CS 353 – Database Systems

Project Design Report

Online Professional Hiring System

Group 39

Bora Ecer

Alp Ege Baştürk

Deniz Alkışlar

Buğra Aydın

02.04.2018

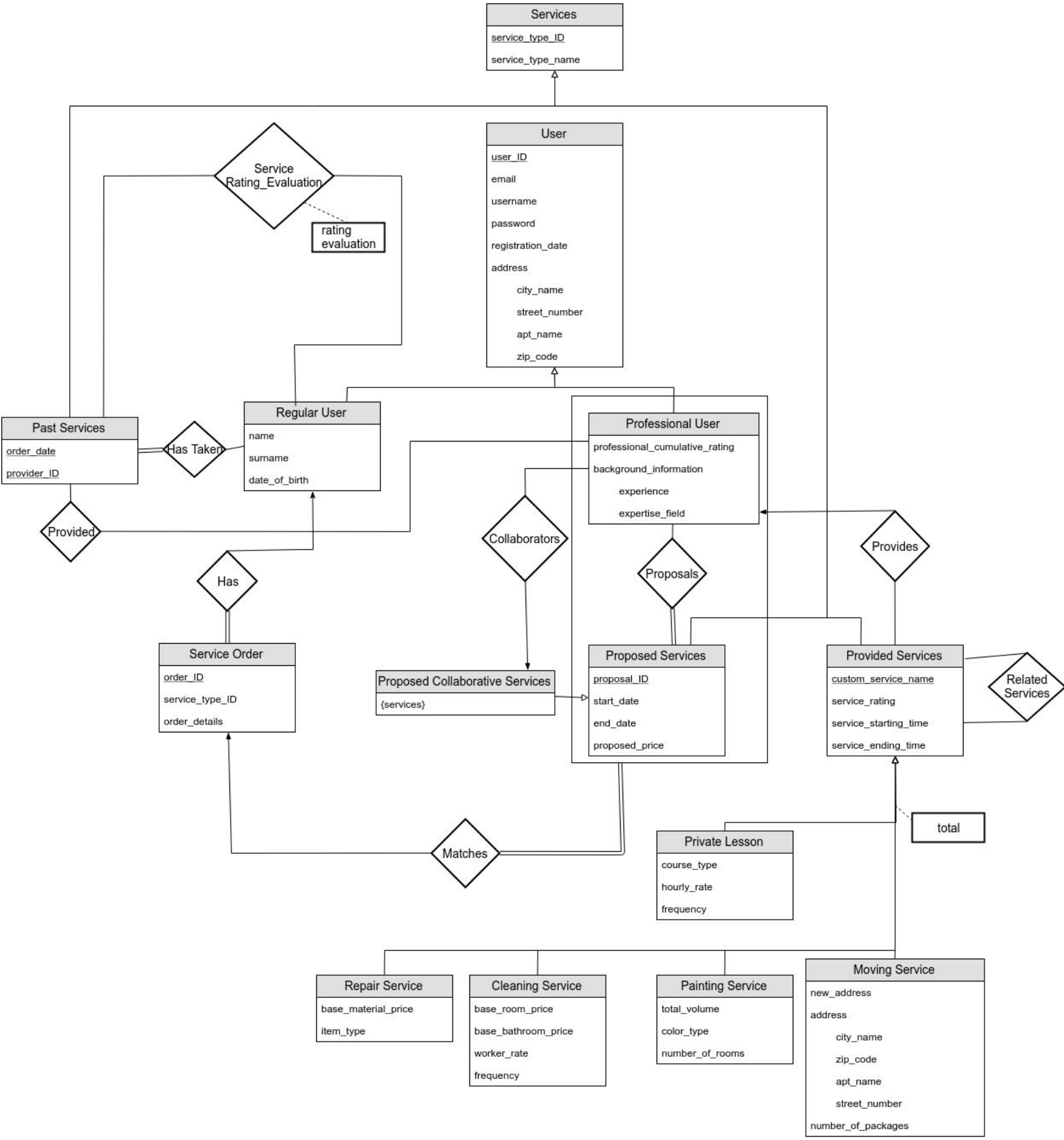
Table of Contents

1. Revised E/R Model

Changes were made in the proposed E/R diagram to provide a better structure and logic. Following changes were made on the proposed E/R diagram according to our TA's feedback;

- Past Services entity was changed from weak to normal entity.
- Aggregation of Past Services and Has Taken relation was removed.
- Service Rating and Service Evaluation relations were into Service Rating_Evaluation relation. Also relation was made between Regular user and Past Services entities. Previously rating and evaluation relations were between regular used and the aggregation of Past Services and Has Taken relation, which was removed.
- password attribute was added to User entity.
- Service Order entity was changed from weak to normal entity.
- Aggregation of Service Order entity and Has relation was removed.
- Matches relation between the Service Order and Has aggregation was connected to Service order directly.
- Proposed Collaborative Services entity was made child of Proposed Services and inherited attributes.
- Professional User entity to Collaborators relation was considering Professional User as a weak entity. This was changed to make Professional User a normal entity.
- Proposals relation was added between Professional User and Proposed Services entities.
- Professional User, Proposed Services entities and Proposals relation were aggregated.
- Proposed Collaborative Services and Proposed Services were both related to Matches relation and causing a ternary relation. Since Proposed Collaborative Services was made child of Proposed Services, this relation was reduced to a binary relation. The mentioned aggregation was used in the Matches relation and also it was made full contribution.
- Ternary Provides relationship was reduces to binary by separating Proposed Services entity from the relation.
- proposal_ID attribute was added to Proposed Services since unique identifiers were not sufficient in the previous diagram.
- Derived attributes such as age() were removed to comply with TA's feedback and attributes such as professional_cumulative_rating was made a proper attribute.

- Provided Services entity and its children were made a total contribution.
- Individual Price attribute was removed from Collaborators relation.
- Provided relation was added in order to connect Past Services and Professional Users entities to allow use of provider_ID.


Services
service_type_ID
service_type_name
User
user_ID
email
username
password
registration_date
address
city_name
street_number
apt_name
zip_code
Service Rating_Evaluation
rating evaluation
Past Services
order_date
provider_ID
Has Taken
Regular User
name
surname
date_of_birth
Professional User
professional_cumulative_rating
background_information
experience
expertise_field
Provided
Has
Collaborators
Proposals
Provides
Service Order
order_ID
service_type_ID
order_details
Proposed Collaborative Services
{services}
Proposed Services
proposal_ID
start_date
end_date
proposed_price
Provided Services
custom_service_name
service_rating
service_starting_time
service_ending_time
Related Services
total
Matches
Private Lesson
course_type
hourly_rate
frequency
Repair Service
base_material_price
item_type
Cleaning Service
base_room_price
base_bathroom_price
worker_rate
frequency
Painting Service
total_volume
color_type
number_of_rooms
Moving Service
new_address
address
city_name
zip_code
apt_name
street_number
number_of_packages

2. Relation Schemas

Following are the Relation Schemas of our database design which corresponds to the E/R diagram.

2.1 Services

Model:
Services(<u>service_type_ID</u>, service_type_name)

Functional Dependencies:
service_type_ID → service_type_name

Candidate Keys:
service_type_ID

Primary Key:
service_type_ID

Normal Form:
BCNF

Table Declaration:

```
CREATE TABLE Services (
        service_type_ID INT PRIMARY KEY,
        service_type_name VARCHAR(32) NOT NULL
        );
```

2.2 User

Model:

User(<u>user_ID</u>, password, email, username, registration_date, city_name, street_number, apt_name, zip_code)

Functional Dependencies:

user_ID → password, email, username, registration_date, city_name, street_number, apt_name, zip_code

email → user_ID, password, username, registration_date, citty_name, street_number, apt_name, zip_code

username → user_ID, email, password, registration_date, citty_name, street_number, apt_name, zip_code

Candidate Keys:

user_ID, email, username

Primary Key:

user_ID

Normal Form:

Table Declaration:

```
CREATE TABLE User (
	user_ID INT PRIMARY KEY AUTO_INCREMENT,
	password VARCHAR(32) NOT NULL,
	email VARCHAR(32) NOT NULL UNIQUE,
	username VARCHAR(32) NOT NULL UNIQUE,
	city_name VARCHAR(32) DEFAULT NULL,
	street_number VARCHAR(32) DEFAULT NULL,
	apt_name VARCHAR(32) DEFAULT NULL,
	zip_code INT DEFAULT NULL
	);
```

2.3 Regular User

Model:

Regular User(user_ID, name, surname, date_of_birth)

FK: user_ID to User

Functional Dependencies:

user_ID → name, surname, date_of_birth

Candidate Keys:

user_ID

Primary Key:

user_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Regular User (
	user_ID INT PRIMARY KEY,
	FOREIGN KEY (user_ID) REFERENCES User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	name VARCHAR (32) DEFAULT NULL,
	surname VARCHAR (32) DEFAULT NULL,
	date_of_birth DATE DEFAULT NULL
	);
```

2.4 Professional User

Model:
Professional User(user_ID, professional_cumulative_rating, experience, expertise_field)
FK: user_ID to User

Functional Dependencies:
user_ID → progessional_cumulative_rating, experience, expertise_field

Candidate Keys:
user_ID

Primary Key:
user_ID

Normal Form:
BCNF

Table Declaration:

```
CREATE TABLE Professional User (
	user_ID INT PRIMARY KEY,
	FOREIGN KEY (user_ID) REFERENCES User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	experience INT DEFAULT NULL,
	expertise_field VARCHAR(32) DEFAULT NULL);
```

2.5 Past Services

Model:
Past Services(<u>service_type_ID, order_date, provider_ID</u>)
FK: service_type_ID to Services
FK: provider_ID to User

Functional Dependencies:
None

Candidate Keys:
{service_type_ID, order_date, provider_ID}

Primary Key:
service_type_ID

Normal Form:
BCNF

Table Declaration:

```
CREATE TABLE Past Services (
	service_type_ID INT PRIMARY KEY,
	FOREIGN KEY (service_type_ID) REFERENCES Services( service_type_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	order_date DATE PRIMARY KEY,
	provider_ID INT PRIMARY KEY,
	FOREIGN KEY (provider_ID) REFERENCES Professional User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

2.6 Service Order

Model:

Service Order(order_ID, service_type_ID, order_details)

FK: service_type_ID to Services

Functional Dependencies:

order_ID → service_type_ID, order_details

Candidate Keys:

order_ID

Primary Key:

order_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Service Order (
        order_ID INT PRIMARY KEY,
        service_type_ID INT
        FOREIGN KEY (service_type_ID) REFERENCES Services( service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        order_details VARCHAR(128));
```

2.7 Proposed Services

Model:

Proposed Services(proposal_ID ,service_type_ID, start_date, end_date,
proposed_price)
FK: service_type_ID to Services

Functional Dependencies:

proposal_ID → service_type_ID, start_date, end_date, proposed_price

Candidate Keys:

proposal_ID

Primary Key:

proposal_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Proposed Services (
        proposal_ID INT PRIMARY KEY,
        service_type_ID INT
        FOREIGN KEY (service_type_ID) REFERENCES Services( service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        start_date DATE,
        end_date DATE,
        proposed_price INT
        );
```

2.8 Proposed Collaborative Services

Model:

Proposed Collaborative Services(proposal_ID)

FK: (proposal_ID) to Proposed Services

Functional Dependencies:

None

Candidate Keys:

proposal_ID

Primary Key:

proposal_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Proposed Services (
	proposal_ID INT PRIMARY KEY,
	FOREIGN KEY (proposal_ID) REFERENCES Proposed Services( proposal_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	services VARCHAR(128)
);
```

2.9 Provided Services

Model:

Provided Services(<u>service_type_ID, custom_service_name</u>, service_rating,
service_starting_date, service_ending_date)
FK: service_type_ID to Services

Functional Dependencies:

service_type_ID, custom_service_name → service_rating, service_starting_date,
service_ending_date

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Provided Services (
        service_type_ID INT PRIMARY KEY,
        FOREIGN KEY (service_type_ID) REFERENCES Proposed Services(
        service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        custom_service_name VARCHAR(32) PRIMARY KEY,
        service_rating INT,
        service_starting_date DATE,
        service_ending_date DATE
        );
```

2.10 Service Rating_Evaluation

Model:

Service Rating_Evaluation(user_ID, service_type_ID, order_date, provider_ID, rating, evaluation)

FK: user_ID to Regular User

FK: (service_type_ID, order_date, provider_ID) to Past Services

Functional Dependencies:

user_ID, service_type_ID, order_date, provider_ID → rating, evaluation

Candidate Keys:

{user_ID, service_type_ID, order_date, provider_ID}

Primary Key:

user_ID, service_type_ID, order_date, provider_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Service Rating Evaluation (
        user_ID INT PRIMARY KEY,
        FOREIGN KEY (user_ID) REFERENCES Regular User( user_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        service_type_ID INT PRIMARY KEY,
        FOREIGN KEY (service_type_ID) REFERENCES Services( service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        order_date DATE PRIMARY KEY,
        provider_ID INT PRIMARY KEY,
        FOREIGN KEY (provider_ID) REFERENCES Professional User( user_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        rating INT,
        evaluation VARCHAR(128));
```

2.11 Has Taken

Model:

Has Taken(user_ID, service_type_ID, order_date, provider_ID)

FK: user_ID to Regular User

FK: (service_type_ID, order_date, provider_ID) to Past Services

Functional Dependencies:

None

Candidate Keys:

{user_ID, service_type_ID, order_date, provider_ID}

Primary Key:

{user_ID, service_type_ID, order_date, provider_ID}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Has Taken (
	user_ID INT PRIMARY KEY,
	FOREIGN KEY (user_ID) REFERENCES Regular User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	service_type_ID INT PRIMARY KEY,
	FOREIGN KEY (service_type_ID) REFERENCES Services( service_type_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	order_date DATE PRIMARY KEY,
	provider_ID INT PRIMARY KEY,
	FOREIGN KEY (provider_ID) REFERENCES Professional User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

2.12 Collaborators

Model:

Collaborators(proposal_ID, user_ID)

FK: proposal_ID to Proposed Collaborative Services

FK: user_ID to Professional User

Functional Dependencies:

proposal_ID → user_ID

Candidate Keys:

proposal_ID

Primary Key:

proposal_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Collaborators (
	proposal_ID INT PRIMARY KEY,
	FOREIGN KEY (proposal_ID) REFERENCES Proposed Services( proposal_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	user_ID INT,
	FOREIGN KEY (user_ID) REFERENCES Professional User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
	);
```

2.13 Proposals

Model:

Proposals(user_ID, proposal_ID,)

FK: professional_ID to Professional User

FK: proposal_ID to Proposed Services

Functional Dependencies:

None

Candidate Keys:

{user_ID , proposal_ID}, user_ID

Primary Key:

{user_ID , proposal_ID}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Proposals (
	professional_ID INT PRIMARY KEY,
	FOREIGN KEY (professional_ID ) REFERENCES Professional User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	proposal_ID INT PRIMARY KEY,
	FOREIGN KEY (proposal_ID) REFERENCES Proposed Services( proposal_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
	);
```

2.14 Has

Model:

Has(user_ID, order_ID)

FK: user_ID to Regular User

FK: order_ID to Service Order

Functional Dependencies:

None

Candidate Keys:

{user_ID, order_ID}

Primary Key:

{user_ID, order_ID}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Provides(
	order_ID INT PRIMARY KEY,
	FOREIGN KEY (order_ID) REFERENCES Service Order (order_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	user_ID INT PRIMARY KEY,
	FOREIGN KEY (user_ID) REFERENCES Regular User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

2.15 Provides

Model:
Provides (<u>user_ID, service_type_ID, custom_service_name</u>)
FK: user_ID to Professional User
FK: (service_type_ID, custom_service_name) to Services

Functional Dependencies:
None

Candidate Keys:
{user_ID, service_type_ID, custom_service_name}

Primary Key:
{user_ID, service_type_ID, custom_service_name}

Normal Form:
BCNF

Table Declaration:

```
CREATE TABLE Provides(
	user_ID INT PRIMARY KEY,
	FOREIGN KEY (user_ID) REFERENCES Professional User( user_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	service_type_ID INT PRIMARY KEY,
	FOREIGN KEY (service_type_ID) REFERENCES Provided Services
		(service_type_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	custom_service_name VARCHAR (32),
	FOREIGN KEY (custom_service_name)  REFERENCES Provided Services
	(custom_service_name)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

2.16 Related Services

Model:

Related Services(<u>service_type_ID</u>, related_service_type_ID)

FK: service_type_ID to Provided Services

FK: related_service_type_ID to Provided Services(service_type_ID)

Functional Dependencies:

service_type_ID → related_service_type_ID

Candidate Keys:

service_type_ID

Primary Key:

service_type_ID

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Related Services(
        service_type_ID INT PRIMARY KEY,
        FOREIGN KEY (service_type_ID)  REFERENCES Provided Services
        (service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        related_service_type_ID INT,
        FOREIGN KEY (related_service_type_ID)  REFERENCES Provided
                Services(related_service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
        );
```

2.17 Matches

Model:

Matches (proposal_ID, order_ID)

FK: proposal_ID to Proposals

FK: order_ID to Service Order

Functional Dependencies:

None

Candidate Keys:

{proposal_ID, order_ID}

Primary Key:

{proposal_ID, order_ID}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Matches(
	order_ID INT PRIMARY KEY,
	FOREIGN KEY (order_ID) REFERENCES Service Order (order_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	proposal_ID INT PRIMARY KEY,
	FOREIGN KEY (proposal_ID) REFERENCES Proposals (proposal_ID)
		ON DELETE CASCADE
		ON UPDATE CASCADE
	);
```

2.18 Private Lesson

Model:

Private Lesson(<u>service_type_ID, custom_service_name</u>, course_type, hourly_rate, frequency)

FK: (service_type_ID, custom_service_name) to Provided Services

Functional Dependencies:

service_type_ID, custom_service_name → course_type, hourly_rate, frequency

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Proposed Services (
        service_type_ID INT PRIMARY KEY,
        FOREIGN KEY (service_type_ID)  REFERENCES Provided Services
        (service_type_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
         custom_service_name VARCHAR(32),
        FOREIGN KEY (custom_service_name)  REFERENCES Provided Services
        (custom_service_name)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        course_type VARCHAR(32),
        hourly_rate INT,
        frequency VARCHAR(32)
        );
```

2.19 Repair Service

Model:

Repair Service(service_type_ID, custom_service_name, base_material_price,
item_type)
FK: (service_type_ID, custom_service_name) to Provided Services

Functional Dependencies:

service_type_ID, custom_service_name → base_material_price, item_type

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Repair Service(
    service_type_ID INT PRIMARY KEY,
    FOREIGN KEY (service_type_ID)  REFERENCES Provided Services
    (service_type_ID),
    custom_service_name VARCHAR(32),
    FOREIGN KEY (custom_service_name)  REFERENCES Provided Services
    (custom_service_name),
    base_material_price INT,
    item_type VARCHAR(32)
    );
```

2.20 Cleaning Service

Model:

Cleaning Service (<u>service_type_ID, custom_service_name</u>, base_room_price,
base_bathroom_price, worker_rate, frequency)
FK: (service_type_ID, custom_service_name) to Provided Services

Functional Dependencies:

service_type_ID, custom_service_name → base_room_price, base_bathroom_price,
worker_rate, frequency

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Cleaning Service(
    service_type_ID INT PRIMARY KEY,
    FOREIGN KEY (service_type_ID)  REFERENCES Provided Services
    (service_type_ID),
    custom_service_name VARCHAR(32),
    FOREIGN KEY (custom_service_name)  REFERENCES Provided Services
    (custom_service_name),
    base_room_price INT,
    base_bathroom_price INT,
    worker_rate INT,
    Frequency VARCHAR(32)
    );
```

2.21 Painting Service

Model:

Painting Service(<u>service_type_ID, custom_service_name</u>, total_volume, color_type, number_of_rooms)

FK: (service_type_ID, custom_service_name) to Provided Services

Functional Dependencies:

service_type_ID, custom_service_name → total_volume, color_type, number_of_rooms

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Painting Service(
    service_type_ID INT PRIMARY KEY,
    FOREIGN KEY (service_type_ID)  REFERENCES Provided Services
    (service_type_ID),
    custom_service_name VARCHAR(32),
    FOREIGN KEY (custom_service_name)  REFERENCES Provided Services
    (custom_service_name),
    total_volume INT,
    color_type VARCHAR(32),
    number_of_rooms INT
    );
```

2.22 Moving Service

Model:

Moving Service(<u>service_type_ID, custom_service_name</u>, new_city_name,
new_zip_code, new_apt_name, new_street_number, city_name, zip_code,
apt_name, street_number, number_of_packages)
FK: (service_type_ID, custom_service_name) to Provided Services

Functional Dependencies:

service_type_ID, custom_service_name → new_city_name, new_zip_code,
new_apt_name, new_street_number, city_name, zip_code, apt_name,
street_number, number_of_packages

Candidate Keys:

{service_type_ID, custom_service_name}

Primary Key:

{service_type_ID, custom_service_name}

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Moving Service(
    service_type_ID INT PRIMARY KEY,
    FOREIGN KEY (service_type_ID) REFERENCES Provided Services
    (service_type_ID),
    custom_service_name VARCHAR(32),
    FOREIGN KEY (custom_service_name) REFERENCES Provided Services
    (custom_service_name),
    new_city_name VARCHAR(32),
    new_zip_code INT,
    new_apt_name VARCHAR(32),
    new_street_number INT,
    city_name VARCHAR(32),
    zip_code INT,
    apt_name VARCHAR(32),
    street_number INT,
```

```
number_of_packages INT
);
```

2.23 Provided

Model:
Provided Services(<u>service_type_ID, order_date, provider_ID</u>)
FK: (service_type_ID, order_date) to Past Services
FK: provider_ID to Professional User

Functional Dependencies:
None

Candidate Keys:
{service_type_ID, order_date, provider_ID}

Primary Key:
service_type_ID

Normal Form:
BCNF

Table Declaration:

```
CREATE TABLE Provided (
    service_type_ID INT PRIMARY KEY,
    FOREIGN KEY (service_type_ID) REFERENCES Past Services(
        service_type_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
order_date DATE PRIMARY KEY,
    FOREIGN KEY (service_type_ID) REFERENCES Past Services(
        service_type_ID)
    ON DELETE CASCADE
ON UPDATE CASCADE,
provider_ID INT PRIMARY KEY,
FOREIGN KEY (provider_ID) REFERENCES Professional User( user_ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE);
```

3. Normalization Of Tables

The Relational Schemas of the design show the functional dependencies. No decomposition or normalization was applied since all relations are in Boyce-Codd Normal Form.

4. Functional Components

4.1 Algorithms

4.1.1 Order and Proposal Related Algorithms

The users of the program will be able to make orders about the services which they want. They need to specify the type of the service and details of the order. Details of the order are taken into the database by following restrictions applied according to the selected service type. Professional Users will propose to the orders by giving time and price information. Many professionals can propose to an order which are stored in the matches table. Regular user will be able to see the proposed orders to his/her order and select one. Regular user will be able to accept an offer which he or she wants. Professional users can propose to many orders also. In addition, the system allows many professionals to propose to a single order. This is a kind of proposal similar to others, which has additional data stored about collaborators.

4.1.2 Logical Requirements

The system should be working without logical errors. Date attributes such as service dates and proposal dates should be checked because ending date cannot come before starting date. Also such dates shouldn’t start before the current date.

4.2 Data Structures

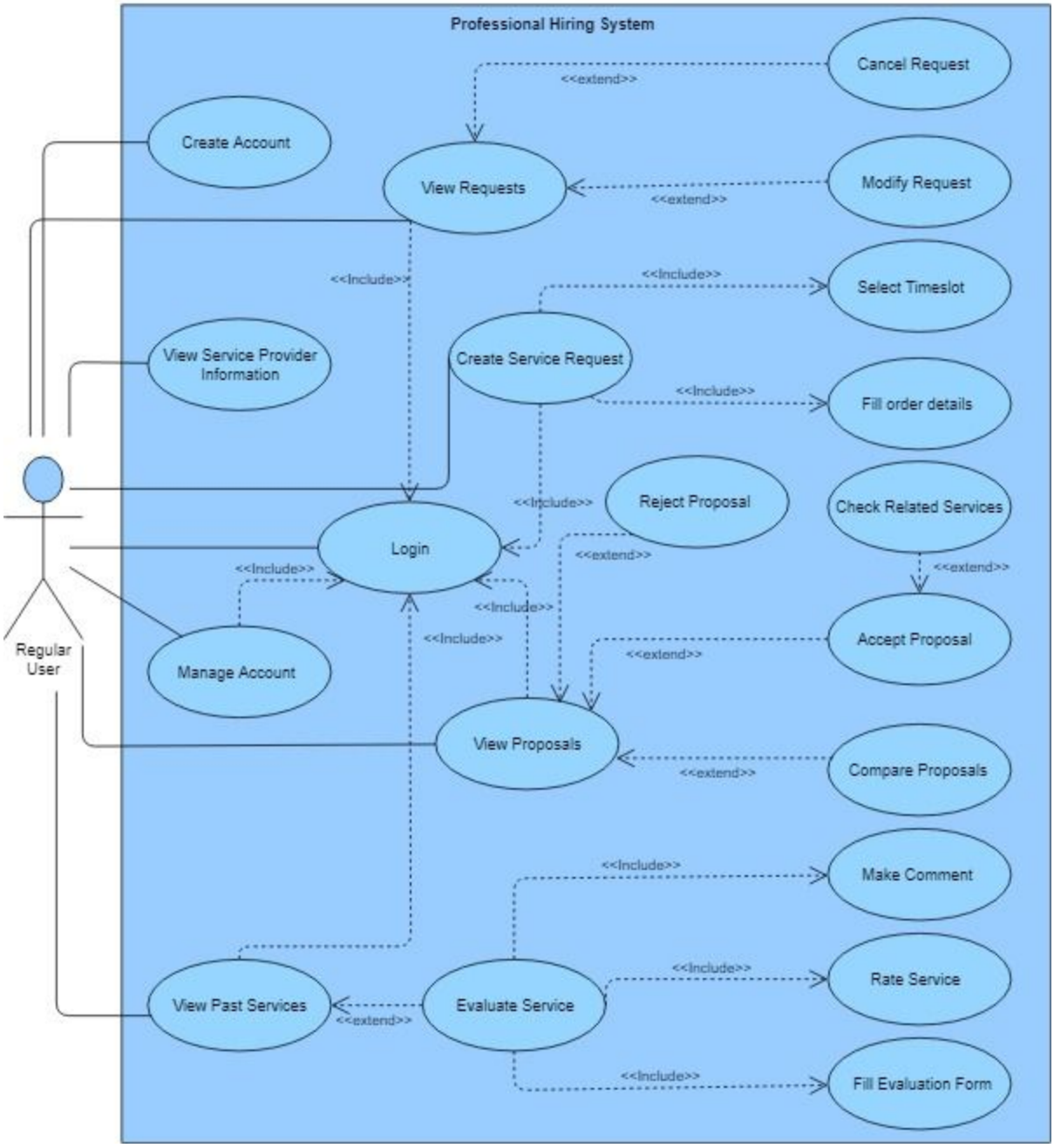
Numeric, alphabetic and date types were used. Numeric values are stored as INT, while String types are stored as VARCHAR. VARCHAR was used for string types because string data which we store are not known before. Date values are stored as DATE type.

4.3 Use Cases / Scenarios

4.3.1 Regular Users

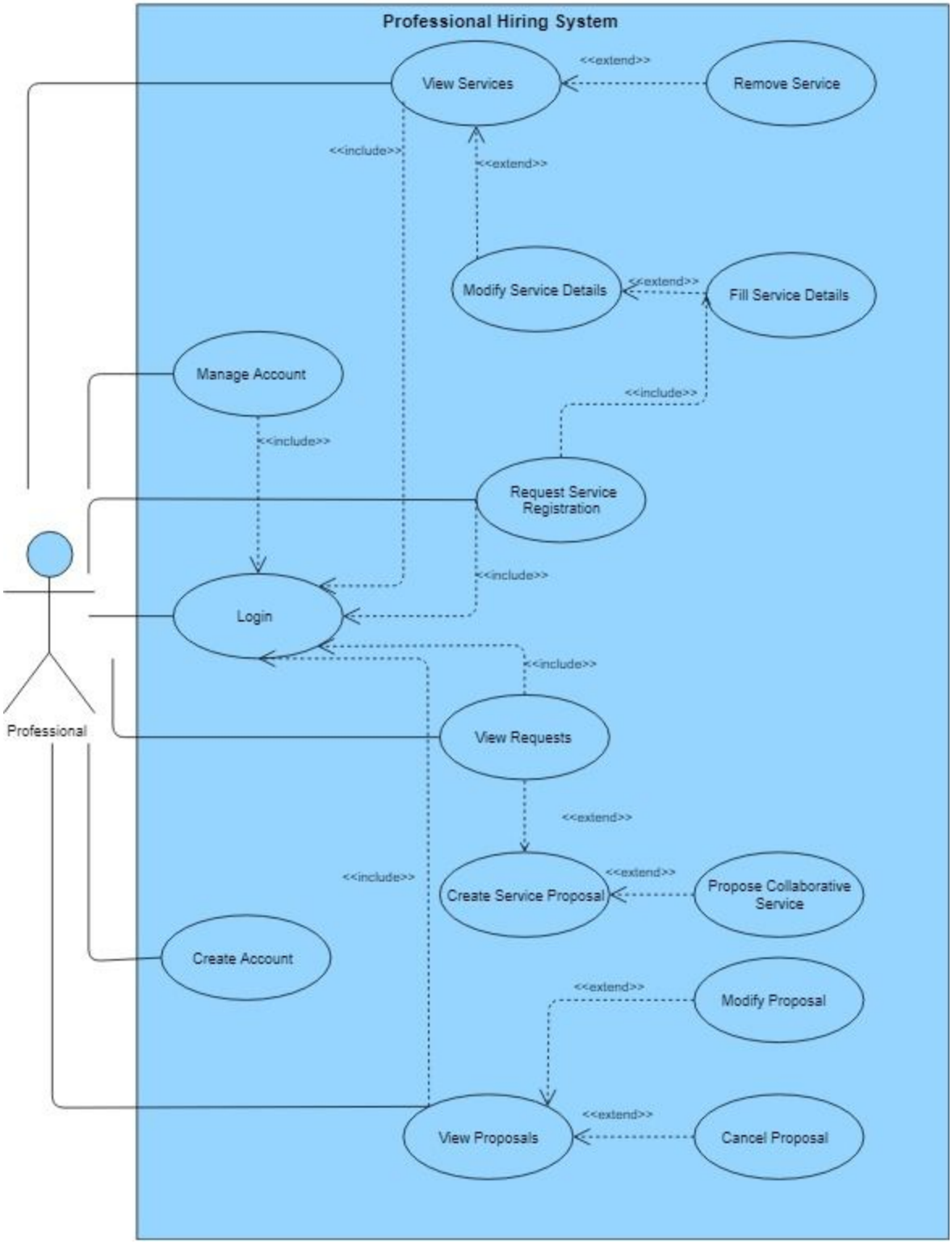
- **Create Account:** Regular users can create an account by providing email address, username, password and address information. Email address and username must be unique in order to create an account.
- **Login:** Regular users can login to the system with their username or email address and password. Once they logged in, regular users can manage their account, create a service request, view their existing service requests, view proposals and view past services. View service provider information operation can be done without logging into the system.
- **Manage Account:** Once logged in, regular users can update their account information such as username, password, email address, name, surname, date of birth and address information. Changing the username and email address requires extra caution since the new username and email address must be unique as well.
- **View Service Provider Information:** Regular users can view a specific service providers information such as their address location, experience and expertise field, cumulative rating, user comments, evaluation forms. Logging into the system is not an requirement.
- **View Requests:** Regular users can view their existing service requests and select a specific request among them to cancel it or modify its order details.
- **Cancel Request:** Regular users can cancel an existing request.
- **Modify Request:** Regular users can modify order details of an service request, which enables them to change service type or any other detail like changing time slot.
- **Create Service Request:** Regular users can create a service request, which will be seen by the professionals after the creation. In order to create a service request, users have to select a time slot and fill the other order details.
- **Select Time slot:** Regular users can select a time slot while creating a service request, which indicates that when the user wants to receive the service. This information can be seen by the professional users enabling them to prepare proposals to the service requests that fit their time schedule.
- **Fill Order Details:** Regular users can fill the order details of the requested service. These information vary with respect to the type of the service. For example, order details of a painting service is different from the order details of a repair service.

- **View Proposals:** Regular users can view the proposals to their existing service requests. These proposals are created by the professional users, which includes information about proposed price, start and end dates.
- **Accept Proposal:** Regular users can accept a proposal if they agree upon the proposed terms.
- **Reject Proposal:** Regular users can reject a proposal if they do not agree upon the proposed terms.
- **Compare Proposals:** Regular users can compare two proposals the find out which one is more beneficial for them. This comparison will show proposed price, start and end dates of both proposals.
- **Check Related Services:** Upon accepting a proposal, regular users can hire extra services which are related to the proposed service. For instance, after a painting service, regular users might want to hire a cleaning service, which will be provided by this operation.
- **View Past Services:** Regular users can view the services they got in the past through the system. Which enables them to get information about the professional that they hired for the service. Also, regular users can evaluate a past service, through this operation.
- **Evaluate Service:** Evaluating a past service includes rating a service, filling an evaluation form about the service and making personal comments about the service and professional.
- **Make Comment:** Regular users can make a comment for a professional and service provided by that professional, which can guide the other regular users who considers hiring the professional.
- **Rate Service:** Regular users can rate a service they received out of 10. Not only regular users will be able to see the user ratings but also, each rating affects the cumulative rating of the professional.
- **Fill Evaluation Form:** Regular users can fill evaluation form to evaluate a service they got. This evaluation form has questions about both the service and the professional. Regular users can access these evaluation forms through view service provider information operation.


Professional Hiring System
Cancel Request
<<extend>>
Create Account
View Requests
Modify Request
<<extend>>
<<Include>>
<<Include>>
Select Timeslot
View Service Provider Information
Create Service Request
<<Include>>
Fill order details
<<Include>>
Reject Proposal
Check Related Services
Login
<<extend>>
<<extend>>
<<Include>>
<<Include>>
<<Include>>
Accept Proposal
Regular User
Manage Account
<<extend>>
View Proposals
Compare Proposals
<<extend>>
<<Include>>
Make Comment
<<Include>>
Rate Service
View Past Services
Evaluate Service
<<extend>>
<<Include>>
Fill Evaluation Form

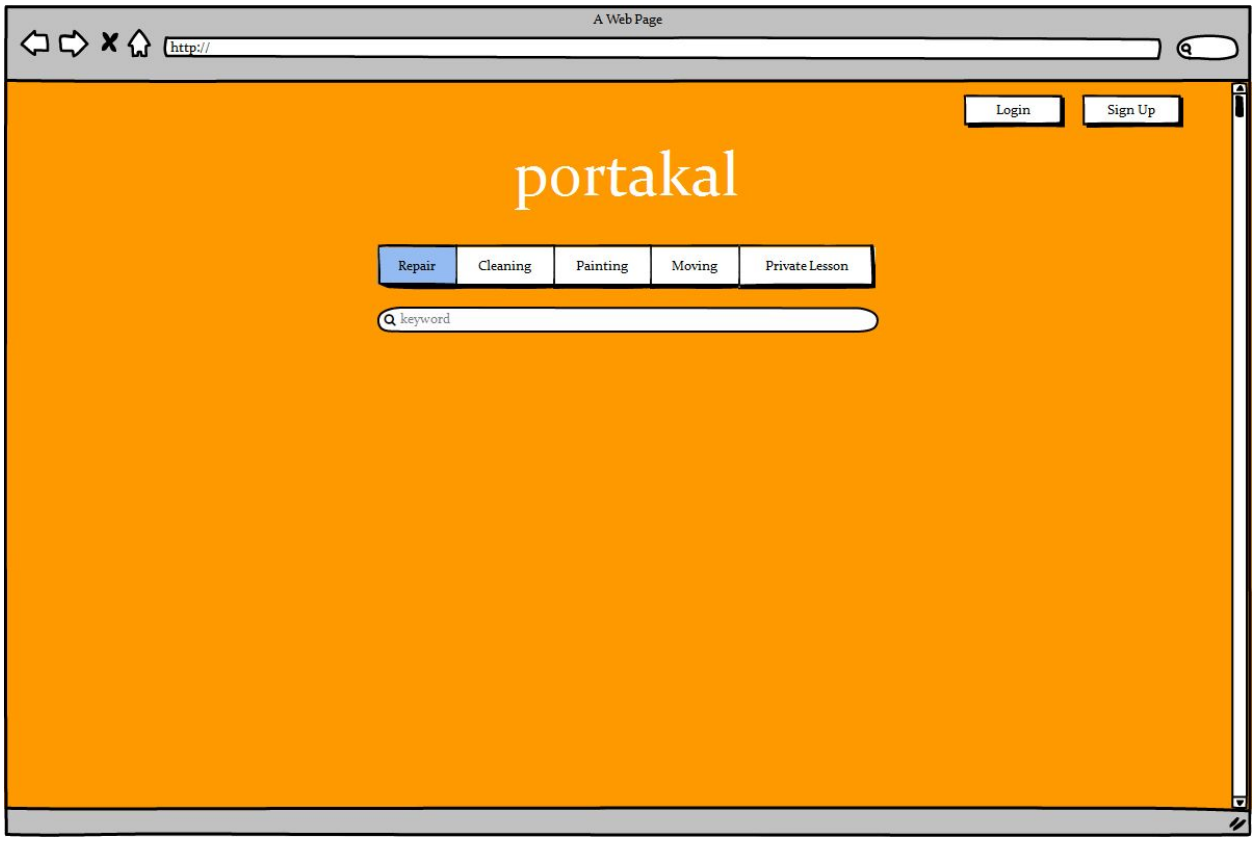
4.3.2 Professionals

- **Create Account:** Professional users can create an account by providing email address, username, password and address information. Email address and username must be unique in order to create an account.
- **Login:** Professionals can login to their account, with the username or email address and password. professionals can change those information using manage account operation. The operations that are described below, requires logging into the system.
- **Manage Account:** As mentioned above, professionals can manage their account, by changing their username, email address, address information, password, experience and expertise field. The new username, and email address must be unique as well.
- **View Proposals:** Professionals can view their proposals, which enables them to modify or cancel a proposal.
- **Modify Proposal:** Professionals can modify an existing proposal by changing proposed price, start and end dates.
- **Cancel Proposal:** Professionals can cancel a proposal, which discards all information about the proposal.
- **Create Service Proposal:** Professionals can create proposals for regular users' service requests. In the proposals, professionals provide a price information which such a service would cost, a starting date which represents when they can start working and a possible end date.
- **Propose Collaborative Service:** While creating a proposal for a service request, professionals can propose a special service, for such a service request, in which they will work with other professionals collaboratively.
- **Service Registration:** Professionals can create registration for a new service. Professionals have to fill service details about the new service and request its registration.
- **Fill Service Details:** Professionals can fill the service details, which includes information about base prices, which will be used to determine the proposed price while creating a proposal.
- **View Provided Services:** Professionals can view the services they provide.
- **Modify Service Details:** Professionals can modify the details about the services they provide.
- **Remove Service:** Professionals can remove a service they provide.


Professional Hiring System
View Services
Remove Service
<<extend>>
<<include>>
<<extend>>
Modify Service Details
<<extend>>
Fill Service Details
Manage Account
<<include>>
<<include>>
Request Service Registration
<<include>>
Login
<<include>>
Professional
View Requests
<<extend>>
<<include>>
Create Service Proposal
<<extend>>
Propose Collaborative Service
Create Account
<<extend>>
Modify Proposal
View Proposals
<<extend>>
Cancel Proposal

5. User Interface Design and Corresponding SQL Statements

5.1 Homepage



Inputs: @service_name
Process: Homepage is first page of the Online Professional Hiring System. The users who are not logged in will be greeted with this page. From this page, the users can browse through service types and check the professional homepages.
SQL Statements:
service_view:

```
CREATE VIEW service_view(service_name, professional_ID, professional_rating,
professional_experience)
AS SELECT custom_service_name, user_ID, professional_cumulative_rating,
experience
FROM Professional User NATURAL JOIN (SELECT *
                                     FROM Provides
                                     WHERE custom_service_name =
                                @service_name)
ORDER BY professional_cumulative_rating DESC;
```

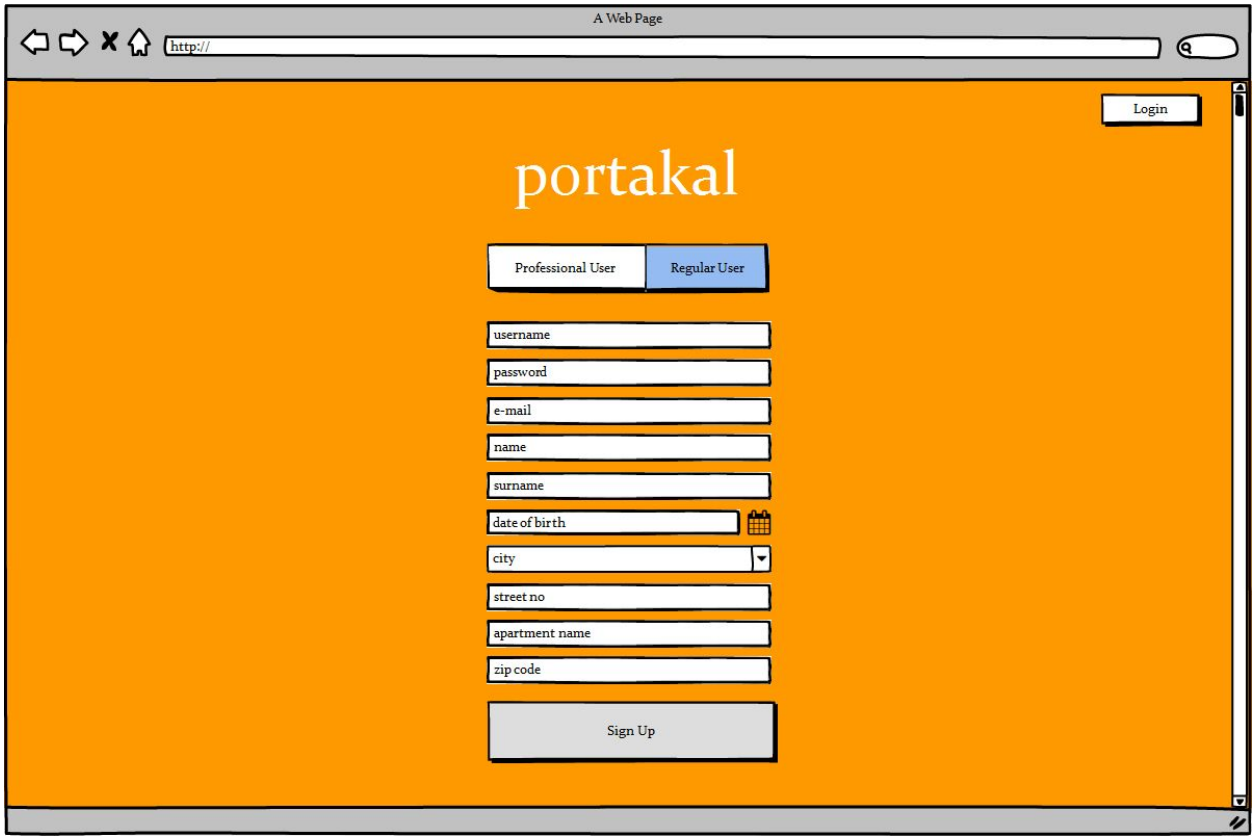
5.2 Login for Regular and Professional User

Inputs: @username or @email and @password.
Process: The login screen is for the users who have not logged into the system. Both professional and regular users can log in to the system by entering either their username or email with their password information.

SQL Statements:

```
SELECT username, email, password
FROM User
WHERE (username = @username OR
        email = @email) AND
        password = @password);
```

5.3 Sign Up for Regular User

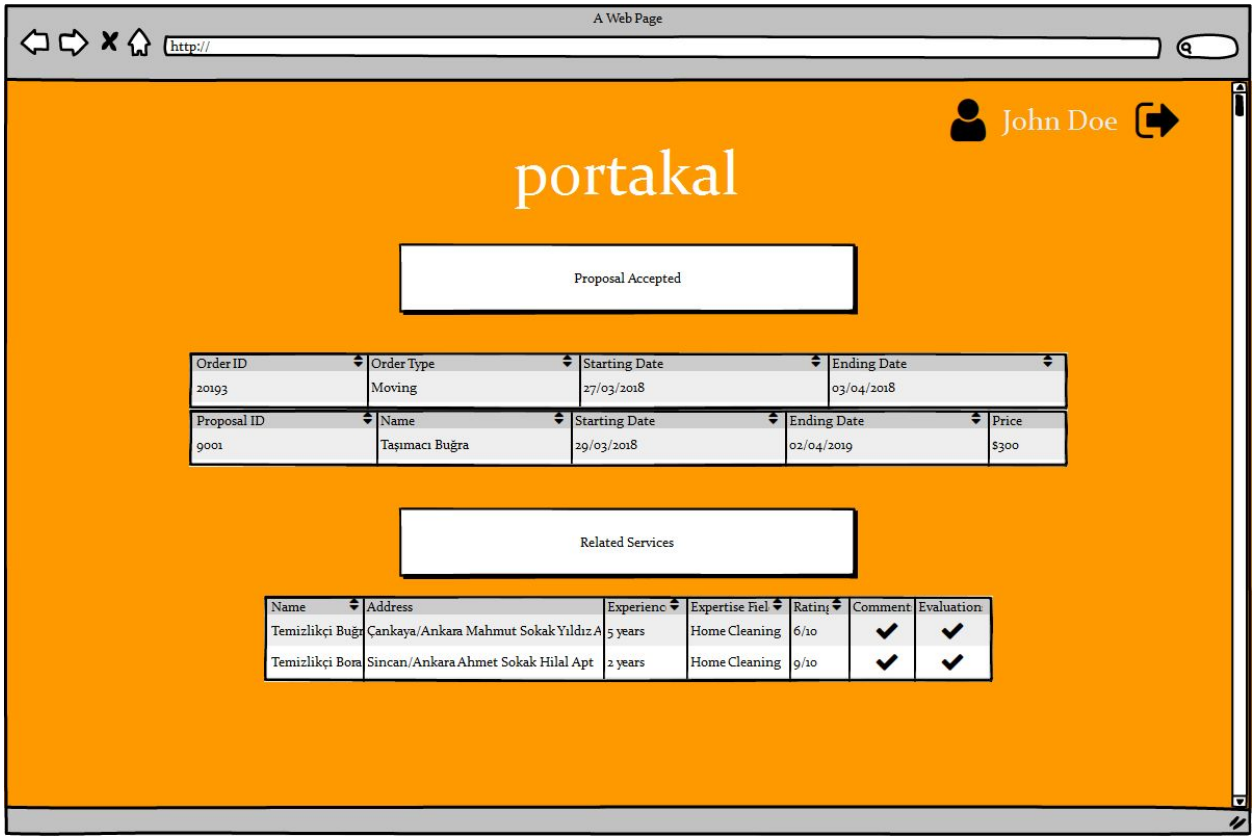


Input: @username, @password, @email, @name, @surname, @date_of_birth, @city @street_no @apartment_name @zip_code
Process: A user can access the sign up page through clicking “sign up” in either homepage, or in login page. In the Sign Up page, users who selects “Regular User” option will be greeted with the page above. In order to create a regular user account, they have to provide at least username, password and email information.

SQL Statements:

INSERT INTO User(password, email, username, city_name, street_number, apt_name, zip_code)
VALUES(@password, @email, @username, @city_name, @street_number, @apt_name, @zip_code);
INSERT INTO Regular User(name, surname, date_of_birth)
VALUES(@name, @surname, @date_of_birth);

5.4 Accept Proposal for Regular User



Inputs: @proposal_ID, @professional_ID,
Process: Upon accepting a proposal, this screen will pop-up and regular users will be able to see the details of accepted proposal, and the services which relates to the accepted proposal.

SQL Statements :
View Order Details

```
SELECT *
FROM Service Order
WHERE order_ID = @order_ID;
```

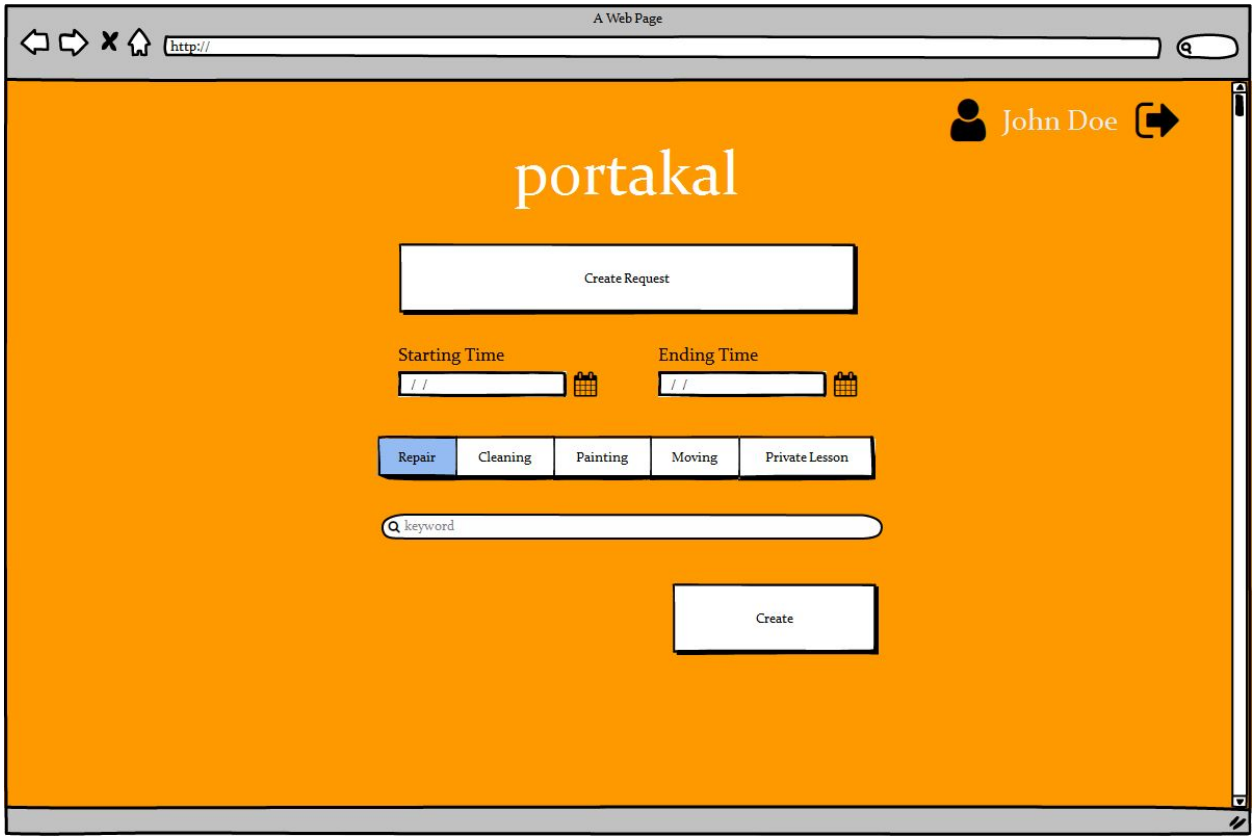
View Proposal Details

```
SELECT *
FROM Proposed Services as P, lateral
        (SELECT *
         FROM (SELECT proposal_ID, user_ID,
                  FROM Professional User NATURAL JOIN Proposals) as T
                  WHERE P.proposal_ID = T.proposal_ID)
        )
WHERE P.proposal_ID = @proposal_ID;
```

View Related Services

```
SELECT *
FROM Provided Services as P,
        (SELECT related_service_type_ID
        FROM Related Services NATURAL JOIN
                (SELECT service_type_ID
                FROM Service Order
                WHERE order_ID = @order_ID)
        ) as T
WHERE P.service_type_ID = T.related_service_type_ID;
```

5.5 Create Service Request for Regular User



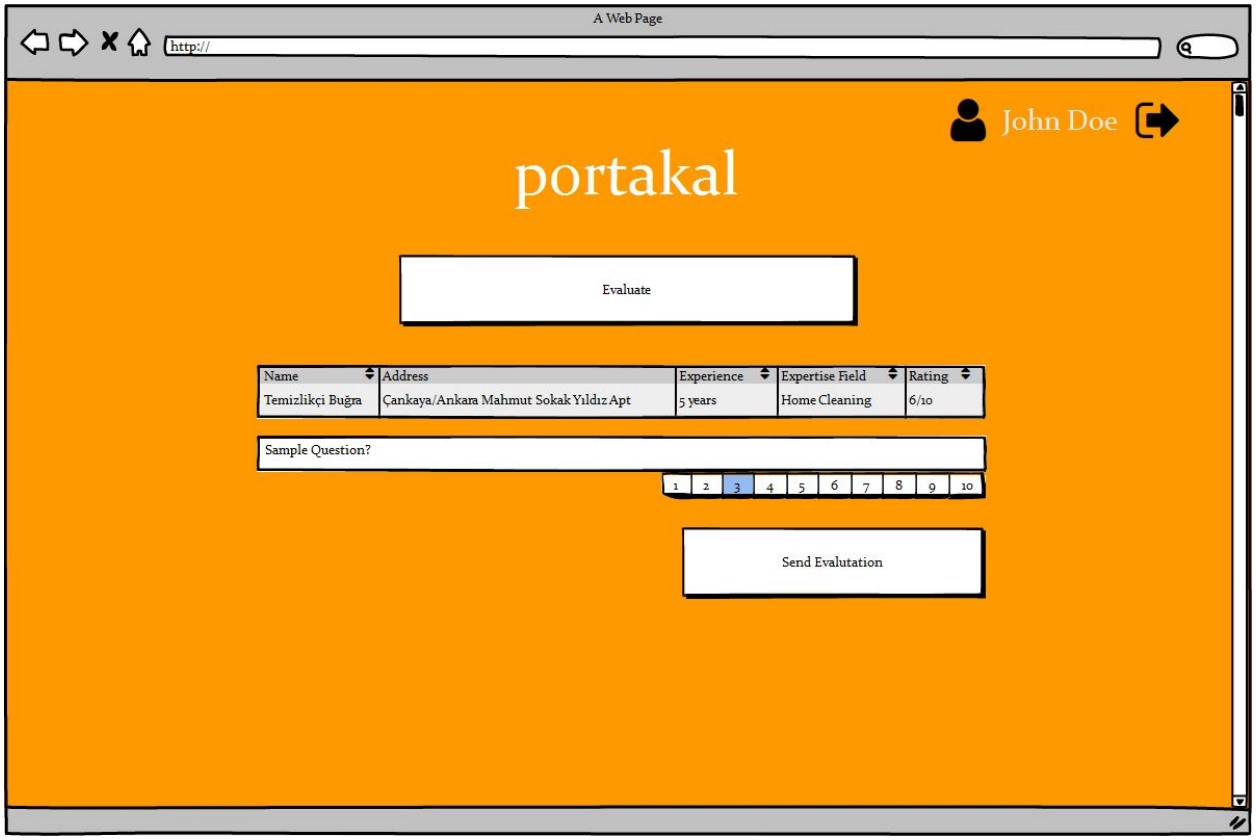
Input: @starting_time, @ending_time, @service_type, @order_details
Process: A regular user can access the create service request page from the log-on screen. In order to create a service request, regular users have to provide starting and ending time of the service. They also need to provide type of the service by selecting one of the service type buttons.

SQL Statements:

Request Service:

INSERT INTO Service Order(service_type_id, service_details)
VALUES(@service_type, @order_details);

5.6 Evaluate Service for Regular User

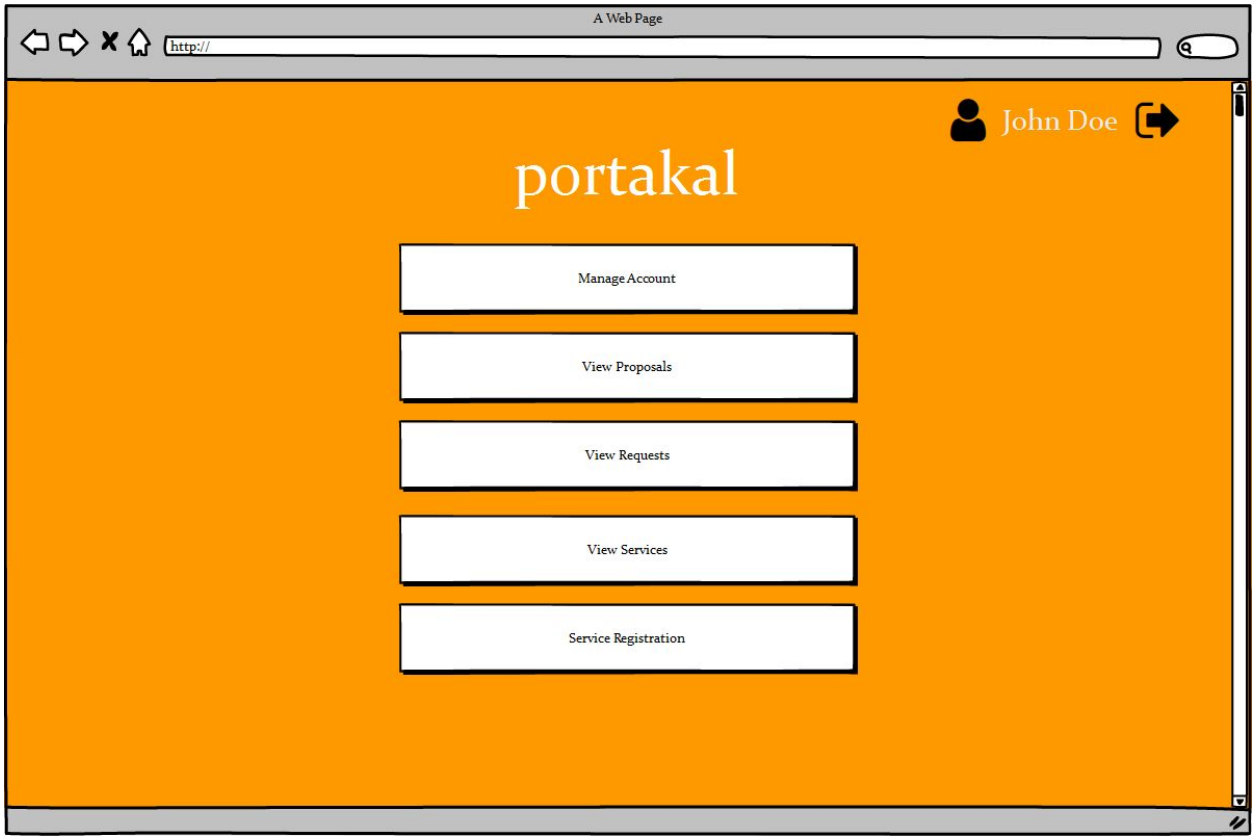


Input: @rating
Process: Regular users can evaluate service from this page. There will be a value calculated from given input to be stored in the database.

Evaluate:

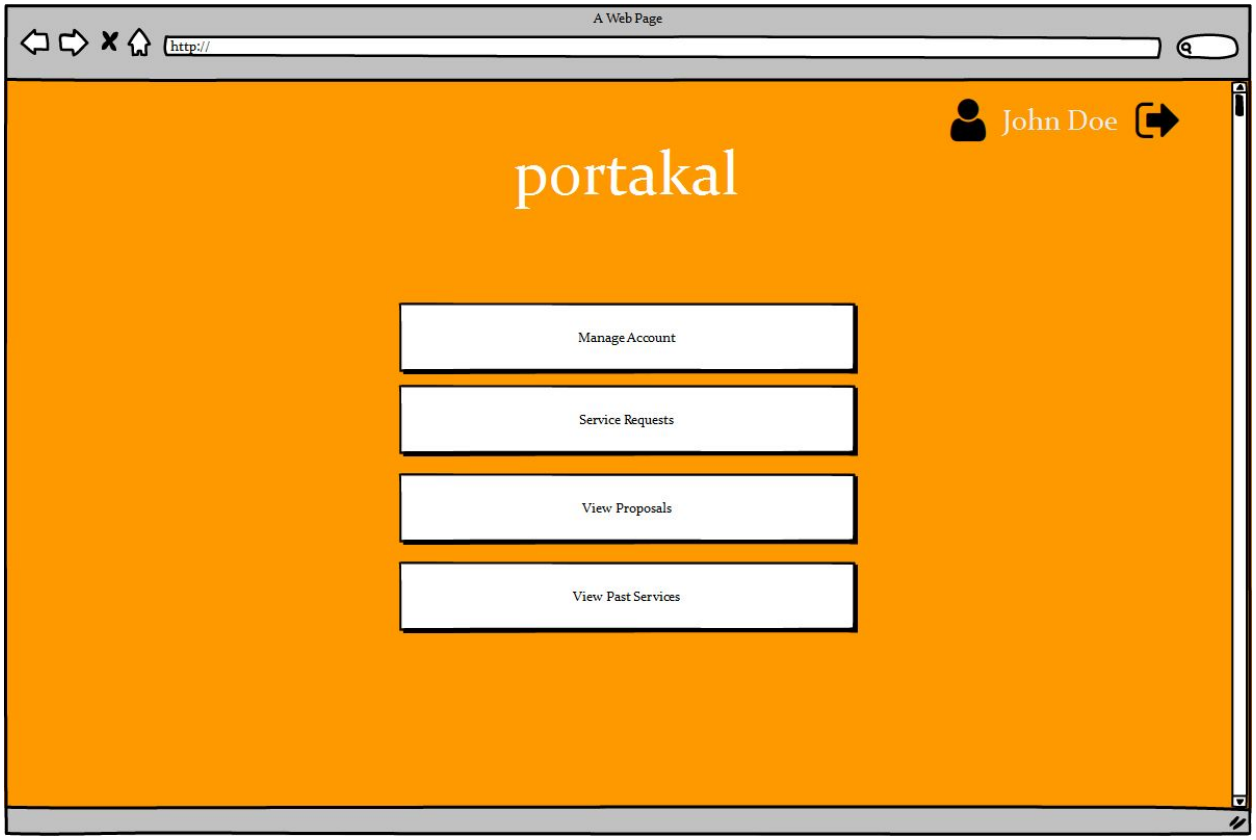
```
UPDATE Service Rating_Evaluation
SET rating = @rating
WHERE user_ID = @user_ID AND order_date = @order_date AND service_type_ID =
        @service_type_ID AND provider_ID = @provider_ID;
```

5.7 Logon for Professional User



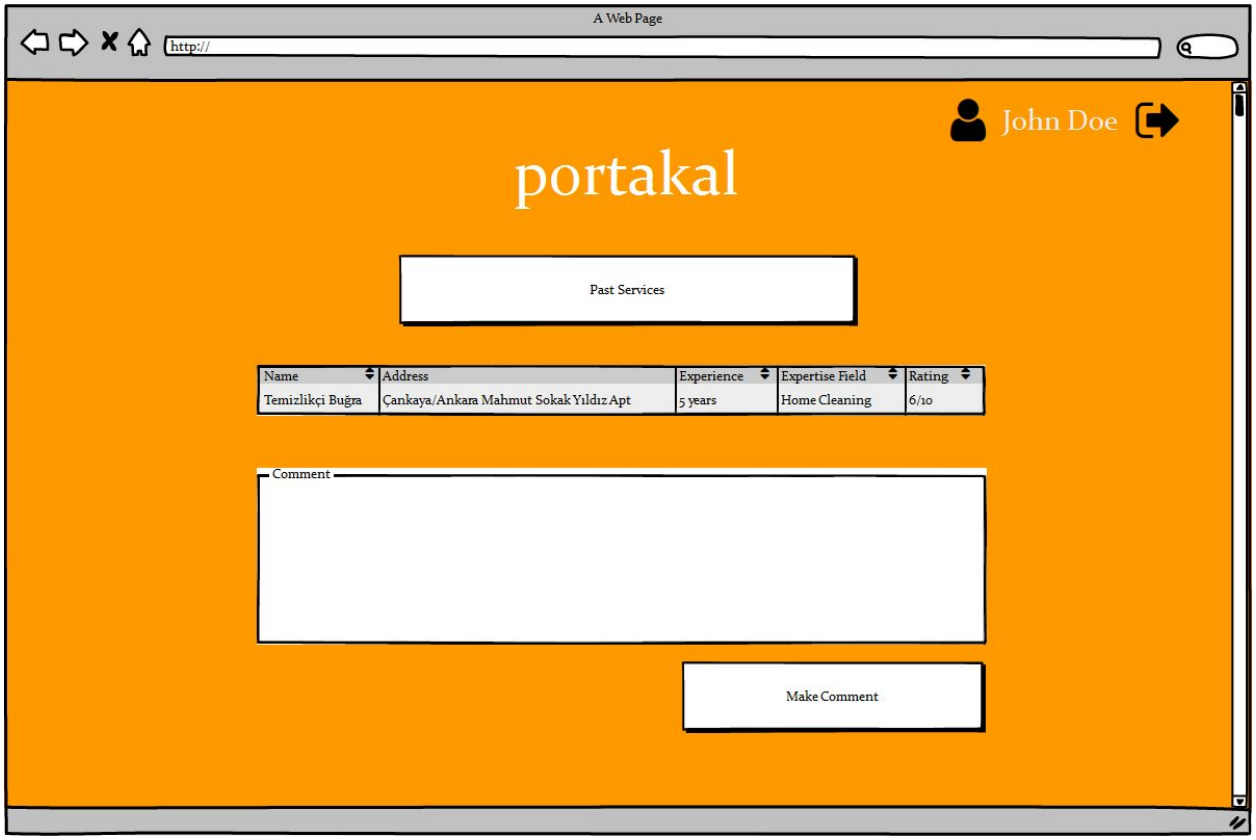
Process: Professional users can access this screen through logging in to the system. By using this page, professional users could manage their accounts, view their time schedules, view proposals, view service requests, view services and request service registration.

5.8 Logon for Regular User



Process: Regular users can access this screen through logging in to the system. By using this page, regular users could manage their accounts, view their service requests, view their proposals and view past service.

5.9 Make Comment for Regular User



Inputs: @evaluation

Process: Regular users can access this screen by selecting a past service and clicking the comment button from the past services screen.

SQL Statements

```
INSERT INTO Service Rating_Evaluation(user_ID, service_type_ID, order_date,
        provider_ID, rating, evaluation)
VALUES(@user_ID, @service_type_ID, @order_date, @provider_ID, @rating,
        @evaluation);
```

5.10 Manage Account for Professional User

Input: @username, @password, @email, @city, @street_number, @apartment_name, @zip_code, @expertise_field, @experience

Process: User can change details of his/her account by writing in to the defined fields. Pressing the Update button saves changes to the database.

SQL Statements:

View Current Account Details:

```
SELECT *
FROM User NATURAL JOIN Professional User
WHERE user_ID = @user_ID;
```

Change Username:

```
UPDATE User
SET username = @new_user_name
WHERE user_ID = @user_ID AND
                    @new_user_name NOT IN (SELECT username
                                                FROM User);
```

Change Password:

```
UPDATE User
SET password = @new_password
WHERE user_ID = @user_ID AND @new_password NOT NULL;
```

Change Email:

```
UPDATE User
SET email = @new_email
WHERE user_ID = @user_ID AND
                    @new_email NOT IN (SELECT email
                                           FROM User);
```

Change City:

```
UPDATE User
SET city = @new_city
WHERE user_ID = @user_ID;
```

Change Street Number:

```
UPDATE User
SET street_number = @new_street_no
WHERE user_ID = @user_ID;
```

Change Zip Code:

```
UPDATE User
SET zip_code = @new_zip_code
WHERE user_ID = @user_ID;
```

Change Apartment Name:

```
UPDATE User
SET apartment_name = @apartment_name
WHERE user_ID = @user_ID;
```

Change Expertise field:

```
UPDATE Professional User
SET expertise_field = @expertise_field
WHERE user_ID = @user_ID;
```

Change Experience:

```
UPDATE Professional User
SET experience = @experience
WHERE user_ID = @user_ID;
```

5.11 Manage Account for Regular User

Inputs: @new_username, @new_password, @new_email, @new_name, @new_surname, @new_date_of_birth, @new_city, @new_street_no, @new_apartment_name, @new_zip_code

Process: Logged in regular users can manage their account information from the screen displayed above. The regular users are able to change their username, password, email, name, surname, date of birth, and address information. However, the username and email address information they provided must be unique for this update.

SQL Statements:

View Current Account Details:

```
SELECT *
FROM User NATURAL JOIN Regular User
WHERE user_ID = @user_ID;
```

Change Username:

```
UPDATE User
SET username = @new_user_name
WHERE user_ID = @user_ID AND
                    @new_user_name NOT IN (SELECT username
                                                FROM User);
```

Change Password:

```
UPDATE User
SET password = @new_password
WHERE user_ID = @user_ID AND @new_password NOT NULL;
```

Change Email:

```
UPDATE User
SET email = @new_email
WHERE user_ID = @user_ID AND
                    @new_email NOT IN (SELECT email
                                           FROM User);
```

Change City:

```
UPDATE User
SET city = @new_city
WHERE user_ID = @user_ID;
```

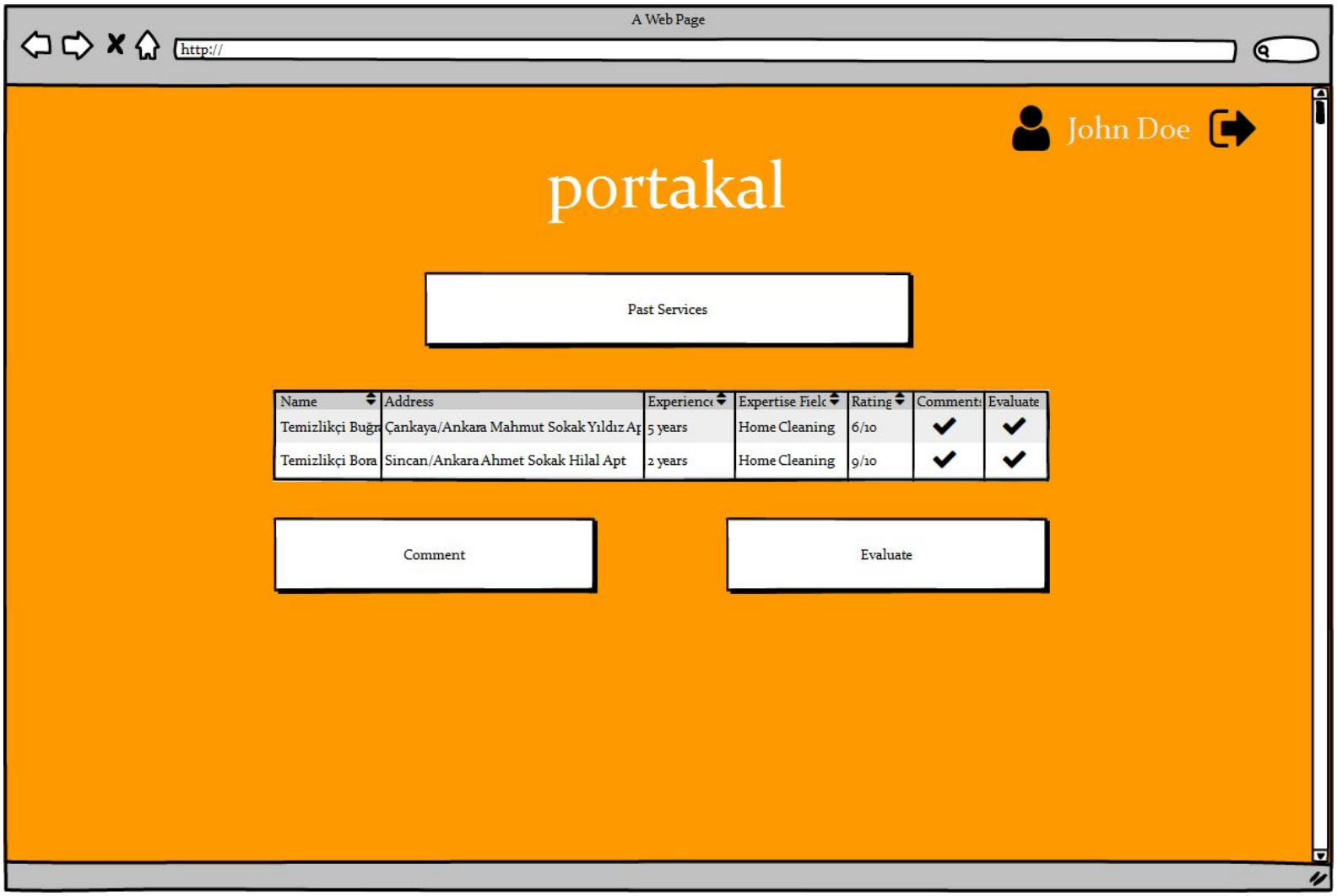
Change Name:

```
UPDATE Regular User
SET name = @new_name
WHERE user_ID = @user_ID;
```

Change Surname:

```
UPDATE Regular User
SET surname = @new_surname
WHERE user_ID = @user_ID;
```

5.12 Past Services for Regular User



Inputs: @rating, @evaluation
Process: Regular User is shown the screen where past services which was taken by that user is shown in the middle. This user will be able to press the Comment button to enter a written comment, which is denoted as evaluation in the database and. Also user is able to click on the Evaluate button which will get a integer value from the new screen. This is denoted as rating in the database.

SQL Statements:

View Past Services Taken:

```
SELECT *
FROM Regular User NATURAL JOIN Has Taken
WHERE user_ID = @user_ID;
```

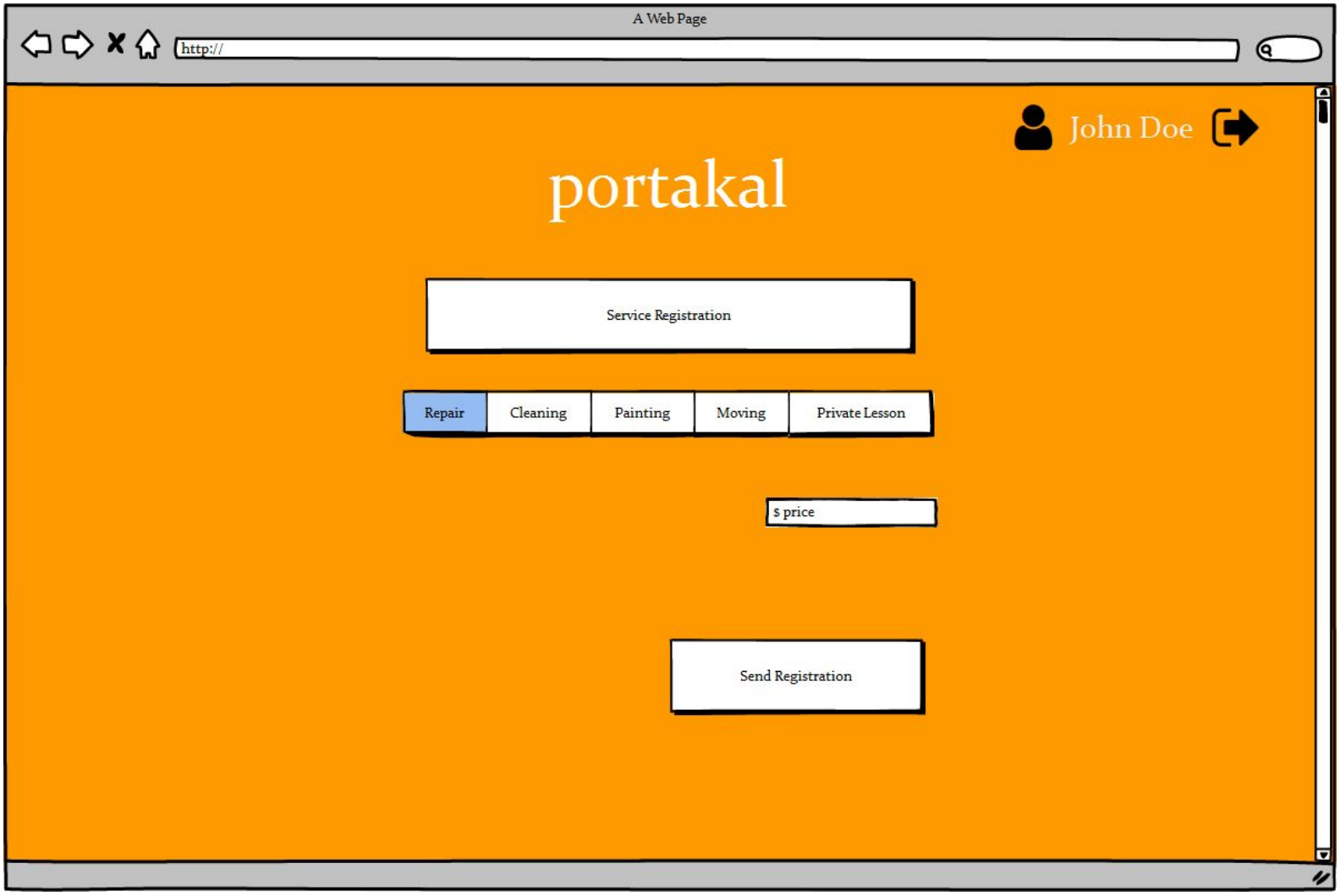
Evaluate:

```
UPDATE Service Rating_Evaluation
SET rating = @new_rating
WHERE user_ID = @user_ID AND order_date = @order_date AND service_type_ID =
        @service_type_ID AND provider_ID = @provider_ID;
```

Comment:

```
UPDATE Service Rating_Evaluation
SET evaluation = @new_evaluation
WHERE user_ID = @user_ID AND order_date = @order_date AND service_type_ID =
        @service_type_ID AND provider_ID = @provider_ID;
```

5.13 Service Registration for Professional User



Input: @user_ID, @service_type_ID, @custom_service_name
Process: Professional user selects a predefined service from the menu and enter details for that service. This service is added to the provides and provided services.

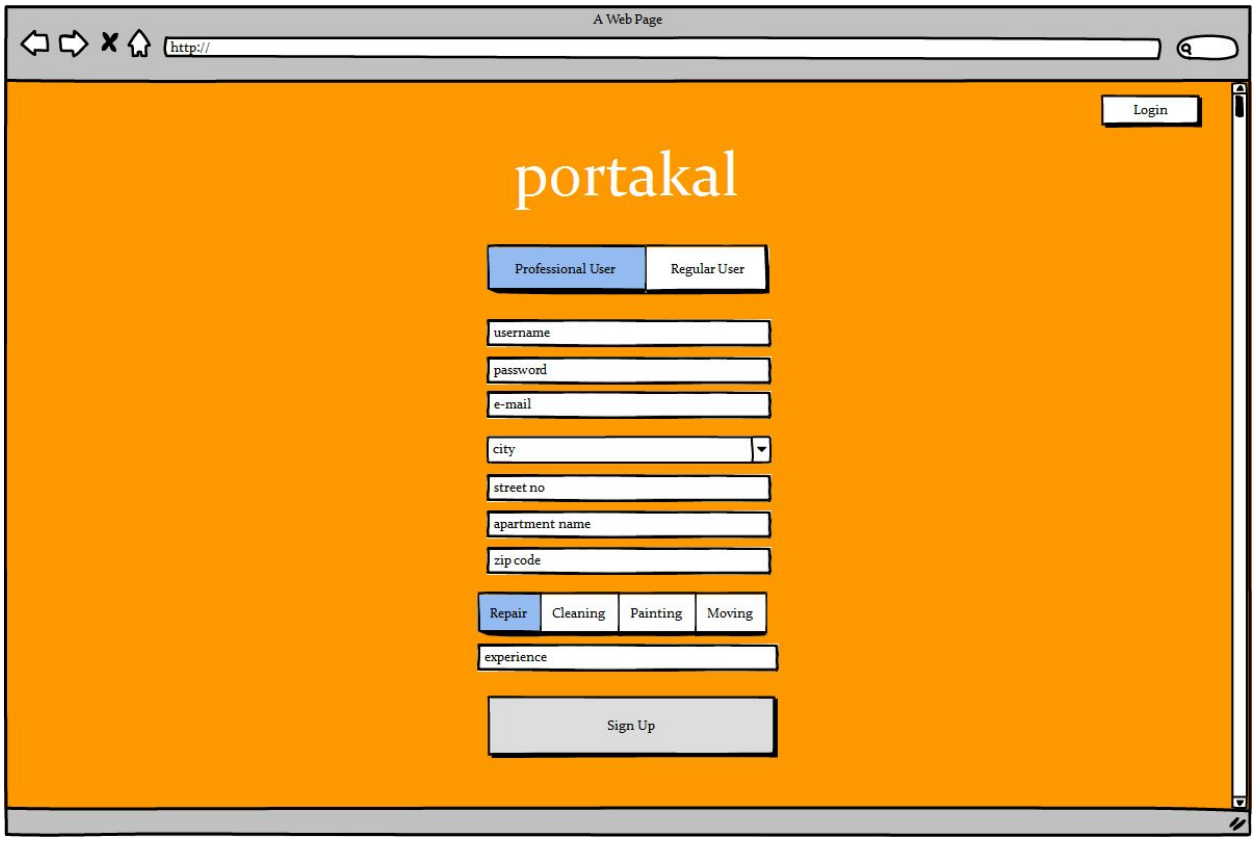
SQL Statements:

Register Service:

```
INSERT INTO Provides(user_ID, service_type_ID, custome_service_name)
VALUES(@user_ID, @service_type_ID, @custome_service_name);

UPDATE Provides
SET user_ID = @new_user_ID,
        service_type_ID = @new_service_type_ID,
        custom_service_name = @new_custom_service_name
FROM Professional User NATURAL JOIN Provides
WHERE service_type_ID IN (SELECT service_type_ID
                                FROM Services)
```

5.14 Sign Up for Professional User

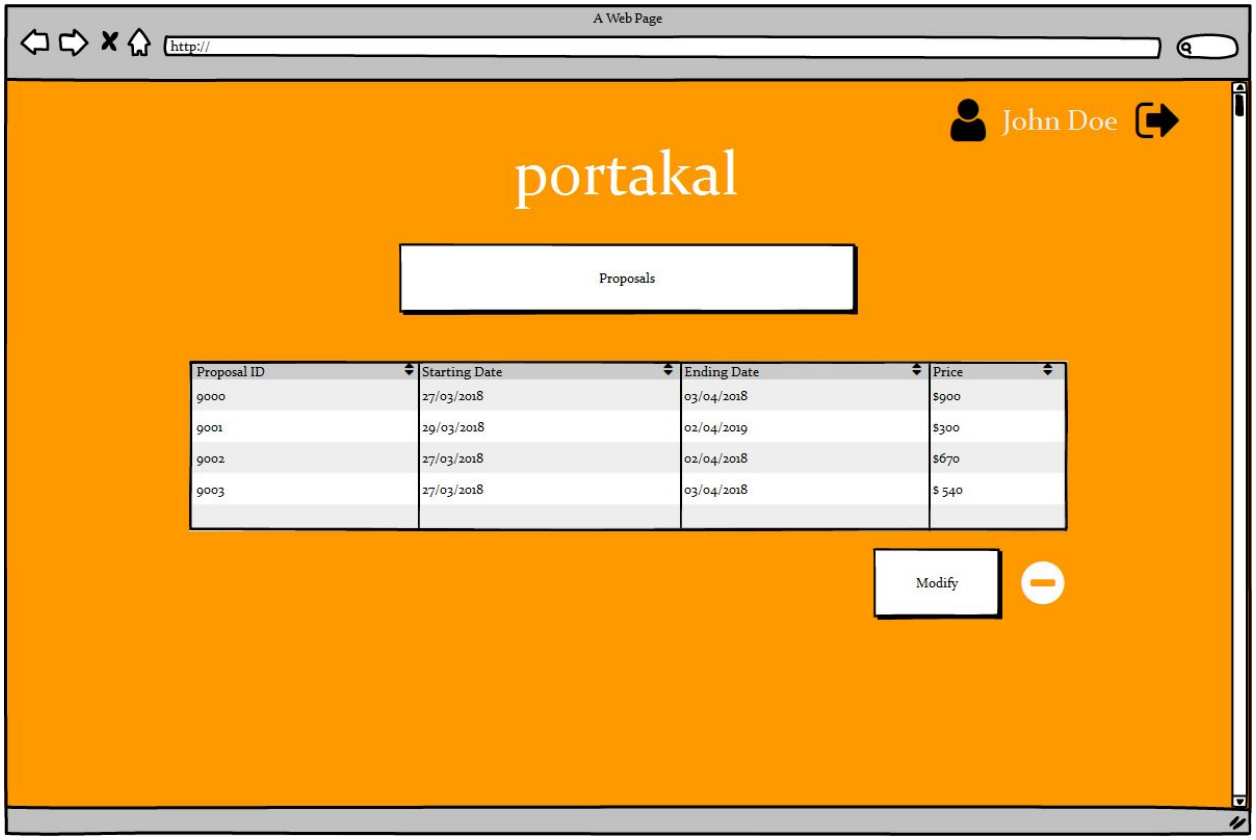


Input: @username, @password, @email,@expertise_field, @experience, @city @street_no @apartment_name @zip_code
Process: A professional user can access the sign up page through clicking “sign up” in either homepage, or in login page. In the Sign Up page, users who selects “Professional User” option will be greeted with the page above. In order to create a professional user account, they have to provide at least username, password and email information.

SQL Statements:

Sign Up:
INSERT INTO User(password, email, username, city_name, street_number, apt_name, zip_code)
VALUES(@password, @email, @username, @city_name, @street_number, @apt_name, @zip_code);
INSERT INTO Professional User(experience, expertise_field)
VALUES(@experience, @expertise_field);

5.15 View Proposals for Professional User



Input: @user_ID, @selected_proposal_ID
Process: Professional users are able to see proposal which they proposed in a list which is on the middle of the screen. User can select one from the list and click Modify Button to modify details of that proposal. Also User can cancel a proposal by the minus button.

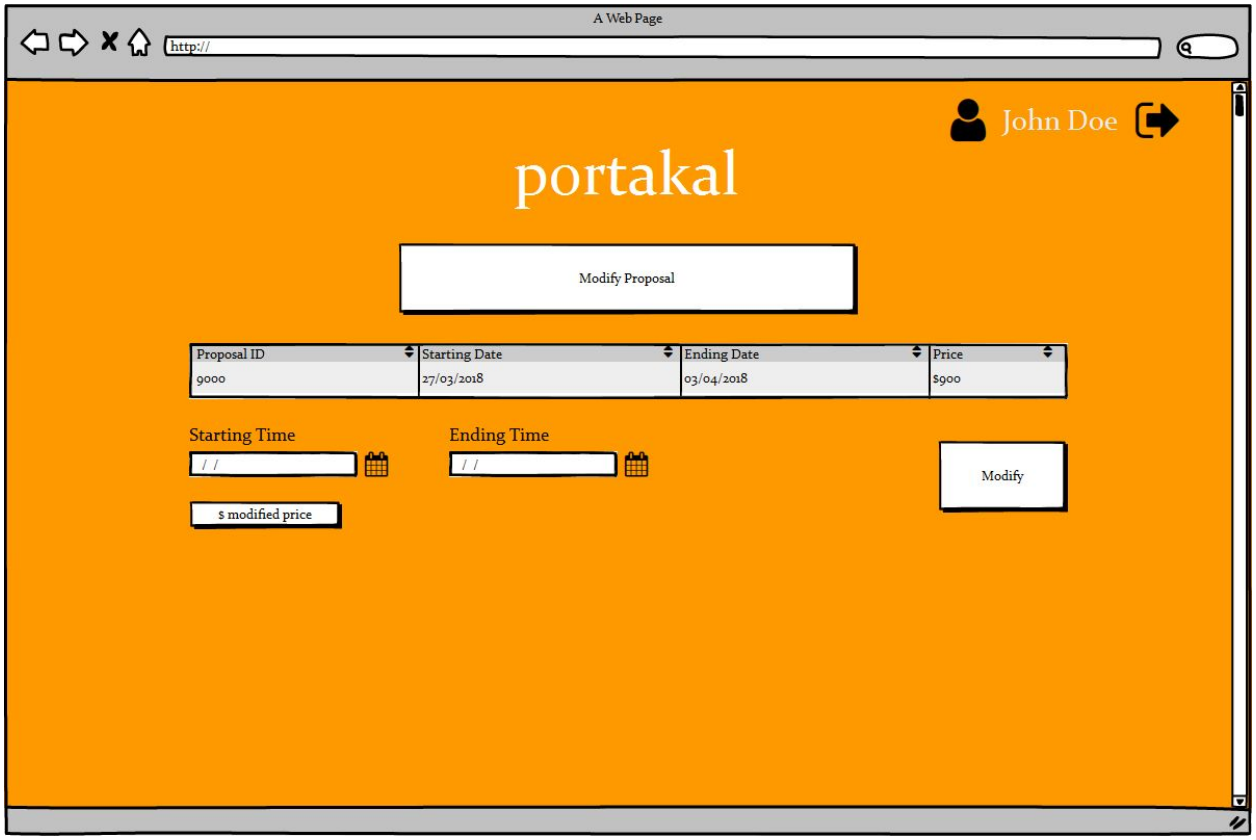
SQL Statements:

View Proposals:

```
SELECT *
FROM Proposed Services as S,
                    (SELECT proposal_ID
                    FROM Proposals as P
                    WHERE P.user_ID = @user_ID) as T
WHERE S.proposal_ID = T.proposal_ID;
```

Cancel Proposal:
DELETE FROM Proposed Services
WHERE proposal_ID= @selected_proposal_ID;

5.16 Modify Proposals for Professional User



Inputs: @selected_proposal_ID, @new_end_date, @new_start_date, @new_proposed_price
Process: Professional Users can modify the selected proposal from this screen. The start, end dates and proposed price informations of the proposal can be modified.
SQL Statements:

Modify Start Date of Proposal:
UPDATE Proposed Services AS P
SET P.start_date = @new_start_date
WHERE P.proposal_ID = @selected_proposal_ID

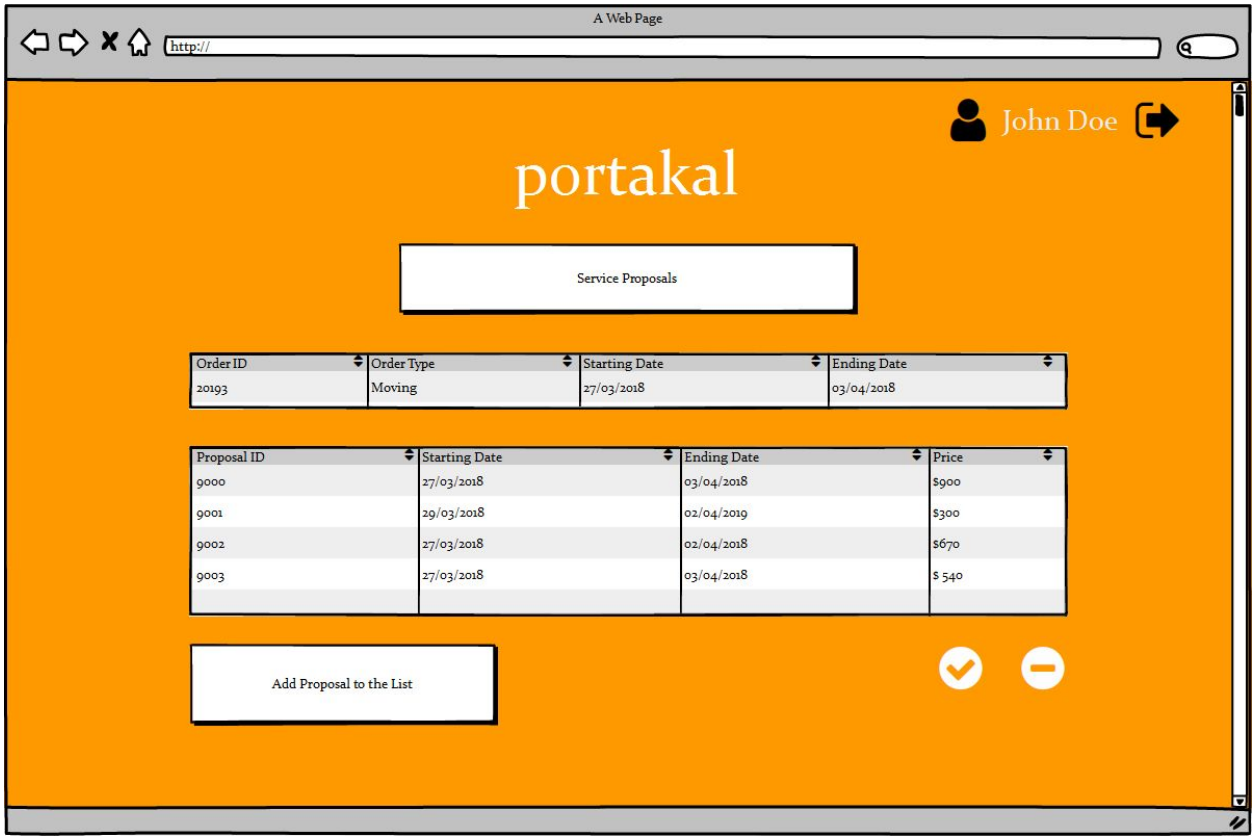
Modify End Date of Proposal:

```
UPDATE Proposed Services AS P
SET P.end_date =
        CASE
                WHEN DATEDIFF(@new_end_date, P.start_date) <= 0 THEN
        P.end_date
                ELSE @new_end_date
        END
WHERE (P.proposal_ID = @selected_proposal_ID)
```

Modify Proposed Price of Proposal:

```
UPDATE Proposed Services AS P
SET P.proposed_price =
        CASE
                WHEN @new_proposed_price >= 0 THEN @new_proposed_price
                ELSE P.proposed_price
        END
WHERE P.proposal_ID = @selected_proposal_ID
```

5.17 View Proposals for Regular User



Input: @order_ID
Process: Regular User can see the proposal
SQL Statements:

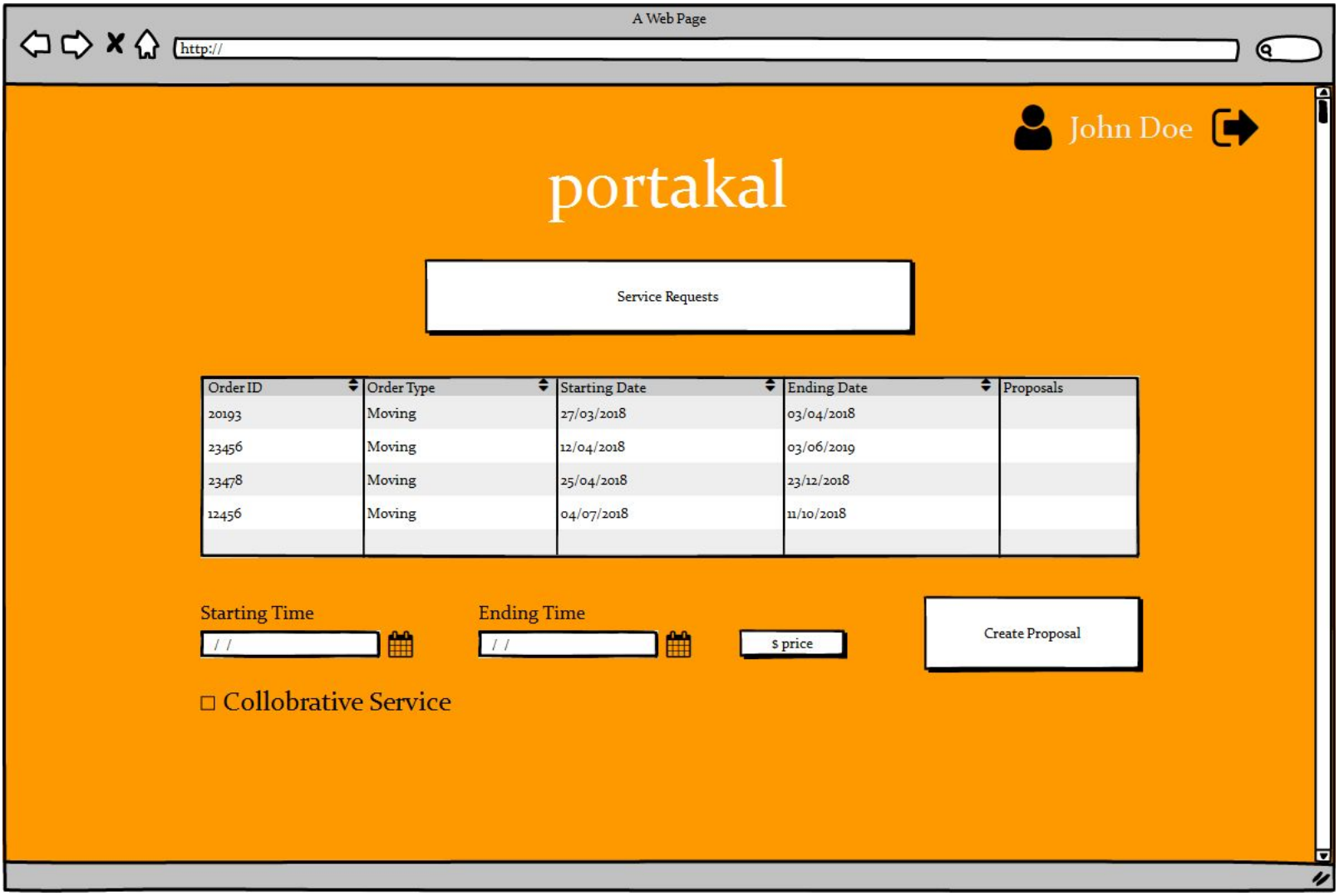
View Order Details

```
SELECT *
FROM Service Order
WHERE order_ID = @order_ID;
```

View Proposals

```
SELECT P.proposal_ID, P.start_date, P.end_date, P.proposed_price
FROM Proposed Services AS P, (SELECT proposal_ID
                              FROM Matches AS M
                              WHERE M.order_ID = @order_ID) AS T
WHERE P.proposal_ID = T.proposal_ID;
```

5.18 View Requests for Professional User



Input: @start_date, @end_date, @proposed_price, @selected_service_order_type
Process: Professional Users are able to see orders in the list. They can propose to these orders by giving details about date and price. Then, they will create the proposal by pressing the Create Proposal button.

SQL Statements:

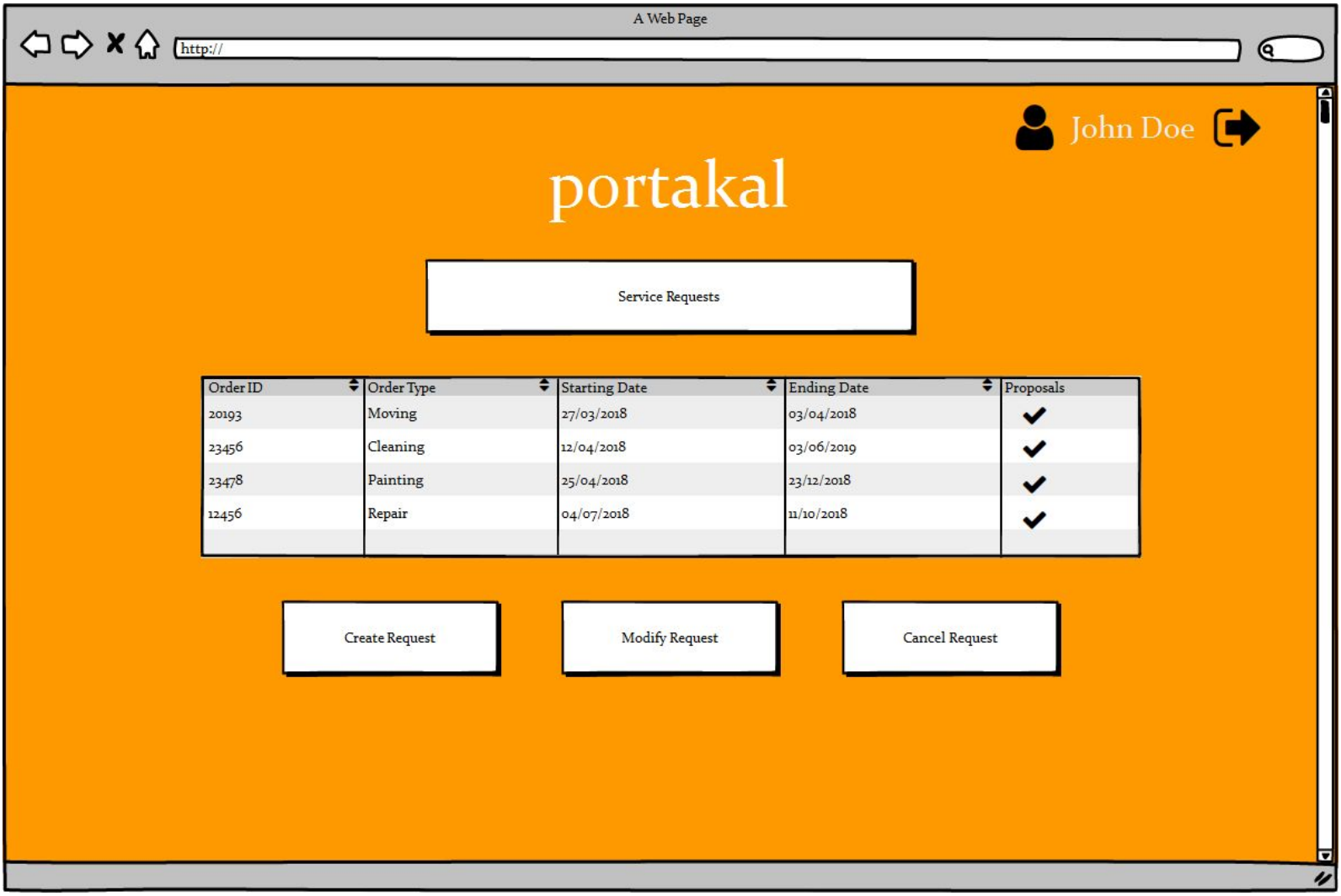
View Service Requests:

```
CREATE VIEW service-requests(order_ID, custom_service_name, start_date,
        end_date)
AS (SELECT order_ID, custom_service_name, start_date, end_date)
FROM (Service Order NATURAL JOIN Matches) NATURAL JOIN Proposals
WHERE user_ID = @user_ID
```

Create Proposal:

```
INSERT INTO Proposed Services(service_order_type, start_date, end_date,
proposed_price)
VALUES(@selected_service_order_type, @start_date, @end_date, @proposed_price);
```

5.19 View Service Orders of Regular User



Input: @user_ID, @selected_order_ID
Process: Create Request takes the user to the Create Service Request for Regular Users page. Orders of the current user are shown in the list. User can modify, cancel or create orders by using respective buttons.

SQL Statement:
View Service Requests

SELECT *
FROM Has as H NATURAL JOIN Service Order

```
WHERE H.user_ID = @user_ID
ORDER_BY H.order_ID;
```

Cancel Request

```
DELETE FROM Service Order
WHERE order_ID = @selected_order_ID;
```

Modify Request:

```
UPDATE Service Order
SET service_order_type = @new_service_order_type, order_details =
        @new_order_details,
WHERE order_ID = @selected_order_ID;
```

5.20 View Service Provider Information for Regular User

Input: @service_type_ID, @custom_service_name
Process: User is able to see the details of the professional user who provides the service.

Professional view:

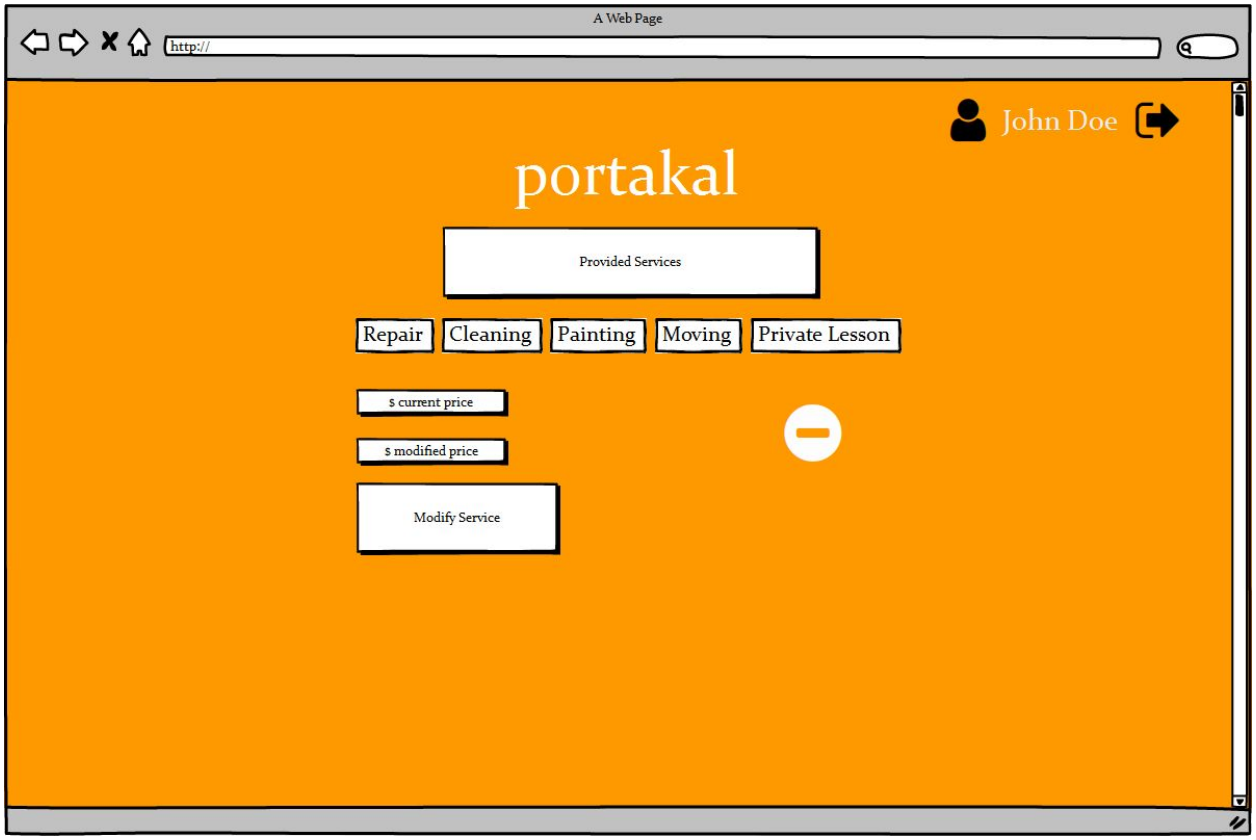
```
CREATE VIEW professional-view(user_ID,  city_name, street_number, apt_name,
zip_code, professional_rating, expertise)
AS SELECT user_ID,  city_name, street_number, apt_name, zip_code,
professional_cumulative_rating, expertise_field
FROM (Professional User NATURAL JOIN User);
```

This view element will be used for the following:

View Service Provider Information:

```
SELECT P.*
FROM professional-view,
        (SELECT user_ID
        FROM Provides AS T
        WHERE (T.service_type_ID = @service_type_ID)
                AND
                (T.custom_service_name = @custom_service_name)
        ) AS C
WHERE P.user_ID = C.user_ID;
```

5.21 View Services for Professional User



Inputs: @service_type, @modified_price, @user_ID, @new_base_material_price, @new_base_room_price, @new_base_bathroom_price, @new_hourly_rate
Process: Professional users can access this screen by clicking the view services button from the log-on screen. In this screen, professionals can edit the three services that has a base price. These services are Repair Service, Cleaning Service and Private Lesson Service. If professionals do provide any of these services, then they can edit the base prices. For Repair Service, professionals edits the base material price according to the item type of the material.

SQL Statements

View Current Price For Repair:

```
SELECT R.base_material_price,
FROM Repair Service AS R, (SELECT service_type_ID, custom_service_name
                           FROM Provides AS P
                           WHERE P.user_ID = @user_ID) AS T
WHERE (R.service_type_ID = T.service_type_ID)
```

```
        AND
        (R.custom_service_name = T.custom_service_name);
```

Modify Service Price for Repair Service:

```
WITH services_of_user (service_type_ID, custom_service_name) AS
        ( SELECT service_type_ID, custom_service_name
        FROM Provides AS P
        WHERE P.user_ID = @user_ID)
UPDATE Repair Service AS R
SET R.base_material_price =
      CASE
            WHEN @new_base_material_price > 0 then @new_base_material_price
            ELSE R.base_material_price
      END
WHERE (R.service_type_ID IS IN
            (SELECT service_type_ID
            FROM services_of_user)
        )
        AND
        (R.custom_service_name IS IN
            (SELECT custom_service_name
            FROM services_of_user)
        );
```

View Current Price For Cleaning Service:

```
SELECT C.base_room_price, C.base_bathroom_price
FROM Cleaning Service AS C, (SELECT service_type_ID, custom_service_name
                                FROM Provides AS P
                                WHERE P.user_ID = @user_ID) AS T
WHERE C.service_type_ID = T.service_type_ID AND
        C.custom_service_name = T.custom_service_name;
```

Modify Base Room Price for Cleaning Service:

```
WITH services_of_user (service_type_ID, custom_service_name) AS
        ( SELECT service_type_ID, custom_service_name
        FROM Provides AS P
        WHERE P.user_ID = @user_ID)
UPDATE Cleaning Service AS C
SET C.base_room_price =
      CASE
            WHEN  @new_base_room_price > 0 THEN @new_base_room_price
            ELSE C.base_room_price
      END
WHERE (C.service_type_ID IS IN
            (SELECT service_type_ID
            FROM services_of_user)
        )
        AND
       (C.custom_service_name IS IN
            (SELECT custom_service_name
            FROM services_of_user)
       );
```

Modify Base Bathroom Room Price for Cleaning Service:

```
WITH services_of_user (service_type_ID, custom_service_name) AS
        ( SELECT service_type_ID, custom_service_name
        FROM Provides AS P
        WHERE P.user_ID = @user_ID)
UPDATE Cleaning Service AS C
SET C.base_bathroom_price =
      CASE
            WHEN  @new_base_bathroom_price > 0 THEN
      @new_base_bathroom_price
            ELSE C.base_bathroom_price
      END
WHERE (C.service_type_ID IS IN
            (SELECT service_type_ID
            FROM services_of_user)
        )
        AND
       (C.custom_service_name IS IN
```

```
            (SELECT custom_service_name
            FROM services_of_user)
        );
```

View Current Price for Private Lesson

```
SELECT C.hourly_rate,
FROM Private Lesson AS C, (SELECT service_type_ID, custom_service_name
                           FROM Provides AS P
                           WHERE P.user_ID = @user_ID) AS T
WHERE C.service_type_ID = T.service_type_ID AND
        C.custom_service_name = T.custom_service_name;
```

Modify Current Price for Private Lesson

```
WITH services_of_user (service_type_ID, custom_service_name) AS
        ( SELECT service_type_ID, custom_service_name
        FROM Provides AS P
        WHERE P.user_ID = @user_ID)
UPDATE Private Lesson AS T
SET T.hourly_rate =
      CASE
            WHEN  @new_hourly_rate > 0 THEN @new_hourly_rate
            ELSE T.hourly_rate
      END
WHERE (T.service_type_ID IS IN
            (SELECT service_type_ID
            FROM services_of_user)
        )
        AND
      (T.custom_service_name IS IN
            (SELECT custom_service_name
            FROM services_of_user)
      );
```

6. Advanced Database Components

6.1 Views

Service_view: This view is used for homepage screen to list a list of professionals and their information who are working in the selected service type.

```
CREATE VIEW service_view(service_name, professional_ID, professional_rating,
professional_experience)
AS SELECT custom_service_name, user_ID, professional_cumulative_rating,
experience
FROM Professional User NATURAL JOIN
                (SELECT *
                FROM Provides
                WHERE custom_service_name = @service_name)
ORDER BY professional_cumulative_rating DESC;
```

Provided_services: People will be able to see Provided services by the registered Professional users and ratings of these services. This information will be public.

```
CREATE VIEW provided_services ( username, custom_service_name, service_rating )
AS SELECT username, custom_service_name, service_rating,
FROM Professional User NATURAL JOIN Provided Services NATURAL JOIN Provides,
ORDER BY custom_service_name DESC,
LIMIT 20;
```

Professional_view: Regular users will be able to see accessible information about the Professional Users. Regular users should not be able to access account information like password, username and email address.

```
CREATE VIEW professional_view(user_ID,  city_name, street_number, apt_name,
zip_code, professional_rating, expertise)
AS SELECT user_ID,  city_name, street_number, apt_name, zip_code,
professional_cumulative_rating, expertise_field
FROM (Professional User NATURAL JOIN User);
```

6.2 Triggers

- When a proposal is created, deleted; corresponding entries should also need to be added to or removed from Proposed and Matches relations. Similar operations are needed for Collaborative Services which also requires updates on the Collaborators relation.
- When a Service is removed, all tuples which include that one should be removed from the Related Services relation.
- When a Service Order is deleted or updated all related Proposal should be removed in order to prevent wrong or undesired matches.

6.3 Constraints

- End dates should not be before start dates and new date entries should not be before current date where dates are used.
- Order details are determined according to the service type and user inputs for this attribute are limited by the system.

6.4 Stored Procedures

Stored procedures are planned to be used for initializing some of the attributes of Services table. Service types should be know at the time of creation.

6.5 Reports

6.5.1 Monthly Proposals Report

Professionals will be able to see their proposals starting this month. This view will be shown in a list similar to other proposal lists shown in different screens mentioned before.

```
CREATE VIEW monthly_prop_report
AS (SELECT proposal_ID, start_date, end_date, proposed_price, service_type_ID
        FROM (Professional User NATURAL JOIN Proposals) NATURAL JOIN
                Proposed Services
        WHERE user_ID = @user_ID AND DATEDIFF(CURDATE(), start_date) <= 30);
```

7. Implementation Plan

We plan to use PHP, HTML, CSS and JavaScript for user interface and functionalities of our system. We plan to use MySQL on the server for the database of our system.

8. Website

Website information can be found in the following page.

https://github.com/BecerZ/hiring_system