Software Design Description For Pharomina Tours: Local System For Tourism Company

Ahmed Emad, Mohamed Abdelhamed , Fady Bassel, Mark Refaat

May 10, 2020

SDD Version	Date	Reason for Change
1.0	8-May-2020	SDD First version's specifications are defined

Table 1: Document version history

GitHub: https://github.com/markRefaat/tourism-company-project/tree/develop

1 Introduction

1.1 Scope

The scope of the project is a local travel system for a company called ”Pharomina Tours”. Pharomina Tours is a way of focus on the company's treasury system, customers data and flight ticket booking. This project's target is to help the company owner and workers to have easier work with better accuracy and easier system with all their needed data and also reduce paperwork. These methods could be used to book flight tickets, add new customer, manage the safe transactions and view generated reports. Workers have stated that the extreme problem that the system's reports generator is not accurate and they need the reports to be well generated with high accuracy so that the system will help them achieve this goal. For the admins: can add/edit/delete customer, refund tickets, add in the safe and view it, view reports, add new ticket and pay them and edit today's orders and view them . For the help desk: can add new ticket, view today's orders, pay tickets and edit today's orders. For the system: send mails and generate reports.

1.2 Purpose

The purpose of software design document is to present detailed description of project system architecture and design. The document will explain the features of the Pharomina Tours system and provides insight into the structure and design of each component. Also, this document will help the project team and the customer to have a full overview about the interface and the functions of the project. The customer can review this document to be aware of the requirements that is finished.

1.3 Overview

The system manages a tourism company locally, and this document shows in detail how is it done. Specifying all the classes, functions and tools used in creating the project. Besides stating any constraints and showing diagrams to make it easier to understand the system.

1.4 Intended audience

The system is mainly intended to make it easier for the company workers to manage their data and for the manager to get reports.

1.5 Definitions and Acronyms

Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

Term	Definition
Late Customer	type of customer that saves his data and doesn't pay for the ticket at the moment of purchase and have a permission to pay after a period of time.
Cash Customer	type of customer that pays for the purchased tickets by cash at the moment of buying it
Receipt	divided into 3 types fatoora w eysal estlam w ezn sarf

2 Project Overview

The system aims to help a tourism company track customers flight tickets payments, manage their finances and generate bills and invoices as well as generating detailed reports at times specified by the user. The system is divided into three main parts customers data, booking flight tickets and safe management. The company divides its customers into two categories regular and ordinary customers, the regular customer has a 15 days span to deliver the ticket(s) payment (15 days upon client request but could be changeable) the system should provide basic information of the customer and if he has any unpaid tickets as well as his previous purchased tickets and notify the company if a customer passed the 15 days allowed for paying the system should also deal with refunding tickets. Booking tickets is done through a form containing basic ticket information (date, destination, price, passenger, etc..) and the result of submitting is a printed bill alongside adding the ticket in the customers account and commencing the 15 day payment period. The system also should deal with the safe, accountants should be able to add expenses/revenues and see detailed general and custom reports generated by the system, also tickets information should be added automatically in the revenues upon payment.

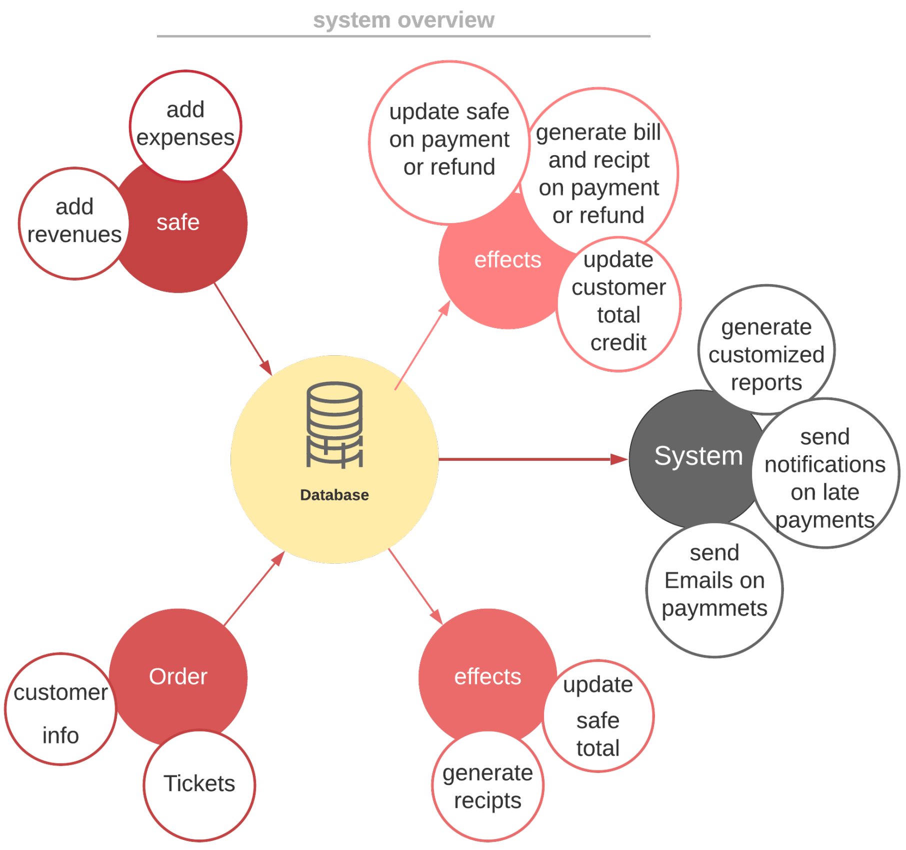

system overview
add expenses
add revenues
safe
update safe on payment or refund
generate bill and recipt on payment or refund
effects
update customer total credit
generate customized reports
Database
System
send notifications on late payments
send Emails on paymmets
Order
customer info
Tickets
effects
update safe total
generate recipts

Figure 1: Architectural Design

2.1 Project Scope

The main goal of the project is to have a web-based local system for the company to replace their paperwork and ease the statistic work for them.
This is achieved by having the system storing each user data, each order data and ticket data.
Then the tickets can be paid one by one and also can be refunded.
All of the transactions are tracked and stored in the safe.
Later a report can be generated to summarize the tickets and receipt flow.
The whole process take few seconds in data input, invoice and reports generation, which is definitely faster than the paper work time.

2.2 Goals and objectives

Reducing excel sheets entries and paperwork by replacing all paper data with MySQL database.
Reducing the work hassle of generating report and getting the data from many excel sheets by auto generating them by the system with the links of the database.
Auto generating some statistics at the system homepage instead of the hassle of manual calculations.

2.3 Project Timeline

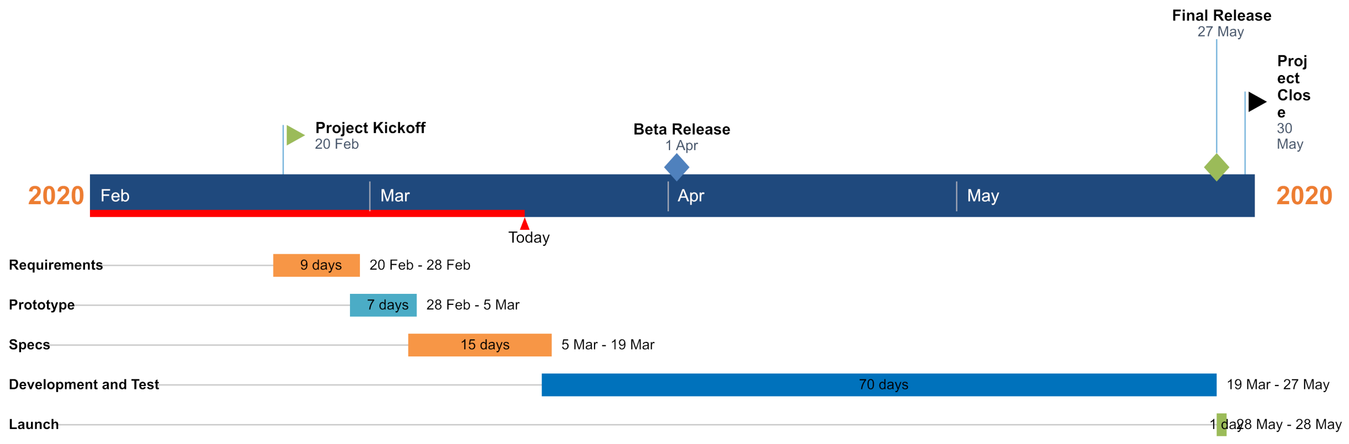


Figure 2: Project Timeline

3 Context viewpoint

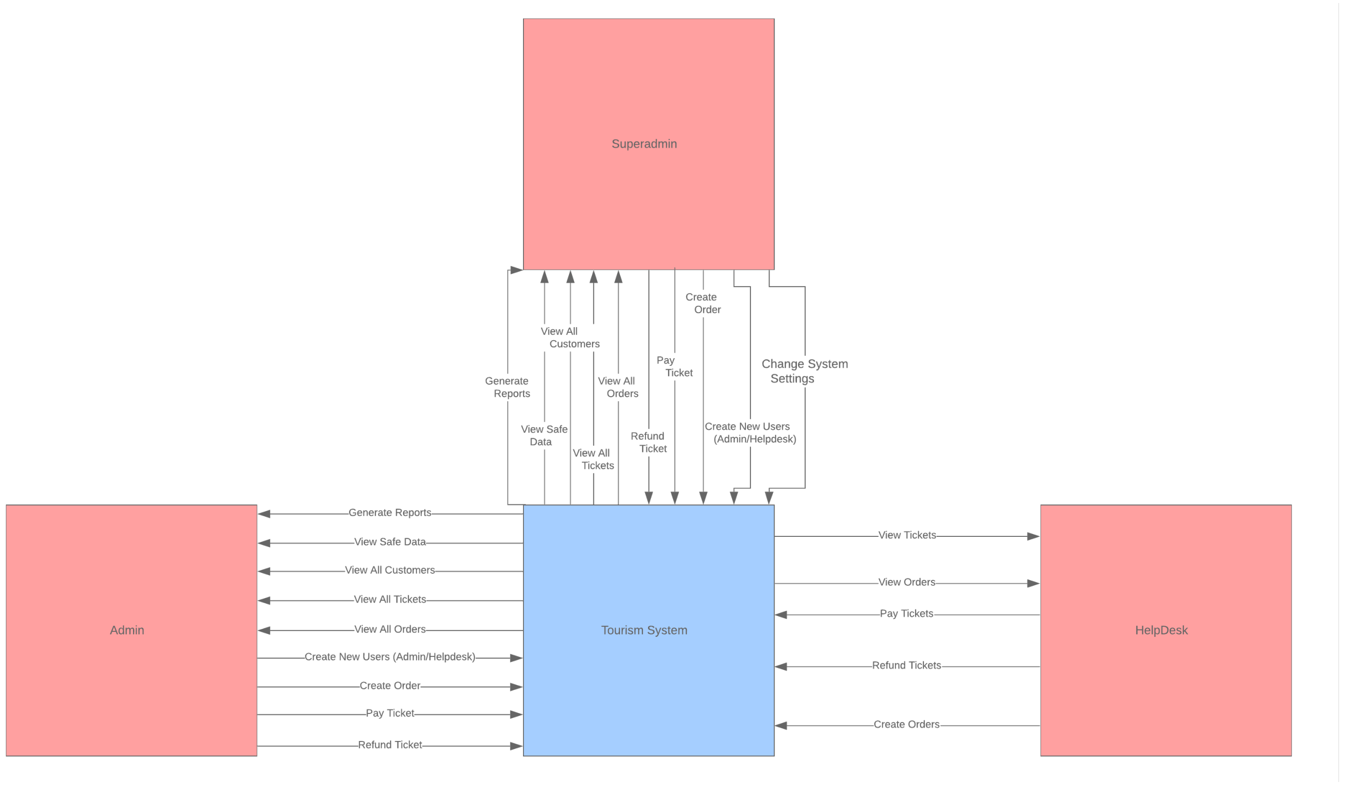


Figure 3: Project Context Diagram

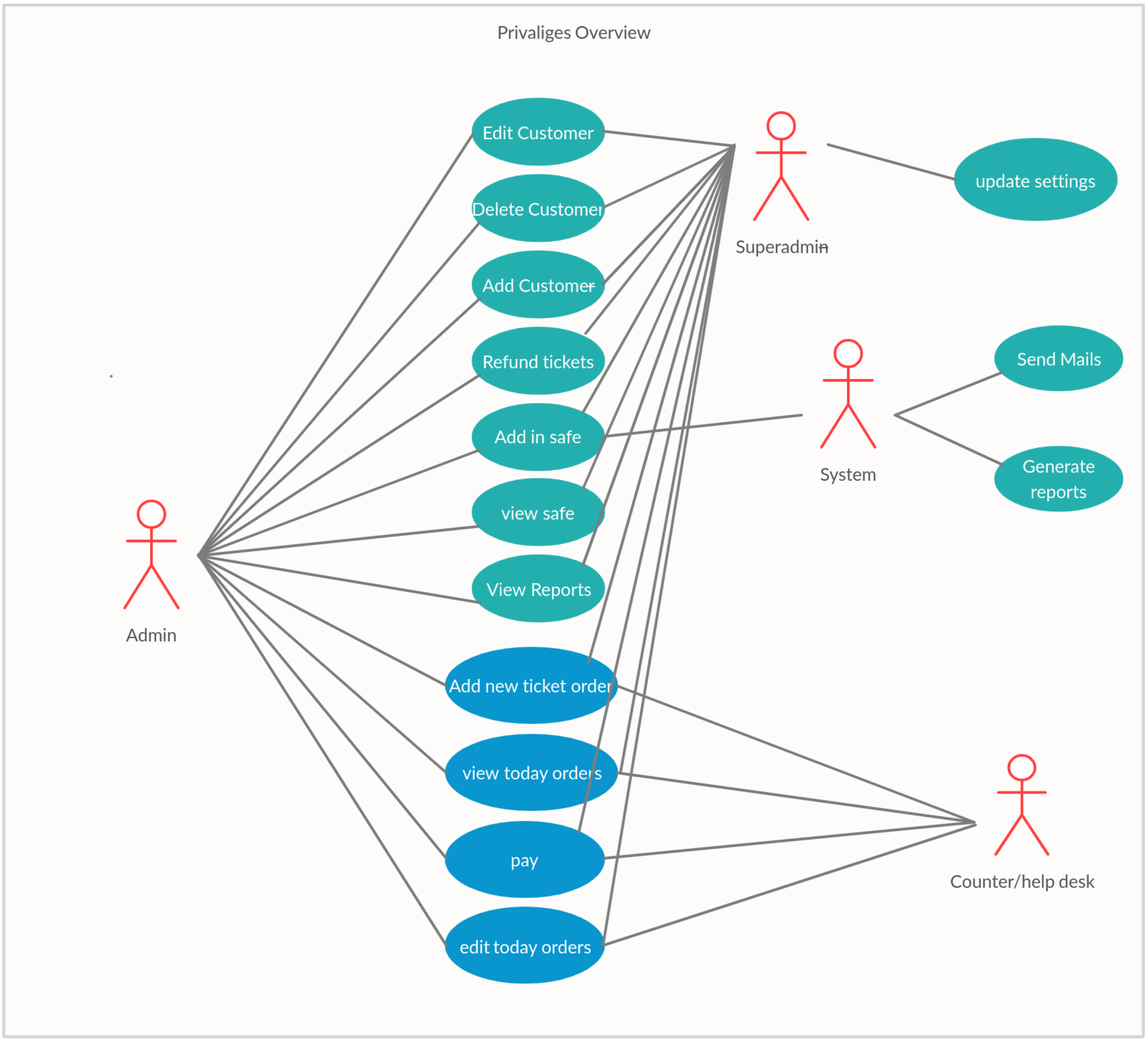


Figure 4: Project Usecase Diagram

4 System Architecture Design

4.1 Logical viewpoint

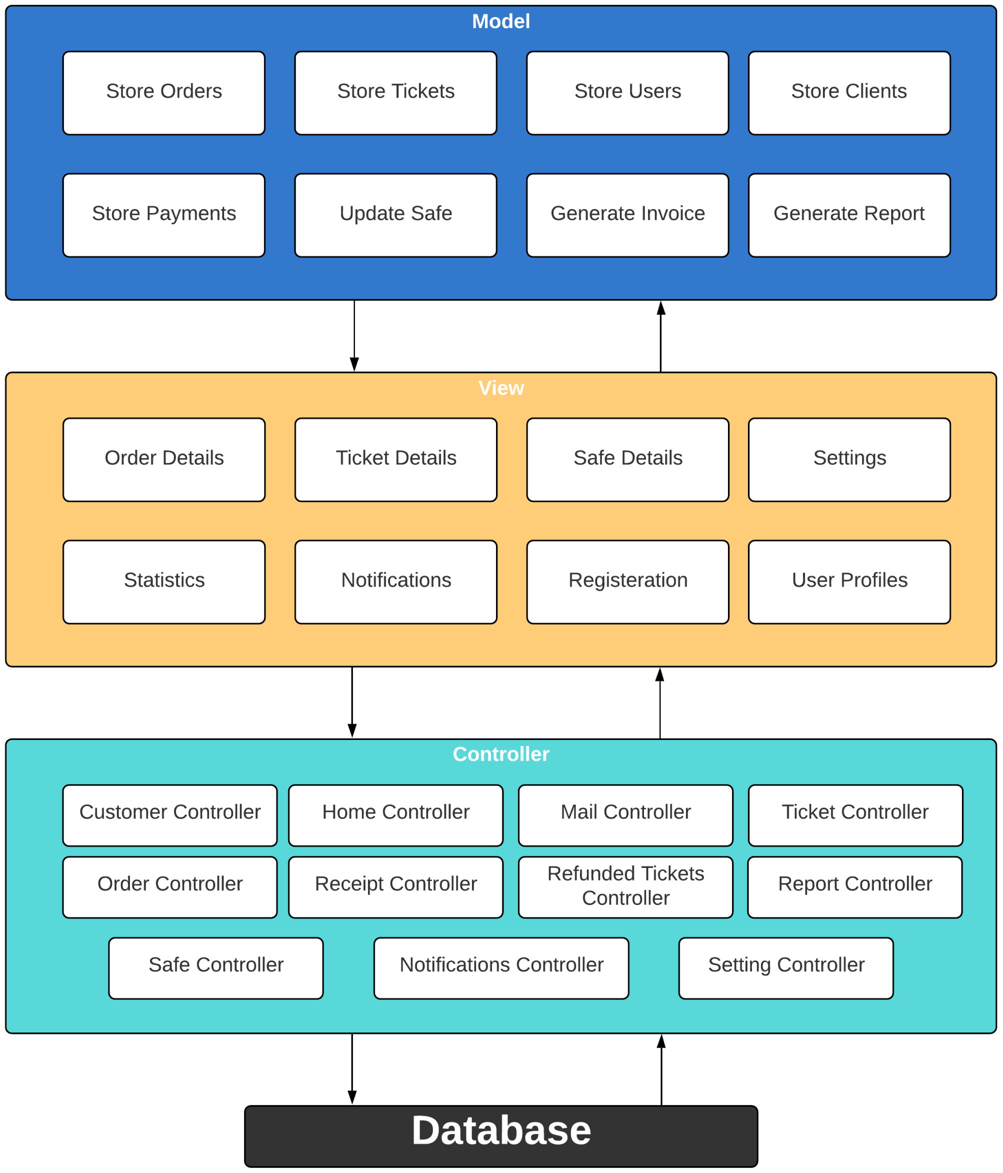


Figure 5: System Architecture

4.1.1 Design Rationale

We chose MVC because it allows us to create many view for the same model, the modifications in views will not affect the model so it will save a lot of hassle in any later design updates and it supports parallel development where each developer can work in separate part (either model, controller or view).

4.2 Patterns use viewpoint

Laravel using design patterns in its structure. Auth and Validation built on facade design pattern. A facade is a class wrapping a complex library to provide a simpler and more readable interface to it. In laravel, Facades provide a "static" interface to classes that are available in the application's service container.

4.2.1 Design Rationale

Facade design pattern helps us in validation on inputs from users using validate method in Validate class, And in getting data of logged in user using built in Auth Class.

4.3 Composition viewpoint

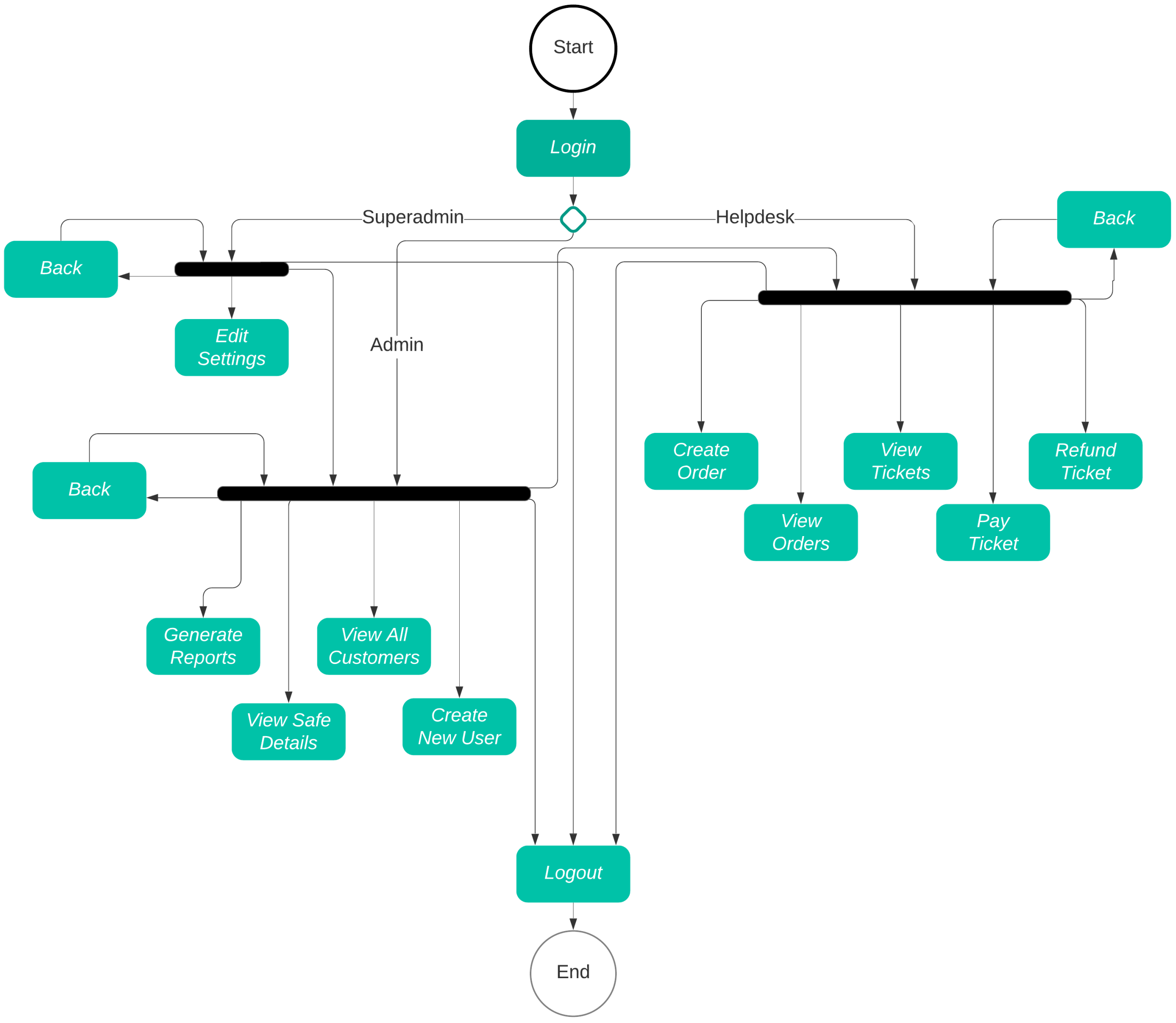
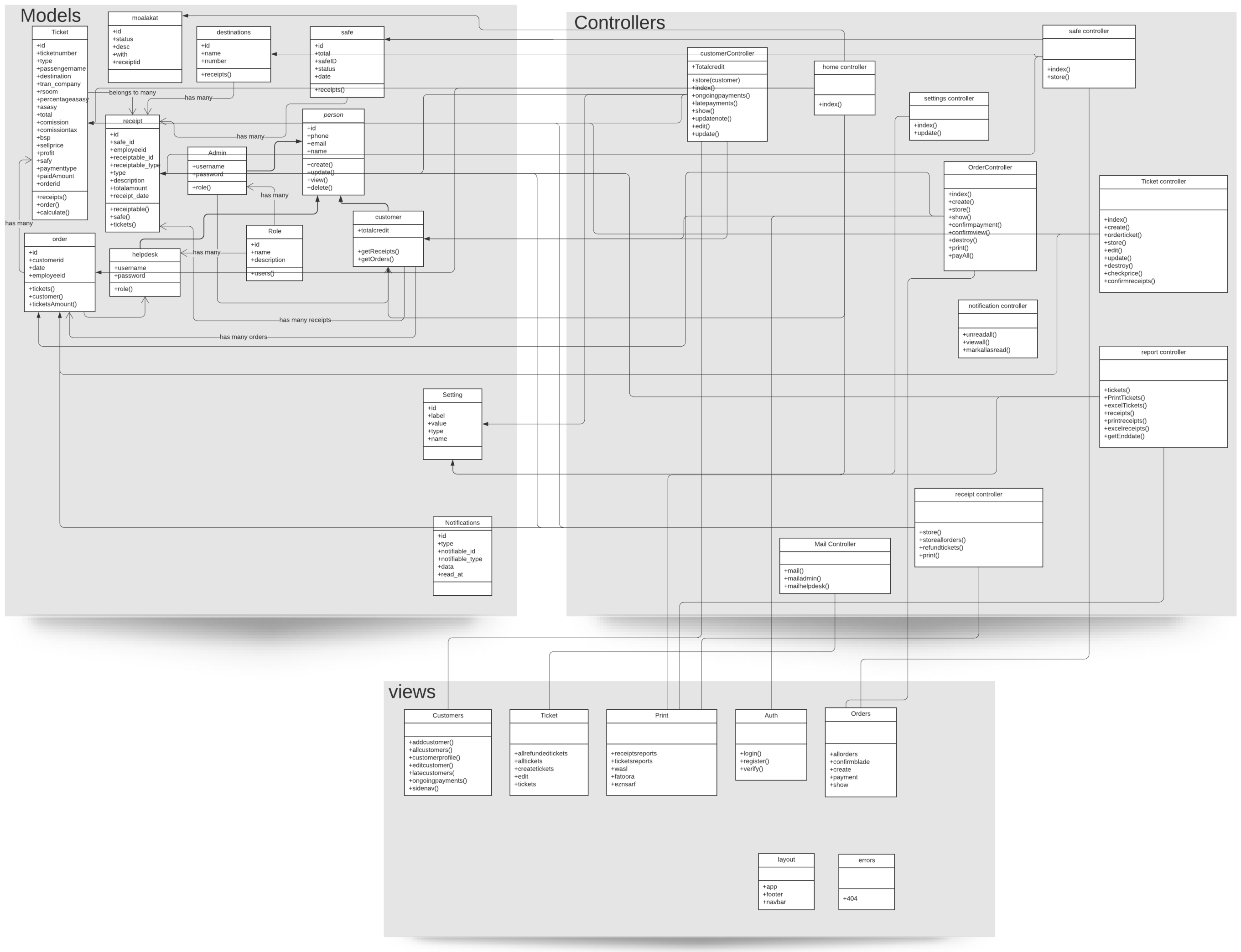


Figure 6: Activity Diagram

4.4 Structure viewpoint



Models
Ticket
+id +ticketnumber +type +passengername +destination +tran_company +rsoom +percentageasasy +asasy +total +comission +comissiontax +bsp +sellprice +profit +safy +paymenttype +paidAmount +orderid
+receipts() +order() +calculate()
moalakat
+id +status +desc +with +receiptid
destinations
+id +name +number
+receipts()
safe
+id +total +safeID +status +date
+receipts()
receipt
+id +safe_id +employeeid +receiptable_id +receiptable_type +type +description +totalamount +receipt_date
+receiptable() +safe() +tickets()
Admin
+username +password
+role()
person
+id +phone +email +name
+create() +update() +view() +delete()
Role
+id +name +description
+users()
customer
+totalcredit
+getReceipts() +getOrders()
order
+id +customerid +date +employeeid
+tickets() +customer() +ticketsAmount()
helpdesk
+username +password
+role()
Setting
+id +label +value +type +name
Notifications
+id +type +notifiable_id +notifiable_type +data +read_at
belongs to many
has many
has many receipts
has many orders
Controllers
customerController
+Totalcredit
+store(customer) +index() +ongoingpayments() +latepayments() +show() +updatenote() +edit() +update()
home controller
+index()
settings controller
+index() +update()
safe controller
+index() +store()
OrderController
+index() +create() +store() +show() +confirmpayment() +confirmview() +destroy() +print() +payAll()
Ticket controller
+index() +create() +orderticket() +store() +edit() +update() +destroy() +checkprice() +confirmreceipts()
notification controller
+unreadall() +viewall() +markallasread()
report controller
+tickets() +PrintTickets() +excelTickets() +receipts() +printreceipts() +excelreceipts() +getEnddate()
receipt controller
+store() +storeallorders() +refundtickets() +print()
Mail Controller
+mail() +mailadmin() +mailhelpdesk()
views
Customers
+addcustomer() +allcustomers() +customerprofile() +editcustomer() +latecustomers(+ongoingpayments() +sidenav()
Ticket
+allrefundedtickets +alltickets +createtickets +edit +tickets
Print
+receiptsreports +ticketsreports +wasl +fatoora +eznsarf
Auth
+login() +register() +verify()
Orders
+allorders +confirmblade +create +payment +show
layout
+app +footer +navbar
errors
+404

Figure 7: Class Diagram

Class Name: Ticket Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Generating and retrieving ticket info.
Collaborations: N/A
Attributes: id, ticketnumber, type, passengername, destination, trancompany, rsoom, percentageasasy, asasy, total, comission, comissiontax, bsp, sellprice, profit, safy, paymenttype, orderid.
Operations: viewRefunded
Constraints: N/A

Class Name: safe Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Retrieving receipts data.
Collaborations: receipt class to add its data to the total safe data.
Attributes: id, receiptno, total, description, type, destination.
Operations: addreceipt(receipt)
Constraints: N/A

Class Name: moalakat Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Generating and retrieving moalakat data.
Collaborations: Names each class with which this class must interact in order to accomplish its purpose, and how.
Attributes: id, status, desc, with, receiptid.
Operations: N/A
Constraints: N/A

Class Name: person Class
Abstract or Concrete: Abstract.
List of Superclasses: N/A
List of Subclasses: Admin, employee, customer.
Purpose: Generating and retrieving person data.
Collaborations: N/A
Attributes: id, phone, email, name.
Operations: N/A
Constraints: N/A

Class Name: order Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Generating and retrieving order data.
Collaborations: Ticket class, to have many tickets in the order.
Attributes: id, customerid, date, total, employeeid.
Operations: pay, refund, addtickets.
Constraints: N/A

Class Name: receipt Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Generating and retrieving receipt data.
Collaborations: N/A
Attributes: id, employeeid, from, type, description, totalamount.
Operations: N/A
Constraints: N/A

Class Name: Admin Class
Abstract or Concrete: Concrete.
List of Superclasses: Person.
List of Subclasses: N/A
Purpose: Generating and retrieving admin data.
Collaborations: N/A
Attributes: username, password.
Operations: getallcustomers, gettrasurydata, addCustomer.
Constraints: Person class should be abstract.

Class Name: user Class
Abstract or Concrete: Concrete.
List of Superclasses: Person.
List of Subclasses: N/A
Purpose: Generating and retrieving employee data.
Collaborations: order class, to be able to view and edit the orders.
Attributes: username, password.
Operations: addneworder, viewOrders, editorders.
Constraints: Person class should be abstract.

Class Name: customer Class
Abstract or Concrete: Concrete.
List of Superclasses: Person.
List of Subclasses: N/A
Purpose: Generating and retrieving customer data.
Collaborations: receipt class, to create and retrieve receipts. Order class, to retrieve orders history.
Attributes: totalcredit.
Operations: calculatetotalcredit, getdata, getreceipts, getorders.
Constraints: Person class should be abstract.

Class Name: safe controller Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control safe data.
Collaborations: safe Class, to be able to retrieve the data and write over it.
Attributes: N/A
Operations: getsafedata, addinvoice, addexpense, search.
Constraints: N/A

Class Name: customerController Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control customer data.
Collaborations: Customer Class, to be able to retrieve the data and write over it.
Attributes: Totalcredit.
Operations: calculateTotalcredit, getCustomerdata, getreceipts, getorders.
Constraints: N/A

Class Name: admin controller Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control admin data.
Collaborations: Admin Class, to be able to retrieve the data and write over it.
Attributes: N/A
Operations: addcustomer, generateReports, getsafedata, getcustomersdata.
Constraints: N/A

Class Name: EmployeeController Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control employee data.
Collaborations: Employee Class, to be able to retrieve the data and write over it.
Attributes: N/A
Operations: addorder, vieworders, editorders.
Constraints: N/A

Class Name: OrderController Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control order data.
Collaborations: Order Class, to be able to retrieve the data and write over it.
Attributes: N/A
Operations: pay, refund, addtickets.
Constraints: N/A

Class Name: customerdetailview Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/retrieve customer data.
Collaborations: N/A
Attributes: N/A
Operations: viewtotalcredit, viewcustomerdata, viewreceipts, viewOrders.
Constraints: N/A

Class Name: orderdetailsview Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/retrieve order data.
Collaborations: N/A
Attributes: N/A
Operations: pay, refund, addtickets.
Constraints: N/A

Class Name: employee dashboard Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/retrieve employee data.
Collaborations: N/A
Attributes: N/A
Operations: addorder, viewOrders, editOrder.
Constraints: N/A

Class Name: AdminDashboard Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/retrieve admin data.
Collaborations: N/A
Attributes: N/A
Operations: Addcustomer, viewReports, viewsafe, viewcustomers.
Constraints: N/A

Class Name: safeView Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/retrieve safe data.
Collaborations: N/A
Attributes: N/A
Operations: viewsafe, addinvoice, addexpense, search.
Constraints: N/A

Class Name: Mails Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Send mails.
Collaborations: N/A
Attributes: N/A
Operations: SendMail.
Constraints: N/A

Class Name: Notifications Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/Create Notifications.
Collaborations: N/A
Attributes: N/A
Operations: viewNotifications, SendNotifications, MarkAsRead.
Constraints: N/A

Class Name: Destination Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/Create Destination.
Collaborations: N/A
Attributes: N/A
Operations: viewDestinations.
Constraints: N/A

Class Name: Role Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View/Create Destination.
Collaborations: N/A
Attributes: N/A
Operations: viewDestinations.
Constraints: N/A

Class Name: Setting Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: View Settings.
Collaborations: N/A
Attributes: N/A
Operations: viewSettings.
Constraints: N/A

Class Name: SettingController Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control settings data.
Collaborations: Settings Class, to be able to retrieve the data and write over it.
Attributes: N/A
Operations: index, update.
Constraints: N/A

Class Name: ReportController Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Control reports data.
Collaborations: TicketsExport And ReceiptsExport Class, to be able to retrieve the data and download

reports.
Attributes: N/A
Operations: index, update.
Constraints: N/A

Class Name: TicketsExport Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Retrieve tickets for download report.
Collaborations: ReportController class, send data to download it‘
Attributes: N/A
Operations: map, headings, collection.
Constraints: N/A

Class Name: ReceiptsExport Class
Abstract or Concrete: Concrete.
List of Superclasses: N/A
List of Subclasses: N/A
Purpose: Retrieve receipts for download report .
Collaborations: ReportController class, send data to download it‘
Attributes: N/A
Operations: map, headings, collection.
Constraints: N/A

4.5 Algorithm viewpoint

TicketsAmount function:
Because the ticket can be payed only a part of it this function is for calculating the order total and how much payed for each ticket and how much is remaining.
The algorithm check all receipts of order tickets and calculate how much payed in each one and at the same time calculate the order total and returns in the end each ticket with the payed amount ,order total and total payed.

```
 $data = [];
$tickets = Ticket::with('receipts')->where('order_id', $this->id)->get();
$ordertotal = 0;
$payed = 0;
foreach ($tickets as $ticket) {
      if ($ticket->type != 'refunded') {
        $receipts = $ticket->receipts;
        $ordertotal += $ticket->sellprice;
        if ($receipts->count() > 0) {
          $total = 0;
          foreach ($receipts as $receipt) {
            if ($receipt->type == "revenue") {
```

```
            $total += $receipt->pivot->amount;
            $payed += $receipt->pivot->amount;
          }
        }
        array_push($data, [$ticket, $total]);
      } else {
        array_push($data, [$ticket, 0]);
      }
    } else {
      array_push($data, [$ticket, "refunded"]);
    }
  }
  return [$data, $ordertotal, $payed];
```

calculate function for ticket :
this function returns status of ticket (already payed , in progress) , payed amount of ticket , and remaining to be paid.

```
$receipts= $this->receipts;
       $sellprice=$this->sellprice;
       $status="in progress";
       $payed=0;
       if ($this->type=="refunded"){
           $status="ticket already refunded";
       }
       if($receipts->count()>0){
       foreach ($receipts as $receipt) {
           if($receipt->type=="revenue")
          $payed+=$receipt->pivot->amount;
       }
    }
       if ($sellprice==$payed){
           $status = 'already payed';
       }

           return [ "status"=>$status  , "payed"=>$payed ,"left" => $sellprice-$payed];
```

RefundTickets function: this function uses calculate function to make sure that part of the ticket has been payed and then attach that amount to the refund receipt and update the customer total credit ,then it calls function total amount on order to check if there are still payments to be paid or not and update the order status.

snippet of the code::

```
foreach ($tickets as $ticket) {
           $result = $ticket->calculate();

           if ($result['payed'] != 0) {
               $receipt->tickets()->attach("$ticket->id", ['amount' => $result['payed']]);
               $total += $result['payed'];
           }
           $ticket->type = "refunded";
           $ticket->save();
           $customer->totalcredit += $ticket->sellprice - $result['payed'];
```

4.6 Interaction viewpoint

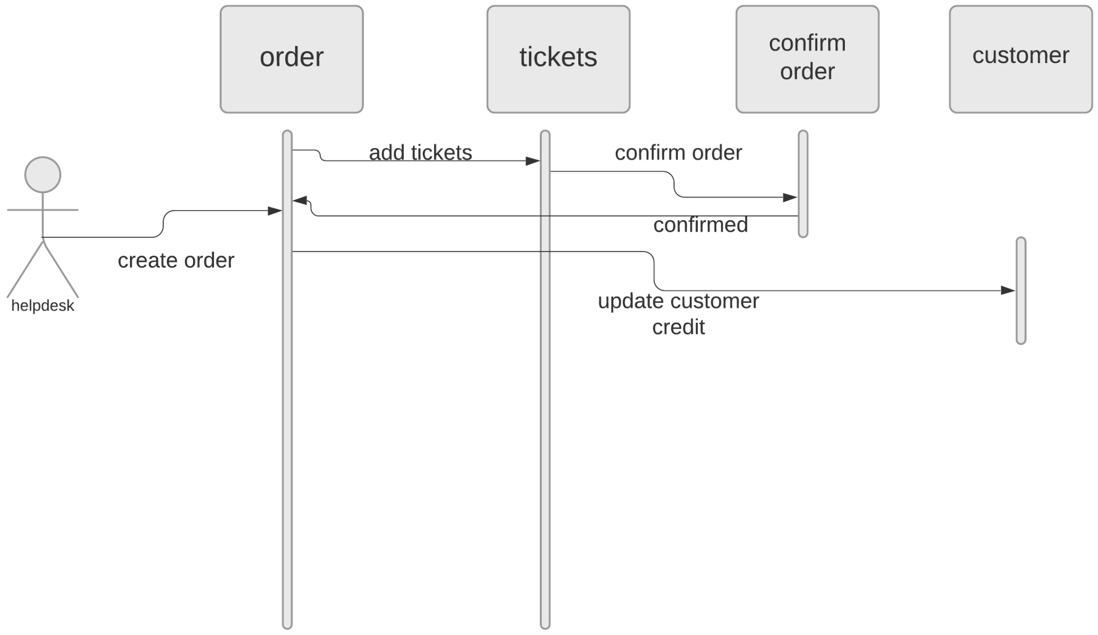


Figure 8: create order Sequence Diagram

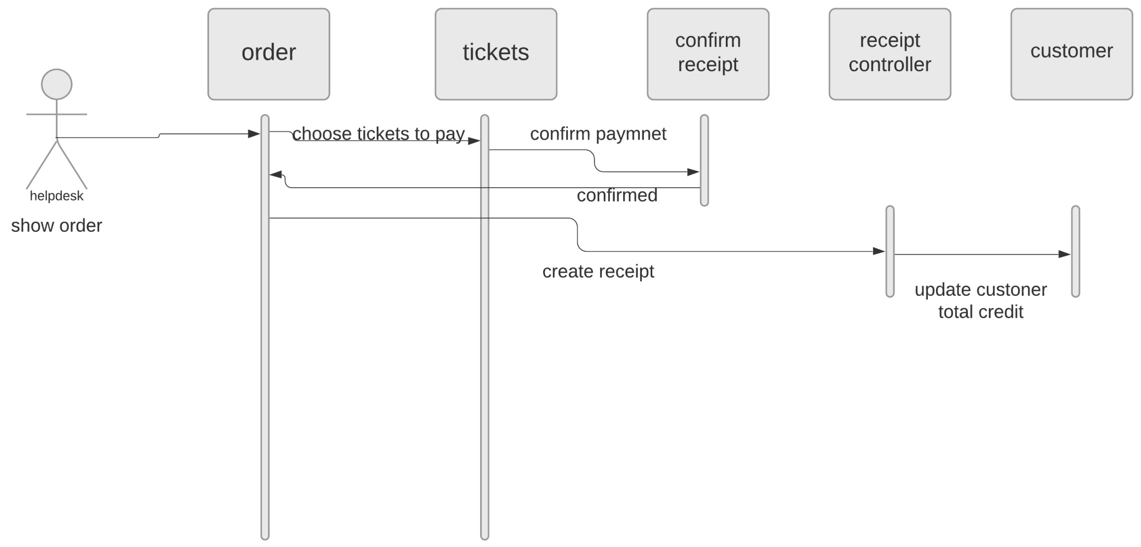


Figure 9: pay order Sequence Diagram

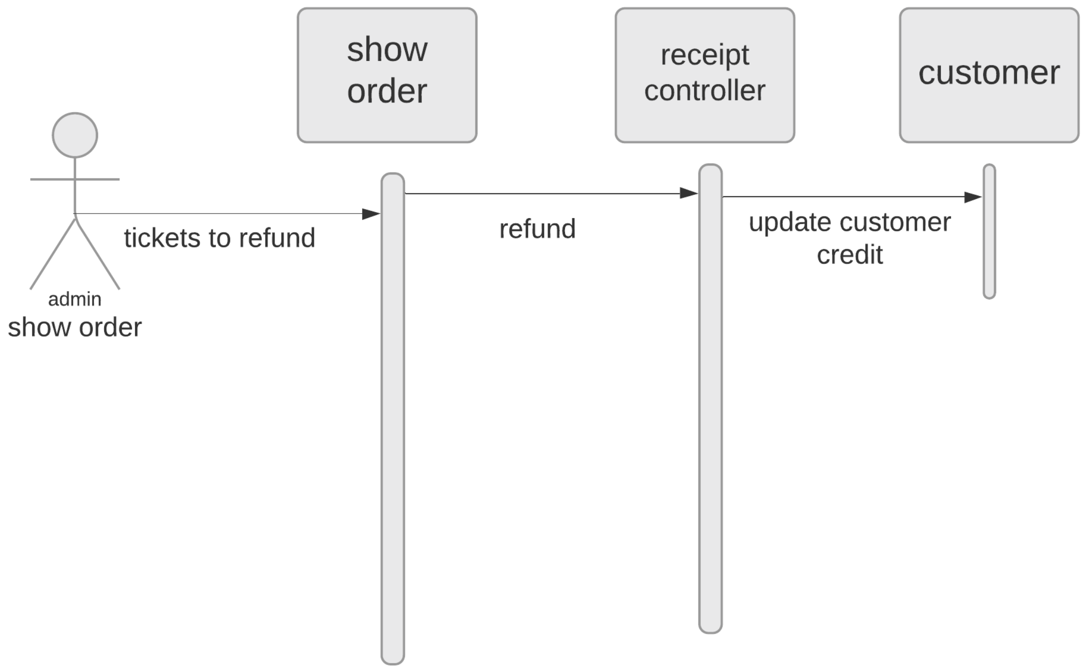


Figure 10: refund Sequence Diagram

4.7 Interface viewpoint

The Interface viewpoint provides programmers, and testers the means to know how to correctly use the provided services. This viewpoint consists of a set of interface specifications for each entity.
NOTE—User interfaces are addressed separately.

5 Data Design

5.1 Data Description

Information entered in the web forms are stored in database tables, linked by IDs if needed and processed through the code by viewing them in web HTML tables or editing them from the web forms.
The original format of the data are paper based and excel files.
Method used to capture the data into the system is web page forms.
The database expected to be maximum of 20 columns in the largest class of tickets.
The expected number of users is 4-7 on the system.
There is no specific code or format for entities ID definition.
By default Laravel tables contain date/time, they are represented in "YYYY-MM-DD HH:MM:SS" format.

5.2 Database design description

Our Data will be stored in a database using MySQL phpmyadmin. Some of the database tables are explained as follows
Users : Contains all user data required to make the user access the system.
Tickets : Contains all sold and refunded data tickets.
Receipts : Contains all receipts made on the system.

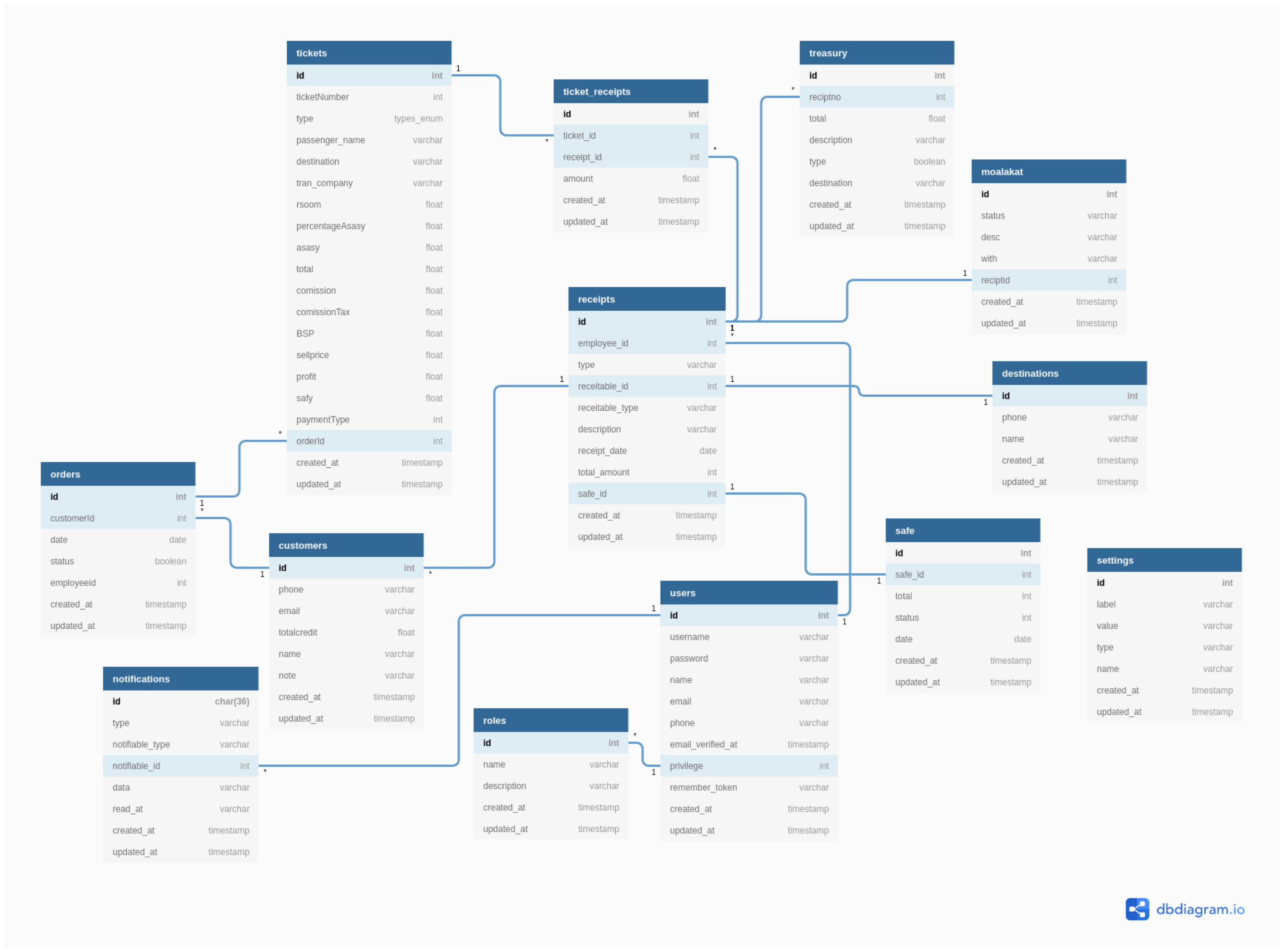


Figure 11: Database

6 Human Interface Design

6.1 User Interface

Users are divided into 3 roles (superadmin, admin and helpdesk). At first, the user has to login.
If he is a help desk, he will be taken to the home page which shows some statistics. Then in the navbar he will have 3 options orders, tickets and logout. He can view all orders and tickets, he can also create new order and refund tickets.
If he is an admin, he will be taken to the home page which shows some more detailed statistics. Then in the navbar he will have 6 options customers, orders, tickets, safe and logout. He can do everything the help desk can plus viewing all the customers profiles, viewing the safe (expenses and revenues) details and generating tickets or receipts reports.
If he is a superadmin he will have an additional privilege to edit some site settings.

6.2 Screen Images

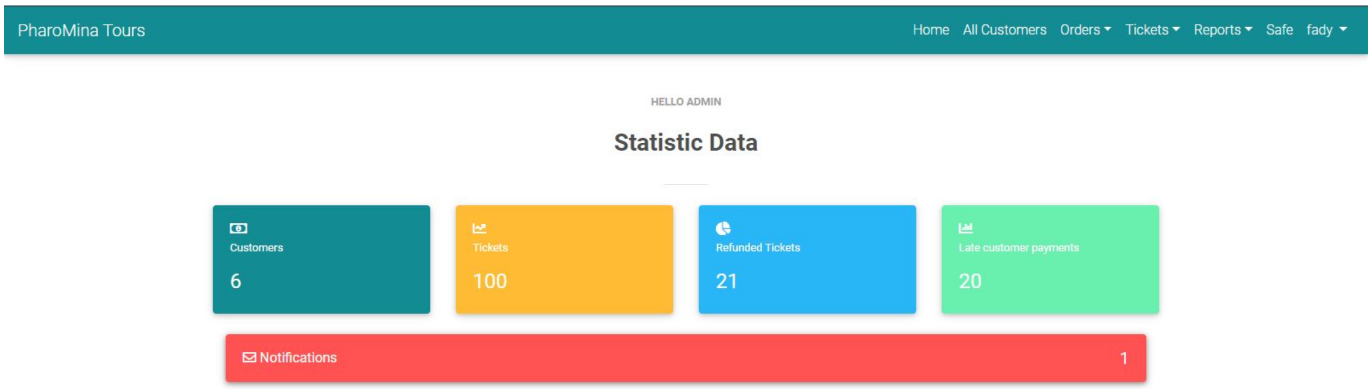
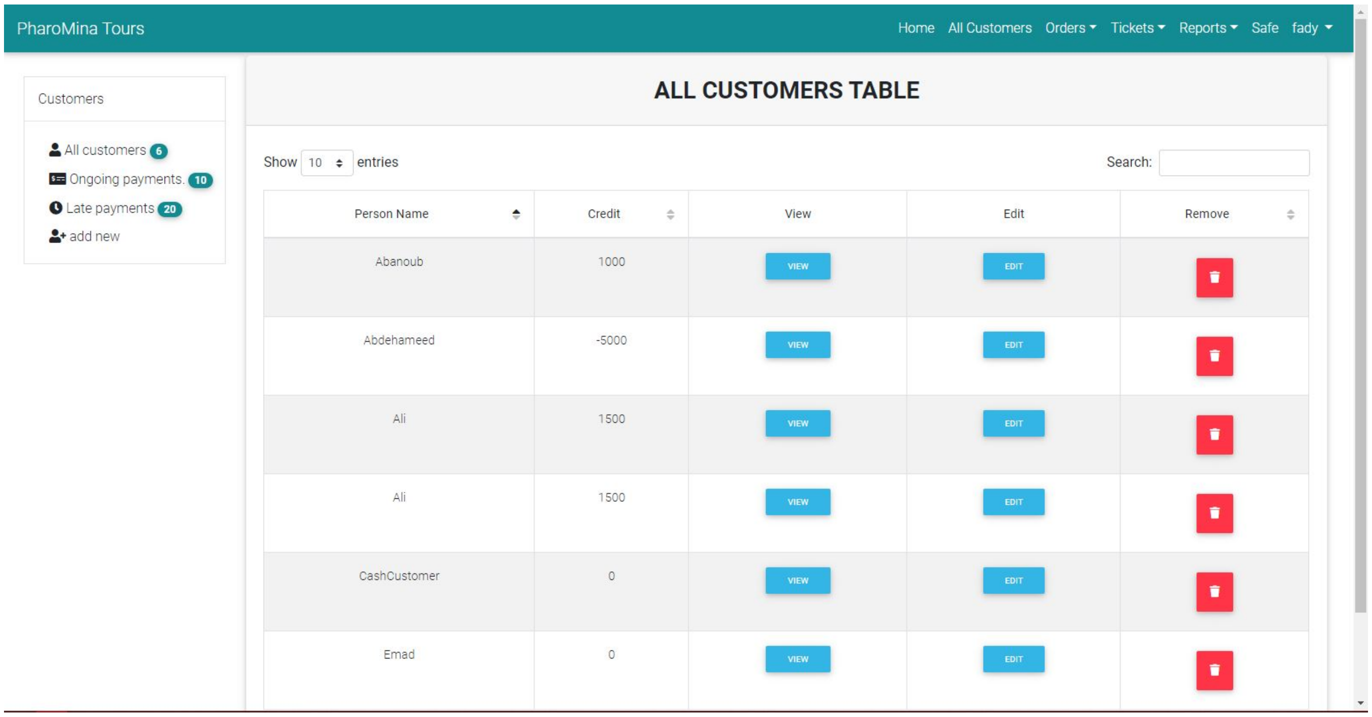


Figure 12: Home

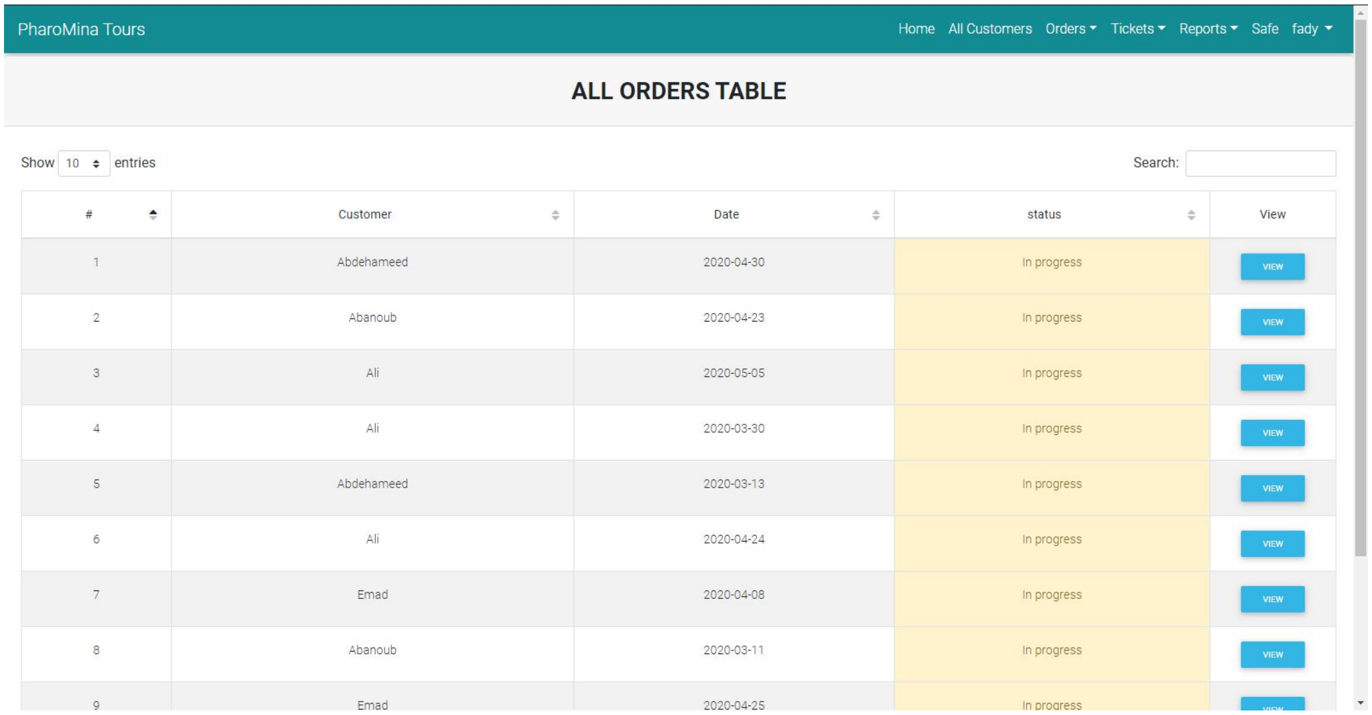


ALL CUSTOMERS TABLE

Show 10 entries Search:

Person Name	Credit	View	Edit	Remove
Abanoub	1000	VIEW	EDIT	
Abdehameed	-5000	VIEW	EDIT	
Ali	1500	VIEW	EDIT	
Ali	1500	VIEW	EDIT	
CashCustomer	0	VIEW	EDIT	
Emad	0	VIEW	EDIT	

Figure 13: All Customers

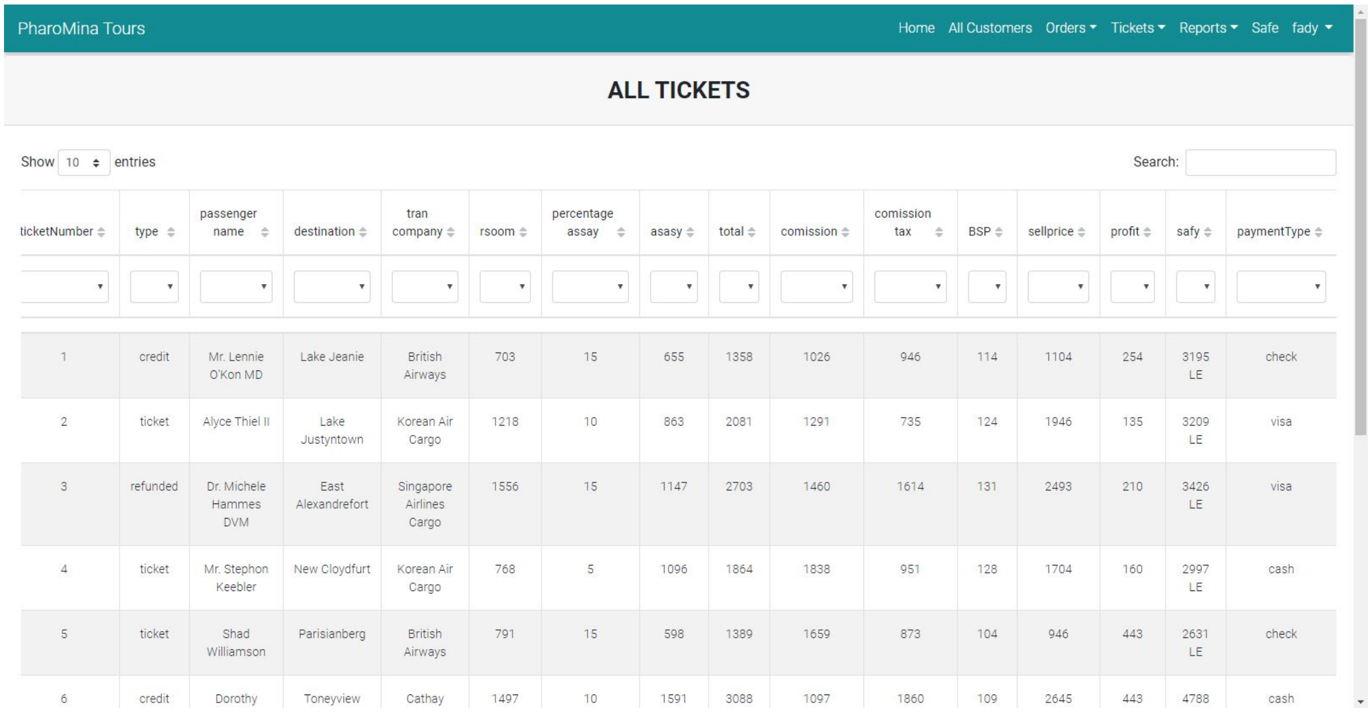


ALL ORDERS TABLE

Show 10 entries Search:

#	Customer	Date	status	View
1	Abdehameed	2020-04-30	In progress	VIEW
2	Abanoub	2020-04-23	In progress	VIEW
3	Ali	2020-05-05	In progress	VIEW
4	Ali	2020-03-30	In progress	VIEW
5	Abdehameed	2020-03-13	In progress	VIEW
6	Ali	2020-04-24	In progress	VIEW
7	Emad	2020-04-08	In progress	VIEW
8	Abanoub	2020-03-11	In progress	VIEW
9	Emad	2020-04-25	In progress	

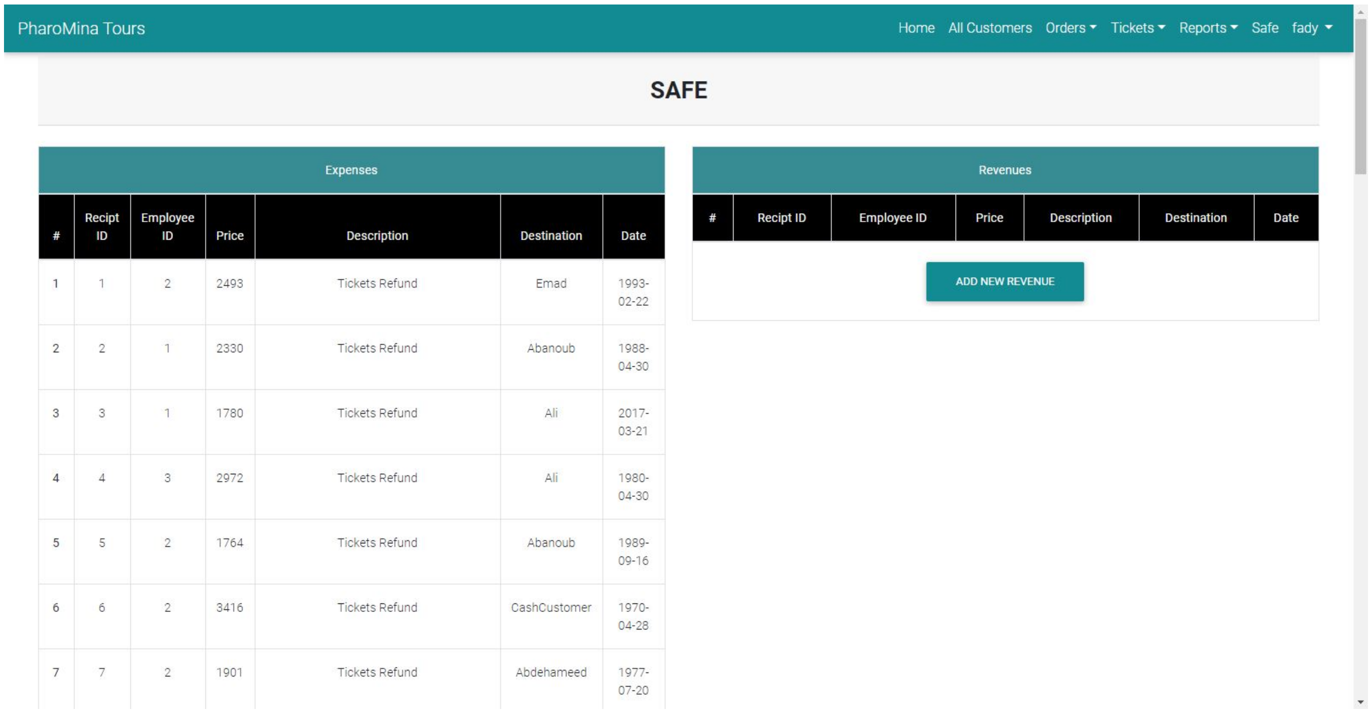
Figure 14: All Orders



ALL TICKETS

ticketNumber	type	passenger name	destination	tran company	rsoom	percentage assay	asasy	total	comission	comission tax	BSP	sellprice	profit	safy	paymentType
1	credit	Mr. Lennie O'Kon MD	Lake Jeanie	British Airways	703	15	655	1358	1026	946	114	1104	254	3195 LE	check
2	ticket	Alyce Thiel II	Lake Justyntown	Korean Air Cargo	1218	10	863	2081	1291	735	124	1946	135	3209 LE	visa
3	refunded	Dr. Michele Hammes DVM	East Alexandrefort	Singapore Airlines Cargo	1556	15	1147	2703	1460	1614	131	2493	210	3426 LE	visa
4	ticket	Mr. Stephon Keebler	New Cloydfurt	Korean Air Cargo	768	5	1096	1864	1838	951	128	1704	160	2997 LE	cash
5	ticket	Shad Williamson	Parisianberg	British Airways	791	15	598	1389	1659	873	104	946	443	2631 LE	check
6	credit	Dorothy	Toneyview	Cathay	1497	10	1591	3088	1097	1860	109	2645	443	4788	cash

Figure 15: All Tickets



SAFE

Expenses

#	Recipt ID	Employee ID	Price	Description	Destination	Date
1	1	2	2493	Tickets Refund	Emad	1993-02-22
2	2	1	2330	Tickets Refund	Abanoub	1988-04-30
3	3	1	1780	Tickets Refund	Ali	2017-03-21
4	4	3	2972	Tickets Refund	Ali	1980-04-30
5	5	2	1764	Tickets Refund	Abanoub	1989-09-16
6	6	2	3416	Tickets Refund	CashCustomer	1970-04-28
7	7	2	1901	Tickets Refund	Abdehameed	1977-07-20

Revenues

#	Recipt ID	Employee ID	Price	Description	Destination	Date

Figure 16: Safe

7 Requirements Matrix

Req. ID	Req. Type	Req. Description	Status
F00	Required	Customer Model - orders	Completed
F01	Required	Customer Model - receipts	Completed
F02	Required	Destination Model - receipts	Completed
F03	Required	Order Model - tickets	Completed
F03	Required	Order Model - customer	Completed
F05	Required	Order Model - ticketsAmount	Completed
F06	Required	Receipt Model - tickets	Completed
F07	Required	Receipt Model - safe	Completed
F08	Required	Role Model - users	Completed
F09	Required	Safe Model - receipts	Completed
F10	Required	Ticket Model - receipts	Completed
F11	Required	Ticket Model - orders	Completed
F12	Required	Ticket Model - calculate	Completed
F13	Required	User Model - role	Completed
F14	Required	Customer Controller - index	Completed
F15	Required	Customer Controller - ongoingpayments	Completed
F16	Required	Customer Controller - latepayments	Completed
F17	Required	Customer Controller - store	Completed
F18	Required	Customer Controller - show	Completed
F19	Required	Customer Controller - updatenote	Completed
F20	Required	Customer Controller - edit	Completed
F21	Required	Customer Controller - update	Completed
F22	Required	Home Controller - index	Completed
F23	Required	Mail Controller - Mail Admin	Completed
F24	Required	Mail Controller - Mail Helpdesk	Completed
F25	Required	Mail Controller - Mail	Completed
F26	Required	Notifications Controller - Notify	Completed
F27	Required	Notifications Controller - Mark All As Read	Completed
F28	Required	Notifications Controller - Unread	Completed
F28	Required	Notifications Controller - View All	Completed
F30	Required	Order Controller - index	Completed
F31	Required	Order Controller - create	Completed
F32	Required	Order Controller - store	Completed
F33	Required	Order Controller - show	Completed
F34	Required	Order Controller - confirmpayment	Completed
F35	Required	Order Controller - confirmview	Completed
F36	Required	Order Controller - destroy	Completed
F37	Required	Order Controller - print	Completed
F38	Required	Order Controller - payAll	Completed
F39	Required	Receipt Controller - store	Completed
F40	Required	Receipt Controller - storeAllOrder	Completed
F41	Required	Receipt Controller - refundTickets	Completed
F42	Required	Receipt Controller - print	Completed

F43	Required	Report Controller - getEndDate	Completed
F44	Required	Report Controller - tickets	Completed
F45	Required	Report Controller - printTickets	Completed
F46	Required	Report Controller - printReceipts	Completed
F47	Required	Report Controller - receipts	Completed
F48	Required	Report Controller - excelTickets	Completed
F49	Required	Report Controller - excelReceipts	Completed
F50	Required	Safe Controller - index	Completed
F51	Required	Safe Controller - store	Completed
F52	Required	Setting Controller - index	Completed
F53	Required	Setting Controller - update	Completed
F54	Required	Ticket Controller - index	Completed
F55	Required	Ticket Controller - create	Completed
F56	Required	Ticket Controller - orderticket	Completed
F57	Required	Ticket Controller - store	Completed
F58	Required	Ticket Controller - refundedTicketsShow	Completed
F59	Required	Ticket Controller - edit	Completed
F60	Required	Ticket Controller - update	Completed
F61	Required	Ticket Controller - destroy	Completed
F62	Required	Ticket Controller - checkprice	Completed
F63	Required	Ticket Controller - confirmReceipt	Completed

8 APPENDICES

The system is built on MVC software design pattern using Laravel[1] framework.
The system front end is responsively built using mdbootstrap[2].
The UML design pattern used is Facade[3] implemented by Laravel.

References

[1] *The PHP Framework for Web Artisans*. URL: `https://laravel.com`.

[2] *Material Design for Bootstrap 4 - the most popular and free UI KIT*. URL: `https://mdbootstrap.com`.

[3] *Facade - A class which provides a static-like interface to services inside the container*. URL: `https://laravel.com/docs/7.x/facades`.