Traveler Assistant Bot for Sri Lankan Tourism

S. Arangajanan

2025

Traveler Assistant Bot for Sri Lankan Tourism

A thesis submitted for the Degree of Master of Information Technology

S. Arangajanan
University of Colombo School of Computing
2025

Declaration

Name of the student: S. Arangajanan
Registration number: 2022/MIT/002
Name of the Degree Programme: Master of Information Technology
Project/Thesis title: Traveler Assistant Bot for Sri Lankan Tourism

1. The project/thesis is my original work and has not been submitted previously for a degree at this or any other University/Institute. To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.
2. I understand what plagiarism is, the various types of plagiarism, how to avoid it, what my resources are, who can help me if I am unsure about a research or plagiarism issue, as well as what the consequences are at University of Colombo School of Computing (UCSC) for plagiarism.
3. I understand that ignorance is not an excuse for plagiarism and that I am responsible for clarifying, asking questions and utilizing all available resources in order to educate myself and prevent myself from plagiarizing.
4. I am also aware of the dangers of using online plagiarism checkers and sites that offer essays for sale. I understand that if I use these resources, I am solely responsible for the consequences of my actions.
5. I assure that any work I submit with my name on it will reflect my own ideas and effort. I will properly cite all material that is not my own.
6. I understand that there is no acceptable excuse for committing plagiarism and that doing so is a violation of the Student Code of Conduct.

Signature of the Student	**Date (DD/MM/YYYY)**
	22/06/2025

Certified by Supervisor(s)
This is to certify that this project/thesis is based on the work of the above-mentioned student under my/our supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard.

	Supervisor 1	**Supervisor 2**	**Supervisor 3**
Name	SPW		
Signature	22nd June 2025		
Date			

Abstract

The Traveler Assistant Bot is a Telegram-integrated service platform designed to streamline service discovery, booking, and payment processing for travelers while providing efficient management tools for service providers and administrators. Traditional service booking methods often involve manual processes, limited accessibility, and security concerns, creating inefficiencies for both users and service providers. This project addresses these challenges by implementing an automated, real-time, and secure service booking system.

The system leverages Spring Boot for backend processing, React.js for the Admin Panel, and Telegram Web Apps for user interaction. Authentication is managed through Telegram OAuth with JWT-based authentication for travelers and JWT-based authentication for administrators. Service providers can register and manage their offerings through a verification-based approval system, ensuring quality control. Payments are securely processed via PayHere, incorporating MD5 hashing for transaction integrity.

Extensive unit, API, integration, and system testing validated the platform's functionality, security, and performance, confirming that the system meets its intended objectives. The results demonstrate an efficient, scalable, and secure solution for service booking and management. Future enhancements include AI-powered service recommendations, multi-payment gateway integration, and native mobile application support to further improve user experience and scalability.

This project successfully bridges the gap between travelers and service providers, enhancing efficiency, transparency, and accessibility in service booking and management.

List of Abbreviations

Abbreviation	Full Form
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
E2EE	End-to-End Encryption
ER	Entity-Relationship
GDPR	General Data Protection Regulation
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IPN	Instant Payment Notification
JPA	Java Persistence API
JWT	JSON Web Token
LTS	Long-Term Support
MITM	Man-in-the-Middle
MVC	Model-View-Controller
RAD	Rapid Application Development
RBAC	Role-Based Access Control
Redis	Remote Dictionary Server
REST	Representational State Transfer
SRS	Software Requirements Specification
SSL/TLS	Secure Sockets Layer / Transport Layer Security
UI/UX	User Interface / User Experience
UML	Unified Modeling Language
XSS	Cross-Site Scripting

Acknowledgement

I sincerely thank my resource person at the University of Colombo School of Computing (UCSC) for their invaluable mentorship, critical feedback, and encouragement throughout this journey. Their expertise and patience in guiding me through complex technical and theoretical challenges were instrumental in shaping this project.

I am deeply grateful to the University of Colombo and UCSC for providing the academic resources, infrastructure, and collaborative environment that enabled the successful completion of this research. Special thanks to the faculty members whose courses and insights laid the foundation for my understanding of software engineering and system design.

I acknowledge the open-source communities behind Spring Boot, Telegram Bot API, MySQL, and Redis for their robust tools and frameworks, which were pivotal in developing the Traveler Assistant Bot. Their contributions to the tech ecosystem empowered this project's implementation.

Table of Contents

List Of Figures

List of Tables

1 Chapter 01: Introduction

1.1 Project Overview

The "Traveler Assistant Bot for Sri Lankan Tourism" project aims to enhance the travel experience by addressing travellers' significant challenges when accessing reliable services in unfamiliar areas. Despite Sri Lanka being one of the top destinations globally (Despina Wilson, 2024) Tourists and travellers often need help efficiently locating and scheduling reputable service providers, such as vehicle rental and repair workers, laundry service providers, local guides, accommodations, etc.

These difficulties are increased by the necessity to install and register with several service applications, which is time-consuming, inconvenient, and frequently demands the exposure of personal information. In addition, tourists may face language challenges, confusion about service quality, and even become victims of fraud. The present ways of finding services, such as web searches or word of mouth, are inefficient and often frustrating.

This project aims to develop a Traveler Assistant Bot for Sri Lankan tourism, leveraging Telegram's platform to connect travellers with local service providers without needing app downloads or registrations. The bot will simplify the service booking process by allowing tourists to limit searches based on specific criteria such as location, service type, and provider ratings. It would also enable direct communication between users and service providers by incorporating capabilities like text chat and photo sharing to improve clarity. The project will also contain an admin interface for confirming service provider identities and monitoring the overall system, guaranteeing a safe and efficient experience for both travellers and providers.

The bottom line of this project is to create a more convenient and reliable way for travellers who come to Sri Lanka to access essential services, thereby improving their overall experience and reducing the anxiety and frustration associated with finding reputable service providers in a foreign country.

1.2 Motivation

Despite Sri Lanka's reputation as a premier travel destination tourists often encounter difficulties in accessing timely and reliable services, while service providers may struggle with inconsistent customer demand. To address these challenges, this project aims to develop a user-friendly platform that simplifies the connection between travellers and service providers. By leveraging a Telegram bot, we seek to create a streamlined, registration-free solution that enables quick and efficient communication. This approach will enhance the overall travel experience, offering convenience and reliability for both travellers and service providers.

1.3 Project Objective(s)

The objective of the Travel Helper Application project is to develop a comprehensive platform that facilitates seamless interaction between travellers, service providers, and administrators in the tourism industry. This application aims to enable travellers to discover, book, and pay for services such as accommodations, vehicle rentals, and tour guides in a user-friendly environment. At the same time, it empowers service providers to manage their services, connect with other providers, and earn commissions through referrals. The application also incorporates administrative functionalities to oversee user management, commission calculations, and multilingual support, ensuring accessibility across different regions.

Application, not a Chatbot: Unlike traditional chatbots that focus on conversation, this is a Telegram-based embedded application that is embedded into Telegram chat. The bot serves as a functional interface, providing users with direct access to services such as travel guides, hotels, local guides, and more. It is designed to perform specific tasks rather than engage in general conversation.

Specific objectives of the Travel Assistant Application include:

1. **Streamlining Booking Processes**: To provide a unified platform for travellers to view, select, and book services based on real-time availability, enhancing user convenience and satisfaction.
2. **Referral and Commission Management**: To support service providers in forming connections with other providers and receiving commissions for successful referrals, fostering a collaborative service ecosystem.
3. **Efficient Payment Processing**: To facilitate secure payment options, including cash and online payments, and automate commission deductions, ensuring accurate financial transactions.
4. **Robust Administrative Controls**: To enable administrators to manage user accounts, approve withdrawal requests, monitor referral integrity, and oversee system security, maintaining a reliable and secure platform.
5. **Multilingual Accessibility**: To provide support for multiple languages, making the application accessible to users from various regions from Sri Lanka and enhancing its usability.

1.4 Structure of the Dissertation

These chapters under this dissertation are relevant to the planned project and are completed at different stages of the project. Every chapter will describe the knowledge needed to understand

the Traveler Assistant Bot system design and implementation milestones.

- Chapter 2 - Background The background chapter explains the specifics of current comparable systems across the world. It also provides information on emerging technologies that are relevant to the planned project.
- Chapter 3 – Design The development approaches utilized to achieve the objectives are described in the design chapter. Object-oriented provides a framework throughout the project, including design diagrams and descriptions of the key client interfaces.
- Chapter 4 - Implementation The implementation chapter outlines the implementation methodologies & tools of the system analysis & design. In this section will discuss the programming language used, database query language & where the database connection is set up. The software & hardware requirements will also be explained.
- Chapter 5 - Testing and evaluation the testing and evaluation chapter describes the system's testing and evaluation methods & procedure. This also explained several test methodologies and how to evaluate the system's functionalities using specific test cases. This testing & evaluation are critical components of the system's long-term errorless functionality. This chapter also includes user feedback on this system.
- Chapter 6 – Conclusion In this last chapter, the conclusion chapter discusses the overall results of the system, which includes project results, a general appraisal of the jobs completed, achievements and proposed future improvements.

2 Chapter 2 – Background

2.1 Introduction

The tourism industry in Sri Lanka has world recognition it is the 5th travel destination(Despina Wilson, 2024) in the tourism realm but it is facing significant challenges despite its recognition as a top travel destination globally. The challenges are blended with economic instability, safety concerns, and inadequate infrastructure, which reduces travellers' travel experiences entertainment and access to reliable services. despite being recognised as one of the top travel destinations globally, travellers often face significant challenges in accessing timely and reliable services and accessing the right service provider to receive their service. The proposed Traveler Assistant Bot aims to address these challenges by providing an ideal platform for travellers to connect with local service providers without the need for app installations or registrations. Thus, Telegram is one of the instant messaging applications that offers various advantages in its features over other instant messaging applications. The most popular feature that is being developed on Telegram is the bot feature, where a third party or user can develop bot features according to user requirements(Rianto et al., 2019). Some old studies showed the positive skewness in the mobile e-commerce business. Further, (Van Eeuwen, 2017) conclude that in 2017 attitudes may change if Messenger chatbots are used for different purposes such as customer service. Despite the neutral overall findings, the slightly left-leaning results suggest a slight positive preference for using Messenger chatbots, although further research is necessary as the technology continues to evolve. (Mero (Järvinen), 2018) this study shows that chat services are becoming an essential aspect of online shopping experiences, and they should be well-managed to maximize the benefits of their business.

2.2 Requirement analysis

The Traveler Assistant Bot aims to enhance the experience of travellers in Sri Lanka by facilitating connections with service providers through a registration-free Telegram-based application. This analysis outlines the requirements based on the project objectives and scope.

2.2.1 Software Requirements Specification (SRS)

2.2.1.1 Purpose

The purpose of this SRS(IEEE Std 830-1998, 1998) describes the functional and non-functional requirements for the "Traveler Assistant Bot for Sri Lankan Tourism." The system is a Telegram bot application designed to help travellers effectively communicate with local service providers (such as hotels, travel guides, and more) without having to download an app. Service

providers must undertake identification verification and may pay a membership fee to use the system.

2.2.1.2 Scope

Travel assistant Bot is a software Product which is A telegram bot application embedded within the Telegram platform, allowing users to search for and book services from verified service providers. The web application will be embedded with a chatbot capable of enabling users to interact with service providers according to users' preferences', manage bookings, provide feedback, make payments etc. Thus, an admin Pannel will be given to administration, a kind of extranet, where the system administrator would have exclusive access for adjusting users' access, and registrations of the service providers, to control the status and performance of the system and even for financial transactions when the system requires to do so.

Users will then utilize the embedded web application in the Telegram chat bot's web view to search for the most appropriate service providers. Concerned service providers will be able to register themselves and their services to the platform being developed. It will provide information such as; service specializations, payment procedures, and customer response options. There will be the creation of an admin panel for the administrative activity; one will be managing the activities of the system; another will be managing the registration of users and the third one will be supervising the system's performance. This brings efficiency in service discovery, provider registration, and transaction monitoring within the system.

The suggested use of this software product will be within the sphere of tourism services, with a focus on the consumers in search of reliable service providers. It will help in the identification of the services, secure handling of transactions, and give the users feedback takers. The website is easy to navigate, which means that the user is not going to waste a lot of time looking for a certain item; the site also has numerous languages and has integrated a secure payment system. It includes the following advantages: The system allows travellers to directly interact with the verified service providers. It has extended communication in different languages thus easing the process of making a booking and payment. Its primary goal is to develop a platform that enables establishing a secure connection between customers, travellers, and trustworthy service providers while guaranteeing user-friendliness and means of expansion. The overall objective is to create a complex of applications with the main Telegram bot, a web application built in the layout of the Telegram web version, and an administrator's panel. This system will enable the services to be retrieved, providers to apply for registration, payments to be made as well as feedback to be collected, there will also be an administrative control to police the system and manage users.

2.3 Functional and Non-Functional Requirements

2.3.1 Functional Requirement

Functional requirements define what the system should do. its specific behaviours or functions of said application.

User Registration and Login

- **Admin Registration**: The system will allow an Admin to register and manage accounts.
- **User Registration**: The system will allow Service Providers and Travelers to register and create accounts.
- **Login Authentication**: The system will authenticate users based on their credentials (username/password/ telegram ID) and provide access accordingly.

User Management

- **Admin User Management**: Admin users will all be able to create, edit, and delete other admin accounts.
- **Service Provider Management**: Admin users will be able to manage Service Providers' profiles, their services, and their connection recommendations.
- **User (Traveller) Management**: Admin users will be able to manage users.
- **Flag Incorrect Information**: The system will allow Admin to flag or correct wrong information provided by users or service providers.

Multilingual Support

- **Language Selection**: Users will be able to choose the system's language from a predefined set of languages.
- **Multilingual Key Management**: Admin will manage localization keys for all languages supported by the system.

Service Provider Connections

- **Service Provider Recommendations**: Service Providers will be able to suggest other providers to users (e.g., a tour guide can recommend a vehicle rental service).
- **Commission for Referrals**: When a traveller hires a service provider referred by another provider, the referring provider will receive a commission automatically.

Payment and Commission Handling

- **Payment Methods**: The system will handle both online and cash payments.
- **Cash Payment Handling**: If a traveller pays by cash, the system will deduct the balance from the traveller’s account, and the system will collect the commission.

- **Payment Review**: The admin will be able to review withdrawal requests from service providers.
- **Withdrawal Approval**: The admin shall approve withdrawals based on the available balance in the service provider's account.
- **Commission Collection**: When users pay via cash, the system will deduct the commission from their account automatically.

Booking System

- **Booking Services**: Travelers will be able to view, select, and book services (e.g., hotels, vehicle rentals, guides, etc) from available Service Providers.
- **Booking Log**: The system will store the booking history of each user and Service Provider.

Fraud Detection and Monitoring

- **Fraud Detection**: The system will check for fraudulent connections or behaviour among service providers and flag suspicious activities.
- **Manual Fraud Review**: Admin users will review flagged connections and decide whether to allow or block them.

Notifications

- **Provider Notifications**: Service Providers will be notified of booking requests, payment status, and withdrawal approvals.
- **Fraud Alerts**: Admin users will be notified if fraudulent behavior is detected.

2.3.2 Non-Functional Requirements

Non-functional requirements define the quality attributes and operational constraints of a system, focusing on how it performs rather than what it does. The Traveler Assistant Bot adheres to the ISO/IEC 9126 software quality model. It defines key quality attributes: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. These attributes were systematically incorporated into the development of the Traveler Assistant Bot to ensure the system meets high security, scalability, and performance standards while allowing future enhancements. The following explains how each attribute was applied:

Functionality

The Traveler Assistant Bot ensures functionality through:

- **Authentication and Access Control:** The system implements Telegram OAuth authentication to securely verify travelers and service providers. Additionally, role-based access control (RBAC) ensures that travelers, service providers, and administrators have access to appropriate functionalities based on their roles.
- **Secure Transactions:** Payment processing is integrated with PayHere, ensuring secure transactions using **MD5** hash verification to prevent unauthorized modifications.
- **Data Integrity and Security:** Service providers can only manage their own bookings, ensuring data isolation and preventing unauthorized modifications by other users.

These features collectively enhance the platform's integrity, security, and compliance with functional requirements.

Reliability

Several mechanisms have been implemented to ensure system stability:

- **Stable Backend Architecture**: The backend is built using Spring Boot, while MySQL with Redis caching enhances database efficiency and reduces system crashes due to high loads.
- **Automated Error Handling**: The system includes error logging and monitoring to detect and resolve unexpected failures efficiently.
- **Resilient Payment Processing**: The system employs retry mechanisms for failed API requests, particularly for payment verification and booking confirmations.
- **Data Redundancy and Backups**: A database backup strategy ensures that critical user data, including bookings and transaction records, is not lost in case of system failures.

By incorporating these measures, the system ensures consistent performance and minimal downtime, enhancing user trust.

Usability

Usability ensures that the system offers an intuitive and user-friendly experience for travelers, service providers, and administrators. The following aspects contribute to the system's usability:

- **Intuitive User Interface**: The Telegram Mini App provides a simplified service discovery and booking experience for travelers, while React.js-based Admin Panel allows administrators to efficiently manage platform operations.
- **Simplified Booking Flow**: The service booking process is designed with minimal steps, allowing users to complete bookings quickly and effortlessly.
- **Responsive Design:** The system is designed to work across multiple devices, ensuring accessibility for users on smartphones, tablets, and desktops.

These usability enhancements contribute to high user adoption rates and ease of system navigation.

Efficiency

- **Optimized Data Handling**: The use of Redis caching reduces the number of direct database queries, leading to faster response times.
- **API Optimization**: The system ensures efficient API calls with pagination, preventing excessive data loads in response processing.
- **Asynchronous Processing**: Booking confirmations, payment verifications, and other background tasks are handled asynchronously, improving system responsiveness.
- **Performance Optimization**: The system applies minimal Data base calls to ensure the speed.

These optimizations reduce system lag, improve scalability, and enhance overall user experience.

Maintainability

The following design principles contribute to system maintainability:

- **Modular Architecture**: The system is designed using Spring Boot Layered archtecture and React.js components, making it easier to update specific features without affecting the entire system.
- **Version Control**: The use of GitHub for source code management allows structured code tracking, facilitating efficient bug fixes and feature enhancements.
- **Centralized Configuration Management**: Configuration parameters such as payment gateway settings and user permissions are centrally managed, allowing quick modifications without altering system code.

- **Comprehensive Documentation**: The system includes well-structured documentation covering APIs, database schemas, and system workflows, ensuring maintainability for future development teams.

These strategies contribute to efficient troubleshooting, adaptability, and ease of implementing future enhancements.

Portability

- **Cloud Deployment Readiness:** The system is designed for deployment on cloud platforms such as AWS and DigitalOcean, ensuring flexibility in hosting options.
- **Containerization:** The use of Docker containers allows the system to run in different environments without dependency issues.
- **Cross-Browser and Multi-Device Support:** The Telegram Mini App and Admin Panel are tested to function correctly on Chrome, Firefox, Edge, and Safari, ensuring cross-browser compatibility.
- **Future Scalability:** The system architecture supports future integration with **native mobile applications**, allowing expansion beyond Telegram-based interactions.

By implementing these portability features, the system ensures smooth deployment, cross-platform compatibility, and long-term adaptability.

The Traveler Assistant Bot has been designed to meet key non-functional requirements, ensuring functionality, security, efficiency, usability, maintainability, and portability. By aligning with the ISO 9126 software quality model, the system achieves high performance, reliability, and adaptability while maintaining robust security mechanisms. These quality attributes ensure that the system remains scalable, user-friendly, and future-proof, providing an efficient and secure service booking experience for travelers, service providers, and administrators.

2.4 Review of Similar System

The Traveler Assistant Bot was developed to address the specific needs of travelers seeking reliable services in unfamiliar locations. Several service provider applications exist in both local and international markets, offering users a platform to connect with service providers across various industries. However, these existing platforms do not fully cater to the real-time service needs of travelers. This section analyzes similar applications, highlighting their strengths, limitations, and the unique value proposition of the Traveler Assistant Bot.

Several service provider applications exist in the local market, such as onawadak (OnaWadak, n.d.) , ikman(ikman, n.d.) and quickhelp (Quickhelp, n.d.) these are some examples aligned with the project, offering users a way to connect with service providers across various industries. while international platforms such as TaskRabbit(Taskrabbit, n.d.), Thumbtack(Thumbtack, n.d.), and Fiverr (Fiverr, n.d.) provide similar services on a global scale. These platforms share similarities with the proposed Traveler Assistant Bot, particularly in their goal of connecting users with service providers. However, a key distinction lies in their target audience, accessibility, and real-time service capabilities. Local platforms such as OnaWadak (OnaWadak, n.d.) and QuickHelp (Quickhelp, n.d.) cater to a broad audience seeking household, repair, and professional services. While they effectively bridge the gap between service seekers and providers, they are not designed for travelers, and their booking processes often require manual negotiation or separate mobile applications. Similarly, international platforms like TaskRabbit (Taskrabbit, n.d.)and Thumbtack (Thumbtack, n.d.) allow users to hire local professionals, but they primarily focus on home maintenance, freelancing, and gig-based services, rather than addressing the real-time, on-the-go needs of travelers.

Travel service providers like HotelsCombined (hotels combined, n.d.), Expedia (Expedia Travel, n.d.), Booking.com(Booking.com, n.d.), Agoda(Agoda, n.d.), and Hotels.com (Hotels.com, n.d.), the bot offers a fundamentally different approach by integrating directly into Telegram. While these traditional platforms focus primarily on accommodation bookings, often requiring users to navigate websites or install dedicated apps, the Traveler Assistant Bot provides a more streamlined experience, allowing travelers to search, book, and communicate with service providers directly within a familiar messaging platform.

Platforms like HotelsCombined (Booking.com, n.d.) and Booking.com (Booking.com, n.d.) aggregate accommodation options, helping travelers compare prices and book hotels across multiple providers. However, they do not offer on-the-go, localized service recommendations such as transport, tour guides, or emergency services. Similarly, Expedia and Agoda provide comprehensive travel packages, including flights, hotels, and rental cars, but they are designed for pre-trip planning rather than real-time assistance during travel. These platforms require users to complete bookings in advance, whereas the Traveler Assistant Bot allows instant service discovery and booking, making it more suitable for travelers who need immediate access to accommodations, transport, and local experiences.

These local platforms share similarities with the proposed Telegram bot in their goal of connecting users with service providers. However, a key distinction lies in their target audience (see table: 2.3.2-1). OnaWadak (OnaWadak, n.d.) and QuickHelp (Quickhelp, n.d.) cater to a broader user base seeking services across various industries in contrast, the proposed Telegram bot specifically targets travellers. While platforms like OnaWadak (OnaWadak, n.d.) and QuickHelp (Quickhelp, n.d.) demonstrate the effectiveness of connecting consumers and providers, a gap remains in addressing the unique needs of tourists. Hence, the proposed Solution Traveler Assistant Bot proposes a novel solution to refine and streamline service-provider interaction specifically for travellers. The bot leverages the familiar interface of Telegram, eliminating the need for additional app downloads and offering a seamless user experience. This approach capitalizes on Telegram's extensive user base and intuitive messaging features to provide travellers with immediate access to service providers (Nikolay R, 2023).

The Traveler Assistant Bot addresses this gap by providing a dedicated, travel-focused service booking system integrated directly into Telegram. By leveraging Telegram’s extensive user base and messaging interface, the bot enhances accessibility, ease of use, and immediacy, providing travelers with instant access to service providers in a way that existing platforms do not. While platforms like said platforms demonstrate the effectiveness of connecting users with service providers, a significant gap remains in addressing the unique, real-time service needs of tourists. The Traveler Assistant Bot fills this gap by introducing a specialized, chatbot-driven travel service marketplace, ensuring faster service discovery, seamless user interaction, and automated booking capabilities, tailored specifically for travelers on the move.

Table 2.3.2-1: Comparison with Existing systems

Platform	***Target Audience***	***App Download***	***Registration***	***Messaging***
OnaWadak	Broad user base	Required	Required	-
Ikman	Broad user base	Required	Required	-
QuickHelp	Broad user base	Required	Required	-
TaskRabbit	Broad user base	Required	Required	In app
Thumbtack	Broad user base	Required	Required	In app
Fiverr	Broad user base	Required	Required	In app
Telegram Bot (Proposed)	Travellers	Not Required	Not/ Likely Required	Telegram

Advantages of Telegram Bots for Travelers are, Research suggests that Telegram bots are transforming communication and simplifying (Nikolay R, 2023). Unlike the conventional interfaces that are typically automated, they provide features that could revolutionize Internet interaction. Telegram bots have the potential to replace entire websites, streamline user experiences, and offer seamless engagement (Nikolay R, 2023). They can act as effective intermediaries, handling tasks from transaction processing to business operations management, ultimately enhancing customer satisfaction and productivity (Nikolay R, 2023) Notably, bots possess the capability to securely receive payments, enabling businesses to directly monetize services within the messaging platform (Nikolay R, 2023). Additionally, Location-based feature set functionalities exist inherently in the telegram tool (Lonami, n.d.) that adapt to the dynamic journey requirements. This feature enables the users to quickly locate service providers in their area so that issues related to travel can be solved quickly or a local guide hired. Through catering for the unique needs of travellers, the proposed Telegram bot presents a unique mode of service delivery customized for the unique mobility and needs of the target demographic as compared to apps such as OnaWadak (OnaWadak, n.d.) and QuickHelp

(Quickhelp, n.d.). thatch.co (ThatchGPT, n.d.), wanderlog.com(wanderlog.com, n.d.), yathra.lk (Yathra.lk, n.d.), are some similar tourism-related service provider applications locally and internationally, Thatch is travellers search presents them with locales where they can use such services as planning and customizing the trip cascading. There are ready-made trips with organized sightseeing, self-designed mini-vacations and many more for the traveller's convenience. ThatchGPT (ThatchGPT, n.d.) for instance is an artificial intelligence travel planner on the platform that assists clients to generate travel ideas and support them in any way that is required. In addition, guides may be marked for future retrieval without any consideration for cost which enhances the overall voyage.

There's also an application called Wanderlog (wanderlog.com, n.d.) Click or tap here to enter text., Its functionality is aimed at working together with others through joint trip planning and trip organization which is essential while on excursions in groups. One good thing about the platform is that it has map properties which makes it easy for the users to view the routes and destinations of their trips.

Yathra (Nurvembrianti et al., 2022; Yathra.lk, n.d.) operates as an agent for proactively marketing packaged tours, preferably in the Island country Sri Lanka which may embody different interests based on shades of adventurous, cultural or natural. local Booking System: Yathra also has a provision for one-stop solutions for users who make bookings for their travel facilities directly on the site.

Table 2.3.2-2: Comparison with similar applications

Feature	Thatch.co	Wanderlog.com	Yathra.lk	Travel Assistant Bot
Primary Function	Personalized travel planning	Trip itinerary planning and management	Tour booking and package deals	Connect Service Providers and Tourist
Target Audience	Travelers seeking expert guidance	Independent travellers planning their trips	Travelers looking for guided tours and packages	Tourists and Service providers

Services Offered	Personalized travel itineraries, travel expert consultations	Trip planning, collaboration, expense tracking, AI-powered recommendations	Pre-planned tours, package deals, customized itineraries	Provide connection between tourists and related service providers and provide customized services
Platform	Web-based	Mobile app and web-based	Web-based	Telegram

The Traveler Assistant Bot introduces an innovative, chatbot-driven approach that bridges the gap between travelers and service providers. By eliminating unnecessary complexity, such as app installations and extensive registrations, the bot provides instant access to verified service providers. The real-time interaction and service discovery further enhance efficiency, allowing travelers to secure bookings within seconds.

Additionally, the bot's integration with secure payment gateways ensures that transactions are safe, reliable, and compliant with financial regulations. Future enhancements, such as AI-powered service recommendations, multilingual support, and mobile wallet integration, will further improve accessibility and convenience for international travelers.

Existing service provider platforms serve a broad audience but fail to address the unique needs of travelers who require instant, location-based services. While some travel-focused applications assist with itinerary planning, they do not provide real-time service connections. The Traveler Assistant Bot fills this gap by offering an accessible, efficient, and app-free service booking experience via Telegram. Its interface, real-time availability, and secure transactions position it as a novel solution for modern travel service management.

Traditional travel booking platforms focus on long-term planning and structured travel packages, the Traveler Assistant Bot addresses a major gap by providing an immediate, real-time travel assistance system that adapts to travelers' changing needs on the go. Its seamless integration with Telegram, elimination of app dependencies, instant booking capabilities, real-time messaging, and secure transactions set it apart as a next-generation travel assistant, revolutionizing the way tourists access local services.

2.5 Related Technologies

2.5.1 Telegram Bot API

In the case of the Traveler Assistant Bot, the most important means of communication between the travellers and the service providers was identified as the Telegram Bot API(Nikolay R, 2023) because it is easy to integrate, widely used and ideal for real-time messaging for Support options in a web application, sharing location and processing secure payments (Abhinav, 2023). This API is suitable for the project since it enables users' interaction within the Telegram environment, which will enable the application's travellers to search for the available services, book them, and interact with providers. This approach corresponds to the goals of the Telegram Platform, where features such as moving between languages and location-based services will improve the value of the project for travellers in unknown areas. Also, the integration with third-party services, for instance, payment services (Nick Bulaienko, 2024), is quite simple and effective to perform secure payments.

2.5.2 Spring Boot Framework

When it came to choosing the backend framework(Spring, n.d.) for the Traveler Assistant Bot project, the preferred choice was Spring Boot since it is very effective for constructing very reliable web applications. Its capacity to facilitate configuration management and the rich choice of solutions for security and data management makes it a suitable instrument to handle the project back-end activities, such as user identification, monetization and booking. This will be used to create Rest APIs which will allow the admin panel to communicate with the backend to help the system administrators adequately manage tasks. On the other hand, webhooks will be implemented to connect the Telegram bot with the backend to allow for the real-time interactivity of the bot with the backend. This approach allows for efficient separation of concerns: the admin panel will be dependent on REST API and the bot will use webhooks for interactivity and real-time flow between the two interfaces.

2.5.3 JPA (Java Persistence API) with Hibernate

As for the database management for the Traveler Assistant Bot project, JPA together with the most used JPA implementation Hibernate was chosen to manage all the interactions between the Java application and the database. These together avoid complications in the database operations in terms of saving different data, creating user profiles, databases of service providers, bookings and feedback in the form of different types of SQL queries. Hibernate then provides a solution to map the Java objects to the database tables and store data without

bothering an individual to handle SQL all the time thus making it very efficient and less demanding in terms of development. This approach also promotes code maintainability in that the team spends more time working on business logic than trying to work out how the database works.

2.5.4 My SQL

MySQL was selected as the database for the Traveler Assistant Bot project because it offers high reliability, extensibility, and compatibility with relational models. Due to these features, it should be used for storing important data like information about travellers and providers, records of bookings, and payment data. MySQL workload capabilities of structured data make certain that the existing system can process heavy transactions and user interactions and thereby expand its system as the size of the users increases. Further, MySQL is open source and acceptable for a great deal of use; it is confirmed that it enjoys steady and strong support from its community, which, in turn, reassures long-term stability and performance.

2.5.5 React Js

React.js was chosen for the development of the front end for the view of the Telegram interface as well as the admin panel of the Traveler Assistant Bot project as it allows one to create highly responsive and interactive interfaces for the user (Md Suny Shaikh, 2024). React is also important for the Telegram web view for the user experience since it provides a dynamic interface that gives a quick response to the actions of the user where engagement with the bot is important. Through the Admin panel, developed in React, a user can easily control the account, payments, and approval.

2.5.6 Redis

Specifically, Redis was selected as a caching tool for improvement of the performance that in turn decreases the load on a database in the framework of the Traveler Assistant Bot project as well as makes the access to the data most frequently requested by users faster. This helps to always cache data such as session, the number of times people have searched for a particular service provider and other multilingual information so that the bot is always responsive even when there is a lot of traffic. This helps to avoid many delays that would have been as a result of frequently querying the database by storing only important data in the memory for easy access thus increasing the user experience and efficiency of the system during these busy times.

2.5.7 Payment Gateway

For handling booking payments in the Traveler Assistant Bot project, a secure payment gateway will be integrated as suggested in the official documentation (core.telegram.org, n.d.;

Nick Bulaienko, 2024). It ensures reliable and protected transactions between travellers and service providers. The chosen gateway will be able to manage payments, refunds, and cancellations in a secured way.

2.5.8 Location Based services

It helps travellers find services near them, like hotels or guides, based on their location. Users' location will be detected through the bot application and suggestions will be given within the available nearby service if there are not any/ enough services found it will increase the search perimeter and provide more suitable search results for the customer. In the bottom line, it will work like this,

- The bot will detect the user's current location and suggest nearby services, such as hotels or local guides.
- If the initial search does not yield enough, the bot will expand the search perimeter to find additional suitable options.
- Users receive tailored suggestions based on their current location, ensuring they find the most relevant and accessible services.

2.5.9 Webhooks

- In implementing real-time communication for the Travel Assistant Bot, Telegram's webhook mechanism will be utilized to manage user interactions and streamline responses. Webhooks allow for efficient, event-driven updates, where Telegram directly sends any new user interactions to the bot's server endpoint, eliminating the need for continuous polling. This approach enables immediate processing of commands and messages, optimizing server resources and reducing latency. By handling requests in real-time and returning responses directly to users, the webhook setup enhances user experience through responsive communication and ensures the bot operates with minimal resource consumption

2.6 Summary

This chapter contextualizes the challenges in Sri Lanka's tourism sector, including fragmented service access, inconsistent infrastructure, and fraud risks faced by travelers. A detailed **requirement analysis** identifies core functionalities: user registration, service booking, PayHere payment integration, and commission management. **Non-functional requirements** emphasize security, scalability (99.9% uptime), and cross-platform accessibility.

A **comparative review** of existing platforms (e.g., OnaWadak, QuickHelp) highlights gaps in traveler-centric solutions, justifying the novel Telegram-based approach. The proposed bot eliminates app downloads, leverages real-time messaging, and integrates PayHere for localized payment processing. **Key technologies** include Spring Boot (backend), Telegram Bot API (user interaction), MySQL (data management), and Redis (caching). While multilingual support is deferred to future work, the architecture is designed for multiple language integration.

The chapter concludes with a rationale for adopting the **Rapid Application Development (RAD)** model, enabling iterative prototyping and early stakeholder feedback.

3 Chapter 3 – Design

3.1 Introduction

This chapter provides a detailed analysis and design of the Travel Helper Application, focusing on user roles such as travellers, service providers, and administrators. The system uses a layered architecture, separating concerns and modular development. The design addresses key functionalities like service provider connections, payment management, multilingual support, and real-time communication through chat. The system's architecture, database schema, and user interface design are discussed in this chapter to illustrate how these requirements are implemented. This chapter will describe the system's architecture, design strategies, and relevant UML diagrams that illustrate various components of the system, their interactions, and key workflows. It will also include a summary of the current progress and refined project timeline, ensuring that all aspects are aligned with the project's objectives.

3.2 System Study

This section describes the need for a Travel Helper Application, focusing on how travellers, service providers, and administrators currently manage their operations and the limitations they face. It also analyses how the proposed system will address these limitations by streamlining the process.

3.2.1 Existing System Overview

In the current scenario, there is no unified platform that is available to connect travellers, service providers (for example: hotels, tour guides, vehicle rentals, vehicle workshops, etc.), and administrators for a smooth booking and payment process. Travellers often need to visit multiple websites or contact service providers individually, while service providers struggle to manage bookings, payments, and connections with other providers. Additionally, there is no standardized process for service providers to earn commissions when recommending other providers.

Challenges in the existing scenario include:

- **Application needs to be installed**: Users are required to install an application and they need to register manually into the system
- **Disorganized Booking Process**: Travelers cannot easily find all services under an umbrella. Service providers must manually track bookings.
- **No Commission Structure**: Service providers lack a way to earn commissions for referring other providers.

- **Manual Payment Handling**: Payments are often handled manually, which can lead to delays, fraud, and tracking difficulties.
- **Multilingual Barriers**: Some Existing platforms typically don't support multilingual features, limiting their accessibility.

3.3 System Design

In the software development process, technical activities, including design, coding, implementation, and testing, are essential for constructing and validating the software system. Among these, the design holds particular significance, as the decisions made during this stage directly impact the overall success of the software implementation and its future maintainability. the design represents the initial and critical phase of the development stage. Effective design is the primary mechanism for accurately translating customer requirements into a functional software product or system. Moreover, it is within the design phase that quality is inherently integrated into the development process.

3.4 Project Development Approach

The Rapid Application Development (RAD) methodology is chosen, because of its flexibility and ability to accommodate changes throughout the development cycle. Unlike traditional models, RAD allowed me to prioritize iterative development over extensive upfront planning, which was particularly useful given the evolving requirements of my project.

3.4.1 Rapid Application Development Process (RAD)

One flexible software development methodology was rapid application development. It prioritizes development by using progress above planning, making it a viable option for research and prototype projects(Thilakarathna, 2023). Developers are capable of making numerous iterations and modifications to the software product without having to start from scratch when rapid application development (RAD) is used (Kissflow, 2024).

RAD offers several advantages to handle the complexity of the system, which includes multiple actors such as Admin, Service Providers, and Users, and functionalities like payment processing, service provider recommendations, commission calculations, and withdrawal management.

RAD's quick Prototyping feature enables the development of working prototypes early in the development cycle. In the context of the Travel Assistant Bot Application, core components like the user service journey, service provider management, and admin functionalities were prototyped and presented for feedback early on. This helped in clarifying requirements and

ensuring alignment with needs. RAD's Frequent Iterations feature enables multiple iterations, and various components of the system like the commission and deduction process for cash payments, multilingual support, and the service provider referral system will be refined. RAD's flexible nature allowed for the easy incorporation of changes.

3.5 The Architecture of the Application

The System architectural diagram (Figure 3.4.1-1) of the Travel Helper Application illustrates the interaction between various system components. The Client communicates with the system via a Telegram Bot API, which serves as an interface for end-users. The Bot Server handles incoming requests from the client through telegram API and forwards them to the Web Service. Then, the web service layer acts as the business logic component and processes the user inputs and interactions. It interacts with the Database, which stores user information, service details, payments, and reviews. Additionally, the Admin Client has direct access to the web service for system management purposes, including service categorization, managing commissions, and overseeing user and provider interactions. This architecture efficiently segregates the presentation, logic, and data layers, ensuring maintainability and scalability.

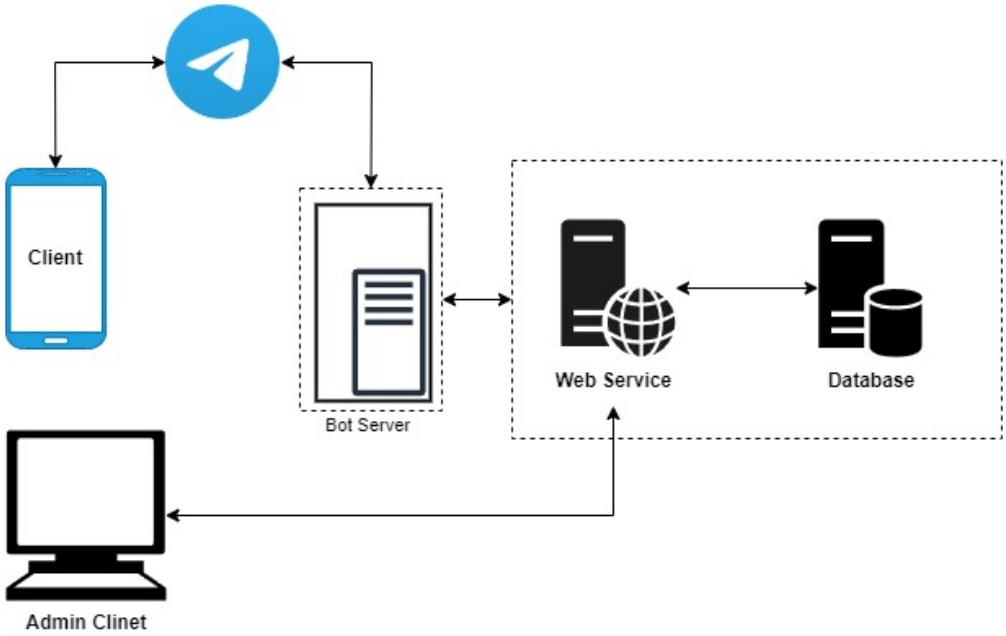


Figure 3.4.1-1: Architectural Design of Travel Helper Application

3.5.1 Software Architecture Patterns

Many design patterns & approaches can be identified for web-based development projects such as Microservices patterns, Model-View-Controller (MVC) patterns, Client-server patterns, layered patterns, and Controller-responder patterns, out of these patterns Layered architecture is Suitable for this application.

3.5.2 Layered Architecture of Travel Helper Application

Layered architecture patterns are shown in Figure 3.5.2 is an n-tiered pattern where the components are organized in horizontal layers. This is the traditional method for designing most software and is meant to be self-independent. This means that all the components are interconnected but do not depend on each other (Priyal Walpita, 2019).

Layered Architecture of Travel Helper Application

1. **Client Layer (Presentation Layer)**

   The client layer interacts with users via a Telegram Bot for travellers and service providers, and through a web interface for admin functions.

   - **Components:**
     - **Telegram Bot Client**: User interface through which users (travellers, service providers) can interact with the bot.
     - **Admin Web Interface**: Admin will use a web interface to manage the business operations.

2. **Controller Layer (API Gateway / Web Layer)**
   - **Components:**
     - **Bot Controller**: Handles bot interactions and processes incoming requests from the client via the Telegram API.
     - **Admin Controller**: Handles HTTP requests from the admin web interface.

3. **Service Layer (Business Logic Layer)**
   - **Components:**
     - All the application-related services and logic are wrapped into this layer. The service layer contains the core business logic of the Travel Helper Application. This layer processes requests from the controller and communicates with the repository layer for data access.

4. **Repository Layer (Data Access Layer)**
    - **Components:**
        - This layer abstracts the data access code and interacts with the database. Using Spring Data JPA, or similar ORM tools, repositories provide an interface for saving, retrieving, and modifying data.

5. **Database Layer (Persistence Layer)**
    - **Components:**
        - The database stores all the persistent data of the system. This layer contains entities and database tables for the application.

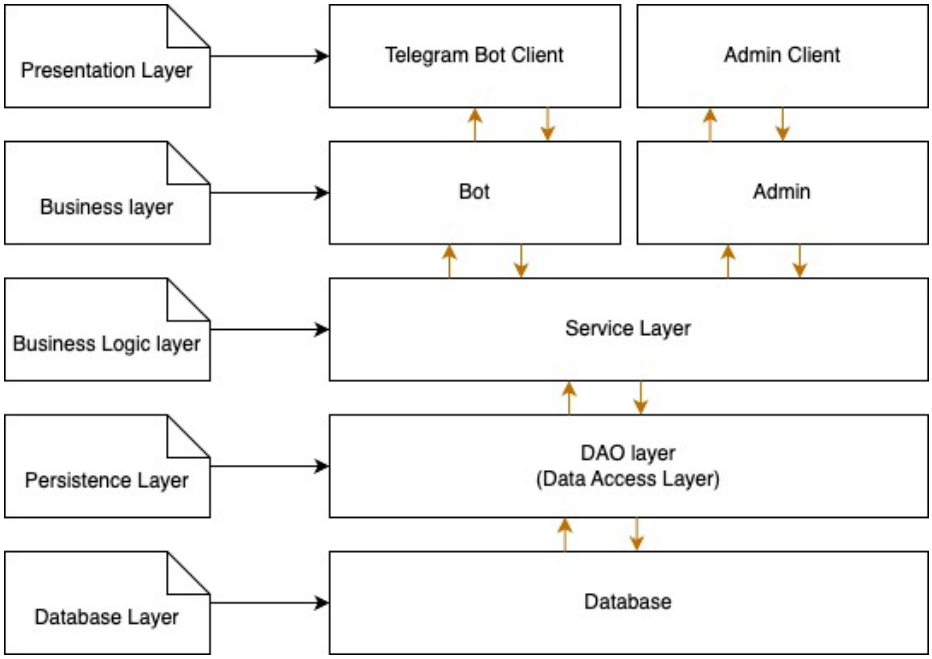


Figure 3.5.2-1: Layered Architectural Diagram for Travel Assistant Application

Telegram Bot Layer

This layer interfaces with the client using webhooks to receive and send messages to the bot server. The webhook mechanism ensures real-time communication between the Telegram client and the bot application, where user requests are processed. Webhooks also eliminate the request lost by managing the internal queue mechanism

Admin Application Layer

The admin client communicates directly with the web server via HTTP requests. This allows the admin to manage tasks like user approvals, managing services, and handling transactions without the need for the webhook mechanism, as it's typically a direct interaction with the system.

webserver

The core logic resides here, processing requests from the Telegram bot and the admin. Common functionalities and business logic are managed at this layer, ensuring a centralized processing unit for the entire system.

Database:

The bot server interacts with web services for processing and communicates with the database to store and retrieve relevant information. The web service acts as a middle layer, managing both admin requests and bot interactions before forwarding them to the database.

The Layered Architecture is selected because of its ability to decouple different parts of the system. This modular approach will help to integrate the different mechanisms such as webhooks and REST API as needed.

3.5.3 Security Design

- **JWT Authentication and Cookies**: For securing access to admin APIs and managing user sessions.
- **SSL/TLS**: Encryption for all data in transit, ensuring that sensitive information is transmitted securely.
- **Role-Based Access Control (RBAC)**: Ensures that only allowed admins can access admin functionality.
- **Data Encryption**: Sensitive data such as passwords, payment details, and personal information will be encrypted both at rest and in transit.

3.6 Design methodology

Unified Modelling Language (UML) is used to visualise and structure the design of the Travel Helper Application. Several UML diagrams will be employed to represent different views of the system.

Use Case Diagram: Identifies the primary actors such as Users, Admins, and Service Providers, and the various actions they can take within the system, such as login, booking a service, payment, and withdrawing funds.

Use Case Narratives: Describe each use case in detail, including actors, preconditions, postconditions, and the flow of events. Each narrative provides a clear understanding of how a specific use case should be executed.

Entity-Relationship Diagram (ER Diagram): A visual representation of the entities (User, Admin, Service Provider, Service, Payment) and their relationships in the database. This diagram helps in understanding the database structure and how the entities relate to one another.

Flowchart: Describes the flow of activities in specific processes like booking a service or making a payment. It shows the decision points, input/output processes, and overall flow within the system.

3.6.1 Use Case Diagram

The use case diagram (Figure 3.6.1) is a type of UML diagram used to capture the functional requirements of a Travel Assistant Bot application, showing interactions between users or service providers' systems admin.

Use case diagrams help in gathering and organizing the functional requirements of a system, by providing a clear, high-level view of how actors are interacting with system functionalities /use cases(Mule and Yashwant Waykar, 2015).

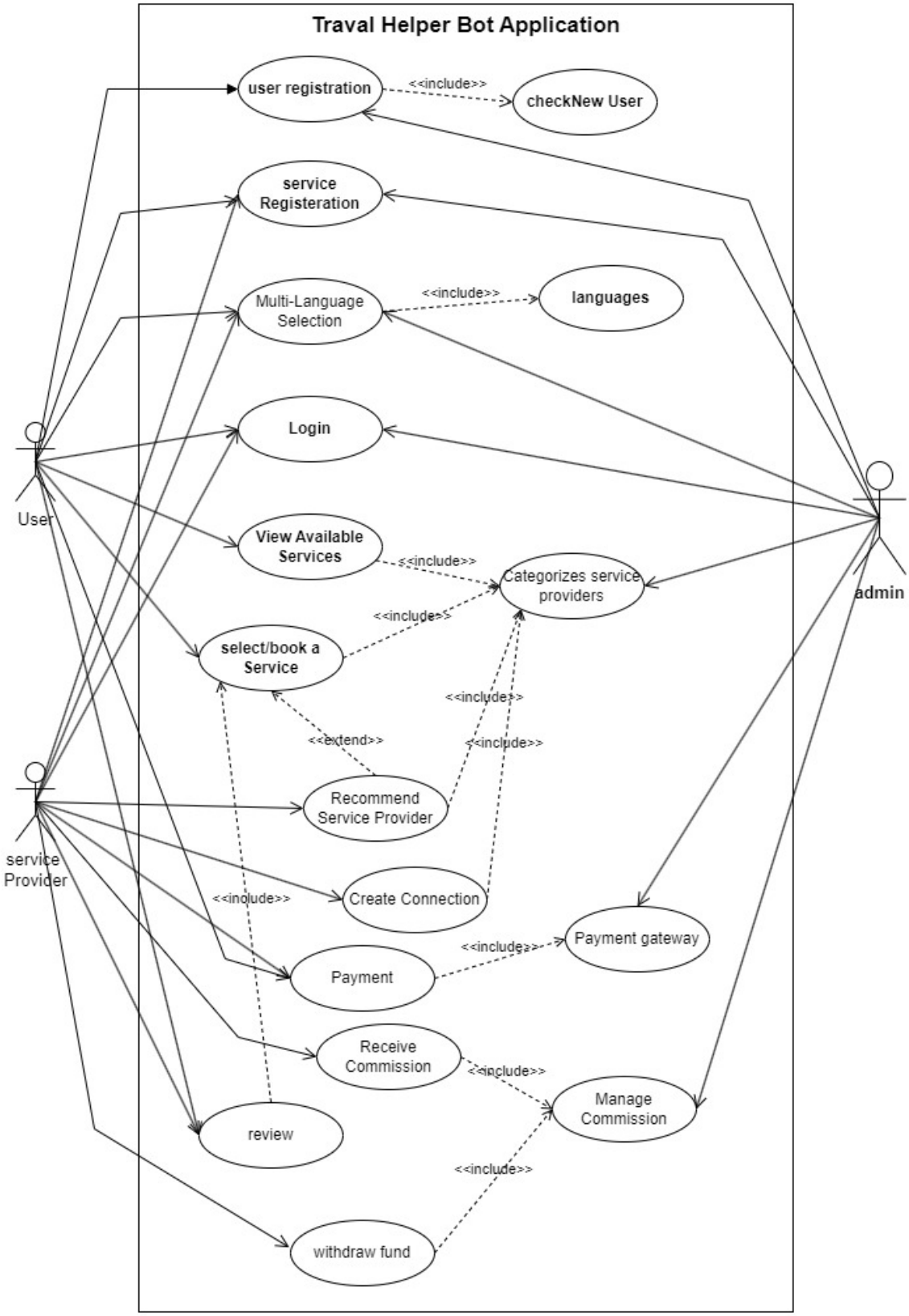


Figure 3.6.1-1: Use case Diagram for Travel Assistant Application

3.6.2 Use Case Narratives

Detail functionalities are described in the use case narratives tables below.

Table 3.6.2-1: User Registration of Traveler Assistant Bot Application

Use Case 01: User Registration (U1)	
Author	S. Arangajanan
Purpose	This use case allows new users to register themselves in the Travel Helper Bot Application. It checks if a user is new and stores their credentials in the system.
Requirements Traceability	1. The system must allow users to register with their details. 2. The system should prevent duplicate accounts by checking if the user is new. 3. The system should notify users upon successful registration.
Priority	High This is a critical feature, and it is a foundation feature of the application that allows new users to onboard into the system.
Preconditions	1. The user has access to the registration interface. 2. The user is not already registered in the system. 3. The user information will be taken from telegram if he is an ordinary user. 4. The system user provides valid input (email, username, password, etc.).
Postcondition	1. A new user account is created and stored in the system database. 2. The user is authenticated and redirected to the welcome screen.
Actors	User & system
Extends	-
Flow of Events	**Basic Flow**: 1. If system user, the user navigates to the registration page (system). Then the user enters the required registration information (email, username, password, etc.). 2. If an ordinary user, the user navigates to the chatbot then registration will automatically be instantiated. 3. The system checks if the user already exists by querying the database.

	4. If the user is new, the system saves their details and creates a new user account. 5. The system sends a confirmation message to the user. 6. The system redirects the user to the login page or home screen after successful registration. **Alternative Flow**: User enters invalid data (e.g., incorrect email format) 1. The system displays an error message prompting the user to enter valid details. 2. The user corrects the information and resubmits the registration form. User already exists in the system. 1. The system informs the user that an account with the provided email or username already exists. 2. The user can choose to recover their account or use another email to register. **Exceptions**: • Database connection error or system failure during registration. 1. The system displays an error message indicating the failure and asks the user to try again later. 2. The user may choose to cancel the registration or retry later. • User's network connectivity issues. 1. The system notifies the user that their connection has failed and prompts them to check their network connection.
Includes	**Use Case 02**
Notes/Issues	

Table 3.6.2-2: Check New User of Traveler Assistant Bot Application

Use Case 02: Check New User (U2)	
Author	S. Arangajanan
Purpose	The system checks if a user is already registered. This prevents duplicate accounts from being created using the same email address or username.

Requirements Traceability	1. The system must be able to verify if a user exists in the database before registering a new user. 2. The system should provide feedback to users if they already have an account.
Priority	Medium
Preconditions	1. The user attempts to register.
Postcondition	1. The system identifies if the user already exists in the database.
Actors	User & system
Extends	-
Flow of Events	**Basic Flow:** 1. The system receives the email/username/mobile number entered by the user or get from bot. 2. The system queries the database to check for existing accounts matching the input. 3. If no existing account is found, the system allows the registration to proceed. 4. If a matching account is found, the system alerts the user that they already have an account. **Alternative Flow**: User already exists: The system finds that the user already exists and informs them to log in or recover their account. **Exceptions**: Database error: 1. The system is unable to query the database. 2. The system shows an error message asking the user to try again later.
Includes	**-**
Notes/Issues	Ensure the check is efficient to avoid delays in the registration process

Table 3.6.2-3: Service Registration of Traveler Assistant Bot Application

Use Case 03: Service Registration (U3)	
Author	S. Arangajanan
Purpose	This use case allows service providers (e.g., travel guides, and hotels) to register their services with the Travel Helper Bot Application.
Requirements Traceability	1. Service providers should be able to register their services by providing the necessary details. 2. The system should validate the details and store the service provider's information.
Priority	High
Preconditions	1. The service provider has not yet registered their service.
Postcondition	1. The service provider's details are stored, and they can be searched or recommended to users.
Actors	Service Provider & system
Extends	-
Flow of Events	**Basic Flow:** 1. The service provider accesses the registration page. 2. The service provider enters the necessary details for their service (e.g., service type, contact info). 3. The system validates the data and stores the service provider's details in the system. 4. The system confirms the registration and makes the service available for users to search. **Alternative Flow**: Invalid service details: 1. The service provider enters incomplete or incorrect information. 2. The system prompts the provider to correct the details and resubmit **Exceptions**: Database error: 1. The system cannot store the service provider's information due to a system error.

	2. The system informs the provider to try again later.
Includes	**-**
Notes/Issues	Ensure that the system can handle different types of services (e.g., travel guide, accommodation, vehicle rental) when registering.

Table 3.6.2-4: Multi-Language Selection of Traveler Assistant Bot Application

Use Case 04: Multi-Language Selection (U4)	
Author	S. Arangajanan
Purpose	This use case allows users to select their preferred language in the Travel Helper Bot Application. The system will present the application interface in the selected language.
Requirements Traceability	1. The system must support multiple languages for user interaction 2. The system must allow users to switch languages from a predefined list. 3. The system should remember the user's language preference for future sessions.
Priority	Medium
Preconditions	1. The user has access to the system interface. 2. The system supports multiple languages.
Post condition	1. The system displays the interface in the selected language. 2. The user's language preference is saved for future sessions.
Actors	Service Provider, User & system
Extends	-
Flow of Events	**Basic Flow:** 1. The user navigates to the settings or language selection option in the application. 2. The system presents a list of supported languages to the user. 3. The user selects their preferred language from the list. 4. The system updates the interface to display content in the selected language. 5. The system saves the language preference for future sessions. **Alternative Flow**: Unsupported language:

	1. If the user selects an unsupported, the system informs the user and falls back to a default language. **Exceptions**: Language switch failure: 1. The system encounters an error while switching languages and fails to update the interface. 2. The system informs the user of the failure and allows them to retry or continue with the current language setting.
Includes	Languages (U5)
Notes/Issues	1. The system should handle cases where text in one language does not translate perfectly into another language. 2. Additional language options may be introduced in future updates. Ensure the system can easily accommodate new languages. 3. Consider implementing automatic language detection based on the user's region or device settings.

Table 3.6.2-5: Languages use case of Traveler Assistant Bot Application

Use Case 05: Languages (U5)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to maintain and manage the list of supported languages in the Travel Helper Bot Application. This use case ensures that the system has an updated list of languages available for user selection and that the system can present these languages correctly.
Requirements Traceability	1. The system must maintain a list of all available languages for user selection. 2. The system must support adding, updating, or removing languages by the admin. 3. The system should present the list of languages in a user-friendly way.
Priority	Medium
Preconditions	1. The system is configured to support multiple languages. 2. The admin has access to manage system settings, including languages.

Post condition	1. The list of supported languages is updated in the system. 2. Users can select from the updated list of languages.
Actors	Admin & system
Extends	-
Flow of Events	**Basic Flow:** 1. The admin logs into the system and navigates to the language management section. 2. The system displays the current list of supported languages. 3. The admin can add a new language by providing the necessary translations for system prompts. 4. The admin can update existing languages or remove languages that are no longer supported. 5. The system saves the changes and updates the list of available languages for user selection. **Alternative Flow**: No languages available: 1. If the list of languages is empty, the system will present a default language until the admin adds languages **Exceptions**: Invalid language addition: 1. The system encounters an error during the addition of a new language (e.g., missing required translations). 2. The system informs the admin of the error and allows them to correct the issue.
Includes	Multi-Language Selection (U4)
Notes/Issues	1. The system should have a fallback mechanism to a default language if there is an issue with any selected language. 2. Consider implementing regional variations of languages (e.g., Tamil, Sinhala and English). 3. Future updates should allow for dynamic language file updates without requiring system downtime.

Table 3.6.2-6: Login use case of Traveler Assistant Bot Application

Use Case 06: Login (U6)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to allow users to authenticate themselves and access their account in the Travel Helper Bot Application, and Admin Application. The login process verifies the user's credentials and grants access to system features based on their role in the system Accordingly (e.g., Admin, User, or Service Provider).
Requirements Traceability	1. The system must verify user credentials. 2. The system must allow multiple types of users (Admin, User, Service Provider). 3. The system must grant appropriate access based on the user's role. 4. The system should provide error messages if login fails.
Priority	High
Preconditions	1. The user has an active account in the system. 2. The admin user knows their login credentials, bot Users should access through telegram.
Post condition	1. The user is successfully logged in and redirected to their dashboard. 2. The system grants access to the features associated with the user's role. 3. In case of failure, the system shows an appropriate error message (e.g., invalid credentials).
Actors	User, Admin & system
Extends	-
Flow of Events	**Basic Flow:** 1. The admin user navigates to the admin panel login page and the system prompts the user to enter their username and password. On the other hand, user and service providers are headed to telegram chatbot and initiate the chat. 2. The user enters their credentials and submits the request. 3. The system verifies the credentials. 4. If the credentials are valid, the system logs the user in and redirects them to their dashboard.

	5. The user gains access to the appropriate system features based on their role (e.g., Admin, User, Service Provider). **Alternative Flow**: Forgot Password: 1. If the admin user forgets the password, who should apply the reset request through management. Disabled user/ service provider 1. If the user is disabled or locked, should be applied through bot. **Exceptions**: Invalid credentials: 1. If the user enters incorrect credentials, the system will display an error message and prompt the user to try again. Account locked 1. If the user fails to enter valid credentials multiple times, the system may lock the account and require manual intervention to unlock it.
Includes	Multi-Language Selection (U4)
Notes/Issues	1. Ensure the system provides descriptive error messages without revealing too much information about account security. 2. Implement session management to ensure only one active session per user.

Table 3.6.2-7: View Available Services use case of Traveler Assistant Bot Application

Use Case 07: View Available Services (U7): **(U7)**	
Author	S. Arangajanan
Purpose	The purpose of this use case is to allow users (both travellers and service providers) to view the list of available services offered in the Travel Helper Bot Application. These services could include accommodations, travel guides, vehicle rentals, and other tourism-related services.
Requirements Traceability	1. The system must retrieve and display all available services. 2. The system should categorize services based on type (e.g., hotels, guides, rentals).

	3. The system should allow users to filter and search for specific services. 4. The system should display detailed information about each service provider.
Priority	High
Preconditions	1. The user is logged into the system. 2. There are available services in the system's database.
Postcondition	1. The user views a list of available services. 2. The user can search, filter, and select specific services for more details. 3. The system records the services viewed for future recommendations.
Actors	User, service provider & system
Extends	-
Flow of Events	**Basic Flow:** 1. The user navigates to the "View Available Services" page. 2. The system fetches a list of all available services from the database. 3. The system categorizes the services (e.g., hotels, guides, rentals) and displays them in groups. 4. The user browses the list of services. 5. The user can click on a service to view more details, including availability, pricing, and provider information. 6. The user may proceed to select or book a service. **Alternative Flow**: Filtering and Searching: 1. The user can use the search bar or filters (e.g., location, category, price range) to narrow down the services displayed. 2. The system updates the list of services based on the user's filters. No Services Available 1. If no services are available, the system displays a message indicating that no services are currently offered. **Exceptions**: Service Provider Unavailable:

	1. If the service provider's account is inactive or temporarily unavailable, the system displays a notification, and the service will not be available for booking. Network Issues: 2. If the system cannot fetch services due to a network error, an appropriate error message will be displayed, and the user can retry.
Includes	Categorize Service Providers (U8):
Notes/Issues	1. Ensure that services display properly based on the user's language preferences from Multi-Language Selection (U4). 2. Consider implementing real-time updates for service availability.

Table 3.6.2-8: Categorize the Service Provider's use case of the Traveler Assistant Bot Application

Use Case 08: Categorize Service Providers (U8)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to categorize service providers (e.g., hotels, travel guides, vehicle rentals etc.) into specific categories within the Travel Helper Bot Application. This helps users easily navigate and select the type of service they need by organizing service providers into well-defined groups.
Requirements Traceability	1. The system must categories service providers based on the type of service they offer. 2. The system should allow filtering of service providers by category. 3. The system should update categories dynamically when new service providers are added.
Priority	Medium
Preconditions	1. Service providers have already registered in the system. 2. Service providers have correctly categorized themselves during registration.
Postcondition	1. Service providers are grouped into categories (e.g., hotels, guides, vehicle rentals). 2. Users can filter and view specific categories of service providers.
Actors	User, service provider, admin & system

Extends	-
Flow of Events	**Basic Flow:** 1. During registration, service providers select their category (e.g., hotel, guide, or vehicle rental). 2. The system assigns service providers to the selected category. 3. The system dynamically updates the list of service providers within each category. 4. Users view available service providers, grouped by category. 5. Users can browse specific categories and view details of each service provider. **Alternative Flow**: Admin Updates Categories: 1. The admin can add, update, or remove categories if new service types emerge. 2. The system automatically updates the categorization of providers based on admin changes. **Exceptions**: Incorrect Categorization: 1. If a service provider is miscategorized, the provider can request an update, or the admin can manually change the category. No Providers in a Category: 1. If a category has no active providers, the system displays a message indicating the lack of available services for that category.
Includes	Manage Categories (Admin): Admin is allowed to modify the list of available categories.
Notes/Issues	1. The categorization logic must be flexible enough to accommodate new types of service providers without requiring significant changes to the system architecture. 2. Ensure that service providers who do not select a category during registration are prompted to do so later

Table 3.6.2-9: Select/Book a Service use case of Traveler Assistant Bot Application

Use Case 09: Select/Book a Service (U9)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to allow users to browse available services provided by different service providers and make bookings for services like hotels, travel guides, or vehicle rentals through the Travel Helper Bot Application.
Requirements Traceability	1. Users must be able to view details of available services. 2. The system should allow users to select and book a service. 3. The system must update the service status after booking.
Priority	High
Preconditions	1. The user must be logged into the system. 2. Service providers must have active services listed in the system. 3. Service providers must be properly categorized and have available slots for booking.
Postcondition	1. The service is booked, and the user receives a confirmation. 2. The system updates the availability of the service provider. 3. The booking details are saved in the booking log for future reference.
Actors	User, service provider & system
Extends	-
Flow of Events	**Basic Flow:** 1. The user navigates to the "View Available Services" page. 2. The user browses through the categorized list of services (e.g., hotels, guides, vehicle rentals). 3. The user selects a specific service provider and views details about the service (e.g., price, availability, description). 4. The user confirms the selection and initiates the booking process. 5. The system prompts the user to input booking details such as date, time, and other preferences. 6. The user submits the booking request. 7. The system verifies availability and processes the booking. 8. The system confirms the booking and provides a reference number to the user.

	9. The service provider is notified of the new booking. 10. The system updates the availability of the service provider. **Alternative Flow**: Booking Unavailable: 1. If the selected service is no longer available, the system will inform the user and provide alternative services within the same category. 2. The user selects another service and repeats the booking process. **Exceptions**: Payment Failure: 1. If payment is required for the booking and the payment fails, the system will cancel the booking and notify the user to retry with a different payment method. Service Provider Declines Booking: 1. In rare cases, if the service provider declines the booking, the system will notify the user and recommend other available providers.
Includes	View Available Services (U7): Users must first view and browse services before selecting a specific one. Payment (U10): If the booking requires payment, the system will direct the user to the payment process.
Notes/Issues	1. The system should include an automated mechanism to send reminders to users about upcoming bookings. 2. There must be a cancellation policy handled within the booking system to allow users or service providers to cancel if needed.

Table 3.6.2-10: Payment use case of Traveler Assistant Bot Application

Use Case 10: Payment (U10)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to facilitate the payment process for users who have selected a service from a service provider. The system allows users to make payments through secure gateways while tracking the payment status and ensuring that service providers receive their earnings.

Requirements Traceability	1. The system must allow users to make payments using various methods (e.g., credit card, online payment). 2. The system must securely handle payment data and ensure encryption. 3. Payment confirmation and receipts must be sent to both the user and the service provider. 4. The payment process must update the transaction history and adjust the balance in the service provider's account.
Priority	High
Preconditions	1. The user has selected a service and initiated the booking process. 2. The service provider has confirmed availability and the total amount is determined. 3. The user must have a valid payment method.
Postcondition	1. The payment is processed successfully, and a receipt is generated for both the user and the service provider. 2. The payment status is updated in the system. 3. The service provider's account balance is updated. 4. The user's transaction history is updated with the details of the payment.
Actors	User, service provider & system/ Payment Gateway
Extends	-
Flow of Events	**Basic Flow:** 1. The user finalizes the service booking and proceeds to the payment page. 2. The system calculates the total amount, including taxes and fees. 3. The user selects a payment method (e.g., credit card, digital wallet). 4. The system redirects the user to a secure payment gateway to enter payment details. 5. The payment gateway processes the payment and confirms with the system. 6. The system receives the payment confirmation and updates the payment status.

	7. The system generates a receipt for the user and service provider. 8. The service provider's account balance is updated to reflect the payment. 9. The user's transaction history is updated with the details of the transaction. **Alternative Flow**: Payment Failure: 1. If the payment fails due to insufficient funds or network issues, the system informs the user of the failure. 2. The user is given the option to retry the payment or select an alternate method. 3. If the user retries and the payment is successful, the system proceeds as per the basic flow. **Exceptions**: Payment Failure/ unavailable: 1. If the payment gateway is unavailable, the system informs the user and provides alternative methods (e.g., manual bank transfer). Service Provider Declines Booking: 1. If the user provides an invalid payment method, the system notifies the user and requests an update.
Includes	View Available Services (U7): Payment is initiated after selecting a service. Transaction History (U12): The system records the transaction details for future reference.
Notes/Issues	1. Ensure that the payment process complies with international standards for financial data security. 2. The system should provide refunds or cancellations if the user chooses to terminate the service after payment.

Table 3.6.2-11: Recommend Service Provider use case of Traveler Assistant Bot Application

Use Case 11: Recommend Service Provider (U11)	
Author	S. Arangajanan
Purpose	The purpose of this use case is to allow service providers to recommend other service providers to users based on user preferences or the nature of the requested service. This recommendation system helps users discover trusted and relevant providers, enhancing their service experience.
Requirements Traceability	1. The system must allow service providers to recommend other providers. 2. The system should ensure that recommended providers have a good reputation and are relevant to the user's needs. 3. The system should notify users of recommended providers during their service selection process.
Priority	Medium
Preconditions	1. The user must have selected or viewed a specific service. 2. Service providers must have established connections with other providers. 3. The recommended provider must have available services for the user's desired time.
Postcondition	1. The user is presented with a recommendation of additional service providers. 2. The recommendation details are stored in the system, which logs the referral. 3. The user may select a recommended service provider and proceed with booking.
Actors	User, service provider & system
Extends	-
Flow of Events	**Basic Flow:** 1. The user views or selects a service from a particular service provider (e.g., a hotel or guide).

	2. The service provider evaluates the user's service needs and suggests additional service providers, either for complementary services or as alternatives. 3. The system retrieves relevant recommendations based on the service provider's connections, user preferences, and service compatibility. 4. The system presents the recommendations to the user, along with details of the suggested providers (e.g., description, price, and reputation). 5. The user may choose to review the recommended provider's details. 6. If interested, the user can initiate booking with the recommended service provider. 7. The system logs the recommendation and establishes any referral commission for the recommending service provider. **Alternative Flow**: No Suitable Recommendations: 1. The user may either proceed with the current service provider or return to the service listing to find another provider. **Exceptions**: Service Provider Connection Issue: 1. If the recommending service provider does not have any valid connections with other providers, the system will prevent the recommendation from being made. Recommendation Declined: 1. If the user declines the recommendation, the system simply logs the action and proceeds with the service provider selection.
Includes	Create Connection (U12): The recommendation is based on prior connections established between service providers. View Available Services (U7): Users must first view services to receive recommendations.
Notes/Issues	1. The recommendation system should provide accurate and valuable suggestions to avoid spamming users with irrelevant providers.

	2. Commission rates for recommendations must be transparent and trackable for both service providers and admins.

Table 3.6.2-12: Create Connection use case of Traveler Assistant Bot Application

Use Case 12: Create Connection (U12)	
Author	S. Arangajanan
Purpose	This use case is designed to enable service providers to create connections with other service providers. The connections allow them to refer services to each other and enhance their network, which impacts service recommendations and commission sharing.
Requirements Traceability	1. The system must allow a service provider to initiate and accept connection requests from other service providers. 2. The system must track the connection status and display a list of connected service providers. 3. Recommendations based on these connections should affect service prioritization and commission allocation.
Priority	Medium
Preconditions	1. The service provider is logged into the system. 2. The service provider has access to a list of other registered service providers. 3. The service provider must have the necessary permissions to create connections.
Postcondition	1. A new connection is successfully established between the service provider and another provider. 2. The connection is stored in the system and is visible in both providers' profiles. 3. The connection impacts service recommendations and commission allocation for referred users.
Actors	Service provider & system
Extends	-
Flow of Events	**Basic Flow:** 1. The service provider logs into the system and navigates to the "Connections" section.

	2. The service provider searches for another service provider from the list or recommendations. 3. The service provider selects another provider and sends a connection request. 4. The system sends a notification to the selected service provider. 5. The recipient service provider accepts the connection request. 6. The system confirms the connection and updates both service providers' profiles with the new connection. 7. The connection impacts service prioritization, allowing the connected providers to recommend each other. **Alternative Flow**: Connection Request Denied: 1. If the recipient denies the connection request, the system notifies the initiating service provider of the denial. 2. No connection is created, and both profiles remain unchanged. Pending Connection 1. If the recipient does not respond immediately, the connection request remains pending. 2. The system periodically reminds the recipient of the request until it is accepted or denied. **Exceptions**: Service Provider not available: 1. If the selected service provider is no longer available (e.g., deactivated), the system informs the initiating provider and cancels the request. Duplicate Connection: 1. If a connection already exists between the two providers, the system prevents the creation of duplicate connections.
Includes	**Recommend Service Provider (U11)** **Receive Commission (U13):** Connections impact commission sharing when referred users hire the connected service provider.
Notes/Issues	1. Consider adding a feature to allow service providers to filter potential connections based on service category, location, or rating

Table 3.6.2-13: Receive Commission use case of Traveler Assistant Bot Application

Use Case 13: Receive Commission (U13)	
Author	S. Arangajanan
Purpose	This use case allows service providers to receive commissions when users hire another service provider based on their recommendations. The commissions are automatically calculated and recorded after a successful transaction between the user and the recommended service provider.
Requirements Traceability	1. The system must track recommendations made by service providers. 2. The system must calculate and allocate a commission percentage to the referring provider when a user hires the recommended provider. 3. The system must store the commission details in the service provider's account
Priority	High
Preconditions	1. A service provider has recommended another provider to a user. 2. The user has hired the recommended provider for a service. 3. The service transaction is completed, and payment has been processed.
Postcondition	1. The system calculates the commission and credits it to the referring service provider's account. 2. The commission details are logged in the transaction history of both service providers involved. 3. The referring service provider can view and withdraw the earned commission.
Actors	Service Provider (Referrer & Referred) & system
Extends	Payment (U10)
Flow of Events	**Basic Flow:** 1. The referring service provider recommends another provider to a user. 2. The user hires the recommended provider for a service. 3. The service is completed, and the payment is made (either via system or in cash, if tracked in the system). 4. The system calculates the commission based on a pre-defined percentage or rate.

	5. The system credits the commission to the referrer's financial account. 6. Both service providers receive a notification about the successful commission transaction. 7. The referring provider can review their commission earnings and request withdrawal if needed. **Alternative Flow**: User Cancels Service: 1. If the user cancels the service before it is completed, the system voids the commission, and no payment is made. **Exceptions**: Failed Payment: 1. If the user's payment fails, the system does not credit the commission to the referring service provider. Provider Not Available: 1. If the preferred provider is not available or rejects the service, no commission is credited
Includes	Transaction History (U14) & Withdraw Funds (U15)
Notes/Issues	1. Ensure that the system handles cases like commission recalculation if the user negotiates a discount or changes the service terms. 2. Commissions should be visible in the service provider's financial account in real-time, but actual withdrawal may require approval from the admin.

Table 3.6.2-14: Transaction History use case of Traveler Assistant Bot Application

Use Case 14: Transaction History (U14)	
Author	S. Arangajanan
Purpose	This use case allows both service providers and users to view a detailed record of their past transactions, including payments, commissions, and withdrawals. The history ensures transparency in all financial dealings within the system.
Requirements Traceability	4. The system must record all transactions, including payments, commissions, and withdrawals.

	5. The system must allow users and service providers to view their respective transaction histories. 6. The system must provide filtering and sorting options to review transactions based on date, status, or type
Priority	High
Preconditions	4. A user or service provider must be registered and logged into the system. 5. At least one transaction must have been completed.
Postcondition	4. The transaction history is displayed to the user or service provider. 5. Transactions can be filtered and sorted. 6. Details of each transaction, such as the amount, date, status, and participants (from/to), are shown.
Actors	User, Service provider & Admin
Extends	-
Flow of Events	**Basic Flow:** 8. The admin, user or service provider logs into the system. 9. They navigate to the "Transaction History" page. 10. The system retrieves all past transactions related to the logged-in user or service provider. • The system displays the transactions in a chronological list, showing the following details: • Transaction type (e.g., booking, payment, commission, withdrawal). • Transaction participants (from/to parties). • Amount. • Date of the transaction. • Status (completed, pending, failed). 11. The user or service provider can filter transactions by: • Date range. • Status. • Transaction type. 12. The user or service provider can click on any transaction to see further details.

	Alternative Flow: No Transactions Available: 2. If no transactions are available, the system shows an appropriate message. Failed Transaction 3. If a transaction fails, it is displayed with a "failed" status, and further information is provided on the issue **Exceptions**: System Unavailable: 2. If the system is down or transaction data cannot be retrieved, an error message is shown, and the user is advised to try again later.
Includes	Receive Commission (U13) & Withdraw Funds (U15)
Notes/Issues	1. Transactions should be displayed in real time to ensure up-to-date information. 2. Admins should have access to all transaction records for auditing purposes, including those of individual users and service providers.

Table 3.6.2-15:Withdraw Funds use case of Traveler Assistant Bot Application

Use Case 15: Withdraw Funds (U15)	
Author	S. Arangajanan
Purpose	This use case allows service providers to request the withdrawal of their earnings from the system. Admins are responsible for reviewing and approving these withdrawal requests before funds are released.
Requirements Traceability	1. The system must allow service providers to request withdrawal of their earnings. 2. The system must track withdrawal requests and their statuses (e.g., pending, approved, rejected). 3. The admin must review, approve, or reject withdrawal requests.
Priority	High
Preconditions	1. The service provider must have an account balance with sufficient earnings to withdraw. 2. The service provider must be logged into the system.
Postcondition	1. A withdrawal request is created and sent to the admin for review.

	2. The admin approves or rejects the request. 3. If approved, the funds are transferred to the service provider's linked account.
Actors	Service provider & admin
Extends	-
Flow of Events	**Basic Flow:** 1. The service provider logs into the system. 2. The service provider navigates to the "Withdraw Funds" section. 3. The system displays the available account balance. 4. The service provider enters the amount to withdraw and submits the request. 5. The system creates a withdrawal request and marks it as "pending." 6. The admin receives the withdrawal request. 7. The admin reviews the request, ensuring the service provider's balance is sufficient. 8. The admin approves or rejects the request. • If approved, the system initiates the fund transfer to the service provider's linked bank account. • If rejected, the system notifies the service provider with the reason for rejection. 9. The system updates the status of the request to "approved" or "rejected." 10. The service provider is notified of the final status of the withdrawal request. **Alternative Flow**: Insufficient Funds: 1. If the service provider attempts to withdraw more than their available balance, the system displays an error message, and the withdrawal request cannot be submitted. **Exceptions**: System Unavailability:

	1. If the system is down during the request submission or fund transfer, the withdrawal request is put on hold, and the service provider is notified to try again later.
Includes	Transaction History (U14)
Notes/Issues	1. The withdrawal approval process should be time-sensitive to avoid delays for service providers. 2. Admins may set withdrawal limits or fees, which must be displayed to the service provider during the request process.

Table 3.6.2-16: Review the Service use case of the Traveler Assistant Bot Application

Use Case 16: Review the Service (U16)	
Author	S. Arangajanan
Purpose	This use case allows users (travellers/tourists) to provide feedback and review the service they received from a service provider. These reviews can influence future service provider recommendations and rankings within the system.
Requirements Traceability	1. Users must be able to review services after booking. 2. Reviews should be visible to other users when selecting service providers. 3. Service providers should be able to view the reviews they receive.
Priority	Medium
Preconditions	1. The user must have successfully booked and used a service. 2. The user must be logged into the system.
Postcondition	1. The user's review is saved in the system. 2. The review is visible to other users and the service provider.
Actors	Service provider & user
Extends	-
Flow of Events	**Basic Flow:** 1. The user logs into the system. 2. The user navigates to the "Booking History" section and selects a completed service.

	3. The system displays the option to leave a review for the service provider. 4. The user rates the service (e.g., 1-5 stars) and provides written feedback. 5. The user submits the review. 6. The system saves the review and associates it with the service provider. 7. The system updates the service provider's profile with the new review and recalculates the average rating. 8. Other users can view the review when selecting service providers. 9. The service provider is notified of the new review and can view the feedback. **Alternative Flow**: Review Not Submitted: 1. If the user chooses not to submit a review, the system does not prompt further, and no data is saved. **Exceptions**: System Unavailability: 1. If a review violates the platform's guidelines (e.g., offensive language, fake reviews), the admin can remove the review after validation.
Includes	Service Selection (U7)
Notes/Issues	1. The system should allow users to edit their reviews within a set period. 2. The admin should oversee all reviews to ensure compliance with guidelines.

3.6.3 Entity-Relationship (ER) Diagram

The Entity-Relationship (ER) diagram for the Travel Assistant Application (Figure 3.6.3-1) outlines the key elements and relationships that form the foundation of the system's database. The diagram emphasizes essential features like managing users, booking services, handling

payments, calculating commissions, and supporting multiple languages. Each entity represents an important part of the system, while the relationships between them show how information is shared and interacts within the system. This diagram provides a clear view of how different components work together to support the application's functionality

The ER diagram's design aligns with the layered architecture of the Travel Assistant Application by dividing key concerns like system, user management, service booking, and payment processing into distinct sections while allowing shared logic between different components. This structured design facilitates the efficient storage and retrieval of data, which is crucial for the system's scalability and adaptability. Detailed transaction and booking logs ensure transparency and foster trust in the system, while the inclusion of multi-language support improves the application's usability for users in various regions.

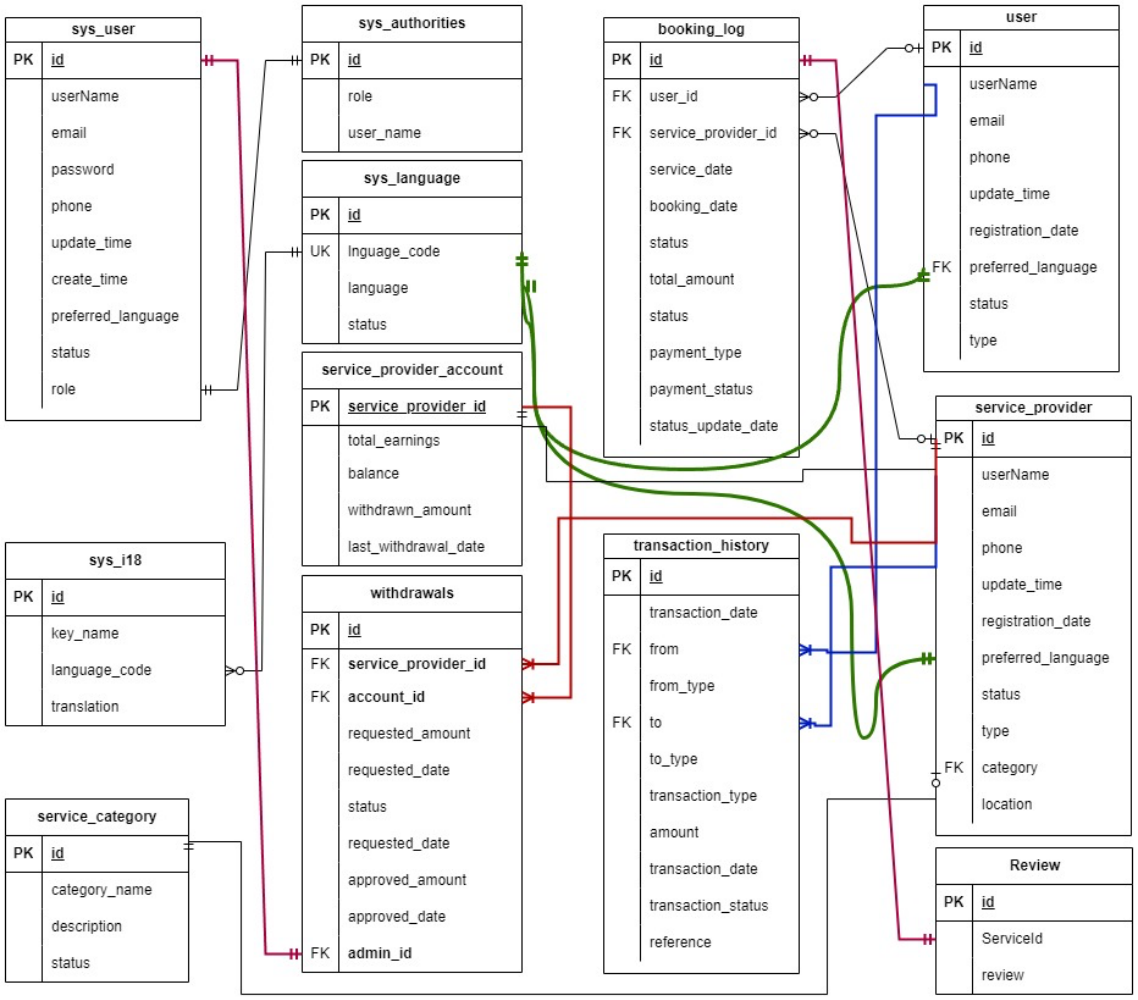


Figure 3.6.3-1: ER diagram for Traveler Assistant Bot

3.6.4 Sequential Diagram

The sequential diagram describes the core functionality of the Travel Assistant Application as shown in Figure 3.6.4-1.

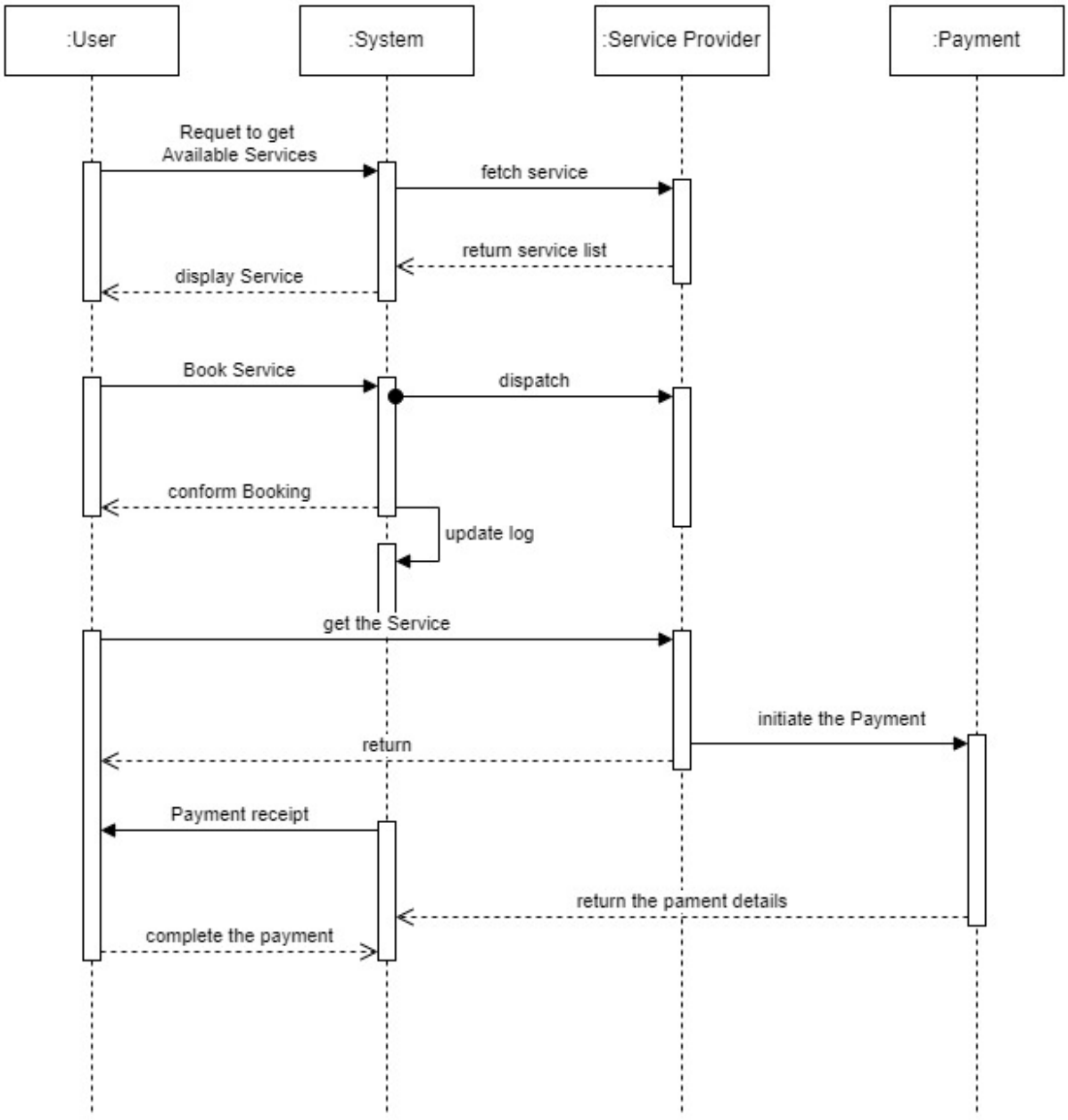


Figure 3.6.4-1: Sequential Diagram for Service Booking function of the Travel Assistant Application

3.6.5 Flow chart

Flow chat describes the system's functionality and flow. It uses simple visual tools that help us understand and represent it easily.

User Journey

The User Journey flowchart (Figure 3.6.5-1) describes the user's interaction process within the Travel Helper Application. It begins with the user logging in to the system, after which the user

can search for service providers that fit their needs. Users need to select a service provider once a service provider is selected; the user can proceed with the booking process. The system checks the payment method chosen by the user either system-based payment or cash. If cash is selected, the user proceeds to pay manually, and if the system-based payment method is chosen, the application will process the payment through its built-in transaction system. The transaction history is then recorded, and the process concludes.

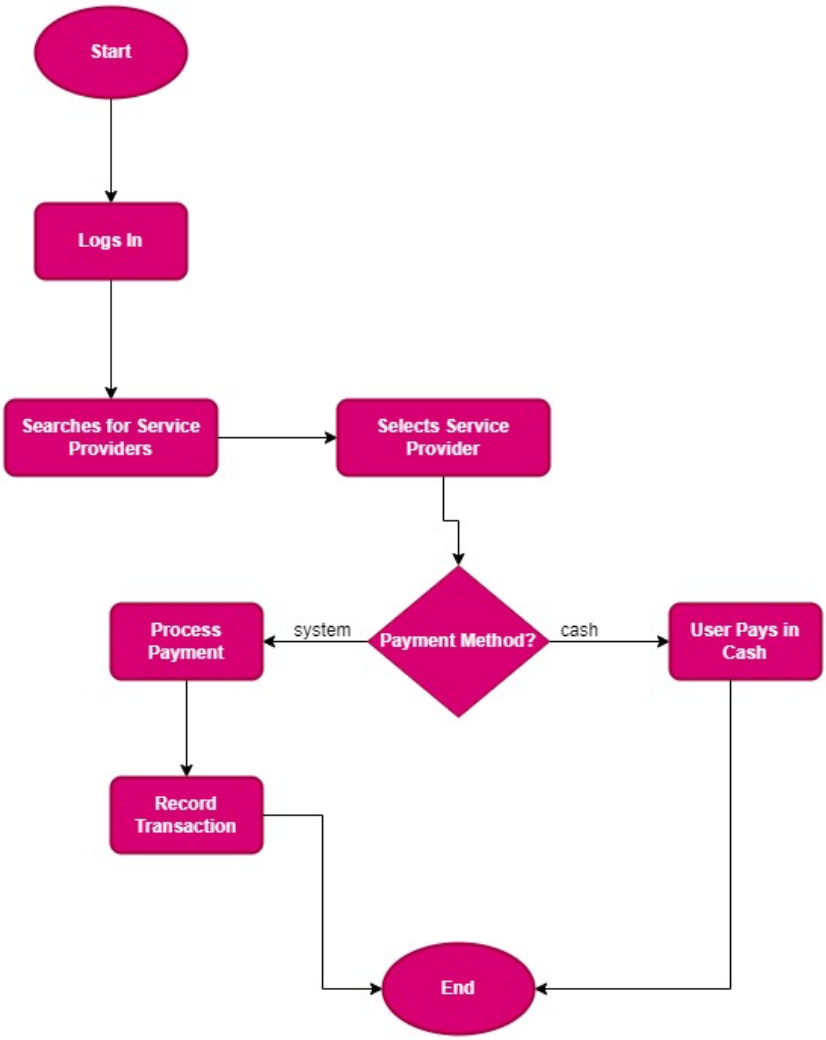


Figure 3.6.5-1: Flowchart of User Journey of Travel Assistant Bot

Service Provider Journey

The Service Provider Journey flowchart (Figure 3.6.5-2) represents the steps involved in interacting with service providers within the Travel Assistant Bot Application. The process starts with the registration and login phase, where a service provider either registers or logs into the system. Once logged in, the provider can manage or create connections with other service providers, influencing service priority and recommendations.

When a service provider receives a service request from a user, they can recommend another provider if needed. This recommendation is sent to the user, and if the recommendation is successful, the system proceeds to pay a commission to the recommending provider. In parallel, a log of the commission is recorded in the system. Once a service is provided and the user pays in cash, the system deducts the appropriate amount from the user's balance.

Providers are also able to withdraw funds, which go through the admin's review process. If the admin approves the withdrawal request, the provider receives the payment. All payment methods (e.g., cash or system-based) are processed accordingly, and the system updates the provider's account and logs the transaction history.

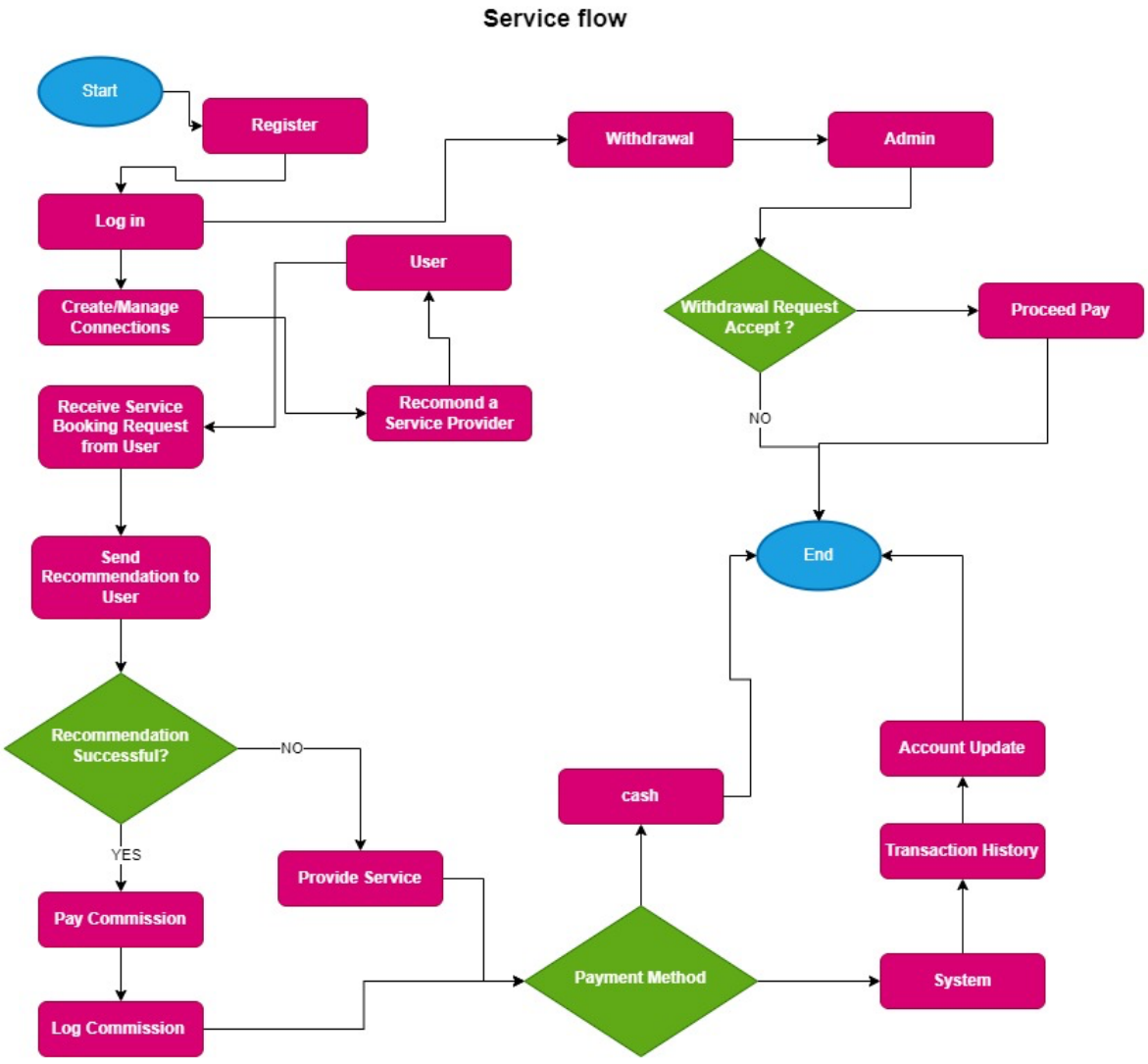


Figure 3.6.5-2: Flowchart of User Journey of Travel Assistant Bot

Admin Journey Flowchart

The Admin Journey flowchart (Figure 3.6.5-3) represents the steps an administrator follows within the Travel Assistant Application to manage users, service providers, multilingual settings, and financial transactions. The journey begins when the admin logs into the system and proceeds to various tasks based on application requirements.

The admin can view and manage users and service providers, correcting any inaccurate information or flagging incorrect data. Additionally, they can create and manage other admin users, granting them specific privileges necessary for their roles within the application.

The admin is also responsible for handling multilingual keys and localization management, ensuring that the application supports multiple languages and provides native language support for users from different regions.

The admin can view and manage users and service providers, correcting any inaccurate information or flagging incorrect data. Additionally, they can create and manage other admin users, granting them specific privileges necessary for their roles within the application.

The admin is also responsible for handling multilingual keys and localization management, ensuring that the application supports multiple languages and provides a seamless experience for users from different regions.

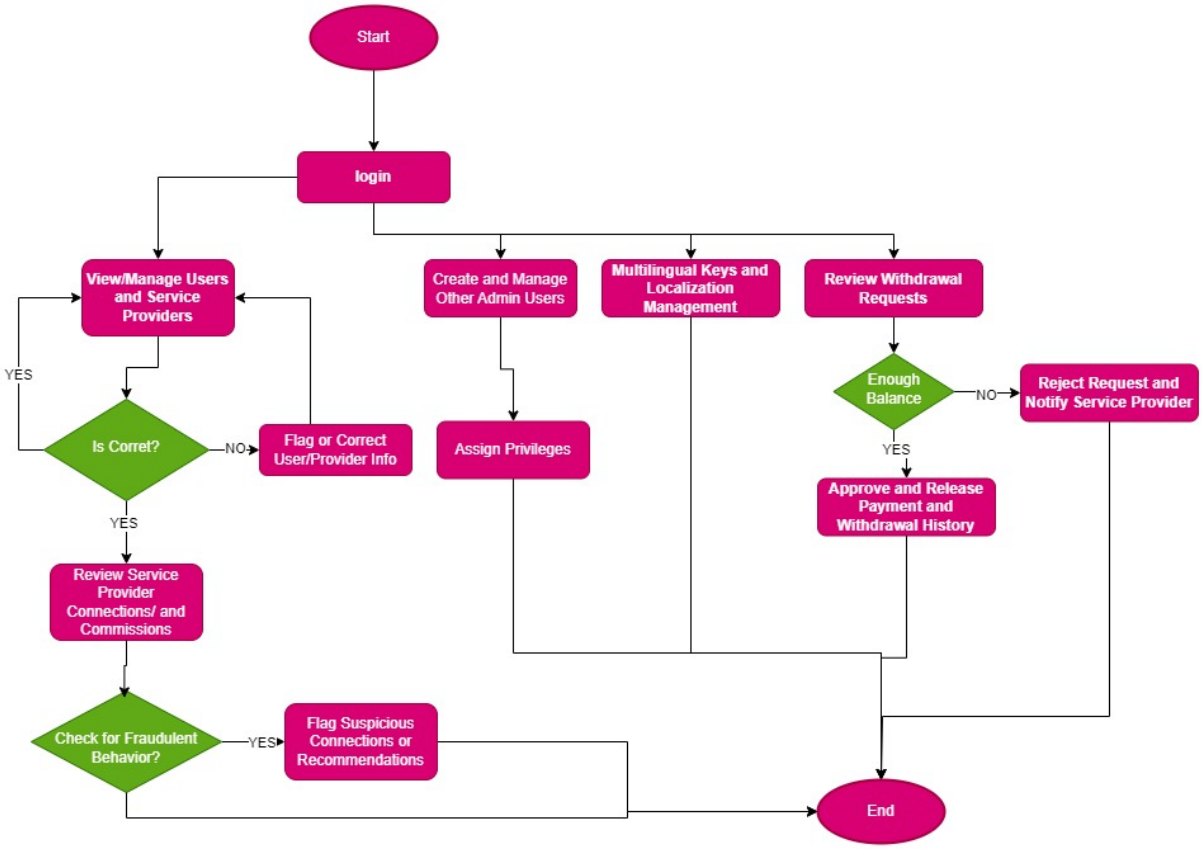


Figure 3.6.5-3: Flowchart of Admin Journey of Travel Assistant Bot

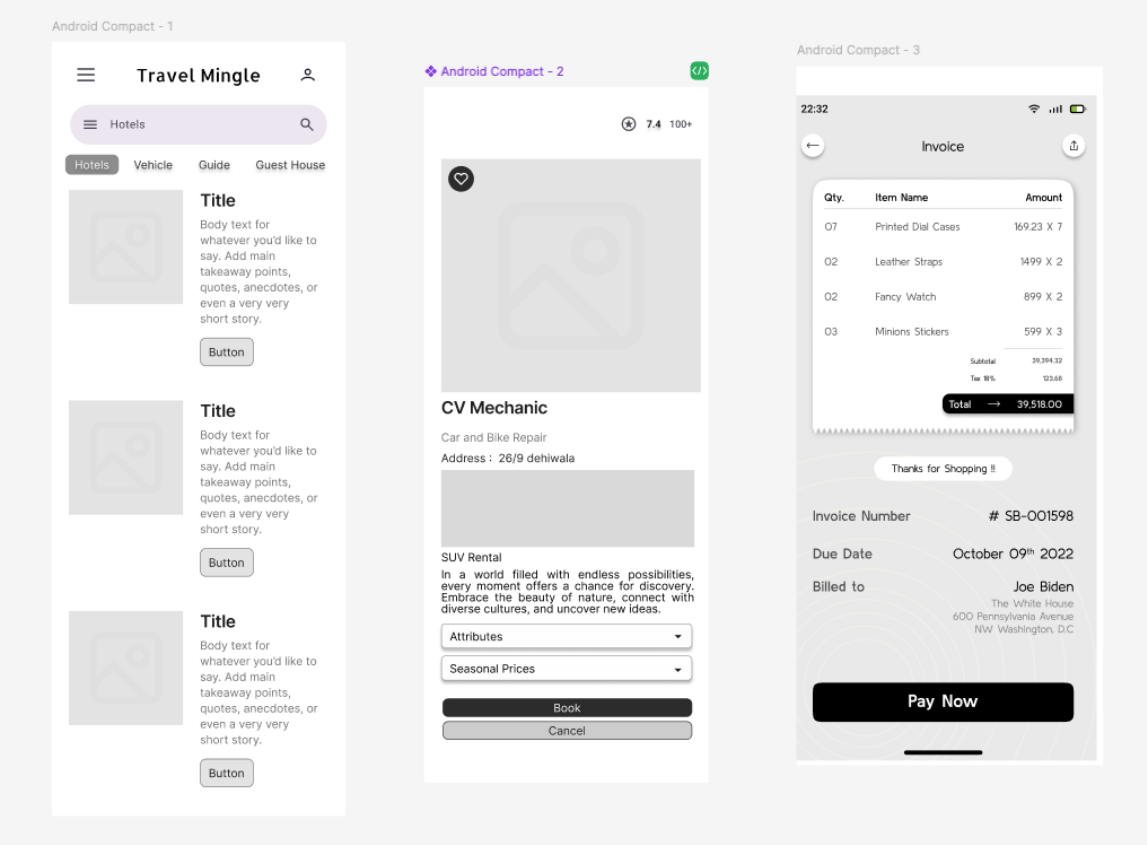


Figure 3.6.5-4 : Mini App UI designs

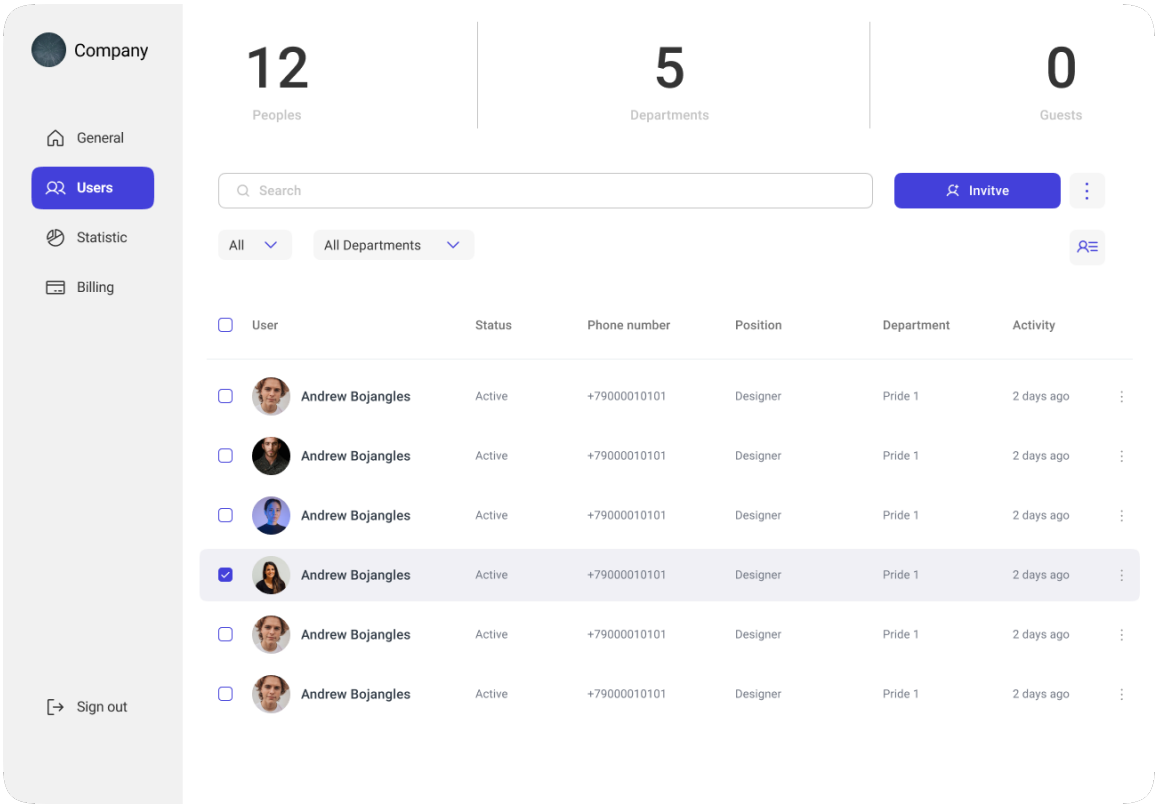


Figure 3.6.5-5: Admin panel user details view UI design

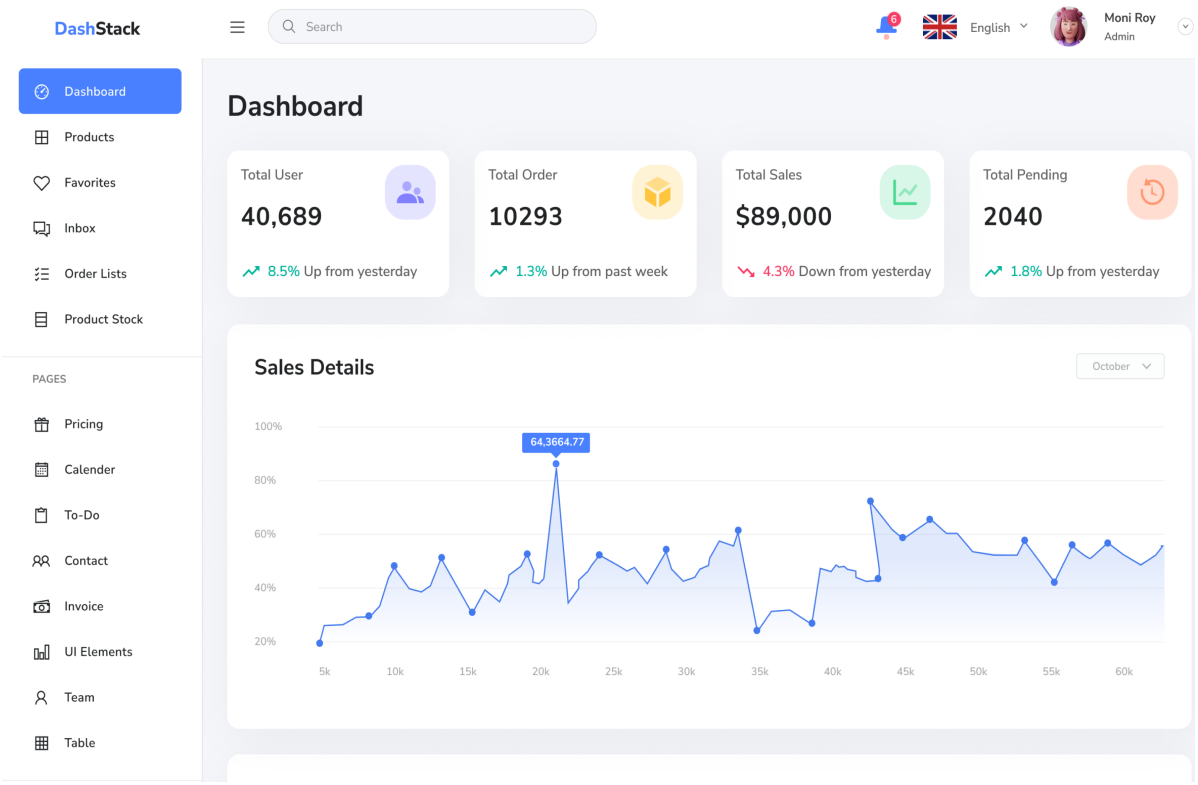


Figure 3.6.5-6: Admin Pannel Dashboard View UI design

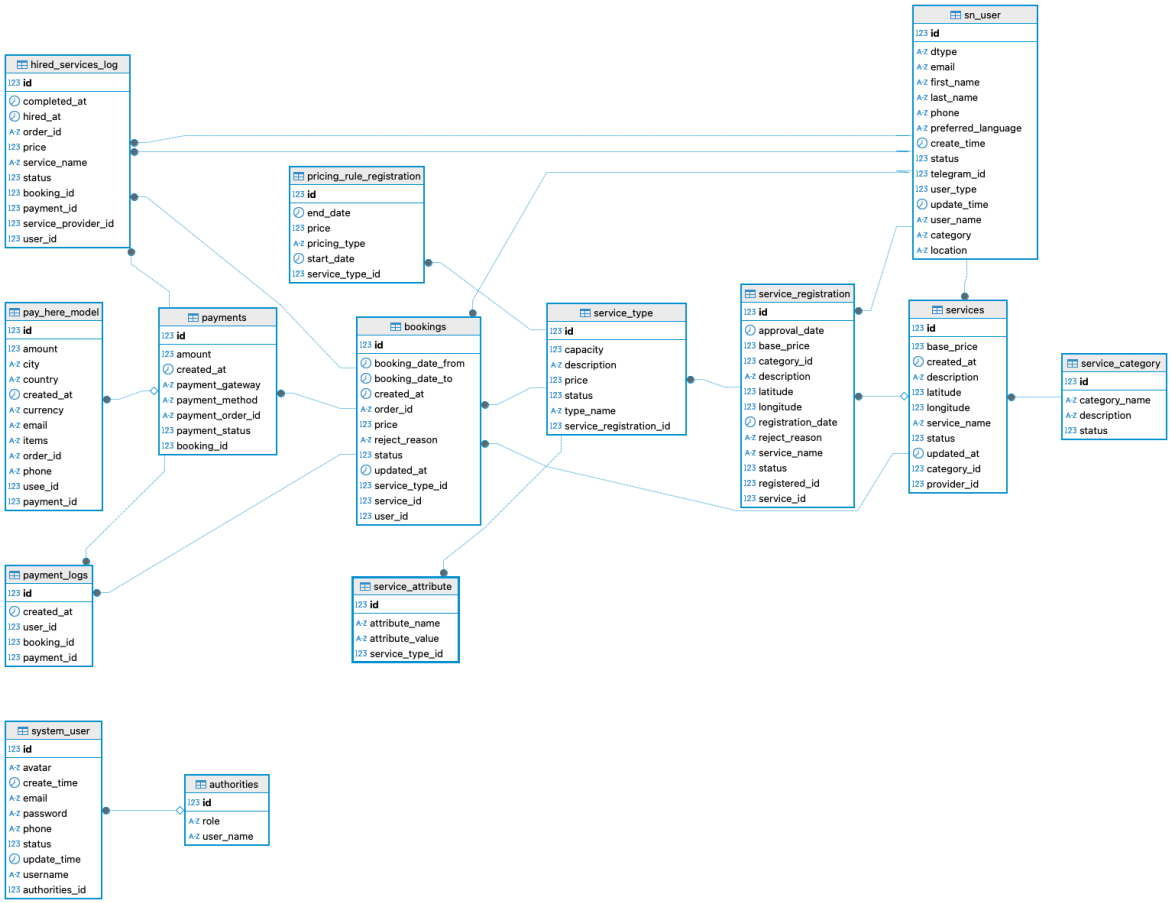


Figure 3.6.5-7: Database Design

3.7 Summary

The Design chapter outlines the structural and technical foundation of the Traveler Assistant Bot system, focusing on system architecture, security mechanisms, and development methodology. The chapter begins with a system study, identifying the limitations of existing service platforms and highlighting the need for an integrated solution to streamline service bookings and management.

The system was developed using a layered architecture, ensuring modularity and scalability. The architecture consists of a client layer (Telegram Bot and Admin Panel), a controller layer for request handling, a business logic layer for processing operations, a repository layer for database interactions, and a persistence layer using MySQL. Security measures such as JWT authentication, role-based access control (RBAC), and data encryption are incorporated to protect sensitive user information and transactions.

To ensure efficient development, the Rapid Application Development (RAD) methodology is adopted, allowing for iterative prototyping and continuous user feedback. Various UML diagrams, including use case diagrams, sequence diagrams, and entity-relationship (ER) diagrams, provide a structured representation of system functionalities and data flow. The integration of Telegram Webhooks enables real-time user interactions, while Redis caching optimizes performance by reducing database query load.

The database design (Figure 3.6.5 1) is structured to efficiently store and manage data related to users, service providers, service categories, bookings, transactions, and payment records. A relational database schema is implemented in MySQL, ensuring data integrity, quick retrieval, and smooth interactions between different modules. Entity-relationship diagrams (ERDs) define the relationships between different data entities, optimizing query execution and system efficiency.

Additionally, this chapter details payment processing mechanisms, multilingual support, and service provider referral systems, ensuring a seamless user experience. The project timeline is refined, ensuring the structured implementation of core system components. This design framework serves as the foundation for successful system implementation, meeting functional, security, and scalability requirements.

4 Implementation

4.1 Introduction

The Travel Assistant Application was developed using the Rapid Application Development (RAD) methodology, which emphasizes quick prototyping and iterative development. The RAD approach was selected due to the evolving nature of requirements and the need for frequent feedback from potential users, including travellers, service providers, and admin stakeholders. This section details the step-by-step implementation process for each module, reflecting the RAD methodology's focus on rapid iteration and component modularity.

4.1.1 Initial Prototyping and User Feedback

The development process began with creating initial prototypes for core functionalities, including user registration, service booking, and admin panel implementation.

4.1.2 Module-Based Iterative Development

To streamline development and testing, the system was divided into distinct modules:

- **Admin Module**: Implemented admin registration, login, and user role management for travellers, service providers, and admins. Security measures, such as **JWT Authentication**, were integrated to ensure secure access control.
- **User Module**: implemented user and service provider Registration using the Telegram Bot Interface.
- **Booking and Payment Module**: A booking system will be developed to allow travellers to select and book services, with payment processing through **PayHere API** integration for online transactions and automated balance deductions for cash payments.
- **Referral and Commission Module**: Created functionality for service provider referrals, where referring providers receive a commission. This module calculates commissions automatically and logs them in the transaction history.
- **Localization Module**: Integrated multilingual support to enhance user accessibility, utilizing **Redis** caching for optimized response times.

Each module was iteratively developed and tested, aligning with RAD's emphasis on rapid feedback and continuous improvement.

4.1.3 Integration of Telegram Bot via Webhooks

The **Telegram Bot** was integrated using webhook technology, allowing real-time communication between the bot and travellers. This webhook approach enabled travellers to

interact with the system through the Telegram platform, providing an accessible chat-based interface for booking services, receiving booking confirmations, and viewing payment statuses.

4.1.4 Database and Data Management

The database was designed to support the layered architecture, with tables for user data, bookings, transactions, and service provider earnings. Data integrity and consistency were maintained through **MySQL relational databases** and **Redis caching** for frequently accessed data such as multilingual keys.

4.1.5 Testing and Validation

Following each development iteration, the modules were subjected to unit testing and integration testing, ensuring that each component functioned correctly and seamlessly integrated with others.

4.2 System Architecture

The "Traveler Assistant Bot for Sri Lankan Tourism" is structured with a layered architecture, which divides the application into five distinct layers. This design enhances modularity, simplifies maintenance, and improves scalability. Each layer has a specific role:

- **Client Layer**: Manages user interactions through the Telegram bot and mini app and the admin web interface.
- **Controller Layer**: Routes requests from the client layer to the appropriate services.
- **Service Layer**: Contains the core business logic, processing requests and interacting with the data access layer.
- **Repository Layer**: Handles data access and management, interfacing with the database.
- **Database Layer**: Stores the application's data.

This architecture supports a clear separation of concerns, allowing for independent scaling and updates of individual layers.

The **Traveler Assistant Bot** follows a **layered architecture** to ensure modularity, scalability, and maintainability. The system consists of three primary components: the **frontend, backend, and database**, which communicate through well-defined interfaces.

The **frontend** comprises two distinct applications: the **Telegram Mini App**, which enables travelers to interact with the system within the Telegram ecosystem, and the **Admin Panel**, which provides administrative functionalities for managing services, users, and transactions.

Both applications are developed using **React.js** and **TypeScript** to ensure a dynamic and responsive user experience.

The **backend** is implemented using **Spring Boot**, which serves as the core business logic layer, processing requests from the frontend and executing operations such as user authentication, service management, and payment processing. It exposes **RESTful APIs** to facilitate interaction between system components. Additionally, a **webhook-based mechanism** is used to integrate the Telegram Mini App with the backend for real-time communication.

The **database layer** consists of **MySQL**, which stores structured data, including user profiles, service listings, booking transactions, and payment records. To enhance performance, **Redis caching** is employed to reduce query response times for frequently accessed data.

This architecture ensures an **efficient, secure, and scalable** system capable of handling high user traffic while maintaining data integrity and fast response times. The use of a **layered approach** promotes separation of concerns, making the system more maintainable and extensible for future enhancements.

4.3 Security Implementations

Security is a critical aspect of the Traveler Assistant Bot system to ensure data integrity, authentication, authorization, and secure transactions. The system incorporates multiple security measures across its components: Admin Panel, Telegram Mini App, Payment, and Backend API.

In the Traveler Assistant Bot system, the Admin Panel is developed using React.js and TypeScript, providing administrators with a user-friendly interface to manage services, monitor transactions, and oversee system operations. Ensuring secure access to this panel is crucial, and this is achieved through robust authentication and authorization mechanisms. The Traveler Assistant Bot implements a secure authentication mechanism for the Admin Panel using JWT-based authentication with access and refresh tokens. This approach ensures secure and efficient authentication while mitigating potential risks associated with token expiration and misuse.

4.3.1 Token-Based Authentication Strategy for Admin Pannel

The Admin Panel employs a **JSON Web Token (JWT)**-based authentication system:

Access Token (Bearer Token)

- This short-lived token is used to authenticate API requests.
- It is stored securely in **Redis** to enhance security and is retrieved during each request using the refresh token.

- The access token follows the Bearer Token format and is included in the *Authorization* header when making API calls.
- The limited lifespan of the access token helps mitigate security risks such as token theft or misuse.

Refresh Token

- This token has a **longer lifespan** and is used to generate new access tokens without requiring the user to log in again.
- It is stored **securely in HTTP-only cookies** in the frontend, preventing **Cross-Site Scripting (XSS) attacks**.
- The refresh token is sent to the backend when the access token expires to request a **new access token**.

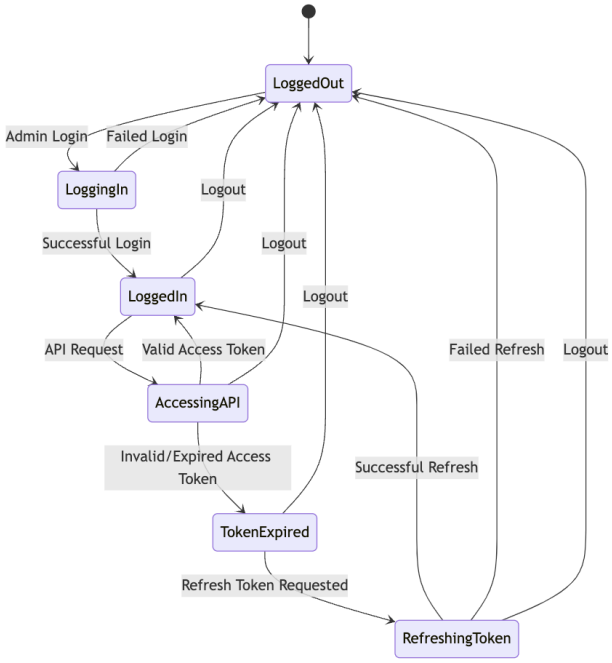
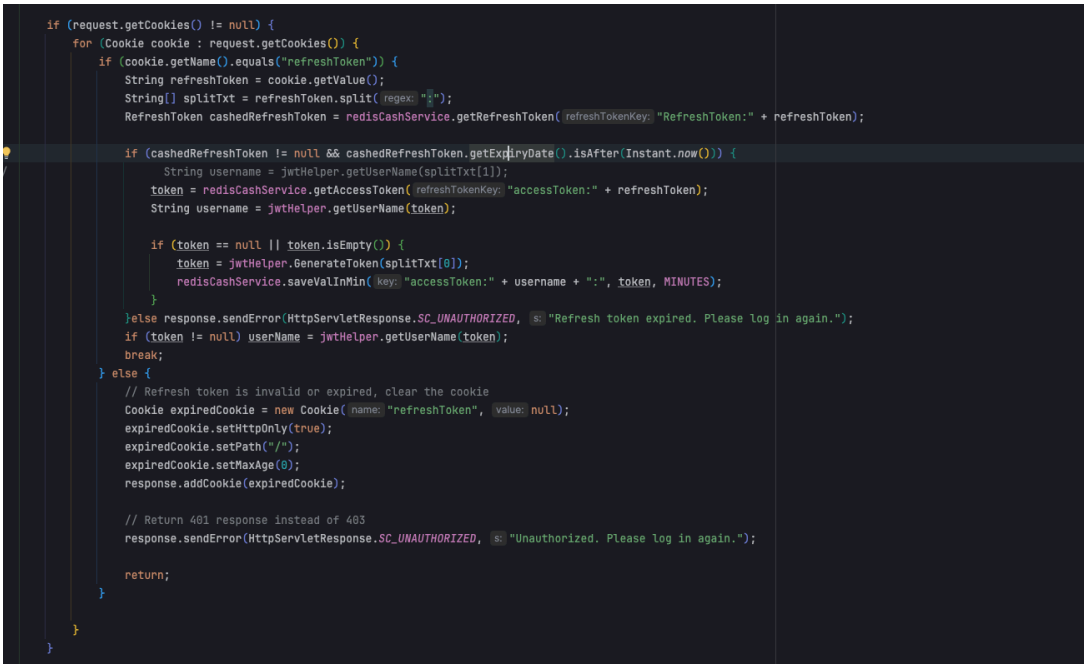


Figure 4.3.1-1: State Diagram for the Security Flow



```
if (request.getCookies() != null) {
    for (Cookie cookie : request.getCookies()) {
        if (cookie.getName().equals("refreshToken")) {
            String refreshToken = cookie.getValue();
            String[] splitTxt = refreshToken.split( regex: ":");
            RefreshToken cashedRefreshToken = redisCashService.getRefreshToken( refreshTokenKey: "RefreshToken:" + refreshToken);

            if (cashedRefreshToken != null && cashedRefreshToken.getExpiryDate().isAfter(Instant.now())) {
                  String username = jwtHelper.getUserName(splitTxt[1]);
                token = redisCashService.getAccessToken( refreshTokenKey: "accessToken:" + refreshToken);
                String username = jwtHelper.getUserName(token);

                if (token == null || token.isEmpty()) {
                    token = jwtHelper.GenerateToken(splitTxt[0]);
                    redisCashService.saveValInMin( key: "accessToken:" + username + ":", token, MINUTES);
                }
            }else response.sendError(HttpServletResponse.SC_UNAUTHORIZED, s: "Refresh token expired. Please log in again.");
            if (token != null) userName = jwtHelper.getUserName(token);
            break;
        } else {
            // Refresh token is invalid or expired, clear the cookie
            Cookie expiredCookie = new Cookie( name: "refreshToken", value: null);
            expiredCookie.setHttpOnly(true);
            expiredCookie.setPath("/");
            expiredCookie.setMaxAge(0);
            response.addCookie(expiredCookie);

            // Return 401 response instead of 403
            response.sendError(HttpServletResponse.SC_UNAUTHORIZED, s: "Unauthorized. Please log in again.");

            return;
        }

    }
}
```

Figure 4.3.1-2 Token Based Authentication Mechanism Implementation

Secure Token Handling and Redis Integration

To ensure enhanced security and avoid token leakage, the system **stores access tokens in Redis**. The process is as follows (Figure 4.3.1-1,2,4):

- When an admin logs in, a refresh token is stored in HTTP-only cookies, while the access token (Bearer Token) is securely stored in Redis.
- Every time an API request is made, the Bearer Token is retrieved from Redis and used for authentication.
- When the access token expires, the frontend sends the refresh token from the cookies to the backend.
- The backend validates the refresh token and retrieves the corresponding access token from Redis or issues a new access token.
- This approach ensures that access tokens are never persistently stored on the client side, reducing the risk of exposure.

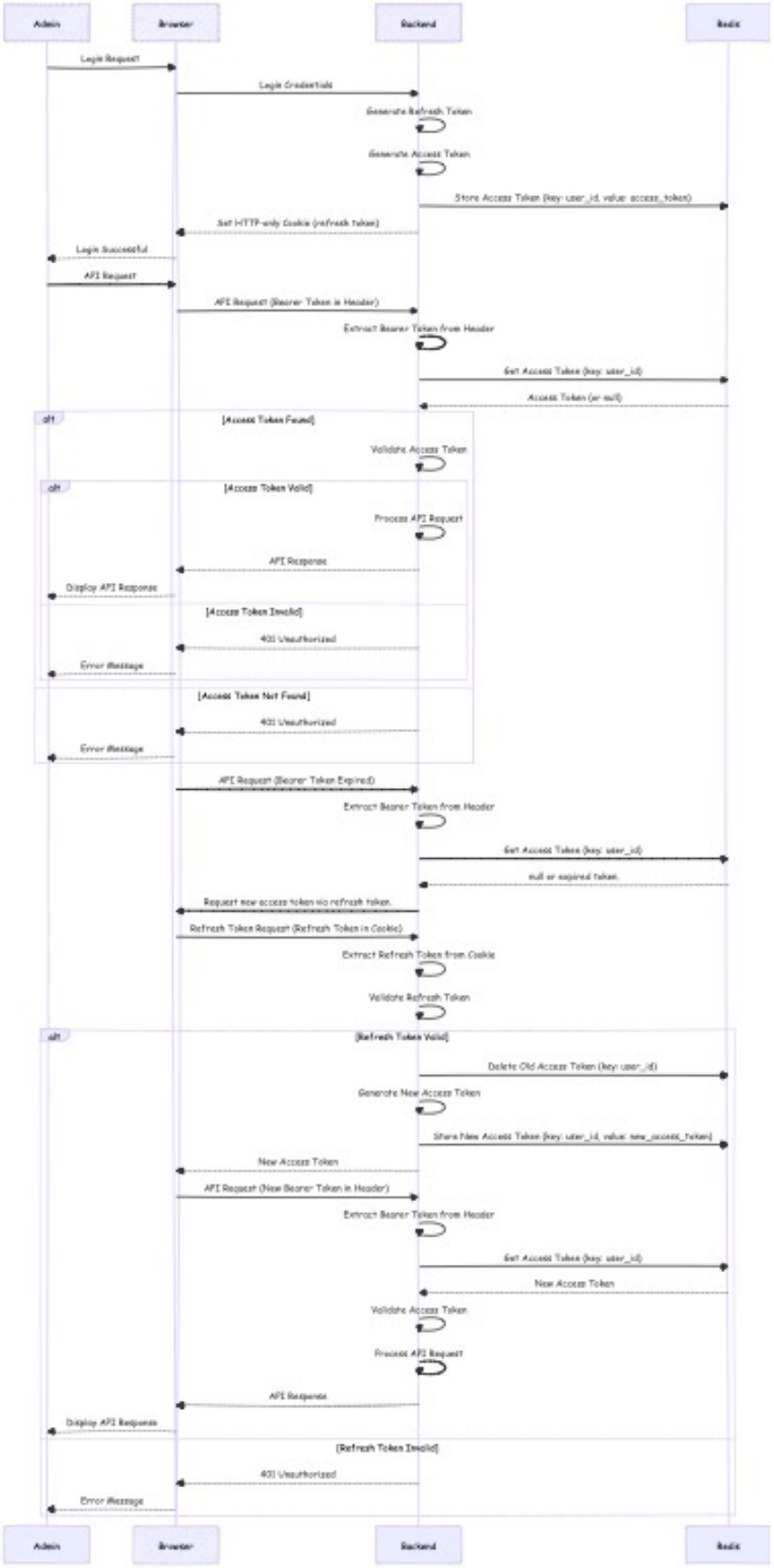


Figure 4.3.1-3 : Secure Token Lifecycle (edited using mermide.live)

The JWT (JSON Web Token) is utilized to encapsulate authentication information within a digitally signed token, ensuring secure and efficient authentication. Each JWT consists of three main components: the header, which specifies the signing algorithm (e.g., HMAC SHA256); the payload, which contains user-specific claims such as user Id (identifying the authenticated admin user), role (defining user permissions), IAT (issued at timestamp), and exp (expiration time); and the signature, which ensures token integrity and authenticity. A key security concern with JWT tokens is that they cannot be invalidated until they expire. Unlike session-based authentication, where tokens can be explicitly revoked, a valid JWT remains usable until its expiration time, even if the user logs out. To address this limitation and prevent token leakage, the system implements token removal from Redis upon logout (Figure 4.3.1 4). This ensures that even if an access token is valid, it cannot be used after logout. The logout process involves deleting the access token from Redis, thereby preventing further authentication requests, invalidating the refresh token stored in HTTP-only cookies, ensuring that the user cannot re-authenticate without logging in again, and clearing user session data to enforce a strict logout mechanism. This approach ensures that the authentication system remains secure, efficient, and resistant to unauthorized access, mitigating risks associated with token misuse.

```
@PostMapping(value = "/logout")  new *
public ResponseEntity<String> logout(HttpServletRequest request, HttpServletResponse response) throws JsonProcessingException {
    if (request.getCookies() != null) {
        for (Cookie cookie : request.getCookies()) {
            if (cookie.getName().equals("refreshToken")) {
                String refreshToken = cookie.getValue();
                String[] splitTxt = refreshToken.split( regex: ":");
                RefreshToken cashedRefreshToken = redisCashService.getRefreshToken( refreshTokenKey: "RefreshToken:" + refreshToken);
                boolean isRemoved = authorizationService.removeToken( key: "RefreshToken:" + refreshToken);
                if (!isRemoved) throw new RuntimeException("errorInRemovingRefreshToken");
                String token = redisCashService.getAccessToken( refreshTokenKey: "accessToken:" + refreshToken);
                if (token == null || token.isEmpty()) {
                    tokenBlacklistService.addToBlacklist(token);
                }
                boolean isRemovedAccessToken = authorizationService.removeToken( key: "accessToken:" + refreshToken);
            }
        }
        SecurityContextHolder.clearContext();
        // Blacklist the JWT token
        final String authorizationHeader = request.getHeader( s: "Authorization");
        if (authorizationHeader != null && authorizationHeader.startsWith("token ")) {
            String jwtToken = authorizationHeader.substring( beginIndex: 6);
            tokenBlacklistService.addToBlacklist(jwtToken);
        }
        HttpSession session = request.getSession( b: false);
        if (session != null) {
            session.invalidate();
        }
        // Remove cookies
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                cookie.setMaxAge(0);
                cookie.setValue(null);
                cookie.setPath("/");
                response.addCookie(cookie);
            }
        }
        return ResponseEntity.ok( body: "logout");
    } else return ResponseEntity.status(HttpStatus.NOT_FOUND).body("no cookies found");
}
```

Figure 4.3.1-4: Implementation of Strict logout Mechanism

4.3.2 Security Implementation for the Telegram Bot and Mini App

Security is a fundamental aspect of the Traveler Assistant Bot and its Telegram Mini App, ensuring that user interactions and data transactions remain secure, authenticated, and protected from potential threats. The system implements different security measures for the Telegram Bot and the Mini App, incorporating Webhooks, Telegram's built-in security layers, and JWT-based authentication for the Mini App.

4.3.2.1 Webhook Security

The bot uses **Telegram Webhooks** (Figure 4.3.2 1) instead of **polling,** ensuring that messages and user requests are processed in real-time with enhanced security.

- The webhook endpoint is **exposed only to Telegram's servers**, and no external sources can send requests.
- Telegram signs every request with a **secret token** to verify its authenticity.
- Requests are validated against a **predefined bot token** before being processed.

```
11  @Component  new *
12  public class TelegramWebHooksService extends TelegramWebhookBot {
13
14      @Autowired
15      private TelegramBotChats telegramBotChats;
16
17      @Value("${telegram.bot.username}")
18      private String botUsername;
19
20      @Value("${telegram.bot.token}")
21      private String botToken;
22
23      @Value("${telegram.bot.webhookPath}")
24      private String webhookPath;
25
```

Figure 4.3.2-1: Webhook Inheritance

4.3.2.2 Telegram's Built-in Security Layers

Telegram itself provides **strong security measures** to protect communication between the bot and users:

- **End-to-End Encryption (E2EE)** for messages (client-side encryption).
- **OAuth 2.0 authentication** for secure login verification when integrating with Mini Apps.
- **Built-in user verification**, ensuring that user IDs and authentication requests come from real Telegram users.
- **Anti-spoofing mechanisms** prevent unauthorized bots from impersonating the official Traveler Assistant Bot.

4.3.2.3 Secure Command Execution and Data Validation

- All user commands sent to the bot are **sanitized** to prevent **command injection attacks**.
- User input is validated using **strict parameter rules**, ensuring that **malicious payloads** cannot be injected into system responses as shown in Figure 4.3.2-2.

```
@Override  no usages  new *
public String getBotPath() { return webhookPath; }

@Override  no usages  new *
public String getBotUsername() { return botUsername; }

@Override  no usages  new *
public String getBotToken() { return botToken; }

@Override  1 usage  new *
public void onRegister() { super.onRegister(); }

@Override  1 usage  new *
public BotApiMethod<?> onWebhookUpdateReceived(Update update) {

    // switch case to catch the incoming message
    if (update.hasMessage() || update.hasCallbackQuery()) {
        Long chatId = null;
        if (update.hasMessage()) {
            chatId = update.getMessage().getChatId();
        } else if (update.hasCallbackQuery()) {
            chatId = update.getCallbackQuery().getFrom().getId();
        }
        String text = "";
        if (update.getMessage() != null) {
            text = update.getMessage().getText();
            if (text == null) {
                text = "/";
            }
        }
        else if (update.getCallbackQuery() != null) {
            text = update.getCallbackQuery().getData();
        }else if (update.getMessage().getLocation() !=null){
            System.out.println("location: "+ update.getMessage().getLocation());
        }
        assert chatId != null;
        switch (text.toLowerCase()) {
            case "/start":
```

Figure 4.3.2-2: Integration of Webhook Communication service

4.3.2.4 Security in the Telegram Mini App

Unlike the Telegram Bot, the Mini App is a web application embedded inside Telegram, which requires additional security measures for authentication and authorization (Figure 4.3.2-4).

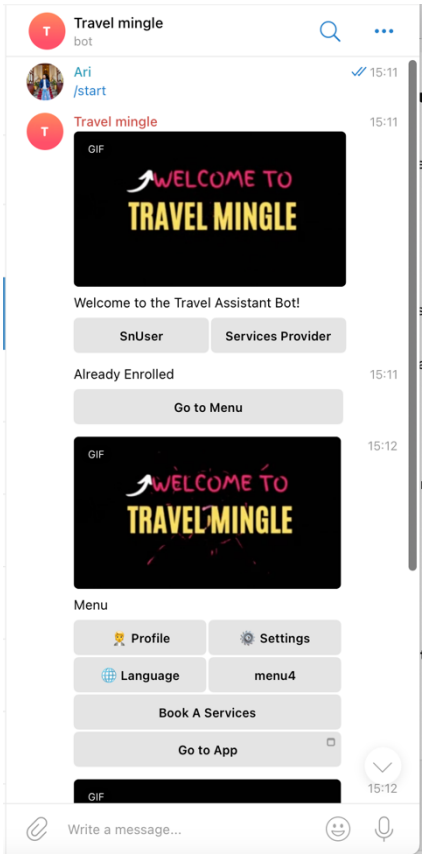


Figure 4.3.2-3: Bot UI view

Telegram OAuth Authentication (First Layer)

- The Mini App does not require traditional login credentials.
- Telegram handles authentication using OAuth, where it securely verifies the user identity before launching the app.
- Upon login, Telegram sends a signed authentication payload, which is verified on the backend.

JWT Token-Based Authentication and Authorization (Second Layer see -Figure 4.3.2-7**)**

- After verifying the Telegram authentication payload, the backend generates a JWT token.
- This token is used for API authentication when the Mini App interacts with Taraval Assistant App backend services.
- The token is stateless, reducing session handling complexities.

Unlike the Admin Panel, which implements Role-Based Access Control (RBAC), the Telegram Mini App follows a simplified authorization model based on user authentication and user type. The Mini App (Figure 4.3.2 3) has two types of users:

1. **General Users** – Regular travelers who can browse services, make bookings, and interact with service providers.
2. **Service Providers** – Verified users who can manage their service listings, receive bookings, and handle customer interactions.

Authorization Model

The Mini App authorizes users based on their JWT authentication status and user type (Figure 4.3.2-5), ensuring that each user can only access relevant functionalities. JWT Authentication: Every user must have a valid JWT token (Figure 4.3.2-6*)*, which is issued after Telegram OAuth authentication and stored securely in local mini app storage.

- **General Users**:
    - Can browse and search for available services.
    - Can initiate service bookings.
    - Can interact with service providers through the platform.

- **Service Providers**:
    - Can create, modify, and manage only their own service listings.
    - Cannot modify or access other providers' data.
    - Can view bookings related to their services but not bookings associated with other providers.

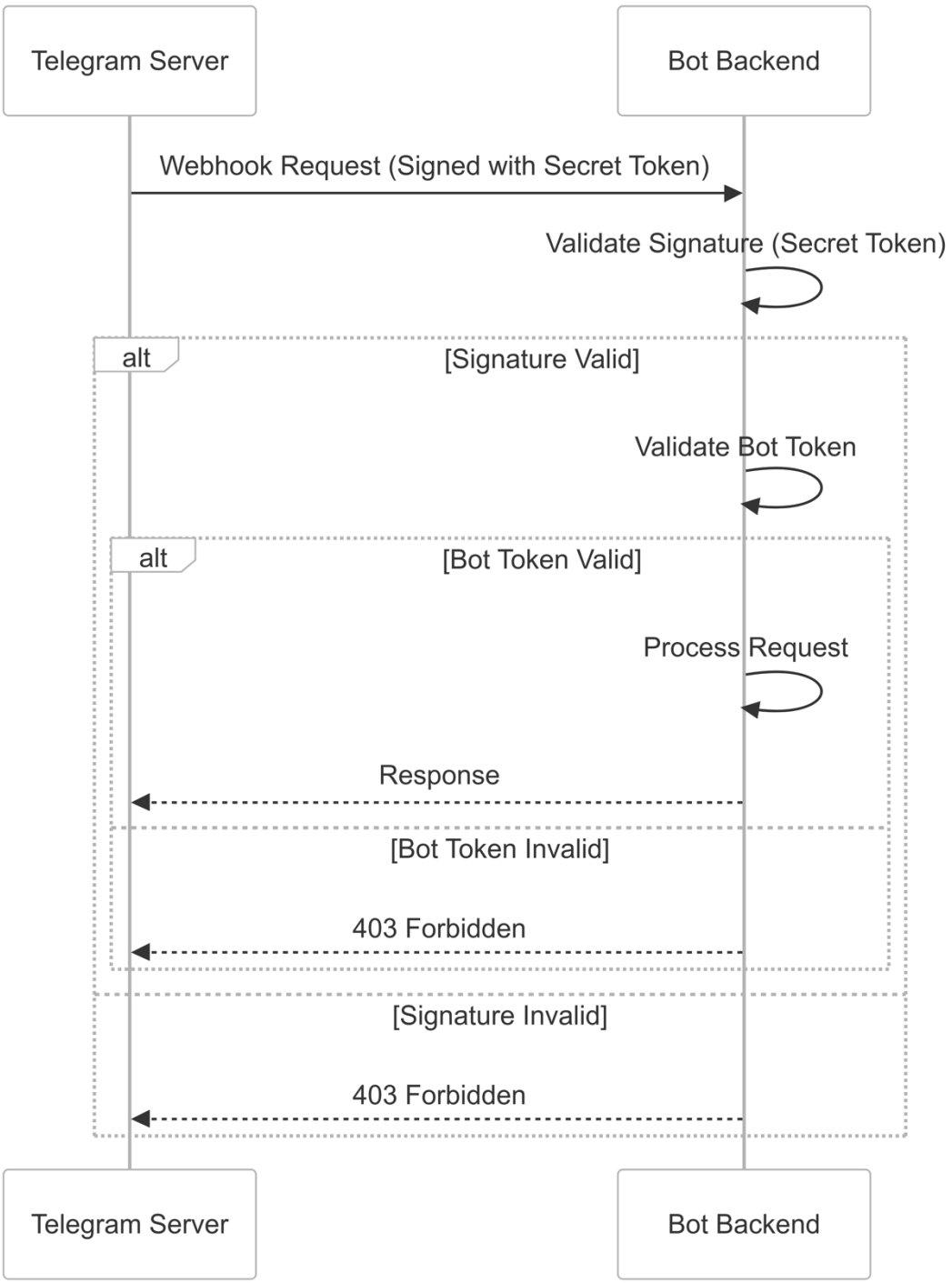
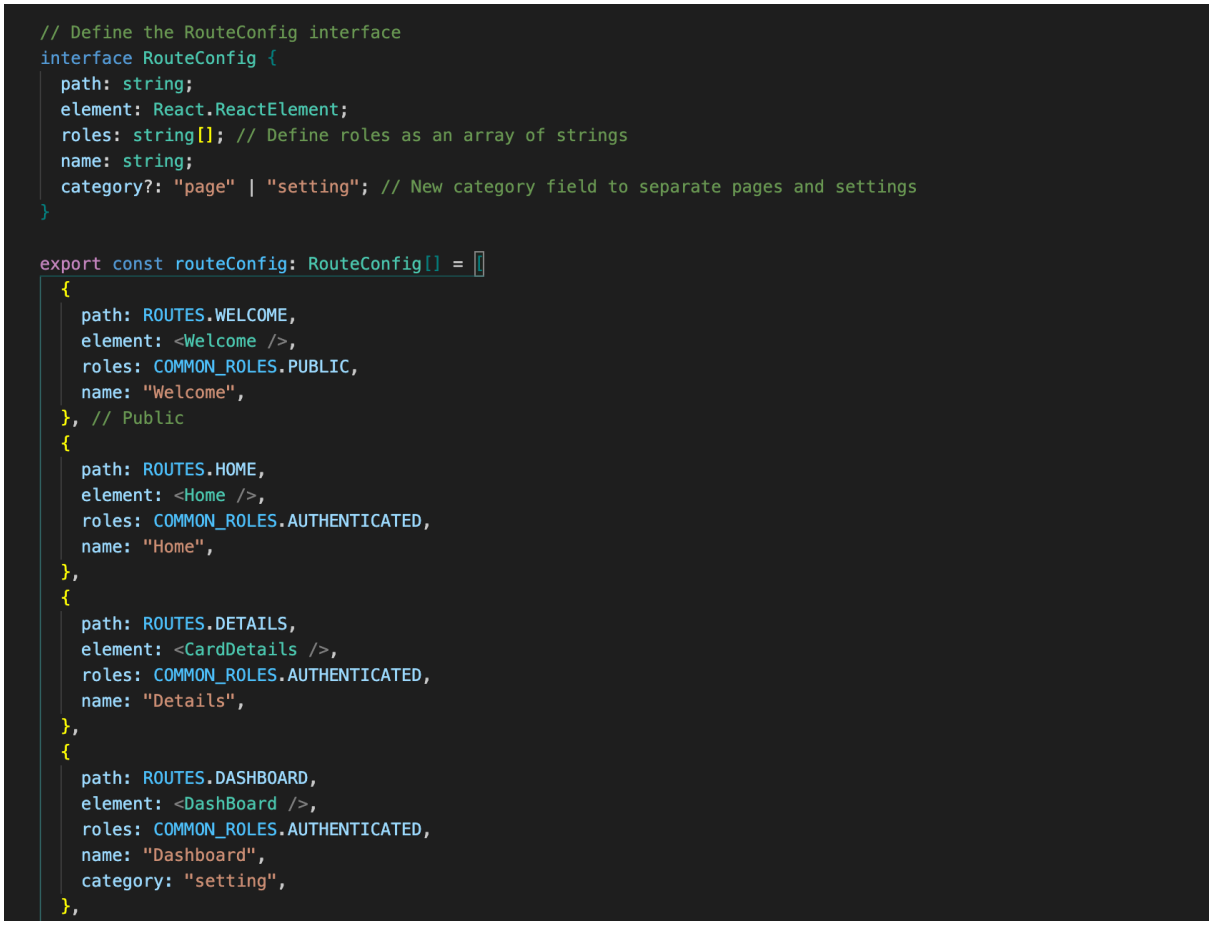


Figure 4.3.2-4: Sequence Diagram for bot's functions



```
// Define the RouteConfig interface
interface RouteConfig {
  path: string;
  element: React.ReactElement;
  roles: string[]; // Define roles as an array of strings
  name: string;
  category?: "page" | "setting"; // New category field to separate pages and settings
}

export const routeConfig: RouteConfig[] = [
  {
    path: ROUTES.WELCOME,
    element: <Welcome />,
    roles: COMMON_ROLES.PUBLIC,
    name: "Welcome",
  }, // Public
  {
    path: ROUTES.HOME,
    element: <Home />,
    roles: COMMON_ROLES.AUTHENTICATED,
    name: "Home",
  },
  {
    path: ROUTES.DETAILS,
    element: <CardDetails />,
    roles: COMMON_ROLES.AUTHENTICATED,
    name: "Details",
  },
  {
    path: ROUTES.DASHBOARD,
    element: <DashBoard />,
    roles: COMMON_ROLES.AUTHENTICATED,
    name: "Dashboard",
    category: "setting",
  },
```

Figure 4.3.2-5: Mini App Front End Authorization

```
@Component  new *
public class CustomAuthenticationEntryPoint implements AuthenticationEntryPoint {
    @Override  no usages  new *
    public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {
        response.setContentType("application/json");
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        response.getWriter().write( s: "{\"error\": \"Unauthorized\", \"message\": \"Invalid or missing token\"}");
    }
}
```

Figure 4.3.2-6: Common Authentication handler

```
@Component  new *
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    @Override  no usages  new *
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        String authHeader = request.getHeader( s: "Authorization");
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            String token = authHeader.substring( beginIndex: 7);
            try {
                Claims claims = JwtUtil.validateToken(token);
                long telegramId = Long.parseLong(claims.get("telegramId").toString());
                String role = claims.get("userType").toString();

                SimpleGrantedAuthority authority = new SimpleGrantedAuthority( role: "ROLE_"+role.toUpperCase());
                UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(telegramId, credentials: null, Collections.singletonList(authority));
                SecurityContextHolder.getContext().setAuthentication(authentication);
            } catch (Exception e) {
                response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                return;
            }
        }
        filterChain.doFilter(request, response);
    }
```

Figure 4.3.2-7: Mini app Security Filter

In the Traveler Assistant Bot system, CORS (Cross-Origin Resource Sharing) is managed through backend configurations to enforce controlled access between the frontend applications and backend services. The backend explicitly defines allowed origins, restricting API requests to those originating from the authorized domains of the Mini App and Admin Panel. Additionally, permitted HTTP methods such as GET, POST, PUT, and DELETE are specified to ensure that only necessary operations can be executed. The server also enforces allowed headers, including Authorization, to securely transmit JWT tokens for authentication. To further enhance security, credentials support is enabled, allowing the frontend to send authenticated requests while ensuring that sensitive cookies, such as refresh tokens, are transmitted only under secure conditions. These settings are implemented within the Spring Boot backend using global CORS configuration (Figure 4.3.2-8), ensuring structured and secure communication between system components while mitigating unauthorized cross-origin requests.

```
@Bean  new *
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration corsConfiguration = new CorsConfiguration();
    corsConfiguration.setAllowedOrigins(
            Arrays.asList(
                    "http://localhost:5175", webAppUrl,
                    "https://**",
                    webhookUrl,
                    "https://*.telegram.org",
                    "https://*.telegram.me"
            ));
    corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"));
    corsConfiguration.setAllowedHeaders(List.of( e1: "*"));
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration( pattern: "/**", corsConfiguration);
    return source;
}
```

Figure 4.3.2-8: CORS Policy handling method

4.3.3 Secure Payment Integration

The Traveler Assistant Bot integrates PayHere as its payment gateway while implementing MD5 hashing for transaction validation and security. The system generates a secure hash using the merchant ID, order ID, formatted payment amount, currency, and a hashed merchant secret, ensuring that payment requests remain unaltered and verifiable. The MD5 hash function is implemented using Java's MessageDigest API, where the input string is processed to generate a 128-bit hash value. The resulting hash is converted to hexadecimal format and padded to ensure a consistent 32-character length. This cryptographic hash is included in the payment request sent to PayHere, allowing the gateway to verify transaction integrity before processing

payments. Upon receiving an Instant Payment Notification (IPN) from PayHere, the backend recalculates the hash using the same logic and compares it with the received hash to ensure data consistency and prevent tampering or unauthorized modifications. This prevents transaction replay attacks and ensures that only valid payment notifications are processed. Additionally, the system restricts IPN handling to requests from PayHere's whitelisted IP addresses, enhancing protection against spoofed transactions. All communication between the frontend, backend, and PayHere APIs is secured using HTTPS encryption, preventing man-in-the-middle (MITM) attacks and unauthorized access to payment data. Furthermore, rate limiting mechanisms are enforced to mitigate brute-force attacks, and JWT authentication ensures that only authorized users can initiate and verify transactions and each successful transaction is updated and stored successfully. By implementing MD5 hashing for transaction security, strict IP validation, and secure API communication, the system ensures reliable, tamper-proof payment processing while minimizing fraud risks (Figure 4.3.3 1).

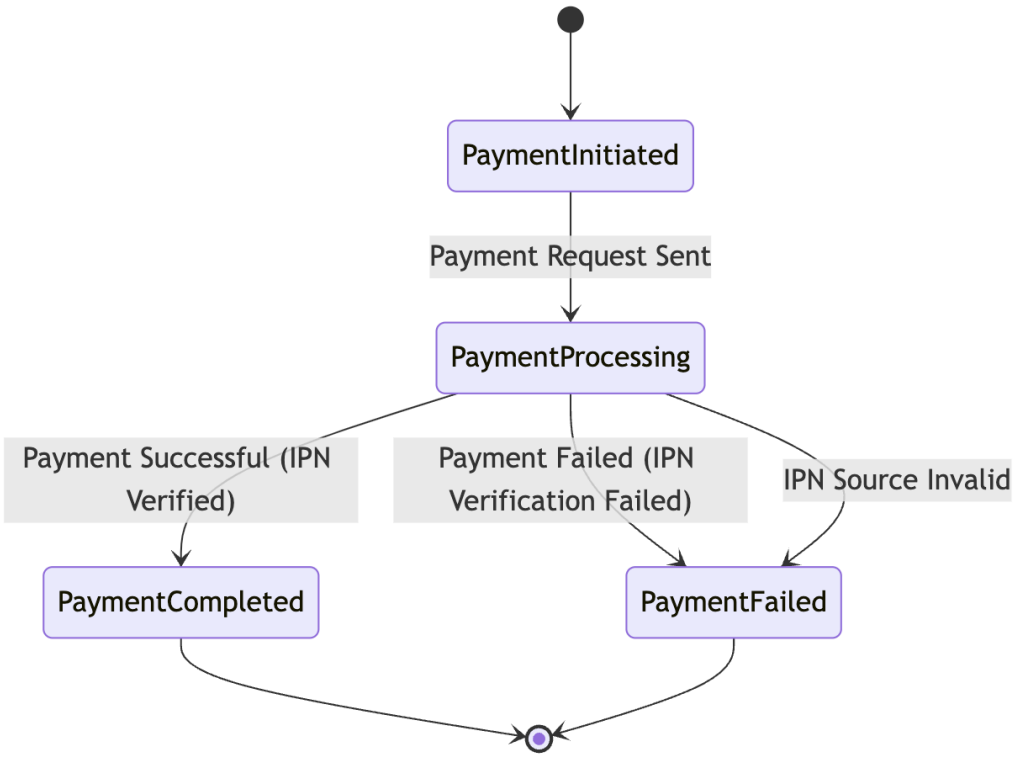


Figure 4.3.3-1: Secure Payment Flow Diagram

4.4 Technologies and Tools Used in the Development of the Traveler Assistant Bot System

The Traveler Assistant Bot system is developed using a combination of modern technologies to ensure efficient, scalable, and maintainable software development. The backend is built with

Spring Boot, developed using IntelliJ IDEA, which provides a comprehensive development environment with advanced debugging, code analysis, and integration support for Spring Boot projects. The project is managed using Maven, a build automation tool that handles dependency management, compilation, testing, and packaging of the Java-based application. The backend is implemented using Java 17, which offers enhanced performance, security improvements, and long-term support (LTS), ensuring the stability and reliability of the application.

For the frontend, the system uses React.js, a widely adopted JavaScript library for building dynamic and interactive user interfaces. The frontend communicates with the backend using Axios, a promise-based HTTP client that simplifies API requests, error handling, and response transformations. To ensure version control and collaborative development, the project is managed using Git, a distributed version control system, allowing efficient tracking of code changes, branching, and repository management. The combination of these technologies enhances the development workflow, enabling efficient backend development with Spring Boot, optimized build processes with Maven, modern UI (Figure 4.3.2-3, Figure 4.3.3-1, Figure 4.3.3-2 and Figure 4.3.3-3) design with React, and API communication using Axios, while Git ensures structured version control and collaboration across the development lifecycle.

4.5 Summary

The Implementation chapter provides a comprehensive overview of the development process of the Traveler Assistant Bot system, covering both backend and frontend implementations, database design, security measures, payment integration, and deployment strategies. The backend is developed using Spring Boot with Java 17, leveraging JWT-based authentication, Redis caching, and MySQL for data storage, while the frontend comprises two separate applications: the Telegram Mini App and the Admin Panel, both implemented using React.js and TypeScript.

The system follows a RESTful API architecture, with secure communication between components managed through CORS policies, HTTPS encryption, and token-based authentication. The Mini App integrates directly with Telegram OAuth for user authentication, while the Admin Panel enforces Role-Based Access Control (RBAC) to restrict access to authorized users only. MD5 hashing is used for transaction security in PayHere integration, ensuring the integrity of payment data, and Instant Payment Notifications (IPN) are validated to prevent fraudulent transactions. Additionally, webhooks are employed for real-time

interaction with Telegram's messaging API, providing a dynamic and responsive experience for users.

For system security, various measures are implemented, including password hashing, API rate limiting, session validation, and restricted access controls

The implementation phase focused on scalability, security, and efficiency, integrating modern technologies and best practices to build a robust and reliable system. Challenges encountered during development, such as optimizing API performance and securing third-party integrations, were addressed through code optimizations, caching strategies, and strict authentication mechanisms. Future improvements may include enhancing AI-based service recommendations, expanding payment options, and further optimizing database performance. This chapter serves as a detailed technical reference for understanding the system's architecture, development methodologies, and security strategies implemented in the Traveler Assistant Bot (Figure 4.3.3 1).

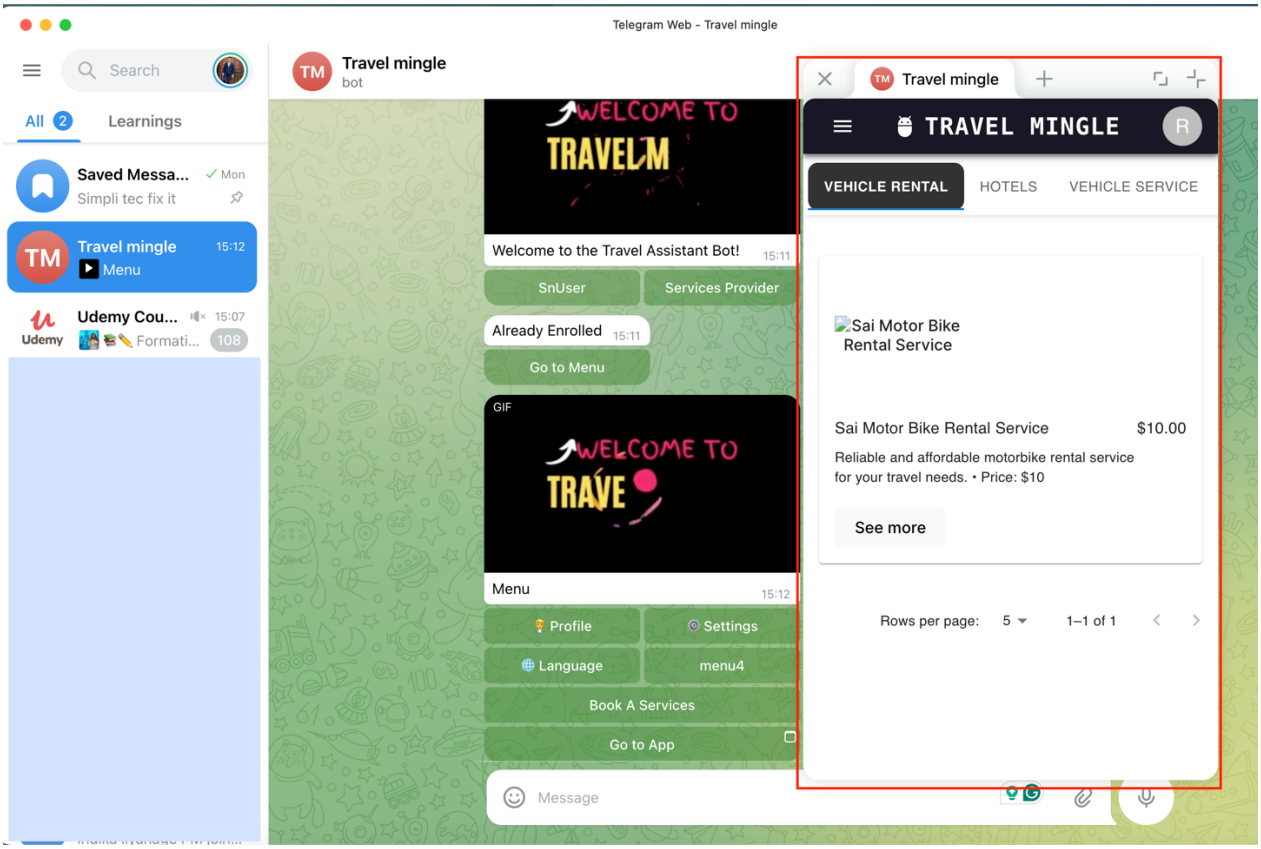


Figure 4.3.3-1: Bot Application Client

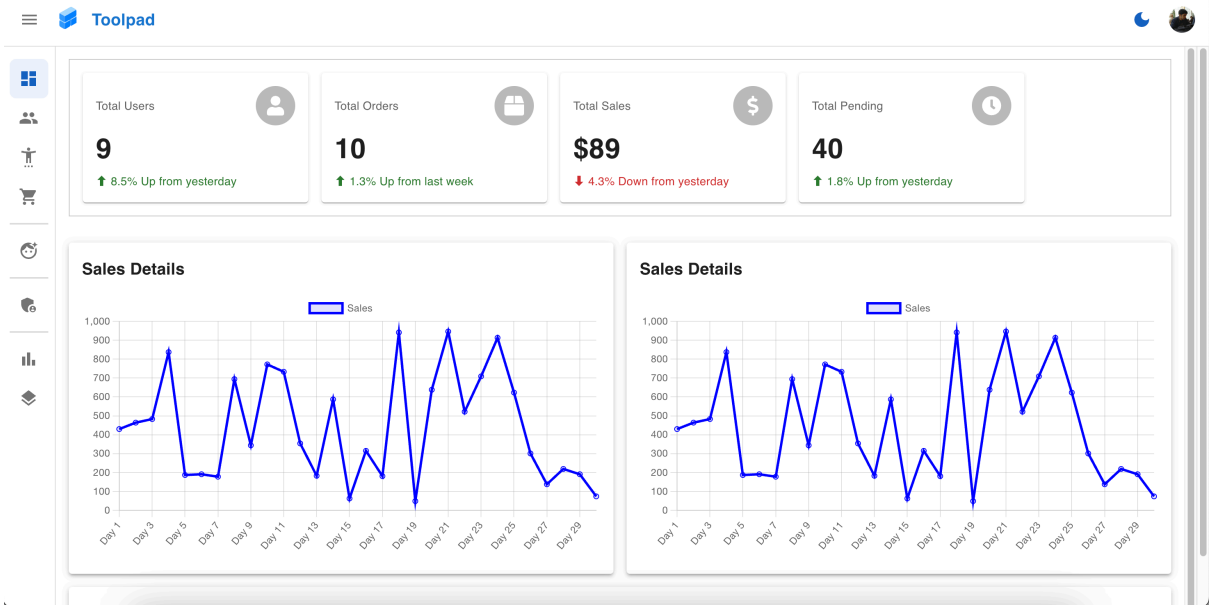


Figure 4.3.3-2: Admin Pannel Dashboard View of Travel Assistant Application

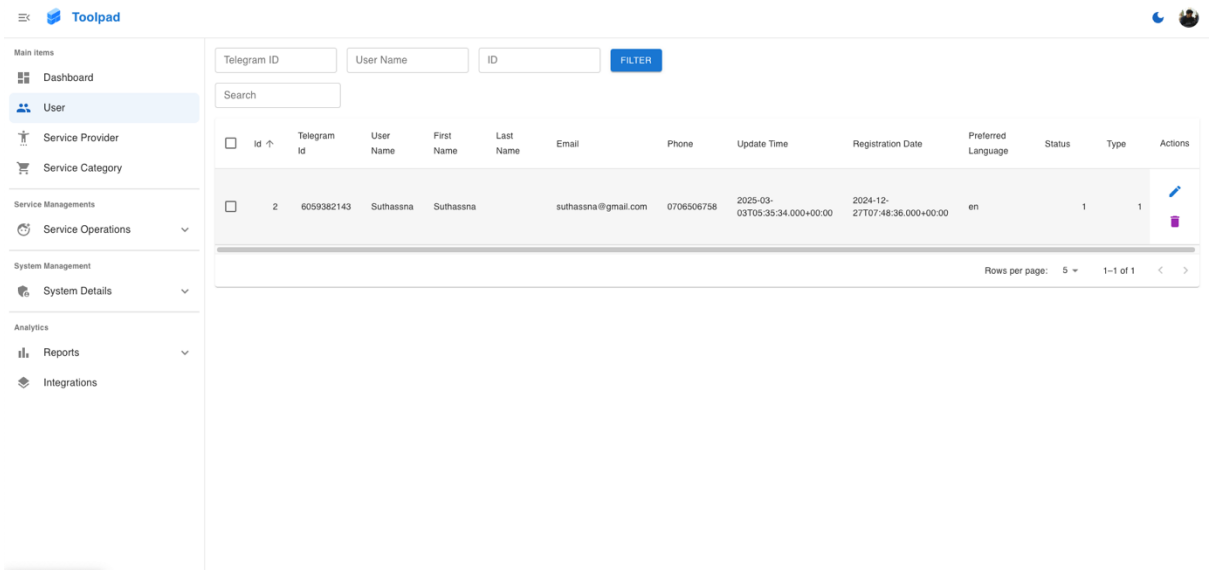


Figure 4.3.3-3: Admin Functional dash board

5 Chapter 5 – Testing and Evaluation

5.1 Introduction

The Testing and Evaluation phase is a critical aspect of the Traveler Assistant Bot system, ensuring that all components function correctly, meet performance expectations, and adhere to security standards. This chapter outlines the testing methodologies used to validate system functionality, including API testing, integration testing, system testing, security testing, and usability evaluation. The primary objective is to identify and resolve potential issues before deployment, ensuring that the system provides reliable, secure, and efficient services to its users, including travelers, service providers, and administrators.

Various testing approaches were employed to examine individual software modules, their integration, and the overall system performance. unit testing focused on verifying the correctness of isolated components, while integration testing assessed interactions between different modules, such as frontend-backend communication and payment gateway integration. System testing evaluated end-to-end functionality, ensuring that the platform meets the defined requirements. Additionally, security testing was conducted to identify vulnerabilities in authentication, authorization, payment processing, and data protection mechanisms.

Furthermore, performance testing was performed to measure the system's ability to handle multiple concurrent requests, ensuring that response times remain within acceptable limits. User evaluation was also carried out to assess usability and gather feedback on system interaction and accessibility. The findings from these evaluations provide insights into potential improvements and future enhancements, ensuring that the Traveler Assistant Bot system operates as expected in real-world scenarios.

5.2 Software Testing Types

The proposed system was examined in two categories throughout this testing procedure as follows.

1. Functional testing approach
2. Non-functional testing approach (Performance Testing approach)

5.2.1 Functional testing approach

Following tests are covered under functional testing approach.

1. Unit Testing
2. Integrating Testing
3. System Testing

5.2.1.1 Unit Testing

Unit testing was conducted to verify the correctness and reliability of individual components within the backend, frontend, and database layers. The Traveler Assistant Bot system uses Mockito and JUnit for backend unit testing and Jest for frontend component testing. The primary objective of unit testing was to ensure that each function or module works independently before integrating it with other system components.

For the backend, unit tests were written to validate authentication, service booking, payment handling, and data processing. The Spring Boot framework was used in conjunction with Mockito to simulate dependencies and test individual methods in isolation.

- Authentication Tests: Verified JWT token generation, validation, and expiration.
- Service Booking Tests: Ensured that users could only book available services.
- Payment Processing Tests: Checked MD5 hash generation and PayHere transaction handling.

5.2.1.2 API Testing

API testing was conducted to **validate the functionality, security, and reliability** of the **Traveler Assistant Bot** system's backend services. The **Admin Panel API** and **Bot API** were tested using **Postman** to ensure that all endpoints responded correctly under different conditions, including valid and invalid requests. The primary focus of API testing was to verify authentication mechanisms, service management, booking processes, payment handling, and security enforcement.

5.2.1.2.1 API Testing Approach

API testing was performed using **Postman** to simulate various HTTP requests to the backend, checking for correct responses, error handling, and security mechanisms. The tests covered:

- Authentication and Authorization: Ensuring JWT token-based authentication works correctly.
- Admin API Testing: Verifying admin access control, service management, and dashboard functionalities.
- Bot API Testing: Testing user authentication, service booking, and Telegram Bot interactions.
- Payment Processing: Ensuring PayHere integration correctly verifies transactions and updates booking statuses.

- Security Enforcement: Testing API protection against unauthorized access, SQL injection, and other security threats.

5.2.1.2.2 Admin Dashboard API Testing

The Admin Dashboard API in the Traveler Assistant Bot system was tested to ensure that administrators could access system analytics, manage users, services, transactions, and enforce role-based access control (RBAC). API testing (Figure 5.2.1 1) was conducted using Postman to validate the correctness of responses, error handling, and security enforcement.

The primary objectives of testing included:

- Ensuring that only authenticated administrators could access the dashboard.
- Verifying that unauthorized user received 403 Forbidden responses.
- Checking that key system statistics were retrieved correctly.
- Validating that the API performed efficiently under high loads.

5.2.1.2.3 Security and Role-Based Access Control (RBAC) Testing (Table 5.2.1 1)

1. The Admin Dashboard API enforces RBAC to prevent unauthorized access. The following security aspects were tested:
2. Ensuring that only users with the “ADMIN” role could access the dashboard.
3. Attempting to access admin-only functionalities with a regular user token.
4. Testing for token manipulation attacks.

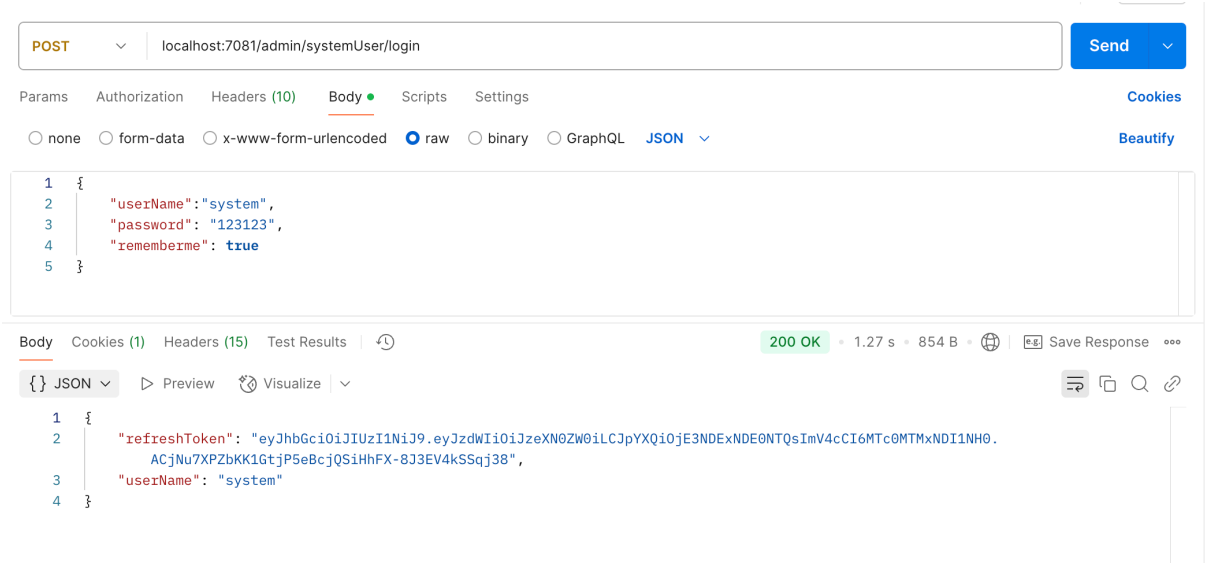


Figure 5.2.1-1: System User Authentication Api Testing

Table 5.2.1-1: Authentication Based Test Summary

Test Scenario	Expected Result	Status
Admin accesses dashboard with valid token	Dashboard data retrieved successfully	Passed
Traveler attempts to access admin dashboard	Access Denied (403 Forbidden)	Passed
Missing or expired token	Unauthorized Access (401)	Passed
Large number of concurrent requests	Response time < 200ms	Passed
SQL Injection attempt in query parameters	API rejects invalid input	Passed

5.2.1.2.4 Bot API Testing

The Bot API in the Traveler Assistant Bot system was tested using Postman to ensure that the API endpoints function correctly, handle requests efficiently, enforce security policies, and respond appropriately under various conditions. The tests focused on user authentication, service retrieval, booking management, payment processing, and security validation.

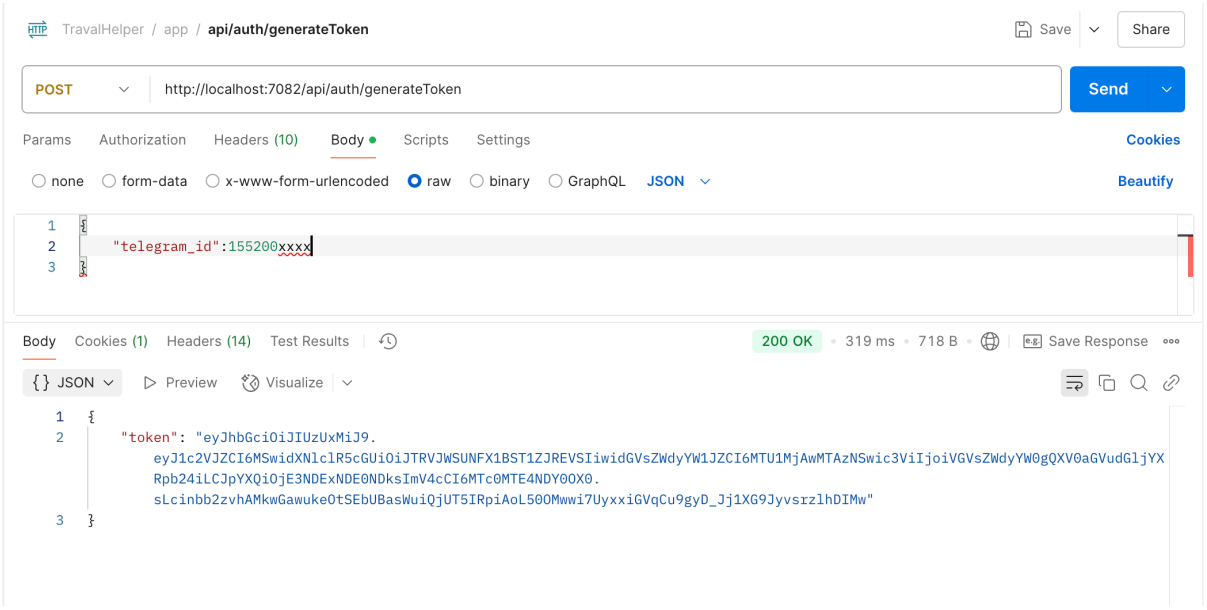


Figure 5.2.1-2: Mini App 2nd Layer Authentication API Testing

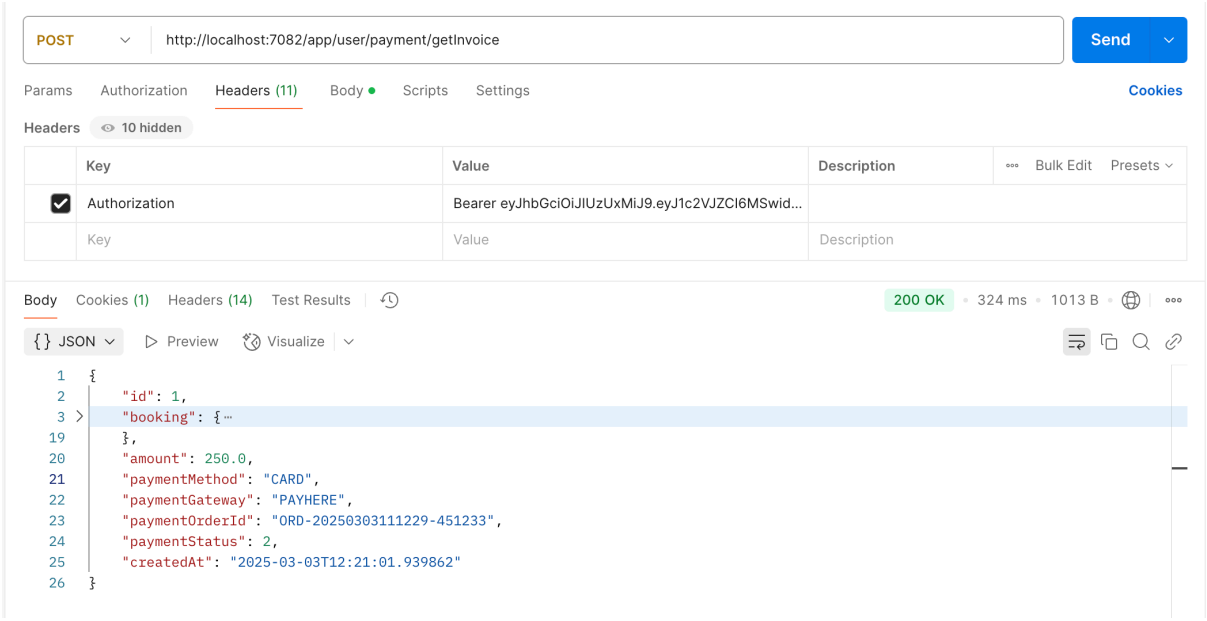


Figure 5.2.1-3: Payment Integration Testing

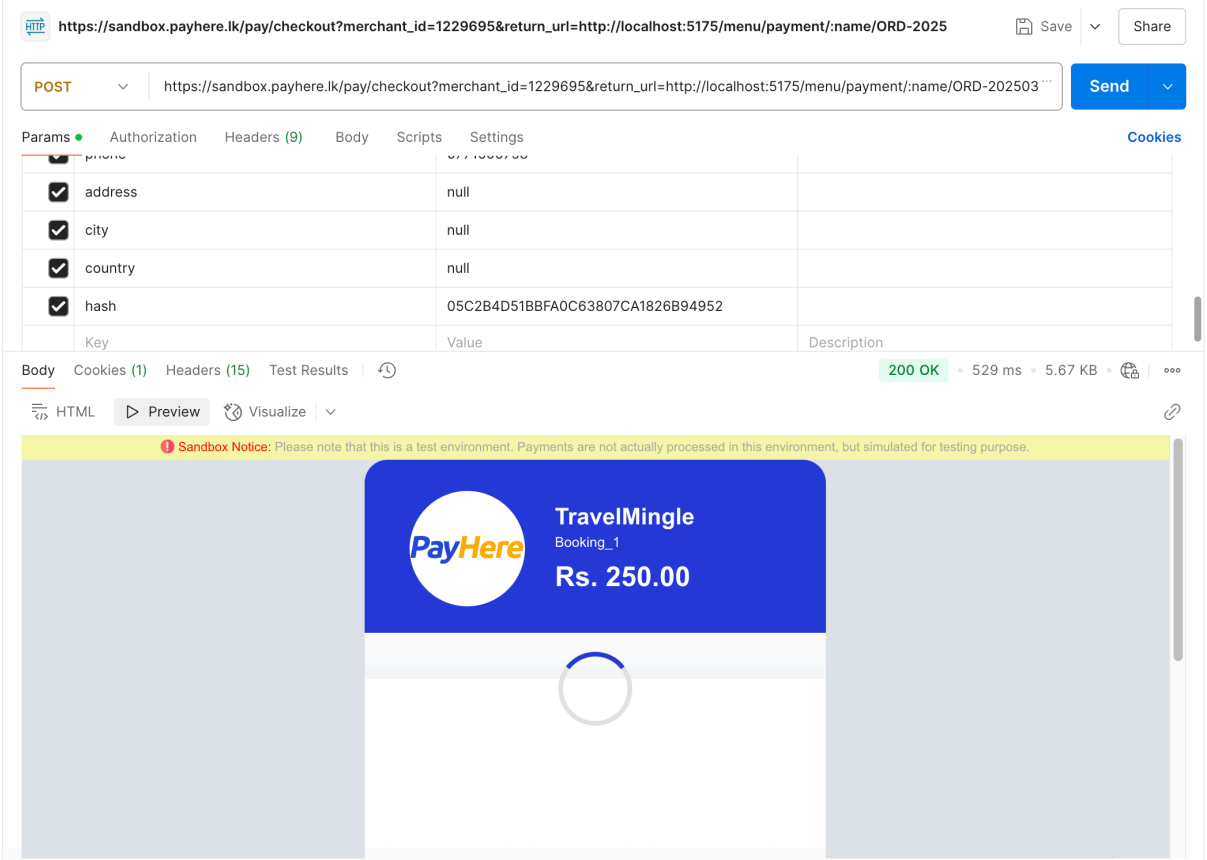


Figure 5.2.1-4: PayHere Payment Gateway Integration

Table 5.2.1-2: Table of Bot and mini-App test Summary

Test Scenario	Expected Result	Status
User login via Telegram OAuth	Authentication successful	Passed
Fetch available services	Returns service list	Passed
Book a service	Booking confirmed	Passed
Attempt to book an unavailable service	Error returned	Passed
Process a valid PayHere payment	Payment successful	Passed
Attempt fraudulent payment	Payment rejected	Passed
Unauthorized API access	Access denied	Passed
SQL Injection attempt	Request rejected	Passed

5.2.1.3 Integration Testing

Integration testing focused on validating interactions between system components, including the Telegram bot, backend APIs, admin panel, and third-party services (e.g., Payhere, Redis, Telegram Api etc.). The goal was to ensure contentious data flow and error handling across modules.

5.2.1.3.1 Integration Testing Strategy

- Scope:
    - Telegram Bot ↔ Backend API (Spring Boot).
    - Mini App (React) ↔ Telegram Bot ↔ Backend API (Spring Boot).
    - Backend Services ↔ MySQL Database & Redis Cache.
    - Admin Panel (React) ↔ Backend APIs
    - Payment Gateway (PayHere) ↔ Payment Calculation Logic (Figure 5.2.1 6).
    - Layered Architecture (Controller ↔ Service ↔ Repository).

Table 5.2.1-3: Summary of Integration Testing Results

Test Scenario	Components Tested	Status
User login via Telegram OAuth	Mini App ↔ Backend	Passed
Service booking updates database	Mini App ↔ Backend ↔ MySQL	Passed
Admin dashboard access control	Admin Panel ↔ Backend	Passed
Secure PayHere transaction processing	Backend ↔ PayHere API	Passed
Service caching in Redis	Backend ↔ Redis	Passed

Integration testing validated that all system components communicate effectively, including the Mini App, Admin Panel, backend, payment gateway, and database. API requests were processed correctly, security mechanisms were enforced, and data consistency was maintained across MySQL and Redis. The PayHere payment verification process was successfully tested, ensuring secure financial transactions. These tests confirmed that the system is functional, secure, and performs as expected in real-world conditions.

5.2.2 System Testing

System Testing evaluated the Traveler Assistant Bot as a fully integrated application to validate compliance with functional and non-functional requirements (Section 2.3). This phase focused on end-to-end workflows, performance under realistic conditions, and alignment with user expectations.

This testing phase aimed to verify that all components, including the Mini App, Admin Panel, Backend, Database, and External Integrations (Telegram API & PayHere API), work together correctly in a real-world environment. The goal was to ensure that the system behaves as expected under various conditions, including normal usage, edge cases, and high traffic loads.

Objectives of System Testing

The main objectives of system testing were to:

- Validate that the system meets all functional requirements.
- Ensure that all modules and components work together as expected.
- Test the end-to-end workflows, including user registration, service browsing, booking, payments, and admin management.
- Assess system stability, security, and performance under different conditions.
- Identify and resolve any bugs, inconsistencies, or usability issues before deployment.

Test Environment

- Backend: Spring Boot v3.0 (Java 17) with Hibernate/JPA.
- Frontend: Telegram Mini App (React.js) + Admin Panel (TypeScript).
- Database: MySQL v8.0.
- Caching: Redis v7.2.4 for session management and translations.
- Payment Gateway: Payhere Sandbox API.
- Tools: JMeter (load testing), Mokito

Table 5.2.2-1 :Summary of System Testing

Test Scenario	Expected Result	Status
User Registration and Authentication	User logs in and accesses dashboard	Passed
Booking Service	Booking stored in database, confirmation sent	Passed
Payment Processing via PayHere	Payment verified, booking status updated	Passed
Admin Dashboard Access	Admin retrieves user and transaction data	Passed
Unauthorized API Access	System blocks request with 401 error	Passed
SQL Injection Attempt	System rejects invalid input	Passed
UI/UX Evaluation	Users find app intuitive	Passed

System testing validated the functionality, security, performance, and usability of the Traveler Assistant Bot. The system was tested under various real-world scenarios, confirming that authentication, service booking, payments, and admin functionalities worked as intended. Security mechanisms effectively prevented unauthorized access and input manipulation. Usability testing provided insights for future improvements. Overall, system testing confirmed that the Traveler Assistant Bot is fully functional, secure, and ready for deployment.

5.3 Summary

The Testing and Evaluation phase ensured that the Traveler Assistant Bot system met all functional, security, and integration requirements. Various testing methodologies, including unit testing, API testing, integration testing, and system testing, were conducted to validate the correctness of individual components and their interactions.

Unit testing was performed on critical backend and frontend modules to verify authentication mechanisms, service booking, payment processing, and database operations. Automated tests using JUnit for Spring Boot confirmed that individual functions operated as expected, focusing on error handling and security validation.

API testing was conducted using Postman, covering all major endpoints for authentication, booking, payments, and admin functionalities. Tests ensured that API requests were processed correctly, responses were returned in the expected format, and security protections like token-based authentication and input validation were enforced.

Integration testing validated the interactions between system components, including the Mini App, Admin Panel, Backend, Payment Gateway (PayHere), and Database. These tests ensured that user requests were correctly processed, booking data was stored and updated in the database, and payment verification was securely handled. Special attention was given to authentication flows, data consistency between MySQL and Redis, and external API interactions to ensure system reliability.

System testing was conducted to evaluate the platform's overall functionality, stability, and security. Tests confirmed that all system workflows, including registration, service browsing, booking, payments, and admin management, worked correctly under different scenarios. Security measures were validated through unauthorized access testing, SQL injection prevention, and token validation, ensuring that only authorized users could access protected resources.

Overall, the Traveler Assistant Bot system was successfully validated through extensive testing, demonstrating its functionality, security, and stability. Identified issues were resolved, and optimizations were implemented to enhance system efficiency. The system is now ready for deployment, meeting all technical and business requirements.

6 Chapter 6 – Conclusion

The Traveler Assistant Bot for Sri Lankan Tourism project addressed the critical challenges faced by travelers in accessing reliable local services, as outlined in Chapter 1. By leveraging Telegram's platform, the system eliminates the need for app downloads and streamlines service discovery, booking, and payment processes. This chapter summarizes the project's achievements, reflects on challenges overcome, and proposes future enhancements.

The system meets its core objectives by offering a streamlined service booking process through Telegram, allowing travelers to search, filter, and book accommodations, tour guides, and vehicle rentals etc. in real-time. Secure payment processing was integrated via PayHere, ensuring compliance with financial regulations while automating transaction tracking. Additionally, cash payment handling was implemented with an automated, enhancing transparency in financial transactions. A React-based admin panel was developed to enable administrators to oversee user accounts, process withdrawal approvals, and monitor fraudulent activities, ensuring system integrity and security.

Several challenges were addressed throughout the development process. PayHere API latency initially caused delays in payment confirmations, which were mitigated by optimizing API call retries and implementing asynchronous transaction logging. Fee calculation errors due to floating-point rounding discrepancies were resolved by incorporating Java's BigDecimal, ensuring precise financial transactions. The Telegram bot's scalability was improved by implementing hooks-based request throttling, which prevents rate-limiting issues under high user loads.

While the current system meets its fundamental objectives, future enhancements will further improve its usability and scalability. Multilingual support will be integrated, allowing users to access services in Sinhala, Tamil, and English. The referral and commission system will be developed to enable service providers to recommend trusted partners and receive automated commission payments. Expanding PayHere's capabilities to support subscription billing will allow recurring payments for long-term services, such as extended vehicle rentals. Additionally, AI-driven fraud detection will be introduced to analyze booking patterns and detect suspicious transactions.

The Traveler Assistant Bot successfully bridges the gap between travelers and local service providers, creating a secure, efficient, and scalable booking ecosystem. By leveraging modern technologies such as Spring Boot for backend processing, React.js for administration, and PayHere for transactions, the system establishes a strong foundation for digital transformation in the tourism sector. With planned enhancements such as AI-based recommendations, multilingual support, and mobile wallet integration, the platform has the potential to evolve into a comprehensive digital marketplace, improving accessibility and service quality for both travelers and service providers.

7 Reference

Abhinav, 2023. Building a Telegram Bot [WWW Document]. URL https://medium.com/@abhinavv.singh/building-a-telegram-bot-and-integrating-it-with-a-spring-boot-api-a-step-by-step-guide-dcd583cae7e0 (accessed 4.12.24).

Agoda, n.d. Agoda Official Site | Free Cancellation & Booking Deals | Over 2 million Hotels [WWW Document]. URL https://www.agoda.com/?site_id=1922893&tag=e8270226-244d-4884-85d9-8fe4141dae0b&gad_source=1&device=c&network=g&adid=695886434540&rand=7844378938835798209&expid=&adpos=&aud=kwd-2230651387&gclid=Cj0KCQiAlbW-BhCMARIsADnwasqsrWhQrrArLdivX8QQ9Tz1gW-6SUHvdjqxlWR9F5-GZHVUCaJxg9gaAgVjEALw_wcB&pslc=1&ds=CsLH%2Filz1eK2jCBO (accessed 3.10.25).

Booking.com, n.d. Booking.com | Official site | The best hotels, flights, car rentals & accommodations [WWW Document]. URL https://www.booking.com/ (accessed 3.10.25).

core.telegram.org, n.d. Bot Payments API [WWW Document]. URL https://core.telegram.org/bots/payments (accessed 9.10.24).

Despina Wilson, 2024. World's Best Countries To Visit In Your Lifetime, 2024 - CEOWORLD magazine [WWW Document]. URL https://ceoworld.biz/2024/04/15/worlds-best-countries-to-visit-in-your-lifetime-2024/ (accessed 9.11.24).

Expedia Travel, n.d. Expedia Travel: Vacation Homes, Hotels, Car Rentals, Flights & More [WWW Document]. URL https://www.expedia.com/?locale=en_US&siteid=1&semcid=US.B.GOOGLE.BT-c-EN.GT&semdtl=a118255096950.b1143063384553.g1kwd-12670071.e1c.m1Cj0KCQiAlbW-BhCMARIsADnwasolzmSPIvdohFxAIhhHIU8Y3rvtb3PH_ew6Bh5L1zvMVlYYoJg6-ToaAnTNEALw_wcB.r186c5ceb1813f82e09a3d77e1d04605e5d07dc63d3419b29b7e84c18818d0c03e.c1.j19069431.k1.d1720994358040.h1e.i1.l1.n1.o1.p1.q1.s1.t1.x1.f1.u1.v1.w1&gad_source=1&gclid=Cj0KCQiAlbW-BhCMARIsADnwasolzmSPIvdohFxAIhhHIU8Y3rvtb3PH_ew6Bh5L1zvMVlYYoJg6-ToaAnTNEALw_wcB (accessed 3.10.25).

Fiverr, n.d. Fiverr | Freelance services marketplace | Find top global talent [WWW Document]. URL https://www.fiverr.com/ (accessed 3.6.25).

hotels combined, n.d. HotelsCombined | Find Deals on Travel Accommodations [WWW Document]. URL https://www.hotelscombined.com/ (accessed 3.10.25).

Hotels.com, n.d. Hotels.com - Deals & Discounts for Hotel Reservations from Luxury Hotels to Budget Accommodations [WWW Document]. URL https://www.hotels.com/?locale=en_AS&pos=HCOM_ASIA&siteid=300000034 (accessed 3.10.25).

IEEE Std 830-1998, 1998. IEEE Recommended Practice for Software Requirements Specifications IEEE Computer Society. IEEE Computer Society.

ikman, n.d. About us | ikman [WWW Document]. URL https://ikman.lk/en/help/about#help-content (accessed 9.11.24).

Kissflow, 2024. Rapid Application Development (RAD) | Definition, Steps & Full Guide [WWW Document]. URL https://kissflow.com/application-development/rad/rapid-application-development/ (accessed 10.17.24).

Lonami, n.d. Get geolocation via Telegram bot using inline - Stack Overflow [WWW Document]. URL https://stackoverflow.com/questions/68867988/get-geolocation-via-telegram-bot-using-inline (accessed 4.2.24).

Md Suny Shaikh, 2024. Why Should We Use React for the Front End? | LinkedIn [WWW Document]. URL https://www.linkedin.com/pulse/why-should-we-use-react-front-end-md-suny-shaikh-kbxdc/ (accessed 8.14.24).

Mero (Järvinen), J., 2018. The effects of two-way communication and chat service usage on consumer attitudes in the e-commerce retailing sector. Electronic Markets 28, 205–217.

Mule, S.S., Yashwant Waykar, M., 2015. ROLE OF USE CASE DIAGRAM IN S/W DEVELOPMENT.

Nick Bulaienko, 2024. Integrating Payment Systems into Telegram Mini Apps: A Comprehensive Guide - BAZU [WWW Document]. URL https://bazucompany.com/blog/integrating-payment-systems-into-telegram-mini-apps-a-comprehensive-guide/ (accessed 9.10.24).

Nikolay R, 2023. Bots for Telegram: top examples, use cases, and benefits for companies [WWW Document]. URL https://umnico.com/blog/telegram-bots/ (accessed 4.2.24).

Nurvembrianti, I., Arianti, N., Noftalina, E., 2022. The Use of Telegram Chatbot Application Services as a Means of Communication in Increasing Satisfaction and Knowledge of

Mothers Who have Toddlers. Galore International Journal of Health Sciences and Research 7, 19–25.

OnaWadak, n.d. ඕනවැඩක් (OnaWadak) [WWW Document]. URL https://www.onawadak.lk/ (accessed 3.9.24).

Priyal Walpita, 2019. Software Architecture Patterns — Layered Architecture | by Priyal Walpita | Medium [WWW Document]. URL https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057 (accessed 10.17.24).

Quickhelp, n.d. Quickhelp [WWW Document]. URL https://www.quickhelp.lk/ (accessed 3.9.24).

Rianto, R., Rahmatulloh, A., Firmansah, T.A., 2019. Telegram Bot Implementation in Academic Information Services with The Forward Chaining Method. Sinkron 3, 73–78.

Spring, n.d. Spring Boot [WWW Document]. URL https://spring.io/projects/spring-boot (accessed 3.9.24).

Taskrabbit, n.d. Taskrabbit: Same Day Handyman, Moving & Mounting Services [WWW Document]. URL https://www.taskrabbit.com/ (accessed 3.6.25).

ThatchGPT, n.d. ThatchGPT | AI Travel Assistant [WWW Document]. URL https://www.thatch.co/thatchgpt (accessed 3.6.25).

Thilakarathna, K.G.D.C., 2023. Web Based Industrial Training Management System (ITMS) for National Water Supply & Drainage Board.

Thumbtack, n.d. Thumbtack | Care for Your Home | Find Local Pros & Reviews [WWW Document]. URL https://www.thumbtack.com/?return=https%3A%2F%2Fthumbtack.57ib.net%2Fc%2F4791133%2F1819752%2F4348%3FsubId1%3D%26sharedid%3D365326707%26u%3Dhttp%253A%252F%252Fwww.thumbtack.com%26level%3D1&cid=4348&tpsync=yes&auth=5890da0a58ae4777&gad_source=1&gclid=CjwKCAiArKW-BhAzEiwAZhWsIGSn4DYKfAnsHIezjv-h-kN5UsisMxUOXFNR4o3YAQoNnqm4_4OvWxoCOZUQAvD_BwE (accessed 3.6.25).

Van Eeuwen, M., 2017. Mobile conversational commerce: messenger chatbots as the next interface between businesses and consumers.

wanderlog.com, n.d. Wanderlog: travel itinerary, vacation & road trip planner [WWW Document]. URL https://wanderlog.com/home (accessed 9.10.24).

Yathra.lk, n.d. About us – Yathra Travels [WWW Document]. URL https://www.yathra.lk/about/ (accessed 9.11.24).

Appendix A – Test Cases

Below is the categorized test cases conducted for the Traveler Assistant Bot system, divided into Functional Testing, Unit Testing, and System Testing as outlined in the thesis.

Test Case ID	Test Scenario	Test Steps	Expected Outcome	Status
TC-001	System User Login with Valid Credentials	System User enters correct credentials and submits login request	User is authenticated and receives a valid JWT token	Pass
TC-002	System User Login with Invalid Credentials	User enters incorrect credentials and submits login request	System returns 'Invalid username or password' error	Pass
TC-003	Access Protected API without Token	User attempts to access an API endpoint without providing a JWT token	System returns 'Unauthorized access' error	Pass
TC-004	Booking a Service Successfully	User selects a service and confirms booking	Booking is stored in the database and user receives confirmation	Pass
TC-005	Booking an Unavailable Service	User attempts to book a service that is fully booked	System returns 'Service not available' error	Pass
TC-006	View Booked Services	User requests a list of their booked services	System returns a list of user bookings	Pass
TC-007	Successful Payment Processing	User initiates a payment through PayHere	Payment is verified, and booking status is updated to 'PAID'	Pass
TC-008	Payment with Invalid Hash Signature	User attempts to process payment with a modified hash signature	System rejects the transaction with 'Invalid payment signature' error	Pass
TC-009	Admin Dashboard Access	Admin logs in and accesses the dashboard	System displays correct user and transaction statistics	Pass
TC-010	Unauthorized Admin Access	Non-admin user attempts to access the admin panel	System returns 'Access Denied' error	Pass
TC-011	Service Provider Registration	User registers as a service provider via Telegram Mini App	System stores provider details and grants provider role	Pass
TC-012	Service Provider Adds New Service	Service provider submits service details	System stores the new service and makes it available for booking	Pass
TC-013	Service Provider Edits Service	Provider modifies existing service details	System updates the service information successfully	Pass

TC-014	Successful Payment Processing	User initiates a payment through PayHere	Payment is verified, and booking status is updated to ‘PAID’	Pass
TC-015	Payment with Invalid Hash Signature	User attempts to process payment with a modified hash signature	System rejects the transaction with ‘Invalid payment signature’ error	Pass

Appendix B- User Documentation

Travel mingle Admin Pannel Login Page

To access the **Travel mingle Admin Pannel**, by entering http://localhost:5174/login in the address bar in any web browser. and, the user will find the following home page in

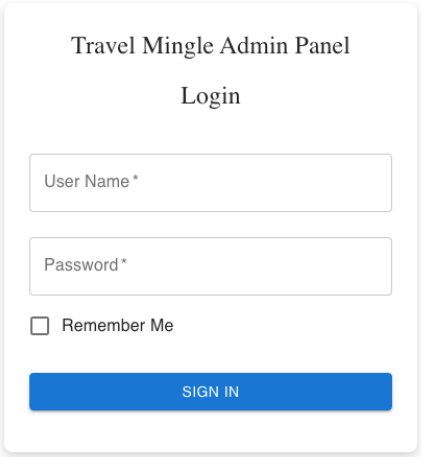


Figure 5.2.2-1: Admin Pannel Login

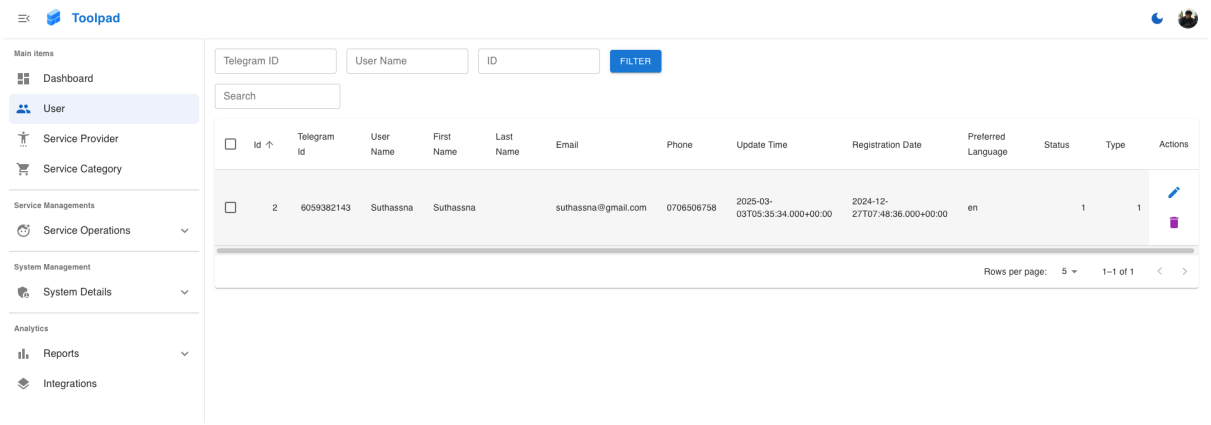
Login

The System Login (Figure 5.2.2-1) API allows administrators to authenticate into the Admin Panel using their credentials. The API verifies the username and password, and upon successful authentication, issues a JWT access token and a refresh token for session management.

Admin Dash Board

Figure 5.2.2-2: Admin Dashboard

Upon successful authentication, the admin dashboard (Figure 5.2.2-2) is displayed, providing administrators with access to system statistics, user management, transaction monitoring, and service management functionalities. The dashboard serves as the centralized control panel, allowing administrators to oversee platform activities, manage service providers and travelers, and ensure the secure operation of the system. Key performance indicators, including total users, active bookings, pending transactions, and revenue insights, are presented to facilitate informed decision-making. Additionally, administrators can navigate through various modules, such as user management, service approval, payment verification, and report generation, ensurin



g efficient platform administration.

Service Category management

Administrators have the privilege to create and manage service categories (Figure 5.2.2-3) within the platform, ensuring that the system accommodates a structured and well-organized service catalog. This functionality allows admins to define the categories under which service providers can register and list their services, enhancing the platform's usability and accessibility.

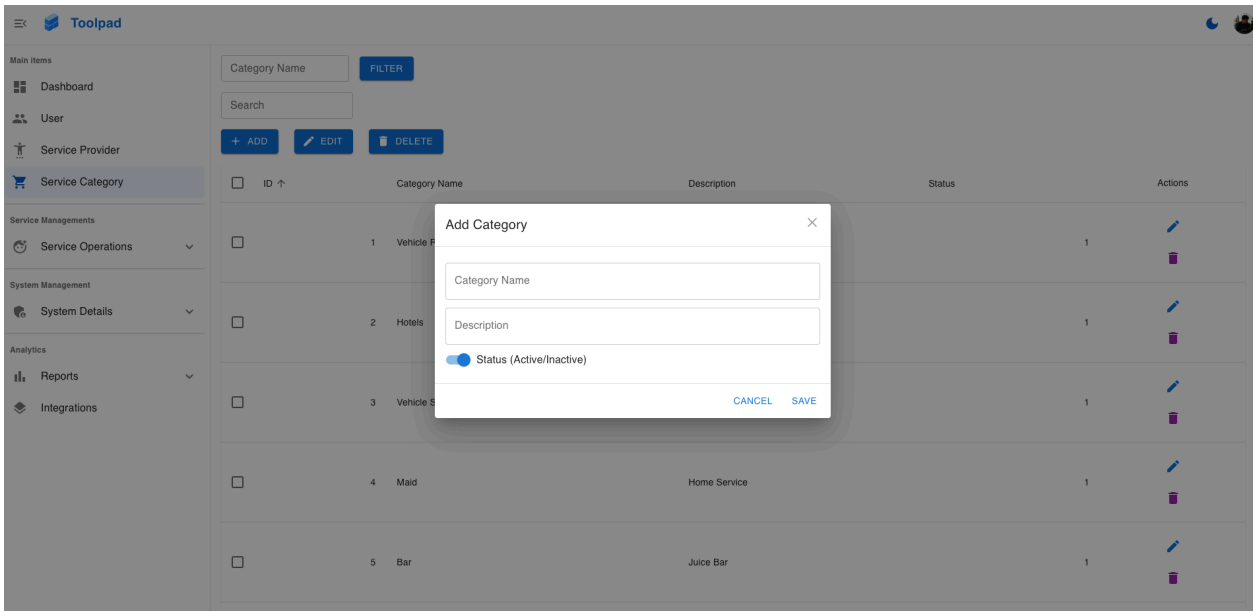


Figure 5.2.2-3: Service Category Management

Through the Admin Panel, administrators can:

- Create new service categories based on user demand and platform requirements.
- Modify existing categories to reflect changes in service offerings.
- Deactivate or remove categories that are no longer relevant.

This feature ensures that service providers can accurately categorize their offerings, allowing travelers to efficiently search and book services based on well-defined classifications. Proper categorization enhances service discoverability, user experience, and system scalability.

Service Managemnt

Service providers have the ability to register and list their services on the platform through the Telegram Mini App. However, to ensure quality control, authenticity, and compliance with platform policies, each service registration must be reviewed and approved by an administrator (Figure 5.2.2-4) before it becomes visible to travelers.

Upon submission, the service details, pricing, and availability are recorded in the system and placed under a pending verification status. Administrators can then review the submitted services through the Admin Panel, where they can:

- Verify service details to ensure accuracy and relevance.
- Approve the service, making it publicly accessible to travelers.
- Reject or request modifications if the service does not meet platform guidelines.

This verification process ensures that only legitimate, high-quality services are made available to travelers, enhancing trust, reliability, and overall user experience on the platform.

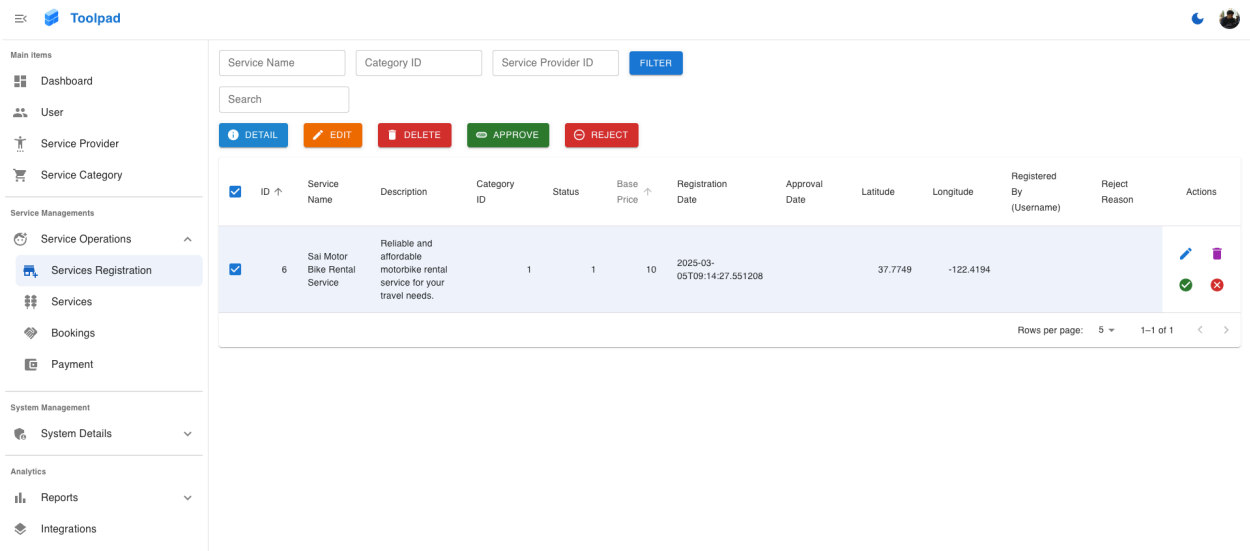


Figure 5.2.2-4: Service Management Board

Travel Mingle Application

On the Bot side (Figure 5.2.2 -5&6), when a user initiates interaction with the **Travel Mingle** (Traveler Assistant Bot) on Telegram by clicking "/Start", they are prompted to complete the registration process. The system first asks the user to choose their role:

1. Traveler (General User) – A user who can browse available services, book appointments, and make payments.
2. Service Provider – A user who offers services and manages bookings through the platform.

Once the user selects their role:

- If registering as a Traveler, the system verifies their Telegram ID and automatically registers them as a user.
- If registering as a Service Provider, the system prompts them to submit business details (e.g., service category, contact information) for admin verification before they can list services.

After successful registration, the user is directed to the main menu, where they can:

- Access the Telegram Mini App, which provides a user-friendly interface for browsing, booking, and managing services.
- Explore available services (for travelers).
- Manage service listings (for service providers).
- View and update personal profile details.

This structured registration process ensures that users are properly categorized and granted the appropriate system privileges, facilitating a smooth and intuitive onboarding experience.

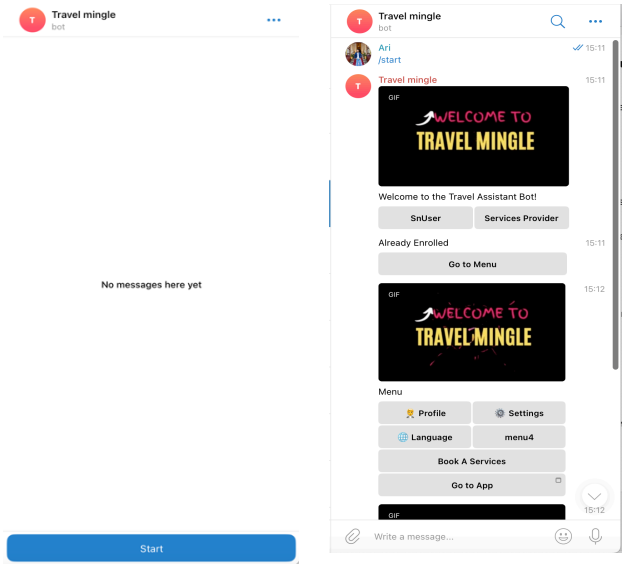


Figure 5.2.2-5: Bot Initialization and User Registration

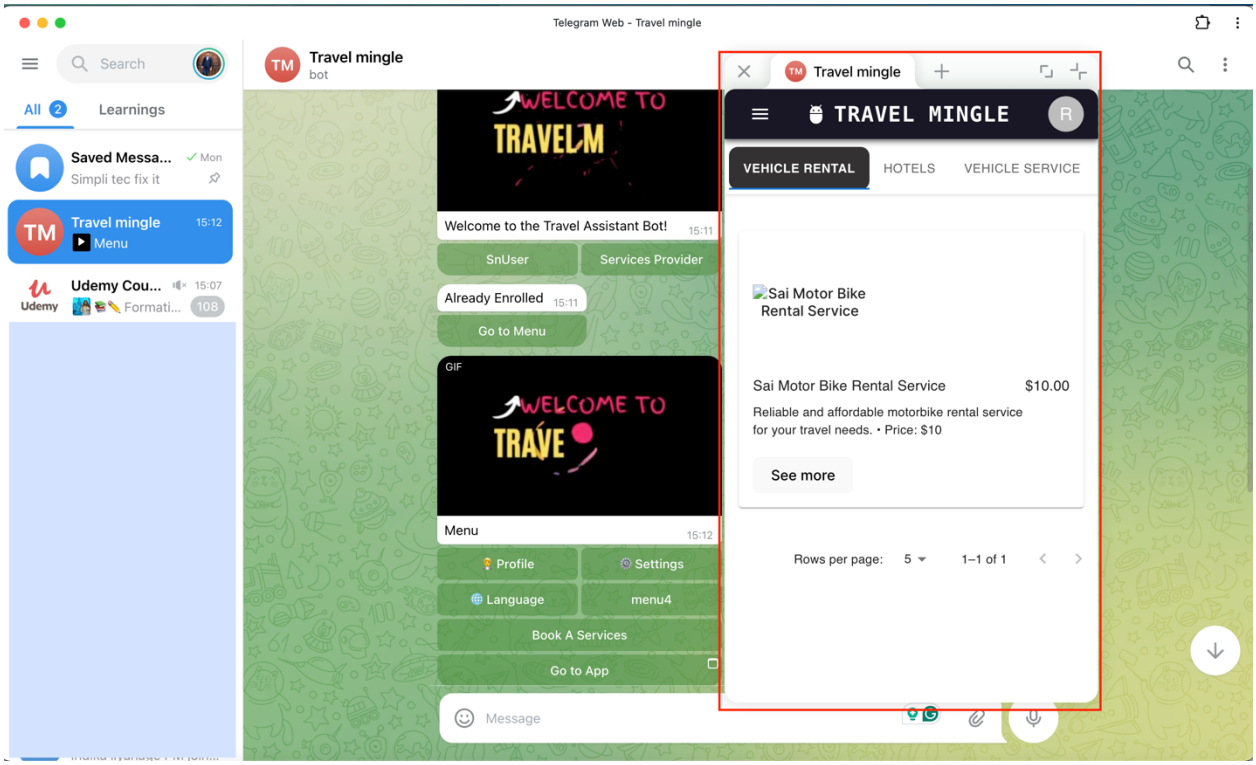


Figure 5.2.2-6: Mini app in the telegram chat

Payment Implementation

For payment processing (Figure 5.2.2-7), the system ensures that users can only proceed with payments after the service has been successfully delivered. Once a service provider marks a booking as "Completed", the user receives a notification and gains access to the payment gateway to finalize the transaction.

Payment Flow:

1. Service Completion Confirmation:
    - The service provider updates the booking status to "Completed" through the Telegram Mini App.
    - The system notifies the user that the service has been successfully provided and payment is now required.
2. Payment Initiation:
    - The user accesses the payment gateway (PayHere) via the Telegram Bot menu or Mini App.
    - The system generates a payment request, including service details, amount, and transaction reference.
3. Secure Transaction Processing:
    - The user enters payment details and completes the transaction through PayHere.

- The system validates the payment using MD5 hash verification to ensure security.
- Upon successful payment, the booking status is updated to “Paid”, and the service provider receives confirmation of payment.

4. Admin & System Monitoring:
   - Administrators can monitor pending and completed transactions through the Admin Panel.
   - Any failed or disputed transactions can be reviewed for necessary action.

This structured post-service payment model ensures trust, transparency, and accountability between travelers and service providers, preventing advance payment risks and enhancing transaction security.

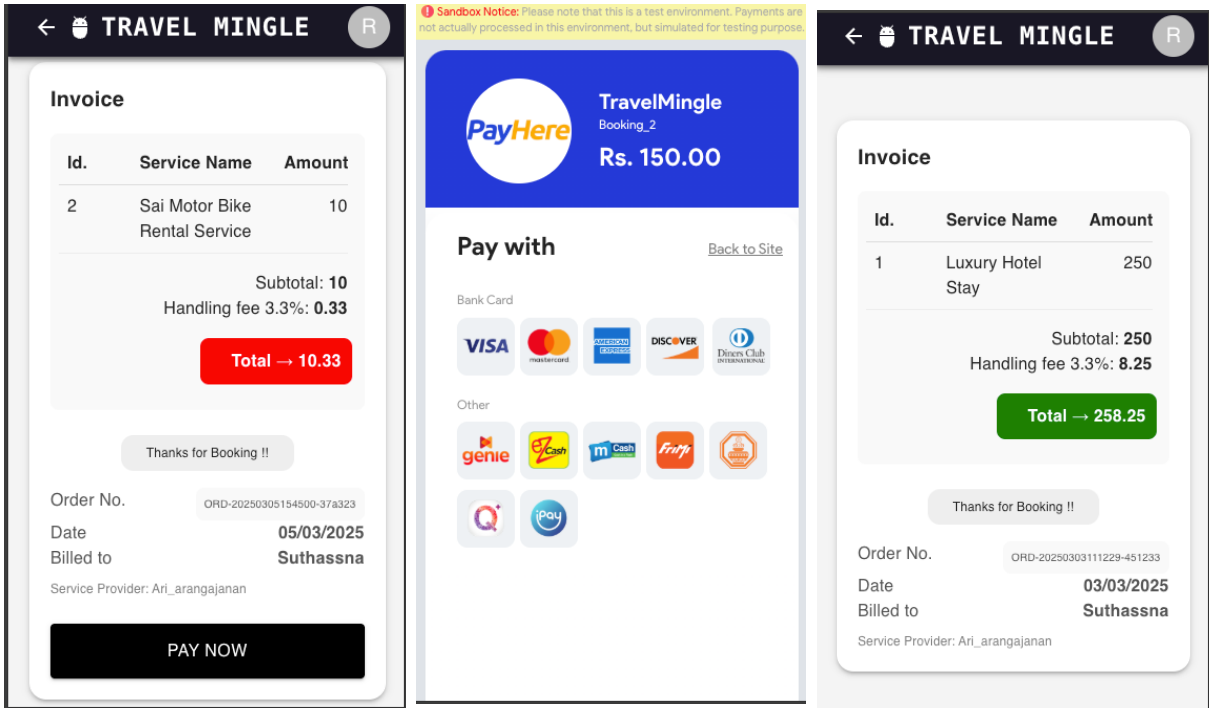


Figure 5.2.2-7: payment Process of the Application