CS 353 Project Design Report

Group 8

Maintenance Data Management System - FixItUp

Web Page Link: https://github.com/Seftali/CS-353-Project-Reports

Berk Ataç - 21200623 - Section 1

Eren Ayture - 21200559 - Section 1

Umut Balkan - 21401911 - Section 1

Derviş Mehmed Barutcu - 21302589 - Section 1

Table of Content

1. Revised E\R Model

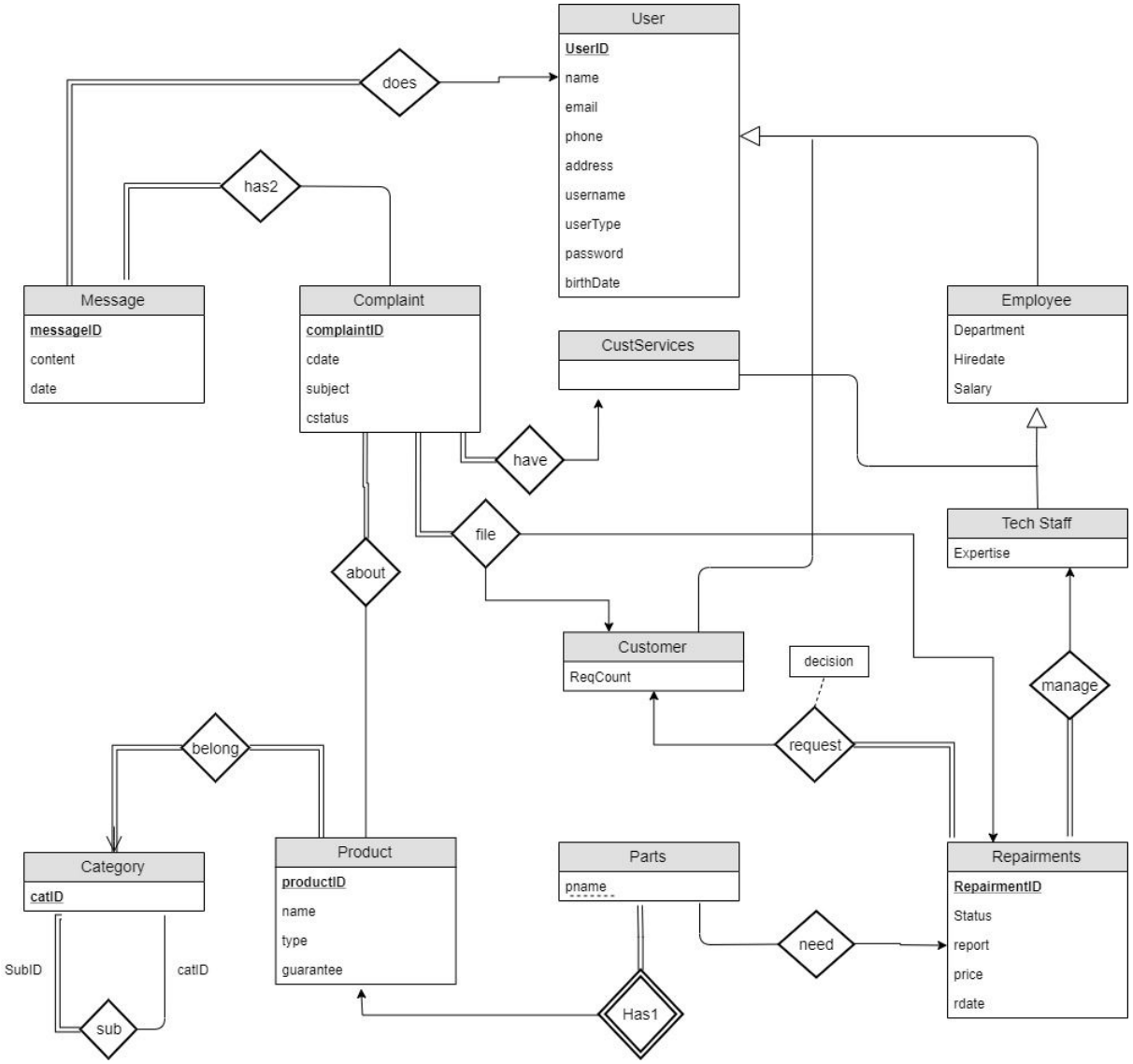

Figure 1: E/R Diagram

According to the feedback given by the assistant, we revised our E/R model as follows:

- We created a table called User which Employee and Customer tables will inherit attributes from.
- We removed our weak entity called Conversation. We made a new table called Message and it is not a weak entity anymore. We related it with our new table User. “does” relation.

- We increased our overall attribute number in our tables and relocated some of them.
- We corrected cardinality mistakes in our diagram. Relation between Category and Product, relation between Complaints and Product, relation between Complaint and CustServices.
- We removed relation between Customer and Product.
- We created a ternary relationship between Complaint, Customer and Repairments.
- We removed relation called Repair between Product and Repairments, instead we created a weak entity called Parts. We created relation “Has1” and “need” from Parts to Product and Repairments respectively.
- We added decision attribute to our relation called “request”, between Customer and Repairments. Added userType to User table.
- We added a relation called sub under Category table. For sub categories.
- As **a new added function** we added a **campaign system**. Attribute called ReqCount in table Customer is the count of requests customers make. One of every three repairs will be free for a customer (Only if all three are paid repairs and not under guarantee).

2. Relational Schemas

2.1. User

Relational Model:

User(<u>userID</u>, username, password, email, birthDate, name, phone, address)

Functional Dependencies:

userID -> username password email birthDate name phone address

Candidate Keys:

{(userID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE User(
     userID    CHAR(16) NOT NULL AUTO_INCREMENT,
     username  VARCHAR(255) NOT NULL UNIQUE,
     password  VARCHAR(255) NOT NULL,
     email     VARCHAR(255) NOT NULL UNIQUE,
     birthDate DATE,
     name      VARCHAR(255) NOT NULL,
     phone     INT(10) UNSIGNED NOT NULL,
     address   VARCHAR(255),
     PRIMARY KEY (userID)
)
```

2.2. Customer

Relational Model:

Customer(<u>custID</u>, iban, creditNum)

Functional Dependencies:

custID -> iban creditNum

Candidate Keys:

{(custID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Customer(
     custID    CHAR(16) NOT NULL,
     iban      CHAR(34) NOT NULL,
     creditNum INT(16) NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (custID)     REFERENCES
     User(userID)   ON   UPDATE   CASCADE   ON   DELETE
     CASCADE,
     PRIMARY KEY (custID)
)
```

2.3. Employee

Relational Model:

Employee(empID, department, salary)

Functional Dependencies:

empID -> department salary

Candidate Keys:

{(empID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Employee(
     empID     CHAR(16) NOT NULL,
     department VARCHAR(255),
     hireDate  DATE NOT NULL,
     salary    INT UNSIGNED NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (empID)     REFERENCES
User(userID) ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY (empID)
)
```

2.4. CustServices

Relational Model:

CustServices(empID)

Functional Dependencies:

empID -> empID

Candidate Keys:

{(empID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE CustServices(
     empID     CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (empID)     REFERENCES
Employee(empID) ON UPDATE CASCADE ON DELETE CASCADE,
     PRIMARY KEY (empID)
)
```

2.5. TechStaff

Relational Model:

TechStaff(empID,expertise)

Functional Dependencies:

empID -> expertise

Candidate Keys:

{(empID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE TechStaff(
     empID     CHAR(16) NOT NULL,
     expertise VARCHAR(64) NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (empID)     REFERENCES
Employee(empID) ON UPDATE CASCADE ON DELETE CASCADE,
     PRIMARY KEY (empID)
)
```

2.6. Message

Relational Model:

Message(<u>messageID</u>, content, date)

Functional Dependencies:

messageID -> content date

Candidate Keys:

{(messageID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Message(
     messageID CHAR(16) NOT NULL AUTO_INCREMENT,
     content   TEXT,
     date      DATE,
     PRIMARY KEY ( messageID )
)
```

2.7. does

Relational Model:

does(messageID,userID)

Functional Dependencies:

messageID userID -> messageID userID

Candidate Keys:

{(messageID,userID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE does(
     messageID CHAR(16) NOT NULL,
     userID    CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN    KEY    (messageID)    REFERENCES
     Message(messageID)  ON UPDATE CASCADE ON DELETE
     CASCADE,
          FOREIGN     KEY     (userID)     REFERENCES
     User(userID)   ON   UPDATE   CASCADE   ON   DELETE
     CASCADE,
     PRIMARY KEY (messageID,userID)
)
```

2.8. has

Relational Model:

has(messageID,complaintID)

Functional Dependencies:

messageID complaintID -> messageID complaintID

Candidate Keys:

{(messageID,complaintID)}

Foreign Keys:

Normal Form:

Table Definition:

```
CREATE TABLE has(
     messageID      CHAR(16) NOT NULL,
     complaintID    CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN    KEY    (messageID)    REFERENCES
     Message(messageID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
          FOREIGN    KEY    (comlaintID)    REFERENCES
     Complaint(comlaintID)   ON   UPDATE   CASCADE   ON
     DELETE CASCADE,
     PRIMARY KEY (messageID,comlaintID)
)
```

2.9. Complaint

Relational Model:

Complaint(complaintID,cdate,csubject,cstatus)

Functional Dependencies:

complaintID` -> cdate csubject cstatus

Candidate Keys:

{(complaintID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Complaint(
     complaintID CHAR(16) NOT NULL AUTO_INCREMENT,
     cdate     DATE,
     csubject  VARCHAR(64),
     cstatus   VARCHAR(255),
     PRIMARY KEY (complaintID)
)
```

2.10. file

Relational Model:

file(complaintID,custID,repairmentID)

Functional Dependencies:

complaintID custID repairmentID -> complaintID custID repairmentID

Candidate Keys:

{(complaintID, custID, repairmentID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE file(
     complaintID CHAR(16) NOT NULL,
     custID CHAR(16) NOT NULL,
     repairmentID CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN KEY (complaintID) REFERENCES
     Complaint(complaintID) ON UPDATE CASCADE ON
     DELETE CASCADE,
          FOREIGN KEY (repairmentID) REFERENCES
     Repairment(repairmentID) ON UPDATE CASCADE ON
     DELETE CASCADE,
          FOREIGN KEY (custID) REFERENCES
     Customer(custID) ON UPDATE CASCADE ON DELETE
     CASCADE,
     PRIMARY KEY (complaintID,custID, repairmentID)
)
```

2.11. about

Relational Model:

about(complaintID,productID)

Functional Dependencies:

complaintID productID -> complaintID productID

Candidate Keys:

{(complaintID,productID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE about(
     complaintID    CHAR(16) NOT NULL,
     productID      CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN   KEY   (complaintID)   REFERENCES
     Complaint(complaintID)  ON  UPDATE  CASCADE  ON
     DELETE CASCADE,
          FOREIGN    KEY    (productID)    REFERENCES
     Product(productID) ON UPDATE CASCADE ON DELETE
     CASCADE,
     PRIMARY KEY (complaintID,productID)
)
```

2.12. have

Relational Model:

have(complaintID,empID)

Functional Dependencies:

complaintID empID -> complaintID empID

Candidate Keys:

{(complaintID, empID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE have(
     complaintID    CHAR(16) NOT NULL,
     empID          CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN   KEY   (complaintID)   REFERENCES
     Complaint(complaintID)  ON  UPDATE  CASCADE  ON
     DELETE CASCADE,
          FOREIGN     KEY     (empID)     REFERENCES
     CustServices(empID) ON UPDATE CASCADE ON DELETE
     CASCADE,
     PRIMARY KEY (complaintID,empID)
)
```

2.13. Product

Relational Model:

Product(productID,name,type,guarantee,stock)

Functional Dependencies:

productID -> name, type, guarantee, stock

Candidate Keys:

{(productID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Product(
     productID CHAR(16) NOT NULL AUTO_INCREMENT,
     name      VARCHAR(64),
     type      VARCHAR(64),
     guarantee BOOLEAN,
     stock     SMALLINT UNSIGNED,
     PRIMARY KEY (productID)
)
```

2.14. Category

Relational Model:

Category(catID,catName)

Functional Dependencies:

catID -> catName

Candidate Keys:

{(catID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Category(
     catID     CHAR(16) NOT NULL AUTO_INCREMENT,
     catName   VARCHAR(255),
     PRIMARY KEY (catID)
)
```

2.15. sub

Relational Model:

sub(catID,subID)

Functional Dependencies:

catID subID -> catID subID

Candidate Keys:

{(catID, subID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE sub(
     catID CHAR(16) NOT NULL,
     subID CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (catID)     REFERENCES
     Category(catID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
          FOREIGN     KEY     (subID)     REFERENCES
     Category(catID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
     PRIMARY KEY (catID,subID)
)
```

2.16. belong

Relational Model:

belong(catID, productID)

Functional Dependencies:

catID productID -> catID productID

Candidate Keys:

{(catID,productID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE belong(
     catID     CHAR(16) NOT NULL,
     productID CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN     KEY     (catID)     REFERENCES
     Category(catID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
          FOREIGN    KEY    (productID)    REFERENCES
     Product(productID)  ON UPDATE CASCADE ON DELETE
     CASCADE,
     PRIMARY KEY (catID,productID)
)
```

2.17. Part

Relational Model:

Part(name,productID)

Functional Dependencies:

name productID -> name productID

Candidate Keys:

{(name, productID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Part(
     name      VARCHAR(64) NOT NULL,
     productID CHAR(16) NOT NULL AUTO_INCREMENT,
     CONSTRAINT
          FOREIGN    KEY    (productID)    REFERENCES
     Product(productID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
     PRIMARY KEY (name,productID)
)
```

2.18. need

Relational Model:

need(name,repairmentID)

Functional Dependencies:

name repairmentID -> name repairmentID

Candidate Keys:

{(name, repairmentID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE need(
     name      VARCHAR(255) NOT NULL,
     repairmentID   CHAR(12) NOT NULL,
     CONSTRAINT
          FOREIGN    KEY    (part_name)    REFERENCES
     Part(name) ON UPDATE CASCADE ON DELETE CASCADE,
          FOREIGN   KEY   (repairmentID)   REFERENCES
     Repairment(repairmentID)  ON  UPDATE  CASCADE  ON
     DELETE CASCADE,
     PRIMARY KEY (name,repairmentID)
)
```

2.19. Repairment

Relational Model:

Repairment(repairmentID,status,guarantee,rdate)

Functional Dependencies:

repairmentID -> status guarantee rdate

Candidate Keys:

{(repairmentID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Repairment(
     repairmentID CHAR(16) NOT NULL,
     status    CHAR(255),
     guarantee DATE,
     rdate     DATE,
     PRIMARY KEY (repairmentID)
)
```

2.20. manage

Relational Model:

manage(repairmentID,empID)

Functional Dependencies:

repairmentID empID -> repairmentID empID

Candidate Keys:

{(repairmentID, empID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE manage(
     repairmentID CHAR(16) NOT NULL,
     empID     CHAR(16) NOT NULL,
     CONSTRAINT
          FOREIGN   KEY   (repairmentID)   REFERENCES
     Repairment(repairmentID)  ON  UPDATE  CASCADE  ON
     DELETE CASCADE,
          FOREIGN     KEY     (empID)     REFERENCES
     TechStaff(empID)  ON  UPDATE  CASCADE  ON  DELETE
     CASCADE,
     PRIMARY KEY (repairmentID,empID)
)
```

2.21. request

Relational Model:

request(<u>custID,repairmentID</u>,decision)

Functional Dependencies:

custID repairmentID -> decision

Candidate Keys:

{(custID, repairmentID)}

Foreign Keys:

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE request(
     custID    CHAR(16) NOT NULL,
     repairmentID CHAR(16) NOT NULL,
     decision  VARCHAR(255),
     CONSTRAINT
          FOREIGN KEY (custID) REFERENCES
     Customer(custID) ON UPDATE CASCADE ON DELETE
     CASCADE,
          FOREIGN KEY (repairmentID) REFERENCES
     Repairment(repairmentID) ON UPDATE CASCADE ON
     DELETE CASCADE,
     PRIMARY KEY (custID,repairmentID)
)
```

3. Functional Dependencies and Normalization

We stated the functional dependencies of the tables. While designing the tables, we have designed them with BCNF as you can see in the Figure 1. Therefore, we did not perform any decomposition or normalization on the tables.

4. Functional Components

4.1. Use Case: Customer

Figure 2: Use of the customer

4.1.1. Change Username/Password

Use Case: Customer can change his/her username/password.

Primary Actor: Customer, Tech Staff, Customer Service

Stakeholders and interests:

- User who desires to change username or password.

Pre-conditions:

- User should be logged in.
- User should have pressed the “change username” or “change password” button.
- User should type in new username or password

Successful Scenario Event Flow:

1. User presses “change username” or “change password” button.
2. User enters username, password.
3. User clicks the OK button.
4. User sees the main page of the system.

Unsuccessful Scenario Event Flow:

1. User presses the button.
2. User enters username, password.
3. User clicks the OK button.
4. User receives an error message that indicate that this is an invalid username(not unique)

4.1.2. View Repair Panel

Use Case: Customer can view Repair Panel.

Primary Actor: Customer

Stakeholders and interests:

- Customer who desires to see Repairs.

Pre-conditions:

- User should be logged in.
- User should be on main screen.

Successful Scenario Event Flow:

1. User should be on main screen.

4.1.3. View Reports Panel

Use Case: Customer can view Reports Panel.

Primary Actor: Customer

Stakeholders and interests:

- Customer who desires to see reports.

Pre-conditions:

- User should be logged in.
- User should have pressed the Reports button.

Successful Scenario Event Flow:

1. User presses the Reports button.

4.1.4. View Complaints Panel

Use Case: Customer can view Repair Panel.

Primary Actor: Customer

Stakeholders and interests:

- Customer who desires to see complaints.

Pre-conditions:

- User should be logged in.
- User should be on main screen.

Successful Scenario Event Flow:

1. User should be on main screen.

4.1.5. File Request For Repair

Use Case: Customer can file a request for repair.

Primary Actor: Customer

Stakeholders and interests:

- Customer, Tech Staff.

Pre-conditions:

- User should be logged in.
- User should fill text boxes under Request Repairment.(Enter product name, enter type of product, enter guarantee, enter problem.)
- User should press the Send button.

Successful Scenario Event Flow:

1. User enters product name.

2. User enters type of product.
3. User enters guarantee date.
4. User describes problem.
5. User presses the Send button.

Unsuccessful Scenario Event Flow:

1. If any of the text fields are empty user sees an error message.

4.1.6. Decide

Use Case: Customer can Decide what to do with repair before and after.

Primary Actor: Customer

Stakeholders and interests:

- Customer who desires to see reports.
- Tech Staff.

Pre-conditions:

- User should be logged in.
- User should have pressed the Reports button.
- User should select a report.
- User should press Return, Renew or Approve button.

Successful Scenario Event Flow:

1. User presses the Reports button.
2. User selects a report.
3. User Presses a button.

4.1.7. Pay For Repair

Use Case: Customer can Pay for repair.

Primary Actor: Customer

Stakeholders and interests:

- Customer

Pre-conditions:

- User should be logged in.
- User should have pressed the Reports button.
- User should select a report.
- User should press Approve button after the second report.
- Product should have a valid guarantee.

Successful Scenario Event Flow:

1. User presses the Reports button.
2. User selects a report.
3. User Presses the approve button.
4. User fills in credit card info.
5. User presses Pay button.

Unsuccessful Scenario Event Flow:

1. If any of the credit card info is empty.

4.1.8. Enter Credit Card

This use case is described above (4.1.7)

4.1.9. File Complaint

Use Case: Customer can file a complaint.

Primary Actor: Customer

Stakeholders and interests:

- Customer
- Tech Staff
- Customer Services

Pre-conditions:

- User should be logged in.
- User should be on main screen.
- Repair should be finished.

Successful Scenario Event Flow:

1. User should be on main screen.
2. User should press complaint button in Repairment History.
3. User should fill in the complaint section in the main screen.
4. User should press the Send button.

Unsuccessful Scenario Event Flow:

1. If user leaves complaint section empty.

4.1.10. Message

Use Case: Customer can message with Customer Services

Primary Actor: Customer

Stakeholders and interests:

- Customer
- Customer Services

Pre-conditions:

- User should be logged in.
- User should be on main screen.
- User should have filed a complaint.

Successful Scenario Event Flow:

1. User should be on main screen.
2. User should type messages in chat window.
3. User should press send.

4.2. Use Case Customer Service

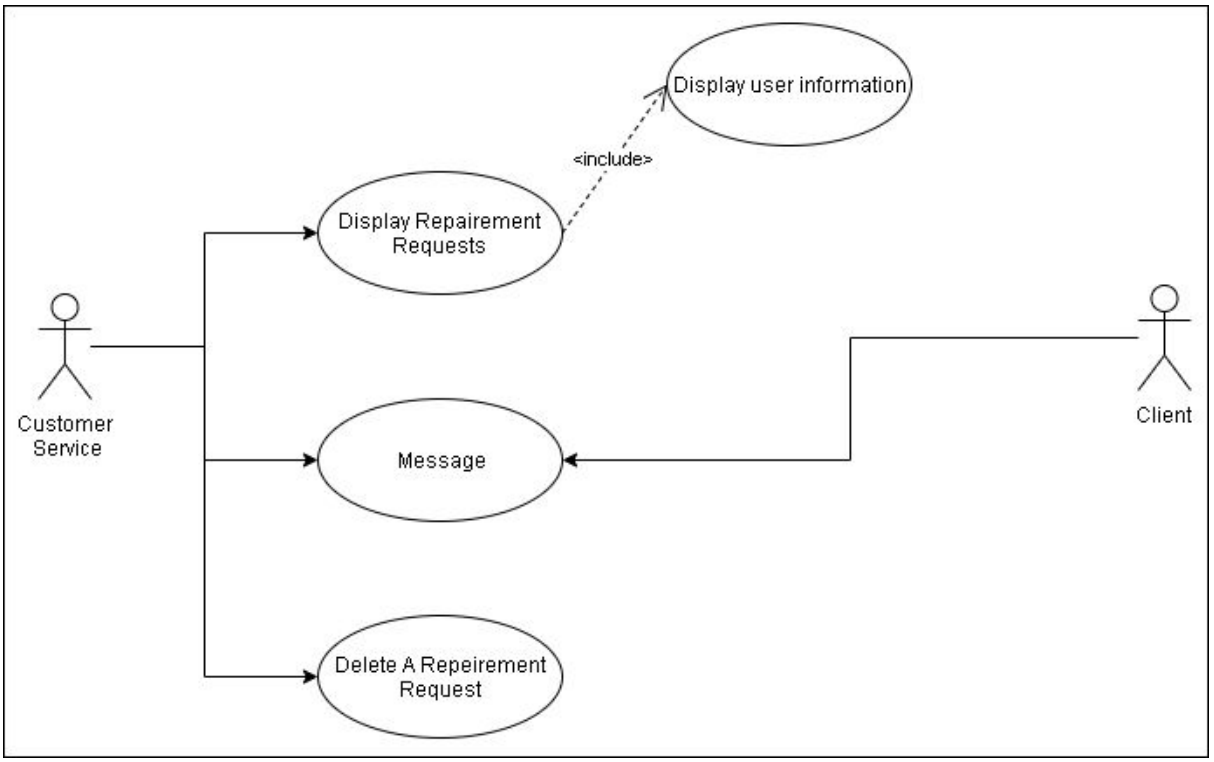

Figure 3: Use Case of the Customer Service

4.2.1. Display Requirement Requests

Use Case: Customer Service can display repairment requests

Primary Actor: Customer Service

Stakeholders and interests:

- Customer Service who desires to see repairment requests

Pre-conditions:

- User should be logged in.

Successful Scenario Event Flow:

- If Customer Service page is opened, customer services can see.

Unsuccessful Scenario Event Flow:

- Login is unsuccessful.
- Browser crash related with internet connection

4.2.2. Delete Repairment Request

Use Case: Customer Service can delete a request that is handled without employee

Primary Actor: Customer Service

Stakeholders and interests:

- Customer Service who received a good feedback from client.

Pre-conditions:

- Product must be repaired
- Employee must have sent a report to client.
- Client must approve repairment.
- Customer Service must log in.

Successful Scenario Event Flow:

- If Customer Service page is opened, customer services can view the status of products.
- When Customer enters remove from list, request is not deleted but removed from the list.

Unsuccessful Scenario Event Flow:

4.2.3. Message

Use Case: Customer Service can message with a customer

Primary Actor: Customer Service, Client

Stakeholders and interests: Customer Service, Client

Pre-conditions:

- Customer Service must logged in.
- Customer Service must click “Message” button

Successful Scenario Event Flow:

- When Message is sent to Customer service to Client

Unsuccessful Scenario Event Flow:

- When Message is not sent to Client

4.2.4. Display User Information

Use Case: Customer Service can display user information

Primary Actor: Customer Service, Client

Stakeholders and interests: Customer Service, Client

Pre-conditions:

- Customer Service must logged in.
- Customer Service must click Requirement Request

Successful Scenario Event Flow:

- When Customer Service see’s user profile

Unsuccessful Scenario Event Flow:

- When Customer Service does not see user profile

4.3. Use Case Tech Staff

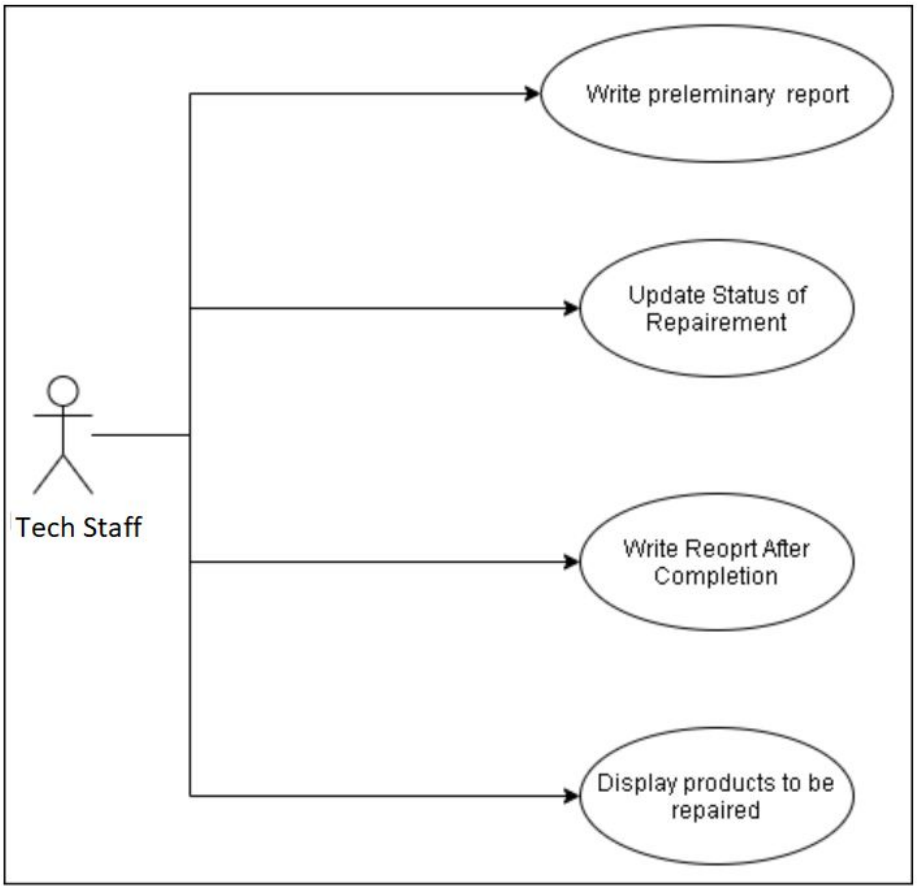

Figure 4: Use Case of the Employee

4.3.1. Write a Preliminary Report

Use Case: Tech staff writes a preliminary report about what will be done about product**.**

Primary Actor: Tech staff

Stakeholders and interests: Tech staff sends report to client

Pre-conditions:

- Tech staff must have logged in.
- Tech staff must see the product.

- Product must be arrived to the customer.

Successful Scenario Event Flow:

- When Tech staff inspects the product and writes a preliminary report to server is successfully stored

Unsuccessful Scenario Event Flow:

- When the product is not arrived
- When the internet crashed and the report is not stored

4.3.2. Update Status of a Report

Use Case: The tech staff updates the status of product like “Product received”, “in repairment”, “Repaired”,”Sending”

Primary Actor: Tech staff

Stakeholders and interests: Tech staff to Client

Pre-conditions:

- When request is made by the user and the customer service is approved.
- When the product is arrived to the customer.
- When the tech staff logged in to the server.

Successful Scenario Event Flow:

- When tech staff updates the combo box.
- When status is updated.

Unsuccessful Scenario Event Flow:

- When internet crashed and status is not updated

4.3.3. Write Report After Completion

Use Case: After the product fixed tech staff can write a report about what is done about the product.

Primary Actor: Tech staff

Stakeholders and interests: Tech staff to Client

Pre-conditions:

- Tech staff must have finalize the status of product

Successful Scenario Event Flow:

- When Tech staff inspects the product and writes a preliminary report to server is successfully stored

Unsuccessful Scenario Event Flow:

- When internet crashed and report is not stored.

4.3.4. Display products to be repaired

Use Case: Tech staff can display the products to be fixed and its problems.

Primary Actor: Tech staff

Stakeholders and interests: Tech staff

Pre-conditions:

- Tech staff must logged in server.

Successful Scenario Event Flow:

- Tech staff must logged in server.

Unsuccessful Scenario Event Flow:

- Tech staff could not log in server.

4.4. Algorithms

4.4.1. Campaign Algorithm

In this site, users can purchase a repair service, if they don't have a guarantee of their product. If the user has a guarantee of repairment requested product, their repairment count will not increase by one. If the product does not have a guarantee, the user can purchase a repairment request of a specific product. If repairment count reaches 3 or multiples of three, application will increase promotion count which is a repairment gift, by one. Then, repairment count will be zero as default. The user will be able to use gift for another repairment request.

4.5. Data Structures

For attributes, utilizing from MySql, numeric types, string, DATE, types will be used.

5. User Interface Design and Corresponding SQL Statements

All SelectedID or Clicked statements will be handled by php functions.

5.1. Login/SignUp page

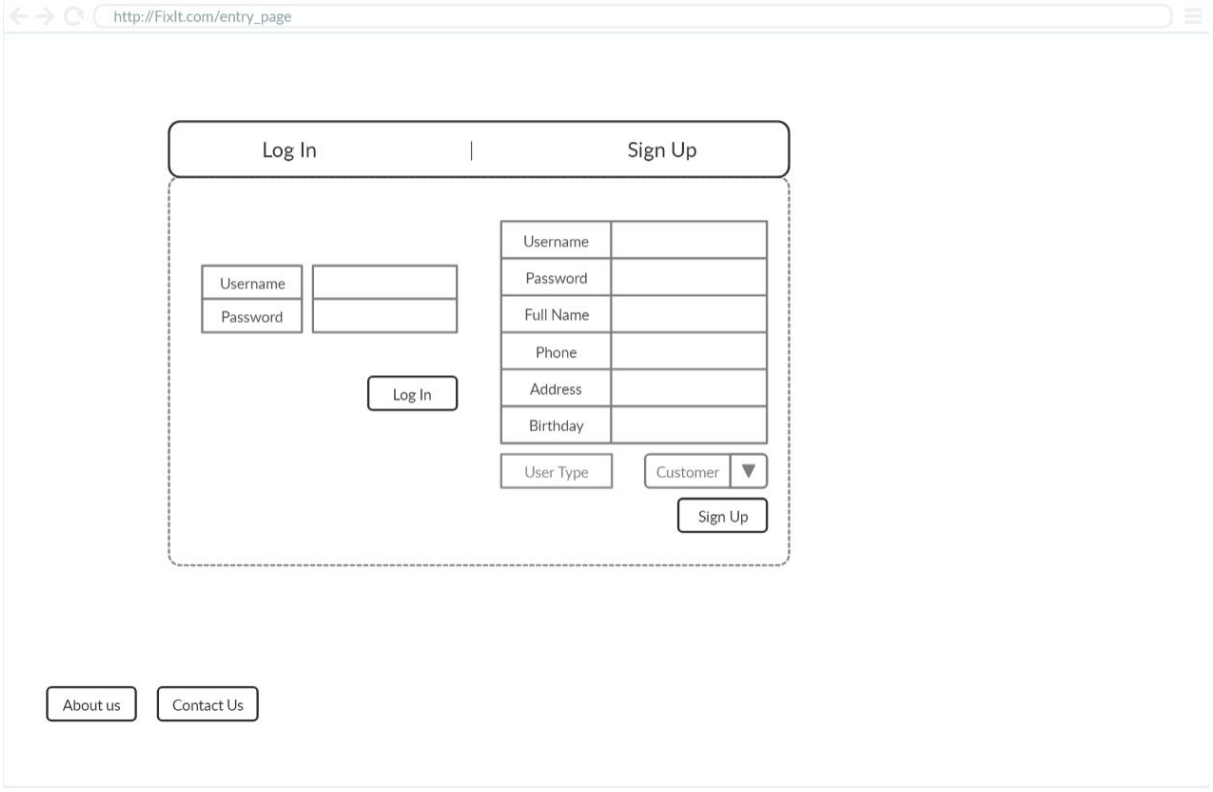

Figure 5: Login/SignUp Page

Sign In:

Insert Into User(name, email, phone, address, username, password, birthDate) **Values** (“Berk Ataç”, “5300406834”, “Ankara Cankaya”, “cBerk”, “passwordStronk1”, 1994-09-28);

Log In:

Select userID,userType

From User

Where username=@username **and** password = @password;

5.2. Main Page: Customer

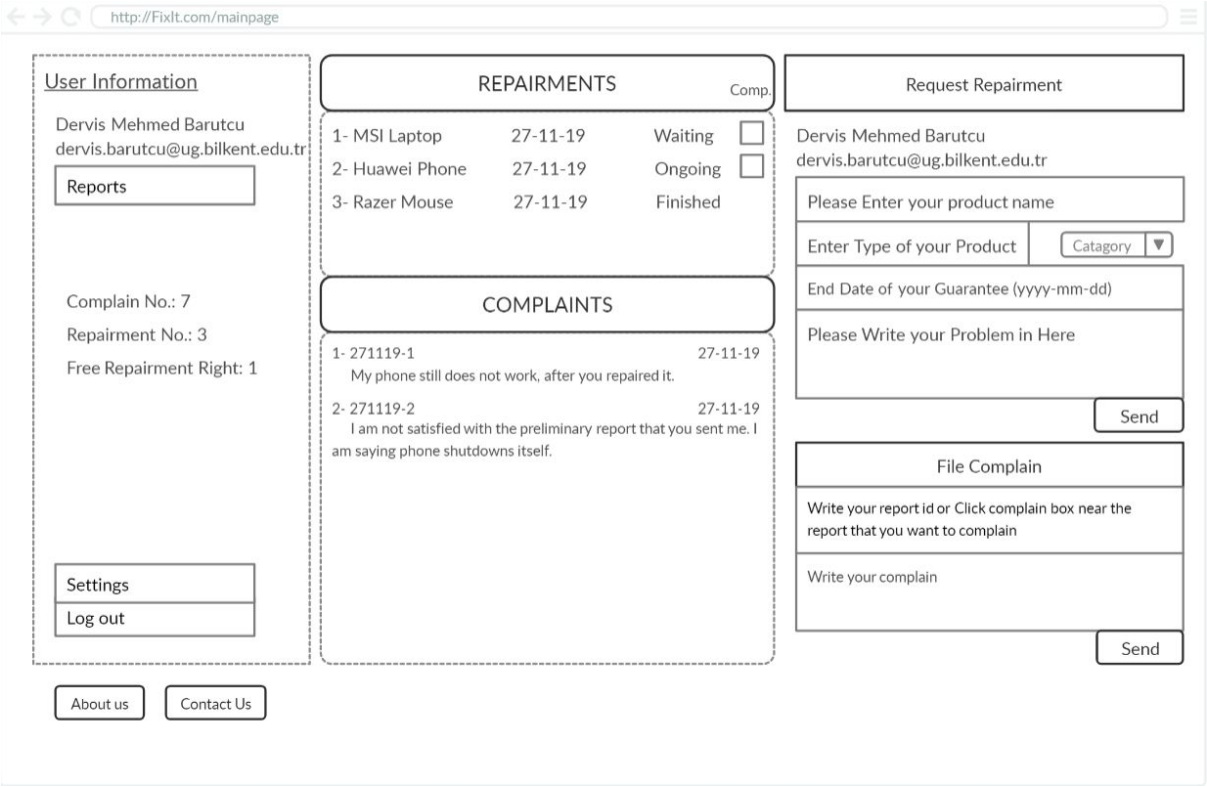

Figure 6: Main Page of the Customer

Repairment list:

Select pname,rdate,Status

From Info_About_Repairments iar

Where iar.custID=@userID;

Complaint list:

Select repairmentID, subject, cdate **From** Info_About_Complaints

Request Repairment:

Insert Into Repairments (Status, report, price, rdate) **Values** (“waiting”, NULL, “100”, 2019-11-27);

Insert Into Product (prname, type, guarantee) **Values** (“Bosch Double Oven”, “Oven”, “1”);

File Complaint:

Insert Into Complaint(cDate, subject, status) **Values** (NOW(), “Broken Screen”, “waiting”);

Send Message:

Insert Into Message (content, date) **Values** (“oven light does not working.”, NOW());

Change Username:

Update User **Set** username = “newUsername”

Where custID=@userID;

Change Password:

Update User **Set** password= “newPassword”

Where custID=@userID;

5.3. Main Page: Customer Service

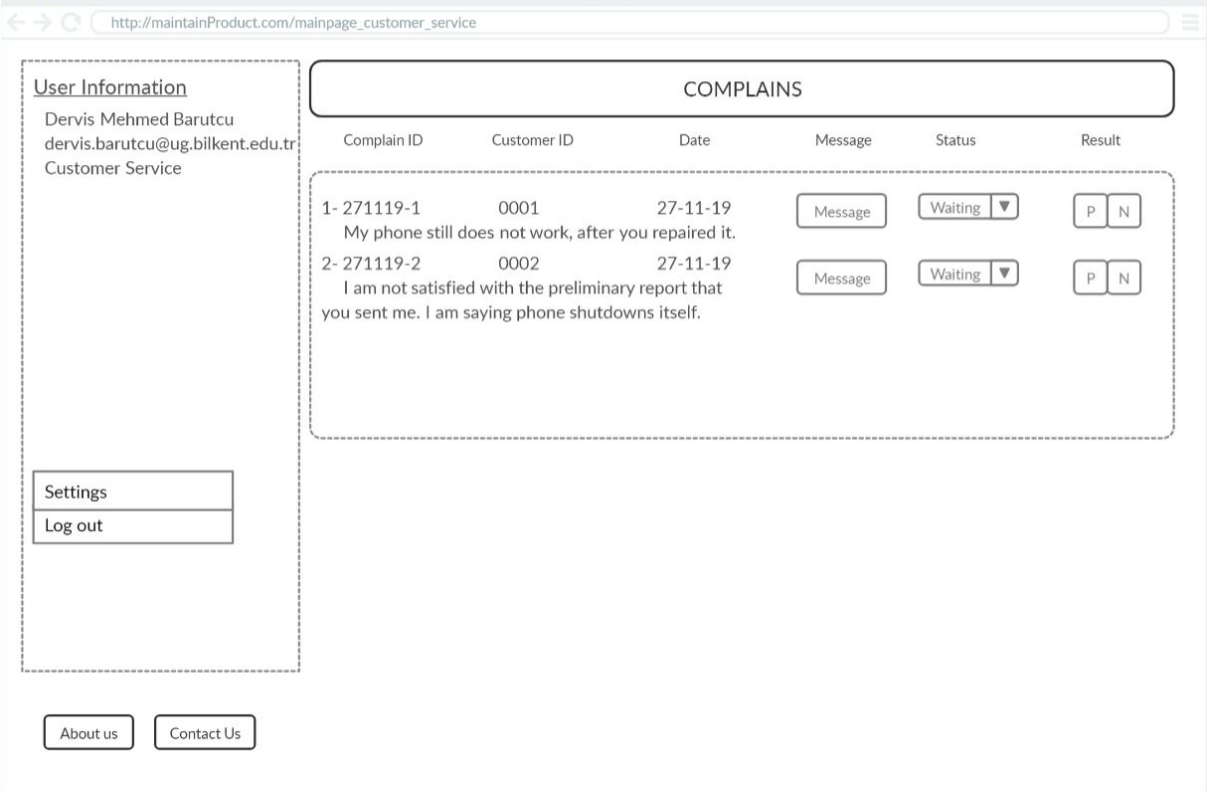

Figure 7: Main Page of Customer Service

User information:

Select name, email, userType

From User where empID=@userID;

Complain list:

Select complainID, custID, cdate, subject

From Info_About_Complaints;

Update status:

Update Complaint

set ctatus='selectedStatus'

where complainID=@selectedComplainID;

5.4. After Reports Button Pressed

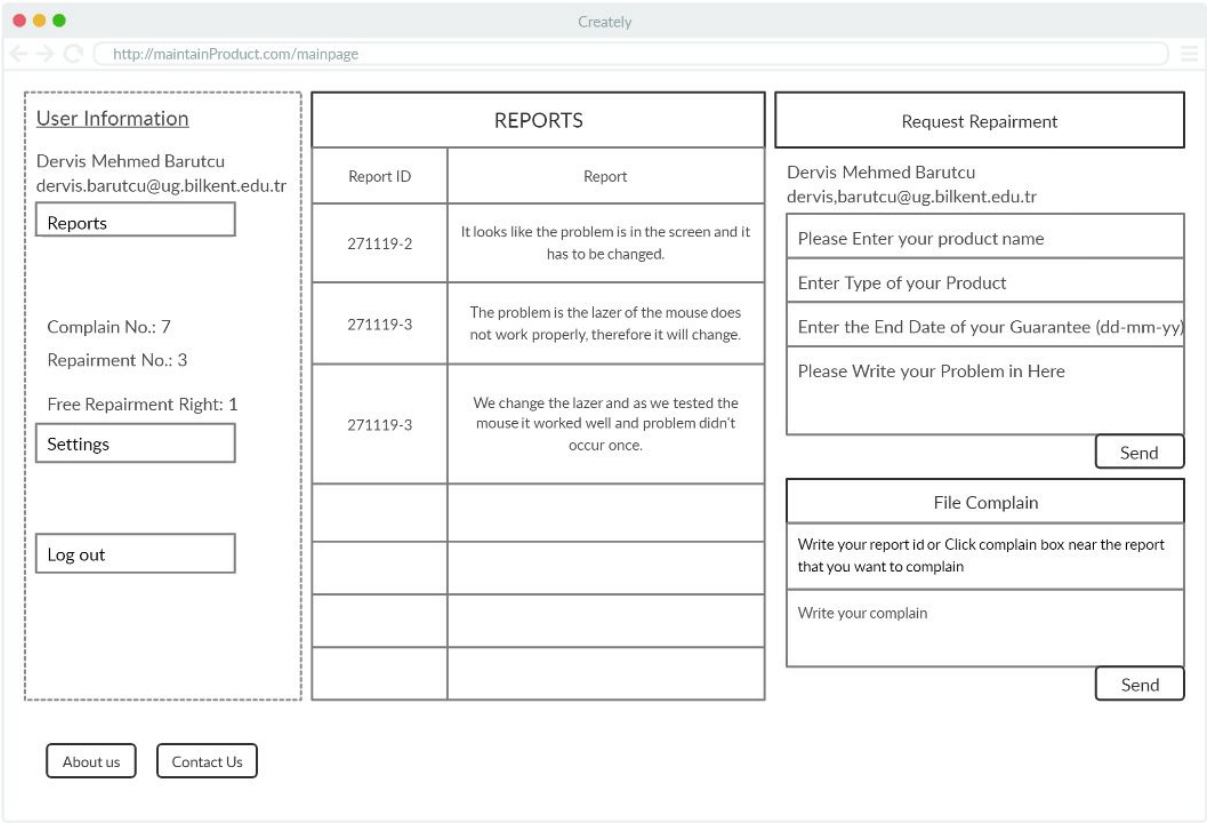

Figure 8: After Reports Button Pressed

Listing reports:

Select * **From** Info_About_Repairments

Where custID= @userID;

5.5. After Report Selected

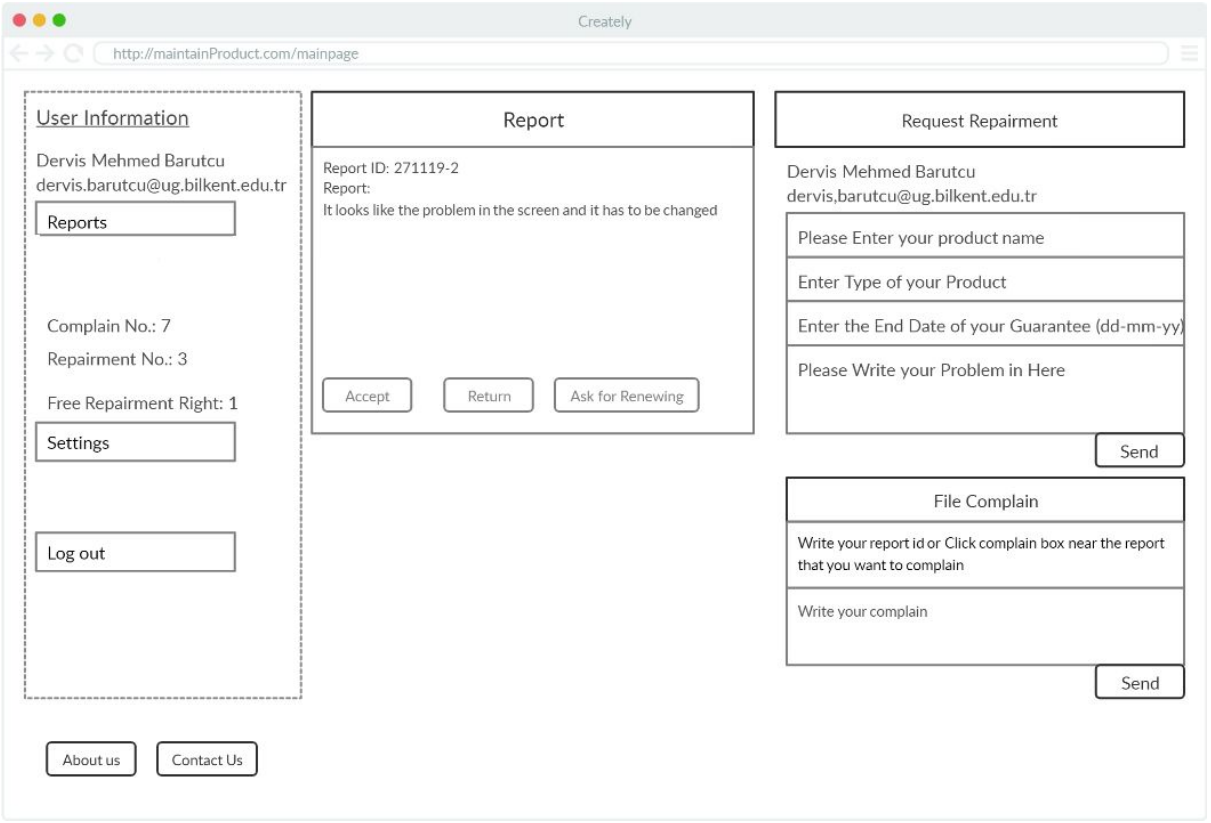

Figure 9: After Report Selected

Update request

set decision =”clickedDecision”

Where custID=@userID and repairmentID=@selectedRepairmentID;

5.6. Main Page: Tech Staff

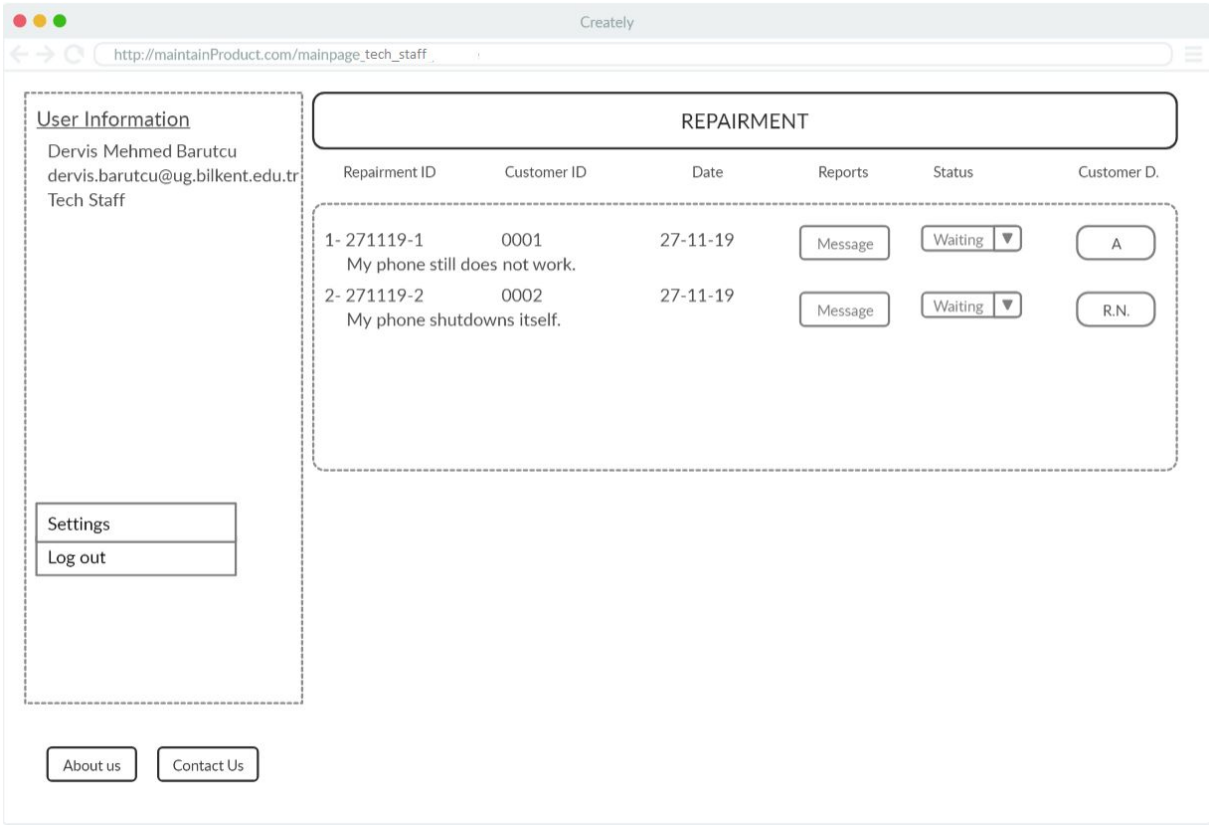

Figure 10: Main Page of Tech Staff

User information:

Select name, email, userType

From User where empID=@userID;

Repairment list:

Select repairmentID, custID,rdate, decision

From Info_About_Repairments;

Update status:

Update Repairment

set status='selectedStatus'

where repairmentID=@selectedRepairmentID;

5.7. Payment Page

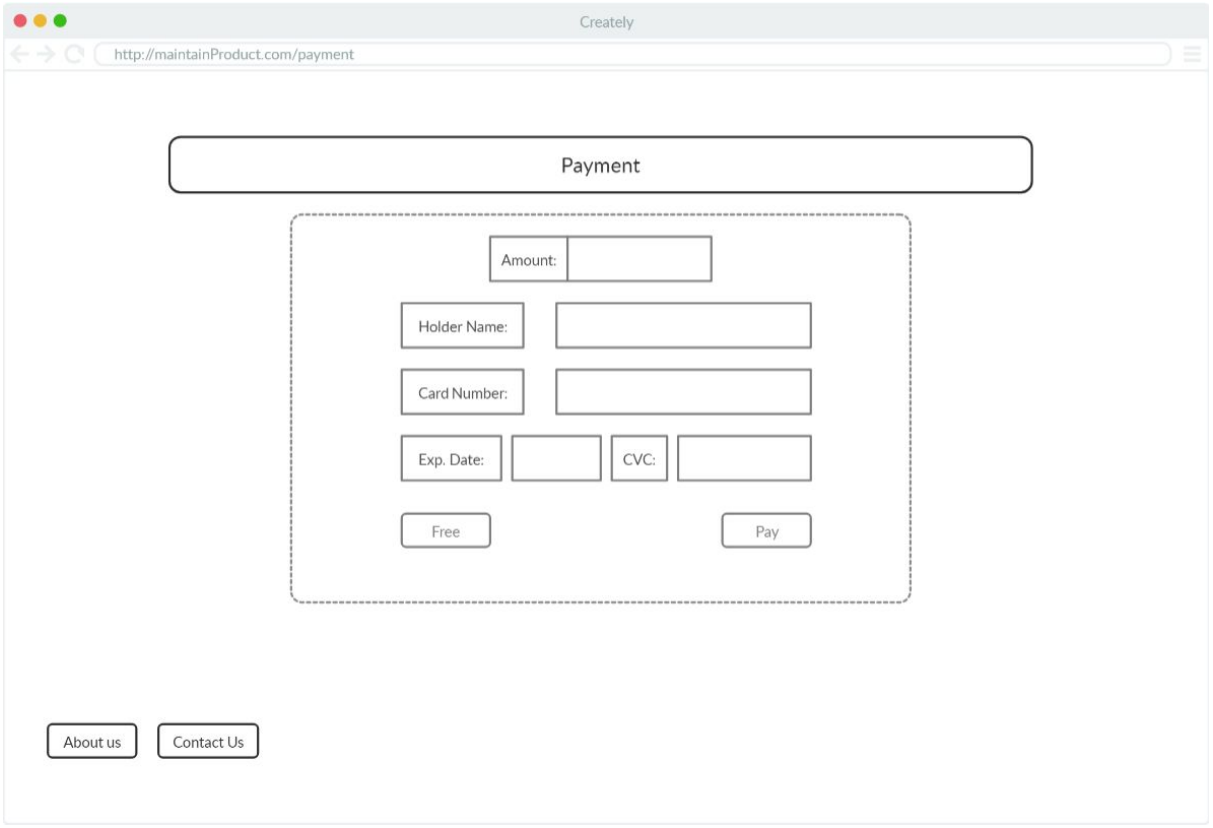

Figure 11: Payment Page

Payment:

Select repCount, guarantee;

From Customer;

Where username =”cBerk” **and** userType= “Customer”;

6. Advanced Database Components

6.1. Views

6.1.1. Info_About_Repairments

Create View Info_About_Repairments **as**

Select * **from** need **Natural join** request **Natural join** product;

6.1.2. Info_About_Complaints

Create View Info_About_Complaints **as**

Select * **From** File **natural join** Complaint;

6.2. Triggers

- When a repairment request is made repairment count (repCount) attribute will be incremented by one only if product is not under guarantee.
- After complaints are created views for customer service will be updated.
- After repairments are created views for tech staff will be updated.
- When an insertion on Message table occurs has2 and does relations updated.
- When an insertion on Complaint table occurs has2 and file relations updated.
- When an insertion on Repairment table occurs manage and request relations together with Category table updated.

6.3. Constraints

- To file a Repairment or complain, customer should create complain.
- To update a status and write file about product, product, situation must be finalized.
- TechStaff can not see complaints
- Customer Service can not see repairments.

- Free payment button can not be active if customers have not made more than three paid repairs.
- Usernames should be unique

6.4. Stored Procedures

6.4.1. Login

CREATE PROCEDURE Login (**IN** UserID CHAR(16), **IN** userType varchar(16), **IN** username varchar(16), **IN** password varchar(16))

BEGIN

IF userType = "Customer" **THEN**

SELECT * **FROM** Customer userID=custID AND password = “” and username = “”;

ELSEIF `userType` = "TechStaff" **THEN**

SELECT * **FROM** TechStaff **WHERE** userID=empID **AND** password = “” and username = “”

ELSEIF `userType` = "CustServices" **THEN**

SELECT * **FROM** CustServices **WHERE** userID=empID **AND** password = “” and username = “”

END;

6.4.2. Message

CREATE PROCEDURE Messaging(**IN** custID CHAR(16), **IN** empID varchar(16))

BEGIN

SELECT * **FROM** has2 **natural join** does **natural join** have **as** CurMess

WHERE CusrMess.custID=custID **AND** CurMess.empID=empID;

END;

7. Implementation Plan

We will be using MySQL Server in our project to handle database. Php and javascript functions will also be used to handle data related functions, page transitions and parameter passing.