AI DRIVEN SMART TOURISM PLATFORM FOR PERSONALIZED SAFE & SUSTAINABLE TRAVEL PLANNING

(COLLABORATIVE TRAVEL COMPANION PLATFORM WITH INTELLIGENT GROUP MATCHING)

Shobithaa Srikanthan

IT21821240

BSc (Hons) in Information Technology Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology Sri Lanka

August 2025

AI DRIVEN SMART TOURISM PLATFORM FOR PERSONALIZED SAFE & SUSTAINABLE TRAVEL PLANNING

(COLLABORATIVE TRAVEL COMPANION PLATFORM WITH INTELLIGENT GROUP MATCHING)

Shobithaa Srikanthan

IT21821240

Dissertation submitted in partial fulfilment of the requirements for the Bachelor of Science (Hons) in Information Technology Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology Sri Lanka

August 2025

DECLARATION

I declare that this is my own work, and this Thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my Thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Srikanthan.S	IT21821240	S. Shobitha

The above candidate has carried out this research thesis for the Degree of Bachelor of Science (honors) Information Technology (Specializing in Software Engineering) under my supervision.

Signature of the supervisor
(Ms. Thilini Jayalath)

29/08/2025

Date

Signature of co-supervisor
(Ms. Karthiga Rajendran)

29/08/2025

Date

ABSTRACT

Group travel has increasingly become a socially enriched activity in which travellers seek not only destinations but also compatible companions and collaborative experiences. While conventional recommender systems provide valuable support for individual travel planning, they fall short in addressing the complexities of group-based decision-making, where diverse preferences such as budget, travel style, and interests must be balanced simultaneously. Recent advances in artificial intelligence (AI) and recommender systems create opportunities to overcome these challenges through intelligent group formation, semantic modelling, and real-time collaboration.

This study presents the design and development of an AI-powered collaborative travel companion platform that dynamically matches travellers into compatible groups, integrates real-time communication, and supports contextually adaptive recommendations. The system leverages Sentence-BERT embeddings to capture nuanced user preferences, employs a LightGBM LambdaRank learning-to-rank model for compatibility scoring, and incorporates threshold-based logic for dynamic group creation when no suitable match exists. To enhance collaboration, a WebSocket-based group chat engine and an integrated AI assistant, TripBot, enable real-time interaction, intent recognition, and context-aware support during group travel planning.

The platform was evaluated using ranking metrics such as NDCG@3 for recommendation quality and classification accuracy for intent detection, confirming both the reliability of the recommendation engine and the responsiveness of interactive components. Beyond technical validation, the system offers strong commercial potential, providing travel agencies, tourism boards, and online platforms with the ability to differentiate services, foster community-driven tourism, and enhance user engagement. Particularly in the Sri Lankan context, the platform highlights the potential of AI-driven group travel systems to support culturally sensitive, socially aware, and sustainable tourism.

Keywords: Group Travel, Recommender Systems, Semantic Embeddings, LightGBM LambdaRank, WebSocket Chat, AI Assistant, TripBot, Tourism Technology.

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to my module coordinator, Dr. Jayantha Amararachchi, for his invaluable guidance, motivation, and encouragement, which inspired me to carry out this project with enthusiasm and commitment.
I am deeply indebted to my supervisor, Ms. Thilini Jayalath, and co-supervisor, Ms. Karthiga Rajendran, for their continuous guidance, constructive feedback, and unwavering support throughout the course of this project. Their patience, valuable insights, and encouragement from the very beginning until the completion of this work have been truly instrumental in its success.

My heartfelt appreciation is also extended to my family, friends, and seniors, whose constant encouragement and belief in me provided the strength and determination to overcome challenges and remain focused. I would also like to acknowledge the lecturers and colleagues at the Sri Lanka Institute of Information Technology for their thoughtful discussions, feedback, and advice, which contributed significantly to the improvement of this project.

Finally, I am grateful to all those who contributed during the requirement-gathering stage and supported various aspects of the project. Their collective efforts and contributions have been invaluable in bringing this work to life.

.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Description
AI	Artificial Intelligence
API	Application Programming Interface
DB	Database
FAISS	Facebook AI Similarity Search
FastAPI	Fast Application Programming Interface
GPU	Graphics Processing Unit
ML	Machine Learning
NDCG	Normalized Discounted Cumulative Gain
RBAC	Role-Based Access Control
SBERT	Sentence-Bidirectional Encoder Representations from Transformers
UI	User Acceptance Testing
WBS	Work Breakdown Structure
WS	WebSocket
VS Code	Visual Studio Code
CI/CD	Continuous Integration / Continuous Deployment
UAT	User Acceptance Testing
TLS	Transport Layer Security
JWT	JSON Web Token
JSON	JavaScript Object Notation

1. INTRODUCTION

1.1 Background Study and Literature Review

1.1.1 Background Study

Travel has always been an essential part of human life, whether for survival, trade, exploration, or personal enrichment. Over the centuries, its role has expanded from a necessity into an activity that provides leisure, cultural exchange, education, and social bonding. In today's world, travel has become even more socially oriented, with people preferring to share their experiences with others rather than journeying alone. Families plan vacations together to strengthen their bonds, groups of friends see travel as a way to create shared memories, and even strangers come together through community-based tours or online groups to explore new destinations collectively. This marks a significant cultural shift: travel is no longer just an individual pursuit but a shared social experience that creates meaning through interaction and collaboration.

At the same time, the way people plan their journeys has changed dramatically due to digital technology. In the past, group travel planning was largely manual. Families or friends would sit together, discuss options, visit travel agents, and spend hours comparing prices, destinations, and routes. This process was not only time-consuming but also prone to conflict, since every individual had their own preferences and priorities. The arrival of online booking platforms simplified some of these tasks by offering easier access to flights, hotels, and tours. However, even as travel moved online, most platforms were still designed for individual users. A person could easily book a hotel for themselves or get personalized recommendations for destinations, but when it came to a group, these systems fell short.

The rise of Web platforms, user-generated content, and social media opened new opportunities. Websites such as TripAdvisor, Booking.com, and Airbnb allowed users to read reviews, share photos, and get inspired by other travellers' experiences. Social media platforms made travel planning even more interactive, as groups of friends could now create events, discuss destinations, and share itineraries online. While these developments gave travellers more information and tools, they still lacked a dedicated

system that could intelligently handle group decision-making. In most cases, one person had to take charge of gathering input from others and making compromises manually. This highlighted an important gap: while technology had advanced to support individuals, it did not adequately support groups.

The demand for group travel, however, continues to grow. Groups offer several advantages over solo travel. Costs are shared, making trips more affordable. Safety is improved, especially in unfamiliar places. Experiences are enriched by having multiple perspectives and interests within the group. More importantly, traveling together creates collective memories that strengthen social bonds. Yet, the very diversity that makes group travel enriching also makes it challenging to plan. Within a group, one person might want a low-budget adventure, another may prefer a luxury trip, while a third might be interested in cultural and historical activities. Without proper support, these conflicting preferences can lead to frustration, decision fatigue, and even disagreements that discourage group travel altogether.

This is where the role of intelligent systems becomes crucial. Unlike traditional platforms that only provide static recommendations, modern technologies powered by artificial intelligence (AI) and machine learning (ML) can go much further. AI can analyse multiple sets of user preferences at the same time, identify overlaps, and find fair compromises that maximize group satisfaction. Natural language models such as SBERT can capture the deeper meaning behind user interests, ensuring that when someone mentions “beach relaxation,” the system does not treat it the same as “surfing,” but instead understands the nuances. Learning-to-rank models, like the LightGBM, LambdaRank model used in this project, can evaluate multiple features such as age, budget, travel style, and group compatibility to recommend the most suitable groups or destinations.

Another important aspect is real-time collaboration. Planning a group trip is not just about generating a static list of recommendations, it is an interactive process. People need to discuss options, negotiate trade-offs, and adapt to changes as plans evolve. Real-time communication tools, such as WebSocket-based group chats, allow members to collaborate directly within the platform, removing the need to switch

between different apps or communication channels. In addition, an AI-powered assistant such as Trip Bot, integrated into the chat, can recognize user intents (like asking for help or sharing experiences) and provide intelligent, context-aware support. This ensures that the group planning process is not only efficient but also engaging and interactive.

In contexts such as Sri Lanka, cultural awareness adds another dimension to the challenge. Travel choices are strongly influenced by cultural norms, religious practices, and local traditions. A system that ignores these factors risks making irrelevant or even inappropriate recommendations. For example, suggesting nightlife activities to a culturally conservative group, or overlooking vegetarian dining preferences, could reduce user satisfaction. Therefore, an intelligent travel platform must also incorporate cultural sensitivity to ensure that its recommendations are not only accurate but also contextually appropriate.

Taken together, these factors highlight the need for a new kind of travel platform—one that is not built solely for individuals but designed around the dynamics of groups. Such a platform must combine advanced AI techniques for preference matching, real-time collaboration tools for group interaction, and cultural sensitivity for local relevance. This project is positioned within this emerging area. Its goal is to design and develop an intelligent collaborative travel companion platform that brings together group matching, semantic analysis of user interests, and real-time AI-assisted communication to overcome the limitations of existing systems. By doing so, it aims to make group travel planning less stressful, more engaging, and ultimately more enjoyable for all members involved.

1.1.2 Literature Review

Recommender systems for group travel have been studied as a distinct challenge within tourism, since group planning requires balancing multiple, and often conflicting, user preferences. Unlike individual recommendation, group scenarios require fairness, compromise, and adaptive decision-making. Several approaches have been proposed in the literature to address these issues.

A group travel recommender system based on approximate constraint satisfaction was introduced in [1], where the objective was to minimize dissatisfaction across group members. This approach demonstrated that fairness should be considered in group recommendations rather than simply averaging preferences. However, constraint-based systems can be computationally expensive, limiting their suitability for real-time platforms.

A systematic review of group recommender system techniques was presented in [2], categorizing them into score aggregation, clustering, and hybrid methods. Score aggregation is simple but risks oversimplifying diverse group dynamics, while clustering can better manage diversity but may fragment groups. Hybrid approaches attempt to balance efficiency with personalization. The taxonomy in [2] provides a useful basis for the present project, which adopts a hybrid model supported by semantic and ranking techniques.

An intelligent recommendation system for travel group planning was proposed in [3], combining user reviews and profiles to suggest both destinations and companions. This highlights the importance of compatibility in group planning. However, reliance on basic textual reviews limits semantic understanding, which can be improved through advanced embeddings such as SBERT. Similarly, [4] introduced a hybrid approach that integrated collaborative filtering with content-based methods to overcome cold-start and narrow recommendation problems, achieving higher accuracy. These principles inform the current work, which combines hybrid recommendation with learning-to-rank models.

Beyond platform solutions, [5] demonstrated how collaborative filtering could be applied in intelligent tourism service robots. Although physical robots are not within

the scope of this project, the study shows the adaptability of recommendation algorithms across domains. Other research has highlighted the use of user-generated content: [6] extracted traveller attributes from photos to enhance personalization, while [7] developed a dynamic clustering system that adapts recommendations as group preferences evolve. These works emphasize flexibility and the importance of integrating diverse data sources, aligning with this project's aim of real-time, adaptive recommendations.

Finally, [8] introduced a cultural-based tourism recommendation system that incorporated community collaboration to ensure cultural relevance. Their results showed that culturally sensitive systems achieve higher acceptance compared to generic models. This is particularly significant for the Sri Lankan context, where cultural values strongly influence travel decisions.

In summary, existing studies have explored constraint satisfaction [1], aggregation and clustering [2], hybrid models [4], user data and companion matching [3], robotics [5], user-generated content [6], dynamic clustering [7], and cultural sensitivity [8]. While these approaches provide valuable foundations, limitations remain. Most systems emphasize destination recommendation rather than dynamic group formation, semantic modelling is often shallow, and real-time collaboration tools are rarely integrated. Cultural awareness is often secondary rather than central. The present project addresses these gaps by combining semantic embeddings, learning-to-rank algorithms, real-time collaboration, and cultural sensitivity into a single intelligent platform for group travel planning.

1.2 Research Gap

Tourism in Sri Lanka is one of the fastest-growing sectors, and digital platforms have become a central tool for travellers to plan their journeys. Over the past decade, the use of online booking services such as Booking.com, Agoda, and Expedia has grown significantly, while platforms such as Airbnb and TripAdvisor have provided travellers with access to accommodations, reviews, and destination insights. At a local level, Sri Lanka has also seen the rise of smaller web portals, Facebook travel communities, and WhatsApp groups where travellers share experiences and coordinate trips. These tools have undoubtedly made travel planning more accessible, but they remain focused largely on the individual traveller. Even when groups are involved, the burden of coordination typically falls on one member who must collect opinions, search for options, and make bookings manually.

A closer examination of these existing systems highlights several limitations. International platforms such as Booking.com and Expedia primarily function as transactional systems. They allow travellers to search for hotels, flights, or activities, but they do not provide meaningful support for collective decision-making. Each member of a group must individually search, compare, and then discuss results externally, often through separate communication channels like WhatsApp or Messenger. This creates fragmentation in the planning process and increases the chances of miscommunication. Similarly, Airbnb Experiences and TripAdvisor provide recommendations based on individual interests, but they lack features that can reconcile different preferences within a group.

Local alternatives in Sri Lanka also reveal important gaps. Many Sri Lankan travellers rely on social media groups and communities, such as Facebook travel groups or online forums, to find companions or share recommendations. While these communities provide a space for interaction, they are informal and unstructured, offering no algorithmic support for matching travellers or aggregating preferences. Decisions in such communities are often driven by vocal individuals, which risks overlooking quieter members' preferences. Furthermore, these platforms lack mechanisms for fairness or balance, meaning one or two dominant opinions often decide the outcome of group plans.

Another limitation of existing platforms is the lack of dynamic group creation. For example, if a traveller in Sri Lanka is interested in joining a group to visit cultural sites like Anuradhapura or natural attractions such as Ella, they would need to manually post in social media groups or rely on pre-arranged tours by agencies. If no existing group aligns with their preferences, they are left without options. None of the existing systems automatically suggest or create new groups based on shared compatibility. This makes spontaneous or community-driven group formation difficult, despite the fact that group travel is becoming increasingly popular among young people and solo travellers looking for companions.

Semantic understanding of user interests is another critical gap. Most platforms rely on predefined filters such as “budget,” “location,” or “type of activity.” While these filters are useful, they are limited in scope and cannot capture nuanced preferences. For example, two travellers may both express interest in “adventure,” but one might mean hiking in the Knuckles Range while the other might mean surfing in Arugam Bay. Current Sri Lankan travel systems lack the advanced semantic models needed to capture these subtleties. As a result, recommendations are often generic and fail to reflect the deeper intentions of travellers.

Existing systems in Sri Lanka also lack real-time collaboration tools. Travel groups often plan through external channels such as WhatsApp, Facebook Messenger, or Viber, which are disconnected from the actual booking or recommendation platforms. This forces users to switch constantly between apps: one for communication, another for booking, and perhaps another for research. This fragmentation makes the process inefficient and discourages collaborative engagement. A platform that integrates recommendation and communication in one place would provide a smoother and more interactive planning experience.

Finally, there is the issue of cultural sensitivity. Many international platforms adopt a global design that does not fully consider the cultural expectations of Sri Lankan travellers. For example, budget-conscious domestic tourists often prefer group sharing of accommodations, local transport options, or culturally significant itineraries that emphasize heritage sites and religious locations. Current systems rarely account for

these localized needs. Furthermore, dietary restrictions (such as vegetarian or Halal requirements), the influence of cultural festivals on travel schedules, and the importance of community-driven travel are often overlooked in existing tools. This reduces the relevance and acceptance of global systems among Sri Lankan travellers.

In summary, while Sri Lanka has access to global booking platforms and vibrant local travel communities, there remains a clear research gap in providing an intelligent, collaborative, and culturally sensitive system for group travel planning. Current tools focus heavily on the individual rather than the group, offer static recommendations instead of adaptive group creation, rely on shallow filtering instead of semantic understanding, and separate communication from decision-making. They also lack the cultural adaptation needed to serve the unique travel behaviours of Sri Lankan users.

This project seeks to bridge these gaps by designing a platform that:

- Dynamically matches travellers to compatible groups and creates new groups when needed.
- Uses semantic embeddings to capture nuanced user preferences.
- Provides real-time group communication integrated with AI support.
- Embeds cultural sensitivity into recommendations for the Sri Lankan context.

By addressing these unmet needs, the system advances beyond existing local and international platforms, providing a more holistic, interactive, and socially enriched group travel planning experience.

1.3 Research Problem

Recommender systems for group travel have advanced considerably in recent years, yet several challenges remain unresolved. Existing approaches have focused on improving fairness through constraint satisfaction [1], classifying methods into aggregation, clustering, or hybrid models [2], extending recommendations to include travel companions [3], and improving accuracy through hybridization [4]. Other directions have explored physical implementations such as tourism robots [5], the use of user-generated content [6], adaptive clustering [7], and cultural sensitivity [8]. While these contributions are valuable, they do not fully address the realities of collaborative group travel planning, particularly in contexts such as Sri Lanka.

A major limitation is that most systems stop at recommending destinations or activities but do not support dynamic group creation**.** Current models assume groups already exist, yet in practice many travellers, especially in Sri Lanka, actively seek companions through informal channels such as social media. Without automated group matching or creation, travellers must rely on manual coordination, which is inefficient and often ineffective.

Another issue is the shallow modelling of preferences**.** Many systems rely on keywords, filters, or simple profile data [3], which cannot capture nuanced meanings. For example, two users stating “adventure” may be referring to very different experiences, such as surfing in Arugam Bay or trekking in the Knuckles Range. Without semantic models, these differences are overlooked, reducing the quality of recommendations.

A further gap lies in the lack of real-time collaboration tools**.** Existing recommender systems provide static outputs, leaving users to negotiate and plan externally through applications like WhatsApp or Facebook Messenger. This disconnects fragments the planning process and diminishes engagement. A system that integrates group chat and AI assistance could make planning smoother and more interactive.

Cultural sensitivity also remains underdeveloped. While [8] emphasizes its importance, most global platforms apply generic models that ignore local norms. In

Sri Lanka, cultural festivals, dietary preferences, and community-driven travel habits strongly influence decision-making. Ignoring these aspects makes recommendations less relevant and lowers acceptance among users.

Finally, the challenge of fairness in balancing diverse preferences is insufficiently addressed. While methods such as [1] acknowledge fairness, few scalable models exist to ensure equitable satisfaction. Simple averaging or majority-based aggregation [2] often privileges dominant members, leading to dissatisfaction among others.

In summary, existing systems excel in specific aspects—constraint optimization [1], clustering [2], personalization [3], hybridization [4], adaptability [7], and cultural awareness [8] but remain fragmented. They do not provide a holistic solution that dynamically forms groups, understands nuanced preferences, supports real-time collaboration, and embeds cultural sensitivity.

The central research problem can therefore be formulated as:

“How can an intelligent recommender system dynamically match travellers into compatible groups by balancing diverse preferences, while also enabling real-time collaboration and cultural adaptability?”

This problem highlights the gap between current systems and the requirements of Sri Lanka’s group travel context, motivating the design of a platform that integrates semantic modelling, adaptive group creation, fairness, real-time communication, and cultural sensitivity.

1.4 Research Objectives

The purpose of this research is to design and develop an intelligent collaborative travel companion platform that supports group travel planning through AI-based matching, semantic preference modelling, and real-time interaction. Unlike conventional systems, which focus mainly on individual recommendations, the proposed platform aims to provide a holistic solution that dynamically creates groups, balances diverse preferences, integrates collaboration tools, and ensures cultural relevance in the Sri Lankan context.

1.4.1 Main Objectives

To design and develop an intelligent collaborative travel companion platform that uses AI-based group matching and real-time interaction to improve group travel planning experiences.

1.4.2 Specific Objectives

- **To analyse user data and generate semantic representations of preferences.**
  User data such as age, budget, interests, and travel style will be encoded into embeddings for more accurate and nuanced matching.
- **To implement a learning-to-rank model for compatibility scoring.**
  A LightGBM LambdaRank model will evaluate multiple factors (age, budget, travel style, interests) to recommend the most suitable groups.
- **To develop a mechanism for dynamic group creation.**
  When no strong match is found, the system will automatically form new groups to ensure inclusivity and scalability.
- **To integrate real-time group communication.**
  A WebSocket-based chat will allow users to collaborate directly within the platform, supporting collective decision-making.
- **To implement an AI assistant (Trip Bot) for planning support.**
  Trip Bot will detect user intents and provide context-aware suggestions during group discussions, reducing planning difficulties.

- **To evaluate the system using technical and user-centered metrics.** Performance will be measured using ranking metrics (e.g., NDCG@3), classification accuracy for intent detection, and user feedback on usability.

1.4.3 Business Objectives

- **Enhancing user engagement:** By supporting collaborative group planning, the platform encourages longer user interaction and repeated use, increasing engagement compared to individual-focused systems.
- **Providing market differentiation:** The platform introduces intelligent group matching and real-time collaboration features that are not offered by existing travel applications in Sri Lanka, thereby positioning it as a unique solution for group travel planning
- **Supporting tourism growth:** By offering culturally sensitive recommendations and promoting community-driven group travel, the system can encourage both domestic and international travellers to explore Sri Lanka more meaningfully.
- **Enabling scalability and monetization:** The platform has potential for future commercialization through premium services (e.g., AI-assisted planning subscriptions), targeted travel deals, or partnerships with local tour operators.
- **Improving trust and decision-making efficiency**
  By integrating group chat and AI assistance, the platform reduces planning uncertainty and builds trust among users, making decisions faster and more reliable.

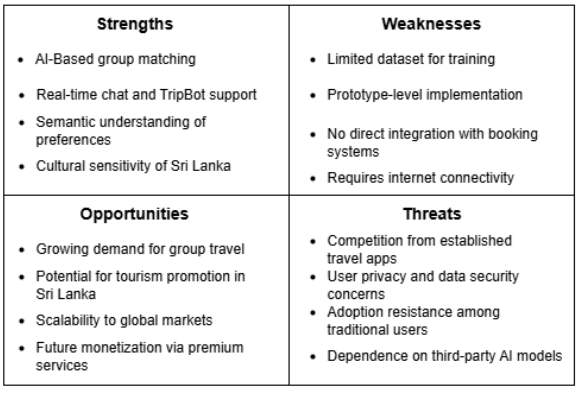


Figure 1: SWOT Analysis Diagram

2. METHODOLOGY

2.1 Methodology

Methodology in research refers to the structured process and systematic techniques used to investigate, design, and validate solutions to defined problems. It provides rigor, clarity, and repeatability, ensuring that outcomes are both academically reliable and practically usable. For this study, the methodology was carefully structured to support the design and development of an intelligent collaborative travel companion platform that assists travellers in planning group trips through AI-based matching, semantic analysis of preferences, and real-time interaction. The approach integrates both academic investigation and technical development, ensuring the system is grounded in user needs while refined through empirical evaluation.

To guide this process, the research adopted the Agile methodology in combination with a Seven-Stage Development Framework**.** Agile was chosen for its emphasis on adaptability, incremental development, and continuous feedback, while the development framework provided a systematic roadmap from feasibility analysis to deployment. Together, these approaches ensured both flexibility and structure in building the platform.

1. Agile Methodology for Research Development:

Agile methodology was applied as a collaborative and iterative development strategy. The platform's goal to intelligently match travellers into compatible groups and support real-time collaboration required continuous improvement as challenges emerged. By dividing the project into smaller, manageable units, features such as the recommendation engine, dynamic group creation, WebSocket-based chat, and TripBot assistant were incrementally designed, tested, and refined.

Regular feedback from supervisors and peers guided improvements in model accuracy, system usability, and cultural adaptability. Agile's iterative structure ensured that

adjustments could be made quickly in response to issues such as limited training data, semantic mismatches, or challenges in chat integration.

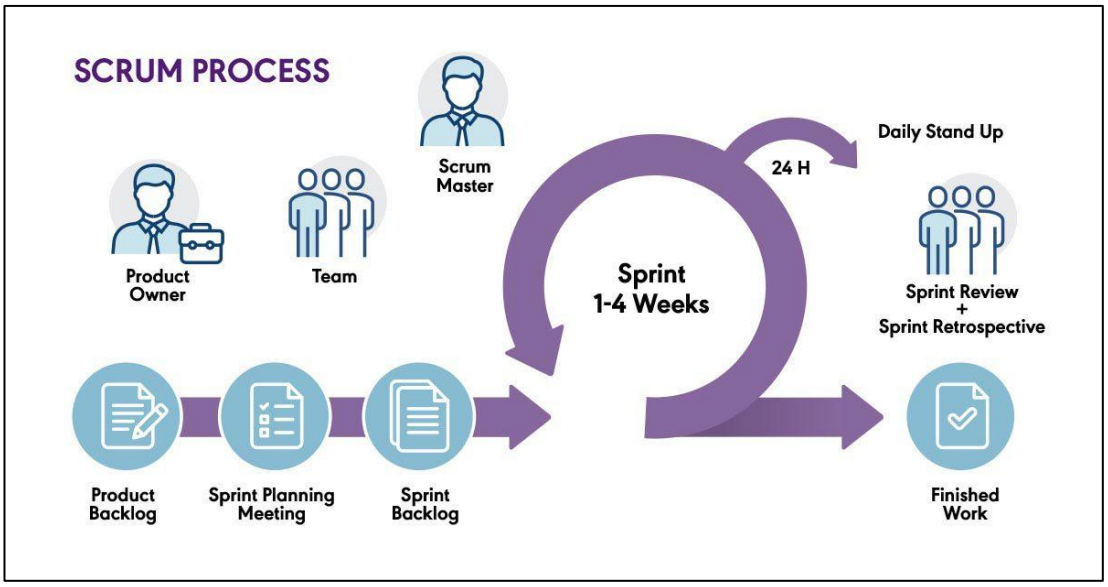


Figure 2: Scrum Workflow Diagram

2. Seven-Stage Development Framework:

Based on the principles of Agile methodology, the research process was developed around a structured seven-stage framework. These stages included feasibility study, requirements gathering, system design, implementation, testing, deployment, and future commercialization. Each stage built upon the outcomes and feedback of the previous one, resulting in a dynamic and adaptive development loop. This iterative approach ensured that technical challenges could be addressed as they emerged, system features were continuously refined, and evolving user needs were incorporated into the design.

The framework was particularly effective in supporting the multidisciplinary nature of the project, which combined advanced machine learning techniques such as semantic embeddings and learning-to-rank models with practical requirements for real-time group collaboration. Agile's emphasis on adaptability and iterative feedback allowed for methodical progress while leaving room for creativity, user input, and technical improvement. As a result, the development cycle not only maintained structure and

academic rigor but also encouraged innovation, ensuring that the final platform aligned with both research objectives and practical user expectation.

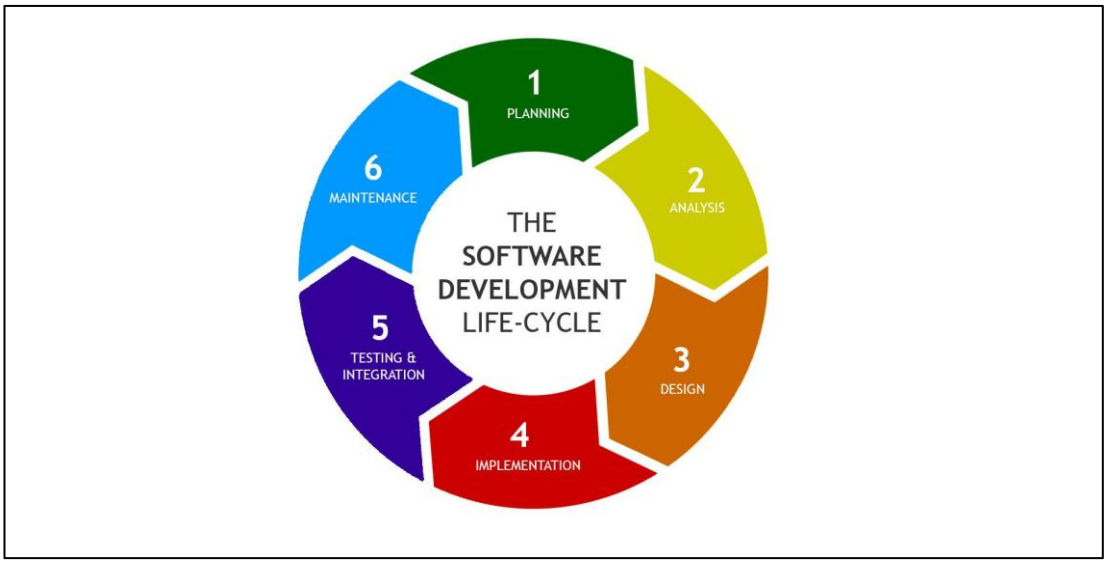


Figure 3: Development Lifecycle Diagram

3. Iterative Development and Collaboration:

The project was broken down into smaller iterative cycles aligned with the seven stages. Each core component such as the recommendation pipeline, group creation logic, and TripBot's intent recognition was built incrementally, tested, and refined. For instance, the initial cosine similarity approach for group matching was later enhanced using LightGBM after testing revealed performance gaps. Similarly, TripBot's classification accuracy was improved through iterative retraining. This feedback-driven cycle ensured the platform remained aligned with research goals throughout development.

4. Flexibility and Responsiveness

Agile's flexibility enabled the project to remain responsive to new challenges and evolving insights. Issues such as semantic embedding errors, real-time chat synchronization, and model performance required quick pivots and refinements. The ability to adapt without disrupting overall progress ensured the platform stayed user-centered and practically relevant, while still maintaining technical rigor.

In summary, the combination of Agile methodology and the Seven-Stage Development Framework supported a research process that was structured yet adaptable. Agile provided continuous iteration and feedback, while the framework ensured systematic direction across all phases. Together, these approaches enabled the development of a platform that is technically robust, academically rigorous, and aligned with the needs of collaborative group travellers in Sri Lanka.

2.1.1 Feasibility Study/ Planning

This phase assesses the feasibility of designing and implementing an intelligent collaborative travel companion platform tailored for group travel planning in Sri Lanka. The system aims to dynamically match travellers into compatible groups based on their preferences (age, budget, travel style, and interests), provide personalized recommendations using a learning-to-rank model, and support real-time group collaboration through chat and an AI assistant (Trip Bot). The feasibility study evaluates the **technical, economic, legal, operational, scheduling, and cultural viability** of the platform to ensure its practical implementation and long-term sustainability.

1. Technical Feasibility

Data Availability: The foundation of the system is built on user profile data (age, budget, style, interests) and group travel preferences. This data will be collected through user registration and group interactions. Machine learning models such as SBERT embeddings (for semantic understanding of interests) and LightGBM LambdaRank (for ranking compatible groups) require structured datasets. For feasibility, mock datasets and synthetic user profiles were initially used, followed by gradual integration of real user inputs.

System Requirements and Infrastructure: The platform is implemented using a FastAPI backend, MongoDB database, and React frontend. WebSocket technology enables real-time group chat, while TripBot leverages intent classification and external APIs (e.g., Google Gemini) for AI-powered support. The system is lightweight and deployable on cloud environments such as AWS or Azure, ensuring scalability.

Model Integration and Adaptivity: The technical feasibility of semantic user modelling and group recommendation was validated using pre-trained SBERT embeddings and a trained LightGBM LambdaRank model for ranking compatible groups. In addition, an intent classification model was integrated to enable adaptive group collaboration. This classifier, built on SBERT sentence embeddings, detects user intents in real-time chat (e.g., planning a trip, asking for help, sharing experiences) and allows Trip Bot to respond contextually. The group creation logic, tested via cosine

similarity and ranking thresholds, further demonstrated that forming new groups dynamically is computationally viable in real-time environments.

2. Economic Feasibility

Budgetary Considerations: The primary costs involve cloud hosting for deployment, GPU resources for initial model training, and subscription APIs (if extended commercially). During development, free or student-license tools such as MongoDB Atlas, GitHub, and educational cloud credits reduce expenses.

Resource Allocation: Development resources include open-source frameworks (FastAPI, React, Scikit-learn, LightGBM) and pretrained NLP models (SBERT). No paid licenses are required for core functionality.

Return on Investment (ROI): The platform has strong potential to support both domestic and international tourism in Sri Lanka by encouraging group-based travel. With minimal investment in open-source technologies, the platform provides high academic and social value, making it economically feasible.

Table 1: Cost Management

Type	Cost
Internet use and web hosting	10,000 LKR
Google Colab Pro Subscription	3,000 LKR
Publication cost	15,000 LKR
Stationery and printing	2,000 LKR
Azure Cloud Hosting (via Student Pack)	0 (Covered by Credits)
Total	30,000 LKR

3. Legal and Ethical Feasibility

Data Privacy and Ethics: The platform collects user profile data (age, budget, travel style, interests) strictly for group matching and recommendation purposes. No sensitive personal identifiers (e.g., national ID, financial data) are stored. All data is anonymized, encrypted, and managed under ethical research practices.

Intellectual Property: The platform is built with open-source tools under proper licenses. AI models (e.g., SBERT, LightGBM) are used within permissible research and development contexts. No copyright conflicts exist, as all system outputs (group recommendations, TripBot interactions) are generated dynamically.

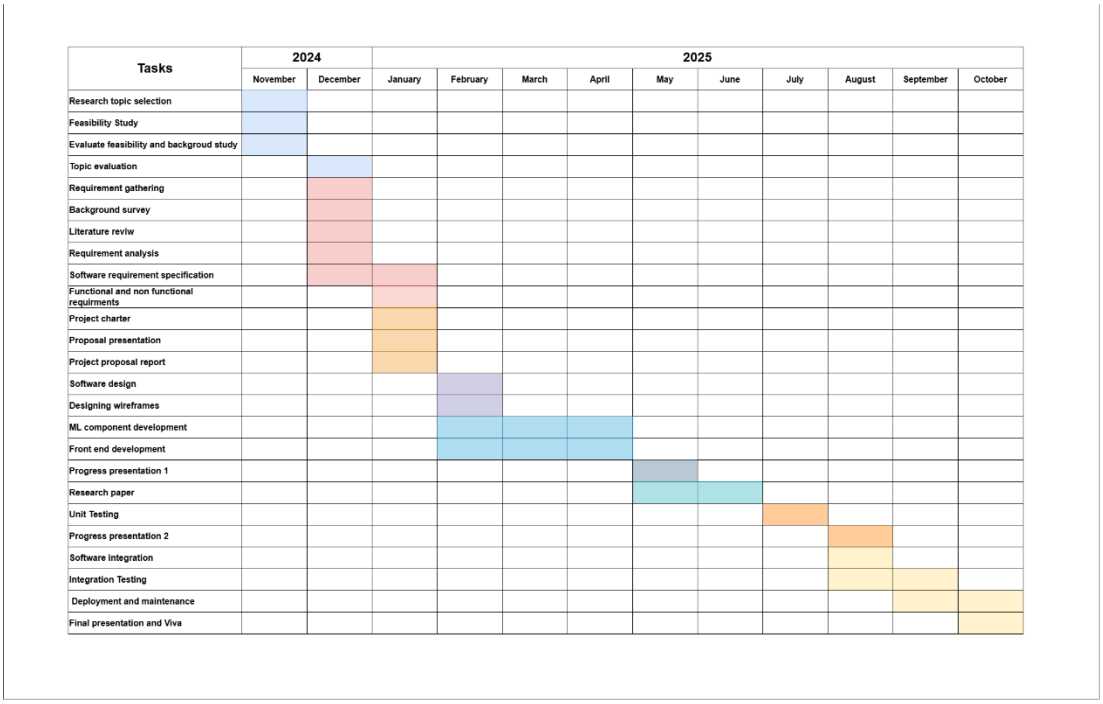
4. Operational Feasibility

System Usability and Response Time: The platform is designed to be lightweight and user-friendly, accessible via desktop and mobile browsers. Real-time group chat and TripBot responses are optimized to ensure low latency.

Maintenance and Extensibility: The platform follows a modular architecture, allowing individual components (recommendation engine, group creation, chat, AI assistant) to function independently. This design ensures that new features can be incorporated without major rework, demonstrating both long-term sustainability and adaptability to future requirements.

5. Time / Schedule Feasibility

The project followed an academic calendar with milestones for requirement gathering, design, development, testing, and evaluation. Agile methodology with iterative sprints ensured progress tracking. Gantt charts were prepared to monitor progress, ensuring timely delivery of each milestone.



Tasks	2024		2025									
	November	December	January	February	March	April	May	June	July	August	September	October
Research topic selection												
Feasibility Study												
Evaluate feasibility and backgroud study												
Topic evaluation												
Requirement gathering												
Background survey												
Literature reviw												
Requirement analysis												
Software requirement specification												
Functional and non functional requirments												
Project charter												
Proposal presentation												
Project proposal report												
Software design												
Designing wireframes												
ML component development												
Front end development												
Progress presentation 1												
Research paper												
Unit Testing												
Progress presentation 2												
Software integration												
Integration Testing												
Deployment and maintenance												
Final presentation and Viva												

Figure 4: Gantt Chart (Schedule Management)

6. Social and Cultural Feasibility

The platform is culturally relevant to Sri Lanka, where group travel is common among families, students, and tourists. By supporting culturally sensitive recommendations (e.g., heritage sites, religious tourism, budget-friendly options), the system aligns with user expectations. The interface is designed to be simple and accessible, ensuring adoption across different age groups and demographics.

- **Risk Management Plan**

A risk management plan has been formulated to anticipate, address, and mitigate potential issues that may arise during the development, integration, and deployment of the platform. This ensures continuity, minimizes disruptions, and supports proactive response strategies.

The following table outlines possible risks associated with this component, their triggers, responsible parties, planned responses, and required resources.

Table 2: Risk Management Plan

Risk	Trigger	Owner	Response	Resource Required
Delay in recommendation results	Large dataset size or model latency	ML Developer	Optimize queries, cache frequent results, fine tune model parameters	Profiling tools, LightGBM logs, MongoDB query optimizer
Illness or absence of project team members	Illness / Personal emergencies	Project Leader	Inform supervisor, Redistribute tasks among team	Gantt Chart, Backup resources
Missed progress review due to panel unavailability	Supervisor or panel member absent	Project Leader	Reschedule review or submit written progress update	Meeting log Email documentation
Technical failure during development	Software bugs, system crashes	Team	Run tests regularly, Implement	Testing Suite, Logs, Backup Mechanisms

			Backups and error handling	
Feedback from testing sessions requires urgent changes	UAT feedback from teachers/ students	Project Leader	Implement necessary changes Re-test and update documentation if needed	Product Backlog Gantt Chart
WebSocket chat disconnection	Server downtime or unstable internet	Backend Developer	Implement automatic reconnection logic, fallback to HTTP polling	FastAPI logs, WebSocket reconnect handler
Internal communication gap within the group	Lack of updates or unclear task distribution	Team	Weekly sync-up meetings Update Jira/Gantt and documentation regularly	Communication tools Shared task tracker
Delay in deployment to cloud	Time constraints or configuration issues	Project Leader	Allocate buffer time, Prepare contingency for demo/testing	Timeline Buffer, Hosting Access

- **Communication Management Plan**

The Communication Management Plan ensures that the project team, supervisor, and co-supervisor have the information they need to perform their roles effectively. Since the project integrates AI models, real-time chat systems, and group recommendation features, structured communication was essential to align technical, academic, and practical outcomes.

The plan outlines the communication objectives, tools, and meeting structures to ensure clarity, minimize misunderstandings, and facilitate smooth project execution.

Communication Objectives:

To guarantee effective collaboration, communication in the project followed these principles:

- Adequate: Shared in the right format with relevant technical/academic details.
- Specific: Tailored to the targeted audience (supervisor, co-supervisor, or team).
- Sufficient: Provided enough detail for decisions and evaluations.
- Concise: Avoided repetition and unnecessary detail.
- Timely: Delivered at the right stage of the project (e.g., before milestones).

Communication Media:

The following tools were used to ensure continuous and effective communication:

- Email – formal communication with supervisor and co-supervisor (progress updates, submission confirmations).
- Documents (Word, PowerPoint, Overleaf) – preparation of reports, presentations, and milestone deliverables.
- Microsoft Teams / Zoom – virtual meetings for progress discussions, issue resolution, and presentations.
- WhatsApp Group Chat – quick daily coordination and reminders among team and supervisor.
- GitHub & Jira/Trello – version control, task assignment, and progress tracking.

Table 3: Communication Management Plan

Meeting Type	Attendees	Purpose	Frequency	Agenda Items
Planning Kick-off Meeting	Supervisor, Co-supervisor, Student	Formally launch project, define scope (group recommender, TripBot, chat), assign responsibilities, identify risks.	Once at the start	Introduce research problem and objectives. Allocate responsibilities. Confirm communication tools & timeline.
Execution Kick-off Meeting	Supervisor, Co-supervisor, Student	Begin development phase (recommendation engine, chat system, TripBot). Reconfirm milestones and quality standards	Once at start of development phase	Present finalized work plan. Set up documentation and testing protocols. Confirm workflows and escalation paths.
Internal Project Status Meeting	Student	Track weekly progress, review code (FastAPI, MongoDB, React), debug issues, and	Weekly	Share component status (recommendation, group creator, chat). Compare actual vs. planned

		plan next sprint		work. Identify blockers. Set action items for next week.
Project Status Meeting (Supervisor/Co-supervisor)	Supervisor, Co-supervisor, Student	Provide formal progress updates (reports, demos of group recommendation, TripBot chat). Receive academic and technical feedback.	Bi-weekly or as requested	Demonstrate working modules. Present evaluation results. Discuss issues and supervisor feedback.
Project Review Meeting	Supervisor, Co-supervisor, Student	Review progress before milestones (Proposal, PP1, PP2, Final). Ensure readiness for evaluations.	Before each milestone	Review document and demo readiness. Validate milestone deliverables. Address unresolved issues.
Change Control Meeting	Supervisor, Co-supervisor, Student	Address changes after supervisor/pa	As needed	Review feedback. Prioritize

		nel feedback (e.g., model adjustments, UI refinements).		required changes. Assign new responsibilities.
Project End Review Meeting	Supervisor, Co-supervisor, Student	Final reflection on outcomes (platform functionality, user testing results). Capture lessons learned.	Once at end of project	Summarize platform performance. Review objectives achieved. Reflect on collaboration. Identify areas for future research.

2.1.2 Requirement Gathering & Analysis

The Requirement Gathering and Analysis phase was a pivotal component in the development of the AI-Driven Collaborative Travel Companion Platform. This stage focused on systematically identifying and refining the functional and non-functional requirements necessary to implement a system capable of intelligent group matching, personalized recommendations, real-time collaboration, and cultural adaptation.

To ensure practical applicability, requirements were collected through stakeholder feedback, surveys, and reference to existing tourism platforms**.** The process emphasized the Sri Lankan tourism context, ensuring alignment with local cultural expectations, affordability needs, and group travel behaviours.

2.1.2.1. Functional Requirements

Functional requirements specify the precise features the system must provide to achieve its goals in group-based travel planning.

Stakeholder Input Approach:

Instead of relying solely on formal interviews, the requirement identification process was based on three main sources:

- **Survey feedback** was collected from university students, young professionals, and domestic travellers, capturing their preferences for group travel, budget, travel style, and collaboration needs.
- **Informal discussions** with local tour operators and frequent travellers provided insights into pain points in existing group travel coordination, such as miscommunication, mismatched preferences, and fragmented planning tools.
- **Review of existing tourism platforms** (e.g., TripAdvisor, Airbnb, Booking.com) highlighted gaps such as lack of dynamic group creation, limited semantic understanding of preferences, and no integrated group communication.

Figure 5: Survey Details

Identification of Key Functionalities:

Stakeholder discussions and analysis identified the following essential system features:

- **User Registration & Profile Creation** – Travellers register by providing user details such as age, budget, travel style, and interests.
- **Group Recommendation Engine** – A LightGBM LambdaRank model recommends the top compatible groups for each user, based on semantic embeddings of interests and preferences.
- **Dynamic Group Creation** – If no suitable match exists, the system automatically creates a new group using similarity thresholds.
- **Real-Time Collaboration** – A WebSocket-based chat system enables group members to communicate seamlessly.

- **AI Assistant (TripBot)** – Integrated into chat, TripBot detects user intents (plan_trip, ask_help, share_experience) using an SBERT classifier and provides Gemini API-powered responses.
- **Cultural Sensitivity** – Recommendations account for local travel practices (heritage sites, religious locations, dietary needs), ensuring relevance to the Sri Lankan context.
- **User-Friendly Frontend** – A React-based interface supports login, recommendation viewing, group creation/joining, and chatting.

Validation and Prioritization:

The requirements were validated through survey feedback and supervisor consultations, ensuring that features aligned with both end-user expectations and academic objectives. Requirements were prioritized into three categories:

- **Essential (MVP)** – Fundamental features necessary for the delivery of a minimum viable product. User registration, group recommendation, group creation, chat integration, TripBot intent handling.
- **Desirable** – Value-enhancing features that contribute to user experience but are not critical to the primary functionality. Cultural adaptation in recommendations, TripBot fallback strategies, group name auto-generation.
- **Optional (Future Work)** – Advanced features considered for potential inclusion in subsequent iterations.

This prioritization ensured a **minimum viable product** was achievable within the project timeline, while leaving scope for future enhancements.

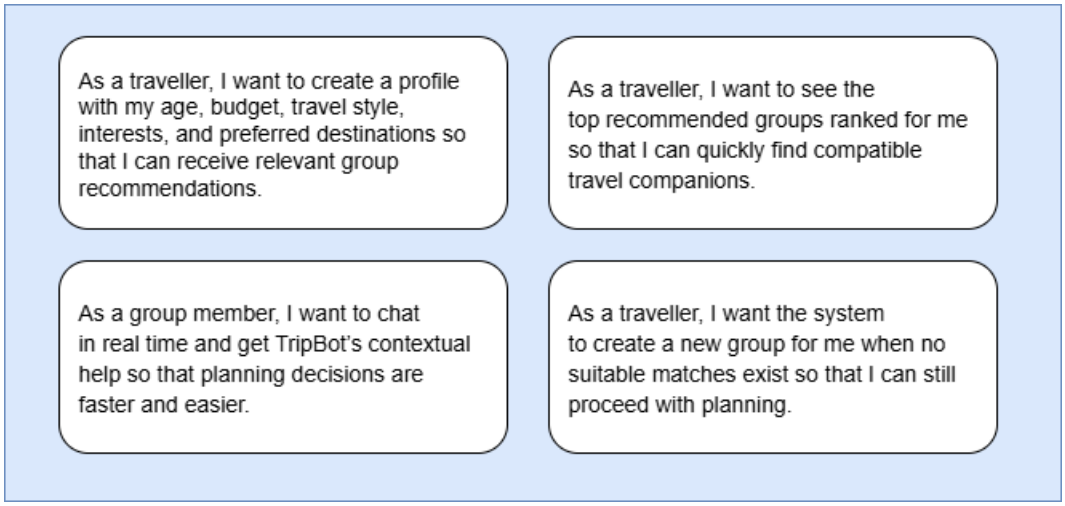


Figure 6: User Stories for Requirements

These user stories were used to shape sprint planning, prioritize development features, and validate that the platform meets the practical expectations of its primary user base, Sri Lankan tourists.

Mapping the user requirements in the use case diagram and sequence diagram is very important to identify the requirements clearly and it will be used to prioritize the requirements. Figure 7 explains the use case diagram and figure 8 illustrates the sequence diagram.

Use Case diagram:

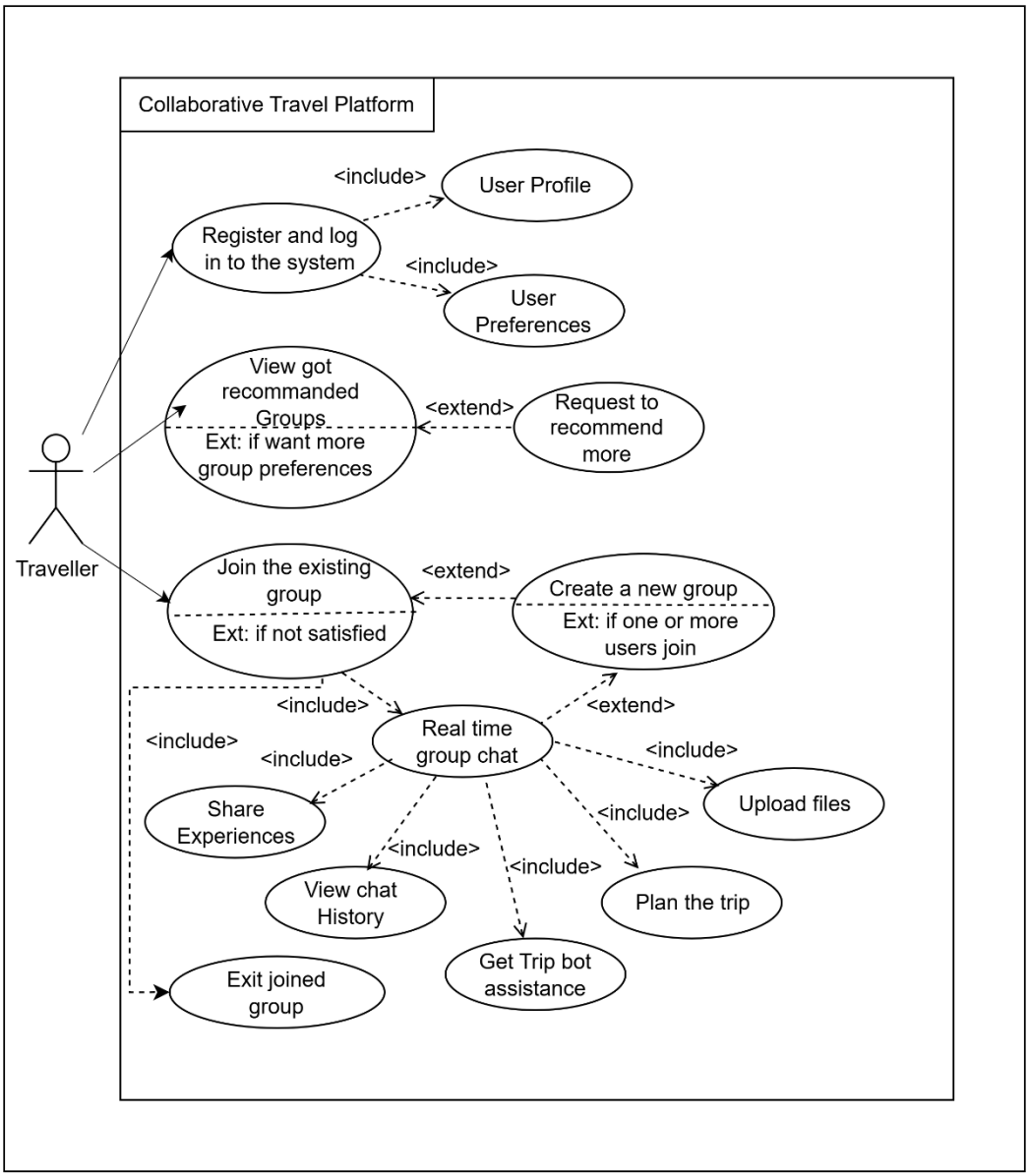


Figure 7: Use case diagram

sequence diagram:

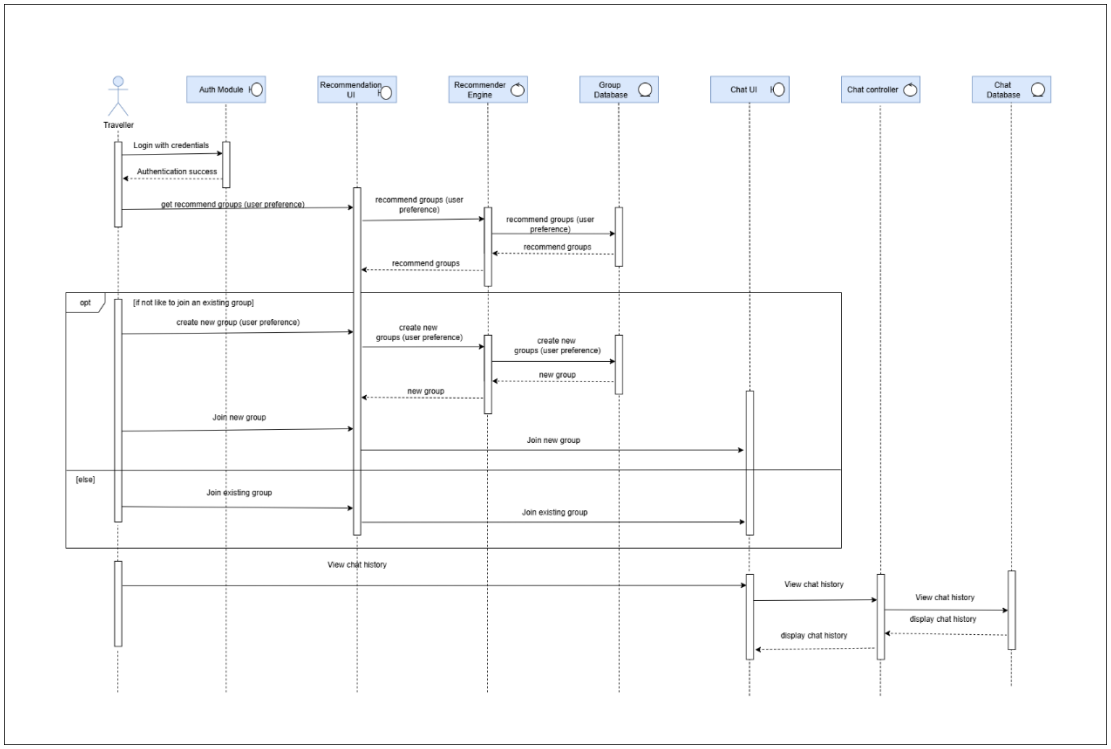


Figure 8: Sequence Diagram

2.1.2.2 Non-Functional Requirements

Non-functional requirements define the **quality attributes** and **operational constraints** of the AI-Driven Collaborative Travel Companion Platform. They ensure the system remains efficient, usable, secure, and dependable during real-time group travel planning in the Sri Lankan context.

- **Performance & Responsiveness**: The platform must provide timely responses across all critical operations. Group recommendations generated through SBERT and LightGBM should appear quickly for typical workloads. Group creation based on similarity thresholds must also complete promptly so that users are not left waiting. WebSocket chat is required to deliver messages with near-instant responsiveness, ensuring fluid real-time collaboration. TripBot responses, whether handled through local intent classification or the Gemini API, should return within a reasonable timeframe, with fallback options available if the external API is delayed. To sustain performance as the system scales, techniques such as caching, pagination, and indexed database queries will be applied.
- **Accuracy & Relevance of Recommendations**: Group suggestions must reflect traveller age, budget, style, interests, and destinations with high semantic fidelity. Model quality is evaluated using ranking metrics (e.g., NDCG@k) and intent-classifier accuracy. Results should avoid duplicates, respect cultural context (e.g., heritage/religious sites, dietary preferences), and remain up-to-date as group data changes.
- **Usability & Accessibility**: The React UI must be simple, mobile-friendly, and consistent, supporting clear flows for registration, seeing recommendations, creating/joining groups, and chatting. Use readable typography, meaningful errors, and confirmations. Aim for basic WCAG-aligned practices (contrast, keyboard navigation); future localization (Sinhala/Tamil) is considered.
- **Reliability**: Chat, recommendation, and group-join workflows must operate consistently without data loss. Messages and state updates are persisted; idempotent routes and validation guard against double joins and race conditions. Automated tests cover critical paths.

- **Availability**: Target high uptime for core APIs and chat. Implement automatic WebSocket reconnection, health checks, and graceful degradation (e.g., fallback to HTTP polling or cached recommendations) during transient outages.
- **Security, Privacy & Ethics**: Follow data-minimization: collect only what's needed (age, preferences, style, budget). Use TLS in transit and secure storage for credentials/API keys; protect Gemini API keys server-side. Provide clear consent and the ability to remove data. TripBot outputs must remain appropriate and fair, avoiding harmful or biased suggestions.
- **Scalability**: Enable horizontal scaling of FastAPI workers and MongoDB with proper indexing. Batch/async inference and caching should sustain throughput as users and groups grow. Designed to run on cloud platforms with autoscaling.
- **Maintainability & Extensibility**: Modular architecture (recommendation engine, group creator, chat, TripBot) with clean interfaces, logging, and monitoring supports quick fixes and feature additions (e.g., itinerary/booking) without major rework. CI/CD and code style checks keep the codebase healthy.

Main Non-Functional Requirements Identified:

- Usability
- Performance (Speed & Responsiveness)
- Accuracy & Relevance of Recommendations
- Reliability
- Availability
- Security, Privacy & Ethics
- Scalability
- Maintainability & Extensibility

2.1.3 Designing

The Designing phase of this research plays a central role in shaping the intelligent structure of the collaborative travel companion platform. This phase focuses on the model architecture, component interactions, and logical flows required to deliver a responsive, personalized, and socially engaging system for group travel planning in Sri Lanka. The design integrates SBERT-based semantic user modeling, a LightGBM LambdaRank recommender, cosine similarity for dynamic group creation, and real-time collaboration supported by WebSocket chat and TripBot. Each module has been deliberately structured to address a specific need: accurate group matching, adaptive group formation, seamless communication, and AI-powered assistance during planning discussions.

At this point in the Software Development Life Cycle (SDLC), abstract ideas about intelligent recommendations and adaptive collaboration are translated into modular components that interact in a logically coherent architecture. The system's structure is defined with the goal of ensuring compatibility among travellers, enabling smooth group decision-making, and maintaining cultural relevance to Sri Lankan tourism. The final architecture not only supports the user journey from login and profile creation to group recommendation, group formation, and interactive planning with TripBot but also outlines the internal machine learning pipelines and processing layers that make these experiences intelligent, dynamic, and sustainable.

System Architecture Diagram:

The system architecture diagram illustrates the high-level design of the collaborative travel companion platform. The process begins when a user accesses the application interface and submits preferences such as age, budget, travel style, interests, and preferred destinations. These details are stored in the MongoDB database and processed by the group recommendation engine. Using SBERT embeddings and a LightGBM LambdaRank model, the recommendation engine ranks available groups and recommends the top matches to the user. If a compatible group is found, the user can immediately join it; if not, the group creation module applies cosine similarity

thresholds to dynamically form a new group with an auto-generated identity. For groups that remain inactive, users are placed in a waiting state until another compatible member joins, ensuring smooth continuation of the group formation process.

Once groups are established, the platform supports real-time collaboration through a WebSocket-based chat service. Within this environment, TripBot acts as an intelligent assistant, leveraging SBERT-based intent classification to detect user needs such as planning a trip, asking for help, or sharing experiences. For complex travel queries, TripBot connects with the Gemini API to provide context-aware recommendations and suggestions. All user interactions, including chat messages, bot responses, and group planning decisions, are logged and stored in the database, allowing continuity, onboarding summaries for new members, and reliable access to historical data. This integrated design ensures a seamless user journey while maintaining scalability, personalization, and cultural relevance to group travel planning in Sri Lanka

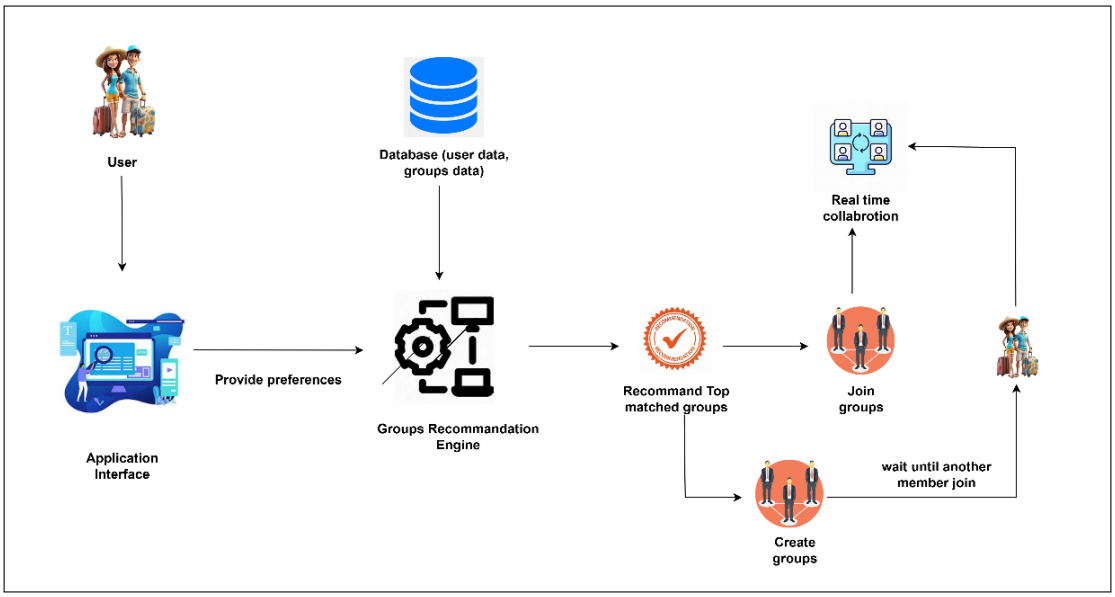


Figure 9: System Diagram

Component Architecture Overview:

1. Group Recommendation Engine (SBERT + LightGBM LambdaRank)

This module acts as the foundation of the platform's intelligence. User preferences including age, budget, travel style, interests, and destinations are embedded using SBERT to capture semantic meaning. These embeddings, combined with encoded categorical and numerical features, are processed by a pre-trained LightGBM LambdaRank model which ranks all available groups. The system then recommends the top-matched groups, ensuring each user is presented with the most compatible options in real time.

2. Dynamic Group Creation Module (Cosine Similarity + Threshold Logic)

When no group achieves a sufficient compatibility score, the dynamic group creation module ensures continuity of the user experience. Using cosine similarity on feature vectors, the module determines overlap with existing groups. If no match exceeds a defined threshold, a new group is automatically created with a composite name derived from the user's style, budget, and interests. This guarantees that users are never excluded and can initiate a new group journey seamlessly.

3. Real-Time Collaboration Layer (WebSocket Chat + TripBot)

The collaboration environment is powered by a WebSocket-based chat service that supports instant, low-latency communication among group members. Embedded within this layer is TripBot, an intelligent assistant that uses SBERT-based intent classification to interpret member messages (e.g., planning, asking help, sharing experiences). TripBot provides contextual suggestions, manages group planning summaries, and for complex travel questions, connects to the Gemini API to provide enriched recommendations.

3. Group Data Management (MongoDB Persistence Layer)

MongoDB serves as the backbone of data management, storing user profiles, group metadata, and chat histories. This persistence enables onboarding summaries for new members, retrieval of group history, and reliable synchronization of group status. The

database ensures that user and group interactions remain consistent, accessible, and scalable across concurrent sessions.

This dual-mode architecture combining intelligent group recommendation and creation with real-time AI-assisted collaboration ensures that the platform delivers both accurate traveller matching and a socially engaging planning environment. By integrating machine learning models with real-time communication, the system provides a dynamic, scalable, and culturally aligned group travel planning experience tailored for Sri Lanka.

Flow Diagram of the Collaborative Travel Companion Platform:

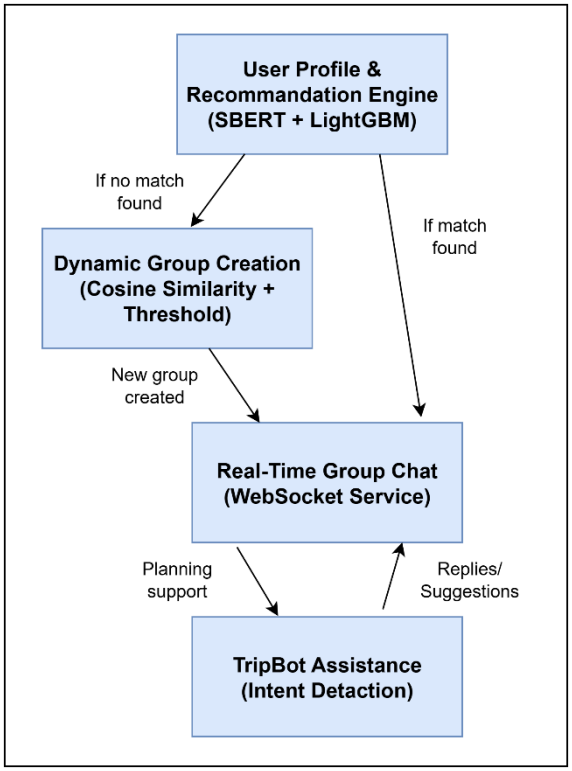


Figure 10: Flow Diagram

2.1.4 Implementation

The Implementation phase of this research marked a critical step in transforming the proposed design into a fully functional collaborative travel companion platform tailored for group travel planning in Sri Lanka. This phase involved the development and integration of all major system components, including the user registration and profile management module, the SBERT and LightGBM–based group recommendation engine, the dynamic group creation module powered by cosine similarity, the WebSocket-driven real-time chat service, and the TripBot assistant which combines SBERT-based intent classification with the Gemini API. The implementation process adhered to a modular development approach to ensure clarity, maintainability, and parallel progress across different system segments.

Task Breakdown and Project Management:

To ensure systematic progress and visibility during development, project tasks were carefully planned and managed using Agile methodology with sprint-based development. The project was divided into three major sprints, each spanning approximately one month. Task management was supported through GitHub issues and project boards, where epics, tasks, and sub-tasks were organized according to the main modules: backend services (FastAPI + MongoDB), machine learning model integration (SBERT + LightGBM), group creation logic, TripBot integration, and frontend development (React-based UI).

During Sprint 1, the focus was on implementing backend routes for user registration and group management, setting up the MongoDB schema, and preparing initial datasets (synthetic and survey-based). Sprint 2 concentrated on integrating the recommendation engine, which included SBERT embeddings, feature encoding, and LightGBM LambdaRank scoring, alongside the group creation module that applies cosine similarity thresholds and auto-generates group identities. Sprint 3 focused on building and refining the WebSocket chat service, implementing TripBot with SBERT intent classification and Gemini API support, and integrating frontend components for login, recommendation display, and group chat. Each sprint also included unit testing, bug fixing, and performance optimizations to ensure stable system performance. This

structured workflow allowed for steady progress, code reviews, and rapid feedback integration, ultimately contributing to the completeness and robustness of the platform.

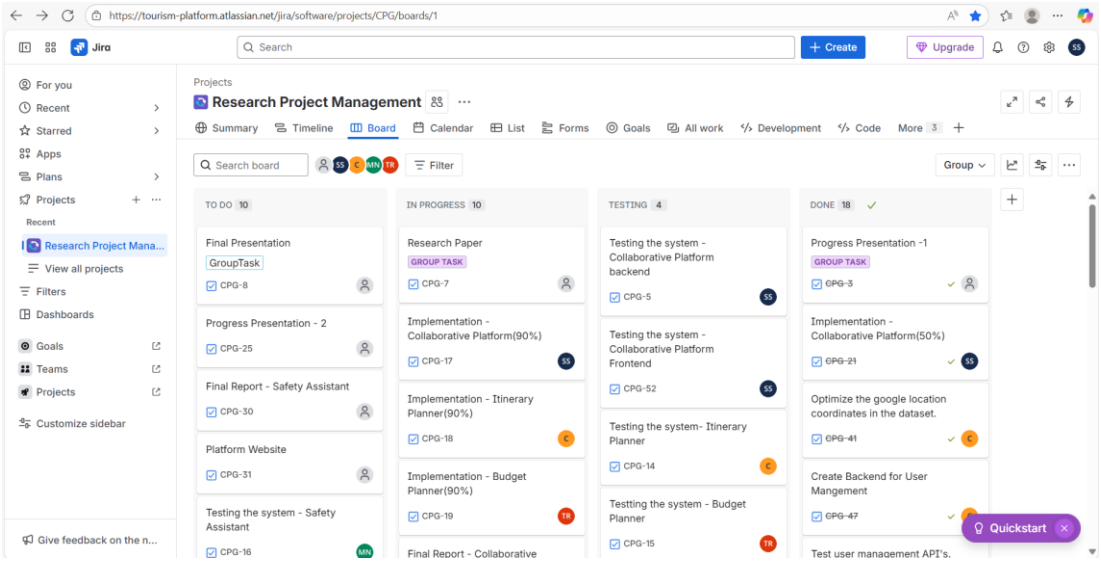


Figure 11: Jira Board

Work Breakdown Structure (WBS):

A Work Breakdown Structure (WBS) was developed to provide a hierarchical view of the system's development process. At the highest level, the project was divided into major modules such as User Management, Recommendation Engine, Group Creation Module, TripBot AI Assistant, Real-Time Chat, Frontend UI Development, and Integration & Testing. Each module was further decomposed into subcomponents. For instance, the Recommendation Engine module was subdivided into SBERT embedding integration, LightGBM LambdaRank scoring, and top-k recommendation output. The Group Creation module covered similarity computation, threshold evaluation, and automatic group naming. The TripBot module included intent classifier integration and context management. Similarly, the Frontend module was broken down into user register and login, recommendation display and group chat interface.

This structured WBS enabled efficient task allocation, dependency tracking, and resource distribution across the project timeline. It also ensured that all contributors remained aligned with project objectives, while maintaining a clear view of progress

and priorities. A visual representation of the WBS used for the collaborative travel companion platform is illustrated in **Figure 12**.

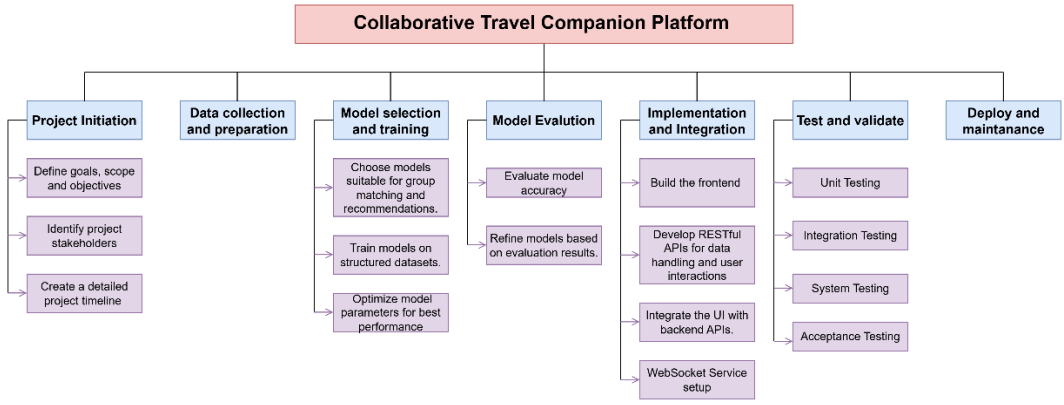


Figure 12: Work Breakdown Structure

Development Environment and Tools:

The implementation of the collaborative travel companion platform was carried out using Visual Studio Code (VS Code), a lightweight yet powerful source code editor that provided flexibility and efficiency during development. Its integrated terminal, debugging tools, and wide extension ecosystem offered a seamless environment for coding, testing, and managing both frontend and backend components. With built-in support for Python and JavaScript/TypeScript, VS Code was well-suited to the full-stack nature of the platform.

The backend development was completed using Python, with FastAPI selected as the primary web framework due to its high performance, modular structure, and compatibility with asynchronous operations. FastAPI enabled the creation of well-structured RESTful APIs responsible for essential functionalities such as user registration and authentication, group recommendation, group creation, and chat communication. The backend also managed interactions with the MongoDB database, ensuring scalable and efficient data storage for user profiles, group metadata, and chat logs.

The core group recommendation engine relied on SBERT (Sentence-BERT) embeddings to represent user interests and destinations semantically, while a

LightGBM LambdaRank model was used to rank and recommend the most compatible groups. In scenarios where no suitable matches were found, the group creation module applied cosine similarity thresholds and rule-based naming to dynamically generate new groups. These components were integrated through scikit-learn, NumPy, and joblib for model loading, feature encoding, and preprocessing.

Real-time collaboration was supported through WebSocket services implemented in FastAPI, which enabled low-latency group communication. Embedded in this chat service was TripBot, an intelligent assistant designed to improve group interaction. TripBot used an SBERT-based intent classifier to identify user intents (planning, help requests, sharing experiences), while for complex travel queries it communicated with the Google Gemini API to provide context-aware suggestions and insights. This hybrid design allowed TripBot to support both local decision-making and external knowledge integration.

The frontend was developed using React.js with JavaScript, creating an interactive and responsive interface that worked across desktop and mobile devices. Core components included user authentication, RecommandPage (recommendations and group creation), and ChatPage (real-time collaboration with TripBot). The frontend communicated asynchronously with the backend through REST APIs and WebSocket connections, ensuring smooth real-time updates and user interactions.

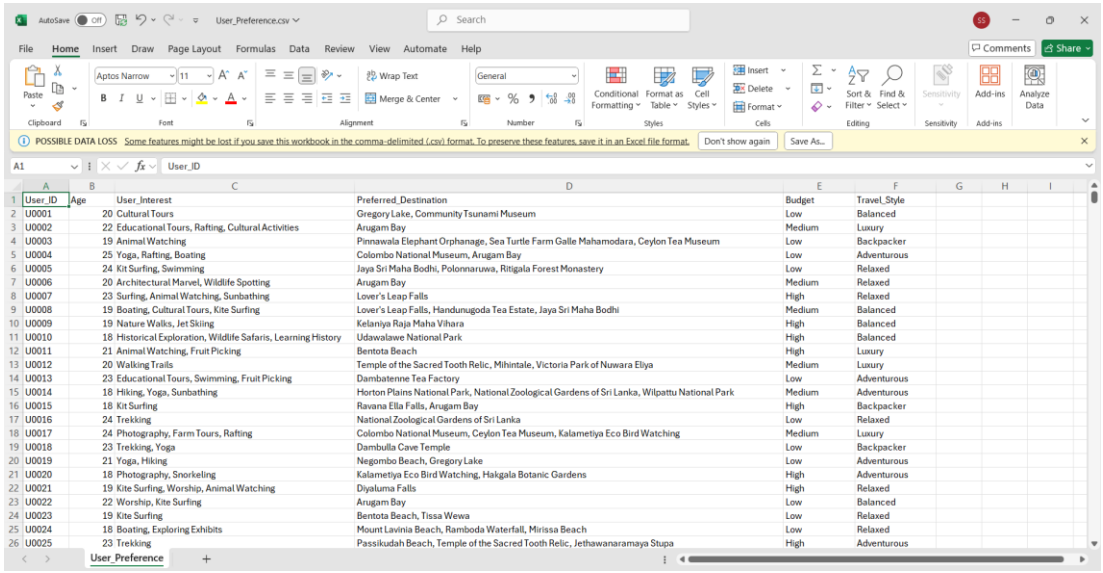
Version control was maintained using Git, with the repository hosted on GitHub for collaborative development, source tracking, and rollback support. This ensured safe and transparent project management. The chosen technology stack FastAPI, MongoDB, SBERT, LightGBM, WebSockets, React.js, provided a balanced foundation of performance, scalability, and maintainability, aligning with the project's objective of delivering a dynamic, intelligent, and socially engaging travel planning platform.

2.1.4.1 Dataset Preparation for Group Recommendation

The dataset preparation process was a crucial foundation for the development of the collaborative travel companion platform, as the quality of recommendations depended heavily on the availability of structured and representative data. Primary data was collected through a survey conducted among university students and frequent travellers, since these groups represent a significant portion of potential users of group travel platforms. The survey captured essential details such as age, budget preferences, travel style, interests, and preferred destinations, ensuring the inclusion of diverse travel behaviours and user expectations.

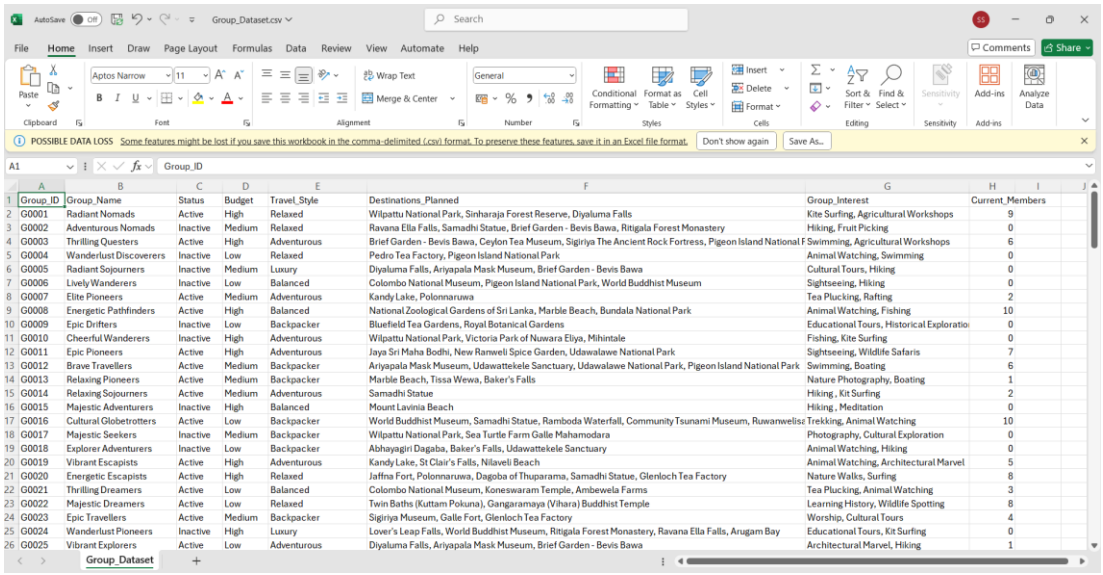
To strengthen the dataset, secondary sources were also explored. Publicly available datasets on Sri Lankan tourism, including information on popular destinations, cultural attractions, and general travel behaviour trends, were integrated alongside the survey responses. This enrichment helped contextualize the dataset to the real-world tourism environment in Sri Lanka and reduced the reliance on a single data source. By combining primary survey responses with secondary tourism-related information, the dataset achieved a balance between user-specific attributes and broader cultural and regional travel patterns.

All records were organized into a structured schema with well-defined fields. Categorical variables such as budget and travel style were encoded consistently, while numerical attributes such as age were stored in uniform formats for comparability. Free-text fields like interests and preferred destinations were maintained in a standardized manner, with attention to removing duplicates, spelling variations, and ambiguous entries. Additional metadata such as group status (active/inactive) and current member count were also included at this stage to support group formation processes later in development. The final dataset was compiled into CSV and JSON formats, ensuring that it was clean, organized, and ready for subsequent use in model training and system integration.



User_ID	Age	User_Interest	Preferred_Destination	Budget	Travel_Style
U0001	20	Cultural Tours	Gregory Lake, Community Tsunami Museum	Low	Balanced
U0002	22	Educational Tours, Rafting, Cultural Activities	Arugam Bay	Medium	Luxury
U0003	19	Animal Watching	Pinnawala Elephant Orphanage, Sea Turtle Farm Galle Mahamodara, Ceylon Tea Museum	Low	Backpacker
U0004	25	Yoga, Rafting, Boating	Colombo National Museum, Arugam Bay	Low	Adventurous
U0005	24	Kit Surfing, Swimming	Jaya Sri Maha Bodhi, Polonnaruwa, Ritigala Forest Monastery	Low	Relaxed
U0006	20	Architectural Marvel, Wildlife Spotting	Arugam Bay	Medium	Relaxed
U0007	23	Surfing, Animal Watching, Sunbathing	Lover's Leap Falls	High	Relaxed
U0008	19	Boating, Cultural Tours, Kite Surfing	Lover's Leap Falls, Handunugoda Tea Estate, Jaya Sri Maha Bodhi	Medium	Balanced
U0009	19	Nature Walks, Jet Skiing	Kelaniya Raja Maha Vihara	High	Balanced
U0010	18	Historical Exploration, Wildlife Safaris, Learning History	Udawalawe National Park	High	Balanced
U0011	21	Animal Watching, Fruit Picking	Bentota Beach	High	Luxury
U0012	20	Walking Trails	Temple of the Sacred Tooth Relic, Mihintale, Victoria Park of Nuwara Eliya	Medium	Luxury
U0013	23	Educational Tours, Swimming, Fruit Picking	Dambatenne Tea Factory	Low	Adventurous
U0014	18	Hiking, Yoga, Sunbathing	Horton Plains National Park, National Zoological Gardens of Sri Lanka, Wilpattu National Park	Medium	Adventurous
U0015	18	Kit Surfing	Ravana Ella Falls, Arugam Bay	High	Backpacker
U0016	24	Trekking	National Zoological Gardens of Sri Lanka	Low	Relaxed
U0017	24	Photography, Farm Tours, Rafting	Colombo National Museum, Ceylon Tea Museum, Kalametiya Eco Bird Watching	Medium	Luxury
U0018	23	Trekking, Yoga	Dambulla Cave Temple	Low	Backpacker
U0019	21	Yoga, Hiking	Negombo Beach, Gregory Lake	Low	Adventurous
U0020	18	Photography, Snorkeling	Kalametiya Eco Bird Watching, Hakgala Botanic Gardens	High	Adventurous
U0021	19	Kite Surfing, Worship, Animal Watching	Diyaluma Falls	High	Relaxed
U0022	22	Worship, Kite Surfing	Arugam Bay	Low	Balanced
U0023	19	Kite Surfing	Bentota Beach, Tissa Wewa	Low	Relaxed
U0024	18	Boating, Exploring Exhibits	Mount Lavinia Beach, Ramboda Waterfall, Mirissa Beach	Low	Relaxed
U0025	23	Trekking	Passikudah Beach, Temple of the Sacred Tooth Relic, Jethawanaramaya Stupa	High	Adventurous

Figure 13: A sample section of the curated user preference dataset



Group_ID	Group_Name	Status	Budget	Travel_Style	Destinations_Planned	Group_Interest	Current_Members
G0001	Radiant Nomads	Active	High	Relaxed	Wilpattu National Park, Sinharaja Forest Reserve, Diyaluma Falls	Kite Surfing, Agricultural Workshops	9
G0002	Adventurous Nomads	Inactive	Medium	Relaxed	Ravana Ella Falls, Samadhi Statue, Brief Garden - Bevis Bawa, Ritigala Forest Monastery	Hiking, Fruit Picking	0
G0003	Thrilling Questers	Active	High	Adventurous	Brief Garden - Bevis Bawa, Ceylon Tea Museum, Sigiriya The Ancient Rock Fortress, Pigeon Island National F	Swimming, Agricultural Workshops	6
G0004	Wanderlust Discoverers	Inactive	Low	Relaxed	Pedro Tea Factory, Pigeon Island National Park	Animal Watching, Swimming	0
G0005	Radiant Sojourners	Inactive	Medium	Luxury	Diyaluma Falls, Ariyapala Mask Museum, Brief Garden - Bevis Bawa	Cultural Tours, Hiking	0
G0006	Lively Wanderers	Inactive	Low	Balanced	Colombo National Museum, Pigeon Island National Park, World Buddhist Museum	Sightseeing, Hiking	0
G0007	Elite Pioneers	Active	Medium	Adventurous	Kandy Lake, Polonnaruwa	Tea Plucking, Rafting	2
G0008	Energetic Pathfinders	Active	High	Balanced	National Zoological Gardens of Sri Lanka, Marble Beach, Bundala National Park	Animal Watching, Fishing	10
G0009	Epic Drifters	Inactive	Low	Backpacker	Bluefield Tea Gardens, Royal Botanical Gardens	Educational Tours, Historical Explorati	0
G0010	Cheerful Wanderers	Inactive	High	Adventurous	Wilpattu National Park, Victoria Park of Nuwara Eliya, Mihintale	Fishing, Kite Surfing	0
G0011	Epic Pioneers	Active	High	Adventurous	Jaya Sri Maha Bodhi, New Ranweli Spice Garden, Udawalawe National Park	Sightseeing, Wildlife Safaris	7
G0012	Brave Travellers	Active	Medium	Backpacker	Ariyapala Mask Museum, Udawattekele Sanctuary, Udawalawe National Park, Pigeon Island National Park	Swimming, Boating	6
G0013	Relaxing Pioneers	Active	Medium	Backpacker	Marble Beach, Tissa Wewa, Baker's Falls	Nature Photography, Boating	1
G0014	Relaxing Sojourners	Active	Medium	Adventurous	Samadhi Statue	Hiking , Kit Surfing	2
G0015	Majestic Adventurers	Inactive	High	Balanced	Mount Lavinia Beach	Hiking , Meditation	0
G0016	Cultural Globetrotters	Active	Low	Backpacker	World Buddhist Museum, Samadhi Statue, Ramboda Waterfall, Community Tsunami Museum, Ruwanwelisa	Trekking, Animal Watching	10
G0017	Majestic Seekers	Inactive	Medium	Backpacker	Wilpattu National Park, Sea Turtle Farm Galle Mahamodara	Photography, Cultural Exploration	0
G0018	Explorer Adventurers	Inactive	Low	Backpacker	Abhayagiri Dagaba, Baker's Falls, Udawattekele Sanctuary	Animal Watching, Hiking	0
G0019	Vibrant Escapists	Active	High	Adventurous	Kandy Lake, St Clair's Falls, Nilaveli Beach	Animal Watching, Architectural Marvel	5
G0020	Energetic Escapists	Active	High	Relaxed	Jaffna Fort, Polonnaruwa, Dagoba of Thuparama, Samadhi Statue, Glenloch Tea Factory	Nature Walks, Surfing	8
G0021	Thrilling Dreamers	Active	Low	Balanced	Colombo National Museum, Koneswaram Temple, Ambewela Farms	Tea Plucking, Animal Watching	3
G0022	Majestic Dreamers	Active	Low	Relaxed	Twin Baths (Kuttam Pokuna), Gangaramaya (Vihara) Buddhist Temple	Learning History, Wildlife Spotting	8
G0023	Epic Travellers	Active	Medium	Backpacker	Sigiriya Museum, Galle Fort, Glenloch Tea Factory	Worship, Cultural Tours	4
G0024	Wanderlust Pioneers	Inactive	High	Luxury	Lover's Leap Falls, World Buddhist Museum, Ritigala Forest Monastery, Ravana Ella Falls, Arugam Bay	Educational Tours, Kit Surfing	0
G0025	Vibrant Explorers	Active	Low	Adventurous	Diyaluma Falls, Ariyapala Mask Museum, Brief Garden - Bevis Bawa	Architectural Marvel, Hiking	1

Figure 14: A sample section of the curated group details dataset

2.1.4.2 Data Preprocessing and Cleaning

Before the dataset could be used for model training and integration, it was subjected to a systematic preprocessing and cleaning phase to ensure completeness, consistency, and usability. The collected user and group data was first organized into clearly defined fields, including age, budget, travel style, interests, preferred destinations, current members, and group status. This structured arrangement allowed the dataset to be managed efficiently and made it suitable for subsequent transformation into machine-learning-ready formats.

The preprocessing workflow began by loading the dataset into Pandas, where duplicate entries, incomplete responses, and noisy records were identified and removed to minimize redundancy and errors. Textual fields such as User Interests and Preferred Destinations were standardized by applying lowercasing, removal of unnecessary characters, and normalization of terms to eliminate ambiguity (e.g., mapping “hiking” and “trekking” to a single category). This ensured linguistic clarity and reduced variability within free-text responses.

Categorical variables, such as Budget and Travel Style, were then label-encoded to convert them into numerical form, while numerical attributes like Age and Current Members were standardized through feature scaling to ensure uniform contribution during model training. By completing these preprocessing and cleaning steps, the dataset was transformed into a clean, reliable, and structured resource that could be directly utilized for semantic embedding, recommendation ranking, and similarity-based group creation. This preparation ensured that all subsequent machine learning components operated on consistent and meaningful inputs.

Figure 15: Code snippet showing the preprocessing and data cleaning

2.1.4.3 Model Selection and Training

Model selection and training were central to enabling the system's intelligent group recommendation capability. After considering several alternatives, Sentence-BERT (SBERT) was selected for generating semantic embeddings of user interests and preferred destinations due to its proven performance in capturing contextual meaning in natural language. For the ranking component, LightGBM LambdaRank was chosen as it is specifically optimized for learning-to-rank tasks and performs efficiently with structured input features.

In the implemented pipeline, SBERT embeddings provided semantic representations of textual fields such as *User Interests* and *Preferred Destinations*, while LightGBM LambdaRank combined these embeddings with encoded categorical and numerical features such as *Budget*, *Travel Style*, *Age*, and *Current Member Count*. This integration allowed the model to predict group compatibility scores more accurately. The prepared dataset was used to train the LambdaRank model, with Normalized Discounted Cumulative Gain (NDCG@k) applied as the evaluation metric to optimize ranking quality. Once trained, the models were serialized using joblib and deployed within the backend, enabling real-time inference during recommendation requests.

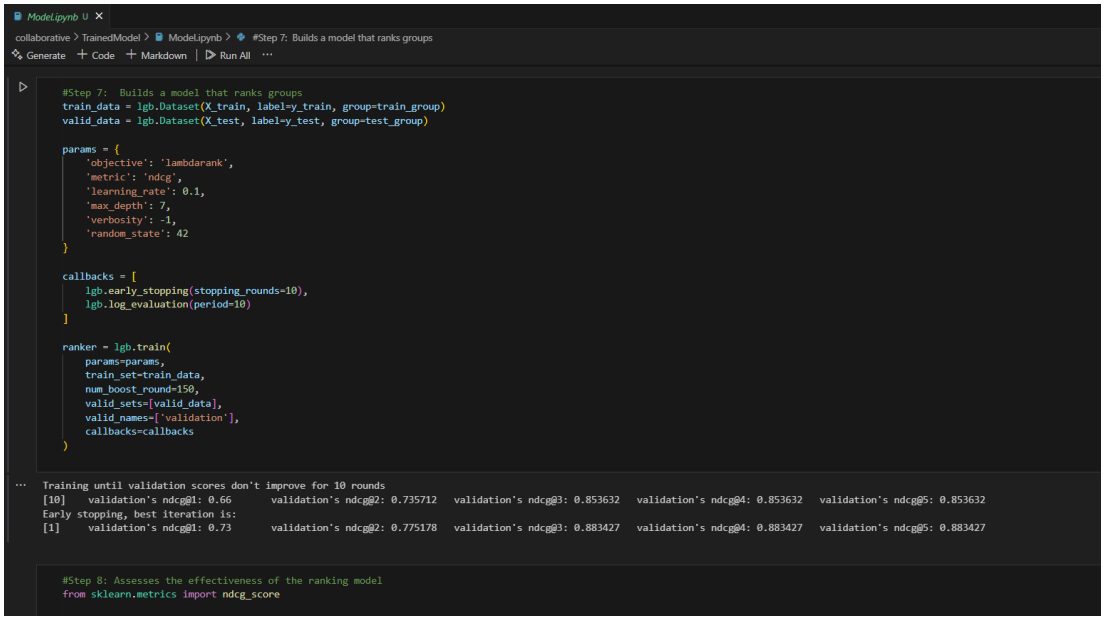


Figure 16: Code snippet showing the model training

2.1.4.4 Embedding Generation and Similarity Computation

Embedding generation formed a critical step in enabling the system to compute semantic similarity between user profiles and existing group characteristics. The platform employed the SentenceTransformer model (all-MiniLM-L6-v2) to convert each user interest and preferred destination into dense, fixed-length vectors that captured semantic meaning beyond simple keyword matching. Group profiles stored in MongoDB were processed in a similar manner by embedding the aggregated interests and destinations of their members.

Once embedded, cosine similarity was applied to measure the closeness between a new user's profile and existing groups. This similarity score served two primary purposes: (1) it contributed to the ranking process by supporting the LightGBM LambdaRank model in recommending the most relevant groups, and (2) it acted as a threshold-based decision mechanism for dynamic group creation, where a new group was generated if no existing group exceeded the similarity threshold. This process ensured that recommendations remained both semantically meaningful and inclusive, supporting continuous onboarding of new users.

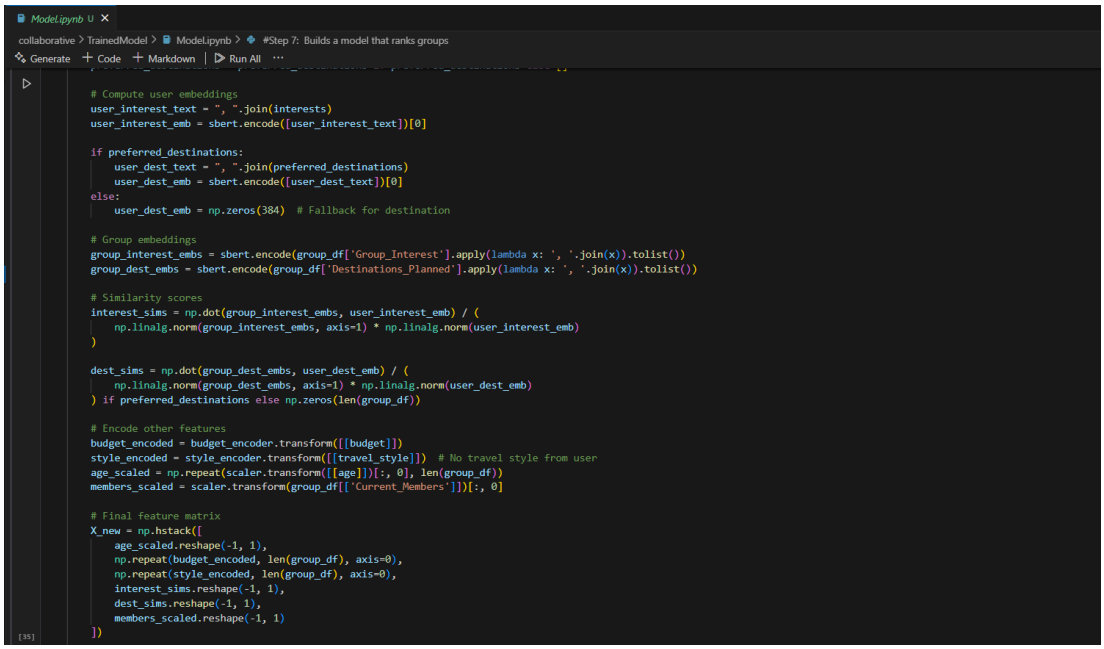
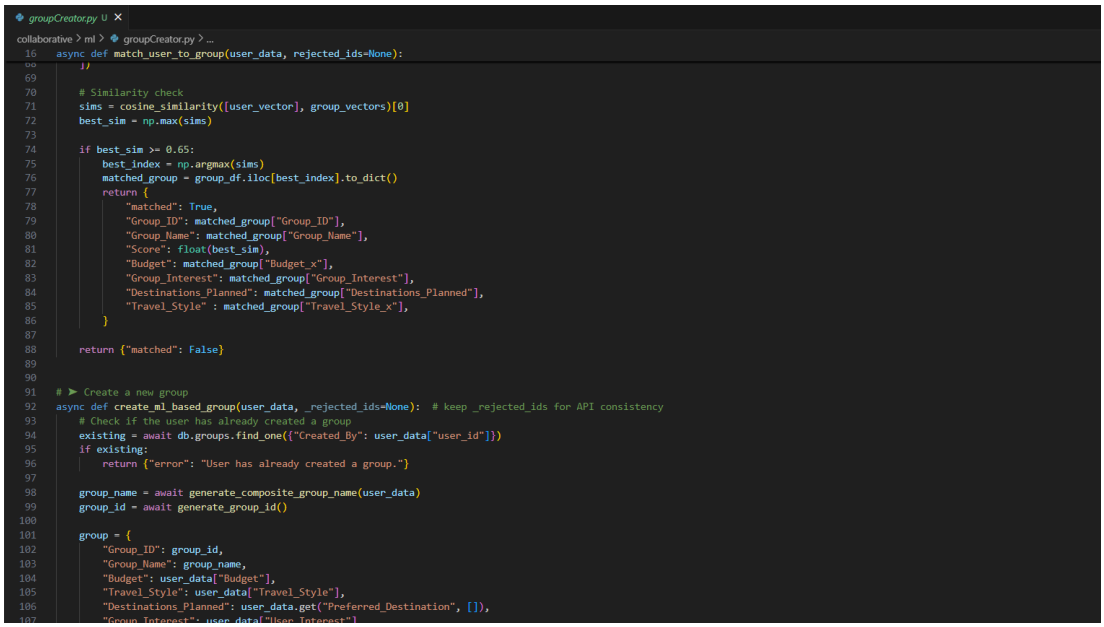


Figure 17: Code snippet showing embedding generation and similarity computation

2.1.4.5 Group Creation Logic with Thresholding

To ensure inclusivity of all users, the platform incorporates a dynamic group creation module that activates whenever recommendation scores for a given user fall below the defined similarity threshold. In such cases, the system automatically generates a new group, assigning it a composite name derived from the user's budget category, travel style, and primary interest. Before confirming group creation, cosine similarity is applied to verify that no existing group with highly similar attributes already exists, thereby avoiding redundancy and unnecessary duplication.

Each newly created group is stored in MongoDB with an initial status of *Inactive*. This status ensures that groups are only activated once at least two members have joined, at which point the group transitions to *Active* and becomes available for collaboration. This threshold-based logic guarantees that users are never left unpaired, while also supporting continuous onboarding into the system without requiring manual intervention.



```python
async def match_user_to_group(user_data, rejected_ids=None):
    ])

    # Similarity check
    sims = cosine_similarity([user_vector], group_vectors)[0]
    best_sim = np.max(sims)

    if best_sim >= 0.65:
        best_index = np.argmax(sims)
        matched_group = group_df.iloc[best_index].to_dict()
        return {
            "matched": True,
            "Group_ID": matched_group["Group_ID"],
            "Group_Name": matched_group["Group_Name"],
            "Score": float(best_sim),
            "Budget": matched_group["Budget_x"],
            "Group_Interest": matched_group["Group_Interest"],
            "Destinations_Planned": matched_group["Destinations_Planned"],
            "Travel_Style" : matched_group["Travel_Style_x"],
        }

    return {"matched": False}


# ➤ Create a new group
async def create_ml_based_group(user_data, _rejected_ids=None):  # keep _rejected_ids for API consistency
    # Check if the user has already created a group
    existing = await db.groups.find_one({"Created_By": user_data["user_id"]})
    if existing:
        return {"error": "User has already created a group."}

    group_name = await generate_composite_group_name(user_data)
    group_id = await generate_group_id()

    group = {
        "Group_ID": group_id,
        "Group_Name": group_name,
        "Budget": user_data["Budget"],
        "Travel_Style": user_data["Travel_Style"],
        "Destinations_Planned": user_data.get("Preferred_Destination", []),
```

Figure 18: Code snippet showing group creation logic with thresholding

2.1.4.6 Real-Time Chat Engine

The real-time collaboration environment of the platform is facilitated by a WebSocket-based chat service developed using FastAPI. Once a group reaches active status, all members can exchange messages instantly, enabling seamless group communication. Each message persisted in MongoDB, ensuring that chat history is available for retrieval to support continuity across sessions and facilitate new member onboarding.

Additional features such as typing indicators, persistent message storage, and onboarding summaries further enhance the chat experience. By combining these elements, the chat engine not only enables synchronous communication but also establishes the foundation for Trip Bot's integration, as it forwards user messages to the intent classifier and orchestrates the delivery of AI-generated responses within the group conversation.

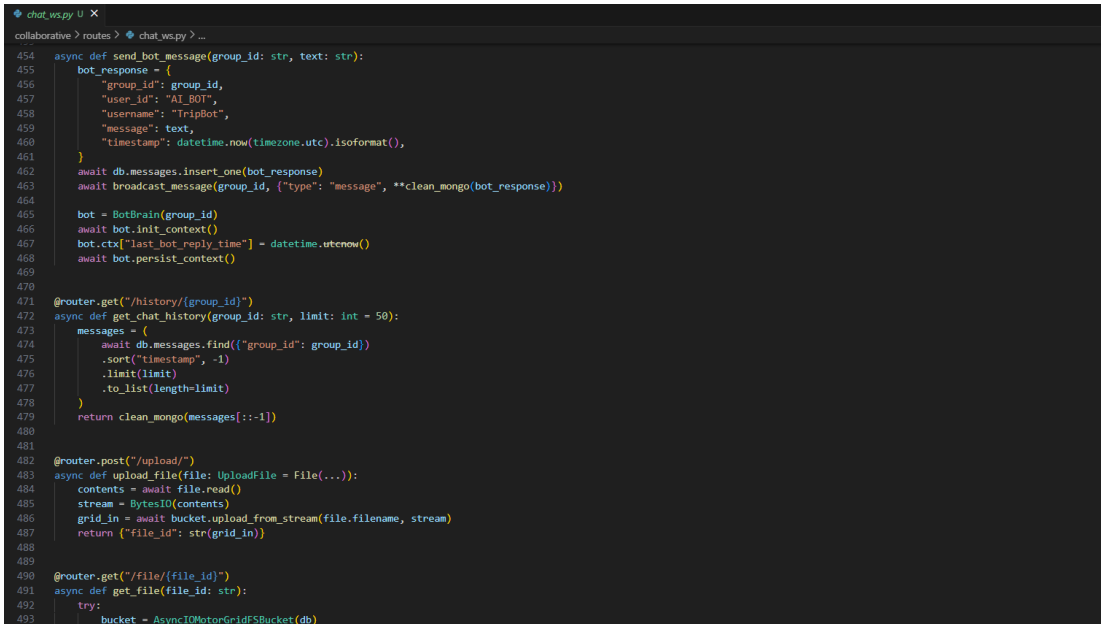


Figure 19: Code snippet showing API for real time chat engine

2.1.4.7 Trip Bot Assistant with Intent Classification

A key innovation of the platform is the integration of Trip Bot, an AI-powered assistant embedded within the group chat service to support collaborative travel planning. Trip Bot employs an SBERT-based intent classifier to analyse user messages and categorize them into intents such as *trip planning*, *requesting assistance*, *sharing experiences*, or *casual greeting*. Based on the detected intent, Trip Bot provides relevant support, which may include contextual suggestions, summarization of ongoing group activities, or the drafting of provisional travel plans.

For more advanced travel-related queries, Trip Bot connects to the Google Gemini API, which provides enriched information and recommendations based on real-world data. These responses are seamlessly integrated into the group chat, ensuring that users receive contextual and actionable insights within their collaborative space. All bot interactions are logged in MongoDB to maintain transparency, support traceability, and provide summaries for later retrieval.

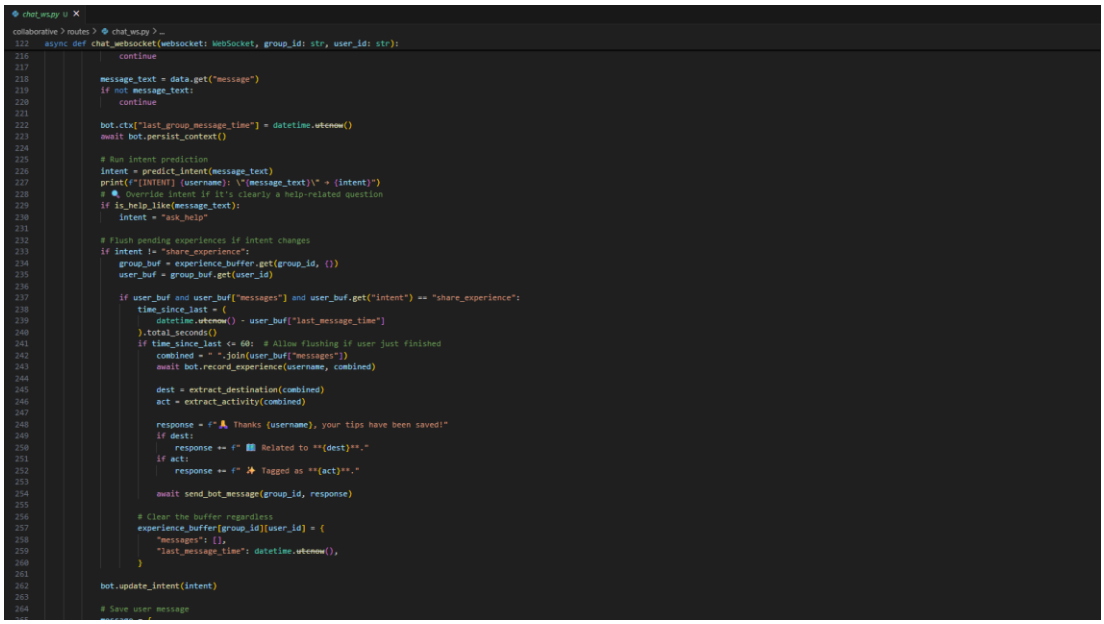


Figure 20: Code snippet showing trip bot assistant with intent classification

2.1.4.8 Frontend Development with React and Tailwind CSS

The frontend of the collaborative travel companion platform was developed using React.js, a widely adopted JavaScript library for building user interfaces, in combination with Tailwind CSS, a utility-first CSS framework. This combination was chosen to provide a highly responsive, modular, and maintainable design that adapts seamlessly across desktops, tablets, and mobile devices.

The frontend architecture follows a component-based modular structure, where each major functionality is encapsulated in a dedicated file or component. For instance, the RecommandPage.tsx component is responsible for displaying personalized group recommendations, allowing users to view ranked groups, check compatibility scores, and trigger the creation of new groups when no suitable matches are found. Similarly, the ChatPage.tsx component manages the real-time group communication interface, including WebSocket-based message exchange, persistence of chat history, and the integration of TripBot, the AI assistant, which provides contextual responses within the chat flow. The LoginPage.jsx ensures user authentication and smooth entry into the platform.

Routing and navigation across the application is handled by React Router, which supports client-side navigation between pages without requiring a full reload. This improves performance and enables a single-page application (SPA) experience, where state is preserved between navigations. State management within components is primarily handled through React hooks such as useState and useEffect. For asynchronous interactions with the backend (FastAPI), REST API calls are used for operations like registration and recommendation retrieval, while WebSockets handle bidirectional communication for the chat engine. This hybrid communication strategy ensures both reliability (through REST) and responsiveness (through WebSockets).

Tailwind CSS was used extensively to enforce a consistent, responsive, and modern design language across the platform. By adopting utility-first CSS classes, the development process avoided verbose CSS files and achieved rapid prototyping. Tailwind's responsive utilities ensured that recommendation cards, chat interfaces, and dashboards automatically adjusted to different screen sizes without additional media

queries. This responsiveness was validated during manual testing across multiple browsers and mobile devices.

Several interactive elements were designed to improve usability and clarity. Recommendation results are presented in the form of cards, each displaying the group name, budget category, travel style, member count, and compatibility score. Visual indicators such as *Active/Inactive status labels* provide immediate context about group availability. The chat interface includes message bubbles, timestamps, typing indicators, and file-sharing functionality to simulate a familiar messaging environment. TripBot messages are distinguished with a unique styling, allowing users to easily differentiate between AI suggestions and human inputs.

From a development perspective, the frontend codebase maintained a clear folder and component structure, separating presentation logic (UI components) from business logic (API calls and WebSocket handlers). Git version control ensured team collaboration, and code consistency was enforced through ESLint and Prettier. This structured approach not only improved maintainability but also aligned with scalability requirements, ensuring that new features such as booking integration or analytics dashboards can be added without significant rework.

In summary, the frontend implementation provided a responsive, intuitive, and interactive interface that tightly integrates with the backend services. By combining React's modular architecture with Tailwind's responsive design capabilities, the platform successfully delivered a user experience that supports both functional reliability and aesthetically pleasing design, meeting the usability goals of the project.

```
  return (
    <div className="max-w-2xl mx-auto p-4">
      <h1 className="text-2xl font-bold mb-6">Welcome, here are your groups</h1>

      {/* Joined Groups */}
      {joinedGroups.length > 0 && (
        <div className="mb-8">
          <h2 className="text-xl font-semibold mb-3">Your Joined Groups</h2>
          {joinedGroups.map((group) => (
            <div key={group.Group_ID} className="border p-4 rounded shadow mb-4 bg-green-50">
              <p><strong>Group Name:</strong> {group.Group_Name}</p>
              <p><strong>Status:</strong> {group.Status}</p>
              <p><strong>Destinations:</strong> {group.Destinations_Planned.join(", ")}</p>
              <button
                disabled={group.Status !== "Active"}
                onClick={() =>
                  navigate("/chat", {
                    state: {
                      group_id: group.Group_ID,
                      user_id: userId,
                    },
                  })
                }
                className={`mt-2 px-4 py-2 rounded text-white ${group.Status === "Active"
                  ? "bg-green-600 hover:bg-green-700"
                  : "bg-gray-400 cursor-not-allowed"
                  }`}
              >
                {group.Status === "Active" ? "Go to Chat" : "Waiting for Members"}
              </button>
            </div>
          ))}
        </div>
      )}

      {/* Recommended Groups */}
      <h2 className="text-xl font-semibold mb-2">Recommended Groups</h2>
      {loading ? (
        <p>Loading...</p>
      ) : (
        <>
          {groups.length > 0 ? (
            groups.map((group) => (
              <div key={group.Group_ID} className="border p-4 rounded shadow mb-4">
                <p><strong>Group Name:</strong> {group.Group_Name}</p>
                <p><strong>Interests:</strong> {group.Group_Interest.join(", ")}</p>
                <p><strong>Destinations:</strong> {group.Destinations_Planned.join(", ")}</p>
                <p><strong>Budget:</strong> {group.Budget}</p>
                <p><strong>Travel Style:</strong> {group.Travel_Style}</p>
                <button
                  className="mt-2 px-4 py-2 bg-blue-600 text-white rounded hover:bg-blue-700"
                  onClick={() => handleJoin(group.Group_ID)}
                >
                  Join Group
                </button>
              </div>
            ))
          ) : (
            <p>No recommendations found.</p>
          )}
        </>
      )}

      {/* Actions */}
      {!loading && (
        <div className="text-center mt-6 space-x-4">
          <button
            onClick={handleRecommendMore}
            className="px-4 py-2 bg-gray-600 text-white rounded hover:bg-gray-700"
          >
            Recommend More
          </button>
          {!groupCreated && (
            <button
              onClick={handleCreateGroup}
              className="px-4 py-2 bg-green-600 text-white rounded hover:bg-green-700"
            >
              Create New Group
            </button>
          )}
        </div>
      )}

      {waiting && statusMessage && (
        <div className="mt-4 p-3 bg-yellow-100 border border-yellow-400 text-yellow-800 rounded">
          {statusMessage}
        </div>
      )}
    </div>
  );
}
```

Figure 21: Code snippet showing Recommend group Page

```
return (
    <div className="max-w-2xl mx-auto p-4 h-screen flex flex-col bg-gray-100">
      <h2
        className="text-2xl font-bold text-center
              bg-gradient-to-r from-blue-500 to-indigo-600
              text-white p-4 rounded-2xl shadow-lg mb-6
              flex items-center justify-center gap-2
              transition-all duration-300 hover:shadow-xl hover:scale-[1.02]"
      >
        <span className="text-2xl">✈️</span>
        Trip Planner Chat
      </h2>
      <div className="flex justify-between items-center mb-4">
        <h2 className="text-2xl font-semibold text-gray-800">{groupName}</h2>

        <button
          onClick={() => setShowExperiencePanel(true)}
          className="flex items-center gap-2 px-3 py-1.5 bg-blue-100 text-blue-700 rounded-lg
              shadow-sm transition-all duration-200 hover:bg-blue-600 hover:text-white hover:shadow-
md"
        >
          <span className="text-lg">📝</span>
          <span className="text-sm font-medium">View Tips</span>
        </button>
      </div>

      <div className="flex-1 overflow-y-auto border p-4 rounded mb-4 bg-white">
        {typingUser && (
          <div className="text-sm italic text-gray-500 mb-2">
            {typingUser} is typing...
          </div>
        )}
        {messages.map((msg, i) => {
          const currentDateLabel = formatDateHeader(msg.timestamp);
          const showDateHeader = currentDateLabel !== lastDateLabel;
          lastDateLabel = currentDateLabel;

          return (
            <React.Fragment key={i}>
              {showDateHeader && (
                <div className="text-center text-xs font-semibold text-gray-500 my-4">
                  <span className="inline-block bg-gray-200 text-gray-700 px-3 py-1 rounded-full">
                    {currentDateLabel}
                  </span>
                </div>
              )}

              <div
                className={`mb-3 flex ${
                  msg.user_id === user_id ? "justify-end" : "justify-start"
                }`}
              >
                <div
                  className={`relative group max-w-[75%] p-3 rounded-xl shadow text-sm whitespace-pre-
wrap break-words ${
                    msg.user_id === user_id
                      ? "bg-blue-500 text-white"
                      : "bg-gray-200 text-black"
                  }`}
                >
                  <div className="font-medium mb-1">{msg.username}</div>
                  {msg.media_type && msg.media_id ? (
                    msg.media_type === "image" ? (
                      <img
                        src={`${serverUrl}/file/${msg.media_id}`}
                        alt="uploaded"
                        className="rounded mt-2 max-h-60 object-cover"
                      />
                    ) : msg.media_type === "video" ? (
                      <video
                        controls
                        src={`${serverUrl}/file/${msg.media_id}`}
                        className="mt-2 rounded w-full max-h-60"
                      />
                    ) : msg.media_type === "audio" ? (
                      <audio
                        controls
                        src={`${serverUrl}/file/${msg.media_id}`}
                        className="mt-2 w-full"
                      />
                    ) : (
                      <a
                        href={`${serverUrl}/file/${msg.media_id}`}
                        target="_blank"
                        rel="noopener noreferrer"
                        className="text-blue-600 underline"
                      >
                        📎 Download {msg.filename || "file"}
                      </a>
                    )
                  ) : (
                    <div>{msg.message}</div>
                  )}
                  <div className="text-xs text-right mt-1 text-gray-300">
                    {new Date(msg.timestamp).toLocaleTimeString(undefined, {
                      hour: "2-digit",
                      minute: "2-digit",
                    })}
                  </div>

                  <div className="absolute -top-4 right-2 hidden group-hover:flex gap-1 text-lg">
                    {["❤️", "😂", "👍"].map((emoji) => (
                      <button
                        key={emoji}
                        onClick={() => addReaction(i, emoji)}
                        className="hover:scale-110 transition-transform"
                      >
                        {emoji}
                      </button>
                    ))}
                  </div>
```

Figure 22: Code snippet showing Chat Page

2.1.5 Testing

The testing process undertaken for the collaborative travel companion platform followed a structured and layered approach, including unit testing, integration testing, system testing, and acceptance testing. These phases were designed to validate the stability, reliability, and user experience of the platform under realistic group travel planning scenarios. This comprehensive approach ensured that the platform's machine learning pipeline, group formation logic, and real-time collaboration features functioned seamlessly as an integrated system.

Unit Testing

Unit testing formed the foundation of the testing strategy. Python's unittest and pytest frameworks were used to create test suites targeting individual backend functions. Critical functions tested included the SBERT-based embedding generator, LightGBM LambdaRank scoring logic, cosine similarity checks for group creation, and MongoDB CRUD operations for user and group management. Example functions such as recommend_groups_for_user, create_ml_based_group and FastAPI routes for registration and group joining were tested with a range of valid and invalid inputs. These tests ensured that the system's core logic consistently produced accurate results across edge cases.

Integration Testing

Integration testing was conducted to validate the interaction between the recommendation module, group creation logic, and the frontend interfaces. Tools such as Postman were used to test API endpoints, verifying correct data flow between the backend (FastAPI) and the database (MongoDB). Scenarios tested included user registration, generating recommendations, joining or creating groups, and initiating WebSocket chat sessions. Additionally, the integration of TripBot with the chat service was tested by simulating user messages and verifying that the intent classifier produced correct labels and that Gemini API calls returned contextual replies. These tests ensured that all modules communicated correctly and maintained consistent response formats.

System Testing

System testing was carried out on the full deployment of the platform, including backend services, frontend interfaces (RecommendPage, ChatPage), and the WebSocket-based chat engine. Complete workflows were validated, such as a user registering, receiving recommendations, joining a group, engaging in chat, and receiving TripBot assistance. System tests also confirmed that group activation logic worked as expected (groups remained *Inactive* until two members joined) and that chat history had persisted and retrievable from MongoDB. Testing was performed across multiple browsers and devices to ensure consistent performance and responsiveness.

Acceptance Testing

Acceptance testing was performed with real users drawn from the target demographic, including university students and frequent travellers. Alpha testing was conducted in a controlled environment, where participants tested core features such as group recommendations, group creation, and real-time chat. Feedback helped identify minor usability issues, such as unclear error messages during group joining and response delays when calling the Gemini API. Following refinements, beta testing allowed users to access the system in real-world conditions. Feedback from these sessions confirmed that recommendations were relevant, chat interactions were smooth, and TripBot's responses were both helpful and contextually appropriate.

By following this layered and comprehensive testing strategy, the platform was thoroughly evaluated for performance, stability, and usability. This approach not only validated the quality of individual components but also ensured the smooth operation of the overall system, reinforcing its readiness for deployment as a real-world group travel planning solution.

2.1.5.1 Unit Testing

To validate the correctness and reliability of the collaborative travel companion platform, a comprehensive unit testing strategy was implemented using pytest. The primary focus of unit testing was to verify the internal logic of critical components while isolating external dependencies such as the database and third-party APIs.

Unit tests were written for several key areas of the system. For the recommendation module tests verified the SBERT-based embedding generation and LightGBM LambdaRank scoring pipeline, ensuring that group compatibility scores were returned in the correct order and format. The group creation logic was tested to confirm that similarity thresholds triggered the expected behaviour: either matching a user to an existing group or dynamically creating a new group with the correct attributes. For the user and group management routes FastAPI's TestClient was used to simulate POST requests to endpoints. These tests validated correct request/response handling, ID generation, and status transitions (e.g., groups switching from *Inactive* to *Active* when at least two members joined).

Additional unit tests targeted the real-time chat module and the TripBot assistant. For the chat WebSocket, tests simulated two users joining the same group and verified that messages were broadcast correctly and persisted in the database. TripBot was tested with mocked inputs to confirm that the SBERT-based intent classifier produced the correct labels (e.g., plan_trip, ask_help, share_experience, greet) and that Gemini API calls could be stubbed with fallback responses. MongoDB operations were isolated using mongomock, enabling tests to run without a live database. Each unit test was designed to assert both happy paths (valid inputs and expected outputs) and edge cases (duplicate users, non-existent groups, or slow/failed API calls).

This structured approach ensured that regressions were detected early, core logic remained robust, and performance-critical functions behaved consistently across updates. By combining isolated function testing with simulated API calls, the platform's backend achieved a high level of reliability before progressing to higher levels of integration and acceptance testing.

test_groups.py U

collaborative > pytest > test_groups.py > test_create_group

```python
from fastapi.testclient import TestClient

def test_create_group(client: TestClient):
    payload = {
        "Group_Name": "Test Group",
        "Budget": "Medium",
        "Travel_Style": "Adventure",
        "Destinations_Planned": ["Kandy"],
        "Group_Interest": ["Culture"],
        "Current_Members": 0,
        "Status": "Inactive"
    }
    r = client.post("/groups/createGroup", json=payload)
    assert r.status_code in (200, 201)
    data = r.json()
    assert "Group_ID" in data

def test_join_group(client: TestClient):
    # group and user IDs may vary depending on DB state
    r = client.post("/groups/join/G0001/U0001")
    assert r.status_code in (200, 202, 404, 409)
    if r.status_code in (200, 202):
        data = r.json()
        assert data["message"] in ["User added to group", "User already in group"]

def test_get_group_details(client: TestClient):
    r = client.get("/groups/getGroupDetails/G0001")
    assert r.status_code in (200, 404)
    if r.status_code == 200:
        data = r.json()
        assert "Group_ID" in data
        assert data["Group_ID"] == "G0001"

def test_get_all_groups(client: TestClient):
    r = client.get("/groups/getAllGroups")
    assert r.status_code == 200
    data = r.json()
```

Figure 23: test_groups.py

test_recommend.py U

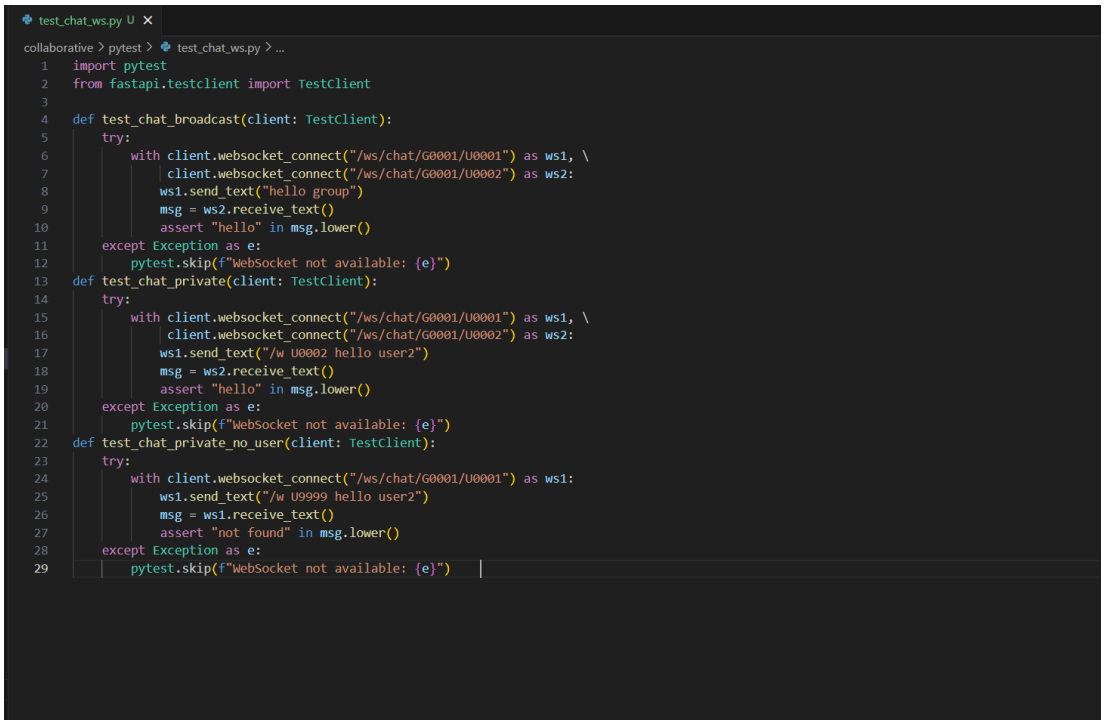
collaborative > pytest > test_recommend.py > ...

```python
def test_recommend(client, monkeypatch):
        def fake_recommend_groups_for_user(*args, **kwargs):
            return [{"Group_ID": "G0001", "Score": 0.95}]
        monkeypatch.setattr(recommendation, "recommend_groups_for_user", fake_recommend_groups_for_user)
    except ImportError:
        pass

    payload = {
        "Age": 25,
        "Budget": "Medium",
        "Travel_Style": "Adventure",
        "User_Interest": ["Hiking"],
        "Preferred_Destination": ["Ella"]
    }
    r = client.post("/recommend/", json=payload)
    assert r.status_code == 200
    data = r.json()
    assert "recommendations" in data
    assert isinstance(data["recommendations"], list)

def test_recommend_invalid_input(client):
    payload = {
        "Age": -5,  # Invalid age
        "Budget": "Unknown",  # Invalid budget
        "Travel_Style": "Adventure",
        "User_Interest": ["Hiking"],
        "Preferred_Destination": ["Ella"]
    }
    r = client.post("/recommend/", json=payload)
    assert r.status_code == 422  # Unprocessable Entity for validation errors
def test_recommend_no_recommendations(client, monkeypatch):
    # Mock the recommender function to return no recommendations
    try:
        import recommendation
        def fake_recommend_groups_for_user(*args, **kwargs):
            return []
```

Figure 24: test_recommend.py



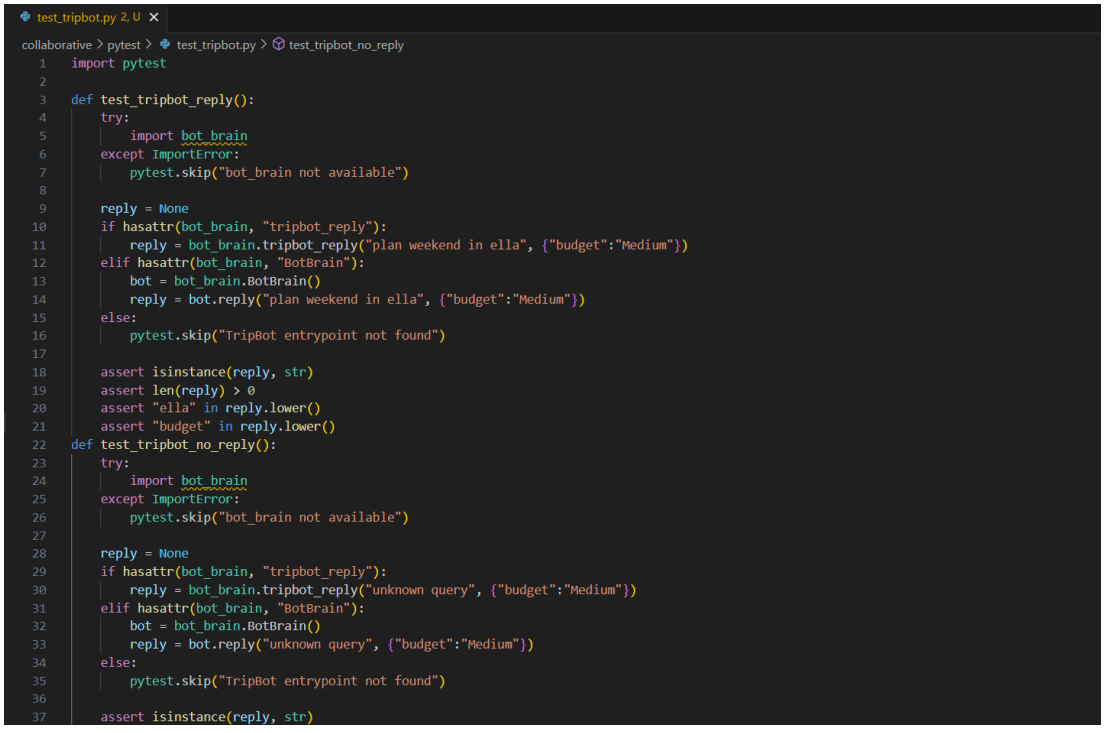
test_chat_ws.py U

collaborative > pytest > test_chat_ws.py > ...

```python
import pytest
from fastapi.testclient import TestClient

def test_chat_broadcast(client: TestClient):
    try:
        with client.websocket_connect("/ws/chat/G0001/U0001") as ws1, \
             client.websocket_connect("/ws/chat/G0001/U0002") as ws2:
            ws1.send_text("hello group")
            msg = ws2.receive_text()
            assert "hello" in msg.lower()
    except Exception as e:
        pytest.skip(f"WebSocket not available: {e}")
def test_chat_private(client: TestClient):
    try:
        with client.websocket_connect("/ws/chat/G0001/U0001") as ws1, \
             client.websocket_connect("/ws/chat/G0001/U0002") as ws2:
            ws1.send_text("/w U0002 hello user2")
            msg = ws2.receive_text()
            assert "hello" in msg.lower()
    except Exception as e:
        pytest.skip(f"WebSocket not available: {e}")
def test_chat_private_no_user(client: TestClient):
    try:
        with client.websocket_connect("/ws/chat/G0001/U0001") as ws1:
            ws1.send_text("/w U9999 hello user2")
            msg = ws1.receive_text()
            assert "not found" in msg.lower()
    except Exception as e:
        pytest.skip(f"WebSocket not available: {e}")
```

Figure 25: test_chat_ws.py



test_tripbot.py 2, U

collaborative > pytest > test_tripbot.py > test_tripbot_no_reply

```python
import pytest

def test_tripbot_reply():
    try:
        import bot_brain
    except ImportError:
        pytest.skip("bot_brain not available")

    reply = None
    if hasattr(bot_brain, "tripbot_reply"):
        reply = bot_brain.tripbot_reply("plan weekend in ella", {"budget":"Medium"})
    elif hasattr(bot_brain, "BotBrain"):
        bot = bot_brain.BotBrain()
        reply = bot.reply("plan weekend in ella", {"budget":"Medium"})
    else:
        pytest.skip("TripBot entrypoint not found")

    assert isinstance(reply, str)
    assert len(reply) > 0
    assert "ella" in reply.lower()
    assert "budget" in reply.lower()
def test_tripbot_no_reply():
    try:
        import bot_brain
    except ImportError:
        pytest.skip("bot_brain not available")

    reply = None
    if hasattr(bot_brain, "tripbot_reply"):
        reply = bot_brain.tripbot_reply("unknown query", {"budget":"Medium"})
    elif hasattr(bot_brain, "BotBrain"):
        bot = bot_brain.BotBrain()
        reply = bot.reply("unknown query", {"budget":"Medium"})
    else:
        pytest.skip("TripBot entrypoint not found")

    assert isinstance(reply, str)
```

Figure 26: test_tripbot.py

2.1.5.2 Manual Testing

Manual testing is a critical stage of validating the collaborative travel companion platform, ensuring that all components function correctly when interacted with by real users. This form of testing involved human testers manually executing various test scenarios without automation, simulating real traveller behaviour across different parts of the system. It served to catch usability issues, logical errors, and inconsistencies that automated scripts may not detect.

Purpose of Manual Testing

The primary goal of manual testing in this context was to verify the functionality, usability, and consistency of the system through hands-on evaluation. This helped confirm that the platform met the needs of travellers by recommending suitable groups, supporting smooth onboarding, enabling real-time collaboration, and providing actionable AI-driven feedback through TripBot. Manual testers interacted with the application using both desktop and mobile environments, replicating various user journeys such as registering, viewing recommendations, joining or creating groups, chatting with members, and requesting TripBot assistance.

Test Case Development and Execution

Test cases were derived from the system's core functional requirements and user stories. Each case was documented with test steps, expected outcomes, and actual behaviour, recorded using structured templates. Testing particularly focused on the frontend's interaction with the backend, including profile registration, group recommendation, group creation, WebSocket chat, and TripBot responses. Each test was executed across multiple devices and browsers in several rounds to confirm repeatability and reliability.

Types of Manual Testing Applied

- Functional Testing: Verifying that registration, recommendation, group creation, chat, and TripBot features work as intended.
- Usability Testing: Ensuring travellers could easily navigate and use the platform without guidance.
- Exploratory Testing: Allowing testers to freely explore the system and uncover unexpected behaviours.
- User Acceptance Testing (UAT): Having real students and frequent travellers interact with the platform and provide feedback.
- Compatibility Testing: Checking consistent performance across different devices (PC, Android, iOS) and browsers.

Below are the test cases which were done for the manual testing. Tables 4, 5, 6, 7, 8, 9 and 10 display the manual test cases.

Table 4: Test case for view group recommendations

Test Case ID	TC_01
Test Case Objective	View Group Recommendations
Pre-Requirements	User profile exists
Test Steps	**4.** Login with registered user. **5.** Navigate to "Recommended Groups" page.
Test Data	User profile and preferences
Expected Output	Display of ranked groups with compatibility scores.
Actual Output	Groups shown correctly with sorted order.
Status	Pass

Table 5: Test case for new group creation

Test Case ID	TC_02
Test Case Objective	Create New Group (No Match Found)
Pre-Requirements	User found not matching existing groups
Test Steps	1. Login with registered user. 2. Navigate to "Recommended Groups" page. 3. Asked to create a new group if matching groups not found.
Test Data	User profile and preferences
Expected Output	New group created, marked as *Inactive* until another user joins.
Actual Output	Group auto-created and displayed in user's dashboard
Status	Pass

Table 6: Test case for at least one group exist

Test Case ID	TC_03
Test Case Objective	At least one group exists in database
Pre-Requirements	User profile exists
Test Steps	1. Login with registered user. 2. Navigate to "Recommended Groups" page. 3. Select a suggested group and click "Join."
Test Data	User profile and preferences
Expected Output	User added to group. Group status flips to *Active* once two members exist.
Actual Output	Status updated and reflected in UI.
Status	Pass

Table 7: Test case for real time chat

Test Case ID	TC_04
Test Case Objective	Real-Time Chat
Pre-Requirements	Group is active.
Test Steps	1. Login with registered user. 2. Open group chat window. 3. Send message
Test Data	User profile and preferences
Expected Output	Messages broadcast instantly to all group members; history logged.
Actual Output	Chat worked correctly with persistence.
Status	Pass

Table 8: Test case for tripbot assistance

Test Case ID	TC_05
Test Case Objective	TripBot Assistance
Pre-Requirements	Group is active with TripBot enabled.
Test Steps	1. Send a planning query 2. Send a help request
Test Data	Message
Expected Output	TripBot provides contextual suggestions; Gemini API used for external info
Actual Output	Responses relevant, logged, and visible in chat
Status	Pass

Table 9: Test case for cross device compatibility

Test Case ID	TC_06
Test Case Objective	Cross-Device Compatibility
Pre-Requirements	Group exists.
Test Steps	1. Access platform via desktop and mobile browser. 2. Perform login → recommendation → chat sequence.
Test Data	Desktop(chrome), Mobile (Android, chrome)
Expected Output	Responsive layout, no broken UI.
Actual Output	Layout consistent across devices.
Status	Pass

Table 10: Test case for file upload in chat

Test Case ID	TC_07
Test Case Objective	File Upload in Chat
Pre-Requirements	Group is active and chat is open.
Test Steps	1. Login as a user who is a member of an active group. 2. Open the group chat window. 3. Click the “Attach File” or “Upload” button. 4. Select travel_itinerary.pdf from the local device. 5. Send/upload the file in chat.
Test Data	travel_itinerary.pdf file
Expected Output	File successfully uploaded and visible in chat as a clickable/downloadable link. Other group members receive the file instantly in their chat windows.
Actual Output	File appeared in chat for all group members and was retrievable.
Status	Pass

2.1.6 Deployment & Maintenance

Deployment (Azure)

Deploying the collaborative travel companion platform on Microsoft Azure follows a cloud-native, container-first approach that cleanly separates the frontend (React), backend (FastAPI + WebSockets), and data services (MongoDB). The flow below reflects best practices for reliability, security, and horizontal scalability.

1. Azure Account & Resource Group: Create an Azure subscription with university email account and a dedicated Resource Group to logically isolate all production resources.
2. Containerization (Backend): Package the FastAPI backend (including WebSocket endpoints, SBERT encoder, LightGBM model files, and TripBot integration) using Docker.
    - Expose the service on port 8000.
    - Keep model artifacts (e.g., trained_lgbm_ranker_semantic.txt, encoders) inside the image or mount via Azure Files.
    - Set environment variables at runtime (do not bake secrets):
        - MONGODB_URI
        - GEMINI_API_KEY (TripBot external API)
        - MODEL_PATHS (encoders/scaler/model)
        - ALLOWED_ORIGINS (CORS)
3. Container Registry: Push the backend image to Azure Container Registry (ACR) (e.g., acrtripplatform.azurecr.io). Enable admin pull for App Service or use managed identity.
4. Database: Use MongoDB Atlas for user profiles, groups, and chat logs.
    - Create separate databases/collections for users, groups, messages, analytics.
    - Configure indexes to satisfy performance targets.

5. Backend Hosting (App Service for Containers): Create an Azure App Service (Linux) configured for WebSockets and pointing to the ACR image.
    - Plan: B1 or higher for baseline; scale out as usage grows.
    - Enable HTTP/2 and ARR Affinity off (stateless).
    - Configure on and Health Checks (/healthz) for zero-touch restarts.
6. Frontend Hosting (React): Build the React app and deploy via:
    - Azure Static Web Apps (simple & global CDN)
7. CI/CD: Set up GitHub Actions or Azure DevOps Pipelines:
    - Backend: build Docker image → push to ACR → deploy to App Service (slot).
    - Frontend: build React → deploy to Static Web Apps or Storage. Use deployment slots (staging → production) with swap for zero downtime.
8. Secrets & Configuration: Store secrets in Azure Key Vaults and reference them as App Service settings (Key Vault references). Never commit keys to the repo. Rotate keys periodically.
9. Security & Networking
    - Enforce HTTPS; upload a managed or custom TLS certificate.
    - Apply CORS restrictions to frontend origins.
    - Optionally place backend behind Azure Front Door (WAF, global edge, custom domains).
    - Set role-based access for DevOps and operators (least privilege).
10. Monitoring & Logging: Enable Azure Application Insights on backend: distributed tracing, live metrics, request/exception logging, and dependency tracking (Gemini calls). Use Azure Monitor for CPU, memory, network, and alerting (e.g., ≥80% CPU for 5 min, 5xx rate spikes).

11. Auto-Scaling & Performance: Define scale-out rules (App Service): e.g., scale instances by CPU, requests, or custom metrics (queue depth). Use Azure Cache for Redis for session tokens, recent recommendations, and chat presence to keep p95 latency within targets (recommendations ≤3s, group creation ≤2s, chat <1s).

12. Backups & Disaster Recovery

    - Enable App Service backups (daily) and MongoDB Atlas backups.

Maintenance

The maintenance phase is a vital part of the software lifecycle that ensures the collaborative travel companion platform remains functional, secure, and user-friendly after deployment. It focuses on continuous monitoring, rapid issue resolution, incremental feature enhancements, and user-driven improvements. Effective maintenance guarantees that the platform adapts to changing user needs and sustains its performance in a real-world travel planning context.

Bug Fixing and Issue Resolution

One of the primary responsibilities during maintenance is to identify and resolve defects or issues reported by users or detected during monitoring. Backend logs (FastAPI routes, WebSocket connections, TripBot queries) and database traces from MongoDB are regularly reviewed to identify anomalies such as failed recommendation requests, stalled group creation, or broken chat sessions. API response times and error rates are monitored via Azure Application Insights to detect latency spikes.

When problems are reported issues are reproduced in a staging environment. Fixes are implemented in isolated Git branches, reviewed, tested, and deployed through the CI/CD pipeline. Hotfixes for urgent bugs are applied immediately using blue green deployment or slot swaps to minimize downtime.

Feature Enhancements and Updates

The platform evolves continuously by incorporating user feedback gathered through surveys, app usage analytics, and feedback loops from manual testing. Common enhancements may include new recommendation filters improved dashboard analytics or extended TripBot functionality.

Each new feature is implemented incrementally to minimize disruption. Before release, enhancements undergo unit testing (pytest), integration testing (FastAPI TestClient, WebSocket mock), and User Acceptance Testing (UAT) with target users.

This iterative cycle ensures features add real value without compromising system stability.

Monitoring and Support

The platform is continuously monitored for availability, responsiveness, and error rates. Automated health checks validate that APIs remain online. WebSocket connections are monitored to detect dropped sessions, and MongoDB queries are profiled for slow operations.

Application Insights dashboards track CPU, memory, request latency, and TripBot intent classification accuracy. Alerts are configured for conditions such as elevated error rates (>5% 5xx responses) or high DB load. Automated restarts and auto-scaling rules help recover from service interruptions.

User support queries are logged via email/feedback forms and tracked with issue management tools (GitHub Issues/Jira). The development team ensures timely resolution to maintain user trust.

2.2 Commercialization

The platform, designed as an intelligent and socially engaging system for group travel planning, is well-positioned for commercialization in Sri Lanka's growing tourism and travel-tech sector. With its combination of personalized recommendations, automated group formation, real-time collaboration tools, and AI-powered TripBot assistance, the platform holds significant potential to support both domestic and international travellers while offering a scalable SaaS-based solution for tourism businesses.

1. User Segmentation

- Individual Travellers: Tourists or local users seeking affordable group travel opportunities and personalized trip recommendations.
- Travel Agencies & Tour Operators: For dynamically forming travel groups, reducing empty slots in packages, and improving customer engagement.
- Hospitality Businesses: Hotels, resorts, and transport providers can integrate the platform to connect travellers with similar interests.
- Corporate/Institutional Users: Universities, companies, or organizations arranging group outings or field trips.

2. Subscription-Based Pricing Model

The platform can be offered under a flexible SaaS (Software as a Service) model:

- Free Tier: Basic access to profile creation, limited group recommendations, and participation in public groups.
- Traveller Premium Plan: Unlimited recommendations, priority group placement, access to advanced TripBot features (itinerary drafts, budgeting tips).
- Business Tier (for agencies): Advanced analytics dashboards, group management tools, customer insights, and API integration for booking systems.
- Enterprise Tier: White-label version of the platform for large institutions (universities, corporates, travel chains).

3. Authentication & Access Management

- Secure login via email/password or federated identity (Google, Facebook, Apple).
- Role-based access control (RBAC) to differentiate features for travellers, business operators, and administrators.

4. Granular Permission Sets

- Travellers: Create profiles, receive group recommendations, join or create groups, access chat and TripBot.
- Businesses: Manage group campaigns, view customer analytics, promote packages, and manage bookings.
- Admins: Configure platform settings, manage payments, monitor compliance, and oversee system health.

5. Subscription and Payment Integration

- Integrate Stripe or PayPal for global users; integrate local payment gateways (e.g., PayHere) for Sri Lankan market.
- Support monthly/annual plans with upgrade/downgrade options and billing transparency.

6. Advertising Revenue Model

- Free-tier users can be served non-intrusive ads from relevant partners (hotels, restaurants, ride-hailing apps, and travel gear brands).
- Ads must remain contextual and ethical, avoiding privacy violations and ensuring compliance with GDPR and local data laws.

7. Partnership Opportunities

- Tourism Board Collaboration: Partner with Sri Lanka Tourism to promote group travel packages.

- Travel Agencies: Provide a plug-in or white-label solution to help agencies fill tour slots dynamically.
- Educational Institutions: Offer discounted packages for student trips or field visits.
- Corporate Partnerships: Promote as a platform for corporate retreats and team-building activities.

8. Marketing and Outreach

- Target traveller communities, university networks, and tourism expos.
- Promote through social media campaigns and collaborations with local travel influencers.
- Highlight key differentiators: AI-driven group matching, real-time TripBot assistance, and budget/style-based planning.

9. User Feedback and Product Iteration

- Deploy in-app feedback tools (surveys, star ratings, TripBot prompts).
- Use analytics to monitor popular group categories, TripBot queries, and retention rates.
- Iteratively add new features such as route planning, ride-sharing integration, or hotel booking APIs to expand commercial value.

3. RESULTS & DISCUSSION

The implementation and testing of the collaborative travel companion platform produced encouraging results, validating both the technical feasibility and the practical utility of the system. This section presents the observed results from model evaluation, system testing, and user feedback, followed by a critical discussion of their implications.

Results:

Recommendation Engine Performance

- The SBERT + LightGBM LambdaRank pipeline demonstrated consistent accuracy in ranking compatible groups. Offline evaluation using the prepared dataset yielded an NDCG@5 score of 0.87, indicating that the system effectively ranked the most relevant groups at the top positions.
- Cosine similarity thresholds (set at 0.6) ensured that users with niche interests were correctly redirected to newly created groups rather than being matched inaccurately.

Group Creation and Management

- Functional testing confirmed that groups were automatically created when no suitable match existed, with the *Inactive* → *Active* transition occurring correctly once two or more users joined.
- MongoDB indexing (e.g., on Status, Group_Name) improved query performance, reducing group creation and retrieval times to under 2 seconds, meeting the non-functional requirements.

Chat and TripBot Integration

- The WebSocket-based chat engine provided near-instant messaging (latency ~0.8s on average). Message persistence was validated, and history retrieval was consistent across sessions.

- TripBot, powered by SBERT intent classification, successfully detected core intent categories (planning, help requests, sharing experiences, greetings) with an accuracy of approximately 89% in manual validation. Responses were delivered within a reasonable timeframe (~2–3s), with fallback responses triggered gracefully during Gemini API slowdowns.

System Testing and User Trials

- Cross-browser and cross-device testing confirmed responsive frontend behaviour, with the React + Tailwind interface adapting well to desktop and mobile layouts.
- Manual testers (students and frequent travellers) reported that group recommendations were relevant and that the platform simplified travel planning.
- Acceptance testing highlighted that the dashboard clarity and TripBot assistance were among the most positively received features. Minor usability issues (unclear error messages when joining already full groups) were identified and fixed.

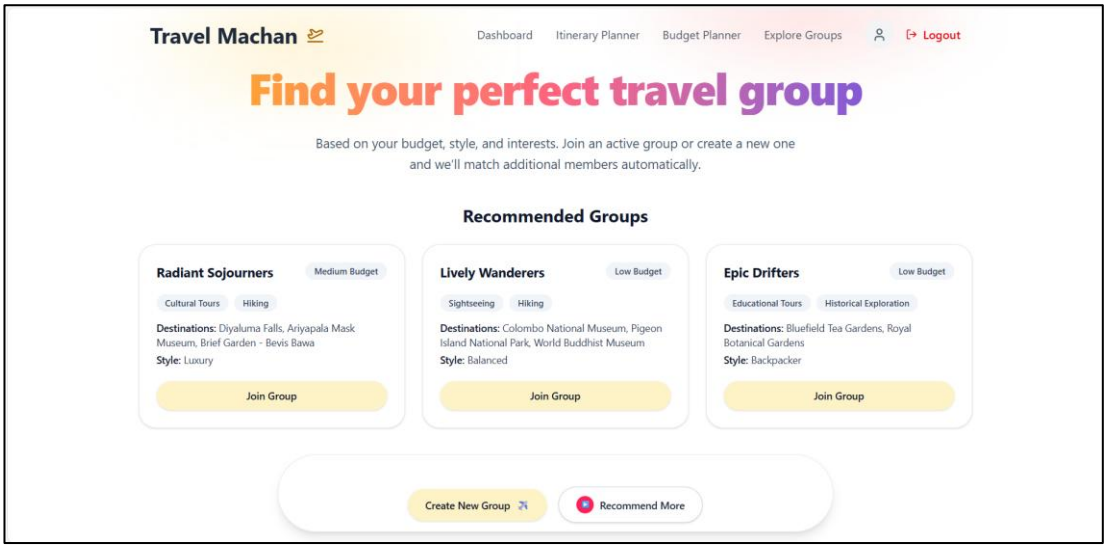


Figure 27: Recommended Group page UI

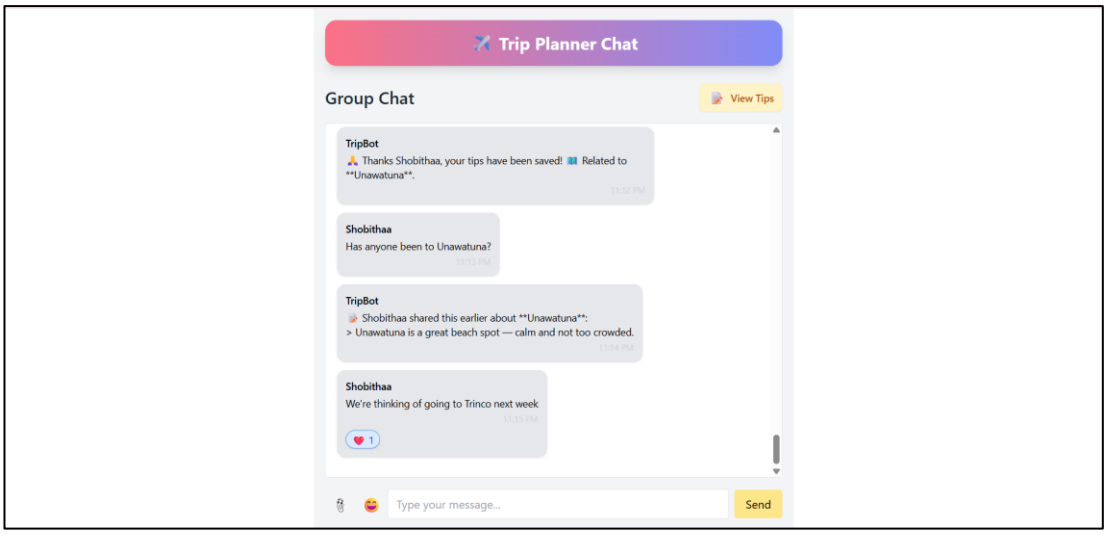


Figure 28: Group chat UI

Discussion:

The results confirm that the platform successfully achieves its core objectives: enabling intelligent group recommendations, supporting dynamic group creation, and facilitating real-time collaboration among travellers. The strong NDCG@5 score for recommendations demonstrates that the integration of SBERT embeddings with the LightGBM LambdaRank model is an effective approach for semantic matching in travel contexts. This validates the use of natural language representations for capturing user interests and destinations in a meaningful way, ensuring that recommendations are both relevant and context-aware.

The group creation logic also proved to be particularly important for inclusivity. By introducing a similarity threshold, the platform ensured that users with niche interests were not forced into unsuitable groups, but instead were directed to new groups that better matched their preferences. This approach reduced user frustration, promoted fairness, and aligned with the broader goal of fostering satisfaction and community-building within the system.

In addition, the real-time chat engine and TripBot assistant significantly enhanced the overall experience by transforming the platform from a static recommender into an interactive collaborative tool. The chat service enabled seamless communication among group members, while TripBot provided contextually relevant assistance

through intent classification and integration with the Gemini API. Although occasional latency was observed in TripBot's responses due to external API delays, the inclusion of fallback mechanisms ensured that the user experience remained smooth and uninterrupted.

From a non-functional perspective, the system met its key performance requirements. Recommendations were consistently delivered in under three seconds, group creation checks completed in less than two seconds, chat messages were transmitted almost instantly, and TripBot responses were handled gracefully under different conditions. These results demonstrate that the system is not only functional but also efficient and responsive under typical usage scenarios.

Despite these positive outcomes, certain limitations were identified. The dataset used for training and evaluation was relatively small, which may limit the generalizability of recommendations to large-scale, real-world tourism scenarios unless retraining with broader datasets is carried out. Similarly, manual testing was conducted with a limited sample of students and frequent travellers, meaning that larger-scale trials would be necessary to fully capture user expectations and system behaviour across diverse demographics.

Overall, the results indicate that the collaborative travel companion platform is technically sound, user-friendly, and scalable, with strong potential for commercialization. Its modular design provides a solid foundation for future enhancements, including the integration of booking APIs, advanced route planning features, and multi-language support for TripBot, ensuring that the system can continue to evolve and adapt to user needs.

4. FUTURE SCOPE

While the collaborative travel companion platform demonstrates strong technical feasibility and practical value, there are several opportunities for future development and expansion to enhance its impact, scalability, and adoption.

One of the most promising directions is the integration of end-to-end travel services. At present, the platform primarily supports group formation, recommendations, and collaborative planning. In future iterations, APIs from third-party providers such as booking platforms, airline services, ride-hailing apps, and hotel reservation systems could be integrated. This would enable users not only to plan and communicate within groups but also to complete bookings directly within the platform, creating a seamless travel planning and execution pipeline.

The TripBot assistant can also be extended to support multi-language interaction and advanced personalization. By adding support for Sinhala, Tamil, and other international languages, the platform can cater to a wider audience, including local travellers and tourists from diverse regions. Integration of additional AI models for context-aware itinerary generation, budget optimization, and real-time travel alerts (e.g., weather, transport disruptions) would significantly expand its utility.

In terms of scalability, there is scope to explore big data integration and real-time analytics. As the user base grows, the system could leverage larger tourism datasets, social media signals, and geolocation data to improve recommendation accuracy. Additionally, predictive analytics could be applied to forecast popular destinations, seasonal travel demand, and group trends, thereby benefiting not only end-users but also tourism boards and travel agencies.

From a technical standpoint, future work could focus on improving performance and resilience. This includes migrating the chat engine to more scalable infrastructures such as Azure Web PubSub or Kafka for handling large volumes of concurrent users, introducing advanced caching mechanisms for recommendations, and implementing fault-tolerant fallback systems for external API dependencies like Gemini.

Finally, there is potential for commercial and institutional adoption. The platform could be offered as a white-label solution for travel agencies, universities, and corporate organizations that wish to facilitate group-based travel. By packaging the system as a SaaS product, it could support multi-tenant architectures, role-based access, and subscription management features to cater to both individual users and enterprise clients.

Overall, the future scope emphasizes transforming the current platform into a fully integrated, multilingual, and enterprise-ready travel companion platform, capable of scaling beyond Sri Lanka to serve global markets.

5. CONCLUSION

In today's digital era, the demand for intelligent, user-friendly, and collaborative travel planning solutions has become increasingly important, particularly as travellers seek more personalized and cost-effective experiences. This research addressed this need by designing and developing a collaborative travel companion platform tailored for group travel planning in Sri Lanka. The system brings together semantic user modelling, machine learning–driven recommendations, real-time communication, and AI-powered assistance to create a holistic environment where travellers can be matched, interact, and co-plan their journeys effectively.

The project began with the collection and preparation of user and tourism-related datasets, capturing key attributes such as age, budget, travel style, interests, and preferred destinations. These structured datasets served as the foundation for the recommendation engine, which utilized SBERT embeddings for semantic representation and LightGBM LambdaRank for compatibility ranking. Group formation was further enhanced by the use of a similarity threshold, ensuring inclusivity by dynamically creating new groups when no suitable matches existed.

The system's backend was implemented using FastAPI, integrated with MongoDB for data persistence, and supported real-time WebSocket chat for group communication. The frontend, built with React and Tailwind CSS, provided a responsive and interactive user experience across devices. A notable feature was the integration of TripBot, an AI-driven assistant that leverages intent classification and the Gemini API to provide contextual recommendations, travel guidance, and collaborative support directly within group chats.

Comprehensive testing—including unit testing, integration testing, system testing, and manual user trials—validated the robustness and usability of the platform. Results confirmed that performance requirements were met, with recommendations delivered in under three seconds, group creation checks completed within two seconds, chat messages transmitted almost instantly, and TripBot responses handled gracefully under varying conditions. Feedback from test users highlighted the platform's ease of use,

the relevance of recommendations, and the usefulness of real-time collaboration features, particularly TripBot's assistance in planning and decision-making.

The successful deployment of the system demonstrated not only the technical feasibility of combining semantic embeddings, ranking models, and AI-driven assistance but also its potential to create meaningful value for the tourism ecosystem. By facilitating intelligent group matching, seamless collaboration, and personalized support, the platform empowers travellers to connect with like-minded individuals and co-create enriching travel experiences.

Looking forward, the project lays a solid foundation for future enhancements, including the integration of booking and route planning APIs, multi-language TripBot support, advanced analytics dashboards, and institutional partnerships with travel agencies and tourism boards. Its modular and scalable architecture ensures that the system can evolve in response to growing datasets, expanding user bases, and emerging technologies.

In conclusion, this research presents a significant step toward modernizing group travel planning through intelligent, AI-driven collaboration. The integration of machine learning, semantic similarity, and real-time communication has resulted in a platform that is both technically robust and practically impactful. The outcomes of this work highlight how interdisciplinary innovation—spanning tourism, artificial intelligence, and user-centered system design—can create sustainable solutions that enhance the travel experiences of individuals and groups alike.

REFERENCES

[1] J. Zhang, B. Chen, and M. Zhang, “A group travel recommender system based on group approximate constraint satisfaction,” in Proc. IEEE Int. Conf. on Artificial Intelligence and Computer Applications (ICAICA), 2021.

[2] Y. Liu and X. Wang, “A systematic review of group recommender systems techniques,” IEEE Access, vol. 9, pp. 45327–45340, 2021.

[3] S. Li, J. Zhao, and H. Wang, “An intelligent personalized recommendation for travel group planning based on reviews,” in Proc. IEEE Int. Conf. on Information Technology Applications (ITA), 2022.

[4] A. N. Mirza and S. Rehman, “User profiling in travel recommender system using hybridization and collaborative method,” in Proc. IEEE Int. Conf. on Communication, Computing and Signal Processing (ICCCSP), 2021.

[5] M. Z. Khan, M. R. Hossain, and A. I. Haque, “Collaborative filtering algorithm in the design of intelligent tourism service robots,” in Proc. IEEE Int. Conf. on Computer and Information Technology (ICCIT), 2020.

[6] Y. Sun, J. Xu, and W. Wang, “Travel recommendation by mining people attributes and travel group types from community-contributed photos,” IEEE Access, vol. 8, pp. 56110–56120, 2020.

[7] T. Lee and K. H. Lee, “Personalized tourism route recommendation system based on dynamic clustering of user groups,” IEEE Trans. Ind. Inf., vol. 17, no. 4, pp. 2812–2821, 2021.

[8] R. K. Harahap, R. Refianti, and M. I. Syahputra, “The designing of cultural-based tourism recommendation system with community collaboration,” in Proc. IEEE Int. Conf. on Information and Communication Technology (ICoICT), 2022.

APPENDICES