CS353 - DATABASE SYSTEMS

PROJECT DESIGN REPORT

29.11.2021

GROUP NO: 5

Aybala Karakaya - 21801630

Halil Özgür Demir - 21801761

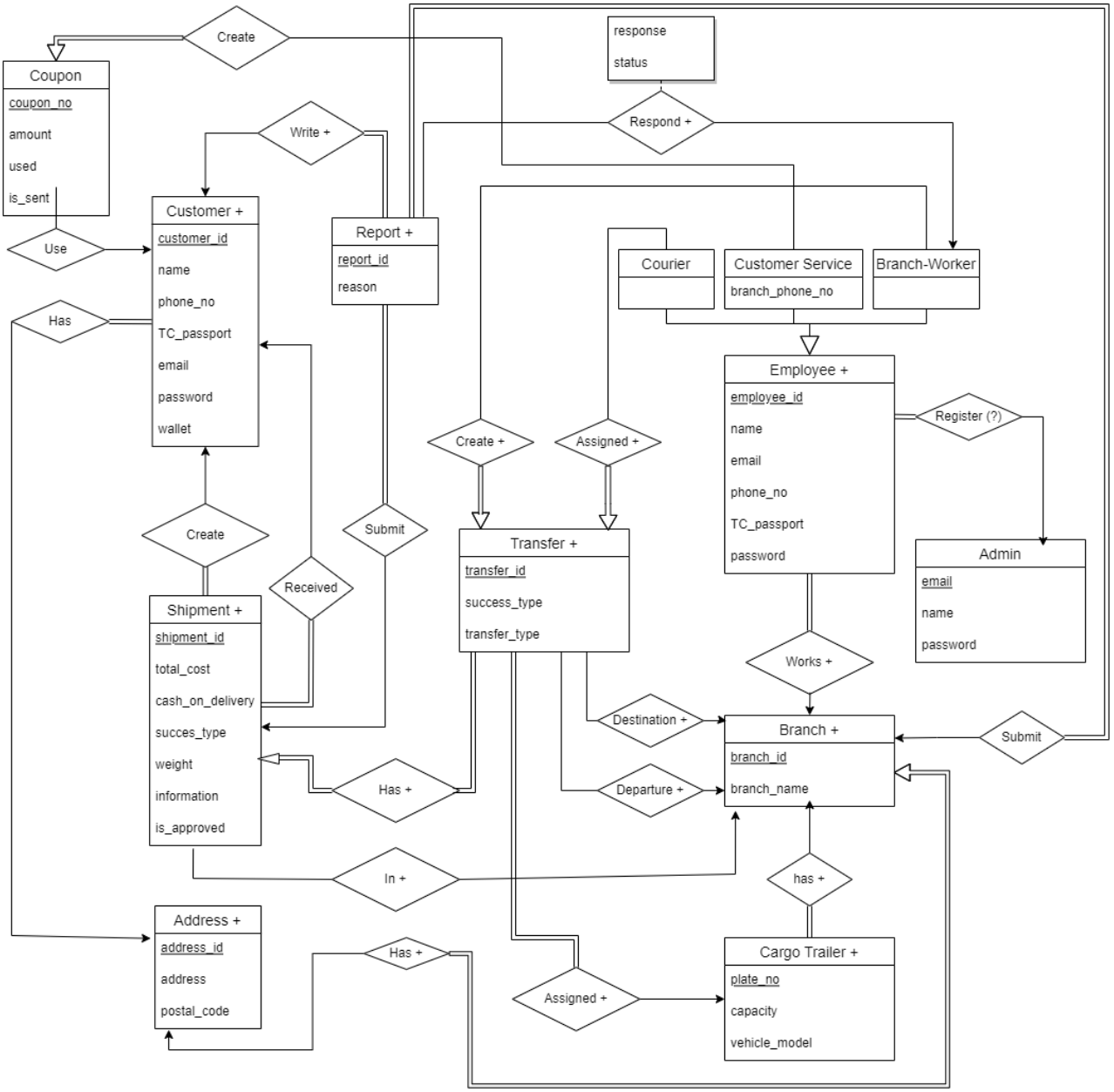
Mustafa Çağrı Durgut - 21801983

Oğuzhan Özçelik - 21802194

Instructor: Özgür Ulusoy

TA: Mustafa Can Çavdar

1) Revised E/R Diagram



2) Database Relations

2.1) Customer

Relational Model

- Customer(<u>customer_id</u>, name, phone_no, TC_passport, email, password, wallet)

Functional Dependencies

- customer_id → rest of them
- TC_passport → rest of them
- email → rest of them

Candidate Keys

- {(customer_id),(TC_passport),(email)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Customer(customer_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, name VARCHAR(64) NOT NULL, phone_no VARCHAR(32) NOT NULL, TC_passport VARCHAR(32) NOT NULL UNIQUE, email VARCHAR(128) NOT NULL UNIQUE, password VARCHAR(32) NOT NULL, wallet INT NOT NULL) ENGINE=INNODB;

2.2) Admin

Relational Model

- Admin(<u>email</u>, name, password)

Functional Dependencies

- email→ rest of them

Candidate Keys

- {(email)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Admin(email VARCHAR(128) NOT NULL UNIQUE PRIMARY KEY, name VARCHAR(64) NOT NULL, password VARCHAR(32) NOT NULL) ENGINE=INNODB;

2.3) Shipment

Relational Model

- Shipment(<u>shipment_id</u>, sender_id, total_cost, cash_on_delivery, success_type, weight, information, receiver_id, is_approved, branch_id)

Functional Dependencies

- shipment_id→ rest of them

Candidate Keys

- {(shipment_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Shipment(shipment_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, sender_id INT NOT NULL, total_cost FLOAT NOT NULL, cash_on_delivery BIT NOT NULL, success_type BIT NOT NULL, weight FLOAT NOT NULL,information TEXT NOT NULL, receiver_id INT NOT NULL, is_approved BIT NOT NULL, branch_id INT, FOREIGN KEY(sender_id) REFERENCES Customer(customer_id), FOREIGN KEY(receiver_id) REFERENCES Customer(customer_id), FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)) ENGINE=INNODB;

2.4) Report

Relational Model

- Report(<u>report_id</u>, reason, reporter_id, shipment_id, respondent_id, response, status, branch_id)

Functional Dependencies

- report_id→ rest of them

Candidate Keys

- {(report_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Report(report_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, reason TEXT NOT NULL, reporter_id INT NOT NULL, shipment_id INT NOT NULL, respondent_id INT, response TEXT, status BIT, branch_id INT NOT NULL, FOREIGN KEY(reporter_id) REFERENCES Customer(customer_id) FOREIGN KEY(shipment_id) REFERENCES Shipment(shipment_id), FOREIGN KEY(respondant_id) REFERENCES Branch_Worker(employee_id), FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)) ENGINE=INNODB;

2.5) Transfer

Relational Model

- Transfer(<u>transfer_id</u>, success_type, transfer_type, shipment_id, trailer_plate, assigned_employee_id, creator_id, departure_id, destination_id)

Functional Dependencies

- transfer_id→ rest of them

Candidate Keys

- {(transfer_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Transfer(transfer_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, success_type BIT, transfer_type BIT NOT NULL, shipment_id INT NOT NULL, trailer_plate VARCHAR(32) NOT NULL, assigned_employee_id INT NOT NULL, creator_id INT NOT NULL, departure_id INT, destination_id INT, FOREIGN KEY(shipment_id) REFERENCES Shipment(shipment_id), FOREIGN KEY(trailer_plate) REFERENCES Trailer(plate_no), FOREIGN KEY(assigned_employee_id) REFERENCES Employee(employee_id), FOREIGN KEY(creator_id) REFERENCES Employee(employee_id), FOREIGN KEY(departure_id) REFERENCES Branch(branch_id),

FOREIGN KEY(destination_id) REFERENCES Branch(branch_id)) ENGINE=INNODB;

2.6) Cargo trailer

Relational Model

- Trailer(<u>plate_no</u>, capacity, vehicle_model, branch_id)

Functional Dependencies

- plate_no→ rest of them

Candidate Keys

- {(plate_no)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Trailer(plate_no VARCHAR(32) NOT NULL UNIQUE PRIMARY KEY, capacity FLOAT NOT NULL, vehicle_model VARCHAR(32), branch_id INT NOT NULL, FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)) ENGINE=INNODB;

2.7) Branch

Relational Model

- Branch(<u>branch_id</u>, branch_name, address_id)

Functional Dependencies

- branch_id→ rest of them

Candidate Keys

- {(branch_id, address_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Branch(branch_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, branch_name VARCHAR(64) NOT NULL, address_id INT NOT NULL, FOREIGN KEY(address_id) REFERENCES Address(address_id)) ENGINE=INNODB;

2.8) Address

Relational Model

- Address(<u>address_id</u>, address, postal_code)

Functional Dependencies

- address_id→ rest of them

Candidate Keys

- {(address_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Address(address_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, address TEXT NOT NULL, postal_code VARCHAR(32) NOT NULL) ENGINE=INNODB;

2.9) Employee

Relational Model

- Employee(<u>employee_id</u>, name, phone_no, TC_passport, email, password, branch_id, adder_admin)

Functional Dependencies

- employee_id → rest of them
- TC_passport → rest of them
- email → rest of them

Candidate Keys

- {(employee_id),(TC_passport),(email)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Employee(employee_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, name VARCHAR(64) NOT NULL, phone_no VARCHAR(32) NOT NULL, TC_passport VARCHAR(32) NOT NULL UNIQUE, email VARCHAR(128) NOT NULL UNIQUE, password VARCHAR(32) NOT NULL, branch_id INT NOT NULL, adder_admin VARCHAR(128) NOT NULL, FOREIGN KEY(branch_id) REFERENCES

Branch(branch_id), FOREIGN KEY(adder_admin) REFERENCES Admin(email)) ENGINE=INNODB;

2.10) Courier

Relational Model

- Courier(<u>employee_id</u>)

Functional Dependencies

- none

Candidate Keys

- {(employee_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Courier(employee_id INT NOT NULL PRIMARY KEY, FOREIGN KEY(employee_id) REFERENCES Employee(employee_id)) ENGINE=INNODB;

2.11) Customer Service

Relational Model

- Customer_Service(<u>employee_id</u>, branch_phone_no)

Functional Dependencies

- employee_id→ rest of them

Candidate Keys

- {(employee_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Customer_service(employee_id INT NOT NULL PRIMARY KEY, branch_phone_no VARCHAR(32) NOT NULL, FOREIGN KEY(employee_id) REFERENCES Employee(employee_id)) ENGINE=INNODB;

2.12) Branch Worker

Relational Model

- Branch_worker(employee_id)

Functional Dependencies

- None

Candidate Keys

- {(employee_id)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Branch_worker(employee_id INT NOT NULL PRIMARY KEY, FOREIGN KEY(employee_id) REFERENCES Employee(employee_id)) ENGINE=INNODB;

2.13) Coupon

Relational Model

- Coupon(coupon_no, amount, used, creator_id, owner_id, is_sent)

Functional Dependencies

- coupon_no→ rest of them

Candidate Keys

- {(coupon_no)}

Normal Forms

- 3NF

Table Definition

- CREATE TABLE Coupon(coupon_no VARCHAR(16) NOT NULL PRIMARY KEY, amount INT NOT NULL, used BIT NOT NULL, creator_id INT NOT NULL, owner_id INT, is_sent BIT NOT NULL, FOREIGN KEY(creator_id) REFERENCES Employee(employee_id), FOREIGN KEY(owner_id) REFERENCES Customer(customer_id)) ENGINE=INNODB;

3) GUI Design and Corresponding SQL Statements

3.1) Login Page

SHIPMIN

Login

Email

Password

LOGIN

Forgot My Password

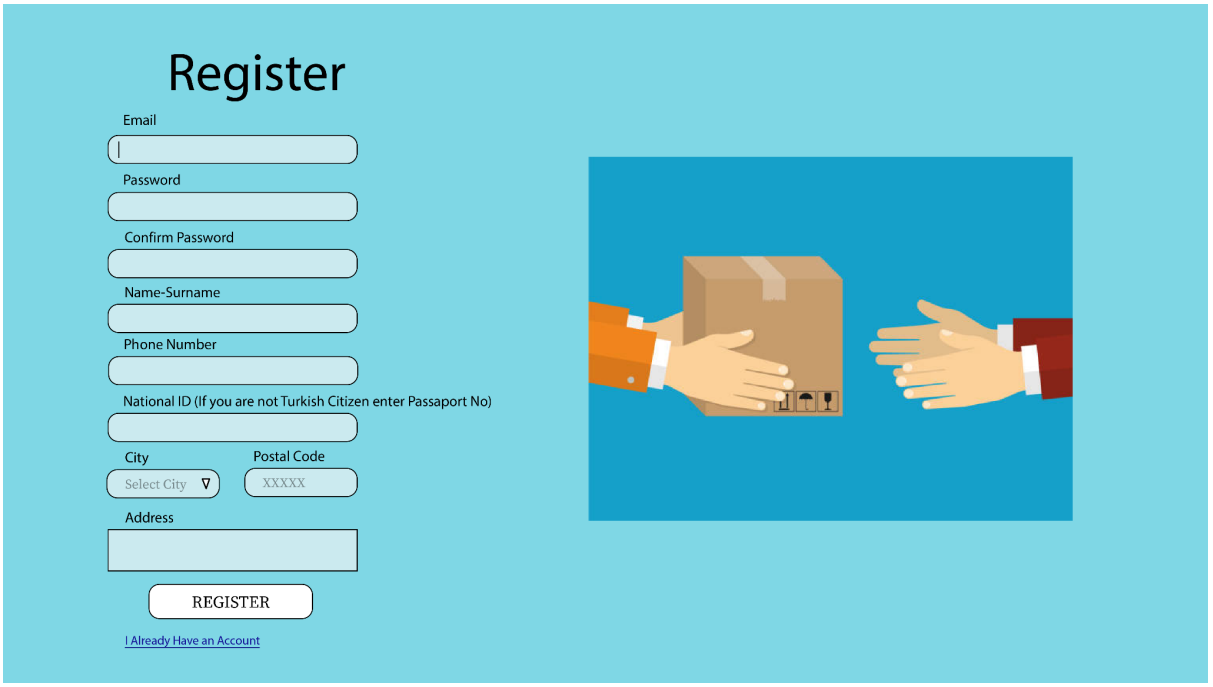
Create new account

Users will enter their email and password in order to login to the system. This page above will prompt them and they will use these two input fields and the button to send the below SQL query to the database:

SELECT * FROM Customer WHERE (email = @email, password = hash(@password));

If the query above returns not null, User will proceed to the system.

3.2) Register Page



If the users do not have an account, they can use the page above in order to register to the system. This page takes an email, confirmed password, name, phone number, National ID, City, Postal Code and their address. If all of the required fields are full the server will run the following SQL query.

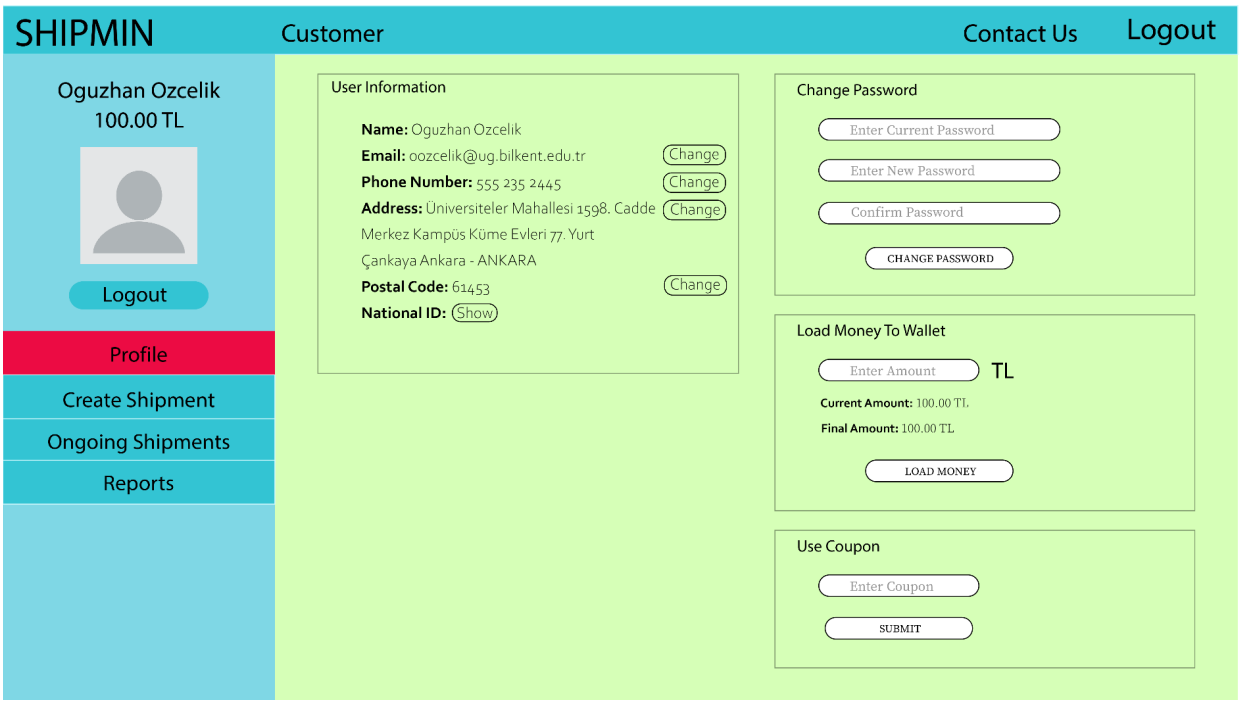
```
SELECT * FROM Customer WHERE (email = @email);
```

If the SQL query above returns not null, The system prompts the user that this email is already registered to the system.

Else, the server will run the following SQL query.

```
INSERT INTO Customer (name, phone_no, TC_passport, email, password)
VALUES (@name, @phone_no, @TC_passport, @email, hash(@password));
```

3.3) Customer Profile Page



This page is the Customer Profile Page, with the help of this page users will be able to edit and see their information. They will be able to change their password and top up money to their wallet and switch to other pages.

The button on the top right will allow users to log off from the system. When users press this button a request to the server that the session is wanted to be ended will be sent. No SQL queries necessary for this function.

Users will be able to switch pages in the application using the menu on the left. Again, no SQL queries needed for this function.

In order to show the relevant information to the user, we need to make a Select query to the database. We will store the customer_id of the user in the session, so that we can make queries efficiently.

```
SELECT * FROM Customer WHERE ( customer_id = @customer_id )
```

We will fetch the information related to this specific user via the SQL query above.

Then, when users wanted to change their information they will make the following SQL query to the database:

```
UPDATE Customer SET ( name = @name, address = @address, ...)
        WHERE (customer_id = @customer_id);
```

Another update is password. Users will be able to update their passwords with the SQL query below. Note that additional checks such as checking the old passwords must be made on the server.

```
UPDATE Customer SET ( password = hash(@new_password))
        WHERE (customer_id = @customer_id);
```

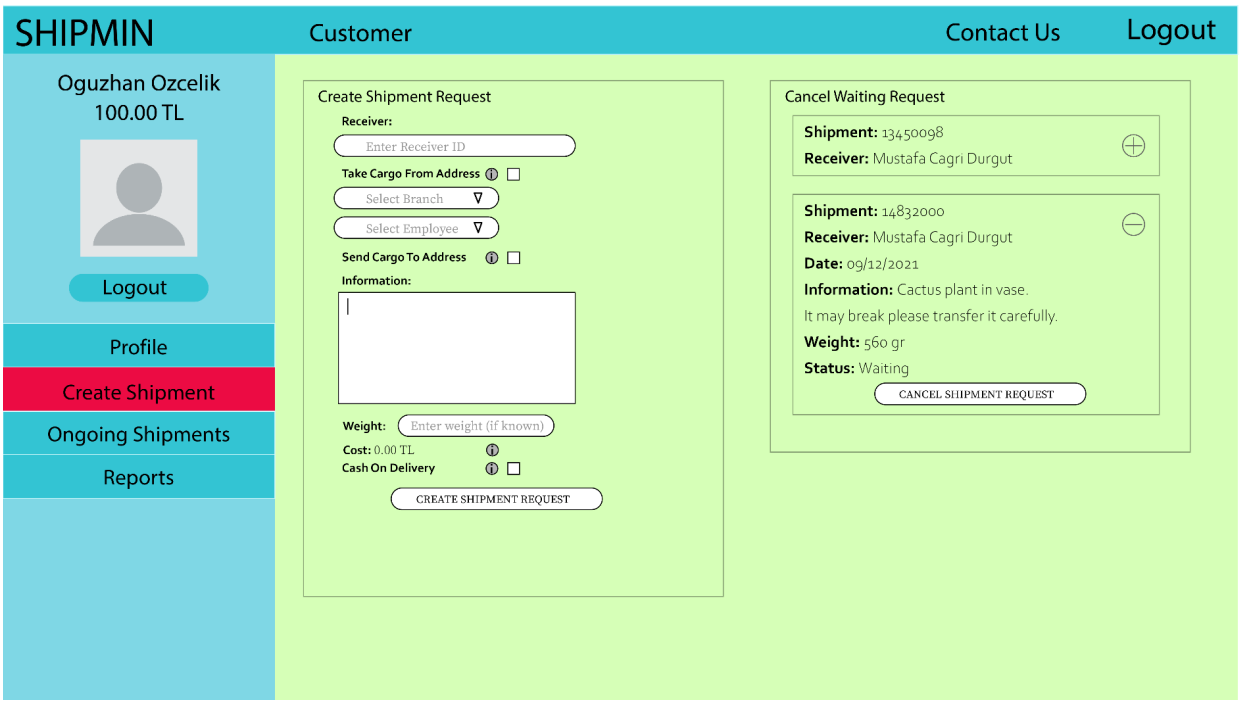
Users will be able to top up money to their wallet. The query below will be sent to the database then.

```
UPDATE Customer SET ( wallet = wallet + @extra_money))
        WHERE (customer_id = @customer_id);
```

Users can also choose to use a discount coupon that was assigned to them. Note that the coupon no would have been sent to the email address of the user. To use a coupon, the users should enter the coupon no from the bottom-right of the page and press the “submit” button. Afterwards, the following queries will be sent to the database and it will be as if some amount of money would be added to the wallet of the customer. Note that the server side should check whether the coupon has already been used before sending the query to the database.

```
SELECT * FROM Coupon WHERE ( coupon_no= @coupon_no )

UPDATE Customer SET ( wallet = wallet + @amount))
        WHERE (customer_id = @owner_id);

UPDATE Coupon SET ( used = 1 )
        WHERE (coupon_no = @coupon_no );
```

3.4) Create Shipment Page



This page is Create Shipment Page in which customers can create new shipment requests and also cancel waiting shipment requests. When a new shipment is created, some shipment id is assigned to that shipment by the system.

```
INSERT INTO shipment(sender_id, cost, cash_on_delivery, success_type,
weight, information, receiver_id, is_approved)
VALUES (@customer_id, @cost, @cash_on_delivery, 0, @weight,
@information, @receiver_id, 0)
```

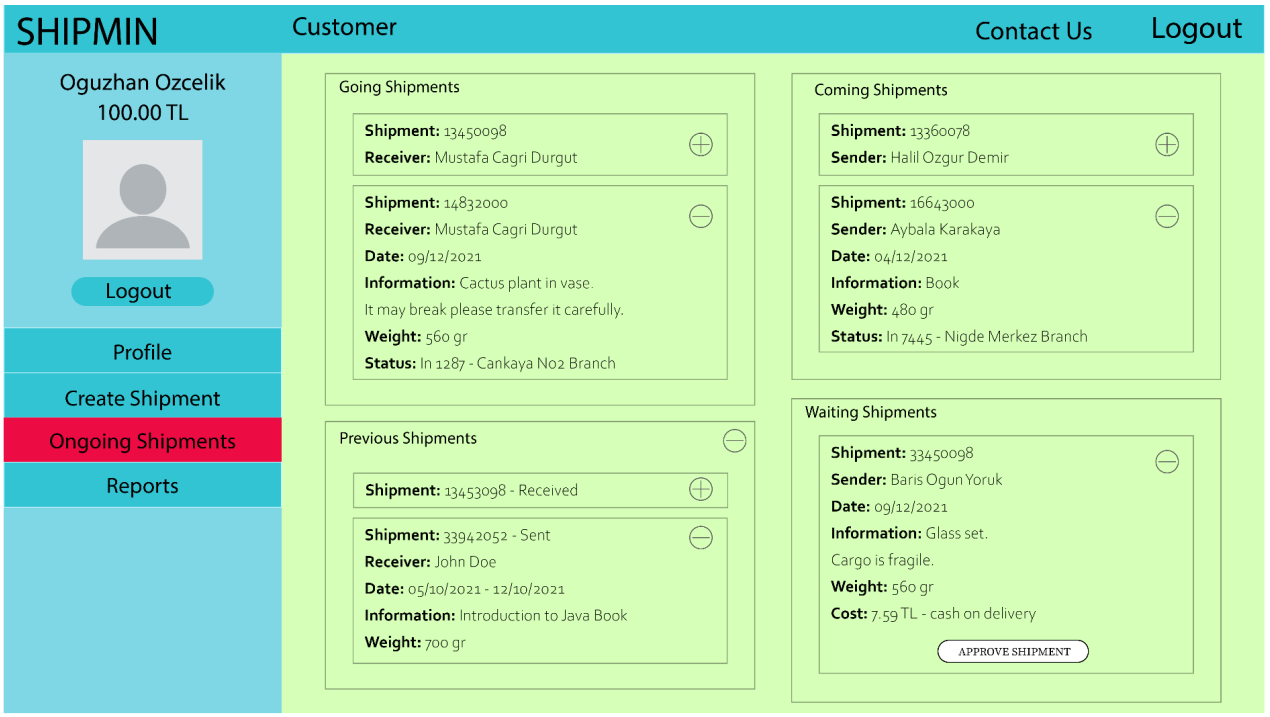
In order to display waiting requests, we select and display non approved shipments.

```
SELECT * FROM shipment
    WHERE sender_id = @customer_id AND is_approved = 0;
```

Customer can also cancel non approved shipments by using “CANCEL SHIPMENT REQUEST” button,

```
DELETE FROM shipment WHERE shipment_id = @shipment_id;
```

3.5) Ongoing Shipments Page



This page is Ongoing Shipments Page which display coming, going, previous and waiting shipments.

```
SELECT * FROM shipment, customer
        WHERE sender_id = @customer_id AND receiver_id = customer_id
        AND success_type = 0 AND is_approved = 1;
```

This query is used for the retrieve information of going shipments.

```
SELECT * FROM shipment, customer
        WHERE receiver_id = @customer_id AND sender_id = customer_id
        AND success_type = 0 AND is_approved = 1;
```

This query is used for retrieve information of coming shipments.

```
SELECT * FROM shipment, customer
        WHERE (receiver_id = @customer_id OR sender_id = @customer_id)
        AND (receiver_id = customer_id OR sender_id = customer_id) AND
        success_type = 1 AND is_approved = 1;
```

This query is used for retrieving previous shipments.

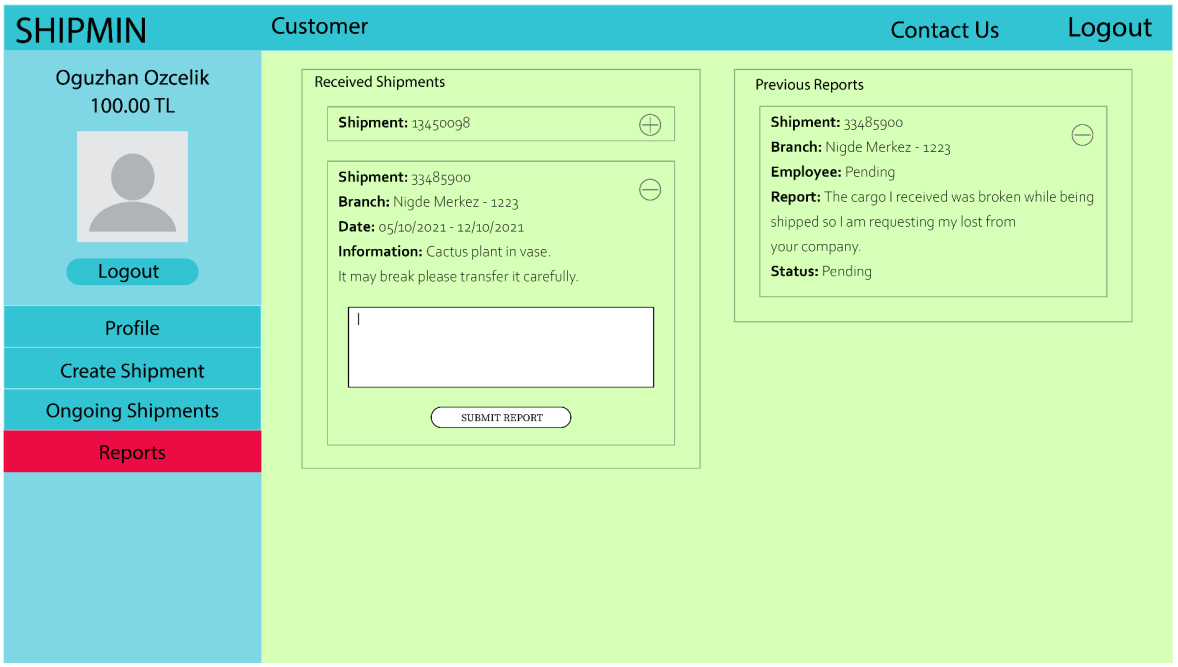
```
SELECT * FROM shipment, customer
```

```
WHERE receiver_id = @customer_id AND sender_id = customer_id
AND is_approved = 0;
```

This query is used for retrieving unapproved shipments. Customers can approve these requests by using the “APPROVE SHIPMENT” button.

```
UPDATE shipment SET ( is_approved = 1 )
    WHERE shipment_id = @shipment_id;
```

3.6) Customer Reports Page



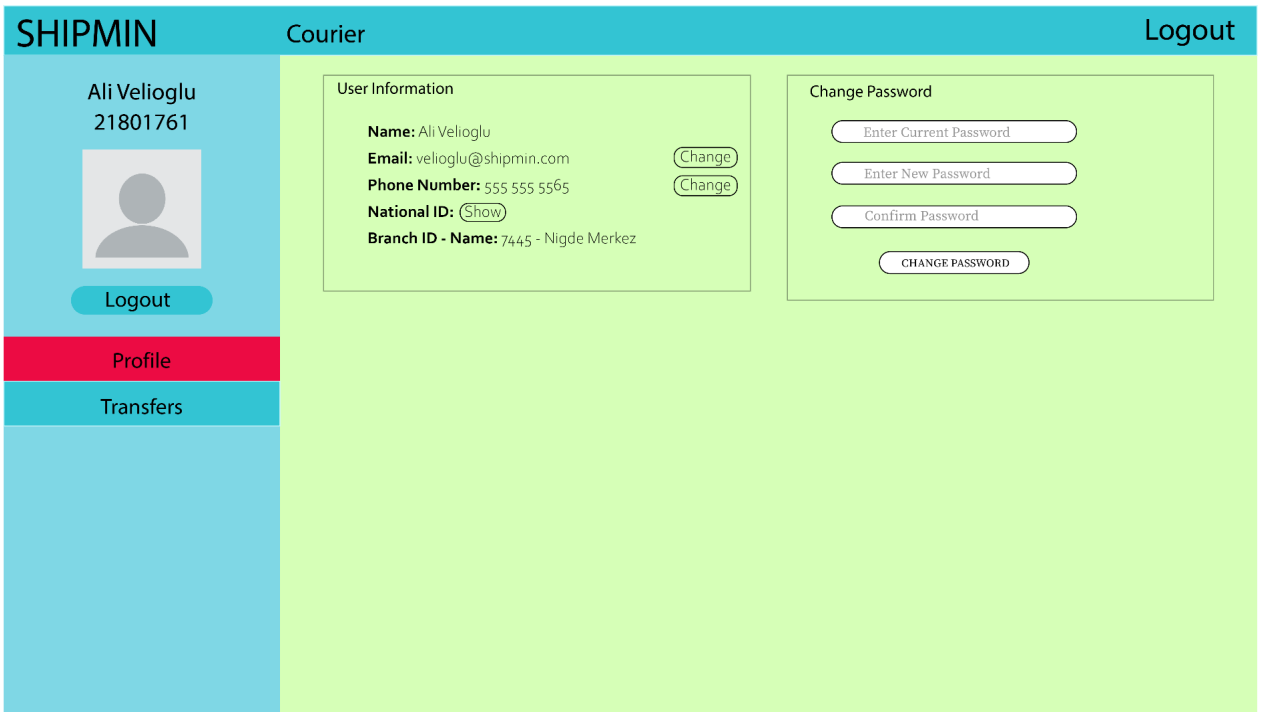
This page is Customer Reports Page which is used for creating reports for received shipments and tracking the previous reports. Customer can create new report by filling the space under the shipment and pressing “SUBMIT REPORT” button.

```
INSERT INTO report(reason, reporter_id, shipment_id, branch_id)
VALUES (@reason, @customer_id, @shipment_id, @coming_branch)
```

Also in order to track report status, system uses customer’s id.

```
SELECT * FROM report
WHERE reporter_id = @customer_id;
```

3.7) Courier Profile Page



This page is the Courier Profile Page, with the help of this page couriers will be able to edit and see their information. They will be able to change their password and switch to other pages.

In order to show the relevant information to the user, we need to make a Select query to the database. We will store the employee_id of the user in the session, so that we can make queries efficiently.

```
SELECT * FROM Employee WHERE ( employee_id = @employee_id )
```

We will fetch the information related to this specific user via the SQL query above.

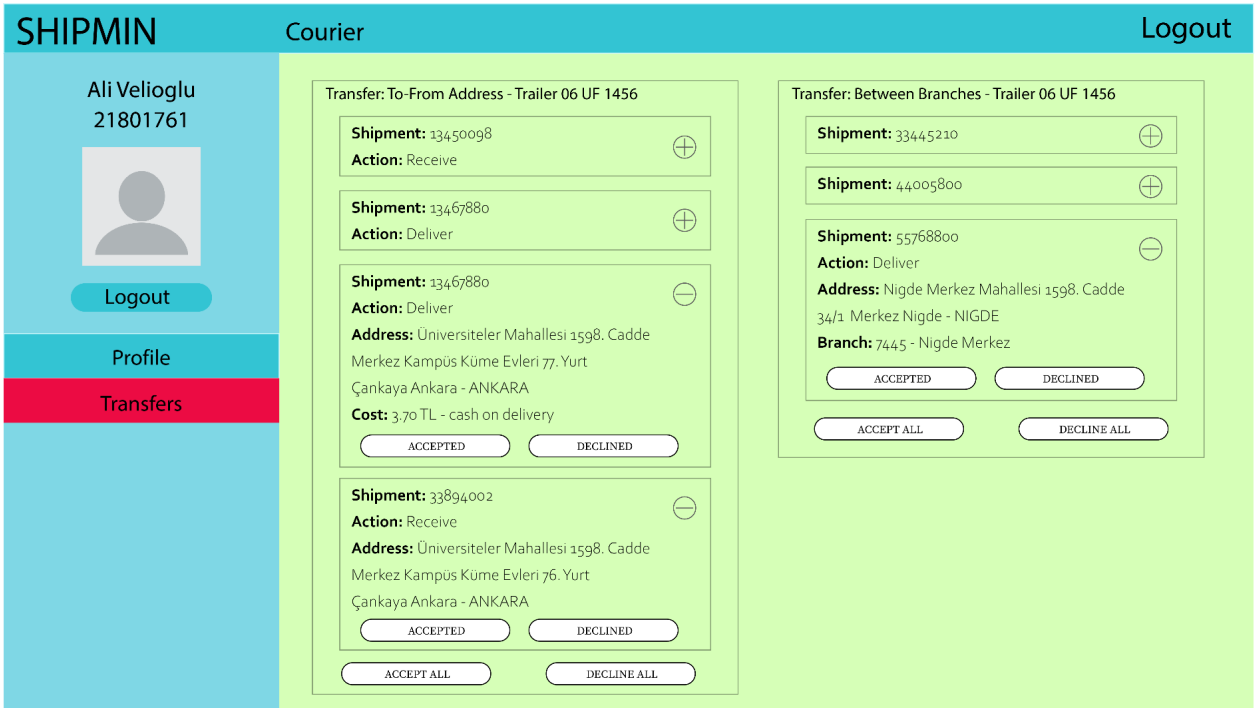
Then, when users wanted to change their information they will make the following SQL query to the database:

```
UPDATE Employee SET ( email= @email, phone = @phone, ...)
        WHERE (employee_id = @employee_id);
```

Another update is password. Users will be able to update their passwords with the SQL query below. Note that additional checks such as checking the old passwords must be made on the server.

```
UPDATE Employee SET ( password = hash(@new_password))
        WHERE (employee_id = @employee_id);
```

3.8) Courier Transfers Page



This page is Courier Transfers Page in which couriers can see the transfer assigned to them which are grouped by their type and cargo trailer. In order to retrieve elements, system uses courier's id.

```
SELECT * FROM transfer
WHERE assigned_employee_id = @employee_id;
```

When all the transfers are retrieved, they are grouped in the system. Couriers also check that whether cargo is accepted or declined by using "ACCEPTED" and "DECLINED" buttons. Also courier can accept or decline all the transfers by using "ACCEPT ALL" and "DECLINE ALL" buttons. They are actually calling accept or decline functions for each transfer.

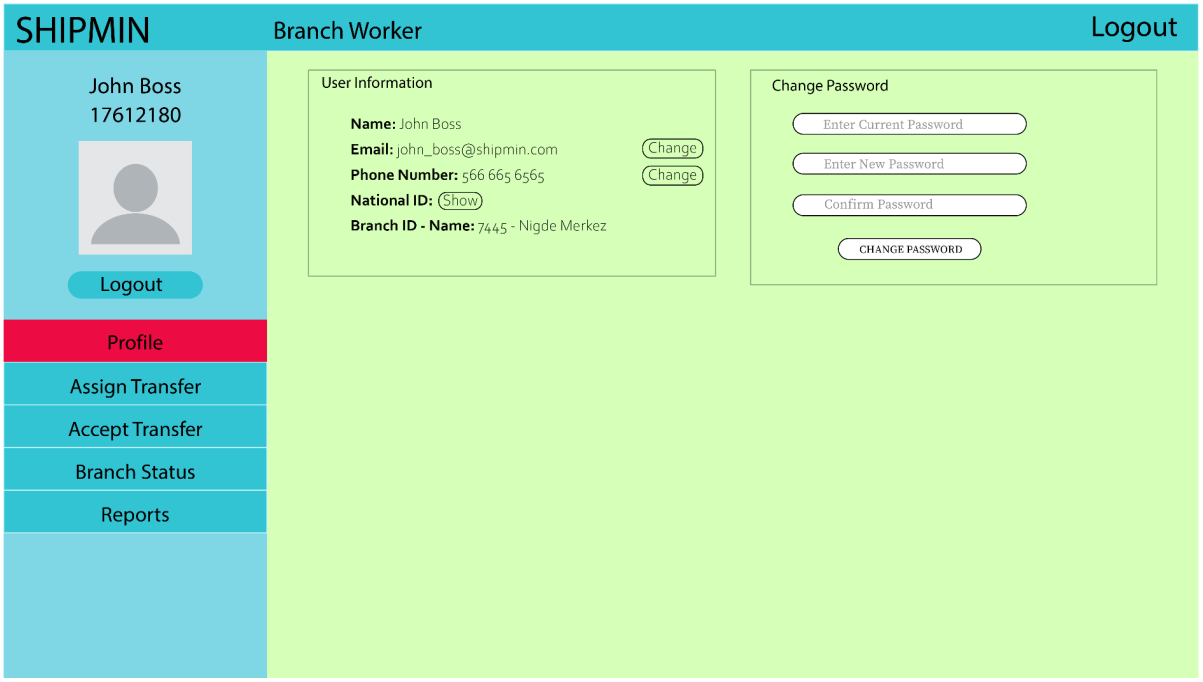
```
UPDATE transfer SET ( success_type = 1 )
WHERE transfer_id = @transfer_id;

UPDATE shipment SET ( branch_id = @current_branch)
WHERE shipment_id = @shipment_id;
```

These queries will be executed when transfer is accepted.

```
UPDATE transfer SET ( success_type = 0 )
WHERE transfer_id = @transfer_id;
```

This query will be executed when transfer is declined.

3.9) Branch Worker Profile Page



This page is the Branch Worker Profile Page, with the help of this page branch workers will be able to edit and see their information. They will be able to change their password and switch to other pages.

In order to show the relevant information to the user, we need to make a Select query to the database. We will store the employee_id of the user in the session, so that we can make queries efficiently.

```
SELECT * FROM Employee WHERE ( employee_id = @employee_id )
```

We will fetch the information related to this specific user via the SQL query above.

Then, when users wanted to change their information they will make the following SQL query to the database:

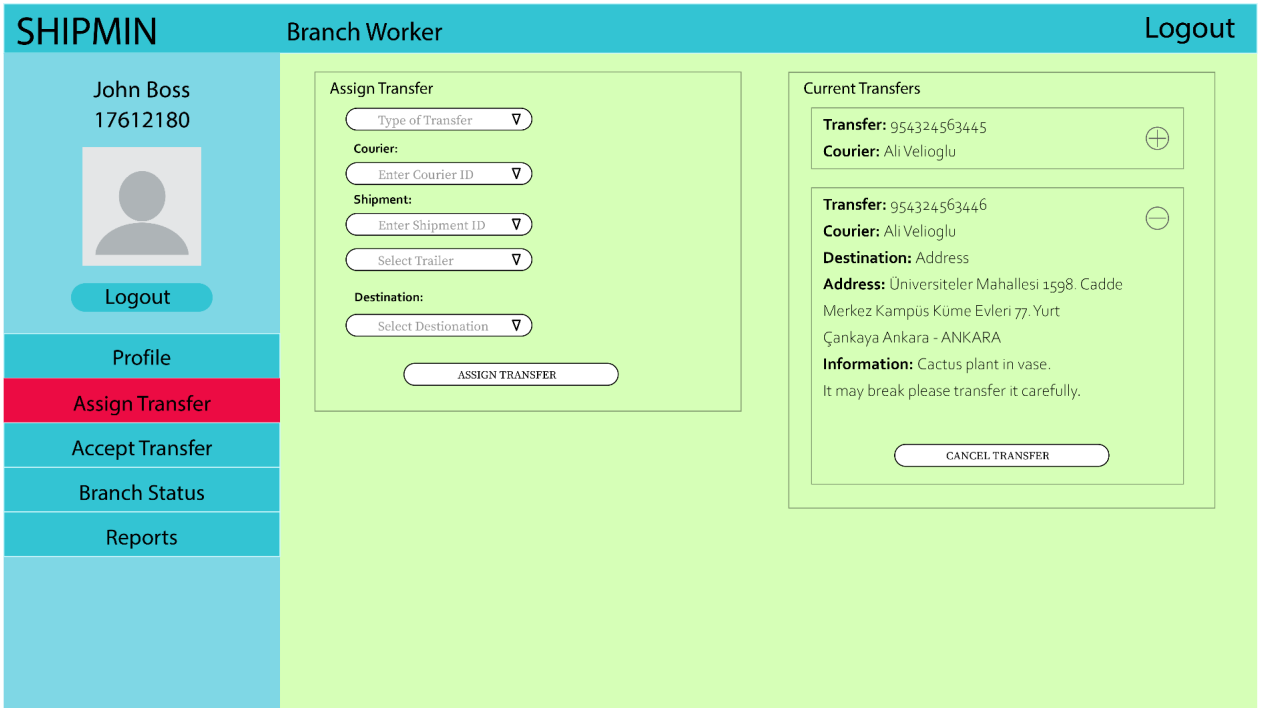
```
UPDATE Employee SET ( email= @email, phone = @phone, ...)
        WHERE (employee_id = @employee_id);
```

Another update is the password. Users will be able to update their passwords with the SQL query below. Note that additional checks such as checking the old passwords must be made on the server.

```
UPDATE Employee SET ( password = hash(@new_password))
```

WHERE (employee_id = @employee_id);

3.10) Assign Transfer Page



This page is Assign Transfer Page where branch workers can assign new transfers to current shipments in branch and couriers. It also provides canceling option to lately assigned transfers.

INSERT INTO transfer(transfer_type, shipment_id, trailer_plate, assigned_employee_id, creator_id, departure_id, destination_id) VALUES (@transfer_type, @shipment_id, @trailer_plate, @courier_id, @employee_id, @branch_id, @destination_id);

This query will create new transfer.

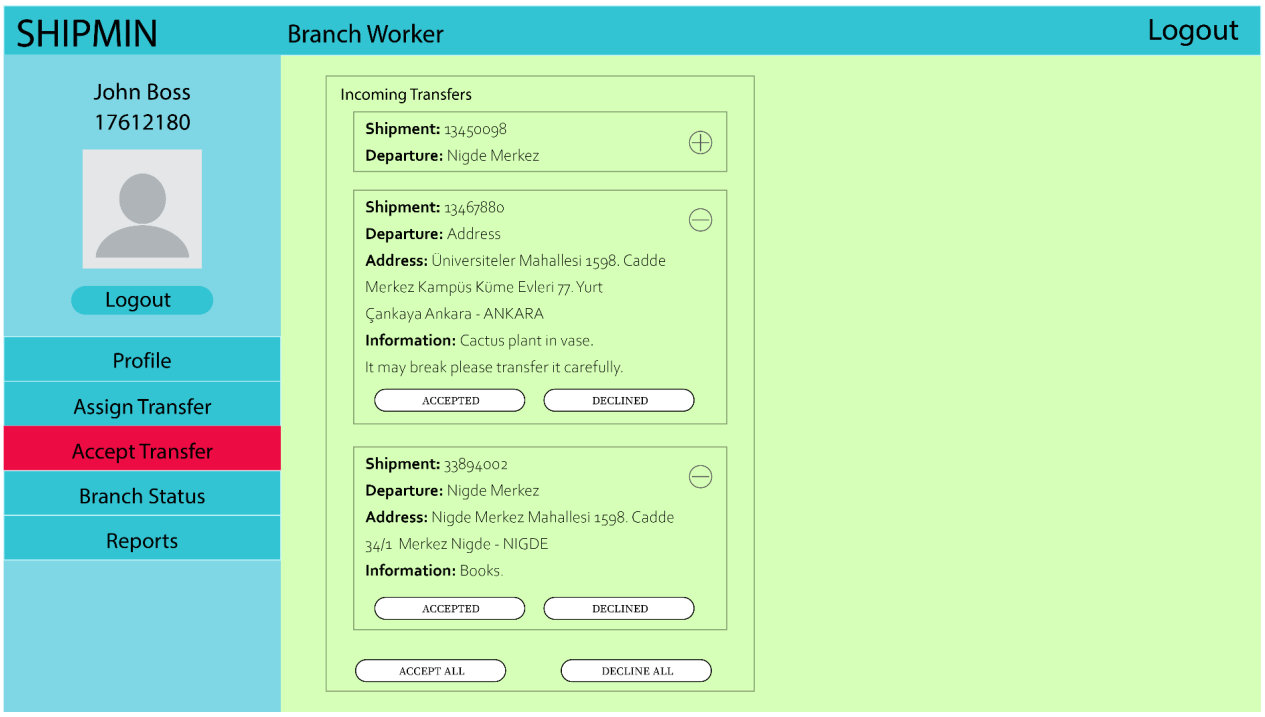
SELECT * FROM transfer, shipment
WHERE creator_id = @employee_id AND transfer.shipment_id = shipment.shipment_id AND transfer.success_type = NULL;

This query will be used for displaying recent transfers.

DELETE FROM transfer WHERE transfer_id = @transfer_id;

This query will cancel the lastly created transfer.

3.11) Accept Transfer Page



This page is Accept Transfer Page which displays incoming transfers to the branch of logged in branch worker. Also branch worker can accept or decline these transfers via “ACCEPTED” and “DECLINED” buttons.

```
SELECT * FROM transfer, shipment
WHERE destination_id = @current_branch AND transfer.shipment_id =
shipment.shipment_id AND transfer.success_type = NULL;
```

This query is used for displaying incoming transfers.

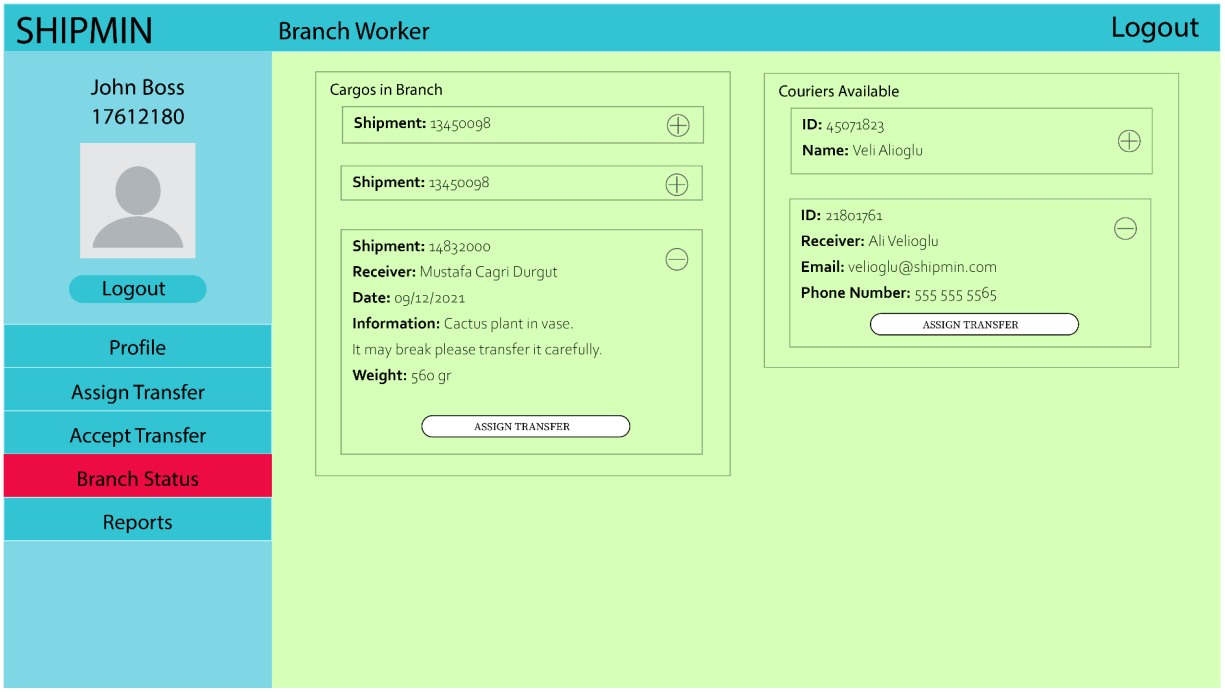
```
UPDATE transfer SET ( success_type = 1 )
WHERE transfer_id = @transfer_id;

UPDATE shipment SET ( branch_id = @current_branch)
WHERE shipment_id = @shipment_id;
```

These queries are used for accepting incoming transfers.

```
UPDATE transfer SET ( success_type = 0 )
WHERE transfer_id = @transfer_id;
```

This query is used for declining incoming transfers.

3.12) Branch Status Page



This page is Branch Status Page which is used for displaying current couriers and cargos exist in branch.

```
SELECT * FROM shipment
WHERE branch_id = @current_branch;
```

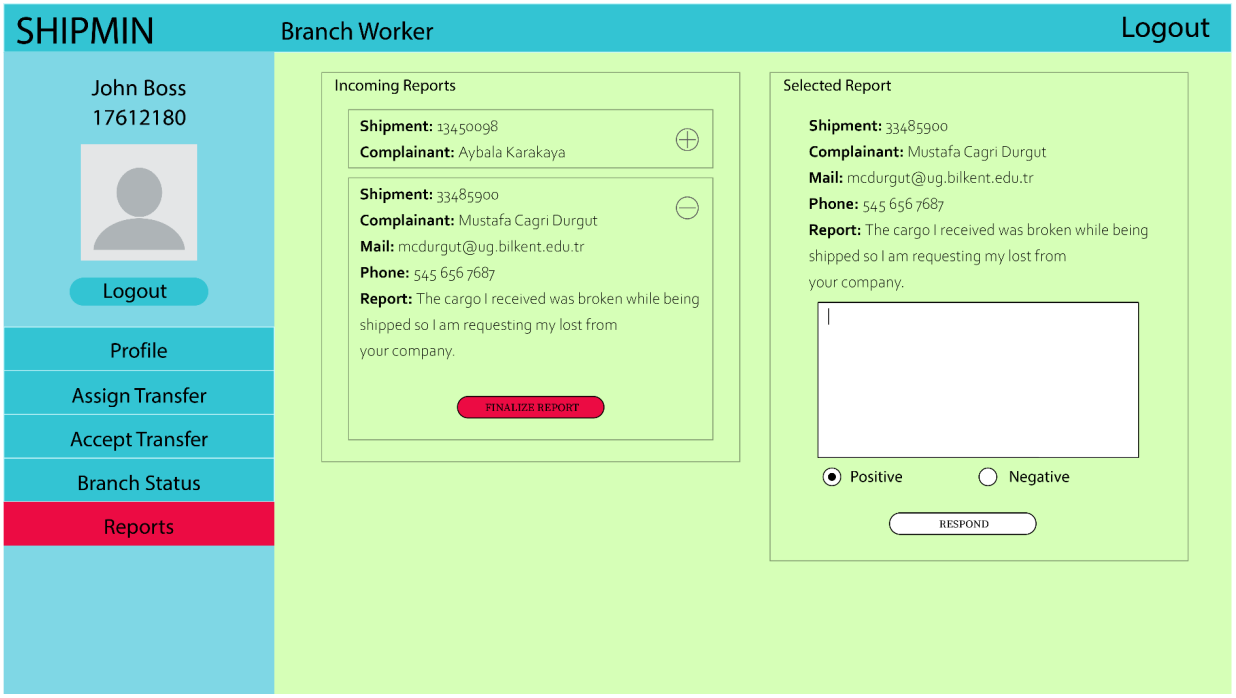
This query is used for displaying cargos.

```
SELECT * FROM courier
WHERE branch_id = @current_branch;
```

This query is used for displaying couriers.

When branch worker clicks on “ASSIGN TRANSFER” button, page will become assign transfer page with spaces in assigning transfer area is filled with necessary information which comes with clicked button. This functionality does not need any query.

3.13) Report Page



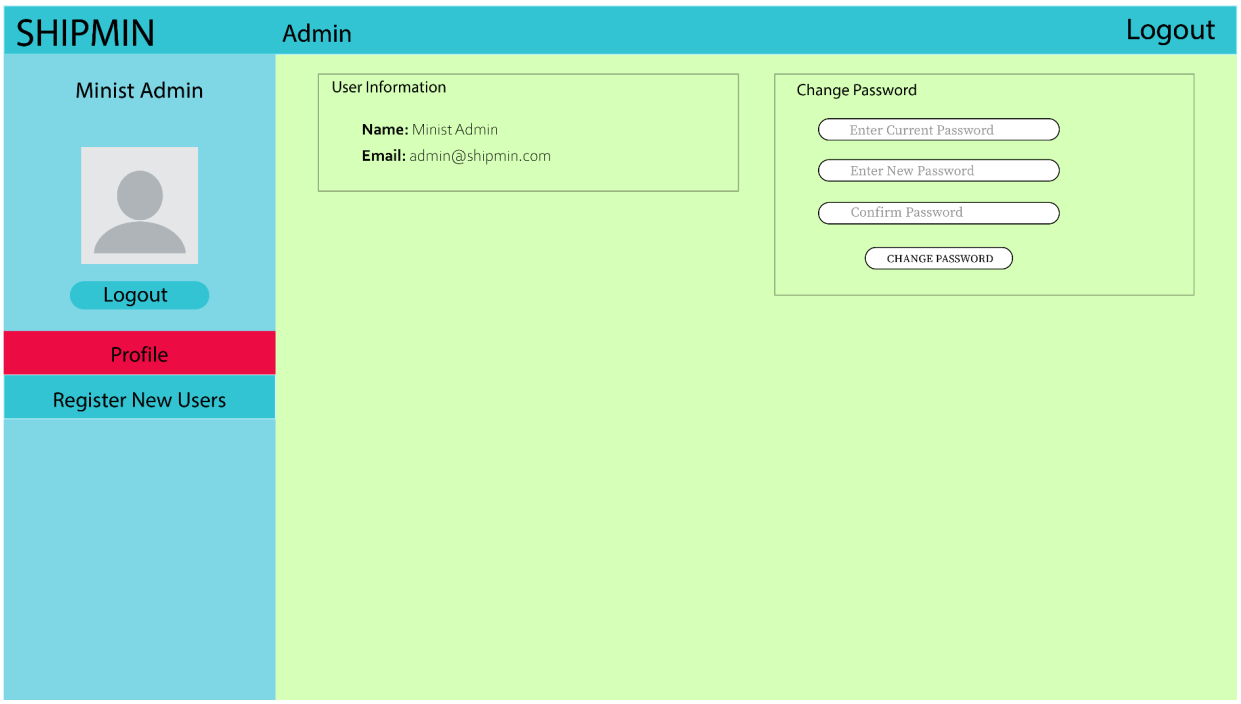
This page is Reports Page where branch workers can select among incoming reports and responses them.

```
SELECT * FROM report
WHERE respondent_id = NULL AND branch_id = @current_branch;
```

This query is used for displaying all the coming reports to that branch. When branch worker clicks on “FINALIZE REPORT” button, extended version of that report will appear in left. By clicking “RESPOND” button, branch worker can respond to that report.

```
UPDATE report SET ( respondent_id = @employee_id, response =
@response, status = @status )
WHERE report_id = @report_id;
```

3.14) Admin Profile Page



This page is the Admin Profile Page, with the help of this page the admins will be able to update their information and see their information. They can also switch to other pages.

The button on the top right will allow the admins to log off from the system. When admins press this button a request to the server that the session is wanted to be ended will be sent. No SQL queries are necessary for this function.

Admins will be able to switch pages in the application using the menu on the left. Again, no SQL queries needed for this function.

In order to show the relevant information to the admin, we need to make a Select query to the database. We will store the email of the admin in the session, so that we can make queries efficiently.

```
SELECT * FROM Admin WHERE ( email = @email )
```

We will fetch the information related to this specific admin via the SQL query above.

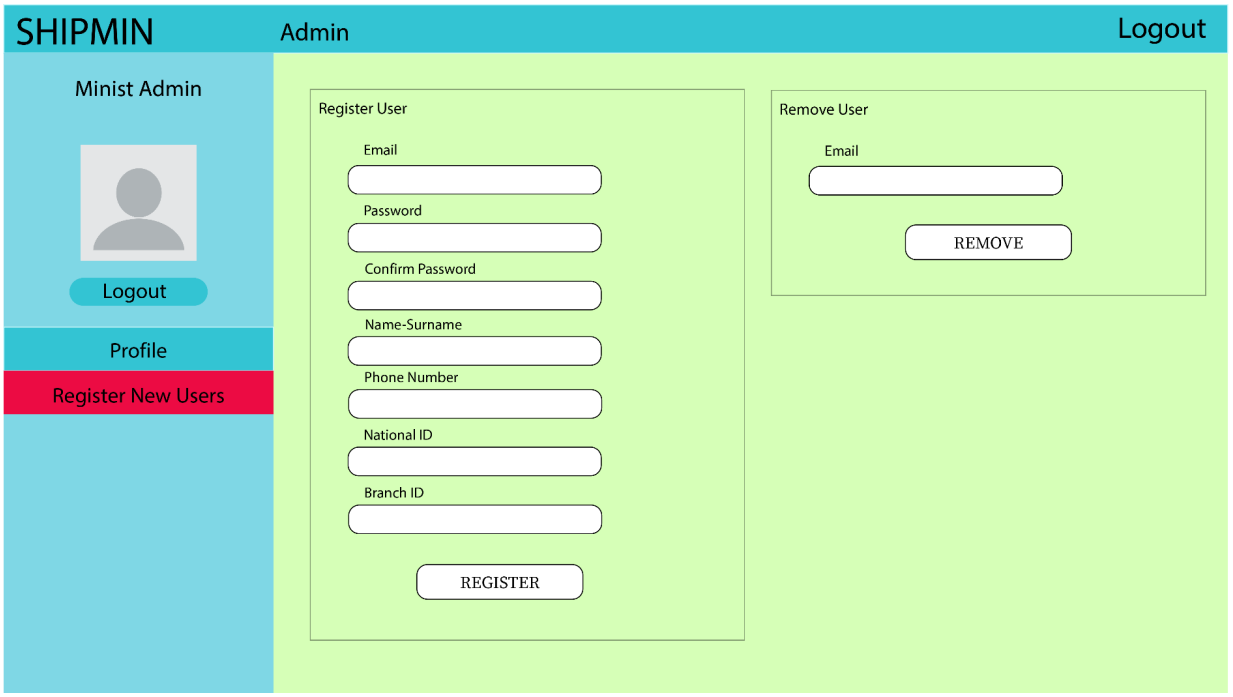
Then, when admins want to change their information they will make the following SQL query to the database:

```
UPDATE Admin SET ( name = @name )
        WHERE (email = @email);
```

Another update is password. Users will be able to update their passwords with the SQL query below. Note that additional checks such as checking the old passwords must be made on the server.

```
UPDATE Admin SET ( password = hash(@new_password))
        WHERE (email = @email);
```

3.15) Admin Register Page



This page is the Admin Register Page. Admins can register new employees and remove the existing ones using this page. They can also switch to other pages and logout from the button on the right top.

To register an employee, an admin should fill the fields of the form on the left side (email, password, confirm password, name-surname, phone number, national ID, and branch ID) and click on the “register” button.

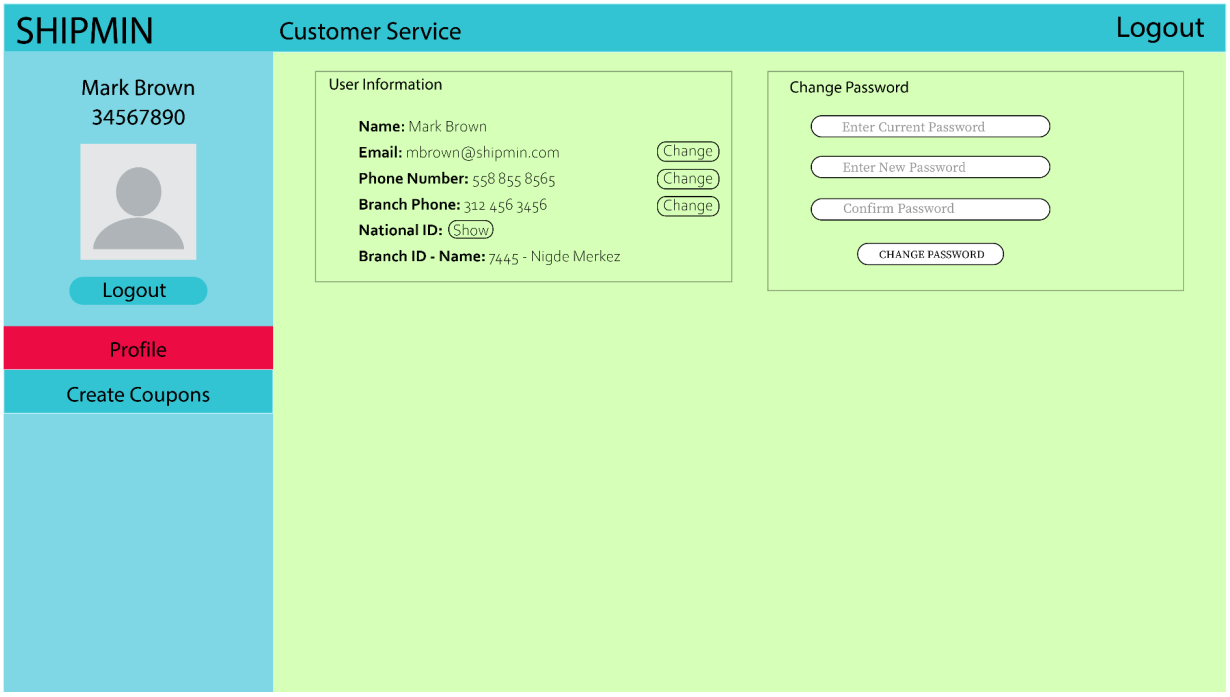
INSERT INTO Employee(name, phone_no, TC_passport, email, password, branch_is, adder_admin) VALUES (@name, @phone_no, @TC_passport, @email, hash(@password), @branch_id, @admin_email);

This query will register a new employee in the database and will be run when an admin clicks on the “register” button after filling the form. Note that checks such as whether the phone number contains an appropriate amount of digits and whether password and “confirm password” fields match should be done on the server side before running this query.

Admins can also delete existing employees. For that, they should enter the email of the employee they want to remove from the database and click on the “remove” button. After they click on this button, this query will be run:

DELETE FROM Employee WHERE email = @email;

3.16) Customer Service Profile Page



This page is the Customer Service Profile Page. Customer Service employees can change their passwords using that page. They can also switch to other pages and logout from the button on the right top.

In order to show the relevant information to the employee, we need to make a Select query to the database.

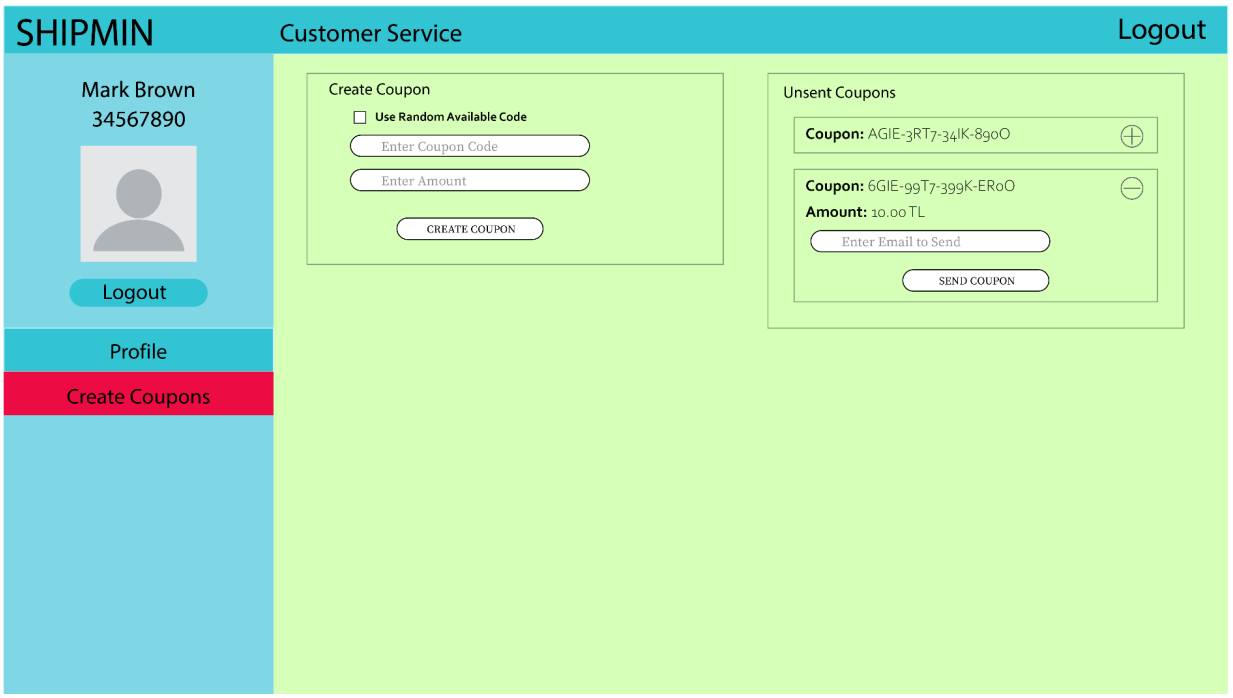
SELECT * FROM Customer_service, Employee WHERE (employee_id = @employee_id)

We will fetch the information related to this specific customer service employee via the SQL query above.

Then, when employees want to change their information they will make the following SQL query to the database. Note that additional checks such as checking the old passwords must be made on the server.

UPDATE Employee SET (password = hash(@new_password))
WHERE (employee_id = @employee_id);

3.17) Create Coupon Page



Customer service employees can create coupons, see unsent coupons and send coupons using this page.

To create a coupon, they should enter a coupon code or choose to use a random available code and they should enter the amount. After they press the “create coupon” button, the server will send the following query to the database:

INSERT INTO Coupon(coupon_no, amount, used, creator_id, owner_id, is_sent) VALUES (@coupon_no, @amount, 0, @employee_id, NULL);

To display the unsent coupons, this query will be used:

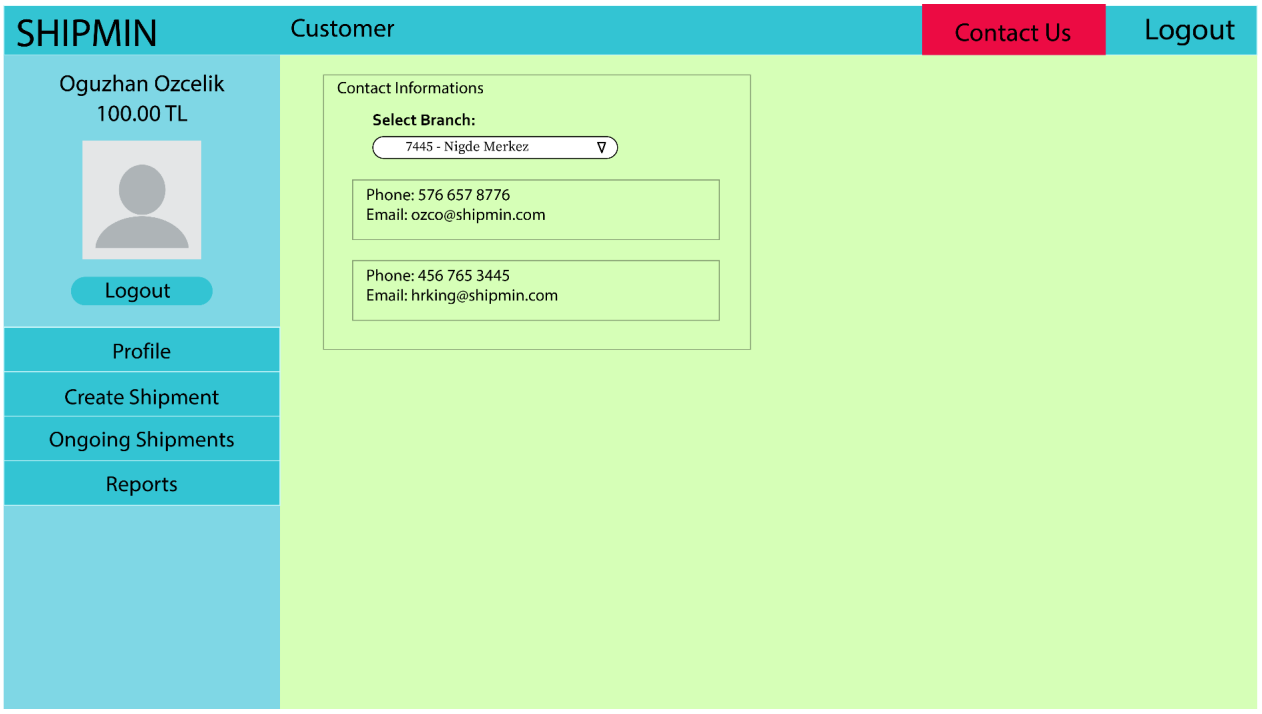
SELECT * FROM Coupon WHERE (is_sent= 0);

Then, for a coupon the customer service employee can enter an email address and send the coupon to the email address by pressing the “send coupon” button. Note that it would be checked whether the email address belongs to a customer on the server side. After they press this button, the following query will be sent to the database:

SELECT customer_id FROM Customer WHERE(email=@customer_email);

UPDATE Coupon SET (is_sent= 1, owner_id=@customer_id)
WHERE (coupon_no = @coupon_no);

3.18) Contact Page



Using this page, a user can see the contact information for a particular branch.

For this, they should select a branch from the drop-down. Afterwards, the following query will be sent to the database:

SELECT branch_phone_no, email FROM Customer_service, Employee WHERE(branch_id = @branch_id);

4) Implementation Plan

We are planning to implement this shipping system as a web application. We are going to use MySQL for database management service. We are going to divide the task between the members of the team.

Firstly, we presented the design of our database in this report, we will implement it according to the feedback of our Teaching Assistant. Every member of the team will be contributing to this implementation.

Halil Ozgur will be the head of the Graphical User Interface team, he will implement the GUI sketches that we presented in this report into HTML/CSS/JavaScript code.

For the server side, we are planning to use PHP to connect our database and implement the functionality. The rest of the team will implement this side of the project. When the server side is finished these members will help to the front-end of the project.