Imam Abdulrahman Bin Faisal University
College of Computer Science & Information Technology
Department of Computer Science

**CS 411 – Software Engineering**
**Term 3 – 2018/2019**

# *Software Design Specifications*

**For**

**Railway.Manage();**

**Version 0.1**

**CS Year 4, G1**

**Ms.Wadha Almattar**

***November 17,2018***

This Software Design Specification was prepared and provided as a deliverable for CS 411, Term 1, and it will be used by Operator and Passenger

This document is based in part on the IEEE Recommended Practice for Software Design Descriptions.

## Team Members:

| # | Name | ID | Role |
|---|---|---|---|
| 1 | Muneera abdullah alhajri | 2160007230 | Leader |
| 2 | Reema Ibrahim Alyousef | 2160003133 | Member |
| 3 | Rahaf Saleh Alzahrani | 2160006662 | Member |
| 4 | Tasneem Hamdy dosoqi | 2160007430 | Member |

## Table of Content

## Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| All members | Nov  17, 2013 | Prepared initial version | 0.1 |

## Table of Tables

## Table of Figures

## 1. Introduction

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The Software Design Document (SDD) shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development. Therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages:

- **The initial design phase:**
  Preliminary high-level design in which the overall system architecture and data architecture is defined
- **The detailed design phase:**
  This phase involves the decomposition of the system's architecture. It comprises of the detailed description of the data, sub-interfaces, and error messages included in end users' interfaces.

This chapter introduces the topics that are related to Railway.Manage(); SDS document. It covers the document's purpose, scope, list of acronyms and abbreviations, and list of references used.

### 1.1 Purpose

The purpose this document is to provide a description of the design of a Railway.Manage(); fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to build. This also includes the architectural features of the system down through details of what operations each code module will perform and the database layout. It also shows how the use cases detailed in the SRS will be implemented in the system using this design.

### 1.2 Scope

This document covers all the design features corresponding to the planned Railway.Manage(); application. The application shall provide specific characteristics and services that will assist RMT Company. The intended audience for this project is:

- **Operators**: They will be responsible about all aspects of the system with no restrictions, since they are permitted to access each component of the system and are also allowed to update or delete entries in time tables.
- **Passengers**: They will be able to access and review timetable of the station, book a trip, or inquire and monitor their ticket.

This document contains the system's architecture and logical design. It includes modeling of the data, relationships between entities and data processes using:

- Entity-relationship Diagram (ER)
- Logical Scheme (ER Mapping)
- Context Data Flow Diagram
- Data Flow Diagrams (DFD)
- Sequence Diagrams

In addition, it will provide a detailed data and database description. Furthermore, it contains the system's prototype (simple illustration of predicted interfaces) and pseudo code of the system's main Functions.

## 1.3 Definitions, Acronyms, and Abbreviations

Table 1.1 below specifies the definitions related to all terms mentioned in this document.

**Table 1.1 List of Definitions**

| Terminology | Definition |
|---|---|
| **Software Requirements Specifications (SRS)** | Software Requirement Specification is a description of the software system to be developed, laying out functional and non-functional requirements. [1] |
| **MySQL** | MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation. [2] |
| **Data Flow Diagram** | A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored. [3] |
| **Context Diagram** | A context diagram is a top level (also known as "Level 0") data flow diagram. It only contains one process node ("Process 0") that generalizes the function of the entire system in relationship to external entities. [3] |

Table 1.2 below specifies the acronyms used in this document:

**Table 1.2 List of Acronyms**

| Acronym | Definition |
|---|---|
| API | Application Programmable Interface |
| CSSN | Customer Social Security Number |
| DFD | Data Flow Diagram |
| E-mail | Electronic Mail |

| | |
|---|---|
| ER | Entity Relationship |
| ESSN | Employee Social Security Number |
| ID | Identification |
| IDE | Integrated Development Environment |
| IEEE | Institute of Electrical and Electronics Engineers |
| INFO | Information |
| INT | Integer |
| JDK | Java Development Kit |
| NO. | Number |
| OOP | Object Oriented Programming |
| OS | Operating System |
| RAN | Reliable And Nimble |
| RDBMS | Relational Database Management System |
| SQL | Structures Query Language |
| SRS | Software Requirements Specification |
| SSN | Social Security Number |
| VARCHAR | Various Characters |

## 1.4 References

Registered below are references referred to in this document:

[1] AlShallali, Bashair M., et al. Software Design Specification. 2016.
[2] MySQL :: MySQL 5.7 Reference Manual :: 1.3.1 What is MySQL? (n.d.). Retrieved
[3] Data Flow Diagram. (n.d.). Retrieved November 29, 2016, from https://www.smartdraw.com/data-flow-diagram/
[4] Shneiderman's Eight Golden Rules of Interface Design. (2018). Retrieved from https://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html
[5] Shneiderman’s Eight Golden Rules Will Help You Design Better Interfaces. (2018). Retrieved from https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces

## 2. System overview

Railway.Manage(); is a system designed to make the management of a train station simpler, this is demonstrated by allowing passengers to book, check their ticket or view the station's time table from their devices. It is possible to do so by dividing the roles into passengers and operators to govern the functionalities of this system, and to allow the system to function as intended.

### 2.1 Overall System Functions

Railway.Manage(); has 2 types of users, Operators and Passengers. This distinction is important to organize the flow of the system. Both users have their respective authorities and functionalities, note that there are common functionalities that both roles could accomplish.

#### 2.1.1 Common Functionality

Both roles (Passengers and operators) are able to view the station's trip schedule. This function is essential to all the users in order to have knowledge about the current flow of the station, and to note which trips are arriving and departing from and to the station.

#### 2.1.2 Passenger

The passenger has the capabilities to do the following:

- ➢ View Ticket
  After booking successfully, the passenger is informed of the completion of the booking and is provided with a generated number that could be used to view their ticket later on if they wish to do so.
- ➢ Book a ticket
  This is done via several stages:
  - 1.Passenger info
    During this stage the user is required to enter their information, this data contains personal information to identify them, alongside contact info.
  - 2.Booking
    The user should specify the trip they wish to travel on, alongside various information to make the booking possible.
  - 3.Selecting a Seat
    After completing the previously mentioned operations, users are required to select a seat in the train.
  - 4.Booking Confirmation
    Here the user is informed of the completion of the booking and is supplied with a generated ticket ID.

#### 2.1.3 Operator

Operators are authorized to modify the trips' table in the database, this is demonstrated by the following functions, note that this is only possible after logging in successfully:

- ➢ Edit a trip's info
  The operator is allowed to update the attributes of the trip by using this function.

- Delete a trip
  If the trip is cancelled or no longer relevant, the operator is able to delete it form the trip schedule.
- Add a new trip
  It is possible to add a new entry in the trips' table in the database.

## 3. Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to plan a complete design solution.

### 2.2 Assumptions and Dependencies

A lot of assumption must be considered and mentioned during the design phase of Railway.Manage(); software. These assumptions regarding to its use, may concern a lot of issues. These issues are frequently fall into 4 categories:

- Related software or hardware
- Operating systems
- End-user characteristics
- Possible and/or probable changes in functionality

#### 3.1.1 Related software or hardware

Railway.Manage(); is standalone desktop application in which any desktop computer is qualified to run this software. Regarding the hardware, Railway.Manage(); software requires 3.8MB memory space, so to install this software, the desktop computer must have at least 3.8MB free storage space.

Regarding the software, Railway.Manage(); is easy to use and maintain because of its simplicity along with the quick access to its database. Rational Database Management System (RDBMS) will be used to implements the database using My Structured Query Language (MySQL). The desktop computer must have NetBeans IDE application installed to run the software. Along with NetBeans IDE application the desktop computer must install MySQL Workbench application which help to store the database and all the require information to store it and retrieve it as needed.

#### 3.1.2 Operating systems

Railway.Manage(); is designed to run in 2 different platforms: Windows and Macintosh operating systems. The java language will be used to write the source code of the system which is capable to run on both systems.

#### 3.1.3 End-user characteristics

Railway.Manage(); has two different key end users: operators who run the software and control the system, and the passengers who use the software to book, inquire, and maintain their tickets.

#### 3.1.4 Possible and/or probable changes in functionality

Changes can occur at any time of the design phase and it cannot be prohibited. Sometimes, there will be change in the design phase if the old design cannot be implemented in the right way. It is acceptable to make any changes in the functionalities if a better design is found as long as it does not consume so much time because the project is time-bound, and each phase has a dedicated time.

## 2.3 General Constraints

There is limitations and constraints that have a significant impact on the design of the Railway.Manage(); 's software (and describe the associated impact). Such constraints may fall into these categories:

- Hardware or software environment
- End-user environment
- Interface requirements
- Data repository and distribution requirements
- Security requirements

### 2.3.1 Software or Hardware Environment

as described in 3.1.1 and 3.1.2, Railway.Manage(); is designed to run in 2 different platforms: Windows and Macintosh operating systems. Also, it will run on NetBeans IDE application and connected to MySQL RDBMS, implemented in MySQL Workbench.

### 2.3.2 End-user environment

For the users, to install Railway.Manage(); on their personal computers, they need to make sure that they have enough memory space at least 3.8MB.

### 2.3.3 Interface requirements

Railway.Manage();'s interfaces should design to be user-friendly and interactive interfaces. The overall structure of the application should be clear and simple. There may be an online manual delivered to the user for the use of the software. There will be some consideration should be put into mind when designing the interfaces, such as:

- Fonts
- Labels
- Buttons
- Visual aids
- Messages

### 2.3.4 Data repository and distribution requirements

All the necessary data and information will be stored in the database implemented in MySQL Workbench.

### 2.3.5 Security requirements

Railway.Manage(); system has a lot of information in its database and most of this information either personal for users and operators, or sensitive for stations and operators, so the system must provide good level of security to keep all these data safe. To implement security, the users has access only for booking or maintaining their tickets and information, so they have no access to the database except for their personal information. For the operators, they must log in to the system using username and password to gain full access without restriction to the system and its database. This will keep the sensitive information protected all the time.

# 4. User Interface Design

This chapter gives an overview of the functionality of the user interfaces and how it designed. It defines their rules, images, objects, and actions.

## 4.1 Overview of User Interface

Railway.Manage(); is user friendly software and easy to use and maintain because of its simplicity along with the quick access to its database. Regarding the end-users, there are two key end users: operators who run the software and control the system, and the passengers who use the software to book, inquire, and maintain their tickets. Each one of the two primary users have different privileges and authorizations. Upon running, the key interface in the application is the home page. The user will be greeted with an introductory frame that contains different functionalities of the program (each one accessed from a separate button) asking them to make a choice of which one they would like to perform. There are five different buttons in this interface and each one is lead to a different interface as follows:

- ❖ **Train Schedule:** if the user chooses this option, a frame will open enabling the user to view all the trips available and their timings (departure and arrival). from another frame that will open when the button is selected, thus preforming a query of the relation storing the trips.
- ❖ **quick booking:** if the user chooses this option, another frame will open asking them to enter their various information and will store them in a relation concerned with collecting the passengers' data. After that, they get to choose which trip they would like to go on (from a pre-defined list), their seat, class...etc via another frame, which is the main mean of booking a trip and will generate ticket ID if the booking was made successfully.
- ❖ **view ticket:** if the user chooses this option, a frame will open asking them to enter the ticket ID that was generated upon booking, when entered successfully, the ticket information will be displayed to the user to view.
- ❖ **Terms:** if the user chooses this option, a file will open enabling the user to view all the terms, conditions and legal policy for the station that may affect their legal rights.
- ❖ **Operator login:** this button will only be useful to operators that wish to make slight modifications or queries to the existing information stored in the database, whether they are trips, passenger's information, tickets...etc. When this button is selected, a frame will open that can perform such functionalities.

## 4.2 Interface Design Rules

The most important reason of design stage it to design clear, productive, cohesive and well-organized user interfaces. The design of Railway.Manage(); user interfaces should reflect Shneiderman's golden rules the "Eight Golden Rules" [1,2] to lead to better interface design. These rules are listed below:

- **Strive for consistency:** use same colors, icons, layout, and menus hierarchy when deigning similar events. Consistency is important because it helps the user to get familiar with Railway.Manage(); software very quickly.
- **Enable frequent users to use shortcuts:** allows the use of keyboard shortcuts by the user to navigate the interface quickly.
- **Offer informative feedbacks:** For every action, there should be human readable feedback to let the user know what is going.
- **Design dialogs to yield closure:** For every sequence of actions, give the user an indication about the successful or failure of those actions. Informative feedback gives the users the satisfaction of accomplishment and let them know that they can move to the next actions.

- **Offer simple error handling:** Design the system to be a fool proof as possible so the user cannot make a serious error. If an error is occurred, the system shall be able to detect the error and offer simple mechanisms to handling that error.
- **Permit easy reversal of actions:** Errors can be undone. Offer users by clear ways to reverse their actions.
- **Support internal locus of control:** Design the software to make the user have full control of Actions.
- **Reduce short-term memory load:** Interfaces must be simple and obvious. Use recognition and avoid designing it in which a user must recall information from the previous interface.

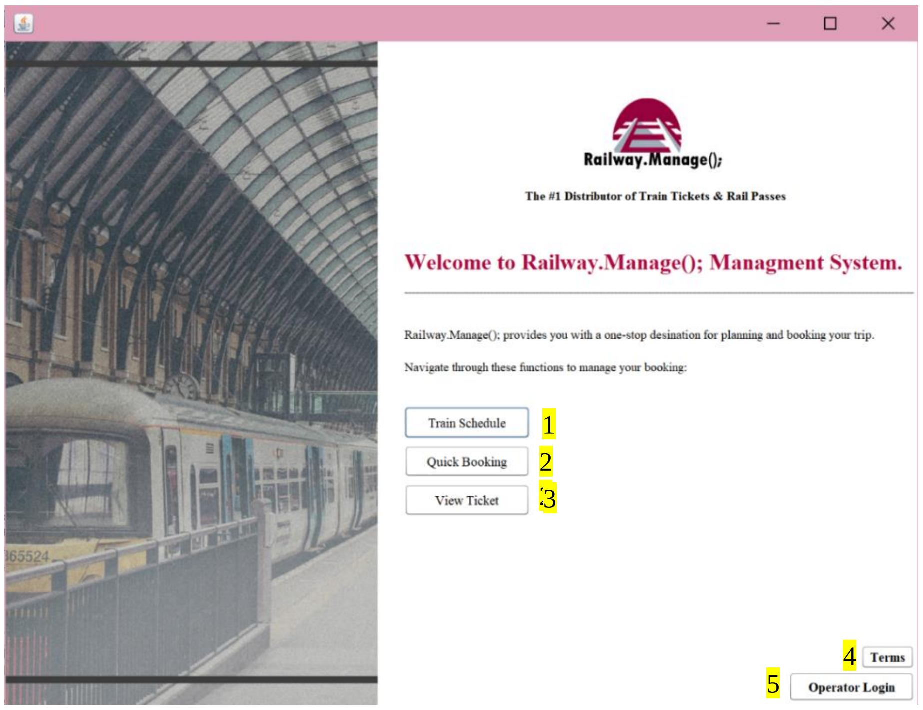
## 4.3 Screen Images

This section shows an initial sample of Railway.Manage(); interfaces. This sample was designed using NetBeans drag and drop to give a strong idea about the software's interfaces before the implementation phase. It must be noted that every interface has yellow numbers that are described in section 4.4.

### 4.3.1 Common Interfaces

#### 4.3.1.1 Main Interface (Home)

Iintroductory interface that shown in **Error! Reference source not found.** below contains different functionalities of the software. The user asked to make a choice of which one they would like to perform. There are five different buttons in this interface and each one is lead to a different interface.



Railway.Manage();

The #1 Distributor of Train Tickets & Rail Passes

Welcome to Railway.Manage(); Managment System.

Railway.Manage(); provides you with a one-stop desination for planning and booking your trip.

Navigate through these functions to manage your booking:

Figure 1: Main Interface screen image.

### 4.3.2 User 1: Passenger

From the passenger perspective, the passenger has three options (buttons): train schedule, quick booking and view ticket.

#### 4.3.2.1 Train Schedule

If the user chooses this option, Figure , enable the user to view all the trips available and their timings (departure and arrival).

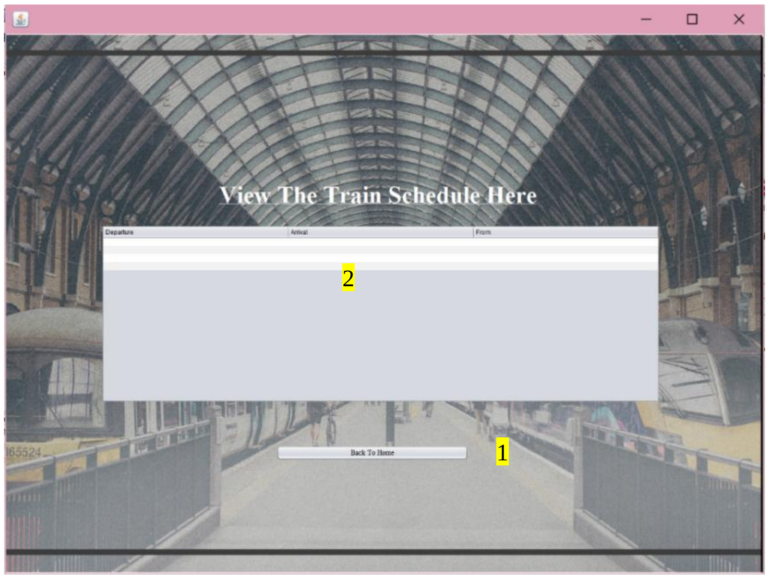


Figure 3: Train Schedule Screen Image

#### 4.3.2.2Quick Booking

##### 4.3.2.2.1 Passenger Information

if the user chooses this option, another frame will open asking them to enter their various information and will store them in a relation concerned with collecting the passengers' data. Figure  below shows that:

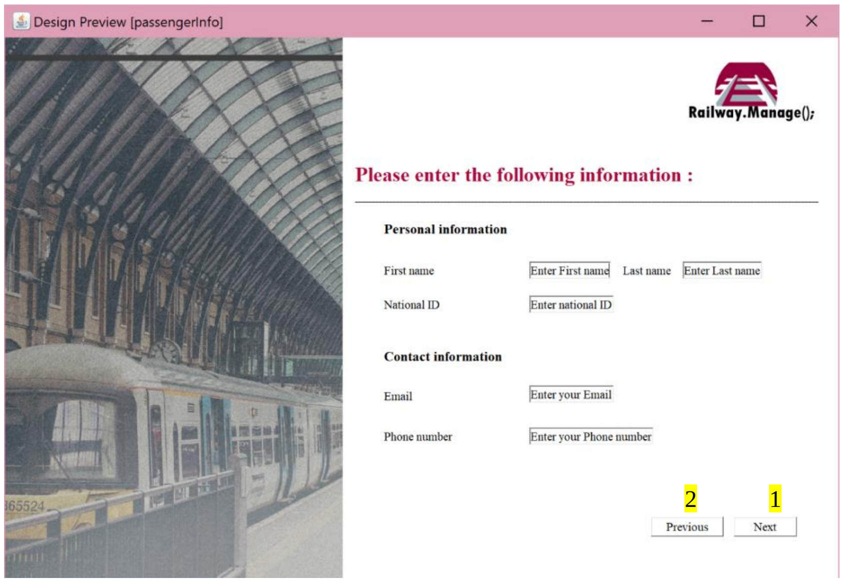


Figure 4: Passenger Information Screen Image

##### 4.3.2.2.2 Booking Information

After the passenger information step is done, the passengers will enter the booking information such as: type of the trip (one way or round trip), station and time and date (departure and arrival), number of adults and children. Figure  below illustrate that:

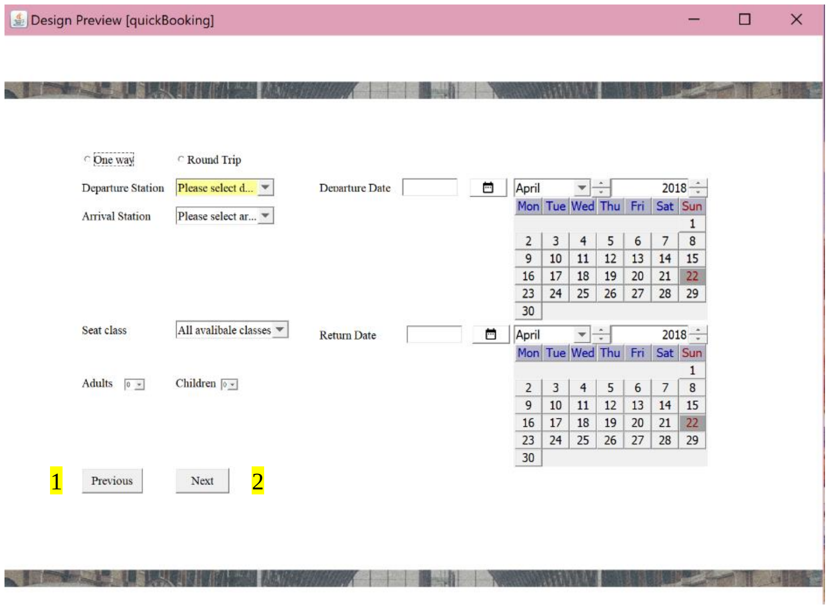


Figure 5: Booking Information Screen Image

#### 4.3.2.2.3 Seat, Class Selection

After the booking information step is done, the passengers will choose their seats and the class from a pre-defined list. Figure  below illustrate that:

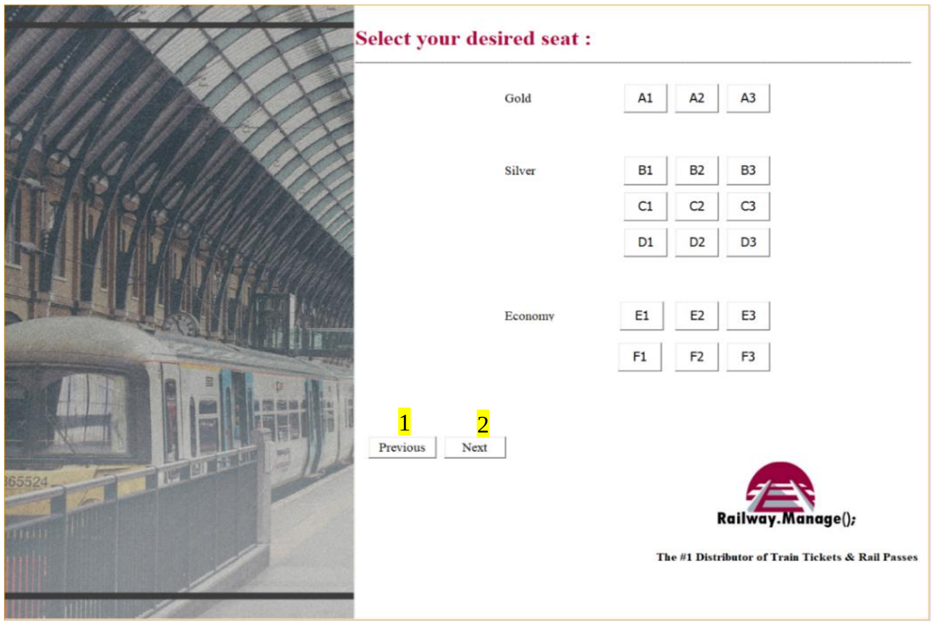


Figure 6: Seat, Class Selection Screen Image

#### 4.3.2.2.4 Booking Completion

After the previous steps completed, another frame will open to inform the user that the booking is done successfully, and a ticket Id will be generated.  Figure 2 below shows that:

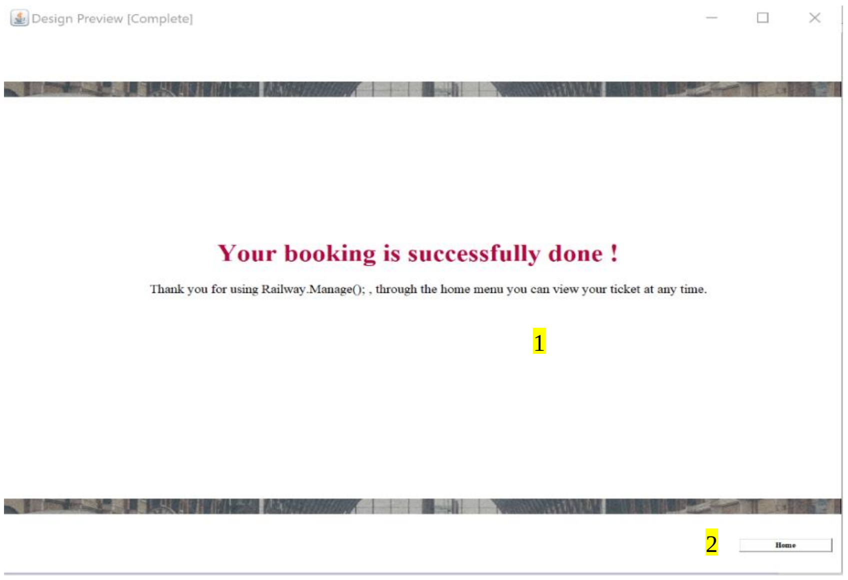

Design Preview [Complete]
Your booking is successfully done !
Thank you for using Railway.Manage(); , through the home menu you can view your ticket at any time.
1
2
Home

Figure 7: Booking Completion Screen Image

### 4.3.2.3 View ticket

if the users choose this option, another frame will open asking them to enter their ticket Id to view their booking information. Figure 3 below shows that:

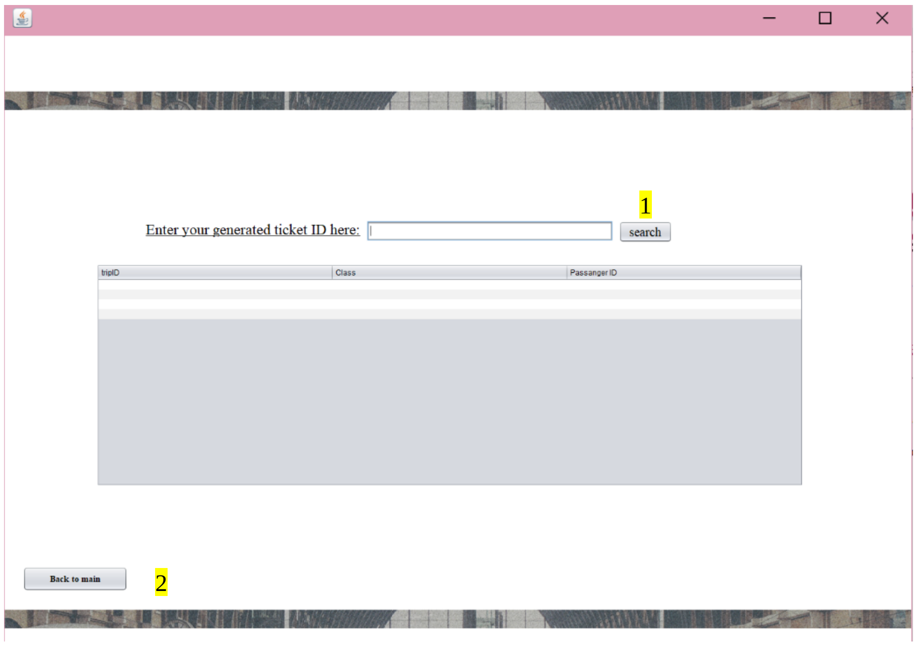


Figure 3: View ticket Screen Image

## 4.3.3 User 2: Operator

From the operator perspective, the operator has more privilege than the passenger. The operator will control the system.

### 4.3.3.1Operator Login

From the home interface, the operator login button will be only for the operators and it will open the frame shown in Figure 4 below when pressed.

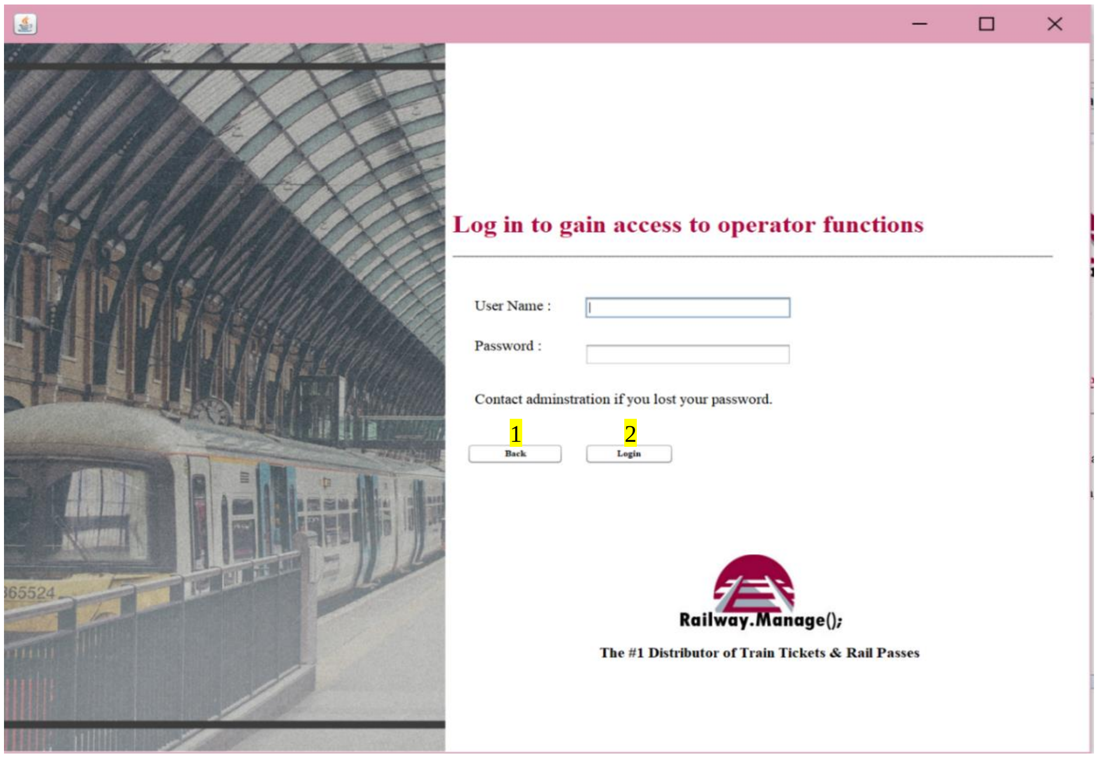

Log in to gain access to operator functions
User Name :
Password :
Contact adminstration if you lost your password.
1
Back
2
Login
Railway.Manage();
The #1 Distributor of Train Tickets & Rail Passes

Figure 4: Operator Login Screen Image.

### 4.3.3.2 Manage Trip's Schedule

This interface accessed only by the operator after the login stage. At this interface, the operator has the authority to make slight modifications such as: save, delete and update the existing information in the schedule. Figure 5 below shows that:

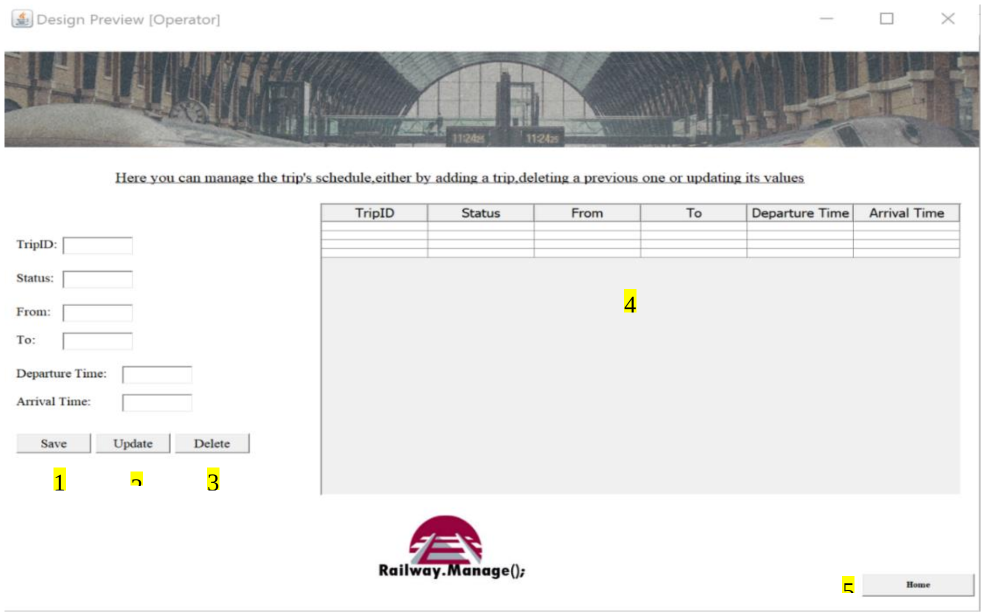


Figure 5: Manage Trip's Schedule Screen Image.

## 4.4 Screen Objects and Actions

This subsection provides a detailed description for every interface's objects and actions. The yellow numbers in the previous interfaces, in section 4.3 is represented in the numbers in the first column in the Table 3 below.

**Table 3: Screen Objects and Actions.**

| No. | Object | Type | Action |
|---|---|---|---|
| | | ***Common Functions*** | |
| | | Main Interface (**Error! Reference source not found.**) | |
| **1** | train schedule | Button | This button will take the user to "train schedule" interface as described in 4.3.2.1. |
| **2** | quick booking | Button | This button will take the user to "quick booking" interface as described in 4.3.2.2. |
| **3** | view ticket | Button | This button will take the user to "view ticket" interface as described in 4.3.2.3. |
| **4** | Terms | Button | This button will take the user to a file and enable the user to read all the terms, conditions. |
| **5** | Operator login | Button | This button will take the user to "Operator login" interface as described in 4.3.3.1. |
| | | User 1: Passenger | |
| | | Train Schedule (Figure 3: Train Schedule Screen Image) | |
| **1** | Bach to home | Button | This button will cancel the process and take the user back to "home" interface. |
| **2** | Trip's schedule | Table | This table will show the trip's schedule and its time. |
| | | quick booking | |
| | | Passenger Information (Figure 4: Passenger Information Screen Image) | |
| **1** | Next | Button | This button will continue the process and take the user next to "booking information" interface. |
| **2** | Previous | Button | This button will cancel the process and take the user back to "home" interface. |
| | | Booking Information (Figure 5: Booking Information Screen Image) | |
| **1** | Previous | Button | This button will take the user back to "passenger information" interface. |
| **2** | Next | Button | This button will continue the process and take the user next to "seat selection" interface. |
| | | Seat, Class Selection (Figure 6: Seat, Class Selection Screen Image) | |
| **1** | Previous | Button | This button will take the user back to "passenger information" interface. |
| **2** | Next | Button | This button will continue the process and take the user next to "seat selection" interface. |
| | | Booking Completion (Figure 2: Booking Completion Screen Image) | |
| **1** | Label | Label | When the booking is successfully done, a ticket id will be generated and displayed in this label. |
| **2** | Home | Button | This button will cancel the process and take the user back to "home" interface. |

| | | | |
|---|---|---|---|
| View ticket (Figure 3: View ticket Screen Image) | | | |
| **1** | Search | Button | This button will search for the ticket based on ticket id and display it in the table if it is exist. |
| **2** | Back to main | Button | This button will cancel the process and take the user back to “home” interface. |
| User 2: Operator | | | |
| Operator Login (Figure 4: Operator Login Screen Image.) | | | |
| **1** | Back | Button | This button will cancel the process and take the user back to “home” interface. |
| **2** | Login | Button | This button enables the operator to enter the system and take the operator next to “manage” interface. |
| Manage Trip’s Schedule (Figure 5: Manage Trip’s Schedule Screen Image.) | | | |
| **1** | Save | | This button enables the operator to save any changes made into trip’s schedule. |
| **2** | Update | | This button enables the operator to update any information in the trip’s schedule. |
| **3** | Delete | | This button enables the operator to delete any information in the trip’s schedule. |
| **4** | Trip’s schedule | Table | This table will show the trip’s schedule and its time. |
| **5** | Home | Button | This button will cancel the process and take the user back to “home” interface. |

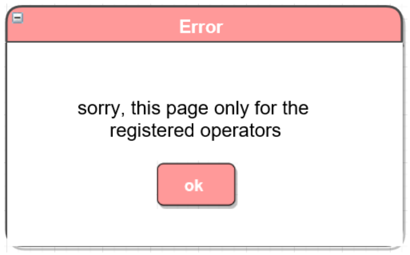
## 4.5 Other Interfaces

This section includes examples of the errors and exceptions that may happen during running the software and the confirmation messages in Railway.Manage();. It is important to inform the user when something unexpected occurred and inform the users to how to correct these errors. These messages are designed using drow.io.

### 4.5.1 Error Messages

Error messages are displayed when something unexpected occurs to inform the users of Railway.Manage(); that they violate the system constrains. The figures below show some samples of the error messages.

Figure 6 shows an error message when the passenger clicks the operator login button in the main interface showed in figure 1.



sorry, this page only for the registered operators

Figure 6: passenger clicks login button error message

Figure 7 shows an error message when the passenger clicks the next button without complete his/her information in the quick booking interface showed in Figure .

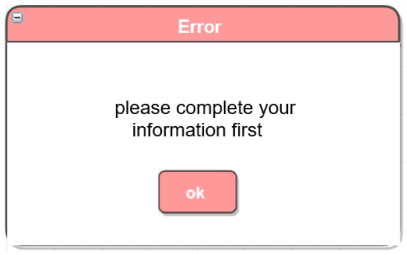


Figure 7: uncomplete information error message.

Figure 8 shows an error message when the passenger clicks the next button without complete the booking information in the booking information interface showed in Figure 4.

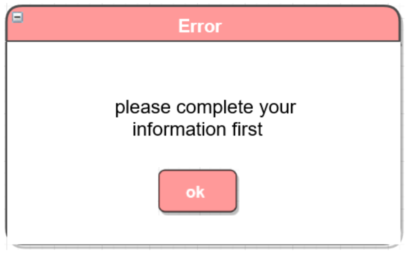


Figure 8: uncomplete booking information error message.

Figure 9 shows an error message when the passenger clicks the next button without choose his/her seat in the seat, class selection interface showed in Figure .

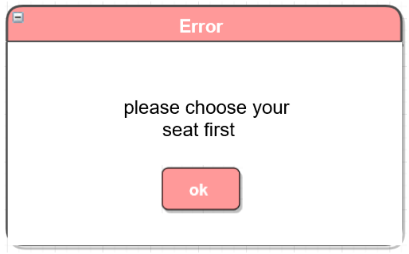


Figure 9: unselected seat error message.

### 4.5.2 Confirmation Messages

confirmation messages are displayed to confirm some user's actions and to make sure that they understand what they do. The figures below show some samples of the confirmation messages.

Figure 10 shows a confirmation message when the passenger clicks the next button after he/she choose his/her seat in the seat, class selection interface showed in Figure 5 for completing the booking.

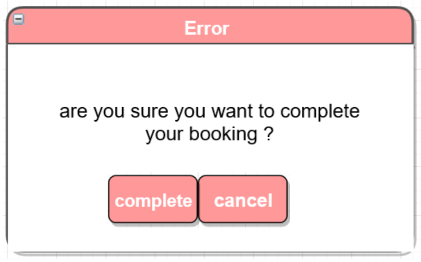


Figure 10: complete booking confirmation message.

Figure 11 shows a confirmation message when the operator clicks the login button in the operator login interface showed in Figure 4.

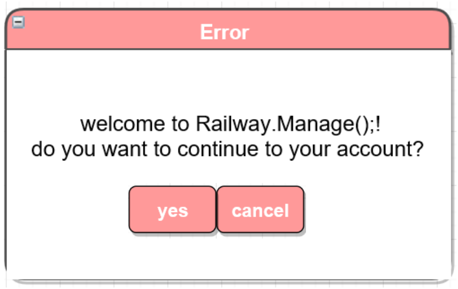

Error
welcome to Railway.Manage();!
do you want to continue to your account?
yes
cancel

Figure 11: operator login confirmation message.

Figure 12 shows a confirmation message when the operator clicks the update button in the manage trip’s schedule interface showed in Figure 5.

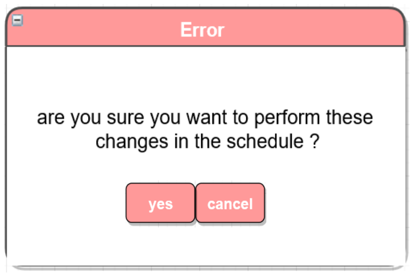

Error
are you sure you want to perform these changes in the schedule ?
yes
cancel

Figure 12: update schedule confirmation message.

Figure 13 shows a confirmation message when the operator clicks the delete button in the manage trip’s schedule interface showed in Figure 5.

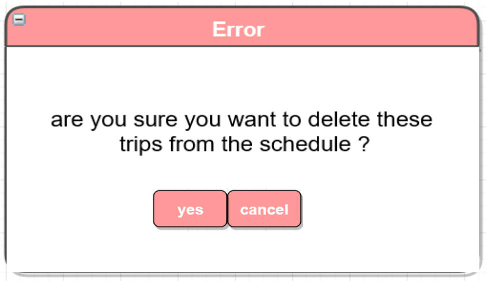

Error
are you sure you want to delete these trips from the schedule ?
yes
cancel

Figure 13: delete trips confirmation message.

# 5. System Architecture

This chapter provides an indication of Railway.Manage();'s architecture. It covers the architectural design approach, the overall system and subsystems architectures.

## 5.1 Architectural Design Approach

The architecture used for the proposed Railway.Manage(); desktop application is the multilayered architecture (N-tier). The architecture consists of 2 basic layers; presentation layer and data layer.

- **Presentation layer:** This is the upper layer that is responsible for managing user's interaction with the system, by displaying information and enabling the passengers to access the time table of the station, book a trip or inquire about their ticket info at any time. The input information of that is then manipulated and processed in the Data layer.
- **Data layer**: This layer provides an access to the database within the system constraints. Data can be stored and retrieved by executing queries. Consequently. Results are sent back to the Presentation layer.

The multilayered architecture helps increase application's maintainability, flexibility and security. If the system was not layered, security threats would be located in one layer, which would be very hard to handle and resolve. Lastly, because the system is multilayered it leads to update processes and addition of more functions could be accomplished without affecting the whole application, which leads to a maintainable system. Similarly, the independency of each layer increases the flexibility of the application.

## 5.2 Architectural Design

The architectural design diagram for the proposed application is shown in Figure 14

- **Presentation layer:** Presentation layer provides tools to help displaying information and enabling the user to access the time table of the station, book a trip or inquire about their ticket information. Moreover, it takes inputs from the intended users (Operator, passengers) and interacts with them accordingly. Java Development Kit (JDK) will be used to create all interfaces. JDK provides different types of components such as combo box, buttons, etc., that satisfy this layer's requirements.
- **Data layer:** The data can be stored in and retrieved from MySQL Relational Database Management System (RDBMS). First, the Java Virtual Machine (JVM) sends queries to the database (MySQL). Accordingly, MySQL RDBMS sends the results back to the JVM. The application helps the intended users (Operator, passengers) interact with the system by entering inputs and storing them into the database and by retrieving data as outputs as well.

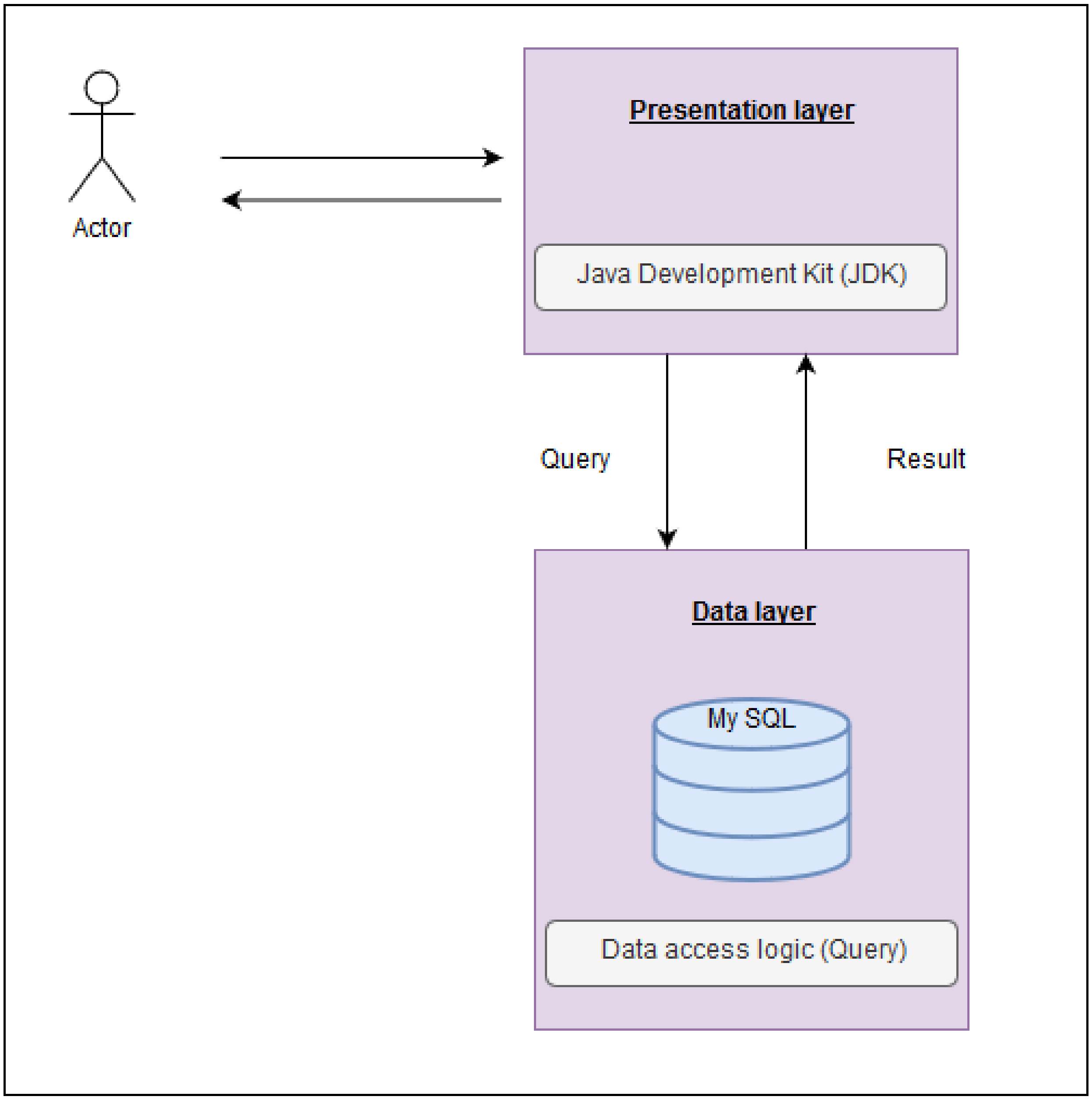


Figure 14 Architectural Design

## 5.3 Subsystem Architecture

This section describes in detail the functions and major processes for the projected Railway.Manage(); application. It defines how data is processed and how the application stored that data. It demonstrates the logical architecture by providing data flow diagrams (DFDs) to define the processes, data flows, and data stores at different levels. Ultimately, this section depicts the association between each and every user with the system.

### 5.3.1 General View of the System

Context DFD illustrates the capacity and limitations of the wished-for Railway.Manage(); application. Context DFD shows the application's boundaries and its relationships with its environment [4]. Figure 15 shows the context DFD to illustrate how the users interchange information within the system.

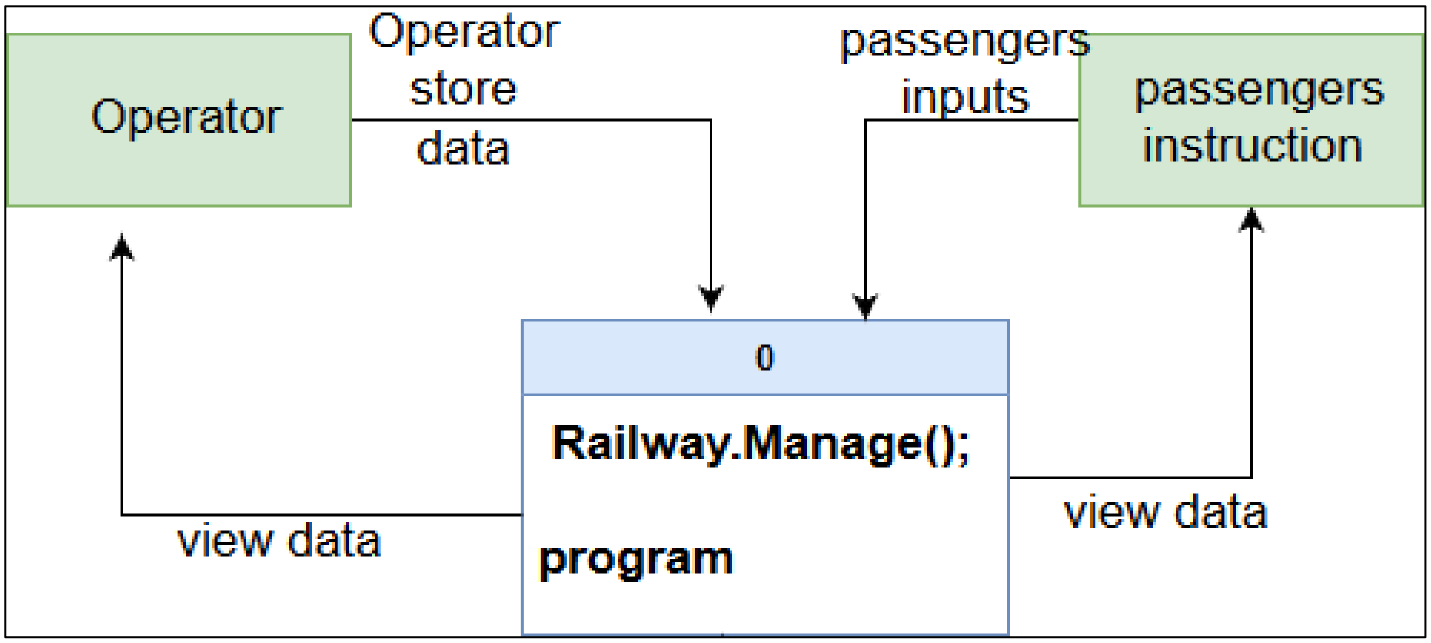
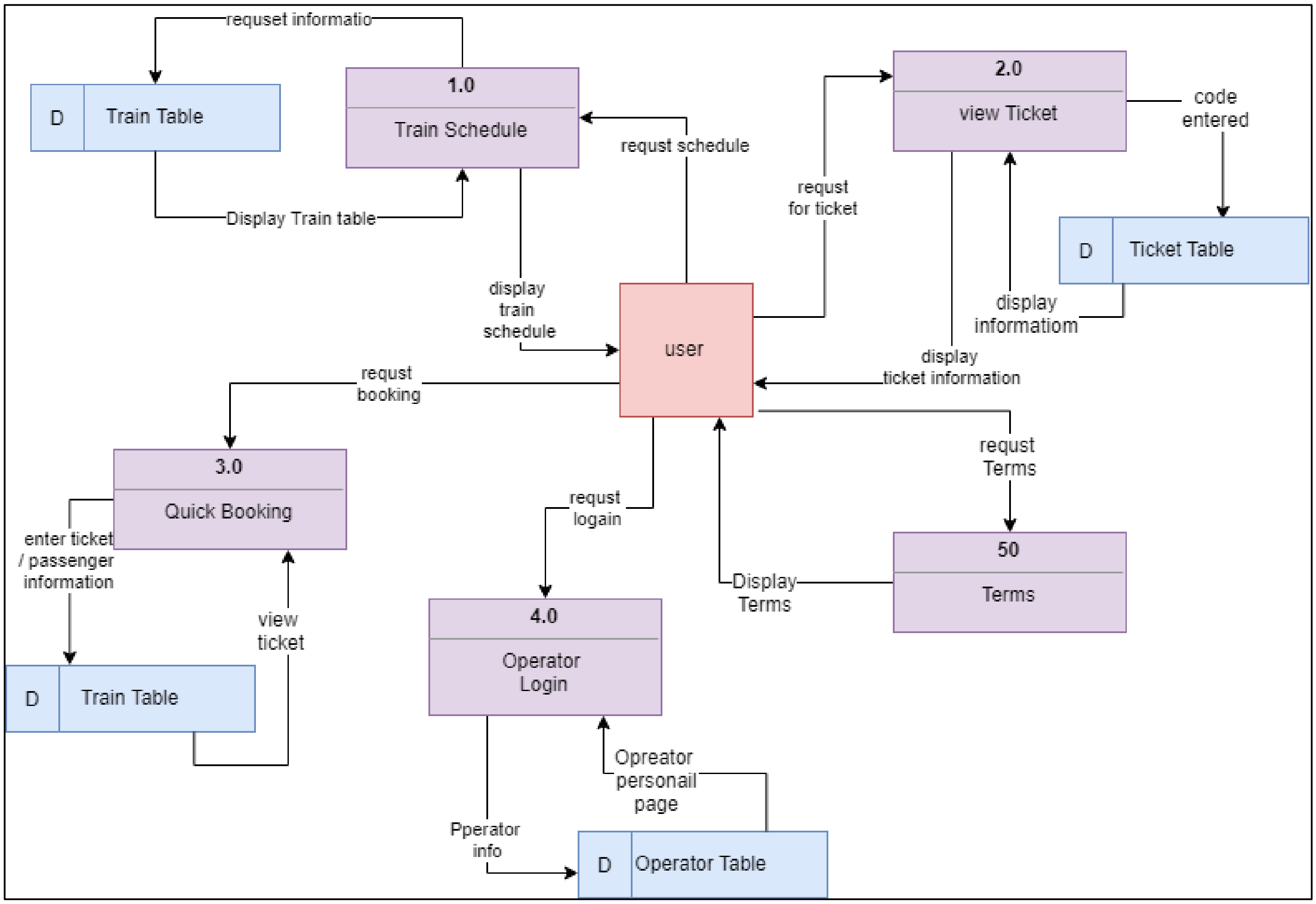


Figure 15 Context Level DFD

### 5.3.2 All Users' Subsystem

This subsection specifies a graphical representation of the common functionalities among all two users, Operator, Passenger and how they interact with the system.

Figure 16 shows the Level-0 DFD of the common functions and illustrates how the data flows between the system and all the users. For the common functionalities, see Software Requirements Specification



(SRS) document, section 3.2.1.

Figure 16 Level-0 DFD for All Users' Subsystem

### 5.3.3 Operator's Subsystem

This section provides a graphical representation of how the Operator of Railway.Manage(); Company interacts with the system. Figure 17 shows the Level-0 DFD of the Operator's subsystem and describes how the data flows between the system and the Operator. For the Operator functional requirements, see Software Requirements Specification (SRS) document, section 3.2.2.

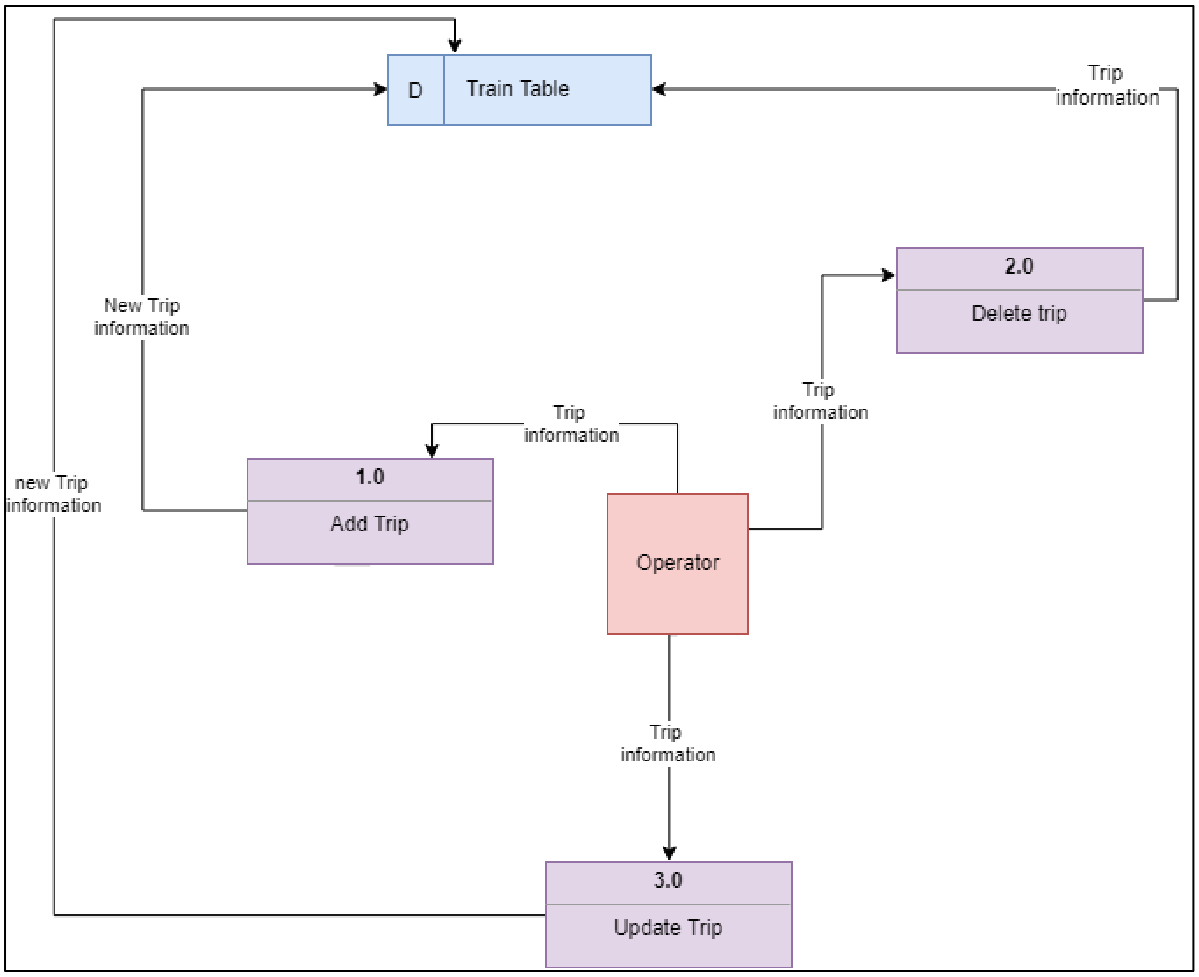


Figure 17 Level-0 DFD for Operator's Subsystem

### 5.3.4 Passenger's Subsystem

This section provides a graphical representation of how a Passenger interacts with the system.

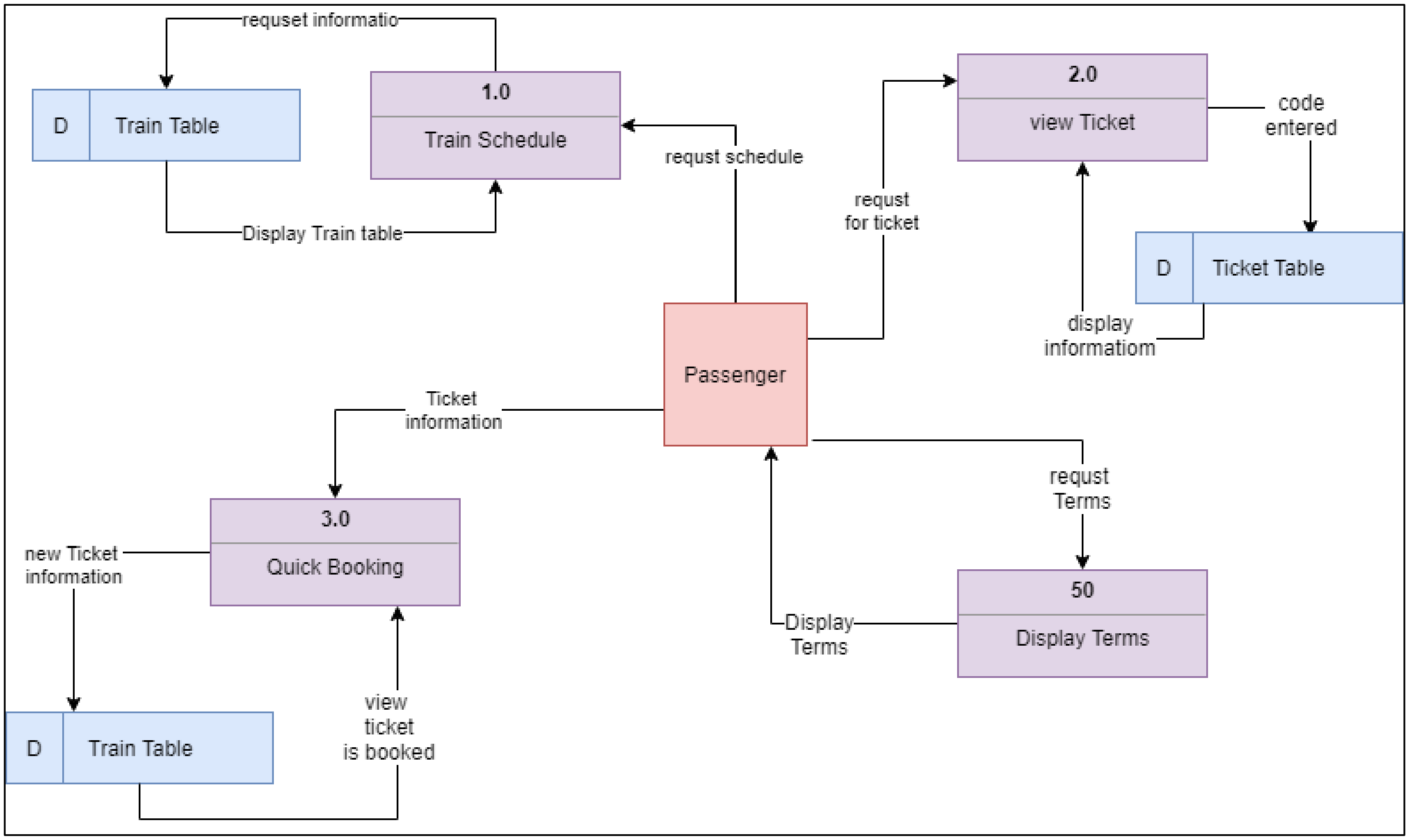


Figure 18 shows the Level-0 DFD of the Passenger's subsystem and describes how the data flows between the system and the Passenger. For the Passenger's functional requirements see SRS document, section 3.2.3.

**Figure 18 Level-0 DFD for passenger's Subsystem**

# 6. Data Design

This chapter provides a description of the data, data types, required fields, list of the application entities and a description of the application's database.

## 6.1 Data Description

Railway.Manage(); data will have stored in single database and it will include all the required data to have a complete functionality. Operators are responsible about all aspects of the database. there are no restrictions for them because they can access any component of the database. In another hand, the passengers can access their own information and the any information related with their ticket. Table 4 below describes the database entities, their required fields, data types, and constraints.

**Table 4: database entities and fields.**

| Entity | Field | Type | Constraints |
|---|---|---|---|
| **Driver** | D_ID | INT(10) | Primary Key, Not Null |
| | First_Name | VARCHAR(45) | Not Null |
| | Last_Name | VARCHAR(45) | Not Null |
| **Train** | Train_ID | INT(10) | Primary Key, Not Null |
| | State | VARCHAR(20) | Not Null |
| | No_Of_Seats | INT(10) | Not Null |
| **Operates** | D_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| | Train_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| **Passenger** | National_ID | INT(10) | Primary Key, Not Null |
| | Train_ID | INT(10) | Foreign Key, Not Null |
| | First_Name | VARCHAR(45) | Not Null |
| | Last_Name | VARCHAR(45) | Not Null |
| | Email | VARCHAR(45) | Not Null |
| | Seat_No | INT(6) | Not Null |
| **Passanger_phone** | National_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| | Phone_number | VARCHAR(10) | Not Null, unique |
| **Station** | Station_number | VARCHAR(10) | Primary Key, Not Null, unique |
| | Location | VARCHAR(45) | Not Null |
| | Name | VARCHAR(25) | Not Null |
| **Stops at** | Station_number | VARCHAR(10) | Primary Key, Foreign Key, Not Null, unique |
| | Train_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| **Trip** | Train_ID | INT(10) | Foreign Key, Not Null |
| | Trip_ID | INT(10) | Primary Key, Not Null |
| | Station_number | VARCHAR(10) | Foreign Key, Not Null, unique |
| | Trip_State | VARCHAR(20) | Not Null |
| | Departure | Date and time | Not Null |
| | Arrival | Date and time | Not Null |
| | Destintion | VARCHAR(20) | Not Null |
| | Ticket_ID | VARCHAR(20) | Foreign Key, Not Null |
| | Trip_ID | INT(10) | Primary Key, Foreign Key, Not Null |

| | | | |
|---|---|---|---|
| **Books** | National_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| | class | Enum('silver','gold' ,'economic') | Not Null |
| **Ticket** | Ticket_ID | VARCHAR(20) | Primary Key, Not Null |
| | National_ID | INT(10) | Foreign Key, Not Null |
| **Luggage** | National_ID | INT(10) | Primary Key, Foreign Key, Not Null |
| | L_ID | INT(10) | Primary Key, Not Null |
| | Weight | VARCHAR(12) | Not Null |

## 6.2 Data Dictionary

This section lists all the entities and describes all the required fields. Table 5 below shows the data dictionary of the anticipated Railway.Manage(); system.

**Table 5: Database Data Dictionary.**

| Entity | Field | Description |
|---|---|---|
| **Driver** | D_ID | Driver's identification number, this attribute is unique for each driver. |
| | First_Name | Driver's first name. |
| | Last_Name | Driver's last name. |
| **Train** | Train_ID | Train's identification number, this attribute is unique for each train. |
| | State | The current state of the train. |
| | No_Of_Seats | Number of seats in all classes in the train. |
| **Operates** | D_ID | Driver's identification number, this attribute is unique for each driver. |
| | Train_ID | Train's identification number, this attribute is unique for each train. |
| **Passenger** | National_ID | Passenger's national identification. This attribute is unique for each passenger. |
| | Train_ID | Train's identification number, this attribute is unique for each train. |
| | First_Name | Passenger's first name. |
| | Last_Name | Passenger's last name. |
| | Email | Passenger's personal email. |
| | Seat_No | Seat number reserved by the passenger. |
| **Passanger_phone** | National_ID | Passenger's national identification. This attribute is unique for each passenger. |
| | Phone_number | Passenger's phone number. This attribute is unique for each passenger. |

| | | |
|---|---|---|
| **Station** | Station_number | Station number. |
| | Location | Location of the station. |
| | Name | Station name. |
| **Stops at** | Station_number | Station number. |
| | Train_ID | Train’s identification number, this attribute is unique for each train. |
| **Trip** | Train_ID | Train’s identification number, this attribute is unique for each train. |
| | Trip_ID | Trip’s identification number, this attribute is unique for each trip. |
| | Station_number | Station number. |
| | Trip_State | Current state of the trip. |
| | Departure | Date and time to depart the station. |
| | Arrival | Date and time to arrive the station. |
| | Destintion | Trip destination. |
| | Ticket_ID | Ticket’s identification number, this attribute is unique for each ticket. |
| **Books** | Trip_ID | Trip’s identification number, this attribute is unique for each trip. |
| | National_ID | Passenger’s national identification. This attribute is unique for each passenger. |
| | class | Train class: gold, silver, economic. |
| **Ticket** | Ticket_ID | Ticket’s identification number, this attribute is unique for each ticket. |
| | National_ID | Passenger’s national identification. This attribute is unique for each passenger. |
| **Luggage** | National_ID | Passenger’s national identification. This attribute is unique for each passenger. |
| | L_ID | luggage’s identification number, this attribute is unique for each luggage. |
| | Weight | Luggage weight. |

## 6.3 Database Description

Railway.Manage(); software is fully designed and depended in its functionalities on the database. It is an essential part of the system. This software project will connect to the database via MySQL RDBMS which is an open source application free for use for anyone to maintain the database.

Based on the previous table in section 6.1, this software has many entities as described. Each entity (table) has many attributes that have common characteristics. These attributes may be single, composite or multi attributes. Entity Relationship Diagram (ER Diagram) is used to display the system's entities, their attributes and their relationship. The following Figure 19: Entity Relationship Diagram. shows the ER diagram.

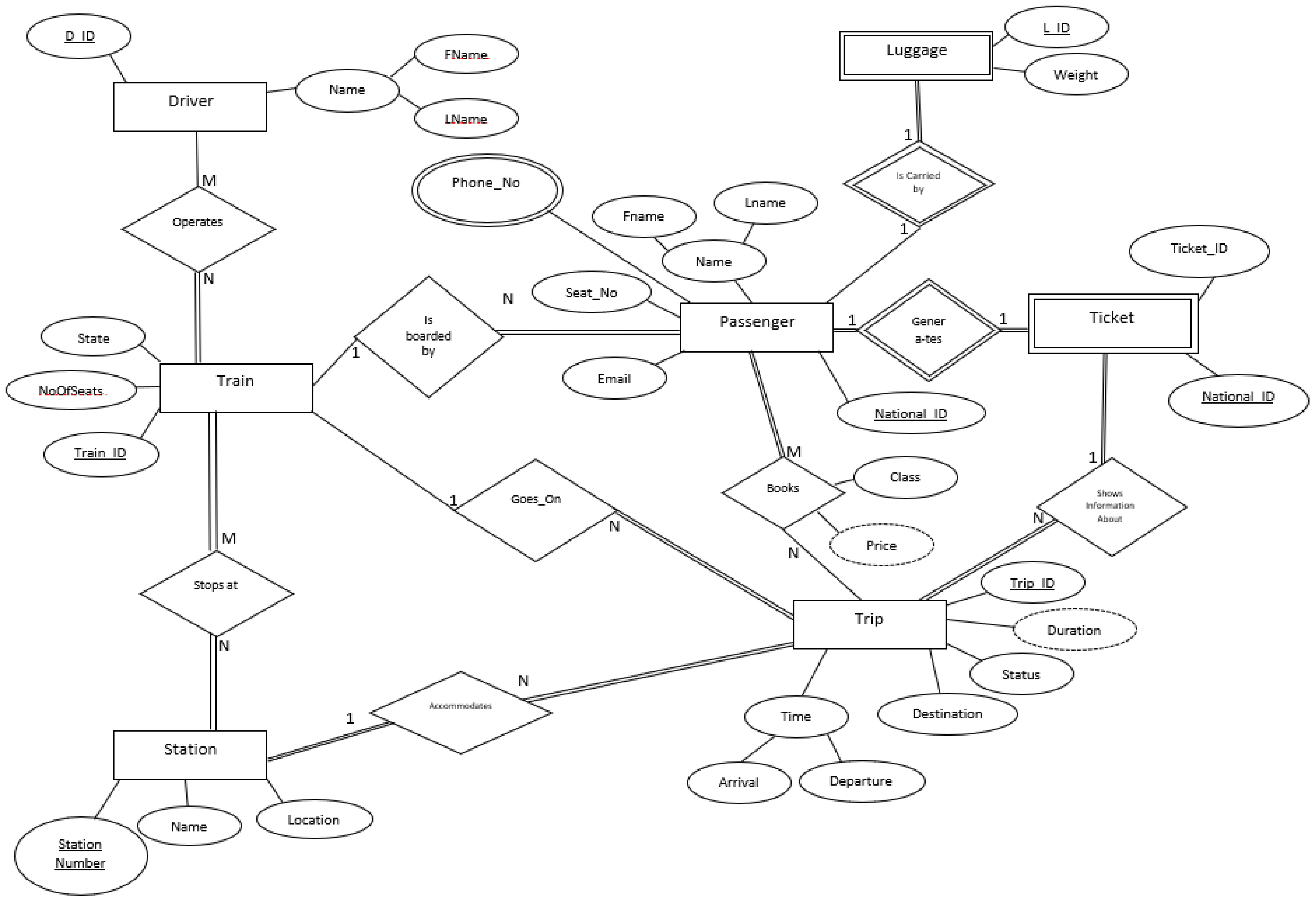


Figure 19: Entity Relationship Diagram.

Moreover, the ER diagram is converted into relational schema diagram (ER mapping) that represent the actual tables and their relationship of a database. Figure 20: logical schema. below shows ER mapping (schema).

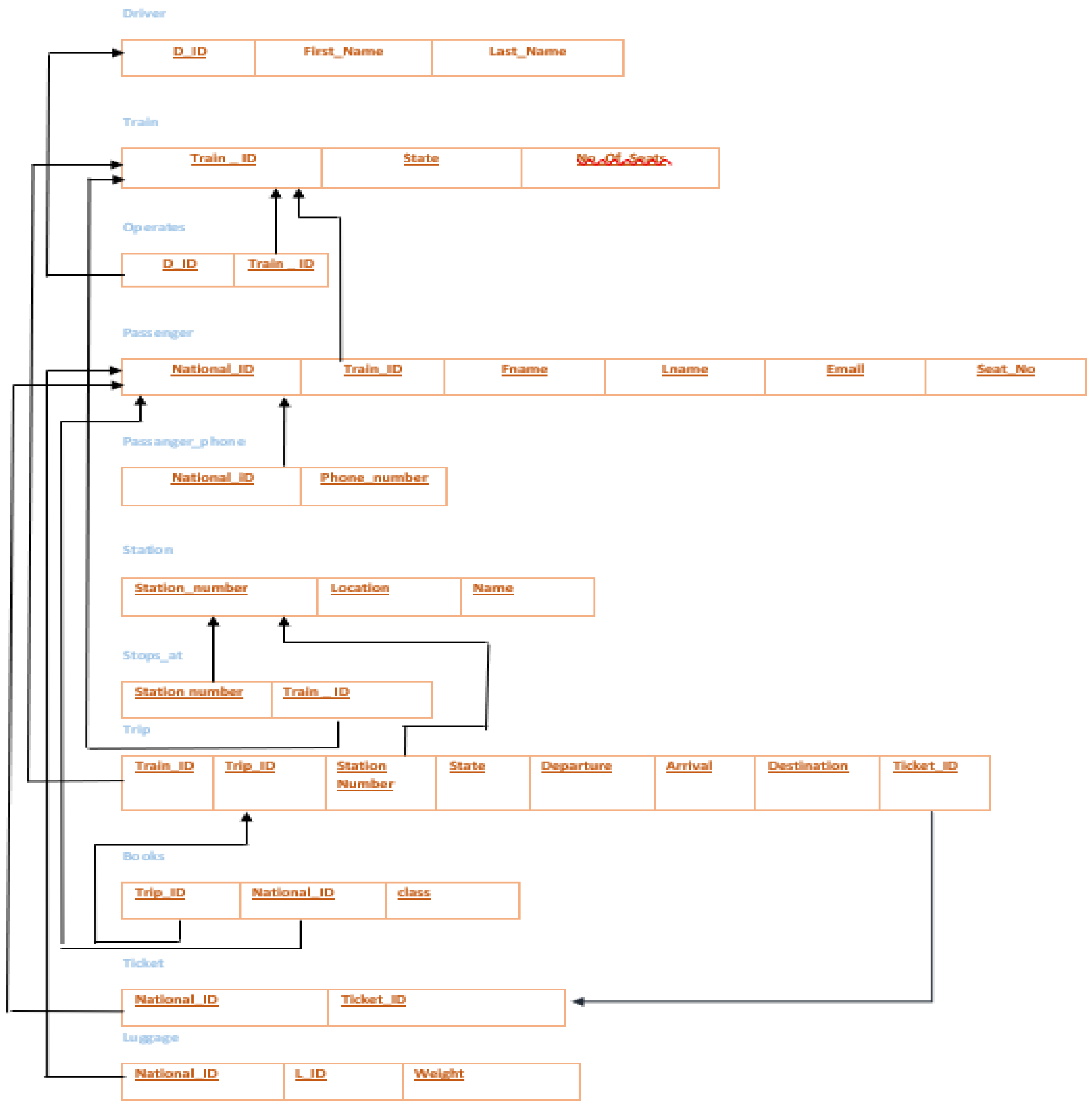


**Figure 20: logical schema.**

## 7. Component Design

In this section, we take a closer look at what each component does in a more systematic way. Pseudocode is an informal language, which is used to help in algorithms development. The following are pseudocode algorithms for the main system's functions. It is expressed in a formally natural language.

### 7.1 Main and Common Functions

#### 7.1.1 Ticket Viewing

```
    Function ViewTicket {

    Input Id_Ticket

If Id_ticket exist in the Database
    → print full information about entered ticket

Else
    → Print "ticket doesn't exist "

}
```

### 7.2 Operator Functions

#### 7.2.1 Log In:

A **log in** feature is provided in the system **and** operators are the ones who must log in before accessing their functions, while passengers are only required to enter their information.

```
Function login()
{
Counter =0
   //  Input username and password
 If the username and password exist in database and counter != 3
         Go to user's "Homepage" interface.
If the username and password don't exist in database and counter ! =3
              Increment counter by 1
             Re-input username and password
If the username and password don't exist in database and counter ==3
             Contact administration if you lost your password.
}
```

### 7.2.2 Modifying Schedule:

The Operator is responsible for editing and maintaining the train schedule, which is shown in the various monitors throughout the train station.

#### *7.2.2.1 Add Trip*

```
Function AddTrip ()
{
Input departure and arrival station
If the departure and arrival station not exist in specific time
  →    New trip will add it in trip Schedule .
Else
  →    Will consider it a trip is exist in the database
}
```

#### *7.2.2.2 Delete Trip*

```
Function DeleteTrip(){

Input Id_Trip

If the state of the Trip is "Arrived "
    → Delete Trip

Else if input of the Id_Trip in the trips Schedule
     → Delete Trip \\ cancelation state }
```

#### *7.2.2.3 Edit Trip*

```
Function EditTrip(){

Input Id_Trip
If Id_Trip exist in Train Schedule
→ edit on trip departure or arrival time
```

## 7.3 Passenger Function

### 7.3.1 Passenger information

```
Function PassengerInfo(){

Enter First and Last Name
Enter national Id
Enter contact information

Store it on the DB

Go to next Step booking

}
```

### 7.3.2 Passenger Booking

```
Function Booking (){

Choose a trip way
    If it's one way or around way

Choose departure and arrival station

Choose a seat class
Economic, Gold, Salver
Choose number of Adults and child

Store all these information on DB}
```

## 8 Detailed System Design

This section includes the classification and definition of every system component and its main responsibility. The classification, definition, and responsibilities for each component is explained in the Table 8.1 below:

### 8.1 Classification, Definition and Responsibilities

This part includes the classification and definition of every system function and its main responsibility. The classification, definition, and responsibilities for each component is explained in the Table 6below:

**Table 6 Classification and Definition of Railway.Manage(); Components**

| Component | Classification | Definition & Responsibilities |
|---|---|---|
| | Common Functions | |
| **Ticket Viewing** | Function | This Function enable users to view their own ticket with full information |
| | Operator Function | |
| **Log In** | Function | This function enables different Operator to access them accounts by providing their usernames and passwords. |
| **Add Trip** | Function | This function enables Operator to add a new trip |
| **Delete Trip** | Function | This function enable operator to delete specific trip either it's canceled or its state is arrived. |
| **Edit Trip** | Function | This function enable operator to update any trip in train schedule. |
| | Passenger Function | |
| **Passenger Information** | Function | This function enable passenger to enter full info before register a trip to print it on the ticket |
| **Booking** | Function | This function enable passenger to book a trip with full specification. |

## 8.2 Constraints and Composition

Table 7 Railway Manage(); Constraints and Conditions

| Component | constraint | Pre-condition | Post-condition |
|---|---|---|---|
| | Common Functions | | |
| **Ticket Viewing** | Function | Ticket_id must be valid | Start search about a ticket with valid Id |
| | Operator Function | | |
| **Log In** | The operator must have an account. | Provide a valid username and password | Start authentication process with entered user name and password |
| **Add Trip** | Departure and arrival station must be existing | Trip does not exist in train schedule. | Train schedule include new trip |
| **Delete Trip** | A trip must be existing in DB | Trip does exist in train schedule. | Consider trip is canceled or already arrived. |
| **Edit Trip** | A trip must be existing and update it with valid input. | Trip does not exist in train schedule. | Update trip schedule with new updates. |
| | Passenger Function | | |
| **Passenger Information** | Function | All entered info are valid | Passenger can not book many ticket at the same time . |
| **Booking** | Function | All entered info are valid | Ticket must valid until trip end. |

## 8.3 Uses/Interactions

### 8.3.1 Common function

*8.3.1.1 view Ticket*

Figure 26 show ticket viewing for operator and passenger.

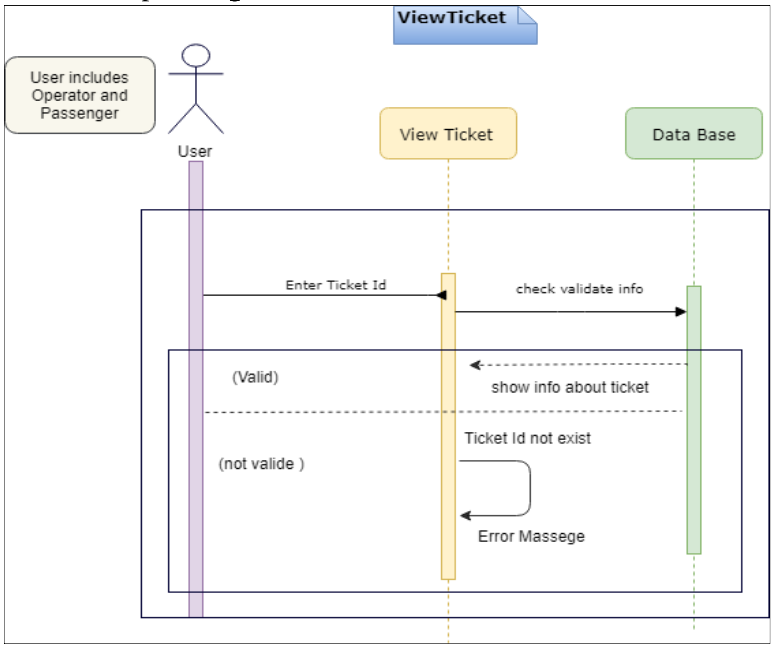


Figure 21 "View Ticket "sequence diagram

### 8.3.2 Operator function

#### *8.3.2.1 Log In*

Figure 27 show “Log In” sequence diagram

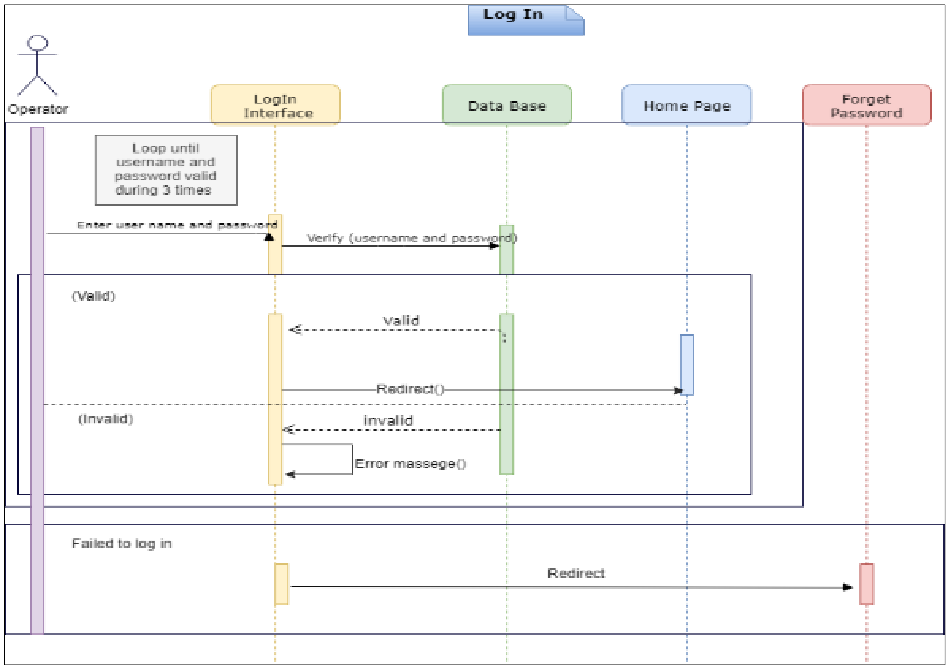


**Figure 22 "Log In" sequence diagram**

#### *8.3.2.2 Add Trip*

Figure 28 show “Add trip” sequence diagram

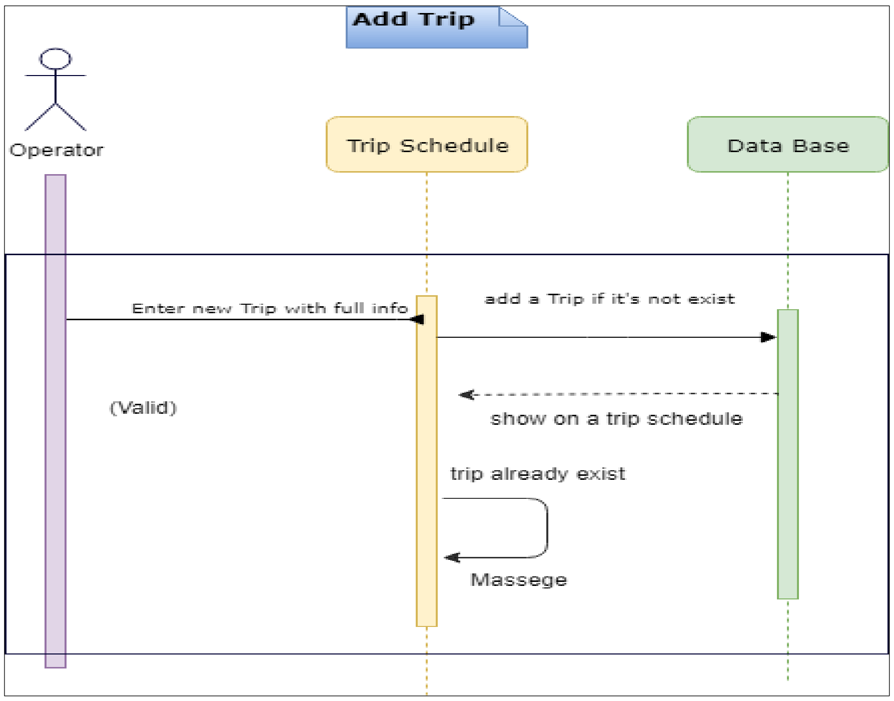


**Figure 23 "Add Trip" diagram**

### *8.3.2.3 Delete Trip*

Figure 29 show “Delete Trip” sequence diagram

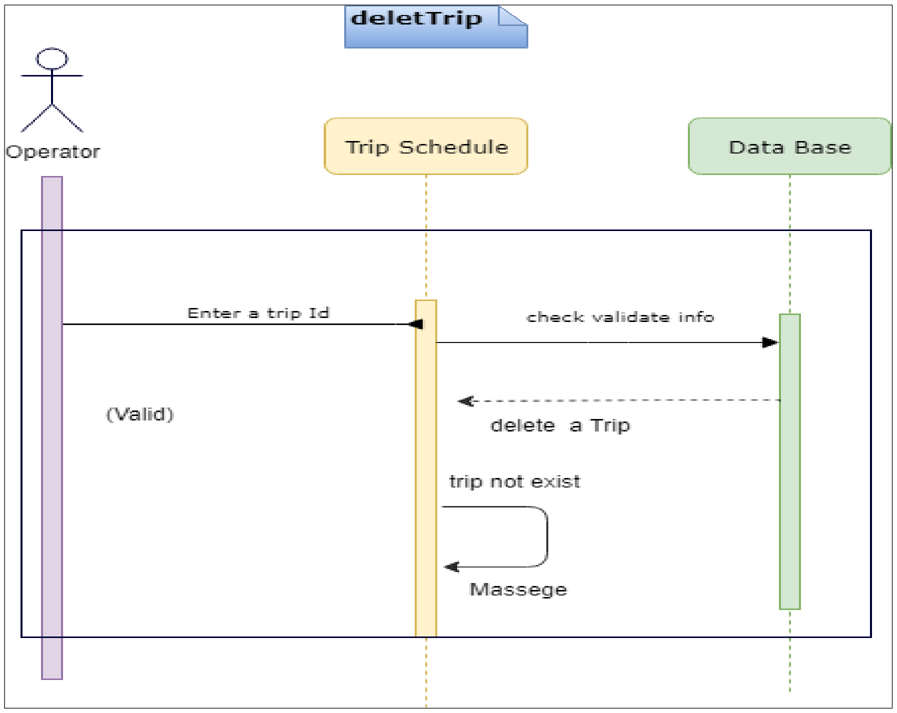


**Figure 24 "Delete Trip" sequence diagram**

### *8.3.2.4 Edit Trip*

Figure 30 show “Edit Trip “sequence diagram

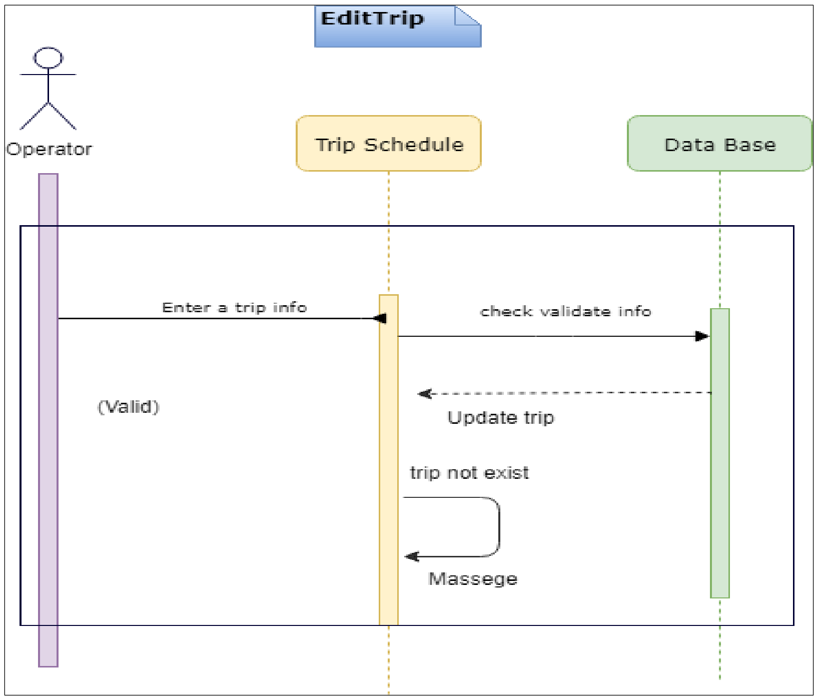


**Figure 25 "Edit Trip" sequence diagram**

### 8.3.3 Passenger function

#### *8.3.3.1 Passenger info and booking functions*

Figure 27 show "passenger info" and "Booking "sequence diagram

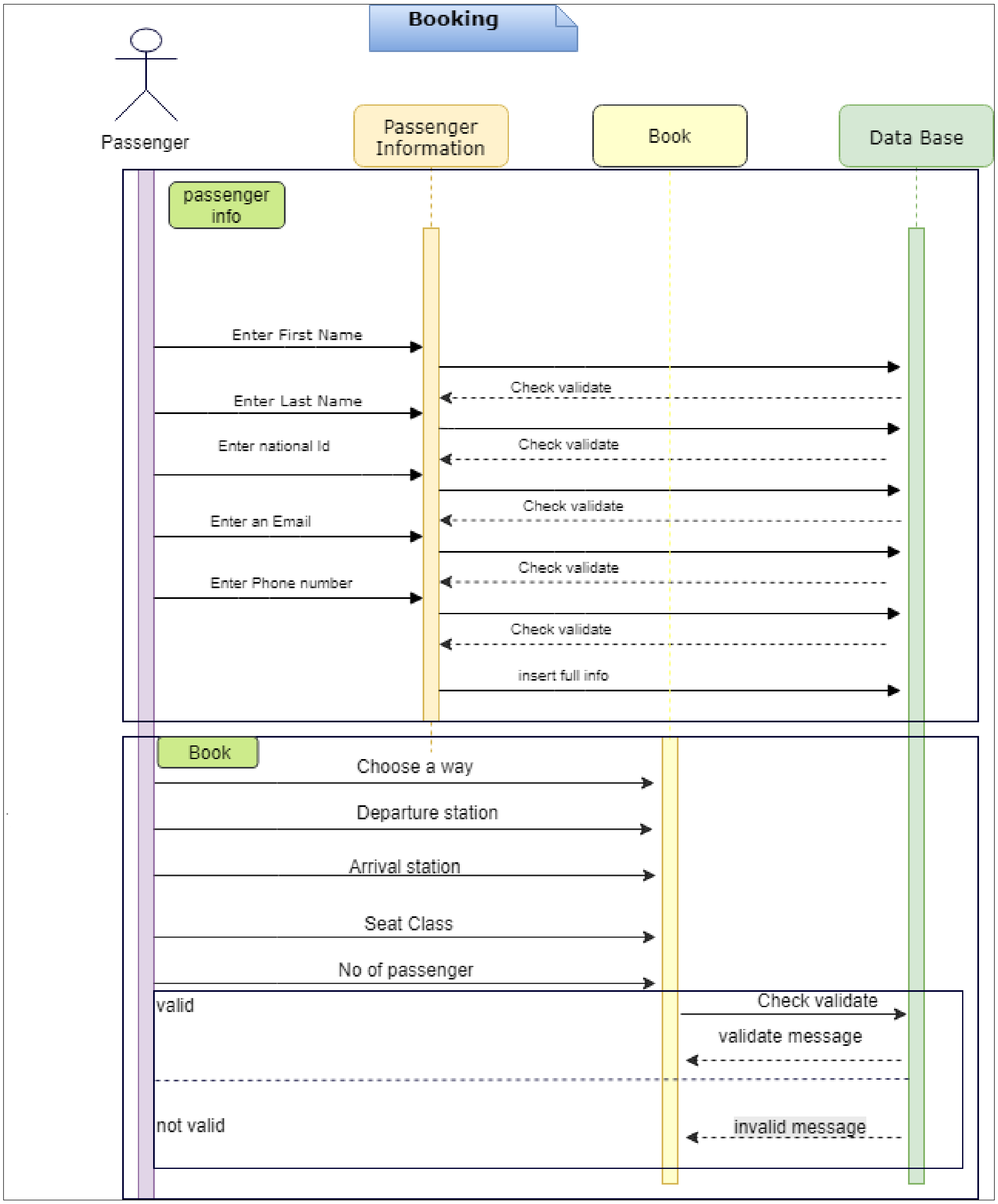


Figure 26 "Passenger info" and "booking " sequence diagram "

## 8.4 Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.
Table 8 shows the resources' specification:

Table 8 Railway.Manage(); Resource Specification

| Component | Classification |
|---|---|
| **Database** | mySQL |
| **Memory Storage** | 4 MB free |
| **OS** | Window 10 and Macintosh operating systems |
| **System Code** | Netbeans |

Table 9 below shows a race condition that may occur in Railway.Manage();:
Table 9 Railway.Manage(); Race Condition

| **Race condition** | solution |
|---|---|
| A passenger may not see a recent update to the Train schedule, if the operator updates some trips as the same time | They should to check their trip from a ticket or they can to contact with operator to make sure . |

Table 10 below show **Deadlock situation may occur in Rail.Manage();**
Table 10 Railway.Manage(); Deadlock situation

| Deadlock situation | Description | Solution |
|---|---|---|
| **Device turned off** | When a Passenger register their own ticket and their device turned off before recording their trip. | They can to check their ticket if it's not exist they should to register again |
| **More than one ticket** | When a passenger wants to register more than one ticket at the same time. | They should to record each ticket separately. |

### 8.5 Processing

In this system, each component is broken into inputs and outputs, with each having some constraints to govern its functionality. The following subsections explain these matters in detail.

**User 1: Passengers.**

#### 8.5.1 View Train Schedule

Through this function, the user is able to view the trips' schedule. Table below describes these functions thoroughly.

| | |
|---|---|
| **Description** | **This function provides the user with the ability to observe the trip schedule in a specific point in time.** |
| **Input** | A stored table with these following attributes:<br>• Trip ID<br>• Status<br>• From (Departing Station)<br>• To (Arriving Station)<br>• Departure time<br>• Arrival time |
| **Output** | The following should be printed to the user accurately.<br>• Trip ID<br>• Status<br>• From (Departing Station)<br>• To (Arriving Station)<br>• Departure Time<br>• Arrival Time |
| **Constraint** | No SQL exceptions should occur during the retrieval of this information. |

Table 11 View Trip Schedule Processing

#### 8.5.2 View Ticket

This function allows the user to review their ticket's information, table below outlines the functions' inputs, outputs and constraints.

| | |
|---|---|
| **Description** | **Though this function, the user is able to review their ticket information.** |
| **Input** | A stored table with these following attributes:<br>• Passenger ID and name.<br>• Trip ID<br>• Class |

| | |
|---|---|
| | • Departure Time<br>• Arrival Time<br>The user should enter their generated ticket number in order to view this information first, this generated number will appear after the user has done their booking successfully. |
| **Output** | The following should be printed to the user accurately.<br>• Passenger ID<br>• Trip ID<br>• Class |
| **Constraint** | • No SQL exceptions should occur during the retrieval of this information.<br>• The user should enter a valid generated number. |

Table 12 View Ticket Processing

### 8.5.3 Quick Booking

In order to book successfully, the user is required to go through several stages. These stages are described in detail below.

#### 8.5.3.1 Passenger Info

Firstly, the user is asked to enter their personal information. This information is essential to identify the passenger in the database. The following table outlines the function's properties.

Table 13 Passenger Info Processing

| **Description** | **The user is asked to enter their personal information in order to book successfully.** |
|---|---|
| **Input** | Personal Information:<br>• First name<br>• Last name<br>• National ID<br>Contact Information:<br>• Phone number<br>• E-mail |
| **Output** | • The user's information will be stored in the database. |
| **Constraint** | • No SQL exceptions should occur during the recording of this information.<br>• The user should enter all the required information.<br>• All entered values should correspond to their specified data types in the database. |

#### 8.5.3.2Quick Booking

After entering their personal information, users are able to advance to booking their trip/s. Table below explain the different aspects of this functionality.

Table 14 Quick Booking Processing

| | |
|---|---|
| **Description** | **Various aspects of booking are done in this stage, where the user is asked to specify several information regarding their booking.** |
| **Input** | • Specifying whether it is a round trip or one-way.<br>• Selecting the departure station and the arrival station.<br>• Selecting the passenger class.<br>• Selecting the number of adults and children.<br>• Selecting from the available timings for departure and arrival. |
| **Output** | • Booking will be stored in the database. |
| **Constraint** | • No SQL exceptions should occur during the recording of this information.<br>• The user should enter all the required information.<br>• All entered values should correspond to their specified data types in the database.<br>• If the user selects a trip outside of the trips mentioned in the database, an error will occur. |

#### 8.5.3.3 Seat Selection

Before finishing their booking, the user is finally asked to select a seat in the train. Refer to the following table for this function’s specifications.

Table 15 Seat Selection Processing

| | |
|---|---|
| **Description** | **User should select their prefered seat during this stage of booking.** |
| **Input** | • Selecting the seat via pressing the button that corresponds to it. |
| **Output** | • Seat number is now saved in the database. |
| **Constraint** | • No SQL exceptions should occur during the recording of this information.<br>• The user should select at least one seat. |

#### 8.5.3.3Successful Booking

After completing the booking successfully, the user is informed of the completion of their booking and is supplied with a generated ticket number that could be user to view the ticket information.

Table 16 Successful Booking Processing

| | |
|---|---|
| **Description** | **This function is necessary to inform the user of the successful completion of their booking, and to provide a generated number that could be used to display the ticket information if user desires to do so.** |
| **Input** | • Successful booking. |
| **Output** | • Generated number. |
| **Constraint** | • No SQL exceptions should occur during the recording of this information. |

**User 2: Operators.**

### 8.5.4 Operator Login

To ensure the security of the operators' functionalities, the user is required to login first.

Table 17 Login Processing

| | |
|---|---|
| **Description** | **Before accessing the operators' portal, successful login should be completed first by entering the valid username and password of the operator.** |
| **Input** | • Username and Password. |
| **Output** | • Access to the operator portal. |
| **Constraint** | • No SQL exceptions should occur during the recording and retrieval of this information.<br>• All values should be entered (Username and Password)<br>• The two values should correspond to the recorded values in the database. |

### 8.5.5 Operator Portal

After the operator log in successfully, they are granted permission to update the trips information in the database. Either by deleting, inserting or editing an entry in the trips' table in the database.

Table 18 Operator Portal Processing

| | |
|---|---|
| **Description** | **In this function, the operator is able to manage the trips' table in the database. This is demonstrated by editing, deleting or adding a trip in the table.** |
| **Input** | • Selection of the desired operation<br>• Trip ID<br>• From (Departing Station) |

| | |
|---|---|
| | • To (Arrival Station)<br>• Departure Time<br>• Arrival Time |
| **Output** | • Successful manipulation of the desired operation (deleting, adding or editing) |
| **Constraint** | • No SQL exceptions should occur during the recording and retrieval of this information.<br>• All values should be entered<br>• All entered values should correspond to their specified data types in the database. |

### 8.6 Interfaces/Exports

Every developer in the team should be aware of the composition of the various components in the system, this knowledge is essential to ensure that the system will function as envisioned by both the client and the developers.
In light of this, this section will be useful in providing an outline and references to the pre and post condition of each requirement in the system, constraints and resources needed to complete these requirements successfully.

Refer to section 8.2 in the SDS document to understand the components pre-conditions, and post conditions. Furthermore, in Section 8.5 functions are discussed in greater depths by outlining each function's inputs, outputs and possible constraints generated by this function.

### 8.7 Detailed Subsystem Design

In order to comprehend the subsystems of the project and how they are interpreted and connected, this section will provide a brief explanation and reference to understand the application's flow and the components' behavior of the system.
This information was previously discussed via activity diagrams in section 3.2 of the SRS document. Please refer to them if necessary.

## 9 Requirements Traceability Matrix

Table 19 explains the requirements traceability matrix.

Table 19 Requirements Traceability Matrix

| Type of User | Technical Assumption(s) and/or Customer Need(s) | Functional Requirement | Associated ID in SDS | System Components |
|---|---|---|---|---|
| **Common Functionalities** | The user should be able to check the trip schedule from the database. | View Train Schedule | 7.1 | Viewing Schedule |

| | | | | |
|---|---|---|---|---|
| **User 1: Passenger** | The user should be able to view their ticket information by entering the generated number after booking successfully. | View Ticket | 7.1.1 | Viewing Ticket |
| | Before Booking, the user is required to enter their personal and contact information. | Enter Passenger Information | 7.3.1 | Passenger Information |
| | The user should able to select the trip they wish to go on and book a ticket. | Quick Booking | 7.3.2 | Passenger Booking |
| | After booking, the user should select their preferred seat in the train. | Seat Selection | | |
| | After booking successfully, the user should be informed of its completion and should be provided with a generated number. | Booking Confirmation | | |
| **User 2: Operator** | Before accessing the operator portal, the operator should login by entering their username and password. | Login | 7.2.1 | Log in |
| | The operator should be able to delete, add, or edit an entry in the trip's table in the database with no issues. | Add a new trip | 7.2.2.1 | Modifying Schedule |
| | | Delete a trip | 7.2.2.2 | |
| | | Edit a trip | 7.2.2.3 | |