



**Lanka Nippon Biztech Institute**

## **SRS Document**

**(Railway Tourism Guide System)**

**Name : G.G.Piyumi Natasha**

**Index Number : UGC0122026**

**Batch : SE 01**

## Dedication

I dedicate this project to my family whose unwavering support and encouragement have been instrumental in my academic journey. I am grateful for their patience and understanding during the countless hours spent researching, designing, and implementing this project. Their belief in my abilities has motivated me to strive for excellence and persevere through challenges. This achievement is a testament to their love and sacrifice, and I am deeply thankful for their continuous encouragement that has propelled me toward success.

# Acknowledgments

I extend my heartfelt gratitude to my supervisor, [Supervisor's Name], whose invaluable guidance and expertise have shaped this project into a rewarding experience. Their constructive feedback and insightful suggestions have enriched my understanding of software development methodologies. I also thank the faculty members of [Your University/Institution] for their support and encouragement throughout this endeavor. Additionally, I express appreciation to my peers and friends for their assistance and encouragement. Finally, I acknowledge the contributions of the developers and researchers whose work served as inspiration and foundation for this project.

## Table of Contents

Dedication .....	
Acknowledgments.....	ii
List of Figures .....	v
List of Acronyms .....	vii
1. Chapter 01: Introduction .....	1
1.1. Introduction .....	1
1.2. Introduction System.....	1
1.3. Motivation .....	2
1.4. Aim .....	2
1.5. Objectives of the Project.....	2
1.6. Scope of the Project.....	3
1.7. Summary.....	3
2. Chapter 02: Analyze.....	4
2.1. Introduction .....	4
2.2. Review Similar System with References.....	4
2.3. Identifying Functional Requirements .....	4
2.4. Identifying Non-Functional Requirements .....	6
2.5. Identifying Suitable Process Model.....	6
2.6. Summary.....	6
3. Chapter 03: Design.....	7
3.1. Introduction .....	7
3.2. Introduction to Clients .....	7
3.3. Scope Identification.....	8
3.4. Use Case Diagram .....	9
3.4.1. Identification of Roles .....	10
3.4.2. Identification of Cases .....	12
3.4.3. Identification of Dependencies .....	12
3.5. ER Diagram .....	13
3.5.1. Identification of Entities .....	14
3.5.2. Justification of Attributes.....	14
3.5.3. Identification of Relationships.....	17
3.6. Class Diagram .....	19
3.7. Sequence Diagram.....	20
3.8. State Chart Diagram .....	25

3.9.	Activity Diagram .....	30
3.10.	Summary.....	35
4.	Chapter 4: Implementation.....	36
4.1.	Introduction .....	36
4.2.	SQL Queries .....	36
4.2.1.	DML .....	36
4.2.2.	Test Plan .....	37
3.	Functional Requirements: .....	50
4.	Non-Functional Requirements: .....	50
5.	Testing .....	51
6.	Annexure.....	56
4.3.	Summary.....	74

## List of Figures

Figure 3.1: Use Case Diagram of Rail Guide Pro .....	9
Figure 3.2: ER Diagram of Rail Guide Pro Project .....	13
3.3:Class Diagram of Rail Guide Pro Project .....	19
3.4:Sequence Diagram 01 of Rail Guide Pro Project.....	20
Figure 3.5:Sequence Diagram 02 of Rail Guide Pro Project .....	21
Figure 3.6::Sequence Diagram03 of Rail Guide Pro Project .....	22
Figure 3.7:Sequence Diagram 04of Rail Guide Pro Project .....	23
Figure 3.8: Sequence Diagram 05 of Rail Guide Pro Project .....	24
Figure 3.9:State Chart Diagram 01 of Rail Guide Pro Project.....	25
Figure 3.10: :State Chart Diagram 02 of Rail Guide Pro Project.....	26
Figure 3.11:State Chart Diagram 03 of Rail Guide Pro Project.....	27
Figure 3.12:State Chart Diagram 04 of Rail Guide Pro Project.....	28
Figure 3.13:State Chart Diagram 05 of Rail Guide Pro Project.....	29
Figure 3.14:Activity Diagram 01 of Rail Guide Pro Project .....	30
Figure 3.15:Activity Diagram 02 of Rail Guide Pro Project .....	31
Figure 3.16:Activity Diagram 03 of Rail Guide Pro Project .....	32
Figure 3.17:Activity Diagram 04 of Rail Guide Pro Project .....	33
Figure 3.18:Activity Diagram 05 of Rail Guide Pro Project .....	34
Figure 4.1:Query 01 of Rail Guide Pro Project .....	38
Figure 4.2:Query 01 Result of Rail Guide Pro Project .....	38
Figure 4.3:Query 02 of Rail Guide Pro Project .....	40
Figure 4.4:Query 02 Result of Rail Guide Pro Projec .....	40
Figure 4.5:Query 03 of Rail Guide Pro Project .....	41
Figure 4.6:Query 03 Result of Rail Guide Pro Project .....	42
Figure 4.7:Query 04 of Rail Guide Pro Project .....	44
Figure 4.8:Query 04 Result of Rail Guide Pro Project .....	45
Figure 4.9:Query 05 of Rail Guide Pro Project .....	46
Figure 4.10:Query 05 Result of Rail Guide Pro Project .....	46
Figure 4.11:Login Administrator .....	48
Figure 4.12:Admin Dashboard.....	49

# List of Table

Table 4.1:Test plan.....	37
Table 4.2:Test Case 01 .....	39
Table 4.3:Test Case 01 Result .....	39
Table 4.4:Test Case 02.....	41
Table 4.5:Test Case 02 Result .....	41
Table 4.6:Test Case 03.....	43
Table 4.7:Test Case 03 Result .....	43
Table 4.8:Test Case 04.....	45
Table 4.9:Test Case 04 Result .....	45
Table 4.10:Test Case 05.....	47
Table 4.11:Test Case 05 Result.....	47

# List of Acronyms

DDL- Data Definition Language

ER- Entity Relationship

SQL- Structured query language

DML- Data Manipulation Language



# 1. Chapter 01: Introduction

## 1.1. Introduction

In this Chapter includes providing seamless travel assistance from departure to destination stations, highlighting nearby tourist stations, offering information on nearby hotels and lodging facilities, maintaining a directory of tourist stations with cultural and historical points of interest, detailing railway station facilities and amenities, and offering real-time updates on facilities and services at each location.

## 1.2. Introduction System

In recent years, there has been a big change in the travel area, with more seeking special and immersive experiences. Railway tourism, in particular, has become a unique and eco-friendly way to travel, providing beautiful views and cultural exploration. However, managing guides for railway tourism and giving tourists the right information at the right time are still big challenges. In the current system, only booking and ticketing are done. Tourists feel difficulty in traveling not only tourists but also nontravelers have a problem. Lack of detailed and easily accessible information for travelers regarding train schedules. With our system, for the convenience of travelers, when they travel from the station that starts to the destination to the next station, the nearby tourist stations, hotels with lodging facilities, railway stations, etc. are available in our system.

### 1.3. Motivation

The motivation for this project arises from the recognition of the untapped tourism potential near railway stations. Despite the cultural and historical significance of these areas, tourists often face challenges in discovering and exploring the available attractions. Moreover, the management of these places lacks a centralized and technologically sophisticated system. This project aims to bridge these gaps and provide an integrated solution for both tourists and administrators.

### 1.4. Aim

This project's main aims are to develop a comprehensive Railway Tourism Management System that enhances the overall experience of railway tourists and improves guide management efficiency. The specific objectives include.

### 1.5. Objectives of the Project

- **Efficient Attractions Management:** To establish a centralized system for managing and promoting tourist attractions around railway stations.
- **Enhanced Tourist Experience:** To provide tourists with a user-friendly guide system, offering real-time information about nearby places of interest, events, and historical landmarks.
- **Optimized Navigation:** To integrate interactive maps and navigation features, aiding tourists in exploring the area seamlessly.
- **Administrative Control:** To empower administrators with tools for effective management, including attraction updates, user feedback analysis, and event coordination.

- **User Engagement Enhancement:** Implement features to enhance user engagement, encouraging tourists to actively participate in the system. By providing feedback, sharing experiences, and contributing to the community aspect of railway tourism.

## 1.6. Scope of the Project

- **Station-to-Station Travel Assistance:** Provide a seamless guide for travelers from the departure station to the destination, highlighting nearby tourist stations and facilitating easy transitions.
- **Accommodation Information:** Include information on hotels and lodging facilities in proximity to railway stations.
- **Tourist Stations Directory:** Maintain a directory of nearby tourist stations, displaying cultural, historical, or scenic points of interest.
- **Railway Station Information:** Provide comprehensive details about each railway station, including facilities, services, and local amenities.
- **Real-Time Updates on Facilities:** Offer real-time updates on the availability of facilities, events, and services at each location.

## 1.7. Summary

In summary, this chapter introduces the Railway Tourism Management System, highlighting the challenges faced by current systems, the motivation behind the project, its aims and objectives, and the scope of work to be undertaken. The subsequent chapters will delve into the methodology, detailed system design, implementation, and evaluation of this innovative solution aimed at enhancing the railway tourism experience.

## 2. Chapter 02: Analyze

### 2.1. Introduction

This chapter focuses on reviewing existing systems related to railway tourism management, identifying functional and non-functional requirements, and selecting a suitable process model for the development of the Railway Tourism Management System.

### 2.2. Review Similar System with References

To understand the landscape of railway tourism management systems, a review of existing systems and related literature will be conducted. This review will provide insights into the functionalities, technologies, and challenges addressed by similar systems, enabling the identification of best practices and areas for improvement.

### 2.3. Identifying Functional Requirements

#### 1. User Registration.

1.1 Customers can be registered with the system.

#### 2. Destination Information.

2.1 Display detailed information about each tourist destination, including

2.2 Implement a search feature allowing users to find specific tourist

Destinations based on keywords.

2.3 Incorporate an interactive map to visually represent the available  
tourist destinations and their proximity to railway stations.

### 3. Train Schedules.

3.1 Provide detailed information about the routes each train takes, including intermediate stops.

3.2 Display a clear and user-friendly schedule of train departures and arrivals with Clearly indicate the arrival times for each train at various stations.

3.3 Specify how often each train service operates (e.g., daily, weekly

### 4. Pricing Management.

4.1 Define pricing tiers for different classes of training (e.g., first class, Second, class) based on services offered.

4.2 Establish pricing structures for hotel packages, considering factors such as accommodation type, duration, and additional services

4.3 Implement a dynamic pricing mechanism that may vary based on factors like demand, seasonality, or special events.

### 5. Admin Management.

5.1 Admin can add Staff to the system.

5.2 Admin cans View Staff 5.3 Admin cans Update Staff of the system .

5.3 Admin cans Update Staff of the system.

5.4 Admin can delete Staff.

## 2.4. Identifying Non-Functional Requirements

- Accommodation in the system to facilitate users with disabilities.
- Support multiple currencies and various payment options for the convenience of users.
- Maintain historical data to allow users to view past schedules.

## 2.5. Identifying Suitable Process Model

For my project, I have opted to use the Waterfall model as the preferred methodology for software development. The decision to choose this model is based on several key factors and considerations that align with the project's requirements and objectives.

## 2.6. Summary

This chapter outlines the need for a Railway Tourism Management System, reviews existing systems, and details the functional and non-functional requirements necessary for its development. The subsequent sections will delve deeper into process model selection and outline the methodology for implementing the system to meet these identified requirements effectively.

## 3. Chapter 03: Design

### 3.1. Introduction

This chapter delves into the design aspects of the Railway Tourism Management System, outlining various diagrams and models that depict the system's structure, functionality, and interactions.

### 3.2. Introduction to Clients

In recent years, the travel landscape has undergone a significant transformation, with a growing interest in unique and immersive experiences. Railway tourism has emerged as a distinctive and eco-friendly mode of travel, offering captivating vistas and opportunities for cultural exploration. However, despite its appeal, managing guides for railway tourism and providing tourists with timely, relevant information remain significant challenges.

Our system addresses these challenges by going beyond traditional booking and ticketing services. We understand that both travelers and non-travelers encounter difficulties in navigating the railway tourism experience, particularly due to a lack of comprehensive and easily accessible information on train schedules and nearby amenities.

### 3.3. Scope Identification

- **Station-to-Station Travel Assistance:** Provide a seamless guide for travelers from the departure station to the destination, highlighting nearby tourist stations and facilitating easy transitions.
- **Accommodation Information:** Include information on hotels and lodging facilities in proximity to railway stations.
- **Tourist Stations Directory:** Maintain a directory of nearby tourist stations, displaying cultural, historical, or scenic points of interest.
- **Railway Station Information:** Provide comprehensive details about each railway station, including facilities, services, and local amenities.
- **Real-Time Updates on Facilities:** Offer real-time updates on the availability of facilities, events, and services at each location.



### 3.4. Use Case Diagram

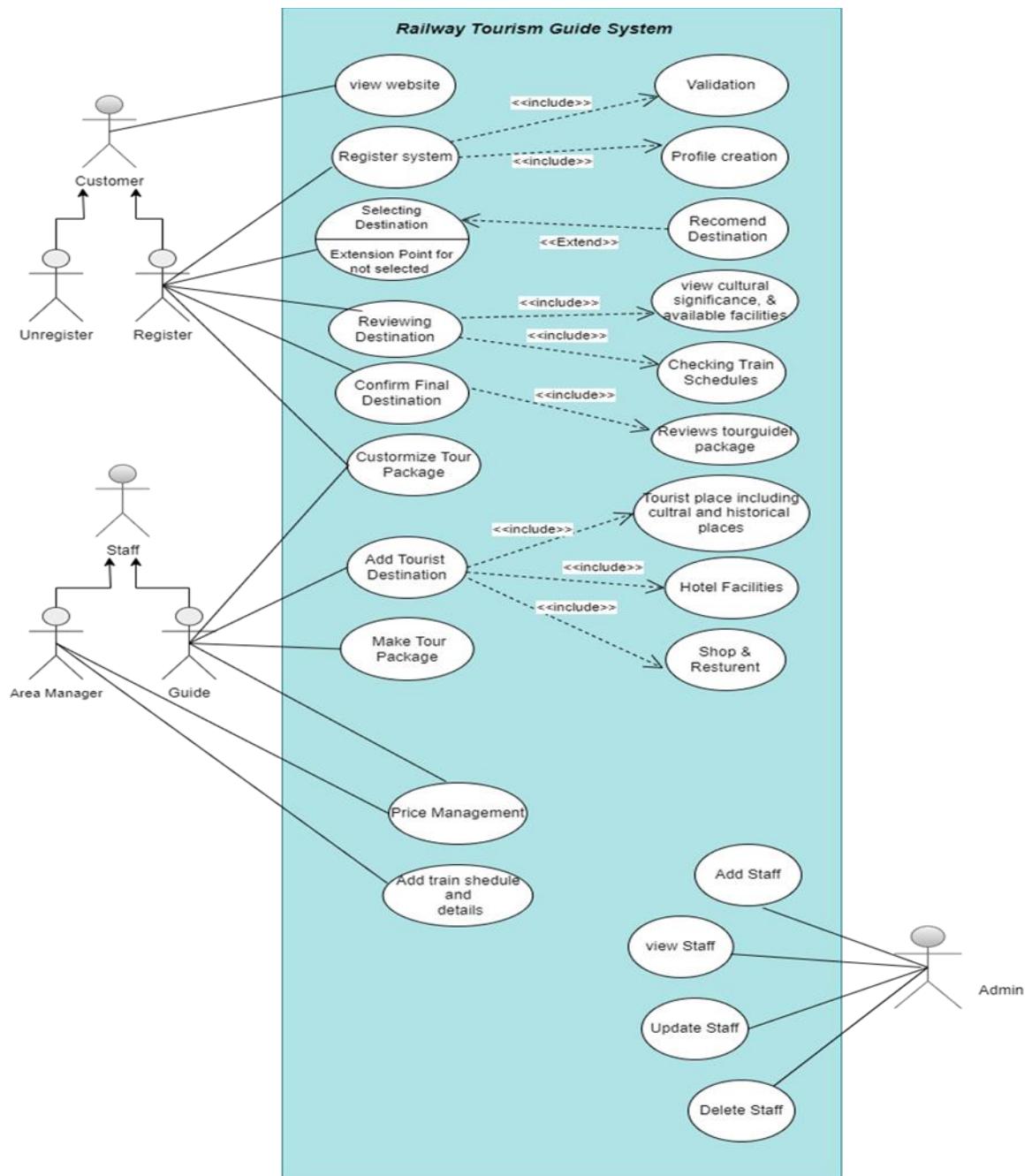


Figure 3.1: Use Case Diagram of Rail Guide Pro

The above diagram (Figure 3.1) focuses on two main actors: customers and staff. Customers can utilize functionalities like "view destination cultural significance" and "view available facilities" to plan their trips. Staff can add destinations, update tourist information, and manage prices through functionalities designed for them.

### 3.4.1. Identification of Roles

#### **Customer**

**Role:** The customer is the end-user of the railway tourism services.

##### **Registering**

- Customers can register themselves on the platform to access various services.

##### **Unregistering**

- Customers have the option to unregister or deactivate their accounts if needed.

##### **Relationships**

- Tour Package to Customer: Many-to-many Relationship.
- Customer to Station: many-to-many Relationship.

#### **Staff**

**Role:** Staff members are employees who work within the railway tourism system.

##### **Area Manager**

- Overseeing Operations: Area managers are responsible for overseeing operations within a specific geographical area or region.
- They allocate resources, monitor performance, and ensure smooth operation.

### **Relationships**

- Area Manager to Station: One to Many Relationships.
- Area Manager to tour package: Many to Many Relationships.

### **Guide**

- Sharing Knowledge: They provide insights into historical, cultural, and scenic aspects of the tour destinations.
- Guides assist customers during their railway tours, offering information, guidance, and commentary.

### **Relationships**

- Guide to Tour package: Many to Many Relationship.
- Guide to Hotel: One to Many Relationship.
- Guide to Tour Location: One to Many Relationship.
- Guide to Shop and Restaurants: One to Many Relationship.

### **Admin**

**Role:** The admin is the administrative user with higher-level privileges.

- Admins manage user accounts, including creating, modifying, or deactivating accounts as needed.
- They configure system settings, including pricing, tour schedules, and other parameters.

### **Relationships**

- Admin to Staff: One-to-Many Relationship.

### 3.4.2. Identification of Cases

#### **Customer Use Cases:**

- **Search for Destinations:** A customer can search for destinations based on various criteria like location, keywords (e.g., historical sites, beach destinations), or travel dates.
- **View Destination Details:** A customer can view detailed information about a specific destination, including cultural significance, attractions, nearby facilities, and travel logistics.
- **Plan a Trip Itinerary:** A customer can create a personalized trip itinerary by selecting destinations and planning travel logistics between them. (This might involve integrating with train booking systems if applicable).

#### **Staff Use Cases:**

- **Add a New Destination:** Staff can add a new destination to the system, including details like description, cultural significance, and location.
- **Update Destination Information:** Staff can edit and update existing information about destinations, including attractions, facilities, and travel logistics.
- **Manage User Accounts:** Staff can create, edit, or delete user accounts and assign access permissions within the system.

### 3.4.3. Identification of Dependencies

- **System Login (Dependent):** The user might need to be logged in to access detailed information not publicly available.
- **Data Availability (Dependent):** The system needs to retrieve and display information about the specific destination, including cultural significance and available facilities. This information should be up-to-date within the system.
- **Search Functionality (Dependent):** This use case might depend on a search function that allows users to find destinations based on various criteria (e.g., location, keywords).

### 3.5. ER Diagram

An Entity-Relationship (ER) diagram is a powerful tool used in database design to visualize and define the relationships between various entities in a system. It presents a clear and structured representation of the data model, showing how different entities relate to each other and the attributes associated with each entity. ER diagrams are essential for understanding the underlying structure of a database.

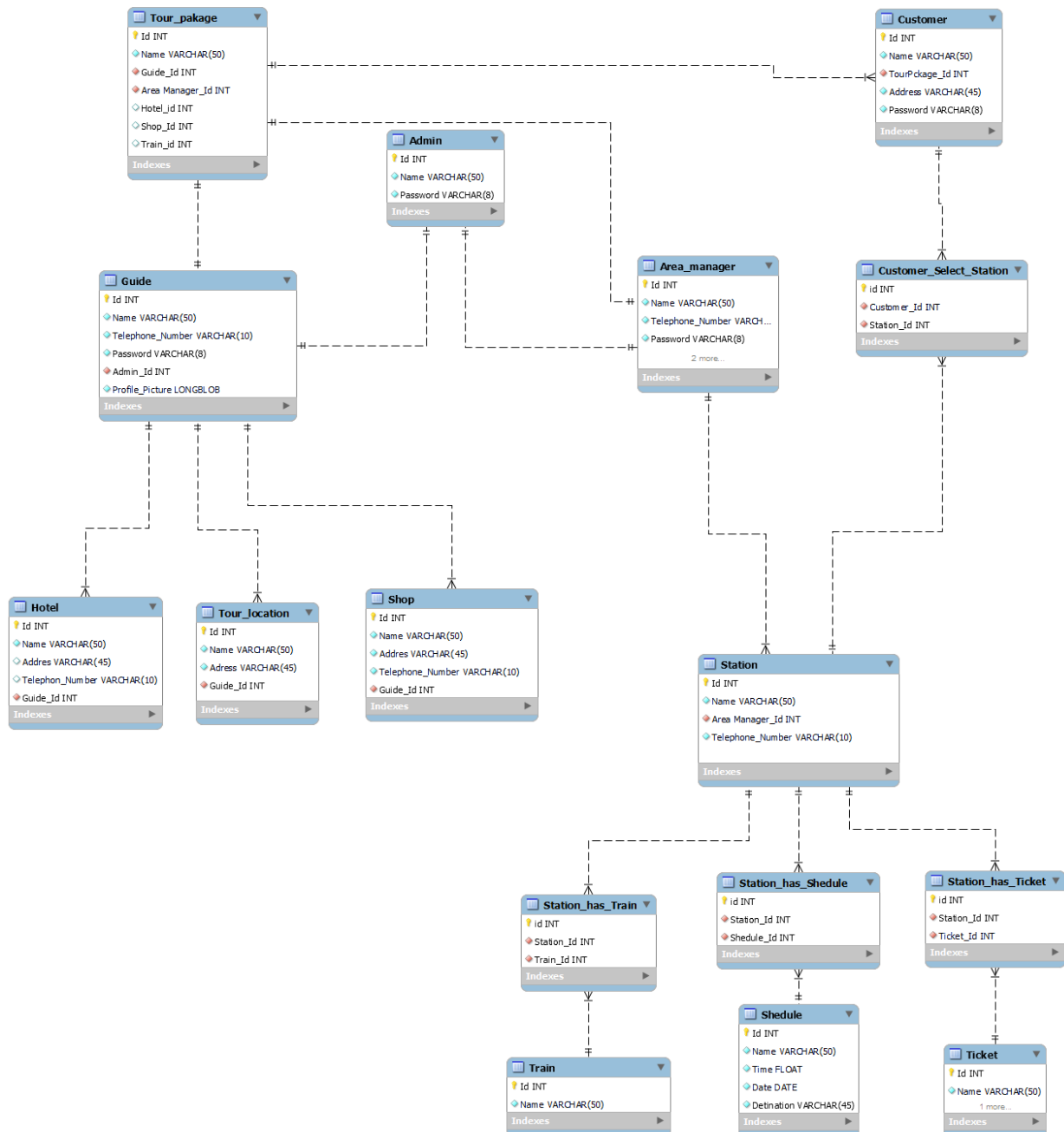


Figure 3.2: ER Diagram of Rail Guide Pro Project

The above ER diagram (Figure 3.2) depicts a database system that is used to manage a train schedule and tour destination service. It allows for storing information about customers, tour packages, guides, hotels, and other entities involved in booking tours.

### 3.5.1. Identification of Entities

The ER diagram you sent shows the following entities:

- Customer
- Tour Package
- Guide
- Hotel
- Station
- Shop
- Admin
- Train
- Schedule
- Ticket

### 3.5.2. Justification of Attributes

#### Customer

- **ID :** A unique identifier for the customer
- **Name:** Identifies the customer.
- **Tour Package:** Likely "Tour Package," stores the chosen package for booking.
- **Address:** Needed for delivery or contact purposes.
- **Telephone Number:** Alternative contact method.
- **Password:** Secures customer account access.
- **Customer Select Station:** Stores the chosen departure station for travel.
- **Profile Picture (optional):** Allows for personalization and identification

## **Admin**

- **ID:** A unique identifier for admin
- **Name:** Identifies the administrator.
- **Password:** Secures customer account access.

## **Guide**

- **ID:** A unique identifier for guide
- **Name:** Identifies the tour guide.
- **Telephone Number:** Enables communication with the guide.
- **Password:** Secures guide account access.

## **Area Manager:**

- **ID:** A unique identifier for area manager
- **Name:** Identifies the area manager.
- **Telephone Number:** Enables communication with the area manager.

## **Hotel:**

- **ID:** A unique identifier for hotel
- **Name:** Identifies the hotel.
- **Address:** Locates the hotel for customers.
- **Telephone Number:** Provides a contact method for the hotel.

**Tour Location:**

- **ID:** A unique identifier for tour location.
- **Name:** Identifies the tour destination.
- **Address:** Locates the tour destination.
- **Guide:** Links the location to the assigned guide.

**Shop:**

- **ID:** A unique identifier for shop.
- **Name:** Identifies the shop.
- **Address:** Locates the tour destination.
- **Telephone Number:** Provides a contact method for the shop.

**Station:**

- **ID:** A unique identifier for station.
- **Name:** Identifies the station.
- **Area Manager:** Links the station to the responsible manager.
- **Telephone Number:** Provides a contact method for the station.

**Train:**

- **ID:** A unique identifier for train.
- **Name:** Identifies the train.
- **Destination:** Indicates the train's final stop.



**Schedule:**

- **ID:** A unique identifier for schedule.
- **Name:** Identifies the specific schedule (e.g., morning, evening).
- **Time:** Departure or arrival time of the train.
- **Date:** Date of the train schedule.

**Ticket:**

- **ID:** A unique identifier for ticket.
- **Name:** Identifies the ticket (e.g., unique ticket number).
- **Destination:** Matches the destination of the chosen train.

### 3.5.3. Identification of Relationships

One-to-many relationship between Admin and Staff:

- Reason: An admin may manage multiple staff members, but each staff member reports to one admin. This relationship enables effective supervision and management of staff within the railway tourism system, ensuring clear lines of authority and responsibility.

One-to-many relationship between Guide and Shop/Restaurant:

- Reason: A guide may recommend or provide information about specific shops or restaurants to tourists during a tour. Each shop or restaurant may be associated with one or more guides. This relationship allows guides to assist tourists in finding suitable shopping or dining options during their tours.

#### Many-to-many relationship between Area Manager and Station

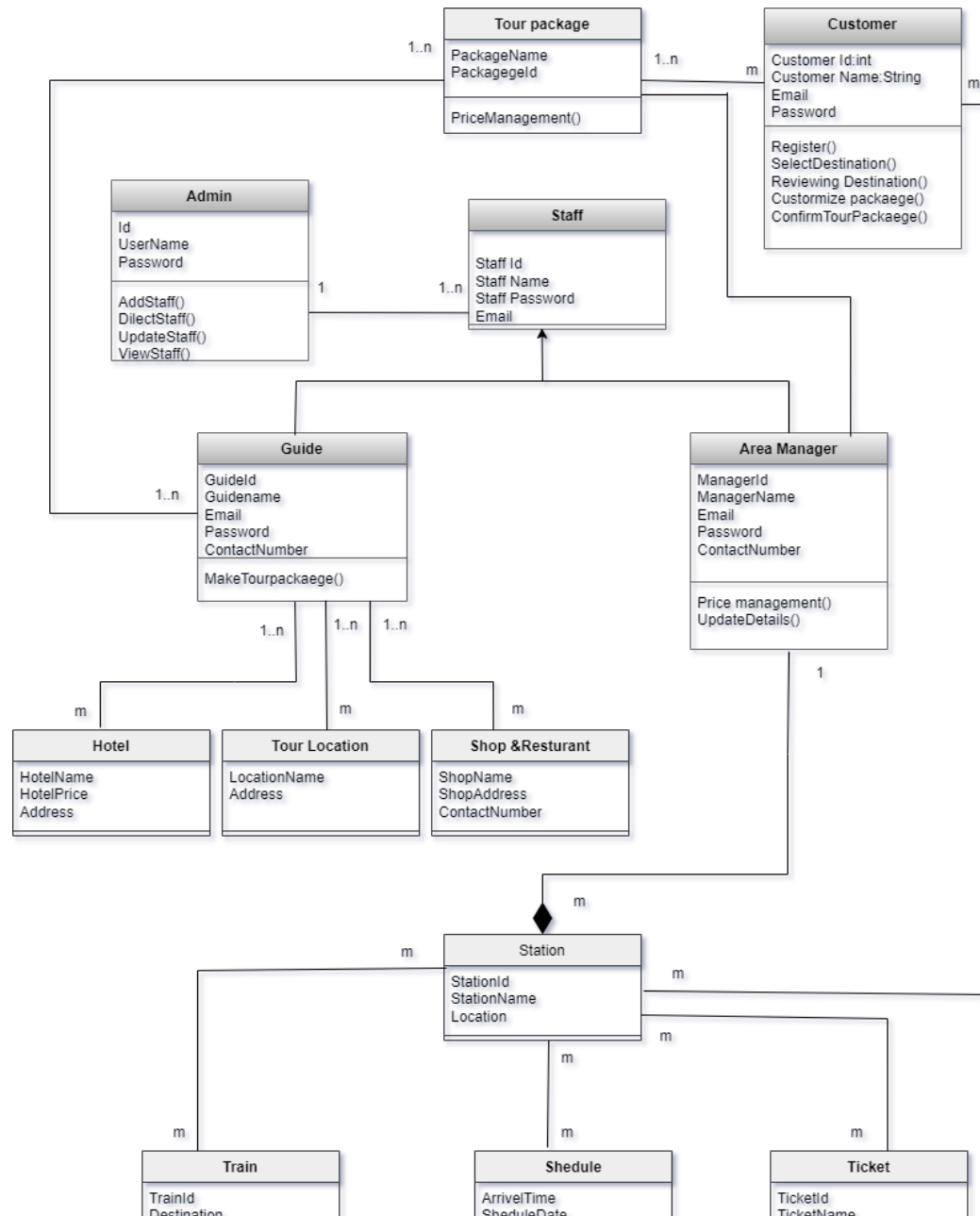
- An area manager can oversee operations at multiple stations, and each station can be supervised by multiple area managers. This relationship enables effective
- manage stations within specific geographical areas or regions by allowing multiple area managers to be responsible for various stations.

#### Many-to-many relationship between Guide and Tour Package:

- Reason: Guides may lead tours for multiple tour packages, and each tour package may be assigned to multiple guides. This relationship enables flexibility in assigning guides to different tour packages based on their availability, expertise, and tour requirements.

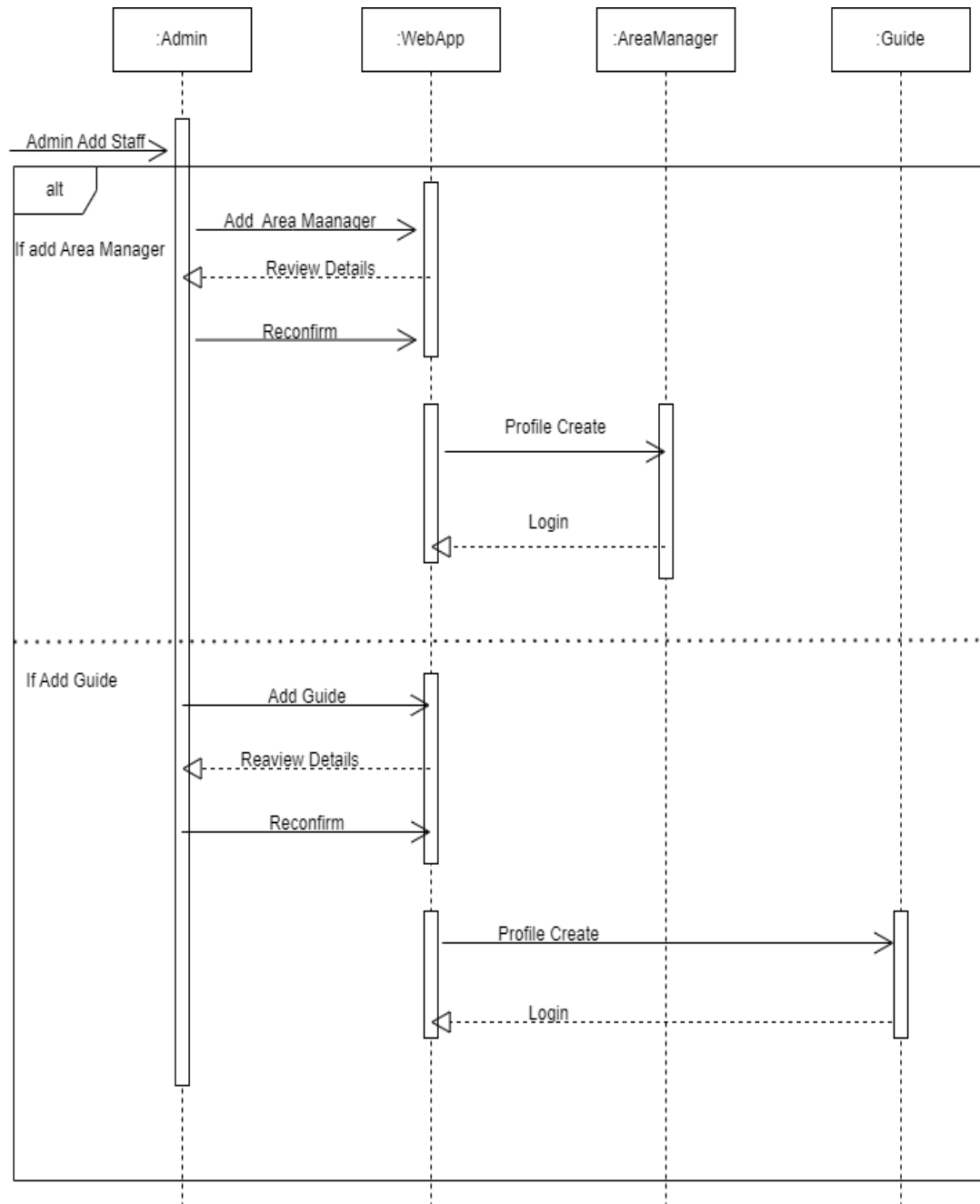
### 3.6. Class Diagram

A class diagram is a type of UML (Unified Modeling Language) diagram that represents the structure of a system by showing the classes, their attributes, methods, and the relationships between them. It's a visual representation of the static design of a system, focusing on the classes and their interactions.



### 3.7. Sequence Diagram

A sequence diagram is a type of UML (Unified Modeling Language) diagram that illustrates the interactions and messages exchanged between objects or components in a system over time. It represents the dynamic behavior of a system, showing the sequence of actions and the order in which they occur.



3.4:Sequence Diagram 01 of Rail Guide Pro Project

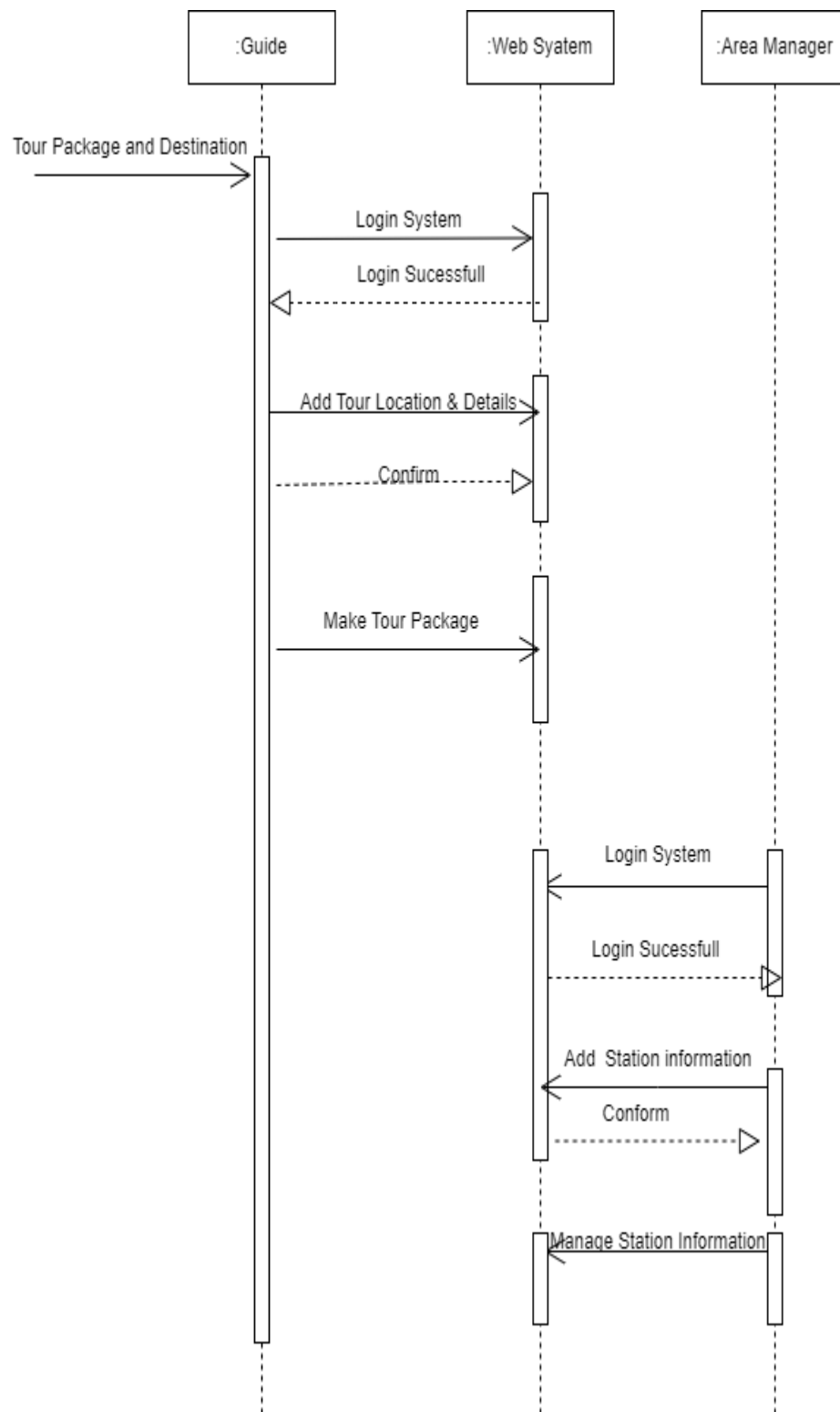


Figure 3.5:Sequence Diagram 02 of Rail Guide Pro Project

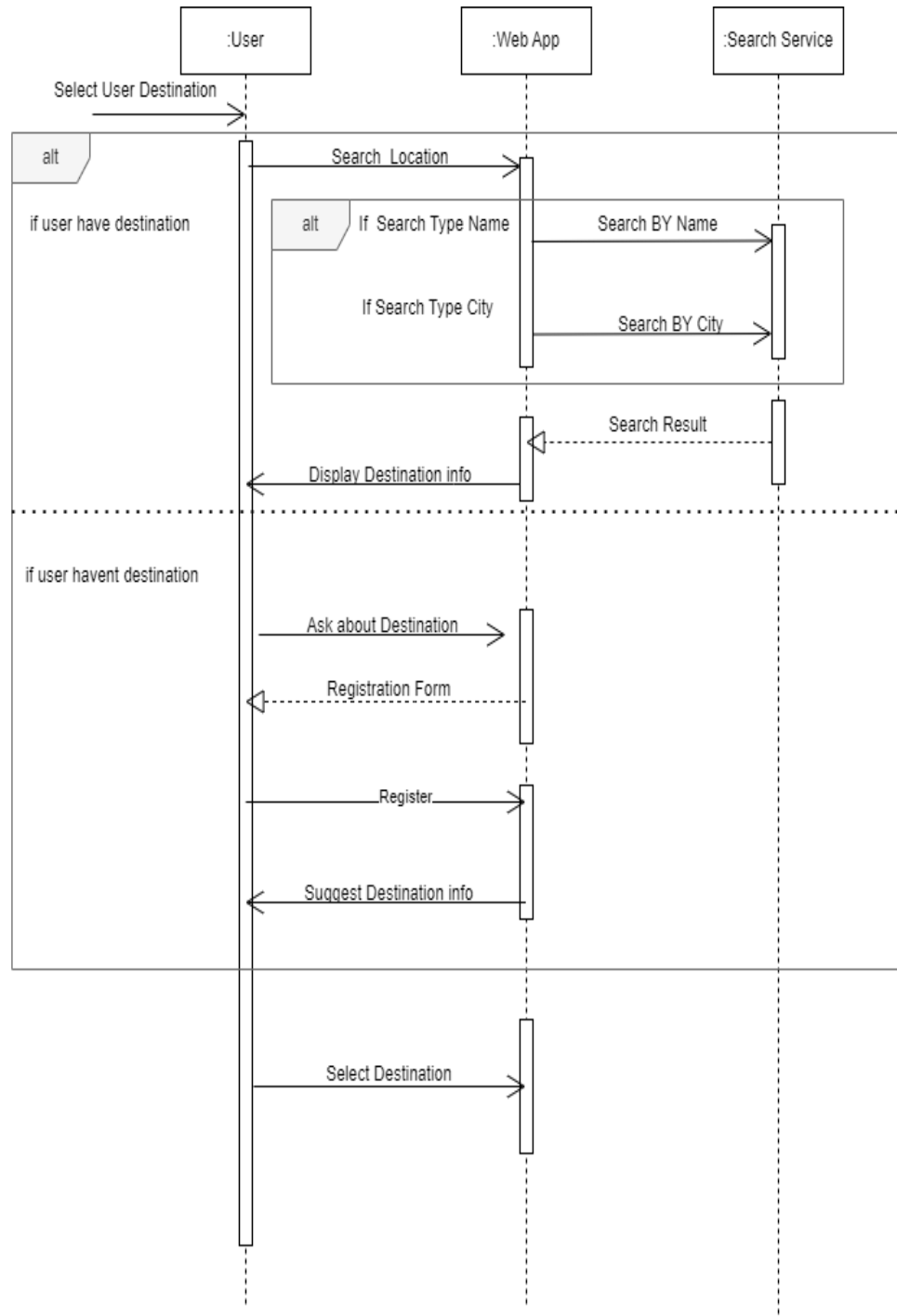


Figure 3.6::Sequence Diagram03 of Rail Guide Pro Project

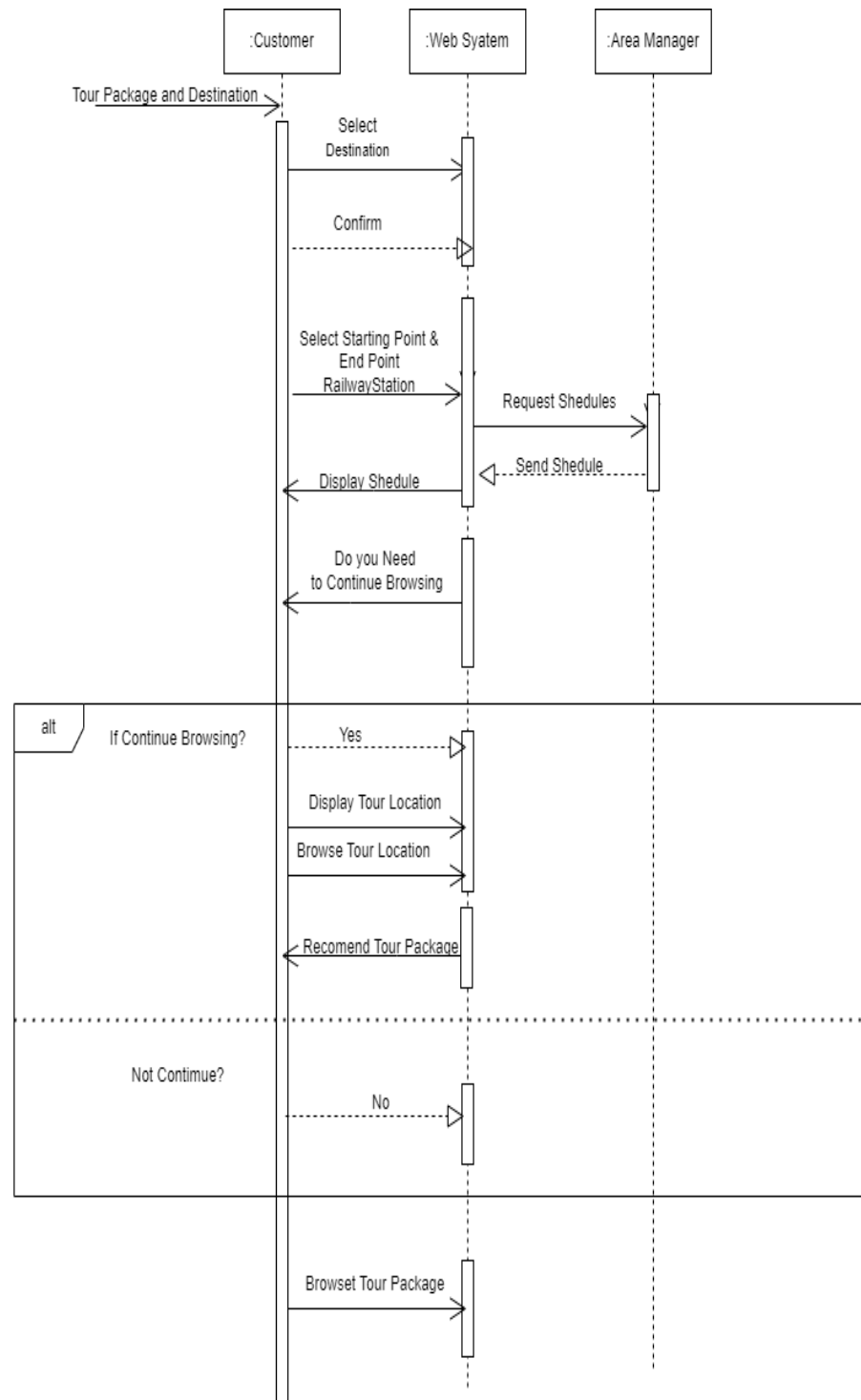


Figure 3.7:Sequence Diagram 04of Rail Guide Pro Project

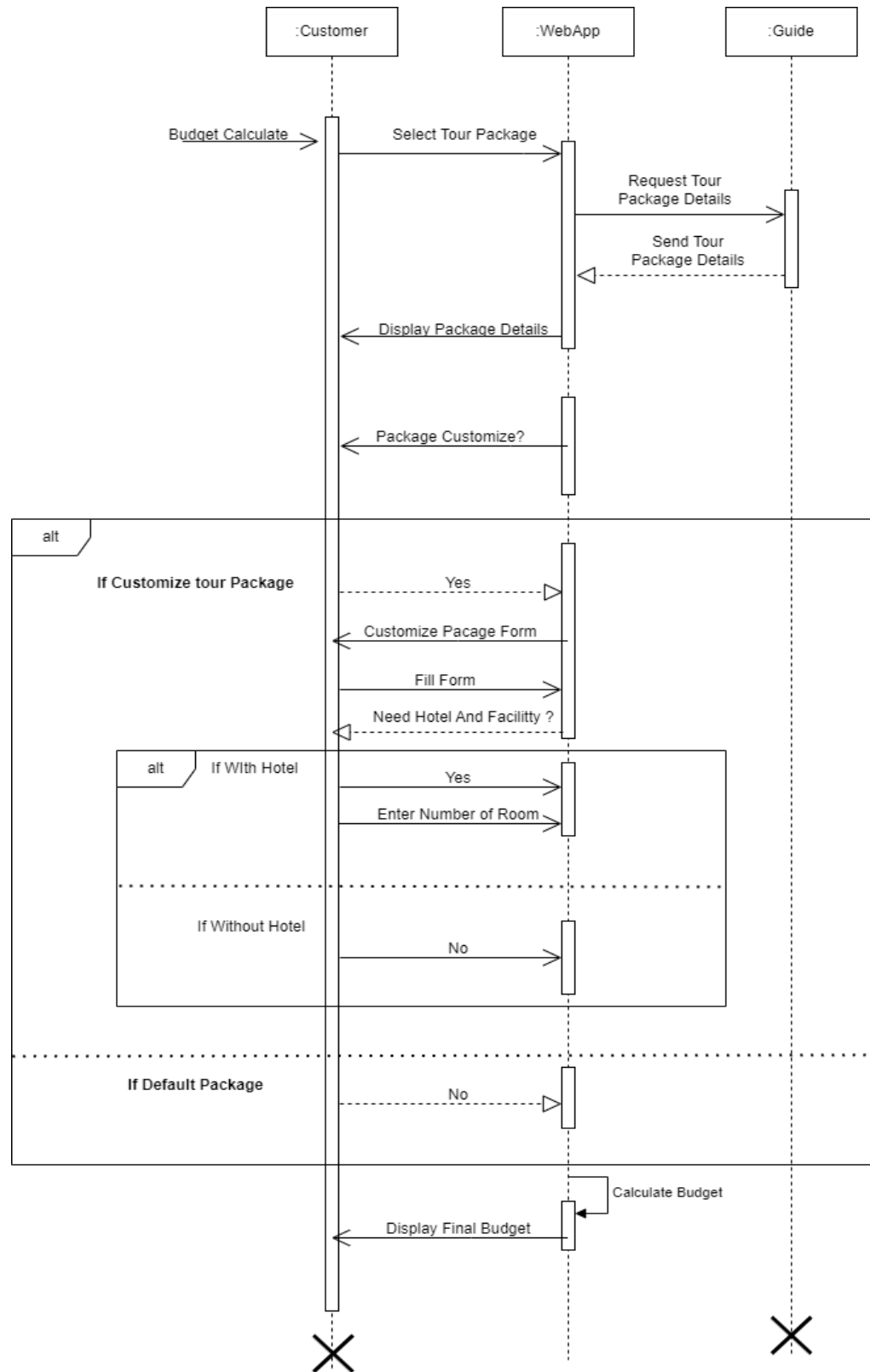


Figure 3.8: Sequence Diagram 05 of Rail Guide Pro Project



### 3.8. State Chart Diagram

A state chart diagram, also known as a state machine diagram, is a type of UML (Unified Modeling Language) diagram used to model the dynamic behavior of a system or an object over time. It represents the various states that an object can be in and the transitions between these states based on events or conditions.

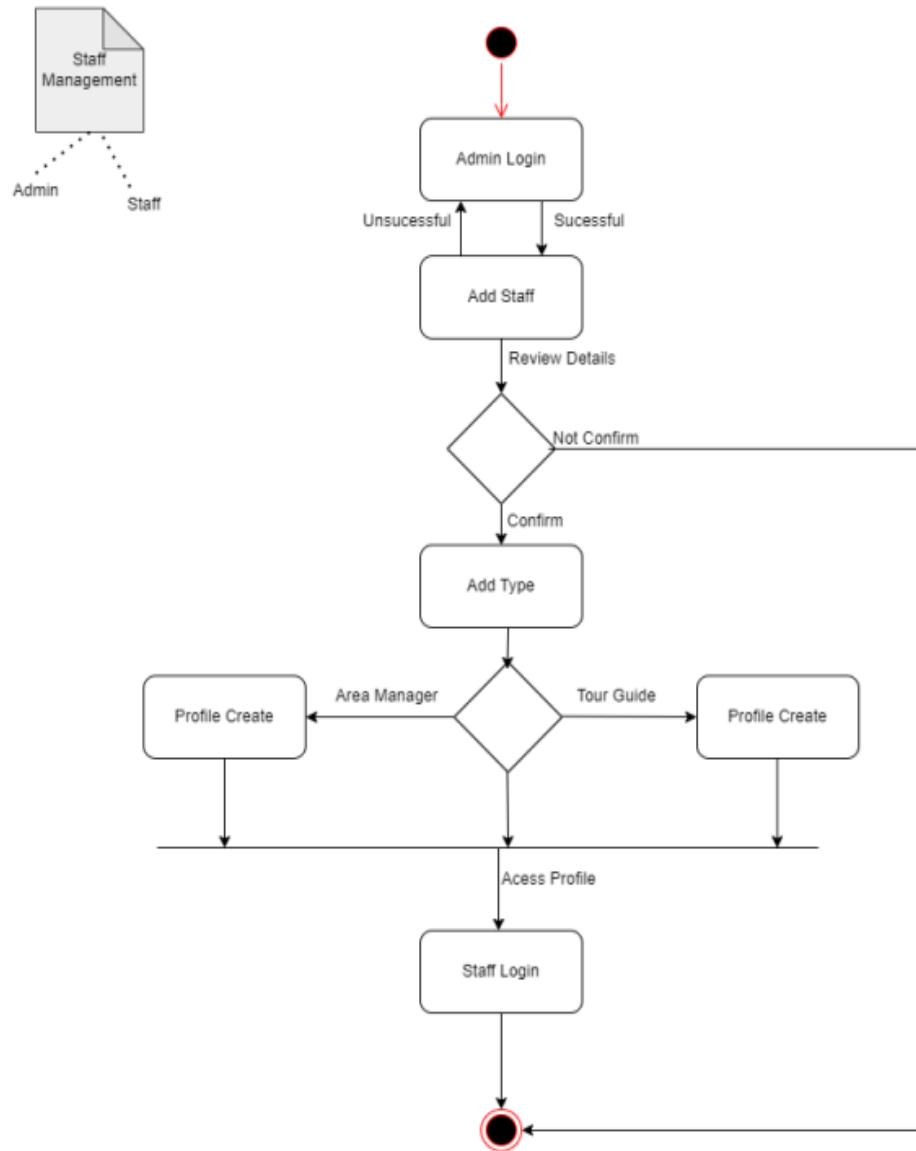


Figure 3.9:State Chart Diagram 01 of Rail Guide Pro Project

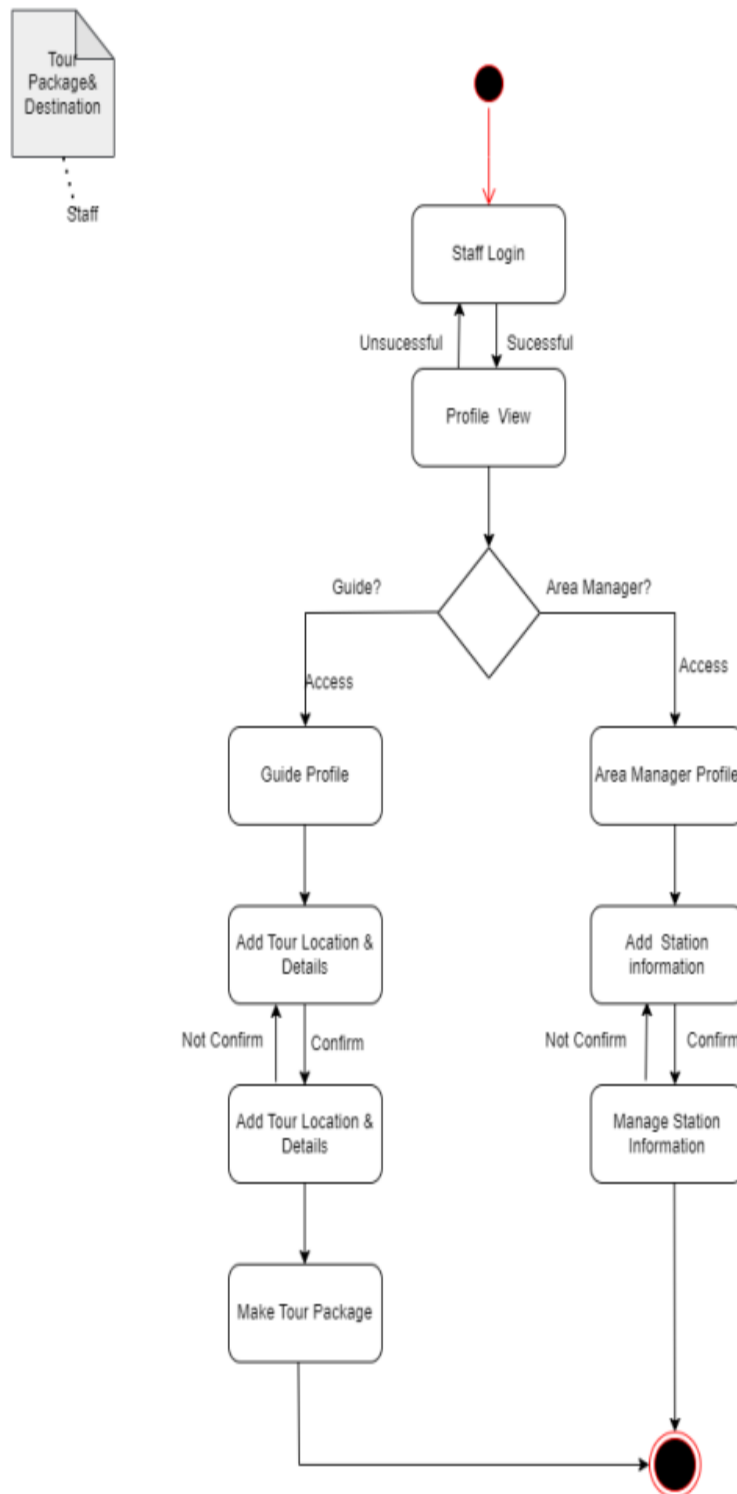


Figure 3.10: :State Chart Diagram 02 of Rail Guide Pro Project

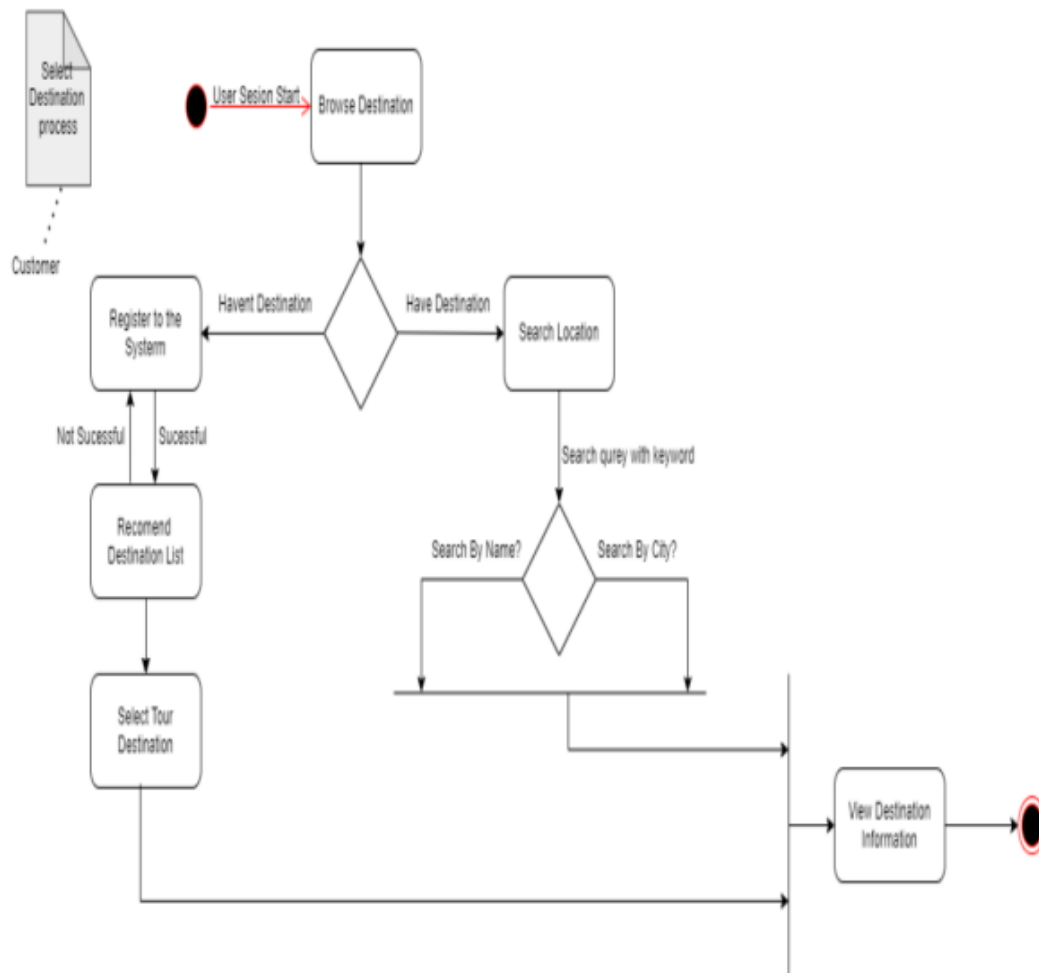


Figure 3.11: State Chart Diagram 03 of Rail Guide Pro Project

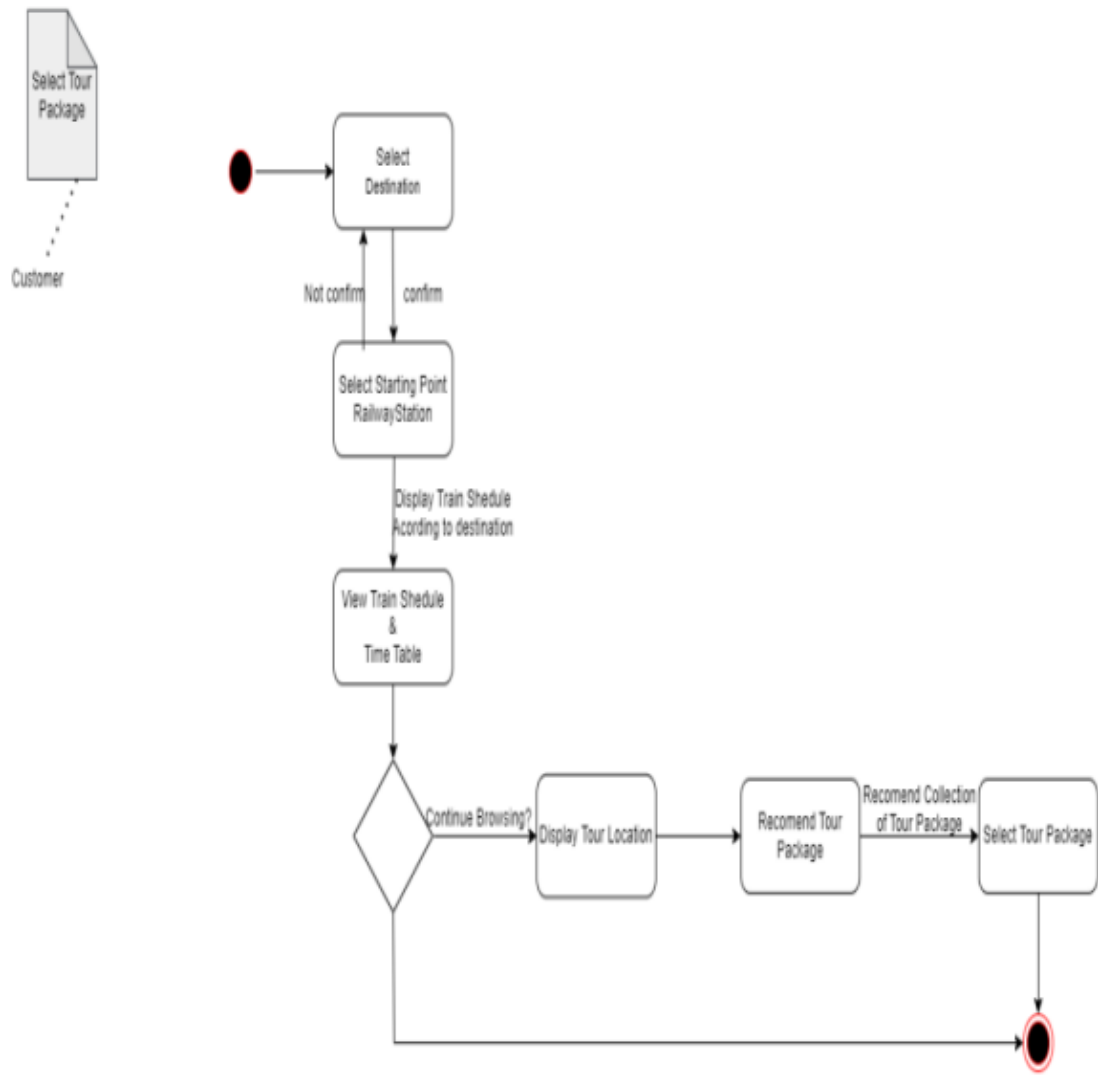


Figure 3.12: State Chart Diagram 04 of Rail Guide Pro Project

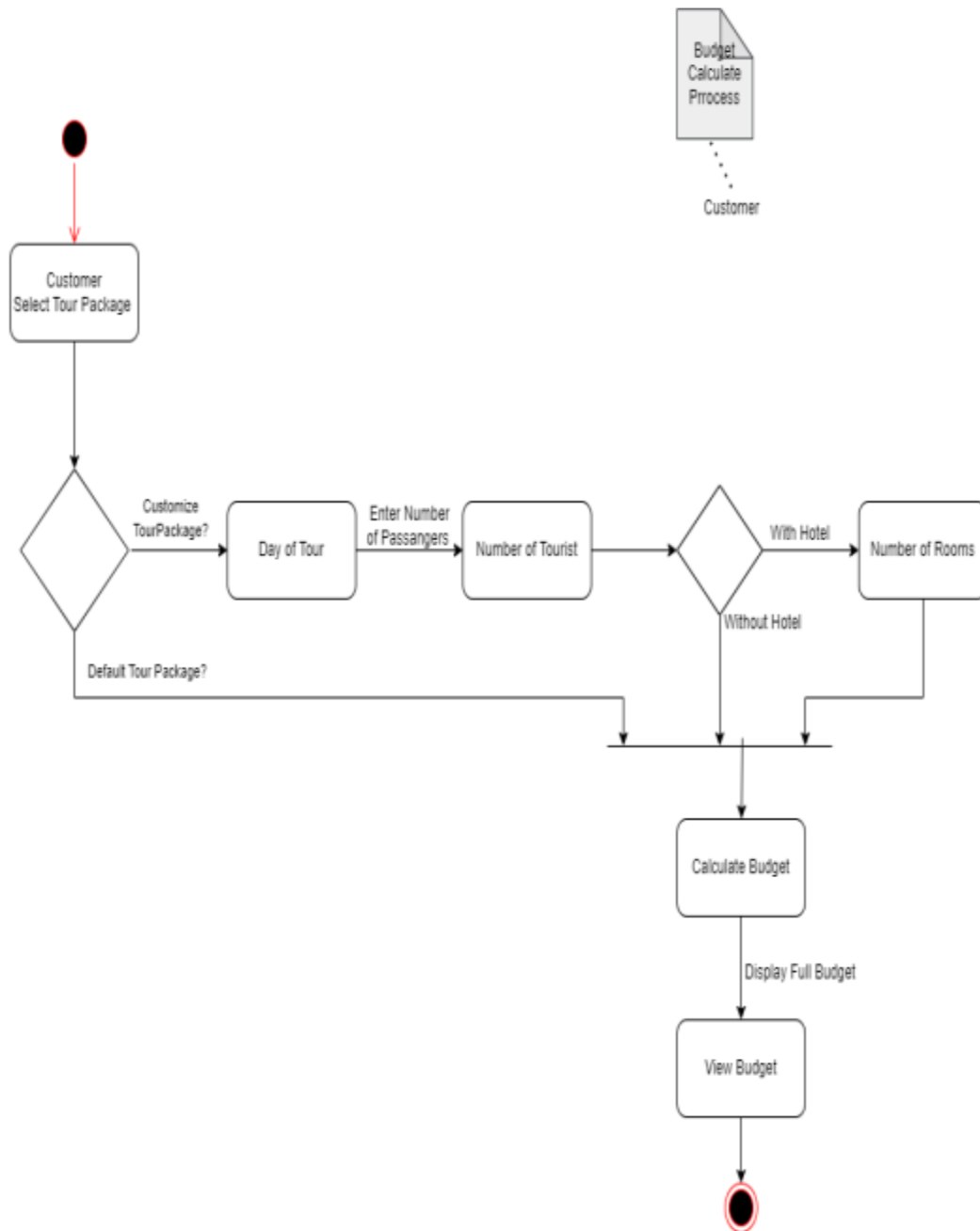


Figure 3.13:State Chart Diagram 05 of Rail Guide Pro Project

### 3.9. Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram that represents the flow of activities or actions within a system or a business process. It visually depicts the sequence of steps, decisions, and parallel or concurrent activities involved in completing a task or achieving a goal.

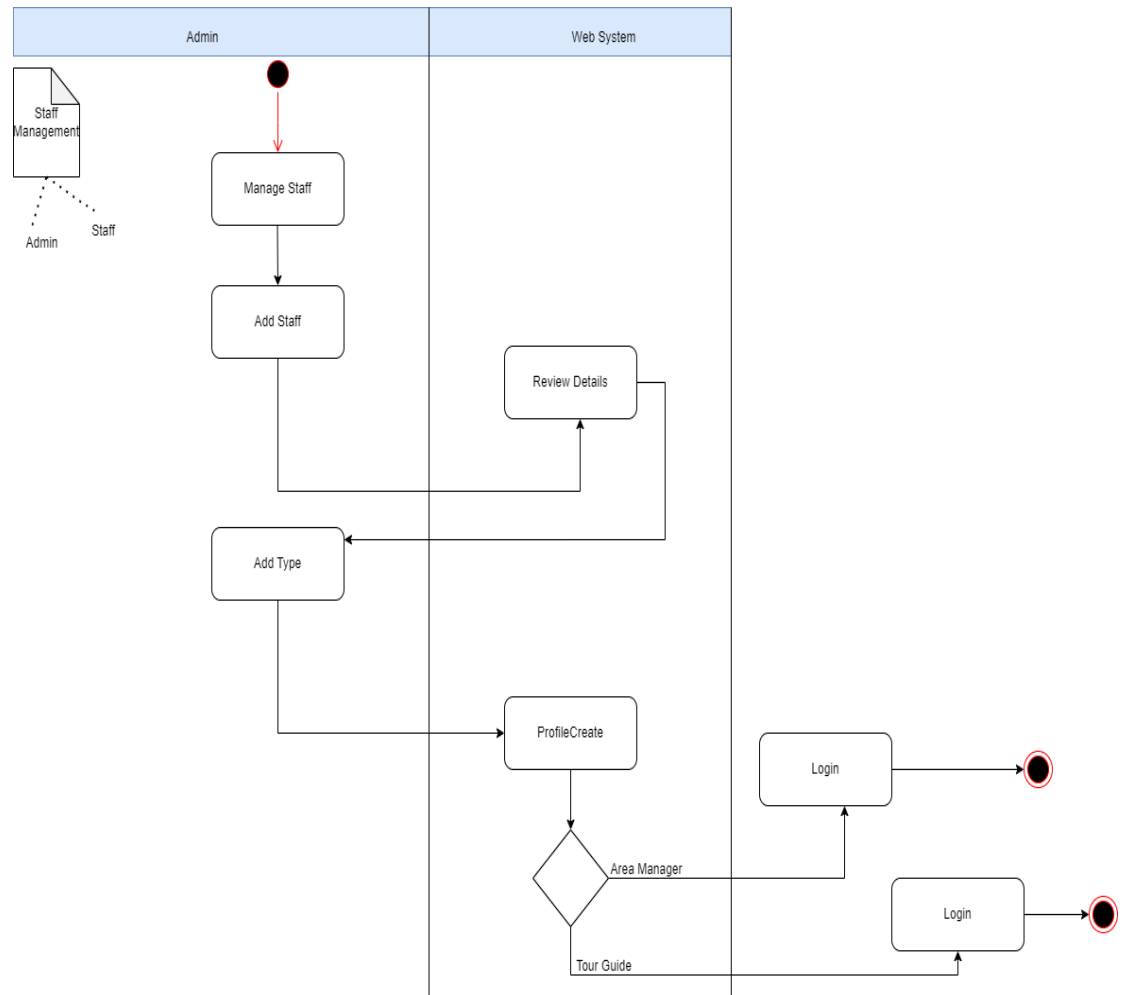


Figure 3.14:Activity Diagram 01 of Rail Guide Pro Project

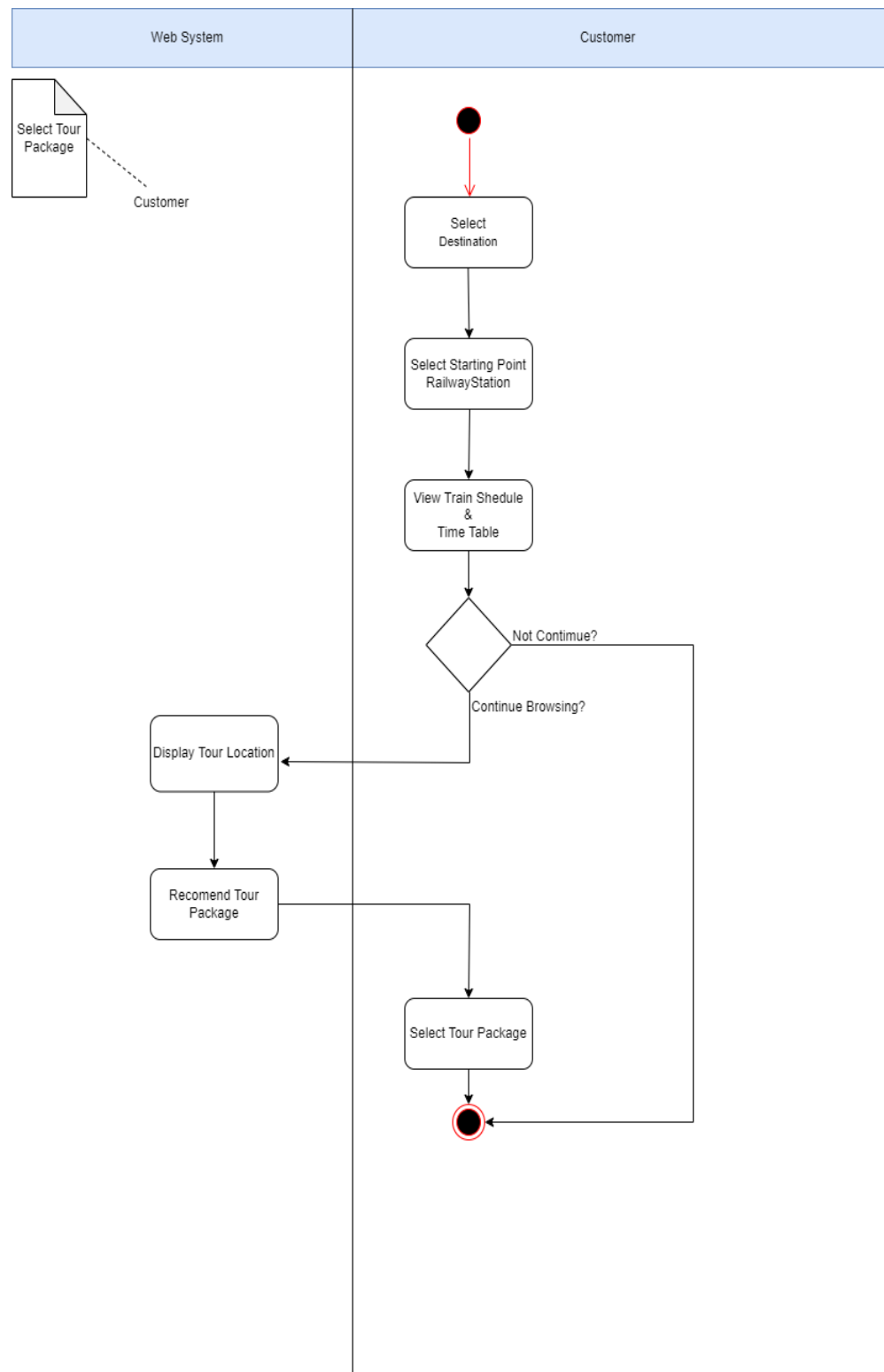


Figure 3.15:Activity Diagram 02 of Rail Guide Pro Project

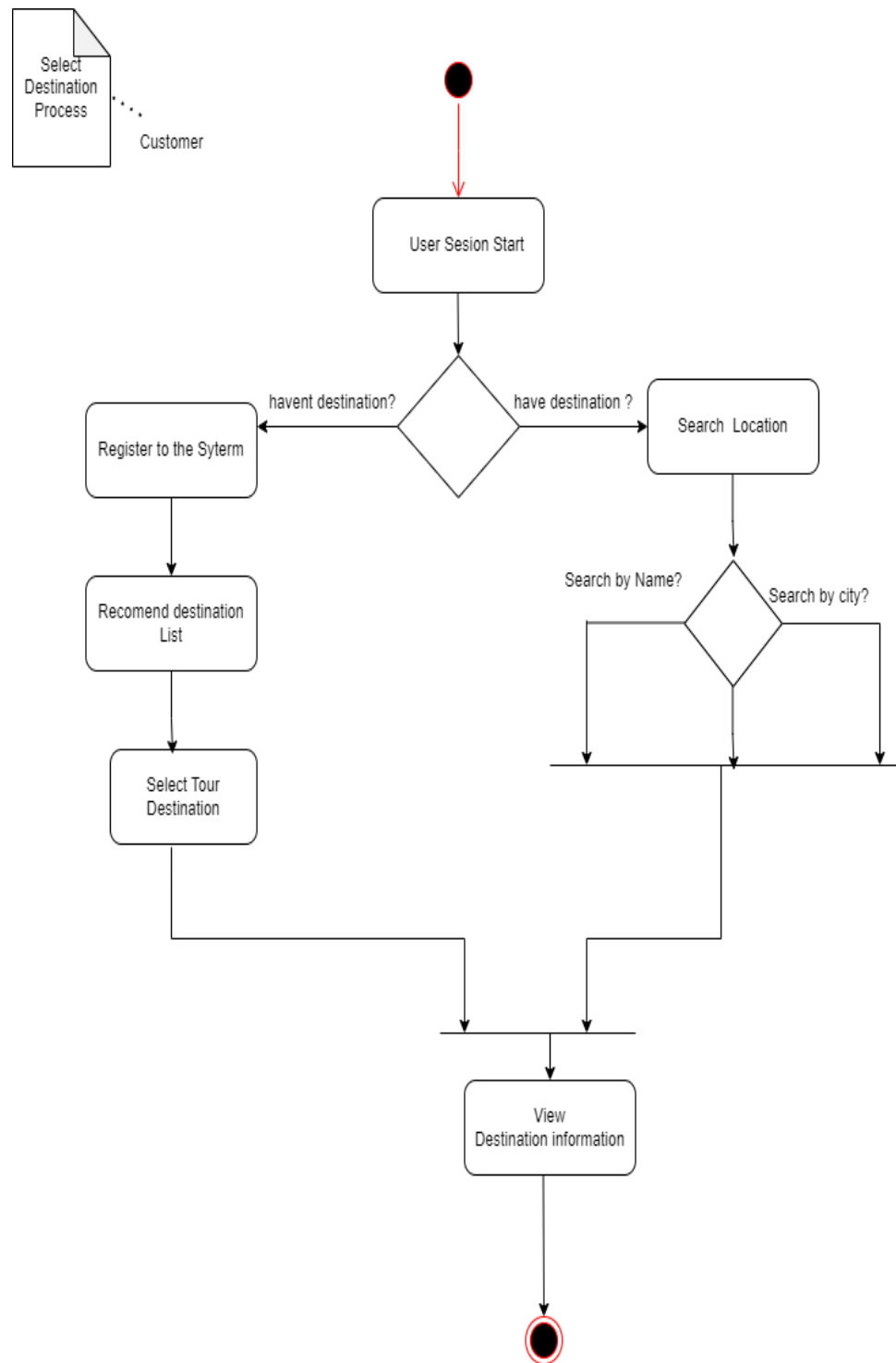


Figure 3.16:Activity Diagram 03 of Rail Guide Pro Project



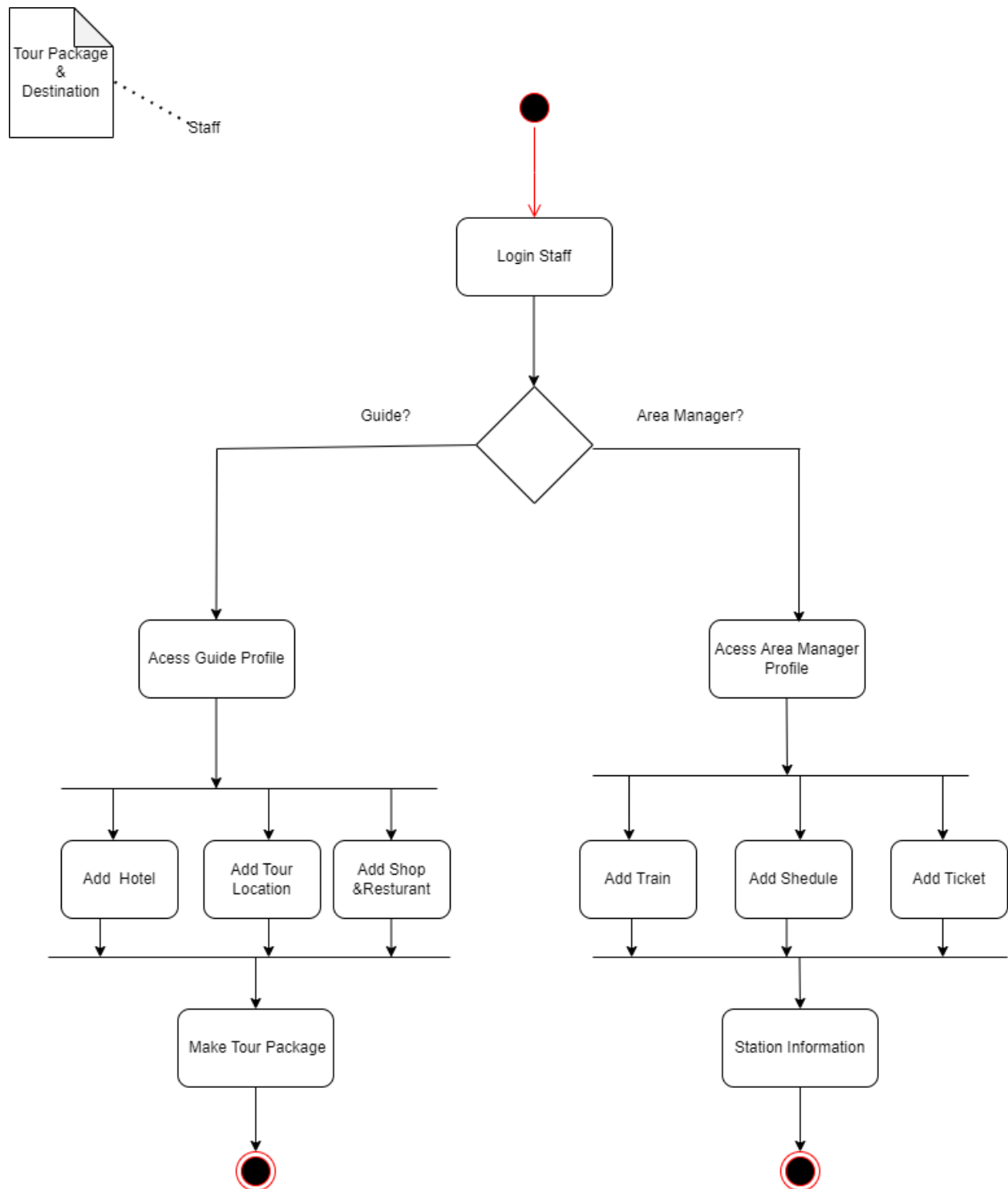


Figure 3.17:Activity Diagram 04 of Rail Guide Pro Project

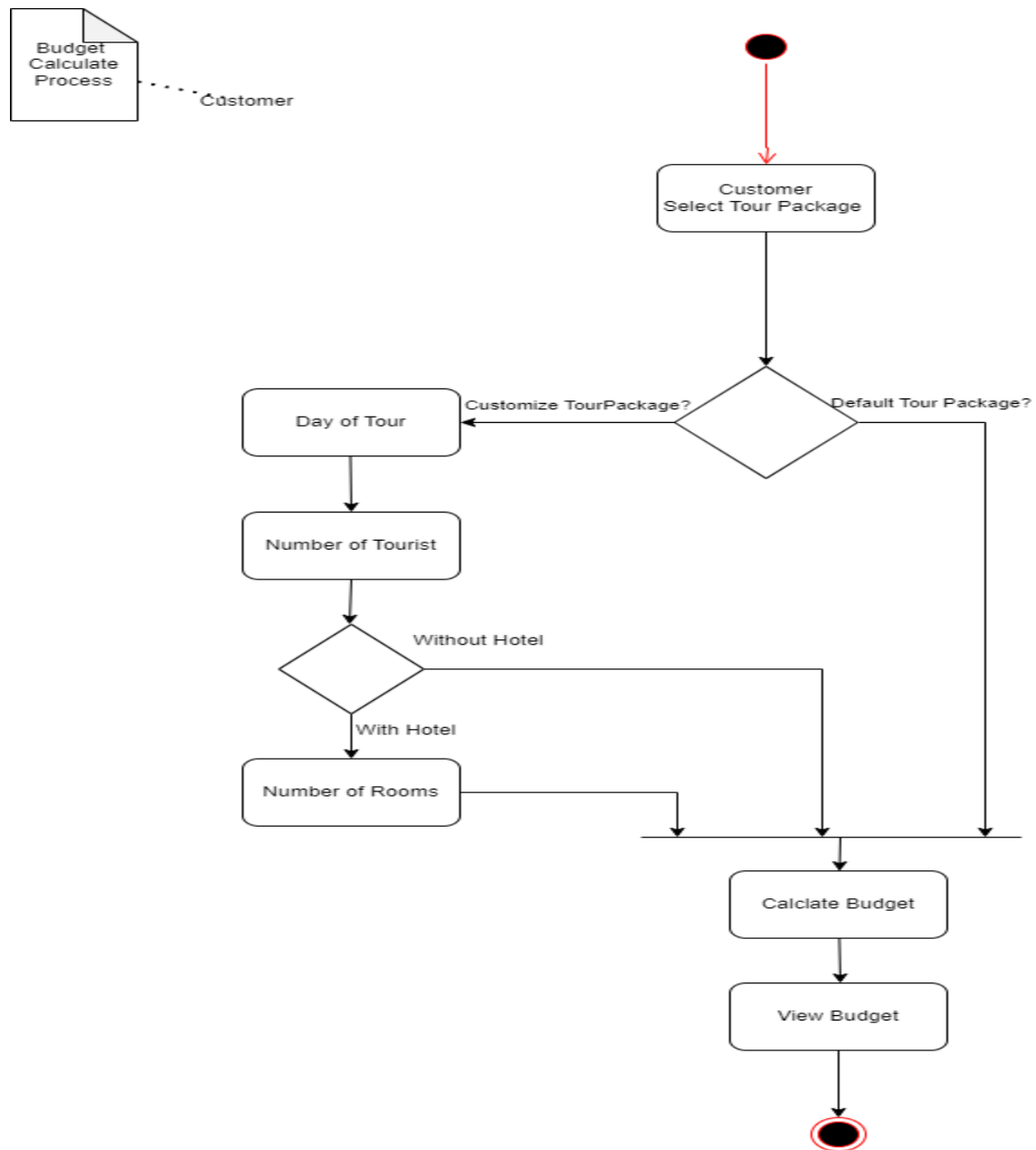


Figure 3.18:Activity Diagram 05 of Rail Guide Pro Project

### 3.10. Summary

The chapter concludes with a summary of the design phase, emphasizing the importance of visual models in conceptualizing and defining the Railway Tourism Management System's architecture and functionality.

## 4. Chapter 4: Implementation

### 4.1. Introduction

Chapter 4 delves into the implementation phase of the Railway Tourism Management System. This chapter focuses on actualizing the design and functionalities outlined in the previous chapters through SQL queries, test plans, and user interface implementations.

### 4.2. SQL Queries

#### 4.2.1. DML

The Data Manipulation Language (DML) queries allow for data retrieval and manipulation within the system's database. Key queries and their purposes are detailed, including JOIN operations to fetch information from related tables.

### 4.2.2. Test Plan

Table 4.1: Test plan

Project Name	Tourism Railway Guide System
Test Plan ID	TRGD01
Brief Introduction about the system.	In response to the changing landscape of travel preferences, this thesis explores the development and implementation of a Railway Tourism Management System. Focused on enhancing the railway tourism experience, the project addresses existing challenges by providing a centralized platform for efficient attractions management, an enhanced guide system for tourists, optimized navigation features, and administrative tools
Test Objectives	<ol style="list-style-type: none"><li>1. Show Train Arrival Station Information details</li><li>2. Shows Hotel Details Who Add to System</li><li>3. Display Customer data by selecting specific hotels, shops, Train</li><li>4. View Destination with Schedule</li><li>5. View Station Location</li></ol>
Features to be tested	Manage Staff
Test Environment	<ul style="list-style-type: none"><li>• Devices - Laptop</li><li>• Software – MYSQL Workbench</li></ul>
Test Approach	<ul style="list-style-type: none"><li>• Testing Levels: System testing, acceptance testing</li><li>• Testing Methods: Manual testing</li><li>• Testing Tools: Junit</li></ul>
Testing Tasks	<ul style="list-style-type: none"><li>• Test Planning: Define test objectives, identify test scenarios, and plan test execution.</li><li>• Test Design: Create test cases for each identified test scenario.</li><li>• Test Execution: Execute test cases and scripts in the test environment.</li></ul>

## Query 01

- ```
SELECT station.*, train.*  
FROM station  
INNER JOIN station_has_train ON station.id = station_has_train.station_id  
INNER JOIN train ON station_has_train.train_id = train.id;
```

Figure 4.1: Query 01 of Rail Guide Pro Project

### FROM station

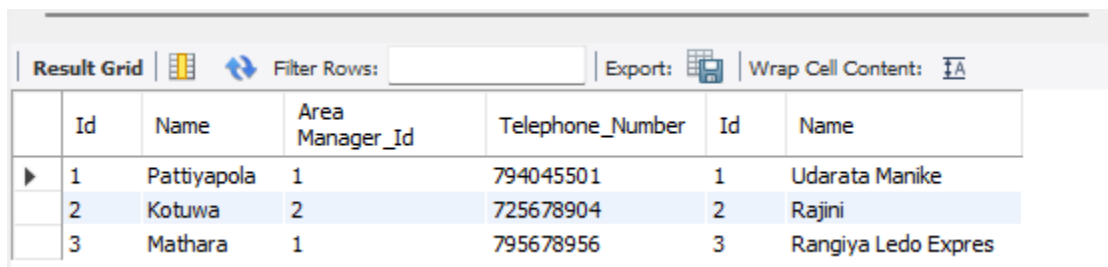
- This specifies that we are selecting data from the station table.

### INNER JOIN station\_has\_train ON station.id = station\_has\_train.station\_id

- This part of the query performs an inner join between the station table and the station\_has\_train table based on the condition that the id column in the station table matches the station\_id column in the station\_has\_train table.

### INNER JOIN train ON station\_has\_train.train\_id = train.id;

- This further joins the result of the previous join with the train table based on the condition that the train\_id column in the station\_has\_train table matches the id column in the train table.



|   | Id | Name        | Area Manager_Id | Telephone_Number | Id | Name                |
|---|----|-------------|-----------------|------------------|----|---------------------|
| ▶ | 1  | Pattiyapola | 1               | 794045501        | 1  | Udarata Manike      |
|   | 2  | Kotuwa      | 2               | 725678904        | 2  | Rajini              |
|   | 3  | Mathara     | 1               | 795678956        | 3  | Rangiya Ledo Expres |

Figure 4.2: Query 01 Result of Rail Guide Pro Project

## Test Case 01

Table 4.2: Test Case 01

| Test Case 01                                    |                |
|-------------------------------------------------|----------------|
| Show Train Arrival Station Information details  | Piyumi Natasha |
| 01                                              | Black box      |
| Show Train arrival station Information for user | 2024.02.24     |
| Staff Details                                   | 3.00 am        |

Table 4.3: Test Case 01 Result

| Step No | Test Step    | Test Case ID | Test Input                                                                                                                                                                                                  | Expected Result                            | Actual result                              | Test result |
|---------|--------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|--------------------------------------------|-------------|
| 01      | Enter Values | 01           | SELECT station.*, train.*<br>FROM station<br><br>INNER JOIN<br>station_has_train ON<br>station.id =<br>station_has_train.station_id<br><br>INNER JOIN train ON<br>station_has_train.train_id =<br>train.id; | Display<br>train<br>Arrival<br>Information | Display<br>train<br>Arrival<br>Information | Pass        |

## Query 02

- ```
SELECT hotel.*, guide.*  
FROM hotel  
INNER JOIN guide ON hotel.Guide_Id = guide.Id;
```

Figure 4.3: Query 02 of Rail Guide Pro Project

SELECT hotel.\*, guide.\*

- This part of the query specifies which columns to select from the tables hotel and guide. The asterisk \* is a wildcard character that means "all columns". So, this query will select all columns from both the hotel and guide tables.

FROM hotel

- This part of the query indicates that the data will be retrieved from the hotel table.

INNER JOIN guide ON hotel.Guide\_Id = guide.Id;

- This part of the query specifies how the tables should be joined. It performs an inner join between the hotel table and the guide table. The condition for the join is that the value of the Guide\_Id column in the hotel table must match the value of the Id column in the guide table. This means that the query will only include rows where there is a matching Id value in both tables.

Result Grid											
		Filter Rows:		Export:		Wrap Cell Content:					
	Id	Name	Address	Telephon_Number	Guide_Id	Id	Name	Telephone_Number	Password	Admin_Id	Profile_Picture
▶	1	Suba Hotel	186/2 Kottawa road Pannipitiya	7635084781	1	1	Piyumi	762653301	1234	1	BLOB
	2	Priya Hotel	21/3 Rathnapura road Embilipitiya	7046679865	2	2	Natasha	786548903	1234	1	BLOB
	3	Hotel King	8016 Maharagama road Nugegoda	7064679669	1	1	Piyumi	762653301	1234	1	BLOB

Figure 4.4: Query 02 Result of Rail Guide Pro Project



## Test Case 02

Table 4.4: Test Case 02

Test Case 02	
Shows Hotel Details Who Add to System	Piyumi Natasha
02	Black box
In this query display Guide details Who add the Hotel	2024.02.24
Guide Details	3.00 am

Table 4.5: Test Case 02 Result

Step No	Test Step	Test Case ID	Test Input	Expected Result	Actual result	Test result
02	Enter Values	02	SELECT hotel.*, guide.*  FROM hotel  INNER JOIN guide ON hotel.Guide_Id = guide.Id;	Display Guide details Who add the Hotel	Display Guide details Who add the Hotel	Pass

## Query 03

```
• SELECT customer.id, customer.name, tour_package.*  
FROM customer  
INNER JOIN tour_package ON customer.Tourpackage_Id = tour_package.Id;
```

Figure 4.5: Query 03 of Rail Guide Pro Project

SELECT customer.id, customer.name, tour\_package.\*

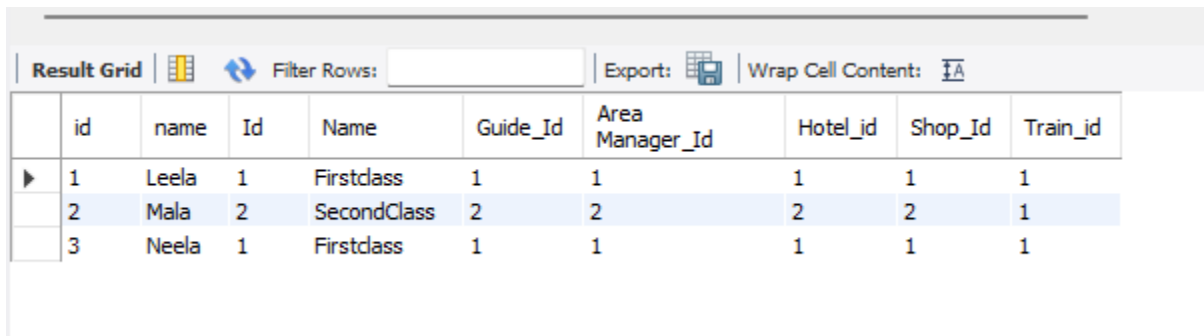
- This part of the query specifies which columns to select. It selects the id and name columns from the customer table, and all columns (\*) from the tour\_package table.

FROM customer

- This part of the query specifies the table from which to retrieve data, which is the customer table.

INNER JOIN tour\_package ON customer.Tourpackage\_Id = tour\_package.Id;

- This part of the query performs an inner join between the customer table and the tour\_package table. The condition for the join is that the Tourpackage\_Id column in the customer table must match the Id column in the tour\_package table. This means that the query will only include rows where there is a matching Id value in both tables. In this query allows an admin to access information about tourist attractions places and hotel packages,



The screenshot shows a database query result grid. At the top, there are controls: 'Result Grid' with a grid icon, a 'Filter Rows:' input field, an 'Export:' button with a document icon, and a 'Wrap Cell Content:' button with a text icon. Below these controls is a table with 10 columns: 'id', 'name', 'Id', 'Name', 'Guide\_Id', 'Area Manager\_Id', 'Hotel\_id', 'Shop\_Id', and 'Train\_id'. The table contains three rows of data. The first row has values: 1, Leela, 1, Firstclass, 1, 1, 1, 1, 1. The second row has values: 2, Mala, 2, SecondClass, 2, 2, 2, 2, 1. The third row has values: 3, Neela, 1, Firstclass, 1, 1, 1, 1, 1.

	id	name	Id	Name	Guide_Id	Area Manager_Id	Hotel_id	Shop_Id	Train_id
▶	1	Leela	1	Firstclass	1	1	1	1	1
	2	Mala	2	SecondClass	2	2	2	2	1
	3	Neela	1	Firstclass	1	1	1	1	1

Figure 4.6: Query 03 Result of Rail Guide Pro Project

### Test Case 03

Table 4.6:Test Case 03

Test Case 03	
Display Customer data by selecting specific tour package	Piyumi Natasha
03	Black box
Display Customer details	2024.02.24
Customer Details	3.00 am

Table 4.7:Test Case 03 Result

Step No	Test Step	Test Case ID	Test Input	Expected Result	Actual result	Test result
03	Enter Values	03	<pre>SELECT customer.id, customer.name, tour_package.* FROM customer INNER JOIN tour_package ON customer.Tourpckage_Id = tour_package.Id;</pre>	Display Customer data by selecting specific hotels, shops, Train	Display Customer data by selecting specific hotels, shops, Train	Pass

## Query 04

```
SELECT shedule.*, station.Name
FROM shedule
INNER JOIN station_has_shedule ON shedule.Id = station_has_shedule.Shedule_Id
INNER JOIN station ON station_has_shedule.Station_Id = station.Id;
```

Figure 4.7: Query 04 of Rail Guide Pro Project

SELECT shedule.\*, station.Name

- This part of the query specifies which columns to select. It selects all columns (\*) from the shedule table and the Name column from the station table.

FROM shedule

- This part of the query specifies the table from which to retrieve data, which is the shedule table.

INNER JOIN station\_has\_shedule ON shedule.Id = station\_has\_shedule.Shedule\_Id

- This part of the query performs an inner join between the shedule table and the station\_has\_shedule table. The condition for the join is that the Id column in the shedule table must match the Shedule\_Id column in the station\_has\_shedule table.

INNER JOIN station ON station\_has\_shedule.Station\_Id = station.Id;

This part of the query further joins the result of the previous join with the station table. The condition for this join is that the Station\_Id column in the station\_has\_shedule table must match the Id column in the station table.



Result Grid						
				Filter Rows:		Export:  Wrap Cell Content: 
	Id	Name	Time	Date	Detination	Name
▶	1	Sunday	03:30:00	2024-02-23	Kandy	Kotuwa
	2	Vacation	05:45:00	2024-02-25	Colombo	Mathara
	3	Monday	09:10:00	2024-02-26	Matara	Pattiyapola

Figure 4.8:Query 04 Result of Rail Guide Pro Project

## Test Case 04

Table 4.8:Test Case 04

Test Case 04	
View Destination with Schedule	Piyumi Natasha
04	Black box
View Destination with Schedule details arrive train	2024.02.24
Train Details	3.00 am

Table 4.9:Test Case 04 Result

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
04	Enter Values	04	SELECT customer.id, customer.name, tour_package.*  FROM customer  INNER JOIN tour_package ON customer.Tourpckage_Id = tour_package.Id;	View Tour Schedule With Tain Arrival to Station	View Tour Schedule With Tain Arrival to Station	Pass

## Query 05

```
• SELECT *  
FROM station  
WHERE Name = 'Pattiyapola';
```

Figure 4.9: Query 05 of Rail Guide Pro Project

SELECT \*

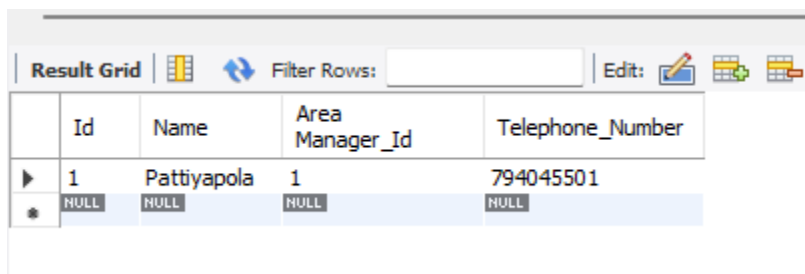
- This part of the query specifies that all columns should be selected.

FROM station

- This part of the query specifies the table from which to retrieve data, which is the station table.

WHERE Name = 'Pattiyapola';

- This part of the query filters the rows to only include those where the value in the Name column is equal to 'Pattiyapola'.
- output of this query will be all columns of the station table for the row(s) where the station name is 'Pattiyapola'. It will contain information about the station(s) with the name 'Pattiyapola'.



	Id	Name	Area Manager_Id	Telephone_Number
▶	1	Pattiyapola	1	794045501
★	NULL	NULL	NULL	NULL

Figure 4.10: Query 05 Result of Rail Guide Pro Project

## Test Case 05

Table 4.10: Test Case 05

Test Case 05	
View Station Location	Piyumi Natasha
05	Black box
Display Station Location according to customer need.	2024.02.24
Station Details	3.00 am

Table 4.11: Test Case 05 Result

Step No	Test Step	Test Case ID	Test Input	Expected Result	Actual result	Test result
01	Insert Station Details	01	City:Pattiyapola	Display the Pattiyapola station information	Display the Pattiyapola station information	Pass

## I. Login Form Interface



The image shows a login form for 'RAILWAY GUIDE PRO LOGIN ADMIN'. The form is set against a solid green background. At the top, the text 'RAILWAY GUIDE PRO LOGIN' is centered in a bold, black, sans-serif font. Below it, the word 'ADMIN' is also centered in a smaller, bold, black, sans-serif font. The form contains two input fields: 'User Name:' and 'Password'. Both labels are in a bold, black, sans-serif font and are positioned to the left of their respective input boxes. The input boxes are white with a subtle blue gradient and a thin black border. Below the input fields is a red rectangular button with the word 'LOGIN' in a bold, black, sans-serif font, centered on the button.

*Figure 4.11:Login Administrator*

An admin login interface is a user interface specifically designed for administrators or privileged users to securely authenticate themselves and gain access to the admin dashboard or administrative features of a system, application, or website.



## II. Admin Dashboard



**RAILWAY GUIDE PRO ADMIN DASHMASTER**

*Staff Register*

Full Name:

Email Address:

Password:

User Roles:

id	name	email	role
1	ss	natsha	s
5	ss	natsha	s

SEARCH

ADD VIEW UPDATE DELETE EXIT

Figure 4.12:Admin Dashboard

An admin dashboard is a user interface specifically designed for administrators or privileged users to manage and oversee various aspects of a system, application, or website. Here's a brief explanation of what an admin dashboard typically includes and its functionalities: Admin dashboards often include features for managing Staff accounts, such as creating new accounts, updating user information, resetting passwords, and managing user permissions or roles.

### 3.Functional Requirements:

#### User Authentication:

- The admin should be able to log in securely using their username and password.

#### Role Management:

- The admin should be able to define different roles for staff members, such as tour guides, Area Manager.
- They should have the ability to assign or revoke roles for staff members as needed.

### 4.Non-Functional Requirements:

#### Compatibility:

- The admin interface should be compatible with a range of devices and web browsers to accommodate different user preferences and accessibility needs.
- It should adhere to web standards and accessibility guidelines to ensure compatibility with assistive technologies.

#### Scalability:

- The system should be designed to accommodate a growing number of staff members and tours without compromising performance.
- It should be scalable to support additional features and functionalities in the future.

## Testing

### 5.1. Test Plan

Project Name	Tourism Railway Guide System
Test Plan ID	TRGD01
Brief Introduction about the system.	In response to the changing landscape of travel preferences, this thesis explores the development and implementation of a Railway Tourism Management System. Focused on enhancing the railway tourism experience, the project addresses existing challenges by providing a centralized platform for efficient attractions management, an enhanced guide system for tourists, optimized navigation features, and administrative tools
Test Objectives	<ol style="list-style-type: none"><li>1. Insert manager Data</li><li>2. Update/Edit Manager Data</li><li>3. Search Manager Data</li><li>4. Delete Manager Data</li><li>5. View Manager Data</li></ol>
Features to be tested	Manage Staff
Test Environment	<ul style="list-style-type: none"><li>• Devices - Laptop</li><li>• Software – Eclipse IDE</li><li>• Database – WAMP server database</li></ul>
Test Approach	<ul style="list-style-type: none"><li>• Testing Levels: System testing, acceptance testing</li><li>• Testing Methods: Manual testing</li><li>• Testing Tools: Junit</li></ul>
Testing Tasks	<ul style="list-style-type: none"><li>• Test Planning: Define test objectives, identify test scenarios, and plan test execution.</li><li>• Test Design: Create test cases for each identified test scenario.</li><li>• Test Execution: Execute test cases and scripts in the test environment.</li></ul>
Test deliverables	<ol style="list-style-type: none"><li>1. Test Plan</li><li>2. Test Environment setup documentation</li><li>3. Test Summary Report</li><li>4. Test Results</li><li>5. Test Evaluation Report</li></ol>
Schedule	<ul style="list-style-type: none"><li>• Test Planning: [10.02.2024] - [14.02.2024]</li><li>• Test Design: [14.02.2024] - [19.02.2024]</li><li>• Test Development: [19.02.2024] -</li></ul>

	[22.02.2024] • Test Execution: [22.02.2024] - [23.02.2024] • Test Evaluation: [23.02.2024] - [24.02.2024]
--	--

## 5.2. Test Cases

### I. Insert Manager Details.

Test Case 01	
Insert Staff Details	Piyumi Natasha
01	Black box
Insert Staff Details to Database	2024.02.24
Staff Details	3.00 am

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
01	Insert Staff Details	01	Full Name: Piyumi Natasha Email Address: piyu@gmail.com Password: 1234 User Role:Guide	Show Staff Insert Successfully message	Staff Insert Successfully	Pass

## II. Update Staff Details.

Test Case 01	
Update Staff Details	Piyumi Natasha
02	Black box
Update Details from Database	2024.02.24
Staff Details	3.10 am

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
02	Update Staff Details	02	Full Name: Maneesha Gamage Email Address: <a href="mailto:piyu@gmail.com">piyu@gmail.com</a> User Role: Guide	Show Staff information updated successfully message	Staff information updated successfully	Pass

### III. Delete Staff Details

Test Case 03	
Insert Manager Details	Piyumi Natasha
03	Black box
Delete Staff Details from the Database	2024.02.24
Staff Details	3.30 am

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
03	Delete Staff Details	03	Id =10	Show Staff Deleted message	Staff Deleted Successfully	pass

### IV. Search Manager Details.

Test Case 04	
Search Manager Details	Piyumi Natasha
04	Black box
Search Staff Details from Database	2024.02.24
Staff Details	5.00 am

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
04	Search Manager Details	04	Email: <a href="mailto:suvi@gmail.com">suvi@gmail.com</a>	Show Staff where email equal suci2gmail.com	Full Name: Tharushi Nimasha Email Address: suvi@gmail.com	Pass

V. View Manager Details.

Test Case 05	
View Staff Details	Piyumi Natasha
05	Black box
View Staff Details from the Database	2024.02.24
Staff Details	4.00 am

Step No	Test Step	Test Case ID	Test Input	Expected result	Actual result	Test result
05	View Staff Details	05	Id=01	View all staff deatails	Full Name: Maneesha Gamage Email Address: <a href="mailto:piyu@gmail.com">piyu@gmail.com</a> User Role:guide	pass

## 6. Annexure

### 6.1 StaffClass.java

```
package railwayguide;

import java.awt.EventQueue;

public class admindashboard extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField nameField;
    private JTextField EmailField;
    private JTextField PasswordField;
    private JTextField textField_1;
    private JTextField roleField;
    private JTable table;
    protected JLabel searchField;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    admindashboard frame = new admindashboard();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public admindashboard() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



```

contentPane = new JPanel();
contentPane.setBackground(new Color(0, 204, 153));
contentPane.setForeground(new Color(0, 255, 255));
contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);
contentPane.setLayout(null);

JPanel panel = new JPanel();
panel.setBounds(48, 91, 620, 557);
panel.setBorder(new MatteBorder(10, 10, 10, 10, (Color) new Color(0, 153, 0)));
panel.setBackground(new Color(0, 153, 0));
contentPane.add(panel);
panel.setLayout(null);

JLabel lblNewLabel_1 = new JLabel("Staff Register");
lblNewLabel_1.setBounds(258, 10, 144, 63);
lblNewLabel_1.setFont(new Font("Tahoma", Font.BOLD | Font.ITALIC, 20));
panel.add(lblNewLabel_1);

nameField = new JTextField();
nameField.setBounds(216, 116, 286, 35);
panel.add(nameField);
nameField.setColumns(10);

EmailField = new JTextField();
EmailField.setBounds(216, 209, 286, 35);
EmailField.setColumns(10);
panel.add(EmailField);

JLabel lblNewLabel_2_1 = new JLabel("Full Name:");
lblNewLabel_2_1.setBounds(54, 125, 117, 13);
lblNewLabel_2_1.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1);

JLabel lblNewLabel_2_1_1 = new JLabel("Email Address:");
lblNewLabel_2_1_1.setBounds(54, 218, 117, 13);

```

```

panel.add(lblNewLabel_2_1_1_1);

JLabel lblNewLabel_2_1_1_1_1 = new JLabel("User Role:");
lblNewLabel_2_1_1_1_1.setBounds(54, 413, 117, 13);
lblNewLabel_2_1_1_1_1.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1_1_1_1);

JLabel lblNewLabel_2_1_1_1_2 = new JLabel("Password:");
lblNewLabel_2_1_1_1_2.setBounds(54, 314, 117, 13);
lblNewLabel_2_1_1_1_2.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1_1_1_2);

PasswordField = new JPasswordField();
PasswordField.setBounds(216, 305, 286, 35);
PasswordField.setColumns(10);
panel.add(PasswordField);

JSeparator separator = new JSeparator();
separator.setBounds(10, 483, 1, 2);
panel.add(separator);

roleField = new JPasswordField();
roleField.setBounds(216, 402, 286, 35);
roleField.setColumns(10);
panel.add(roleField);

JPanel panel_1 = new JPanel();
panel_1.setBounds(688, 91, 595, 557);
panel_1.setBorder(new MatteBorder(10, 10, 10, 10, (Color) new Color(0, 153, 0)));
contentPane.add(panel_1);
panel_1.setBackground(new Color(0, 204, 153));
panel_1.setForeground(new Color(0, 255, 0));
panel_1.setLayout(null);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(25, 56, 541, 462);
panel_1.add(scrollPane);

```

```

table = new JTable();
table.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int selectedRow = 0;
        if (selectedRow != -1) {

            nameField.setText(table.getValueAt(selectedRow, 1).toString());
            EmailField.setText(table.getValueAt(selectedRow, 2).toString());

            roleField.setText(table.getValueAt(selectedRow, 3).toString());

        }
    }
});
scrollPane.setViewportView(table);

JButton btnNewButton = new JButton("VIEW");
btnNewButton.setBounds(392, 688, 101, 26);
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        staff staffobject=new staff();
        DefaultTableModel model=staffobject.view();
        table.setModel(model);

    }
});
contentPane.add(btnNewButton);

```

```

});
contentPane.add(btnNewButton);
btnNewButton.setBackground(new Color(0, 102, 255));
btnNewButton.setFont(new Font("Tahoma", Font.BOLD, 15));

JButton btnUpdate = new JButton("UPDATE");
btnUpdate.setBounds(650, 688, 109, 26);
contentPane.add(btnUpdate);
btnUpdate.setBackground(new Color(0, 204, 102));
btnUpdate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        int selectedRow = 0;
        int id = (int) table.getValueAt(selectedRow, 0); // Retrieve the ID from the selected row
        String name = nameField.getText();
        String email = EmailField.getText();
        String password = PasswordField.getText();
        String role = roleField.getText();

        // Create an instance of the staff class
        staff staffManager = new staff();

        // Call the update method on the instance
        boolean updated = staffManager.update( name, email, password, role);
        if (updated) {
            JOptionPane.showMessageDialog(null, "Staff information updated successfully.");
        } else {
            JOptionPane.showMessageDialog(null, "Failed to update staff information.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});
btnUpdate.setFont(new Font("Tahoma", Font.BOLD, 15));

JButton btnDelete = new JButton("DELETE");
btnDelete.setBounds(916, 688, 109, 26);
contentPane.add(btnDelete);
btnDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        int result=JOptionPane.showConfirmDialog(null, "are you want to delete?","Warning",JOptionPane.YES_NO_OPTION);
        try{
            staff staffObject=new staff();

            int idToDelete=getSelectedIdFromTable();
            boolean sucess= staffObject.delete(idToDelete);

            if(sucess) {
                JOptionPane.showMessageDialog(btnDelete,"Staff Deleted Sucessfully");
            }

            DefaultTableModel model=staffObject.view();

```

```

        DefaultTableModel model=staffObject.view();
        table.setModel(model);
    }else {
        JOptionPane.showMessageDialog(btnDelete,"Staff Deleted Faill");
    }

} catch(SQLException ex) {
    ex.printStackTrace();
}

}

private int getSelectedIdFromTable() {
    int selectedRowIndex = table.getSelectedRow();
    if (selectedRowIndex != -1) {

        return (int) table.getValueAt(selectedRowIndex, 0);
    }

    return 0;
}

);

btnDelete.setBackground(new Color(204, 0, 51));
btnDelete.setFont(new Font("Tahoma", Font.BOLD, 15));

JButton btnAdd = new JButton("ADD");
btnAdd.setBounds(115, 688, 101, 26);
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        String name=nameField.getText();
        String email=EmailField.getText();
        String password=PasswordField.getText();
        String role=roleField.getText();

        staff staffobject=new staff();

```

```

        boolean success = staffobject.insert(name, email, password, role);

        if(success) {

            JOptionPane.showMessageDialog(btnAdd, "Staff Insert Sucessfully");
            nameField.setText("");
            EmailField.setText("");
            PasswordField.setText("");
            roleField.setText("");

        }else {
            JOptionPane.showMessageDialog(btnAdd, "Staff Insert fail");
        }
    }

});
btnAdd.setFont(new Font("Tahoma", Font.BOLD, 15));
btnAdd.setBackground(new Color(204, 255, 255));
contentPane.add(btnAdd);

JButton btnExit = new JButton("EXIT");
btnExit.setBounds(1156, 688, 109, 26);
btnExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
btnExit.setFont(new Font("Tahoma", Font.BOLD, 15));
btnExit.setBackground(new Color(255, 255, 102));
contentPane.add(btnExit);

textField_1 = new JTextField();
textField_1.setBounds(48, 675, 1235, 60);
textField_1.setBackground(new Color(0, 204, 153));
contentPane.add(textField_1);
textField_1.setColumns(10);

JLabel lblNewLabel_2 = new JLabel("RAILWAY GUIDE PRO ADMIN DASHMASTER");
lblNewLabel_2.setBounds(373, 29, 734, 37);
lblNewLabel_2.setFont(new Font("Tahoma", Font.BOLD, 30));
contentPane.add(lblNewLabel_2);

JButton searchbtn = new JButton("SEARCH");
searchbtn.setBounds(633, 654, 89, 23);
searchbtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String searchEmail = searchField.getText().trim();
        if (!searchEmail.isEmpty()) {
            try {
                staff staffManager = new staff();
                ResultSet resultSet = staffManaqer.searchByEmail(searchEmail);
            }
        }
    }
});

```

```

        contentPane.add(lblNewLabel_2);

        JButton searchbtn = new JButton("SEARCH");
        searchbtn.setBounds(633, 654, 89, 23);
        searchbtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String searchEmail = searchField.getText().trim();
                if (!searchEmail.isEmpty()) {
                    try {
                        staff staffManager = new staff();
                        ResultSet resultSet = staffManager.searchByEmail(searchEmail);
                        loadDataIntoTable(resultSet);
                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                }
            }
        });
        contentPane.add(searchbtn);
    }

    protected void loadDataIntoTable(ResultSet resultSet) {
        DefaultTableModel model = new DefaultTableModel();
        table.setModel(model); // Set the new model to the table

        try {
            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();

            // Add columns to the model
            for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
                model.addColumn(metaData.getColumnLabel(columnIndex));
            }

            // Add rows to the model
            while (resultSet.next()) {
                Object[] rowData = new Object[columnCount];
                for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
                    rowData[columnIndex - 1] = resultSet.getObject(columnIndex);
                }
                model.addRow(rowData);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

## 6.2 Connection class

```
package railwayguide;
import java.sql.Connection;

public class MySqlConnection {
    private String username;
    private String pwd;
    private String dbname;
    private Connection conn;

    public MySqlConnection() {
        this.dbname="projecteddb";
        this.username="root";
        this.pwd="";
        connectDb();
    }

    public Connection connectDb() {
        try {
            conn=DriverManager.getConnection("jdbc:mysql://localhost:3309/"+dbname,username,pwd);
        } catch (SQLException e) {
            System.out.print(e);
        }
        return conn;
    }
}
```



### 6.3 login class.java

```
package railwayguide;

import java.awt.EventQueue;

public class Login extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField textField;
    private JPasswordField passwordField;
    protected Connection connection;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Login frame = new Login();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Login() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 856, 492);
        contentPane = new JPanel();
        contentPane.setBackground(new Color(0, 153, 0));
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("RAILWAY GUIDE PRO LOGIN");
        lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 20));
        lblNewLabel.setBounds(300, 24, 363, 41);
        contentPane.add(lblNewLabel);

        JLabel lblNewLabel_1 = new JLabel("User Name:");
        lblNewLabel_1.setFont(new Font("Tahoma", Font.BOLD, 18));
        lblNewLabel_1.setBounds(235, 124, 125, 13);
        contentPane.add(lblNewLabel_1);
    }
}
```

```

JLabel lblNewLabel_1_1 = new JLabel("Password");
lblNewLabel_1_1.setFont(new Font("Tahoma", Font.BOLD, 18));
lblNewLabel_1_1.setBounds(235, 198, 125, 13);
contentPane.add(lblNewLabel_1_1);

JLabel lblNewLabel_2 = new JLabel("ADMIN");
lblNewLabel_2.setFont(new Font("Tahoma", Font.BOLD, 20));
lblNewLabel_2.setBounds(403, 75, 131, 13);
contentPane.add(lblNewLabel_2);

textField = new JTextField();
textField.setBounds(370, 118, 258, 31);
contentPane.add(textField);
textField.setColumns(10);

passwordField = new JPasswordField();
passwordField.setBounds(370, 197, 268, 31);
contentPane.add(passwordField);

JButton btnNewButton = new JButton("Login");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(textField.getText().equals("admin") && new String (passwordField.getPassword()).equals("1234")){
            admindashboard obj=new admindashboard();
            obj.setVisible(true);
        }
        else {
            JOptionPane.showMessageDialog(null,"Invalid Login..Try Again !");
        }
    }
});
btnNewButton.setBounds(445, 297, 89, 23);
contentPane.add(btnNewButton);
}
}

```

## 6.4 Admin Dash Master

```
package railwayguide;

import java.awt.EventQueue;

public class admindashboard extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JTextField nameField;
    private JTextField EmailField;
    private JTextField PasswordField;
    private JTextField textField_1;
    private JTextField roleField;
    private JTable table;
    protected JLabel searchField;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    admindashboard frame = new admindashboard();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public admindashboard() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 1363, 798);
        contentPane = new JPanel();
        contentPane.setBackground(new Color(0, 204, 153));
        contentPane.setForeground(new Color(0, 255, 255));
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        setContentPane(contentPane);
        contentPane.setLayout(null);

        JPanel panel = new JPanel();
        panel.setBounds(48, 91, 620, 557);
        panel.setBorder(new MatteBorder(10, 10, 10, 10, (Color) new Color(0, 153, 0)));
        panel.setBackground(new Color(0, 153, 0));
        contentPane.add(panel);
    }
}
```

```

contentPane.add(panel);
panel.setLayout(null);

JLabel lblNewLabel_1 = new JLabel("Staff Register");
lblNewLabel_1.setBounds(258, 10, 144, 63);
lblNewLabel_1.setFont(new Font("Tahoma", Font.BOLD | Font.ITALIC, 20));
panel.add(lblNewLabel_1);

nameField = new JTextField();
nameField.setBounds(216, 116, 286, 35);
panel.add(nameField);
nameField.setColumns(10);

EmailField = new JTextField();
EmailField.setBounds(216, 209, 286, 35);
EmailField.setColumns(10);
panel.add(EmailField);

JLabel lblNewLabel_2_1 = new JLabel("Full Name:");
lblNewLabel_2_1.setBounds(54, 125, 117, 13);
lblNewLabel_2_1.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1);

JLabel lblNewLabel_2_1_1_1 = new JLabel("Email Address:");
lblNewLabel_2_1_1_1.setBounds(54, 218, 117, 13);
lblNewLabel_2_1_1_1.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1_1_1);

JLabel lblNewLabel_2_1_1_1_1 = new JLabel("User Role:");
lblNewLabel_2_1_1_1_1.setBounds(54, 413, 117, 13);
lblNewLabel_2_1_1_1_1.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1_1_1_1);

JLabel lblNewLabel_2_1_1_1_2 = new JLabel("Password:");
lblNewLabel_2_1_1_1_2.setBounds(54, 314, 117, 13);
lblNewLabel_2_1_1_1_2.setFont(new Font("Tahoma", Font.BOLD, 15));
panel.add(lblNewLabel_2_1_1_1_2);

PasswordField = new JTextField();
PasswordField.setBounds(216, 305, 286, 35);
PasswordField.setColumns(10);
panel.add(PasswordField);

JSeparator separator = new JSeparator();
separator.setBounds(10, 483, 1, 2);
panel.add(separator);

roleField = new JTextField();
roleField.setBounds(216, 402, 286, 35);
roleField.setColumns(10);
panel.add(roleField);

```

```

JPanel panel_1 = new JPanel();
panel_1.setBounds(688, 91, 595, 557);
panel_1.setBorder(new MatteBorder(10, 10, 10, 10, (Color) new Color(0, 153, 0)));
contentPane.add(panel_1);
panel_1.setBackground(new Color(0, 204, 153));
panel_1.setForeground(new Color(0, 255, 0));
panel_1.setLayout(null);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(25, 56, 541, 462);
panel_1.add(scrollPane);

table = new JTable();
table.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int selectedRow = 0;
        if (selectedRow != -1) {

            nameField.setText(table.getValueAt(selectedRow, 1).toString());
            EmailField.setText(table.getValueAt(selectedRow, 2).toString());

            roleField.setText(table.getValueAt(selectedRow, 3).toString());

        }
    }
});
scrollPane.setViewportView(table);

JButton btnNewButton = new JButton("VIEW");
btnNewButton.setBounds(392, 688, 101, 26);
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        staff staffobject=new staff();
        DefaultTableModel model=staffobject.view();
        table.setModel(model);

    }
});
contentPane.add(btnNewButton);
btnNewButton.setBackground(new Color(0, 102, 255));
btnNewButton.setFont(new Font("Tahoma", Font.BOLD, 15));

```

```

JButton btnUpdate = new JButton("UPDATE");
btnUpdate.setBounds(650, 688, 109, 26);
contentPane.add(btnUpdate);
btnUpdate.setBackground(new Color(0, 204, 102));
btnUpdate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        int selectedRow = 0;
        int id = (int) table.getValueAt(selectedRow, 0); // Retrieve the ID from the selected row
        String name = nameField.getText();
        String email = EmailField.getText();
        String password = PasswordField.getText();
        String role = roleField.getText();

        // Create an instance of the staff class
        staff staffManager = new staff();

        // Call the update method on the instance
        boolean updated = staffManager.update( name, email, password, role);
        if (updated) {
            JOptionPane.showMessageDialog(null, "Staff information updated successfully.");
        } else {
            JOptionPane.showMessageDialog(null, "Failed to update staff information.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});
btnUpdate.setFont(new Font("Tahoma", Font.BOLD, 15));

JButton btnDelete = new JButton("DELETE");
btnDelete.setBounds(916, 688, 109, 26);
contentPane.add(btnDelete);
btnDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        int result=JOptionPane.showConfirmDialog(null, "are you want to delete?","Warning",JOptionPane.YES_NO_OPTION);
        try{
            staff staffObject=new staff();

            int idToDelete=getSelectedIdFromTable();
            boolean success= staffObject.delete(idToDelete);

            if(success) {
                JOptionPane.showMessageDialog(btnDelete,"Staff Deleted Successfully");

                DefaultTableModel model=staffObject.view();
                table.setModel(model);
            }else {
                JOptionPane.showMessageDialog(btnDelete,"Staff Deleted Fail");
            }
        }
    }
});

```

```

        DefaultTableModel model=staffObject.view();
        table.setModel(model);
    }else {
        JOptionPane.showMessageDialog(btnDelete,"Staff Deleted Faill");
    }

}

}catch(SQLException ex) {
    ex.printStackTrace();
}

}

private int getSelectedIdFromTable() {
    int selectedIndex = table.getSelectedRow();
    if (selectedIndex != -1) {

        return (int) table.getValueAt(selectedRowIndex, 0);
    }

    return 0;
}

});

btnDelete.setBackground(new Color(204, 0, 51));
btnDelete.setFont(new Font("Tahoma", Font.BOLD, 15));

JButton btnAdd = new JButton("ADD");
btnAdd.setBounds(115, 688, 101, 26);
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        String name=nameField.getText();
        String email=EmailField.getText();
        String password=PasswordField.getText();
        String role=roleField.getText();

        staff staffobject=new staff();

```

```

        if(sucess) {

            JOptionPane.showMessageDialog(btnAdd, "Staff Insert Sucessfully");
            nameField.setText("");
            EmailField.setText("");
            PasswordField.setText("");
            roleField.setText("");

        }else {
            JOptionPane.showMessageDialog(btnAdd, "Staff Insert fail");
        }
    }

});

btnAdd.setFont(new Font("Tahoma", Font.BOLD, 15));
btnAdd.setBackground(new Color(204, 255, 255));
contentPane.add(btnAdd);

JButton btnExit = new JButton("EXIT");
btnExit.setBounds(1156, 688, 109, 26);
btnExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

btnExit.setFont(new Font("Tahoma", Font.BOLD, 15));
btnExit.setBackground(new Color(255, 255, 102));
contentPane.add(btnExit);

textField_1 = new JTextField();
textField_1.setBounds(48, 675, 1235, 60);
textField_1.setBackground(new Color(0, 204, 153));
contentPane.add(textField_1);
textField_1.setColumns(10);

JLabel lblNewLabel_2 = new JLabel("RAILWAY GUIDE PRO ADMIN DASHMASTER");
lblNewLabel_2.setBounds(373, 29, 734, 37);
lblNewLabel_2.setFont(new Font("Tahoma", Font.BOLD, 30));
contentPane.add(lblNewLabel_2);

JButton searchbtn = new JButton("SEARCH");
searchbtn.setBounds(633, 654, 89, 23);
searchbtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String searchEmail = searchField.getText().trim();
        if (!searchEmail.isEmpty()) {
            try {
                staff staffManager = new staff();
                ResultSet resultSet = staffManager.searchByEmail(searchEmail);
            }
        }
    }
});

```



```

JButton searchbtn = new JButton("SEARCH");
searchbtn.setBounds(633, 654, 89, 23);
searchbtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String searchEmail = searchField.getText().trim();
        if (!searchEmail.isEmpty()) {
            try {
                staff staffManager = new staff();
                ResultSet resultSet = staffManager.searchByEmail(searchEmail);
                loadDataIntoTable(resultSet);
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    }
});
contentPane.add(searchbtn);
}

protected void loadDataIntoTable(ResultSet resultSet) {
    DefaultTableModel model = new DefaultTableModel();
    table.setModel(model); // Set the new model to the table

    try {
        ResultSetMetaData metaData = resultSet.getMetaData();
        int columnCount = metaData.getColumnCount();

        // Add columns to the model
        for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
            model.addColumn(metaData.getColumnLabel(columnIndex));
        }

        // Add rows to the model
        while (resultSet.next()) {
            Object[] rowData = new Object[columnCount];
            for (int columnIndex = 1; columnIndex <= columnCount; columnIndex++) {
                rowData[columnIndex - 1] = resultSet.getObject(columnIndex);
            }
            model.addRow(rowData);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

### 4.3. Summary

Chapter 4 focuses on the practical implementation of the Railway Tourism Management System, showcasing SQL queries, test plans, and user interface components essential for system development and validation. The detailed SQL queries demonstrate data retrieval and manipulation capabilities, while the test plan outlines objectives and methods for validating system features. Additionally, user interface elements such as the login form and admin dashboard contribute to the system's usability and functionality. This chapter forms a crucial bridge between system design and actual implementation, laying the groundwork for system testing and refinement.

## 5. Reference

<https://dev.mysql.com/doc/>

<https://www.db-fiddle.com/>

<https://sqlbolt.com/>

<https://gemini.google.com/app>