BİLKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

Design Report

Group 4

İlke Kaş 21803184

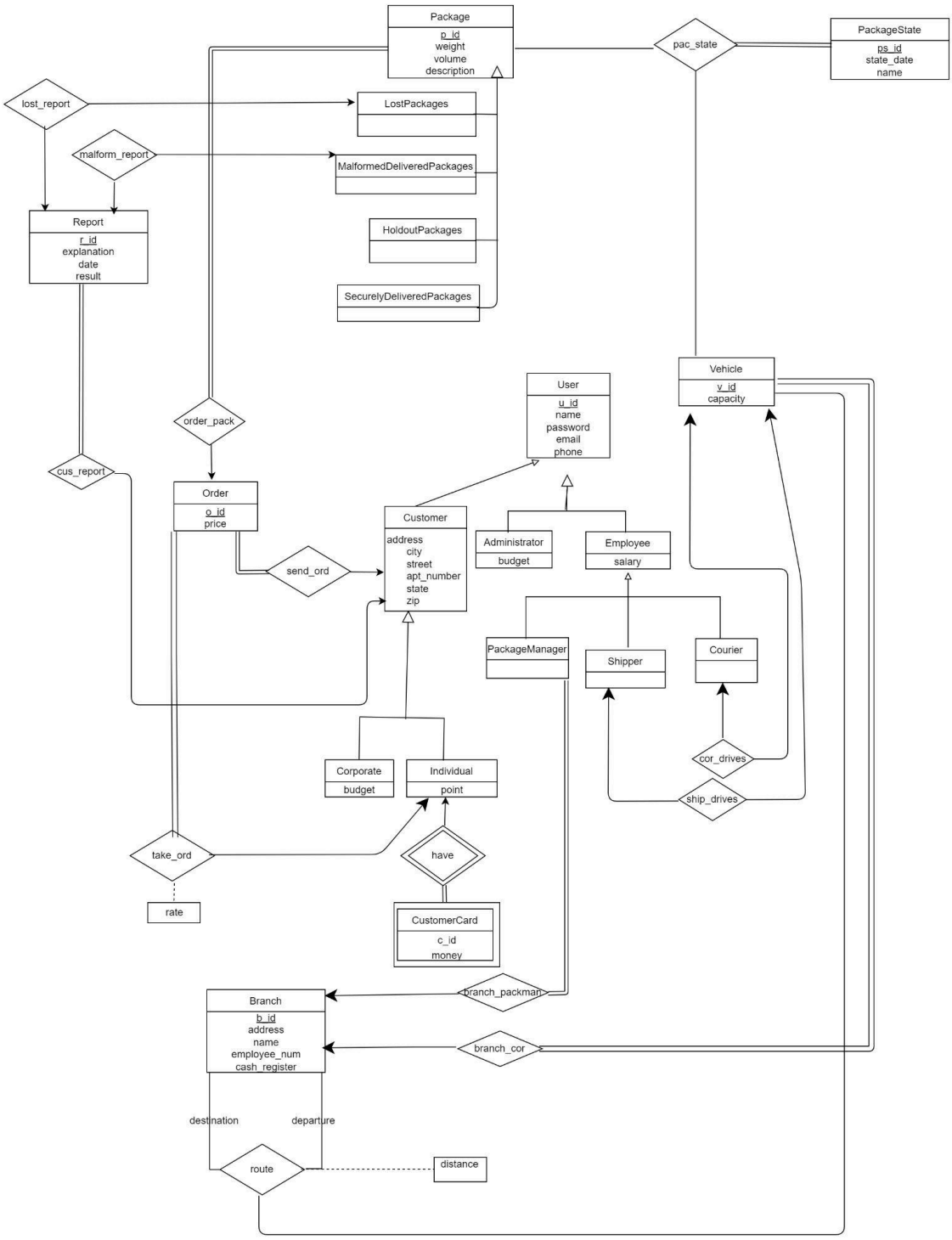
Zeynep Büşra Ziyagil 21802646

Bilgehan Akcan 21802901

Ömer Onat Postacı 21802349

Table of Contents

1. Final E/R Diagram



2. Relational Schemas and SQL

2.1. User

Relational Model:

User(u_id, name, password, surname, email, phone)

Primary Key: u_id

2.2. Administrator

Relational Model: Administrator(u_id, budget)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

2.3. Employee

Relational Model: Employee(u_id, salary)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

2.4. Package Manager

Relational Model: PackageManager(u_id, b_id)

Primary Key: u_id

Foreign Key: b_id references to Branch(b_id)

Foreign Key: u_id references User(u_id)

2.5. Shipper

Relational Model: Shipper(u_id, v_id)

Primary Key: u_id

Foreign Key: v_id references Vehicle(v_id)

Foreign Key: u_id references User(u_id)

2.6. Courier

Relational Model: Courier(u_id, v_id)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

Foreign Key: v_id references Vehicle(v_id)

2.7. Customer

Relational Model: Customer(u_id, city, street, apt_number, state, zip)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

2.8. Corporate

Relational Model: Corporate(u_id, budget)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

2.9. Individual

Relational Model: Individual(u_id, point)

Primary Key: u_id

Foreign Key: u_id references User(u_id)

2.10. Customer Card

Relational Model: CustomerCard(u_id, c_id, money)

Primary Key: u_id, c_id

Foreign Key: u_id references User(u_id)

SQL Definition:

```
CREATE TABLE CustomerCard(
u_id        VARCHAR(30) NOT NULL,
c_id        VARCHAR(20) NOT NULL,
"money"     NUMERIC DEFAULT 0 ,
PRIMARY KEY(u_id, c_id),
FOREIGN KEY(u_id) REFERENCES "User"(u_id)
```

```
ON DELETE CASCADE
);
```

2.11. Order

Relational Model: Order(o_id, take_indv_id, send_customer_id, price, rate)

Primary Key: o_id

Foreign Key: take_indv_id references Individual(u_id)

Foreign Key: send_customer_id references Customer(u_id)

SQL Definition:

```
CREATE TABLE "Order"(
o_id                SERIAL NOT NULL,
take_indv_id        VARCHAR(30) NOT NULL,
send_customer_id    VARCHAR(30) NOT NULL,
price               NUMERIC NOT NULL,
rate                NUMERIC,
PRIMARY KEY(o_id),
FOREIGN KEY(take_indv_id) REFERENCES "User"(u_id)
ON DELETE CASCADE,
FOREIGN KEY(send_customer_id) REFERENCES "User"(u_id)
ON DELETE CASCADE
);
```

2.12. Package

Relational Model: Package(p_id, o_id, weight, volume)

Primary Key: p_id

Foreign Key: o_id references Order(o_id)

SQL Definition:

```
CREATE TABLE Package(
p_id            SERIAL NOT NULL,
o_id            INTEGER NOT NULL,
weight          NUMERIC NOT NULL,
item_dscrptn    VARCHAR not null,
volumeNUMERIC NOT NULL,
PRIMARY KEY(p_id),
FOREIGN KEY(o_id) REFERENCES "Order"(o_id)
ON DELETE CASCADE
);
```

2.13. Malformed Delivered Package

Relational Model: MalformedDeliveredPackage(p_id, report_id)

Primary Key: p_id

Foreign Key: p_id references Package(p_id)

Foreign Key: report_id references Report(r_id)

SQL Definition:

```
CREATE TABLE MalformedDeliveredPackage(
report_id       INTEGER,
FOREIGN KEY(report_id) REFERENCES Report(r_id)
) INHERITS(Package);
```

2.14. Lost Packages

Relational Model: LostPackages(p_id, report_id)

Primary Key: p_id

Foreign Key: p_id references Package(p_id)

Foreign Key: report_id references Report(r_id)

SQL Definition:

```
CREATE TABLE LostPackages(
report_id        INTEGER,
FOREIGN KEY(report_id) REFERENCES Report(r_id)
) INHERITS(Package);
```

2.15. Securely Delivered Packages

Relational Model: SecurelyDeliveredPackages(p_id)

Primary Key: p_id

Foreign Key: p_id references Package(p_id)

SQL Definition:

```
CREATE TABLE SecurelyDeliveredPackages(
) INHERITS(Package);
```

2.16. Holdout Packages

Relational Model: HoldoutPackages(p_id)

Primary Key: p_id

Foreign Key: p_id references Package(p_id)

SQL Definition:

```
CREATE TABLE HoldoutPackages(
) INHERITS(Package);
```

2.17. Transaction

Relational Model: Transaction(t_id, o_id, tot_price)

Primary Key: t_id

Foreign Key: o_id references Order(o_id)

SQL Definition:

```
CREATE TABLE "Transaction"(
t_id          SERIAL NOT NULL,
o_id          INTEGER NOT NULL,
tot_price     NUMERIC,
PRIMARY KEY(t_id),
FOREIGN KEY(o_id) REFERENCES "Order"(o_id)
ON DELETE CASCADE
ON UPDATE CASCADE
) ;
```

2.18. Package State

Relational Model: PackageState(ps_id, state_date, name)

Primary Key: ps_id

SQL Definition:

```
CREATE TABLE PackageState(
ps_id         SERIAL NOT NULL,
"name"        VARCHAR(30) NOT NULL,
state_date    DATE,
PRIMARY KEY(ps_id)
) ;
```

2.19. Report

Relational Model: Report(r_id, u_id, explanation,result, date)

Primary Key: r_id

Foreign Key: u_id references Customer(u_id)

SQL Definition:

```
CREATE TABLE Report(
r_id          SERIAL NOT NULL,
```

```
u_id            VARCHAR(30) NOT NULL,
p_id            INT NOT NULL,
explanation     VARCHAR(300) NOT NULL,
"result"        VARCHAR(10),
"date"          DATE,
PRIMARY KEY(r_id),
FOREIGN KEY(u_id) REFERENCES "User"(u_id)
ON DELETE CASCADE
FOREIGN KEY(p_id) REFERENCES Package (p_id)
ON DELETE CASCADE
) ;
```

2.20. Vehicle

Relational Model: Vehicle(v_id, capacity)

Primary Key: v_id

SQL Definition:

```
CREATE TABLE Vehicle(
v_id            SERIAL NOT NULL,
capacity        NUMERIC,
PRIMARY KEY(v_id)
) ;
```

2.21. Courier Vehicle

Relational Model: CourierVehicle(v_id, b_id)

Primary Key: v_id

Foreign Key: v_id referencesVehicle(v_id)

Foreign Key: b_id references Branch(b_id)

SQL Definition:

```
CREATE TABLE CourierVehicle(
b_id            INTEGER,
FOREIGN KEY(b_id) REFERENCES Branch(b_id)
ON DELETE CASCADE
) INHERITS(Vehicle);
```

2.22. Shipper Vehicle

Relational Model: ShipperVehicle(v_id)

Primary Key: v_id

Foreign Key: v_id referencesVehicle(v_id)

SQL Definition:

```
CREATE TABLE ShipperVehicle(
) INHERITS(Vehicle);
```

2.23. Branch

Relational Model: Branch(b_id, address, name, employee_num, cash_register)

Primary Key: b_id

SQL Definition:

```
CREATE TABLE Branch(
b_id            SERIAL NOT NULL,
address         VARCHAR(100) NT NULL,
"name"          VARCHAR(30) UNIQUE NOT NULL,
employee_num    INTEGER,
cash_register   NUMERIC,
PRIMARY KEY(b_id)
) ;
```

2.24. Pac_State

Relational Model: Pac_State(ps_id,p_id,v_id)

Primary Key: ps_id, p_id, v_id

Foreign Key: v_id referencesVehicle(v_id)

Foreign Key: ps_id referencesPackageState(ps_id)

Foreign Key: p_id references Package(p_id)

SQL Definition:

```
CREATE TABLE Pac_State(
ps_id        INTEGER NOT NULL,
p_id         INTEGER,
v_id         INTEGER,
PRIMARY KEY(ps_id, p_id, v_id),
FOREIGN KEY(ps_id) REFERENCES PackageState(ps_id)
ON DELETE CASCADE,
FOREIGN KEY(p_id) REFERENCES Package(p_id)
ON DELETE CASCADE,
FOREIGN KEY(v_id) REFERENCES Vehicle(v_id)
) ;
```

2.25. Route

Relational Model: Route(destination_b_id, departure_b_id, v_id, distance)

Primary Key: destination_b_id, departure_b_id , v_id

Foreign Key: destination_b_id references Branch(b_id)

Foreign Key: departure_b_id references Branch(b_id)

Foreign Key: v_id referencesVehicle(v_id)

SQL Definition:

```
CREATE TABLE Route(
```

```
destination_b_id        INTEGER,
departure_b_id          INTEGER,
v_id                    INTEGER,
distance                NUMERIC,
PRIMARY KEY(destination_b_id, departure_b_id  , v_id),
FOREIGN KEY(destination_b_id) REFERENCES Branch(b_id)
ON DELETE CASCADE,
FOREIGN KEY(departure_b_id) REFERENCES Branch(b_id)
ON DELETE CASCADE,
FOREIGN KEY(v_id) REFERENCES Vehicle(v_id)
) ;
```

3. User Interface Design and Needed SQL Statements

3.1. Sign in

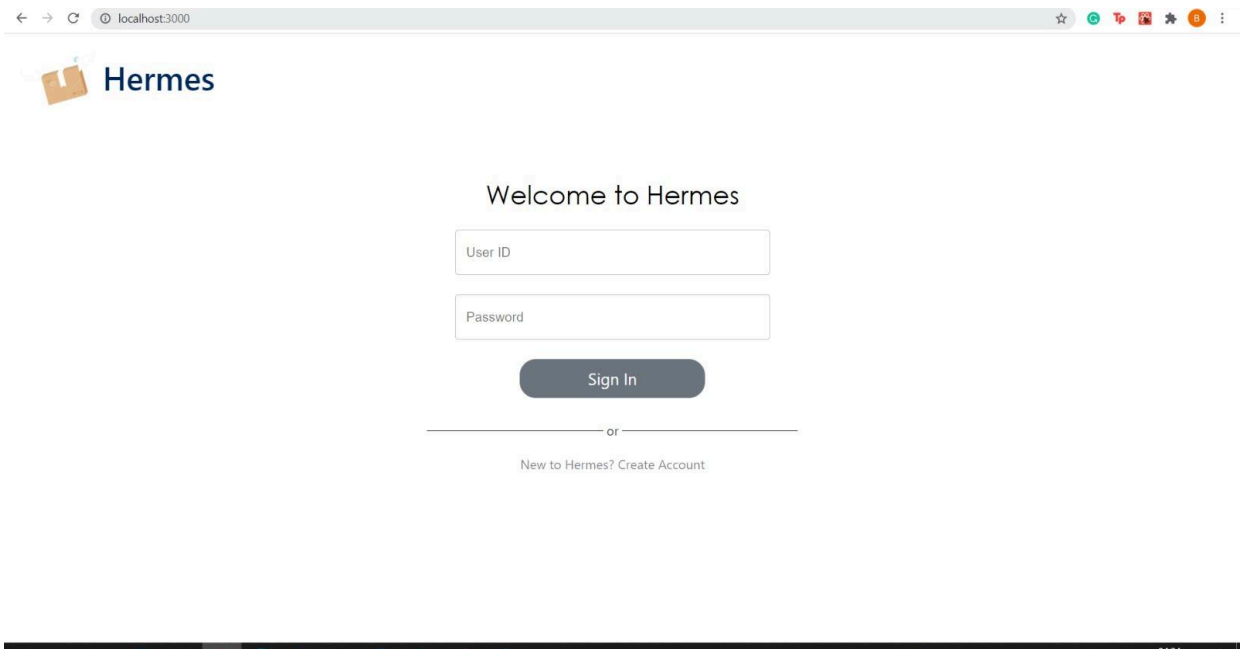


Figure 1

Explanation: Users will sign in to Hermes by using their user ids and passwords

Inputs: @email, @password

Needed SQL Statements:

- SELECT * FROM User WHERE u_id = @u_id AND password = @password

3.2. Registration

3.2.1. Registration as Customer

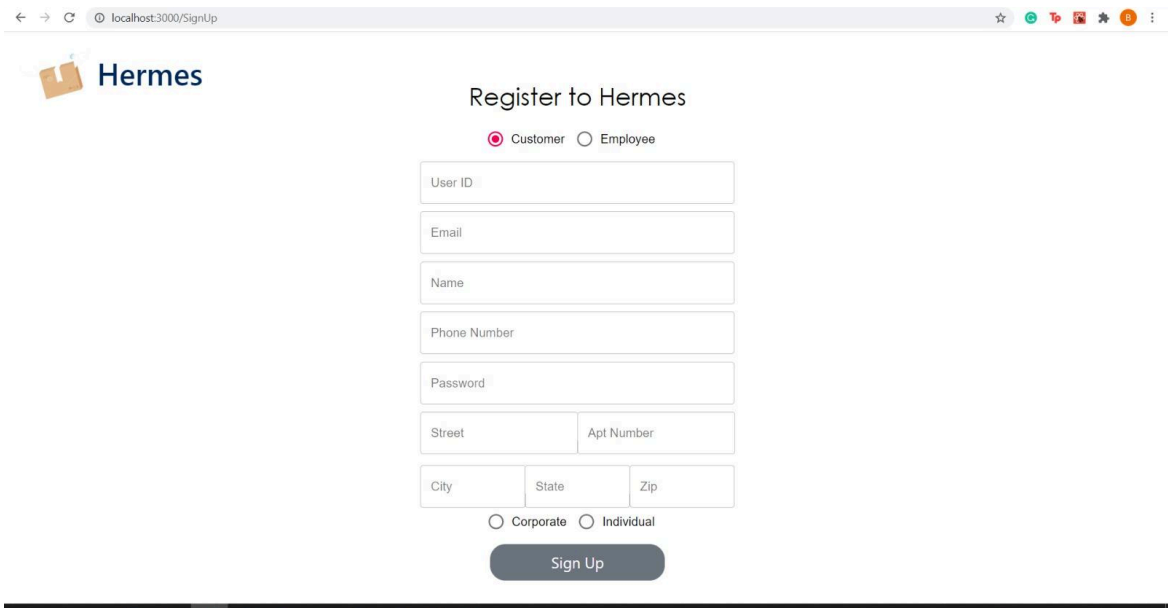


Figure 2

Explanation: Customers will register to Hermes by entering their information. These are: user id that they will use when sign in to Hermes, email, name, password, phone number and their address information. Besides that they have to choose what kind of customer they are: corporate or individual.

Inputs: @userid, @email, @name,@phone, @password, @street, @aptnumber, @city, @state, @zipcode, @customertype

Needed SQL Statements:

- **If @customertype = Corporate**

INSERT INTO Corporate VALUES (@user_id, @name, @password, @email, @phone, @city, @street, @apt_number, @state, @zip, 0);

- **If @customertype = Individual**

INSERT INTO Individual VALUES (@user_id, @name, @password, @email, @phone, @city, @street, @apt_number, @state, @zip, 0);

3.2.2. Registration as Employee

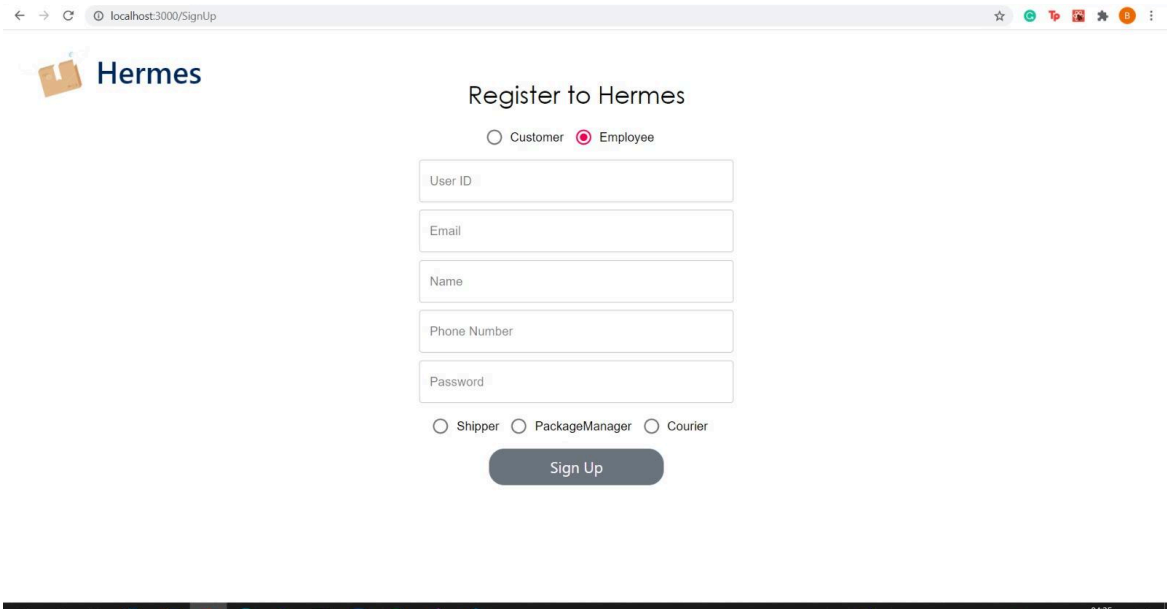


Figure 3

Explanation: Employees will register to Hermes by entering their information. These are: user id that they will use when sign in to Hermes, email, name, password and phone number. Besides that they have to choose what kind of employee they are: shipper, package manager or courier .

Inputs: @userid, @email, @name,@phone, @password , @employeetype

3.2.2.1. Register as PackageManager

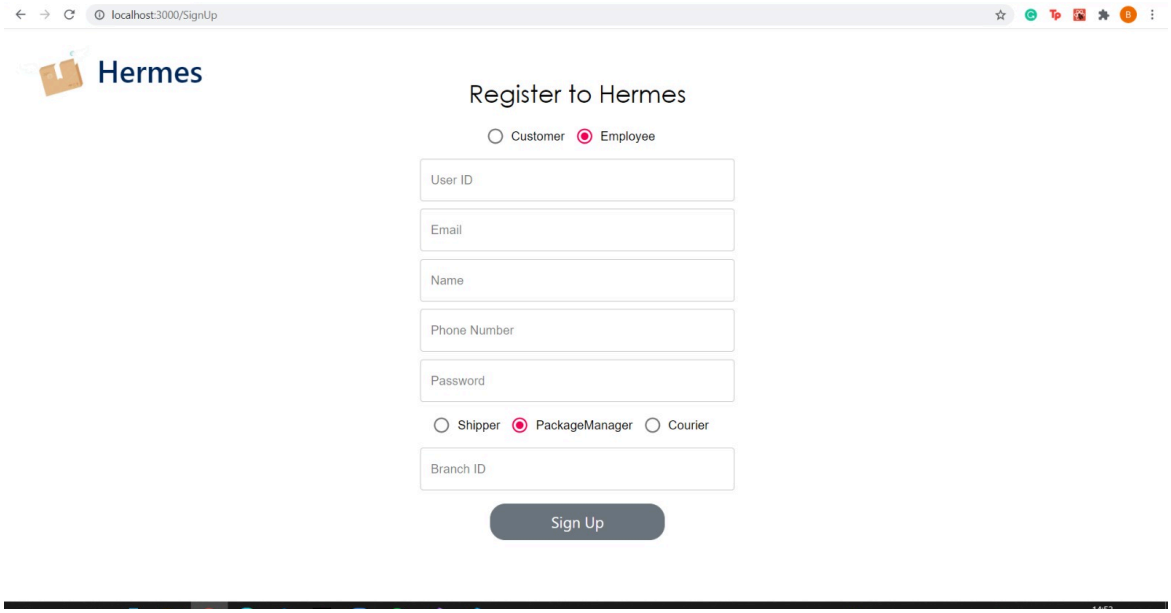


Figure 4

Explanation: According to their employee type they will enter dependent information. If the employee is a package manager, then branch id data will also be taken.

Inputs: @userid, @email, @name,@phone, @password , @employeetype, @branchid

Needed SQL Statements:

- **If @employeetype = PackageManager**

  INSERT INTO PackageManager VALUES (@user_id, @name, @password, @email, @phone, 0, @b_id);

3.2.2.2. Register as Shipper or Courier

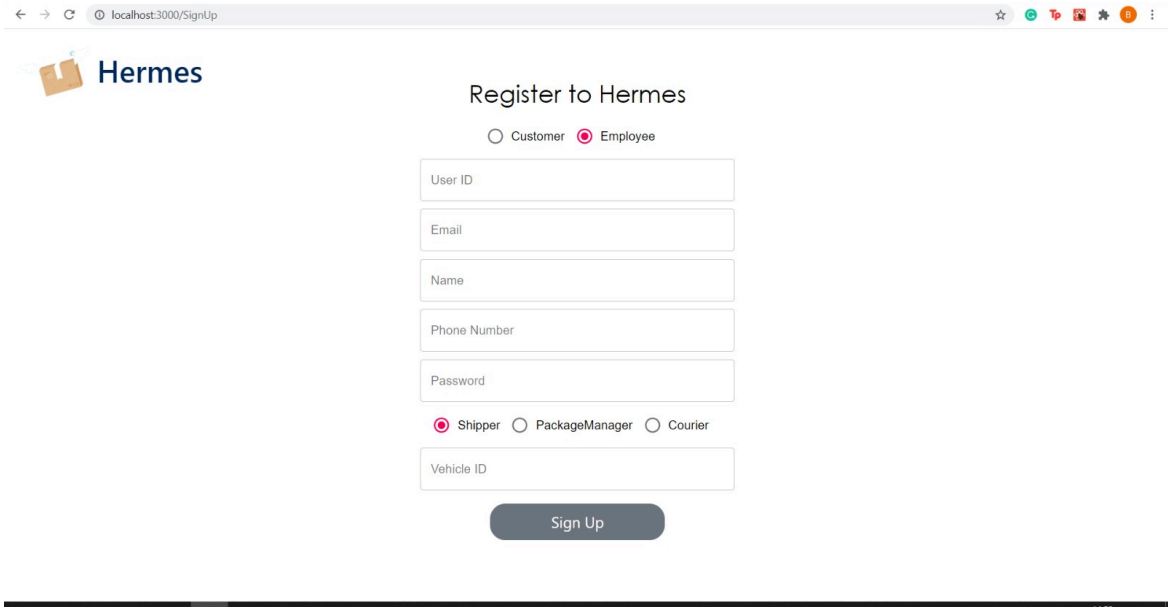


Figure 5

Explanation: According to their employee type they will enter dependent information. If the employee is a courier or a shipper then branch id data will also be taken.

Inputs: @userid, @email, @name,@phone, @password , @employeetype, @vehicleid

Needed SQL Statements:

- **If @employeetype = Shipper**

  INSERT INTO Shipper VALUES (@user_id, @name, @password, @email, @phone, 0, @v_id);

- **If @employeetype = Courier**

  INSERT INTO Courier VALUES (@user_id, @name, @password, @email, @phone, 0, @v_id);

3.3. Home Page

3.3.1. Customer Home Page

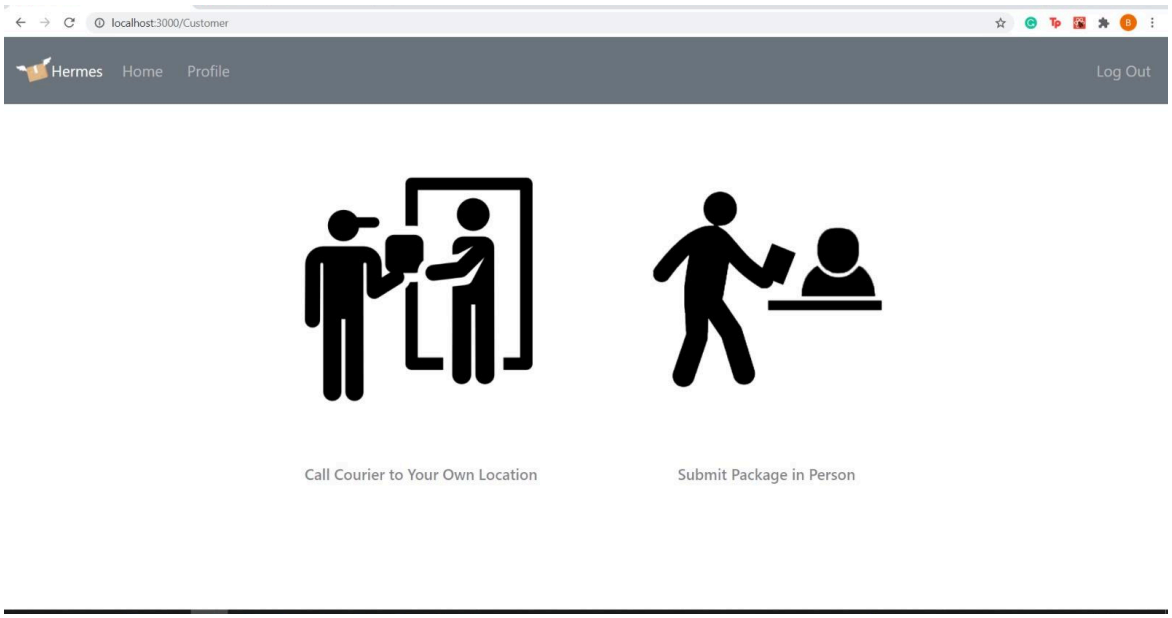


Figure 6

Explanation: At the main page of the customer, the submission type of package will be selected. There are two options: Customers can either call a courier to her/his/its own location or submit a package in person. According to selection, if customers choose to call the courier option, they will be directed to the page shown in ***Figure 11.*** If customers choose to submit the package in person option, they will be directed to the page shown in ***Figure 12.***

3.3.2. Employee Home Page

3.3.2.1. Package Manager Home Page

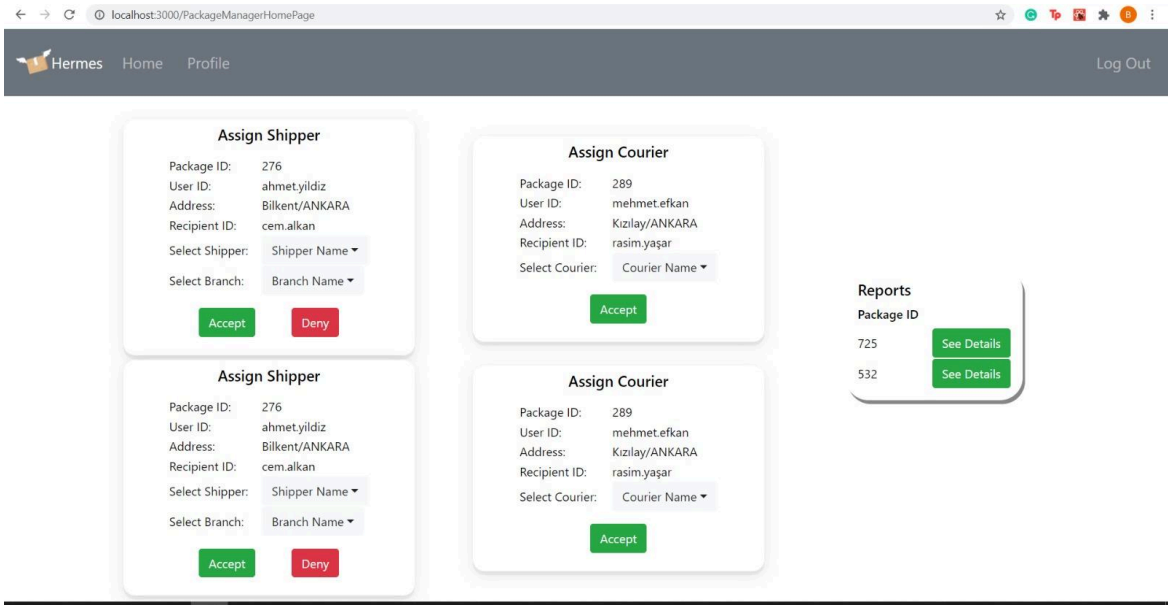


Figure 7

Explanation: Package managers can assign shippers to the packages that come to the branch by a courier. They should first choose the shipper who is currently in this branch and press accept. Then, the shipper will be assigned. If they deny one of the assign shipper sections, the denied package will be shown in the assign courier section of the package manager's home page in order to send the package back to the customer via courier.

They also can assign couriers to the packages that come to the branch by a shipper. They should first choose the courier and press accept. Besides these, package managers can see the list of reports filled by customers in their homepages and finalize the report as positive or negative as in ***Figure 8***.

Inputs: @p_id

Needed SQL Statements:

- SELECT p.p_id, o.send_customer_id, o.take_indv_id, b.address

  FROM (Package p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL

JOIN PackageManager pm NATURAL JOIN Branch b WHERE p.p_id = @p_id;

- SELECT b."name" FROM branch b;
- SELECT s.”name”, s.u_id, s.v_id FROM (Shipper s NATURAL JOIN Route r) INNER JOIN Branch b ON r.departure_b_id = b.b_id WHERE b.”name” = @name
- INSERT INTO PackageState VALUES(‘Shipper’, @date);
- INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);
- SELECT c.”name”, c.v_id FROM Courier c;
- INSERT INTO PackageState VALUES(‘Courier’, @date);
- INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);
- SELECT r.p_id FROM Report;

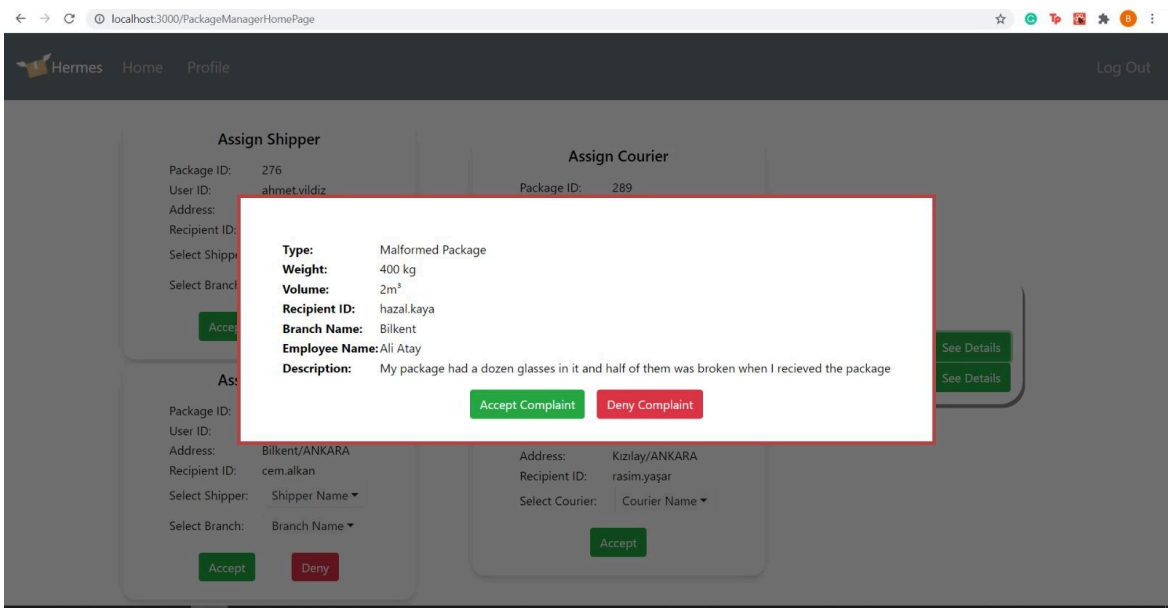


Figure 8

- SELECT r.p_id FROM Report r;
- SELECT p.item_dscrptn, p.weight, p.volume, o.take_indv_id, b."name" , pm.”name”

  FROM (Package p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b WHERE p.p_id = @p_id;

- SELECT * FROM MalformedPackages mp where mp.p_id = @p_id;
- SELECT * FROM LostPackages lp where lp.p_id = @p_id;

3.3.2.2. Courier Home Page

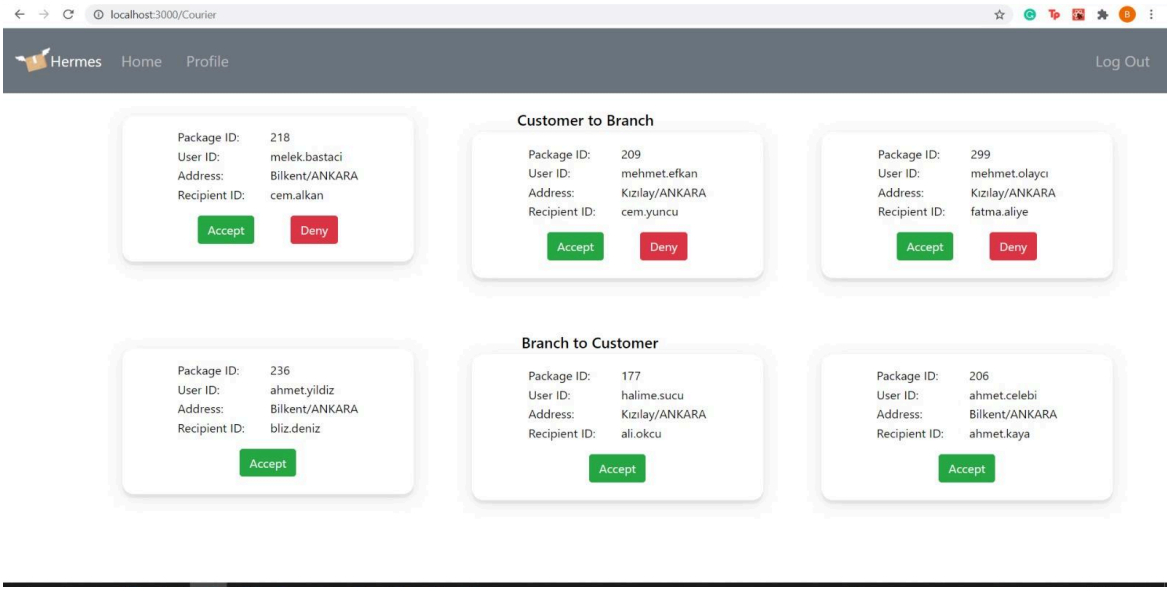


Figure 9

Explanation: Couriers can see the assigned packages that will be delivered from customer address to branch at the top of their home page. Couriers also can see the assigned packages that will be delivered from branch to customer address. They can accept or deny the packages. If they deny the packages in the customer to branch section, notification will be sent to the customer. If they accept the packages in the customer to branch section, this package will be placed in package managers' home page as a shipper assignment as in ***Figure 7***. They will accept the package to deliver it from the branch to the customer.

Needed SQL Statements:

--For Customer to Branch

- SELECT p.p_id, o.send_customer_id, o.take_indv_id, b.address

  FROM (Package p NATURAL JOIN "Order" o)

  INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b WHERE p.p_id = @p_id;

- SELECT c.v_id FROM Courier c WHERE c.u_id = @u_id;

- INSERT INTO PackageState VALUES('Courier', @date);
- INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);

--For Branch to Customer

- SELECT p.p_id, o.send_customer_id, o.take_indv_id, c.state, c.street, c.apt_number

  FROM (Package p NATURAL JOIN "Order" o)

  INNER JOIN Customer c ON o.take_indv_id= c.u_id

  WHERE p.p_id = @p_id;

- SELECT c.v_id FROM Courier c WHERE c.u_id = @u_id;
- INSERT INTO PackageState VALUES('Courier', @date);
- INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);

3.3.2.3. Shipper Home Page

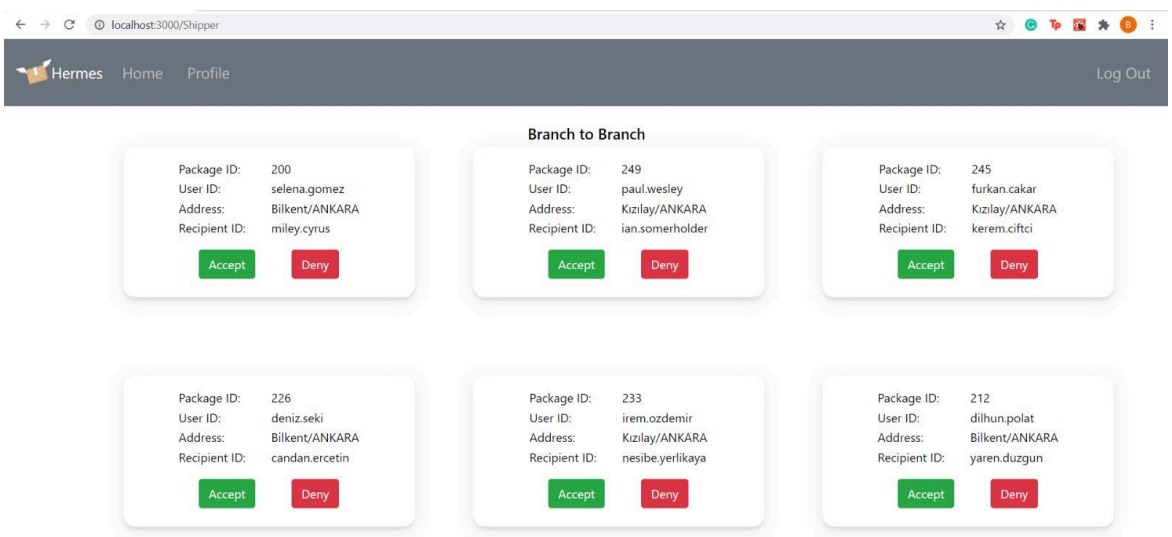


Figure 10

Explanation: Shippers can see the assigned packages that will be delivered from the current branch to the destination branch. They can accept or deny the assigned packages. If they deny the assigned packages, the package will be shown in the package managers' home page to assign the package again in ***Figure 7.***

Needed SQL Statements:

- SELECT p.p_id, o.send_customer_id, o.take_indv_id, b.address FROM (Package p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b

  WHERE p.p_id = @p_id;

- SELECT s.v_id FROM Shipper s WHERE s.u_id = @u_id;
- INSERT INTO PackageState VALUES('Shipper', @date);
- INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);

3.4. Submitting Packages For Customers

3.4.1. Submit Package to Courier

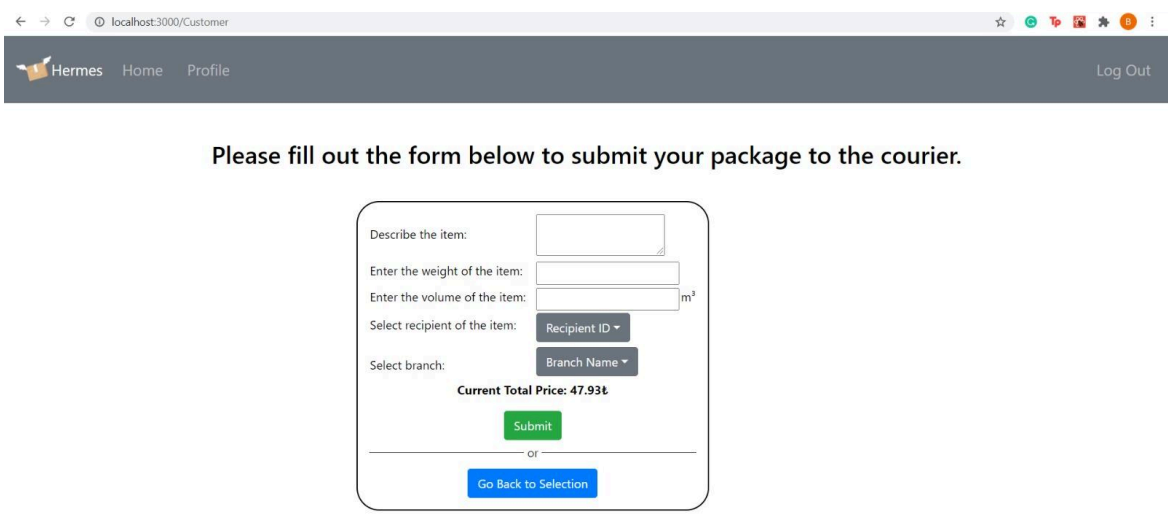


Figure 11

Explanation: Customers can submit packages to couriers by using this page. They will enter the necessary information of the package such as description of the package, weight, volume. They can select a recipient among a list of possible customers. According to this information, they will be able to see the calculated price for the package. They also need to select the closest branch to them to call

the courier. Then they will submit it to the courier. An available courier from the selected branch will be automatically assigned by the system to take the package from the customer.

Inputs: @description, @weight, @volume, @recipentid, @branch

Needed SQL Statements:

- INSERT INTO "Order" VALUES(@take_indv_id , @send_customer_id, @price, @rate);
- INSERT INTO package VALUES(@o_id , @weight, @volume);
- SELECT b."name" FROM branch b;
- SELECT c."name" FROM customer c;

3.4.2. Submit Package In Person

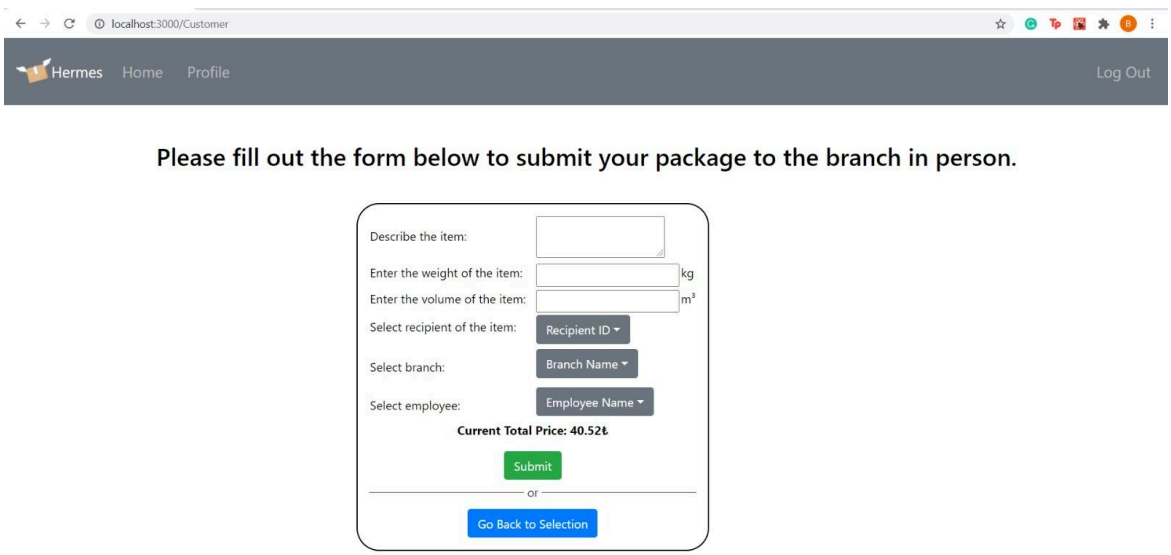


Figure 12

Explanation: Customers can submit packages in person by using this page. They will enter the necessary information of the package such as description of the package, weight, volume. They can select a recipient among a list of possible customers.. They also need to select the closest branch to them and one of the employees in that branch. Then they are able to calculate the price of the package and submit it to the package manager.

Inputs: @description, @weight, @volume, @recipentid, @branch

Needed SQL Statements:

- INSERT INTO "Order" VALUES(@take_indv_id , @send_customer_id, @price, @rate);
- INSERT INTO package VALUES(@o_id , @weight, @volume);
- SELECT b."name" FROM branch b;
- SELECT pm."name" , pm.u_id FROM packagemanager pm WHERE b_id = @b_id;
- SELECT c."name" FROM customer c;

3.5. Profile Page

3.5.1. Customer Profile Page

3.5.1.1. Individual Profile Page

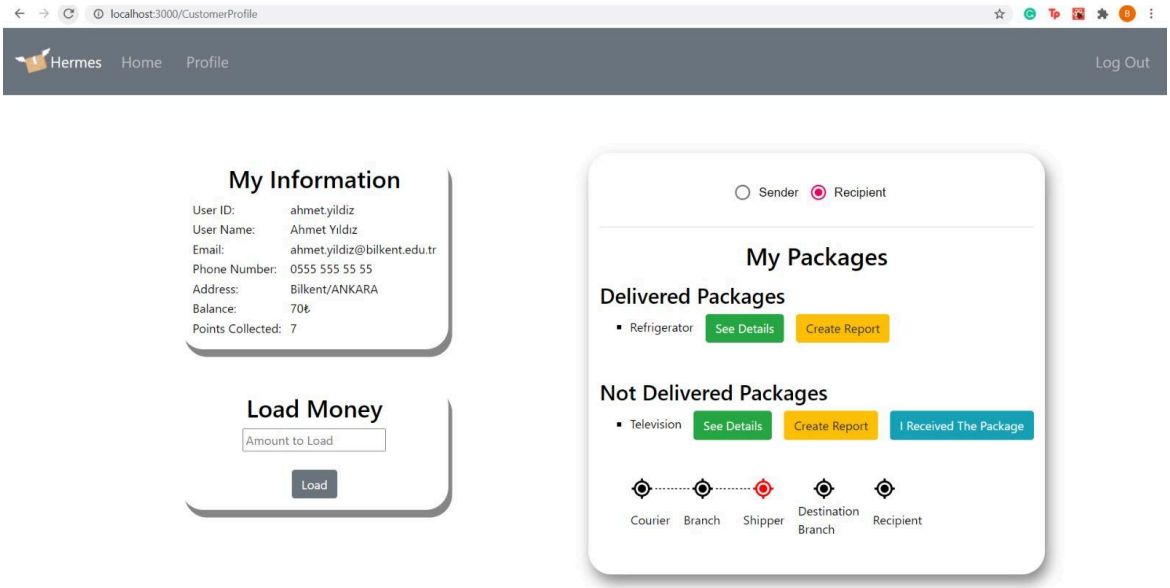


Figure 13

Explanation: Individual customers can see their personal information in their profile pages. They can see the list of delivered and not delivered packages. Besides that, they can see the status of not delivered packages such as, on branch, on courier etc. They are able to load money to their card by using the load button. They can see details of the packages by clicking the "See Details" button as in ***Figure 14.*** They also can create malformed reports for delivered packages as in ***Figure 20.*** Besides, they

can create lost reports for undelivered packages as in ***Figure 21.*** At the final step, they will use the button “I Received The Package” as shown in ***Figure 15***, if they successfully take the package. This button will be disabled until the courier takes the package in order to send it to the customer.

Inputs: @addedmoney

Needed SQL Statements:

–For Customer Info

- SELECT i.u_id , i."name", i.email , i.phone , i.street , i.state , i.zip ,i.city, c".money"

  FROM Individual i NATURAL JOIN customercard c

  WHERE i.u_id = @u_id);

--For Delivered Packages for Sender

- (SELECT p1.item_dscrptn

  FROM "Order" o NATURAL JOIN SecurelyDeliveredPackages p1

  WHERE o.send_customer_id = @send_customer_id

  UNION

  SELECT p2.item_dscrptn

  FROM "Order" o NATURAL JOIN MalformedDeliveredPackage p2

  WHERE o.send_customer_id = @send_customer_id);

--For Non-delivered Packages for Sender

- (SELECT p1.item_dscrptn

  FROM "Order" o NATURAL JOIN LostPackages p1

  WHERE o.send_customer_id = @send_customer_id

  UNION

  SELECT p2.item_dscrptn

  FROM "Order" o NATURAL JOIN HoldoutPackages p2

WHERE o.send_customer_id = @send_customer_id);

--For Delivered Packages for Recipient

(SELECT p1.item_dscrptn

FROM "Order" o NATURAL JOIN SecurelyDeliveredPackages p1

WHERE o.take_indv_id = @take_indv_id

UNION

SELECT p2.item_dscrptn

FROM "Order" o natural join MalformedDeliveredPackage p2

WHERE o.take_indv_id = @take_indv_id);

--For Non-delivered Packages for Recipient

- (SELECT p1.item_dscrptn

  FROM "Order" o NATURAL JOIN LostPackages p1

  WHERE o.take_indv_id = @take_indv_id

  UNION

  SELECT p2.item_dscrptn

  FROM "Order" o NATURAL JOIN HoldoutPackages p2

  WHERE o.take_indv_id = @take_indv_id);

–For load money

- UPDATE CustomerCard SET "money" = “money”+@addedmoney
  WHERE u_id= @u_id;

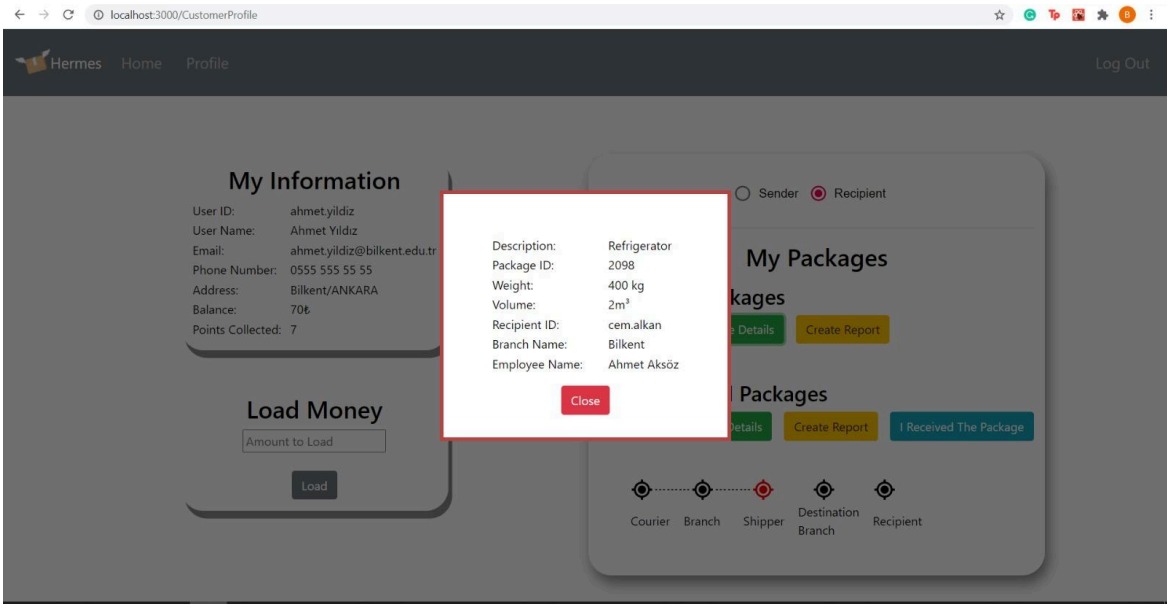


Figure 14

Needed SQL Statements:

- SELECT p.p_id, p.item_dscrptn, p.weight, p.volume, o.take_indv_id, b."name"

  FROM (Package p NATURAL JOIN "Order" o)

  INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b WHERE p.p_id = @p_id;

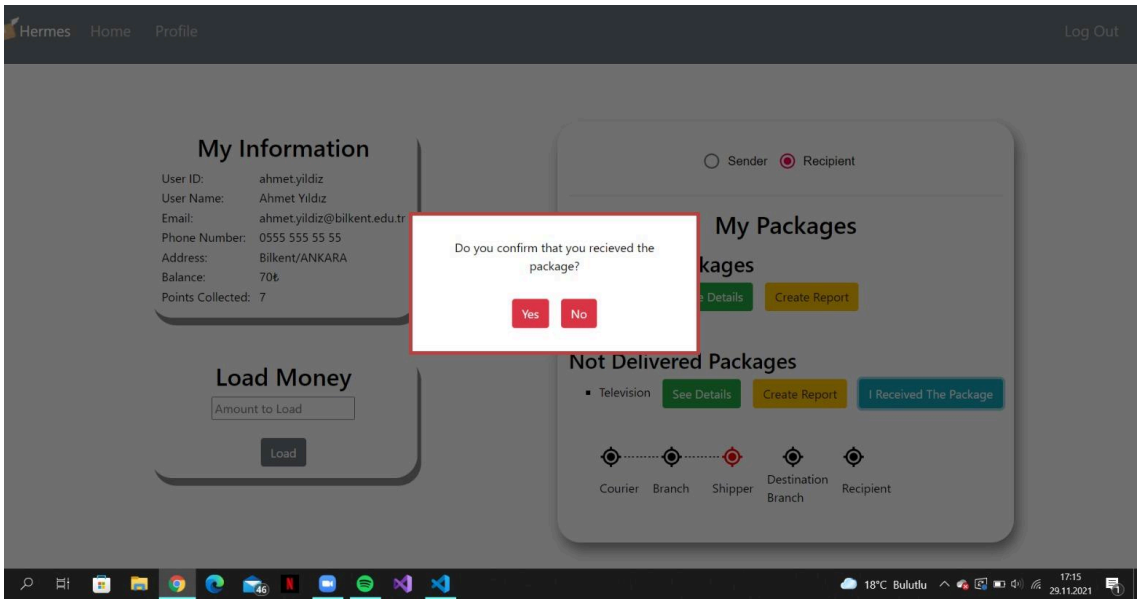


Figure 15

Needed SQL Statements:

INSERT INTO PackageState VALUES('Recipient', @date);

INSERT INTO Pac_State VALUES(@ps_id, @p_id, @v_id);

SELECT * FROM Package p WHERE p.p_id = @p_id;

INSERT INTO SecurelyDelivered VALUES(@p_id, @o_id, @weight, @item_dscrptn, @volume);

3.5.1.2. Corporate Profile Page

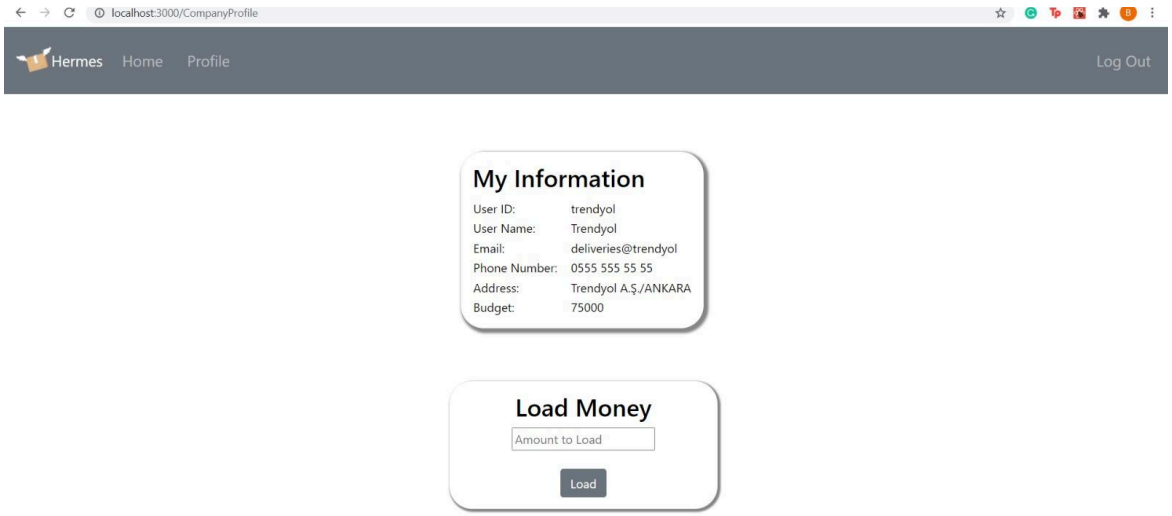


Figure 16

Explanation: Corporate customers can see their institutional information in their profile pages. They are able to load money to budget by using the load button.

Inputs: @addedmoney

Needed SQL Statements:

- SELECT * FROM corporate c WHERE c.u_id = @u_id;
- UPDATE corporate SET budget = budget + @addedmoney WHERE u_id= @u_id;

3.5.2. Employee Profile Page

3.5.2.1. Courier Profile Page

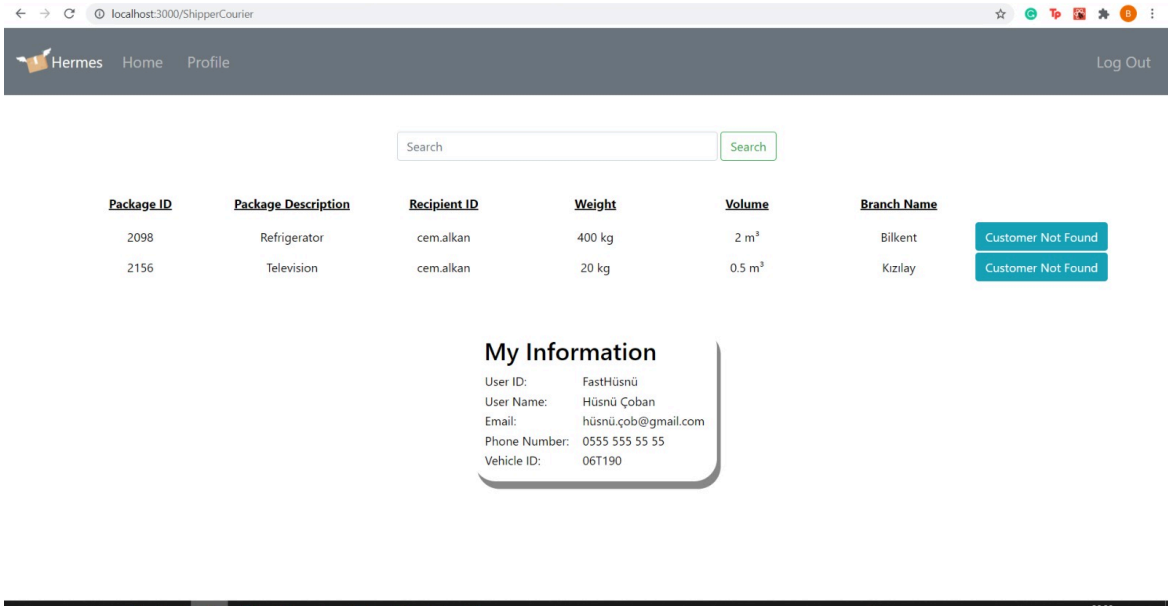


Figure 17

Explanation: Couriers can see their information in their profile pages. Besides, couriers can see the packages they accepted to deliver here. When they try to deliver the package to a customer and cannot reach them, the courier presses the customer not found the button. Then, the package state changes to the holdout and the package takes place in the holdout packages table.

Needed SQL Statements:

- SELECT * FROM courier c WHERE c.u_id = @u_id;
- SELECT pk.p_id FROM packagestate pk WHERE pk.name = ‘Courier’;
- SELECT p.p_id, p.item_dscrptn ,p.volume, p.weight, o.take_indv_id, o.o_id, b.name FROM (Package p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b

  WHERE p.p_id = @p_id;

- INSERT INTO HoldoutPackages VALUES (@p_id, @o_id, @weight, @item_dscrptn, @volume);
- UPDATE packagestate ps SET ps. “Name” = ‘Holdout’ WHERE ps.p_id = @p_id

3.5.2.2. Package Manager Profile Page

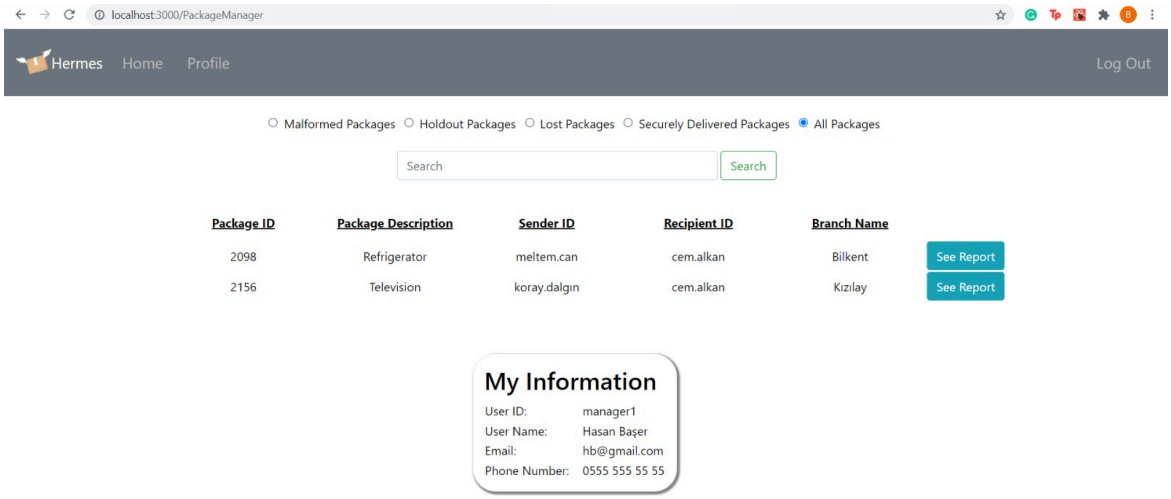


Figure 18

Explanation: Package Managers can see their information in their profile pages. Besides these, they can see malformed packages, holdout packages, lost packages,securely delivered packages and all packages by using radio buttons at the top of the page. They also can search packages by using the search bar. They can see reports by clicking the "See Report" button.

Needed SQL Statements:

- SELECT * FROM PackageManager pm where pm.u_id = @u_id;

- SELECT p.p_id, p.item_dscrptn, o.take_indv_id, b."name"

  FROM (Package p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b;

- SELECT sp.p_id, sp.item_dscrptn, o.take_indv_id, b."name"

  FROM (SecurelyDeliveredPackage sp NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b;

- SELECT lp.p_id, lp.item_dscrptn, o.take_indv_id, b."name"

  FROM (LostPackages lp NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b;

- SELECT hp.p_id, hp.item_dscrptn, o.take_indv_id, b."name"

  FROM (HouldoutPackages hp NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b;

- SELECT hp.p_id, hp.item_dscrptn, o.take_indv_id, b."name"

  FROM (MalformedPackages p NATURAL JOIN "Order" o) INNER JOIN "User" u ON o.send_customer_id= u.u_id NATURAL JOIN PackageManager pm NATURAL JOIN Branch b;

3.5.2.3. Shipper Profile Page

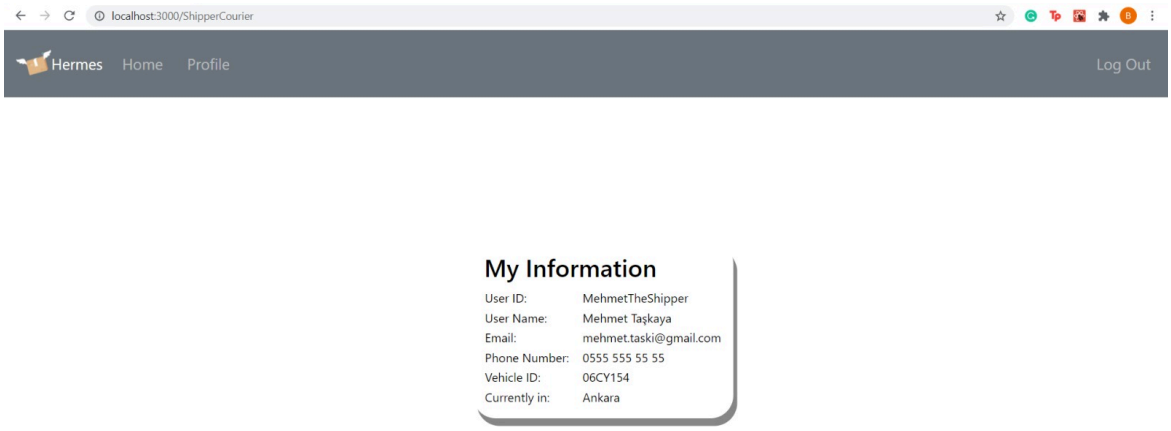


Figure 19

Explanation: Shippers can see their information in their profile pages.

Needed SQL Statements:

- SELECT * FROM shipper s WHERE s.u_id = @u_id;

3.6. Reports

3.6.1. File a Report Customer

3.6.1.1. Malformed Report

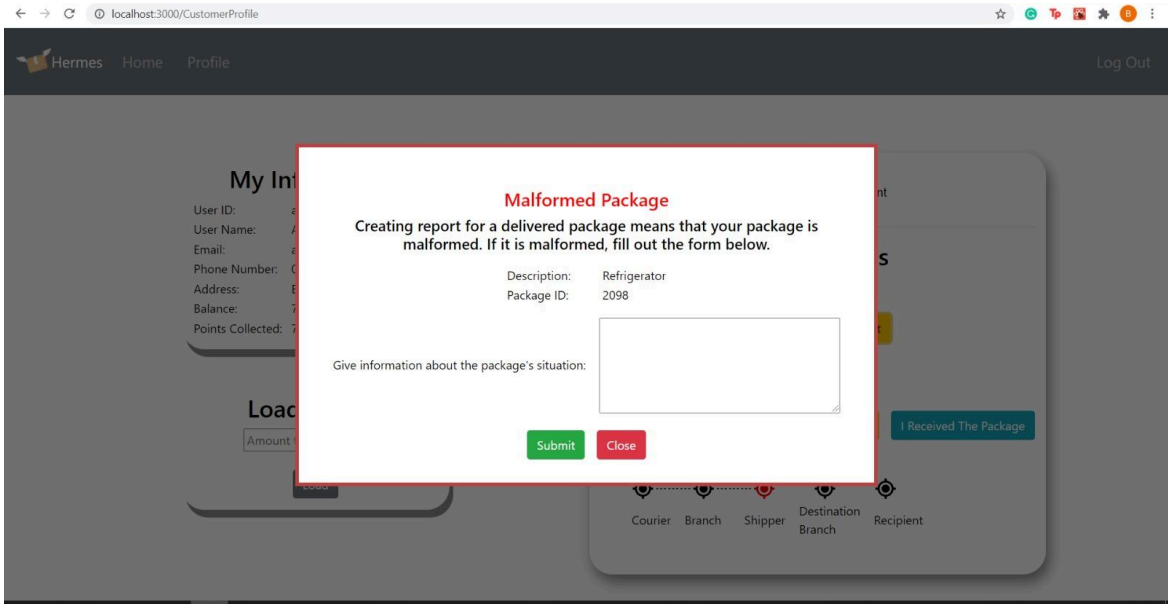


Figure 20

Explanation: Individual customers can report if their delivered package is malformed. They will fill the input text to describe the malformed package's situation.

Inputs: @text

Needed SQL Statements:

--For Package Details

- SELECT p.p_id, p.item_dscrptn FROM MalformedDeliveredPackage p WHERE p.p_id = @p_id **;**

--For Report Creation

- INSERT INTO Report VALUES(@u_id, @p_id, @explanation, null, @date);

3.6.1.2. Lost Report

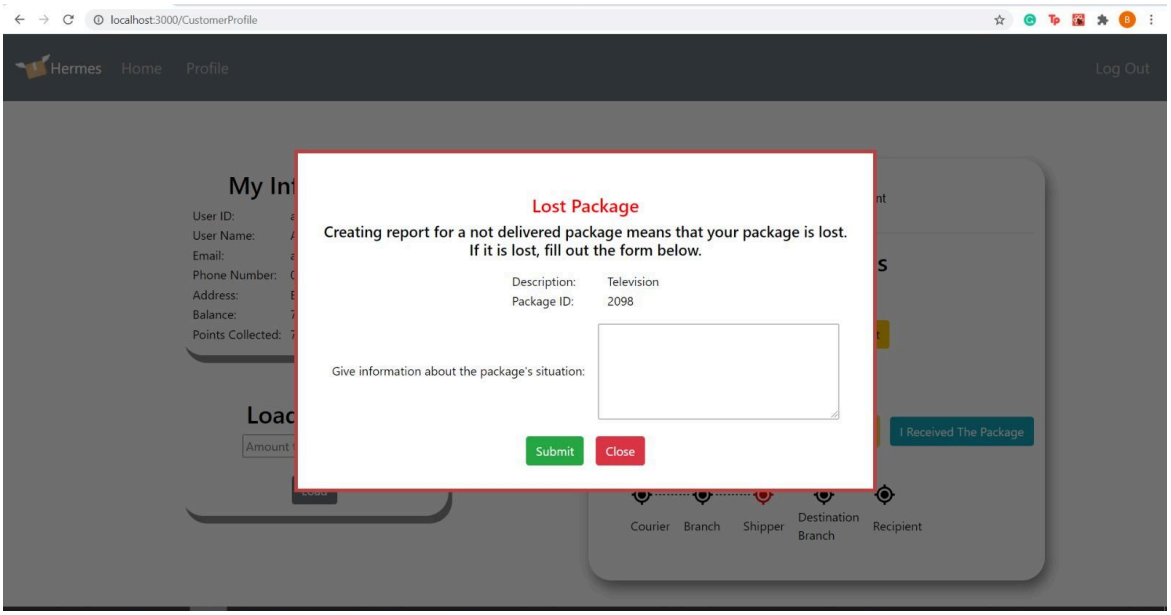


Figure 21

Explanation: Individual customers can report if their undelivered package is lost. They will fill the input text to describe the package's situation.

Inputs: @text

Needed SQL Statements:

--For Package Details

- SELECT p.p_id, p.item_dscrptn FROM LostPackages p WHERE p.p_id = @p_id;

--For Report Creation

- INSERT INTO Report VALUES(@u_id, @p_id, @explanation, null, @date);

4. Implementation Plan

Our projects will be implemented by several programs. The front-end of the website will be implemented using React.js. For the back-end and API handling Node.js will be used. As the query language PostgreSQL will be used. We have implemented user interfaces using React.js and the photos in figures above are screen captures of the program implementation. We have used VisualStudio Code for implementation. For the database management system, pgAdmin will be used in order to keep track of the records.