2017-2018 Spring
CS 353 Database Systems

BilinkedIn
Project Design Report

GROUP 40:
Ata Gün Öğün - 21501187
Deniz Sipahioğlu - 21501526
Deniz Ulusel - 21503896
Mehmet Enes Keleş - 21501348

2.04.2018

Available at: https://github.com/dsipahioglu/bilinkedin

Table of Contents

1. REVISED E/R DIAGRAM

We made the following changes in our model since the project proposal. Our assistant's reviews were taken into account.

- "**email**" attribute added to "**User**"
- "**phone_number**" attribute removed from "**User**"
- "**minimumcost**" attribute removed from "**JobAnnouncement**"
- All attribute names converted to snake_case for consistency.
- Name of "**Admin**" entity changed as "**CustomerSupport**".
- Disjoint specialization was applied to the "**Review**" entity, creating two new entities "**ProfessionalReview**" and "**CustomerReview**". "**ProfessionalReview**" denotes the review that the professional receives and "**CustomerReview**" denotes the review that the customer receives.
- Disjoint specialization was applied to the "**Chat**" entity, creating three new entities "**CommissionChat**","**TicketChat**" and "**OfferChat**". This allowed us to better describe the participation constraints of "**Chat**" entities in several different relations.
- Derived attributes not shown on the ER diagram as per our assistants request.
- Some cardinality and participation constraints are shown using numbers to provide clarity.

  The revised E/R diagram is given in Figure 1.

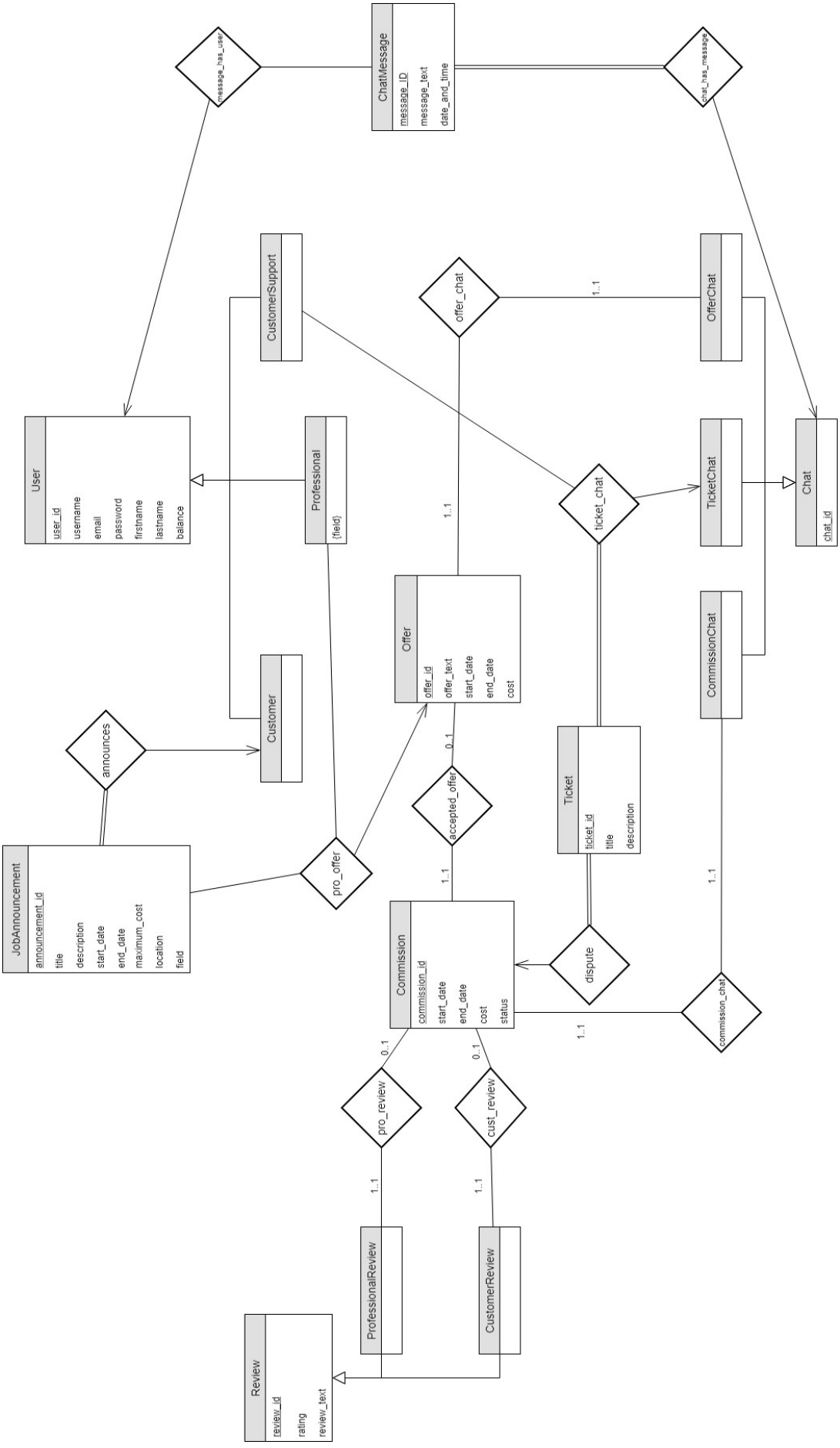

Figure 1: Revised E/R Diagram

2. RELATION SCHEMAS

2.1. User

Relational Model:
User(user_id)

Functional Dependencies:
-

Candidate Keys:
{(user_id)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE User (
     user_id   INT PRIMARY KEY AUTO_INCREMENT
);
```

2.2. Customer

Relational Model:
Customer(user_id, username, password, email, firstname, lastname, balance)

Functional Dependencies:
user_id -> username, password, email, firstname, lastname, balance
email -> user_id, username, password, firstname, lastname, balance

Candidate Keys:
{(user_id),(email)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE Customer (
     user_id     INT PRIMARY KEY,
     username    VARCHAR(32) NOT NULL,
     password    VARCHAR(32) NOT NULL,
     email       VARCHAR(64) NOT NULL UNIQUE,
     firstname   VARCHAR(32) NOT NULL,
     lastname    VARCHAR(32) NOT NULL,
     balance     FLOAT       NOT NULL,
     FOREIGN KEY(user_id) REFERENCES User
);
```

2.3. Professional

Relational Model:
Professional(user_id, username, password, email, firstname, lastname, balance)

Functional Dependencies:
user_id -> username, password, email, firstname, lastname, balance
email -> user_id, username, password, firstname, lastname, balance

Candidate Keys:
{(user_id),(email)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE Professional (
     user_id    INT PRIMARY KEY,
     username   VARCHAR(32) NOT NULL,
     password   VARCHAR(32) NOT NULL,
     email      VARCHAR(64) NOT NULL UNIQUE,
     firstname  VARCHAR(32) NOT NULL,
     lastname   VARCHAR(32) NOT NULL,
     balance    FLOAT       NOT NULL,
     FOREIGN KEY(user_id) REFERENCES User
);
```

2.3.1. ProfessionalField

Relational Model:
ProfessionalField(user_id, field)

Functional Dependencies:
-

Candidate Keys:
{(user_id, field)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE ProfessionalField (
     user_id  INT NOT NULL,
     field    VARCHAR(32) NOT NULL,
     PRIMARY KEY(user_id, field),
     FOREIGN KEY(user_id) REFERENCES Professional
);
```

2.4. CustomerSupport

Relational Model:
CustomerSupport(user_id, username, password, email, firstname, lastname, balance)

Functional Dependencies:
user_id -> username, password, email, firstname, lastname, balance
email -> user_id, username, password, firstname, lastname, balance

Candidate Keys:
{(user_id),(email)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE CustomerSupport (
     user_id   INT PRIMARY KEY,
     username   VARCHAR(32) NOT NULL,
     password   VARCHAR(32) NOT NULL,
     email      VARCHAR(64) NOT NULL UNIQUE,
     firstname  VARCHAR(32) NOT NULL,
     lastname   VARCHAR(32) NOT NULL,
     balance    FLOAT       NOT NULL,
     FOREIGN KEY(user_id) REFERENCES User
);
```

2.5. JobAnnouncement

Relational Model:
JobAnnouncement(announcement_id, user_id, title, description, start_date, end_date, maximum_cost, location, field)

Functional Dependencies:
announcement_id -> title, description, start_date, end_date, maximum_cost, location, field

Candidate Keys:
{(announcement_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE JobAnnouncement (
     announcement_id   INT PRIMARY KEY AUTO_INCREMENT,
     user_id           INT           NOT NULL,
     title             VARCHAR(32)   NOT NULL,
     description       VARCHAR(256)  NOT NULL,
     start_date        DATE,
     end_date          DATE,
     maximum_cost      FLOAT,
     location          VARCHAR(128)  NOT NULL,
     field             VARCHAR(32)   NOT NULL,
     FOREIGN KEY(user_id) REFERENCES Customer
);
```

2.6. Offer

Relational Model:
Offer(offer_id, chat_id, offer_text, start_date, end_date, cost)

Functional Dependencies:
offer_id -> chat_id, offer_text, start_date, end_date, cost
chat_id -> offer_id, offer_text, start_date, end_date, cost

Candidate Keys:
{(offer_id), (chat_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE Offer (
     offer_id     INT PRIMARY KEY AUTO_INCREMENT,
     chat_id      INT UNIQUE   NOT NULL,
     offer_text   VARCHAR(256) NOT NULL,
     start_date   DATE,
     end_date     DATE,
     cost         FLOAT        NOT NULL,
     FOREIGN KEY(chat_id) REFERENCES OfferChat,
);
```

2.6.1. pro_offer

Relational Model:
pro_offer(offer_id, user_id, announcement_id)

Functional Dependencies:
offer_id -> user_id, announcement_id

Candidate Keys:
{(offer_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE pro_offer (
     offer_id         INT PRIMARY KEY,
     user_id          INT NOT NULL,
     announcement_id   INT NOT NULL,
     FOREIGN KEY(offer_id)         REFERENCES Offer,
     FOREIGN KEY(user_id)          REFERENCES Professional,
     FOREIGN KEY(announcement_id) REFERENCES JobAnnouncement
);
```

2.7. Commission

Relational Model:
Commission(commission_id, offer_id, chat_id, start_date, end_date, cost, status)

Functional Dependencies:
commission_id -> start_date, end_date, cost, status, chat_id, offer_id
offer_id -> start_date, end_date, cost, status, chat_id, commission_id
chat_id -> start_date, end_date, cost, status, offer_id, commission_id
Candidate Keys:
{(commission_id),(chat_id),(offer_id)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE Commission (
     commission_id   INT PRIMARY KEY AUTO_INCREMENT,
     offer_id        INT UNIQUE   NOT NULL,
     chat_id         INT UNIQUE   NOT NULL,
     start_date      DATE,
     end_date        DATE,
     cost                 FLOAT        NOT NULL,
     status          VARCHAR(32)  NOT NULL,
     FOREIGN KEY(offer_id) REFERENCES Offer,
     FOREIGN KEY(chat_id)  REFERENCES CommissionChat
);
```

2.8. ProfessionalReview

Relational Model:
ProfessionalReview(review_id, commission_id, rating, review_text)

Functional Dependencies:
review_id -> commission_id, rating, review_text
commission_id -> review_id, rating, review_text

Candidate Keys:
{(review_id), (commission_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE ProfessionalReview (
     review_id       INT PRIMARY KEY AUTO_INCREMENT,
     commission_id    INT UNIQUE     NOT NULL,
     rating          FLOAT          NOT NULL,
     review_text     VARCHAR(512)   NOT NULL,
     FOREIGN KEY(commission_id) REFERENCES Commission
);
```

2.9. CustomerReview

Relational Model:
CustomerReview(review_id, commission_id, rating, review_text)

Functional Dependencies:
review_id -> commission_id, rating, review_text
commission_id -> review_id, rating, review_text

Candidate Keys:
{(review_id), (commission_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE CustomerReview (
     review_id       INT PRIMARY KEY AUTO_INCREMENT,
     commission_id    INT UNIQUE     NOT NULL,
     rating          FLOAT          NOT NULL,
     review_text     VARCHAR(512)   NOT NULL,
     FOREIGN KEY(commission_id) REFERENCES Commission
);
```

2.10. Ticket

Relational Model:
Ticket(ticket_id, commission_id, title, description)

Functional Dependencies:
ticket_id -> commission_id, title, description

Candidate Keys:
{(ticket_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE Ticket (
     ticket_id       INT PRIMARY KEY AUTO_INCREMENT,
     commission_id    INT           NOT NULL,
     title           VARCHAR(32)   NOT NULL,
     description     VARCHAR(512) NOT NULL,
     FOREIGN KEY(commission_id) REFERENCES Commission
);
```

2.10.1. ticket_chat

Relational Model:
ticket_chat(ticket_id, chat_id, user_id)

Functional Dependencies:
ticket_id -> chat_id, user_id
chat_id -> ticket_id, user_id

Candidate Keys:
{(ticket_id), (chat_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE ticket_chat (
     ticket_id   INT PRIMARY KEY,
     chat_id     INT UNIQUE NOT NULL,
     user_id     INT,
     FOREIGN KEY(ticket_id) REFERENCES Ticket,
     FOREIGN KEY(chat_id)   REFERENCES TicketChat,
     FOREIGN KEY(user_id)   REFERENCES CustomerSupport
);
```

2.11. Chat

Relational Model:
Chat(chat_id)

Functional Dependencies:
–

Candidate Keys:
{(chat_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE Chat (
     chat_id INT PRIMARY KEY AUTO_INCREMENT
);
```

2.12. CommissionChat

Relational Model:
CommissionChat(chat_id)

Functional Dependencies:
–

Candidate Keys:
{(chat_id)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE CommissionChat (
     chat_id INT PRIMARY KEY,
     FOREIGN KEY(chat_id) REFERENCES Chat
);
```

2.13. TicketChat

Relational Model:
TicketChat(chat_id)

Functional Dependencies:
–

Candidate Keys:
{(chat_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE TicketChat (
     chat_id INT PRIMARY KEY,
     FOREIGN KEY(chat_id) REFERENCES Chat
);
```

2.14. OfferChat

Relational Model:
OfferChat(chat_id)

Functional Dependencies:
–

Candidate Keys:
{(chat_id)}

Normal Form:
BCNF

Table Definition:
```
CREATE TABLE OfferChat (
     chat_id INT PRIMARY KEY,
     FOREIGN KEY(chat_id) REFERENCES Chat
);
```

2.15. Chat Message

Relational Model:
ChatMessage(message_id, chat_id, user_id, message_text, date_and_time)

Functional Dependencies:
message_id -> chat_id, user_id, message_text, date_and_time

Candidate Keys:
{(message_id)}

Normal Form:
BCNF

Table Definition:

```
CREATE TABLE ChatMessage (
     message_id    INT PRIMARY KEY AUTO_INCREMENT,
     chat_id       INT           NOT NULL,
     user_id       INT           NOT NULL,
     message_text  VARCHAR(256) NOT NULL,
     date_and_time          TIMESTAMP    NOT NULL,
     FOREIGN KEY(chat_id) REFERENCES Chat,
     FOREIGN KEY(user_id) REFERENCES User
);
```

3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

Functional dependencies and normal forms are shown in Relational Schemas section of our report. Since all of the tables in our design is in Boyce-Codd Normal Form, there is no need for normalization.

4. FUNCTIONAL COMPONENTS

4.1. Use Cases

BilinkedIn is a system that allows people to put up announcements of jobs they want done, and to contact people who want to do these jobs. We have three types of users: Customer, Professional and Customer Support. Use case diagrams for these three types of users are given in Figure 2, Figure 3 and Figure 4.

4.1.1. Customer

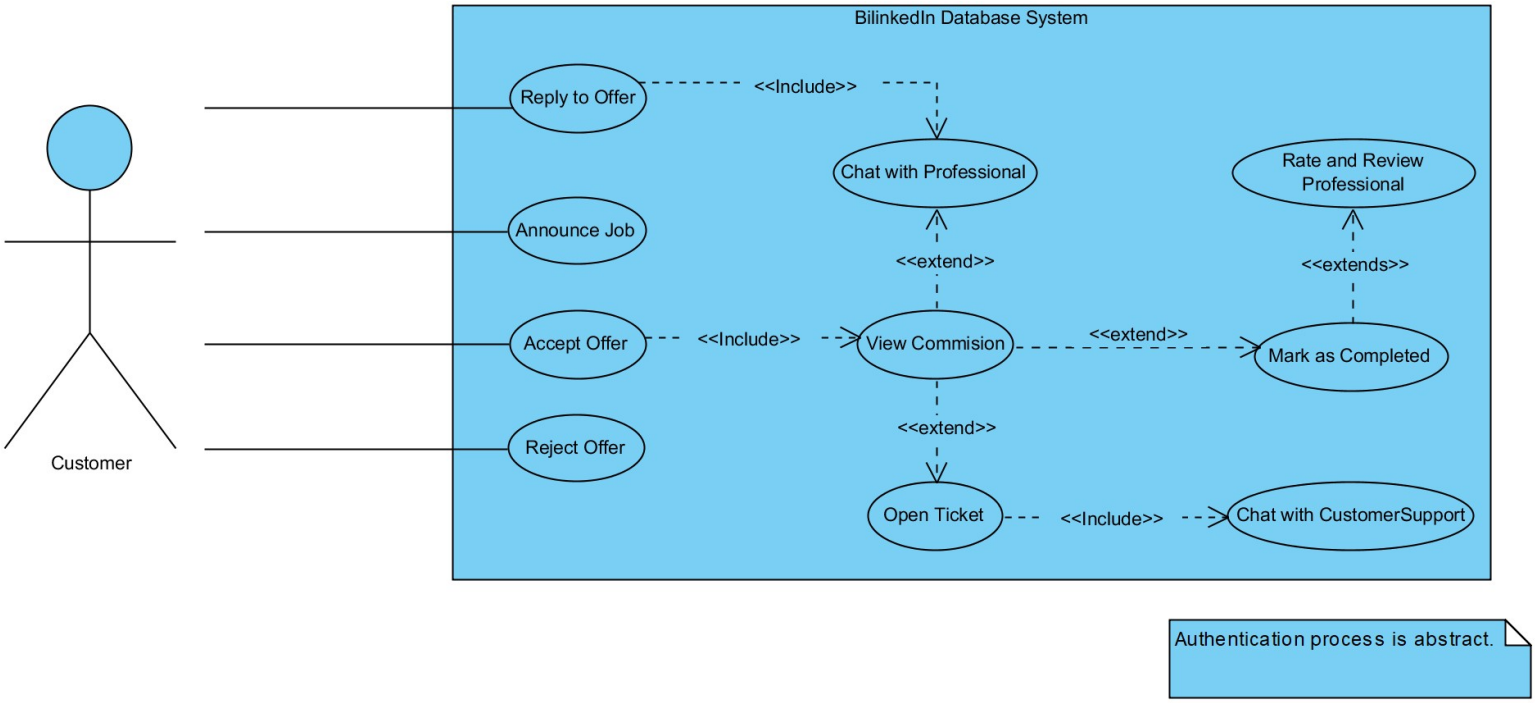

Figure 2: Customer Use Case Diagram

Announce Job: Customers can put on job announcements which includes details about the job they want to get done. Here, they enter the field, description, start date, end date and the cost information about the job.

Reply to Offer: Customers can reply to the offers of professionals who are interested in taking the job.

Chat with Professional: Customers can chat with the professionals regarding their offers and ongoing commissions.

Accept Offer: Customers can accept the offers if they agree with the service that a professional is providing. Consequentially a commission, an ongoing job, is opened.

Reject Offer: Customers can reject the offers that are made to their job announcements.

View Commision: Customers can see the details about their current commissions.

Open Ticket: Customers can open a ticket about a commission if they have a complaint about a professional they hired.

Mark as Completed: Customers can mark the commissions as completed if they think that the job is done.

Chat with Customer Support: Customers can chat with the customer support regarding their complaints about a professional they hired.

Rate and Review Professional: Upon completion of a job, the customers can rate and review a professionals' performance during the job.

4.1.2. Customer Support

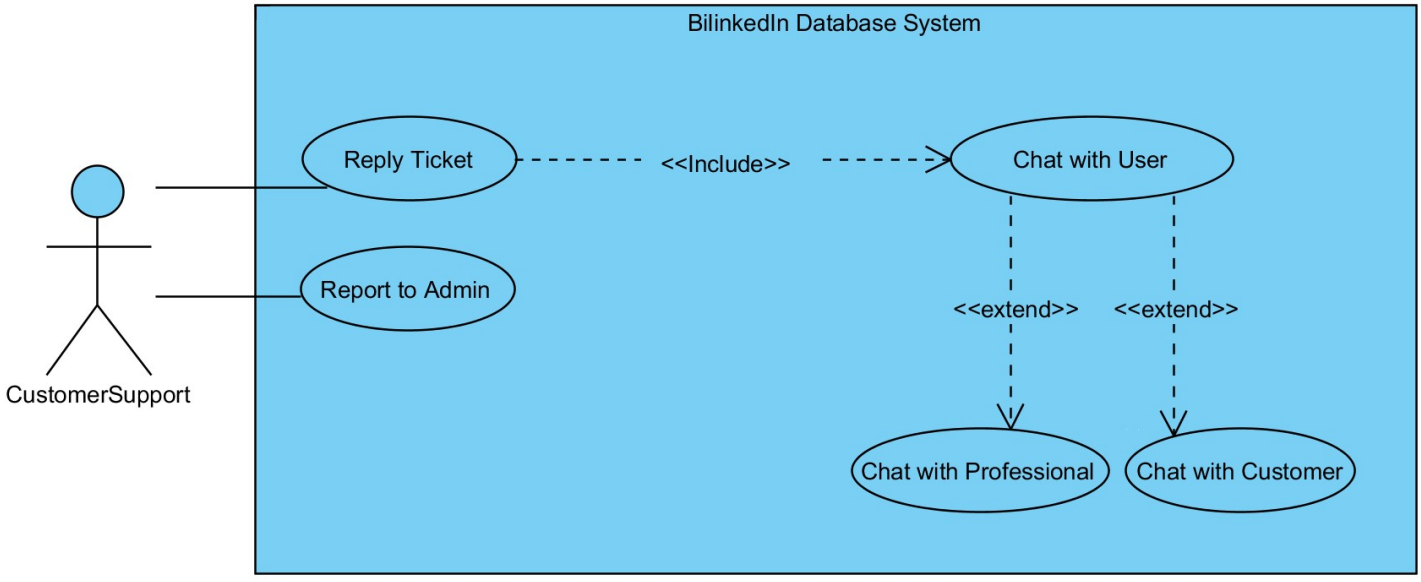

Figure 3: CustomerSupport Use Case Diagram

Reply Ticket: Customer supports can reply to the tickets opened by the professionals and the customers about the commissions or the complaints that they might have.

Report to Admin: Customer supports can report the users to the admins/authorities if they are convinced that there is a problem after the manual investigation of the ticket.

Chat with User: Customer supports can chat with the customers and the professionals.

Chat with Professional: Customer Supports can chat with the professionals about the issues they have about a customer they hired upon receiving a ticket.

Chat with Customer: Customer Supports can chat with the customers about the issues they have about a professional they hired upon receiving a ticket.

4.1.3. Professional

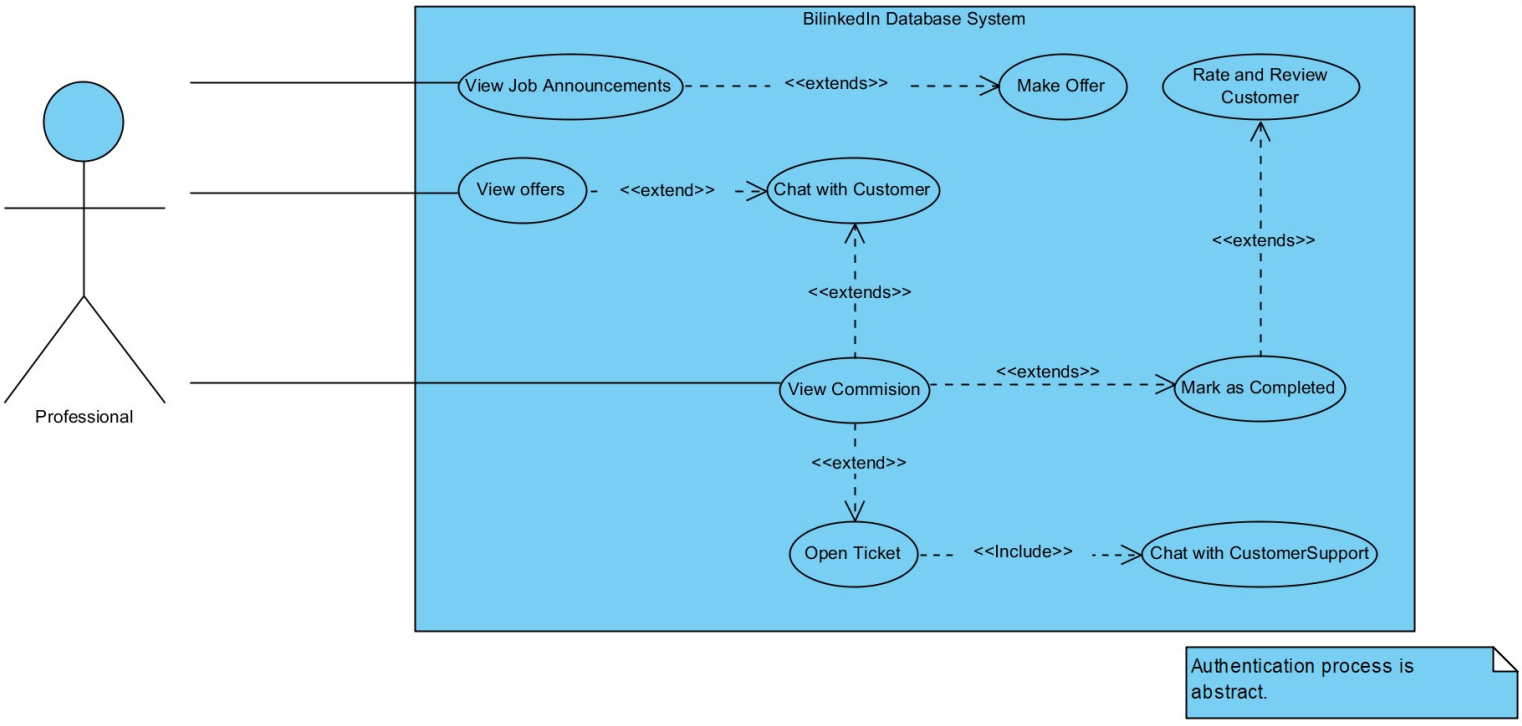

Figure 4: Professional Use Case Diagram

View Job Announcements: Professionals can view the job announcements put on by the customers, which include details about the announced job. They can see the field, description, start date, end date and the cost information about the job.

Make Offer: Professionals can make offers to the announcements of customers if they are interested in taking the job.

View Offers: Professionals can view the offers that they made.

Chat with Customer: Professionals can chat with the customers regarding their ongoing commissions and the offers they made.

View Commission: Professionals can view the commissions that they made.

Mark as Completed: Professionals can mark the commissions as completed if they think that the job is done.

Rate and Review Customer: Upon completion of a job, the professionals can rate and review a customer.

Open Ticket: Professionals can open a ticket about a commission if they have a complaint about a customer who hired them.

Chat with Customer Support: Professionals can chat with the customer support regarding their complaints about a customer who hired them.

4.2. Algorithms

4.2.1. User-Related Algorithms

The BilinkedIn Database system will make sure that the users will be able to edit their personal information except their name and the surname.

To calculate the average rating of a professional, his completed commissions will be taken into account. Customer reviews that belong to the professional's completed commissions will be used. A similar method is used also for the customers.

4.2.2. Commission-Related Algorithms

To manage the transactions, after the job is done and it is marked as completed by both the customer and the professional, the amount will be transferred from the customer's balance to the professional's balance.

Other than this, the customer won't be able to offer a higher amount than his balance.

The professionals will be able to rate and review the customer after the job is done. The customers will also be able to rate and review the professional that they hired, regarding the professional's performance.

4.2.3. Offer-Related Algorithms

The offers that are rejected by the customers are deleted from the system.

The accepted offers turn into commissions.

4.2.4. Announcement-Related Algorithms

The job announcements with a past end-date that weren't done by any of the professionals are deleted from the system.

4.2.5. Customer Support-Related Algorithms

In order for the tickets opened by the customers and professionals to be handled, a customer support is assigned to each one of the tickets. For the tickets to be managed, the customer support that has the least ongoing tickets at that time is assigned to an new coming ticket. If they have the same amount of ongoing tickets, the customer support that has the smaller id will be assigned to the new coming ticket.

4.2.6. Logical Requirements

To make sure that there are no logical errors in the BilinkedIn Database System, there are some possible issues that should be handled beforehand. First one of these is the cost constraints in job announcements.

The cost of a job should not have a negative value since it reflects the possible amount to be paid by the customer to the professional who takes the job. Other than this, if the cost of an offer that a professional made is not between “0 - cost determined by the user”, it won’t be accepted.

The second one is about the end and the start dates of announced jobs. The start-date shouldn’t be after the end-date, and the end-date shouldn’t be before the start-date of a job announcement.

4.3. Data Structures

The attribute domains that are used are Numeric Types, Date and Time Types and String Types of MySQL.

Numeric types are used to store the numeric data like the id’s, balance, cost and rating. **INT** and **FLOAT** domains are used.

Date and Time types are used to store the end and start dates of job announcements, offers, commissions and to store the timestamps of chats. **DATE** and **TIMESTAMP** domains are used.

String types are used to store the username, password, email, first name, last name, field, location, offer, status, review, title and description of tickets, and message text in chat messages. **VARCHAR(32)**, **VARCHAR(64)**, **VARCHAR(128)**, **VARCHAR(256)** and **VARCHAR(512)** domains are used.

5. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

5.1. Login Screen

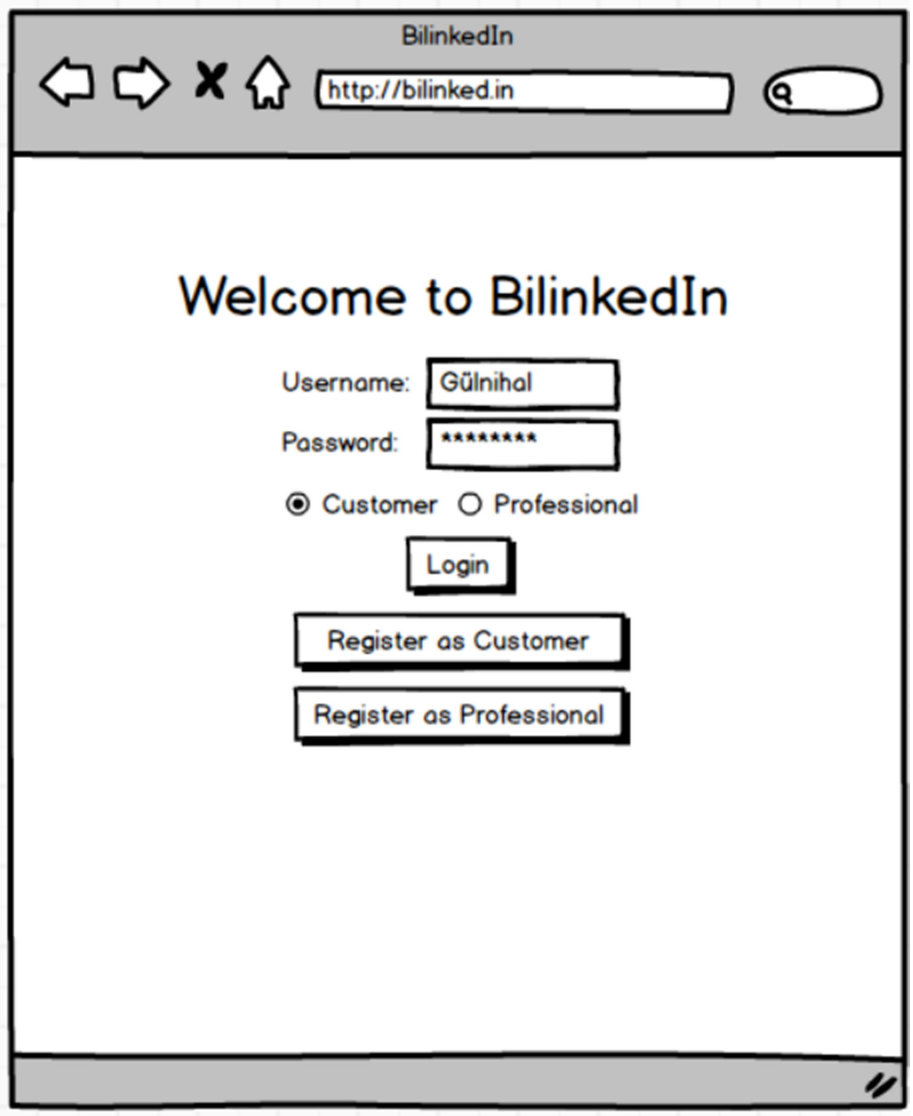

Figure 5: The login screen of the BilinkedIn Website

```
-- entry page
-- Inputs: @username, @password

SELECT user_id
FROM Customer
WHERE username = @username AND
      password = @password
```

```
SELECT user_id
FROM Professional
WHERE username = @username AND
      password = @password
```

5.2. Register as Customer

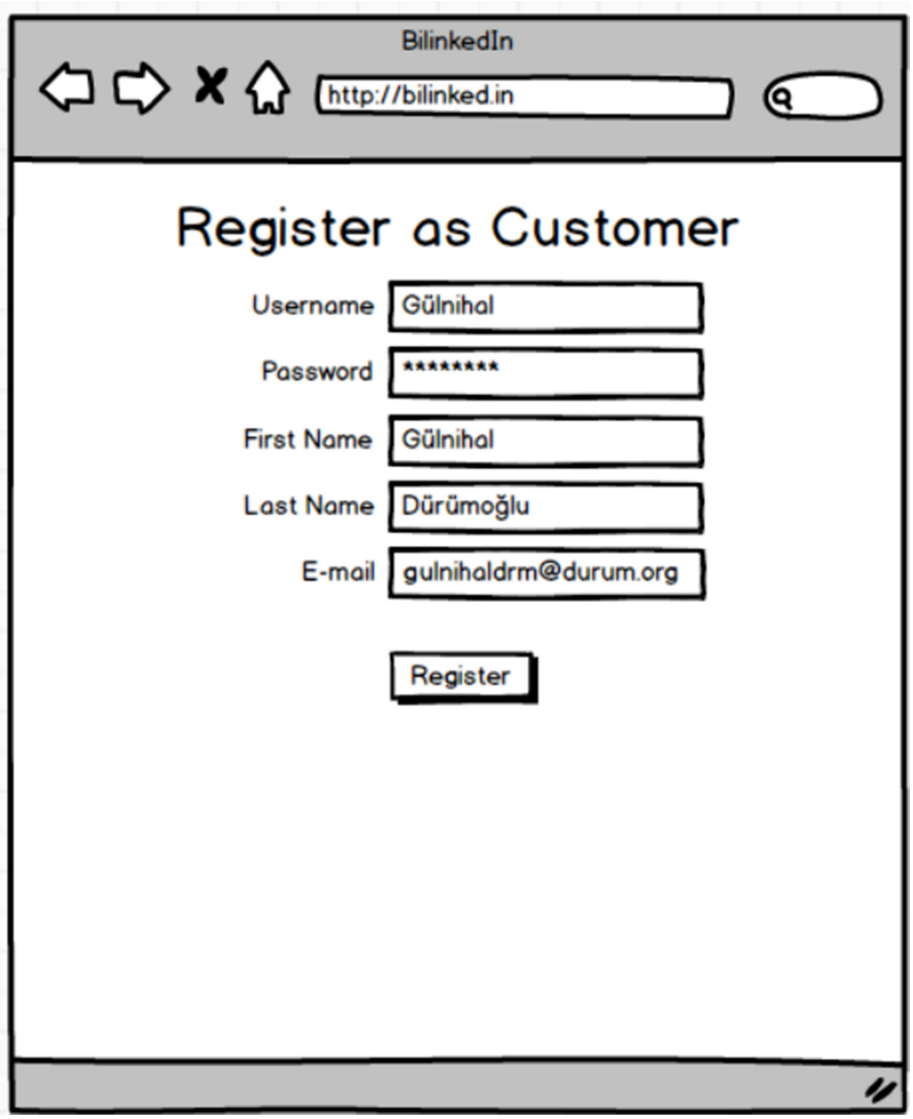

Figure 6: Customer Registration Screen

```
-- customer register
-- Inputs: @username, @password, @firstname, @lastname,
@email
```

```
INSERT INTO Customer(username, password, firstname,
lastname, email, balance) VALUES
            (@username, @password, @firstname, @lastname,
@email, 0)
```

5.3. Register as Professional

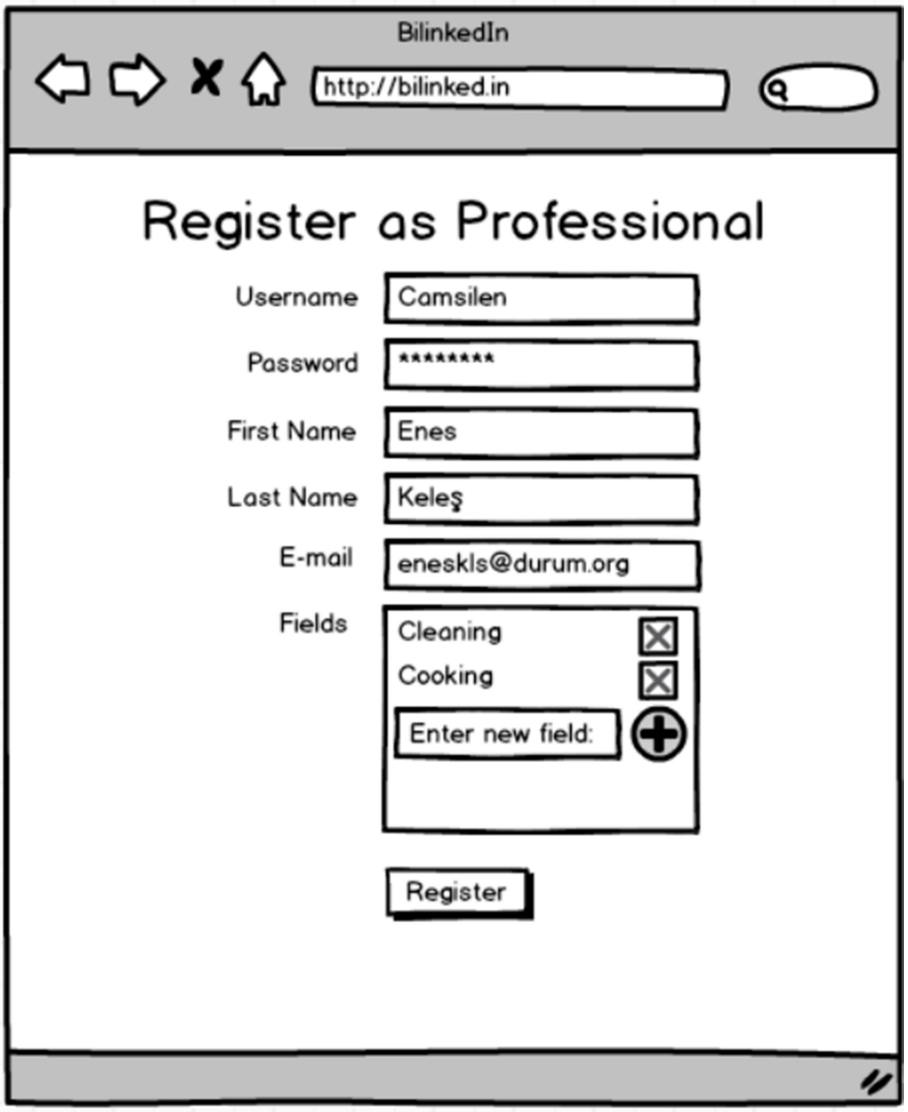

Figure 7: Professional Registration Screen

```
-- professional register
-- Inputs: @username, @password, @firstname, @lastname,
@email, @field
INSERT INTO User
```

```
INSERT INTO Professional(user_id, username, password,
firstname, lastname, email, balance) VALUES
            (@username, @password, @firstname, @lastname,
@email, 0)

-- for each field
INSERT INTO ProfessionalField (user_id, field) VALUES
            ((SELECT user_id FROM Professional WHERE
email=@email), @field)
```

5.4. Professional Profile - Self View

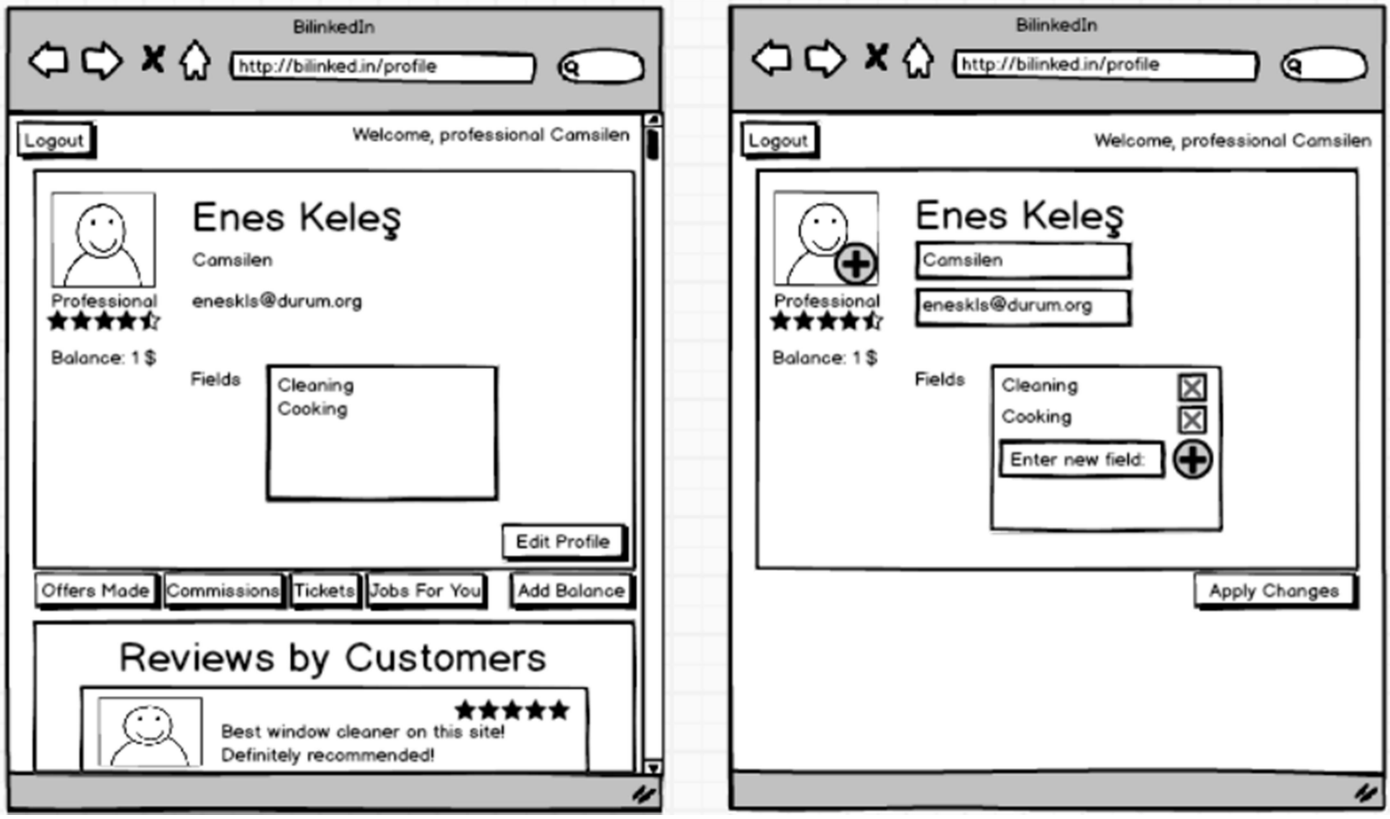

Figure 8: Professional Profile Screen – Profile Owner's View

```
-- professional self-profile
-- Inputs: @user_id

SELECT firstname, lastname, username, balance, avg_rating
      FROM Professional NATURAL JOIN
professional_avg_ratings
      WHERE Professional.user_id = @user_id
```

```
SELECT field, FROM ProfessionalField WHERE user_id=@user_id

SELECT Customer.username, customer_avg_ratings.avg_rating,
       ProfessionalReview.review_text,
ProfessionalReview.rating
FROM professional_reviews, Customer, ProfessionalReview,
customer_avg_ratings
WHERE professional_reviews.receiver_id = @user_id AND
      professional_reviews.sender_id = Customer.user_id AND
      customer_avg_ratings.user_id = Customer.user_id

-- professional self-profile update
-- Inputs: @username, @email, @deletefield, @addfield

UPDATE Professional
SET username=@username
WHERE user_id=@user_id

UPDATE Professional
SET email=@email
WHERE user_id=@user_id

INSERT INTO ProfessionalField(user_id, field) VALUES
(@user_id, @addfield)

DELETE FROM ProfessionalField WHERE user_id=@user_id AND
field=@deletefield
```

5.5. Customer Profile - Self View

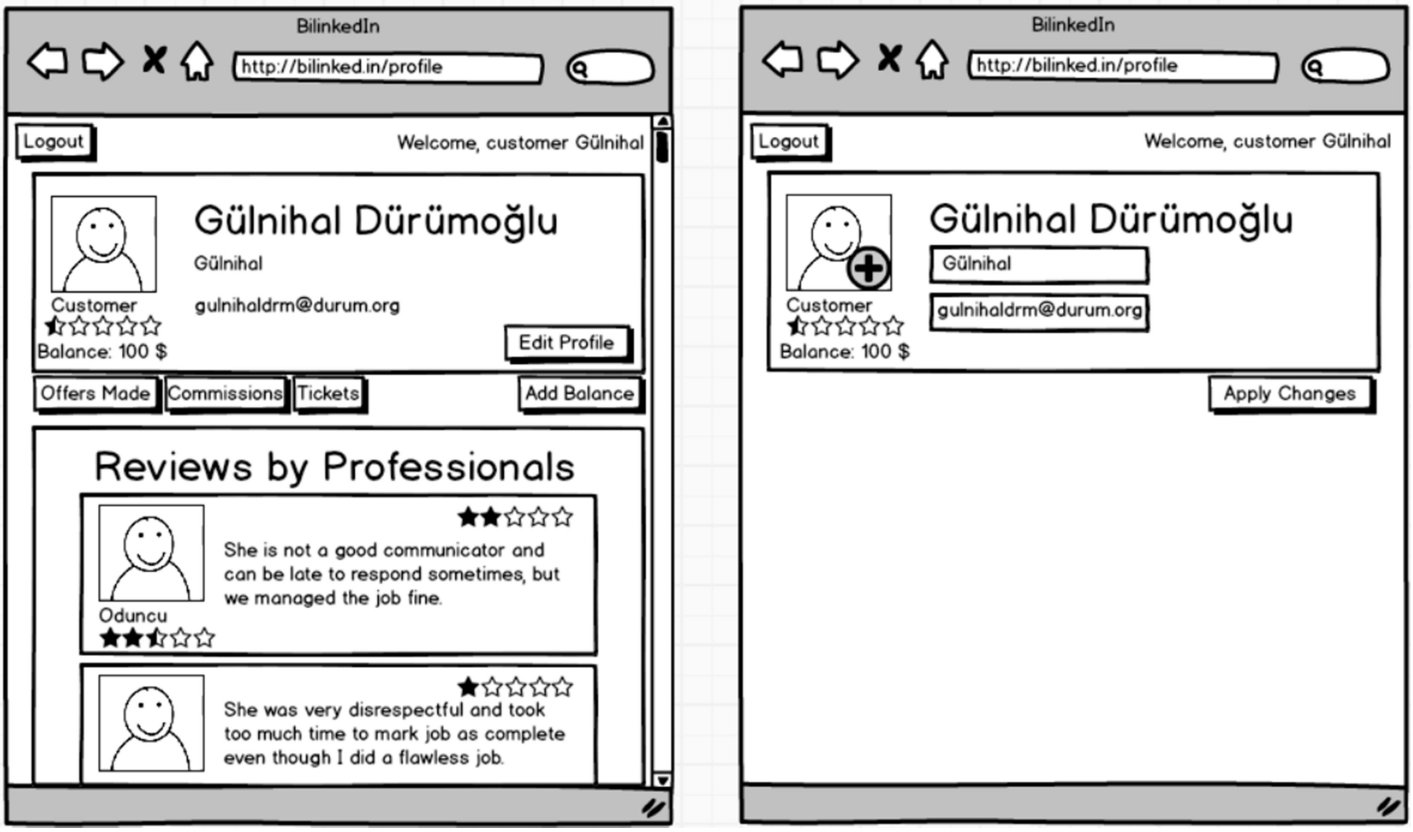

Figure 9: Customer Profile Screen – Profile Owner's View

```
-- customer self-profile
-- Inputs: @user_id

SELECT firstname, lastname, username, balance, avg_rating
      FROM Customer NATURAL JOIN customer_avg_ratings
      WHERE Customer.user_id = @user_id

SELECT Professional.username,
professional_avg_ratings.avg_rating,
       CustomerReview.review_text, CustomerReview.rating
FROM customer_reviews, Professional, CustomerReview,
professional_avg_ratings
WHERE customer_reviews.receiver_id = @user_id AND
      customer_reviews.sender_id = Professional.user_id AND
      professional_avg_ratings.user_id =
Professional.user_id

-- customer self-profile update
-- Inputs: @username, @email
```

```
UPDATE Customer
SET username=@username
WHERE user_id=@user_id

UPDATE Customer
SET email=@email
WHERE user_id=@user_id
```

5.6. Deposit Screen

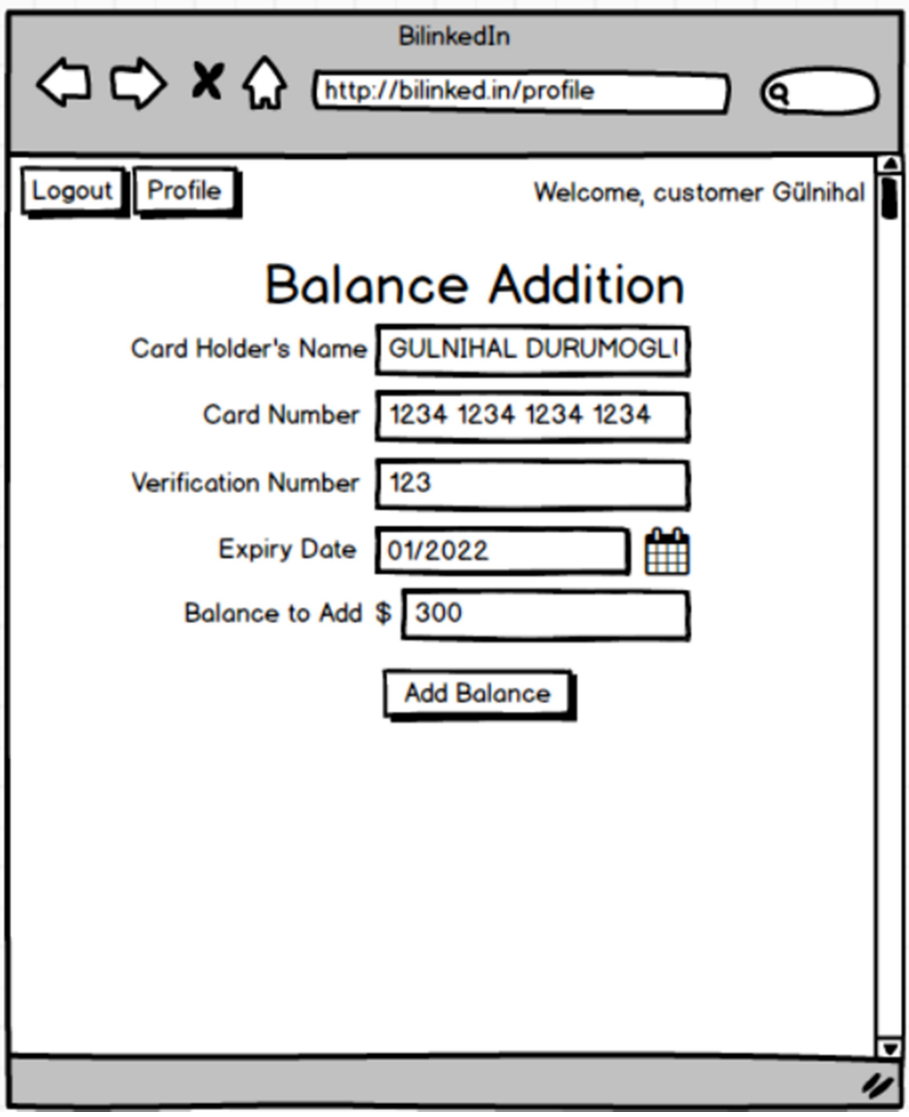

Figure 10: Deposit Screen

```
-- add balance
-- Inputs: @user_id, @amount
```

```
UPDATE Customer
SET balance=balance + @amount
WHERE user_id = @user_id

UPDATE Professional
SET balance = balance + @amount
WHERE user_id = @user_id
```

5.7. Professional Profile - Others View

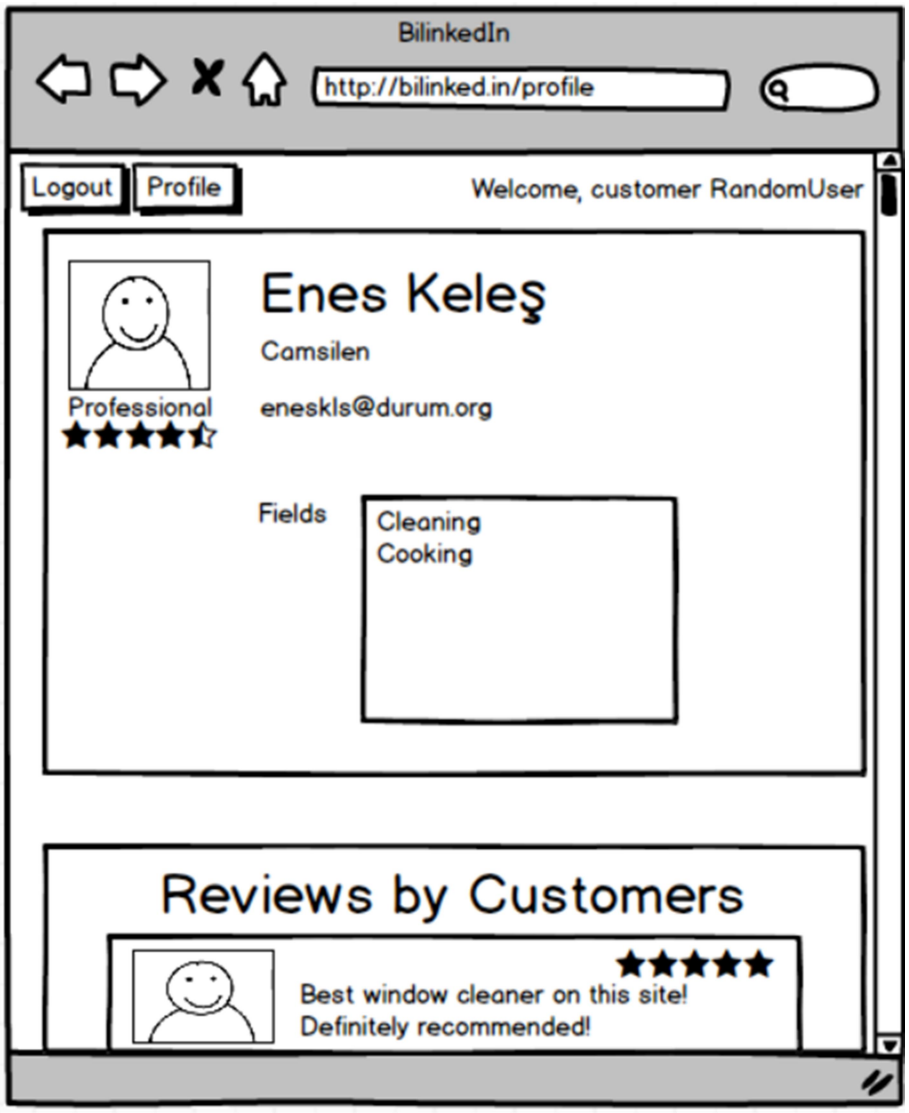

Figure 11: Professional Profile - Others' View

Previous SQL Queries given in *"5.4. Professional Profile - Self View"* are reused.

5.8. Customer Profile – Others View

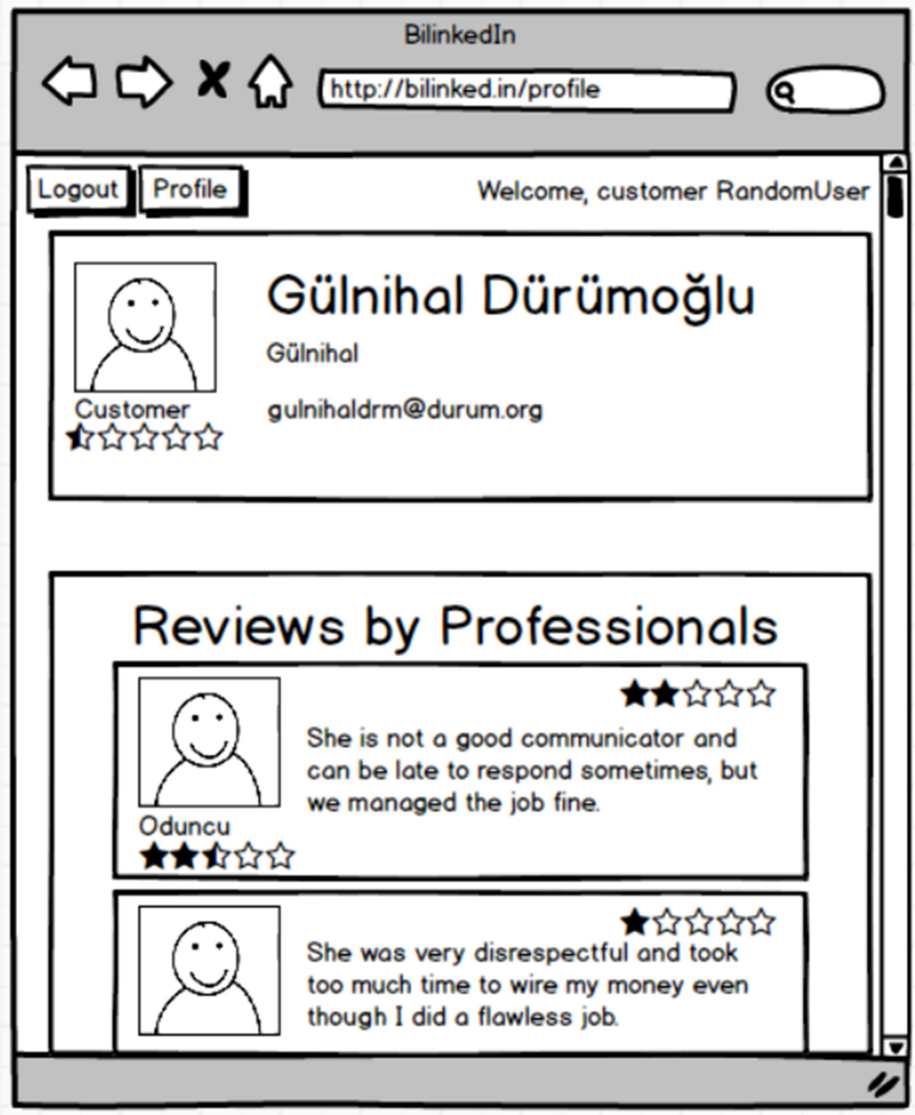

Figure 12: Customer Profile - Others' View

Previous SQL Queries given in *"5.5. Customer Profile – Self View"* are reused.

5.9. Customer Publish Job Announcement Screen

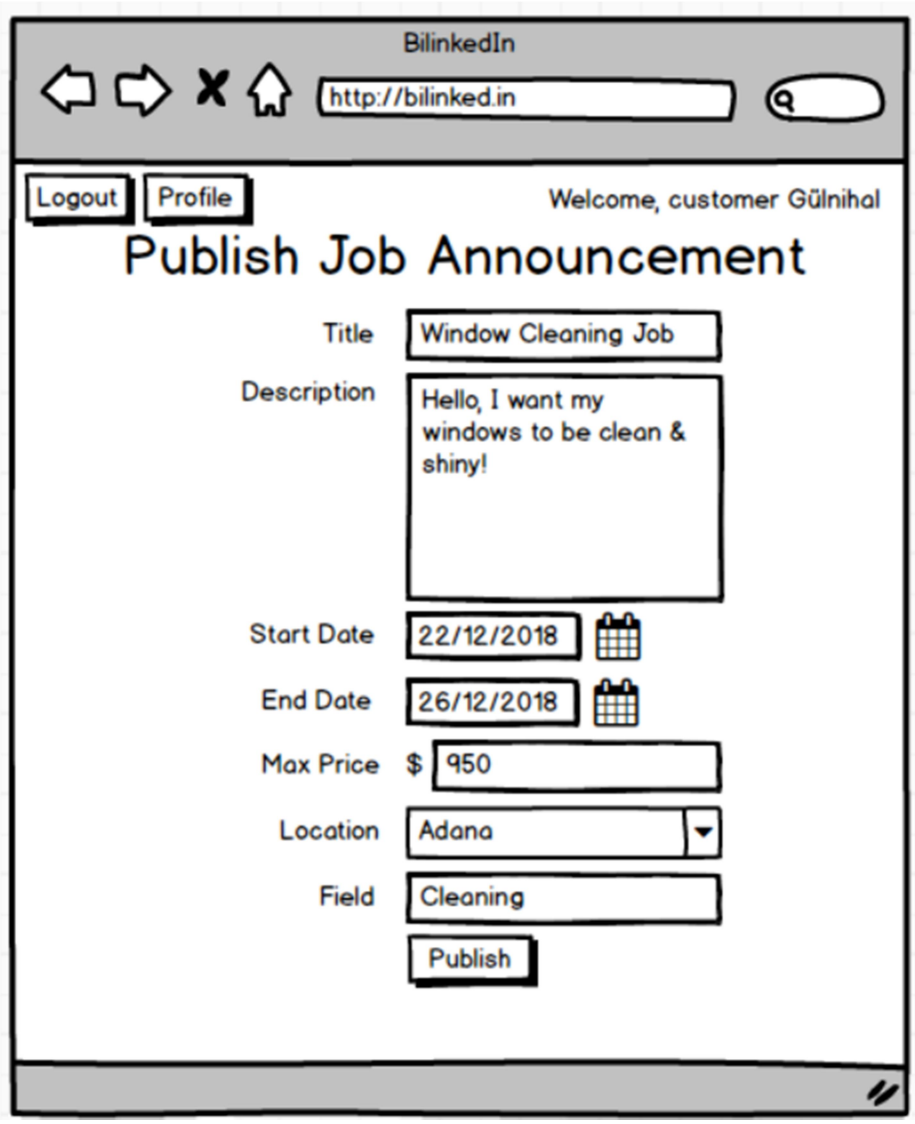

Figure 13: Publish Job Announcement Screen

```
-- publish job announcement
-- Inputs: @user_id, @title, @description, @start_date,
@end_date, @maximum_cost, @location, @field

INSERT INTO JobAnnouncement(user_id, title, description,
startdate, end_date, maximum_cost, location, field)
VALUES (@user_id, @title, @description, @startdate,
@end_date, @maximum_cost, @location, @field)
```

5.10. Available Jobs Screen

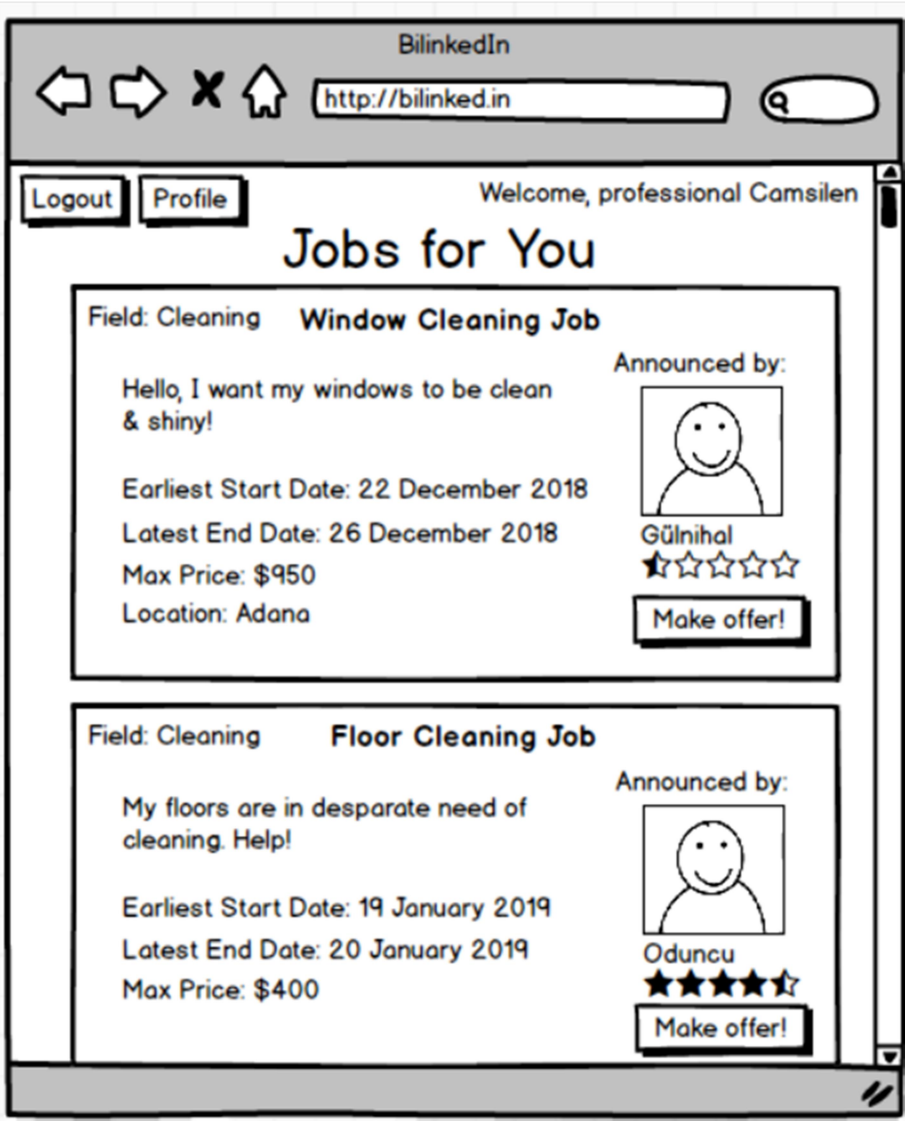

Figure 14: The screen which shows available jobs to professionals

```
-- prof view announcements
-- Inputs: @user_id

SELECT field, title, description, start_date,
       end_date, maximum_cost, location, Customer.username,
avg_rating,
FROM JobAnnouncement, Customer, ProfessionalField,
customer_avg_ratings
WHERE JobAnnouncement.field = ProfessionalField.field AND
      JobAnnouncement.user_id = Customer.user_id AND
      customer_avg_ratings.user_id = Customer.user_id
      ProfessionalField.user_id = @user_id
```

5.11. Professional Make Offer Screen

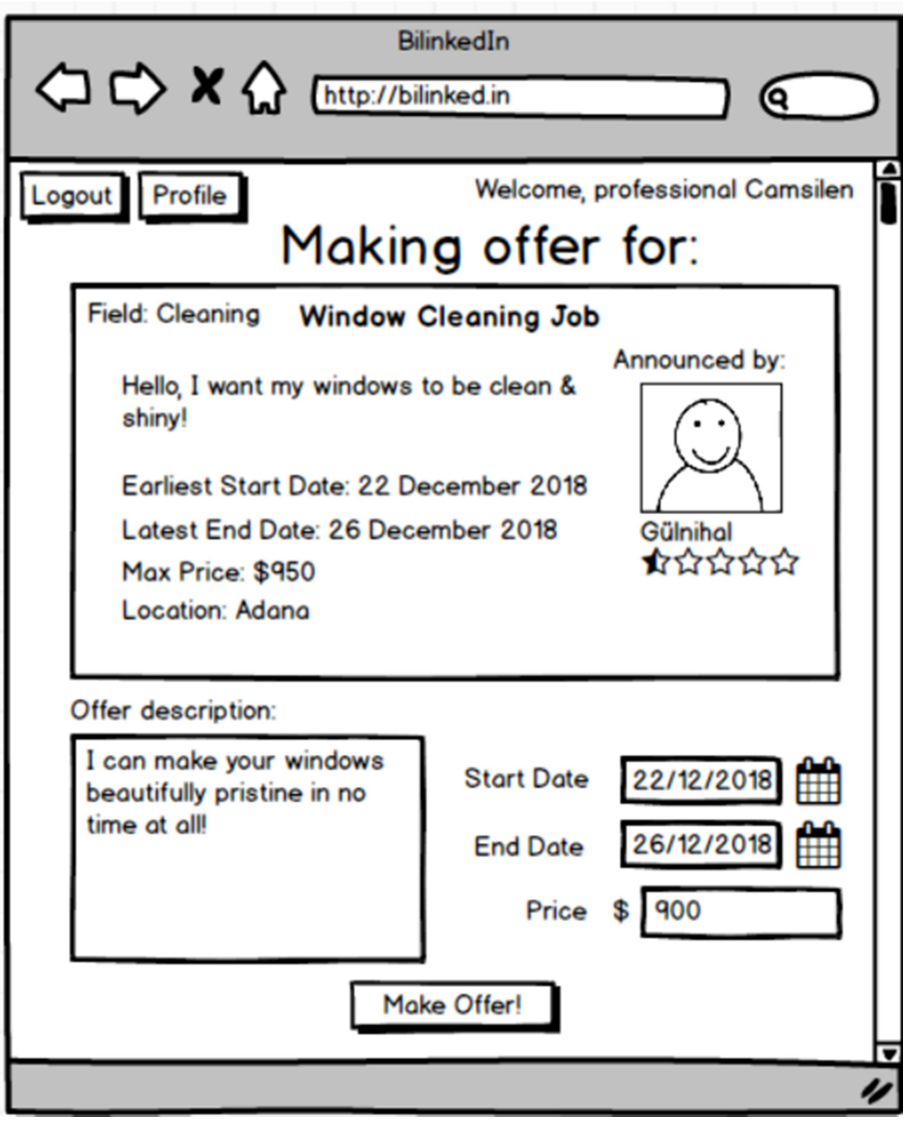

Figure 15: The screen used by professionals for making offers to job announcements

```
-- prof make offer
-- Inputs: @user_id, @announcement_id @description,
@start_date, @end_date, @cost

-- same query in view announcements is used for seeing the
announcement
INSERT INTO Chat
INSERT INTO Offer(offer_text, start_date, end_date, cost)
VALUES (@description, @start_date, @end_date, @cost)
INSERT INTO pro_offer(user_id, announcement_id)
VALUES (@user_id, announcement_id)
```

5.12. Customer View Offers Screen

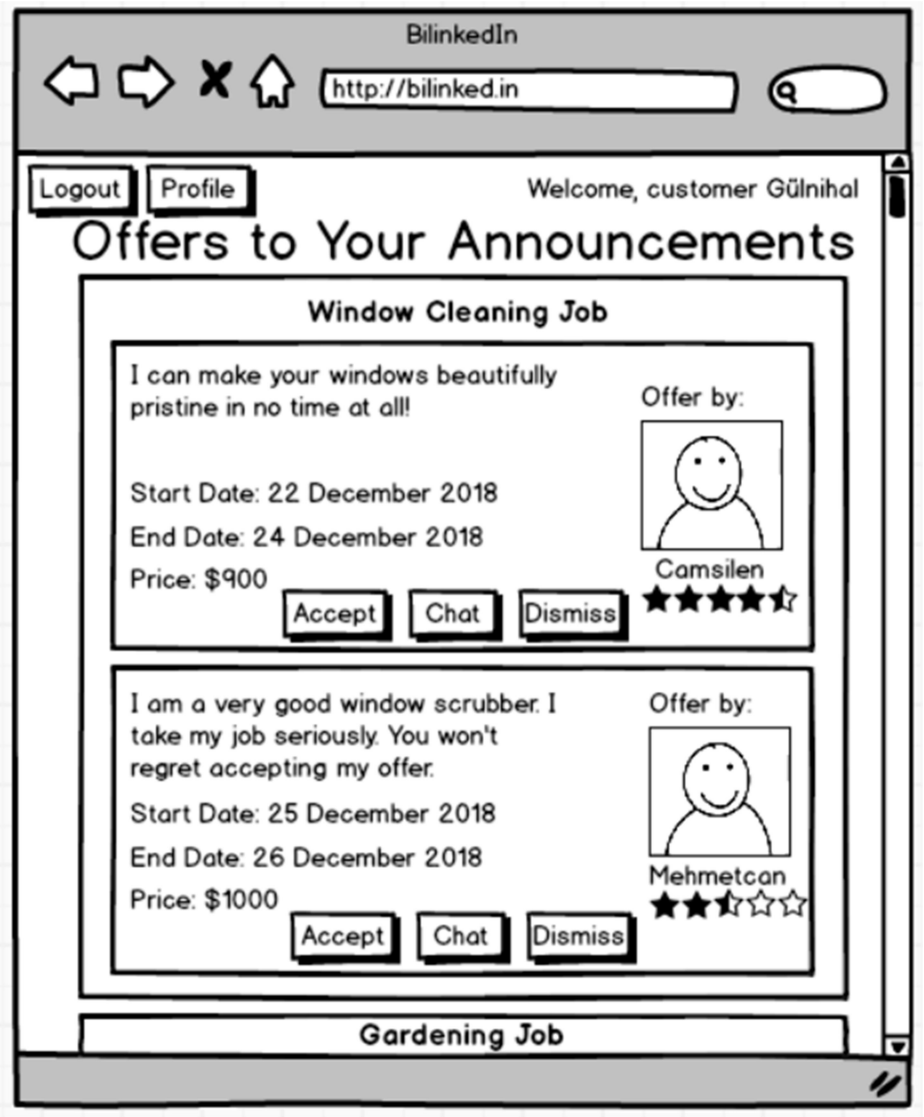

Figure 16: The screen which shows offers received for previous announcements made by the customer

```
-- cust view offers
-- Inputs: @user_id

SELECT offer_id, description, end_date, start_date, cost,
avg_rating, username
FROM pro_offer, JobAnnouncement, Professional,
professional_avg_ratings
WHERE pro_offer.user_id = Professional.user_id AND
      JobAnnouncement.announcement_id =
pro_offer.announcement_id AND
      JobAnnouncement.user_id = @user_id AND
      professional_avg_ratings.user_id =
Professional.user_id
```

5.13. Professional View Offers

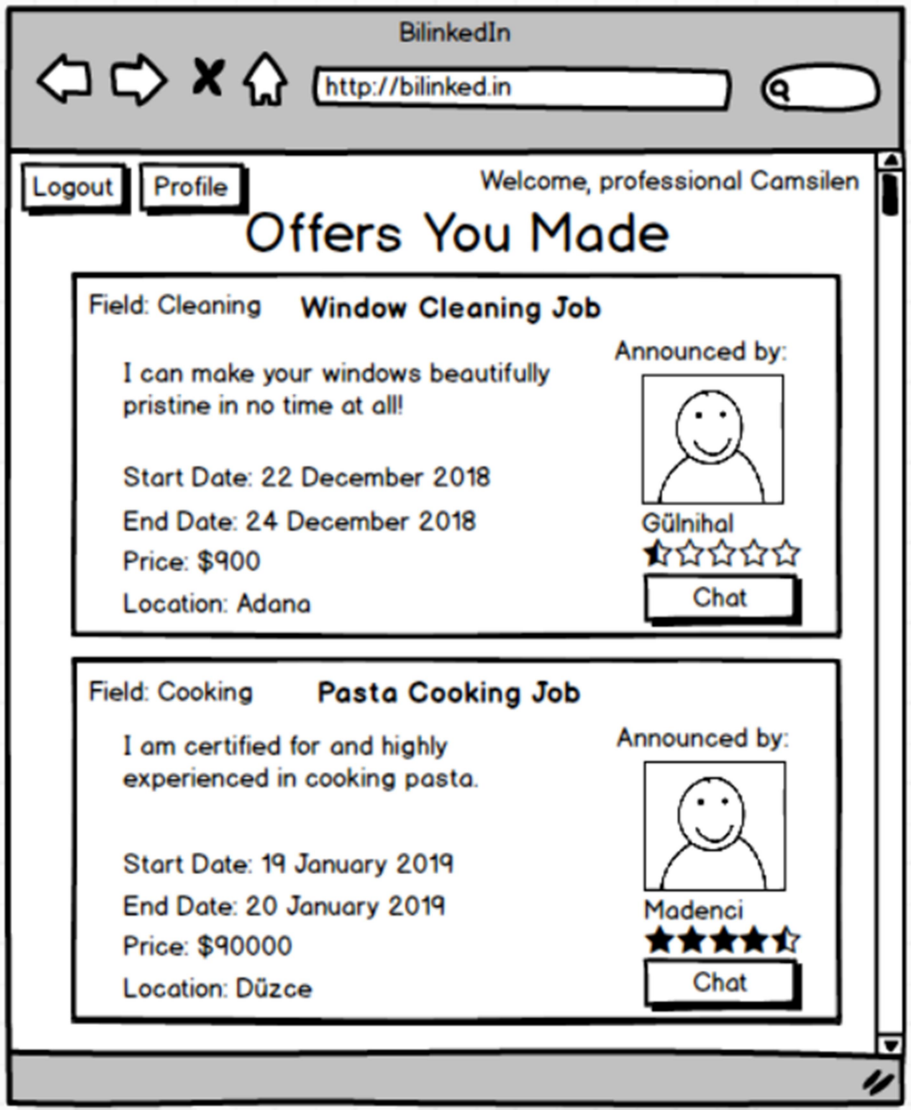

Figure 17: The screen which shows offers sent by the professional

```
-- prof view offers made
-- Inputs: @user_id

SELECT Offer.offer_id, field, title, description,
Offer.start_date, Offer.end_date,
       Offer.cost, location, Customer.username, avg_rating
FROM pro_offer, Offer, JobAnnouncement, Customer,
customer_avg_ratings
WHERE pro_offer.user_id = @user.id AND
      JobAnnouncement.user_id = Customer.user_id AND
      Customer.user_id = customer_avg_ratings.user_id AND
      pro_offer.offer_id = Offer.offer_id
```

5.14. Customer Offer Chat Screen

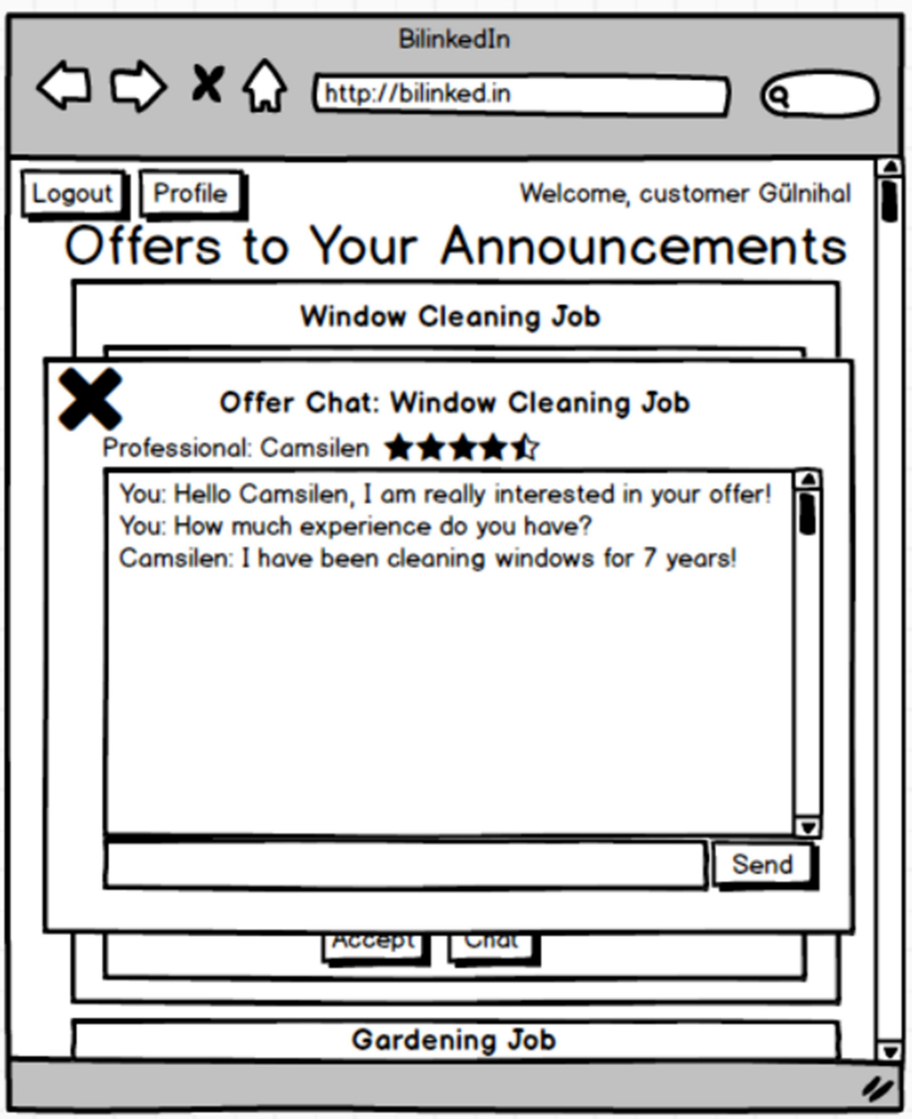

Figure 18: The offer chat screen seen by the customer

The chat is used for discussing the details of an offer between the professional who sent the offer and the customer who made the job announcement.

```
-- cust offer chat
-- Inputs: @offer_id

SELECT message_text, user_id_name.username
FROM user_id_name, Offer, OfferChat, ChatMessage
WHERE Offer.offer_id = @offer_id AND
      Offer.chat_id = OfferChat.chat_id AND
      ChatMessage.chat_id = OfferChat.chat_id
ORDER BY date_and_time ASC
```

5.15. Professional Offer Chat Screen

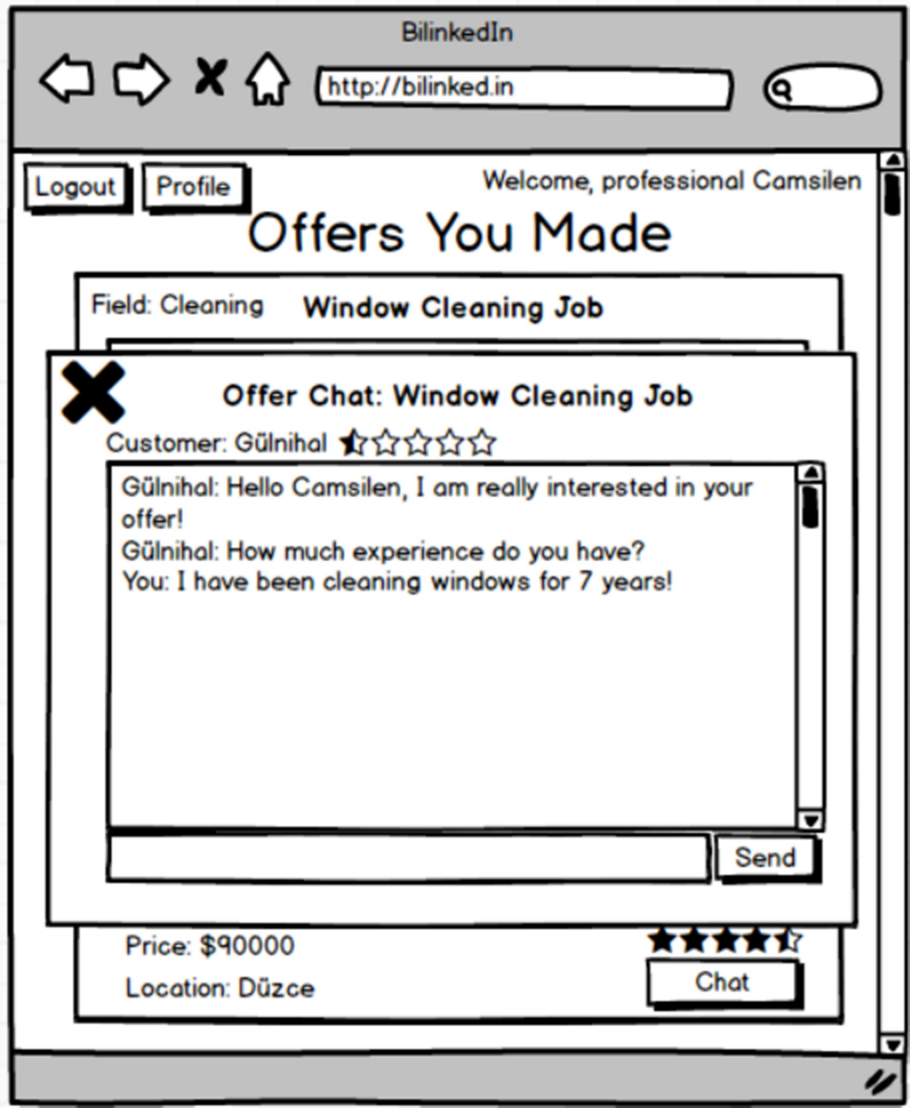

Figure 19: The offer chat screen seen by the professional

The same chat that can be seen on Figure 18, but from the professionals perspective. Previous SQL Queries given in *"5.14. Customer Offer Chat Screen"* are reused.

5.16. Professional Ongoing Commissions Screen

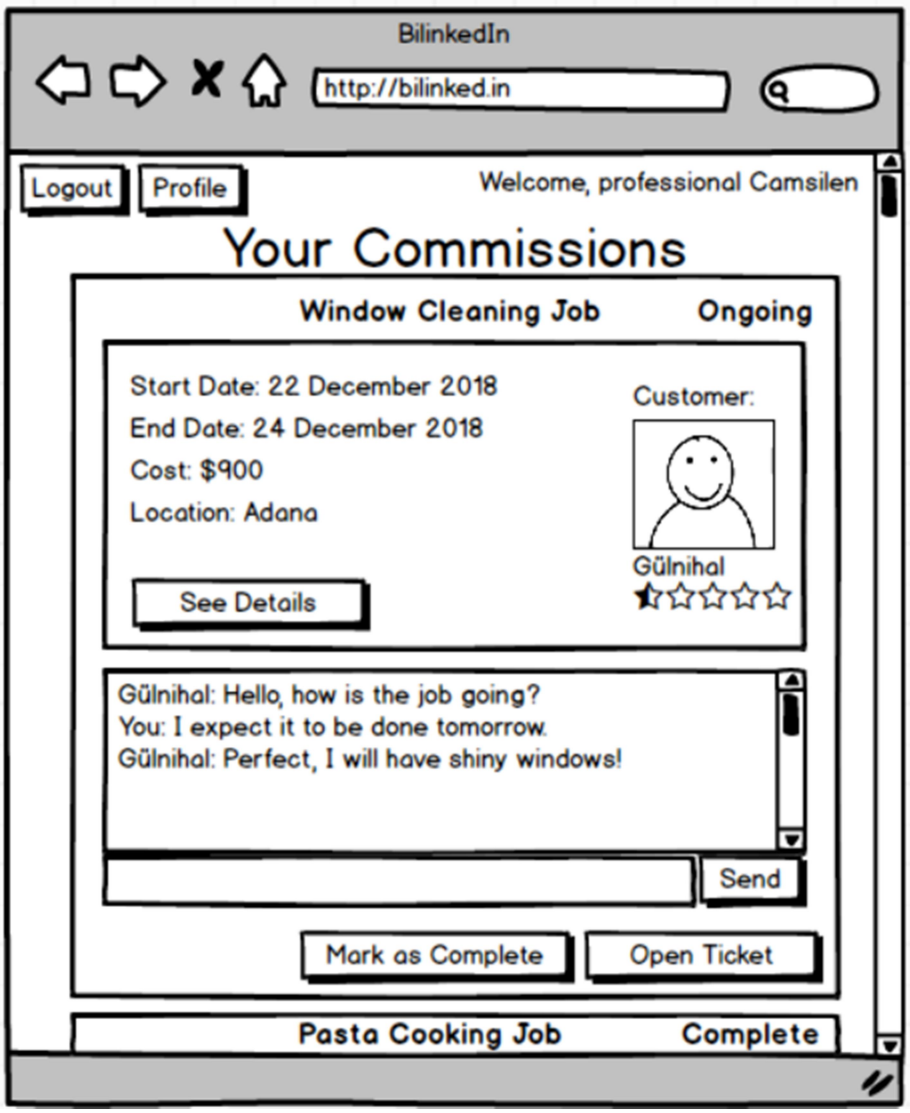

Figure 20: The screen which shows the ongoing jobs of the professional

```
-- prof ongoing commissions
-- Inputs: @user_id
SELECT title, Commission.start_date, Commission.end_date,
Commission.cost, JobAnnouncement.location, username,
avg_rating
FROM JobAnnouncement, pro_offer, professional_commissions,
Commission, professional_avg_ratings
WHERE professional_commissions.user_id = @user_id AND
      professional_commissions.commission_id =
Commission.commission_id AND
      Commission.offer_id = pro_offer.offer_id AND
```

```
        pro_offer.announcement_id =
JobAnnouncement.announcement_id AND
        professional_avg_ratings.user_id = @user_id AND
        Commission.status = 'Ongoing'
```

5.17. Customer Ongoing Commissions Screen

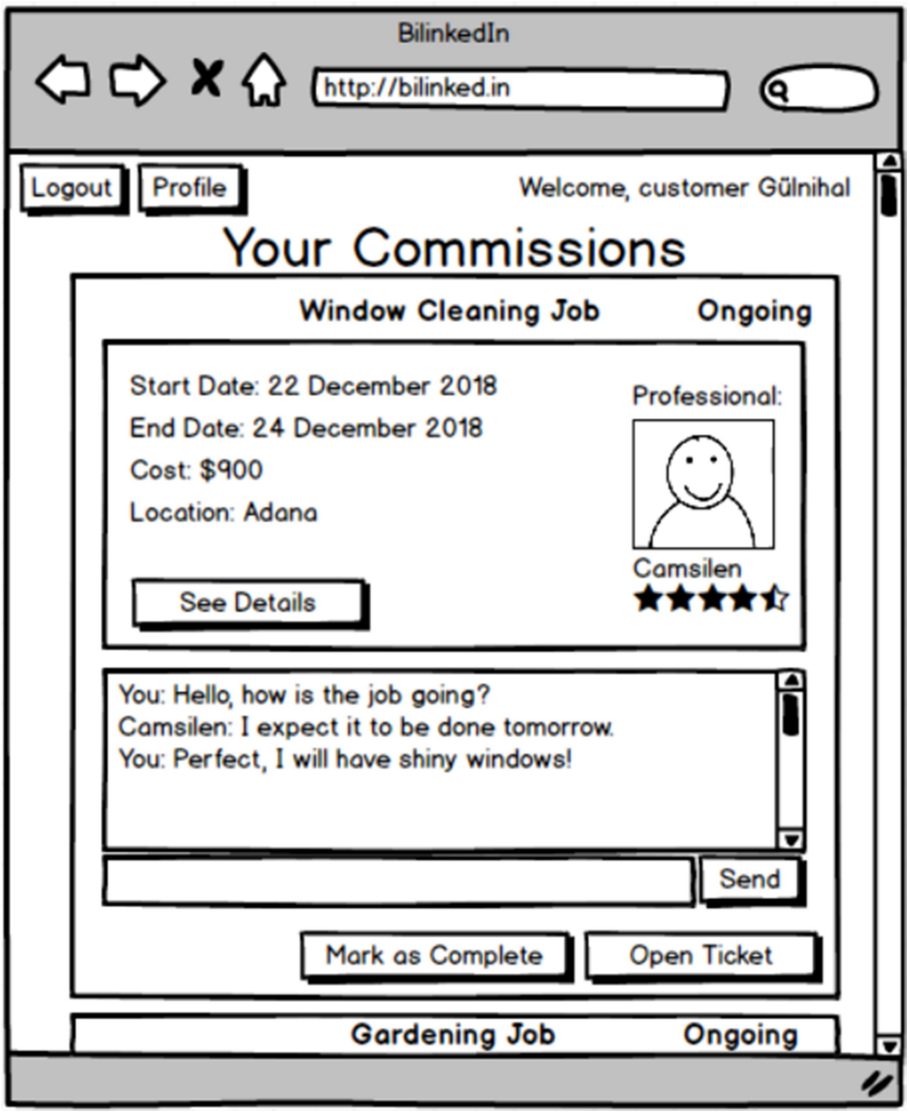

Figure 21: The screen which shows the ongoing commissions of the customer

Previous SQL Queries given in *"5.16. Professional Ongoing Commissions Screen"* are reused.

5.18. Customer Commission Details Screen

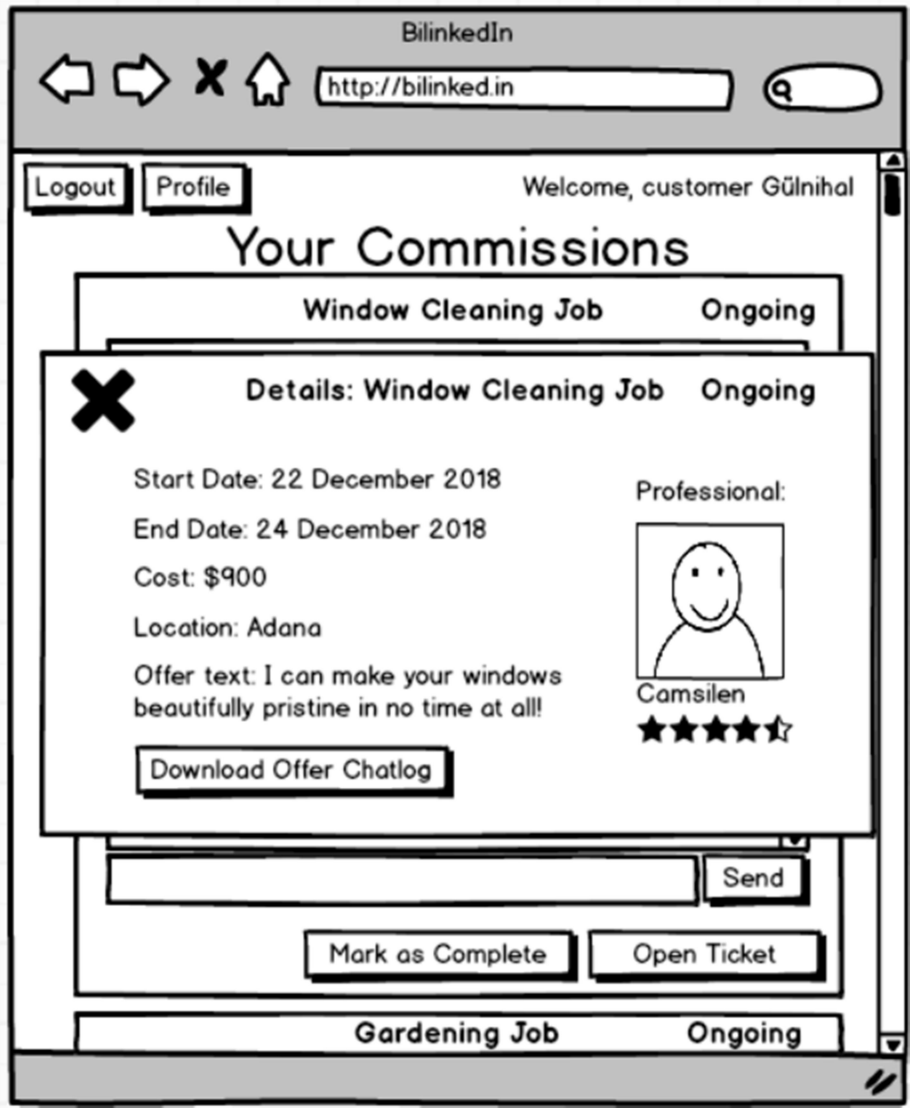

Figure 22: Detailed view of a specific commission seen by the customer

This screen shows details about a specific commission that the customer wishes to see. Previous SQL Queries given in *"5.14. Customer Offer Chat Screen"* and *"5.16. Professional Ongoing Commissions Screen"* are reused.

5.19. Professional Commission Details Screen

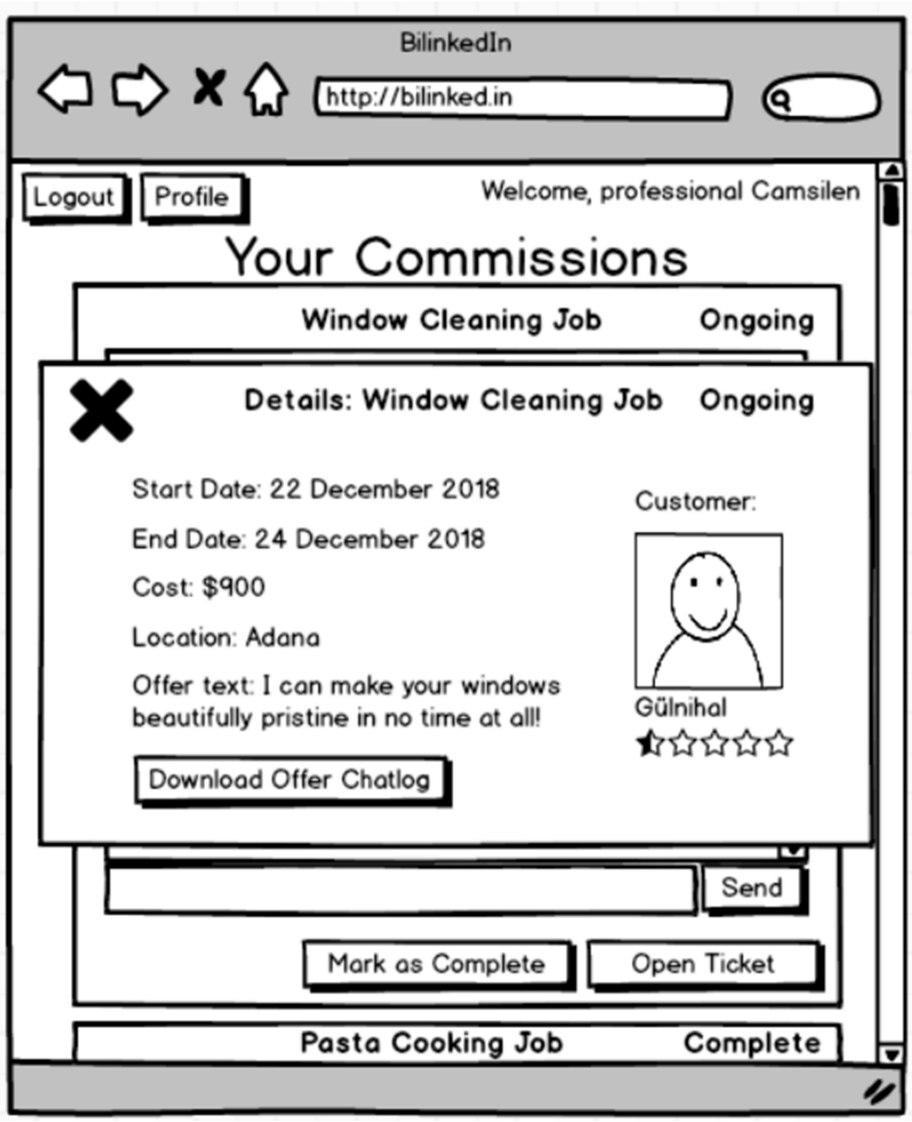

Figure 23: Detailed view of a specific commission seen by the professional

This screen shows details about a specific commission that the professional wishes to see. Previous SQL Queries given in *"5.14. Customer Offer Chat Screen"* and *"5.16. Professional Ongoing Commissions Screen"* are reused.

5.20. Open Ticket Screen

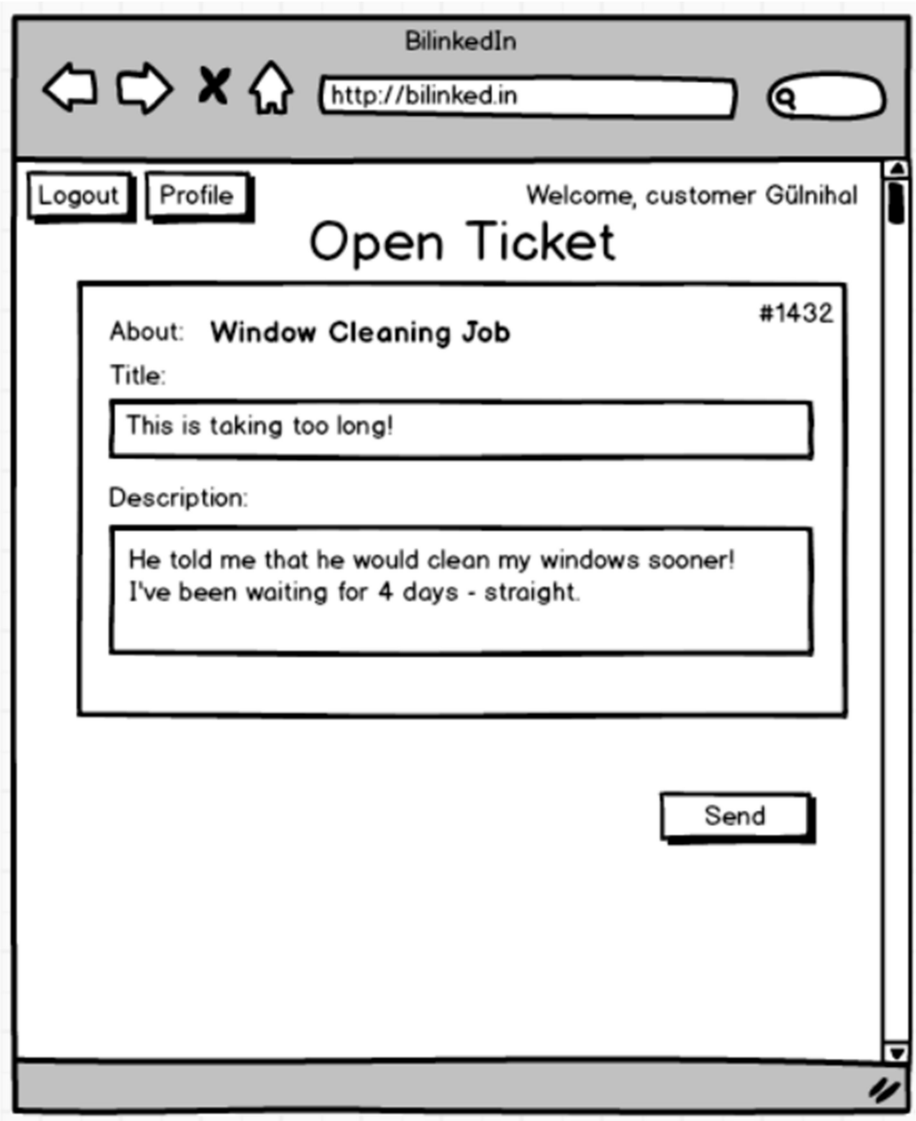

Figure 24: The screen which is used for opening a ticket

A ticket is opened when there is a disagreement between the professional and the customer. Both parties can open a ticket.

```
-- open ticket
-- Inputs: @title, @description

INSERT INTO Ticket(title, description) VALUES (@title,
@description)
```

5.21. Tickets Screen

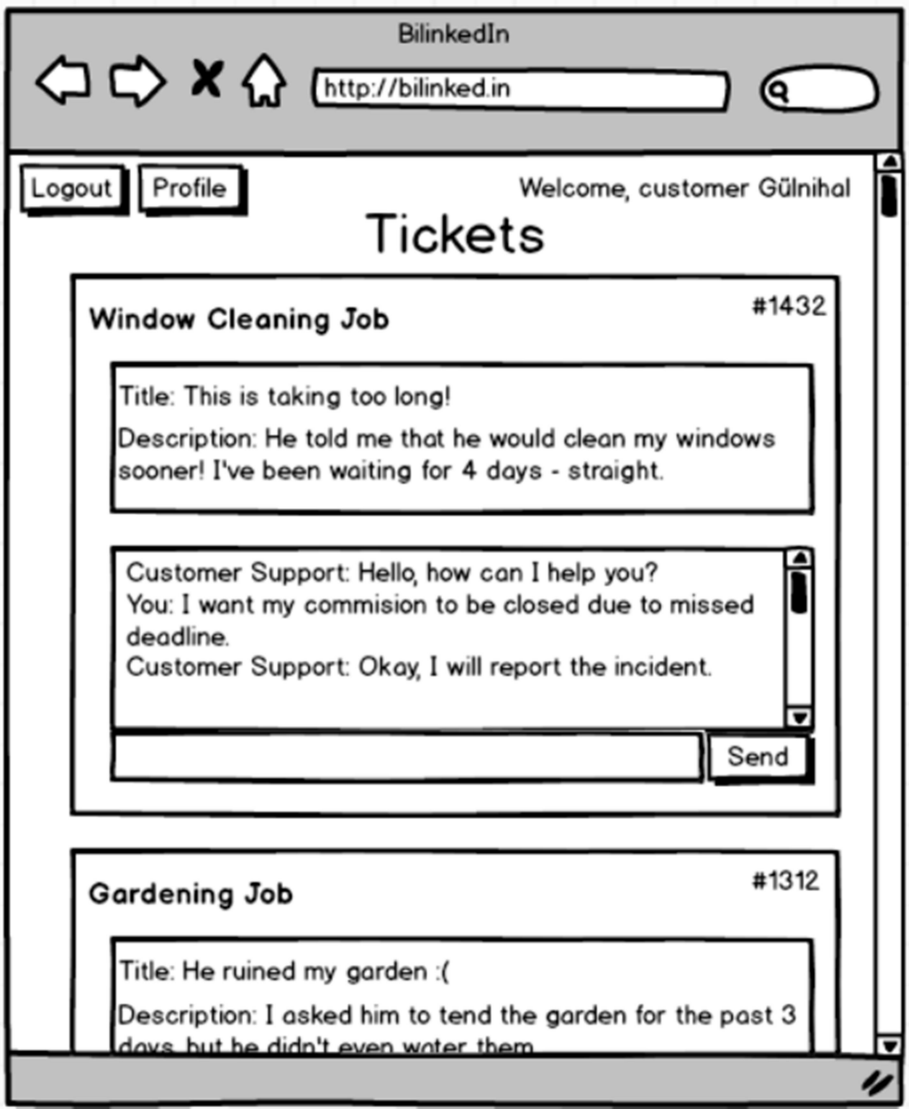

Figure 25: The screen which shows the tickets the user is involved in

```
-- see tickets
-- Inputs: @user_id

SELECT Ticket.title, Ticket.description
FROM customer_tickets, Ticket
WHERE user_id = @user_id AND
      Ticket.ticket_id = customer_tickets.ticket_id

SELECT Ticket.title, Ticket.description
FROM professional_tickets, Ticket
WHERE user_id = @user_id AND
      Ticket.ticket_id = professional_tickets.ticket_id
```

5.22. Rate and Review Screen - Customer View

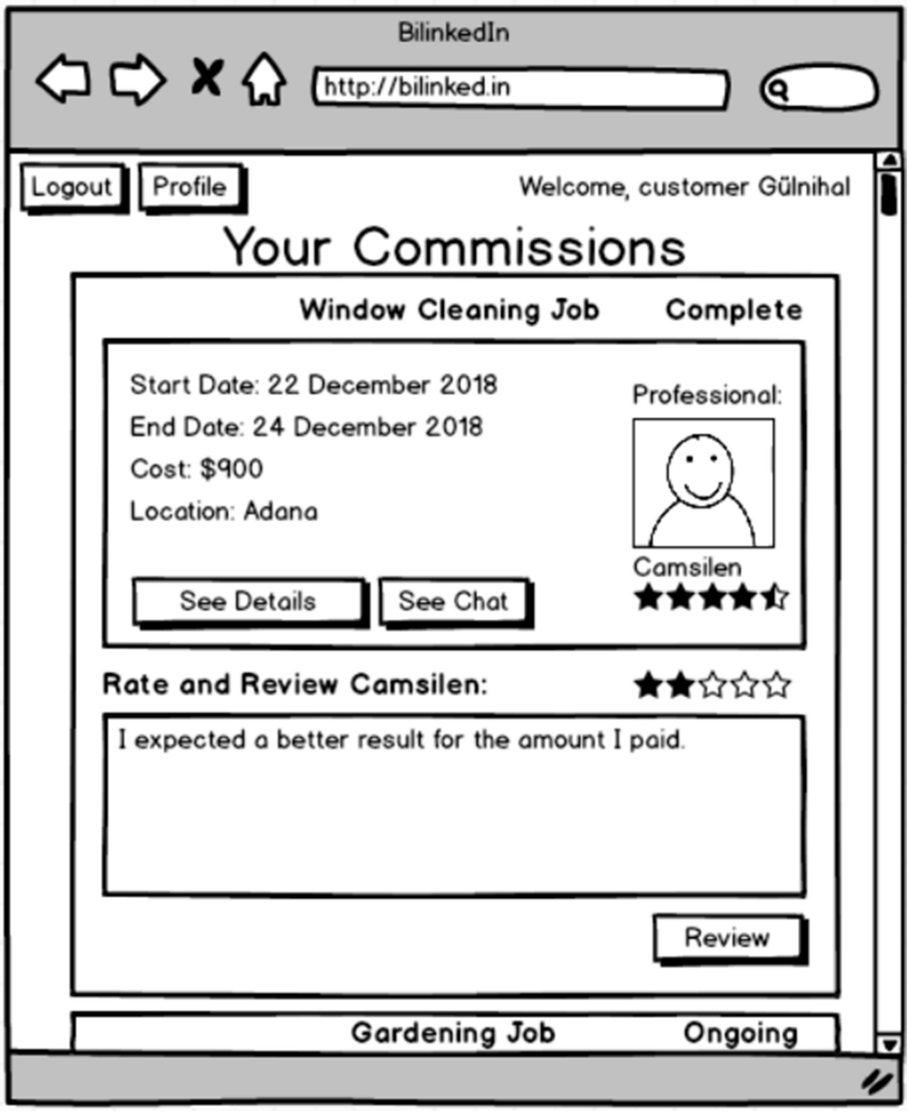

Figure 26: The screen which is used after a commission is done to rate and review the professional

```
-- make review by customer to professional
-- Inputs: @commission_id, @review_text, @rating

INSERT INTO
ProfessionalReview (commission_id, review_text, rating)
VALUES (@commision_id, @review_text, @rating)
```

5.23. Rate and Review Screen - Professional View

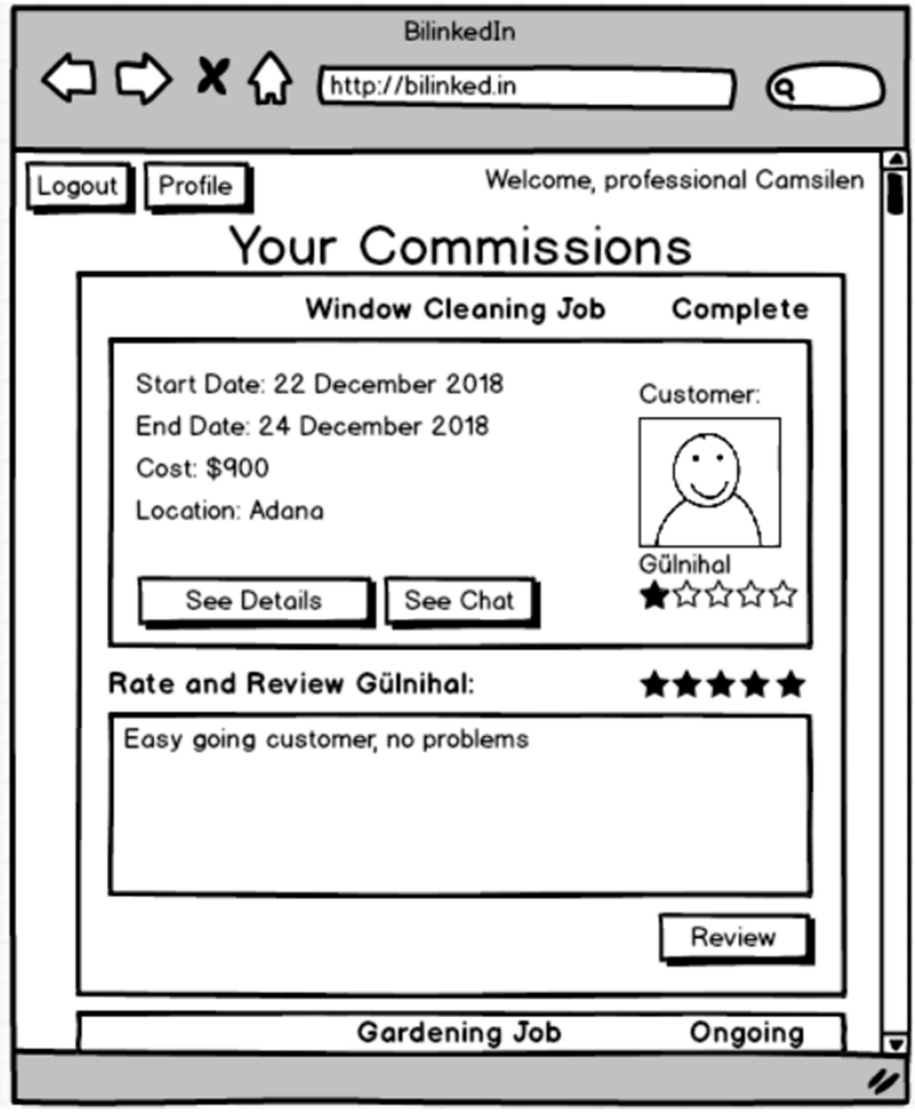

Figure 27: The screen which is used after a commission is done to rate and review the customer

```
-- make review by professional to customer
-- Inputs: @commission_id, @review_text, @rating

INSERT INTO
CustomerReview (commission_id, review_text, rating)
VALUES (@commision_id, @review_text, @rating)
```

6. ADVANCED DATABASE COMPONENTS

6.1. Views

6.1.1. Professional Commissions View

This view will be used to find commissions of professionals.

```
CREATE VIEW professional_commissions AS (
 SELECT pro_offer.user_id, Commission.commission_id,
 FROM pro_offer, Offer, Commission
 WHERE Offer.offer_id = pro_offer.offer_id AND
       Commission.offer_id = Offer.offer_id
)
```

6.1.2. Customer Commissions View

This view will be used to find commissions of customers.

```
CREATE VIEW customer_commissions AS (
 SELECT JobAnnouncement.user_id,
       Commission.commission_id,
 FROM JobAnnouncement, pro_offer, Offer, Commission
 WHERE pro_offer.announcement_id =
       JobAnnouncement.announcement_id AND
       Offer.offer_id = pro_offer.offer_id AND
       Commission.offer_id = Offer.offer_id
)
```

6.1.3. Professional Average Ratings View

This view will be used to find average ratings of professionals.

```
CREATE VIEW professional_avg_ratings AS (
 SELECT professional_commissions.user_id,
       AVG(ProfessionalReview.rating)
FROM professional_commissions, ProfessionalReview
WHERE ProfessionalReview.commission_id =
       professional_commissions.commission_id
GROUP BY professional_commissions.user_id
)
```

6.1.4. Customer Average Ratings View

This view will be used to find average ratings of professionals.

```
CREATE VIEW customer_avg_ratings AS (
  SELECT customer_commissions.user_id,
       AVG(CustomerReview.rating)
FROM customer_commissions, CustomerReview
WHERE CustomerReview.commission_id =
       customer_commissions.commission_id
GROUP BY customer_commissions.user_id
)
```

6.1.5. Professional Reviews View

This view will be used to list down reviews concerning professionals.

```
CREATE VIEW professional_reviews AS (
  SELECT professional_commissions.user_id,
       ProfessionalReview.review_id
  FROM professional_commissions, ProfessionalReview
  WHERE ProfessionalReview.commission_id =
       professional_commissions.commission_id
)
```

6.1.6. Customer Reviews View

This view will be used to list down reviews concerning customers.

```
CREATE VIEW customer_reviews AS (
  SELECT customer_commissions.user_id,
       CustomerReview.review_id
  FROM customer_commissions, CustomerReview
  WHERE CustomerReview.commission_id =
       customer_commissions.commission_id
)
```

6.1.7. Professional Tickets View

This view will be used to list down tickets that are opened by professionals.

```
CREATE VIEW professional_tickets AS (
  SELECT professional_commissions.user_id,
       Ticket.ticket_id
  FROM professional_commissions, Ticket
  WHERE Ticket.commission_id =
       professional_commissions.commission_id
)
```

6.1.8. Customer Tickets View

This view will be used to list down tickets that are opened by customers.

```
CREATE VIEW customer_tickets AS (
  SELECT customer_commissions.user_id, Ticket.ticket_id
  FROM customer_commissions, Ticket
  WHERE Ticket.commission_id =
       customer_commissions.commission_id
)
```

6.2. Stored Procedures

The SQL statements given under “5. User Interface Design and Corresponding SQL Statements”, are all stored procedures. The procedures are not written in this part to avoid repetition. Refer to part 5 to see the stored procedures and their usage.

6.3. Reports

We currently don’t plan on using reports in our system since they are not crucial for the functionality of the system.

6.4. Triggers

- When a dispute is made(a ticket is created), a new TicketChat entity is created and an according entity is inserted to ticket_chat relation, linking the TicketChat and the Ticket. If the user_id attribute in the new ticket_chat entry is NULL after the insertion, a trigger finds the customer support with the least amount of tickets and sets the user_id of the ticket_chat entry as the user_id of the customer support with the least amount of tickets.

- For insertions to specialization tables (Customer, Professional, CustomerSupport, Professional, Chats, Reviews, etc.) a trigger creates an entity in the “parent” table and sets the value of the primary key of the entry being inserted as the primary key of the created entry. For example, when a new entry is inserted to Customer table, the trigger inserts a new entry to the User table. The trigger then sets the user_id of the NEW Customer equal to the user_id of the new User created in the trigger.(User has auto-incremented user_id)

6.5. Constraints

- The system requires logging in.

- The name and surname cannot be changed.

- The customers won’t be able to offer a higher amount than their balance.

- The announcements are deleted if no offer was accepted before the end-date. All the offers made for that announcement are deleted as well.(ON DELETE CASCADE)
- An ongoing job cannot be cancelled.

- Transactions cannot be completed without both sides marking the job as completed.

- The cost of a job can’t have a negative value.

- The amount that a professional requests should be in the range of “0-cost determined by the customer”.

- The start-date of an announced job can't be after the end-date, and the end-date of an announced job can't be before the start-date.

7. IMPLEMENTATION PLAN

For web-based UI, we are planning to use Bootstrap, Core UI and Vue.js. For the database, we are planning to use the MySQL Server. For the server endpoints, flask microframework in Python will be used.

8. WEBSITE

The project information is available at:
https://github.com/dsipahioglu/bilinkedin