A MACHINE LEARNING APPROACH TO TOURISM RECOMMENDATION SYSTEM

BY

CHIA AN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Dr. Phan Koo Yuen and my moderator, Dr. Ahmad Hakimi Bin Ahmad Sa'ahiry who have given me a golden opportunity to involve in the machine learning and mobile app development field study. Besides that, they have given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course

ABSTRACT

This project aims to develop a tourism attractions recommendation system by integrating machine learning recommendation algorithms. The main problem encountered when developing a powerful recommendation system is cold start problem, data sparsity and scalability problems. Cold start problem occurs when there is insufficient data about new users, new items or both. Data sparsity is a situation where there exists null value in the dataset, making it difficult to make predictions. Scalability problems arise when the system struggles to handle large volumes of data or a growing number of users and items. To overcome this problem, this project implements machine learning algorithms with collaborative filtering, content-based filtering and hybrid filtering approaches. Algorithms like Singular Value Decomposition, K-Nearest Neighbor and Co-clustering will be compared in this project. Model with the highest accuracy will be integrated into a tourism recommendation mobile application. A high portability and mobility mobile application will be developed by using React Native, and the dataset used in developing will be obtained from Google API. By developing this powerful recommendation system, travelers, tour guides and tourism agents will benefit by reducing their massive workload on planning trip itinerary.

Area of Study: Machine Learning, Recommendation Application

Keywords: Tourism Application, Recommendation, Mobile Application, Artificial Intelligence, React Native

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
KNN	K-Nearest Neighbour
ML	Machine Learning
CF	Collaborative Filtering
SVR	Support Vector Regression
PCA	Principal Component Analysis
EM	Expectation Maximization
SVD	Singular Value Decomposition
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
CB	Content-based Filtering
DF	Demographic Filtering
LSI	Latent Semantic Indexing
RS	Recommendation System
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
MF	Matrix Factorization
CDN	Content Delivery Network

Chapter 1

Introduction

The chapter is structured as follows. Section 1.2 introduces the problem statement met by other researchers while conducting research on recommendation AI models, and developers who are trying to develop a robust tourism recommendation mobile app. Section 1.2 also includes motivation or solutions to solve the specific problem. Section 1.3 clearly explains the objective of this project. Following section 1.4, section 1.5 and section 1.6, the project scope and direction, contribution of this project and lastly organization of this report.

1.1 Project Background

Between 2023 and 2024, international tourism receipts grew from approximately USD 1.5 trillion to USD 1.6 trillion, reflecting a 6.7% increase. This growth surpassed pre-pandemic levels, with 2024 receipts being about 4% higher than in 2019 when adjusted for inflation and exchange rate fluctuations. Additionally, total exports from tourism, including passenger transport, reached a record USD 1.9 trillion in 2024, approximately 3% higher than before the pandemic. For a visual representation of this upward trend, please refer to Figure 1.1

Travelling is one of the method people in this era to relax and escape from busy work. In Malaysia, citizens are more cherish to the opportunity and time of travelling especially after experienced the Malaysian Movement Control Order in year 2020 and 2021. To enjoy a wonderful and deserved journey, people are advised to plan their itinerary and choose tourist attractions wisely. However, it is difficult and time consuming while reviewing the enormous volume of information about destination online[1]. As new technologies emerge, people do not need to map out itinerary manually. Artificial intelligence(AI) is a powerful tool to make human life more convenient, people at present can schedule a journey and select attractions to be visited automatic by using AI. Unfortunately, some trending AI tools do not have enough customization recommendations for every user. It is further noted that a customized tourism recommendation system is a crucial system needed at the current time. This proposal aims to develop a Tourism Attraction Recommendation System by using machine learning.

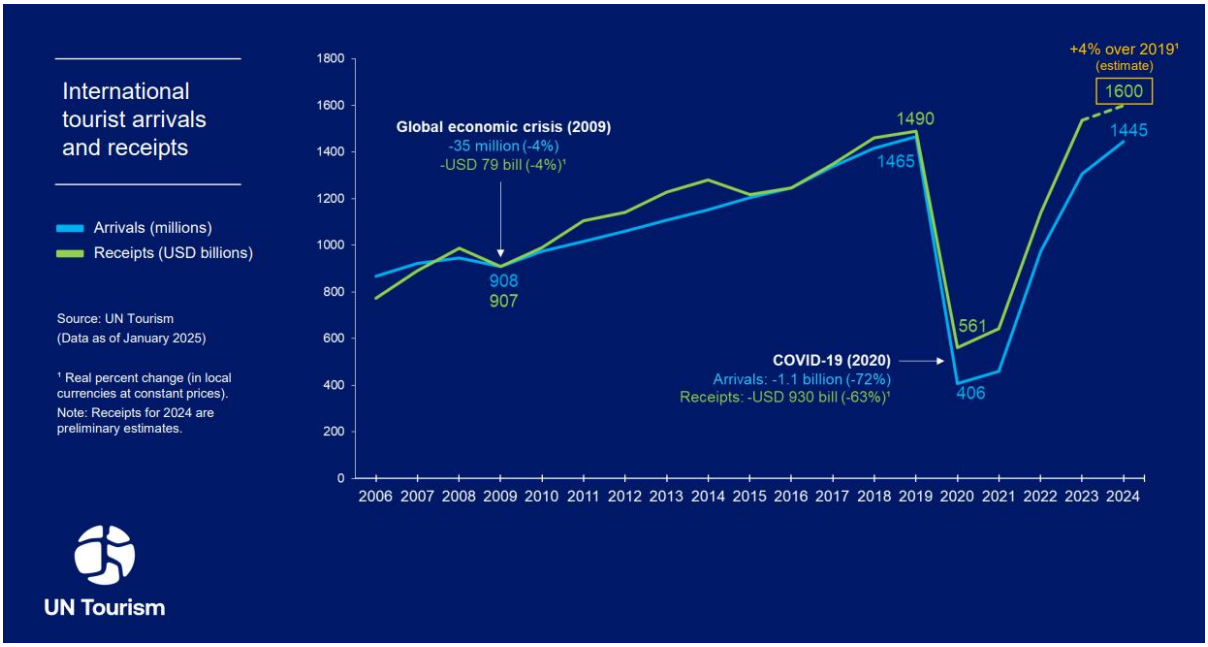


Figure 1.1 Tourism Industry Statistic [37]

1.2 Problem Statement and Motivation

1.2.1 Cold Start Problem

One of the biggest and most known challenges faced while developing a recommendation system is the cold start problem. This problem arises when there is a new user or new item signed in or added to the system. Since the system does not have sufficient information about the user and item, it is impossible for the system to propose an accurate recommendation[2]. This concept can be roughly explained as the difficulty of turning on a car engine in winter[3]. For example, when a new user logs into the recommendation system, there does not exist any data or preferences of the new user in the database of the system. This makes the system or algorithm unable to recommend or propose the most suitable item or content to the new user by following his preferences. This can also happen on new items included. A cold start situation is the main obstacle in the recommendation system while achieving excellent quality of services[4]. Moreover, cold start problems can be classified into three types: new user problem while recommending new users; new item problem while recommending new items; recommendations on new items for new users. In is paper, items are considered as hotels, restaurants, tourism attractions, etc. There are several previous works that had encountered this cold start problem.[1] stated that content-based filtering he implemented had undergo a new

user cold start problem whereby it is unable to generate results due to an absence of user-rated items for comparison.

Previous work [5] also stated that new items arise which have not been rated before had led to a new item problem. Another recommendation system development work [6] discussed that a cold start problem is one of the major limitations while applying collaborative filtering recommendation method. To develop an outstanding tourism recommendation system, the cold start problem is an issue which is crucial to be overcome. One of the solutions to overcome the cold start problem is, call a user to input his demographic while first using the application, this is the easiest way to provide user information in the beginning

1.2.2 Personalized and Reliability Challenges

In this modern era, using traveling apps like Google Travel, RoadTriper and Airbnb is quite often while people try to travel to another country, or even states. However, tourists encounter obstacles in getting accurate and personalized recommendations from conventional recommendation systems. Many traditional solutions are based on static data or general popularity measures, which do not keep pace with changing requirements and specific travelers' preferences. This problem has commonly occurred, the positive and excitement of tourists while travelling to a new place was extinguished while finding that the accommodation or restaurants they booked are not suitable for them. Luckily, given the increasing presence of users, generated data and behavior traces in this era of technology, a promising direction to improve recommendation accuracy is through machine learning, which processes the user behavior to learn patterns and to cater to the special interest of users [13]. One the other hand, travelers have little time to find personalized, accurate, and current suggestions on a wide range of activities, accommodations, restaurants, and experiences, especially when visiting outside their comfort zones. The vast quantity of material online is often disorientating and if not held within a broader framework of understanding actually makes comprehension of the subject matter worse [23]. In addition, current services often propose generic recommendations that do not meet personal tastes or current local conditions, leading to dissatisfactory user experiences [20]. These challenges emphasize the demand for intelligent situational- aware systems that can provide reliable, dynamic, and personalized recommendations to satisfy the diverse requirements of tourists today.

To address these challenges, machine learning techniques such as collaborative filtering, content-based filtering, and hybrid approaches have emerged as powerful solutions, enabling more accurate, personalized, and context-aware recommendations for travelers.

1.2.3 Data Sparsity

Machine learning as a subset and technique used to achieve AI, innumerable data are needed to train the model well. This need for large amounts of data may lead to a limitation, data sparsity. Data sparsity is a frequent problem observed in many data-driven sectors, especially those required large datasets such as recommender system and natural language processing[7]. It describes circumstances in which a dataset has a higher percentage of missing or zero values rather than non-zero or actual values. Most entries in sparse datasets are zero or null, signifying the lack of specific features or interaction. Large-scale datasets withs lots of dimensions or features, like user-item matrices in recommender systems are frequently linked to sparsity[8]. Let's state a more understandable example, in user- item interaction matrix. Each row represents a user, and each column represents an item(e.g. a restaurant) or feature of the item, the matrix is often sparse because each user with typically interact only small subset of item. It is challenging to provide recommendations for new users or items with few interactions in recommender systems due to the sparsity of user-item interactions[9].Previous work [1] had faced the problem of data sparsity and the research successfully to reduce the sparse by combining the 20 possible styles activities into five groups. Another research[10] states that the weakness of collaborative filtering system is that it has an opportunity of suffering from sparsity problem, where it could be challenging to identify similar users for particular products or users. To overcome that limitation, project[11] suggested matrix factorization. The ability of matrix factorization to handle large and sparse datasets with flexibility has set it distinct from other collaborative filtering techniques like k-nearest neighbor(KNN)[12]. Co-clustering, a collaborative filtering method, is also a good model to handle data sparsity, it discovers "tribes" of users and "categories" of places. The co-clustering reveals which "tribes" like which "categories," making it effective for recommendations even with sparse data.

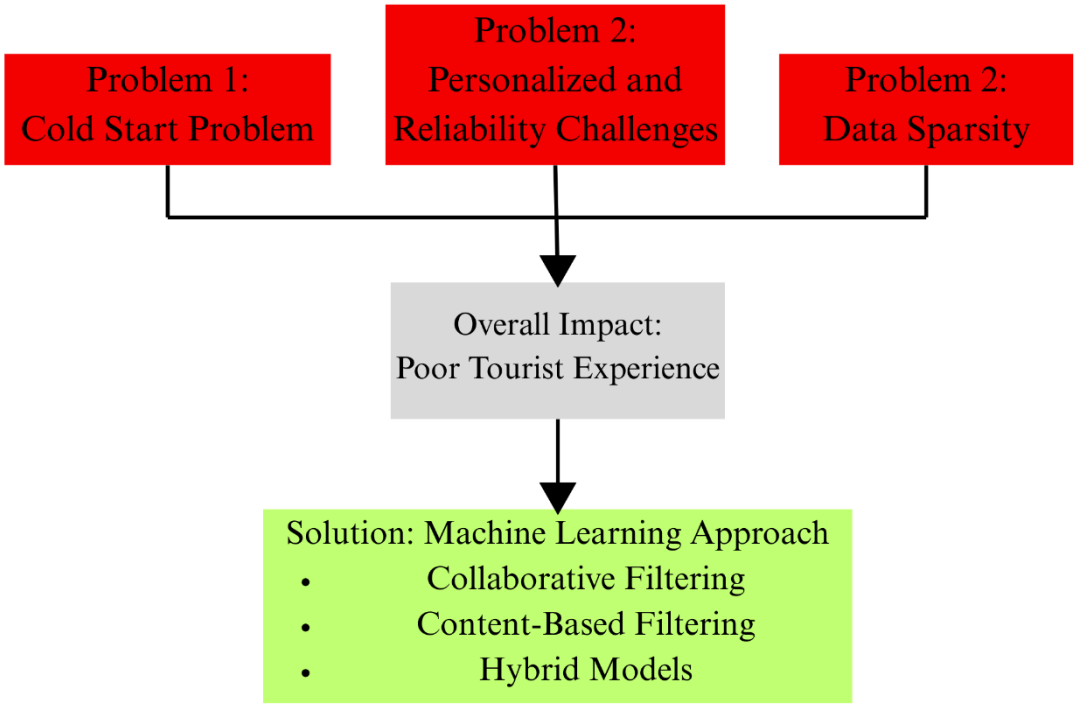


Figure 1.2 Challenges Leading to Machine Learning Solution

1.3 Project Objectives

The objectives of this project are as follows:

1. To evaluate and compare three popular recommendation AI model and choose the most accurate one to implement into a tourism recommendation mobile application.

    - By doing literature review, find three popular recommendations techniques used in this era. Conduct performance analysis by comparing the RMSE of each model and select the best one and integrate it in a mobile application.

    - By implementing the most accurate machine learning model. Cold start problems and data sparsity problems can be solved.

    - By optimizing the best model, a recommendation with higher accuracy and reliability can be published.

2. To develop a portable and high-mobility mobile application to enhance people tourist experience by consolidating various travel-related functions into a single, powerful platform.

    - Provide users with an accurate, reliable and personalized list of destinations based on their preferences and interactions. In the application, users can plan multiple tourist itineraries with different preferences. In short, users are not bound to a single set of preferences, instead, users can have different preferences, it depends on how users wish to set their own trip card in the beginning of tourist itinerary planning.

    - A powerful and highly portable mobile application will be developed by using React Native at the end of this project. Convenience functions trip destinations recommendation, users' community, destination exploration and so on will be implemented in the application.

    - Ensure the application is portable and user-friendly, facilitating easier and efficient travel planning on the go.

3. To integrate the optimized most accurate machine learning model into a highly portable mobile application with various functionalities.
    - Through integration of recommendation engine into mobile application. A highly personalized trip plan can be provided to users.

    - This application can perfectly solve the personalization and reliability problems. Users can receive a robust trip plan suggestion when the recommendation engine is integrated.

    - The best model will be optimized in the end to better handle cold start problem.

1.4 Project Scope and Direction

This project aims to develop a tourism attraction recommendation system by applying machine learning methods. A highly portable mobile application will be developed at the end of the project. First, three machine learning recommendations models; KNN with mean, Funk SVD and Co-Clustering will be computed. The computation and comparison technique used is RMSE. Model with the lowest RMSE will be considered as the most accurate model. Then the outperformed model will be optimized and implemented into the mobile application. The model is the core of the application to recommend accurate and suitable tourist attractions for users. In this project, the weaknesses and strengths of each model will be explained. The reason for a specific model being outperformed also will be discussed.

With several integrated features, a mobile application which aims to improve user experience will be developed. React Native is the framework used for development. One of the important functions is user profile management, where users can safely sign up and authenticate, make and edit profiles with interests, past experiences. They can also specify preferred destinations and kinds of attractions. The system gathers and analyzes ratings, feedback, and user interactions data to continuously improve its recommendations and deliver suggestions in real time. By using this system, users have no need to do massive research on attractions before travelling. The proposed system will recommend tourism attractions to users based on their preferences, behaviors and historical data.

In addition, this system proposed is supported by the attraction database, which stores a large repository of tourist attractions along with descriptions, reviews, ratings, image and attributes. Three databases will be used, Firebase, Cloudinary and PostgreSQL. Firebase is mainly used for user authentication. PostgreSQL is mainly for storage of structured data, while cloudinary is used for unstructured data, user uploaded image storage, for example, profile image and trip card image. The system, in turn, enhances trip planning assistance by using an itinerary generator that can suggest complete travel plans based on user preferences and recommended attractions. User interaction and feedback enable the user to rate and review attractions, giving feedback for improvement in the system's accuracy, thereby enhancing user satisfaction. Notifications and alerts will help users stay updated through personal notifications on new attractions, special events, and travel reminders, thus keeping users informed about what is new and making them better prepared for trips. Users' community is also a powerful function which will be developed, this makes users communicate with each other and share their trip plan with others.

All personal and sensitive data pertaining to the proposed system will be managed in accordance with legal and ethical requirements. Since the proposed system will be deployed and used in real-time, the proposed system should be sufficiently lightweight to minimize waiting times for users looking for tourist destinations and attractions.

1.5 Contributions

1.5.1 Body of Knowledge

This project gives value to machine learning (ML) as a discipline and more particularly to the discipline of personalized tourist recommendation systems. By the use and experimentation of various recommendation algorithms, the project establishes the most effective algorithm with the capability of achieving a Root Mean Square Error (RMSE) of 0.7, denoting high prediction performance. The most accurate algorithm performer, say Singular Value Decomposition (SVD) or K-Nearest Neighbors (K-NN) Collaborative Filtering, can serve as a benchmark for any subsequent system that is designed to make recommendations to ensure improved user satisfaction.

Furthermore, the findings of this project offer a valuable reference point for researchers and developers working on developing similar systems. It demonstrates how combining user behavior data, content-based filtering, and collaborative filtering techniques can enhance recommendation performance. The insights gained from evaluating algorithmic performance not only enable the development of more accurate and consistent systems but also encourage the application of hybrid approaches in the ML community for tourism and related fields.

1.5.2 Practical Implications

The paper presents several contributions. Firstly, the tourism recommendation system can bring benefits to **travel agency companies**. Travel agencies have the power to enhance client engagement and satisfaction by providing customized travel suggestions based on customer favor and previous behavior. Higher booking rates for travel packages and tours can be achieved by offering personalized tourism planning to customers. Operational efficiency can be improved by the automated suggestions from the recommendations system. The system can

automate the recommendation process, hence the burden on human agents, freeing them up to concentrate on more difficult support duties.

As the proposed personalized tourist recommendation system developed and implemented, a **private tour guide** is also a beneficiary. The system is a useful and powerful tool for tour guides to tailor itineraries to the interests and preferences of specific travelers and groups. Personalized tours that align with tourists' preference led to higher satisfaction and positive reviews. Moreover, a recommendation system can estimate tourist preferences and demand for specific tours, enabling tour guides to more effectively allocate resources. By optimizing schedule and routes, insights into tourist behaviors can help in planning tours more efficiently during peak and off-peak season.

The practical implications of a tourism recommendation system are quite extensive and beneficial to different actors in the tourism sector. For the travel agency companies, it helps in facilitating customer engagement and enhancing efficiency in their operations. To tour guides, this calls for customized itineraries and efficient distribution of resources. It offers personalized holiday experiences, convenience and increased satisfaction to ordinary users, thereby making travelling arrangements enjoyable and stress-free at last.

1.6 Report Organization

The report is organized into seven chapters to maintain clarity and soundness of logic. The literature review is contained in Chapter 2 and provides discussion about research already conducted and theoretical bases applicable to the study. Chapter 3 is then descriptive about system methodology and approach to developing the suggested solution. Chapter 4 then deals with system design and describes the architectural framework and structural elements, while Chapter 5 talks about system implementation and how the design was turned into a workable system. Thereafter, Chapter 6 is system evaluation and discussion and contains examination of the results, system performance, and findings. Lastly, Chapter 7 concludes the report by highlighting overall study and providing recommendations for future work and improvement.

Chapter 2

Literature Review

2.1 Previous Works

2.1.1 A Recommendation System for Tourism Industry using Cluster Ensemble and Prediction Machine Learning Techniques

Previous research [16] stated that implementation of collaborative filtering (CF) techniques in recommendation system is especially popular and had been applied in various online retail platforms like Amazon and Taobao. CF recommendation is widely used due to its capacity to meaningfully connect users and their product preferences and it can work well with sparse data. However, standard CF recommender systems match users who are similar to each other by recommending items to them based on their individual ratings. To improve the accuracy of traditional CF algorithms, the researcher proposed a method which used multi-criteria ratings instead of single rating. In detail, this research implemented model-based CF with clustering and prediction model. Several techniques like support vector regression (SVR), Principal Component Analysis (PCA), Expectation Maximization (EM) had been applied in this previous work to perform data clustering, prediction and dimensions reduction. Data clustering is an important and great influence factor to accuracy in CF, particularly in how it handles user preferences and item similarities. However, every clustering technique has a different behavior when clustering data, and no single technique can produce a satisfactory result for every kind of dataset. To overcome this limitation, the researcher applied clustering ensembles. The logic of clustering ensembles is combining several dataset partitions using a consensus function to produce a final partition. Clustering ensembles typically perform better than single clustering because they combine the advantages of multiple clustering solutions. In conclusion, overfitting of the recommendation system model to the clusters can be a drawback, especially if clusters are too small or overly specific. The researcher also stated that research on CF and multi-criteria CF should focus more on utilizing prediction and clustering ensemble algorithms for their incremental updates.

2.1.2 A Semantic Approach to Solve Scalability, Data Sparsity and Cold-Start Problems in Movies Recommendation System.

As mentioned in chapter 1, data sparsity, scalability and cold start problems are the most encountered problems while developing recommendations system. Previous work [15] had proposed single value decomposition (SVD) techniques which are matrix factorization and collaborative filtering approach technique. SVD is a mathematical technique that divides a matrix into three smaller matrices. It is widely used to locate and extract the data's underlying structure, and it is especially helpful for managing big, sparse datasets. By applying SVD, latent factors and patterns in user-item interactions can be effectively captured. This holds true even in cases where the data is scarce and limited. In the research [15], SVD had successfully decomposed the user-item interaction matrix of dataset into three separate matrices, which are user matrix, diagonal matrix and item matrix. By further selecting the top-k singular values and corresponding singular vectors from the three matrices, dimensionality of the SVD representation can be reduced. This enables us to keep the most significant latent factors and eliminate the ones that are not as significant. SVD is accurate in predicting the ratings given by a user and is capable of modelling complex patterns in the data and is also very flexible as it allows its parameters for instance the number of factors to be tuned for better performance. However, despite SVD being able to handle cold start problem, initial data still required to make accurate recommendations. New system without any historical data may find this difficult. Even though SVD offers interpretable factors, understanding each factor's meaning can be difficult. The characteristics of the user or the item may not be clearly correlated with the factors.

2.1.3 A Machine Learning Approach to Building a Tourism Recommendation System using Sentiment Analysis

As we all know, people basically not only give rating score while reviewing a place or attractions but also write down what they thought about the item. These comments largely determine how much the commenter likes the item. [17] had proposed some algorithms to perform sentiment analysis on digital text. The model is trained to determine how much a review is positive or negative using the reviews that already exist. The system's rating of the tourist destination is determined by the degree of positively or negatively. Through experiment, the researcher obtained a result of Recurrent Neural Network (RNN) with an outstanding

accuracy of 94.56%. Following by Convolutional Neural Network with accuracy of 94.40%. Instead of traditional RNN, this research applied Long Short-Term Memory RNNs (LSTM RNN). Normally, traditional RNNs are not suitable for natural language processing (NLP) tasks such as text classification. In such a scenario, researchers stated that one option is to use an alternative RNN model, like the LSTM model. Because LSTMs have input, forget and output gates that regulate the information flow through the network, they are more appropriate for NLP tasks. To perform a powerful and accurate recommendation system, [17] stated that other researchers can try on breaking down the comment more specifically and classify them into different core categories. For example, extracting features from a review about a tourist destination, such as parking availability, cleanliness, and child safety, may prove useful and warrant further research in future.

2.1.4 A Personalized Hybrid Tourism Recommendation System

Collaborative filtering (CF) and content-based filtering (CB) are the two main techniques in recommendation algorithms. However, applying only one technique is not enough to develop a powerful and accurate recommendation system. To maximize the advantages of each technique and get around the cold start problem, hybridization is a wise move. [1] proposed a hybrid recommendation system by combining CF, CB and demographic filtering (DF). The researchers had further divided the hybridization schemes into weighted and switching approaches. These two approaches are the main techniques applied to find the best combination of CF, CB and DF to increase the accuracy of predictions. When conducting the experiment, the researchers found that there does not exist a single algorithm that can solve all the problems encountered. For example, the KNN for CB, although it can handle the issue of the new item cold start problem well, it still has the issue of the new user cold start problem, which prevents it from giving results since there are no rated items from the active user to compare. Figure 2.1.4 shows the architecture of the proposed system. In the perspective of recommender engine, an algorithm of summarizing the hybrid method was implemented. The algorithm checks if there is no cold start situation, it uses the average weighted sum of DF, CB and CF results. If a new user cold start problem is detected, it uses the DF recommender result and CB recommender result will be used if new item cold start situation is detected. In conclusion, the result of this previous work shows that hybrid method gave an outstanding accuracy prediction compared to other methods used separately. However, evaluating hybrid systems can be more

complicated because performance needs to be assessed across multiple dimensions and algorithms.

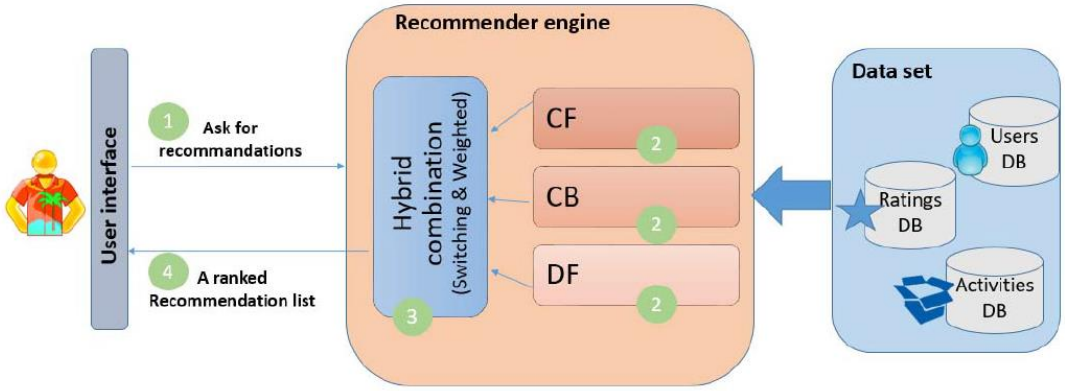


Figure 2.1.4 Architecture of Hybrid Recommendation System [1]

2.1.5 Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems

According to previous research [18], collaboration filtering techniques are becoming a crucial tool in E-commerce recommendation domain. However, collaboration filtering encountered a problem when handling huge volume of users and items in existing corporate databases. Researchers stated that singular value decomposition (SVD)-based approach can perform a result that better than traditional collaboration filtering algorithm mostly. However, SVD-based recommender systems have a significant drawback that causes them to be less suitable for widespread use. The drawback is that the matrix factorization in SVD is computationally very expensive. It is a crucial obstacle to achieving high scalability. To overcome this issue, the researcher has proposed incremental SVD-based recommendations that can achieve high scalability and maintain excellent predictive accuracy. In [18], researchers conduct an experiment of applying Latent Semantic Indexing (LSI), which uses SVD as its underlying dimensionality reduction algorithm and test its performance. The researcher concludes that despite SVD-based algorithms have an outstanding on-line performance and merely a few basic arithmetic operations are needed for every recommendation, the off-line decomposition step is computationally very expensive. However, there are still numerous additional ways that SVD could be used to solve problems with recommender system. For instance, it could be used to

identify important products that would aid in launching the recommender system or to produce low-dimensional visualizations of ratings space.

2.1.6 Movie Recommendation System Using Cosine Similarity and KNN

Previous work [19] utilized cosine similarity in conjunction with content-based filtering to evaluate how similar two items are to one another based on their attributes. Cosine similarity is a useful tool for comparing two movies because it computes the cosine of the angle between the two vectors in multidimensional space. A value close to 1 indicates high similarity, while a value near 0 suggests minimal similarity. When combined with the KNN algorithm, this method finds the closest neighbors based on similarity scores and efficiently suggests movies by using a normalized popularity score to improve distance calculations. [19] stated that to improve performance and user experience, recommendation systems can be further advanced by incorporating deep learning techniques with other filtering methods, such as collaborative and hybrid filtering.

2.1.7 The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review

To provide users with better recommendations, machine learning (ML) algorithms are being used in recommendation system (RS). However, due to the abundance of approaches and variations suggested in the literature, the machine learning field lacks a clear classification scheme for its algorithms [21]. A review [22] has been conducted to identify patterns in the application or study of machine learning algorithms for recommender systems. Through experiments, the researcher made a conclusion that in the development of RS, collaborative approaches are becoming increasingly popular, particularly when neighborhood-based techniques are employed. There is still a need for further research into hybrid approaches. Both supervised and unsupervised learning in relation to ML algorithms are being thoroughly studied. The most popular algorithms are ensemble, support vector machines (SVM) and clustering algorithms like K-Means. Research of performance metrics method to evaluate ML algorithms also been conducted in [22] with conclusion of MAE, Precision, Recall and F-measure are the popular used. Further research can be conducted in the future to examine the effects of the use, effectiveness and utility of Clustering, Ensemble and SVM algorithms in

RSs. Moreover, [22] stated that research examining requirements and design phases as well as maintenance phases is also lacking in RS development.

2.1.8 Games Recommender System Using Singular Value Decomposition

CF is a highly popular and effective recommender system paradigm. Memory-based CF is a subcategory of CF that searches for similarities between users (user-based CF) and items (item-based CF). Model-based CF, on the other hand, uses data mining or machine learning techniques to forecast ratings. SVD is one of the techniques of matrix factorization (MF) and MF is one of the methods in model-based CF. Dimensionality reduction in SVD is achieved by first decreasing the number of input variables, or dimensions, in the modelling process. When compared to memory-based collaborative filtering (CF) techniques, SVD, which is model-based, produces predictions that are more accurate [24]. [24] suggested using SVD in place of model-based CF in the game recommender system since it is a more effective solution to the data sparsity issue and Non-negative Matrix Factorization as a base model to test and compare their performance. Despite this project is a game recommendation system, the algorithms proposed still can be used in tourism recommendation as recommendation systems have the same concept in algorithm. In this work, SVD was trained on a user ratings matrix obtained from the Steam Reviews 2021 dataset to predict ratings with high accuracy. The performances of SVD were evaluated using Root Mean Square Error and cross-validation metrics and compared to those of Non-Negative Matrix Factorization. These results pointed out that SVD outperformed NMF for accuracy, since its average RMSE was decidedly lower. Moreover, SVD was faster in computation, at minimum 172 seconds versus NMF's of 231 seconds. The results showed that SVD is more efficient in making recommendations for games and that further work may be extended with an integration of deep learning methods or by adding more features like playing time, genre, and price to improve the system's performance.

2.1.9 Co-Clustering: A Survey of the Main Method, Recent Trends and Open Problems

Co-clustering is a dual clustering technique applied to rows and columns of a data matrix with the aim of finding latent patterns in huge high-dimensional and sparse data. According to the argument posed by Battaglia et al. [27], co-clustering improves clustering efficiency but also enables dimensionality reduction as well as enhanced interpretability of output, which is

especially useful in complex decision-making systems. Their survey groups co-clustering methods into four broad categories: spectral methods, which transform co-clustering into a graph partitioning problem; matrix factorization-based methods, such as Non-negative Matrix Tri-Factorization, which factor matrices into lower-dimensional representations; model-based probabilistic methods such as Latent Block Models; and information-theoretic methods that maximize mutual information or analogous measures. Recent trends also include deep learning techniques, e.g., DeepCC, and optimal transport-based methods, e.g., CCOT, that address issues of automatic identification of cluster counts and scalability. Although the quantity of newly proposed algorithms has decreased to some extent, citation patterns depict ongoing strong scientific interest in the subject. Co-clustering remains relevant to text mining, bioinformatics, recommendation systems, and image segmentation tasks, with future directions including tensor co-clustering, multi-view data analysis, and integration with manifold learning techniques to enhance processing of non-linear data structures [27].

2.1.10 Simultaneous Co-Clustering and Learning to Address the Cold Start Problem in Recommender Systems

The cold start problem has always been a critical and important issue for recommender systems, which is caused by the insufficient information of new users and items [26]. This serious issue has made new users feel disappointed while using the system. To improve the quality, user experience and accuracy of the recommender system, [26] proposed a hybrid approach which combines collaborative filtering with demographic information. The algorithm the author used is Simultaneous Co-Clustering and Learning (SCOAL). SCOAL iteratively divides data into co-clusters, with row represent users and column represents items while simultaneously building predictive model for each co-cluster. An important point in this paper is that the authors do not just cluster users with items using raw data values, instead clustering is based on its own model predictions. The predictive model is based on the characteristics of users, for example of movie recommendations system, users with similar attributes like age and interested genre will be clustered together. This predictive model mechanism is very essential since the similarity of each cluster will directly impact the accuracy of this kind of clustering model. In this paper, SCOAL was extended to address cold start problems. With massive experiments conducted on MovieLens, Jester, and Netflix datasets, authors showed that the methods (Cluster with Minimum Error, Weighted Prediction, Dynamic Classification)

outperform baselines similarity-based methods and matrix-factorization-KNN approach, especially when dealing with new users who never rated anything. Dynamic Classification (DC) has been a better method but costs more computationally.

2.2 Comparison of Reviewed Techniques

2.2.1 Comparison of Reviewed Algorithms

K-Nearest-Neighbors (KNN):

```python
import numpy as np
from sklearn.neighbors import NearestNeighbors

# Example user-item interaction matrix (ratings from 1 to 5, 0 means no rating)
user_item_matrix = np.array([
    [4, 0, 0, 5, 1],
    [5, 5, 4, 0, 0],
    [0, 0, 0, 2, 4],
    [0, 3, 0, 0, 5],
    [5, 0, 4, 0, 0]
])

# Normalize the matrix by subtracting the mean rating of each user
mean_user_rating = np.mean(user_item_matrix, axis=1).reshape(-1, 1)
normalized_matrix = user_item_matrix - mean_user_rating

# Fit the KNN model
knn = NearestNeighbors(metric='cosine', algorithm='brute')
knn.fit(normalized_matrix)

# Find the k nearest neighbors for a target user (e.g., user index 0)
target_user_index = 0
distances, indices = knn.kneighbors(normalized_matrix[target_user_index].reshape(1, -1), n_neighbors=3)

# Aggregate ratings from the nearest neighbors
neighbors_ratings = user_item_matrix[indices.flatten()]
predicted_ratings = neighbors_ratings.mean(axis=0)

# Recommend items with the highest predicted ratings that the target user hasn't rated
unrated_items = np.where(user_item_matrix[target_user_index] == 0)[0]
recommended_items = unrated_items[np.argsort(predicted_ratings[unrated_items])[::-1]]
```

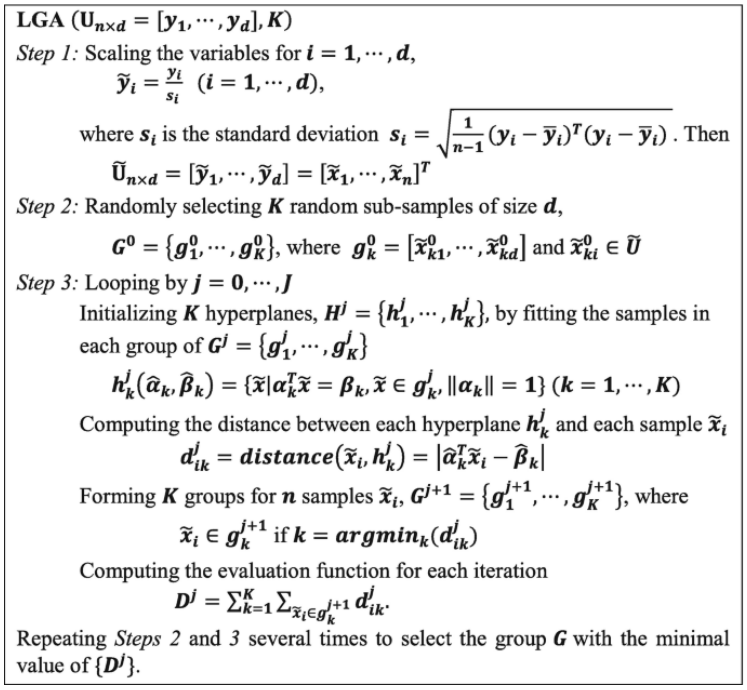
Figure 2.2.1.1 Example of code of KNN [34]

KNN is an algorithm used in recommendation systems. It is simple and effective in searching similar items or users based on their features. KNN is based on comparing the similarity of items or users. Usually, distance metrics like Manhattan distance, cosine similarity [19] and Euclidean distance are used for this [9]. The types of data and the recommendation task will determine which metric is best. For a given user or item, KNN identifies the ‘k’ nearest neighbors. The neighbors could be similar users or similar items. One hyperparameter that can be adjusted according to the dataset and performance needs in the value ‘k’.

KNN finds users who are similar to the target user in user-based collaborative filtering and suggests products that those users liked to the target user. In item-based collaborative filtering, KNN finds items similar to those the target user has already interacted with and recommends to these similar items. After the neighbors are found, recommendations are created by

combining their preferences like rating, interactions, etc. Simple averaging, weighted averaging, where neighbors with greater nearer have more influence, or more advanced methods can all be used for this aggregation [8][9]. In short, this algorithm predicts the likelihood of user liking an item based on the preferences of similar users/items and recommendation in the top items.

Co-Clustering:



LGA $(\mathbf{U}_{n\times d} = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_d], K)$

Step 1: Scaling the variables for $i = 1, \cdots, d$,

$$\tilde{\boldsymbol{y}}_i = \frac{\boldsymbol{y}_i}{s_i} \ (i = 1, \cdots, d),$$

where s_i is the standard deviation $s_i = \sqrt{\frac{1}{n-1}(\boldsymbol{y}_i - \bar{\boldsymbol{y}}_i)^T(\boldsymbol{y}_i - \bar{\boldsymbol{y}}_i)}$. Then

$$\tilde{\mathbf{U}}_{n\times d} = [\tilde{\boldsymbol{y}}_1, \cdots, \tilde{\boldsymbol{y}}_d] = [\tilde{\boldsymbol{x}}_1, \cdots, \tilde{\boldsymbol{x}}_n]^T$$

Step 2: Randomly selecting K random sub-samples of size d,

$$G^0 = \{g_1^0, \cdots, g_K^0\}, \text{ where } g_k^0 = [\tilde{x}_{k1}^0, \cdots, \tilde{x}_{kd}^0] \text{ and } \tilde{x}_{ki}^0 \in \tilde{U}$$

Step 3: Looping by $j = 0, \cdots, J$

Initializing K hyperplanes, $H^j = \{h_1^j, \cdots, h_K^j\}$, by fitting the samples in each group of $G^j = \{g_1^j, \cdots, g_K^j\}$

$$h_k^j(\hat{\alpha}_k, \hat{\beta}_k) = \{\tilde{x} | \alpha_k^T \tilde{x} = \beta_k, \tilde{x} \in g_k^j, \|\alpha_k\| = 1\} \ (k = 1, \cdots, K)$$

Computing the distance between each hyperplane h_k^j and each sample $\tilde{x}_i$

$$d_{ik}^j = distance(\tilde{x}_i, h_k^j) = |\hat{\alpha}_k^T \tilde{x}_i - \hat{\beta}_k|$$

Forming K groups for n samples $\tilde{x}_i$, $G^{j+1} = \{g_1^{j+1}, \cdots, g_K^{j+1}\}$, where

$$\tilde{x}_i \in g_k^{j+1} \text{ if } k = argmin_k(d_{ik}^j)$$

Computing the evaluation function for each iteration

$$D^j = \sum_{k=1}^K \sum_{\tilde{x}_i \in g_k^{j+1}} d_{ik}^j.$$

Repeating *Steps 2* and *3* several times to select the group G with the minimal value of $\{D^j\}$.

Figure 2.2.1.2 Example of Pseudocode of Co-Clustering [28]

Co-clustering is a collaborative filtering technique widely used in recommendation systems. By using co-clustering, the users or items will be separated into clusters, and each user cluster will then combine with its most suitable item cluster. The model will then optimize the cluster to make all the users or items can be clustered into their own similar group. The flow of co-clustering is as follows:

The algorithm starts by defining the number of clusters to be formed, this is a parameter which can be modified to obtain the highest accuracy. Depending on the dataset, the number of clusters can be (3,3), (5,5) or (7,7). Let's take an example of (3,3), this means that 3 similar user clusters and 3 similar item clusters will be formed. In the end there will be a combination of 9 co-clusters. The similarity of user inside the cluster will further calculate to make sure that

he is suitable for that cluster. In the end all users or items will allocate it to the most suitable cluster.

Singular Value Decomposition (SVD):

```python
#Libraries and Data Loading
import numpy as np
import pandas as pd
data = pd.io.parsers.read_csv('data/ratings.dat', names=['user_id', 'movie_id', 'rating', 'time'], engine='python', delimiter='::')
movie_data = pd.io.parsers.read_csv('data/movies.dat', names=['movie_id', 'title', 'genre'], engine='python', delimiter='::')

#data preparation
ratings_mat = np.ndarray(shape=(np.max(data.movie_id.values), np.max(data.user_id.values)), dtype=np.uint8)
ratings_mat[data.movie_id.values-1, data.user_id.values-1] = data.rating.values

#normalization
normalised_mat = ratings_mat - np.asarray([(np.mean(ratings_mat, 1))]).T
A = normalised_mat.T / np.sqrt(ratings_mat.shape[0] - 1)

#Singular Value Decomposition
U, S, V = np.linalg.svd(A)

#Function for Recommendation
#Top Cosine Similarity
def top_cosine_similarity(data, movie_id, top_n=10):
    index = movie_id - 1
    movie_row = data[index, :]
    magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))
    similarity = np.dot(movie_row, data.T) / (magnitude[index] * magnitude)
    sort_indexes = np.argsort(-similarity)
    return sort_indexes[:top_n]


#Print Similar Movies
def print_similar_movies(movie_data, movie_id, top_indexes):
    print('Recommendations for {0}: \n'.format(movie_data[movie_data.movie_id == movie_id].title.values[0]))
    for id in top_indexes + 1:
        print(movie_data[movie_data.movie_id == id].title.values[0])

#Recommendation Process
k = 50
movie_id = 10
top_n = 10
sliced = V.T[:, :k]
indexes = top_cosine_similarity(sliced, movie_id, top_n)

print_similar_movies(movie_data, movie_id, indexes)
```

Figure 2.2.1.3 Example of Code of SVD [36]

Singular Value Decomposition (SVD) remains one of the most popular matrix factorization techniques in recommendation system, especially because of the application of collaborative filtering methods. It helps decrease the dimensionality by pulling out latent factors that represent the underlying data pattern and makes predictions for user-item interactions [8][12]. The works of SVD are as follows:

Recommendation systems use a matrix representation of user-item interactions (like rating) where users are represented by rows, item by columns, and interactions are represented by matrix elements. Because some users have not rated every item, the matrix is frequently sparse. The purpose is to predict the sparse matrix so accurate recommendations can be suggested. SVD decomposes the original user-item matrix A into three metrices:

U: A matrix representing users and their relation to latent factors.

S: A diagonal matrix containing singular value that represents the strength of each latent factor.

V: A matrix representing items and their relation to latent factors.

Next step is dimensionality reduction, by decreasing the number of singular values in S along with the corresponding rows in U and V, using a lower-rank matrix to estimate the original matrix. This reduction eliminates noise and less important information from the data while capturing the most important patterns. Once the matrices have been decomposed and reduced, SVD can predict the missing value like rating that users have not provided. This is done by the process of multiplying the matrices together again to make estimates of the interactions based on the latent factors. The new predicted user-item interaction matrix is represented by $A^{\wedge}$. Based on the predicted matrix $A^{\wedge}$, recommendations can be made by suggesting the highest predicted ratings or interactions that a user has not yet experienced.

Algorithm	K-Nearest Neighbors (KNN)	Co-Clustering	Singular Value Decomposition (SVD)
Strengths	- Handle user or item cold start problem well[1] - Straightforward and intuitive	- Simultaneous row and column clustering - Dimensionality reduction [27] - Better interpretability	- Work well with sparse data [14][15] - Extract data's underlying structure[15] - flexible[15] - excel in modelling complex pattern in data[15]

			- efficiently capture latent factors in user-item interactions[12]
Weakness	- difficult while both cold start problems encountered[1] - low scalability[29] - worst performance on large datasets[29] - less effective with sparse data[8] - does not handle cold start problem well[30]	- Scalability issues because of high computational complexity.[26] - sensitive to sparsity[26] - requires predefined number of clusters	- difficult to understand each factor's meaning[15] - computational expensive[18]

Table 2.2.1 Algorithms Comparison

Critical Reviews:

Among the algorithms that are reviewed for developing a tourist attractions recommendation system, SVD would be most ideal because it boasts a host of strengths and, subsequently, is quite effective in managing complex recommendation tasks. This again means that SVD models complex patterns within data much better, capturing latent factors in user-item interactions more powerfully than other dimensionality reduction techniques; hence, it is quite effective for personalized recommendations. Besides this, SVD has advantages when the

data is sparse, as it usually is in recommendation systems. Previous works [12] and [25] have demonstrated that SVD performs very well in similar applications.

Co-Clustering is equally a viable option, which clusters rows and columns simultaneously to achieve better capture of the latent pattern in the data. It is equally praised for its dimension reduction and interpretability by forming significant groups. It is, however, marred by several disadvantages such as poor scalability with big data, vulnerability to sparsity and failure to pre-specify the number of clusters. Although Co-clustering is a powerful tool for clustering high-dimensional data, methods like SVD may be more adaptable to dealing with high sparse data and have greater feature extraction ability for more effective analysis.

In conclusion, given that SVD has superiority in handling such highly sparse data with rich information on complex user-item relationships, SVD is the main algorithm for the recommendation system. To make full use of the unique advantages of each algorithm, a hybrid approach, combining multiple algorithms by probably combining SVD with Co-Clustering or KNN, will balance respective strong points and compensate for the individual weak points, further enhancing system performance.

2.2.2 Comparison of Existing Website or Application

	TripAdvisor[31]	**Google Travel[32]**	**Booking.com[33]**	**Proposed System**
Search and explore destinations	√	√	√	√
Personalized trip planner	√	√		√
Reviews and ratings	√	√	√	√
Travel forums and community	√		√	√
Account and profile management	√		√	√

Push notification	√	√	√	√
Saved interesting trip plan of other users				√
Add destinations if not provided				√
Detail and accurate specific attractions recommendation				√
Detail day itinerary scheduling with time slot				√

Table 2.2.2 Website and Application Comparison

Critical Review:

The proposed tourist attraction recommendation system is unique, as it will offer more complete functionalities than the currently operating systems like TripAdvisor, Google Travel, and Booking.com. All these systems have presented essential functionalities that include searching and exploring hotels and restaurants, personalized trip planners, reviews and ratings, and push notifications. The proposed system will incorporate all these functionalities with more added capabilities.

Besides this, Google Travel does not have travel forums and community features, this function will be incorporated in the proposed system for enhancing user experience. Other than that, users can save trip plans of other users if interested. This can improve interactive and communication between users in the application. The proposed system will also go on to make some specific and detailed recommendations for attractions, not comprehensively included by any of the other platforms. Added emphasis on the precision of recommendations regarding attractions is furthered by the introduction of advanced machine learning algorithms: recommendations are pertinent and highly personalized, with accuracy rates of over 90%. The

strength of such an ability makes the proposed system particularly useful for users who are in search of specific tourist attractions and customized experiences that would cater to their interests and preferences. More detailed time scheduling features also make the application more robust. Available time slot for users to filter from 12am until 12pm will be shown in the app and let user to assign activities into that specific time slot. This makes users have a clearer understanding of the entire trip and lets them control the time wisely.

Chapter 3

System Methodology/Approach

3.1 System Architecture Diagram of Mobile Application

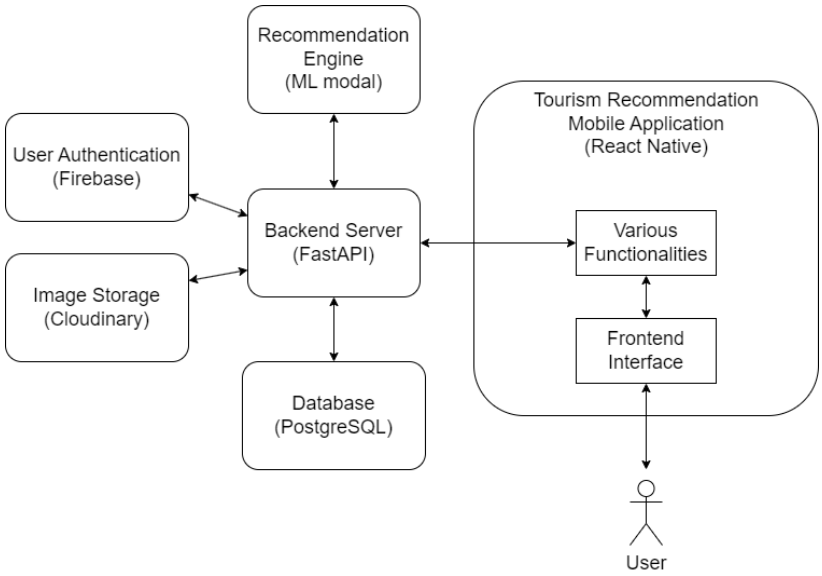


Figure 3.1 Block Diagram of Mobile Application

This project is built on a modern client-server paradigm to maintain stringent separation of concerns, facilitating modularity and ease of future modification. The client-tier is a cross-platform mobile app built on React Native and responsible for presenting the user interface as well as processing all user interactions. This platform was chosen because it can create a native-like experience from a single codebase, significantly boosting development efficiency. Consequently, frontend only deals with display, forwarding user requests and fetching processed data from the backend server.

Server-tier is built using FastAPI, a high-performance Python framework that builds the logical core of the application. Server-tier performs all business logic, performs data validation, and serves as a centralized hub, managing communication between the client and all downstream services. It handles key tasks such as user authentication flows, data querying, and serving personalized content. Its asynchronous architecture and auto-generated API documentation enable the system to be skilled at handling many simultaneous requests effectively while minimizing integration testing complexity. This design provides the backend system with resilience, security, and scalability with demand growth.

To organize data, a polyglot persistence strategy is employed, leveraging databases for specific uses: PostgreSQL for structured data, Firebase for authentication, and Cloudinary to store media. This manner of choosing the best tool for every task ensures maximum performance and reliability for every form of data operation. There is also an independent machine learning microservice with individual recommendations, thus model iteration and deployment can be performed independently without affecting the core application. This modular design with data services specific to each domain integrated with an ML module separated from them results in the system being scalable, secure, and long-term maintainable.

3.2 Use Case Diagram of the Mobile Application

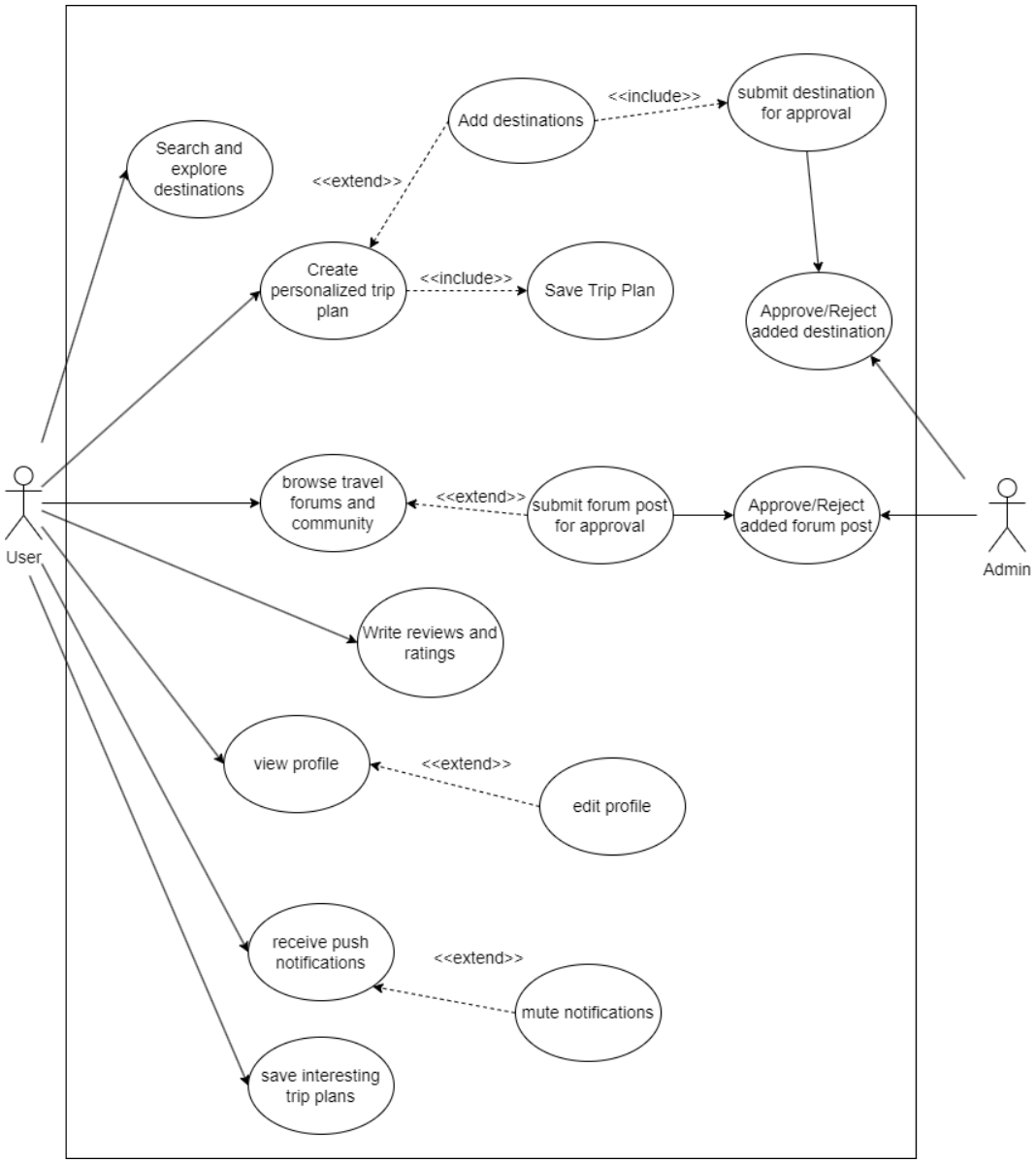


Figure 3.2 Use Case Diagram for Tourism Recommendation Mobile Application

The use case diagram represents the interaction between two actors, the user and the administrator in a tourism application intended to offer individualized travel planning and community involvement. The diagram illustrates how each actor engages in various systems activities and how “include” and “extend” associations structure the relationships between these activities.

Since most of the features are made to facilitate their travel planning and involvement in the community, the user is the main actor in the system. The ability to search and explore destinations is one of the primary features that the user can utilize to examine the various travel options that are available to them. Since the user may choose to create their own itinerary after exploring destinations, this case can optionally expand to create a customized trip plan. The system requires certain mandatory actions when the user decides to create a trip plan. This is shown through “include” relationships: creating a personalized trip plan necessarily includes the actions add destinations and save trip plan. Adding destinations may further lead to submit destination for approval, which is required when the User wants to introduce new destinations that are not already in the system. These submissions are then handled by Admin.

Another substantial action for the User is to browse forums and community travel, whereby they can view discussion and join chat with other travelers. This use case can optionally include submitting forum post for approval, where the User chooses to create new content. Like submitting destinations, posts on forums require moderation, and that is the job of the Admin. The Admin therefore plays a crucial role in moderation through the use cases approve/reject added destination and approve/reject added forum post. These tasks ensure that the system maintains quality and reliability by filtering user-generated content before it is published.

In addition to planning and community activities, the User might also interact with profile and notification settings. The view profile use case allows Users to see their own details and interests, optionally including edit profile if they want to make a change. Similarly, the receive push notifications use case keeps Users in the loop with timely alerts. This can be extended to mute notifications, where they have the option whether they want to receive such updates. Other secondary functionalities include writing reviews and ratings, wherein the User can rate services or destinations, and save interesting trip plans, where they can bookmark itineraries for later use.

The Admin is the secondary actor for this system. While the User engages with most of the functional features, the Admin's role is one of validation and approval. Whenever a User inserts

a new destination or forum entry, it is the Admin's responsibility to verify the contents and approve or reject them. Both use cases are important to maintain the integrity of the application by ensuring that objectionable, inaccurate, or low-quality content is not presented within the system.

Briefly, the diagram shows how the tourism app enables the User to plan their trip, interact with the community, update their profile, and receive personalized updates, while the Admin ensures that all content posted meets system standards. The equilibrium of User-generated features and Admin moderation renders the travel planning interactive and personalized yet moderated for accuracy and quality.

3.3 Activity Diagram

The main item in this system is the trip card. Even the community post also is the trip card. In this system, trip cards have three statuses; planning, ongoing and completed. Ongoing trip card have further “Ongoing”, “Ready” and “Past” status. Since combining these three statuses in one activity diagram may cause confusion. The following structure had separated these statuses into three different activity diagrams.

3.3.1 Trip Card Management (Planning) Module

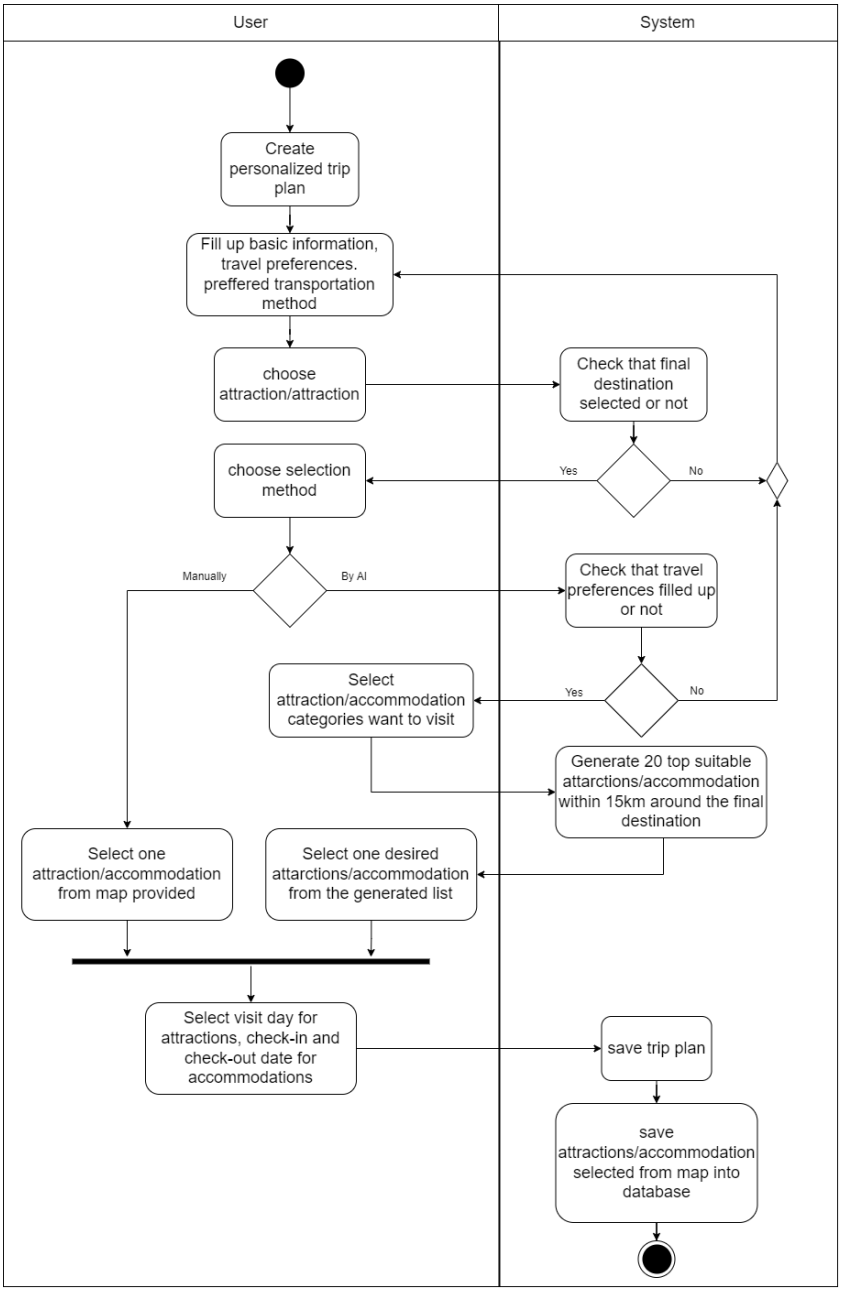


Figure 3.3.1 Activity Diagram for Trip Card Planning

The activity diagram illustrates the activity flow of a customized trip plan, where both the User and the System share duties. The process is initiated as soon as the User selects to develop a customized trip plan and provides overall travel information such as travel preferences and desired transportation method. The System first verifies if the destination has been selected by the User. If a destination is not selected, the system prompts user to select destination first, but if a destination is selected, the flow continues. The User chooses either attractions or accommodations of interest and then chooses the mode of selection, manually or AI.

If the User selects manually, he chooses one attraction or accommodation directly from the map that is shown. Or else, if the AI-method is chosen, the System checks whether the travel preferences are fully filled. If not, the User must fill them in first before continuing. Once preferences are filled, the User selects the categories of attractions or accommodations, and the System generates the top 20 best-matching options within 15 km of the destination. The User then selects one from the filled list. Next, the User sets the visit date for a destination or a check-in/check-out date for a hotel. Finally, the System saves the trip plan as customized, which completes the procedure. Additionally, if there are attractions or accommodations selected from map, it will be saved into the pending attractions table for admin to add it into system attractions table.

3.3.2 Trip Card Management (Ongoing) Module

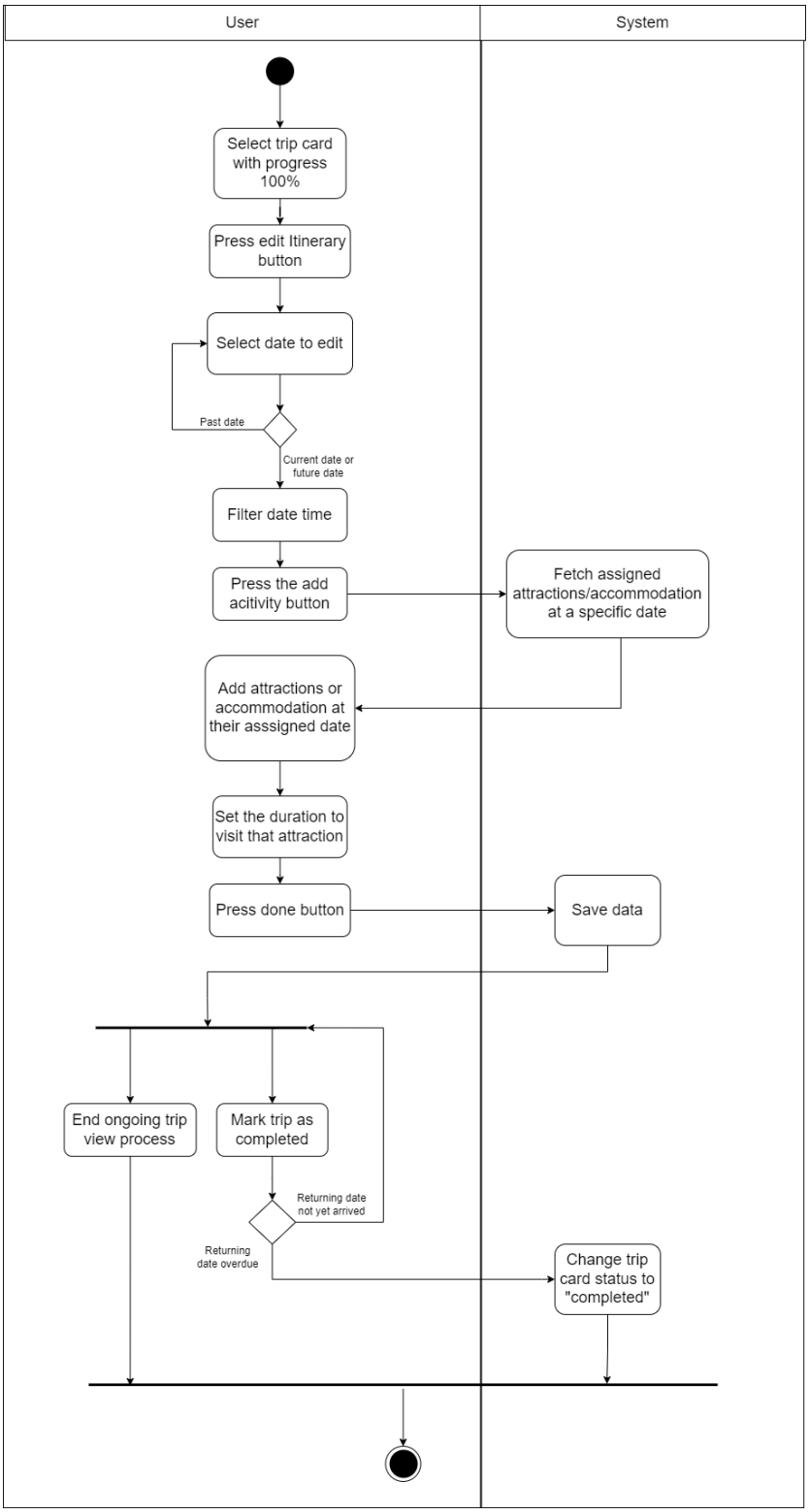


Figure 3.3.2 Activity Diagram for Ongoing Trip Card Actions

This process diagram illustrates the steps involved in editing and revising a trip plan in the system, with a focus on User-System interaction. The user begins by selecting a 100% progress trip card or in other word, trip card with status “ongoing” in database, and then decides to edit

the plan. After selecting a date to edit, the system checks whether the date is valid (either the current or future date). If it is a past date, the user must choose a valid date again. Once the date is confirmed, the user can filter date and time and proceed to add activities

The system then supports the process by bringing back assigned attractions or hotels for the selected date. The user continues by adding attractions or hotels, assigning visit durations and clicking the done button to save. The system during this step saves the altered data to ensure updates in the itinerary are created.

Finally, the diagram shows the decision-making process after stamping the status of the trip. When the return date has passed, the user can select to stamp the status of the trip card as "completed" or directly end reviewing the trip plan. When the return date has not yet arrived, the trip is maintained active and still able to edit until it reaches the completion status. The flow maintains consistency in managing itinerary so that actions of the user are validated and data integrity is maintained from the user input to the system's backend process.

3.3.3 Trip Card Management (Completed) Module

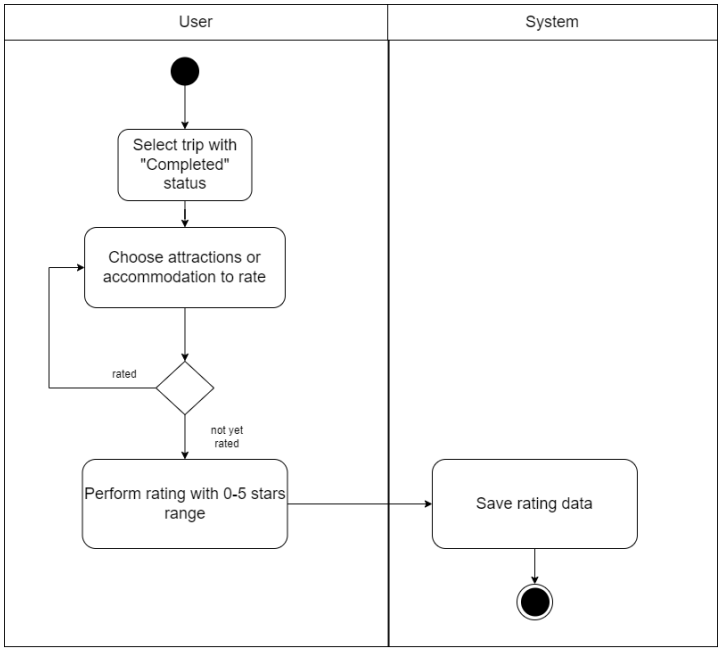


Figure 3.3.3 Activity Diagram for Attractions/Accommodations Rating

This activity diagram models the user-driven process of rating a trip component, beginning with the user selecting a trip that has a "Completed" status and then choosing a specific attraction or accommodation from that trip to rate. The user then performs the rating action,

which involves assigning a score within a 0–5-star range, before the system finally saves this new rating data to complete the process.

3.3.4 Attractions & Accommodations Adding Module.

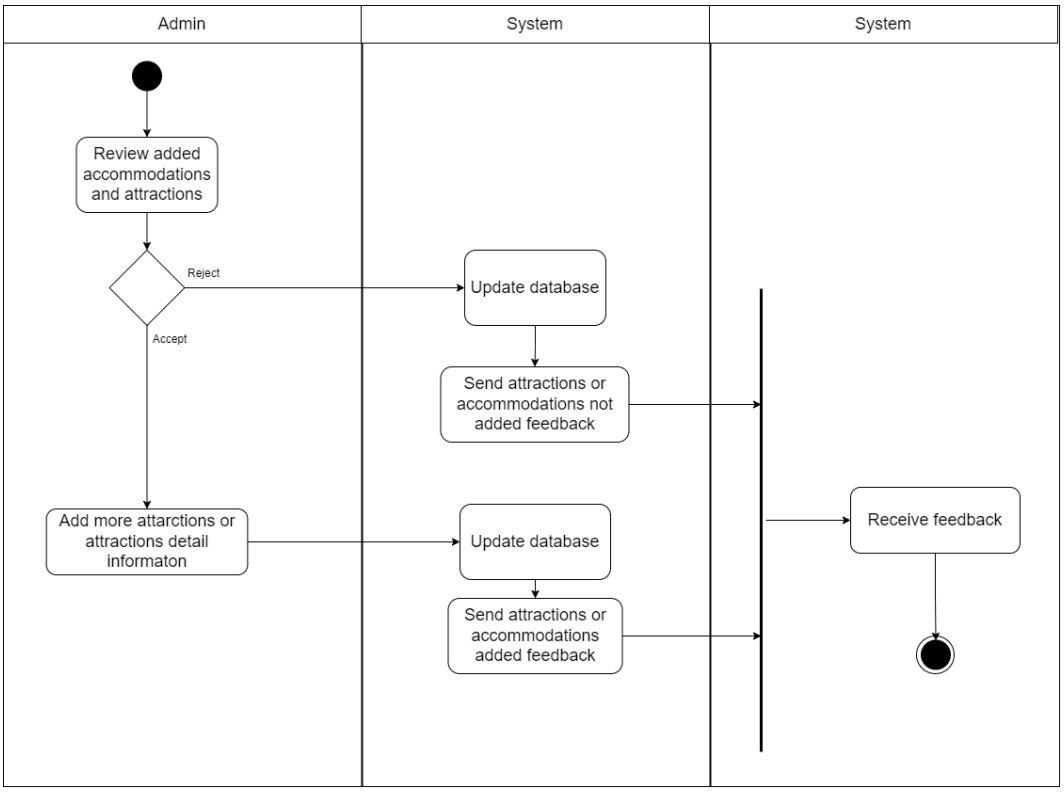


Figure 3.3.4 Activity Diagram for Attractions/Accommodations Adding Module

This activity diagram represents the process of validating and managing newly added accommodations or attractions within the system. Before this workflow begins, User previously saved the attractions or accommodations simple and basic data into database when selecting places from map manually and hence trigger this workflow done by Admin. The workflow begins with the Admin reviewing the details of accommodation and attractions submitted to the platform. At this stage, a decision point determines whether the submission is accepted or rejected. If the attractions or accommodations already request for adding by another user before, the status of the places will update to “Rejected”, and rejected feedback will be sent to User.

If the attractions or accommodation is a whole new place that had no request for adding by other User before. Admin then provide additional details or enrich the information about the attraction or accommodation. Once this step is completed, the system updates the database to include the verified data. After updating, the system generates confirmation feedback

indicating that the accommodation or attraction has been successfully added, which is then received and stored by the system for record-keeping.

Overall, the process ensures only valid and duly validated accommodations or attractions are warehoused in the database. By involving both the Admin and the system in this exchange, the workflow maintains data integrity with the added flexibility of allowing for improvement. The feedback facility gives transparency to the point where any submission—whether accepted or rejected—will have open communication back to the system. This ensures there is an ordered validation pipeline before new data becomes accessible to users.

3.3.5 Community Module

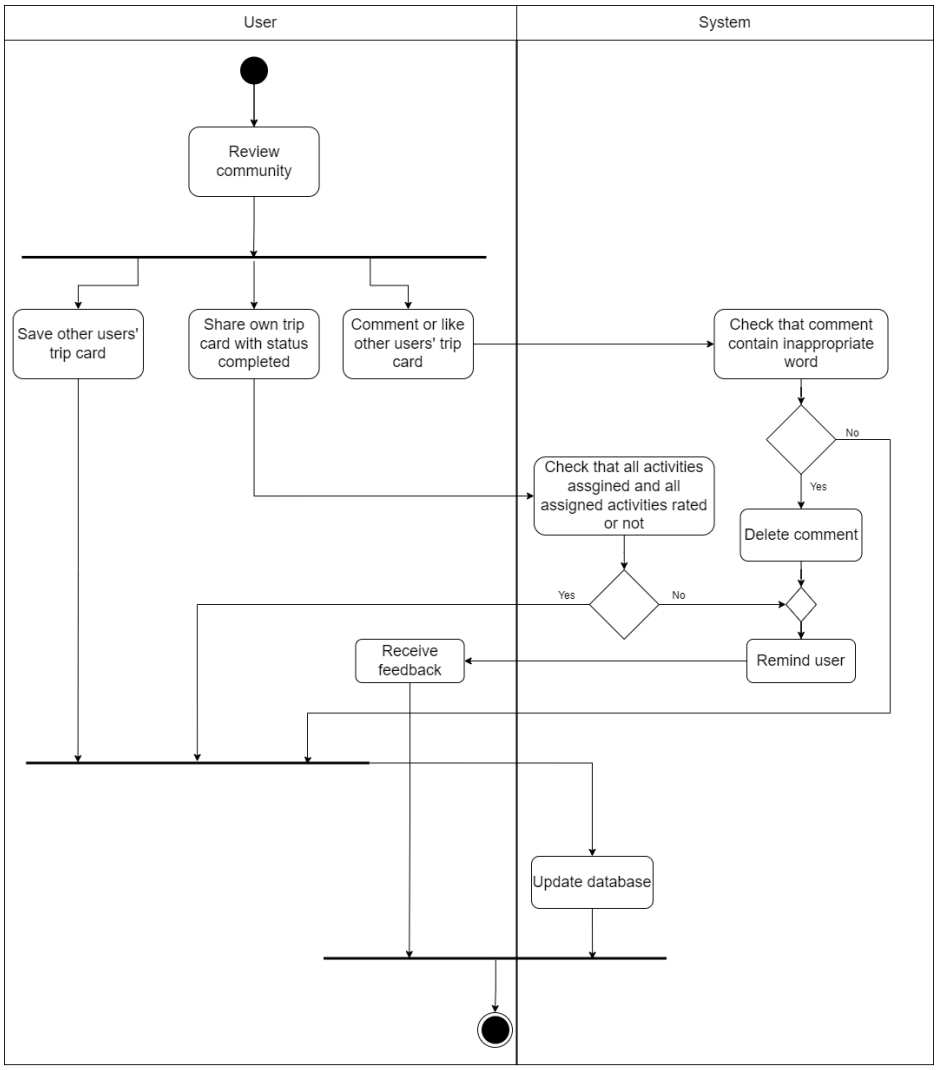


Figure 3.3.5 Activity Diagram for Community Module Actions

This activity diagram shows how the users engage with the community feature in the system. After they get to the community, they can save other users' trip cards, upload their own

completed trip cards, or social interact via liking and commenting. These features aim to encourage sharing, learning, and collaboration among travelers. Through all these modes of interaction, the system gives a platform where users can both contribute their own expertise and learn from others.

On the system's side, appropriate and reliable interactions are ensured through checks. The comments are scanned for inappropriate words by default, and the user is reminded if the activity is not complete before posting. Once all prerequisites are in place, the database is updated to represent the user's activity, and confirmation is fed back to ensure the process is being followed. This process strikes a balance between freedom for users and control from the system while ensuring a secure, supportive, and trustful travel environment.

3.3.6 User Profile(Registration) Module

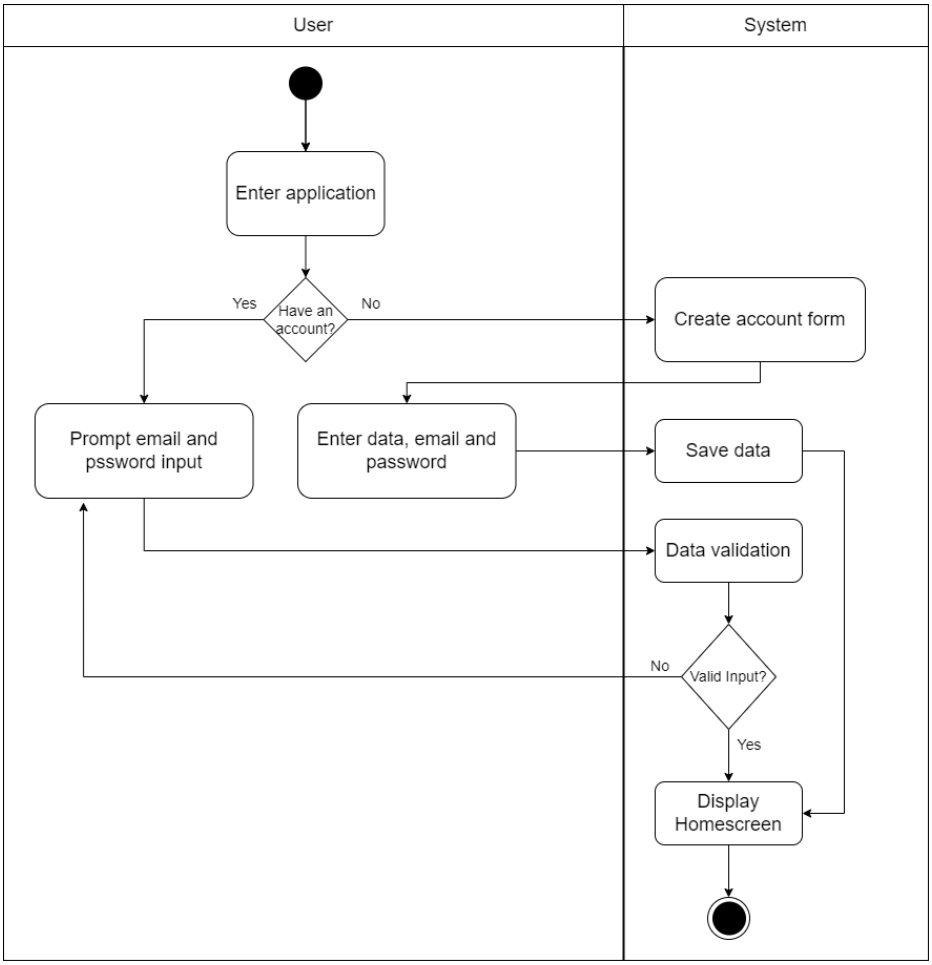


Figure 3.3.6 Activity Diagram for Registration Actions

The flowchart illustrates a sequential process of authentication initiated with the entry of a user in the application. The system first determines whether the user has a prior account; if yes, it requests his/her email and password and otherwise leads to a form to create a new account. Information entered by the user is then validated against certain requirements.

If the input is incorrect, the user is sent back to the input process so that the corrections may be done. Once correct credentials are input, the system saves the information (if new) or verifies the same (if already present), granting access finally and displaying the application home screen. This ensures only authenticated users gain access, keeping system security and data integrity intact.

Chapter 4

System Design

4.1 System Block Diagram

4.1.1 System Overflow of Recommendation Engine

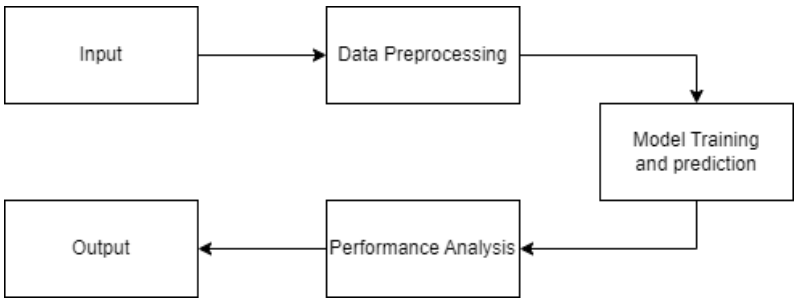


Figure 4.1.1 Block Diagram of the Recommendation System

Input :

The user provides data to the recommendation engine, including destination coordinate, budget, dietary restrictions, traveling group, pet owner status and lastly types of attractions or accommodation want to travel for result filtering.

Data Preprocessing:

In this project, a standardized data preprocessing pipeline is implemented to ensure the dataset is clean, consistent, and suitable for building accurate recommendation models. Data preprocessing source code is provided in Appendix A.1. The key steps and techniques used in the preprocess_data function are as follows:

1. **Handling Missing Values**

- Technique used**: Mean Imputation**
  For numeric columns like 'Rating Given', 'Price Level', 'Rating', and 'Total Ratings', any missing values are filled using the mean of each column. This ensures that the dataset

maintains numerical consistency and avoids the removal of potentially useful data due to null values.

2. **Encoding Categorical Variables**

   Depending on the selected encoding method, two main techniques are used:

   - **Label Encoding :**

     Technique : Each unique category in a feature is converted to an integer label using LabelEncoder().

     Application : Applied to features such as 'Traveling Group', 'Budget Preference', 'Dietary Restriction', and 'Types'.

   - **One-Hot Encoding:**

     Technique : Converts categorical variables into binary indicator using OneHotEncoder()

     Application : The same categorical features as above are transformed into separate binary columns for each unique category.

3. **Binary Feature Normalization**

   - Technique used: **Boolean Mapping and Imputation**

     Binary features such as Serves Vegetarian Food, Allows Pet, Open_Mon to Open_Sun, and others are cleaned by filling missing values with 0 (interpreted as "No" or "Not Applicable") and converting all values to integers.

4. **User-Item Matrix Construction**

   - Technique used**: Pivot Table Creation**

     A user-item matrix is created using pivot_table(), where rows represent Place Name, columns represent Author Name (users), and values represent the Rating Given. This matrix format is essential for collaborative filtering models, as it maps user preferences across available destinations.

Model Training and Prediction:

Following data preprocessing, the next step involves model training and prediction. The process of model training and prediction is using the collected and processed data to develop a recommendation model that will provide destinations based on user input. The components include:

- **Data Preparation for Model Training:** The data set includes user variables, like age and likes, destination variables, like destination attributes and types, and user ratings. It is a structured data set that will allow the model to be trained to learn human behavioral trends.
- **Model Selection:** The best-performing model, selected based on earlier evaluation, is used for training and prediction in the system.
- **Model Training:** The data to be modeled will be used in training the selected model. The selected model will be trained to learn how user profiles, characteristics of different destinations, and rating patterns of the destinations interact.
- **The Prediction phase:** When the user inputs new information, such as a destination constraint, the trained model will return a list of destinations from the input and the user's preferences. The model will rank the potential destinations based on which destinations it finds a better fit. Following the prediction process, the ranked destinations will be proposed to the user.

Performance Analysis:

The performance of the chosen recommendation model is quantified using Root Mean Square Error (RMSE) where RMSE measures how far, on average, the predicted ratings are from the real user ratings. The smaller the value of RMSE, the greater the chance of predicting ratings. The performance of the chosen recommendation model is quantified using Root Mean Square Error (RMSE) where RMSE measures how far, on average, the predicted ratings are from the real user ratings. The smaller the value of RMSE, the greater the chance of predicting ratings.

While the proposed system is currently running using the best-performing model in terms of the performance metrics, comparisons and assessments between the three candidate models will constantly be done throughout the development of the system. The outputs will continuously test and check to ensure that the system always runs using the best-performing model.

Output :

The recommendation algorithm then produces an individualized set of locations based on user preference input and prediction by the model. Recommended places will be scored based on forecasted fit for user budgets, chosen attraction categories, group number, dietary restrictions and other environmental variables, etc. The user will be provided with a list of recommended destinations that suit them the best and speed up the process of planning their trip. The system desires the output to be as precise, useful and user centric as can be to optimize and enhance their travel experience.

4.1.2 Mobile Application Hierarchy Diagram

Figure 4.1.2.1 illustrates the complete mobile application hierarchy, showing the navigation hierarchy, authentication flow, and core feature integration throughout the tourism application.

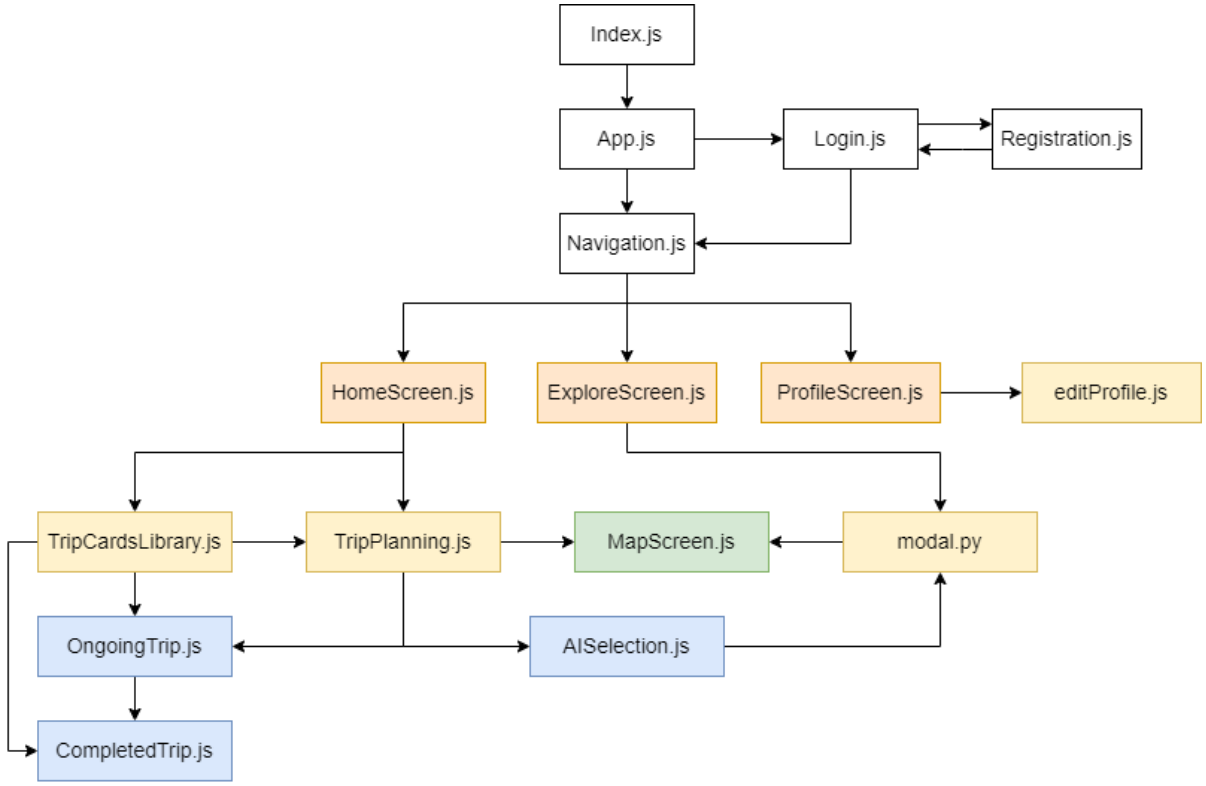


Figure 4.1.2.1 File Hierarchy Level

The system begins with index.js, which initializes the application and loads core configurations. Control is then passed to App.js, which manages the global context and user authentication state before rendering the navigation system (Navigation.js). From the entry point, users are directed to the authentication flow, where Login.js and Registration.js handle account access and new user creation. Once authenticated, users are routed into the main navigation hub. Navigation.js acts as the central navigation controller, linking the primary screens:

- HomeScreen.js for trip cards and quick access to planning
- ExploreScreen.js for discovering new attractions
- ProfileScreen.js for user profile management

Each of these screens leads to secondary modules, creating a structured and organized flow.

The Home screen ecosystem connects to modules such as TripCardsLibrary.js and TripPlanning.js. Within trip planning, users can access:

- AISelection.js for recommendation results
- MapScreen.js for location visualization
- OngoingTrip.js and CompletedTrip.js for managing trip status

This structure ensures that a trip lifecycle is well supported, from initial planning to completion and storage in the trip library.

From the Profile screen, users can edit their details through editProfile.js. Media uploads are supported by Cloudinary integration, while authentication and persistence are managed with Firebase and AsyncStorage. Besides, the frontend communicates with the FastAPI backend (main.py), which integrates the SVD recommender (model.py) and the PostgreSQL database. Third-party services like Firebase Authentication, Google Maps API, and Cloudinary are connected seamlessly to support login, mapping, and media handling.

Data Flow Summary:

The overall user journey follows a clear cycle:

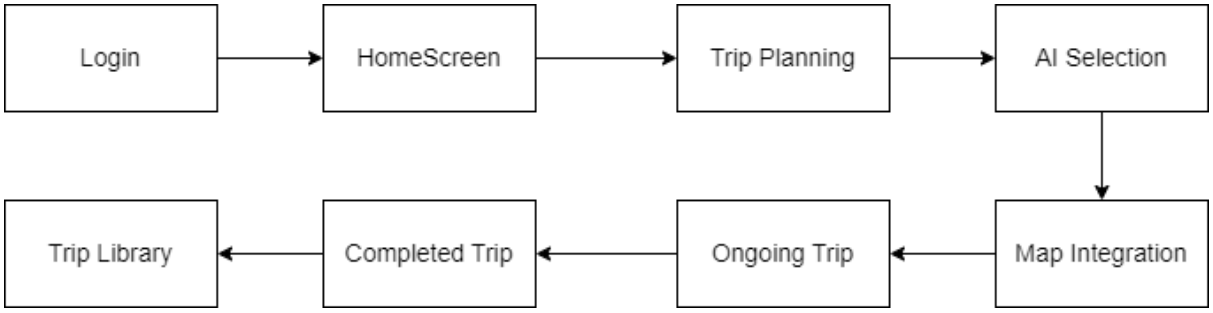


Figure 4.1.2.2 Block Diagram of the Data Flow

At each step, global states are managed through UserContext.js, ensuring consistency in authentication, trip progress, and personalization.

4.2 System Components Specifications

4.2.1 Mobile Application Stack

The mobile application stack for the system is primarily developed using **React Native** (for cross-platform development) and **Firebase Authentication** (for once secure user identifying mechanisms). React Native offers one codebase for both Android and iOS, allowing for consistent UIs and seamless navigation, and Firebase Authentication adds reliable user registration, login, and session management capabilities.

React Native:

React Native 0.79.5, a cross-platform framework that facilitates simultaneous development of Android and iOS devices from the same JavaScript codebase is used while developing the mobile app frontend. This architectural decision can significantly save development time and offer an identical user experience across all mobile platforms. The framework is augmented with Expo SDK53, which includes additional tools for rapid prototyping, testing and deployment without having to set up complex native development.

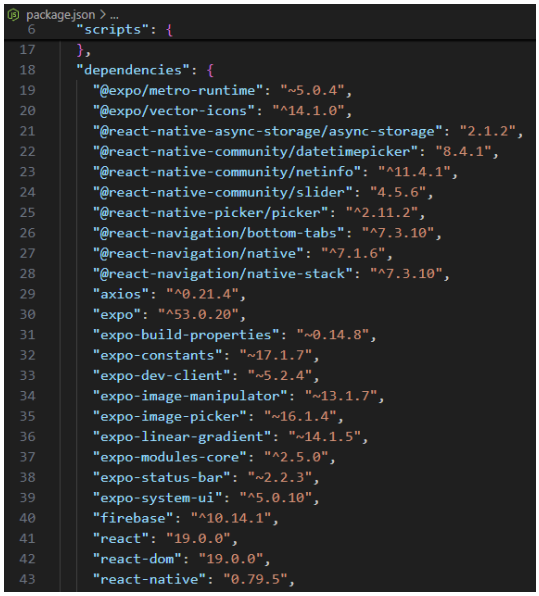


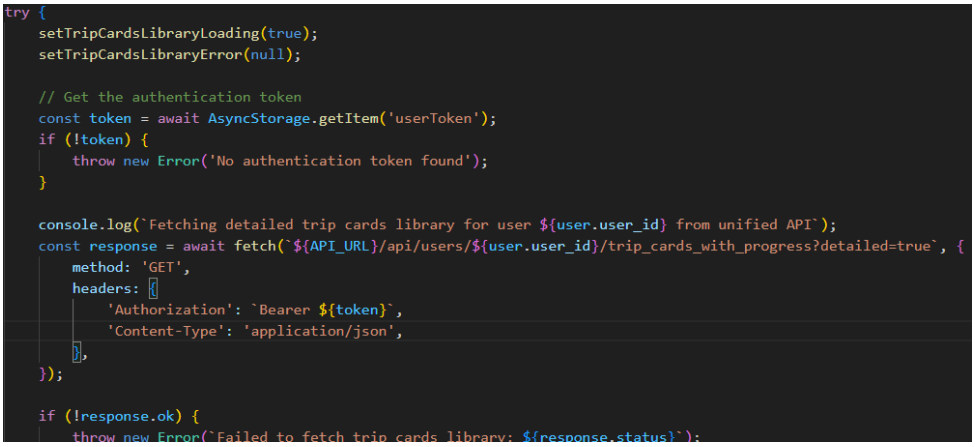
Figure 4.2.1.1 Frontend Technology Stack and Dependencies

The application's UI structure depends on a component-based architecture where every screen has a specific functional purpose. Navigation is done through a specific NavigationBar.js component using an animated bottom tab implementation that provides users with easy access to any main sections for example Home, Profile, Trip Planning, Maps, and

Edit Profile. This design enables users to easily switch between different features without losing their session state in doing so throughout the application.

State management of the React Native app is implemented with React Context API directly in UserContext.js for overall user state management. This places user authentication information, preferences, and session data in central storage and exposes it to all components without prop drilling. The authentication system runs in harmony with Firebase Authentication services, providing secure login functionality with cache storage through AsyncStorage. User sessions are defaulted to automatically expire after 21 days, with the security levels retained via token refresh measures built in.

For backend communication, the app uses the Axios HTTP client with custom interceptors to manage API requests, responses, and errors. This setup provides reliable interaction with the FastAPI backend, includes built-in retry logic, and shows user-friendly error messages. The API setup dynamically determines the correct server endpoint from the development environment, including local development and tunnel connections for testing on physical hardware.



```
try {
    setTripCardsLibraryLoading(true);
    setTripCardsLibraryError(null);

    // Get the authentication token
    const token = await AsyncStorage.getItem('userToken');
    if (!token) {
        throw new Error('No authentication token found');
    }

    console.log(`Fetching detailed trip cards library for user ${user.user_id} from unified API`);
    const response = await fetch(`${API_URL}/api/users/${user.user_id}/trip_cards_with_progress?detailed=true`, {
        method: 'GET',
        headers: {
            'Authorization': `Bearer ${token}`,
            'Content-Type': 'application/json',
        },
    });

    if (!response.ok) {
        throw new Error(`Failed to fetch trip cards library: ${response.status}`);
```

Figure 4.2.1.2 Sample Code for Axios HTTP

The app also uses React Native Maps for location features, so users can see marked destinations and plot their routes. Its modular design lets each feature run independently while sharing services like authentication and API calls, resulting in a maintainable, scalable codebase that can add new features later without major restructuring.

Firebase Authentication:

Firebase Authentication is the primary identity management system of the travel app, providing secure user registration, login, and session management. Firebase Authentication was selected because it is easy to integrate with React Native apps and provides authentication through multiple modalities like email/password, sign-in using social media, and phone authentication. Firebase Authentication avoids coding authentication functionality from scratch, making development easier while maintaining enterprise-level security.

The design of the authentication system is token-based, where Firebase generates JSON Web Tokens (JWT) upon successful user login. The tokens are managed automatically by the Firebase SDK and securely stored with AsyncStorage for long-term app session authentication. The system has a 21-day token expiration, after which users must re-authenticate to remain online. This approach balances security requirements with the convenience of the user, excluding unauthorized access while preventing the user from repeatedly being prompted to log in.

The React Native frontend integration is facilitated through the UserContext.js component that covers the whole app and provides access to global authentication state. When users attempt to login, the Firebase SDK handles the authentication request and provides back user credentials and access tokens. The credentials are stored in the app context state, thereby providing access to user data in all components without prop drilling or complex state management patterns.

The authentication process also interfaces with the custom FastAPI backend through token validation procedures. After Firebase authentication, the mobile app then sends the Firebase token to the backend API for verification and session creation for the user. This two-layered authentication ensures both frontend and backend systems to possess synchronized user states. The backend also authenticates Firebase tokens through Firebase Admin SDK, which adds additional security layers as well as server-side user management support.

```
// Get Firebase ID token
const idToken = await getIdToken(firebaseUser);
await AsyncStorage.setItem('userToken', idToken);

// Try to get user details from backend
try {
    const endpoint = `${API_URL}/api/users/profile`;
    console.log('Fetching user profile at:', endpoint);

    const response = await axios.get(endpoint, {
        timeout: 10000, // 10 second timeout
        headers: {
            'Authorization': `Bearer ${idToken}`
        }
    });
```

Figure 4.2.1.3 Token Storage and Token Persistence

Firebase Authentication configuration includes AsyncStorage persistence options that resume the user's sessions automatically when the app is restarting. This method keeps users as signed in between app restarts unless they sign out themselves or their tokens are expiring. The system also handles authentication state changes via Firebase listeners, and it dynamically updates the UI and navigation path of the app automatically according to the current authentication status of the user, giving a seamless user experience across the life cycle of the app.

4.2.2 Backend API Stack

The backend of the tourism recommendation system is designed using **FastAPI** and **PostgreSQL**, providing excellent performance, scalability, and reliability. FastAPI enables the development of high-performance RESTful APIs with built-in data validation and asynchronous processing, while PostgreSQL serves as the core relational database, ensuring secure and efficient data storage.

FastAPI:

FastAPI architecture utilizes a layered design pattern where incoming HTTP requests are passed through middleware layers to the relevant endpoint handlers. Each API endpoint is built to handle specific functionalities such as user authentication, trip planning, recommendation generation, and data retrieval operations. The framework automatically validates incoming request data against predefined Pydantic schemas, ensuring data integrity and reducing manual validation overhead. Response serialization is automatically managed, converting Python

objects to JSON format while maintaining consistent API response structures throughout all endpoints.

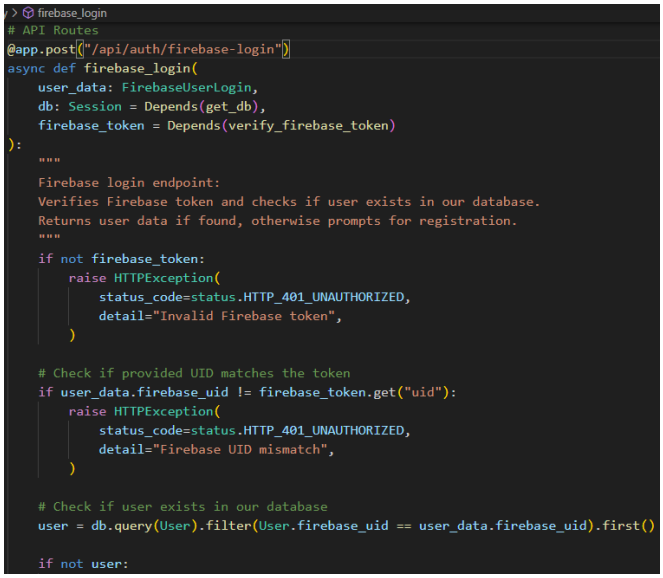


Figure 4.2.2.1 FastAPI Route Handler Implementation

Database integration in the FastAPI backend utilizes SQLAlchemy ORM for PostgreSQL database operations, providing an abstraction layer that simplifies working with complex database queries and relationships. Connection pooling and session management are utilized by the system to enhance database performance during concurrent user loads. Database models are defined using SQLAlchemy declarative base classes, which automatically generate corresponding database tables and handle schema migrations. This approach guarantees database consistency with the potential for making future schema modifications without impacting existing functionality.

Figure 4.2.2.2 SQLAlchemy Database Models

The authentication system integrates Firebase Admin SDK for token validation, creating a secure connection between the Firebase authentication of the mobile application and the backend's secured endpoints. When the mobile application sends authenticated requests, FastAPI middleware intercepts the Firebase tokens, validates them with Firebase's authentication servers, and extracts user information for authorization. The implementation ensures that sensitive operations require authentication correctly while maintaining stateless API design principles that allow horizontal scaling and load distribution.

PostgreSQL:

PostgreSQL is the central relational database management system for the tourism application to offer effective data storage, retrieval, and management for user details, trip information, recommendations, and analytics for the system. It is an open-source database that has been chosen because it has some advanced features such as ACID compliance, complex query optimization support, extensible data types support, and good support for JSON operations to support both structured as well as semi-structured data requirements. PostgreSQL is also proven to be highly reliable with good scalability to effectively support increasing concurrent user sessions as well as data volumes with expanding applications.

The database design enforces a normalized relational schema optimized to reduce data redundancy but ensures referential integrity throughout all system entities. Central tables consist of user profiles, trip cards and attractions with foreign key links defining definite data dependencies. In schema design, a third normal form is upheld where every chunk of data is stored in a single place only but with support for effective join operations to handle complex queries with efficiency. Indexing strategies are also deployed on columns often retrieved such as user IDs, trip IDs, and timestamp columns to maximize query efficiency.

Specialized data types and database functions support the tourism application's distinctive features. Flexible user preference information and trip itineraries are stored as JSON columns, whereas location coordinates for destination plotting and calculating distances utilize geometric data types. Recommendation algorithms are also custom implemented as database-level database functions to minimize data transfer overheads and minimize recommendation query response times. Full text searchability is also implemented using PostgreSQL's native support for destination and attraction searching functionality. Data security is ensured with

complete constraint definition and user roles. Referential integrity among related tables is ensured with foreign key constraints, and data ranges and formatting are checked prior to insertion with check constraints. User roles for databases are set up with least possible privilege with a principle of least access, and sensitive data like user passwords are dealt with via secure authentication routines. Automated regular database backups and point-in-time recovery setups offer data defense against database failures and also ensure business continuity for the tourism app.

4.2.3 External Services Integration

The system integrates external services Cloudinary and the Google Maps API to enhance functionality beyond core backend and database capabilities. Cloudinary provides reliable image storage and management, enabling secure profile and trip-related image handling, while the Google Maps API delivers location-based services, including map visualization, place search, and attraction categorization.

Cloudinary:

Cloudinary for this project adopts a hybrid client-server architecture that allows React Native clients to conduct direct uploads while the FastAPI server assumes secure delete operations through secured endpoints. This architecture decouples upload tasks from security enforcements, yet it enables seamless client-side image management in addition to server-side control over delete operations.

The frontend implementation in this project utilizes a standardized upload pattern across four main components (Register.js, editProfile.js, TripPlanning.js, and OngoingTrip.js), each containing an uploadImageToCloudinary function for direct uploads. This project implements intelligent URL validation to prevent unnecessary re-uploads by checking if URIs already contain Cloudinary domains. The error handling includes network connectivity checks,

response validation, and graceful fallback mechanisms ensuring user experience continuity even when uploads fail.

Backend implementation allows for a secure delete endpoint that accepts Firebase authentication tokens to allow only users with permission to delete images. URL parsing routines that take the Cloudinary PUBLIC IDS by pattern matching are included in the project. Image handling is done with the "upload-then-delete" paradigm where images are first uploaded, profiles are then updated with the latest URLs, and old images only deleted after successful updates, with the system maintaining operation even upon failure of deletion. The configuration architecture is based on Cloudinary's upload preset system for unsigned uploads that is secured through built-in protections.

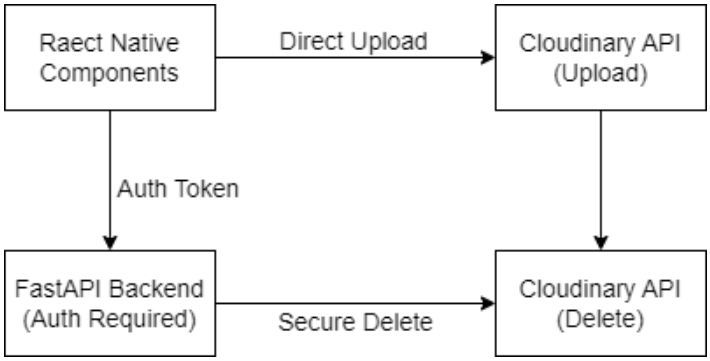


Figure 4.2.3.1 Cloudinary System Architecture Diagram

Google Maps API:

This project adopts an integrated Google Maps API solution to enable location functionality, as well as place searching and interactive map functionality throughout the tourist app. Google Maps API in the project is utilized as the underlying location infrastructure to support destination choosing, attraction finding, accommodation displaying, and visual travel planning by means of integrated map user interface. Both Google Maps JavaScript API and Google Places API are utilized for implementation to enable effortless location-based experiences within the React Native app by utilizing React Native Maps with Google provider implementation and Google Places Autocomplete support.

Front-end project implementation uses dynamic map rendering with different operating modes including destination selection, browsing of attraction list, display of accommodation map, and visualization of trip. Project utilizes point-of-interest click handling, real-time

location services with the option to position users. The map component supports multiple navigation flows from AI-based place selection to manual point selection with regionally oriented intelligence based on selected attractions or accommodations. Google Maps API for the project extends simple mapping to advanced Google Places API-based categorization of Google place types to enable the AISelection component to convert user preference into specific Google place types such as tourist sites, museums, national parks, restaurants, and accommodations.

Figure 4.2.3.2 shows the system architecture of Google Maps API. The GooglePlacesAutocomplete Component is the main user interface for searching locations, submitting queries to the Google Places API for instantaneous suggestions of places and comprehensive details of places. The MapView Component in PROVIDER_GOOGLE integration displays interactive maps through the Google Maps SDK, showing chosen locations with markers and offering visual navigation features. The AISelection Component is an intelligent broker that translates user-friendly category picks to Google's vast place type taxonomy, allowing advanced filtering of attractions and accommodations while user select place by AI modal. The Place Type Categories System analyzes more than 100 various Google Maps place types, translating user preference into executable API queries for advanced location finding. The combined architecture facilitates effortless data flow from user input to Google's services to visual representation of maps, underpinning the entire location-based experience of the tourism application.

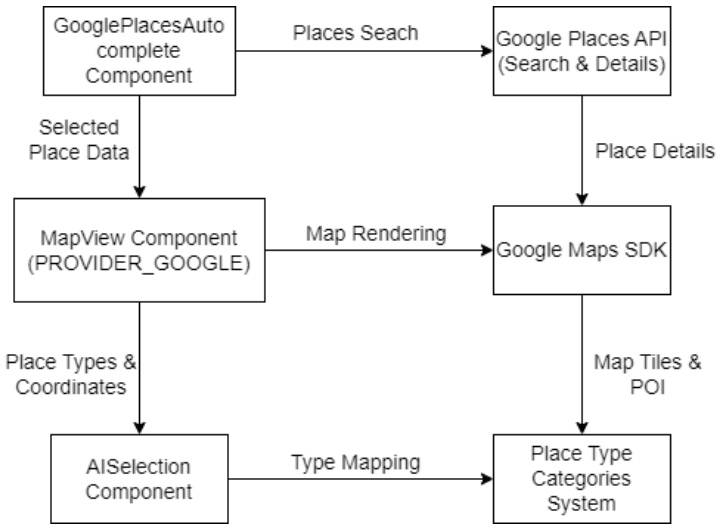


Figure 4.2.3.2 Google Maps API Integration Architecture Diagram

4.3 Application Design Architecture

4.3.1 Class Diagram of the Mobile Application

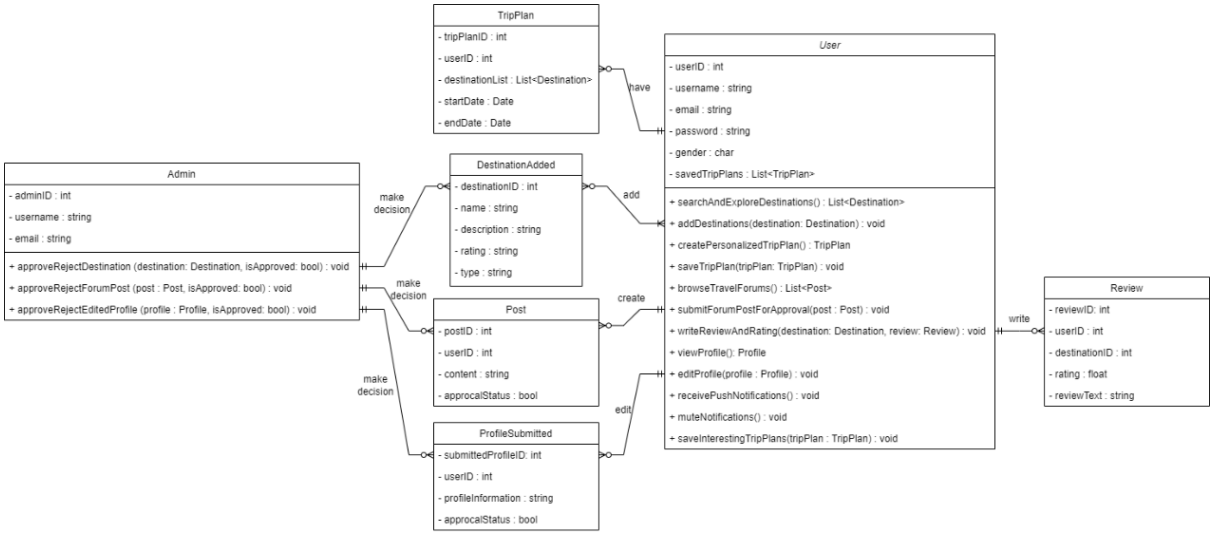


Figure 4.3.1 Class Diagram for Tourism Recommendation Mobile Application

The mobile app's class diagram illustrates user interactions with destinations and trip planning along with forums, reviews, profiles, and administrative functions. The User class serves as the foundational class because it provides functionality for users to browse destinations and search them along with other capabilities like creating trip plans and reading travel forums. Admin class performs independent approval and denial of destinations, forum posts and edited profiles as part of its admin-specific task management. Support classes like Destination, TripPlan, Post, Profile, and Review establish data models for these tasks.

Two destination adding methods are available in this application. Users can choose to add the attractions and accommodation by manually selecting it from map provided or choosing from attractions recommended by AI modal. As users choose the attraction manually from map, he will be prompt that the attractions he chosen will be sent to admin for review and add into system database for system performance improvement. If the user attempts to add a new destination through the addDestination (destination: Destination): void method to create a new destination, the system automatically sends it to admin approval without requiring any submission action. The system supports multiple users adding the same destination because there is a many-to-many relationship between User and Destination which prevents system errors. The system sends a notification about pending approval instead of blocking the

user. After the destination undergoes admin approval through approveRejectDestination (destination, isApproved), the user will receive either of two possible notifications:

- If the destination was already submitted by another user, the user is informed: **"Your destination added with ID xxx has already been added by another user. Please check again."**
- If the destination is unique and accepted, the user receives: **"The destination you added has been successfully approved."**

This design ensures smooth user experience, supports multiple simultaneous submissions without conflict, and delegates final verification responsibility to admins, maintaining data integrity across the application.

4.3.2 Entity- Relationship Diagram (ERD)

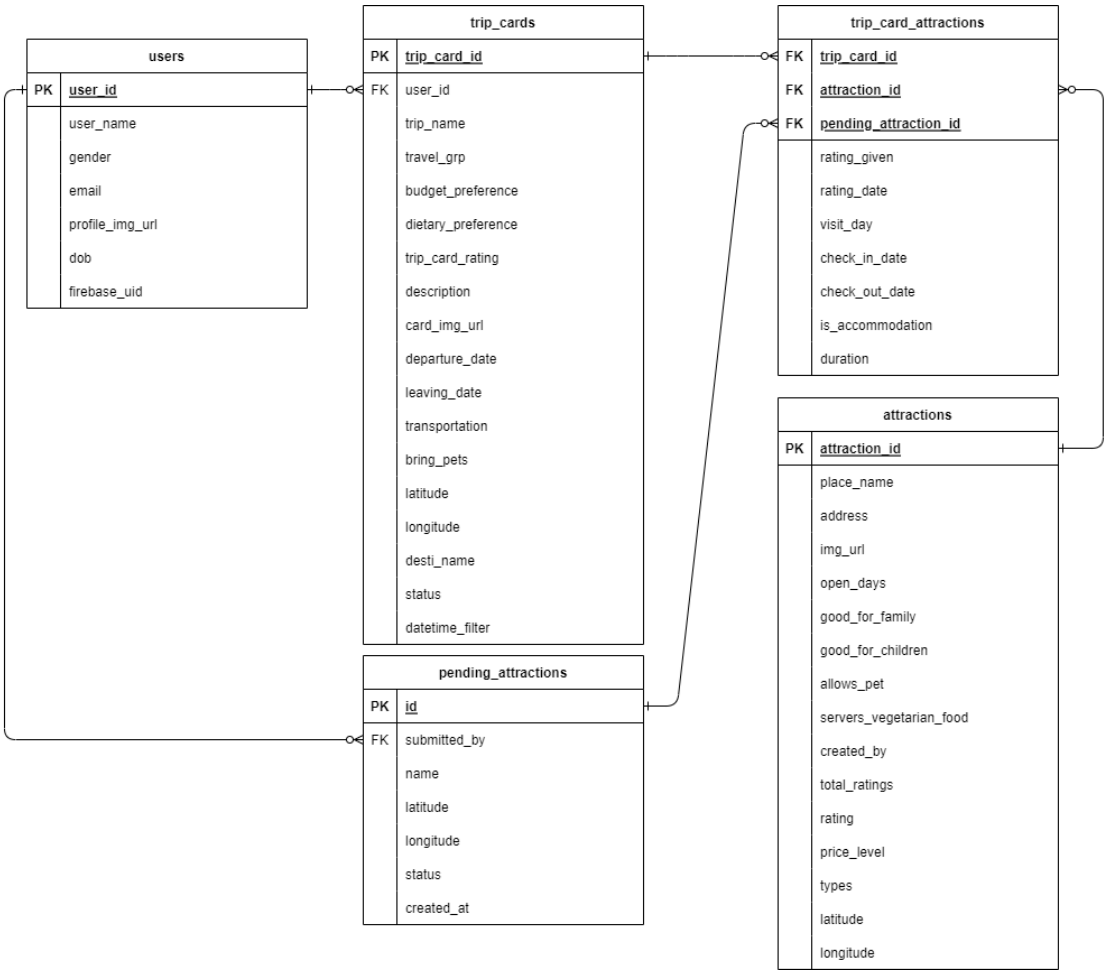


Figure 4.3.2 Mobile Application Database ERD

This Entity Relationship Diagram depicts the detailed database architecture underlying the React Native application for tourism constructed with Expo and Firebase-based integration. The Users table is the underlying entity that holds user profile information and login credentials seamlessly integrated with Firebase Authentication system, wherein the firebase_uid field builds upon the pivotal interconnection of database rows and Firebase authentication tokens. The Trip Cards entity is the hub for all operations of trip planning and holds detailed travel metadata encompassing geographical location for map insertion, temporal scheduling data, user requirements such as budget limitation and dietary needs, mode of transportation, and several status flags that underline the application's fundamental trip management capability.

Attractions management system enforces advanced many-to-many relation architecture by three interlinked entities: the attractions table acting as the master repository of validated tourism sites, the trip_card_attractions junction table that supports agile linking of trips to various attractions, and the pending_attractions entity that supports user-generated content submission with integrated moderation workflows. The normalized design pattern supports reuse of attractions within various trip itineraries in an efficient manner while upholding referential integrity and accommodating scaleable content management. The geographical information kept in trips and attractions entities supports location-based services, map visualization functionality, and proximity-based attraction recommendations.

The database schema directly facilitates the application's primary user journey from logon through trip planning, points of interest choice, and profiling. All database access is secured by Firebase authentication with backend token verification to ensure data protection for all CRUD operations and optimize performance through query pattern optimization. The architecture scales seamlessly from local development environments to production deployment, accommodating real-time data synchronization for many application screens and resilient error handling for network availability failures. This design foundation allows the application to provide thorough tourism planning capability while ensuring data consistency and accommodating future feature augmentation.

4.3.3 API Design and Endpoints Structure

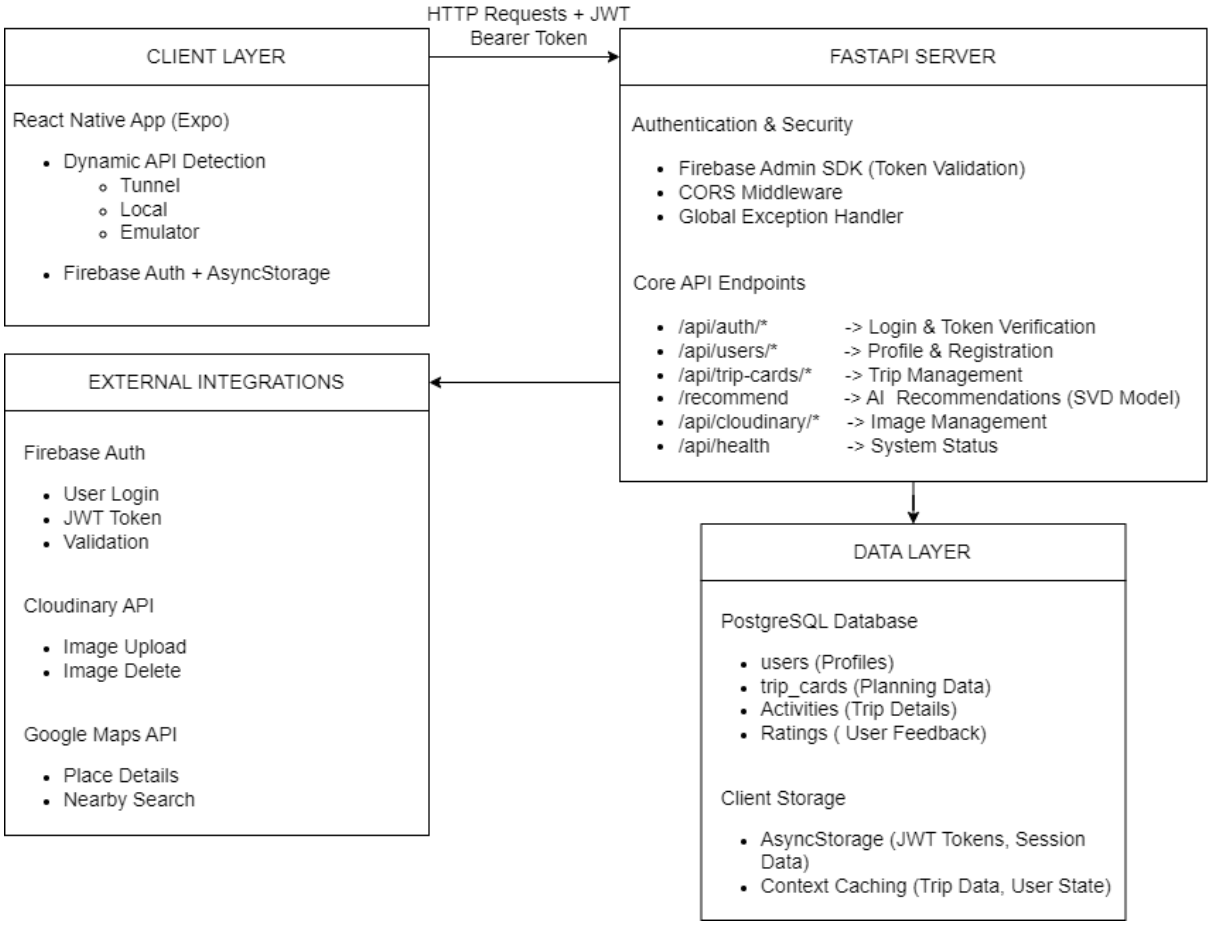


Figure 4.3.3 API Design and Endpoints Structure

This project employs a clean and up-to-date API architecture based on FastAPI with a focus on simplicity and consistency. The system is based upon three crucial principles: smart environment detection, secure authentication, and streamlined data management. Rather than switching server URLs by hand during development time, the app itself detects its environment automatically. Tunnels connect to physical devices via **ngrok** URLs, local development is done via LAN addresses, and emulators are handed their respective localhost configurations. This negates the constant annoyance of switching API endpoints when testing through various environments. On the other hand, Firebase handles user authentication with third-party backend authentication for data management. Upon login by users, Firebase generates a JWT token retrieved locally and added to all API requests. FastAPI server verifies these tokens and handles user data stored in PostgreSQL. This provides us with Firebase's security advantage while allowing full control over our business logic implementation.

The endpoints follow a logical pattern:

Endpoint Description	Endpoint Pattern	Functionalities
Authentication endpoints	(/api/auth/*)	handle login and token validation
User management	(/api/users/*)	manages profiles, registration, and trip data
Trip operations	(/api/trip-cards/*)	handle everything from trip creation to completion
AI recommendations	(/recommend)	processes user data through SVD model for personalized suggestions

Table 4.3.3 API Design Pattern Table

The app's client-side caches trip data cleverly, fetching new data only when necessary. The app stays responsive while always displaying up-to-date info to users. Third-party services like images managed by Cloudinary and location info managed by Google Maps are easily integrated and don't cause any dependency. Error handling is exhaustive to ensure users are presented with useful feedback when things go awry. Automatic retry logic is present along with connection tracking and graceful expiry of session management. Additionally, there is a health check endpoint to show server status briefly for debugging and monitoring purposes. This architecture provides a solid base that is developer-friendly and user-centric yet capable enough to handle its main tourism app functionalities while remaining accommodating to future augmentations.

4.4 System Components Interaction Operations

4.4.1 Authentication Flow Sequence Diagram

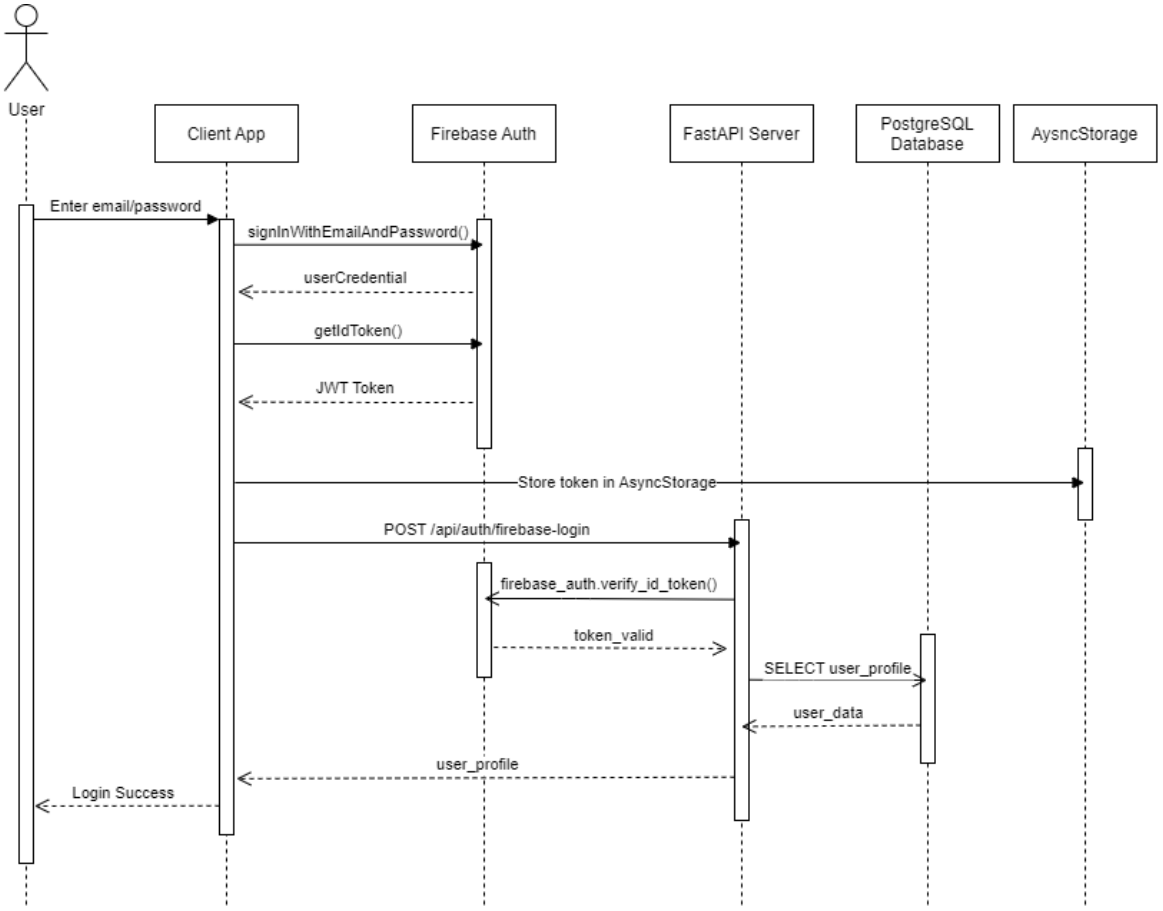


Figure 4.4.1.1 Login Flow Sequence Diagram

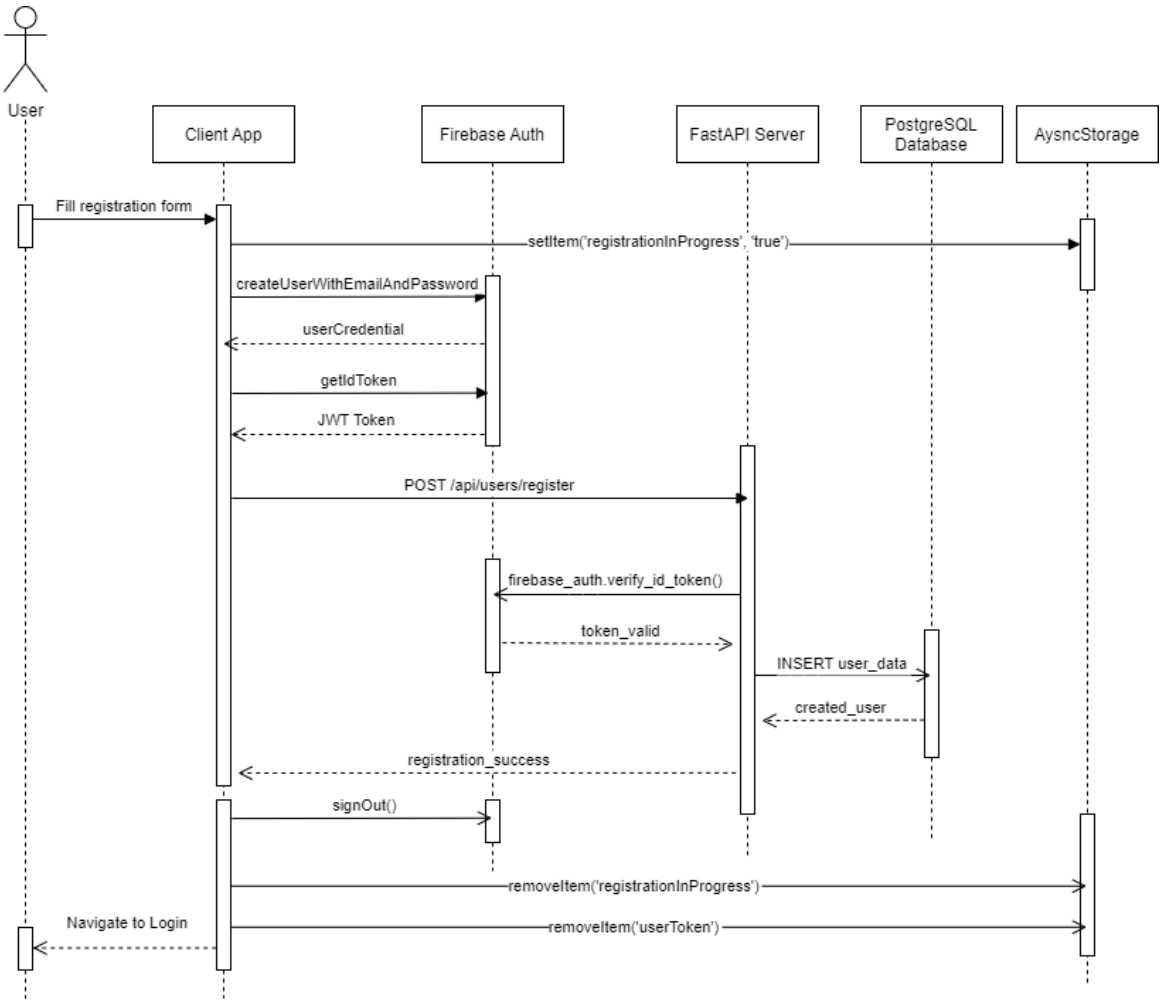


Figure 4.4.1.2 Registration Flow Sequence Diagram

4.4.2 Trip Card Module Workflow Sequence Diagram

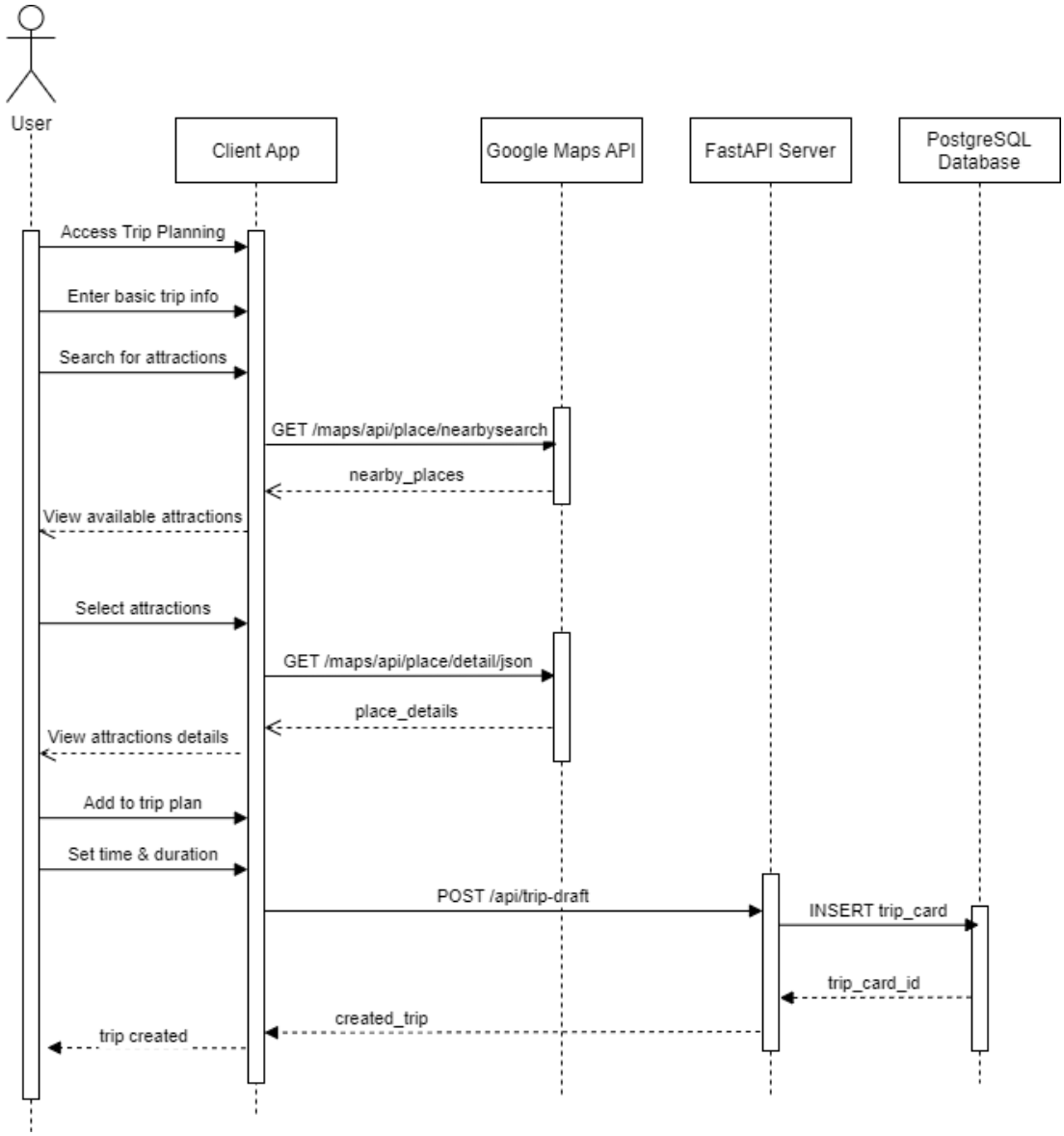


Figure 4.4.2.1 Trip Planning Sequence Diagram

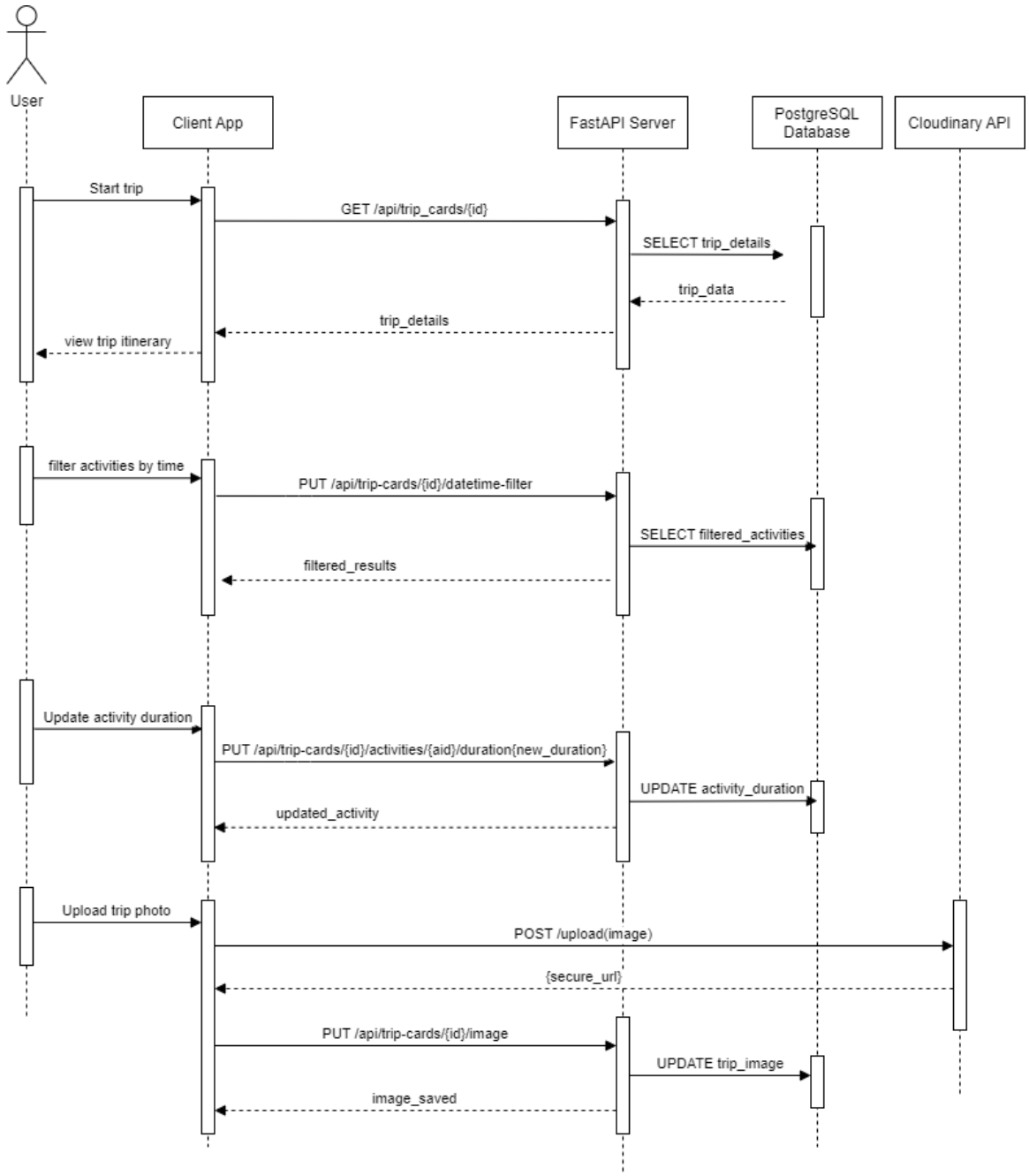


Figure 4.4.2.2 Ongoing Trip Reviewing Sequence Diagram

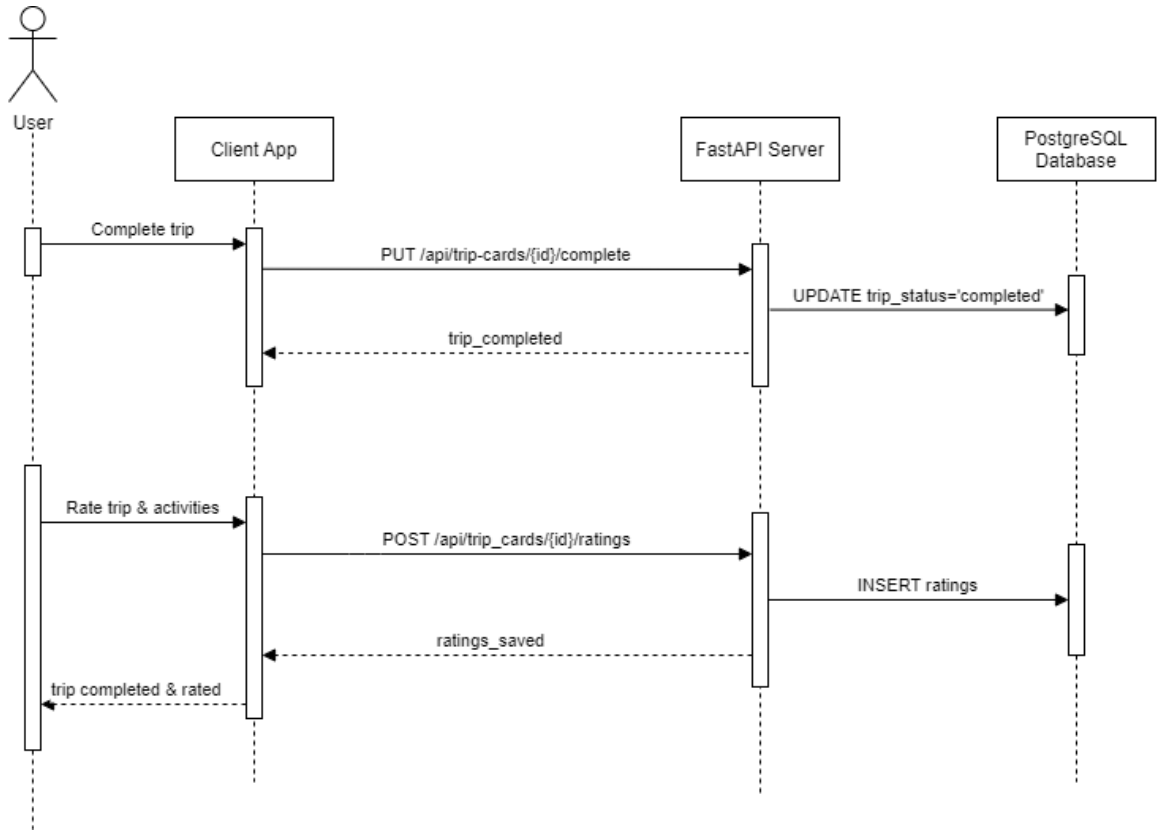


Figure 4.4.2.3 Completed Trip Rating Sequence Diagram

4.4.3 Recommendations Workflow Sequence Diagram

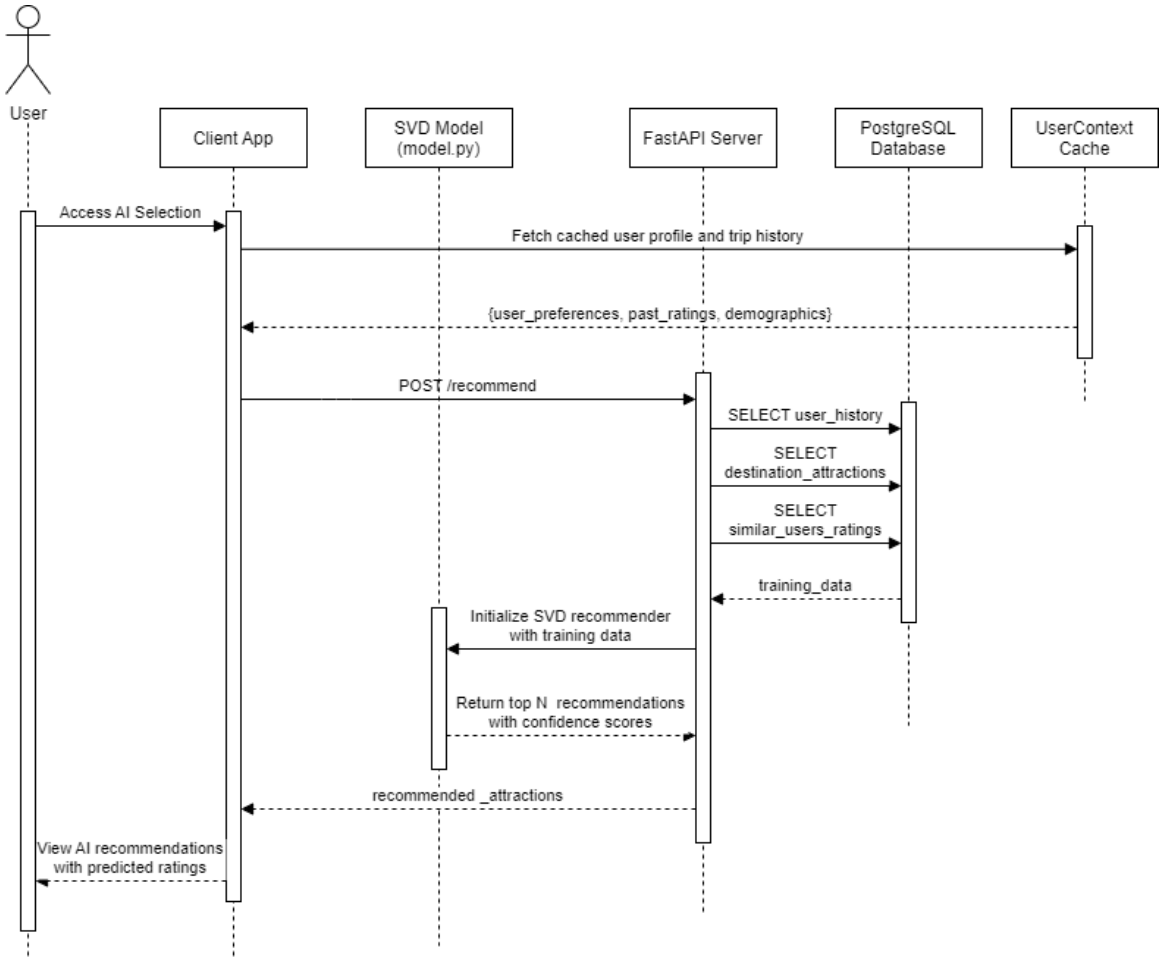


Figure 4.4.3.1 Recommendation Request Sequence Diagram

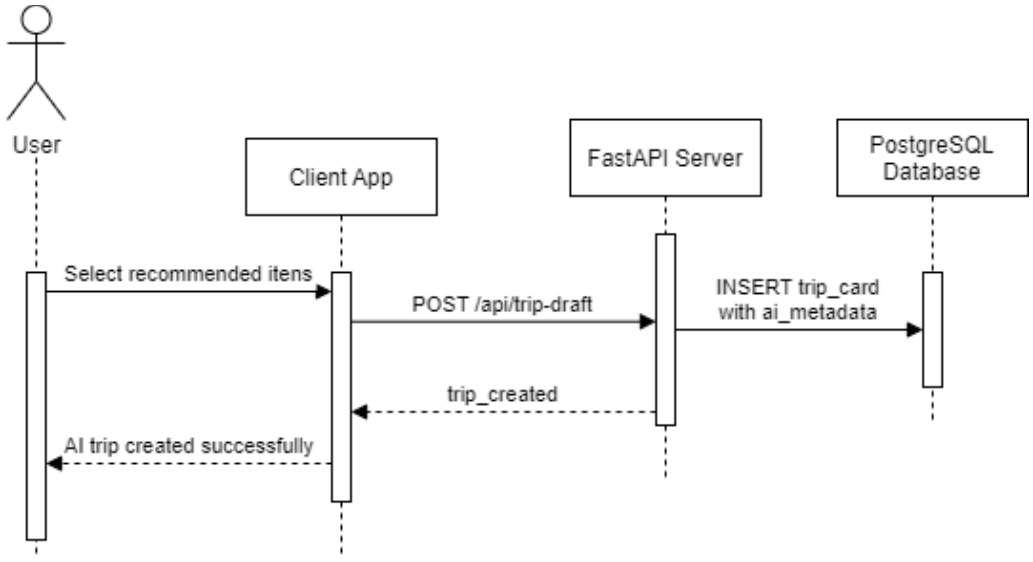


Figure 4.4.3.2 Recommendation Selection Sequence Diagram

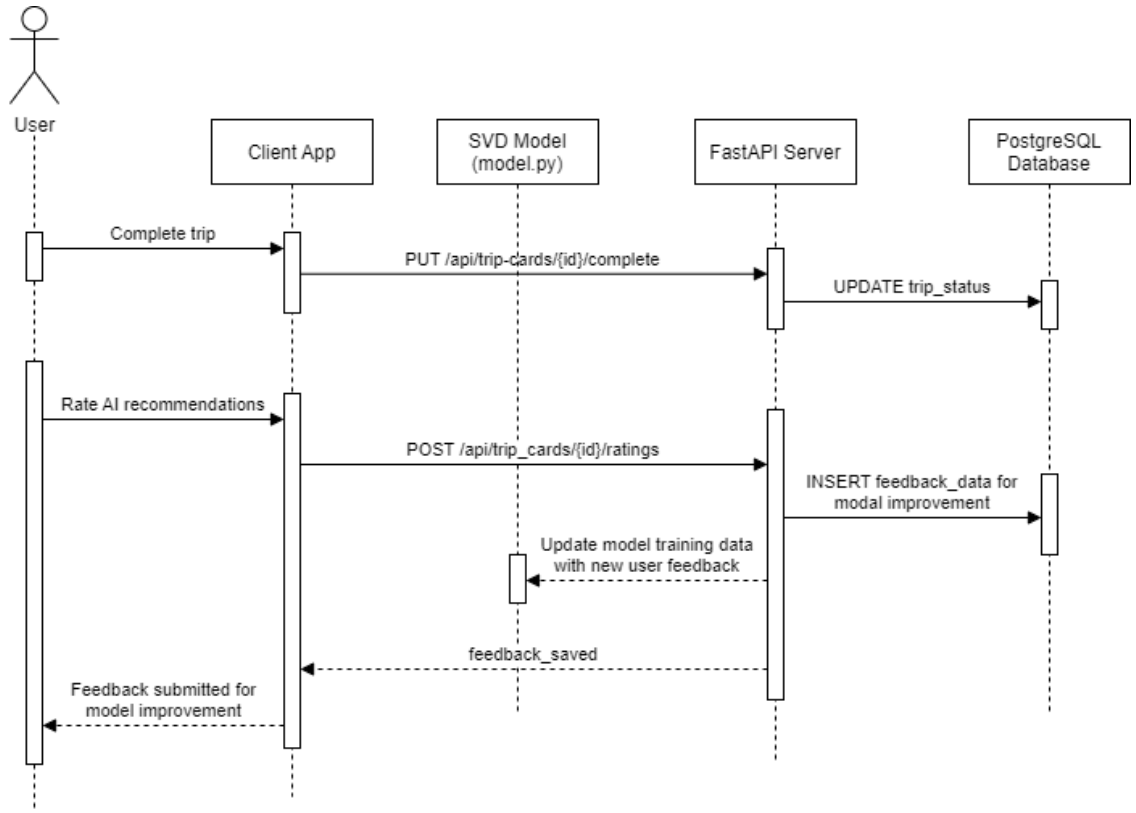


Figure 4.4.3.3 Ratings Feedback Sequence Diagram

4.4.4 Search and Explore Sequence Diagram

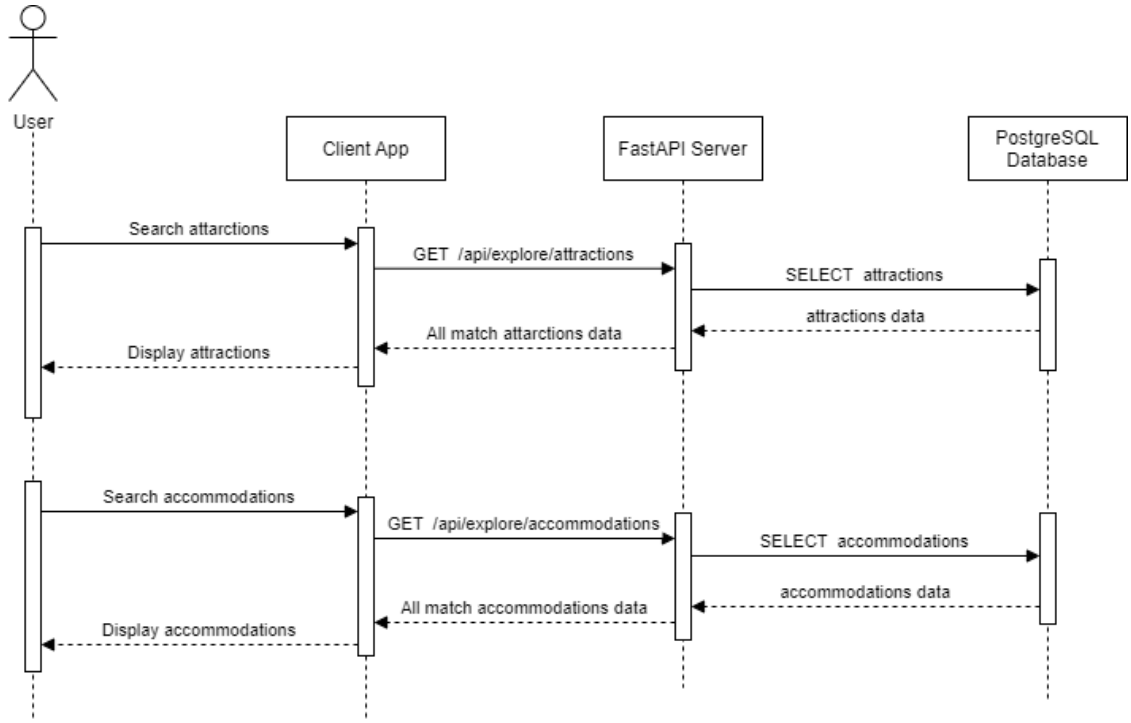


Figure 4.4.4 Ratings Feedback Sequence Diagram

Chapter 5

System Implementation

This chapter explains how the tourism mobile application was developed and put into operation. It is divided into six sections: 5.1 Proposed Methodology, 5.2 Hardware Setup, 5.3 Software Setup, 5.4 Application Interface and Key Features, 5.5 Implementation Issues and Challenges and 5.6 Project Timeline. Each section describes the steps, tools and considerations involved in turning the system design into a functional application.

5.1 Proposed Methodology

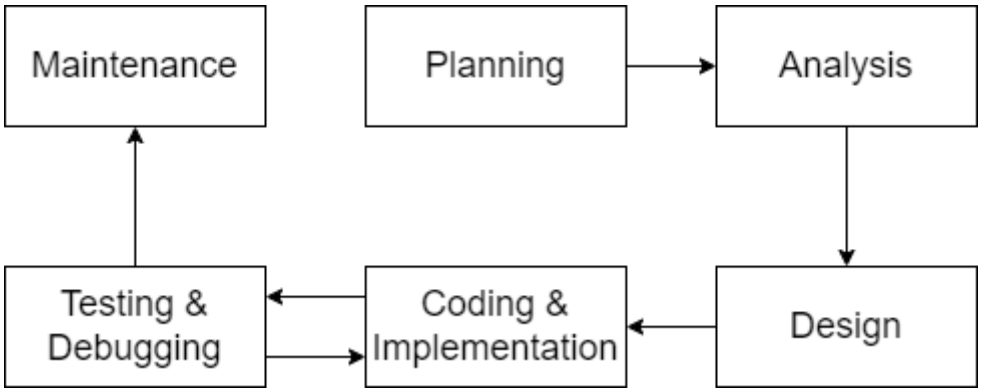


Figure 5.1 System Development Life Cycle

Planning:

In this planning phase, background of the tourism industry and recommendation machine learning industry will be investigated. Problems faced by other researchers or developers will be defined. Three problem statements will then be outlined as the main problems I wish to solve in this project. Project objectives and project scope will be proposed in this phase. Software used will be defined in this phase. React Native for frontend development, FastAPI as backend server, Firebase as user authentication, PostgreSQL for structured data storage and Cloudinary primary for image storage. Besides, Google Places API will be integrated to include the google map and destination manually selection features.

Analysis:

The literature will be reviewed to determine important characteristics, difficulties, and successful strategies employed in similar systems. System requirements will be defined by reviewing previous research to identify the functional and technical requirements, including data sources and integration requirements. All functionality that will be implemented in coding phase will be clarified by reviewing existing application and brainstorming for any possible robust function ca be added. Furthermore, a review of algorithms will be analyzed, compared and chosen for recommendation system. Other than the task mentioned. Below are the additional activities that will be conducted in this phase, which focus on recommendation model machine learning :

Activity 1: Data collection for machine learning.

Activity 2: Data preprocessing for model training.

Activity 3: Training of each machine learning model.

Activity 4: Evaluation and analysis of each machine learning model performance. And the model with highest accuracy will be integrated into mobile applications in Implementation Phase.

Activity 5: Brainstorming functionality can be added and system flow.

Design:

According to the functionalities and system flow outlined at the previous analysis phase, this phase is mainly focused on designing the architecture and other important diagrams for the system. Rough UI of mobile application also will be conducted in this phase. Below are the activities that will be conducted in this phase :

Activity 1: Design system overflow of recommendation engine which will be implemented into the mobile application.

Activity 2: Design system architecture diagram of the mobile application.

Activity 3: Design use case diagram of the mobile application.

Activity 4: Design class diagram of the mobile application.

Activity 5: Design activity diagram of the mobile application.

Activity 6: Design User Interface of the mobile application.

Coding and Implementation:

The coding and implementation phase of this tourism attraction recommendation system will be associated with the development of the system according to design specifications. During this stage of the coding phase, the core functionalities such as a recommendation engine, which is the machine learning model, user profile management, and other important functions stated before will be coded and implemented. This coding and implementation phase is focused on development of functions. Recommendations engine will be integrated into the mobile application in this stage. Sketch diagram of UI in design phase will be coded in this phase. This phase is the most time-consuming phase, which coding, implementing, integrating and debugging and the system always takes a huge amount of time. This phase is an interchanging phase with testing and debugging phase.

Testing :

This phase is conducted simultaneously with the coding and implementation phase; the system will be tested right after new function is coded and implemented to make sure that it is error free and bug free. While a main function is implemented, the overall performance of the system will be tested to prevent any newly added code affecting the existing code's performance and functionality. This phase associated with coding and implementation phase may take a huge amount of time.

Maintenance:

The maintenance phase makes sure that the system is always current, responsive to user needs, and operational. During this phase, the system's performance is regularly monitored, bugs will be fixed, and algorithms will be optimized to increase the accuracy of recommendations as new data becomes available. Updates might be required for integrating new functionality, enhanced UI components, or adapting to modifications in external services like APIs. Additionally,

ongoing data management includes refining data collection methods, cleaning data, and retraining machine learning models will be carried out to keep recommendations relevant. It is a constant process to collect and evaluate feedback from users to make improvements that increase user satisfaction.

5.2 Hardware Setup

This section specifies the necessary hardware setup components used to support the tourism recommendation system. Hardware needed for the application is smart phone with Android operating system. While the local server that handles the application backend processes, including user authentication and database queries. Database for the application also host on this hardware. The server and database specifications are as follows:

Description	Specifications
Model	NBLK-WAX9X
Processor	AMD Ryzen 7 3700U
Operating System	Windows 11 Home Single Language
Graphic	Radeon Vega Mobile Gfx
Memory	8.00 GB RAM
Storage	512 GB SSD

Table 5.2 Specifications of laptop

5.3 Software Setup

5.3.1 Development Environment Configuration

To ensure smooth development of this tourism recommendations system, a properly configured development environment is required. This section describes the step-by-step setup process, including Node.js version, dependency installation and environment variable management.

```
PS C:\Projects\tourism-app> node --version
v22.14.0
```

Node.js provides the JavaScript runtime environment necessary for running React Native applications, while npm is used to install and manage project dependencies.

```
"dependencies": {
  "firebase": "^10.14.1",
  "react": "19.0.0",
  "react-dom": "19.0.0",
  "react-native": "0.79.5",
  "react-native-get-random-values": "^1.11.0",
  "react-native-google-places-autocomplete": "^2.5.7",
  "react-native-maps": "1.20.1",
  "react-native-reanimated": "~3.17.4",
  "react-native-safe-area-context": "5.4.0",
```

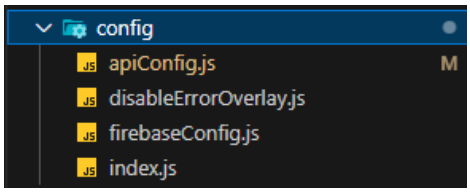
All required project dependencies are listed in the package.json file. They include 31 production dependencies and 12 development dependencies, such as:

- React Native 0.79.5
- React 19.0.0
- Firebase 10.14.1
- React Navigation 7x

Install dependencies using:

```
PS C:\Projects\tourism-app> npm install

up to date, audited 991 packages in 2m

93 packages are looking for funding
  run `npm fund` for details
```

This command downloads all required libraries, including Expo modules, UI components, and third-party integrations, for example google map.



Environment-specific settings are stored in the config/ directory. This centralized configuration approach separates sensitive data and platform-specific settings from the application code. Examples include Firebase authentication setup, API URL management and platform-specific settings.

5.3.2 Build Configuration

To implement and distribute the tourism mobile application, this project uses Expo EAS Build, a cloud-based build and deployment service provided by Expo. Unlike running the app only in Expo Go during development, EAS Build compiles the project into standalone APK, making it production-ready. The setup involves configuring build profiles, initiating builds from the terminal, monitoring progress in the EAS dashboard and finally downloading and installing the generated build on a physical device. The following screenshots illustrate the setup process.

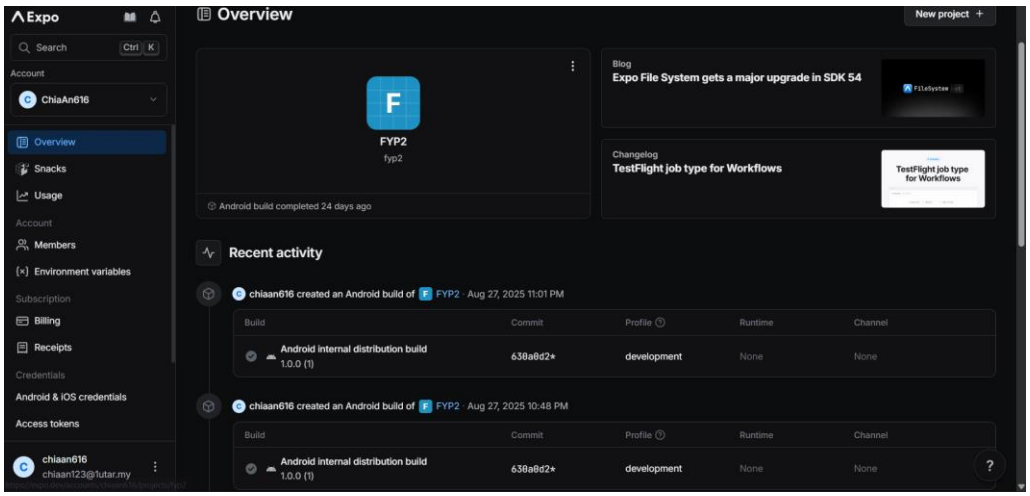
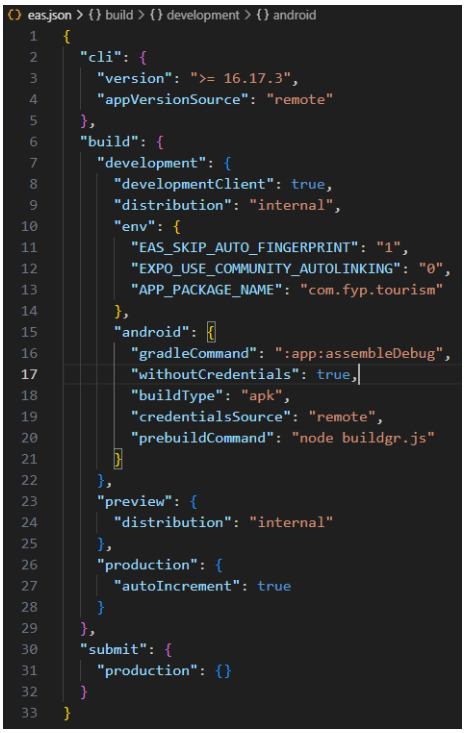


Figure 5.3.2.1 Expo EAS dashboard

The Expo EAS dashboard is the central management panel for cloud builds. In this page, the project is registered, and developers can review build history, logs and configurations details.

This interface provides quick visibility into past and present builds, helping to debug any build failures.



This eas.json file is the backbone of the build setup which defines build profiles such as development, preview and production. Each profile specifies parameters like platform (Android), environment, variables and credential requirements.

```
PS C:\Projects\tourism-app> eas build --platform android
★ eas-cli@16.19.3 is now available.
To upgrade, run:
npm install -g eas-cli
Proceeding with outdated version.

Resolved "production" environment for the build. Learn more: https://docs.expo.dev/eas/environment-variables/#setting-t
No environment variables with visibility "Plain text" and "Sensitive" found for the "production" environment on EAS.

Specified value for "android.package" in app.json is ignored because an android directory was detected in the project.
EAS Build will use the value found in the native code.
✓ Incremented versionCode from 2 to 3.
✓ Using remote Android credentials (Expo server)
√ Generate a new Android Keystore? ... yes
✓ Created keystore

Compressing project files and uploading to EAS Build. Learn more: https://expo.fyi/eas-build-archive
✓ Compressed project files 8s (37.1 MB)
✓ Uploaded to EAS 34s
⌛ Computing the project fingerprint is taking longer than expected...
⏩ To skip this step, set the environment variable: EAS_SKIP_AUTO_FINGERPRINT=1
✓ Computed project fingerprint

See logs: https://expo.dev/accounts/chiaan616/projects/fyp2/builds/7a5bd640-0b8d-43c4-882f-d3ef9fa0c930

Waiting for build to complete. You can press Ctrl+C to exit.
- Build queued...
```

Builds are triggered directly from the terminal using the **eas build** command. For example, e**as build –platform android** uploads the project to Expo servers where it is compiled in a managed environment. This approach eliminates the need for heavy local SDK installations, as Expo handles dependencies and platform-specific compilers in cloud

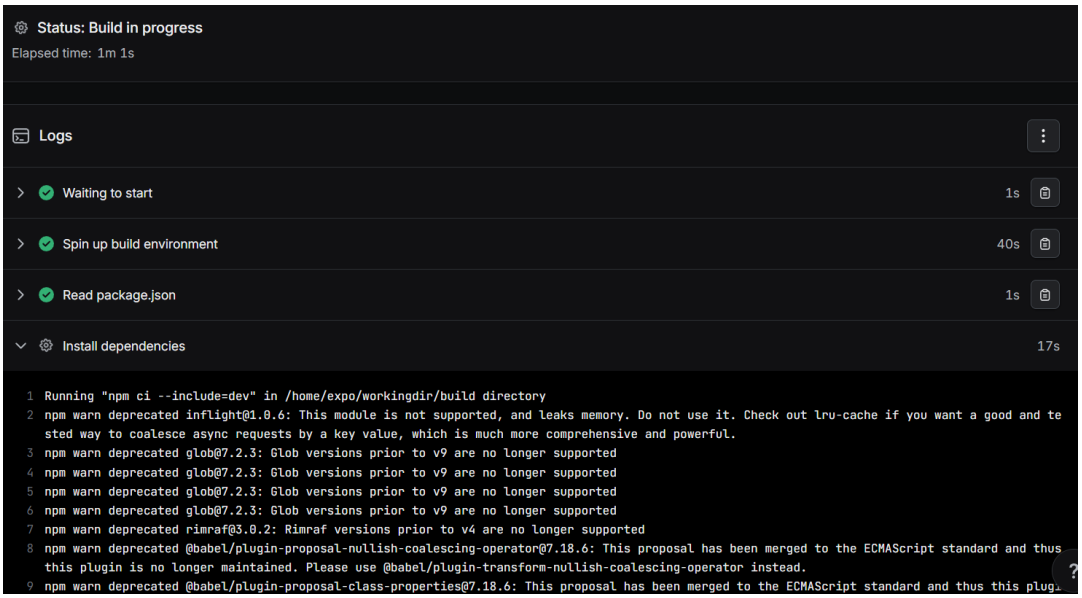
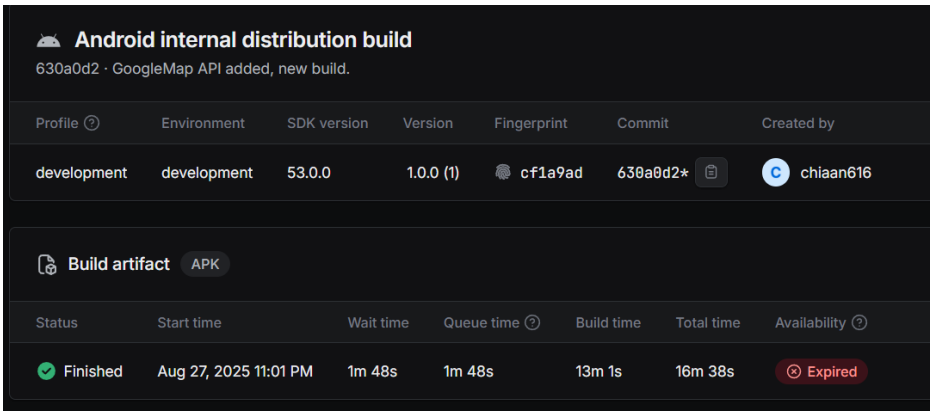


Figure 5.3.2.2 EAS build progress interface

Once a building is initiated, its live progress can be monitored from the Expo dashboard. The progress page displays real-time logs, system output, and the estimated remaining build time.



After a successful build, Expo generates a direct download link for the build artifact (APK for Android, IPA for iOS). The result page summarizes the build details, including the build profile used, commit ID, and output type. The expired keyword is because this project was built on 27 Aug, and this screenshot is taken on 20 September. In normal building flow, the SDK download link will be available for a week.

Figure 5.3.2.3 Expo Go app scanning QR Code

During development, Expo Go plays a key role in quickly testing app changes without rebuilding. By scanning the QR code provided by Expo, the app can be previewed instantly on a physical device.

5.3.3 Firebase Integration Setup

Firebase is integrated into this application to provide secure authentication, persistent user sessions, and platform-specific service configurations. The setup ensures seamless synchronization between the mobile client and backend services while maintaining enterprise-level security standards.

config > firebaseConfig.js > firebaseConfig

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { initializeAuth, getReactNativePersistence } from "firebase/auth";
import AsyncStorage from '@react-native-async-storage/async-storage';

// Firebase configuration
const firebaseConfig = {
  apiKey: "API_KEY",
  authDomain: "tourismfyp-a8085.firebaseapp.com",
  projectId: "tourismfyp-a8085",
  storageBucket: "tourismfyp-a8085.firebasestorage.app",
  messagingSenderId: "978204429155",
  appId: "1:978204429155:android:0d6da32f2897a9f9632156"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Auth with AsyncStorage persistence
const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(AsyncStorage)
});

export { auth };
export default app;
```

Firebase Authentication is initialized with React Native persistence using AsyncStorage. The setup imports initializeAuth and getReactNativePersistence from Firebase v10.14.1, enabling user authentication states to persist across app sessions. The exported auth object is then used throughout the app for login, registration and secure token handling. This ensures authentication tokens are automatically managed and securely refreshed in the background.

5.3.4 Cloudinary Integration Setup

Cloudinary is integrated into the application to manage user-generated images such as profile pictures, trip cover photos, and trip completion uploads. The service provides cloud storage, automatic optimization, and content delivery through a content delivery network (CDN), ensuring fast performance and efficient storage management.

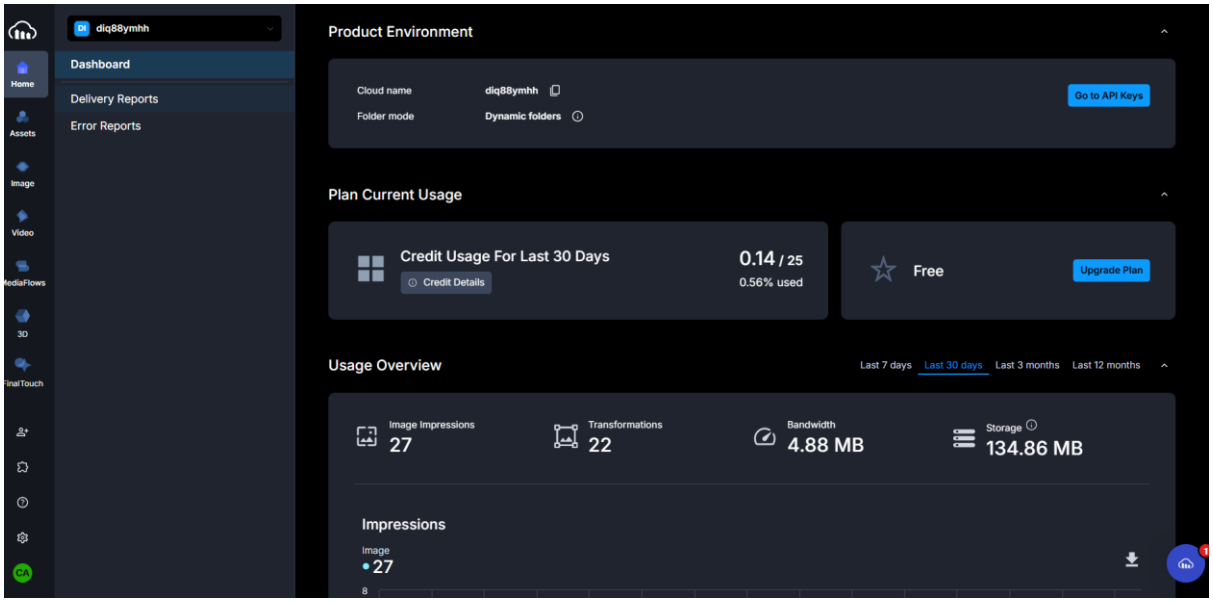


Figure 5.3.4.1 Cloudinary Dashboard

A Cloudinary account was created to handle all image storage and delivery. The account provides a centralized space where uploaded images are managed, optimized, and served securely through Cloudinary's global CDN.

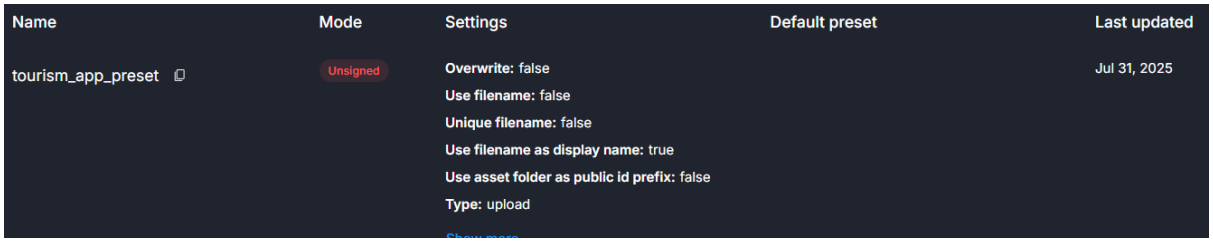


Figure 5.3.4.2 Cloudinary Upload Preset Configuration

An upload preset was configured to allow secure, client-side image uploads from the mobile app. The preset defines rules such as allowed file types, storage folders, and transformations, ensuring that all images meet the application's requirements without exposing sensitive API keys.

```javascript
const uploadImageToCloudinary = async (imageUri) => {
  const data = new FormData();
  data.append('file', {
    uri: imageUri,
    name: 'profile.jpg',
    type: 'image/jpeg'
  });
  data.append('upload_preset', 'tourism_app_preset');

  try{
    const response = await fetch(`https://api.cloudinary.com/v1_1/diq88ymhh/image/upload`, {
      method: 'POST',
      body: data,
    });

    const result = await response.json();
    if (result.secure_url) {
      return result.secure_url;
    } else {
      console.error('Cloudinary did not return a secure URL:', result);
      return null;
    }
  } catch (error) {
    console.error('Error uploading image to Cloudinary:', error);
    return null;
  }
};
```

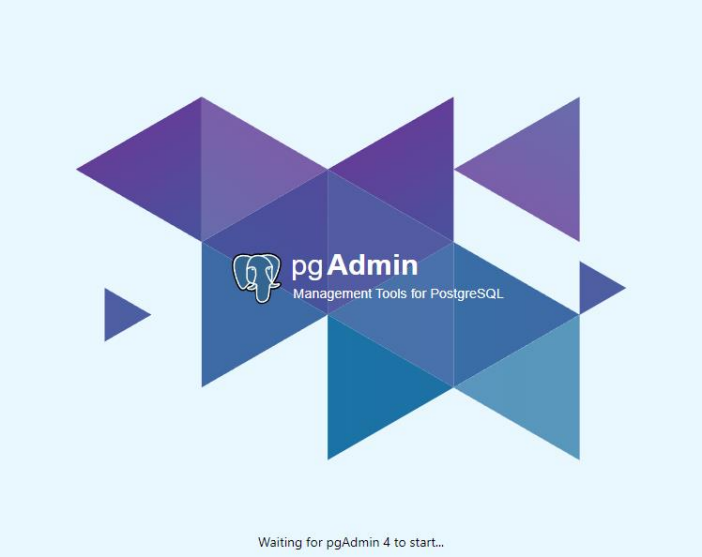
The application allows users to upload images directly from their devices. Images are packaged into a request, sent to Cloudinary's upload API, and then returned as secure URLs. These URLs are saved in the app's backend and displayed throughout the app in features like profile management and trip planning.

```javascript
// Delete image from Cloudinary via backend API
const deleteImageFromCloudinary = async (imageUrl) => {
    if (!imageUrl || !imageUrl.includes('cloudinary.com')) return false;

    try {
        // Get current Firebase user token for authentication
        const currentUser = auth.currentUser;
        if (!currentUser) {
            console.log('User not authenticated, cannot delete image');
            return false;
        }

        // Get fresh ID token
        const idToken = await getIdToken(currentUser, true);

        // Call our backend API endpoint to handle secure deletion
        const endpoint = `${API_URL}/api/cloudinary/delete`;
        console.log('Deleting image via:', endpoint);

        const response = await axios.post(endpoint,
            { image_url: imageUrl },
            {
                timeout: 10000, // 10 second timeout
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${idToken}`
                }
            }
        );
```

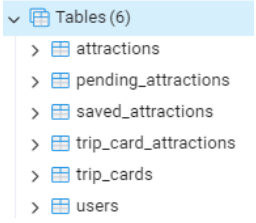
To prevent unused images from accumulating, the system automatically deletes old files when users upload new ones (e.g., updating a profile picture). This ensures efficient storage usage while keeping the user's content up to date.

5.3.5 PostgreSQL Database Configuration

PostgreSQL was chosen as the primary relational database for the tourism application due to its reliability, scalability, and strong support for structured queries. It stores core application data including user profiles, trip information, attractions, and recommendation system results.



PostgreSQL was installed and configured as the main database server. A dedicated database was created for the project, with proper user roles and privileges to ensure secure access. pgAdmin 4 was used for database administration, making it easier to visualize tables, run queries, and monitor data.



```python
# Define ORM models for existing tables
class User(Base):
    __tablename__ = "users"

    user_id = Column(Integer, primary_key=True)
    user_name = Column(String(100))
    gender = Column(String(20), nullable=True)  # Changed from String(1) to allow more options
    email = Column(String(100), unique=True)
    profile_img_url = Column(Text, nullable=True)
    firebase_uid = Column(String(128), nullable=True, unique=True)  # Added for Firebase authentication
    dob = Column(Date, nullable=True)  # Added to store date of birth

    trip_cards = relationship("TripCard", back_populates="user")
    saved_places = relationship("SavedPlace", back_populates="user")
    created_places = relationship("Place", back_populates="creator")

# Other model definitions remain unchanged
class TripCard(Base):
    __tablename__ = "trip_cards"

    trip_card_id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.user_id"))
    trip_name = Column(String(50))
    travel_grp = Column(String(50))
    budget_preference = Column(String(10))
    dietary_preference = Column(ARRAY(Text))
    trip_card_rating = Column(Float)
    description = Column(Text)
    card_img_url = Column(Text)
    departure_date = Column(Date)
```

The backend (FastAPI) connects to PostgreSQL using an ORM (Object Relational Mapper) to simplify query handling. This allows the system to perform operations like:

- Creating new users during registration
- Fetching trip details for planning and ongoing trips
- Storing attraction feedback for recommendation training
- Updating user profiles and preferences

The use of ORM improves code maintainability while preventing SQL injection through parameterized queries.

```
select * from trip_cards
```

Data Output | Messages | Notifications

Showing rows: 1 to 21 | Page No: 1 of 1

	trip_card_id [PK] integer	user_id integer	trip_name character varying (150)	travel_grp character varying (50)	budget_preference character varying (10)	dietary_preference text[]	trip_card_rating double precision	description text	card_img_url text	departure_date date
1	13	3	First Time with Destination	Friends	high	(Vegetarian,Halal)	0			2025-08-27
2	15	3	Testing	Family	medium	(Halal,Vegetarian)	0			2025-08-27
3	16	3	Testing	Family	medium	(Halal,Vegetarian)	0			2025-08-27
4	17	3	Testing With custom destination	Friends	high	(Halal,Vegetarian)	0			2025-08-27
5	26	6	I am Doogy going to Turkey	Friends	high	(Halal,Vegetarian)	0		https://res.cloudinary.com/diq88ymnh/image/upload/v1757090762/xlphr6r69yyrzitqvdl3.jpg	2025-09-06
6	20	5	测试	Couple	low	(Vegetarian)	0	你好玛测试及哈哈糊糊	https://res.cloudinary.com/diq88ymnh/image/upload/v1757924455/jzqk9xpmk9icqoqomkwu...	2025-08-30
7	19	5	测试	Couple	low	(Vegetarian)	0	你好玛测试		2025-08-30
8	32	6	Test with one attractions only	Friends	low	()	0		https://res.cloudinary.com/diq88ymnh/image/upload/v1757157119/h9fix0om8gzkictnkyya.jpg	2025-09-06
9	22	5	Test2 with image upload	Friends	medium	()	0		https://res.cloudinary.com/diq88ymnh/image/upload/v1756574493/zxs9htipijrvzc50bt61.jpg	2025-08-30
10	23	5	Test Saving in Progress	Family	high	(Vegetarian)	0	Test	https://res.cloudinary.com/diq88ymnh/image/upload/v1756575055/j5adr1wirbnddgrcprho.jpg	2025-08-30

PostgreSQL ensures persistent data storage and supports scaling as the system grows. With indexing and optimized queries, the system can handle large datasets efficiently, which is essential for recommendation features that rely on historical user interactions.

5.3.6 AI Recommender Connection Configuration

The core of the system uses a **Singular Value Decomposition (SVD)** based collaborative filtering approach, enhanced with content-based filtering. This hybrid model considers not only user-item interaction history but also contextual trip factors such as **budget, group type, dietary preferences, and travel destination**. The recommender is pre-trained on historical interaction data and can be dynamically updated when new user ratings are added, ensuring that recommendations remain accurate over time.

```python
@app.post("/recommend")
def recommend(req: RecommendationRequest):
    try:
        # Validate dietary options
        valid_dietary_options = ["None", "Vegetarian", "Halal"]
        if req.dietary:
            for dietary_item in req.dietary:
                if dietary_item not in valid_dietary_options:
                    raise HTTPException(
                        status_code=400,
                        detail=f"Invalid dietary value '{dietary_item}'. Must be one of: {', '.join(valid_dietary_options)}"
                    )

        # Run the recommendation engine
        recommendations = recommender.get_recommendations(
            user_id=req.user_id,
            budget=req.budget,
            group=req.group,
            dietary=req.dietary,
            pet_owner=bool(req.pet),
            age=req.age,
            types=req.types,
            destination_lat=req.destination_lat,
            destination_lon=req.destination_lon
        )

        # If error is returned from model (backwards compatibility)
        if isinstance(recommendations, list) and len(recommendations) > 0 and isinstance(recommendations[0], dict) and 'erro
            raise HTTPException(status_code=404, detail=recommendations[0]['error'])
```

The backend provides a dedicated endpoint (/api/recommend) where the mobile app sends user context information. The API processes parameters such as budget, travel group, dietary restrictions, and location, then returns a ranked list of recommended attractions or accommodations. The number of results can be customized from 1 to 20 recommendations providing flexibility for different use cases.

5.4 Application Interface and Key Features

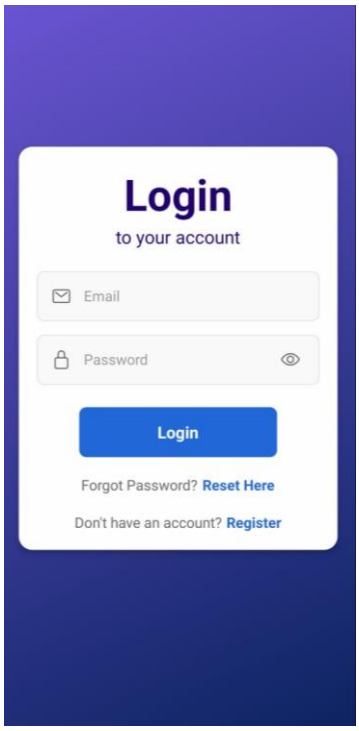


Figure 5.4.1 Login Page

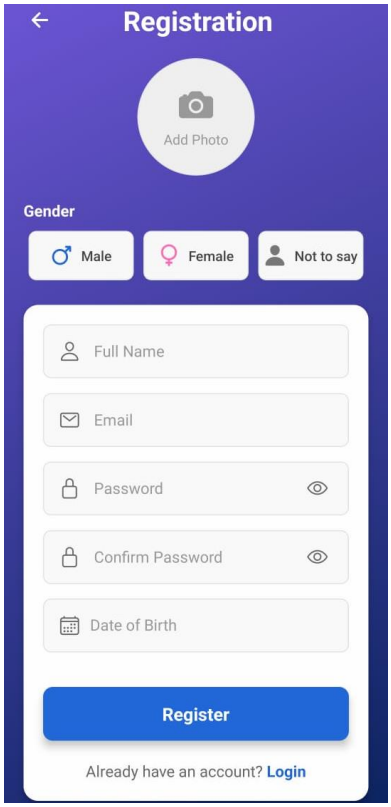


Figure 5.4.2 Registration Page

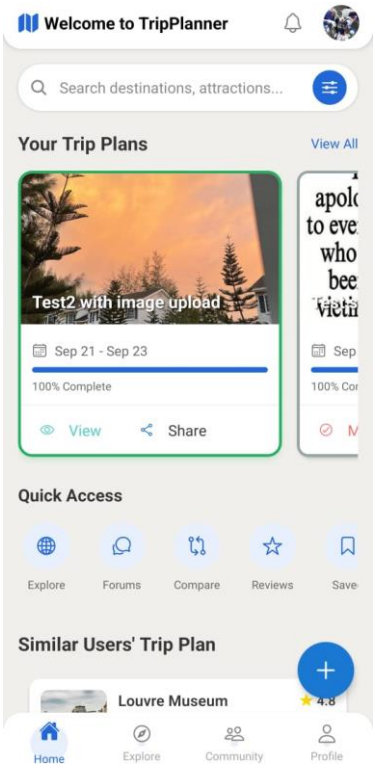


Figure 5.4.3 Home Page

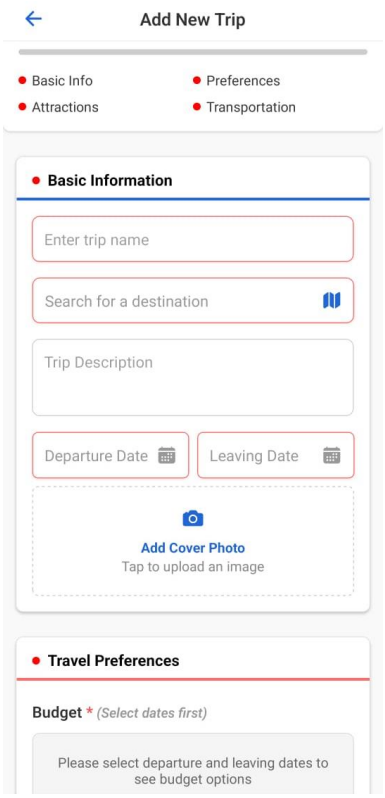


Figure 5.4.4 Trip Planning Page

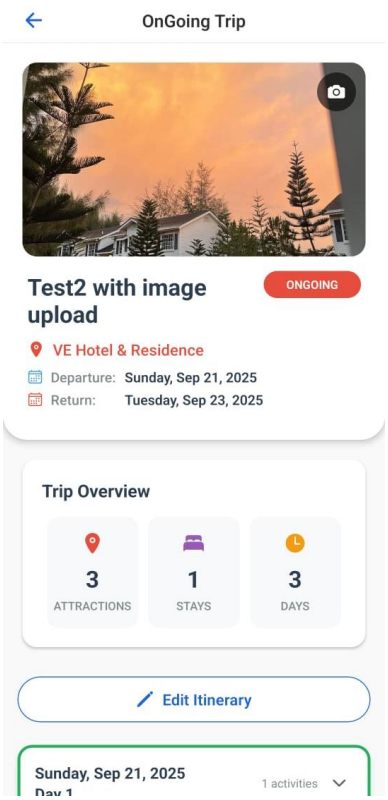


Figure 5.4.5 Ongoing Trip Page

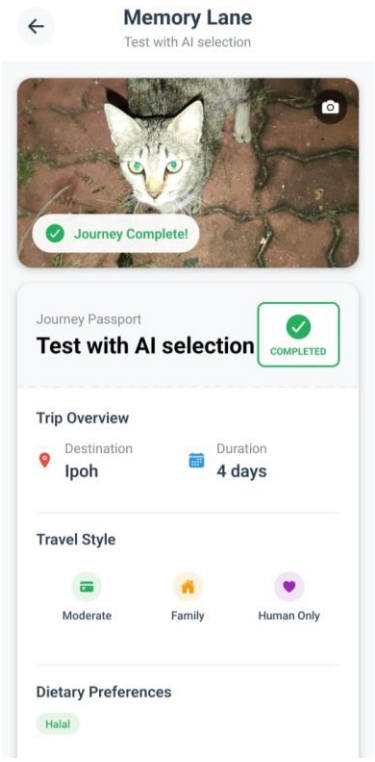


Figure 5.4.6 Completed Trip Page (1)

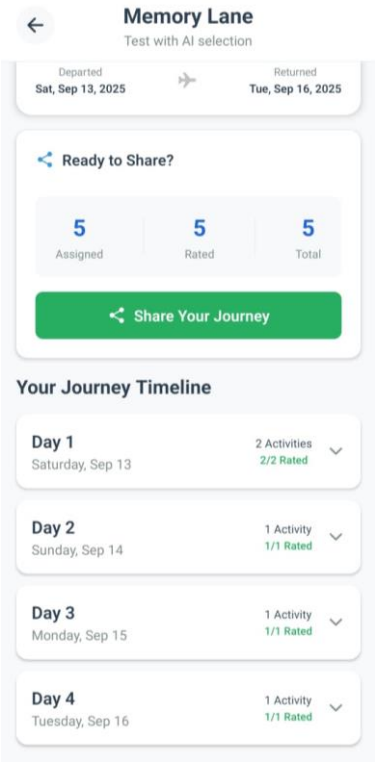


Figure 5.4.7 Completed Trip Page (2)

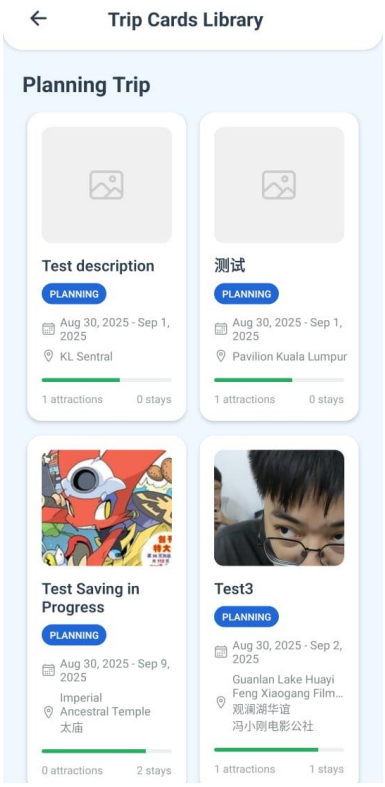


Figure 5.4.8 Trip Cards Library Page(1)

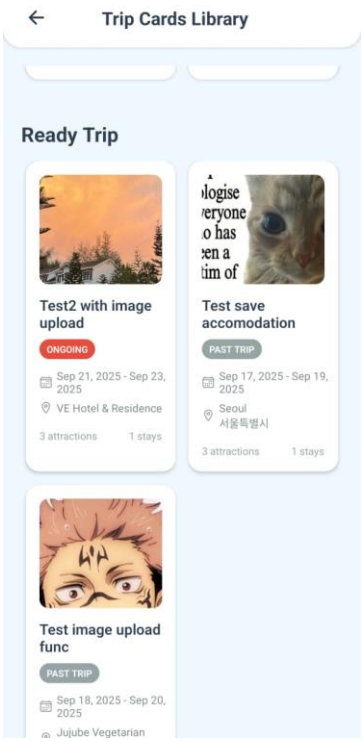


Figure 5.4.9 Trip Cards Library Page(2)

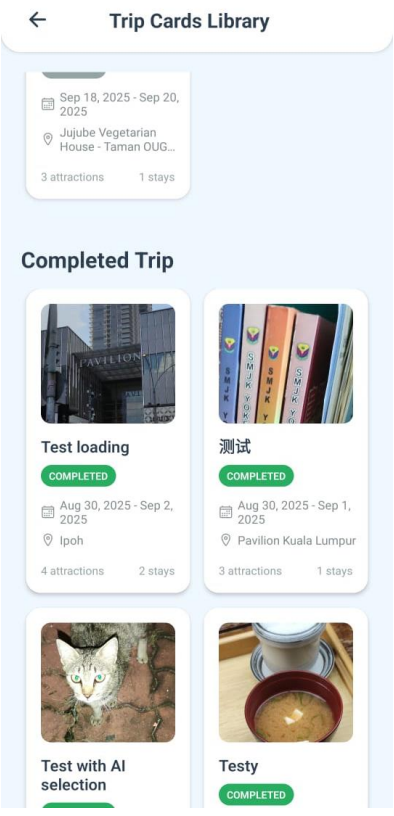


Figure 5.4.10 Trip Cards Library Page(3)

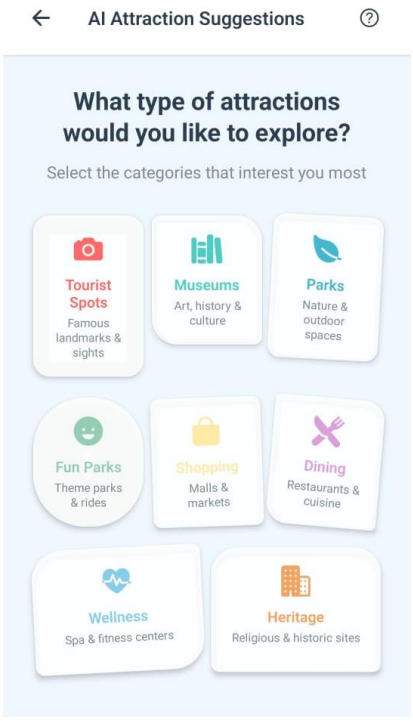


Figure 5.4.11 Attractions/Accommodations Selection Page

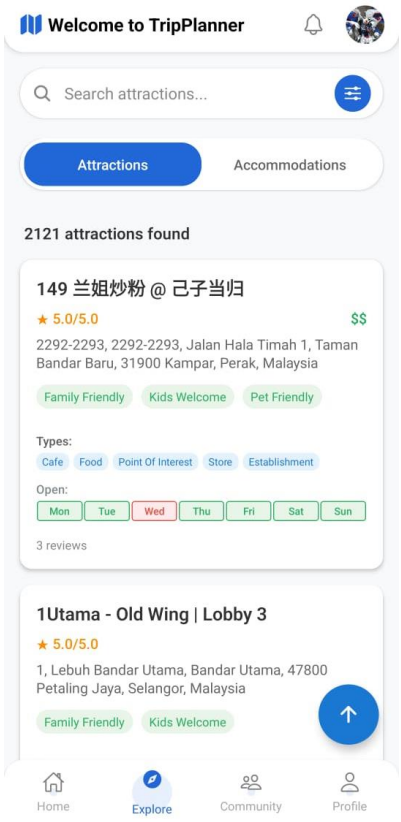


Figure 5.4.12 Attractions/Accommodations Recommendations List

5.5 Implementation Issues and Challenges

Model Selection and Evaluation:

Selecting and comparing models was especially complicated due to the depth and nuances of the analysis. Each of the three algorithms (Funk SVD, Co-Clustering, and KNN with Mean) had different behaviors and bodied differing performance qualities, and it was difficult to determine which would work best given the constraints of a mobile recommendation system. Even a comparison that could be deemed reasonable required a breakdown of the technical issues such as easily implementing a standardized preprocessing pipeline, as well as being mindful of data to avoid bias. This complicating factor heightened the technical challenge.

Also, there was a need to fine-tune the hyperparameters for each model by many trials, which took significant time and computational resources. Not only did it take time and computational resources to run the models, but it took time to verify small performance differences using RMSE and verifying if the performance was consistent over the many runs. Actually, this was the assignment that took the most time because it needed accuracy as well as the capacity to think critically in order to make a decision regarding which model worked best.

Data Preprocessing Complexity:

Data preprocessing was difficult because of the missing and incomplete data that was being returned in the Google API. The data set contained missing values, mixed types, and heterogeneous sets of categorical and binary features that had to be standardized prior to training the model. The challenge was how to deal with the null values for numeric fields such as ratings and price levels without undermining the integrity of the data. This required us to execute multiple imputation processes with care not to add bias.

Further, ensuring categorical attributes like dietary restrictions, travel group, and attraction categories were accurately encoded added to the increased complexity. Binary properties like "Open_Mon" or "Allows Pets" required standardized formatting throughout models as well. Creating an organized and tidy user-item matrix for collaborative filtering methods was extremely time-consuming. Any mistake at this stage would have impacted considerably the performance and equity of the model evaluation, and hence this is an important but demanding part of the pipeline.

Integration of Machine Learning Model into Mobile Application:

There were technical difficulties in integrating the machine learning model with the mobile application. Since the model was trained in Python, it could not be embedded directly into the React Native environment. To overcome this, the model had to be deployed through FastAPI as a backend server, and interaction between the application and the model had to be through Axios for HTTP requests. This required proper serialization of inputs and outputs, secure data handling, and efficient endpoint design to avoid bottlenecks.

Another of the other major challenges was incorporating low-latency responses for real-time recommendations. The system needed to make good recommendations quickly without compromising the user experience through slowdowns. Performance tuning of the model without compromising on accuracy included backend optimization, caching mechanisms, and careful coordination between front-end and back-end layers. Debugging issues across multiple layers, mobile app, API, model, and database also added to the complexity of the integration process.

Third-Party API Limitations

Another concern is the risk of dependency on third-party APIs. Because the application is so reliant upon Google Maps for core functionality, a disruption of the system's functioning would occur with any outage of the services, update of the APIs, or shift in pricing model. For instance, if Google changes up the endpoints of the APIs or authentication mechanism, it may lead to temporary downtime until the application gets updated correspondingly. It poses the problem of retaining system integrity and qualitative end-user experience, particularly in live productions. To counter it, there is the need for prudent API usage tracking, optimal cache policy usage, and adoption of other providers like OpenStreetMap or Mapbox as fallback services in order to retain sustainability in the long run and curb excessive dependency on one single third-party offering.

5.6 Project Timeline

Tourism Recommendation System

Task Name	Duration (week)	Start Date	End Date
Planning Phase			
Review Proposal of Project	1	10/2/2025	16/2/2025
Define Problem Statement	1	10/2/2025	16/2/2025
Identify Objective and Project Scopes	1	10/2/2025	16/2/2025
Plan Project Timeline	1	17/2/2025	23/2/2025
Plan Tools	1	17/2/2025	23/2/2025
Analysis Phase			
Literature Review	3	24/2/2025	16/3/2025
Rview on Algorithms	2	10/3/2025	23/3/2025
Review on Existing Application	2	24/3/2025	6/4/2025
Design Phase			
System overflow Design	1	7/4/2025	13/4/2025
System Architeture Design	1	7/4/2025	13/4/2025
Use Case Design	1	14/4/2025	20/4/2025
Class Diagram Design	1	14/4/1015	20/4/42025
Mobile Application UI Design	3	14/4/1015	4/5/2025
Coding, Implementation and Testing Phase			
AI model development and comparison	5	24/3/2025	27/4/2025
Final Year Project 1 Report Preparation	1	28/4/2025	4/5/2025
Final Year Project 1 Report Presentation	2	5/5/2025	18/5/2025
User Authentication Module	2	23/6/2025	6/7/2025
Tourism Recommendation Engine Integration	2	30/6/2025	13/7/2025
Search and Explore Module	2	7/72025	20/7/2025
Review and Ratings Module	1	21/7/2025	27/7/2025
Travel forums and Community Module	2	28/7/2025	10/8/2025
Activate and Mute Push Notification Module	1	3/8/82025	10/8/2025
Saved Interesting Trip Plan Module	1	11/8/2025	17/8/2025
Edit Profile Module	1	18/8/2025	24/8/2025
Add Destinations Module	1	18/8/2025	24/8/2025
Finalise Product and Maintenance Phase			
Project Testing and Maintenance	3	25/8/2025	14/9/2025
Final Year Project 2 Report Preparation	2	1/9/2025	14/9/2025
Final Year Project 2 Report Presentation	2	15/9/2025	28/9/2025

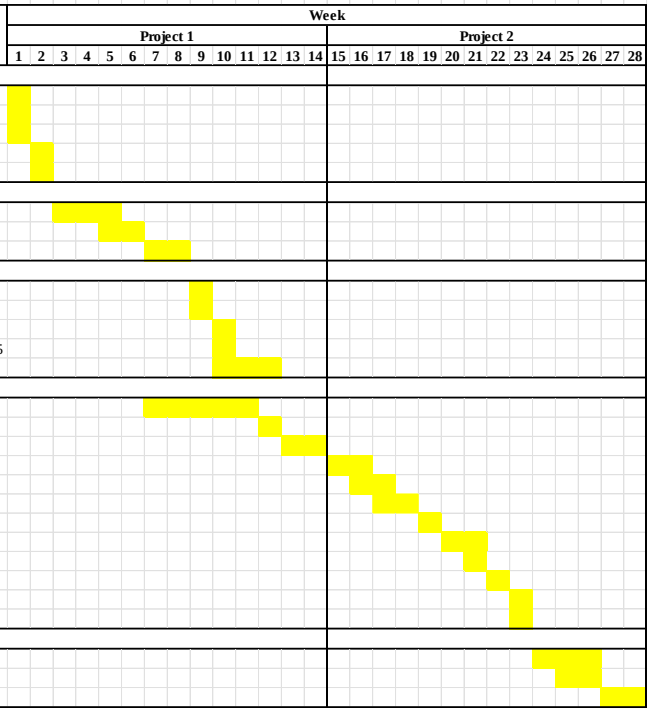


Figure 5.6 Project Timeline Gantt Chart

Chapter 6

System Evaluation and Discussion

6.1 Recommendation Model Performance Analysis

6.1.1 Models Result Comparison

Figure 6.1.1.1 and Figure 6.1.1.2 show the RMSE result of Funk SVD, Co-Clustering and KNN with means. All the models are trained and tested using the same set of pre-processed data. This makes the result reliable and fair to each model. **Analysis and Explanation of result of each model are presented in Chapter 6.1.2.**

```
Running cross-validation comparison...

Cross-validating Funk SVD...
34 iterations to reach convergence

Fold 1/5 - RMSE: 0.8113
34 iterations to reach convergence

31 iterations to reach convergence

Fold 2/5 - RMSE: 0.8365
33 iterations to reach convergence

Fold 3/5 - RMSE: 0.8543
37 iterations to reach convergence

Fold 4/5 - RMSE: 0.8211
36 iterations to reach convergence

Fold 5/5 - RMSE: 0.8548

Cross-validating Co-clustering...
Fold 1/5 - RMSE: 0.9214
Fold 2/5 - RMSE: 0.9259
Fold 3/5 - RMSE: 0.9679
Fold 4/5 - RMSE: 0.9376
Fold 5/5 - RMSE: 0.9514

Model Comparison Results (5-fold Cross-Validation)
==================================================
Model                  Mean RMSE    Std RMSE
--------------------------------------------------
Funk SVD             0.8356 ± 0.0175
Co-clustering        0.9408 ± 0.0170
==================================================
Best model: Funk SVD with Mean RMSE: 0.8356
Model saved for mobile implementation at: mobile_models\best_model_funk_svd_cv.json
```

Figure 6.1.1.1 Results of Funk SVD and Co-Clustering

```
Grid search completed in 399.15 seconds
Best RMSE: 1.0311
Best parameters: {'top_k': 5, 'similarity_method': 'cosine', 'use_pca': False, 'pca_components': None, 'use_item_cf': False, 'alpha': 0.3, 'normalize_pre
dictions': True, 'normalize_ratings': False}

Top 10 Best Results:
   top_k similarity_method  use_pca pca_components  use_item_cf  alpha  \
0      5            cosine    False           None        False    0.3
1      5            cosine    False           None        False    0.3
2      5            cosine    False           None        False    1.0
3      5            cosine    False           None        False    1.0
4     10            cosine    False           None        False    0.3
5     10            cosine    False           None        False    0.3
6     10            cosine    False           None        False    1.0
7     10            cosine    False           None        False    1.0

   normalize_predictions  normalize_ratings      rmse       time
0                   True              False  1.031142  50.320174
1                   True               True  1.031142  53.108348
2                   True              False  1.031142  49.462442
3                   True               True  1.031142  47.563197
4                   True              False  1.032080  51.648017
5                   True               True  1.032080  48.842704
6                   True              False  1.032080  49.720221
7                   True               True  1.032080  47.758962
Results saved to grid_search_results.csv

=== FINAL RESULTS ===
Best RMSE: 1.0311
Best parameters: {'top_k': 5, 'similarity_method': 'cosine', 'use_pca': False, 'pca_components': None, 'use_item_cf': False, 'alpha': 0.3, 'normalize_pre
dictions': True, 'normalize_ratings': False}
```

Figure 6.1.1.2 Results of KNN with means

6.1.2 Model Performance Analysis and Explanation

From section 6.1.1, we can draw the conclusion that Funk SVD is the most accurate tourism recommendation model with RMSE 0.8356. Following by Co-clustering, RMSE 0.9408. Lastly is the KNN with means, RMSE 1.0311. So, the recommendation engine to be integrated into the mobile application will be Funk SVD. Below is the analysis of Explanation of each model:

Funk SVD (Funk Singular Value Decomposition):

```
@numba.njit
def funk_svd_epoch(arr, miu, b_u, b_i, p_u, q_i, alpha, lambda_):
    # Initialize
    error = 0

    # Only iterate known ratings
    for i in range(arr.shape[0]):
        for u in range(arr.shape[1]):
            r_ui = arr[i, u]

            # Skip NaN values
            if np.isnan(r_ui):
                continue

            # Compute error
            epsilon_ui = r_ui - miu - b_u[u] - b_i[i] - q_i[i].T @ p_u[u]
            error += epsilon_ui**2

            # Update parameters
            b_u[u] += alpha * (epsilon_ui - lambda_ * b_u[u])
            b_i[i] += alpha * (epsilon_ui - lambda_ * b_i[i])
            p_u[u] += alpha * (epsilon_ui * q_i[i] - lambda_ * p_u[u])
            q_i[i] += alpha * (epsilon_ui * p_u[u] - lambda_ * q_i[i])

    return error, b_u, b_i, p_u, q_i
```

Figure 6.1.2.1 Funk SVD Sample Code

Funk SVD gave the most efficient performance out of the three models being compared with an RMSE of 0.8356, indicating the highest accuracy level in prediction in the approximate of user ratings of tourist destinations. This makes it the best model to be included in the mobile recommendation system.

Funk SVD is a **matrix factorization algorithm** that factors the user-item rating matrix as latent features vectors of users and item (destinations). The vectors get learned iteratively through stochastic gradient descent to push down the disparity between actual ratings and predicted one. The method also has items' and users' bias terms and uses regularization to

prevent overfitting. Figure 4.3.2.2 shows the visualization of concept of matrix factorization. K serves as the main latent factor which the algorithm aims to find. Right after figure 4.3.2.2 is the main formula in Funk SVD.

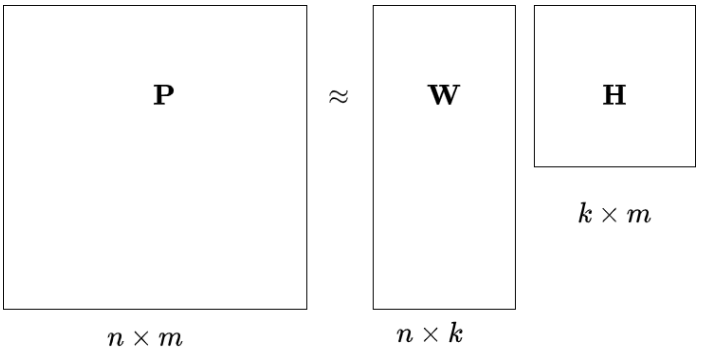


Figure 6.1.2.2 Matrix factorization Visualization

r̂_ui = miu + b_u[u] + b_i[i] + q_i[i].T @ p_u[u]

r̂_ui	predicted rating that user u would give to item i
miu	global average rating across all users and all items
b_u[u]	User bias terms
b_i[i]	Item bias terms
q_i[i].T @ p_u[u]	Latent factors (n*k)(k*m)

Table 6.1.2 Funk SVD Equation Explanation Table

The computational is optimized using Numba acceleration, which speeds up numerical computation on big data. User biases (b_u), item biases (b_i), and the latent factors (p_u, q_i) are updated in each iteration using the error (epsilon_ui) between predicted and actual ratings. One key advantage of Funk SVD in this context is that it is able to handle sparse rating matrices, commonly in recommendation cases where users provide ratings for only a small fraction of available items. Through identification of latent preferences, Funk SVD is able to generalize robustly and effectively predict even items not rated.

Co-Clustering:

```
@numba.njit
def compute_avg(arr, num_of_cluster_item, num_of_cluster_user, cluster_item, cluster_user):
    global_mean = np.nanmean(arr)
    bar_c_ui = np.zeros((num_of_cluster_item, num_of_cluster_user))

    for c_i in range(num_of_cluster_item):
        for c_u in range(num_of_cluster_user):
            row_id = np.where(cluster_item == c_i)[0]
            col_id = np.where(cluster_user == c_u)[0]
            if len(row_id) > 0 and len(col_id) > 0:
                cocluster_sub = arr[row_id][:, col_id].copy().flatten()
                non_nan_values = cocluster_sub[~np.isnan(cocluster_sub)]
                if len(non_nan_values) > 0:
                    bar_c_ui[c_i][c_u] = non_nan_values.mean()
                else:
                    bar_c_ui[c_i][c_u] = global_mean
            else:
                bar_c_ui[c_i][c_u] = global_mean

    miu_i = []
    for i in range(arr.shape[0]):
        subset = arr[i][~np.isnan(arr[i])]
        if len(subset) > 0:
            miu_i.append(subset.mean())
        else:
            miu_i.append(global_mean)

    miu_u = []
    for u in range(arr.shape[1]):
        subset = arr[:, u][~np.isnan(arr[:, u])]
        if len(subset) > 0:
            miu_u.append(subset.mean())
        else:
            miu_u.append(global_mean)

    return bar_c_ui, miu_i, miu_u
```

Figure 6.1.2.3 Co-clustering sample code

Co-clustering achieved an RMSE of 0.9408 and ranked itself as the second most accurate model after Funk SVD. The model clusters both items and users and predicts using the combination of cluster-level statistics, per-item/user biases, and co-cluster means. The algorithm iteratively refines user/item cluster assignments through Expectation-Maximization (E-M) to slowly converge to minimized prediction error. Figure below shows concept visualization of co-clustering:

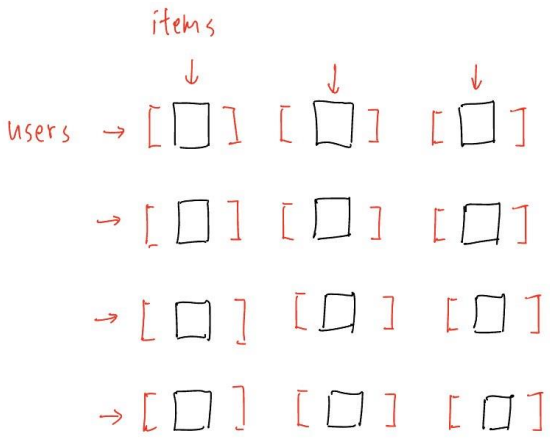


Figure 6.1.2.4 Concept Visualization of Co-Clustering

Each block represents a (4,3) co-cluster where users within that row group have similar preferences for items in that column group.

Back to the algorithm implemented, it initiates by randomly initializing both the users' and items' cluster. In the M-step, mean ratings for each co-cluster, as well as item and user bias measurement, are calculated. Subsequently, in E-step, each user and item are re-assigned to the cluster with the lowest prediction error, determined from the cluster-wise average and personal biases. Predicted ratings of each user will then calculate using the formula below:

estimated_rating = co_cluster_average + (user_average - user_cluster_average) + (item_average - item_cluster_average)

This formula highlights how individual user and item biases are adjusted based on their respective cluster behaviours. So, clustering of user and item is very essential and important, it can directly affect the accuracy of predictions.

This approach balances world structure with local taste in modelling both user- and item-specific behaviour within a lower-dimensional space. Although grounded in discrete cluster affiliation rather than the latent factor decomposition seen in Funk SVD, Co-clustering maintains a respectable rate of prediction accuracy, making it an option for specific situations—especially those where interpretability and group-based results are paramount. However, due to its relatively larger RMSE, it may be less suitable than Funk SVD for applications in which high-resolution personalization is the priority.

KNN with mean:

```python
def calculate_similarity_matrix(user_features, method='cosine', use_pca=False, n_components=None):
    """Calculate similarity matrix between users based on their features."""
    try:
        # Check for NaN values first
        if user_features.isnull().any().any():
            print("WARNING: NaN values found in user_features. Filling with means.")
            user_features = user_features.fillna(user_features.mean())

        # Standardize all features
        scaler = StandardScaler()
        user_features_scaled = scaler.fit_transform(user_features)

        # Apply PCA if requested
        pca = None
        if use_pca and user_features.shape[1] > 3:
            # PCA code remains the same...
            pass

        # Calculate similarity based on chosen method
        if method == 'cosine':
            # Check for NaN values after scaling
            if np.isnan(user_features_scaled).any():
                print("WARNING: NaN values found after scaling. Filling with zeros.")
                user_features_scaled = np.nan_to_num(user_features_scaled)

            similarity_matrix = cosine_similarity(user_features_scaled)
        elif method == 'pearson':
            # For Pearson correlation
            if np.isnan(user_features_scaled).any():
                print("WARNING: NaN values found after scaling. Filling with zeros.")
                user_features_scaled = np.nan_to_num(user_features_scaled)

            similarity_matrix = np.corrcoef(user_features_scaled)
            similarity_matrix = np.nan_to_num(similarity_matrix)
        else:
            raise ValueError(f"Unknown similarity method: {method}")

        # Check for NaN in the similarity matrix
        if np.isnan(similarity_matrix).any():
            print("WARNING: NaN values in similarity matrix. Replacing with zeros.")
```

Figure 6.1.2.5 KNN with mean sample code

This KNN with Mean uses memory-based collaborative filtering to make the missing ratings of user-item matrices. It makes its predictions on the basis of finding similar users or items in an attempt to estimate unknown tastes. For each missing rating, the method calculates the users who have rated the same item (user-based) or items rated by the user previously (item-based) and then goes on to calculate the weighted average of these ratings with mean values tempered.

The word “mean” makes this algorithm different from other normal KNN. The "mean" refers to the average rating that each user gives. The algorithm adjusts for these differences by working with how much each rating deviates from a user's typical rating pattern, rather than

the raw rating values themselves. **For example**, if Alice typically gives 4-star ratings and Bob typically gives 2-star ratings, a 5-star rating from Alice (1 star above her average) and a 3-star rating from Bob (1 star above his average) actually represent the same level of enthusiasm.

This method generates better recommendations since it is reliant on relative preference as compared to the actual values of the ratings. Despite this, KNN with Mean is not as good as methods like Funk SVD. The algorithm does not function well when users have only a few common ratings, prohibiting the identification of meaningful similarity relations. Though computationally simple to read and compute, Mean KNN is not mathematically elegant enough to capture the complex, n-dimensional wants underlying tourism recommendations, in which the idiosyncratic consideration of individuals commonly determines travel destinations in ways not easily imitable by simple similarity measures.

Overall Review:

Based on the comparison of three collaborative filtering models, Funk SVD, Co-Clustering, and KNN with Mean, Funk SVD was the most predicting algorithm for the users' ratings for tourist spots with the minimum RMSE being 0.8356. As a matrix factorization technique, Funk SVD **learns latent item and user features**, contains user and item biases, and uses regularization to avoid overfitting, making it very accurate and efficient in **handling sparse datasets** prevalent in recommendation systems. Numba acceleration also increased computational efficiency, wherein it was able to do well if mobile real-time use was needed. Co-Clustering with an RMSE = 0.9408 provided relatively moderate levels of prediction accuracy by clustering both users and items into co-clusters and iteratively optimizing using Expectation-Maximization. Although less accurate in its predictions, Co-Clustering offers higher interpretability and is better suited to scenarios where group behaviour interpretation is paramount. Lastly, the memory-based algorithm KNN with Mean, which corrects for the user rating trends, performed the lowest across the three. While elementary and easy to implement, it is still not sophisticated enough to identify the complex, high-dimensional flavor profiles characteristic of travel recommendations, especially with data scarcity. In general, **Funk SVD is the most suitable model** to be implemented into the mobile travel recommendation system, offering the best balance between precision, scalability, and responsiveness to user activities.

Additionally, Funk SVD will then be optimized by include content-base filtering approaches. This can make the Funk SVD be more powerful in handling cold start problem, not only good handling data sparsity.

6.2 System Testing

System testing is carried out to verify that the tourism recommendation system functions as intended, meets user requirements, and ensures reliable operation across different environments. Testing was performed to validate both functional correctness and integration of the mobile application, backend services, and database, as well as to assess usability and performance.

6.2.1 Testing Setup and Result

6.2.1.1 User Authentication and Profile Management

Test Case ID	AUTH-001
Test Scenario	User Registration - Valid Data
Test Steps	1. Open registration screen 2. Enter valid email 3. Enter strong password 4. Confirm password 5. Click Register
Expected Result	User account created successfully, redirected to profile setup
Test Data	Email: test@example.com Password: Test123!

Table 6.2.1.1.1 Authentication Test AUTH-001

Test Case ID	AUTH-002
Test Scenario	User Registration - Invalid Email
Test Steps	1. Open registration screen 2. Enter invalid email format 3. Enter password 4. Click Register
Expected Result	Error message displayed: "Invalid email format"
Test Data	Email: invalid-email Password: Test123!

Table 6.2.1.1.2 Authentication Test AUTH-002

Test Case ID	AUTH-003
Test Scenario	User Login - Valid Credentials
Test Steps	1. Open login screen 2. Enter valid email/password 3. Click Login
Expected Result	User logged in, redirected to Home screen
Test Data	Existing user credentials

Table 6.2.1.1.3 Authentication Test AUTH-003

Test Case ID	AUTH-004
Test Scenario	User Login - Invalid Credentials
Test Steps	1. Open login screen 2. Enter invalid credentials 3. Click Login
Expected Result	Error message: "Invalid email or password"
Test Data	Wrong credentials

Table 6.2.1.1.4 Authentication Test AUTH-004

Test Case ID	AUTH-005
Test Scenario	Profile Creation - Complete
Test Steps	1. Complete registration 2. Enter full name 3. Select gender 4. Set date of birth 5. Upload profile picture
Expected Result	Profile created successfully, redirected to Home
Test Data	Valid profile data

Table 6.2.1.1.5 Authentication Test AUTH-005

6.2.1.2 Trip Planning and Management

Test Case ID	TRIP-001
Test Scenario	Profile Creation - Complete
Test Steps	1. Navigate to Trip Planning 2. Click "Create New Trip" 3. Enter trip name 4. Set start/end dates 5. Add description
Expected Result	Trip created with "Draft" status
Test Data	Name: "Tokyo Adventure" Dates: 7 days Description: "Exploring Japan"

Table 6.2.1.2.1 Trip Management Test TRIP-001

Test Case ID	TRIP-002
Test Scenario	Add Attractions - Manual Selection
Test Steps	1. Open existing draft trip 2. Navigate to Attractions section 3. Click "Add Manually" 4. Search and select attractions
Expected Result	Attractions added to trip itinerary
Test Data	Tourist attractions in destination

Table 6.2.1.2.2 Trip Management Test TRIP-002

Test Case ID	TRIP-003
Test Scenario	Add Accommodations with Dates
Test Steps	1. Open draft trip 2. Navigate to Accommodations 3. Search for hotels 4. Set check-in/out dates 5. Confirm selection
Expected Result	Accommodation added with date ranges
Test Data	Hotel: 3 nights, specific dates

Table 6.2.1.2.3 Trip Management Test TRIP-003

Test Case ID	TRIP-004
Test Scenario	Schedule Activities - Time Slots
Test Steps	1. Open trip with attractions 2. Navigate to Schedule section 3. Drag activities to time slots 4. Save schedule
Expected Result	Activities scheduled in time grid
Test Data	Morning/afternoon/evening slots

Table 6.2.1.2.4 Trip Management Test TRIP-004

Test Case ID	TRIP-005
Test Scenario	Trip Status Change - Confirm Trip

Test Steps	1. Complete all trip sections 2. Click "Confirm Trip" 3. Verify status change
Expected Result	Trip status changes from "Draft" to "Ongoing"
Test Data	Completed trip details

Table 6.2.1.2.5 Trip Management Test TRIP-005

Test Case ID	TRIP-006
Test Scenario	Invalid Date Selection
Test Steps	1. Create new trip 2. Set end date before starting date 3. Try to save
Expected Result	Error message: "End date must be after start date"
Test Data	Invalid date range

Table 6.2.1.2.6 Trip Management Test TRIP-006

Test Case ID	TRIP-007
Test Scenario	Maximum Trip Duration
Test Steps	1. Create trip with 31+ days 2. Try to save
Expected Result	Warning about extended trip duration
Test Data	35 days duration

Table 6.2.1.2.7 Trip Management Test TRIP-007

6.2.1.3 AI Recommendation System

Test Case ID	AI-001
Test Scenario	Get Attraction Recommendations
Test Steps	1. Navigate to AI Selection 2. Select "Attractions" 3. Choose categories (Tourist Spots, Museums) 4. Click "Get AI Recommendations"
Expected Result	AI recommendations displayed with ratings and details
Test Data	Multiple attraction categories

Table 6.2.1.3.1 AI Recommendation Test AI-001

Test Case ID	AI-002
Test Scenario	Select Recommendation for Trip
Test Steps	1. Get AI recommendations 2. Click on recommendation card 3. View details 4. Click "Confirm Selection"
Expected Result	Recommendation added to trip planning
Test Data	Selected attraction/hotel

Table 6.2.1.3.2 AI Recommendation Test AI-002

Test Case ID	AI-003
Test Scenario	No Recommendations Available
Test Steps	1. Select very specific categories 2. Request recommendations 3. Check response
Expected Result	Message: "No recommendations found for selected criteria"
Test Data	Rare category combinations

Table 6.2.1.3.3 AI Recommendation Test AI-003

Test Case ID	AI-004
Test Scenario	Check Recommendation on Map
Test Steps	1. Get recommendations 2. Select recommendation 3. Click "Check on Map"
Expected Result	Map opens showing selected location
Test Data	Recommendation with coordinates

Table 6.2.1.3.4 AI Recommendation Test AI-004

6.2.1.4 Map Integration and Location Services

Test Case ID	MAP-001
Test Scenario	Search Location - Autocomplete
Test Steps	1. Open Map screen 2. Type location name 3. Select from suggestions 4. Confirm selection
Expected Result	Location found and displayed on map
Test Data	"Tokyo Tower, Japan"

Table 6.2.1.4.1 AI Recommendation Test MAP-001

Test Case ID	MAP-002
Test Scenario	Select Location by Tap
Test Steps	1. Open map interface 2. Tap on location 3. View location details 4. Add to trip
Expected Result	Location selected and added to trip
Test Data	Map coordinates

Table 6.2.1.4.2 AI Recommendation Test MAP-002

Test Case ID	MAP-003
Test Scenario	Location Search - No Results
Test Steps	1. Search for non-existent places 2. Check results 3. Try alternative search
Expected Result	"No results found" message displayed
Test Data	"Nonexistent Place XYZ"

Table 6.2.1.4.3 AI Recommendation Test MAP-003

Test Case ID	MAP-004
Test Scenario	Map Navigation - Zoom/Pan
Test Steps	1. Open map interface 2. Zoom in/out using gestures 3. Pan around map 4. Reset view
Expected Result	Smooth map navigation and reset
Test Data	Touch gestures

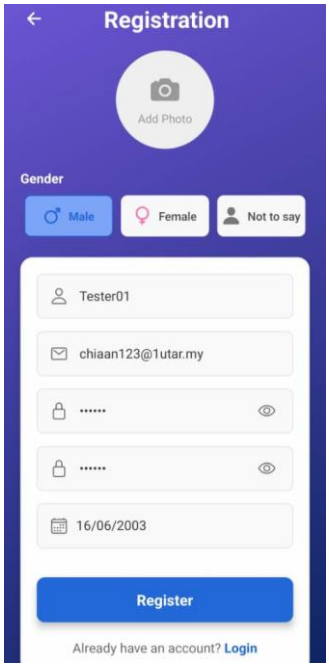
Table 6.2.1.4.4 AI Recommendation Test MAP-004

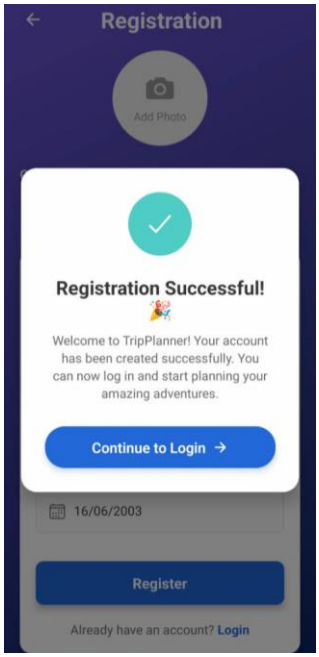
Test Case ID	MAP-005
Test Scenario	GPS Permission Denied
Test Steps	1. Open map 2. Deny location permission 3. Try to access current location
Expected Result	Error message: "Location permission required"
Test Data	Permission denied

Table 6.2.1.4.5 AI Recommendation Test MAP-005

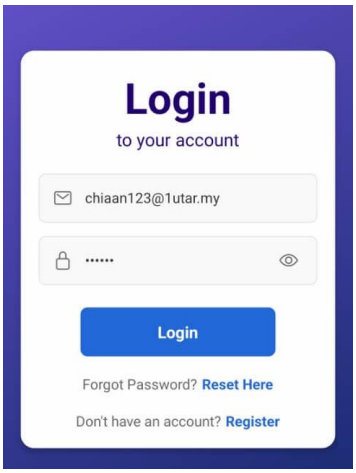
6.2.2 Functional Flow Testing

First, we conduct functional testing by registering a new account first.

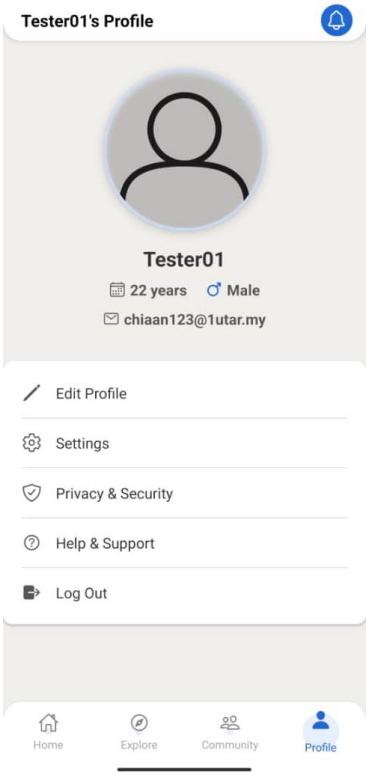




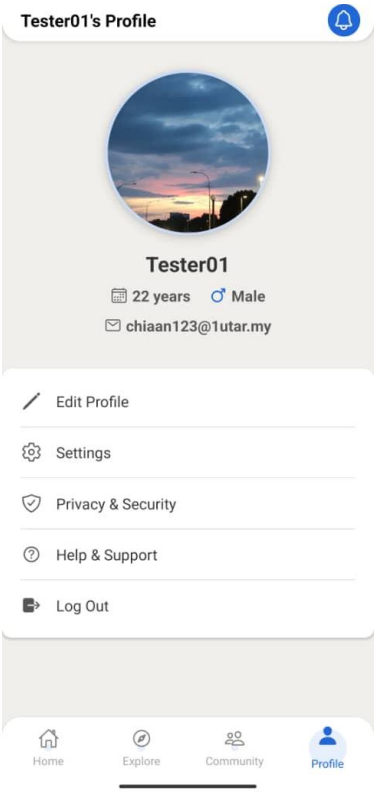
After successfully register, login with registered credentials



This profile page shows that user successfully registered and log in.

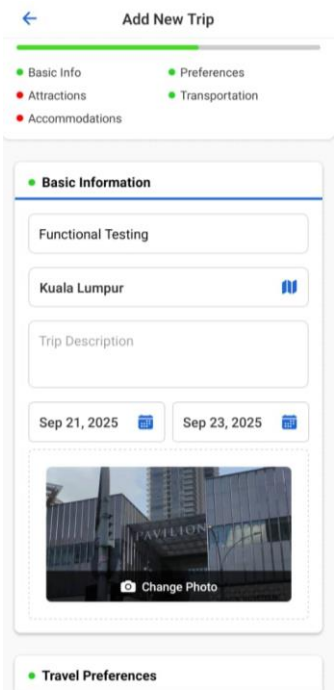


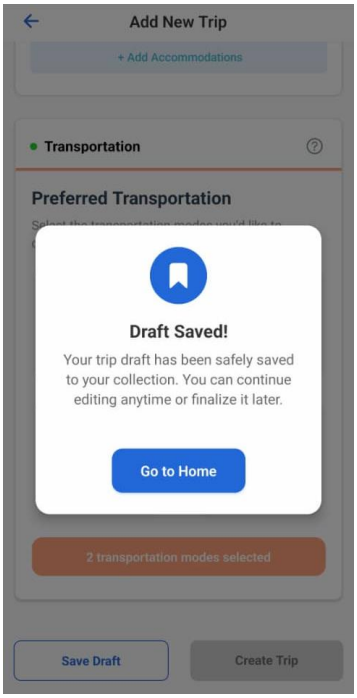
Next, we try the image upload features by adding user profile image.



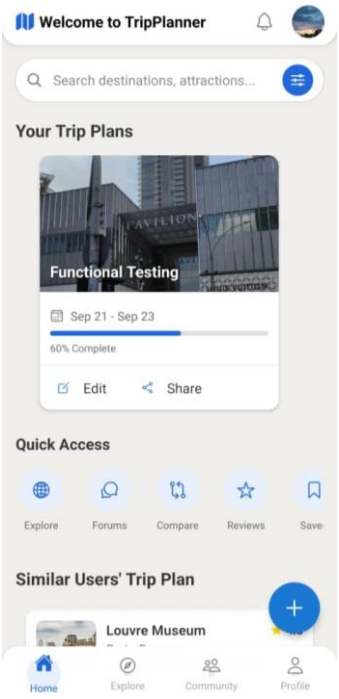
User profile image successfully updated.

Next, we start testing the main features, trip card module. First, we create a trip plan, with all necessary data input and save it as draft.

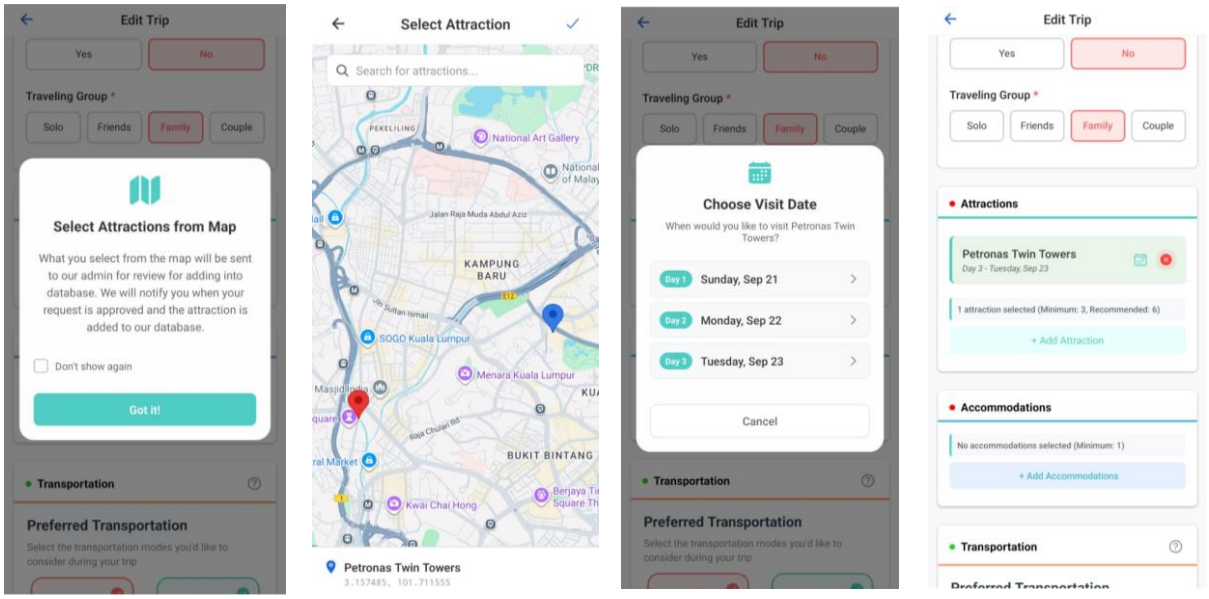




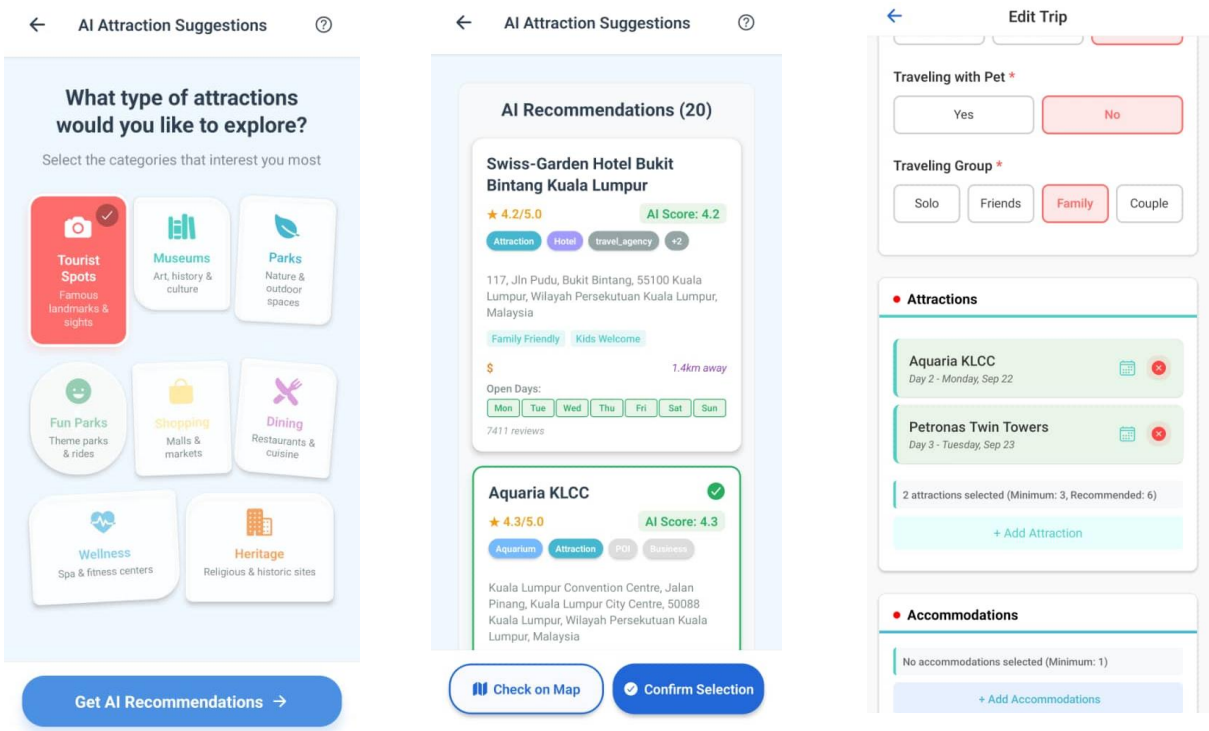
Trip draft successfully saved and showing at home screen.



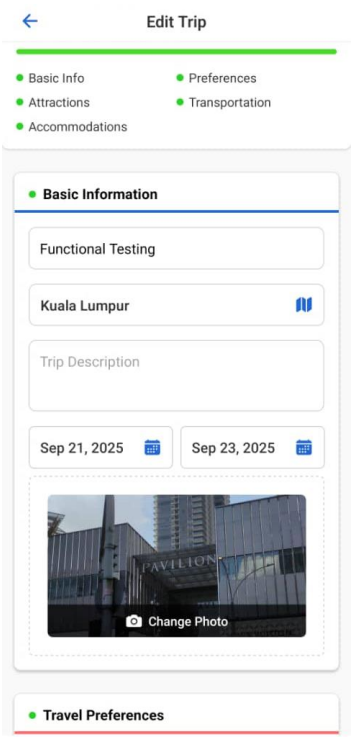
Continue editing the trip plan draft by selecting attractions manually by map provided. Which red marker is the coordinate of destination and blue marker as the attractions spot currently select.

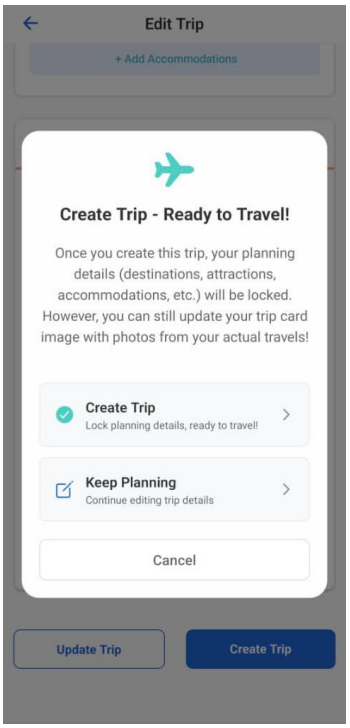


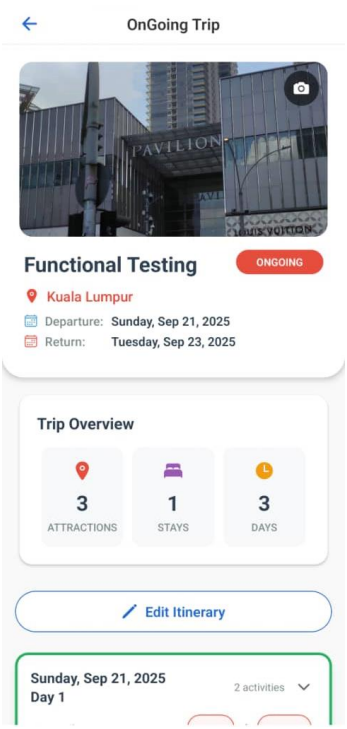
Next, we try select attarction from AI recommendations. Firstly, we choose which attractions category we would like to visit. Then we can choose one from the 20 recommended attractions. If the there is no recommendation output, which means there is not attarctions around the destination in system database, system will recommend user to choose attarctions manually from map first.



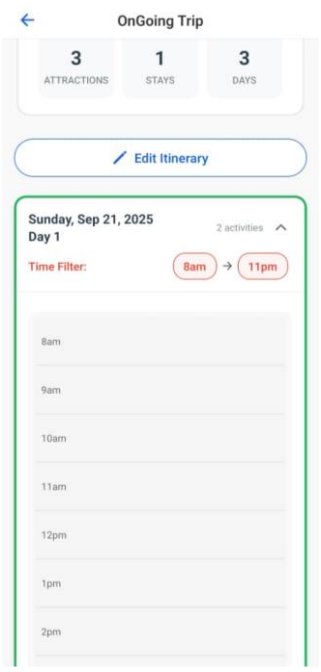
After comlete the trip draft and make the progress to 100%, press the create trip button. Trip draft is now turn to ongoing status. User is now ready to travel.

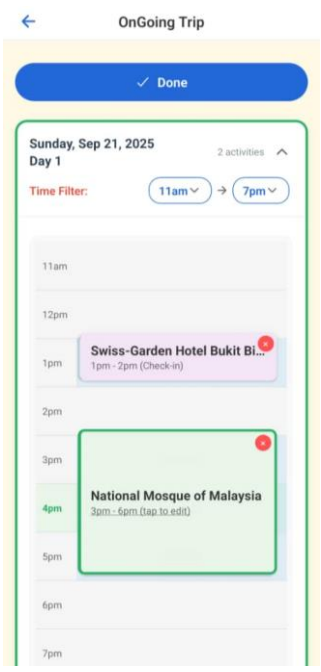




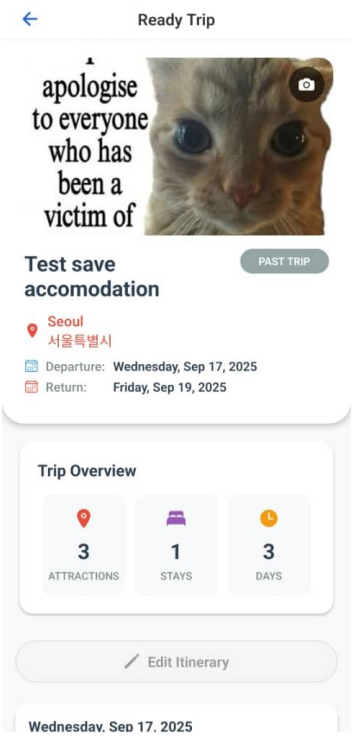


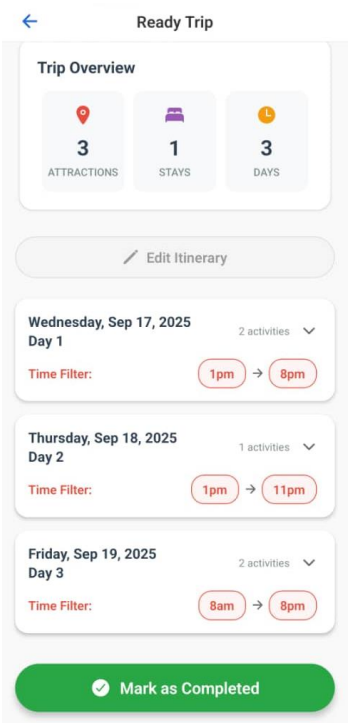
Then user can edit and arrange his itinerary by assigning activity into specific time slot of the day.

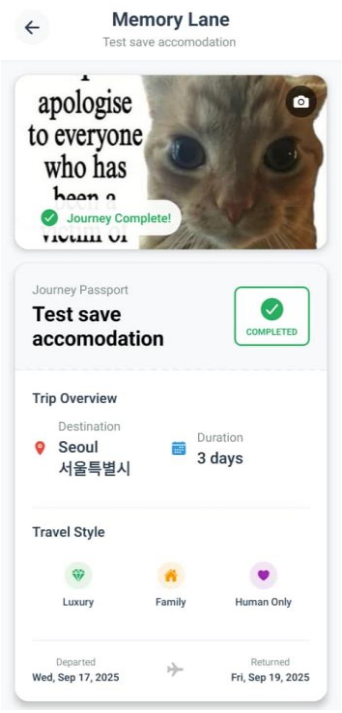




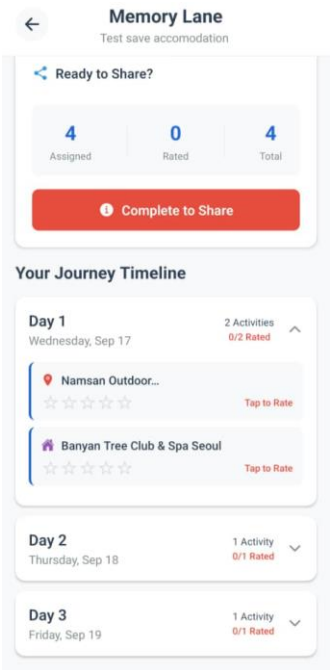
Next we try mark the ongoing trip to completed, since a trip card can only be mark as completed when the traveling date is past date. So we try this feature by using another user account. The testing trip card is end at 19 September 2025, so user is able to mark it as completed.

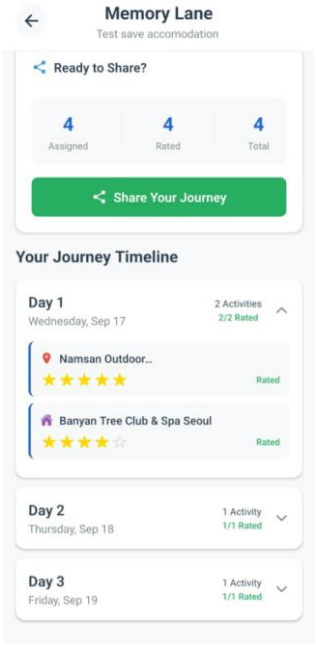


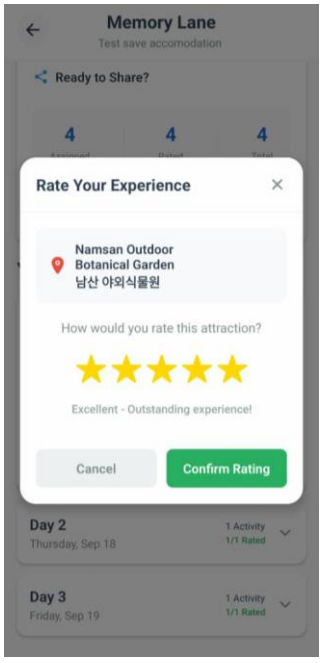




User can rate all the attractions and accommodations he visited in his journey after the trip is completed

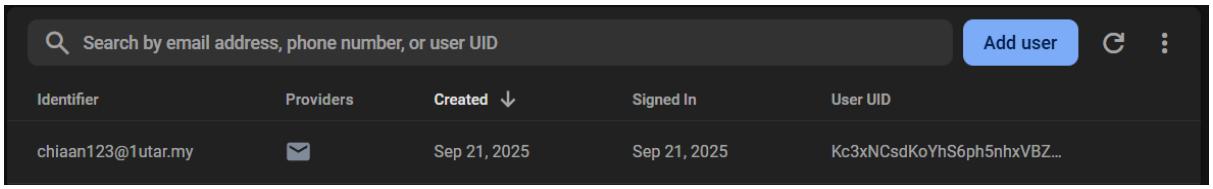






6.2.3 Integration Testing

Looking back to the registration testing, user email is successfully saved into Firebase Authentication:



User profile image successfully uploaded to Cloudinary storage in a URL form:

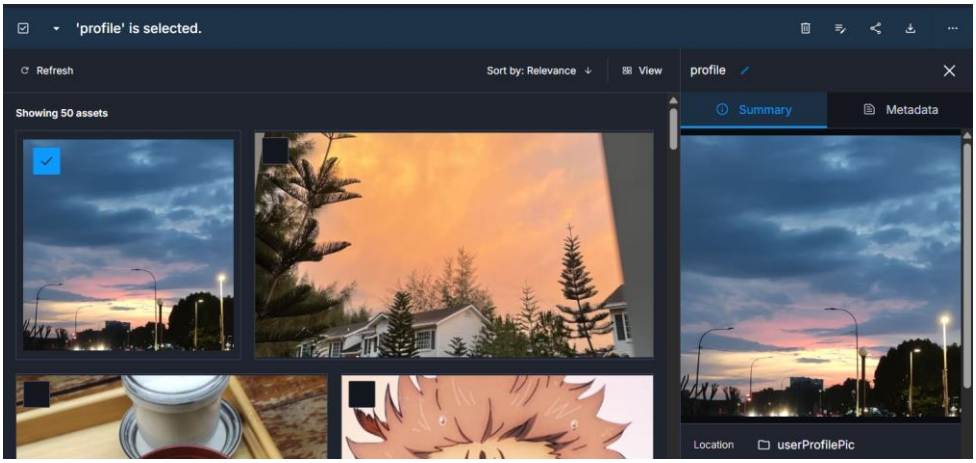
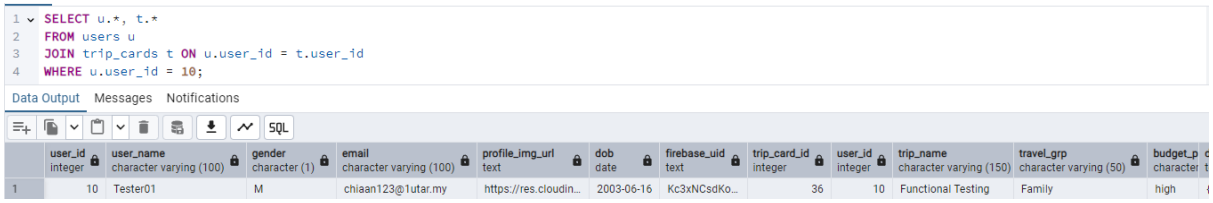


Figure below shows user credentials and trip card information added successfully saved into database securely.



```
SELECT u.*, t.*
FROM users u
JOIN trip_cards t ON u.user_id = t.user_id
WHERE u.user_id = 10;
```

Data Output | Messages | Notifications

	user_id integer	user_name character varying (100)	gender character (1)	email character varying (100)	profile_img_url text	dob date	firebase_uid text	trip_card_id integer	user_id integer	trip_name character varying (150)	travel_grp character varying (50)	budget_p character
1	10	Tester01	M	chiaan123@1utar.my	https://res.cloudin...	2003-06-16	Kc3xNCsdKo...	36	10	Functional Testing	Family	high

Attractions information successfully saved into database securely. With attraction_id attribute is the attraction selected by AI recommendations and pending_attraction_id is the attraction selected manually by map.

```
select * from trip_card_attractions where trip_card_id = 36
```

Data Output | Messages | Notifications

	trip_card_id integer	attraction_id integer	rating_given double precision	rating_date timestamp without time zone	pending_attraction_id bigint	visit_day integer	check_in_date date	check_out_date date	is_accommodation boolean	duration text
1	36	266	[null]	[null]	[null]	2	[null]	[null]	false	[null]
2	36	[null]	[null]	[null]	75	3	[null]	[null]	false	[null]

Attractions visit duration and accommodation check in time assign in specific timeslot successfully saved into database. 1518 means that visit from 3pm until 6pm as assigned. And 1314 is the check in time of the accommodation. There is a , and empty behind is because user have not assigned check out time yet.

```
select * from trip_card_attractions where trip_card_id = 36
```

Data Output | Messages | Notifications

	trip_card_id integer	attraction_id integer	rating_given double precision	rating_date timestamp without time zone	pending_attraction_id bigint	visit_day integer	check_in_date date	check_out_date date	is_accommodation boolean	duration text
1	36	266	[null]	[null]	[null]	2	[null]	[null]	false	[null]
2	36	[null]	[null]	[null]	75	3	[null]	[null]	false	[null]
3	36	269	[null]	[null]	[null]	1	[null]	[null]	false	1518
4	36	265	[null]	[null]	[null]	[null]	2025-09-21	2025-09-23	true	1314,

User rating on attractions and accommodation successfully saved into database.

```
select * from trip_card_attractions where trip_card_id = 34
```

Data Output | Messages | Notifications

	trip_card_id integer	attraction_id integer	rating_given double precision	rating_date timestamp without time zone	pending_attraction_id bigint	visit_day integer	check_in_date date	check_out_date date	is_accommodation boolean	duration text
1	34	[null]	4	2025-09-21 16:59:00	63	[null]	2025-09-17	2025-09-19	true	1415,1213
2	34	[null]	2	2025-09-21 16:59:00	30	2	[null]	[null]	false	2123
3	34	[null]	5	2025-09-21 16:59:00	31	3	[null]	[null]	false	1618
4	34	[null]	5	2025-09-21 16:59:00	29	1	[null]	[null]	false	1720

6.2.4 Performance Testing

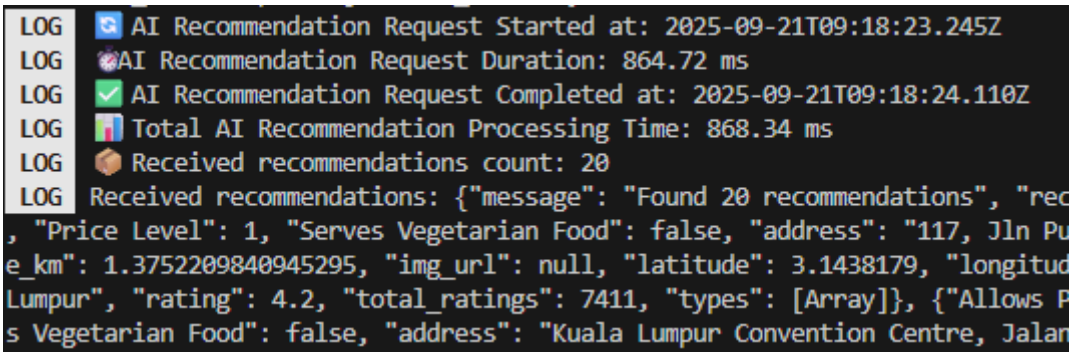


Figure 6.2.4.1 AI Recommendations Performance Testing Result

The AI recommendation request duration was measured at **864.72 ms**, while the recommendation processing time was **868.34 ms**. The closeness of these two values indicates that nearly the entire duration of handling a recommendation request is dominated by the recommendation engine's processing itself. In other words, the system spends most of its time running the model to generate predictions, while the overhead from other tasks such as request handling, communication, or data preparation is negligible. This demonstrates that the system is highly efficient in managing peripheral processes, with model execution being the primary factor influencing total response time.

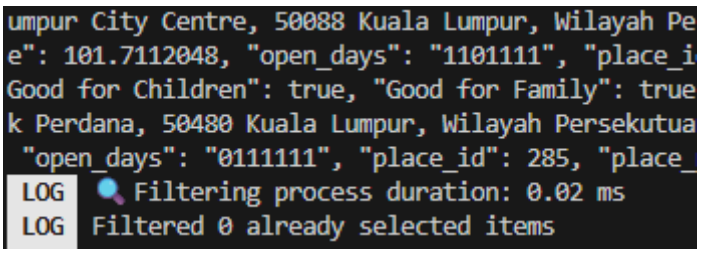


Figure 6.2.4.2 Filtering Process Performance Testing Result

Filtering is the step applied after generating raw recommendations, where the system applies user-specific constraints (e.g., budget, group size, dietary restrictions, pet-friendliness). The extremely low duration indicates that filtering is very lightweight compared to the model execution. This suggests that the system efficiently narrows down results without adding significant latency.

6.3 Result Interpretation

The comparison of the performance of the recommendation models highlights clear differences in predictive accuracy and usability in a mobile tourism recommendation system. Among the three collaborative filtering techniques tried out, Funk SVD was the best performing with the lowest RMSE of 0.8356 that was most consistent in predicting user ratings for tourist attractions. Its strength lies in its capacity to successfully model latent user–item relationships, incorporate user and item biases, and employ regularization to fight overfitting. With the added benefit of Numba acceleration, Funk SVD also computed faster, which is necessary to provide responsiveness in real-time mobile applications. In the meantime, Co-Clustering with RMSE 0.9408 showed modest performance with greater interpretability of user group activities at the cost of accuracy. KNN with Mean, while easy to implement and simple, performed the worst and was afflicted with data sparsity, which was less suitable for the complex and dynamic nature of travel recommendations. Among these findings, Funk SVD stands out as the best model to implement, especially when combined with content-based filtering to address cold-start problems and further improve recommendation quality.

In terms of system testing, the implemented features performed as expected. Functional testing confirmed that core modules, including user authentication, trip card management, and rating functionalities, operated smoothly without critical issues. Integration testing verified that external services, such as Cloudinary for image uploads, Firebase for user management, and PostgreSQL for database operations, were successfully interconnected, ensuring reliable data flow across components. Furthermore, performance testing showed promising results, with an overall recommendation request duration of 864.72 ms, a recommendation processing time of 868.34 ms, and a filtering duration of only 0.02 ms. These values indicate that the system can generate recommendations in under one second, with most of the processing time spent on executing the recommendation model, while filtering and overhead processes contributed negligibly to latency.

Overall, the evaluation results confirm that the system not only leverages the most effective recommendation model (Funk SVD) for accuracy and efficiency but also delivers stable functionality, seamless integration with external services, and real-time responsiveness suitable for enhancing user experience in mobile tourism applications.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

This project effectively designed and deployed a tourism recommendation framework that uses machine learning methods to solve fundamental issues in the tourism industry, more specifically data sparsity and scalability. Three known recommendation models—Funk SVD, Co-Clustering, and KNN with Mean—were trained and evaluated against each other, with Funk SVD being the most precise model, recording the lowest RMSE of 0.8356. Through matrix factorization and bias removal, the system proved to have high prediction abilities, hence the best for integration into the mobile app.

Apart from the recommendation engine, the extremely mobile-responsive app was developed with React Native along with a FastAPI backend and PostgreSQL database. The app has prominent travel-related functionalities such as personalized destination recommendations, user profile handling, and itinerarization of trip plans. Facilitative APIs like Firebase Authentication for authentication, Cloudinary for media storage, and Google Maps API for location services were used to enable the optimal user experience and allow for seamless communication between frontend and backend.

The initial objective of comparing and evaluating three models of recommendations was achieved to the maximum. Funk SVD's performance, particularly in dealing with sparse data sets and providing sound predictions, confirmed why it was utilized. This achievement validated the method employed in the system and ensured the most accurate and scalable model was included.

The second objective, the design of an accessible and user-focused mobile application, was also achieved. The application was effectively developed with core functions such as the handling of user profiles and destination discovery. Although some of the more sophisticated features such as real-time notifications and social sharing are yet to be developed, the application shows promise for the consolidation of several different travel-related tasks into one high-mobility platform, thereby improving the tourism experience for final consumers.

The third objective, deploying the optimized Funk SVD model within the mobile app, was successfully achieved but requires optimization. The recommendation engine was deployed with FastAPI and integrated into the frontend, which has support for personalized recommendations. There are additional optimizations required in API endpoints to improve performance, stability, and scalability, which will be addressed in future development to facilitate stronger real-time capability.

Lastly, this project made significant contributions to the creation of an intelligent tourism assistant by combining accurate recommendation modeling with actual-world mobile application programming. Not only does it establish a framework for mitigating cold-start and sparsity problems but also enhances the travel planning process as a whole for users. With more optimization and enhancement of features, this system possesses the potential to evolve into an end-to-end solution for tourists, tour guides, and tourism companies.

7.2 Recommendations

While this project successfully achieved its main objectives, there are several ways the system can be refined and developed further in the future.

Firstly, the algorithm for recommendations can also be enhanced by trying advanced methods such as deep learning-based recommenders (e.g., Neural Collaborative Filtering or Transformers). They have been shown to handle more complex user–item interactions and would therefore improve recommendation accuracy better than the current SVD-based approach. Additionally, incorporating context-aware factors such as season, current weather, and cultural events could make recommendations more dynamic and personalized.

Second, the mobile application can be enhanced further with additional features that enhance user engagement. For example, adding social and community features would allow users to post itineraries, remarks, and experience on the app, thereby increasing system adoption and usefulness. Similarly, integrating real-time alerts for trip updates, transport changes, or proximity-based alerts for attractions would enhance the interactivity and usefulness of the app for transit travelers in motion.

Third, scalability and performance enhancement is recommended to be implemented in the future. While PostgreSQL and FastAPI already provide a solid backend solution, further optimization such as caching mechanisms, cloud deployment, and load balancing can make the

system handle even more datasets and concurrent users. Scaling up the dataset size from the original 5,000 records to extend to larger geographic areas will also make the system relevant and versatile for real-world applications.

In summary, the test outcomes affirm that the system not only employs the most efficient recommendation model (Funk SVD) for performance and precision but also ensures stable performance, seamless external service integration, and real-time responsiveness suitable for enhancing user experience in mobile tourism apps.

REFERENCES

[1] M. E. B. H. Kbaier, H. Masri, and S. Krichen, "A Personalized Hybrid Tourism Recommender System," IEEE Xplore, Oct. 01, 2017. https://ieeexplore.ieee.org/document/8308293 (accessed May 16, 2022).

[2] Bilal Hawashin, Shadi AlZu'bi, Ala Mughaid, Farshad Fotouhi, and A. Abusukhon, "An Efficient Cold Start Solution for Recommender Systems Based on Machine Learning and User Interests," Apr. 2020, doi: https://doi.org/10.1109/sds49854.2020.9143953.

[3]"Machine Learning Solutions for Cold Start Problem in Recommender Systems," Express Analytics, Jul. 30, 2021.

[4]B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," Expert Systems with Applications, vol. 41, no. 4, pp. 2065–2073, Mar. 2014, doi: https://doi.org/10.1016/j.eswa.2013.09.005.

[5] B. Ramzan et al., "An Intelligent Data Analysis for Recommendation Systems Using Machine Learning," Scientific Programming, vol. 2019, pp. 1–20, Oct. 2019, doi: https://doi.org/10.1155/2019/5941096.

[6] Q. Shambour and J. Lu, "A Framework of Hybrid Recommendation System for Government-to-Business Personalized E-Services," 2010 Seventh International Conference on Information Technology: New Generations, 2010, doi: https://doi.org/10.1109/itng.2010.114.

[7] C. C. Aggarwal, Data mining : the textbook. Cham: Springer, 2015.

[8] F. Ricci and E. Al, Recommender systems handbook. New York, N.Y.: Springer, 2011.

[9] C. C. Aggarwal and Springer International Publishing Ag, Recommender Systems The Textbook. Cham Springer International Publishing Springer, 2018.

[10] M. Badouch and M. Boutaounte, "Personalized Travel Recommendation Systems: A Study of Machine Learning Approaches in Tourism," April-May 2023, no. 33, pp. 35–45, Apr. 2023, doi: https://doi.org/10.55529/jaimlnn.33.35.45.

[11] Reham Alabduljabbar, "Matrix Factorization Collaborative-Based Recommender System for Riyadh Restaurants: Leveraging Machine Learning to Enhance Consumer Choice," Applied sciences, vol. 13, no. 17, pp. 9574–9574, Aug. 2023, doi: https://doi.org/10.3390/app13179574.

[12]Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," Computer, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[13] Z. Xiang, Q. Du, Y. Ma, and W. Fan, "A comparative analysis of major online review platforms: Implications for social media analytics in hospitality and tourism," *Tourism Management*, vol. 58, no. 1, pp. 51–65, Feb. 2017, doi: https://doi.org/10.1016/j.tourman.2016.10.001.

[14] Y. Xin, “Challenges in Recommender Systems: Scalability, Privacy, and Structured Recommendations,” 2015. Available: https://dspace.mit.edu/bitstream/handle/1721.1/99785/927438195-MIT.pdf?sequence=1

[15] T. S. Kumari and K. Sagar, “A Semantic Approach to Solve Scalability, Data Sparsity and Cold-Start Problems in Movie Recommendation Systems,” International Journal of Intelligent Systems and Applications in Engineering, vol. 11, no. 6s, pp. 825–837, May 2023, Accessed: Jul. 27, 2024. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/2917

[16] M. Nilashi, K. Bagherifard, M. Rahmani, and V. Rafe, “A recommender system for tourism industry using cluster ensemble and prediction machine learning techniques,” Computers & Industrial Engineering, vol. 109, pp. 357–368, Jul. 2017, doi: https://doi.org/10.1016/j.cie.2017.05.016.

[17] A. Kulkarni, R. M., P. Barve, and A. Phade, “A Machine Learning Approach to Building a Tourism Recommendation System using Sentiment Analysis,” International Journal of Computer Applications, vol. 178, no. 19, pp. 48–51, Jun. 2019, doi: https://doi.org/10.5120/ijca2019919031.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems.” Available: https://files.grouplens.org/papers/sarwar_SVD.pdf

[19] “Movie Recommendation System using Cosine Similarity and KNN,” International Journal of Engineering and Advanced Technology, vol. 9, no. 5, pp. 556–559, Jun. 2020, doi: https://doi.org/10.35940/ijeat.e9666.069520.

[20] U. Gretzel and K. H. Yoo, "Use and impact of online travel reviews," in *Information and Communication Technologies in Tourism 2008*, P. O’Connor, W. Höpken, and U. Gretzel, Eds. Springer, 2008, pp. 35–46.

[21] H. Lv and H. Tang, “Machine Learning Methods and Their Application Research,” IEEE Xplore, Oct. 01, 2011. https://ieeexplore.ieee.org/document/6103548?msclkid=2630bbb4afca11ec9429120d7864a627 (accessed Mar. 30, 2022).

[22] I. Portugal, P. Alencar, and D. Cowan, “The use of machine learning algorithms in recommender systems: A systematic review,” Expert Systems with Applications, vol. 97, pp. 205–227, May 2018, doi: https://doi.org/10.1016/j.eswa.2017.12.020.

[23] D. Buhalis and R. Law, “Progress in Information Technology and Tourism management: 20 Years on and 10 Years after the Internet—The State of eTourism Research,” *Tourism Management*, vol. 29, no. 4, pp. 609–623, Aug. 2008, Available: https://www.sciencedirect.com/science/article/pii/S0261517708000162

[24] Arfan Jadulhaq and Z. K. A. Baizal, “Games Recommender System Using Singular Value Decomposition,” Dec. 2023, doi: https://doi.org/10.1109/icicyta60173.2023.10428876.

[25] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of Dimensionality Reduction in Recommender System --A Case Study." Available: https://files.grouplens.org/papers/webKDD00.pdf

[26] A. L. Vizine Pereira and E. R. Hruschka, "Simultaneous co-clustering and learning to address the cold start problem in recommender systems," *Knowledge-Based Systems*, vol. 82, pp. 11–19, Jul. 2015, doi: https://doi.org/10.1016/j.knosys.2015.02.016.

[27] E. Battaglia, F. Peiretti, and R. G. Pensa, "Co-clustering: a Survey of the Main Methods, Recent Trends and Open Problems," *ACM Computing Surveys*, Oct. 2024, doi: https://doi.org/10.1145/3698875.

[28] Kah Phooi Seng, L. Ang, Alan Wee-Chung Liew, and J. Gao, *Multimodal Analytics for Next-Generation Big Data Technologies and Applications*. Springer, 2019.

[29] H. Liu, Z. Hu, A. Mian, H. Tian, and X. Zhu, "A new user similarity model to improve the accuracy of collaborative filtering," Knowledge-Based Systems, vol. 56, pp. 156–166, Jan. 2014, doi: https://doi.org/10.1016/j.knosys.2013.11.006.

[30] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," Knowledge-Based Systems, vol. 46, pp. 109–132, Jul. 2013, doi: https://doi.org/10.1016/j.knosys.2013.03.012.

[31] "Tripadvisor: Read Reviews, Compare Prices & Book," Tripadvisor. https://www.tripadvisor.com.my/

[32] "Google," Google.com, 2021. https://www.google.com/travel/?dest_src=ut&tcfs=UgRgAXgB&ved=2ahUKEwjtrqmIjauIAxWtn2YCHelKKlkQyJABegQIABAE&ictx=2&hl=en&gl=MY (accessed Sep. 05, 2024).

[33] "Booking.com: The largest selection of hotels, homes, and holiday rentals," Booking.com, 2024. https://www.booking.com/index.html?label=gen173nr-

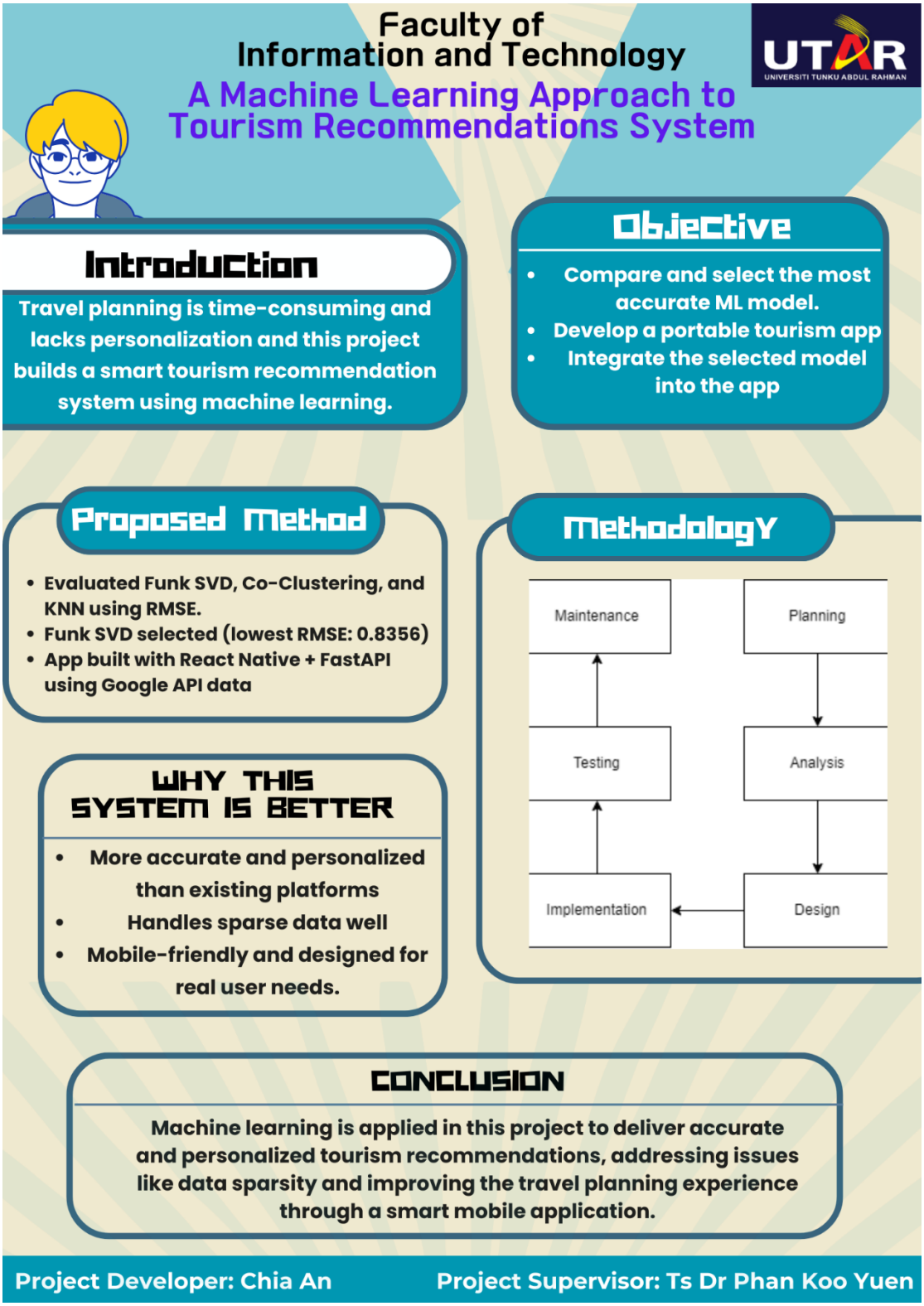
[34] "Recommender Systems using KNN," GeeksforGeeks, Jun. 11, 2024. https://www.geeksforgeeks.org/recommender-systems-using-knn/

[35] ahmadibraheeem, "Movie Recommendation System Using Kmeans and HC," Kaggle.com, Nov. 08, 2023. https://www.kaggle.com/code/ahmadibraheeem/movie-recommendation-system-using-kmeans-and-hc (accessed Sep. 09, 2024).

[36] Dr. Vaibhav Kumar, "Singular Value Decomposition (SVD) & Its Application In Recommender System," AIM, Mar. 25, 2020. https://analyticsindiamag.com/ai-mysteries/singular-value-decomposition-svd-application-recommender-system/ (accessed Sep. 09,

[37] UNWTO, "International tourism recovers pre-pandemic levels in 2024," *UNWTO.org*, Jan. 20, 2025. https://www.unwto.org/news/international-tourism-recovers-pre-pandemic-levels-in-2024

POSTER


Faculty of
Information and Technology
UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
A Machine Learning Approach to
Tourism Recommendations System
Introduction
Travel planning is time-consuming and lacks personalization and this project builds a smart tourism recommendation system using machine learning.
Objective
Compare and select the most accurate ML model.
Develop a portable tourism app
Integrate the selected model into the app
Proposed Method
Evaluated Funk SVD, Co-Clustering, and KNN using RMSE.
Funk SVD selected (lowest RMSE: 0.8356)
App built with React Native + FastAPI using Google API data
Methodology
Maintenance
Planning
Testing
Analysis
Implementation
Design
WHY THIS SYSTEM IS BETTER
More accurate and personalized than existing platforms
Handles sparse data well
Mobile-friendly and designed for real user needs.
CONCLUSION
Machine learning is applied in this project to deliver accurate and personalized tourism recommendations, addressing issues like data sparsity and improving the travel planning experience through a smart mobile application.
Project Developer: Chia An
Project Supervisor: Ts Dr Phan Koo Yuen