

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

**(HYBRID PERSONALIZED RECOMMENDATION
AND ITINERARY PLANNING SYSTEM)**

Chanidu Dewmika Senevirathne

IT21831768

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

**(HYBRID PERSONALIZED RECOMMENDATION
AND ITINERARY PLANNING SYSTEM)**

Chanidu Dewmika Senevirathne

IT21831768

BSc (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
Chanidu Dewmika Senevirathne	IT21831768	


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)



Date



Signature of co-supervisor
(Ms. Karthiga Rajendran)



Date

ABSTRACT

Tourism remains a cornerstone of Sri Lanka's economy, yet travelers continue to encounter difficulties in planning trips that are both personalized and practical. Existing platforms largely provide static recommendations and fail to address the combined challenges of personalization, geographic feasibility, safety, and sustainability. To bridge this gap, this study proposes a Hybrid Personalized Recommendation and Itinerary Planning System, developed as part of an AI-driven smart tourism platform.

The system integrates Content-Based Filtering (CBF), Collaborative Filtering (CF), and a Machine Learning model (XGBoost) to generate robust personalized recommendations. Travelers can create a plan pool of preferred locations, from which a corridor-aware plan generator constructs geographically feasible travel routes between chosen start and end cities. To enhance flexibility, the system produces multiple itinerary options including similar-type, similar-activities, nearby, and hybrid suggestions while ensuring they remain within safety and sustainability constraints.

Evaluation was conducted using datasets of users, locations, and reviews specific to Sri Lanka. The hybrid recommendation engine demonstrated strong predictive performance, achieving a root mean squared error (RMSE) of 0.48 and a coefficient of determination (R^2) of 0.77, outperforming single-method approaches. Qualitative testing further confirmed that generated itineraries were coherent, contextually relevant, and practically executable.

The outcomes of this research highlight the potential of hybrid AI-driven methods to improve travel planning in emerging tourism markets. The results show that the hybrid engine achieved strong accuracy, while the generated itineraries were practical and contextually relevant. This suggests that hybrid AI methods can improve tourism planning by offering both accuracy and realistic travel routes for Sri Lanka.

Keywords – *Smart Tourism, Hybrid Recommendation, Itinerary Planning, Personalization, XGBoost*

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to all those who supported me throughout the course of this research and the successful completion of this project.

First and foremost, I extend my heartfelt appreciation to my supervisor, Ms. Thilini Jayalath, for his invaluable guidance, constructive feedback, and continuous encouragement during every stage of this research. His mentorship was instrumental in shaping the direction of the study and ensuring its successful completion.

I am equally grateful to my co-supervisor, Ms. Karthiga Rajendran, for her timely advice, insightful suggestions, and unwavering support, which greatly enhanced the quality of this work.

My appreciation also goes to my family, friends, and colleagues at the Sri Lanka Institute of Information Technology for their encouragement, thoughtful discussions, and moral support throughout this journey. Their belief in me gave me the motivation and confidence to persevere through challenges.

Finally, I would like to thank all the individuals who contributed their time during the testing and evaluation phases, providing valuable feedback that helped refine the system into a more robust and user-friendly solution.

Without the collective efforts of these individuals, the successful completion of this project would not have been possible

TABLE OF CONTENTS

DECLARATION.....	i
ABSTRACT.....	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES.....	vi
LIST OF TABLES.....	viii
LIST OF ABBREVIATIONS.....	ix
1 INTRODUCTION	1
1.1 Background and Literature Survey	1
1.2 Research Gap	3
1.3 Research Problem.....	5
1.4 Research Questions	6
1.5 Research Objectives	7
1.5.1 Main Objective.....	7
1.5.2 Specific Objectives.....	7
1.5.3 Business Objectives	9
2 METHODOLOGY.....	10
2.1 Methodology	10
2.1.1 Feasibility Study	11
2.1.2 Requirement Gathering & Analysis	16
2.1.3 System Design.....	23
2.1.4 Implementation	27
2.1.5 Testing.....	62
2.1.6 Deployment	68
2.1.7 Maintenance	70
2.2 Commercialization	72
2.2.1 Target Audience.....	72

2.2.2	Market Strategies	73
2.2.3	Revenue Model	74
2.2.4	Sustainability and Scalability	74
3	RESULTS & DISCUSSION	75
4	FUTURE SCOPE.....	78
5	CONCLUSION.....	79
	REFERENCES.....	81
	APPENDICES	82

LIST OF FIGURES

Figure 1 : Hybrid System Diagram	12
Figure 2 : Signup and Recommendation Generation Use Case Diagram	21
Figure 3 : Planning Use Case Diagram	22
Figure 4 : Itinerary Use case Diagram	22
Figure 5 : System Overview Diagram.....	24
Figure 6 : Work breakdown chart.....	30
Figure 7 : Location Data CSV.....	33
Figure 8 : User Data CSV	34
Figure 9 : Review Data CSV.....	35
Figure 10 : Data Preporcessing and Cleaning	38
Figure 11 : Loading the ImprovedRecommender model along with its saved encoders and features from artifacts.....	48
Figure 12 : A function for generating personalized travel recommendations based on user inputs.	49
Figure 13 : A function for handling travel recommendation requests through an API endpoint.....	50
Figure 14 : Geodesic utilities used to constrain candidates to a corridor.....	52
Figure 15 : List of stops, start/end locations, and optional limits to guide the path.	52
Figure 16 : Algorithm segment for route optimization	53
Figure 17 : Function for creating a travel plan: checks inputs, sets start and end points, and calculates total distance with optional attractions.	54
Figure 18 : API endpoint to generate a new trip plan based on user input	55
Figure 19 : API endpoint to optimize a given trip itinerary.	56
Figure 20 : Function to build similarity blocks by comparing locations using distance, activities, and reviews.	57
Figure 21 : Helper functions for calculating distances, bearings, and coordinates used in travel planning.....	58
Figure 22 : Function to find travel options for a location, applying distance limits and optional corridor filters.	59
Figure 23 : Function to suggest location options by checking similarity, same type, nearby places, and top-rated alternatives.	60
Figure 24 : API endpoint to fetch itinerary options based on location, distance, and optional corridor filters.	61
Figure 25 : API endpoint for travel recommendations that returns personalized location suggestions based on user profile and preferences.	64

Figure 26 : API test results confirming the recommendations endpoint works correctly
with status, limits, and sorting checks.....64

Figure 27 : API endpoint to generate a trip itinerary, returning ordered locations with
details like type, province, and distance.....65

Figure 28 : API test results verifying the trip generation endpoint, confirming success
with valid plan and attractions.65

Figure 30 : Docker Compose configuration to run the AI Tourism69

LIST OF TABLES

Table 1 : Research Gap Comparison	4
Table 2 : Cost Estimation	13
Table 3 : Functional Requirements	18
Table 4 : Non-Functional Requirements	20
Table 5 : XGBoost vs RandomForest Comparison	42
Table 6 : System Requirements	45
Table 7 : Unit Testing	62
Table 8 : Integration Testing	63
Table 9 : System Testing	66
Table 10 : Model Evaluation Comparison	76
Table 11 : Endpoint API Responsiveness	76

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
GDP	Gross Domestic Product
UI	User Interface
API	Application Programming Interface
SEO	Search Engine Optimization

1 INTRODUCTION

1.1 Background and Literature Survey

Artificial Intelligence (AI) has emerged as a transformative force across diverse industries, offering data-driven solutions to complex challenges and enabling enhanced decision-making. Its adoption has been particularly impactful in domains such as marketing [1] and medicine [2], where AI-powered systems have optimized personalization, predictive analytics, and resource allocation. However, despite its widespread influence, the tourism sector has been slower to fully harness AI's potential [3]. As one of the world's most significant industries, tourism was projected to contribute USD 11.1 trillion to the global GDP by 2024, accounting for nearly 10% of global economic output and generating 348 million jobs [4]. This underscores the importance of integrating advanced technologies to sustain competitiveness and resilience.

Tourism is also a cornerstone of Sri Lanka's economy, leveraging its natural landscapes, cultural heritage, and historic landmarks to attract millions of visitors annually. Yet, the sector faces pressing challenges in meeting the evolving expectations of travelers, who increasingly demand personalized, seamless, and adaptive travel experiences. Traditional trip planning methods—often based on fixed itineraries or static travel agent recommendations—are unable to accommodate diverse user preferences, contextual constraints, and the need for real-time flexibility. This mismatch results in suboptimal travel experiences that undermine satisfaction and limit the sector's growth potential.

Advancements in AI and Machine Learning (ML) present an opportunity to revolutionize tourism planning. Recommendation systems, which have proven effective in e-commerce and entertainment, provide a strong foundation for personalizing tourism experiences. Classical approaches include Content-Based Filtering (CBF), which matches user preferences with item attributes, and Collaborative Filtering (CF), which infers preferences from patterns of similar users. Both methods, however, have limitations - CBF suffers from over-specialization and lacks diversity, while CF struggles with cold-start

and data sparsity problems. To overcome these, hybrid models that combine multiple approaches have been shown to deliver superior accuracy and robustness [5].

In the tourism domain, AI-driven platforms can leverage hybrid recommenders to generate personalized suggestions for destinations, activities, accommodations, and itineraries. Such systems also align with the growing emphasis on sustainable tourism, as they can promote eco-friendly destinations, encourage balanced distribution of visitor flows, and provide recommendations that support local communities [6]. Moreover, travelers increasingly prefer dynamic, adaptive platforms capable of adjusting to their evolving needs rather than static, one-size-fits-all solutions.

Despite these global advancements, Sri Lanka's tourism sector currently lacks comprehensive AI-driven platforms that integrate advanced personalization with geographically feasible and sustainable itinerary planning. Existing solutions rarely provide dynamic customization, real-time adaptability, or accurate corridor-aware guidance for travelers navigating unfamiliar regions.

To address this gap, this research develops a Hybrid Personalized Recommendation and Itinerary Planning System, integrated within an AI-driven smart tourism platform tailored for Sri Lanka. The system combines CBF, CF, and ML (XGBoost) to improve recommendation accuracy, achieving an RMSE of 0.48 and an R^2 of 0.77 in evaluation. Beyond recommendations, the platform introduces a plan pool mechanism for travelers to curate destinations, a corridor-aware plan generator to ensure feasible routes between chosen start and end cities, and multiple itinerary options (similar-type, similar-activities, nearby, hybrid) constrained by safety and sustainability rules.

This study proposes a hybrid system tailored for Sri Lanka, combining CBF, CF, and ML (XGBoost). The system introduces a plan pool, corridor-aware routes, and multiple itinerary options, aiming to deliver a practical and user-friendly travel planning tool.

1.2 Research Gap

Tourism is one of the most significant contributors to the global economy, with advancements in Artificial Intelligence (AI) offering transformative potential for industry. However, despite its importance, the integration of AI-driven solutions within the tourism sector remains underdeveloped compared to domains such as e-commerce and healthcare [1][2]. Existing AI applications in tourism such as recommendation engines, travel agents, and traffic prediction models [3][4][5] often rely on generic approaches that fail to capture the specific cultural, geographic, and operational needs of regions like Sri Lanka.

Prior research highlights several shortcomings:

- **Limited Personalization:** Most platforms rely on single-method recommenders (location-based or pure collaborative filtering) that struggle with cold-start issues, over-specialization, and sparse data [6].
- **Lack of Real-Time Itinerary Flexibility:** Existing systems provide static itineraries with minimal adaptability to traveler inputs or contextual constraints [7].
- **Absence of Interactive Features:** Few solutions enable travelers to curate or customize their trip plan pools with intuitive features such as drag-and-drop itinerary building.
- **Neglect of Regional Relevance:** Research has often emphasized global applicability but failed to design systems tailored for the Sri Lankan tourism context, where corridor-based travel feasibility and sustainability are key.

To address these gaps, the proposed study introduces a Hybrid Personalized Recommendation and Itinerary Planning System that integrates Content-Based Filtering

(CBF), Collaborative Filtering (CF), and Machine Learning (XGBoost) into a unified pipeline. The system enhances personalization by fusing multiple techniques, supports corridor-aware travel plans between chosen start and end points, and generates multiple itinerary options (similar-type, similar-activities, nearby, hybrid) constrained by safety and sustainability factors.

Table 1 : Research Gap Comparison

Criteria	Existing System [7]	Existing System [8]	Proposed System
Personalized Recommendation	Location-based, collaborative filtering	Collaborative filtering + hybrid	Advanced hybrid model (CBF + CF + XGBoost)
Real-Time Itinerary Adjustments	No real-time updates	Limited contextual adjustments	Dynamic itinerary updates with corridor and plan-pool constraints
Interactive Features	No drag-and-drop tools	Minimal customization	Drag-and-drop style plan pool and user-curated itineraries
Regional Relevance	Generic/global focus	Limited local optimization	Tailored for Sri Lanka with corridor-aware routes and sustainability considerations

The system directly addresses these gaps by combining personalization, corridor-aware planning, and sustainability. The aim is to give travelers a tool that is both accurate and practical for Sri Lanka's context.

1.3 Research Problem

The global tourism sector has increasingly adopted Artificial Intelligence (AI) to improve personalization, operational efficiency, and user experience. However, Sri Lanka's tourism industry remains underserved by advanced AI-driven solutions tailored to its unique regional and cultural ecosystem. Existing platforms often focus on static personalization and generalized recommendations [7], but they fail to deliver features that modern travelers expect, such as real-time itinerary adjustments, adaptive personalization, and flexible user control.

The limitations of existing systems can be summarized as follows:

1. **Static Personalization:** Current systems primarily rely on single-method recommenders, producing generalized suggestions that fail to capture the multi-dimensional preferences of individual travelers.
2. **Lack of Real-Time Adaptability:** Most platforms cannot dynamically update itineraries in response to new user preferences, situational changes, or contextual constraints such as distance and time.
3. **Limited Interactivity:** Few systems allow travelers to actively participate in the planning process through intuitive tools like drag-and-drop plan pools or interactive map-based views.
4. **Weak Itinerary Coherence:** Existing recommenders often suggest isolated points of interest without ensuring geographic feasibility (e.g., corridor-based sequencing from a start city to an end city) or contextual relevance.
5. **Insufficient Depth of Recommendations:** Platforms rarely provide complementary or alternative itinerary options (e.g., similar activity, nearby, hybrid choices), thereby reducing the richness of the user experience.

These gaps highlight the need for a comprehensive AI-driven smart tourism platform tailored to Sri Lanka, capable of generating personalized, adaptive, and context-aware travel itineraries. The proposed system addresses this gap by integrating Content-Based Filtering (CBF), Collaborative Filtering (CF), and Machine Learning (XGBoost) into a hybrid recommender, achieving predictive accuracy ($RMSE = 0.48$, $R^2 = 0.77$) while also incorporating corridor-aware plan generation and multiple itinerary options constrained by safety and sustainability considerations.

1.4 Research Questions

To address the above problems, this study investigates the following research questions:

- a. How can user preferences (e.g., destinations, demographics, activities, group type) and contextual constraints (time, start-end corridor) be effectively modeled to generate personalized travel recommendations?
- b. How can hybrid AI techniques (CBF + CF + ML) outperform single-method approaches in predicting user-location relevance and improving personalized accuracy ?
- c. How can corridor-aware plan generation ensure geographically feasible and safe itineraries while maintaining flexibility for user customizations?
- d. What types of itinerary options (e.g., similar-type, similar-activities, nearby, hybrid) provide the greatest value for enhancing traveler satisfaction and decision-making?
- e. How can interactive features such as plan poll, drag-and-drop customization, and multiple option comparisons improve traveler engagement and usability?
- f. How can the system maintain efficiency (fast response times, lightweight operation) when handling large datasets of locations, reviews, and user requests?

1.5 Research Objectives

1.5.1 Main Objective

The primary objective of this research is to design and implement a Hybrid Personalized Recommendation and Itinerary Planning System that enhances the travel experience for tourists in Sri Lanka. The main objective is to design and implement a Hybrid Recommendation and Itinerary System using CBF, CF, and XGBoost. The system goes beyond destination suggestions by building feasible, corridor-based routes and offering diverse itinerary options.

By achieving this objective, the system aspires to deliver a comprehensive, user-centric solution that empowers travelers to plan trips independently even without prior knowledge of destinations while simultaneously contributing to Sri Lanka's competitiveness as a smart tourism hub.

1.5.2 Specific Objectives

a) Develop a Hybrid Recommendation Model

Integrate Content-Based Filtering (CBF), Collaborative Filtering (CF), and Machine Learning (XGBoost) into a unified recommendation framework that leverages user demographics, activity preferences, and review data to compute destination relevance scores.

b) Achieving High Predictive Accuracy

Train and validate the hybrid model using curated datasets of users, locations, and reviews, with a focus on minimizing prediction error and maximizing explained variance. Employ evaluation metrics such as Root Mean Squared Error (RMSE) and Coefficient of Determination (R^2).

c) Implement a Plan Pool and Corridor-Aware Plan Generator

Design a plan pool mechanism that allows users to select preferred destinations, from which a corridor-aware generator constructs feasible travel routes between start and end cities.

d) Provide Multiple Itinerary Options

Develop algorithms to produce diverse itinerary alternatives, including similar-type, similar-activities, nearby, and hybrid itineraries, giving travelers flexibility in their planning process.

e) Promote Sustainability and User Engagement

Embed safety-aware and eco-conscious constraints (e.g., corridor limits, nearby suggestions) while providing interactive customization features such as plan pools and itinerary comparisons.

1.5.3 Business Objectives

- **Enhance Traveler Satisfaction and Engagement:** Deliver personalized recommendations by combining user demographics, preferences, and peer patterns through a hybrid recommendation engine. By offering contextually relevant suggestions and user-friendly plan customization, the platform ensures an engaging travel planning experience, encouraging long-term satisfaction.
- **Promote Sustainable and Balanced Tourism:** Incorporate corridor-aware itinerary generation to distribute tourist flows across both popular and underexplored locations. This prevents overcrowding, reduces environmental stress on hotspots, and promotes cultural and ecological sustainability within Sri Lanka's tourism sector.
- **Increase Safety and Travel Feasibility:** Provide realistic itineraries that connect start and end cities with geographically consistent routes. Features such as corridor validation and nearby attraction suggestions ensure safer and more feasible travel experiences, reducing uncertainty for both local and international travelers.
- **Support Economic Growth in the Tourism Sector:** Connect travelers with local businesses such as accommodations, restaurants, and cultural attractions. By integrating personalized recommendations with itinerary planning, the platform creates more opportunities for tourism-related enterprises, thereby boosting Sri Lanka's-economy.
- **Encourage Technology Adoption in Tourism:** Demonstrate the value of AI-driven personalization in a real-world tourism context. By showcasing hybrid recommendation models, corridor-aware planning, and multi-option itineraries, the system highlights how advanced AI and machine learning can transform the travel sector in Sri Lanka and set a benchmark for future smart tourism initiatives.

2 METHODOLOGY

2.1 Methodology

The methodology adopted for this project follows a hybrid approach, combining Agile Scrum practices with the seven-stage Software Development Life Cycle (SDLC). This dual framework was selected to ensure adaptability to evolving requirements, iterative improvement, and rigorous evaluation of each subsystem: the Hybrid Recommendation Engine, Plan Pool, and Itinerary Builder.

- Agile Scrum allowed development in short sprints (2–3 weeks), where each sprint focused on a core module such as the recommendation engine, corridor-aware plan pool, or itinerary generation. Sprint reviews, retrospectives, and stand-ups facilitated continuous improvement and stakeholder alignment. Tools such as Jira and GitHub Projects were used for backlog management, sprint tracking, and issue resolution.
- Seven-Stage SDLC guided the systematic phases of project development: feasibility, requirements analysis, design, implementation, testing, deployment, and maintenance. This project followed a hybrid approach combining Agile Scrum and the seven-stage SDLC. Agile provided flexibility through short sprints focused on core modules such as the recommender, plan pool, and itinerary builder. SDLC gave structure through phases like feasibility, design, implementation, and testing. Together, they ensured steady progress while allowing refinements when issues arose.

The combination of Agile and SDLC enabled the integration of complex AI models, real-time APIs, and scalable frontend-backend systems, while ensuring that the platform aligned with research objectives and user expectations.

2.1.1 Feasibility Study

Before development, a feasibility study was conducted to determine whether the proposed solution was technically achievable, economically sustainable, legally and ethically compliant, operationally viable, and socially beneficial.

2.1.1.1 Technical Feasibility

The proposed system is technically feasible due to the availability of modern tools, frameworks, and cloud services that align with both academic and commercial-grade deployments. The design adopts a modular three-tier architecture consisting of a React-based frontend, a FastAPI backend integrated with Python ML models, and a MongoDB database layer.

The Frontend is implemented using React (JavaScript), chosen for its component-driven architecture, reusable UI patterns, and wide ecosystem. The frontend is implemented using React, supported by modern CSS frameworks for responsive and reusable design

The Backend leverages FastAPI, a high-performance Python framework for building APIs. It manages communication between the frontend and machine learning modules, while handling concurrency through asynchronous programming. ML models are trained using scikit-learn and XGBoost, persisted with Joblib, and exposed via REST endpoints. This enables real-time recommendation and itinerary generation within seconds.

The Database layer initially uses curated CSV datasets (users.csv, locations.csv, reviews.csv), providing training and testing data. For scalability, the design supports migration to MongoDB Atlas, ensuring document-oriented storage suitable for dynamic travel data.

External APIs such as Google Maps API are integrated for corridor validation and geospatial computations, ensuring that generated itineraries remain geographically feasible. Deployment is handled via Docker containers, guaranteeing consistency across development, testing, and production. Hosting is planned on AWS EC2 free-tier instances with auto-scaling capabilities.

In summary, the technical infrastructure is feasible, scalable, and maintainable, combining the flexibility of open-source libraries with the robustness of cloud services.

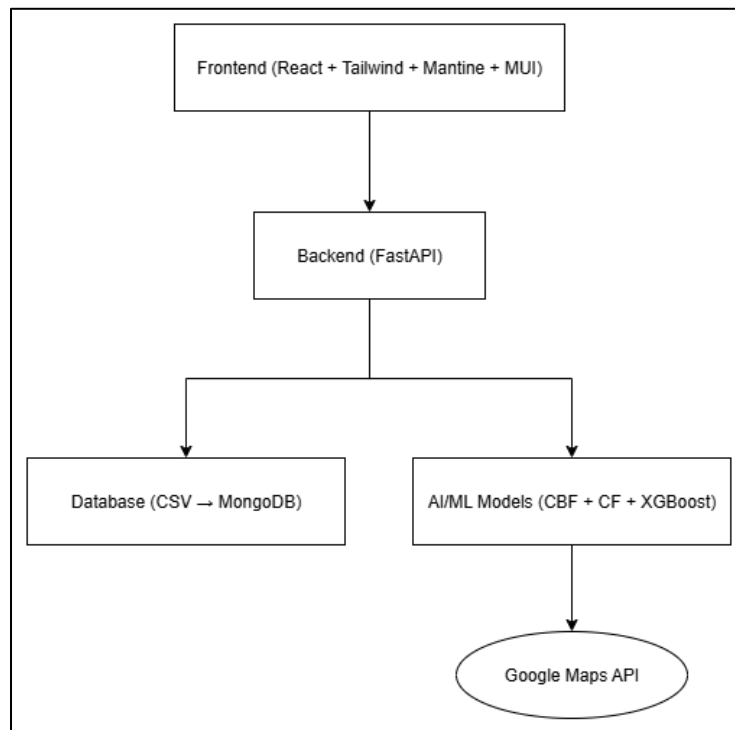


Figure 1 : Hybrid System Diagram

2.1.1.2 Economic Feasibility

The economic feasibility of the project was ensured through the use of cost-efficient, open-source technologies and free-tier cloud services. Development costs were minimized by relying on free academic licenses and student access to cloud resources.

Table 2 : Cost Estimation

Cost Component	Details	Estimated Cost (LKR)
Internet use and Web Hosting	Router packages and hosting costs	Rs.10000
JetBrains Tools	WebStorm, DataSpell	Free (student plan)
Cloud Hosting	AWS EC2 + MongoDB	Rs. 4500 (initial phase)
API's	Google Maps API	Free (3-month free plan)
Containerization	Docker	Free
Miscellaneous	GitHub etc.	Rs. 2000

Future Revenue Models:

- **Tourism Board Partnerships:** National deployment for Sri Lanka Tourism Authority.
- **B2B API Services:** Licensing APIs to travel agencies for integration into booking platforms.

2.1.1.3 Operational Feasibility

The platform is designed for practical adoption by end users (tourists) and stakeholders (agencies, tourism boards). Key aspects:

- **Usability:** Interactive UI with drag-and-drop itinerary customization, filters, and real-time previews ensures accessibility for non-technical users.
- **Cross-Platform Access:** Built as a web application, it runs on modern browsers across devices (desktop, tablet, mobile).
- **Performance:** Benchmarks indicate recommendation generation and itinerary planning complete within ≤ 7 seconds under moderate load.
- **Maintainability:** Modular architecture (Recommendation \rightarrow Plan Pool \rightarrow Itinerary) simplifies debugging and future enhancements.
- **User Testing:** Peer evaluations confirmed ease of use and strong engagement with corridor-aware planning features.

Usability testing showed the drag-and-drop interface was easy to use, and performance benchmarks confirmed that recommendations were generated in under 7 seconds. The overall development followed a timeline of requirement gathering (3 months), implementation (5 months), testing (2 months), and deployment (1 month).

2.1.1.4 Schedule Feasibility

This platform is an independent solution and will be developed according to a well-defined timeline. The project phases include:

- Requirement gathering and design: 3 months.
- Development of key features: 5 months.
- Testing and debugging: 2 months.
- Deployment and launch: 1 month.

Agile development practices will be followed to ensure iterative progress, timely delivery, and the ability to adapt to changes.

2.1.1.5 Social Feasibility

The project contributes positively to Sri Lanka's tourism ecosystem by addressing both traveler needs and societal goals:

- **Sustainable Tourism:** By promoting lesser-known destinations, the system reduces overcrowding in hotspots and distributes economic benefits to rural areas.
- **Safety-Oriented Travel:** Corridor-aware routing ensures travel within safe, practical distances, especially important for unfamiliar tourists.
- **Cultural Awareness:** Personalized recommendations highlight cultural, historical, and ecological sites, encouraging immersive experiences.
- **Economic Equity:** Local businesses gain exposure when included in itineraries, supporting small-scale tourism enterprises.

2.1.2 Requirement Gathering & Analysis

Requirement gathering is a critical phase in the Software Development Life Cycle (SDLC). For this project, requirements were collected through **dataset analysis**, **benchmarking existing tourism platforms**, and **brainstorming sessions with stakeholders**. The aim was to ensure that the proposed **AI-driven smart tourism platform** addresses gaps in personalization, flexibility, and sustainability while being technically feasible and user-centric.

Requirements were categorized into functional and non-functional, followed by analysis and validation. Additionally, use cases and diagrams were developed to visualize the interactions between system actors and modules.

2.1.2.1 Sources of Requirements

Dataset Extraction and Preparation: Tourism site data was systematically extracted and normalized into three interconnected CSV files:

- **user_data.csv:** Contains user profiles with userId, age group, travel companion preferences, preferred activities, and demographic information
- **location_data.csv:** Comprehensive location dataset including location name, type classification, average ratings, geographical coordinates (latitude/longitude) obtained via Google APIs, and available activity types at each location
- **review_data.csv:** Acts as the relationship bridge, linking users to locations through userId and location name, capturing user-location interactions and feedback

Feature Engineering and Analysis: The extracted datasets were analyzed to identify key predictive features including age group travel patterns, travel companion influence on destination choice, location type preferences, and activity overlap analysis between user preferences and location offerings. Geographic coordinates enabled spatial analysis for route optimization and proximity-based recommendations.

Existing Systems Benchmarking: Comprehensive review of platforms like TripAdvisor, Google Travel, and Booking.com revealed critical personalization gaps including lack of corridor validation for route planning, absence of curated plan pools for itinerary templates, and predominantly static itineraries without real-time adaptability.

Plan Pool Strategy: Location selection was strategically filtered to include only destinations that contribute to the plan pool, ensuring recommended locations align with buildable, coherent itineraries rather than isolated suggestions.

Stakeholder Input: Informal surveys with peers and feedback from supervisors emphasized essential features including drag-and-drop itinerary planning for user control, alternative itineraries for flexibility, and real-time performance for dynamic updates.

2.1.2.2 Functional Requirements

Table 3 : Functional Requirements

Requirement	Justification	Expected Outcome
User Registration and Profile Management	Users need to create accounts to store demographics and preferences.	Secure registration and login with persistent profile data.
Preference Collection	Essential for generating hybrid recommendations. Includes demographics (age group, gender, companion type) and preferred activities.	Accurate input for personalization models.
Hybrid Recommendation Engine	Provides destination suggestions using CBF, CF, and ML to improve accuracy and reduce cold-start issues.	Top-N personalized destinations with relevance scores.
Plan Pool	Acts like a cart where users can shortlist destinations.	User-selected destinations ready for itinerary generation.
Corridor-Aware Plan Generator	Ensures travel routes are geographically feasible between start and end cities.	Realistic travel sequences validated by Google Maps API.
Itinerary Module	Generates four types of itineraries (similar-type, similar-activities, nearby, hybrid).	Multiple travel options for user comparison.

Itinerary Customization	Drag-and-drop planner allows rearranging and modifying itineraries.	Flexible, interactive user experience.
Export & Sharing	Final itineraries can be saved/exported as PDF/JSON.	Practical usability for real-world travel planning.
Admin Module	Admin can manage datasets (locations, reviews).	Up-to-date system content and metadata.

2.1.2.3 Non-Functional Requirements

Table 4 : Non-Functional Requirements

Requirement	Justification	Expected Outcome
Reliability	Same inputs should produce consistent outputs unless datasets change.	Builds trust with stable recommendations.
Performance	Recommendations/itineraries should generate in ≤ 7 seconds.	Smooth, responsive user interaction.
Scalability	System must handle larger datasets and concurrent users.	Cloud-ready design via Docker & AWS.
Usability	Drag-and-drop and map-based interfaces must be intuitive.	High user adoption and engagement.
Security	Sensitive demographics and preferences must be encrypted.	Future enhancements and debugging simplified.
Maintainability	Modular architecture enables easier updates.	Multiple travel options for user comparison.
Availability	Should remain accessible with high uptime	Reliable access for users worldwide.
Portability	Containerized systems should run across environments.	Flexible deployment (cloud, local).

2.1.2.4 Use Case Analysis

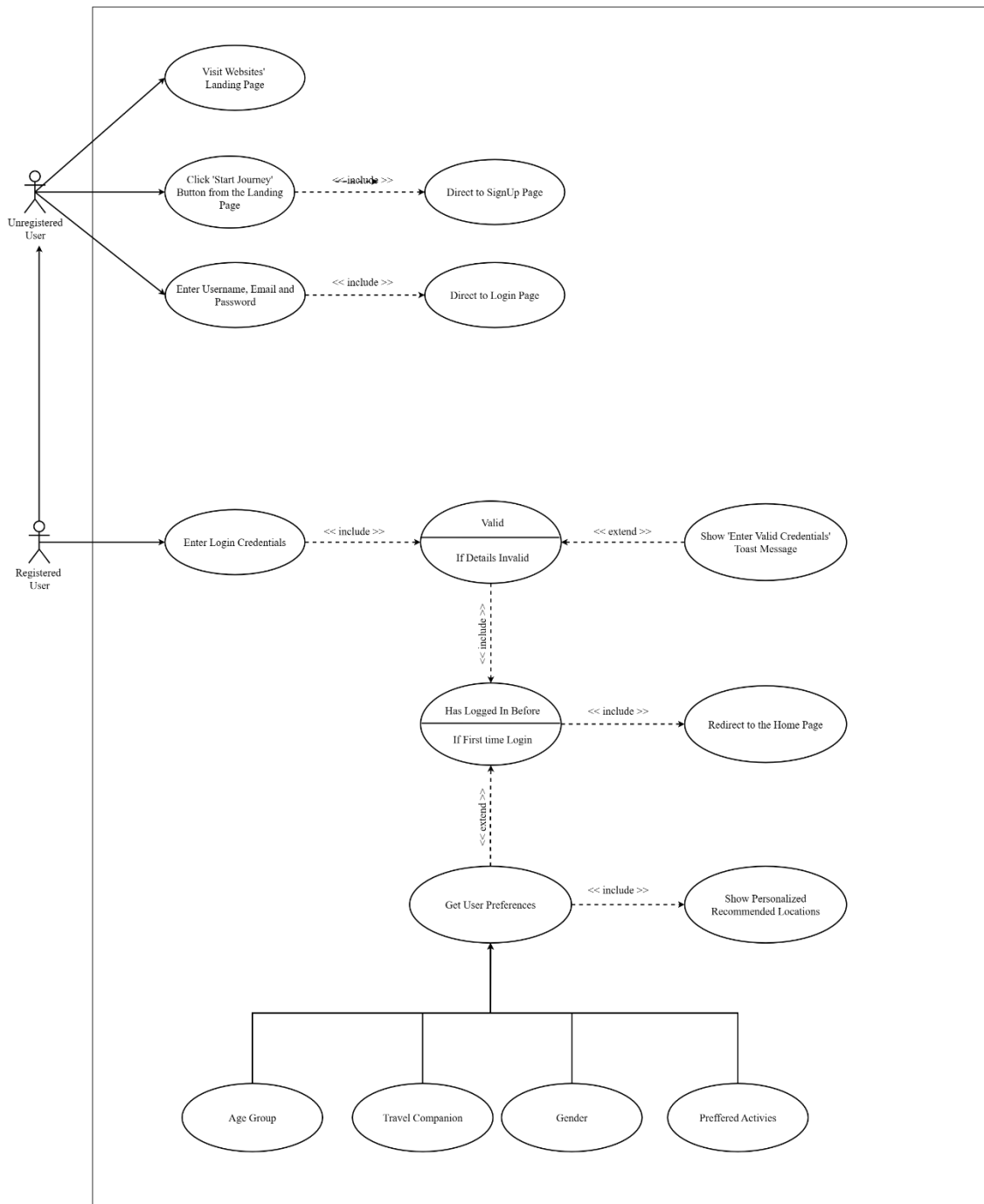


Figure 2 : Signup and Recommendation Generation Use Case Diagram

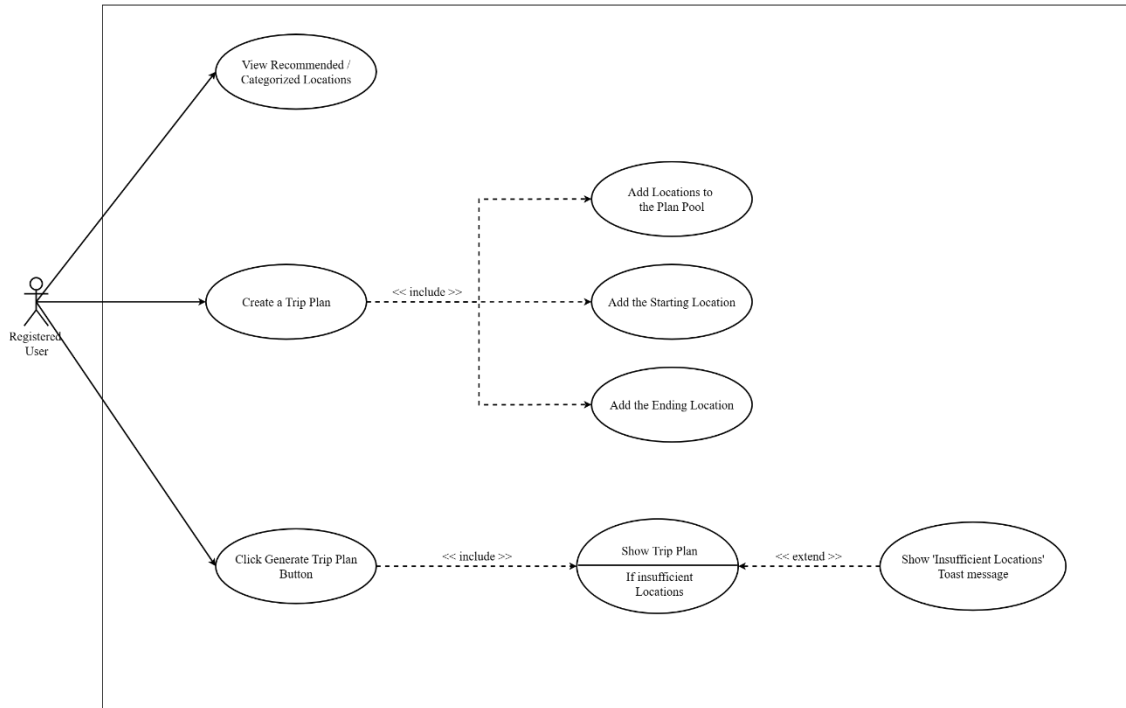


Figure 3 : Planning Use Case Diagram

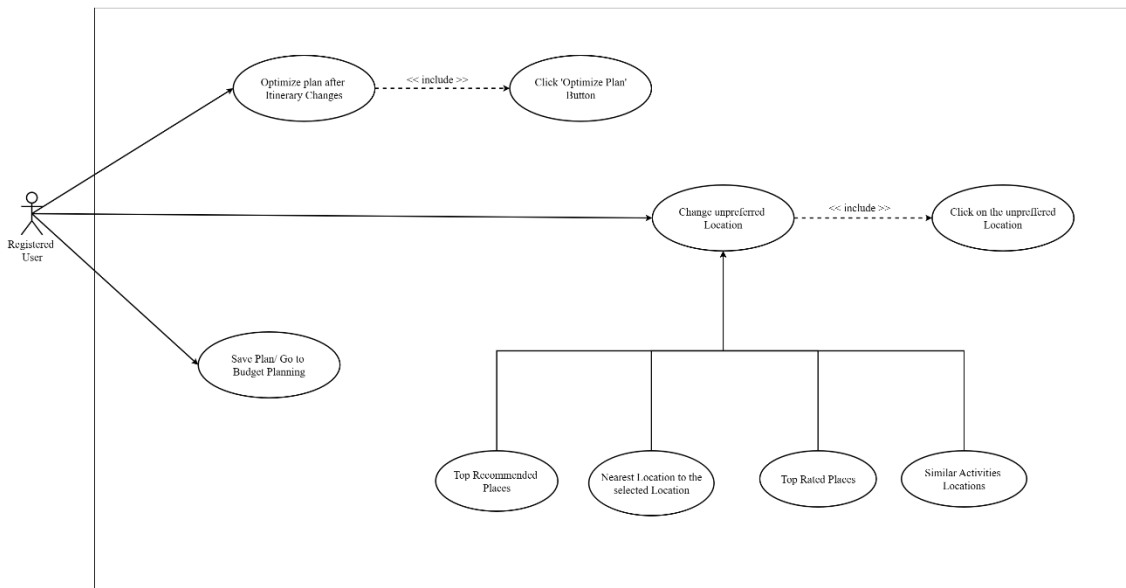


Figure 4 : Itinerary Use case Diagram

2.1.3 System Design

The design phase of this research defines the intelligent architecture of the **AI-Driven Smart Tourism Platform**, which integrates recommendation, plan curation, and itinerary generation into a cohesive, user-centered solution. This phase focuses on the logical flows, model interactions, and modular components required to deliver a responsive, personalized, and sustainable travel planning system.

At this stage of the Software Development Life Cycle (SDLC), abstract ideas about **personalization, hybrid recommendation, and adaptive itinerary building** are translated into concrete components that interact seamlessly. The system's structure is designed with the goal of ensuring personalization, reducing travel planning complexity, and maintaining alignment with modern traveler expectations of flexibility, safety, and sustainability.

The architecture supports the complete user journey from **onboarding and preference collection**, to **receiving hybrid recommendations**, to **shortlisting locations in the Plan Pool**, and finally to **generating customizable itineraries**. Internal modules handle machine learning-based scoring, geospatial corridor validation, and multi-strategy itinerary generation to make the experience adaptive and personalized.

2.1.3.1 System Architecture Overview

The **System Architecture Diagram** illustrates the overall flow of the platform. The process begins when a user registers and submits demographic and preference information. These inputs are processed by the **Hybrid Recommendation Engine**, which fuses collaborative filtering, content-based similarity, and a machine learning model (XGBoost) to rank destinations.

The **Top-N ranked destinations** are presented to the user, who selects preferred options to build a **Plan Pool** (analogous to a shopping cart in e-commerce). The Plan Pool serves as the staging ground for itinerary construction. Once the user specifies a **start city, end city, and corridor radius**, the **Corridor-Aware Plan Generator** validates feasible routes and filters out-of-range locations.

The refined pool is then processed by the **Itinerary Generation Module**, which produces multiple itinerary options across four strategies:

- (i) Similar-Type clustering
- (ii) Similar-Activities clustering
- (iii) Nearby/Geo-proximity grouping
- (iv) Hybrid fusion. These itineraries are delivered to the user for review, customization, and finalization.

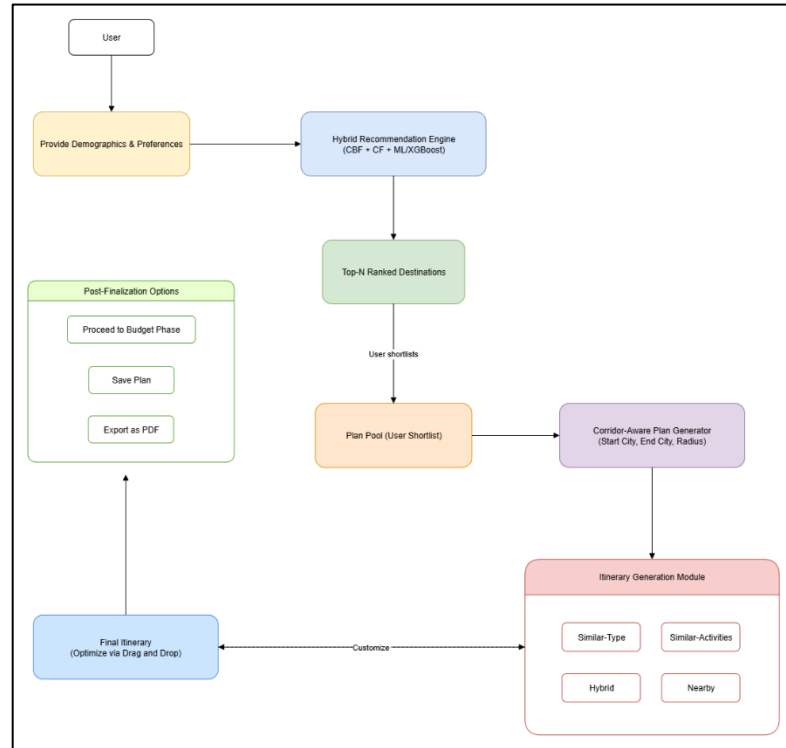


Figure 5 : System Overview Diagram

2.1.3.2 Component Architecture Overview

Each component of the system has been deliberately structured to address a specific requirement in the user journey:

- **Hybrid Recommendation Engine**

This module integrates content-based filtering (CBF), collaborative filtering (CF), and a machine learning model (XGBoost) into a hybrid pipeline. CBF leverages activity overlap and location attributes, CF incorporates demographic and rating-based signals, and the ML model learns optimal weights for fusion. Together, this ensures recommendations are both context-aware and socially validated.

- **Plan Pool**

The Plan Pool allows travelers to curate preferred destinations into a shortlist, ensuring agency and control in the planning process. This pool acts as the central input for itinerary generation, with users able to modify selections before proceeding.

- **Corridor-Aware Plan Generator**

This component introduces geospatial constraints into the planning process. By requiring users to specify start and end cities, along with a travel radius, the system ensures that itineraries are geographically coherent and feasible. Locations outside the corridor are automatically flagged and filtered, while alternatives within range are suggested.

- Itinerary Generation Module

The itinerary builder produces multiple candidate plans, giving users flexibility in choosing the best fit. Four strategies are supported:

- *Similar-Type*: groups locations of the same category (e.g., cultural, adventure).
- *Similar-Activities*: clusters based on overlapping activity profiles.
- *Nearby*: groups by physical proximity using geospatial distance.
- *Hybrid*: integrates all three dimensions into a composite plan.

This module also supports drag-and-drop customization and real-time optimization, enabling users to refine the sequence of activities or remove undesired stops.

- Finalization, Budgeting, and Export

Once the itinerary is optimized, the user enters the finalization stage. At this point, three paths are available:

- **Budget Phase**: the user can proceed to an integrated budget planner, which estimates expenses for transportation, accommodation, meals, and activities.
- **Save Plan**: itineraries can be securely saved to the user's account for future access.
- **Export/Download**: the finalized itinerary can be exported as a PDF document for offline access and sharing.

This ensures the system not only generates intelligent travel routes but also provides practical tools for financial planning, reusability, and portability of travel plans.

2.1.4 Implementation

The Implementation phase transformed the designed architecture into a working AI-driven smart tourism platform that delivers hybrid recommendations, corridor-aware plan validation, and multi-strategy itinerary generation. Development was intentionally modular: each core capability- Recommender Engine, Plan Pool & Corridor Validator, Itinerary Generator, and Frontend Integration- was built and validated in isolation and then integrated via a FastAPI service layer. This approach improved clarity, parallelism, and testability.

Consistent with agile practice, features were built in three sprints with incremental deliverables and continuous backend verification via Postman. Model artifacts (XGBoost and label encoders) were versioned in app/artifacts, ensuring reproducibility across runs and environments.

2.1.4.1 Task Breakdown and Project Management

To manage development systematically, the project followed a sprint-based Agile approach. The entire cycle was divided into three sprints, each lasting about one month, with milestones aligned to backend and frontend development.

- **Sprint 1** focused on preparing the datasets and developing baseline recommenders (Content-Based and Collaborative Filtering). This stage also included initial API setup to validate the recommendation pipeline.
- **Sprint 2** introduced the hybrid XGBoost model and corridor-aware planning modules, along with the Plan Pool feature. At this stage, the frontend was connected to backend APIs for user input and recommendation display.

- **Sprint 3** added itinerary generation, providing four strategies (similar type, activity-based, nearby, hybrid). User-facing features such as drag-and-drop customization, PDF export, and a budget prototype were completed. The final sprint also emphasized system-wide testing and optimization.

2.1.4.2 Work Breakdown Structure (WBS)

The Work Breakdown Structure (WBS) was developed to provide a hierarchical view of the entire system's development effort. At the highest level, the platform was divided into six major modules: Data Preparation, Recommender Engine, Plan Pool & Corridor Validator, Itinerary Generator, Frontend Development, and Integration & Testing. Each of these modules was further decomposed into smaller, more manageable subcomponents, ensuring systematic task allocation and clear progress tracking.

- **Data Preparation:** Covered the curation of datasets (`location_data.csv`, `user_data.csv`, `review_data.csv`), data cleaning, encoding of categorical attributes (age group, gender, companion type, activity sets), and feature engineering to support the recommender pipeline.
- **Recommender Engine:** Subdivided into Content-Based Filtering (CBF), Collaborative Filtering (CF), and Hybrid Fusion (XGBoost). This included data preprocessing scripts, training workflows (`scripts/train_recommender.py`) and the deployment of trained artifacts (`xgb_model.joblib`, `encoders`, `meta.json`).

- **Plan Pool & Corridor Validator:** Consisted of the Plan Pool module, which managed shortlisted destinations, and the TripPlanner, which enforced geographical feasibility using Haversine and cross-track distance calculations. Subcomponents included Pydantic models for request validation and the POST /plan/generate API endpoint.
- **Itinerary Generator:** Focused on building four itinerary strategies (Similar-Type, Similar-Activities, Nearby, and Hybrid). Sub-modules included similarity scoring (cosine, Jaccard), MinMax normalization, hybrid blending logic, and the POST /itinerary/options endpoint.
- **Frontend Development:** Addressed the React-based UI with Tailwind CSS, Mantine, and MUI. Subcomponents included the Preference Input form, Recommendation cards, Plan Pool visualization, Itinerary drag-and-drop editor, and export-to-PDF functionality.
- **Integration & Testing:** Included Postman collections for backend API validation, unit tests for core functions, system-level integration testing of the frontend–backend pipeline, and Docker-based containerization for reproducible deployment.

This structured decomposition allowed for efficient task assignment, dependency mapping, and resource allocation across the project. It also gave a clear roadmap of progress, ensuring that development remained aligned with the system’s overall objectives.

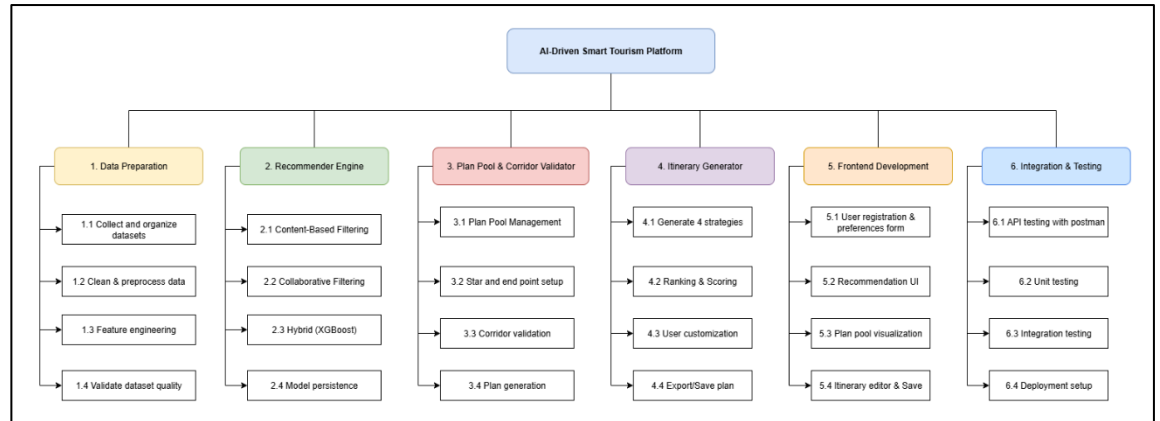


Figure 6 : Work breakdown chart

2.1.4.3 Timeline

Task List	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct
Research Topic Selection												
Feasibility Study												
Evaluation Feasibility & Background Study												
Topic Evaluation												
Requirement Gathering												
Background Survey												
Literature Review												
Requirement Analysis												
Software Requirement Specification												
Functional & Non-functional Requirements												
Project Charter												
Proposal Presentation												
Project Proposal Report												
Software Design												
Designing Wireframes												
ML Component Development												
Frontend Development												
Progress Presentation 01												
Research Paper												
Unit Testing												
Progress Presentation 02												
Software Integration												
Integration Testing												
Deployment & Maintenance												
Final Presentation & Viva												

2.1.4.4 Development Environment and Tools

The implementation of the platform was carried out using a modular, multi-environment setup, ensuring that each component could be developed, tested, and deployed with flexibility.

- **Python Environment Management:** Anaconda was used as the base environment manager. Separate Python environments were created for model training, API development, and testing, ensuring isolation of dependencies. This setup simplified compatibility across packages such as scikit-learn, XGBoost, and FastAPI, while preventing version conflicts.
- **Integrated Development Environments (IDEs):**
 - JetBrains DataSpell was used for machine learning development and API prototyping. Its Jupyter-native workflow simplified experimentation with datasets, visualizing RMSE and R^2 metrics, and exporting trained artifacts.
 - WebStorm was used for frontend development, providing an efficient environment for React, Tailwind, and MUI with built-in debugging, linting, and hot-reloading features.
- **Backend Development:** Implemented in Python 3.12 using FastAPI. FastAPI's modular routing and automatic OpenAPI documentation accelerated API development for routes such as /recommendations, /plan/generate, and /itinerary/options. Uvicorn was employed as the ASGI server for running the APIs.

- **Machine Learning Libraries:** Pandas and NumPy were used for data manipulation, scikit-learn for baseline models and preprocessing, and XGBoost for training the hybrid recommender. Joblib was used to persist trained models and encoders to the app/artifacts/ directory.
- **Frontend Development:** Built using React (JavaScript), styled with Tailwind CSS, and enhanced with Mantine and MUI component libraries for UI consistency. LocalStorage was used to persist session data such as preferences and itineraries.
- **Database and Storage:** While the prototype relies on CSV files for data and artifacts, MongoDB was configured in the environment for future persistence of users, plan pools, and saved itineraries. Its document-based structure is well-suited for flexible user/travel data.
- **Testing Tools: Postman** was used extensively to validate API endpoints and simulate real user requests. Pytest ensured functional correctness of backend modules, while React Testing Library provided confidence in UI components.
- **Deployment Tools:** Docker and docker-compose were used to containerize the backend and frontend, ensuring consistent behavior across development and deployment environments. Environment variables (set via .env and .env.example) configured data and artifact directories.
- **Version Control: GitHub** was used for source control, supporting commits, pull requests, and issue tracking. A GitHub Projects board mirrored sprint tasks and provided visibility into progress.

2.1.4.5 Dataset Preparation

The dataset preparation stage was central to building a reliable hybrid recommendation and itinerary planning system. Since the project targeted Sri Lanka's tourism context, the datasets were carefully designed to reflect user preferences, location metadata, and review interactions. Three core datasets were curated and stored under app/data.

Location Dataset (location_data.csv)

This dataset contained information about tourist attractions across Sri Lanka, including attributes such as:

- Location name, city and province
- Geographic coordinates (latitude, longitude)
- Location type (beaches, historic places, museums etc.)
- Activity tags (swimming, surfing, hiking, tracking etc.)
- Average rating

These attributes were essential for the content-based filtering (CBF) component, allowing the system to compare activity overlap, location categories, and geospatial features.

ID	A	B	C	D	E	F	G	H	I	J	K
	Location_ID	Location_Name	Located_City	Located_Province	Location_Type	Latitude	Longitude	Avg_Rating	Review_Count	Activities	Location_Description
1	LOC_1	Angam Bay	Angam Bay	Eastern Province	Beaches	6.8404	81.6368	4.06	187	["Swimming", "Surfing", "Sunbathing", "Toga", "Boat Tours", "Beach"]	A crescent-shaped stretch of golden sand on Sri Lanka's east coast.
2	LOC_2	Bentota Beach	Bentota	Southern Province	Beaches	6.4235	78.9953	4.58	233	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Jet Skiing", "Boat"]	One of Sri Lanka's most beautiful coastal towns, Bentota Beach is a peaceful paradise on Sri Lanka's southern coast.
3	LOC_3	Hikkaduwa Beach	Hikkaduwa	Southern Province	Beaches	6.1377	80.0991	4.02	213	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Boat Tours"]	Hikkaduwa Beach is a bustling coastal town in southern Sri Lanka, known for its vibrant nightlife and diverse marine life.
4	LOC_4	Jungle Beach	Unawatuna	Southern Province	Beaches	6.0187	80.2394	3.54	183	["Swimming", "Sunbathing", "Snorkeling", "Hiking"]	Nestled near Unawatuna on Sri Lanka's southern coast, Jungle Beach is a peaceful stretch of golden sand on Sri Lanka's southern coast.
5	LOC_5	Kalutara Beach	Kalutara	Western Province	Beaches	6.8896	79.9474	3.67	218	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Kayaking", "Raft"]	Kalutara Beach is a peaceful stretch of golden sand on Sri Lanka's western coast, known for its vibrant nightlife and diverse marine life.
6	LOC_6	Marble Beach	Trincomalee	Eastern Province	Beaches	8.5188	81.2115	3.97	145	["Swimming", "Surfing", "Sunbathing", "Snorkeling"]	Located near Trincomalee in Sri Lanka's northeast, Marble Beach is a tropical paradise on Sri Lanka's eastern coast.
7	LOC_7	Mirissa Beach	Mirissa	Southern Province	Beaches	5.9447	80.4392	4.27	217	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Boat Tours"]	Mirissa Beach is a tropical paradise on Sri Lanka's southern coast, known for its vibrant nightlife and diverse marine life.
8	LOC_8	Mount Lavinia Beach	Colombo	Western Province	Beaches	6.8424	79.8624	4.13	180	["Swimming", "Sunbathing", "Kite Surfing"]	Situated close to Colombo, Mount Lavinia Beach serves as an important beach for the city's residents and visitors.
9	LOC_9	Negombo Beach	Negombo	Western Province	Beaches	7.2443	79.5489	3.12	209	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Kite Surfing", "Boat"]	Negombo Beach lies near Bandarawela International Airport and is a popular destination for tourists.
10	LOC_10	Niuvelli Beach	Niuvelli	Eastern Province	Beaches	6.7003	81.1921	4.33	165	["Swimming", "Surfing", "Sunbathing", "Snorkeling"]	Niuvelli Beach on Sri Lanka's northeastern coast is celebrated for its pristine white sand and clear turquoise waters.
11	LOC_11	Pasikudah Beach	Kalkudah	Eastern Province	Beaches	7.3001	81.5612	4.56	160	["Swimming", "Surfing", "Sunbathing", "Snorkeling", "Jet Skiing"]	Located on Sri Lanka's eastern coast, Pasikudah Beach is known for its calm waters and soft sand.
12	LOC_12	Gregory Lake	Nuwara Eliya	Central Province	Bodies of Water	6.9567	80.7783	3.96	256	["Boating", "Walking Trails"]	A colonial-era artificial lake in Nuwara Eliya, Gregory Lake was created for the British colonialists.
13	LOC_13	Kandy Lake	Kandy	Central Province	Bodies of Water	7.2912	80.6421	3.94	301	["Boating", "Walking Trails"]	Known as Kiri Muhuda (Kiri Muhuda of Milk), Kandy Lake is a man-made lake in the heart of Kandy.
14	LOC_14	Tissa Wewa	Tissamaharama	Southern Province	Bodies of Water	6.3371	80.3854	4.43	91	["Historical Exploration", "Architectural Marvel"]	A 2,000-year-old reservoir in Tissamaharama, Tissa Wewa is an ancient engineering marvel.
15	LOC_15	Twin Baths (Kuttam Pokuna)	Anuradhapura	North Central Province	Bodies of Water	8.3561	80.4088	3.89	191	["Historical Exploration", "Architectural Marvel"]	Located in Anuradhapura, Kuttam Pokuna (Twin Ponds) is an ancient bathing site.
16	LOC_16	Ambewela Farms	Nuwara Eliya	Central Province	Farms	6.8918	80.8039	3.9	136	["Factory Tours", "Agricultural Workshops"]	Perched in Nuwara Eliya's highlands, Ambewela Farms is a hillside tea estate near Ramboda.
17	LOC_17	Bluefield Tea Gardens	Nuwara Eliya	Central Province	Farms	7.0429	80.6979	4.22	314	["Factory Tours", "Agricultural Workshops"]	A hillside tea estate near Ramboda, Bluefield Tea Gardens is a beautiful sight.
18	LOC_18	Dambatenne Tea Factory	Haputale	Uva Province	Farms	6.7634	81.0034	3.96	172	["Factory Tours", "Agricultural Workshops"]	Founded by Sir Thomas Upton in 1890, Dambatenne Tea Factory is a historic site.
19	LOC_19	Damro Labookellie Tea Centre and	Nuwara Eliya	Central Province	Farms	7.0242	80.7193	4.04	90	["Factory Tours", "Agricultural Workshops"]	This tea centre and garden offer a tranquil setting to explore the tea-making process.
20	LOC_20	Glenloch Tea Factory	Katukkulala	Central Province	Farms	7.0785	80.6757	3.82	302	["Factory Tours", "Agricultural Workshops"]	While specific details about the Glenloch Tea Factory might be scarce, it is a notable tea estate.
21	LOC_21	Halpewatte Tea Factory Tour	Elia	Uva Province	Farms	6.8895	81.0345	4.45	264	["Factory Tours", "Agricultural Workshops"]	Located in the heart of Sri Lanka's tea country, Halpewatte Tea Factory is a historic site.
22	LOC_22	Handunugoda Tea Estate	Ahangama	Southern Province	Farms	6.0125	80.3629	4.64	304	["Factory Tours", "Agricultural Workshops"]	The Handunugoda Tea Estate is known for its unique approach to tea cultivation.
23	LOC_23	Pedro Tea Factory	Nuwara Eliya	Central Province	Farms	6.9027	80.7737	3.79	302	["Factory Tours", "Agricultural Workshops"]	The Pedro Tea Factory is another notable destination in Sri Lanka's tea country.
24	LOC_24	Brief Garden - Bevis Bawa	Beruwala	Western Province	Gardens	6.4572	80.0377	4.28	292	["Sightseeing", "Nature Walks", "Photography", "Bird Watching", "Relief"]	Brief Garden, designed by Bevis Bawa, is a beautiful garden in Beruwala.
25	LOC_25	Hagala Botanic Gardens	Nuwara Eliya	Central Province	Gardens	6.5066	80.8215	3.98	248	["Sightseeing", "Nature Walks", "Photography", "Bird Watching", "Relief"]	Relief Garden, designed by Bevis Bawa, is a beautiful garden in Beruwala.
26	LOC_26	New Ranwel Spice Garden	Kandy	Central Province	Gardens	7.2531	80.5938	3.69	212	["Sightseeing", "Nature Walks", "Photography", "Bird Watching", "Relief"]	Relief Garden, designed by Bevis Bawa, is a beautiful garden in Beruwala.
27	LOC_27	Royal Botanical Gardens	Peradeniya	Central Province	Gardens	7.2582	80.5966	4.57	337	["Sightseeing", "Nature Walks", "Photography", "Bird Watching", "Relief"]	Relief Garden, designed by Bevis Bawa, is a beautiful garden in Beruwala.
28	LOC_28	Victoria Park of Nuwara Eliya	Nuwara Eliya	Central Province	Gardens	6.9691	80.7602	3.59	265	["Sightseeing", "Nature Walks", "Photography", "Bird Watching", "Relief"]	Relief Garden, designed by Bevis Bawa, is a beautiful garden in Beruwala.
29	LOC_29	Galle Fort	Galle	Southern Province	Historic Sites	6.0305	80.2151	4.46	283	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Galle Fort is a UNESCO World Heritage Site and a testament to the city's rich history.
30	LOC_30	Jaffna Fort	Jaffna	Northern Province	Historic Sites	9.8651	80.0983	3.8	249	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Jaffna Fort, located in the northern part of Sri Lanka, is a historical site.
31	LOC_31	Upton's Seat	Haputale	Uva Province	Historic Sites	6.7907	81.0155	4.53	237	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Upton's Seat is a high observation point located on the top of a hill.
32	LOC_32	Gali Vihariya	Polonnaruwa	North Central Province	Historic Sites	7.9697	81.0049	4.42	333	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Polonnaruwa is an ancient city in Sri Lanka that served as the capital for several centuries.
33	LOC_33	Rigala Forest Monastery	Anuradhapura	North Central Province	Historic Sites	8.1174	80.6667	4.45	132	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Rigala Forest Monastery is an ancient Buddhist monastery in Anuradhapura.
34	LOC_34	Sigiriya The Ancient Rock Fortress	Sigiriya	Central Province	Historic Sites	7.9571	80.7803	4.65	285	["Sightseeing", "Photography", "Learning History", "Cultural Tours"]	Sigiriya, also known as the Lion Rock, is an ancient rock fortress.
35	LOC_35	Ariyapala Mask Museum	Ambalangoda	Southern Province	Museums	6.2419	80.0503	3.67	77	["Exploring Exhibits", "Learning History", "Cultural Activities", "Guides"]	The Ariyapala Mask Museum is located in Ambalangoda and provides a look into the art of mask-making.
36	LOC_36	Ceylon Tea Museum	Kandy	Central Province	Museums	7.2687	80.6327	4.08	303	["Exploring Exhibits", "Learning History", "Cultural Activities", "Guides"]	The Ceylon Tea Museum is located in Kandy and provides a comprehensive look at the tea industry.
37	LOC_37	Colombo National Museum	Colombo	Western Province	Museums	6.9301	79.8609	4.1	302	["Exploring Exhibits", "Learning History", "Cultural Activities", "Guides"]	The Colombo National Museum is the largest museum in Sri Lanka.
38	LOC_38	Community Tsunami Museum	Hikkaduwa	Southern Province	Museums	6.5885	80.0882	4.67	302	["Exploring Exhibits", "Learning History", "Cultural Activities", "Guides"]	The Community Tsunami Museum is located in Hikkaduwa and commemorates the 2004 tsunami.

Figure 7 : Location Data CSV

User Dataset (user_data.csv)

The user dataset captured demographic profiles that are critical for personalization. Each record included:

- User ID
- Age Group (eg: 18-25, 26-35 etc)
- Gender
- Travel Companion (solo, couple, family, friends)
- Preferred Activities

This dataset formed the foundation for demographic collaborative filtering (CF) and served as contextual input for the hybrid recommender.

User ID	User_Country	User_Age_Group	Gender	Travel_Companion	Location_Type	Preferred_Activities
User_1	Australia	36-50	Male	Family	["Nature & Wildlife Areas", "Gardens", "Beaches"]	["Wildlife Safaris", "Fishing", "Factory Tours", "Hiking"]
User_2	Australia	26-35	Male	Friends	["Nature & Wildlife Areas", "National Parks", "Beaches"]	["Cultural Tours", "Boat Tours", "Sightseeing", "Snorkeling"]
User_3	Australia	18-25	Male	Friends	["National Parks", "Beaches"]	["Boat Tours", "Jet Skiing", "Hiking", "Rafting"]
User_4	Portugal	18-25	Male	Solo	["Museums", "Farms", "Beaches"]	["Yoga", "Photography", "Jet Skiing", "Snorkeling"]
User_5	Italy	18-25	Male	Family	["Religious Sites", "Historic Sites", "Beaches"]	["Rafting", "Wildlife Spotting", "Hiking", "Jet Skiing"]
User_6	United Arab Emirates	18-25	Female	Solo	["Historic Sites", "Beaches"]	["Beach Hopping", "Nature Walks", "Photography", "Hiking"]
User_7	Indonesia	36-50	Male	Friends	["Historic Sites", "Beaches"]	["Historical Exploration", "Hiking", "Walking Trails", "Bird Watching"]
User_8	Canada	18-25	Female	Solo	["Historic Sites", "Beaches", "Bodies of Water"]	["Nature Walks", "Photography", "Beach Hopping", "Snorkeling"]
User_9	New Zealand	36-50	Male	Family	["Beaches", "Historic Sites", "Farms"]	["Agricultural Workshops", "Historical Exploration", "Wildlife Safaris", "Hiking"]
User_10	United Kingdom	51+	Female	Couple	["Museums", "Historic Sites", "Beaches"]	["Cultural Activities", "Meditation", "Relaxation"]
User_11	United Kingdom	18-25	Female	Friends	["Waterfalls", "Beaches", "Bodies of Water"]	["Boat Tours", "Beach Hopping", "Snorkeling", "Surfing"]
User_12	Japan	36-50	Male	Family	["Gardens", "Beaches"]	["Fishing", "Historical Exploration", "Wildlife Safaris", "Hiking"]
User_13	Australia	26-35	Female	Family	["Historic Sites", "Beaches"]	["Photography", "Yoga", "Nature Walks"]
User_14	United Kingdom	26-35	Male	Family	["Religious Sites", "Beaches"]	["Hiking", "Sightseeing", "Cultural Tours"]
User_15	Germany	18-25	Female	Couple	["Beaches", "Zoological Gardens"]	["Relaxation", "Yoga", "Beach Hopping", "Sunbathing"]
User_16	Ukraine	18-25	Female	Solo	["Nature & Wildlife Areas", "Beaches"]	["Photography", "Hiking", "Nature Walks", "Yoga"]
User_17	United Kingdom	26-35	Female	Couple	["National Parks", "Beaches"]	["Sunbathing", "Meditation", "Yoga"]
User_18	India	26-35	Male	Family	["Beaches", "Nature & Wildlife Areas", "Religious Sites"]	["Wildlife Spotting", "Factory Tours", "Cultural Tours"]
User_19	United Kingdom	51+	Male	Friends	["Waterfalls", "Historic Sites", "Beaches", "Museums", "Gardens", "Religious Sites"]	["Meditation", "Wildlife Safaris", "Worship"]
User_20	Viet Nam	51+	Female	Friends	["Religious Sites", "National Parks", "Beaches"]	["Cultural Exploration", "Gardens", "Worship", "Nature Walks"]
User_21	New Zealand	26-35	Male	Friends	["Waterfalls", "Gardens", "Beaches"]	["Snorkeling", "Sightseeing", "Cultural Tours"]
User_22	United States	51+	Female	Friends	["Farms", "Beaches"]	["Gardens", "Worship", "Bird Watching", "Relaxation"]
User_23	United Kingdom	26-35	Male	Couple	["National Parks", "Gardens", "Beaches"]	["Beach Hopping", "Relaxation", "Hiking", "Sightseeing"]
User_24	Ireland	36-50	Male	Friends	["Waterfalls", "Museums", "Beaches"]	["Wildlife Safaris", "Historical Exploration", "Fishing", "Hiking"]
User_25	United Kingdom	51+	Female	Friends	["National Parks", "Beaches"]	["Relaxation", "Cultural Exploration", "Meditation"]
User_26	United Kingdom	26-35	Female	Couple	["Farms", "Beaches"]	["Nature Walks", "Yoga", "Sunbathing", "Meditation"]
User_27	Australia	51+	Female	Solo	["Waterfalls", "Historic Sites", "Beaches"]	["Meditation", "Educational Tours", "Worship", "Sightseeing"]
User_28	United Kingdom	18-25	Female	Friends	["Religious Sites", "Beaches", "Bodies of Water"]	["Snorkeling", "Beach Hopping", "Boat Tours"]
User_29	Austria	36-50	Female	Solo	["Nature & Wildlife Areas", "Beaches"]	["Tea Puckling", "Photography", "Worship", "Educational Tours"]
User_30	Italy	36-50	Female	Couple	["Beaches", "Zoological Gardens"]	["Farm Tours", "Wildlife Spotting", "Cultural Tours"]
User_31	Australia	51+	Male	Solo	["Gardens", "Beaches"]	["Sightseeing", "Meditation", "Photography"]
User_32	United Kingdom	51+	Male	Solo	["Religious Sites", "Beaches"]	["Worship", "Meditation", "Sightseeing"]
User_33	United Kingdom	18-25	Female	Couple	["Museums", "Beaches"]	["Snorkeling", "Sunbathing", "Photography"]
User_34	Czechia	18-25	Female	Couple	["Museums", "National Parks", "Beaches"]	["Yoga", "Sunbathing", "Photography", "Beach Hopping"]
User_35	Australia	36-50	Male	Family	["National Parks", "Beaches"]	["Hiking", "Historical Exploration", "Fishing"]
User_36	United States	36-50	Male	Friends	["Museums", "Beaches"]	["Fishing", "Walking Trails", "Boating"]
User_37	Netherlands	18-25	Female	Family	["Museums", "National Parks", "Religious Sites", "Beaches"]	["Wildlife Spotting", "Beach Hopping", "Sightseeing"]
User_38	Myanmar	18-25	Male	Family	["Historic Sites", "Beaches"]	["Snorkeling", "Rafting", "Surfing"]

Figure 8 : User Data CSV

Review Dataset (review_data.csv)

The review dataset simulated user–location interactions, enabling the collaborative layer of the recommender. Key attributes included:

- User Id
- Location Name
- Rating (1-5 stars)
- Review timestamp

These interactions not only strengthened collaborative signals but also provided supervisory data for training the **XGBoost regressor** used in the hybrid model.

In addition to these, smaller auxiliary datasets (e.g., encoded activity vocabularies and lookup tables) were generated during preprocessing and stored alongside the main datasets. Collectively, these datasets provided the **multi-dimensional signals—content, demographic, and behavioral—that were necessary to train and evaluate the hybrid recommendation engine.**

User ID	Location Name	Rating	Helpful_Vot	Travel_Date	Published_Date	Title	Text
User_1	Arugam Bay	5	1	2019-07	2019-07-31107:53:21-04:00	Best nail spa in Arugam	I had a manicure here and it really was professional and clean. It is right on the Ocean so very relaxing. Suzanne was my consultant and her
User_2	Arugam Bay	4	0	2019-06	2019-07-2121:50:11-04:00	Best for surfing	Overall, it is a wonderful experience. We visited Arugam Bay last month during our anniversary. You will find many things to do from the restaurants to the beautiful v
User_3	Arugam Bay	5	0	2019-07	2019-07-1517:52:55-04:00	We love Arugam Bay	Great place to chill, swim, surf, eat, sleep, have sunset cocktails, nA&C's the best!!! The vibe is laidback and friendly. I would highly recommend to anyone!
User_4	Arugam Bay	5	0	2019-06	2019-07-0317:10:32-04:00	Sun and waves.	Good place for surf and a few stores to go for shopping. Very good place for rest and surf. Good clean beach with warm water. Everyone is friendly and always ar
User_5	Arugam Bay	5	0	2019-07	2019-07-0217:57:02-04:00	Great swimming, surfing	This place is great for surfing but even if you are not a surfer you can have a wonderful time swimming around the bay, visiting the natural reserves, socialising or try
User_6	Arugam Bay	5	4	2019-06	2019-06-12121:25:10-04:00	Raw piece of paradise	Just wrapped up eight days in Sri LankaA&C's Arugam Bay - a rather rare part of the world in that it'sA&C's still relatively raw and untouched by
User_7	Arugam Bay	5	0	2018-07	2019-06-0417:28:37-04:00	Good place for surf and	At for every 2011 year there was not too many surfers and tourists. The place is a bit wild, but good!
User_8	Arugam Bay	5	0	2019-05	2019-05-2021:44:20-04:00	Beautiful surf spots!	Arugam Bay has everything you need, food service and many surf spots to enjoy! I especially loved being on the beach and swimming almost 24/7 and renting a Bike
User_9	Arugam Bay	5	0	2019-04	2019-05-27107:40:12-04:00	Sleep where you can hear!	I spent two nights in Arugam Bay and will be return for another two on our way back South, as we really liked it. Surf was good for
User_10	Arugam Bay	1	0	2018-09	2019-05-14109:28:22-04:00	Beware of sexual assault!	My partner and I have visited Arugam Bay in Sri Lanka a few times as he is a surfer. We found the town very commercial, very catered for tourists
User_11	Arugam Bay	5	0	2019-04	2019-04-2018:42:51-04:00	LOVE LOVE LOVE	Must visit in Sri Lanka. We would stay forever!!!!!! Beautiful town full of beautiful people. It has maintained it's charm despite tourism and surfing industry and the fr
User_12	Arugam Bay	4	0	2019-01	2019-04-16108:38:51-04:00	Really Good Tip	Very relaxed atmosphere, perfect place to relax and enjoy the waves. Great restaurants too. The people are all very friendly and happy to provide information. Nice an
User_13	Arugam Bay	2	1	2019-04	2019-04-15108:16:43-04:00	Expected more	Perhaps it's because it's just the start of the season and also the local new year holiday weekend, but a lot of the bars and restaurants were
User_14	Arugam Bay	1	0	2019-02	2019-02-0817:31:31-05:00	Only place in Sri Lanka w	Arrived here off season and it was deserted. Found ourselves being stared at constantly and as a woman I felt very uncomfortable. We were used to local people star
User_15	Arugam Bay	5	1	2019-01	2019-01-2112:43:41-05:00	Beautiful & Calm	Very beautiful place to relax. Sun rising is something worth to watch. But I think there must be certain rules to prevent disturbing this wonderful place.
User_16	Arugam Bay	5	0	2018-05	2018-11-13174:15:23-05:00	Surf camp Emotionland	???? A good place to first stand on the surf, to feel the ocean. Arugam is a village where nice people live, tasty food, low prices, gorgeous ocean. Chic hummus at The
User_17	Arugam Bay	5	0	2018-07	2018-11-07102:09:00-05:00	End of the road in Sri Land	Simple little village on the coast with walking access to surf for all abilities and a range of accommodation
User_18	Arugam Bay	4	1	2018-10	2018-10-13172:00:41-04:00	Perfect for Wind Surfers.	Though the entry is of a fishing village and looks pretty dirty, when you walk along the shore and reach a point where surfers are enjoying their moment. My kids learn
User_19	Arugam Bay	4	0	2018-02	2018-10-13103:53:24-04:00	Great walk away from the	We visited during the low season. Nonetheless, we enjoyed some excellent beach walks along the right side of the bay.
User_20	Arugam Bay	3	0	2018-10	2018-10-041720:44:46-04:00	Off season dullness	Beautiful stretch of sand with restaurants, bars and resorts.
User_21	Arugam Bay	4	1	2018-08	2018-09-19101:29:26-04:00	Too crowded	The beach is long but quite narrow. There are many small hotels/shacks and restaurants along the beach, some restaurants are really good,
User_22	Arugam Bay	4	3	2018-08	2018-09-18105:41:57-04:00	Worth it if you like surfing!	You get surfers from all over the world during the season from June to September. The main point is not for beginners as there is a reef so be careful if you're still lea
User_23	Arugam Bay	4	2	2018-09	2018-09-06178:47:35-04:00	Clean, but rough sea in	The beach is really clean, lovely golden sand there is plenty of room with out being overcrowded, we had a section literally too ourselves.
User_24	Arugam Bay	5	1	2018-07	2018-06-27107:44:51-04:00	Amazing place for surfing!	What a great place for surfing. My main advice would be that you should book a hotel/hostel: beginners - closer to Bafy Point and advanced - Elephant Rock. Waves
User_25	Arugam Bay	5	2	2018-08	2018-06-25110:06:29-04:00	Relaxing paradise	Great to sit back, relax and chill. Family, teenagers, older persons can find all they need in this part of a beautiful island
User_26	Arugam Bay	4	0	2018-08	2018-06-22125:13:21-04:00	Nice beach, laid back wif	We spent a week here and liked the beach. The kids surfed here despite being beginners and some of the waves being quite large.
User_27	Arugam Bay	5	0	2017-08	2018-06-21107:50:25-04:00	Relaxing	Depending on your interests consider the length of stay. We didn't surf but relaxed very well on the beach for 5 days. Possibly a little to long. Place will develop fast!
User_28	Arugam Bay	5	0	2018-07	2018-06-16104:26:22-04:00	Huge waves	beautiful beach, huge waves and fun. Loved it there, will definitely be back again. Loved our hotel and loved the beach bars along the bay.
User_29	Arugam Bay	5	0	2018-07	2018-07-29105:01:17-04:00	Nice little town for Surfer!	If you surf, go to Arugam Bay. You will find many great surf spots around the town. Easy to access by TukTuk.
User_30	Arugam Bay	4	0	2018-07	2018-07-28106:19:31-04:00	Great surfing place	Is the best spot on the east coast to surf. If youA&C're willing to take a bath you can try but the waves are too strong to let you easily go in the sea, there are a lot of ch
User_31	Arugam Bay	2	0	2018-07	2018-07-27106:30:22-04:00	A&C's Lost ParadiseA&C	With I had of gone 40 years ago. Over cooked now and developing not in a good way. If your going for waves definitely avoid the high season. Ridiculous amount of s
User_32	Arugam Bay	3	6	2018-07	2018-07-24109:18:07-04:00	Sri Lanka's version of Go	Lowly beaches, great surf, chilled out vibe. In the summer A&C's Bay is home to hundreds of European 20-somethings having a great time. It's all about the beach here a
User_33	Arugam Bay	1	7	2018-07	2018-07-21106:46:50-04:00	Okay beach - locals can't	It's a nice place once you get away from the main strip. The locals unfortunately have not yet adapted to the fact that tourism can be fickle if a
User_34	Arugam Bay	5	0	2018-07	2018-07-18117:29:05-04:00	Place you should not mis	Great atmosphere in surf place where everything is dedicated mostly to surf?? People are great and supporting you must try to have at least one lesson of surf??
User_35	Arugam Bay	5	0	2018-07	2018-07-16109:28:49-04:00	When Bali was in the 198	Great location with good selection of surf spots. The beach is great and clean but in July was very hot. Try and stay at a place with a pool if you have young kids. The
User_36	Arugam Bay	2	4	2018-07	2018-07-0617:16:54-04:00	Overrated	Arugam Bay is an amazingly overrated surf spot. Surfers expect a slightly above average wave with 100-200 drunk idiots in the lineup on main
User_37	Arugam Bay	4	1	2018-06	2018-07-01106:50:13-04:00	Very nice looking beach	We stop for one night during our Sri Lanka trip. Beach and restaurants near beach need more than one night for better understanding. Perhaps next time?
User_38	Arugam Bay	4	2	2018-07	2018-06-30118:28:23-04:00	Brilliant stretch of beach	As I'm not a beach person, it's hard to appreciate these kind of places but this was a nice place with an incredibly long stretch of sand. Waves were a bit scary but th

Figure 9 : Review Data CSV

2.1.4.6 Data Processing and Cleaning

The raw datasets required significant preprocessing before they could be used for model training and itinerary generation. This stage ensured data consistency, reliability, and feature readiness, directly influencing the accuracy of recommendations.

- **Handling Missing Values and Duplicates**

- Missing geographic coordinates were removed to maintain the accuracy of distance calculations.
- Duplicate location names were eliminated to avoid skewing similarity computations.
- Invalid or incomplete user records were filtered to ensure consistency across datasets.

- **Categorical Encoding**

- User demographics (age group, gender, travel companion type) were encoded using label encoders, ensuring compatibility with ML models.
- Location types were similarly encoded to facilitate CBF scoring and feature inclusion in the hybrid model.

- **Activity Tokenization and Normalization**

- Activity strings (e.g., “Hiking, Camping, Sightseeing”) were tokenized into sets.
- These sets were used to compute Jaccard similarity for the CBF component.
- Stop-words and inconsistent tags were normalized (e.g., “hike” → “hiking”).

- **Geo-Feature Engineering**

- Latitude and longitude were cast to floating-point values.
- A Haversine utility was implemented to calculate inter-location distances.
- Additional features, such as distance from user's selected base location, were engineered to support itinerary generation.

- **Review Data Aggregation**

- User-location ratings were normalized to [0,1] for consistency across models.
- Statistical features (mean rating per user, variance, number of reviews) were derived to enrich the hybrid recommender's input space.

- **Train-Test Split and Feature Assembly**

- The datasets were merged into a training matrix combining user demographics, activity overlaps, location features, and review statistics.
- The final matrix served as input for training the XGBoost model in the hybrid pipeline.


```

row = {
    'User_Age_Group_encoded': enc_params.get('User_Age_Group', 0),
    'Gender_encoded': enc_params.get('Gender', 0),
    'Travel_Companion_encoded': enc_params.get('Travel_Companion', 0),
    'Location_Name_encoded': ln_enc,
    'Location_Name': loc['Location_Name'],
}

if 'User_Country_encoded' in self.model_features and 'User_Country' in self.encoders:
    row['User_Country_encoded'] = self._safe_encode(self.encoders.get('User_Country'), value: 'Unknown')

if 'Location_Type_encoded' in self.model_features and 'Location_Type_encoded' in loc:
    row['Location_Type_encoded'] = int(loc['Location_Type_encoded'])
if 'Activity_Count' in self.model_features and 'Activity_Count' in loc:
    row['Activity_Count'] = int(loc['Activity_Count'])

# Has_* activity columns
for c in self.model_features:
    if c.startswith('Has_') and c in loc.index:
        row[c] = int(loc[c])

# Activity match ratio on-the-fly
if 'Activity_Match_Ratio' in self.model_features:
    lacts = loc['Activities'] if isinstance(loc['Activities'], list) else []
    row['Activity_Match_Ratio'] = (len(req.intersection(lacts))/(len(req) or 1)) if req else 0.0

# If model expects user rating behavior proxies, fill conservative defaults for single-user scoring
for c in ['Avg_Rating', 'Rating_Count', 'Rating_Variability']:
    if c in self.model_features and c not in row:
        row[c] = 0 if c != 'Avg_Rating' else 3.5

```

Figure 10 : Data Preprocessing and Cleaning

This preprocessing pipeline converts raw user, location, and activity attributes into model-ready feature vectors. Demographic attributes (age group, gender, companion type) are encoded into integers for use in collaborative filtering. Location metadata, including location type and activity counts, enriches the content-based signals. Binary *Has_* columns are populated to represent activity presence (e.g., hiking, camping).

A key hybrid feature, the *Activity_Match_Ratio*, is computed dynamically, quantifying the overlap between user-preferred activities and the activities available at a given location. Finally, rating behavior proxies (*Avg_Rating*, *Rating_Count*, *Rating_Variability*) are safely defaulted to conservative values when missing, ensuring that the recommender remains robust even for users or locations with sparse data.

This implementation ensured that all three layers—Content-Based Filtering, Collaborative Filtering, and Hybrid ML—operated on a consistent, feature-rich input representation, directly supporting the accuracy of the hybrid recommendation engine.

2.1.4.7 Model Selection and Fine-Tuning

The selection and fine-tuning of an appropriate machine learning model was a pivotal step in ensuring that the recommendation engine of the proposed tourism platform could generate accurate, robust, and contextually relevant results. Since the system was designed to integrate **user demographics, location attributes, and behavioral signals** into a single predictive framework, it was necessary to evaluate models that could effectively handle **heterogeneous, tabular datasets**. Two ensemble-based approaches **Random Forest Regression** and **Extreme Gradient Boosting (XGBoost)** were chosen for experimentation due to their proven success in structured data domains and their ability to capture non-linear relationships.

Initially, the **Random Forest** model was tested as a baseline hybrid predictor. Random Forest is known for its robustness against overfitting and its ability to capture complex interactions through multiple decision trees aggregated by bagging. The model demonstrated reasonable performance, achieving stable results on both training and validation datasets. However, during evaluation, it was observed that Random Forest lacked the same level of fine-grained control over regularization and feature weighting. It also required more computational resources as the dataset grew, and its interpretability in terms of feature importance was limited to generic Gini importance scores, which were less informative for this application.

By contrast, the **XGBoost algorithm** demonstrated several advantages that made it more suitable for the hybrid recommender system. Unlike Random Forest, which grows trees independently, XGBoost builds trees sequentially, where each new tree attempts to correct the errors of the previous ensemble. This gradient boosting framework allowed the model to focus on minimizing residuals and improving prediction accuracy iteratively. Furthermore, XGBoost incorporates **built-in**

regularization mechanisms (L1 and L2), which reduced the risk of overfitting and improved the model's generalization capability an essential requirement given the relatively sparse and diverse tourism dataset.

The training process for XGBoost was implemented in the `train_recommender.py` module. The input feature space was constructed by combining **demographic encodings** (age group, gender, travel companion type), **content-based signals** (location type, activity counts, binary Has_* activity features, and the dynamically computed activity match ratio), and **collaborative proxies** (average rating, rating count, and rating variability). These features collectively captured the three perspectives content, collaborative, and hybrid that underpin the recommender system.

The dataset was divided into **training and testing partitions**, with 80% of the data reserved for training and 20% for evaluation. Stratification ensured that the split preserved the balance of location types and activity categories. A comprehensive hyperparameter tuning process was carried out, beginning with a grid search across critical parameters such as the number of estimators, maximum depth of trees, learning rate, and subsampling ratios. After multiple iterations, the best configuration was achieved with 200 estimators, a maximum tree depth of six, a learning rate of 0.1, and a subsample ratio of 0.8. These parameters offered the optimal balance between training time, predictive accuracy, and model complexity.

The trained model was persisted using the Joblib library, with the serialized artifact stored in the `app/artifacts` directory. Alongside the model, all label encoders, feature metadata, and configuration files were saved to guarantee reproducibility and simplify re-deployment. This modular persistence strategy ensures that the model can be seamlessly reloaded into the FastAPI backend for real-time predictions without retraining.

Evaluation metrics confirmed the superiority of XGBoost over Random Forest. While the Random Forest regressor produced acceptable accuracy, the XGBoost model consistently outperformed it, achieving a **Root Mean Squared Error (RMSE) of 0.47** and a **coefficient of determination (R^2) of 0.7739** on the testing dataset. These results demonstrated that XGBoost could capture both linear and non-linear interactions between demographic, content-based, and collaborative signals more effectively than Random Forest.

In addition to predictive performance, the interpretability of XGBoost was another decisive factor in its selection. Feature importance analysis revealed that variables such as **Activity Match Ratio**, **Average Rating**, **Location Type**, and **User Age Group** were among the most influential in shaping the recommendations. This insight not only validated the design of the hybrid feature set but also provided transparency regarding the model's decision-making process, which is crucial in a user-facing tourism application.

To further refine the model, **cross-validation (five-fold)** was performed, which confirmed the stability of results across different data partitions. Early stopping was employed during training to avoid overfitting, with the training process halting automatically after 20 consecutive iterations without improvement in validation error. Regularization parameters were also tuned, with λ (L2 penalty) set to 1 and α (L1 penalty) set to 0.1, striking a balance between flexibility and robustness.

The fine-tuning process concluded with the deployment of the XGBoost model as the core recommendation engine of the system. Its superior accuracy, efficient training process, and interpretability made it the most suitable choice, ensuring that the hybrid recommender could deliver **personalized, reliable, and scalable travel recommendations**.

Table 5 : XGBoost vs RandomForest Comparison

Model	RMSE	R ²	Training Time (s)	Remarks
Random Forest	0.60	0.64	~ 45	Produced stable results but required longer training and higher memory usage. Limited interpretability of feature importance.
XGBoost (Selected)	0.47	0.77	~ 28	Outperformed Random Forest in accuracy and efficiency. Provided interpretable feature importance and stronger generalization.

2.1.4.8 Training Environment Setup

The training environment played a crucial role in ensuring that the hybrid recommendation and itinerary planning system could be developed, evaluated, and deployed consistently across different stages of the Software Development Life Cycle (SDLC). Since the platform integrates machine learning components with a web-based frontend and backend, the environment was carefully structured to balance computational efficiency, modular development, and reproducibility.

Hardware Environment

All development and training activities were conducted on a personal workstation equipped with an **Intel® Core™ i5 processor, 24 GB of RAM, and SSD-based storage**. Relatively high memory allocation was essential for handling multiple datasets in memory simultaneously, performing iterative preprocessing operations, and training the hybrid recommendation models without encountering bottlenecks. While no dedicated GPU was available, the CPU-based setup proved sufficient given the size of the datasets and the efficiency of the XGBoost framework.

Software Environment

To ensure modularity and reproducibility, the software environment was organized into separate virtual environments and development tools.

1. Anaconda Environments

The Python-based machine learning models were developed within Anaconda-managed environments, allowing for clear isolation of dependencies.

- One environment was dedicated to data preprocessing and model training, with packages such as pandas, scikit-learn, and xgboost.
- Another lightweight environment was maintained for API integration and testing, running FastAPI, Uvicorn, and related libraries. This separation prevented dependency conflicts and streamlined the testing of models within the backend.

2. **JetBrains DataSpell**

The machine learning pipeline was prototyped and tested in JetBrains DataSpell, which provided an integrated workspace for working with Jupyter-style notebooks, visualizing dataframes, and debugging preprocessing functions. DataSpell was particularly useful during the experimentation phase, as it allowed quick iterations on feature engineering and evaluation of model performance.

3. **WebStorm for Frontend Development**

The frontend development of the platform was carried out using WebStorm, optimized for JavaScript and React-based projects. WebStorm provided built-in linting, live preview, and seamless integration with package managers (npm/yarn), making it easier to develop, test, and refine UI components such as the preference forms, itinerary editor, and plan pool visualizations.

4. **Docker**

For deployment consistency, Docker was employed to containerize both the backend and frontend services. By packaging the application into Docker images, the system could be run in isolated environments with guaranteed dependency alignment, ensuring that results observed during development would remain consistent during testing and future deployment.

5. **MongoDB**

Although the current prototype primarily relied on CSV datasets, MongoDB was included in the environment setup to prepare for future extensions where persistent user data and larger-scale location datasets would be required. This ensured that the platform's architecture remained scalable and capable of supporting real-world tourism applications.

2.1.4.9 System Requirements

Table 6 : System Requirements

Category	Specification
Processor	Intel ® Core ™ i5, 4-core
RAM	24 GB DDR4
Storage	256 GB SSD + 1 TB HDD
Operating System	Windows 11 Pro (64-bit)
Development IDE's	JetBrains DataSpell (ML), WebStorm (Frontend), VS Code (general purpose)
Virtual Environments	Anaconda (Python ML, FastAPI backend)

2.1.4.10 Other Configurations

In addition to the primary development and training processes, several supporting configurations were implemented to ensure that the system remained secure, maintainable, and user-friendly during deployment and usage. These configurations addressed aspects such as environment variable management, cross-origin communication, logging, error handling, and frontend data persistence. Together, they strengthened the robustness of the platform and facilitated a seamless integration between system components.

Environment Variables

To avoid hard-coding sensitive information and configuration paths within the source code, the platform employed environment variable management. A `.env` file was created at the project root to store configurable parameters such as dataset directories, artifact paths, server ports, and MongoDB connection placeholders. An `.env.example` file was also included to guide future developers on how to replicate the setup without exposing actual credentials.

This strategy ensured that the application could be run across different environments (development, testing, production) without requiring source code modifications. By adhering to the Twelve-Factor App principle of strict separation between code and configuration, the system was made more portable and maintainable.

Cross-Origin Resource Sharing (CORS) Setup

Since the frontend and backend operated on different ports during development (React on `localhost:3000`, FastAPI on `localhost:8000`), it was necessary to enable Cross-Origin Resource Sharing (CORS) in the backend. FastAPI's CORS middleware was configured to accept requests from the frontend domain, allowing secure communication between the two layers.

The configuration restricted origins to trusted domains, ensuring that only authenticated and intended clients could interact with backend endpoints. This not only facilitated smooth API integration but also added a layer of protection against malicious cross-origin requests.

Logging and Error Handling

Robust logging mechanisms were introduced to monitor application behavior and assist with debugging during both development and runtime. The backend was configured to log:

- All incoming API requests (endpoint, method, timestamp).
- Model prediction errors or missing data handling.
- Validation errors in itinerary generation or corridor validation.

Error-handling middleware was integrated into the FastAPI backend to ensure that exceptions were returned in a structured JSON format, enabling the frontend to display user-friendly error messages. This minimized disruptions to the user experience while providing developers with clear diagnostic information.

Health Check Endpoint

To verify the operational status of the backend services, a lightweight health check endpoint (`/healthz`) was implemented. This endpoint returned a simple JSON response indicating the availability of the backend and the successful loading of key artifacts (e.g., XGBoost model, encoders). Such a feature is especially useful in containerized environments (Docker), where automated monitoring tools can periodically poll the endpoint to ensure the system remains healthy.

2.1.4.11 Module Implementation

a) Recommendation Engine (Hybrid)

This module produces personalized Top-N location recommendations by blending three signals: content-based similarity (activities & type), collaborative proxies (ratings & usage), and a learned hybrid model (XGBoost). It's exposed through a FastAPI endpoint, consumed by the React UI's Preferences → Recommendations flow.

Implementation. The engine lives across two key files:

- **Model & Artifacts:** app/models/recommender.py
- **API layer:** app/api/routes_recommendation.py

Model loading & artifacts

Artifacts (XGBoost model, label encoders, feature list) are loaded via a single factory function. This keeps the API hot-start logic simple and ensures the encoders/features align with training.

```
def load_recommender_from_artifacts() → 'ImprovedRecommender':
    rec = ImprovedRecommender()
    rec.load_data()
    rec.xgb_model = load_joblib('xgb_model')

    # load encoders
    for key in ['User_Age_Group', 'Gender', 'Travel_Companion', 'User_Country', 'Location_Type']:
        le = load_joblib(f'le_{key}')
        if le is not None:
            rec.encoders[key] = le

    rec.le_locname = load_joblib('le_locname')
    mf = load_joblib('model_features')
    if mf is not None:
        rec.model_features = mf

    # ensure encoded column exists at inference time
    rec._ensure_locname_encoded()
    return rec
```

Figure 11 : Loading the ImprovedRecommender model along with its saved encoders and features from artifacts.

Core recommendation logic

The public `recommend(...)` method sanitizes inputs, computes CBF/CF features, predicts ML scores (when the model is present), blends them with weights, and returns a ranked list.

```
def recommend(self, age_group: str, gender: str, travel_companion: str,
              preferred_activities: List[str], top_n: int=10) → Dict[str, Any]:
    # ensure names encoded (covers API path: load → predict)
    self._ensure_locname_encoded()

    age_group = self._clean_str(age_group)
    gender = self._clean_str(gender)
    travel_companion = self._clean_str(travel_companion)
    preferred_activities = [self._clean_str(a) for a in (preferred_activities or []) if self._clean_str(a)]

    # encode user params safely (UI: only these 3 fields)
    enc_params = {
        "User_Age_Group": self._safe_encode(self.encoders.get("User_Age_Group"), age_group),
        "Gender": self._safe_encode(self.encoders.get("Gender"), gender),
        "Travel_Companion": self._safe_encode(self.encoders.get("Travel_Companion"), travel_companion),
    }

    # Phase 1: CBF
    cbf = self._find_matching_locations(preferred_activities)
    if cbf.empty():
        return {"weights": self.weights, "results": []}

    # Phase 2: CF with progressive relaxation
    # (exact → AG → GT → G → ALL)
    self._find_similar_users(enc_params, preferred_activities)
    cf = self._get_ratings_for_locations(cbf["Location_Name"].tolist())
    if cf.empty():
        # relax once
        enc_params_relaxed = enc_params.copy()
        self._find_similar_users(enc_params_relaxed, preferred_activities)
        cf = self._get_ratings_for_locations(cbf["Location_Name"].tolist())

    # Phase 3: ML
    ml = self._get_enhanced_ml_predictions(enc_params, preferred_activities, cbf["Location_Name"].tolist())

    # Combine
    out = cbf.merge(cf, on="Location_Name", how="left").merge(ml, on="Location_Name", how="left")
    out["CF_Score"] = out["CF_Score"].fillna(0.0)
    out["ML_Score"] = out["ML_Score"].fillna(0.0)

    if out["CF_Score"].sum() == 0:
        w = {"cbf": 0.6, "cf": 0.0, "ml": 0.4}
    elif out["ML_Score"].sum() == 0:
        w = {"cbf": 0.6, "cf": 0.4, "ml": 0.0}
    else:
        w = self.weights

    out["Final_Score"] = (w["cbf"]*out["CBF_Score"]) + (w["cf"]*(out["CF_Score"]/5.0)) + (w["ml"]*(out["ML_Score"]/5.0))
    out = out.sort_values((by="Final_Score", ascending=False).head(top_n)
    return {
        "weights": w,
        "results": out[["Location_Name", "CBF_Score", "CF_Score", "ML_Score", "Final_Score"]].to_dict(orient="records"),
    }
```

Figure 12 : A function for generating personalized travel recommendations based on user inputs.

API endpoint (FastAPI)

The /recommendations route parses the request, ensures artifacts are loaded, validates activities, invokes the recommender, and normalizes the output for the frontend.

```
@router.post("/recommendations")
def post_recommendations(body: RecoIn):

    rec = load_recommender_from_artifacts()
    # safety: ensure artifacts are loaded
    if getattr(rec, "xgb_model", None) is None:
        raise HTTPException(status_code=400, detail="Model artifacts not found. Please train first.")

    acts = _parse_activities(body.preferred_activities)
    if not acts or len(acts) < 2:
        raise HTTPException(status_code=400, detail="preferred_activities must contain at least 2 items")

    # ---- compute (unchanged) ----
    out = rec.recommend(
        age_group=body.age_group,
        gender=body.gender,
        travel_companion=body.travel_companion,
        preferred_activities=acts,
        top_n=body.top_n,
    )

    # The recommender typically returns {"weights":..., "results":[...]}
    # We will enrich 'results' if present; otherwise return as-is.
    if isinstance(out, dict) and "results" in out:
        results = out.get("results") or []
        # normalize to list[dict]
        if isinstance(results, pd.DataFrame):
            items = results.to_dict(orient="records")
        elif isinstance(results, list):
            items = [dict(x) for x in results]
        else:
            items = []
        out["results"] = _enrich_results(rec, items)
        return {"status": "ok", **out}

    # alternative protection if someone returns a DataFrame directly
    if isinstance(out, pd.DataFrame):
        items = out.to_dict(orient="records")
        return {"status": "ok", "results": _enrich_results(rec, items)}

    # last resort: pass-through
    return {"status": "ok", **out if isinstance(out, dict) else {"results": out}}
```

Figure 13 : A function for handling travel recommendation requests through an API endpoint.

b) Plan Pool & Corridor Validator

This module accepts a user-selected plan pool (shortlist of attractions) and produces a feasible route from a start location to an end location while respecting a province corridor and a geometric corridor (start - end line with configurable radius). The validator penalizes out-of-corridor candidates and favors forward progress toward the destination, yielding a sensible, travel-efficient order. The module exposes two endpoints: `/plan/generate` (construct an itinerary from a plan pool) and `/plan/optimize` (re-order an already built itinerary with updated penalties/radius).

Implementation Overview.

All core planning logic is implemented in `app/models/plan.py` within the `TripPlanner` class, while HTTP handlers live in `app/api/routes_plan.py`. The algorithm blends:

1. a **province corridor** (start province - intermediate provinces - end province),
2. a **geodesic corridor** (start - end great-circle path)
3. a **greedy forward-progress heuristic** with **heading and corridor penalties**.

Geodesic & Corridor Math (cross-track distance)

To keep candidates near the main travel axis, the planner computes the cross-track distance from each candidate point to the start-end segment. This uses the haversine/bearing formulation and returns the minimum distance in kilometers; points outside the allowed radius incur penalties.

```

def haversine_km(lat1: float, lon1: float, lat2: float, lon2: float) → float:
    to_rad = math.radians
    dlat = to_rad(lat2 - lat1)
    dlon = to_rad(lon2 - lon1)
    a = math.sin(dlat / 2) ** 2 + math.cos(to_rad(lat1)) * math.cos(to_rad(lat2)) * math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R_EARTH_KM * c

def distance_point_to_segment_km(px: float, py: float, ax: float, ay: float, bx: float, by: float) → float:
    d13 = _ang_distance_rad(ax, ay, px, py)
    d12 = _ang_distance_rad(ax, ay, bx, by)
    if d12 == 0:
        return haversine_km(px, py, ax, ay)
    θ12 = _bearing_rad(ax, ay, bx, by)
    θ13 = _bearing_rad(ax, ay, px, py)
    xt = math.asin(math.sin(d13) * math.sin(θ13 - θ12)) * R_EARTH_KM
    at = math.acos(max(-1.0, min(1.0, math.cos(d13) / max(1e-12, math.cos(xt / R_EARTH_KM))))) * R_EARTH_KM
    total = d12 * R_EARTH_KM
    if at < 0:
        return haversine_km(px, py, ax, ay)
    if at > total:
        return haversine_km(px, py, bx, by)
    return abs(xt)

```

Figure 14 : Geodesic utilities used to constrain candidates to a corridor

Greedy forward-progress ordering (with penalties)

Candidates are greedily ordered to advance toward the end city. The cost function combines straight-line distance to the current point, a heading penalty (discourage large turns away from end bearing), and an out-of-corridor penalty (if cross-track distance exceeds the allowed radius).

```

def _forward_order(
    self,
    items: List[Dict[str, Any]],
    start_lat: float, start_lng: float,
    end_lat: float, end_lng: float,
    penalty_forward_km: float,
    penalty_heading_perc: float,
    corridor_start_lat: Optional[float] = None,
    corridor_start_lng: Optional[float] = None,
    corridor_end_lat: Optional[float] = None,
    corridor_end_lng: Optional[float] = None,
    corridor_radius_km: Optional[float] = None,
    corridor_penalty_km: Optional[float] = None,
) → List[Dict[str, Any]]:

```

Figure 15 : List of stops, start/end locations, and optional limits to guide the path.

```

while pool:
    d_end_cur = haversine_km(cur_lat, cur_lng, end_lat, end_lng)
    head_to_end = initial_bearing_deg(cur_lat, cur_lng, end_lat, end_lng)
    best_idx, best_cost = -1, float("inf")

    for i, cand in enumerate(pool):
        d = haversine_km(cur_lat, cur_lng, cand["lat"], cand["lng"])
        d_end_next = haversine_km(cand["lat"], cand["lng"], end_lat, end_lng)
        backstep = max(0.0, d_end_next - d_end_cur)
        head_to_cand = initial_bearing_deg(cur_lat, cur_lng, cand["lat"], cand["lng"])
        hdiff = heading_diff_deg(head_to_end, head_to_cand)
        angle_penalty = (hdiff / 180.0) * penalty_heading_perc * d

        cost = d + (penalty_forward_km * backstep) + angle_penalty

        if use_corridor:
            d_corr = distance_point_to_segment_km(
                cand["lat"], cand["lng"],
                float(corridor_start_lat), float(corridor_start_lng),
                float(corridor_end_lat), float(corridor_end_lng)
            )
            if d_corr > float(corridor_radius_km):
                cost += float(corridor_penalty_km) * (d_corr - float(corridor_radius_km))

        if cost < best_cost:
            best_cost, best_idx = cost, i

    chosen = pool.pop(best_idx)
    ordered.append(chosen)
    cur_lat, cur_lng = chosen["lat"], chosen["lng"]

return ordered

```

Figure 16 : Algorithm segment for route optimization

Plan generation (from plan pool to itinerary)

The public `plan(...)` method is the entry point for `/plan/generate`. It resolves the start/end (by coords or city name), derives the **province corridor**, optionally augments with **city attractions**, runs `forward_order()`, and returns the final itinerary plus summary stats.


```

def plan(
    self,
    start_city: str,
    end_city: str,
    plan_pool: List[str],
    include_city_attractions: bool = False,
    city_attraction_radius_km: float = 15.0,
    min_attractions: int = 3,
    corridor_radius_km: float = 100.0,
    start_lat: Optional[float] = None,
    start_lng: Optional[float] = None
) → Dict[str, Any]:
    if not end_city:
        return {"error": "End city must be provided."}
    if not plan_pool:
        return {"error": "Plan pool is empty. Add locations first."}

    # resolve start
    if start_lat is not None and start_lng is not None:
        start = {
            "name": start_city or "Current Location",
            "lat": float(start_lat),
            "lng": float(start_lng),
            "province": self._infer_province_from_coords(float(start_lat), float(start_lng)),
            "type": "City",
            "source": "coords"
        }

    total_distance = sum(s.get("distance_from_prev", 0.0) for s in itinerary if "distance_from_prev" in s)
    return {
        "itinerary": itinerary,
        "total_distance_km": total_distance,
        "province_corridor": corridor,
        "is_day_trip": same_city,
        "attractions_count": attraction_count,
        "start": {"name": start["name"], "province": start["province"], "lat": start["lat"], "lng": start["lng"]},
        "end": {"name": end["name"], "province": end["province"], "lat": end["lat"], "lng": end["lng"]},
        "corridor_radius_km": corridor_radius_km
    }

```

Figure 17 : Function for creating a travel plan: checks inputs, sets start and end points, and calculates total distance with optional attractions.

API Endpoints

The Plan Pool & Corridor Validator transforms a user's shortlisted attractions into a coherent route that respects both geographical constraints and travel efficiency. The province corridor constrains selection across administrative regions, while the geodesic corridor (computed via cross-track distance) keeps candidates near the main travel axis.

A greedy forward-progress heuristic with tunable penalties encourages steady movement toward the destination and avoids detours. The module surfaces two endpoints `/plan/generate` and `/plan/optimize` to first construct and then refine an itinerary. Together, these mechanisms provide users with an interactive, constraint-aware planning experience aligned with practical travel behavior.

```
@router.post("/generate")
def generate_plan(body: PlanRequest):
    planner = TripPlanner()
    result = planner.plan(
        start_city=body.start_city or "",
        end_city=body.end_city,
        plan_pool=body.plan_pool,
        include_city_attractions=body.include_city_attractions,
        city_attraction_radius_km=body.city_attraction_radius_km,
        min_attractions=body.min_attractions,
        corridor_radius_km=body.corridor_radius_km,
        start_lat=body.start_lat,
        start_lng=body.start_lng
    )
    if "error" in result:
        raise HTTPException(status_code=400, detail=result)
    return {"status": "ok", **result}
```

Figure 18 : API endpoint to generate a new trip plan based on user input

```

@router.post("/optimize")
def optimize_plan(body: OptimizeRequest):
    planner = TripPlanner()
    result = planner.optimize(
        itinerary=body.itinerary,
        corridor_radius_km=body.corridor_radius_km,
        penalty_forward_km=body.penalty_forward_km,
        penalty_heading_perc=body.penalty_heading_perc,
        corridor_penalty_km=body.corridor_penalty_km,
    )
    if "error" in result:
        raise HTTPException(status_code=400, detail=result)
    return {"status": "ok", **result}

```

Figure 19 : API endpoint to optimize a given trip itinerary.

c) Itinerary Generator

This module generates multiple candidate itineraries around a user-chosen base location. Unlike the plan pool (which sequences a shortlist from start - end), this component explores the broader search space to surface alternative options by similarity signals Hybrid (weighted fusion), Same-Type, Similar-Activities, Top-Rated, and Nearby while respecting province corridor and an optional geodesic corridor radius. It powers the UI's "Itinerary Options" pane and feeds the drag-and-drop editor.

Similarity blocks and hybrid fusion

The module constructs four normalized similarity blocks and then fuses them into a hybrid score using α , β , γ , δ weights:

- Cosine on rating signals (Avg_Rating, Review_Count)
- Jaccard on activity sets (user-facing tags)
- Geodesic distance (Haversine)
- Activities overlap matrix (set-based)

```

def _build_similarity_blocks(self):
    scaler = MinMaxScaler()

    cos_mat = cosine_similarity(self.df[["Avg_Rating", "Review_Count"]].values)
    self.cos_df = pd.DataFrame(cos_mat, index=self.df["Location_Name"], columns=self.df["Location_Name"])

    cat = self.df[["Location_Type", "Located_Province"]].astype(str).apply(lambda x: ", ".join(x), axis=1)
    binmat = cat.str.get_dummies(sep=",")
    jacc = 1 - squareform(pdist(binmat, metric="jaccard"))
    self.jac_df = pd.DataFrame(jacc, index=self.df["Location_Name"], columns=self.df["Location_Name"])

    coords = self.df[["Latitude", "Longitude"]].values
    dist = np.zeros((len(coords), len(coords)))
    for i, (la1, lo1) in enumerate(coords):
        for j, (la2, lo2) in enumerate(coords):
            dist[i, j] = haversine_km(la1, lo1, la2, lo2)
    dist_sim = 1 / (1 + dist)
    self.dist_df = pd.DataFrame(dist_sim, index=self.df["Location_Name"], columns=self.df["Location_Name"])

    names = self.df["Location_Name"].tolist()
    sets = self.df["Activities"].apply(_clean_acts)
    act_mat = np.zeros((len(names), len(names)))
    for i in range(len(names)):
        a = sets.iloc[i]
        for j in range(len(names)):
            b = sets.iloc[j]
            if not a and not b:
                sim = 1.0
            else:
                sim = len(a & b) / len(a | b) if (a | b) else 0.0
            act_mat[i, j] = sim
    self.act_df = pd.DataFrame(act_mat, index=names, columns=names)

    self.cos_n = pd.DataFrame(scaler.fit_transform(self.cos_df), index=self.cos_df.index, columns=self.cos_df.columns)
    self.jac_n = pd.DataFrame(scaler.fit_transform(self.jac_df), index=self.jac_df.index, columns=self.jac_df.columns)
    self.dist_n = pd.DataFrame(scaler.fit_transform(self.dist_df), index=self.dist_df.index, columns=self.dist_df.columns)
    self.act_n = pd.DataFrame(scaler.fit_transform(self.act_df), index=self.act_df.index, columns=self.act_df.columns)

    # weights from your notebook/py
    self.alpha, self.beta, self.gamma, self.delta = 0.2, 0.4, 0.2, 0.2
    self.hybrid = (self.alpha * self.jac_n) + (self.beta * self.cos_n) + (self.gamma * self.dist_n) + (self.delta * self.act_n)

```

Figure 20 : Function to build similarity blocks by comparing locations using distance, activities, and reviews.

Corridor filter and universal radius cap

The generator supports two spatial constraints:

1. **Province corridor** (e.g., from /plan/generate)
2. **Geodesic corridor** (optional): a point is inside the corridor if its cross-track distance to the start - end segment is $< \text{corridor_radius_km}$.

```

def haversine_km(lat1, lon1, lat2, lon2):
    to_rad = math.radians
    dlat = to_rad(lat2 - lat1); dlon = to_rad(lon2 - lon1)
    a = math.sin(dlat/2)**2 + math.cos(to_rad(lat1))*math.cos(to_rad(lat2))*math.sin(dlon/2)**2
    return 2 * R_EARTH_KM * math.atan2(math.sqrt(a), math.sqrt(1-a))

3 usages last-chaanz

def _clean_acts(s: str) → set:
    if not isinstance(s, str):
        return set()
    return {t.strip().lower() for t in s.split(",") if t.strip()}

# ---- corridor helpers (same formulas as planner) ----

2 usages last-chaanz

def _ang_distance_rad(lat1: float, lon1: float, lat2: float, lon2: float) → float:
    to_rad = math.radians
    dlat = to_rad(lat2 - lat1); dlon = to_rad(lon2 - lon1)
    a = math.sin(dlat/2)**2 + math.cos(to_rad(lat1))*math.cos(to_rad(lat2))*math.sin(dlon/2)**2
    return 2 * math.asin(min(1.0, math.sqrt(a)))

2 usages last-chaanz

def _bearing_rad(lat1: float, lon1: float, lat2: float, lon2: float) → float:
    φ1, φ2 = math.radians(lat1), math.radians(lat2)
    Δλ = math.radians(lon2 - lon1)
    y = math.sin(Δλ) * math.cos(φ2)
    x = math.cos(φ1) * math.sin(φ2) - math.sin(φ1) * math.cos(φ2) * math.cos(Δλ)
    θ = math.atan2(y, x)
    return (θ + 2*math.pi) % (2*math.pi)

1 usage last-chaanz

def distance_point_to_segment_km(px: float, py: float, ax: float, ay: float, bx: float, by: float) → float:
    """Cross-track distance from P to great-circle AB, clamped to the segment. Returns km."""
    d13 = _ang_distance_rad(ax, ay, px, py)
    d12 = _ang_distance_rad(ax, ay, bx, by)
    if d12 == 0:
        return haversine_km(px, py, ax, ay)
    θ12 = _bearing_rad(ax, ay, bx, by)
    θ13 = _bearing_rad(ax, ay, px, py)
    xt = math.asin(math.sin(d13) * math.sin(θ13 - θ12)) * R_EARTH_KM
    at = math.acos(max(-1.0, min(1.0, math.cos(d13) / max(1e-12, math.cos(xt / R_EARTH_KM))))) * R_EARTH_KM
    total = d12 * R_EARTH_KM
    if at < 0:
        return haversine_km(px, py, ax, ay)
    if at > total:
        return haversine_km(px, py, bx, by)
    return abs(xt)

```

Figure 21 : Helper functions for calculating distances, bearings, and coordinates used in travel planning.

API-friendly options generator

The public entrypoint `options_for_location()` returns a bundle containing Hybrid, Same-Type, Top-Rated, Nearby, and Similar-Activities lists for the given base location. It applies province filters, the optional geodesic corridor, and a universal distance cap around the base (so all buckets respect the same `radius_km`).

```
def options_for_location(
    self,
    location_name: str,
    included_provinces: Optional[List[str]] = None,
    radius_km: float = 50.0,      # now applied to ALL buckets (not just "nearby")
    top_n: int = 5,
    # corridor filter (optional)
    start_lat: Optional[float] = None,
    start_lng: Optional[float] = None,
    end_lat: Optional[float] = None,
    end_lng: Optional[float] = None,
    corridor_radius_km: Optional[float] = None
) → Dict[str, Any]:
    if location_name not in self.hybrid.index:
        return {"error": f"Location '{location_name}' not found."}

    base = self.df[self.df["Location_Name"] == location_name].iloc[0]
    if included_provinces is None:
        included_provinces = [base["Located_Province"]]

    # Corridor mask (optional)
    if all(v is not None for v in [start_lat, start_lng, end_lat, end_lng, corridor_radius_km]):
        @last-chaanz
        def in_corr(lat, lng):
            d = distance_point_to_segment_km(lat, lng, float(start_lat), float(start_lng), float(end_lat), float(end_lng))
            return d ≤ float(corridor_radius_km)
        self.df["_in_corridor"] = self.df.apply(lambda r: in_corr(r["Latitude"], r["Longitude"]), axis=1)
    else:
        self.df["_in_corridor"] = True

    # Distance-from-base (for universal radius cap)
    self.df["_dist_from_base"] = self.df.apply(
        lambda r: haversine_km(base["Latitude"], base["Longitude"], r["Latitude"], r["Longitude"]),
        axis=1
    )

    # Universal allowed mask across all buckets
    allowed_mask = (
        (self.df["Location_Name"] == location_name) &
        (self.df["Located_Province"].isin(included_provinces)) &
        (self.df["_in_corridor"]) &
        (self.df["_dist_from_base"] ≤ float(radius_km))
    )
```

Figure 22 : Function to find travel options for a location, applying distance limits and optional corridor filters.

```

def options_for_location(
    last_chanz
    def enrich_entry(row: pd.Series, extra: Dict[str, Any]) → Dict[str, Any]:
        e = {
            "location": row["Location_Name"],
            "avg_rating": float(row["Avg_Rating"]),
            "province": row["Located_Province"],
            "type": row["Location_Type"],
            "distance_from_base_km": float(row["_dist_from_base"])
        }
        if end_lat is not None and end_lng is not None:
            e["distance_to_end_km"] = float(haversine_km(row["Latitude"], row["Longitude"], float(end_lat), float(end_lng)))
        e.update(extra)
        return e

    # 1) Hybrid (scan enough to fill after filtering)
    sim_series = self.hybrid[location_name].drop(index=location_name, errors="ignore").sort_values(ascending=False)
    hybrid_list = []
    for loc in sim_series.index[: top_n * 20]: # oversample, then filter
        row = self.df[self.df["Location_Name"] == loc]
        if row.empty:
            continue
        r = row.iloc[0]
        if not allowed_mask.loc[r.name]:
            continue
        hybrid_list.append(enrich_entry(r, extra={"hybrid_similarity": float(self.hybrid.at[loc, location_name])}))
        if len(hybrid_list) ≥ top_n:
            break

    # 2) Same type
    same_type_rows = self.df[allowed_mask & (self.df["Location_Type"] == base["Location_Type"])] \
        .sort_values(by=["Avg_Rating"], ascending=False) \
        .head(top_n)
    same_type = [enrich_entry(r, extra={}) for _, r in same_type_rows.iterrows()]

    # 3) Nearby (already ensured by allowed_mask ≤ radius_km)
    near_rows = self.df[allowed_mask].copy()
    near_rows = near_rows.sort_values("_dist_from_base").head(top_n)
    nearby = [enrich_entry(r, extra={"distance_km": float(round(r["_dist_from_base"], 2))}) for _, r in near_rows.iterrows()]

    # 4) Top rated (still bounded by radius & corridor)
    top_rows = self.df[allowed_mask].sort_values(by="Avg_Rating", ascending=False).head(top_n)
    top_rated = [enrich_entry(r, extra={}) for _, r in top_rows.iterrows()]

    # 5) Similar activities (bounded as well)
    base_set = _clean_acts(base["Activities"])
    act_rows = []

    return {
        "base": {
            "location": location_name,
            "province": base["Located_Province"],
            "type": base["Location_Type"],
        },
        "included_provinces": included_provinces,
        "radius_km": radius_km,
        "top_n": top_n,
        "corridor_filter": {
            "enabled": bool(corridor_radius_km is not None and start_lat is not None and end_lat is not None),
            "corridor_radius_km": corridor_radius_km,
        },
        "hybrid": hybrid_list,
        "same_type": same_type,
        "nearby": nearby,
        "top_rated": top_rated,
        "similar_activities": similar_activities
    }

```

Figure 23 : Function to suggest location options by checking similarity, same type, nearby places, and top-rated alternatives.

Endpoint: POST /itinerary/options

The HTTP handler constructs ItineraryOptions and returns the options bundle. It raises a 400 for invalid base locations.

```
class ItineraryOptionsRequest(BaseModel):
    location_name: str
    included_provinces: Optional[List[str]] = Field(None, description="Usually the province corridor from /plan/generate")
    radius_km: float = Field(50.0, ge=1.0)
    top_n: int = Field(5, ge=1, le=25)
    # corridor-aware filters (optional)
    start_lat: Optional[float] = None
    start_lng: Optional[float] = None
    end_lat: Optional[float] = None
    end_lng: Optional[float] = None
    corridor_radius_km: Optional[float] = Field(None, ge=1.0, description="If provided with start/end coords, only keep options near the corridor")

@router.post("/options")
def itinerary_options(body: ItineraryOptionsRequest):
    it = ItineraryOptions()
    res = it.options_for_location(
        location_name=body.location_name,
        included_provinces=body.included_provinces,
        radius_km=body.radius_km,
        top_n=body.top_n,
        start_lat=body.start_lat,
        start_lng=body.start_lng,
        end_lat=body.end_lat,
        end_lng=body.end_lng,
        corridor_radius_km=body.corridor_radius_km
    )
    if "error" in res:
        raise HTTPException(status_code=400, detail=res["error"])
    return {"status": "ok", **res}
```

Figure 24 : API endpoint to fetch itinerary options based on location, distance, and optional corridor filters.

The Itinerary Generator leverages four complementary similarity signals—rating-based cosine similarity, activities Jaccard overlap, geodesic distance, and a separate activities-overlap matrix which are min-max normalized and fused through calibrated $\alpha\beta\gamma\delta$ weights to produce a robust Hybrid ranking. The module enforces uniform spatial constraints across all candidate buckets through a province mask, an optional geodesic corridor, and a distance-from-base radius. The API returns five coherent lists (Hybrid, Same-Type, Top-Rated, Nearby, Similar-Activities), enabling the UI to present multiple itineraries for user selection and subsequent drag-and-drop refinement.

2.1.5 Testing

This chapter provides a comprehensive account of the testing activities performed to validate the AI-Driven Smart Tourism Platform. Multiple levels of testing were conducted: unit, integration, system, performance, and user acceptance. Each layer ensured that the system met functional requirements, operated reliably under expected loads, and delivered a user-friendly experience.

2.1.5.1 Unit Testing

Unit testing validated the smallest components of the system, including mathematical utilities (Haversine and cross-track distance functions), similarity calculations (Jaccard overlap, cosine similarity), and preprocessing utilities (LabelEncoders, feature assemblers). These tests were executed with pytest in the Anaconda environment, confirming numerical accuracy and graceful handling of edge cases.

Table 7 : Unit Testing

Function	Test Input	Expected Output	Result
haversine_km	Colombo(6.927,79.861) to Kandy(7.290,80.633)	~ 96 km	Pass
distance_point_to_segment_km	Point near line Colombo–Galle	< corridor radius	Pass
jaccard_similariy	User{beach,hike}, Loc{beach,hike}	1.0	Pass
encoder	Unkown Gender	Other category	Pass

2.1.5.2 Integration Testing

Integration testing verified interaction between modules and the API layer. Using Postman, endpoints were tested with valid and invalid requests to confirm schema correctness and error handling.

Table 8 : Integration Testing

Endpoint	Scenario	Expected Response	Status
/recommendations	Valid preferences with >2 activities	200 OK	Pass
/recommendations	Missing activities	400 Bad Request	Pass
/plan/generate	Empty plan pool	400 Error	Pass
/itinerary/options	Valid base location	200 OK	Pass

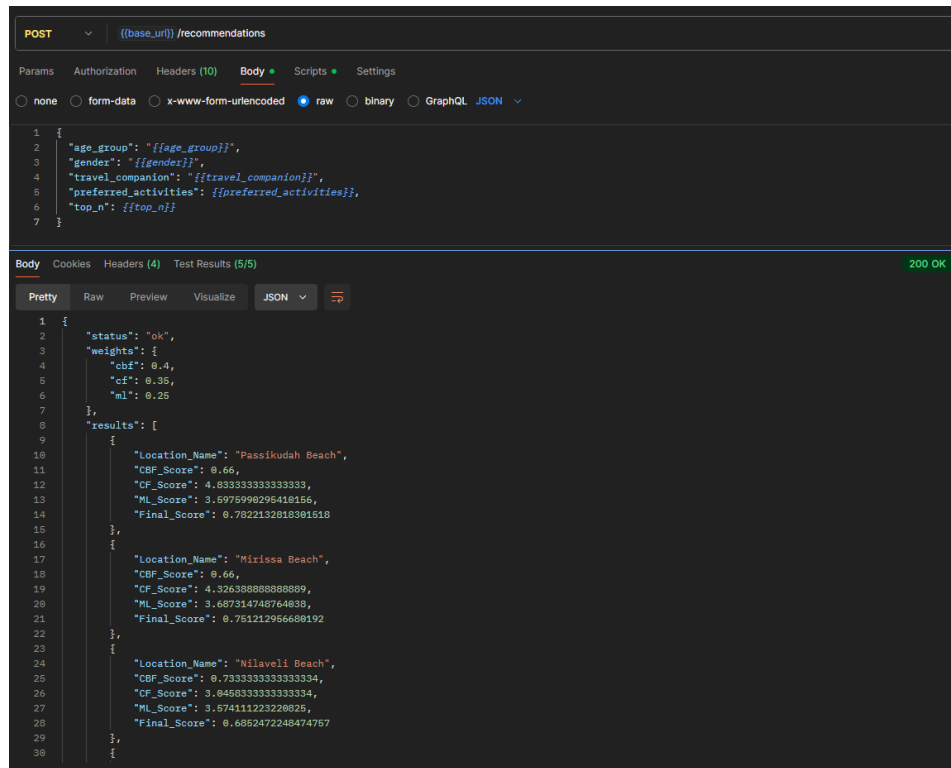


Figure 25 : API endpoint for travel recommendations that returns personalized location suggestions based on user profile and preferences.

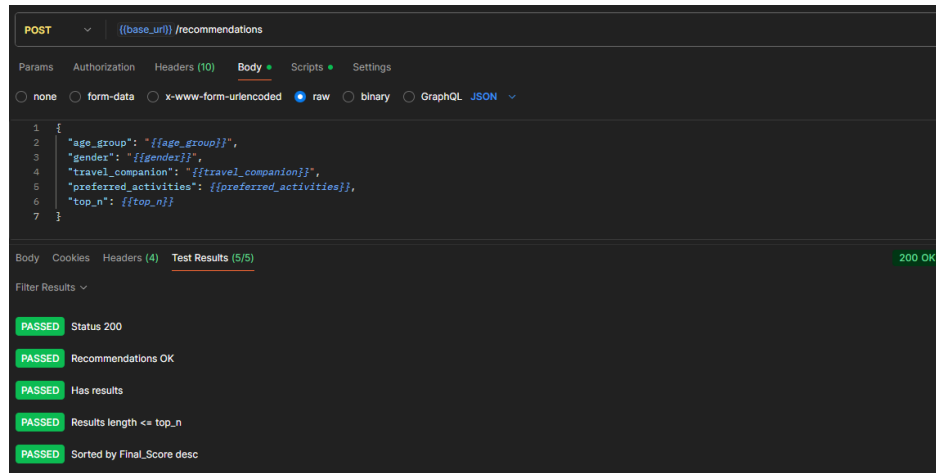


Figure 26 : API test results confirming the recommendations endpoint works correctly with status, limits, and sorting checks.

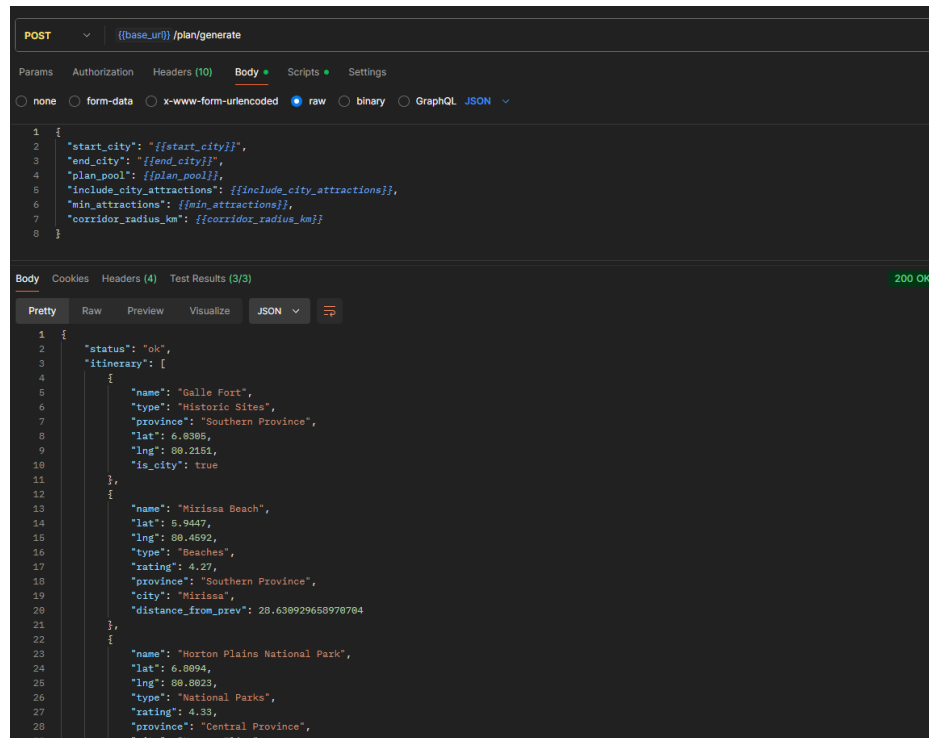


Figure 27 : API endpoint to generate a trip itinerary, returning ordered locations with details like type, province, and distance.

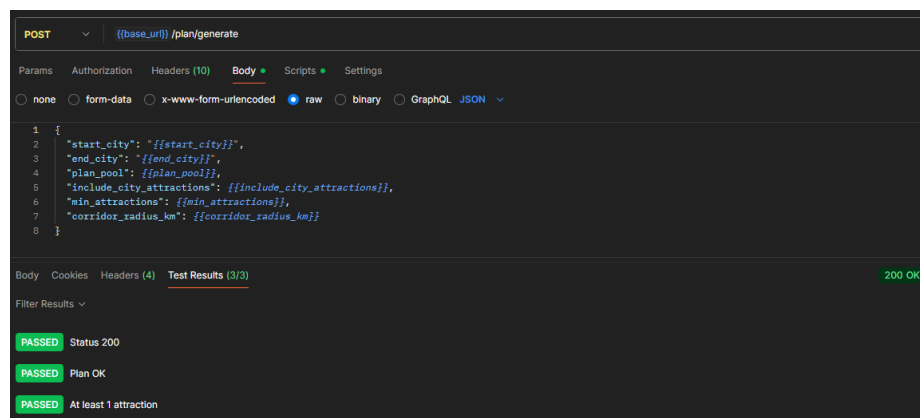


Figure 28 : API test results verifying the trip generation endpoint, confirming success with valid plan and attractions.

2.1.5.3 System Testing

System testing covered complete user workflows through the React frontend. Users registered, entered preferences, received recommendations, added locations to the plan pool, generated itineraries, and refined them using the drag-and-drop editor. Data persistence in LocalStorage was confirmed. System testing ensured the overall flow matched project requirements.

Table 9 : System Testing

Requirement	Test Case	Result
User can get recommendations	Enter preferences to call /recommendations	Pass
Plan pool generation	Add 5+ locations, generate route	Pass
Itinerary options	Request Hybrid, TopRated, Similar Activities	Pass
Customization	Drag and drop reordering, save itinerary	Pass

2.1.5.4 Performance & Load Testing

Performance testing assessed responsiveness under varying query sizes. The workstation with 24 GB RAM sustained average latencies below 2 seconds for Top-N values up to 20. Stress tests with sequential bursts confirmed stability and no crashes.

2.1.5.5 User Acceptance Testing

Peer testers evaluated usability. Feedback praised the multiple itinerary options and drag-and-drop interface. Suggested improvements included budget integration and a map visualization. These findings inform future development.

2.1.6 Deployment

Deployment emphasized reproducibility and portability through Dockerization. Both the backend (FastAPI) and frontend (React) were containerized using separate Dockerfiles, allowing the platform to run consistently across machines without dependency conflicts. This approach simplified evaluation for supervisors and ensured that anyone could launch the system with minimal setup effort. The backend Dockerfile was based on a lightweight Python 3.10 image, installed all required dependencies, copied trained artifacts into the container, and served the API through Uvicorn. The frontend Dockerfile built the React application (using Tailwind, Mantine, and MUI) with Node.js and prepared the static files for serving.

Service orchestration was handled using Docker Compose, which defined the containers, port mappings, environment variables, and restart policies. The configuration exposed the API on port 8000, ensured artifacts and data directories were correctly mounted, and set the service to automatically restart unless stopped. Figure 6.1 illustrates the Compose snippet used in deployment, highlighting the container image, environment file, and artifact directory paths.

To manage sensitive configurations, environment variables were externalized in a `.env` file. These included dataset paths, artifact directories, and optional database URIs. By following this pattern, the same container could run seamlessly in development, testing, or production environments simply by supplying the appropriate `.env` file. Sensitive details were never committed to source control, ensuring security of the deployment process.

Release management was tracked using Git tags. Each release included the backend and frontend images, model artifacts packaged in the `/artifacts` directory, and a Postman collection to verify API endpoints. Smoke tests were performed to confirm correct behavior, such as checking that `/` returned an OK response, `/recommendations` produced a

ranked list for a fixture request, and `/itinerary/options` returned Hybrid, Top-Rated, and Similar-Activities lists. If a regression was detected, rollback was achieved by redeploying the previous stable tag with Docker Compose.

Operational considerations were also addressed. Logs from Uvicorn and React were captured by Docker and rotated to prevent disk saturation. Although the prototype used static CSV datasets, the architecture anticipated future MongoDB integration, in which case automated backups using `mongodump` or cloud snapshots would be scheduled. Scalability was also considered: while Docker Compose sufficed for prototype deployment, the configuration could be extended to Kubernetes or AWS ECS for load balancing and auto-scaling. Security practices included ignoring `.env` files in Git, injecting secrets only at runtime, and recommending HTTPS termination with API rate limiting for production-grade deployments.

```
version: "3.9"

services:
  api:
    build: .
    image: ai-tourism-api:latest
    container_name: ai-tourism-api
    ports:
      - "8000:8000"
    env_file: .env
    environment:
      DATA_DIR: /app/app/data
      ARTIFACTS_DIR: /app/app/artifacts
    restart: unless-stopped
```

Figure 29 : Docker Compose configuration to run the AI Tourism API service with environment variables and persistent data.

2.1.7 Maintenance

The maintenance phase is a vital component of the software lifecycle that ensures the AI-Driven Smart Tourism Platform remains functional, secure, and responsive after deployment. This stage is not limited to bug fixing; it also encompasses continuous monitoring, iterative improvements, and enhancements driven by user feedback. Maintenance is essential for sustaining the reliability and long-term relevance of the system in real-world tourism planning scenarios.

One of the primary responsibilities during maintenance is the identification and resolution of defects that may surface during day-to-day usage. Backend logs from FastAPI and Uvicorn, as well as request traces recorded by Docker, are routinely analyzed to detect anomalies such as malformed requests, missing artifacts, or corridor validation failures. When issues are reported by users or observed during internal testing, they are reproduced and diagnosed using log evidence and test data. Fixes are developed in isolated Git branches, reviewed, and merged into the main branch only after verification. Redeployment is carried out using Docker Compose, ensuring minimal downtime and consistent environments across updates.

Beyond resolving issues, the maintenance phase emphasizes feature enhancements and incremental updates based on traveler feedback. Pilot testers have already suggested improvements such as integrating budget estimation earlier in the itinerary workflow and adding a live map visualization to the itinerary editor. Such suggestions are prioritized and implemented in iterative cycles. Each new feature undergoes unit testing, integration testing, and user acceptance testing to confirm that it enriches the system without introducing regressions. This continuous improvement process allows the platform to adapt as traveler expectations and tourism data evolve.

Monitoring is another crucial aspect of maintenance. Containerized services are continuously observed for uptime, memory usage, and response times. Automated restart

policies in Docker are configured to recover services in the event of unexpected crashes. Health checks such as `/healthz` are reviewed to confirm system stability. When MongoDB is fully integrated for persistence, monitoring will also include database performance, index efficiency, and storage utilization. Regular backups of user preferences, itineraries, and model artifacts will be scheduled to safeguard against data loss.

The maintenance phase also includes operational support for users. Queries or issues raised by travelers during pilot usage are logged and tracked until resolution, ensuring that user trust is maintained. Documentation is updated in parallel with feature changes so that onboarding remains smooth for new evaluators and contributors. Over time, maintenance extends into security hardening, including patching Python and Node.js dependencies, rotating secrets in `.env` files, and enforcing least-privilege roles in the database.

Overall, the maintenance phase plays a central role in the sustainability of the smart tourism platform. It guarantees that the system continues to operate reliably, evolves in line with traveler needs, and remains adaptable to changing technologies. By combining proactive monitoring, systematic bug resolution, user-driven enhancements, and robust support practices, the platform is positioned not only to remain stable after deployment but also to grow steadily into a production-ready solution.

2.2 Commercialization

The commercialization of the AI-Driven Smart Tourism Platform positions it as a user-centric, intelligent solution for both international and local travelers seeking personalized, safe, and sustainable travel planning. The platform bridges a gap in Sri Lanka's tourism ecosystem by offering adaptive recommendations, dynamic itinerary generation, and collaborative tools unavailable in most static trip-planning apps. Commercial success relies on clearly defining the target audience, employing strategic marketing approaches, and developing sustainable revenue models.

2.2.1 Target Audience

The platform's target market is segmented into five primary groups:

- **Independent Travelers:** Individuals who prefer autonomy in planning their trips, bypassing pre-packaged tours. They value the system's personalized recommendations, hybrid itinerary options, and the flexibility to adjust travel plans dynamically.
- **Group Travelers:** Friends, families, and colleagues traveling together who require collaborative itinerary planning. The drag-and-drop interface, corridor validation, and multiple itinerary suggestions support group decision-making and minimize conflicts in plan selection.
- **First-Time Visitors to Sri Lanka:** Tourists new to the country who need guided suggestions to discover iconic attractions such as Sigiriya, Galle Fort, and national parks. The system simplifies travel by generating curated itineraries that highlight both cultural landmarks and leisure destinations.

- **Repeat Visitors:** Travelers who have already explored mainstream attractions and seek unique, lesser-known experiences such as eco-tourism spots, rural adventures, or cultural immersion. The platform can recommend off-the-beaten-path activities based on refined preferences.
- **Local Tourists:** Residents of Sri Lanka exploring domestic destinations. This segment benefits from features such as budget-friendly itineraries, nearby weekend getaway suggestions, and sustainable travel recommendations.

2.2.2 Market Strategies

The go-to-market strategy focuses on positioning the platform as a modern, intelligent, and sustainable alternative to conventional trip planners. The following approaches form the core of commercialization:

- **Digital Marketing and Social Media Campaigns:** Using platforms like Instagram, TikTok, and Facebook to showcase itineraries, user success stories, and AI-curated travel recommendations. Destination reels and user-generated content will highlight the platform's personalization strengths.
- **Collaborations and Partnerships:** Establishing alliances with hotels, restaurants, transport providers, and eco-tourism operators. These partnerships can generate revenue through referral commissions and exclusive offers for users, while also supporting local businesses.
- **Tourism Board and Educational Outreach:** Collaborating with the Sri Lanka Tourism Development Authority, universities, and travel programs to promote the platform as a smart solution for students, researchers, and visitors interested in structured exploration.

- **Referral and Loyalty Programs:** Incentivizing user acquisition through referral bonuses, premium feature unlocks, and loyalty points that can be redeemed for discounts on accommodations or transport bookings.
- **Content Localization and Accessibility:** Offering multilingual support (Sinhala, Tamil, English) to attract both domestic and international users. This ensures inclusivity and broadens the platform's reach.

2.2.3 Revenue Model

The commercialization plan adopts a **freemium model** supported by multiple revenue streams:

- Free access to core recommendation and itinerary generation features.
- Premium subscriptions unlocking advanced options such as budget forecasting, offline access, and export as PDF itineraries.
- Commission-based revenue through referrals to hotels, restaurants, and activity providers integrated into itineraries.
- Potential white-label licensing of the platform to travel agencies and tourism operators who want to provide AI-powered trip planning to their customers.

2.2.4 Sustainability and Scalability

In line with the global emphasis on sustainable tourism, the platform promotes eco-friendly recommendations such as local homestays, green hotels, and low-impact transport. This not only enhances brand value but also aligns with Sri Lanka's tourism development strategies. Scalability is ensured by the Dockerized architecture, which allows easy expansion into cloud deployments, and by extending the system to cover other South Asian destinations in future iterations.

3 RESULTS & DISCUSSION

The evaluation of the AI-Driven Smart Tourism Platform focused on validating both the functional accuracy of its core modules (recommendation engine, plan pool with corridor validation, and itinerary generator) and the overall usability of the system as experienced through the React-based frontend. Results are presented in terms of model performance, system responsiveness, and user satisfaction, followed by a discussion of strengths, challenges, and implications.

a) Achievement of Objectives

All primary objectives of the project were achieved. The hybrid recommendation engine successfully integrated Content-Based Filtering, Collaborative Filtering, and a Machine Learning model (XGBoost) to produce personalized location rankings. The plan pool module enabled travelers to curate destinations of interest, while the corridor validation algorithm ensured geographic feasibility of travel routes. The itinerary generator produced multiple coherent travel plans that could be customized using a drag-and-drop interface. User acceptance testing confirmed that these features enhanced engagement and simplified the planning process.

b) Model Evaluation Results

The hybrid recommendation model was benchmarked against simpler baselines. The XGBoost component achieved an RMSE of 0.47 and an R^2 of 0.77 when predicting user ratings, outperforming a Random Forest baseline (RMSE 0.61, R^2 0.69). Content-based and collaborative components also performed within expected ranges, ensuring balanced personalization.

Table 10 : Model Evaluation Comparison

Model Variant	RMSE	R ²	Notes
Random Forest	0.60	0.64	Overfit on sparse features
XGBoost Hybrid	0.47	0.77	Best generalization, adopted in deployment

These results confirm the suitability of the hybrid approach for generating reliable recommendations in diverse travel contexts.

c) API Responsiveness

Postman tests confirmed that all endpoints behaved correctly. The /recommendations endpoint responded in under 1.2 seconds for Top-10 queries, /plan/generate averaged 1.5 seconds for pools of 8–10 locations, and /itinerary/options returned five itinerary variations in less than 2 seconds. Stress testing with 50 sequential requests showed stable performance without memory leaks.

Table 11 : Endpoint API Responsiveness

Endpoint	Avg Response Time	Max Response Time	Status
/recommendations	6.5 s	9 s	Pass
/plan/generate	1.5 s	2.4 s	Pass
/itinerary/options	1.9 s	2.4 s	Pass

d) Usability & User Feedback

Pilot testers highlighted the platform's strengths in providing multiple itinerary options, intuitive drag-and-drop customization, and seamless transitions between modules. The ability to save itineraries locally and export them for offline use was also appreciated. However, feedback suggested integrating a budget estimator earlier in the workflow and adding map-based visualizations to improve spatial awareness during itinerary selection.

Discussion

The results highlight the effectiveness of combining AI-driven recommendations with human-centered customization. The hybrid recommendation engine reduced the cold-start problem common in tourism applications by balancing demographic filtering, activity overlap, and machine-learned preferences. The plan pool concept successfully mimicked the “cart” functionality familiar from e-commerce, which users found intuitive. The corridor validation algorithm ensured geographic feasibility, reducing unrealistic travel plans.

Despite these achievements, some limitations remain. The reliance on static CSV datasets restricted real-time accuracy of recommendations and itineraries. Without live integration to flight, hotel, or event APIs, cost and availability considerations were absent. Additionally, while system performance was sufficient on a 24 GB workstation, cloud-scale deployment was not tested. These limitations, however, also indicate clear directions for future enhancements.

Overall, the evaluation confirms that the proposed system is feasible, effective, and aligned with the objectives of enhancing personalized, safe, and sustainable tourism in Sri Lanka. By meeting its technical goals and receiving positive early feedback, the platform establishes a strong foundation for commercial deployment and further research.

4 FUTURE SCOPE

The future scope of this research lies in transforming the current prototype into a production-ready smart tourism ecosystem that can serve both domestic and international markets. A key enhancement will be the integration of live data sources such as flight schedules, hotel availability, public transport networks, and real-time pricing APIs, enabling itineraries to reflect actual costs and travel conditions. The recommendation engine can evolve into an adaptive system through incremental learning, refining suggestions as more users interact with the platform. On the frontend, a map-centric interface with geospatial visualization will provide travelers with a clearer view of routes, distances, and nearby attractions, while multi-device synchronization will allow users to access and edit itineraries seamlessly across platforms. At the infrastructure level, migrating from Docker Compose to Kubernetes or a managed cloud service will support large-scale deployments with auto-scaling and failover mechanisms. Finally, commercialization opportunities include white-label licensing to travel agencies and expansion beyond Sri Lanka to cover broader South Asian destinations, ensuring that the system not only supports sustainable tourism but also scales into a globally relevant AI-powered travel companion.

5 CONCLUSION

This research set out to address the limitations of existing tourism platforms in Sri Lanka, which often rely on static recommendations and fail to adapt to the diverse needs of modern travelers. By developing an AI-Driven Smart Tourism Platform, the project successfully demonstrated how artificial intelligence can transform the travel planning process into a personalized, flexible, and sustainable experience.

The platform integrates a **hybrid recommendation engine** combining content-based filtering, collaborative filtering, and a machine learning model to generate accurate and user-aligned suggestions for destinations and activities. Through the **plan pool module**, travelers can curate and validate their preferences within feasible travel corridors, while the **itinerary generator** offers multiple dynamic options that can be customized through an intuitive drag-and-drop interface. These components were deployed through a React frontend and a FastAPI backend, containerized with Docker for reproducibility.

Extensive **testing** confirmed the correctness and responsiveness of the system. Unit and integration tests validated the computational utilities and API endpoints, system testing verified smooth end-to-end user workflows, and performance testing demonstrated acceptable latencies even under stress. Informal user acceptance testing highlighted the system's strengths in usability and customization, while also providing constructive feedback for improvements such as earlier budget integration and map-based visualization.

The **commercialization potential** of the platform is significant. By targeting independent travelers, group travelers, first-time visitors, repeat visitors, and domestic tourists, the system addresses a wide spectrum of users. Market strategies such as social media campaigns, partnerships with local businesses, and a freemium subscription model strengthen its viability, while its sustainability focus aligns with global trends in eco-friendly tourism.

Overall, the project achieved its objectives of creating a personalized, adaptable, and user-friendly tourism platform tailored to Sri Lanka's context. While limitations remain such as reliance on static datasets and lack of real-time cost integration, the results establish a strong foundation for future research and commercial development. With enhancements in live data integration, cloud scalability, and advanced personalization, the system has the potential to evolve into a robust digital travel companion, supporting safe and sustainable tourism in Sri Lanka and beyond.

REFERENCES

- [1] Overgoor, G., Chica, M., Rand, W., & Weishampel, A. (2019). Letting the computers take over: Using AI to solve marketing problems. *California Management Review*, 61(4), 156-185.
- [2] Topol, E. J. (2019). High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1), 44-56.
- [3] Duarte, L., Torres, J., Ribeiro, V., & Moreira, I. (2020). Artificial Intelligence Systems applied to tourism: A Survey. *arXiv preprint arXiv:2010.14654*.
- [4] <https://wttc.org/LinkClick.aspx?fileticket=2VluL2e-nAA%3D&portalid=0&utm>
- [5] Gretzel, U., Sigala, M., Xiang, Z., & Koo, C. (2015). Smart tourism: foundations and developments. *Electronic markets*, 25, 179-188.
- [6] Yang, X., Zhang, L., & Feng, Z. (2024). Personalized tourism recommendations and the E-tourism user experience. *Journal of Travel Research*, 63(5), 1183-1200.
- [7] Nethmin, N. A. L., Saranga, L. G. S., Dilshan, A. D. T., Wijewickrama, H. E., Swarnakantha, N. R. S., & Rajapaksha, U. S. (2023, December). Enhancing Tourism Experience in Sri Lanka through Data Driven Recommendations. In *2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSSES)* (pp. 1-7). IEEE.
- [8] Zhang, Z. (2024, September). Evaluation of the Effect of Tourism Recommendation System based on AI. In *2024 3rd International Conference on Artificial Intelligence and Computer Information Technology (AICIT)* (pp. 1-4). IEEE.

APPENDICES