**HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH-QUALITY TRAINING**

**COURSE NAME: Database Management System**

**🙡🙠✵🙢🙣**



**FINAL PROJECT REPORT**

**Project name:**

BUS TICKET BOOKING MANAGEMENT SYSTEM

**Lecturer:** Prof. Nguyễn Thành Sơn

**Course ID:** DBMS330284E\_22\_2\_01FIE

**Group:** 5

**Date:** 2nd Sem/2022-2023

*Ho Chi Minh city, May , 2023*

*(This page was intentionally left blank)*

**LIST OF STUDENTS – GROUP 5**

**Project:** Bus ticket booking management system

|  |  |
| --- | --- |
| ***ID*** | ***Full name*** |
| 21110758 | Lê Xuân Cường |
| 21110092 | Bùi Quốc Thông |
| 21110066 | Phạm Vũ Bảo Nhân |
| 21110785 | Mai Nguyễn Nhật Nam |

**Professor’s comment**

Ho Chi Minh city, May …, 2023

Grading

**CONTENTS**

[PROLOGUE 4](#_Toc133495881)

[INTRODUCTION 5](#_Toc133495882)

[CHAPTER I: SYSTEM OVERVIEW 6](#_Toc133495883)

[1. Specifications 6](#_Toc133495884)

[1.1. Problem statement 6](#_Toc133495885)

[1.2. Overview 6](#_Toc133495886)

[2. Real-life functions 9](#_Toc133495887)

[3. Main application functions 10](#_Toc133495888)

[CHAPTER II: SYSTEM ANALYSIS AND DESIGN 12](#_Toc133495889)

[1. Conceptual level database design 12](#_Toc133495890)

[2. Logical level database design 12](#_Toc133495891)

[3. Required constraints 13](#_Toc133495892)

[4. Database and constraints implementation 18](#_Toc133495893)

[5. Other constraints 24](#_Toc133495894)

[6. Constraint-checking triggers 27](#_Toc133495895)

[7. Database views implementation 28](#_Toc133495896)

[CHAPTER III: FUNCTION AND PROCEDURE DESIGN 34](#_Toc133495897)

[1. Connect to database 34](#_Toc133495898)

[2. Function design 35](#_Toc133495899)

[3. Procedure design 37](#_Toc133495900)

[CHAPTER IV: USER CREATION AND PRIVILEGE DISTRIBUTION 44](#_Toc133495901)

[CHAPTER V: SYSTEM INTERFACE DESIGN 52](#_Toc133495902)

[1. Applications and services used 52](#_Toc133495903)

[2. Software interface 52](#_Toc133495904)

[3. Functionalities **Error! Bookmark not defined.**](#_Toc133495905)

# PROLOGUE

Firstly, we would like to express our gratitude to Prof. *Nguyễn Thành Sơn* for his whole-hearted instructions that helped us finish our final project for the Database Management System course. Thanks to the knowledge the professor has provided us, we were able to firmly grasp the basic knowledge and foundation for building a database management system. And through this project, our group would like to present the development process of a database management system and demonstrate by programming a related project once again.

During the process of executing this project, it will be hard to avoid mistakes. Because of that, we would love to get the professor’s suggestion on improving our work so it would be more functional and complete. We wish you good health and the best of luck pursuing the path of teaching.

Finally, we would like to thank all the teachers and classmates who studied with us on this course and offered us support while we carried out our final project.

# INTRODUCTION

In recent years, the Information and Technology (IT) field has been integrated into our society and daily lives, regardless of any field and/or occupations. It also plays an important part of booking management in Vietnam and especially in almost every country as there are many applications made to help fix problems that big organizations frequently face.

The creation of the bus ticket booking management system is the result of many developers’ creativity and hard work with the aim of aiding companies in managing their businesses.

With that in mind, to better understand the application and role of Information and Technology (IT) in Database Management, we have decided on the **“Bus ticket booking management system”** as our final project.

# CHAPTER I: SYSTEM OVERVIEW

## Specifications

### Problem statement

The bus ticket booking management system will:

* Manage the employees, passengers, bus, trips, routes easier.
* Convenient for users to check and book trips.
* Check the state and location of the trip more clearly through a map.
* More convenient for the bus company to obtain statistics: revenue, number of passengers, number of trips, employee salary, outcome, etc. per day, per month, per year.

Vehicle management: Manage travel vehicles including their location, date and time of arrival/departure, price, etc.

System management: Manage employees, drivers, customers, travel curriculum.

Statistics: Employee statistics, vehicle statistics, daily sales, etc.

### Overview

A bus company needs to have a bus ticket reservation system. The bus ticket reservation system should contain the following data:

The bus company manages a lot of agents. Each agent has: agent ID, place id, cash reserve ID, address, agent name.

Each agent has only one cash reserve. A cash reserve includes cash reserve ID and counter.

An agent has many employees. Each employee has: employee ID, position ID, account ID, agent ID, name, address, phone number, identity number, salary, email, date of birth, state.

The employee state can be:

* Not working
* Working

Each employee is provided with an account to access into the system (username and password). Each employee type has a different position.

The information of the position group contains: position ID, type.

There are several types:

* Administrator
* Travel planner
* Travel supervisor
* Driver
* Ticket seller
* Service guide
* Security guard
* Porter

Each position group has separate privileges. The information of the privileges group includes: privilege ID, name.

The agent manages passengers. Each passenger has: passenger ID, name, phone number, address, identity number, gender, email.

The gender attribute of passenger above has two options:

* Male
* Female

Easily manage and filter the address of stations in the general local area, there is information of places: place ID, region.

Each passenger can choose a pickup station and drop-off station. Each station has: station ID, detailed address, name, capacity, parked bus number.

The bus of each brand has: bus ID, registration number, model, capacity, status, type.

Status of the bus can be:

* Ongoing
* Idle
* Break
* Incident

Type of the bus can be:

* Interprovince
* Transit

Routes involving the journey have: route ID, start bus station ID, final bus station ID, travel distance.

Each trip is set up by the travel planner which includes: trip ID, drivers ID, bus ID, route ID, departure time, duration, number of booked seats, state.

The state of trip above has three options:

* Waiting
* Going
* Finish
* Cancel

The drivers ID in the trip relation is an attribute of TRIP\_DRIVER relation: trip ID, driver ID. Note that driver ID is a multivalued attribute.

The agent distributes tickets to the passenger. Each ticket has: ticket ID, trip ID, passenger ID, status, fare, type, seat number.

The status of the ticket can be:

* Available
* Bought

The type of ticket has two options:

* Seat ticket
* Sleeper ticket

The agent manages the booking transaction. Each booking transaction includes: transaction ID, ticket ID, passenger ID, employee ID, booking time.

Each driver has an employee ID number, license level and type of driver (long-haul driver and transit driver), state.

The state of driver can be:

* Not drive
* Is driving

General rules:

* Each employee can take on more than 1 position
* Each passenger can book more than 1 ticket
* Each trip can have more than 1 driver

The bus company provides a delivery service so that the customers can send a package without booking a ticket. They must provide information about their packages such as: mass, the phone number of sender and receiver. This package will have an ID and price. The package’s price is determined by a pre-determined pricing policy: ID, mass of package and price\_per\_km.

When a big event happens, the bus company will hold discount periods to lower the price of tickets.

Besides, the refund policy can help the passengers receive part of the fare when they cancel their trips and tickets.

## Real-life functions

#### Booking period

*\* Offline booking:*

The service guide records the passenger’s full field information including: their name, ID number, phone number, address, gender, email. Then, the ticket seller checks again to guarantee all the required fields are correctly fielded.

Then, the passenger picks a trip by choosing from multiple options: destination, pickup station, drop-off station, departure time, the available seat, ticket type. Options will be planned by the travel planner, so the passenger must follow this template.

Then, the ticket seller verifies the customer’s selection. If valid, the ticket seller informs the passenger and waits for their confirmation. If they confirm, the ticket seller prints the ticket, gives it to them and reminds them to arrive at the correct time on the ticket. Else if they refuse, the customer needs to modify the information.

*\* Online booking:*

First of all, the passengers must have an account to access the bus ticket booking application. If they don’t have an account yet, they have to register and log in to book the ticket. If they have an account, they only need to log in to book.

Afterwards, passengers will access the system to book their ticket. They will fill in the information about their name, ID number, phone number, address, gender, email, destination, pickup station, drop-off station, departure time, the available seat, ticket type. The system will send a verification code through email, then passengers fill in the app to verify their booking action.

Next, the system will provide information about the ticket and passengers will have to pay the ticket fare via online payment.

#### Departure period

The passengers wait for the agent. 15 minutes before the departure time of the trip, the vehicle will take the passengers to the bus station.

At the bus station, the porter put the passengers’ luggage into the trunk.

When it’s time, the service guide instructs passengers to the vehicle, and provides water and tissues to them.

#### Drop-off period

When the bus arrives at the last bus station, the porter takes passengers’ luggage from the bus and gives it to the passenger.

#### Ticket cancellation/time change period

*\* Offline cancellation/time change (in the agent):*

The passengers must go to the agent of the bus system and have the ticket-selling employee cancel or change their ticket. In some different cases, the fee of the cancellation/change is also different.

*\* Online cancellation/time change (on the application system):*

The passengers must access the application that they had booked their tickets to change or cancel their ticket. In this case, they must pay for this change.

#### Delivery period

The customer must take their packages before the time of departure of the bus. The employee will measure the weight of the packages and inform the customers with the incurred fees.

## Main application functions

Administrator (global):

* Add, modify, delete, authorize for positions
* Add, modify, delete employee of the position
* Statistic information about trip, the number of sold tickets

Travel planner:

* Add, modify, delete trips
* Add, modify, delete routes
* Add (distribute the tickets of the trip), modify, delete tickets

Travel supervisor:

* Add, delete passengers of the trip
* Report errors (trip, route, passenger, booking)

Ticket-selling:

* Add, modify, delete passenger
* Export bill
* Export ticket
* Change the state attribute of trips

Passenger (when booking online):

* Check price ticket of each route
* Check the information about booked tickets
* Book one or many tickets
* Change the information about ticket (information of passenger, the route, departure time, departure date)
* Cancel their tickets
* Export their tickets

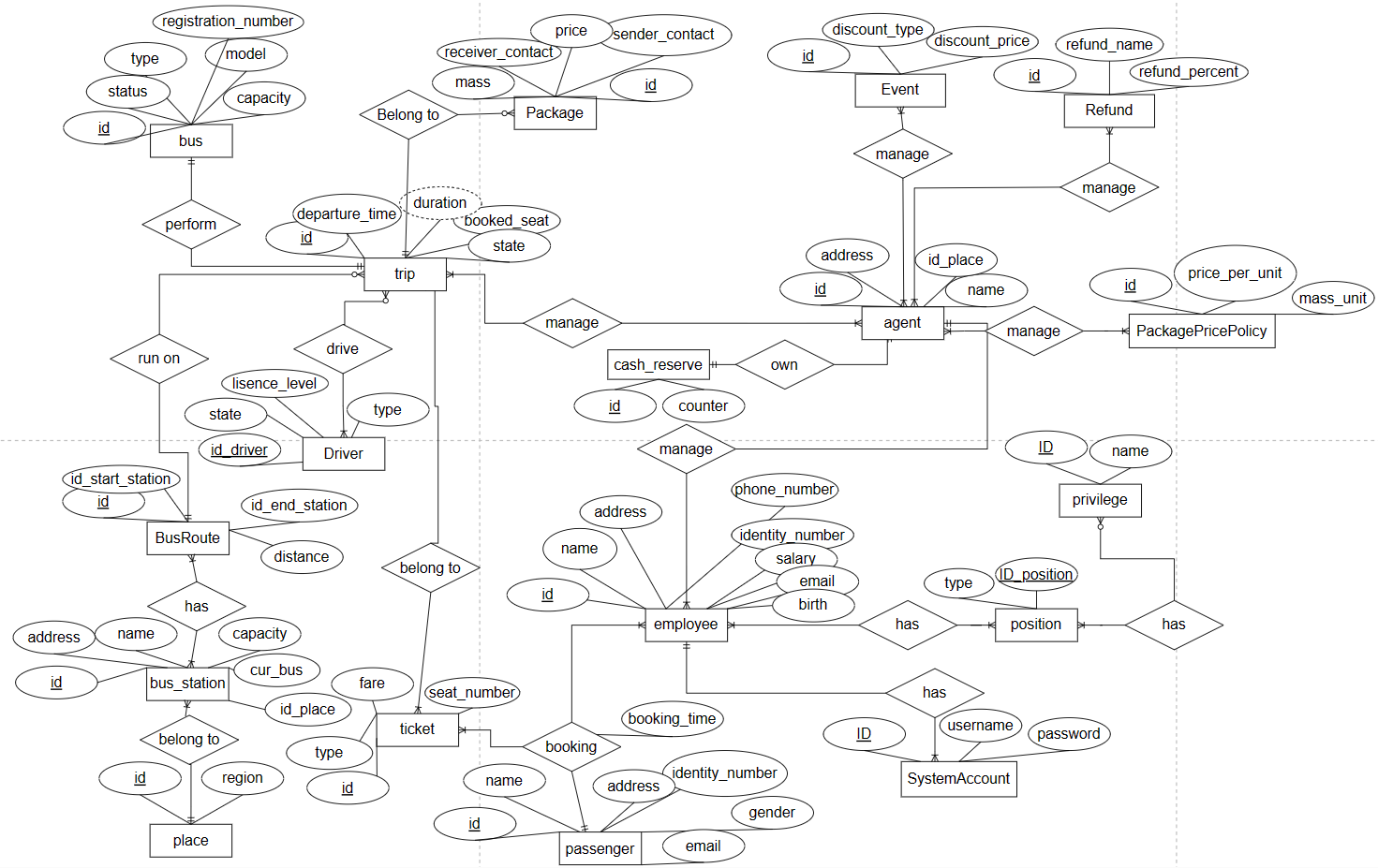
\* Authorization:

* Admin: Full control on the whole system – Global privilege
* […] (Other privileges): Local privilege

# CHAPTER II: SYSTEM ANALYSIS AND DESIGN

## Conceptual level database design

From the necessary data in description of the specifications, the following Entity Relationship Model (ERD) is formed.



Total 18 relations with 9 N-N, 1 three-way relation.

Sharp image: [ERD sharp image (busticketbookingerd.netlify.app)](https://busticketbookingerd.netlify.app/)

## Logical level database design

From the Entity Relationship Model (ERD), we have:

1. BUS (id\_bus, registration\_number, model, capacity, status, type)
2. TRIP (id\_trip, id\_bus, id\_bus\_route, departure\_time, duration, booked\_seat, status)
3. TRIP\_DRIVER (id\_trip, id\_driver)
4. AGENT (id\_agent, id\_place, id\_cash\_reserve, address, name)
5. CASHRESERVE (id\_cash\_reserve, counter)
6. BUSSTATION (id\_bus\_station, id\_place, name, bus\_capacity, count\_current\_bus)
7. DRIVER (id\_driver, license\_level, type, state)
8. BUSROUTE (id\_route, id\_start\_station, id\_end\_station, distance)
9. PLACE (id\_place, region)
10. EMPLOYEE (id\_employee, id\_account, id\_agent, name, address, phone\_number, identity\_number, salary, email, birthdate, state)
11. POSITION (id\_position, type)
12. PRIVILEGE (id\_privilege, name)
13. SYSTEMACCOUNT (id\_account, username, password)
14. TICKET (id\_ticket, id\_trip, status, fare, type, seat\_number)
15. EVENT (id\_event, discount\_type, discount\_percent)
16. REFUND (id\_refund, refund\_name, refund\_percent)
17. PASSENGER (id\_passenger, name, phone\_number, address, identity\_number, gender, email)
18. BOOKING (id\_booking, id\_ticket, id\_passenger, id\_employee, booking\_time)
19. PACKAGE (id\_package, id\_trip, mass, price, sender\_contact\_phone, receiver\_contact\_phone)
20. PACKAGEPRICEPOLICY (id\_policy, price\_per\_km, mass\_unit)
21. AGENT\_TRIP (id\_agent, id\_trip)
22. AGENT\_EVENT (id\_agent, id\_event)
23. AGENT\_REFUND (id\_agent, id\_refund)
24. AGENT\_POLICY (id\_agent, id\_policy)
25. BUSROUTE\_BUSSTATION (id\_bus\_route, id\_bus\_station)
26. EMPLOYEE\_POSITION (id\_employee, id\_position)
27. EMPLOYEE\_TICKET (id\_employee, id\_ticket)
28. POSITION\_PRIVILEGE (id\_position, id\_privilege)

## Required constraints

|  |  |  |
| --- | --- | --- |
| **No.** | **Table** | **Constraint** |
| **1** | BUS | Primary key:  id\_bus  Check:  CHK\_bus\_capacity |
| **2** | TRIP | Primary key:  id\_trip  Foreign keys:  FK\_trip\_id\_bus, FK\_trip\_id\_bus\_route  Check:  CK\_trip |
| **3** | TRIP\_DRIVER | Foreign keys:  FK\_trip\_driver\_id\_trip, FK\_trip\_\_driver\_id\_driver |
| **4** | DRIVER | Primary key:  id\_driver |
| **5** | AGENT | Primary key:  id\_agent  Foreign key:  FK\_agent\_id\_cash\_reserve, FK\_agent\_id\_place |
| **6** | CASHRESERVE | Primary key:  id\_cash\_reserve |
| **7** | BUSSTATION | Primary key:  id\_bus\_station  Foreign keys:  FK\_busstation\_id\_place  Check:  CHK\_busstation |
| **8** | BUSROUTE | Primary key: id\_route  Foreign keys:  FK\_busroute\_id\_start\_bus\_station, FK\_busroute\_id\_end\_bus\_station  Check:  CHK\_busroute |
| **9** | PLACE | Primary key:  id\_place  Check:  CHK\_place |
| **10** | EMPLOYEE | Primary key:  id\_employee  Foreign keys:  FK\_employee\_id\_account, FK\_employee\_id\_agent  Check:  CHK\_employee |
| **11** | POSITION | Primary key:  id\_position  Check:  CHK\_position |
| **12** | PRIVILEGE | Primary key:  id\_privilege  Check:  CHK\_privilege |
| **13** | SYSTEMACCOUNT | Primary key:  id\_account |
| **14** | TICKET | Primary key:  id\_ticket  Foreign key:  FK\_trip\_id\_trip  Check:  CHK\_ticket |
| **15** | EVENT | Check:  CHK\_event |
| **16** | REFUND | Check:  CHK\_refund |
| **17** | PASSENGER | Primary key:  id\_passenger  Check:  CHK\_passenger |
| **18** | BOOKING | Primary key:  id\_booking  Foreign keys:  FK\_booking\_id\_ticket, FK\_booking\_id\_passenger, FK\_booking\_id\_employee |
| **19** | PACKAGE | Primary key:  id\_package  Foreign keys:  FK\_package\_id\_trip  Check:  CHK\_package |
| **20** | PACKAGEPRICEPOLICY | Primary key:  id\_policy  Check:  CHK\_packagepricepolicy |
| **21** | AGENT\_TRIP | Primary key:  id\_agent, id\_trip  Foreign key:  FK\_agent\_trip\_id\_agent, FK\_agent\_trip\_id\_trip |
| **22** | AGENT\_EVENT | Primary key:  id\_agent, id\_event  Foreign key:  FK\_agent\_event\_id\_agent, FK\_agent\_event\_id\_event |
| **23** | AGENT\_REFUND | Primary key:  id\_agent, id\_refund  Foreign key:  FK\_agent\_refund\_id\_agent, FK\_agent\_refund\_id\_refund |
| **24** | AGENT\_POLICY | Primary key:  id\_agent, id\_policy  Foreign key:  FK\_agent\_policy\_id\_agent, FK\_agent\_policy\_id\_policy |
| **25** | BUSROUTE\_BUSSTATION | Primary key:  id\_bus\_route, id\_bus\_station  Foreign key:  FK\_busroute\_busstation\_id\_bus\_route, FK\_busroute\_busstation\_id\_bus\_station |
| **26** | EMPLOYEE\_POSITION | Primary key:  id\_employee, id\_postion  Foreign key:  FK\_employee\_position\_id\_employee, FK\_employee\_position\_id\_position |
| **27** | EMPLOYEE\_TICKET | Primary key:  id\_employee, id\_ticket  Foreign key:  FK\_employee\_ticket\_id\_employee, FK\_employee\_ticket\_id\_ticket |
| **28** | POSITION\_PRIVILEGE | Primary key:  id\_position, id\_privilege  Foreign key:  FK\_position\_privilege\_id\_position, FK\_position\_privilege\_id\_privilege |

## Database and constraints implementation

[BUS]

create table BUS

(

id\_bus varchar(20) primary key,

registration\_number char(15) unique not null,

model varchar(50) not null,

capacity tinyint default 32,

status char(15) not null default 'idle',

type bit not null default 0

-- 0: interprovince, 1: transit

);

[TRIP]

create table TRIP

(

id\_trip varchar(20) primary key,

id\_bus varchar(20) ,

id\_bus\_route varchar(20),

departure\_time datetime not null,

duration int not null,

-- unit: hour,

booked\_seat tinyint default 0,

status char(15) default 'waiting'

);

[TRIP\_DRIVER]

create table TRIP\_DRIVER

(

id\_trip varchar(20),

id\_driver varchar(20),

primary key(id\_trip, id\_driver)

);

[AGENT]

create table AGENT

(

id\_agent varchar(20) primary key,

id\_cash\_reserve varchar(20),

id\_place varchar(20),

name varchar(50) not null,

address char(100) not null

);

[CASHRESERVE]

create table CASHRESERVE

(

id\_cash\_reserve varchar(20) primary key,

counter money default 0,

);

[BUSSTATION]

create table BUSSTATION

(

id\_bus\_station varchar(20) primary key,

id\_place varchar(20),

name varchar(50) not null,

address char(100) not null,

bus\_capacity int not null,

count\_current\_bus int null default 0,

);

[DRIVER]

create table DRIVER

(

id\_driver varchar(20) primary key,

-- foreign key

lisence\_level char(10) not null,

type bit default 0,

-- 0: interprovince, 1: transit

state bit default 0

-- 0: not drive, 1: is driving

);

[BUSROUTE]

create table BUSROUTE

(

id\_route varchar(20) primary key,

id\_start\_station varchar(20),

id\_end\_station varchar(20),

distance int not null

-- unit: km

);

[PLACE]

create table PLACE

(

id\_place varchar(20) primary key,

region char(50) default 'TP.Ho Chi Minh'

);

[EMPLOYEE]

create table EMPLOYEE

(

id\_employee varchar(20) primary key,

id\_account varchar(20),

id\_agent varchar(20),

name varchar(50) not null,

address char(100) not null,

phone\_number varchar(20) not null,

identity\_number char(20) not null,

salary money not null,

email char(50) null,

birthdate date not null,

state bit default 1

-- 0: not working, 1: is working

);

[POSITION]

create table POSITION

(

id\_position varchar(20) primary key,

type varchar(50) not null

);

[PRIVILEGE]

create table PRIVILEGE

(

id\_privilege varchar(20) primary key,

name char(50)

);

[SYSTEMACCOUNT]

create table SYSTEMACCOUNT

(

id\_account varchar(20) primary key,

username varchar(20) not null unique,

pass varchar(50) not null,

);

[TICKET]

create table TICKET

(

id\_ticket varchar(20) primary key,

id\_trip varchar(20) ,

status bit default 0,

fare money not null,

type bit default 0,

-- 0: seat, 1: lie

seat\_number char(15) not null unique

);

[EVENT]

create table EVENT

(

id\_event varchar(20) primary key,

discount\_type char(50) not null unique default 'normal',

discount\_percent float default 0.0

);

[REFUND]

create table REFUND

(

id\_refund varchar(20) primary key,

refund\_name char(50) not null unique default 'cancel',

refund\_percent float default 0.0

);

[PASSENGER]

create table PASSENGER

(

id\_passenger varchar(20) primary key,

name varchar(50) not null,

phone\_number varchar(20) not null,

address char(100) not null,

identity\_number char(20) null,

gender bit default 0,

-- 0: male, 1: female

email char(50) null,

);

[BOOKING]

create table BOOKING

(

id\_booking varchar(20) primary key,

id\_ticket varchar(20),

id\_passenger varchar(20),

id\_employee varchar(20),

booking\_time datetime default getdate(),

);

[PACKAGE]

create table PACKAGE

(

id\_package varchar(20) primary key,

id\_trip varchar(20),

mass smallint default 0,

price money ,

-- is calculated by the formula

sender\_contact\_phone char(20) not null,

receiver\_contact\_phone char(20) not null

);

[PACKAGEPRICEPOLICY]

create table PACKAGEPRICEPOLICY

(

id\_policy varchar(20) primary key,

price\_per\_km money not null,

mass\_unit int not null

-- /5kg, /1kg

);

[AGENT\_TRIP]

create table AGENT\_TRIP

(

id\_agent varchar(20),

id\_trip varchar(20),

primary key(id\_agent, id\_trip)

);

[AGENT\_EVENT]

create table AGENT\_EVENT

(

id\_agent varchar(20),

id\_event varchar(20),

primary key(id\_agent, id\_event)

);

[AGENT\_REFUND]

create table AGENT\_REFUND

(

id\_agent varchar(20),

id\_refund varchar(20),

primary key(id\_agent, id\_refund)

);

[AGENT\_POLICY]

create table AGENT\_POLICY

(

id\_agent varchar(20),

id\_policy varchar(20),

primary key(id\_agent, id\_policy)

);

[BUSROUTE\_BUSSTATION]

create table BUSROUTE\_BUSSTATION

(

id\_bus\_route varchar(20),

id\_bus\_station varchar(20),

primary key(id\_bus\_route, id\_bus\_station)

);

[EMPLOYEE\_POSITION]

create table EMPLOYEE\_POSITION

(

id\_employee varchar(20),

id\_position varchar(20),

primary key(id\_employee, id\_position)

);

[EMPLOYEE\_TICKET]

create table EMPLOYEE\_TICKET

(

id\_employee varchar(20),

id\_ticket varchar(20),

primary key(id\_employee, id\_ticket)

);

[POSITION\_PRIVILEGE]

create table POSITION\_PRIVILEGE

(

id\_position varchar(20),

id\_privilege varchar(20),

primary key(id\_position, id\_privilege)

);

## Other constraints

**\* Constrain bus ID** **after add one**

-- Set constraint bus identity automatically.

USE BusManagement

ALTER TABLE Bus

ADD CONSTRAINT AUTO\_ID\_Bus

DEFAULT DBO.AUTO\_ID\_Bus() FOR ID\_bus;

GO

**\* Constrain passenger ID and ticket ID** **after customer buy a ticket**

-- Set constraint ticket identity automatically.

USE BusManagement

ALTER TABLE TICKET

ADD CONSTRAINT AUTO\_ID\_ticket

DEFAULT DBO.AUTO\_ID\_ticket() FOR ID\_ticket;

GO

-- Set constraint passenger identity automatically.

USE BusManagement

ALTER TABLE PASSENGER

ADD CONSTRAINT AUTO\_ID\_passenger

DEFAULT DBO.AUTO\_ID\_passenger() FOR ID\_passenger;

GO

**\* Constrain trip ID** **after add one**

-- Set constraint trip identity automatically.

USE BusManagement

ALTER TABLE TRIP

ADD CONSTRAINT AUTO\_ID\_trip

DEFAULT DBO.AUTO\_ID\_trip() FOR ID\_trip;

GO

**\* Constrain route ID** **after add one**

-- Set constraint route identity automatically.

USE BusManagement

ALTER TABLE BUS\_ROUTE

ADD CONSTRAINT AUTO\_ID\_route

DEFAULT DBO.AUTO\_ID\_route() FOR ID\_route;

GO

**\* Constrain position ID** **after add one**

-- Set constraint route identity automatically.

USE BusManagement

ALTER TABLE POSITION

ADD CONSTRAINT AUTO\_ID\_position

DEFAULT DBO.AUTO\_ID\_position() FOR ID\_position;

GO

**\* Constrain employee ID** **after add one**

-- Set constraint route identity automatically.

USE BusManagement

ALTER TABLE EMPLOYEE

ADD CONSTRAINT AUTO\_ID\_employee

DEFAULT DBO.AUTO\_ID\_employee() FOR ID\_employee;

GO

**Constrain agent ID** **after add one**

-- Set constraint bus identity automatically.

USE BusManagement

ALTER TABLE Agent

ADD CONSTRAINT AUTO\_ID\_Agent

DEFAULT DBO.AUTO\_ID\_Agent() FOR ID\_agent;

GO

**\* Constrain booking ID** **after add one**

-- Set constraint bus identity automatically.

USE BusManagement

ALTER TABLE Booking

ADD CONSTRAINT AUTO\_ID\_Booking

DEFAULT DBO.AUTO\_ID\_Booking() FOR ID\_booking;

GO

**\* Constrain package ID** **after add one**

-- Set constraint bus identity automatically.

USE BusManagement

ALTER TABLE Package

ADD CONSTRAINT AUTO\_ID\_Package

DEFAULT DBO.AUTO\_ID\_Package() FOR ID\_package;

GO

**\* Constrain package price policy ID** **after add one**

-- Set constraint bus identity automatically.

USE BusManagement

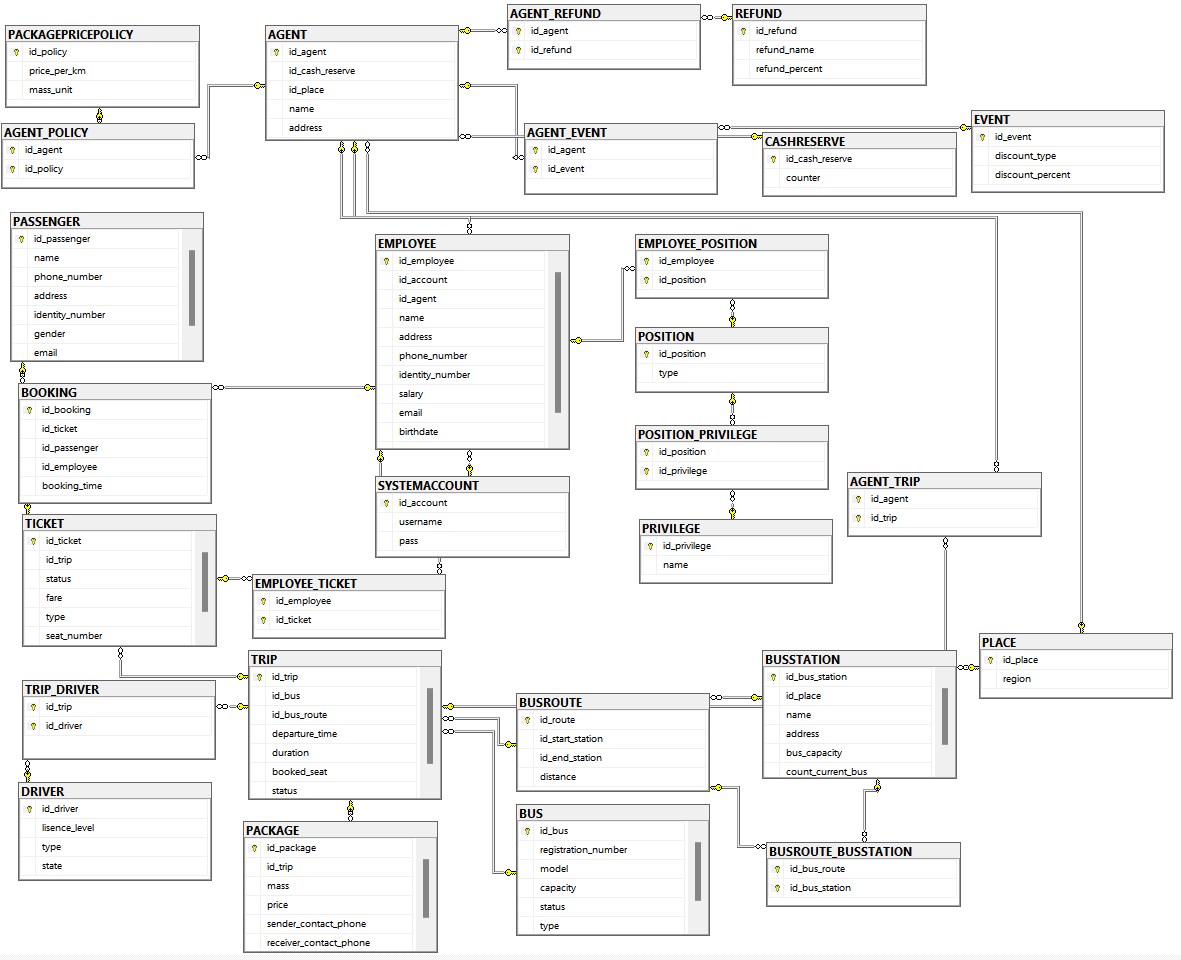
ALTER TABLE PackagePricePolicy

ADD CONSTRAINT AUTO\_ID\_PackagePricePolicy

DEFAULT DBO.AUTO\_ID\_PackagePricePolicy() FOR ID\_policy;

GO

After setting up the necessary tables and relations along with their respective constraints and triggers, a Physical level diagram will be created:



## Constraint-checking triggers

**\* Update state of an employee on Employee and Driver relation**

CREATE TRIGGER tr\_employee\_update\_stateEmployee

ON Employee

AFTER UPDATE

AS

BEGIN

DECLARE @employee\_id CHAR

DECLARE @new\_state CHAR

SELECT @employee\_id = ID\_employee, @new\_state = state FROM inserted

-- Kiểm tra nếu trạng thái mới của nhân viên là 0

IF @new\_state = 0

BEGIN

-- Cập nhật trạng thái của nhân viên trong bảng Employee thành 0

UPDATE Employee SET state = 0 WHERE ID\_employee = @employee\_id

UPDATE Driver SET state = 0 WHERE ID\_driver = @employee\_id

END

SELECT state FROM Driver WHERE ID\_driver = @employee\_id

END

**\* Delete employee account and update state of inactive Employee (state = 0)**

CREATE TRIGGER tr\_employee\_deleteAccount

ON Employee

AFTER UPDATE

AS

BEGIN

DECLARE @account\_id CHAR

DECLARE @new\_state CHAR

SELECT @account\_id = ID\_account, @new\_state = inserted.state FROM inserted

IF @new\_state = 0

BEGIN

-- Delete account

DELETE FROM SystemAccount WHERE @account\_id = ID\_account

END

END

## Database views implementation

**\* View for list of active employee which is working (state = 1)**

CREATE VIEW [dbo].[ActiveEmployee] AS

SELECT Employee.ID\_employee, Employee\_ID\_account, Employee.name, Employee.address, Employee.phone\_number, Employee.identity\_number, Employee.salary, Employee.email,

Employee.birthday, Agent.name, Position.type

FROM

Employee AS temp1 INNER JOIN Agent AS temp 2

ON temp1.ID\_agent = temp2.ID\_agent

INNER JOIN Position as temp3

ON temp1.ID\_position = temp3.ID\_position

WHERE temp1.status = 1

**\* View for list of waiting trip:**

CREATE VIEW [dbo].[WaitingTrip] AS

temp1.ID\_trip, temp1.departure\_time, temp1.duration, temp1.booked\_seat, temp1.registration\_number, temp1.type,

temp2.name AS start\_point, temp3.name AS end\_point

FROM

((SELECT Trip.\*, Bus.registration\_number, Bus.type

FROM Trip INNER JOIN Bus

ON Trip.ID\_bus = Bus.ID\_bus) AS temp0

INNER JOIN BusRoute

ON BusRoute.ID\_route = temp0.ID\_route) AS temp1

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station1, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station1 = BusStation.ID\_bus\_station

) AS temp2

ON temp1.ID\_route = temp2.ID\_route

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station2, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station2 = BusStation.ID\_bus\_station

) AS temp3

ON temp1.ID\_route = temp3.ID\_route

WHERE temp1.status = 'Waiting'

**\* View for of going trip:**

CREATE VIEW [dbo].[GoingTrip] AS

temp1.ID\_trip, temp1.departure\_time, temp1.duration, temp1.booked\_seat, temp1.registration\_number, temp1.type,

temp2.name AS start\_point, temp3.name AS end\_point

FROM

((SELECT Trip.\*, Bus.registration\_number, Bus.type

FROM Trip INNER JOIN Bus

ON Trip.ID\_bus = Bus.ID\_bus) AS temp0

INNER JOIN BusRoute

ON BusRoute.ID\_route = temp0.ID\_route) AS temp1

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station1, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station1 = BusStation.ID\_bus\_station

) AS temp2

ON temp1.ID\_route = temp2.ID\_route

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station2, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station2 = BusStation.ID\_bus\_station

) AS temp3

ON temp1.ID\_route = temp3.ID\_route

WHERE temp1.status = ‘Going’

**\* View for list of finished trip:**

CREATE VIEW [dbo].[FinishTrip] AS

temp1.ID\_trip, temp1.departure\_time, temp1.duration, temp1.booked\_seat, temp1.registration\_number, temp1.type,

temp2.name AS start\_point, temp3.name AS end\_point

FROM

((SELECT Trip.\*, Bus.registration\_number, Bus.type

FROM Trip INNER JOIN Bus

ON Trip.ID\_bus = Bus.ID\_bus) AS temp0

INNER JOIN BusRoute

ON BusRoute.ID\_route = temp0.ID\_route) AS temp1

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station1, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station1 = BusStation.ID\_bus\_station

) AS temp2

ON temp1.ID\_route = temp2.ID\_route

INNER JOIN (

SELECT temp1.ID\_route, temp1.ID\_bus\_station2, BusStation.name

FROM temp1 INNER JOIN BusStation

ON temp1.ID\_bus\_station2 = BusStation.ID\_bus\_station

) AS temp3

ON temp1.ID\_route = temp3.ID\_route

WHERE temp1.status = ‘Finish'

**\* View for list of idle interprovince bus:**

CREATE VIEW [dbo].[IdleInterprovinceBus]

AS

SELECT rel.ID\_bus, rel.registration\_number, rel.model, rel.capacity

FROM Bus as rel

WHERE Bus.status = 'idle' AND Bus.type = 'interprovince'

**\* View for list of break interprovince bus:**

CREATE VIEW [dbo].[BreakInterprovinceBus]

AS

SELECT rel.ID\_bus, rel.registration\_number, rel.model, rel.capacity

FROM Bus as rel

WHERE Bus.status = 'break' AND Bus.type = 'interprovince'

**\* View for list of incident interprovince bus:**

CREATE VIEW [dbo].[IncidentInterprovinceBus]

AS

SELECT rel.ID\_bus, rel.registration\_number, rel.model, rel.capacity

FROM Bus as rel

WHERE Bus.status = 'incident' AND Bus.type = 'interprovince'

**\* View for list of ongoing interprovince bus:**

CREATE VIEW [dbo].[OnGoingInterprovinceBus]

AS

SELECT rel.ID\_bus, rel.registration\_number, rel.model, rel.capacity

FROM Bus AS rel INNER JOIN (

Select Trip.ID\_bus, Trip.ID\_trip

FROM Trip

WHERE Trip.status = 'going'

) as rel2

ON rel.ID\_bus = rel2.ID\_bus

WHERE Bus.status = 'ongoing' AND Bus.type = 'interprovince'

**\* View for (detailed) list of passenger booking information:**

CREATE VIEW [dbo].[BookingInfor]

AS

SELECT rel.ID\_booking, temp1.ID\_ticket, temp1.ID\_trip, temp1.seat\_number, temp1.type, temp1.fare, temp2.name AS passenger\_name, temp2.phone\_number AS passenger\_phone\_number, temp2.address AS passenger\_address, temp2.email AS passenger\_email, temp2.gender AS passenger\_gender, temp3.name AS employee\_name, temp3.phone\_number AS employee\_phonee\_number

FROM Booking AS rel INNER JOIN Ticket AS temp1

ON rel.ID\_ticket = temp1.ID\_ticket

INNER JOIN Passenger AS temp2

ON rel.ID\_passenger = temp2.ID\_passenger

INNER JOIN Employee AS temp3

ON rel.ID\_employee = temp3.ID\_employee

**\* View for (detailed) list of current bus route information:**

CREATE VIEW [dbo].[BusRouteInfor]

AS

SELECT rel.ID\_route, temp1.start\_point, temp2.end\_point, rel.distance

FROM BusRoute AS rel INNER JOIN (

SELECT rel.ID\_Route, BusStation.name as start\_point

FROM rel INNER JOIN BusStation

ON rel.ID\_bus\_station1 = BusStation.ID\_bus\_station

) AS temp1

ON rel.ID\_route = temp1.ID\_route

INNER JOIN (

SELECT rel.ID\_Route, BusStation.name as end\_point

FROM rel INNER JOIN BusStation

ON rel.ID\_bus\_station2 = BusStation.ID\_bus\_station

) AS temp2

ON rel.ID\_route = temp2.ID\_route

**\* View for list of Going trip driver:**

CREATE VIEW [dbo].[WaitingTripDriverInfor]

AS

SELECT rel.\*, temp2.ID\_driver, temp2.name AS driver\_name, temp2.phone\_number AS driver\_phone\_number

FROM WaitingTrip AS rel INNER JOIN TripDriver AS temp1

ON rel.ID\_trip = temp1.ID\_trip

INNER JOIN (

SELECT Driver.ID\_driver, Employee.name, Employee.phone\_number

FROM Driver INNER JOIN Employee

ON Driver.ID\_driver = Employee.ID\_employee

) AS temp2

ON temp1.ID\_driver = temp2.ID\_driver

**\* View for list of Going trip driver:**

CREATE VIEW [dbo].[GoingTripDriverInfor]

AS

SELECT rel.\*, temp2.ID\_driver, temp2.name AS driver\_name, temp2.phone\_number AS driver\_phone\_number

FROM GoingTrip AS rel INNER JOIN TripDriver AS temp1

ON rel.ID\_trip = temp1.ID\_trip

INNER JOIN (

SELECT Driver.ID\_driver, Employee.name, Employee.phone\_number

FROM Driver INNER JOIN Employee

ON Driver.ID\_driver = Employee.ID\_employee

) AS temp2

ON temp1.ID\_driver = temp2.ID\_driver

**\* View for list of Going trip driver:**

CREATE VIEW [dbo].[FinishTripDriverInfor]

AS

SELECT rel.\*, temp2.ID\_driver, temp2.name AS driver\_name, temp2.phone\_number AS driver\_phone\_number

FROM FinishTrip AS rel INNER JOIN TripDriver AS temp1

ON rel.ID\_trip = temp1.ID\_trip

INNER JOIN (

SELECT Driver.ID\_driver, Employee.name, Employee.phone\_number

FROM Driver INNER JOIN Employee

ON Driver.ID\_driver = Employee.ID\_employee

) AS temp2

ON temp1.ID\_driver = temp2.ID\_driver

**\* View for list of employee accounts:**

CREATE VIEW [dbo].[EmployeeAccount]

AS

SELECT temp1.ID\_employee, temp1.name, temp2.username, temp2.password

FROM Employee AS temp1, SystemAccount AS temp2

**\* View for list of waiting trip which still has empty seats:**

CREATE VIEW [dbo].[TripWithAvailableChair]

AS

SELECT temp1.\*, temp2.capacity - temp1.booked\_seat AS available\_position

FROM WaitingTrip AS temp1 INNER JOIN (

SELECT Bus.capacity, Trip.ID\_trip

FROM Trip INNER JOIN Bus

ON Trip.ID\_bus = Bus.ID\_bus

) as temp2

ON temp1.ID\_trip temp2.ID\_trip

WHERE temp2.capacity - temp1.booked\_seat > 0

**\* View for list of waiting trip which has full seats:**

CREATE VIEW [dbo].[TripWithAvailableChair]

AS

SELECT temp1.\*, temp2.capacity - temp1.booked\_seat AS available\_position

FROM WaitingTrip AS temp1 INNER JOIN (

SELECT Bus.capacity, Trip.ID\_trip

FROM Trip INNER JOIN Bus

ON Trip.ID\_bus = Bus.ID\_bus

) as temp2

ON temp1.ID\_trip temp2.ID\_trip

WHERE temp2.capacity - temp1.booked\_seat = 0

**\* View for sum of all agents’ cash reserve:**

CREATE VIEW [dbo].[V\_TotalCashReserve]

AS

SELECT SUM(counter) AS sum\_counter

FROM dbo.CASHRESERVE

# CHAPTER III: FUNCTION AND PROCEDURE DESIGN

## Connect to database

This project uses Entity Framework in order to connect to the BusManagement database, hence the Connection string is stored in App.config instead of a variable in the application, but this feature is still part of the .NET framework that the project is based on.

The Entity Framework library maps the database into the C# Project so we will have a dbConnection file that helps us perform operations (functions and procedures) and get database entities (views, relations,…) without having to directly use SQL commands.

A screenshot of a computer

Description automatically generated with medium confidence

Connection string (For Server Admins):

<connectionStrings>

<add name="BusTicketManagementApplication.Properties.Settings.BusManagementConnectionString" connectionString="Data Source=.;Initial Catalog=BusManagement;Integrated Security=True" providerName="System.Data.SqlClient" /><add name="BusManagementEntities" connectionString="metadata=res://\*/src.dbConnection.BusManagementModel.csdl|res://\*/src.dbConnection.BusManagementModel.ssdl|res://\*/src.dbConnection.BusManagementModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local);initial catalog=BusManagement;persist security info=True;user id=admin01;password=admin;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

Model tree of BusManagementModel.edmx:

Graphical user interface

Description automatically generatedA picture containing text

Description automatically generated

Text

Description automatically generated with medium confidenceText

Description automatically generated

## Function design

**NOTE**: All functions with the name [func\_auto\_...] have the same functionality and call syntax in the program so we will display only ONE example.

A screenshot of a computer

Description automatically generated with low confidence

All SQL functions are passed in this function - using only their names - in order to run in the program.

public static string RunFunc(string funcName)

{

BusManagementEntities db = new BusManagementEntities();

string query = $"select dbo.{funcName}()";

return db.Database.SqlQuery<string>(query).ToList().FirstOrDefault().ToString();

}

**\* Example for [func\_auto\_...] to automatically create a default passenger ID:**

CREATE function [dbo].[func\_auto\_id\_passenger]()

returns char(20)

as

begin

declare @id\_no char(20)

set @id\_no = (

select max(id\_passenger)

from PASSENGER

)

if( @id\_no is null)

set @id\_no = concat('pas\_', '0000000000')

declare @no int

set @no = right(@id\_no, 10) + 1;

return concat('pas\_', format(@no, '0000000000'))

end

public string GetNewPassengerId()

{

BusManagementEntities db = new BusManagementEntities();

string funcName = "func\_auto\_id\_passenger";

return BSMain.RunFunc(funcName);

}

**\* Function to add a new passenger:**

create function func\_AddPassenger(@name nvarchar(50), @phone char(20))

returns char(20)

as

begin

declare @id\_passenger char(20)

set @id\_passenger = dbo.func\_auto\_id\_passenger();

exec dbo.pro\_AddPassenger @id\_passenger, @name, @phone;

return @id\_passenger

end

public void AddPassenger(string name, string phone)

{

BusManagementEntities db = new BusManagementEntities();

string idPassenger = BSMain.RunTableValuedFunc("func\_AddPassenger", new List<string> { name, phone }).FirstOrDefault();

}

**\* Function to get available seats in a trip:**

create function func\_GetAvailableSeat(@idTrip char(20), @type bit)

returns table

as

return (select TICKET.seat\_number from TICKET where TICKET.id\_trip = @idTrip and TICKET.type = @type and TICKET.status = 0)

public List<string> GetAvailableSeat(string idTrip, int type) // type 0: seat, 1: sleeper

{

BusManagementEntities db = new BusManagementEntities();

string funcName = "func\_GetAvailabelSeat";

List<string> ticketList = BSMain.RunTableValuedFunc(funcName, new List<string> { idTrip, type.ToString() });

return ticketList;

}

## Procedure design

**\* Procedure for cancelling a ticket:**

create proc pro\_CancelTicket @id\_ticket char(20)

as

begin

update TICKET set status = 0 where id\_ticket = @id\_ticket

end

private void BtnCancel\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(this.LbSelectedId.Text.Trim()))

{

MessageBox.Show("Please select the ticket to cancel!");

return;

}

BusManagementEntities db = new BusManagementEntities();

//

db.pro\_CancelTicket(this.LbSelectedId.Text.Trim());

this.LbSelectedId.Text = string.Empty;

MessageBox.Show("Cancel successfully!");

FilterBookedTickets();

}

**\* Procedure for setting a trip status as ‘cancel’:**

create proc pro\_SetCancelTrip @id\_trip char(20)

as

begin

update TRIP set status = 'cancel' where id\_trip = @id\_trip

end

public void SetCancelTrip(string tripId)

{

BusManagementEntities db = new BusManagementEntities();

db.pro\_SetCancelTrip(tripId);

}

**\* Procedure for setting a trip status as ‘finish’:**

create proc pro\_SetFinishTrip @id\_trip char(20)

as

begin

update TRIP set status = 'finish' where id\_trip = @id\_trip

end

public void SetFinish(string tripId)

{

BusManagementEntities db = new BusManagementEntities();

db.pro\_SetFinishTrip(tripId);

}

**\* Procedure for setting a trip status as ‘going’:**

create proc pro\_SetGoingTrip @id\_trip char(20)

as

begin

update TRIP set status = 'going' where id\_trip = @id\_trip

end

public void SetGoing(string tripId)

{

BusManagementEntities db = new BusManagementEntities();

db.pro\_SetGoingTrip(tripId);

}

**\* Procedure for adding default ticket information while booking:**

create proc pro\_AddDefaultBooking @id\_ticket char(20), @id\_passenger char(20)

as

begin

update TICKET set TICKET.status = 1 where TICKET.id\_ticket = @id\_ticket;

insert into BOOKING(id\_ticket, id\_passenger, id\_employee, booking\_time) values (@id\_ticket, @id\_passenger, dbo.func\_AutoDefaultIdEmployee(), GETDATE());

end

BusManagementEntities db = new BusManagementEntities();

//

db.pro\_AddDefaultBooking(this.TbIDTicket.Text.Trim(), UserData.GetPassengerId());

// payment logic ( directly, online )

**\* Procedure for booking a ticket for a passenger:**

create proc pro\_AddBooking @id\_ticket char(20), @id\_passenger char(20), @id\_employee char(20)

as

begin

update TICKET set TICKET.status = 1 where TICKET.id\_ticket = @id\_ticket;

insert into BOOKING(id\_ticket, id\_passenger, id\_employee, booking\_time) values (@id\_ticket, @id\_passenger, @id\_employee, GETDATE());

end

public void AddBooking(string ticketId, string passengerId, string employeeId)

{

BusManagementEntities db = new BusManagementEntities();

db.pro\_AddBooking(ticketId, passengerId, employeeId);

}

**\* Procedure for adding a passenger:**

create proc pro\_AddPassenger @id\_passenger char(20), @name nvarchar(50), @phone char(20)

as

begin

insert into PASSENGER(id\_passenger, name, phone\_number) values (@id\_passenger, @name, @phone);

end

string funcName = "func\_auto\_id\_passenger";

passengerId = BSMain.RunFunc(funcName);

if (!string.IsNullOrEmpty(passengerId))

{

db.pro\_AddPassenger(passengerId, name, phone);

db.pro\_AddPassengerAccount(passengerId, username, password); // add passenger account and assign privilege

errMsg = "Create new user successfully!. No error";

}

else

{

errMsg = "Can't get new passengerId";

}

**\* Simulate [C#]: Try/Catch SQL exception in ValidateUser function (Check whether the user information is in the database or not)**

The function will pass an error variable into the function to get the exception message. When an exception occurs, the function will automatically assign the message to the error variable then the program will show it to the user.

create PROCEDURE pro\_CheckUniqueUser(@username varchar(50))

AS

BEGIN

DECLARE @count int, @errMsg nvarchar(MAX)

SET @count = 0

SELECT @count = COUNT(\*) FROM PASSENGERACCOUNT WHERE PASSENGERACCOUNT.username = @username

IF @count > 0

BEGIN

SET @errMsg = 'Username has already been taken.';

RAISERROR(@errMsg, 16, 1);

END

SELECT @count

END

public bool CreateNewUser(string username, string password,string name, string phone,ref string passengerId, ref string errMsg)

{

passengerId = string.Empty;

try

{

foreach (char c in username)

{

if (c < 48 || (c > 57 && c < 65) || (c > 90 && c < 97) || c > 122)

{

errMsg = "Username is invalid. Username must be letters in alphabet and digits";

return false;

}

}

BusManagementEntities db = new BusManagementEntities();

// way 1

//bool uniqueUser = db.PASSENGERACCOUNTs.Count(d => d.username == username) == 0;

//if(!uniqueUser){

// errMsg = "Username has exist in the system!";

// return false;

//}

// way 2: check whether unique username

db.Database.SqlQuery<int>("EXEC pro\_CheckUniqueUser @username", new SqlParameter("@username", username)); //if user exist,throw an sql exception

// incase of unique username

string funcName = "func\_auto\_id\_passenger";

passengerId = BSMain.RunFunc(funcName);

if (!string.IsNullOrEmpty(passengerId))

{

db.pro\_AddPassenger(passengerId, name, phone);

db.pro\_AddPassengerAccount(passengerId, username, password); // add passenger account and assign privilege

errMsg = "Create new user successfully!. No error";

}

else

{

errMsg = "Can't get new passengerId";

}

}

catch (SqlException err)

{

errMsg = err.Message;

MessageBox.Show(err.Message.ToString());

return false;

}

catch (Exception ex)

{

MessageBox.Show("An error occurred: " + ex.Message);

return false;

}

return true;

}

**\* Procedure for changing user passwords**

CREATE PROC [dbo].[pro\_ChangeSystemPassword] (@username varchar(50), @new\_password varchar(50))

AS

BEGIN

SET XACT\_ABORT ON;

BEGIN TRAN;

BEGIN TRY

UPDATE SYSTEMACCOUNT SET pass = @new\_password WHERE username = @username;

DECLARE @query nvarchar(MAX);

SET @query = 'ALTER LOGIN ' + QUOTENAME(@username) + ' WITH PASSWORD = ''' + @new\_password + ''';';

EXEC (@query);

COMMIT TRAN;

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH;

END;

CREATE PROC [dbo].[pro\_ChangePassengerPassword] (@username varchar(50), @new\_password varchar(50))

AS

BEGIN

SET XACT\_ABORT ON;

BEGIN TRAN;

BEGIN TRY

UPDATE PASSENGERACCOUNT SET password = @new\_password WHERE username = @username;

DECLARE @query nvarchar(MAX);

SET @query = 'ALTER LOGIN ' + QUOTENAME(@username) + ' WITH PASSWORD = ''' + @new\_password + ''';';

EXEC (@query);

COMMIT TRAN;

END TRY

BEGIN CATCH

ROLLBACK TRAN;

THROW;

END CATCH;

END;

public bool ChangeUserPassword(string username, string newPassword)

{

bool res = true;

try

{

BusManagementEntities db = new BusManagementEntities(StaticEnv.GetDefaultEFConnectionString());

if (UserData.IsPassenger)

{

db.pro\_ChangePassengerPassword(username, newPassword);

}

else

{

db.pro\_ChangeSystemPassword(username, newPassword);

}

}

catch (SqlException err)

{

MessageBox.Show(err.Message);

return false;

}

catch (Exception err)

{

MessageBox.Show(err.Message);

return false;

}

return res;

}

# CHAPTER IV: USER CREATION AND PRIVILEGE DISTRIBUTION

**Applicable towards users (Passengers, Staff and Server Admin)**

**\* For passengers**

exec sp\_addrole rol\_Passenger;

-- grant tables

grant select on AGENT to rol\_Passenger;

grant select, insert, delete, references on BOOKING to rol\_Passenger;

grant select on BUS to rol\_Passenger;

grant select on BUSROUTE to rol\_Passenger;

grant select on BUSSTATION to rol\_Passenger;

grant select, insert, update, references on PASSENGER to rol\_Passenger;

grant select, insert, update, references on PASSENGERACCOUNT to rol\_Passenger;

grant select on PLACE to rol\_Passenger;

grant select on TICKET to rol\_Passenger;

grant select on TRIP to rol\_Passenger;

--

--grant views

grant select on V\_AGENTINFOR to rol\_Passenger;

grant select on V\_AVAILABLETRIP to rol\_Passenger;

grant select on V\_BOOKEDTICKET to rol\_Passenger;

grant select on V\_BOOKINGINFOR to rol\_Passenger;

grant select on V\_BUSSTATIONINFOR to rol\_Passenger;

grant select on V\_ROUTEINFOR to rol\_Passenger;

grant select on V\_TRIPINFOR to rol\_Passenger;

grant select on V\_USERINFOR to rol\_Passenger;

--

-- grant procedures

grant execute on dbo.pro\_AddBooking to rol\_Passenger;

grant execute on dbo.pro\_AddDefaultBooking to rol\_Passenger;

grant execute on dbo.pro\_AddPassenger to rol\_Passenger;

grant execute on dbo.pro\_AddPassengerAccount to rol\_Passenger

grant execute on dbo.pro\_CancelTicket to rol\_Passenger;

--

-- grant functions

grant execute on dbo.func\_auto\_id\_booking to rol\_Passenger;

grant execute on dbo.func\_auto\_id\_passenger to rol\_Passenger;

--grant execute on dbo.func\_GetAvailabelSeat to rol\_Passenger;

--

-- deny

deny delete on PASSENGER to rol\_Passenger;

deny update, insert, delete, references on AGENT to rol\_Passenger;

deny update, insert, delete, references on BUS to rol\_Passenger;

deny update, insert, delete, references on BUSROUTE to rol\_Passenger;

deny update, insert, delete, references on BUSSTATION to rol\_Passenger;

deny select, update, insert, delete, references on CASHRESERVE to rol\_Passenger;

deny select, update, insert, delete, references on EMPLOYEE to rol\_Passenger;

deny delete on PACKAGE to rol\_Passenger;

deny update, insert, delete, references on PLACE to rol\_Passenger;

deny select, update, insert, delete, references on POSITION to rol\_Passenger;

deny select, update, insert, delete, references on SYSTEMACCOUNT to rol\_Passenger;

deny update, insert, delete, references on TRIP to rol\_Passenger;

**\* For staff**

exec sp\_addrole rol\_Staff;

-- grant tables

grant select on AGENT to rol\_Staff;

grant select, insert, delete, references on BOOKING to rol\_Staff;

grant select on BUS to rol\_Staff;

grant select on BUSROUTE to rol\_Staff;

grant select on BUSSTATION to rol\_Staff;

grant select, insert, update, delete, references on PASSENGER to rol\_Staff;

grant select, insert, update, delete, references on PASSENGERACCOUNT to rol\_Staff;

grant select on PLACE to rol\_Staff;

grant select, update on TICKET to rol\_Staff;

grant select on TRIP to rol\_Staff;

grant select, update on EMPLOYEE to rol\_Staff;

grant select, update on SYSTEMACCOUNT to rol\_Staff;

--

--grant views

grant select on V\_AGENTINFOR to rol\_Staff;

grant select on V\_AVAILABLETRIP to rol\_Staff;

grant select on V\_BOOKEDTICKET to rol\_Staff;

grant select on V\_BOOKINGINFOR to rol\_Staff;

grant select on V\_BUSSTATIONINFOR to rol\_Staff;

grant select on V\_ROUTEINFOR to rol\_Staff;

grant select on V\_TRIPINFOR to rol\_Staff;

grant select on V\_USERINFOR to rol\_Staff;

grant select on V\_EMPLOYEEINFOR to rol\_Staff;

grant select on V\_DRIVERINFOR to rol\_Staff;

--

-- grant procedures

grant execute on dbo.pro\_AddBooking to rol\_Staff;

grant execute on dbo.pro\_AddDefaultBooking to rol\_Staff;

grant execute on dbo.pro\_AddPassenger to rol\_Staff;

grant execute on dbo.pro\_AddPassengerAccount to rol\_Staff

grant execute on dbo.pro\_CancelTicket to rol\_Staff;

--

-- grant functions

grant execute on dbo.func\_auto\_id\_booking to rol\_Staff;

grant execute on dbo.func\_auto\_id\_passenger to rol\_Staff;

--grant execute on dbo.func\_GetAvailabelSeat to rol\_Staff;

grant execute on dbo.func\_auto\_id\_employee to rol\_Staff;

--

-- deny

deny delete on PASSENGER to rol\_Staff;

deny update, insert, delete, references on AGENT to rol\_Staff;

deny update, insert, delete, references on BUS to rol\_Staff;

deny update, insert, delete, references on BUSROUTE to rol\_Staff;

deny update, insert, delete, references on BUSSTATION to rol\_Staff;

deny select, update on CASHRESERVE to rol\_Staff;

deny insert, delete, references on EMPLOYEE to rol\_Staff;

deny update, insert, delete, references on PLACE to rol\_Staff;

deny select, update, insert, delete, references on POSITION to rol\_Staff;

deny insert, delete, references on SYSTEMACCOUNT to rol\_Staff;

deny update, insert, delete, references on TRIP to rol\_Staff;

**\* For server admin**

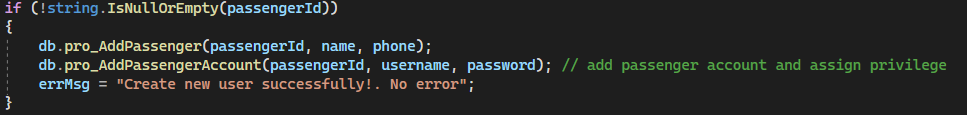
exec sp\_addrole rol\_admin;

grant control on DATABASE::BusManagement to rol\_admin;

**\* [C#] Create an account for a passenger and assign the Passenger role to the account**

When the user doesn’t have an account to use the app, they will have the option to sign up for a brand new account.

This is when they will input certain information including their name, phone number, the account’s username and password, which will be used by these procedures: pro\_AddPassenger, pro\_AddPassengerAccount.

****

pro\_AddPassenger will insert the new information into Passenger table.

CREATE proc [dbo].[pro\_AddPassenger] @id\_passenger char(20), @name nvarchar(50), @phone char(20)

as

begin

insert into PASSENGER(id\_passenger, name, phone\_number) values (@id\_passenger, @name, @phone);

end;

pro\_AddPassengerAccount will do 2 things: Insert new username and password into PassengerAccount table then assign Passenger role to the user.

CREATE proc [dbo].[pro\_AddPassengerAccount] @id\_passenger char(20), @username varchar(50), @password varchar(50)

as

begin

insert into PASSENGERACCOUNT values (@id\_passenger, @username, @password);

exec dbo.pro\_AssignPassengerPrivilege @id\_passenger;

end;

**\* Procedure to assign privileges to passengers**

CREATE proc [dbo].[pro\_AssignPassengerPrivilege] (@id\_passenger char(20))

as

begin

SET XACT\_ABORT ON

begin tran

begin try

declare @username varchar(50), @sqlString nvarchar(MAX)

select @username = PASSENGERACCOUNT.username from PASSENGERACCOUNT where PASSENGERACCOUNT.id\_passenger = @id\_passenger

set @sqlString = 'exec sp\_addrolemember ''rol\_Passenger'', ''' + @username + ''''

exec (@sqlString)

commit tran

end try

begin catch

rollback

end catch

end

**\* Trigger for PassengerAccount table when a new row of data is created**

CREATE trigger [dbo].[tr\_CreatePassengerAccount] on [dbo].[PASSENGERACCOUNT]

after insert

as

declare @username varchar(30), @password varchar(10)

select @username = ins.username, @password = ins.password from inserted ins

begin

begin tran

begin try

declare @sql nvarchar(max);

set @sql = 'create login ' + quotename(@username) + ' with password = ''' + @password + ''', DEFAULT\_DATABASE=[BusManagement], CHECK\_EXPIRATION=OFF, CHECK\_POLICY=OFF';

exec sp\_executesql @sql;

set @sql = 'create user ' + quotename(@username) + ' for login ' + quotename(@username);

exec sp\_executesql @sql;

commit tran;

end try

begin catch

rollback

end catch

end

**\* Procedure to assign system privileges to staff members and admins**

CREATE proc [dbo].[pro\_AssignSystemPrivilege] (@id\_employee char(20), @id\_position char(20))

as

begin

SET XACT\_ABORT ON

begin tran

begin try

declare @username varchar(50), @position\_name varchar(50), @sqlString varchar(1000)

--

select @username = b.username

from EMPLOYEE as a inner join SYSTEMACCOUNT as b on a.id\_account = b.id\_account

where a.id\_employee = @id\_employee;

--

select @position\_name = a.type from POSITION as a where a.id\_position = @id\_position;

--

if(@position\_name = 'administrator')

set @sqlString = 'exec sp\_addrolemember ''rol\_Admin'', ''' + @username + '''';

else

set @sqlString = 'exec sp\_addrolemember ''rol\_Staff'', ''' + @username + '''';

exec (@sqlString);

insert into EMPLOYEE\_POSITION values(@id\_employee, @id\_position);

commit tran;

end try

begin catch

rollback;

end catch

end

**\* Sensitive class to load users’ information after logging in**

Text

Description automatically generated

**\* C# Function to check whether the login credentials are of a Passenger’s, an Admin’s or a Staff’s and save their information**

Text

Description automatically generated

Text

Description automatically generated

When logging in, the app will use the default connection string. After logging in, the app will switch to a different connection string that contains the saved information that was input by the user.

**\* [C#] Code inside StaticEnv**

public static string GetEFConnectionString(string username, string password)

=> $"metadata=res://\*/src.dbConnection.BusManagementModel.csdl|res://\*/src.dbConnection.BusManagementModel.ssdl|res://\*/src.dbConnection.BusManagementModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=(local);initial catalog=BusManagement;user id={username};password={password};MultipleActiveResultSets=True;App=EntityFramework\"";

public static string GetEFConnectionString()

=> $"metadata=res://\*/src.dbConnection.BusManagementModel.csdl|res://\*/src.dbConnection.BusManagementModel.ssdl|res://\*/src.dbConnection.BusManagementModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=(local);initial catalog=BusManagement;user id={UserData.Username};password={UserData.Password};MultipleActiveResultSets=True;App=EntityFramework\"";

public static string GetDefaultEFConnectionString()

=> "metadata=res://\*/src.dbConnection.BusManagementModel.csdl|res://\*/src.dbConnection.BusManagementModel.ssdl|res://\*/src.dbConnection.BusManagementModel.msl;provider=System.Data.SqlClient;provider connection string=\"data source=(local);initial catalog=BusManagement;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework\"";

**\* Creating a new database instance to connect to the database using the Staff/Employee Connection String:**

public BusManagementEntities(string connectionStr) : base(connectionStr)

{

}

BusManagementEntities db = new BusManagementEntities(StaticEnv.GetDefaultEFConnectionString());

**\* Creating a new database instance to connect to the database using the User Connection String:**

public BusManagementEntities() : base(StaticEnv.GetEFConnectionString())

{

}

BusManagementEntities db = new BusManagementEntities();

# CHAPTER V: SYSTEM INTERFACE DESIGN

## Applications and services used

This part contains all the applications used during the making of this project.

* Microsoft SQL Server 2022
* Microsoft SQL Server Management Studio 19 (SSMS 19)
* Windows Forms App (.NET Framework) built using Visual Studio 2022
* Packages: Entity Framework, Mailkit, Mimekit, BouncyCastle.Cryptography

## Software interface

**\* Login & Sign up page**

This is where users will have to input their username and password in order to use the software’s booking functions or access the database in general.

In the case a user does not possess an account to use the software, they can sign up for a new account after filling out certain information criteria.

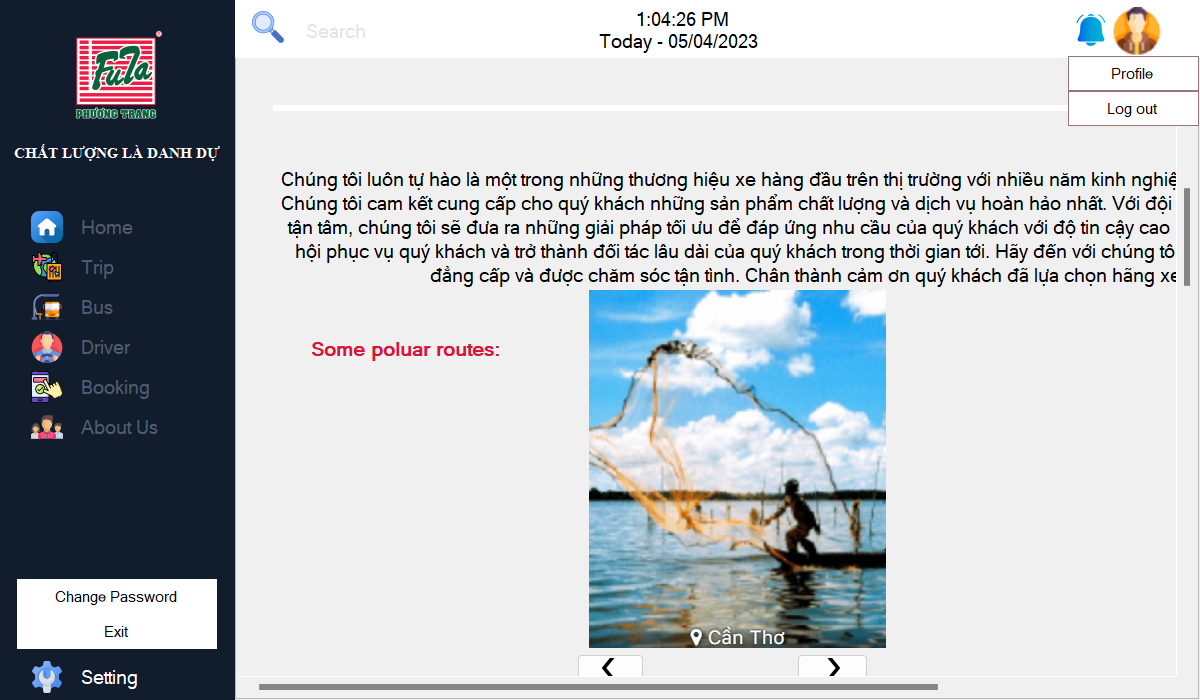
Graphical user interface, application, Teams

Description automatically generatedGraphical user interface, application

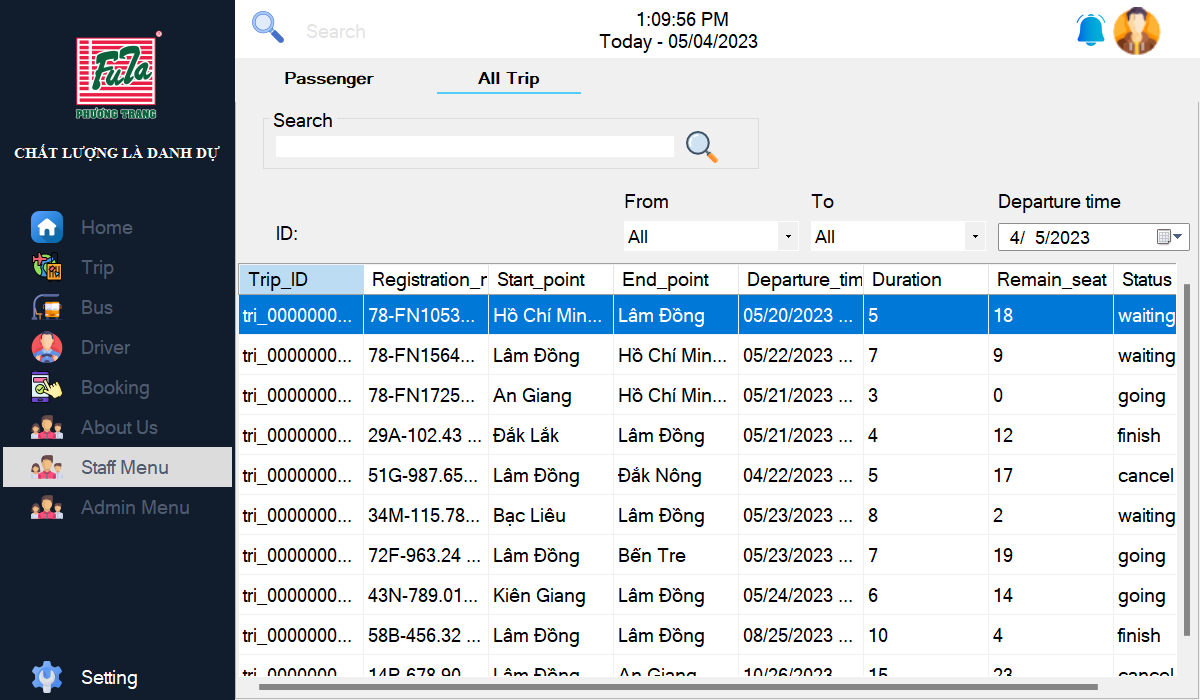
Description automatically generated

**\* Home screen**

**- For passengers**



**- For server admins**

****

**\* Change password screen**

Graphical user interface, application

Description automatically generated

**\* Profile screen**

**Graphical user interface

Description automatically generated**

**\* Trip screen**

**Graphical user interface, text, application, email

Description automatically generated**

**- Get all trip data**

public List<V\_TRIPINFOR> GetAllTrips(DateTime dateTime)

{

BusManagementEntities db = new BusManagementEntities();

var res = db.V\_TRIPINFOR.ToList();

return res;

}

**- Search for available trips**

public List<V\_AVAILABLETRIP> SearchAvailableTrips(string input, string src, string des, DateTime dateTime)

{

BusManagementEntities db = new BusManagementEntities();

var res = db.V\_AVAILABLETRIP.Where(d => d.Departure\_time > dateTime);

if (src != "All")

{

res = res.Where(d => d.Start\_point == src);

}

if (des != "All")

{

res = res.Where(d => d.End\_point == des);

}

if (!string.IsNullOrEmpty(input))

{

res = res.Where(d => d.Trip\_ID.Contains(input.Trim()));

}

return res.ToList();

}

**\* Bus screen**

Graphical user interface, table

Description automatically generated

**- Get all bus information**

public List<BUS> GetAllBus()

{

BusManagementEntities db = new BusManagementEntities();

var res = db.BUSes.ToList();

return res;

}

**- Search bus by ID**

public List<BUS> SearchBusByID(string input, bool type)

{

BusManagementEntities db = new BusManagementEntities();

var res = FilterBus(type);

if (!string.IsNullOrEmpty(input))

{

res = res.Where(d => d.id\_bus.Contains(input)).ToList();

return res.ToList();

}

return res.ToList();

}

**- Search bus by registration number**

public List<BUS> SearchBusByRegistrationNumber(string input, bool type)

{

BusManagementEntities db = new BusManagementEntities();

var res = FilterBus(type);

if (!string.IsNullOrEmpty(input))

{

res = res.Where(d => d.registration\_number.Contains(input)).ToList();

return res.ToList();

}

return res.ToList();

}

**- Filter bus by type (Interprovince/Transit)**

public List<BUS> FilterBus(bool type)

{

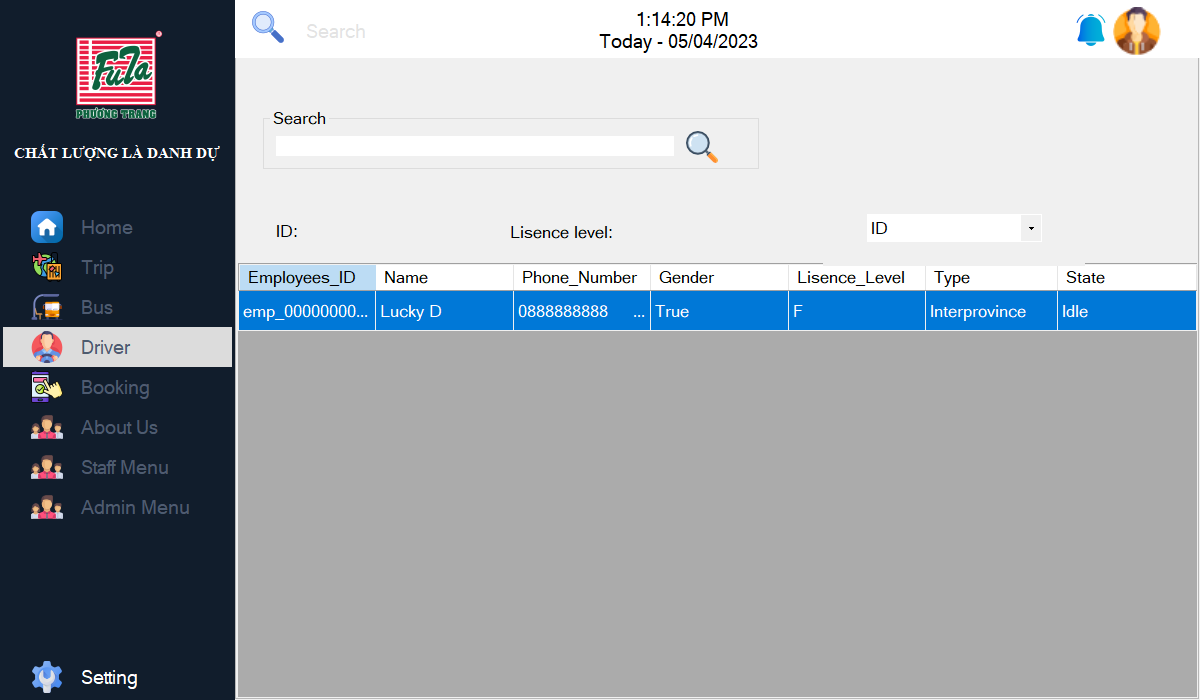
BusManagementEntities db = new BusManagementEntities();

var res = db.BUSes.Where(d => d.type == type);

return res.ToList();

}

**\* Driver screen**

****

**- Get all drivers information**

private void LoadMainData()

{

BSDriver bSDriver = new BSDriver();

this.DgvMainData.DataSource = bSDriver.GetAllDrivers();

}

**\* Booking & Booked screen**

Graphical user interface, website

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

**- Get available seats**

public List<string> GetAvailableSeat(string idTrip, int type) // type 0: seat, 1: sleeper

{

BusManagementEntities db = new BusManagementEntities();

string funcName = "func\_GetAvailabelSeat";

List<string> ticketList = BSMain.RunTableValuedFunc(funcName, new List<string> { idTrip, type.ToString() });

return ticketList;

}

**- Get booked ticket information**

public List<V\_BOOKEDTICKET> SearchBookedTickets(string passengerId,string input, string src, string des, DateTime dateTime)

{

BusManagementEntities db = new BusManagementEntities();

var res = db.V\_BOOKEDTICKET.Where(d => d.Departure\_time > dateTime && d.Passenger\_ID == passengerId);

if (src != "All")

{

res = res.Where(d => d.Start\_point == src);

}

if (des != "All")

{

res = res.Where(d => d.End\_point == des);

}

if (!string.IsNullOrEmpty(input))

{

res = res.Where(d => d.Ticket\_ID.Contains(input.Trim()));

}

return res.ToList();

}

**\* Email received to confirm after successful booking**

A picture containing text

Description automatically generated

**\* [C#] Code inside IEmailSender.cs**

internal interface IEmailSender

{

Task SendEmailAsync(string email, string subject, string body);

}

**\* [C#] Code inside EmailSender.cs**

public Task SendEmailAsync(string email, string subject, string body)

{

string host = "smtp.gmail.com";

int port = 465;

string username = LocalEnv.EmailServerName;

string password = LocalEnv.EncodedEmailServerPassword;

//

MimeMessage message = new MimeMessage();

message.From.Add(new MailboxAddress("FUTA Bus Lines", "noreply03@futa.vn"));

message.To.Add(MailboxAddress.Parse(email));

message.Subject = subject;

//

BodyBuilder builder = new BodyBuilder();

builder.HtmlBody = body;

message.Body = builder.ToMessageBody();

//

SmtpClient smtpClient = new SmtpClient();

//

smtpClient.Connect(host, port, true);

smtpClient.Authenticate(username, password);

return smtpClient.SendAsync(message);

}

**\* About us screen**

****

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**\* Passenger & All trips screen (Only visible to server admin)**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**- Get passenger information**

private void FilterPassengers()

{

BSPassenger bsbooked = new BSPassenger();

int tag = this.CbField.SelectedIndex;

this.DgvMainData.DataSource = bsbooked.SearchPassenger(searchInput, tag); // 0: ID, 1: Name

}

**- Get all trip information**

public List<V\_TRIPINFOR> SearchTrips(string input, string src, string des, DateTime dateTime)

{

BusManagementEntities db = new BusManagementEntities();

var res = db.V\_TRIPINFOR.Where(d => d.Departure\_time > dateTime);

if (src != "All")

{

res = res.Where(d => d.Start\_point == src);

}

if (des != "All")

{

res = res.Where(d => d.End\_point == des);

}

if (!string.IsNullOrEmpty(input))

{

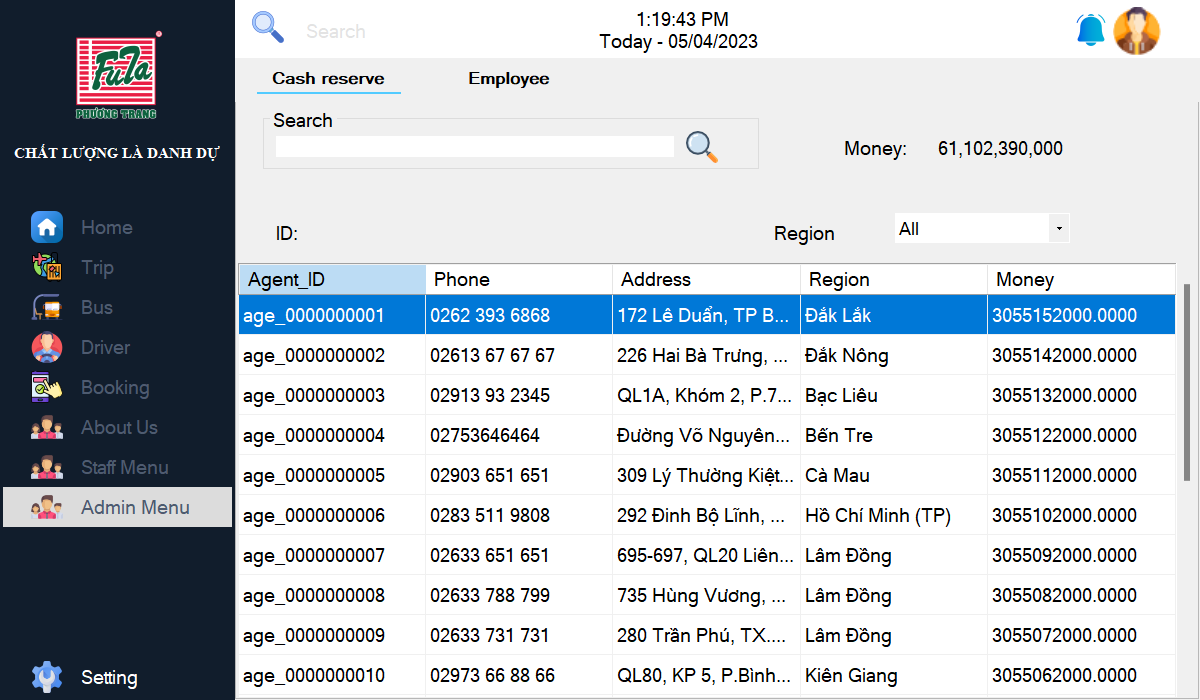
res = res.Where(d => d.Trip\_ID.Contains(input.Trim()));

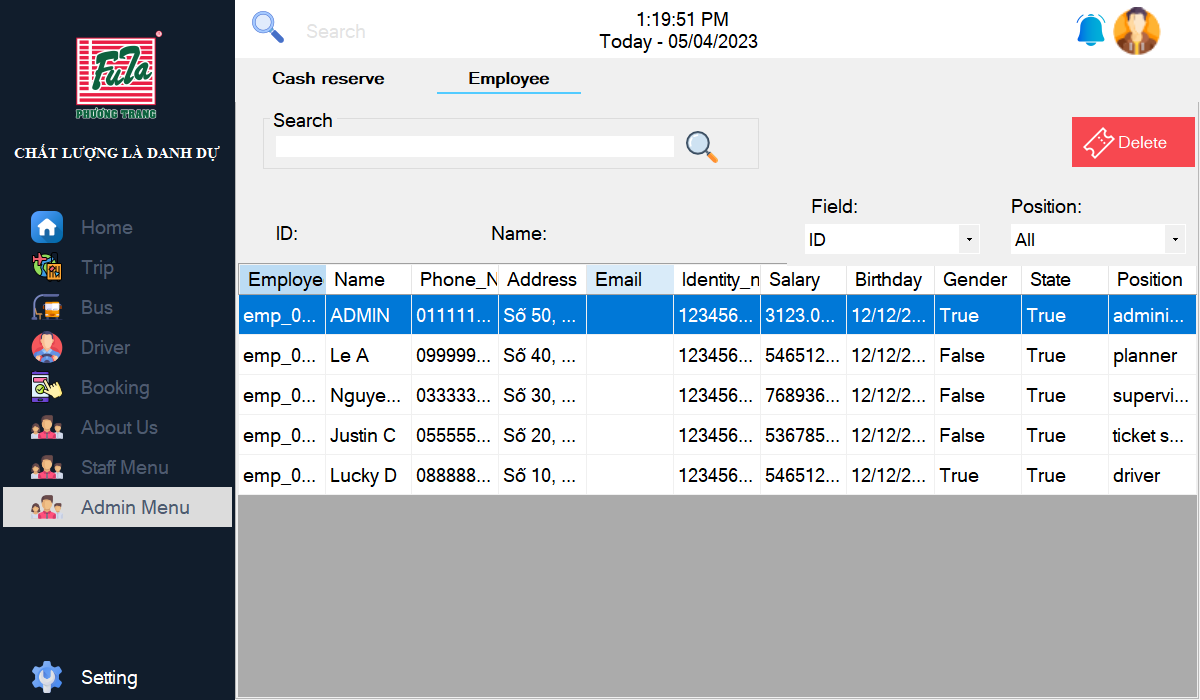
}

return res.ToList();

}

**\* Admin menu – Cash reserve & Employee (Only accessible by admins)**

****

****

**- Get cash reserve information**

private void FilterAgent()

{

BSAgent bsagent = new BSAgent();

List<V\_AGENTINFOR> dataSource = bsagent.SearchAgents(this.searchInput.Trim(), this.CbRegion.Text.Trim());

this.DgvMainData.DataSource = dataSource;

this.LbSumMoney.Text = dataSource?.Aggregate(0m, (s, d) => s + (decimal)d.Money).ToString("###,###,###,###");

}

**- Get employee information**

private void FilterEmployees()

{

BSEmployee bsemployee = new BSEmployee();

int tag = this.CbField.SelectedIndex;

string position = this.CbPosition.Text.Trim();

this.DgvMainData.DataSource = bsemployee.SearchEmployees(this.searchInput.Trim(), tag, position);

}

*(This page was intentionally left blank)*