Bilkent University

Department of Computer Engineering

CS 353 Term Project

Travel Agency Data Management System - Group 9

Design Report

Project Group Members:

Zübeyir Bodur - 21702382

Funda Tan - 21801861

Emine Ezgi Saygılı - 21802871

Abdul Razak Daher Khatib - 21801340

TA: Mustafa Can Çavdar

Table of Contents

1. Entity-Relationship Diagram

We have revised our E/R model based on the feedback given on the Proposal of the project as follows:

- The guides_feedback relation, which is between Guide, Tour, and Feedback tables was changed accordingly. All pairs of this relation are now one-to-one, i.e. a Guide can give one feedback on a specific tour, feedback belongs to a single tour and a tour has at most one feedback associated with it.
- There is total participation of Discounts in applies relation (between Discount and Agents), i.e. All Discounts are applied by Agents.
- “make_review” relation (between Customer and Review tables) is now one-to-one, i.e. a review can be written by one customer.
- The notation error in the “receive” relation is fixed.
- There is total participation of TourReviews in the “tour_rev” relation (between TourReview and Tour tables), i.e. all tour reviews belong to a single tour.
- There relation “visits”” (between SightseeingPlace and Tour tables) is now many-to-many, i.e. a SightseeingPlace might be available in multiple tours.
- The relation “assign_guide” is changed accordingly. One agent assigns only one guide to some tours. The same agent can assign the same guide to any tour, however, the time conflicts will be checked by the API. For the system to allow the same (agent, guide) pair to assign the guide to any tour, the Tour side of the relationship should be Many.
- Discounts are changed. Since it does not make sense to apply discounts to reservations one by one, hotels, tours, and activities receive discounts in bulk, i.e. if a hotel receives a discount, all rooms of the hotels receive a discount and the user benefits from the discount before making the reservation. The similar applies to extra activities and tours.
- Tour’s attributes have been changed. Now, every tour has a predetermined length in days, e.g. 30 days, and they have a start date. Customers only choose the start date for tour reservations.

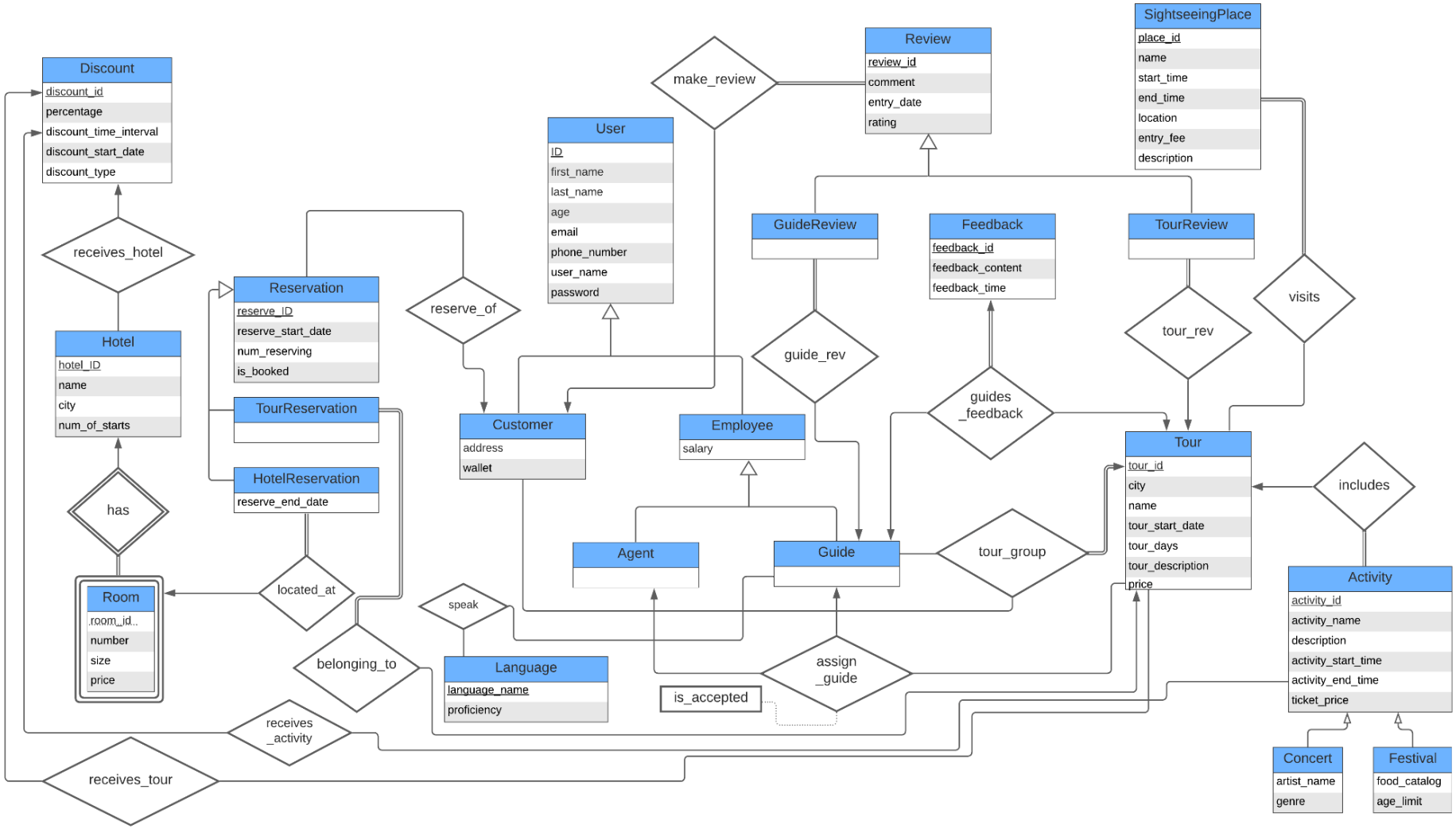
Discount
discount_id
percentage
discount_time_interval
discount_start_date
discount_type
receives_hotel
Hotel
hotel_ID
name
city
num_of_starts
has
Room
room_id
number
size
price
Reservation
reserve_ID
reserve_start_date
num_reserving
is_booked
reserve_of
TourReservation
HotelReservation
reserve_end_date
located_at
belonging_to
User
ID
first_name
last_name
age
email
phone_number
user_name
password
Customer
address
wallet
Employee
salary
Agent
Guide
speak
Language
language_name
proficiency
make_review
Review
review_id
comment
entry_date
rating
GuideReview
guide_rev
Feedback
feedback_id
feedback_content
feedback_time
guides
_feedback
TourReview
tour_rev
SightseeingPlace
place_id
name
start_time
end_time
location
entry_fee
description
visits
Tour
tour_id
city
name
tour_start_date
tour_days
tour_description
price
tour_group
includes
Activity
activity_id
activity_name
description
activity_start_time
activity_end_time
ticket_price
Concert
artist_name
genre
Festival
food_catalog
age_limit
assign
_guide
is_accepted
receives
_activity
receives_tour

Figure 1: ER Diagram of the Travel Agency Web Application

2. Schemas

2.1. Tour

Tour (tour_id, city, name, tour_start_date, tour_days, tour_descritption, price)

PK: tour_id

Functional Dependencies: tour_id -> city, name, tour_start_date, tour_end_date, tour_description, price

Form: 3NF

2.2. User

User (ID, first_name, last_name, age, email, phone_number, user_name, password)

PK: ID

Functional Dependencies: ID -> first_name, last_name, age, email, phone_number, user_name, password

Form: 3NF

2.2.1. Customer

Customer (ID, first_name, last_name, age, email, phone_number, user_name, password, address, wallet)

PK: ID

Functional Dependencies: ID -> first_name, last_name, age, email, phone_number, user_name, password, address, wallet

Form: 3NF

2.2.2. Employee

Employee (ID, first_name, last_name, age, email, phone_number, user_name, password, salary)

PK: ID

Functional Dependencies: ID -> first_name, last_name, age, email, phone_number, user_name, password, salary

Form: 3NF

2.2.2.1. Guide

Guide(ID, first_name, last_name, age, email, phone_number, user_name, password, salary)

PK: ID

Functional Dependencies: ID -> first_name, last_name, age, email, phone_number, user_name, password, salary

Form: 3NF

2.2.2.2. Agent

Agent(ID, first_name, last_name, age, email, phone_number, user_name, password, salary)

PK: ID

Functional Dependencies: ID -> first_name, last_name, age, email, phone_number, user_name, password, salary

Form: 3NF

2.3. Review

Review(review_id, comment, entry_date, rating, customer_ID)

PK: review_id

FK: customer_ID references Customer

Functional Dependencies: review_id -> comment, entry_date, rating, customer_ID

Form: 3NF

2.3.1. GuideReview

GuideReview(review_id, comment, entry_date, rating, customer_ID, guide_ID)

PK: review_id

FK: customer_ID references Customer

FK: guide_ID references Guide

Functional Dependencies: review_id -> comment, entry_date, rating, customer_ID, guide_ID

Form: 3NF

2.3.2. TourReview

TourReview(review_id, comment, entry_date, rating, ID, tour_id)

PK: review_id

FK: ID references Customer

FK: tour_id references Tour

Functional Dependencies: review_id -> comment, entry_date, rating, ID, tour_id

Form: 3NF

2.4. Activity

Activity(activity_id, activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id)

PK: activity_id

FK: tour_id references Tour

FK: discount_id references Discount (if no discount is applied NULL)

Functional Dependencies: activity_id -> activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id

Form: 3NF

2.4.1. Festival

Festival(activity_id, activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id, food_catalog, age_limit)

PK: activity_id

FK: tour_id references Tour

FK: discount_id references Discount (if no discount is applied NULL)

Functional Dependencies: activity_id -> activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id, food_catalog, age_limit

Form: 3NF

2.4.2. Concert

Concert(activity_id, activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id, artist_name, genre)

PK: activity_id

FK: tour_id references Tour

FK: discount_id references Discount (if no discount is applied NULL)

Functional Dependencies: activity_id -> activity_name, description, activity_start_time, activity_end_time, ticket_price, tour_id, discount_id, artist_name, genre

Form: 3NF

2.5. SightseeingPlace

SightseeingPlace(place_id, name, start_time, end_time, location, entry_fee, description)

PK: place_id

Functional Dependencies: place_id -> name, start_time, end_time, location, entry_fee, description

Form: 3NF

2.6. Hotel

Hotel(hotel_ID, name, city, num_of_stars, discount_id)

PK: hotel_ID

FK: discount_id references Discount

Functional Dependencies: hotel_ID -> name, city, num_of_stars, discount_id

Form: 3NF

2.7. Discount

Discount(discount_id, percentage, discount_time_interval, discount_start_time, discount_type)

PK: discount_id

Functional Dependencies: discount_id -> percentage, discount_time_interval, discount_start_time, discount_type

Form: 3NF

2.8. Reservation

Reservation(reserve_ID, reserve_start_date, num_reserving, is_booked, customer_ID)

PK: reserve_ID

FK: customer_ID references Customer

Functional Dependencies: reserve_ID -> reserve_start_date, num_reserving, is_booked, customer_ID

Form: 3NF

2.8.1. TourReservation

TourReservation(reserve_ID, reserve_start_date, num_reserving, is_booked, customer_ID)

PK: reserve_ID

FK: customer_ID references Customer

Functional Dependencies: reserve_ID -> reserve_start_date, num_reserving, is_booked, customer_ID

Form: 3NF

2.8.2. HotelReservation

HotelReservation(reserve_ID, reserve_start_date, num_reserving, is_booked, customer_ID, reserve_end_date, room_id, hotel_ID)

PK: reserve_ID

FK: customer_ID references Customer

FK: room_id, hotel_ID references Room

Functional Dependencies: reserve_ID -> reserve_start_date, num_reserving, is_booked, customer_ID, reserve_end_date, room_id, hotel_ID

Form: 3NF

2.9. Room

Room(room_id, hotel_ID, number, size, price)

PK: room_id, hotel_ID

FK: hotel_ID references Hotel

Functional Dependencies: room_id, hotel_ID -> number, size, price

Form: 3NF

2.10. Feedback

Feedback(feedback_id, content, feedback_time)

PK: feedback_id

Functional Dependencies: feedback_id -> content, feedback_time

Form: 3NF

2.11. Language

Language(language_name, proficiency)

Primary key: language_name

Functional Dependencies:

Form: 3NF

2.12. visits

visits(place_id, tour_id)

Primary key: place_id, tour_id

FK: place_id references Sightseeing

FK: tour_id references Tour

Form: 3NF

2.13. speak

speak(ID, language_name)

Primary Key: ID, language_name

FK: ID references Guide

FK: language_name references Language

Form: 3NF

2.14. guides_feedback

This is the relation between Guide, Tour, and Feedback, which represents how a Guide gives feedback at the end of the Tour.

guides_feedback(feedback_id, ID, tour_id)

PK: feedback_id, ID

FK: feedback_id references Feedback

FK: ID references Guide

FK: tour_id references Tour

Functional Dependencies: feedback_id, ID -> tour_id

2.15. assign_guide

This is the relation between Agent, Tour, and Guide, which represents how an Agent assigns a Guide into a Tour.

assign_guide(guide_ID, agent_ID, tour_id, is_accepted)

PK: guide_ID, agent_ID, tour_id

Unique Key: guide_ID, tour_id

Unique Key: agent_ID, tour_id

FK: tour_id references Tour

FK: agent_ID references Agent

FK: guide_ID references Guide

Functional Dependencies: guide_ID, tour_id -> agent_ID

Functional Dependencies: agent_ID, tour_id -> guide_ID

2.16. tour_group

This is the relation between a Guide, Tour, and Customers

tour_group(customer_ID, guide_ID, tour_id)

PK: customer_ID, guide_ID

FK: tour_id references Tour

FK: agent_ID references Agent

FK: guide_ID references Guide

Functional Dependencies: customer_ID, guide_ID -> tour_id

3. User Interface Design and Corresponding SQL Statements

Below, for each functionality, the necessary SQL Queries and UI mockups are given. In addition, the common functionalities specified in the Project Functionality Document and the functionalities unique to this project are separated as well.

3.1. Common Functionalities

Below are the SQL Queries for each functionality. The common functionalities specified in the Project Functionality Document and the functionalities unique to this project are separated as below.

3.1.1. Login and Register

3.1.1.1. UI Mockups

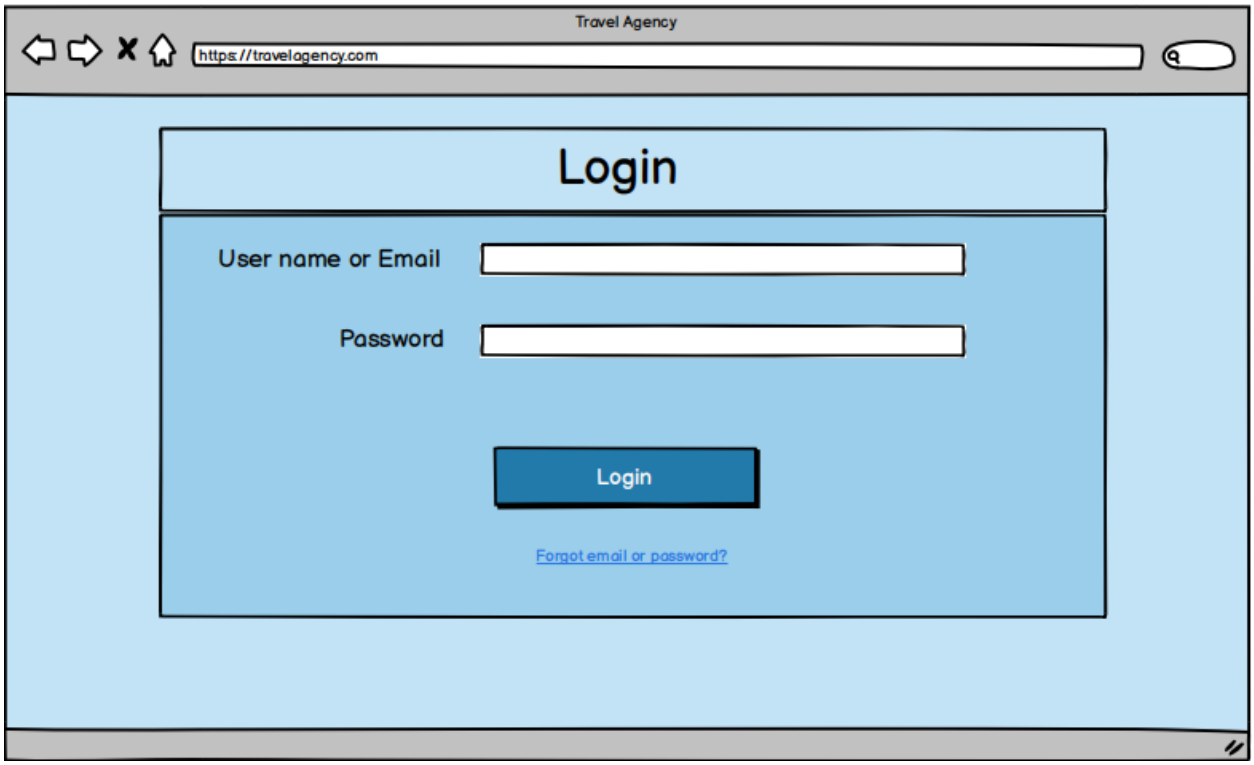

Figure 2: Login Page

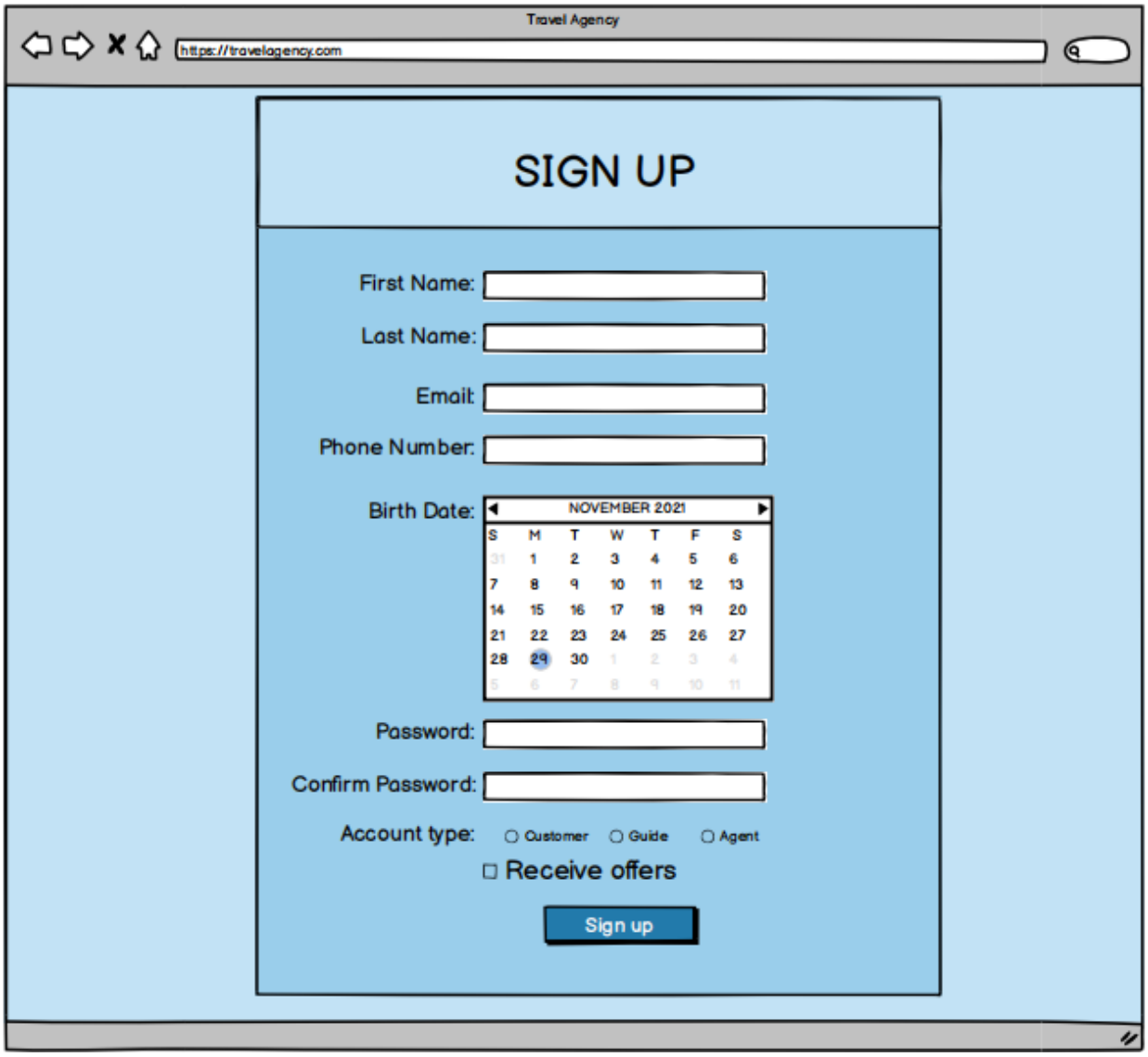

Figure 3: Register Page

3.1.1.2. SQL Queries

Check credentials

The following query string will return the row associated with the Customer if credentials are correct. The number of rows of this output can be used for the action to be done in the backend.

SELECT *

FROM Customer

WHERE Customer.user_name = @user_name **AND** Customer.password = @password;

However, to understand which type of user logged in to the system, the following queries should be also considered:

SELECT *

FROM Agent

WHERE Agent.user_name = @user_name **and** Agent.password = @password;

SELECT *

FROM Guide

WHERE Guide.user_name = @user_name **and** Guide.password = @password;

Check if the username exists

The following queries first check if the same username already exists, when a user tries to register to the system. If the number of rows returned is more than 1, the same username already exists. This query will be run before registering any user.

SELECT Sum.username

FROM (

(**SELECT** Customer.user_name

FROM Customer

WHERE Customer.user_name = @user_name) **UNION** (

SELECT Guide.user_name

FROM Guide

WHERE Guide.user_name = @user_name

) **UNION** (

SELECT Agent.user_name

FROM Agent

WHERE Agent.user_name = @user_name

)

) **AS** Sum;

Register a new user

After making sure that the new username is unique, the user will be registered to the system. From the checkbox, if the “Customer” option is checked, the following will be run:

INSERT INTO Customer

VALUES(@first_name, @last_name, @age, @email, @phone_number, @user_name, @password, @address);

If Guide is checked, this will be run instead.

INSERT INTO Guide

VALUES(@first_name, @last_name, @age, @email, @phone_number, @user_name, @password, @salary);

If Agent is checked, this will be run instead.

INSERT INTO Agent

VALUES(@first_name, @last_name, @age, @email, @phone_number, @user_name, @password, @salary);

3.1.2. Book a Tour

3.1.2.1. UI Mockups

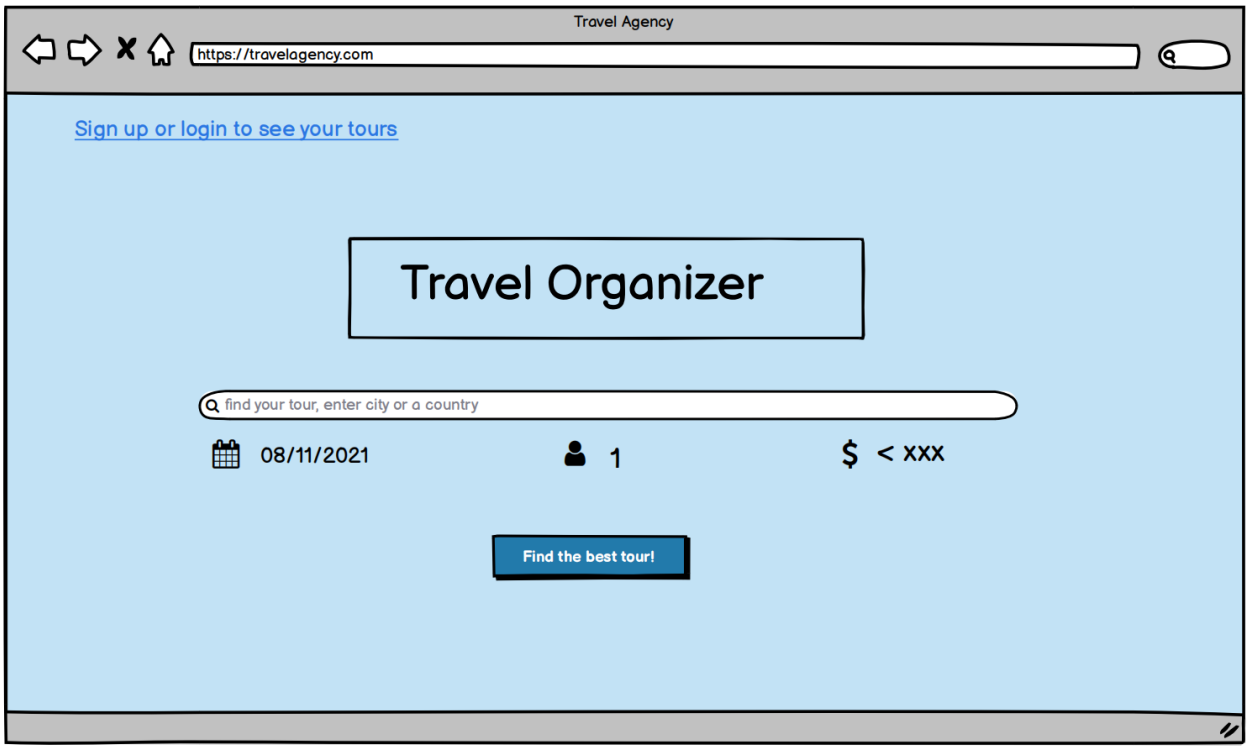

Figure 4: Main Page of the Project

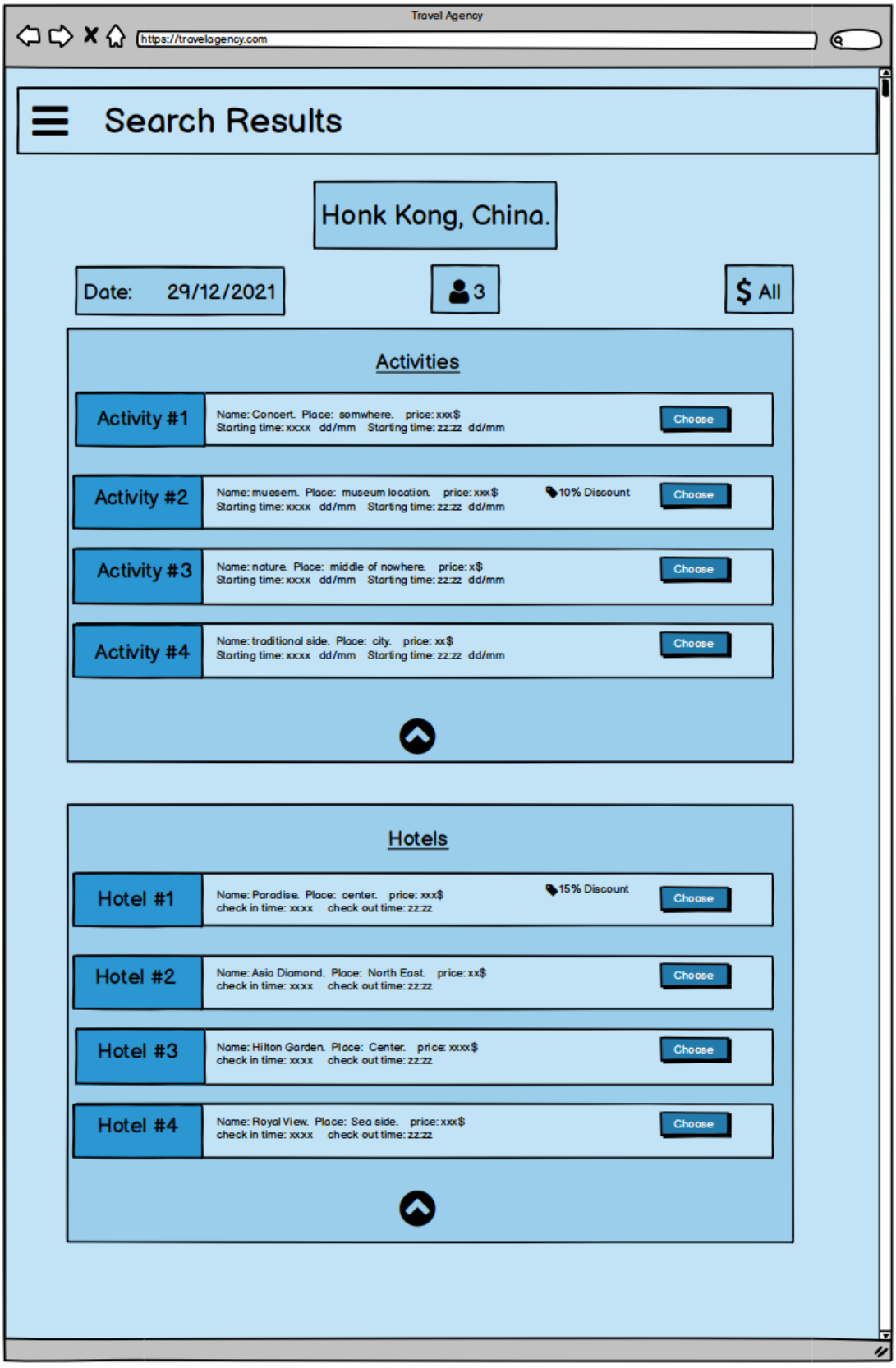

Figure 5: Results of the given search, where the city is Honk Kong, and the price of the tours is not limited

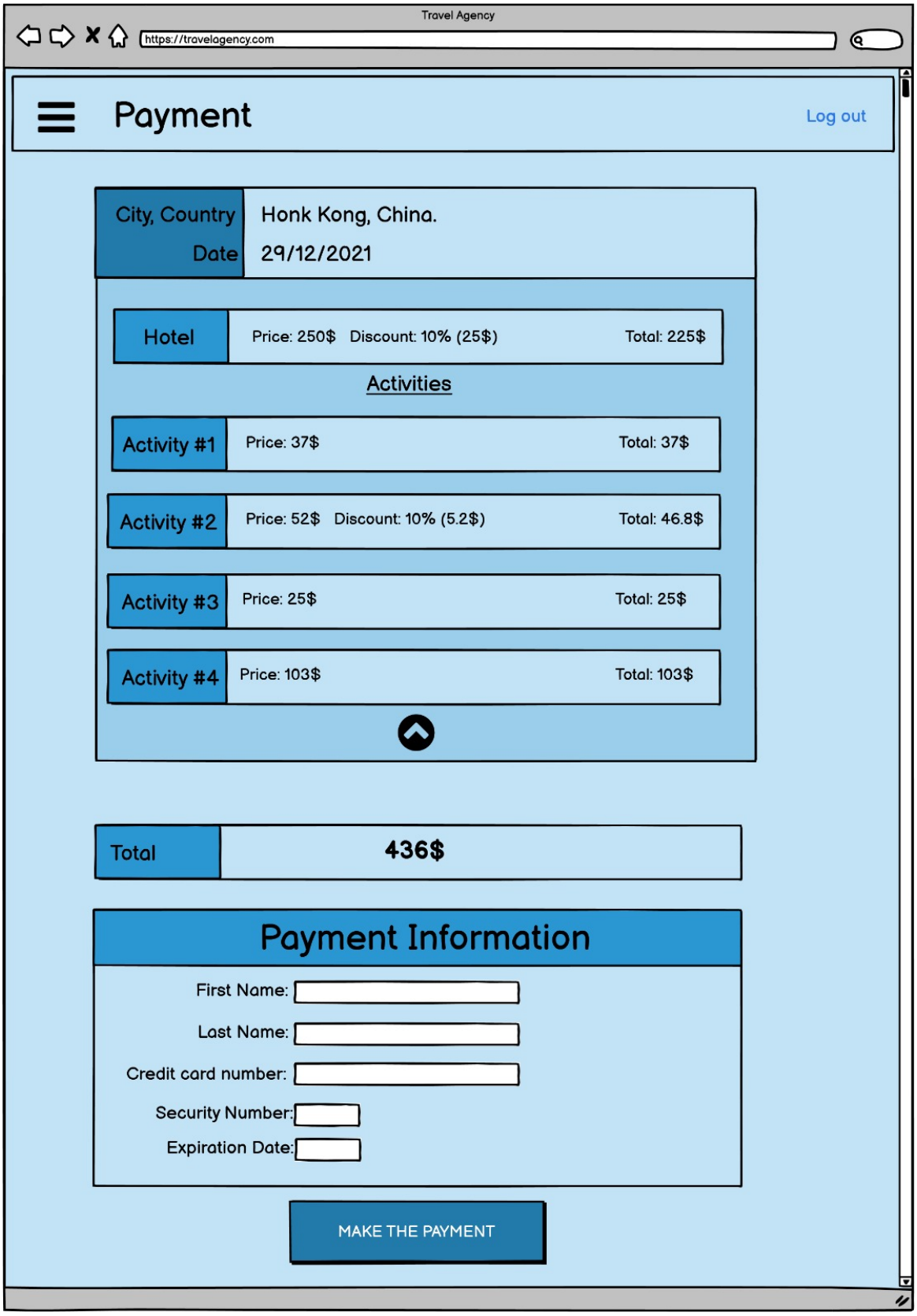

Figure 6: Payment page

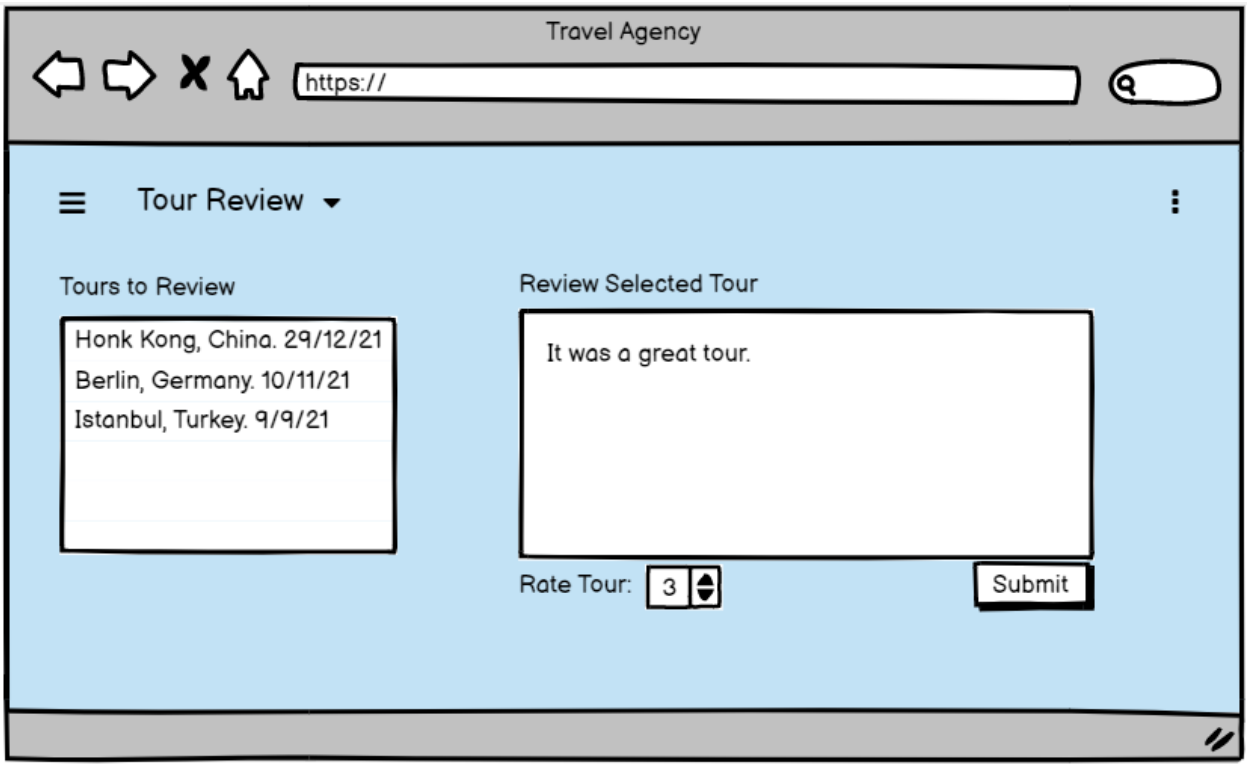

Figure 7: Tour Review Page

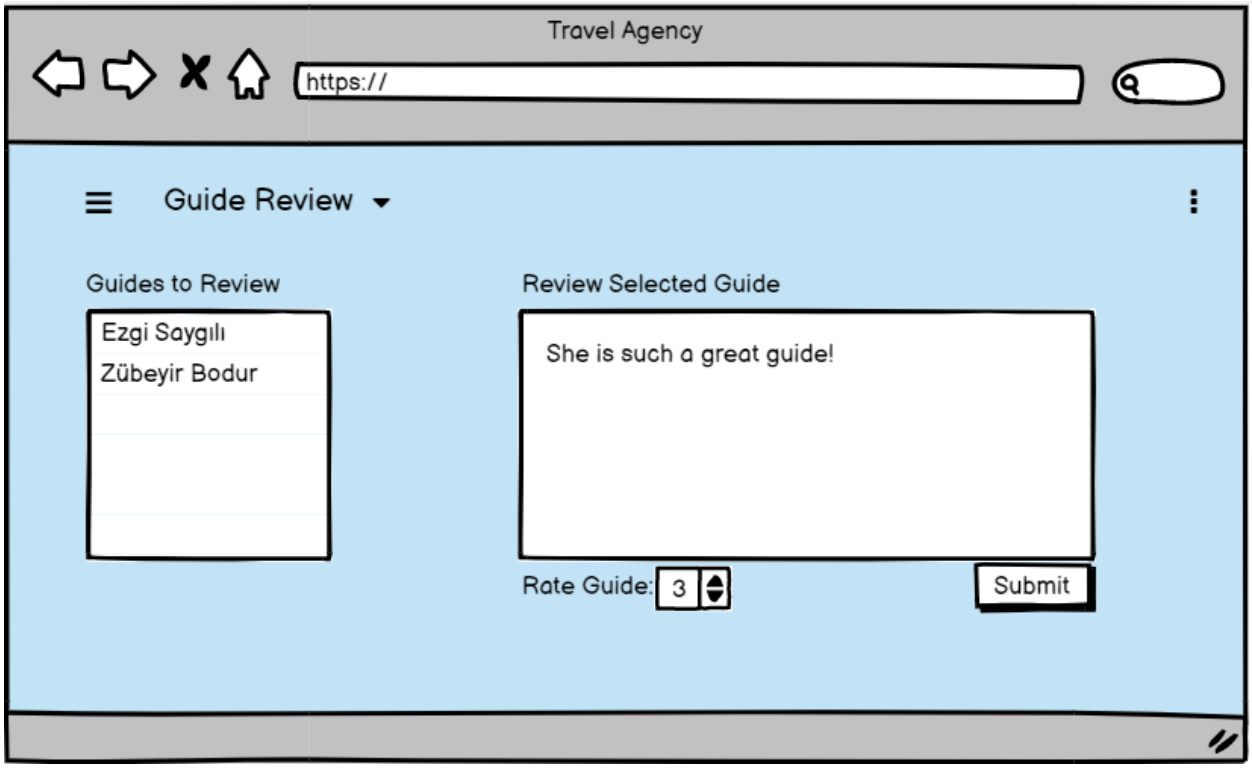

Figure 8: Guide Review Page

3.1.2.2. SQL Queries

List all available tours and apply filters

If a user passes any filter value to the query/procedure, it should consider that filter otherwise the query should ignore that filter and return all records when listing all available tours.

```
SELECT *
FROM Tour
WHERE (1=(CASE WHEN @tour_start_date IS NULL THEN 1 ELSE 0 END) OR [tour_start_date]=@tour_start_date)
AND (1=(CASE WHEN @city IS NULL THEN 1 ELSE 0 END)
OR [city]=@city) AND (1=(CASE WHEN @price IS NULL THEN 1 ELSE 0 END)
OR [price]=@price)
ORDER BY
(CASE
    WHEN tour_start_date IS NOT NULL AND city IS NULL AND price IS NULL THEN tour_start_date
    WHEN tour_start_date IS NULL AND city IS NOT NULL AND price IS NULL THEN city
    WHEN tour_start_date IS NULL AND city IS NULL AND price IS NOT NULL THEN price
    WHEN tour_start_date IS NOT NULL AND city IS NOT NULL AND price IS NULL THEN tour_start_date, city
     WHEN tour_start_date IS NOT NULL AND city IS NULL AND price IS NOT NULL THEN tour_start_date, price
     WHEN tour_start_date IS NULL AND city IS NOT NULL AND price IS NOT NULL THEN city, price
    ELSE tour_start_date, city, price
```

END) **ASC**, name **ASC**;

List all available activities of the selected tour

The following query will return the rows associated with the Activities corresponding to the selected tour.

SELECT activity_id, activity_name, description, acitivity_start_time, activity_end_time, activity_type

FROM Activity **NATURAL JOIN** Tour

WHERE activity_name = @activity_name AND activity_type = @activity_type;

Book the tour with the given parameters

The following query will update the associated tables so that a reservation is created for the booking operation. The row will differ from the reservations that are not paid with an “is_paid” boolean attribute, which in this case will be true.

This operation includes the number of people that are booking, selected tour, and selected start date of the customer.

INSERT INTO TourReservation(reserve_id, reserve_start_date, tour_id, num_reserving, is_booked)

VALUES(@reserve_id, @reserve_start_date, @tour_id, @num_reserving, 1);

Make the payment accordingly

The following query will update the rows associated with the Customer making the payment.

UPDATE Customer

SET wallet = wallet - @payment_amount

WHERE Customer.ID = @ID;

Give feedback (Review) about the tour and the guide

Review the tour

The following query will insert a new row associated with the Tour Review associated with the specified tour.

INSERT INTO tour_rev(tour_id, review_id, comment, entry_date, rating_stars

VALUES (@tour_id, @review_id, @comment, @entry_date, @rating_stars);

Review the guide

The following query will insert a new row associated with the Guide Review associated with the specified tour.

INSERT INTO guide_rev(ID, review_id, comment, entry_date, rating_stars)

VALUES (@ID, @review_id, @comment, @entry_date, @rating_stars);

3.2. Unique Functionalities

3.2.1. Sightseeing Places

3.2.1.1. UI Mockups

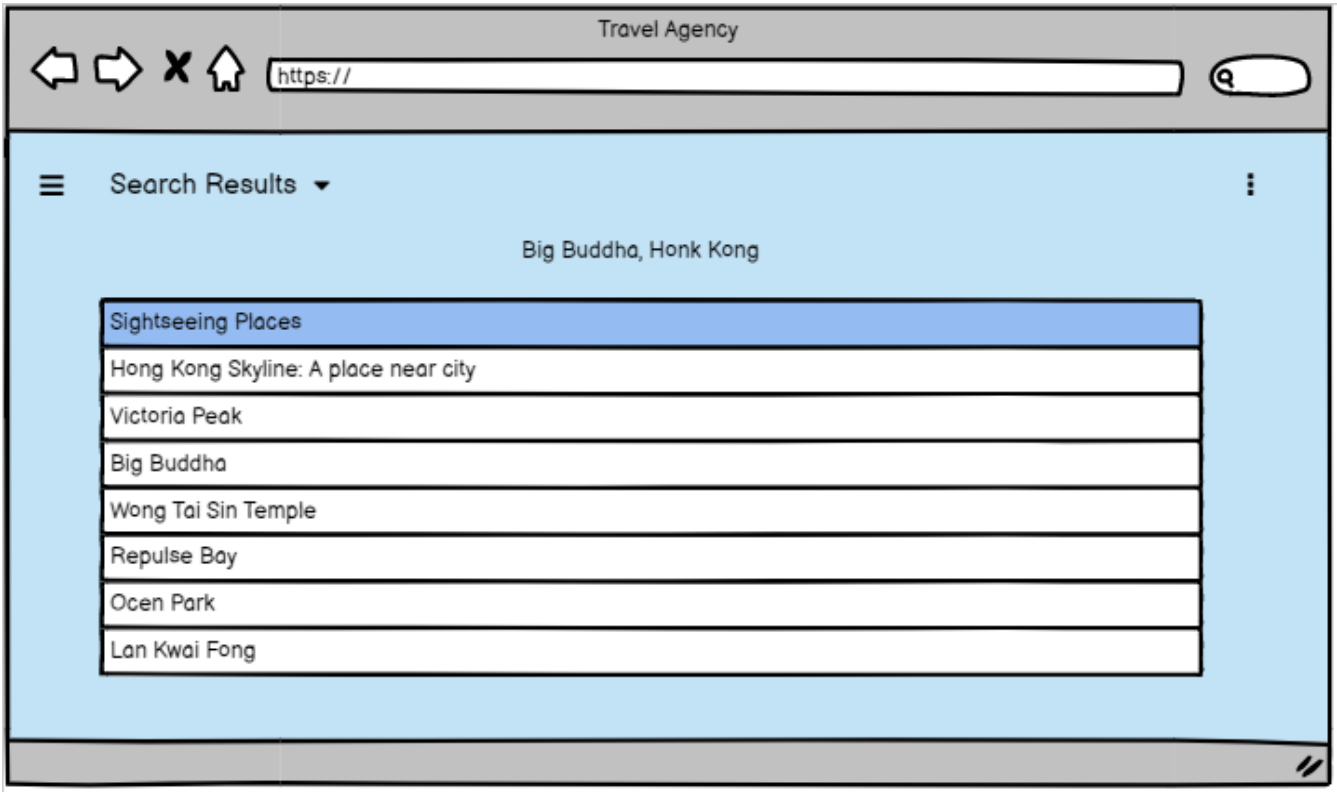

Figure 10: Sightseeing Places Preview page

Figure 11: Sightseeing Place information page

3.2.1.2. SQL Queries

List all Sightseeing Places Associated with the Tour

When the tours are listed, the user should be able to look at the sightseeing activities involved in a specific tour. Hence, this query will list all sightseeing places associated with that tour.

SELECT place_id, name, start_time, end_time, location, entry_fee, description

FROM Visits **NATURAL JOIN** SightseeingPlaces

WHERE tour_id = @tour_id;

3.2.2. Wallet

3.2.2.1. UI Mockups

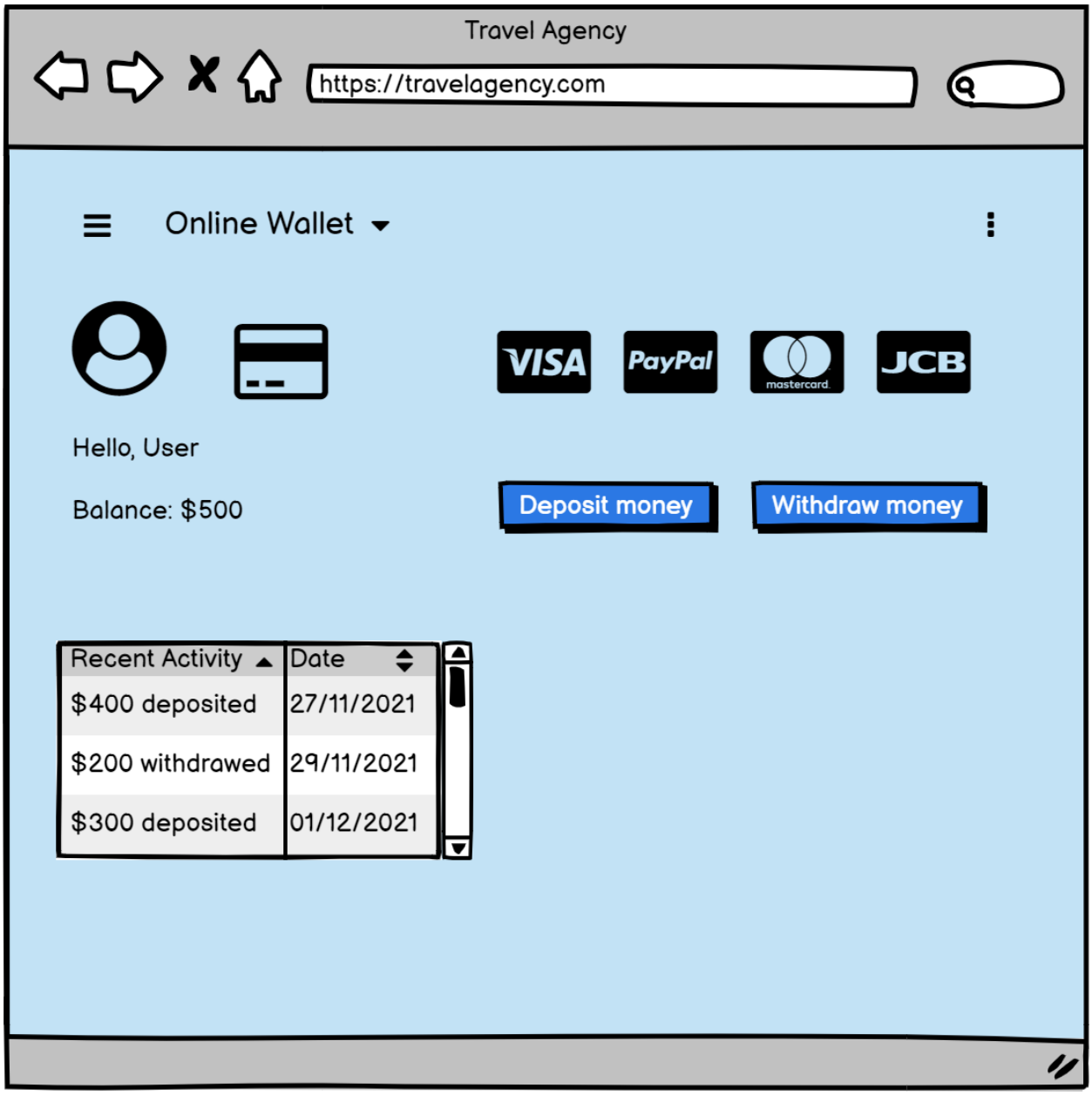

Figure 12: Online Wallet Page

3.2.2.2. SQL Queries

Customers make all of their payments from their online wallets. They can deposit money into this wallet, and when they book a tour or hotel, the payment is done from this wallet (See 3.1.2.2. - Make the Payment Accordingly).

Deposit Cash into the Wallet

UPDATE Customer

SET Customer.wallet = Customer.wallet + @deposit_amount

WHERE Customer.username = @username;

3.2.3. Applying Discounts

3.2.3.1. UI Mockups

Below, Agent Mustafa has created four discount templates, which are stored in the Discount table of the system. Mustafa can apply those discounts into their associated entities, by clicking apply. The application of those discounts are described in the next pages.

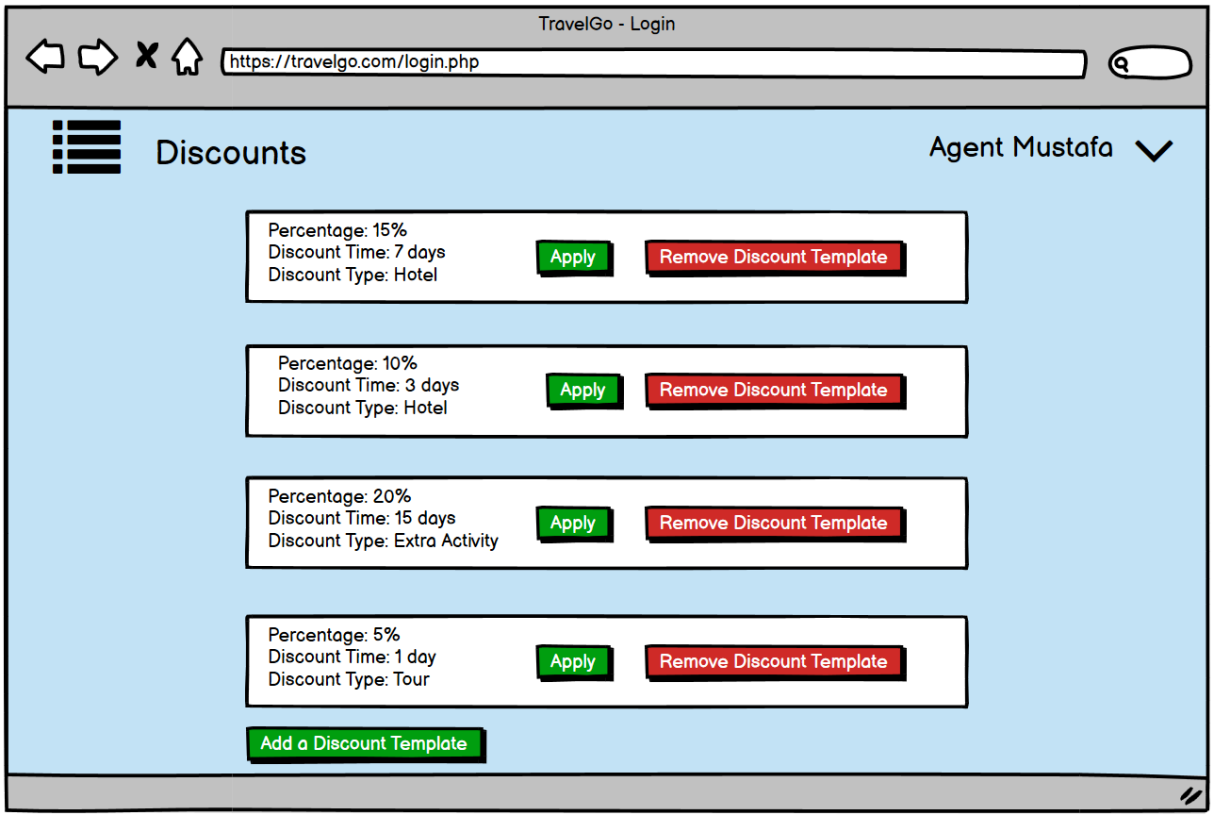

Figure 13: Managing Discounts

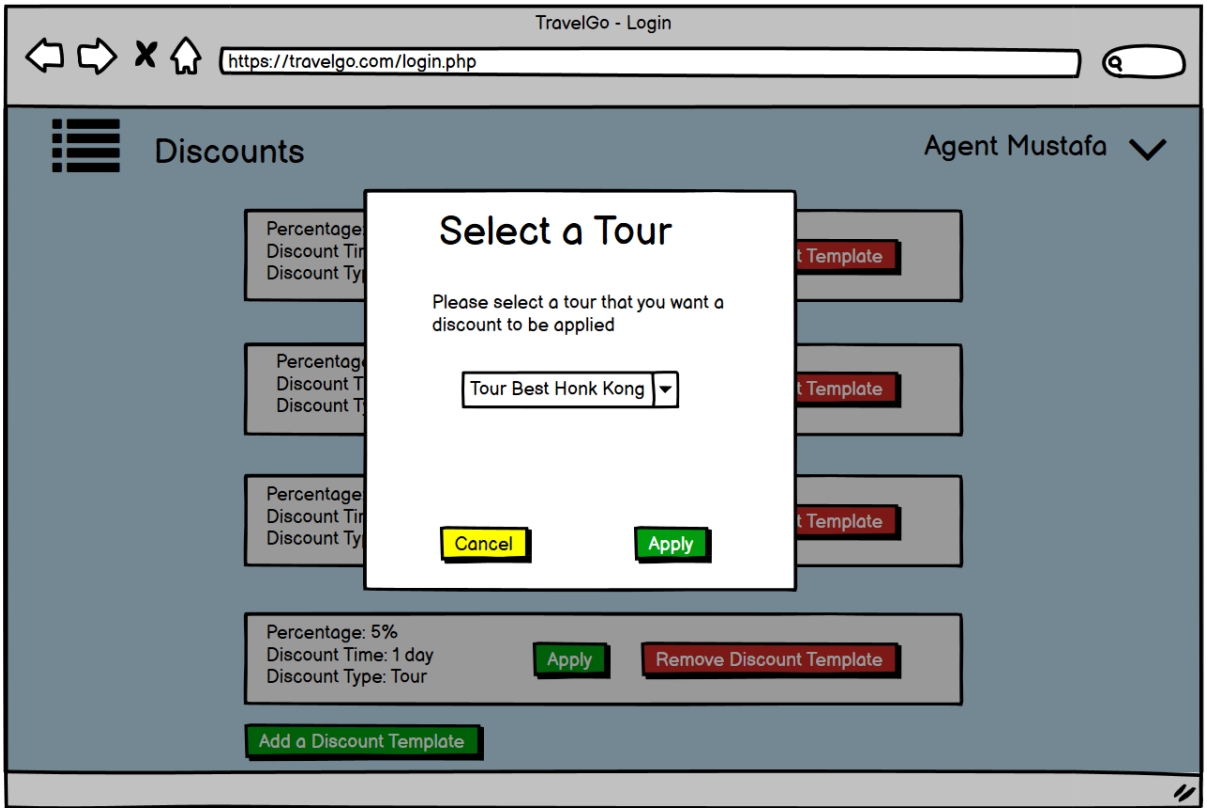

Figure 14 : Agent Mustafa applies a discount to the Tour "Tour Best Honk Kong" from a discount template

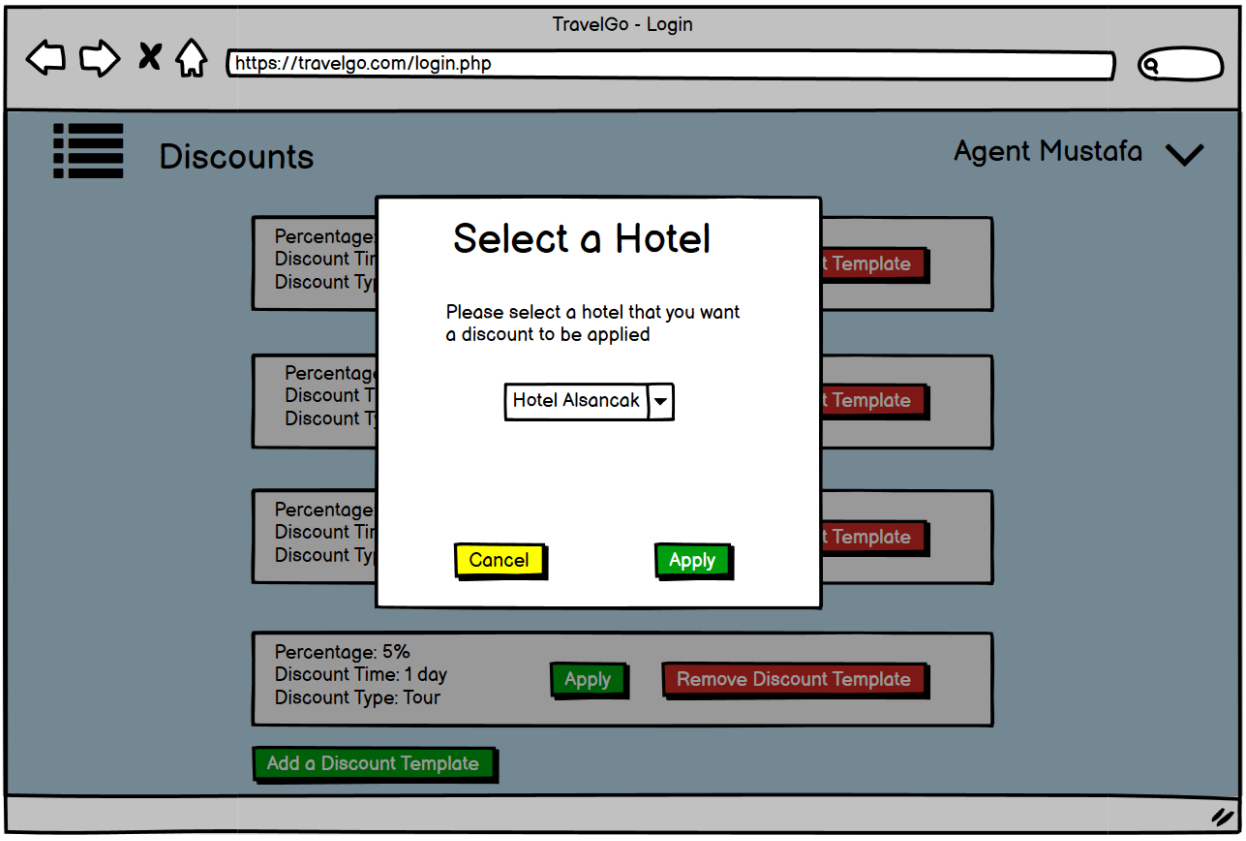

Figure 15: Agent Mustafa applies a discount to the "Hotel Alsancak" from a discount template

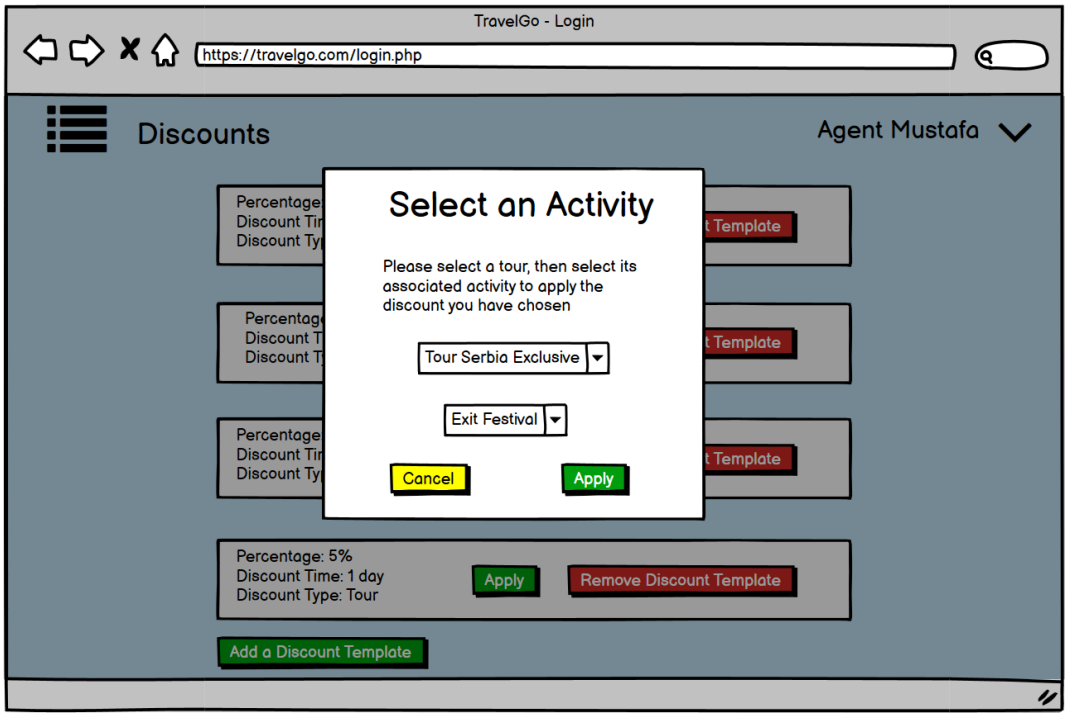

Figure 16: Agent Mustafa applies a discount to the "Exit Festival" from a discount template

3.2.3.2. SQL Queries

An agent can apply discounts to the hotel reservations, tour reservations, and extra activities of a specific tour. There are predetermined discounts, which are located in the Discount table. The agent can also create new discounts.

The discounts to hotels are done in bulk. The entry is saved into the Discount table, then it is linked with the Hotel table. The agent can also choose to apply from discounts that have already been done.

Check if the Current User is an Agent

```
SELECT *
FROM Agent
WHERE Agent.user_name = @user_name;
```

Create a Discount Template

INSERT INTO Discount

VALUES(@discount_id, @percentage, @discount_time_interval, @discount_strart_date, @discount_type);

Apply a Discount to Hotels

UPDATE Hotel

SET discount_id = @discount_id

WHERE hotel_ID = @hotel_id;

Apply a Discount to Tours

UPDATE Tour

SET Tour.discount_id = @discount_id

WHERE Tour.tour_id = @tour_id;

Apply a Discount on Extra Activities

UPDATE Activity

SET Activity.discount_id = @discount_id

WHERE activity_id = @activity_id;

3.2.4. Languages of Guides

3.2.4.1. UI Mockups

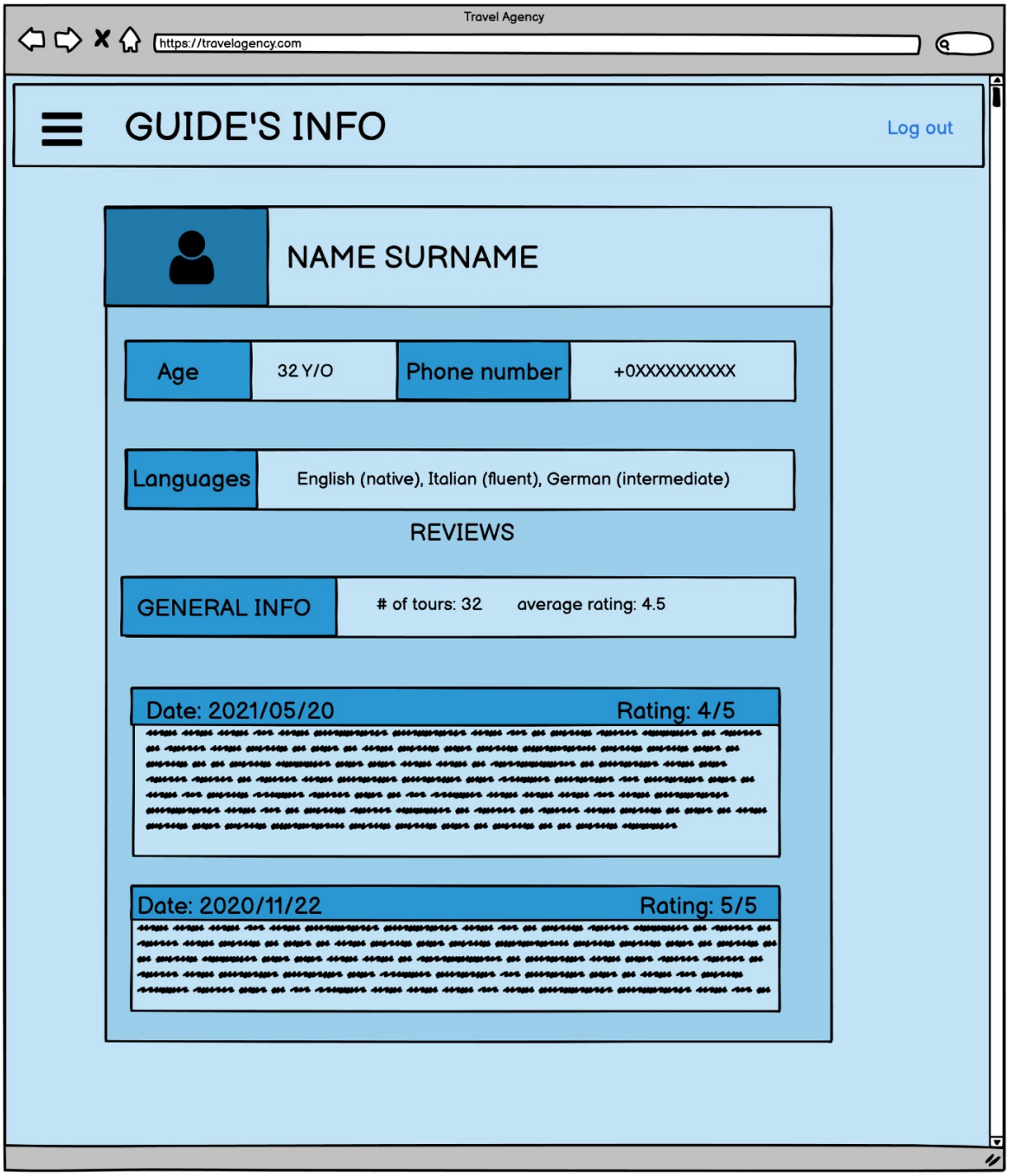

Figure 17: Guide Information Page

3.2.4.2. SQL Queries

List the name of the Languages a Guide Can Speak

Customers will see the languages their guides can speak, as well as their proficiency, through the website. The following query will list those.

SELECT language_name, proficiency

FROM Language

WHERE Language.ID = @guide_id;

4. Implementation Plan

We are going to use the MySQL server in our project as a database management system. In the backend of the website, we will use .NET Core, and in the front-end, we may use jQuery as a front-end framework. As a CSS library, we will use Bootstrap as well. In addition, HTML, CSS, and Javascript will be used throughout the project.

5. Website

The project website [1] is the following: https://cs353-travel-agency-system.github.io/

6. References

[1] “Travel Agency Management System | https://cs353-travel-agency-system.github.io,” *GitHub Pages,* [Online]. Available : https://cs353-travel-agency-system.github.io/. [Accessed Oct. 20, 2021].