CS353 DATABASE SYSTEMS

PROJECT DESIGN REPORT

GROUP 13

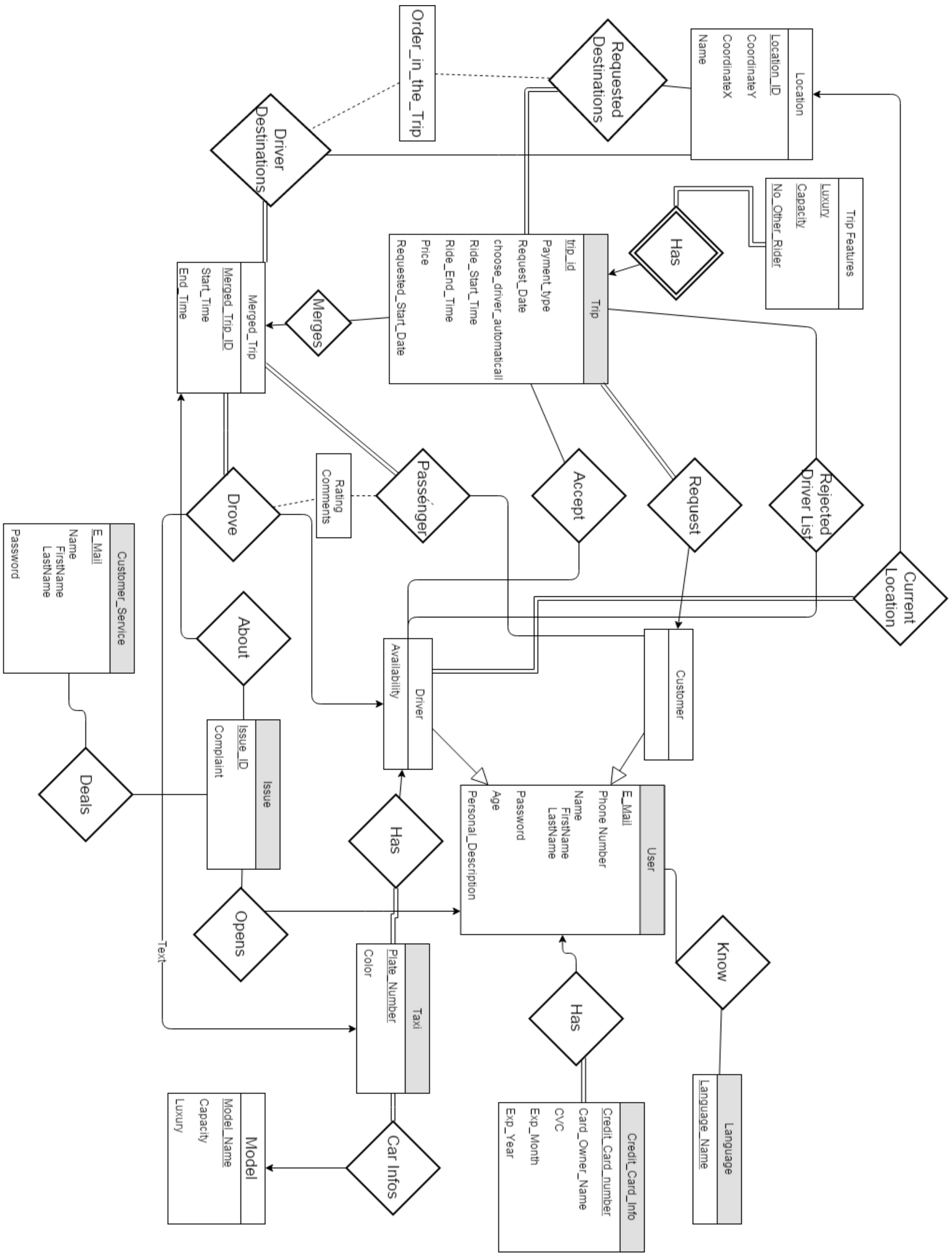
İbrahim Taha Aksu

Onur Kulaksızoğlu

Berke Soysal

Gökçe Şakir Özyurt

1 Revised E-R Diagram



2. Table Schemas

User (E_Mail, Phone_Number, FirstName, LastName, Personal_Info, Password, Age, Credit_Card_Number)

FK: Credit_Card_Number references Credit_Card_Info

CK: Phone_Number, E_Mail

PK: E_Mail

Composite Value: Name

Table Definition with domains and constraints:

```
Create Table User{
E_Mail varchar (320) NOT NULL PRIMARY KEY,
Phone_Number char(15) NOT NULL UNIQUE,
FirstName varchar (20) NOT NULL,
LastName varchar (40) NOT NULL,
Name AS
  (FirstName + " " + LastName ),
Personal_Info varchar(250),
Password varchar(32) NOT NULL CHECK (LENGTHB(Password) >= 4,
Age int(3,0) CHECK (Age>=18),
Credit_Card_Number char(19) REFERENCES Credit_Card_Info(Credit_Card_Number)
}
```

Language(<u>Language_Name</u>)

CK: Language_Name

PK: Language_Name

Table Definition with domains and constraints:

Create Table Language{

Language_Name varchar(24) NOT NULL PRIMARY KEY

}

Know(<u>E_Mail, Language_Name</u>)

CK: {E_Mail, Language_Name}

PK: {E_Mail, Language_Name}

FK: E_Mail REFERENCES User(E_Mail), Language_Name REFERENCES Language(Language_Name)

Table Definition with domains and constraints:

Create Table Know{

Language_Name varchar(24) NOT NULL REFERENCES Language(Language_Name),

E_Mail varchar (320) NOT NULL REFERENCES User(E_Mail),

CONSTRAINT PK_KNOW PRIMARY KEY (E_Mail, Language_Name)

}

Credit_Card_Info(<u>Credit_Card_Number</u>, Card_Owner_Name, Exp_Month, Exp_Year, CVC)

CK: Credit_Card_Number

PK: Credit_Card_Number

Table Definition with domains and constraints:

/* data will be encrypted, though currently we don’t know how to do.

And credit card part is just for a thought experiment, we won’t have transactions managed by Taxim in the implementation */

Create Table Credit_Card_Info

{

Credit_Card_Number char(19) NOT NULL PRIMARY KEY,

Card_Owner_Name   varchar(50) NOT NULL,

Exp_Month   numeric(2,0) NOT NULL,

Exp_Year   numeric(2,0) NOT NULL,

CVC   numeric(3,0)   NOT NULL,

CONSTRAINT CHK_VALID_THRU CHECK (Exp_Year>17 OR (Exp_Year=2017 AND Exp_Month>11))

/*There could be additional checks, but as mentioned above, these are just thought experiments*/

}

Customer(<u>E_Mail</u>)

CK: E_Mail

PK: E_Mail

FK: E_Mail REFERENCES User(E_Mail)

Table Definition with domains and constraints:

Create Table Customer{

E_Mail varchar (320) NOT NULL PRIMARY KEY REFERENCES User(E_Mail)

}

Driver(<u>E_Mail</u>, Current_Location, Availability)

CK: E_Mail

PK: E_Mail

FK: E_Mail REFERENCES User(E_Mail), Current_Location REFERENCES Location(Location_ID)

Table Definition with domains and constraints:

```
Create Table Driver{
E_Mail varchar (320) NOT NULL PRIMARY KEY REFERENCES User(E_Mail),
Current_Location int REFERENCES Location(Location_ID),
Availability BIT
}
```

Taxi(<u>Plate_Number</u>, Color, Driver_ID, Model_Name)

CK: Plate_Number

PK: Plate_Number

FK: Driver_ID references Driver(E_Mail), Model_Name references Model(Model_Name)

Table Definition with domains and constraints:

```
Create Table Taxi{
Plate_Number varchar(16) NOT NULL PRIMARY KEY,
Color varchar(16) NOT NULL,
Driver_ID varchar (320) NOT NULL REFERENCES Driver(E_Mail),
Model_Name varchar (40) NOT NULL REFERENCES Model(Model_Name)
}
```

Model(<u>Model_Name</u>, Capacity, Luxury)

CK: Model_Name

PK: Model_Name

Table Definition with domains and constraints:

Create Table Model{

Capacity int NOT NULL,

Luxury char(1) NOT NULL CHECK Luxury in (‘L’, ‘Q’, ‘M’),

/*Luxury, High in Quality, Moderate/*

Model_Name varchar (40) NOT NULL PRIVATE KEY

}

Trip(<u>trip_id</u>, Payment_Type, Request_Date, Choose_Driver_Automatically, Ride_Start_Time, Ride_End_Time, Price, Requested_Start_Date, requester_id)

FK: requester_id REFERENCES Customer(E_Mail)

CK: trip_id

PK: trip_id

Table Definition with domains and constraints:

Create Table Trip{

trip_id int NOT NULL PRIVATE KEY,

Payment_Type char(1) NOT NULL CHECK Payment_Type in (‘R’, ‘A’),

/*Unluckily both credit and cash start with C, so we use their second syllables/*

Request_Date datetime NOT NULL,

Choose_Driver_Automatically BIT NOT NULL,

Ride_Start_Time datetime CHECK(Ride_End_Time>Ride_Start_Time>=Request_Date),

Ride_End_Time datetime,

Price numeric(5,2),

Requested_Start_Date NOT NULL CHECK (Requested_Start_Date >=Request_Date),

requester_id   varchar (320) NOT NULL REFERENCES Customer(E_Mail)

}

Trip_Features(<u>Luxury, Capacity, No_Other_Rider, trip_id</u>)

FK: trip_id REFERENCES Trip(trip_id)

CK: {Luxury, Capacity, No_Other_Rider, trip_id}

PK: {Luxury, Capacity, No_Other_Rider, trip_id}

Table Definition with domains and constraints:

Create Table Trip_Features{

trip_id int NOT NULL REFERENCES Trip(trip_id),

Capacity int NOT NULL,

Luxury char(1) NOT NULL CHECK Luxury in (‘L’, ‘Q’, ‘M’),

/*Luxury, High in Quality, Moderate/*

No_Other_Rider BIT NOT NULL,

CONSTRAINT PK_Trip_Features PRIMARY KEY(Luxury_Car, Capacity, No_Other_Rider, trip_id)

}

Rejected_Driver_List(<u>E_Mail, trip_id</u>)

FK: E_Mail REFERENCES Driver(E_Mail), trip_id REFERENCES Trip(trip_id)

CK: {E_Mail, trip_id}

PK: {E_Mail, trip_id}

Table Definition with domains and constraints:

Create Table Rejected_Driver_List{

E_Mail varchar (320) NOT NULL REFERENCES Driver(E_Mail),

Trip_id int NOT NULL REFERENCES Trip(trip_id),

CONSTRAINT PK_RDL PRIMARY KEY(E_Mail, trip_id)

}

Accept(trip_id, E_Mail)

FK: E_Mail REFERENCES Driver(E_Mail), trip_id REFERENCES Trip(trip_id)

CK: {E_Mail, trip_id}

PK: {E_Mail, trip_id}

Table Definition with domains and constraints:

```
Create Table Accept{
E_Mail varchar (320) NOT NULL REFERENCES Driver(E_Mail),
Trip_id int NOT NULL REFERENCES Trip(trip_id),
CONSTRAINT PK_RDL PRIMARY KEY(E_Mail, trip_id)
}
```

Requested_Destinations(trip_id, Location_ID, Order_in_the_Trip)

FK: trip_id REFERENCES Trip(trip_id), Location_ID REFERENCES Location(Location_ID)

CK: {trip_id, Location_ID}

PK: {trip_id, Location_ID}

Table Definition with domains and constraints:

```
Create Table Requested_Destinations{
trip_id int NOT NULL REFERENCES Trip(trip_id),
Location_ID int NOT NULL REFERENCES Location(Location_ID),
Order_in_the_Trip int NOT NULL
CONSTRAINT PK_R_DESTINATION PRIMARY KEY(trip_id, Location_ID)
}
```

Merged_Trip(Merged_Trip_ID, Start_Time, End_Time, E_Mail, Plate_Number, Rating, Comment)

CK: Merged_Trip_ID, {Start_Time, E_Mail}, {End_Time, E_Mail}, {Start_Time, Plate_Number}, {End_Time, Plate_Number}

/*this looks kinda wrong, but it applies to 3NF, and since these are time related and not defined on creation time constraints are not in this definition /*

PK: Merged_Trip_ID

FK: E_Mail REFERENCES Driver(E_Mail), Plate_Number REFERENCES Taxi(Plate_Number)

Table Definition with domains and constraints:

```
Create Table Merged_Trip{
Merged_Trip_ID int NOT NULL PRIVATE KEY,
Start_Time datetime,
End_Time datetime,
E_Mail varchar (320) NOT NULL REFERENCES Driver(E_Mail),
Plate_Number varchar(16) NOT NULL REFERENCES Taxi(Plate_Number)
Rating numeric(2,1),
Comment varchar(250)
}
```

Merges(trip_id, Merged_Trip_ID)

FK: trip_id REFERENCES Trip(trip_id), Merged_Trip_ID REFERENCES Merged_Trip(Merged_Trip_ID)

CK: {trip_id, Merged_Trip_ID}

PK: CK: {trip_id, Merged_Trip_ID}

Table Definition with domains and constraints:

Create Table Merges{

trip_id int NOT NULL REFERENCES Trip(trip_id),

Merged_Trip_ID NOT NULL REFERENCES Merged_Trip(Merged_Trip_ID),

CONSTRAINT PK_MERGES PRIMARY KEY(trip_id, Merged_Trip_ID)

}

Driver_Destinations(<u>Merged_Trip_ID, Location_ID</u>, Order_in_the_Trip)

FK: Merged_Trip_ID REFERENCES Merged_Trip(Merged_Trip_ID), Location_ID REFERENCES Location(Location_ID)

CK: {trip_id, Location_ID}

PK: {trip_id, Location_ID}

Table Definition with domains and constraints:

Create Table Driver_Destinations{

Merged_Trip_ID int NOT NULL REFERENCES Merged_Trip(Merged_Trip_ID),

Location_ID int NOT NULL REFERENCES Location(Location_ID),

Order_in_the_Trip int NOT NULL

CONSTRAINT PK_D_DESTINATION PRIMARY KEY(trip_id, Location_ID)

}

Location(<u>Location_ID</u>, CoordinateY, CoordinateX, Name)

CK: {CoordinateX, CoordinateY}, Location_ID

PK: Location_ID

Table Definition with domains and constraints:

Create Table Location{

Location_ID int NOT NULL PRIVATE KEY,

CoordinateY int NOT NULL,

CoordinateX int NOT NULL,

CONSTRAINT COORDINATES UNIQUE (CoordinateY, CoordinateX),

Name varchar(100) NOT NULL

}

Passenger(E_Mail, Merged_Trip_ID, Rating, Comment)

CK: {E_Mail, Merged_Trip_ID}

PK: {E_Mail, Merged_Trip_ID}

FK: E_Mail REFERENCES Customer(E_Mail), Merged_Trip_ID REFERENCES Merged_Trip(Merged_Trip_ID)

Table Definition with domains and constraints:

Create Table Passenger{

E_Mail varchar (320) NOT NULL REFERENCES Customer(E_Mail),

Merged_Trip_ID int NOT NULL REFERENCES Merged_Trip(Merged_Trip_ID),

Rating numeric(2,1),

Comment varchar(250),

CONSTRAINT PK_PASSENGER PRIVATE KEY (E_Mail, Merged_Trip_ID)

}

Customer_Service(E_Mail, FirstName, LastName, Password)

CK: E_Mail

PK: E_Mail

Composite Value: Name

Table Definition with domains and constraints:

```
Create Table Customer_Service{

E_Mail varchar (320) NOT NULL PRIVATE KEY,

FirstName varchar (20) NOT NULL,

LastName varchar (40) NOT NULL,

Name AS

  (FirstName + " " + LastName ),

Password varchar(32) NOT NULL CHECK (LENGTHB(Password) >= 4,

}
```

Issue(Issue_ID, Complaint, Complainer_E_Mail, Merged_Trip_ID)

FK: Complainer_E_Mail REFERENCES User(E_Mail), Merged_Trip_ID REFERENCES Merged_Trip(Merged_Trip_ID)

CK: Issue_ID, {Complainer_E_Mail, Merged_Trip_ID}

/*A user can’t complain about a trip twice/*

PK: Issue_ID

Table Definition with domains and constraints:

```
Create Table Issue{

Issue_ID int NOT NULL PRIVATE KEY,

Complaint varchar(1500) NOT NULL,

Complainer_E_Mail varchar (320) NOT NULL  REFERENCES User(E_Mail),

Merged_Trip_ID int NOT NULL REFERENCES Merged_Trip(Merged_Trip_ID),

CONSTRAINT ISSUE_SPAM UNIQUE (Complainer_E_Mail, Merged_Trip_ID)

}
```

Deals(Issue_ID, CS_E_Mail)

FK: Issue_ID REFERENCES Issue(Issue_ID), CS_E_Mail REFERENCES Customer_Service(E_Mail)

CK: Issue_ID, CS_E_Mail

PK: CK: Issue_ID, CS_E_Mail

Table Definition with domains and constraints:

```
Create Table Deals{
Issue_ID int NOT NULL REFERENCES Issue(Issue_ID),
CS_E_Mail varchar (320) NOT NULL REFERENCES Customer_Service(E_Mail),
Constraint PK_DEALS PRIVATE KEY (Issue_ID, CS_E_Mail)
}
```

3. Functional Components

3.1 Use Cases:

Taxim web/mobile application provides a communication service for drivers and passengers. There are two options that a passenger can make. First, he can accept to ride with other customers. Then, the passenger will plan a route.Then, the Taxim system will search for another trip that coincides with the passenger’s trip. If that is the case, and the other passenger has accepted travelling with stranger passengers, then the passenger will be assigned to this ride automatically. This type of ride will be cheaper than the other option below.

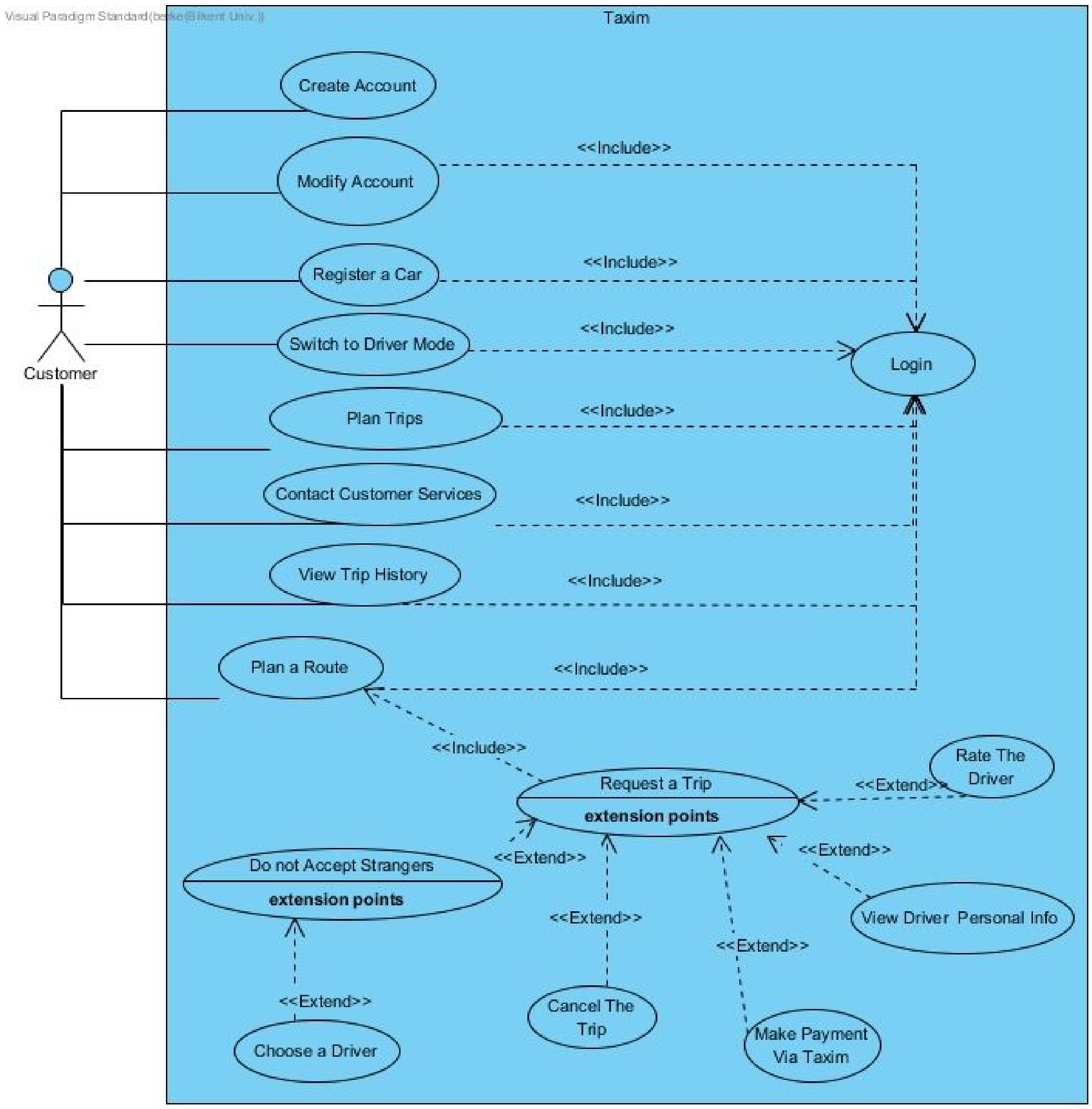
The second option is, if the passenger does not accept stranger passengers, then he/she will make a request to system. The active drivers will be notified, and the drivers may accept

or decline the request. Among drivers that accepted the request, the passenger will choose a driver. This type of ride will be more expensive.
Passengers can make his payment via Taxim, if the driver has accepted that option.

Drivers can choose passengers that make requests. They can ride multiple passengers, or ride single passengers, based on passengers' and their choice.

Customer Service is another user for Taxim. They can view detailed information about completed trips, or ongoing trips, or can review a driver's trip history. They can view drivers' car information. If there is a complained trip, then they can view the personal information about customer and driver. They can communicate with complainer users.

3.1.1 Customer Use Cases:



Login: Customer can log in to Taxim with his username and password in order to request a trip. Login is required for all operations.

Create account: Customer (driver or passenger) can create an account with specifying these information:

Name, e-mail, phone number, password.

This information is mandatory, they can add age and personal description too.

Register a car: Customers can register their car to be a driver. They should specify these information. A customer then can change to driver mode, and become a driver.:

Car model, Car Capacity, Car Luxury, Plate Number and Color.

Switch to Driver Mode: Customers can switch to driver mode, and look for customers using "Switch to Driver" option. They need to have a car to be registered for using this option.

Contact Customer Services: If customer has a complain about his trip etc., he can contact to customer services, and get help.

Plan Trips: User can make a reservation for a later time. Then the same procedures will apl

View Trip History: Users can view their trip history, e.g duration of trips, rating of the drivers,

Location distance.

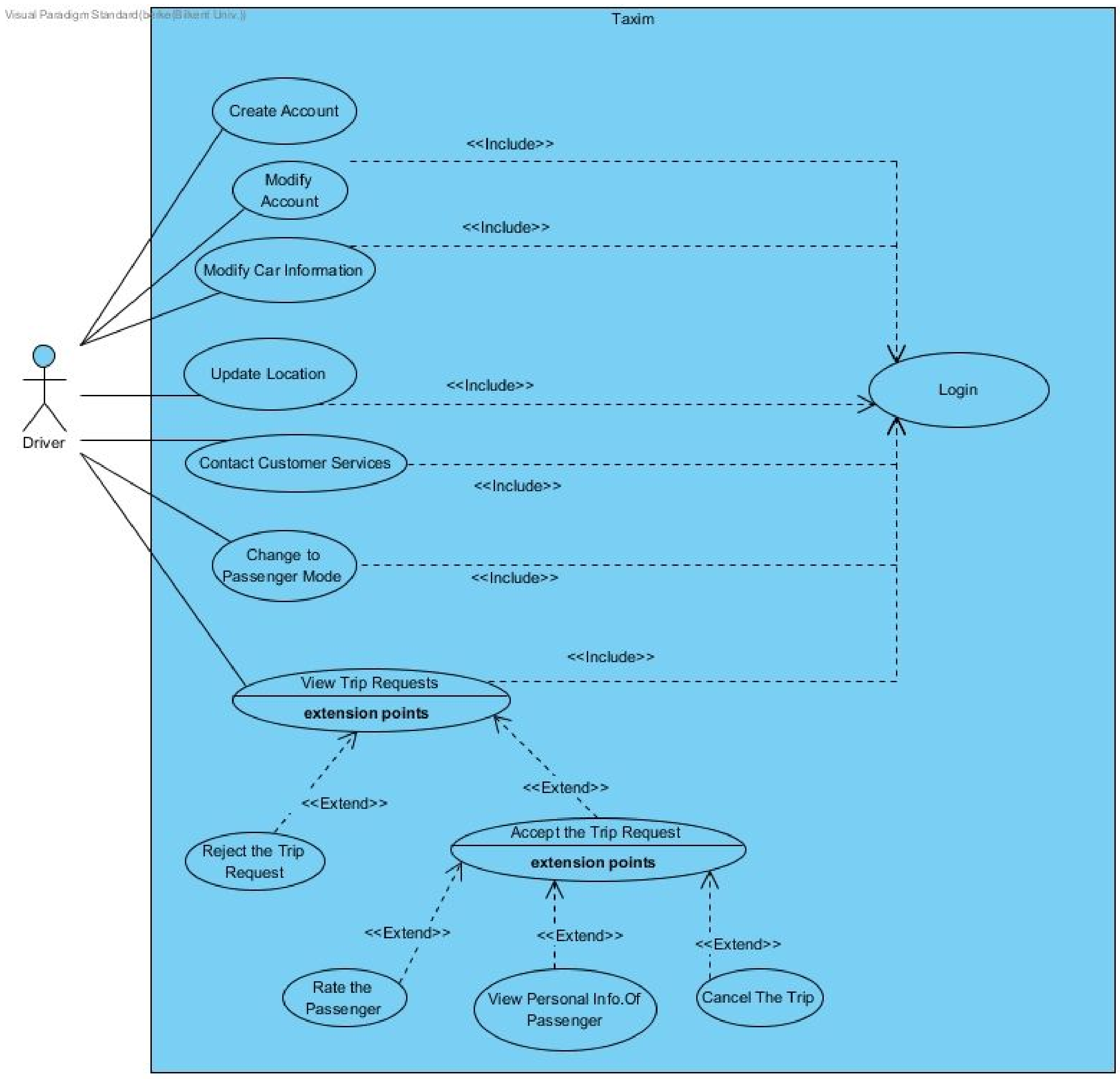
Plan a Route: Users can plan a route using the Taxim system. The route can have multiple or single destination points.

Request a Trip: After planning a route, user can request a trip, then the active drivers will be notified

Do not accept stranger Passengers: The passenger may choose to ride without stranger passengers, then the drivers will be notified about the trip request, and the drivers will accept or reject the trip.

Choose a Driver: Among the drivers that accepted the request, the passenger will choose a rider, and the driver will be notified about it.

3.1.2 Driver Use Cases:



Log in: Drivers should log in to Taxim, before they do anything on Taxim.

Create Account: The driver can create an account with specifying these personal information:

Name, e-mail, phone number, password, age.

This information is mandatory, they can add personal description too.

They should also specify these car information:

Car model, Car Capacity, Car Luxury, Plate Number and Color.

Modify account: Drivers can modify their personal information: their e-mail, phone number etc.

Modify car information: Drivers can modify their can information: car model, car capacity etc.

Update Location: Driver can update their current location, by entering manually, or using a GPS.

Contact Customer Services: If drivers have a complain about their passenger etc., they can contact to customer services, and get help.

Change to Passenger Mode: Driver can change to passenger mode and request a trip.

View Trip Requests: Driver can take requests while he’s active o Taxim, and select a passenger among them. He can filter the requests by passenger rating, distance etc.

Reject Trip Request: Driver can reject a trip request or ignore it. Passenger will be informed if the request is rejected by the driver.

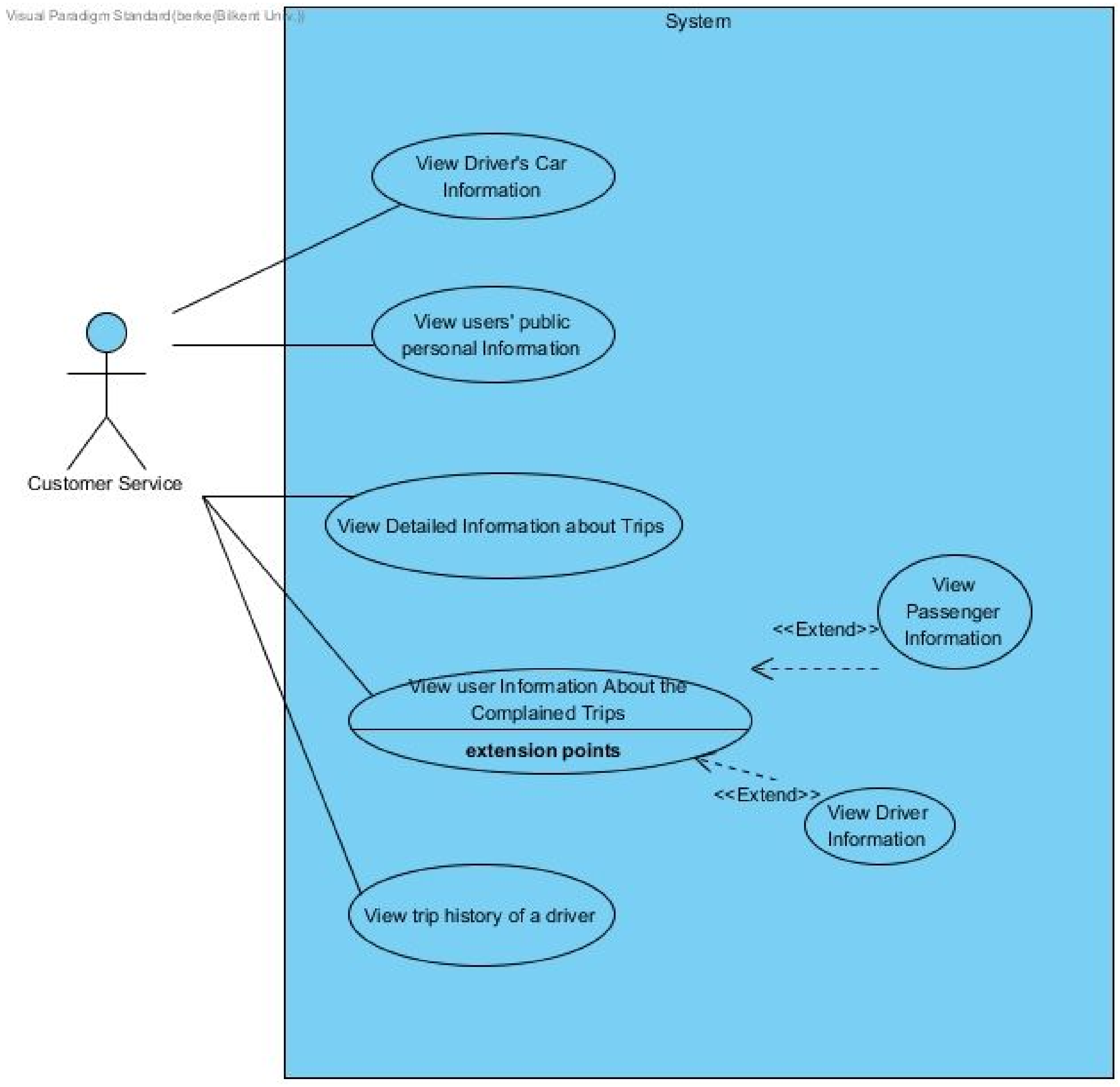
Accept Trip Request: Driver can accept the trip request. Then the passenger will be informed.

Rate The Passenger: Based on the experience on specific trip, driver can rate the passenger.

View personal Information of Passenger: Driver can view personal information of the passenger:phone,name etc.

Cancel the Trip: The driver still has right to cancel the trip, after he/she accepted the request.

3.1.2 Customer Service Use Cases:


Visual Paradigm Standard(berke(Bilkent Univ.))
System
View Driver's Car Information
View users' public personal Information
Customer Service
View Detailed Information about Trips
View Passenger Information
<<Extend>>
View user Information About the Complained Trips
extension points
<<Extend>>
View Driver Information
View trip history of a driver

If customer has a complain about his trip etc., he can contact to customer services, and get help.

View Driver’s Car Information: CS(Customer Service) cab view any drivers’ car information: car model, plate etc.

View User’s Public Personal Information: CS can view any users’s public personal information: name, age etc.

View Detailed Information About Complained Trips: CS can view a detailed information about a complained trip. Trip duration, Distance etc.

View Passenger Information: CS can view a passenger’s personal information (his/her trip history, private information included), if the passenger has been involved in a complained trip.

View Driver Information:CS can view a driver’s personal information (his/her trip history, private information included), if the passenger has been involved in a complained trip.

View Trip History of a Driver: CS can view trip history of a driver, and CS can make his search with filters, e.g. last 2 week, trips with more than 3 km.

3.2 Algorithms:

3.2.1 Single Passenger Destination Algorithm:

The driver will be assigned to a trip routed by Taxim. If the passenger does not accept stranger driver, the system will choose the most direct route. The route will not have any zigzags.

Example figure: The passenger will go C from A, the most direct path is selected as route and the driver is assigned with it.

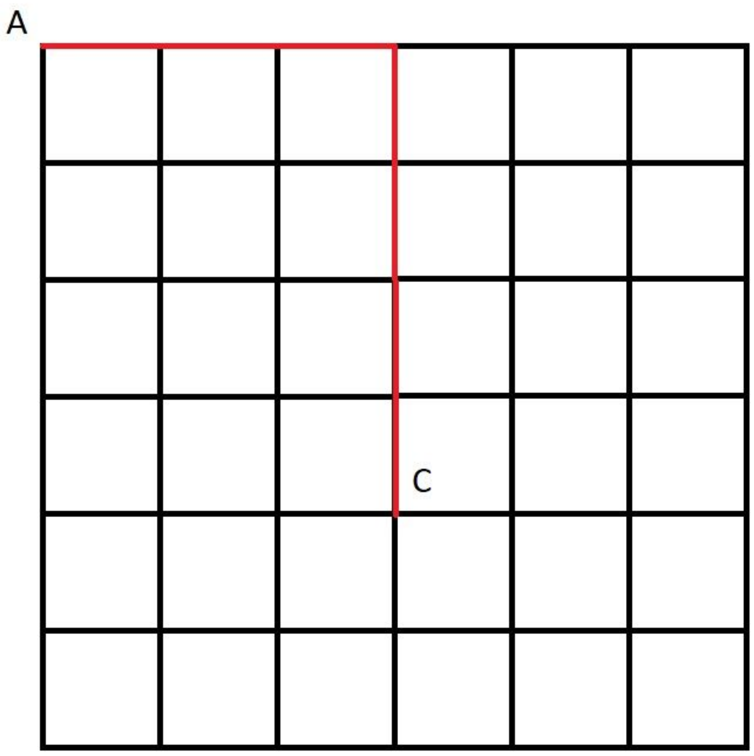


Figure 1: The most direct route will be selected if single passenger option is used.

3.2.2 Multiple Passenger Destination Algorithm:

If the passenger is accepted multiple passenger option, the most “indirect route” will be assigned the driver. So that the probability of getting another passenger, who has a route that coincides with driver's route, will be increased. If driver rides a passenger with that option from A to B, the route of the trip will be like the below figure.

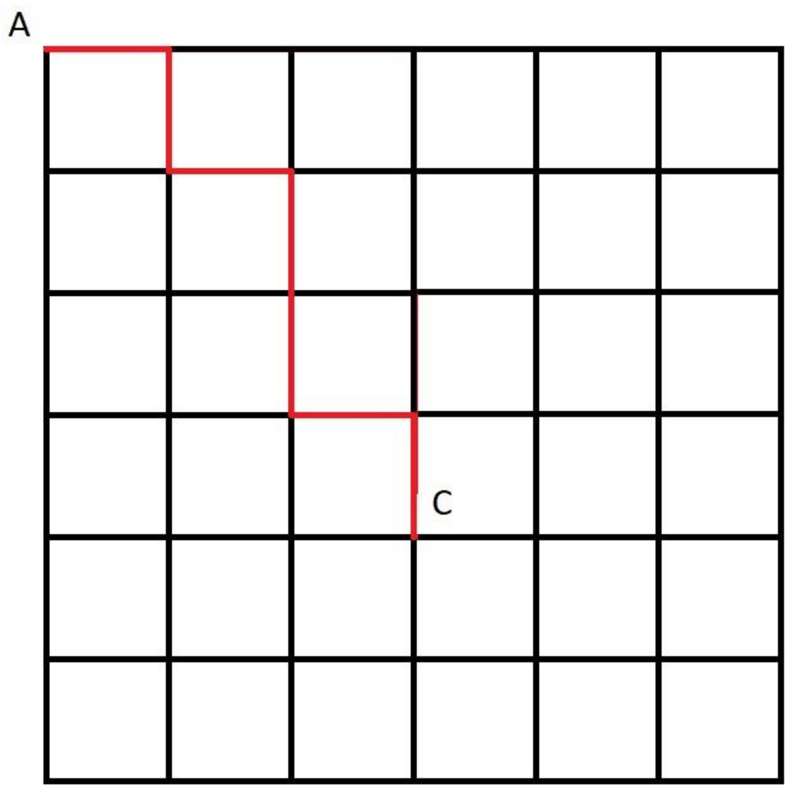


Figure 2: Zigzags will occur in route if there is multiple passenger option

This will be advantageous to the driver. Driver can take another driver, but the distance of the trip of the passenger on the vehicle should not be increased. For example in the figure below, the driver is notified when he is at point B, about a passenger at is point D, that wants to go point E, and the driver accepted it, a new route will be created that includes B, and goes to C with zigzags again.

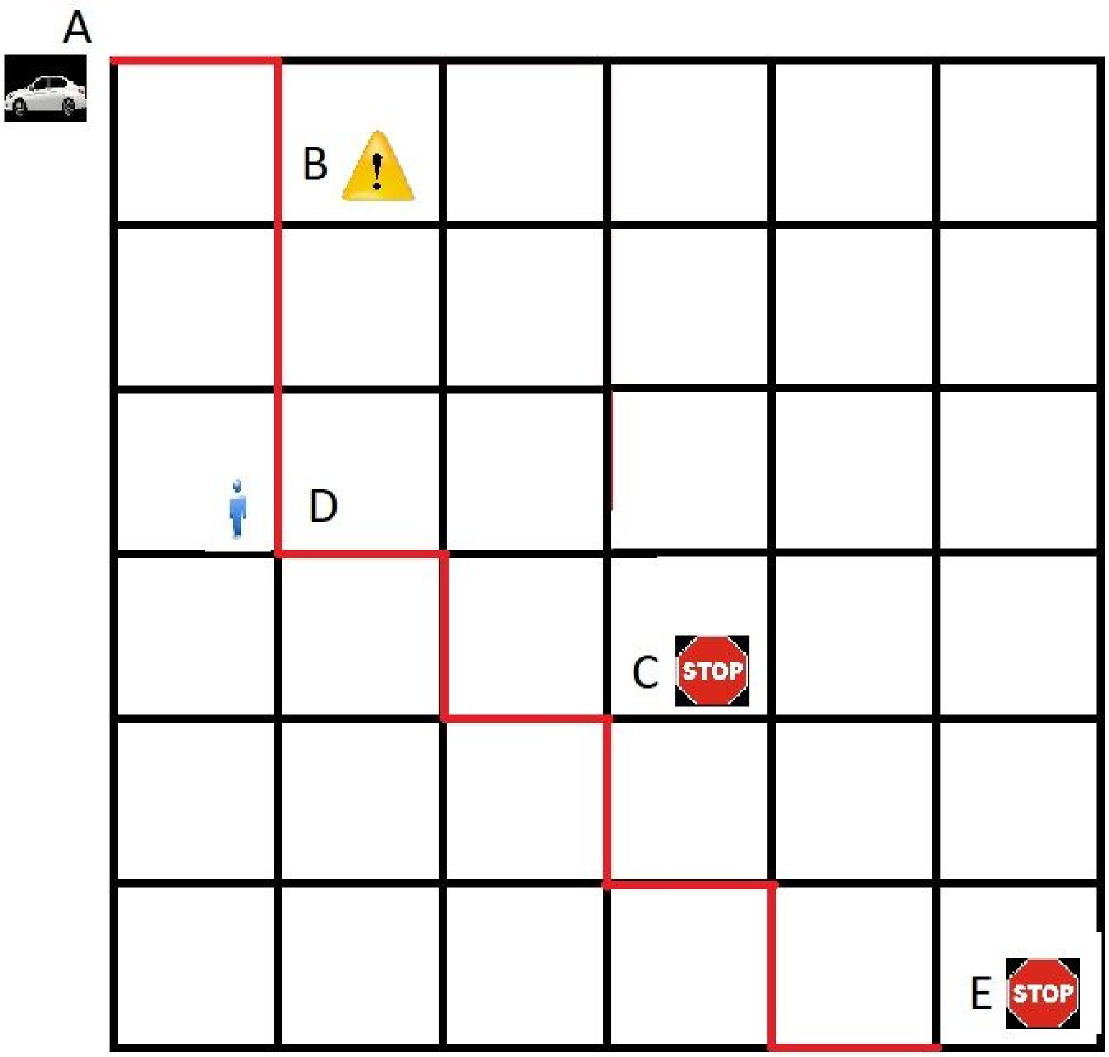


Figure 3: Driver is notified at Point B, about the Passenger at D, and the drive accepted the request, a direct route from B to D is routed, and driver continues to C, then E with zigzags.

Note that the driver can’t accept a passenger that increases the distance of the first passenger, in below figure, the driver can’t accept the passenger at B.

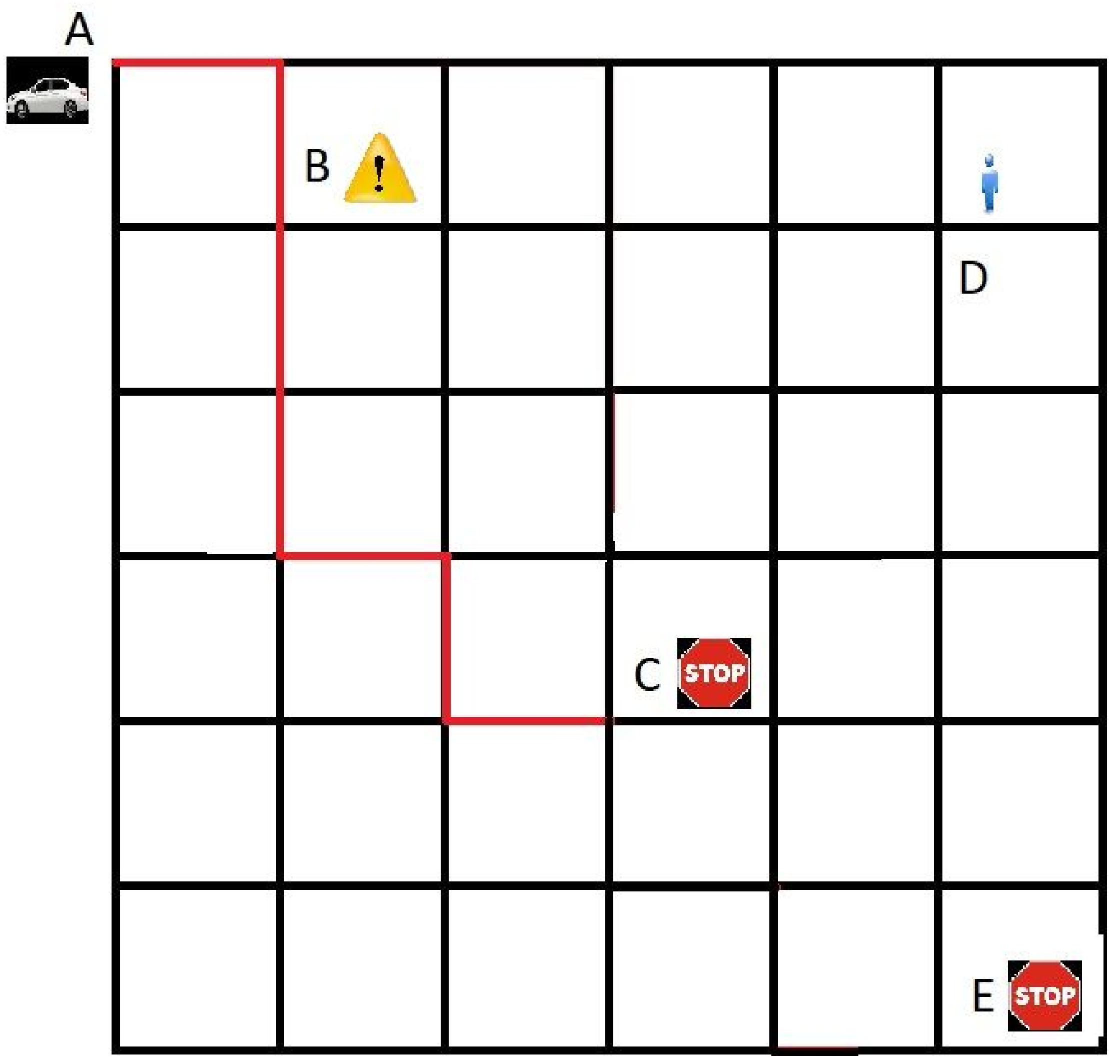


Figure 4: The driver can’t accept the passenger that he is notified at point B, because taking him makes the trip longer for the passenger at vehicle.

Driver can’t accept a passenger that has requested a trip that is, either of the drivers trip, can’t be inside of shortest path of the passenger at the seat. That means, if one of the passengers will experience a longer trip than his shortest path, then the driver can’t accept it.

In the below example, the user can’t accept the passenger goes from point D to E, because one of the passengers will have to experience a longer trip than shortest path if he takes him.

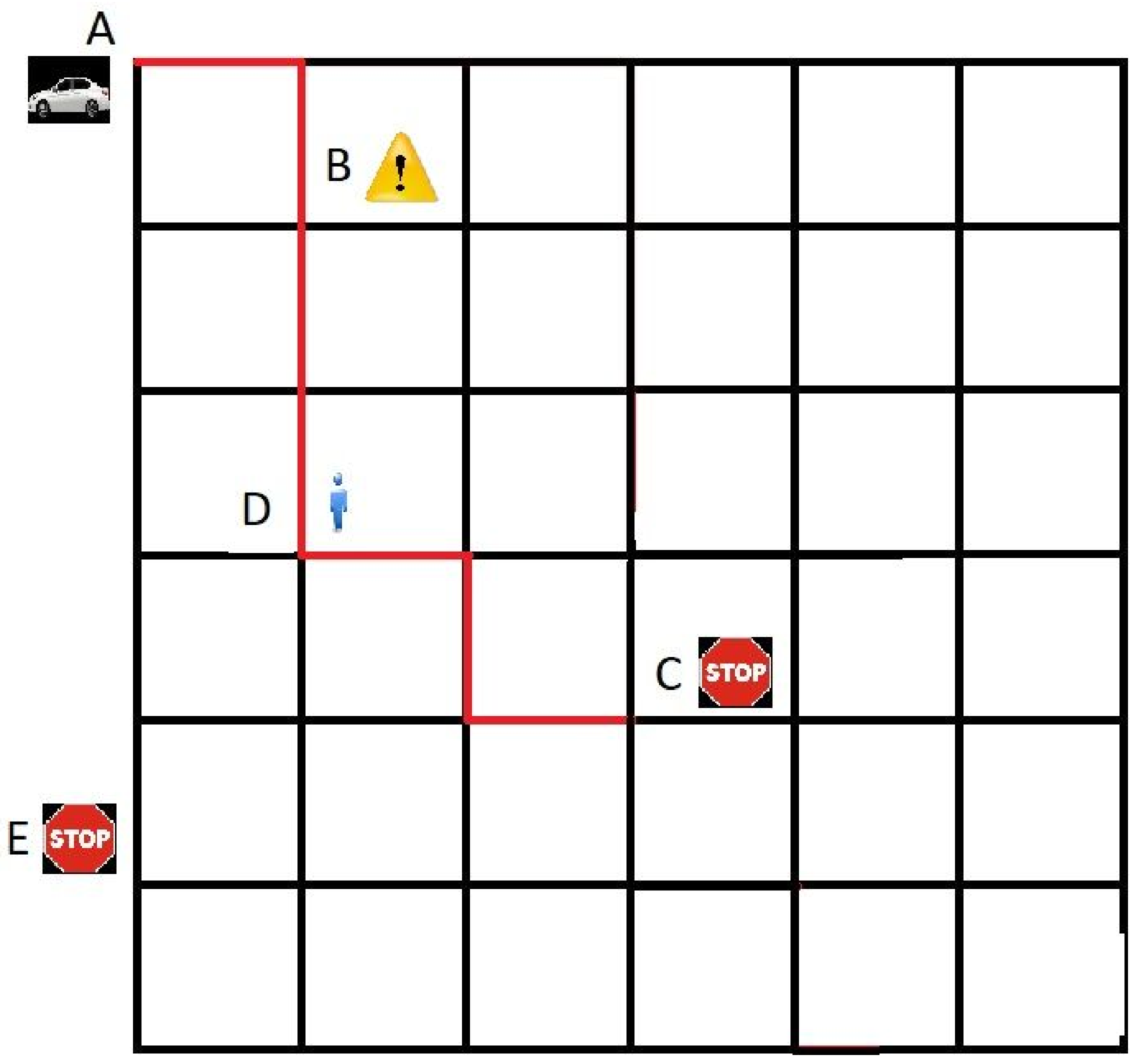


Figure 5: Driver can’t take the passenger at point D that goes to E.

Passengers can select multiple destinations. Than the routes are created again with zigzags but including the stop points.

Driver can’t accept more than the car capacity.

3.2.3 Price Algorithm:

The price will be calculated using: distance, single/multiple user option, total passengers infos, and the quality of the car.

The driver’s earnings will be increased nonlinearly but proportionally by the number of the passengers he accepted e.g. when going from A to B, if he takes multiple passengers, each passenger will pay less than single passenger fee, but also the driver will get more than bringing a single passenger from A to B. And if a rider chooses to allow multiple passengers option, even if there are no other riders chooses to share this trip, the rider pays less in the end to encourage this behavior.

Price per kilometer by a rider who chooses to share his trip is calculated as:
Default Price Per Kilometer * 0.9 / sqrt(Number of Riders Currently in the trip) * Luxury Coefficient

Luxury Coefficient and the Default Price Per Kilometer are not defined yet, and are only used to illustrate a simple part of our algorithm

An additional algorithmic feature related to this will be that, people who choose to share their trip will be directed to the taxis who has more capacity than the required request, so incase of a trip request on the road, another driver group can fit in.

4. User Interface Design and Corresponding SQLs

We have 2 different user types in our system; **Customer and Taxi Driver.**

4.1 Register Page

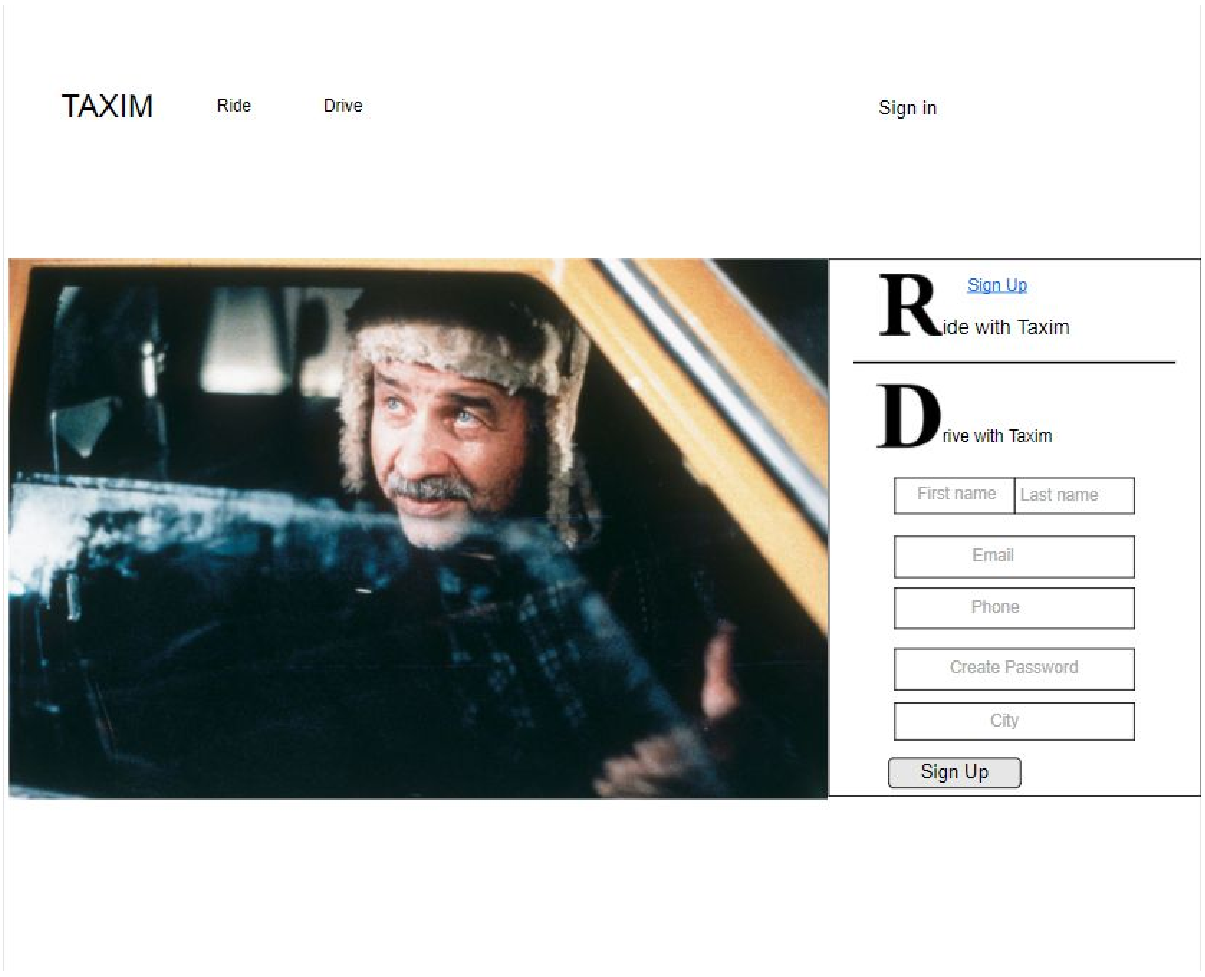
4.1.1 Customer Register Page

Process: Customers will be able to register with their personal information.

Input: @first_name, @last_name, @phone_no, @e-mail,@password

SQL: INSERT INTO User(first_name,last_name,phone_no,e_mail,password) VALUES (@first_name, @last_name, @phone_no, @e-mail,@password);

4.1.2 Driver Register Page



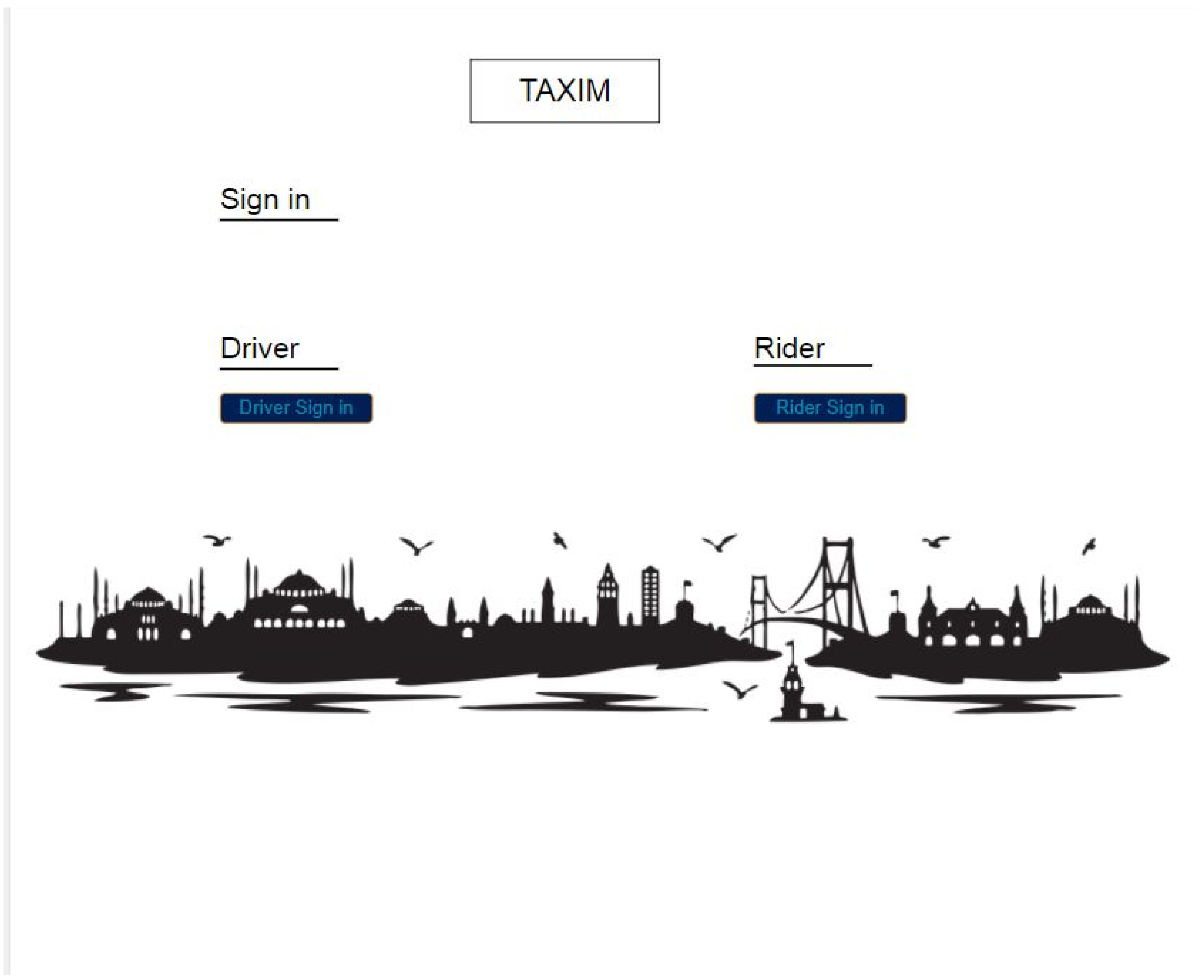
Process: Drivers will be able to register with their personal information.

Input: @first_name, @last_name, @phone_no, @e-mail,@password @City

SQL: INSERT INTO User(first_name,last_name,phone_no,e_mail,password,City) VALUES

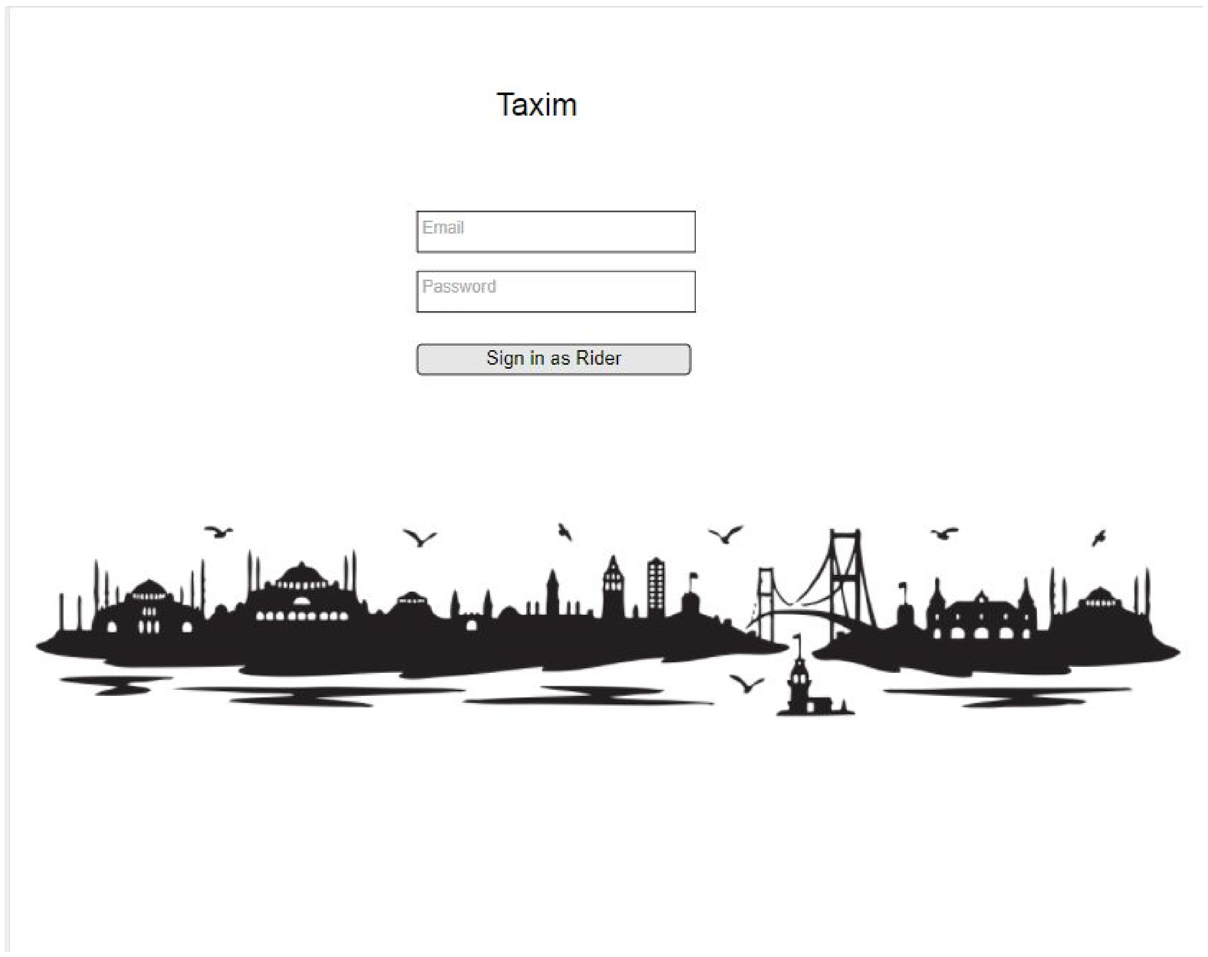
(@first_name, @last_name, @phone_no, @e-mail,@password,@City);

4.2 Login Page



There are 2 seperate login pages for **Customers** and **Drivers.**

5.2.1 Customer Login Page

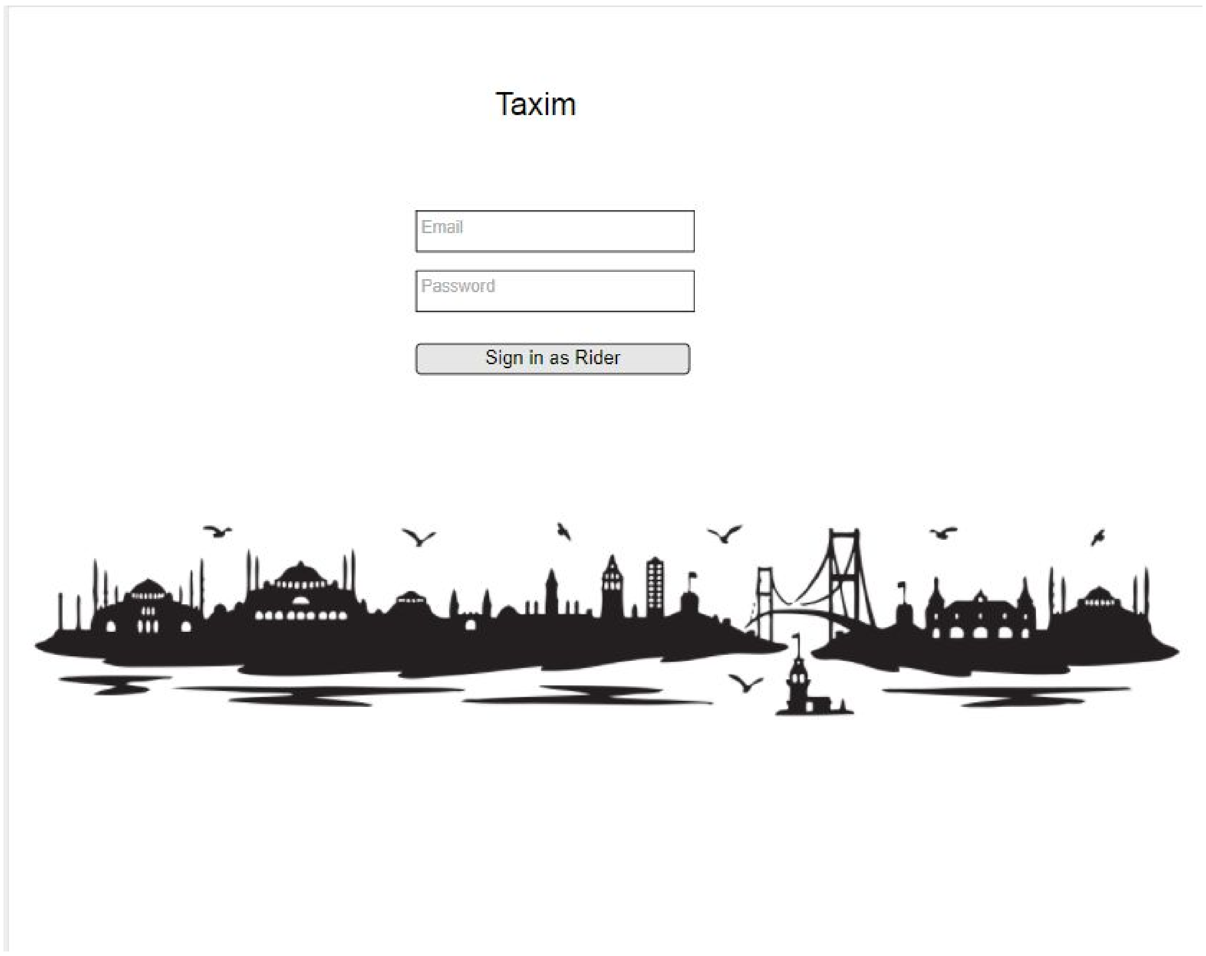


Process: Customers will be able to login with their username and password hey used in register page.

Input: @e_mail,@password

SQL: SELECT E_Mail,password from User WHERE E_mail = @e_mail AND password=@password;

4.3 Driver Login Page

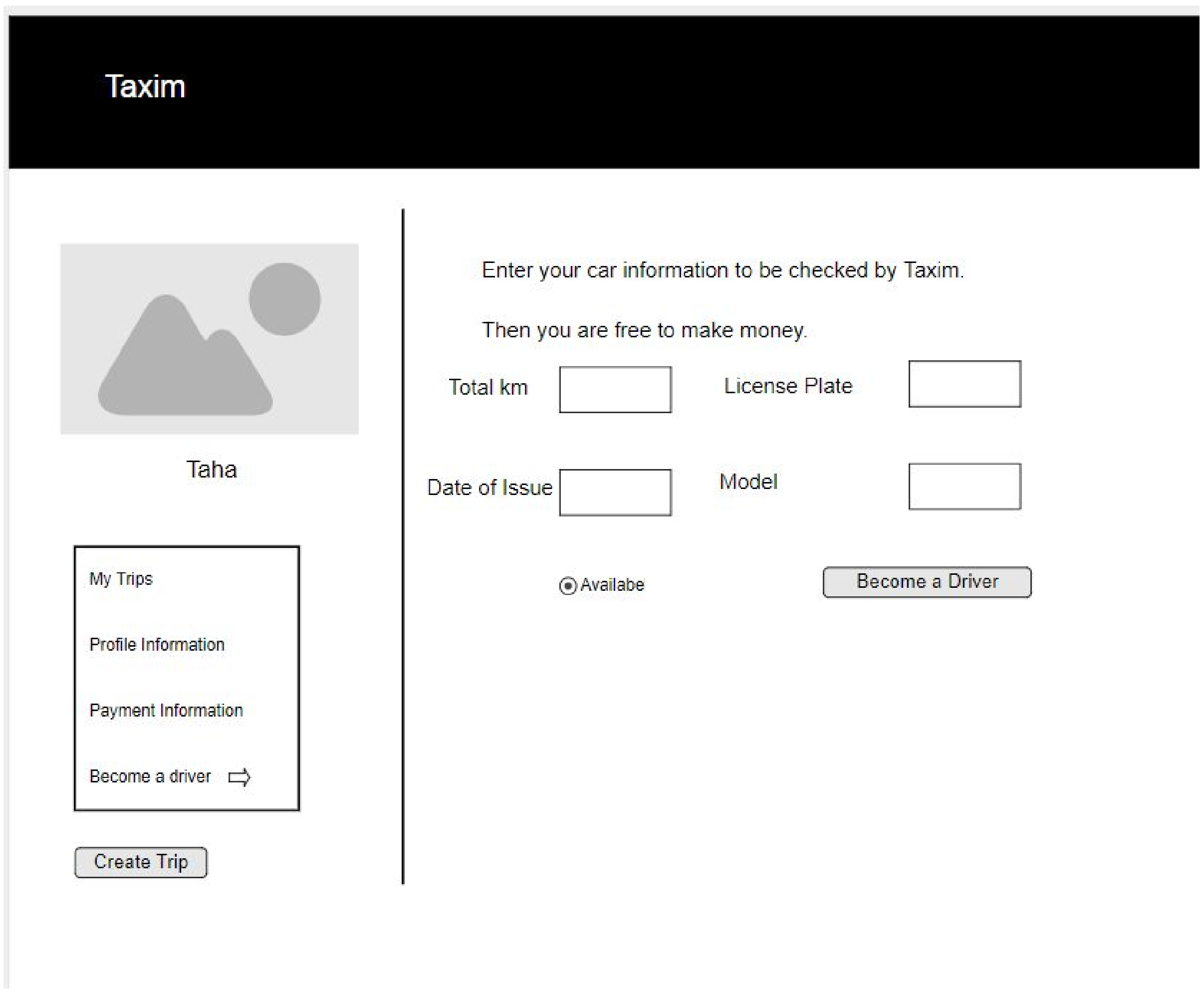


Process: Drivers will be able to login with their username and password hey used in register page.

Input: @e_mail,@password

SQL: SELECT @e_mail,password from Driver WHERE user_name = @user_name AND password=@password;

4.4 Become Driver Page



Enter your car information to be checked by Taxim.

Then you are free to make money.

Process: Drivers will be able to enter their car information to become an authorized driver.

Taha is already included in the system as rider. He wants to become a driver.

Current location of the driver is automatically added by the system using Google Maps API.

calculateLocation(@e_mail) -> current_location

Input: @availability

SQL:INSERT INTO DRIVER (Current_Location,Availability) VALUES

(current_location,@availability

)

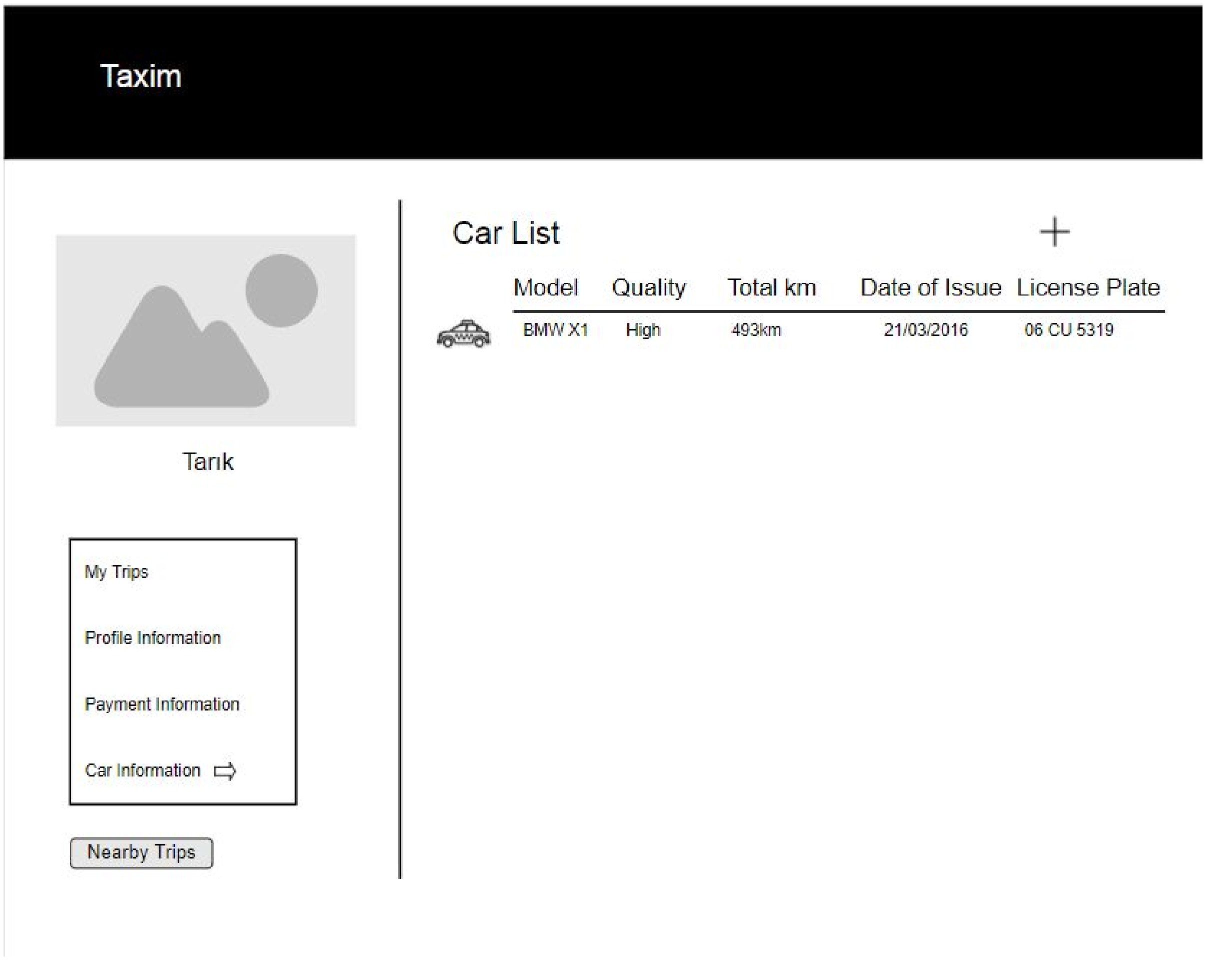
After driver add himself to the system. In that page, he will enter information about his car.

Input: @total_km,@plate_Number,@date_of_Issue ,@model_Name

SQL:INSERT INTO TAXI (Total_km,Plate_Number,Date_of_Issue,Model_Name) VALUES

(@total_km,@plate_Number,@date_of_Issue ,@model_Name)

4.5 List Car Information Page



Car List

Model	Quality	Total km	Date of Issue	License Plate
BMW X1	High	493km	21/03/2016	06 CU 5319

Process: User will be able to list information about cars.

The users email is already known by the system and it is stored as @e_mail

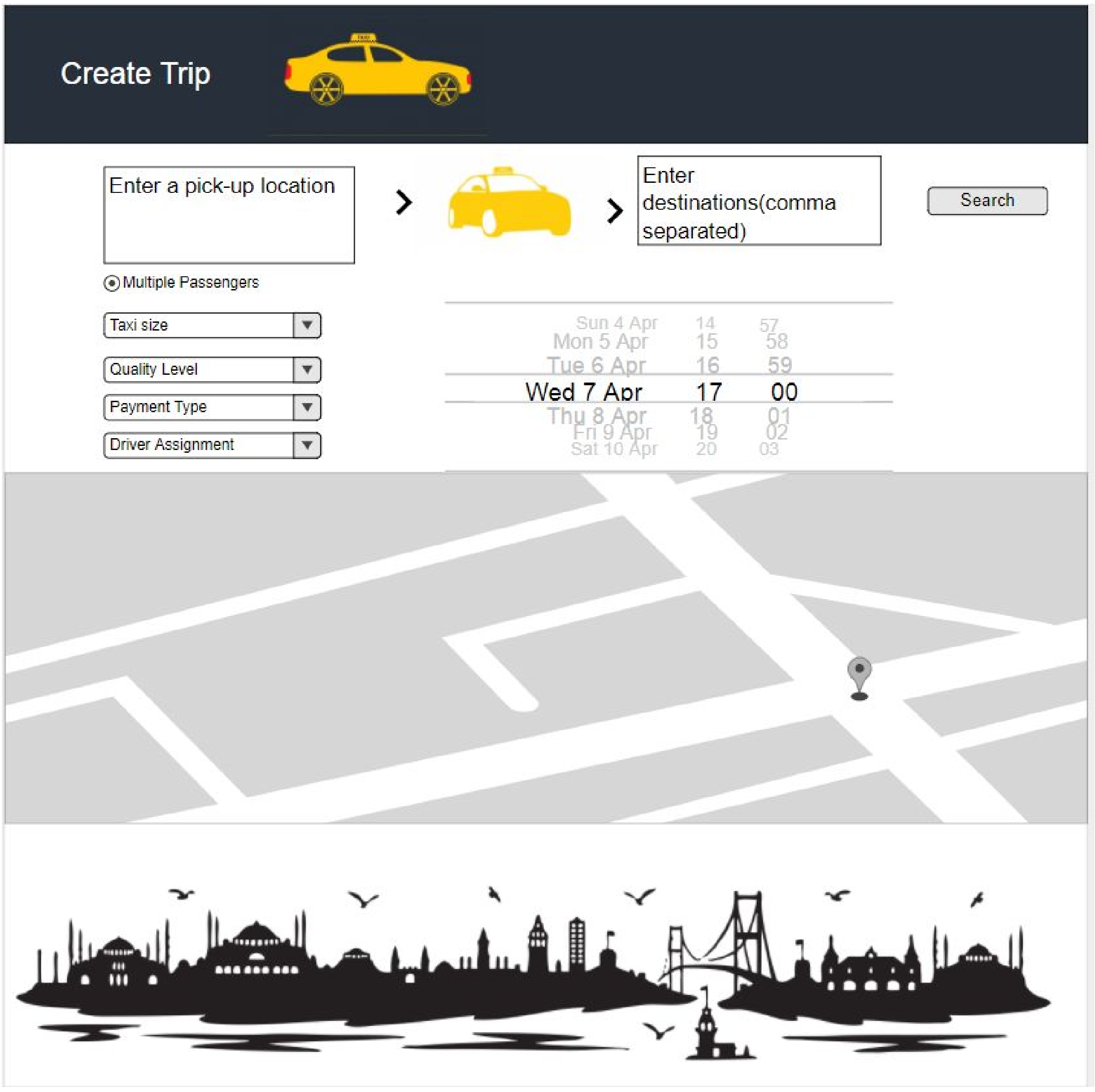
Input: None
SQL:

```
 SELECT Model_Name,Quality,Total_Km,Plate_Number FROM Taxi

FROM TAXI NATURAL JOIN DRIVER

 WHERE E_Mail= @e_mail
```

4.6 Create Trip Page



Process: Users will create a trip to a desired location by entering **pick up location, destination, taxi size, quality level and date.**

Input:

@pick_up_location,@destination,@taxi_size,@quality_level,@requestedDate,@time,@PaymenType,@DriverAssignment

Creating a trip process will consist of multiple insertions into different tables.

We have a model table in which we pre-define the cars. Once the driver enters a car with a model name we will automatically assign the quality of the car.
Inserting the quality of the car and the taxi-size

SQL: INSERT INTO Trip Features(Luxury Car, Capacity, No Other Rider) VALUES (@quality_level,@taxi_size,@Multiple Passengers)

Price value is calculated with a function in backend
-> @price =calculatePrice(startingLocation,destinationLocation,LuxuryCar,Capacity,MultiplePassengers);
->@date = todaysDate();
INSERT INTO Trip(Payment_type,Request_date,choose_driver_automatically,Requested_Start_Time,Price,Requested_Start_Date) VALUES
(@PaymentType,@date,@DriverAssignment,@time,@price,@requestedDate)

The customer will enter a pick up location; by using a location_id generator algorithm in back-end side of our page we will generate our location id to put in our **Requested Destination** table.

We will do the same process for destination since location ids will differ for different places.

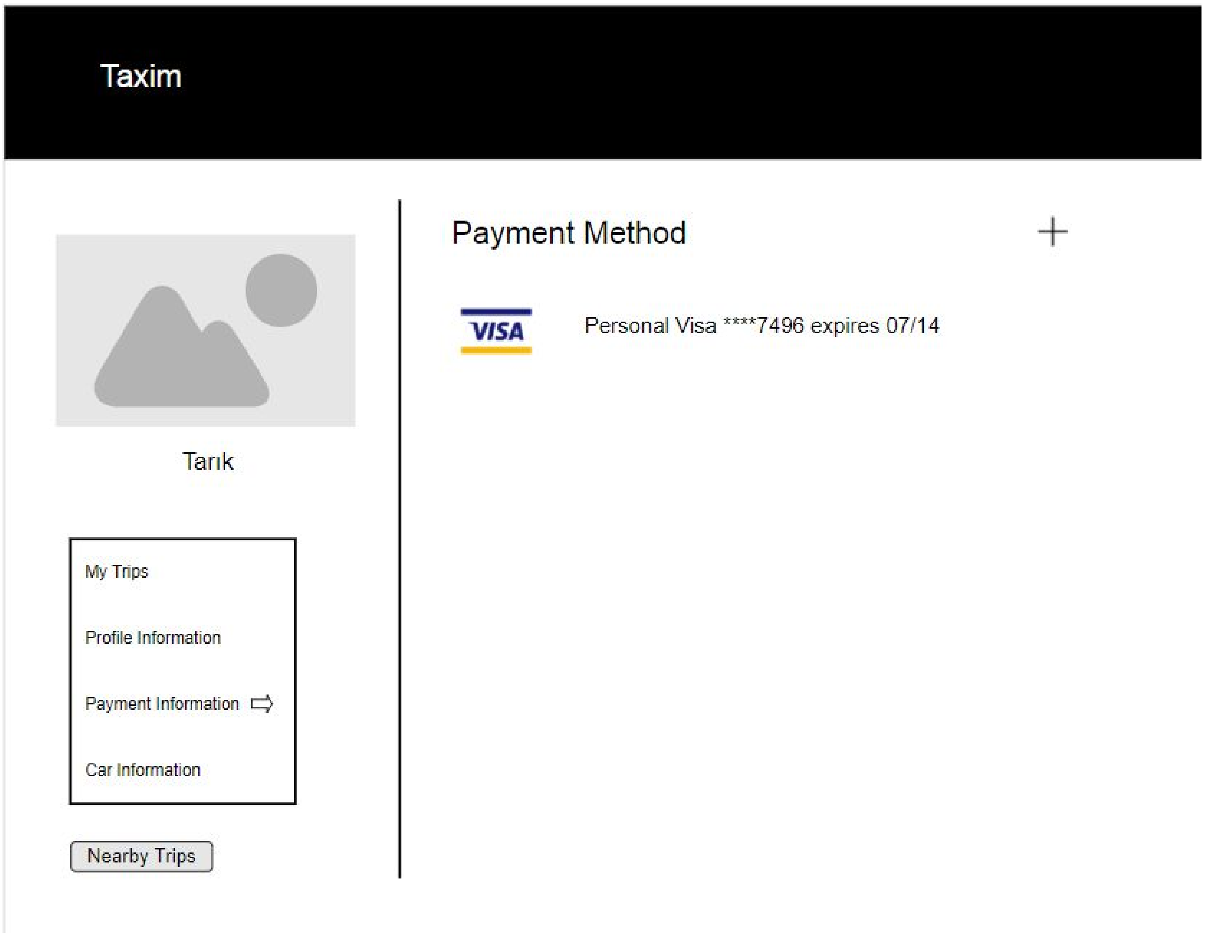
generateLocation(@destination); ->destination_id,coordainateY,coordinateX,name

SQL: INSERT INTO REQUESTED DESTINATIONS

(Location_ID,CoordinateY,CoordinateX,Name) VALUES

(destination_id,coordainateY,coordinateX,name);

4.7 Payment Method Page



Personal Visa ****7496 expires 07/14

Process: Customers will select the existing payment method or add a new one.

Input: none

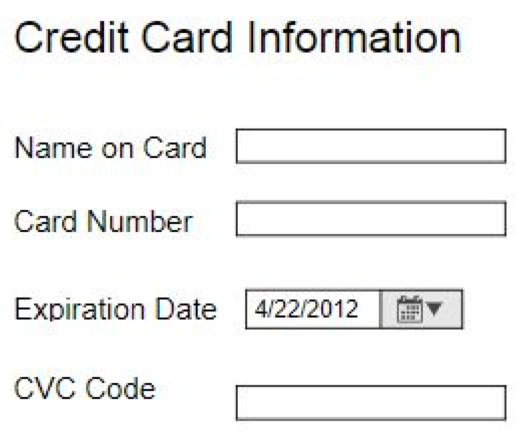
The user’s mail that has logged in is already stored by the systtem as @e_mail

Select existing payment methods.

SQL: SELECT * FROM Credit_Card_info NATURAL JOIN User

WHERE E_mail=@e_mail;

4.7.1 Credit Card Information Page


Credit Card Information

Name on Card	
Card Number	
Expiration Date	4/22/2012
CVC Code	

Process: Customer will enter credit card information in order to create a trip.

All payment process will be handled using external Credit Card Validation protocols.

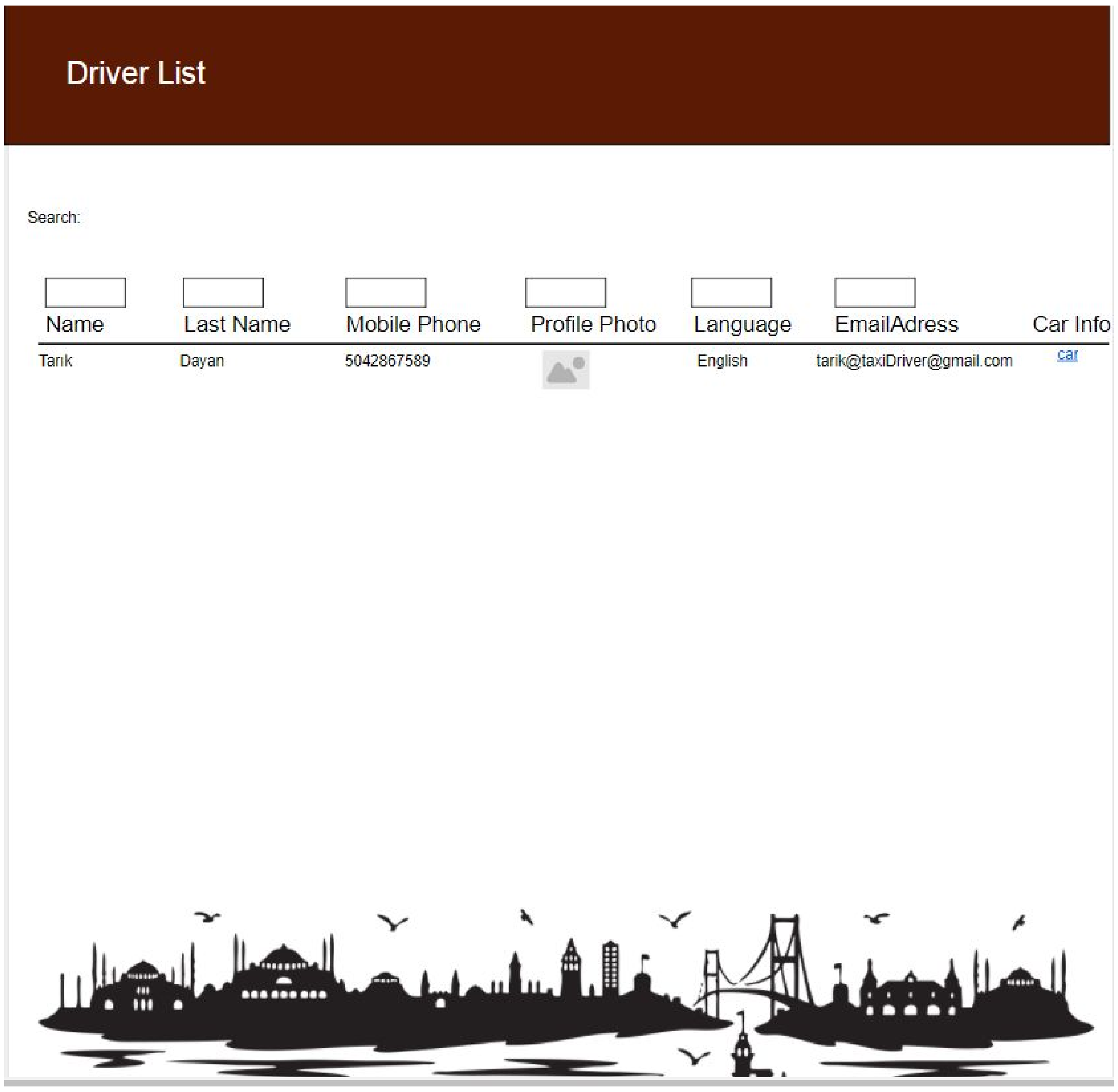
Input: @name_on_card, @card_no,@expiration_date,@cvc

We will split expiration date into expiration month and expiration year.

parseDate(@expiration_date) -> expiration_month, expiration_year);

SQL: INSERT INTO Credit_Card_Info(<u>Credit_Card_Number</u>, Card_Owner_Name, Exp_Month, Exp_Year, CVC) VALUES (@card_no,@name_on_card,@expiration_month,expiration_year,@cvc)

4.8 Driver List Page



Driver List

Search:

Name	Last Name	Mobile Phone	Profile Photo	Language	EmailAdress	Car Info
Tarık	Dayan	5042867589		English	tarik@taxiDriver@gmail.com	car

Process: Customer will be able search the drivers according to different parameters and display the resulting operation..

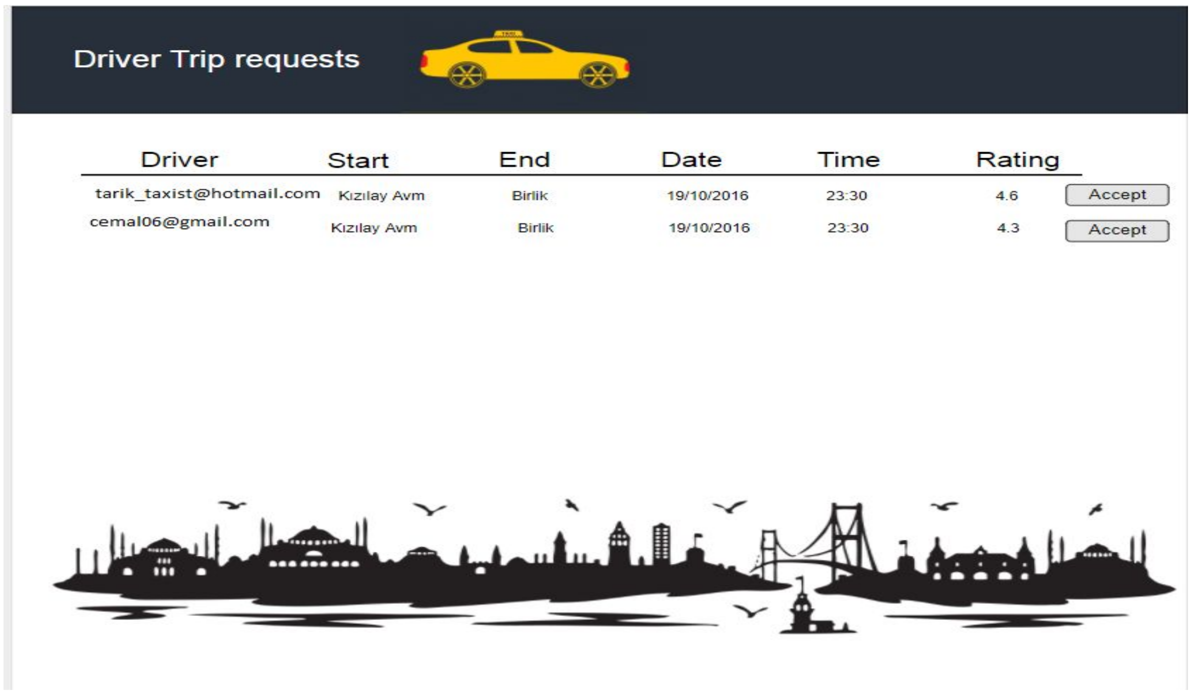
Input: @first_name,@last_name,@e_mail

Since driver is a user we can simply query the User Table

SQL for first_name parameter

SQL: SELECT first_name FROM User NATURAL JOIN Languge where Name LIKE %@first_name% AND LastName LIKE %@last_name% MobilePhone LIKE %@MobilePhone% LanguageName LIKE %@Language% E_mail LIKE %@e_mail%

4.9 Driver Trip Request Page



Driver	Start	End	Date	Time	Rating	
tarik_taxist@hotmail.com	Kızılay Avm	Birlik	19/10/2016	23:30	4.6	Accept
cemal06@gmail.com	Kızılay Avm	Birlik	19/10/2016	23:30	4.3	Accept

Process: Customers will be able to see the available drivers using the Driver Trip Request Page.

Input:@e_mail,@trip_id,@date

We already entered the information about the trip in **Create Trip Page.** At the end of Create Trip operation we have a @trip_id for that particular request.

//SQL for updating the accept relation. We already have the

SQL: INSERT INTO Merged_Trip(Merged_trip_id,Start_Time,E_mail) VALUES (@trip_id,@Start_time,@e_mail)

The other drivers who are not accepted will be added to rejected driver list.

Accepted driver will be stored with accepted status= “True” in our back end implementation of the page. Other drivers will remain unaccepted with accepted status=”False”. Based on that we will find out the non accepted drivers using their email addresses. After that we simply add them into Rejected Driver List.

SQL: INSERT INTO RejectedDriverList(trip_id,E_mail)

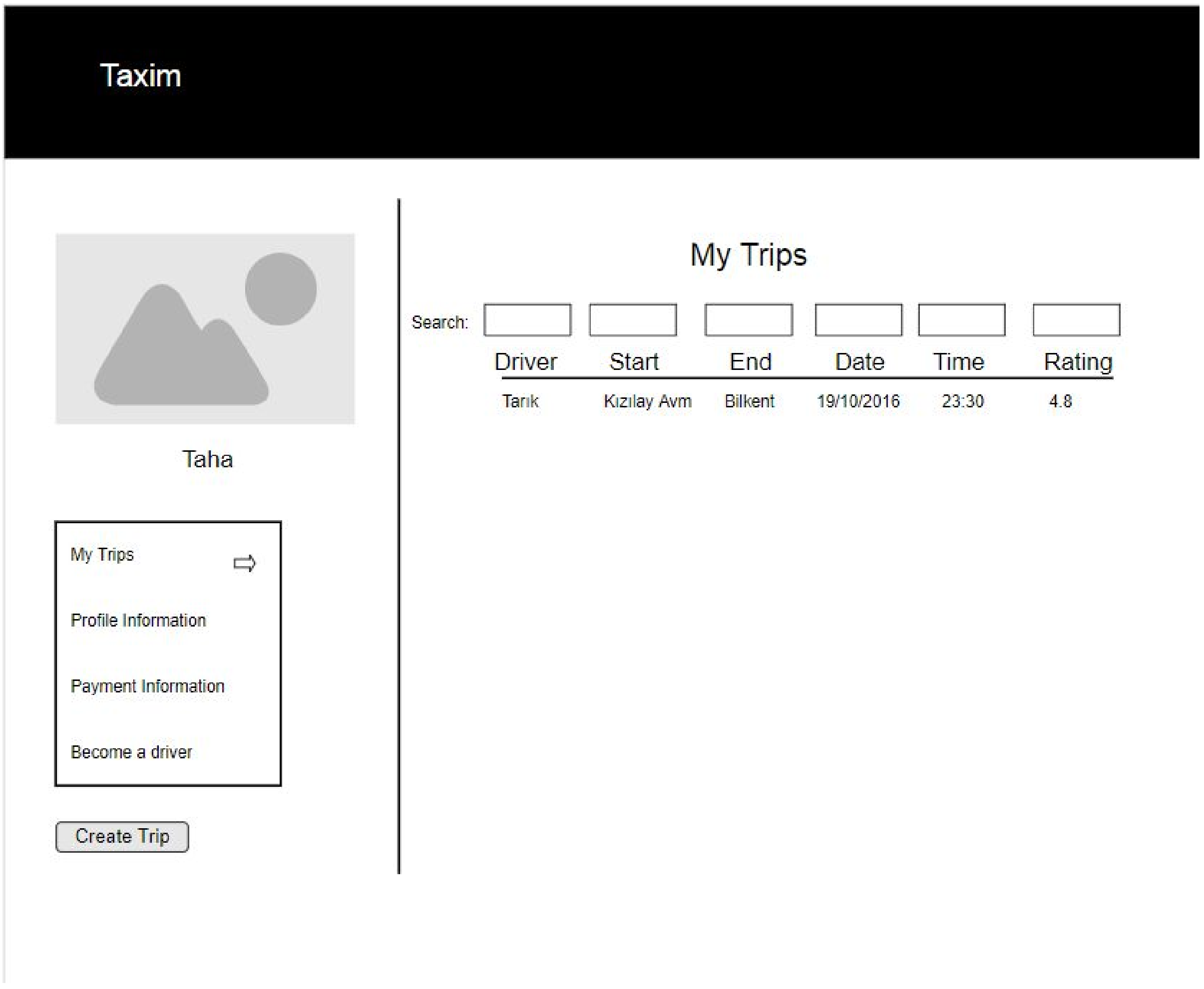
SELECT (trip_id,E_mail)

From Driver NATURAL JOIN Trip

WHERE E_mail <> @e_mail AND trip_id = @trip_id

4.10 Customer Trip Page

We have 2 pages for displaying trips for Customer and Driver.



Driver	Start	End	Date	Time	Rating
Tarık	Kızılay Avm	Bilkent	19/10/2016	23:30	4.8

Process: Customer will be able to access recent trips in **Customer Trip Page.**

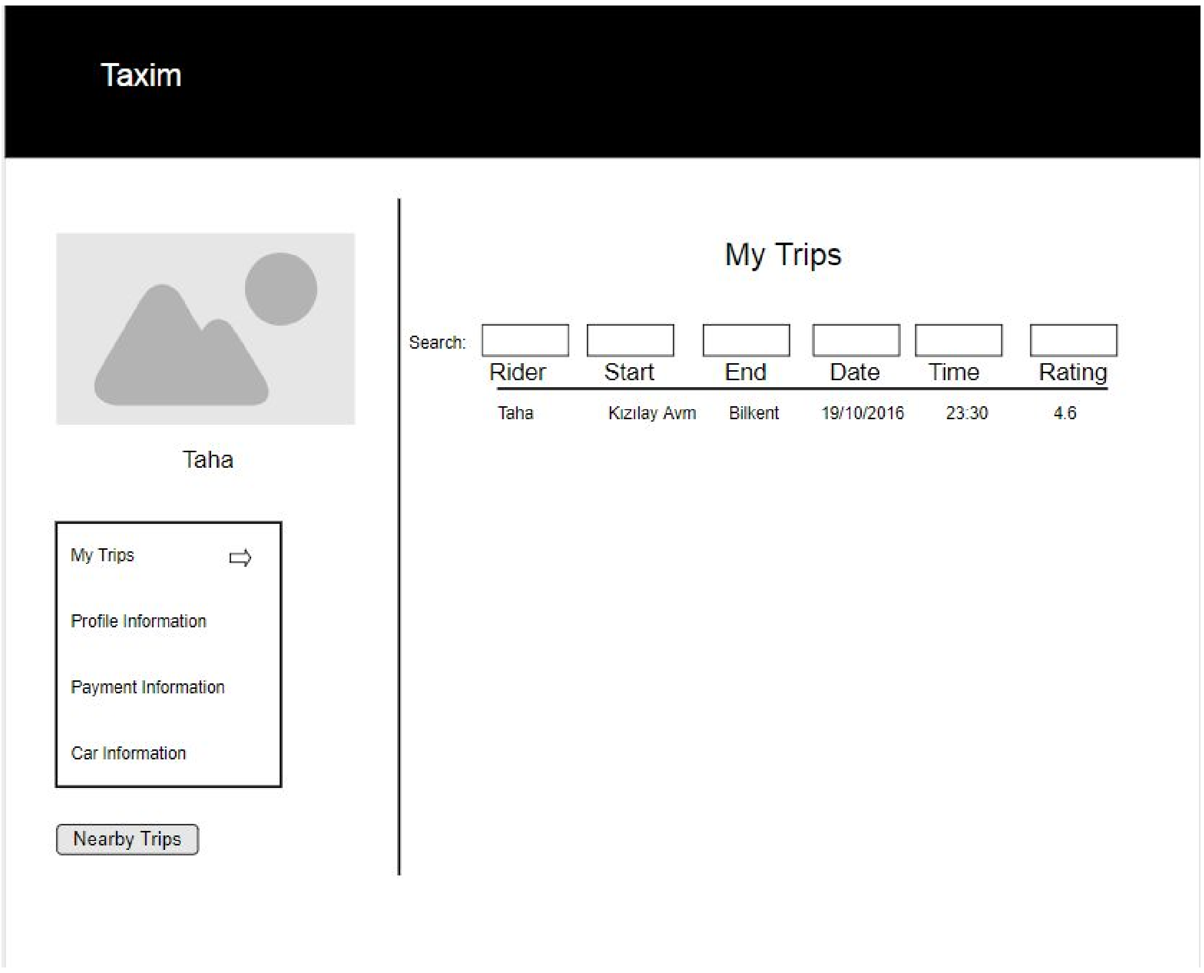
Input: @driver_name, @pick_up_location, @destination, @date,@time

We already know the currently logged in users email.

SQL: SELECT * FROM Trip NATURAL JOIN Driver

WHERE E_mail = @email AND First_Name LIKE %@driver_name% AND Where Request_Date = @date AND ride_start_time = @time

5.9 Driver Trip Page



Rider	Start	End	Date	Time	Rating
Taha	Kızılay Avm	Bilkent	19/10/2016	23:30	4.6

If the user logged in the system using the Driver session; this page will be shown. We already have the email-id

Process: Drivers will be able to display recent trips they made. They can also search riders and date of trip.By default all information about trip is displayed.

Input: @rider_name, @pick_up_location, @destination, @date,@time

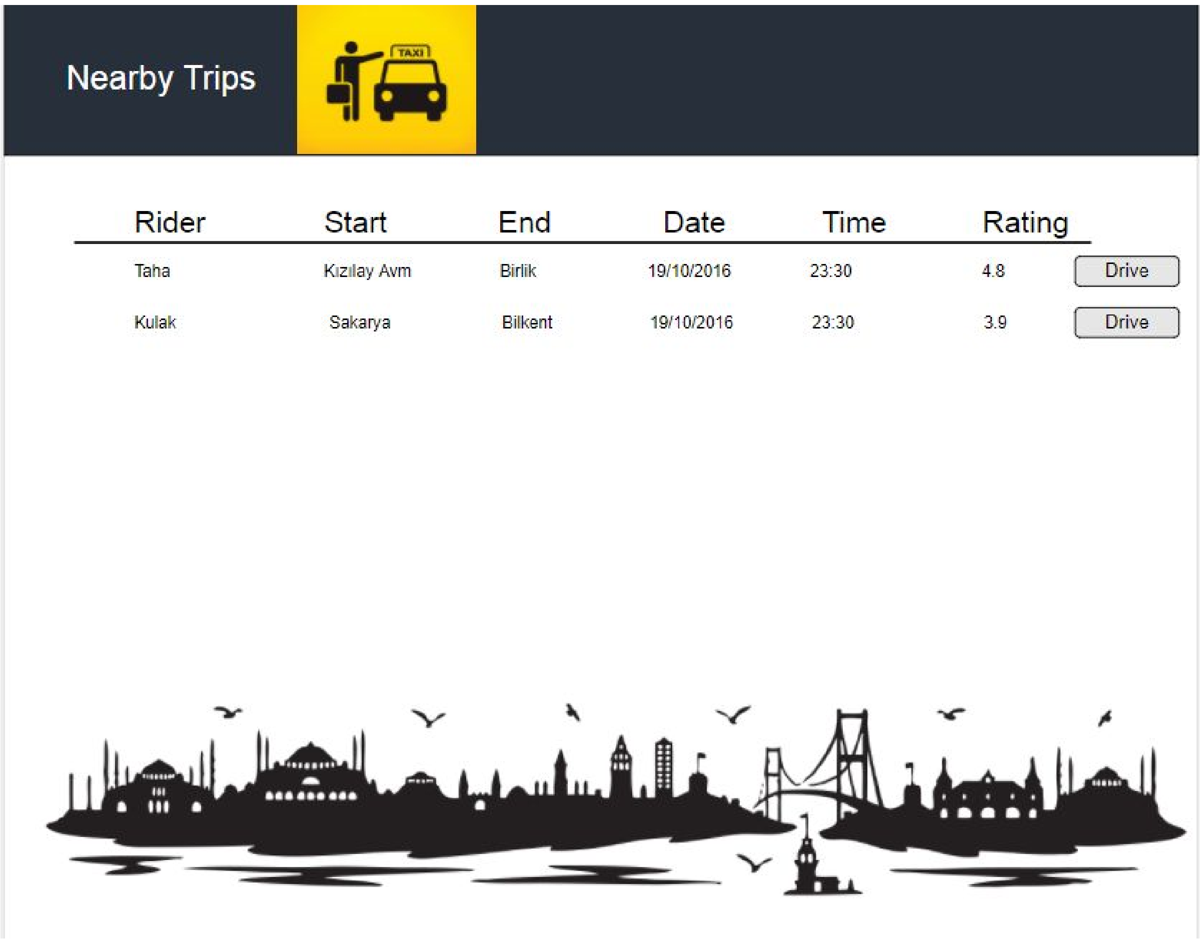
We already know the currently logged in users email.

SQL: SELECT * FROM Trip NATURAL JOIN User

WHERE E_mail = @email AND First_Name LIKE %@rider_name% AND Where

Request_Date = @date AND ride_start_time = @time

5.9 Nearby Trip Page



Rider	Start	End	Date	Time	Rating	
Taha	Kızılay Avm	Birlik	19/10/2016	23:30	4.8	Drive
Kulak	Sakarya	Bilkent	19/10/2016	23:30	3.9	Drive

The accept relation is where all the drivers who are accepted the trip are held. Thus whenever a driver presses one the Driver buttons in this page they are added to this very table with specific

We have the email of the user stored as @e_mail from the current session.

We are defining the distance of the driver to a location by the vertical and horizontal distance sum and if it is less then 5 km total we display the trip in drivers’ nearby trips list.

SQL: SELECT *

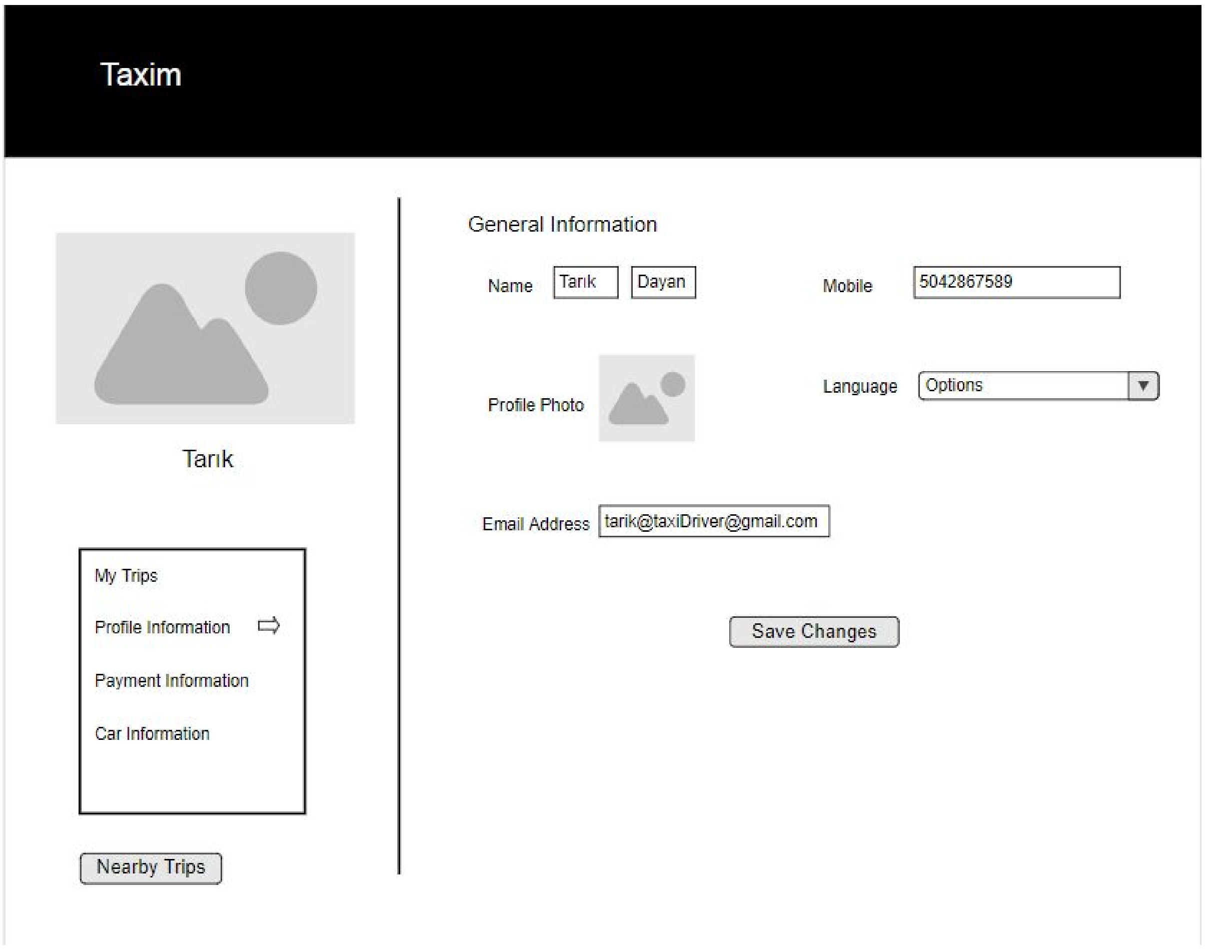
FROM Request natural join RequestedDestinations as CurrentDest, (Driver natural join RequestedDestinations) as DriverDest,

WHERE E_mail = @e_mail AND (abs(CurrentDest.CoordinateX-DriverDest.CoordinateX) +abs(CurrentDest.CoordinateY-DriverDest.CoordinateY)< 5)

//SQL for updating the accept relation. We already have the

SQL: INSERT INTO ACCEPT(trip_id,e_mail) VALUES (generated_trip_id,e_mail)

5.10 Driver Information Page



Name: Tarık Dayan
Mobile: 5042867589
Profile Photo
Language: Options
Email Address: tarik@taxiDriver@gmail.com

This is the page for the driver to edit their profile information. The email of the driver is already stored in the particular session they are logged in to.

Process: Drivers will be able to edit information about themselves.

Input: @name, @phone_no,@profile_photo,@language,@e_mail

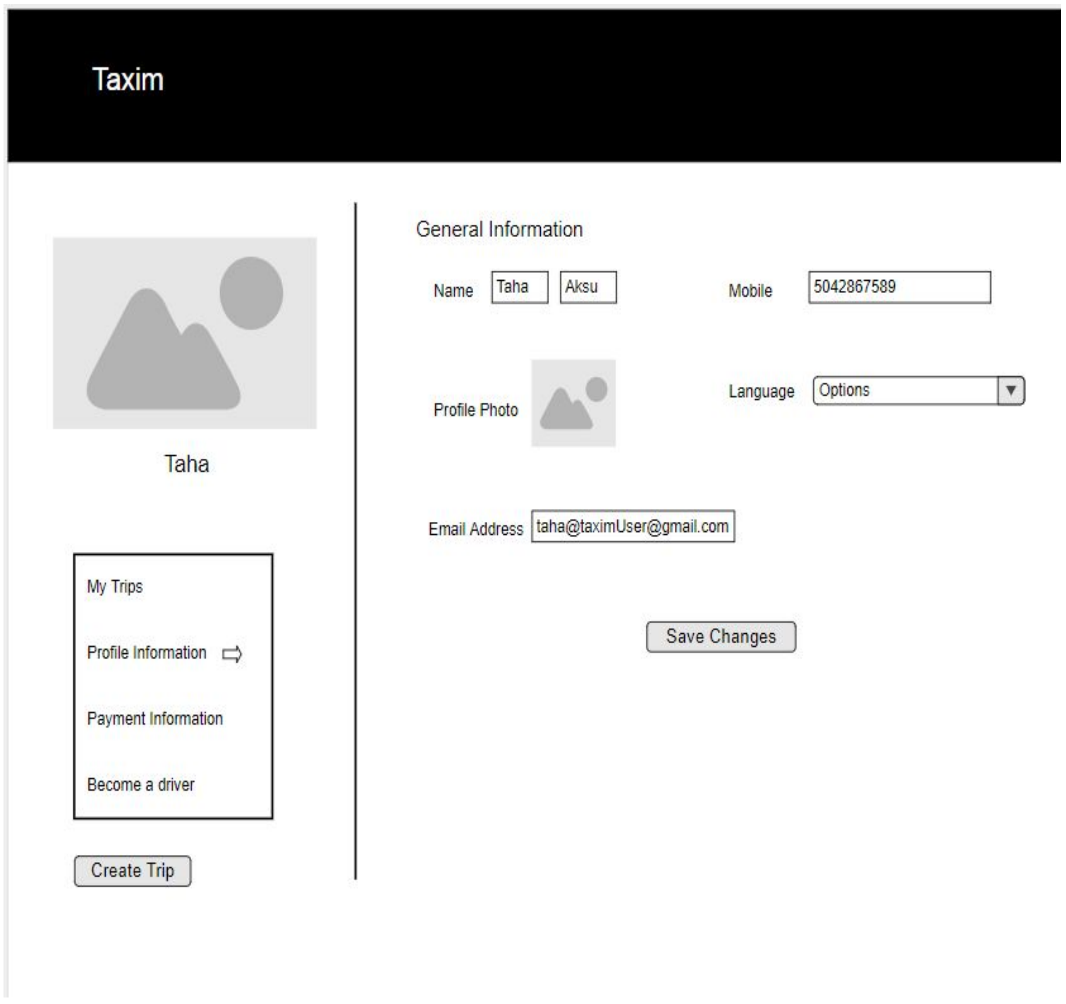
SQL: UPDATE Driver

SET

Name=@name,Phone_Number=@phone_no,Personal_Description=@profile_photo,Language=@language,E-mail=e_mail

WHERE E_mail =@e_mail

5.11 Profile Information Page



Name: Taha Aksu
Mobile: 5042867589
Profile Photo
Language: Options
Email Address: taha@taximUser@gmail.com

As stated before customers will see this page once they are logged in with Customer session and their email is stored in the session already.

Process: Customers will be able to edit information about themselves.

Input: @name, @phone_no,@profile_photo,@language,@e_mail

SQL: UPDATE User

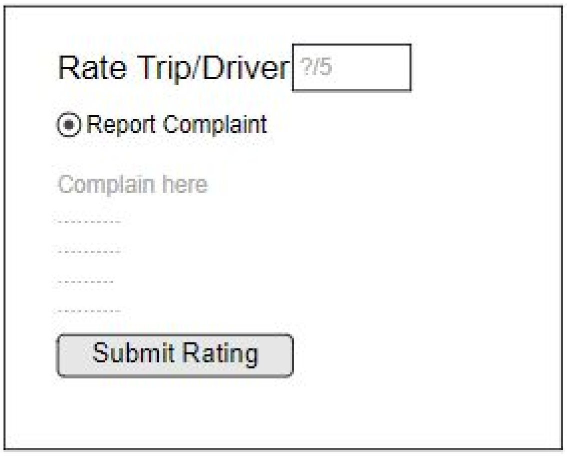
SET

Name=@name,Phone_Number=@phone_no,Personal_Description=@profile_photo,Langua

ge=@language,E-mail=@e_mail

WHERE E_mail =@e_mail

5.12 Rate Trip/Rider Page

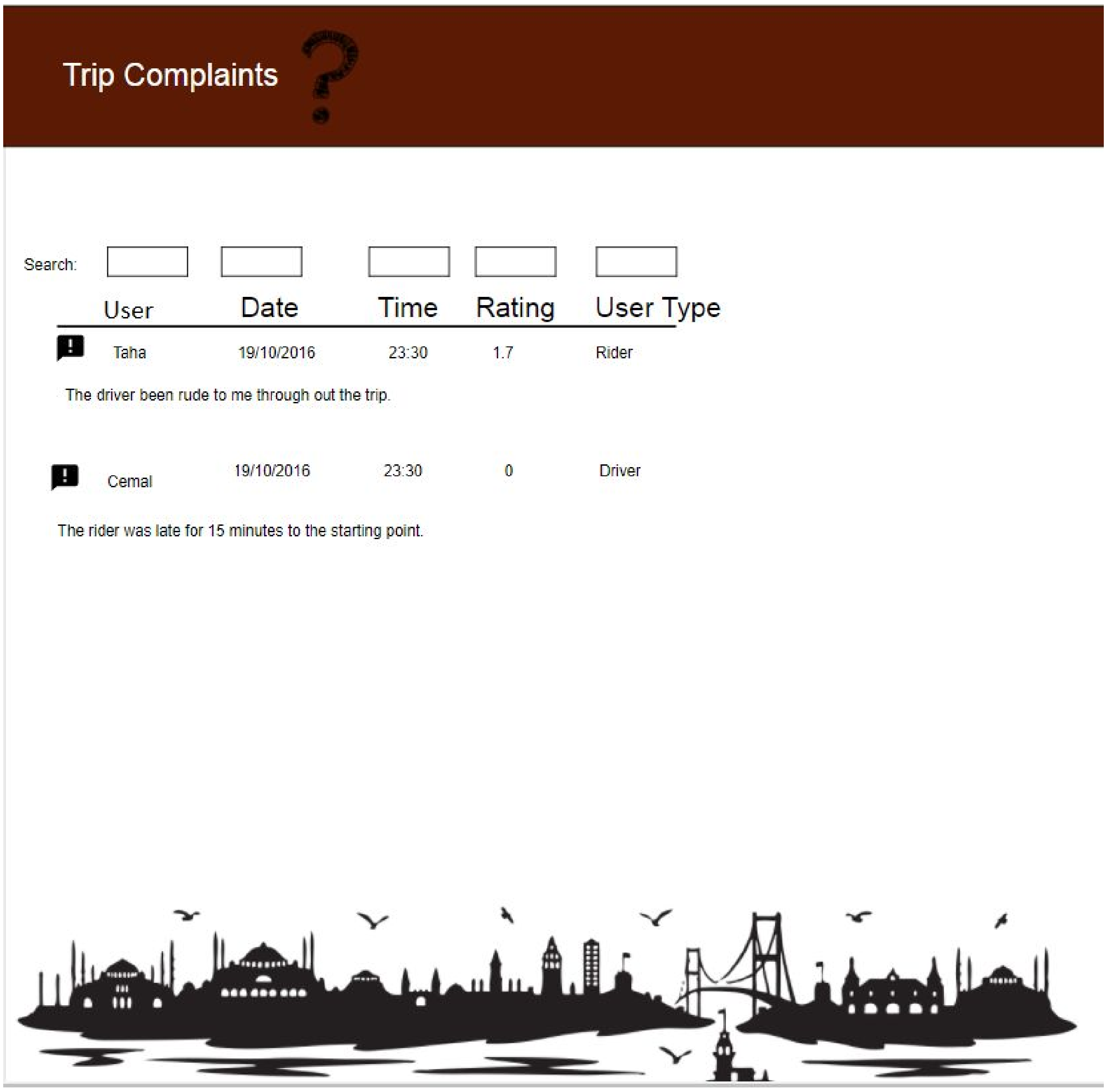


Process: Users will be able to rate their trip or driver using this page.

Input: @comment

SQL: INSERT INTO PASSENGER (Comment) VALUES @comment

5.13 Trip Complaints Page


Trip Complaints

Search:

	User	Date	Time	Rating	User Type
❗	Taha	19/10/2016	23:30	1.7	Rider

The driver been rude to me through out the trip.

	User	Date	Time	Rating	User Type
❗	Cemal	19/10/2016	23:30	0	Driver

The rider was late for 15 minutes to the starting point.

Return all trips that are reported with a complaint using an issue.

Process: Customer service will list the complaints of the drivers and riders if any complaints associated with them.

Input: None

Command: SELECT FirstName,Start_Time,End_Time,RatingComments

FROM user NATURAL JOIN opens NATURAL JOIN Issue NATURAL JOIN About NATURAL JOIN Merged_Trip NATURAL JOIN Drove

5. ADVANCED DATABASE COMPONENTS

5.1 Views

There are lot of restrictions on how users and drivers can see trips, past information, requests etc. for both security and to make easier to use Taxim. For example there's no need for a driver to see requests from far away places, or the requests he can't provide such as a request with a requirement of larger capacity. Or a customer service worker can't access user information who doesn't have a complaint, again for security and privacy reasons.

In the following view codes inputs are denoted with an @ sign.

5.1.1 Seeing Past Trips by a Customer

Customer can access driver's name, start and end dates, and his rating at first in the gui, if he clicks more he can see additional info such as his comments, detailed destination etc.

CREATE VIEW Initial_Past_Trips_of_a_User(DriverName, StartDate, EndDate, Rating){

SELECT Name, Ride_Start_Time, Ride_End_Time, Rating

```
FROM Merged_Trip JOIN Passenger JOIN Merges JOIN Trip
WHERE @UserE-Mail = E-Mail
}
```

5.1.2 Driver Request View

A driver can only see requests that are close to his vicinity, and the requests he can provide. This query is done assuming driver currently has no rider in his car since this would require more complex algorithms in c#. We need another view for the destinations in the trip since they are stored as row by row.

```
CREATE VIEW Driver_See_Requests(PickUp_Location_Name, Payment_Method,
Price, PeopleCount){
WITH NonStartedTrips as
                ( SELECT *
                FROM TRIP
                WHERE (Requested_Start_Time >  @CurrentTime AND
                Ride_Start_Time = NULL) OR (Ride_Start_Time = NULL AND
                Requested_Start_Time = NULL)),
        NonStartedFeatures as (
                FROM NonStartedTrips JOIN TripFeatures),
        PickUpLocations as ( SELECT *
                FROM NonStartedTrips JOIN Requested_Destinations
                WHERE          Order_in_the_Trip = 0),
        CarQualifications  as ( SELECT Capacity, Luxury
                FROM Taxi JOIN Model
                WHERE          Driver_ID = @driverEmail)
```

```
SELECT Name, Payment_Type, Price, Capacity
FROM NonStartedFeatures JOIN PickUpLocations
Where CarQualifications.Capacity>=NonStartedFeatures.Capacity AND
CarQualifications.Luxury>=NonStartedFeatures.Luxury
```

5.2 Stored Procedures

5.2.1 Sign In Sign Up Stored Procedures

For all of the sign in and sign up operations we are going to have predefined stored procedures for making sure that our system is secure. By taking the username and password as inputs to those procedures we would make sure that no unauthorized user can Manipulate the sql queries in a way to reach the database. Most basicly for sign up we will take the parameters required in registering and pass them to the related stored procedure which will add these informations to the driver table if the register operation is being done by the driver or to the Customer table if it is a customer registering.

5.2.2 Create Trip Stored Procedure

Creating trips will be done through a predefined stored procedure. We are going to take the required parameters through UI which will be fed into the stored procedure as input any value that is not fed for an input of the procedure will be set as null wont be taken in to consideration later in the filtering of the trips. For example if the customer does not provide a luxury level for the car, the trip can be seen by every nearby driver whatever ther cars’s luxury level is.

5.2.3 Add credit card Stored Procedure

In order to add additional credit cards we will use predefined stored procedure this both decreases redundancy and makes the retrieval and assignment of information between database and system secure. By not doing the operations directly through sql queries we prevent the access to the credit card table by unauthorized users. The procedure takes name, card number, expiration date and cvc code and puts it into the credit_card_info table.

5.2.4 Updating Driver Information

The drivers are able to update their information which are displayed in their profile page. In order to do that they basically have to type new values in and press the button save changes. These values entered by the user will be fed to the stored procedures as inputs and the drivers information will be updated accordingly.

5.2.5 Updating Customer Information

The customers like drivers are able to update their profile information from their profile pages. The same context is present here as well. We have a very similar stored procedure with one slight difference, that is it updates not the driver table but the customer table.

5.2.6 Distance Calculating Stored Procedure Given IDs

A stored procedure is used for calculating the distance between two locations given location ids. The stored procedure takes two location ids as input and returns the distance between two points back according to the differences in CoordinateX and CoordinateY.

5.2.7 Distance Calculating Stored Procedure Given a Location ID and Two Coordinates

A stored procedure is used for calculating the distance between two locations given a location id and two coordinate values CoordinateX and CoordinateY. The stored procedure takes two location ids as input and returns the distance between two points back according to the differences in CoordinateX and CoordinateY.

5.3 Reports

5.3.1 Total Number of Complaints from Users, Total Number of Complaints from Customers, Total Number of Complaints from Drivers

```
1)SELECT COUNT(Issue_ID)
FROM ISSUE
2)SELECT COUNT(Issue_ID)
FROM issue NATURAL JOIN  user NATURAL JOIN opens NATURAL JOIN driver
WHERE  E_mail in (SELECT E_mail
                  FROM customer)
3)SELECT COUNT(Issue_ID)
FROM issue NATURAL JOIN  user NATURAL JOIN opens NATURAL JOIN driver
WHERE  E_mail in (SELECT E_mail
                  FROM driver)
```

5.3.2 Total Number Of Rejections Made During a Particular Date

For customer service this very query provides the opportunity to see how many drivers were rejected in that very day at total.

```
Var @date => todays date
SELECT COUNT(trip_id)
FROM trip NATURAL JOIN RejectedDriverList
WHERE Request_date= @date
```

5.4 Triggers

5.4.1 Updating Rejected Driver List

Whenever a driver is accepted for a particular trip request, that is a new entry for the merged_trip table is created, other drivers who were also candidates to drive that particular request are all put into rejected driver list with a trigger.

5.4.2 Updating Nearby Trip List

Whenever a new trip is created the nearby trip of all drivers are updated with a trigger. If the trip is near to a driver the trip has to be added to the nearby list of that drivers’ nearby list.

5.4.3 Updating Driver Trip Request and Nearby Trip List In Case of an Acceptance of a Trip

Whenever a trip is accepted that is offered by a driver that very trip has to be deleted from the nearby List of all other drivers. Moreover if the trip is rejected by the customer then the drivers nearby List should be updated in way to not to include that very trip anymore.

5.5 Constraints

5.5.1 Capacity Constraint for Taxis

The maximum number of passenger cannot exceed 4.

5.5.2 Destination Constraint for Trip

User cannot assign more than one destination while creating a trip.

5.5.3 Simultaneous Ride Requests Constraint for Driver

Driver cannot accept ride request from different users that have same date and ride time for the customer.

5.5.4 Simultaneous Drive Requests Constraint for User

User cannot accept ride request from different drivers that offer rides that have same date and time for the customer.

5.5.5 Driver Profile Photo Constraint for Driver

Drivers cannot leave the profile photo empty due to security concerns and standard online transportation protocols.

5.5.5 Customer Profile Photo Constraint for Customer

Customers cannot leave the profile photo empty due to security concerns and standard online transportation protocols.

6. IMPLEMENTATION PLAN

In back-end of the project, we are going to use MSSQL server to manage and store the data of the web application. Other than that, to handle the necessary logical operations in our servers we are going to use ASP .NET Framework using C#.
To design the front-end side of the project we are going to use HTML for structure, CSS and standard JavaScript APIs Bootstrap,JQuery for design,animation and front-end logic.