Mary Wilson
Carlos Leon
Database Design
Final Project

Retail System Final Report

For our final project, we successfully created a retail store website with customer, employee, and manager functionality. To do this, we made a web based interface with HTML, CSS, and JavaScript. This was connected to a PostgreSQL server with node.js. (Don't be scared by the size of this report, the length is mostly from pictures.)

Contents of this Report (Linked to each section)

Back to Top

Tables in the Database

- **Comments** - Stores the timestamp of the comment and the comment itself from the user
  {timestamp : timestamp, comment : varchar(305)}

- **Customers** - Stores the CustID, name, and email of each customer
  {CustID : int4, name : varchar(50), email : varchar(50)}

- **Employees** - Stores the EmpID, name, position, and department of each employee
  {EmpID : int4, Name : varchar(50), Position : varchar(50), Department : varchar(30)}

- **InStock** - Stores the item's UPC along with the StoreID and the quantity that store has in their inventory
  {UPC : int4, StoreID : int4, Qty : int4}
  FKs: UPC to Items, StoreID to Stores

- **Items** - Stores key information about an item such as UPC, size, color, department, description, brand, and price
  {UPC : int4, Size : varchar(30), Color : varchar(30), Department : varchar(30), Description : varchar(50), Brand : varchar(30), Price : money}

- **Stores** - Stores key information about a store such as the StoreID, store name, city, state, rating, and year it was founded
  {StoreID : int4, StoreName : varchar(30), City : varchar(30), State : varchar(30), rating : int4, year : int4}

- **Transactions** - Stores key information about a transaction such as the TransID, EmpID of cashier, CustID of customer, StoreID of store purchased at, timestamp of purchase, and the total amount for the entire purchase
  {TransID : int4, EmpID : int4, StoreID : int4, CustID : int4, Timestamp: timestamp, Total: float8}
  FKs: EmpID to Employees, StoreID to Stores, CustID to Customers

- **Users** - Stores CustID (if customer) or EmpID (if employee) along with the usernames and hashed passwords of all users
  {id : int4, custid : numeric, empid : numeric, username : text, password : text}

Back to Top

Pages on the Website

Login

- Login - Takes in username and password, compares username with hashed passwords to log you in and take you to the home page
- Create user - Takes in username, password, name, and email to create a customer profile

Home

- Customers & Managers - Displays three buttons “Transaction History,” “Item Catalog,” and “Comment”
- Employees - Displays two buttons “Transaction History” and “Item Catalog”
- Both - Has links in the header to the about page, profile, and store search

Transaction History

- Customers - Displays a list of transactions that they completed as a customer
- Employees - Displays a list of transactions that they completed as an employee
- Managers - Displays a list of all transactions

Item Catalog

- Customers - Displays all items, searchable by store, brand, size, and department
- Employees - Displays all items in their department, minor features are editable with the edit button next to each item. There is also a quantity edit button that disables all fields besides quantity until you hit save
- Managers - Displays all items with the same editing functionality explained for employees

Comments

- Customers - Displays a text box for customers to leave comments about their experience at the store
- Managers - Displays a table of customer entered comments with the timestamp attached

Back to Top

Store Search

- Displays the store information that is searchable by store name or state located in

About

- Blurb with a mission statement and a picture of our dog (that the site is named after)

Profile

- Customers - Displays name, CustID, and email - name and email are editable
- Employees and Managers - Displays name, EmpID, position, and department - name is editable

Back to Top

Frontend Details

The frontend consists of a series of web pages that show data retrieved from the database. These web pages are written as EJS templates. These templates are populated with the retrieved data, rendered to HTML, and then sent to the user. The EJS templates are essentially standard HTML with some extra functionality for iterating over data. That page then loads the CSS and JavaScript files for styling. The the front-end JavaScript files contain functions for making requests back to the back-end as well as showing/hiding certain elements and showing the status of database operations.

Backend Details

The backend for this project was written in node.js which is a server side JavaScript interpreter built around the V8 engine. We choose node.js for this project because JavaScript is built around the idea of asynchronous programming. This is particularly useful for creating servers because it allows for requests to be processes in a non-blocking manner without spinning a new thread for each response. As an example, while the backend makes a query into the database to search for store items, it does not have to block incoming connections for users logging in. When the store item search query is complete, the function will call the callback function to return a response to the user. In this project, we particularly made heavy use of advance asynchronous features in JavaScript called Promises and Async/Await which work much in the same way as passing around callback functions but have the added benefit of keeping code tidy and making error handling much easier.

The backend server is built using express.js which is a node.js library. Express provides middleware between the moment that a request gets to the server and when we process it. It allows for functions to parse the incoming request to make it easier to handle. In particular we used multer.js, another node.js library, for parsing form data, and Cookie Parser for parsing cookie data. Furthermore, express makes routing requests to specific endpoints much easier. We route each of our pages to their specific GET endpoints for EJS rendering, and each XMLHttpRequest made on the front end by things like searching and submissions to their specific POST endpoints for processing.

We used pg.js for connecting to the postgres database. This library is a JavaScript wrapper around libpq (the PostgreSQL C library). This library also allowed us to check for SQL injections by using parameterized queries. Furthermore, we used bcrypt for password hashing. This is a JavaScript wrapper around the C Bcrypt library put out by the same people.

Back to Top

A Brief Explanation of our Cookie Logic

When the user types in the website address, we receive the get request, assign the user a cookie, and then send the cookie along with the webpage back to the user. From there, on all subsequent requests, we put the cookie through a parser to make it an easy-to-read Javascript object. We do have an add_cookie function that checks if the user has a cookie, if not, it assigns them one. The cookie is a randomly generated 8 digit number. Once the user logs in, we query the database once for the user's information and store in two structure on the backend: one structure is an array of objects that holds the user's data and the other is a hashtable that uses the cookie tracking id to look up the index into the user array . We keep track of the cookie tracking id, the user type (customer, employee, or manager), and the user's id which is either their CustID or EmpID so we can access their information faster. For each webpage, it checks the user type to decide which version of the webpage to show, and whether or not we need to redirect them to the login page.

Back to Top

Queries per Webpage

(For full functions, all of these queries are located in db_connector.js in our components folder.)

Login

- Creating a new user: For this specific code, we are initially inserting the customer's name and email into the customer table with the insert statement in the first picture below. In addition to adding to the customer table, we also add the CustID, username, and password to the users table once the hashing is complete (second image).

```
function create_user_cust(customer){
    return new Promise((resolve, reject) => {
        // Validate input
        if(customer.username.length === 0 || customer.password.length === 0){
            resolve("error");
            return;
        }
        let email_pattern = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
        if(customer.name.length === 0 || customer.email.length === 0 || !email_pattern.test(customer.email) ){
            resolve("error");
            return;
        }

        // Get a connection
        let conn = GetConnector();
        let query = 'INSERT INTO customers ("name", "email") VALUES ($1, $2) Returning "CustID";'
        let values = [customer.name, customer.email];

        // Insert the query into the customer table
        conn.query(query, values, (err1, res1) => {
            // Check for error
            if(err1){
                resolve("error");
                return;
            }

            // Get the new employee id
            let custID = res1.rows[0].CustID;

            // Validate
            if(typeof custID !== "number"){
                resolve("error");
                return;
            }
```

Back to Top

```
// hash the password
bcrypt.hash(customer.password, 10, (err2, hash) => {
    // Check error
    if(err2){
        resolve("error");
        return;
    }

    // Set new query for users table
    query = "INSERT INTO users (custid, username, password) VALUES ($1, $2, $3);";
    values = [custID, customer.username, hash];

    // Insert the query
    conn.query(query, values, (err3, res3) => {
        // Close the connection
        conn.end();

        // Check error
        if(err3){
            resolve("error");
            return;
        }

        // If no error, then resolve true
        resolve("success");
    });
});
```

- Logging in with an existing user (to spare you the report space, only the query code is shown below for the rest of the pages): For this functionality, we select the left join of users and employees to get the position of the employee (if they are an employee). If they are a customer, they will have a CustID in the users table so the join won't matter. This is only important to distinguish if an employee is a manager or not. (Shown below)

```
// Get a connection
let conn = GetConnector();
let query = 'select * from users u left join employees e on u.empid = e."EmpID" where "username"=$1;';
let values = [username];
```

Back to Top

Transaction History

- Showing the table (this one is not searchable): In this code, the query is built from all the different items that you view in the transaction history, along with the if statement that determines the where condition.

```
let values = [];
let query = 'SELECT t."TransID", to_char(t."Timestamp", \'MM-DD-YY HH12:MI AM\') as tm, t."Total",';
query += 'c.name AS cust_name, s."StoreName", '
query += 'e."Name" AS emp_fname '
query += "FROM transactions t "
query += 'LEFT JOIN employees e ON e."EmpID" = t."EmpID"'
query += 'LEFT JOIN customers c ON c."CustID" = t."CustID"'
query += 'LEFT JOIN stores s ON s."StoreID" = t."StoreID"'
if(user.type !== "manager"){
    if(user.type === 'employee'){
        query += ' where t."EmpID" = $1;'
    }
    else{
        query += ' where t."CustID" = $1;'
    }
    values.push(user.id);
}
```

Item Catalog

- Shows the whole query first: In this query, we join the items table with the inStock table to display the quantity per item per store.

```
let query = 'select * from items i JOIN instock s ON i."UPC" = s."UPC" JOIN stores o ON s."StoreID" = o."StoreID" ';
let param = 1;
let where = "where "
let values = [];
```

Back to Top

- Can search per store number, size, brand, and department: This code adds onto the code in the last example to refine the selection by department, brand, size, or store number.

```
// For searching
if(dept || brand || size || store){
    if(dept){
        where += (param > 1 ? " and " : "");
        where += ' i."Department" like $' + param.toString() + " ";
        param++;
        values.push("%" + dept + "%");
    }
    if(brand){
        where += (param > 1 ? " and " : "");
        where += ' i."Brand" like $' + param.toString() + " ";
        param++;
        values.push("%" + brand + "%");
    }
    if(size){
        where += (param > 1 ? " and " : "");
        where += ' i."Size" like $' + param.toString() + " ";
        param++;
        values.push("%" + size + "%");
    }
    if(store){
        where += (param > 1 ? " and " : "");
        where += ' s."StoreID"= $' + param.toString() + " ";
        param++;
        values.push(store);
    }
    query += (param > 1? where : "");
}
if(user.type === "employee" && param === 2){
    query += where;
}
query += ' Order By i."UPC";'
```

- Employees can only see items in their department: In this query, we select the items from the employee's department only.

```
if(user.type === "employee"){
    query = 'SELECT * FROM users u JOIN employees e ON e."EmpID" = u.empid JOIN items i ON i."Department" = e."Department" JOIN instock s ON s."UPC" = i."UPC" ';
    where += " u.empid=$"+ param.toString() + " ";
    values.push(user.id);
    param++;
}
```

- Employees and Managers can edit item details: If the edit button is pressed next to an item, all the information changed in the text boxes for that item will be added to the database as an update to the current information.

```
let query = 'update items Set "Size"=$1, "Color"=$2, "Description"=$3, "Brand"=$4, "Price"=$5 where "UPC"=$6;';
let values = [item.size, item.color, item.desc, item.brand, item.price, item.upc];
```

Back to Top

- Employees and Managers can edit item stock: For simplicity sake, we separated the edit quantity option and the edit item option to be two different buttons. When the save button is pressed for the item quantity, the item quantity for that specific item at that store will be updated.

```
let query = 'update instock Set "Qty"=$1 where "UPC"=$2 and "StoreID"=$3;';
let values = [item.qty, item.upc, item.store];
```

Comments

- Customers can leave comments: Takes the comments from the user's textbox and stores it in the comments table. When the comment is inserted, the

```
let query = 'insert into comments (comment) values($1);';
let values = [comment];
```

- Managers can view comments: The managers only view comments, therefore this code selects the comments and timestamp and formats it accordingly.

```
conn.query('SELECT *, to_char(Timestamp, \'MM-DD-YY HH12:MI AM\') as tm FROM comments;', (err1, res1) => {
    if(err1){
        resolve("error");
    }else{
        resolve(res1.rows);
    }
});
```

Back to Top

Store Search

- Shows all stores and searchable by state and store name: Same setup as the item search, without the editing functionality.

```
let query = 'select * from stores ';
let where = "where "
let values = [];
if(state || name){
    if(state && name){
        where += '"State" like $1 and "StoreName" like $2'
        values.push("%" + state + "%", "%" + name + "%");
    }
    else if(state){
        where += '"State" like $1';
        values.push("%" + state + "%");
    }
    else if(name){
        where += '"StoreName" like $1';
        values.push("%" + name + "%");
    }
    else{
        where = "";
    }
    query += where + ";";
}
```

Profile

- Customers and employees are shown their information: This is done with a simple select statement based on the type of user.

```
if(user.type === "customer"){
    query = 'SELECT * FROM customers WHERE "CustID"=$1;'
}
else{
    query = 'SELECT *, "Name" AS name FROM employees WHERE "EmpID"=$1;'
}
```

Back to Top

- Customers can edit name and email: This updates the email and name of a customer depending on which edit button they press.

```
if(user.type === "customer"){
    if(typeof name === "undefined"){
        query = 'UPDATE customers SET email=$1 WHERE "CustID"=$2;'
        values.unshift(email);
    }
    else{
        query = 'UPDATE customers SET name=$1 WHERE "CustID"=$2;'
        values.unshift(name);
    }
}
```

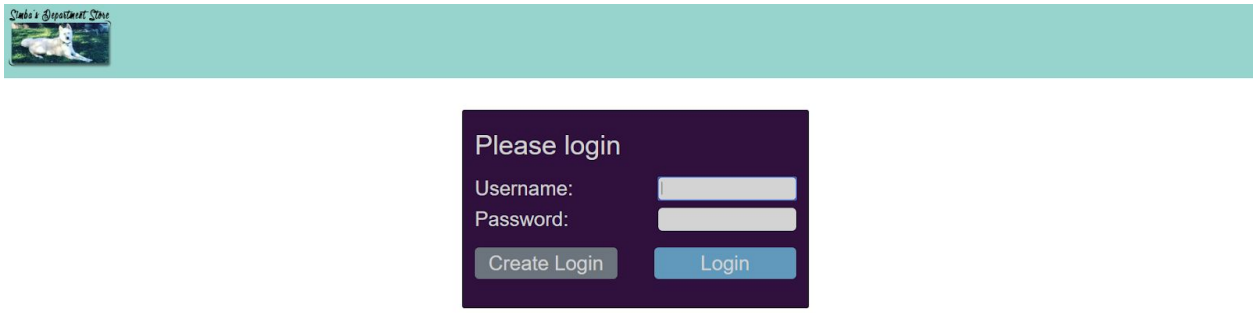
- Employees can edit name: This is the same as for a customer, employees just can't edit any other information about themselves.

```
else{
    query = 'UPDATE employees SET "Name"=$1 WHERE "EmpID"=$2;'
    values.unshift(name);
}
```

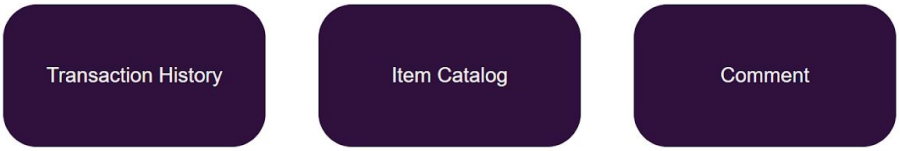
Back to Top

Customer View

Login:

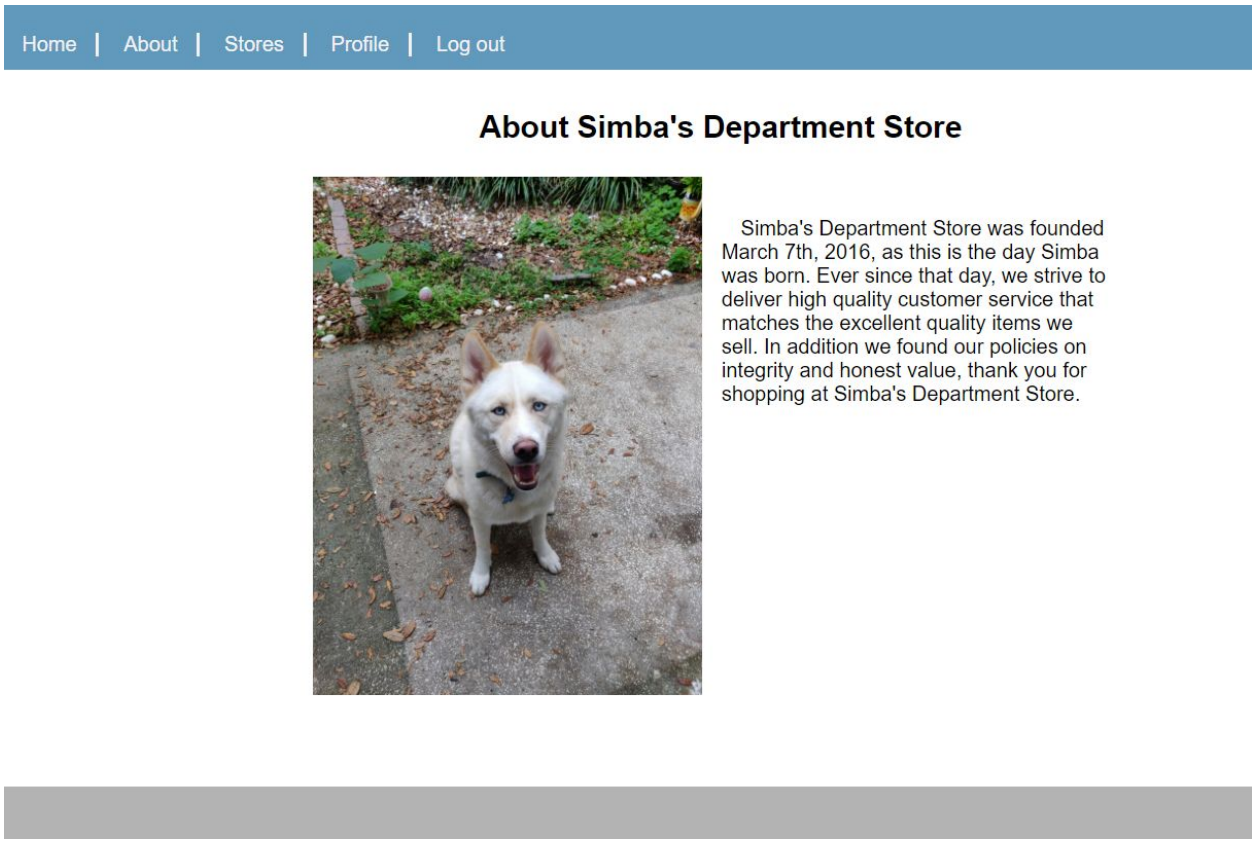


Home Page:

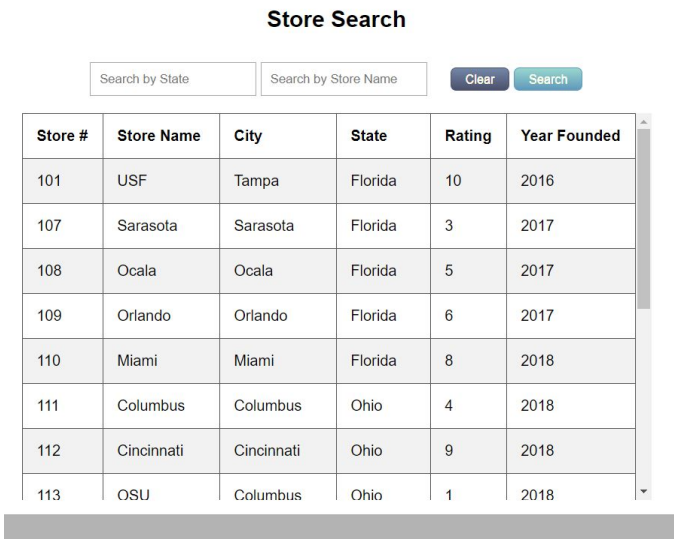


Back to Top

About Page:

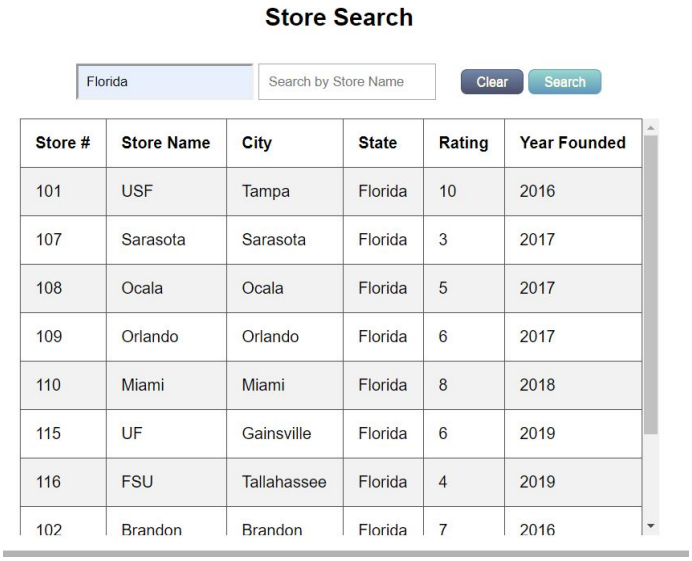


Store Search:

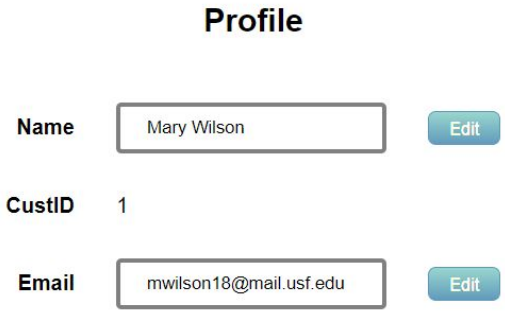


Back to Top

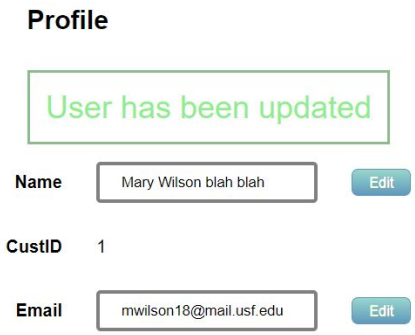
Store Search after Search:



Profile:

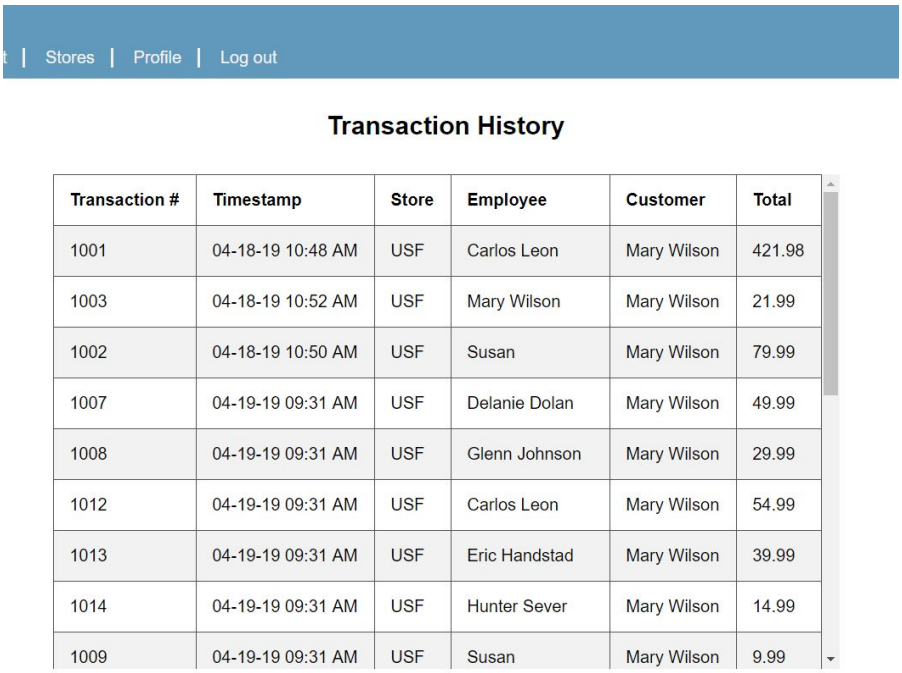


Profile after Edit:



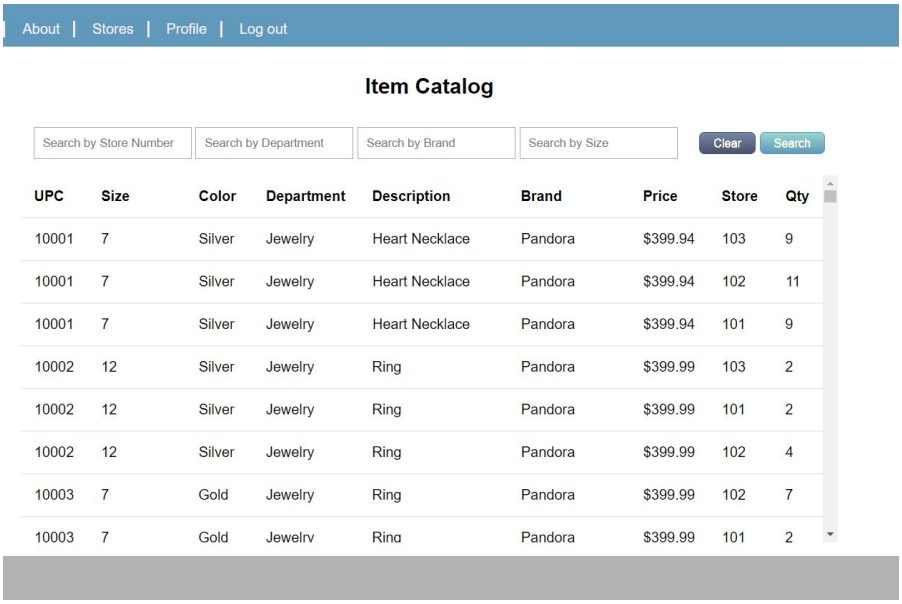
Back to Top

Transaction History:



Transaction #	Timestamp	Store	Employee	Customer	Total
1001	04-18-19 10:48 AM	USF	Carlos Leon	Mary Wilson	421.98
1003	04-18-19 10:52 AM	USF	Mary Wilson	Mary Wilson	21.99
1002	04-18-19 10:50 AM	USF	Susan	Mary Wilson	79.99
1007	04-19-19 09:31 AM	USF	Delanie Dolan	Mary Wilson	49.99
1008	04-19-19 09:31 AM	USF	Glenn Johnson	Mary Wilson	29.99
1012	04-19-19 09:31 AM	USF	Carlos Leon	Mary Wilson	54.99
1013	04-19-19 09:31 AM	USF	Eric Handstad	Mary Wilson	39.99
1014	04-19-19 09:31 AM	USF	Hunter Sever	Mary Wilson	14.99
1009	04-19-19 09:31 AM	USF	Susan	Mary Wilson	9.99

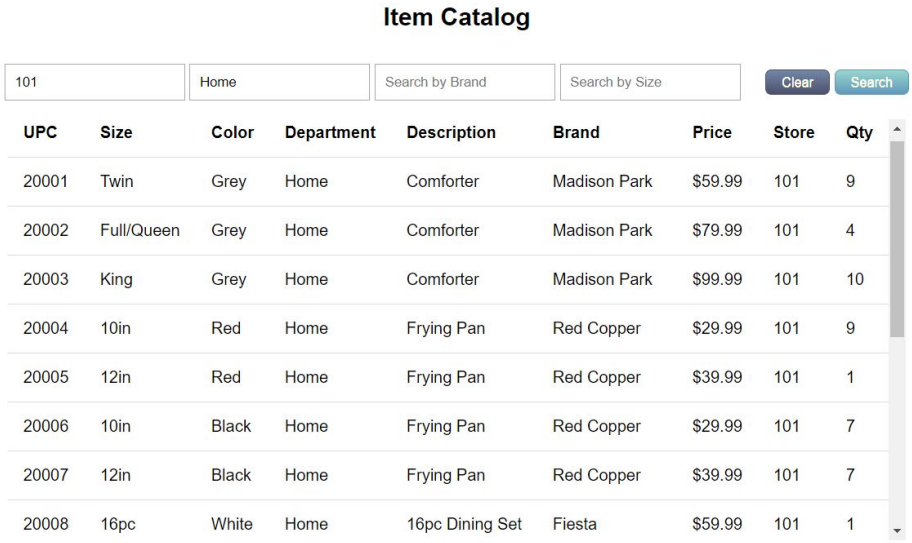
Item Catalog:



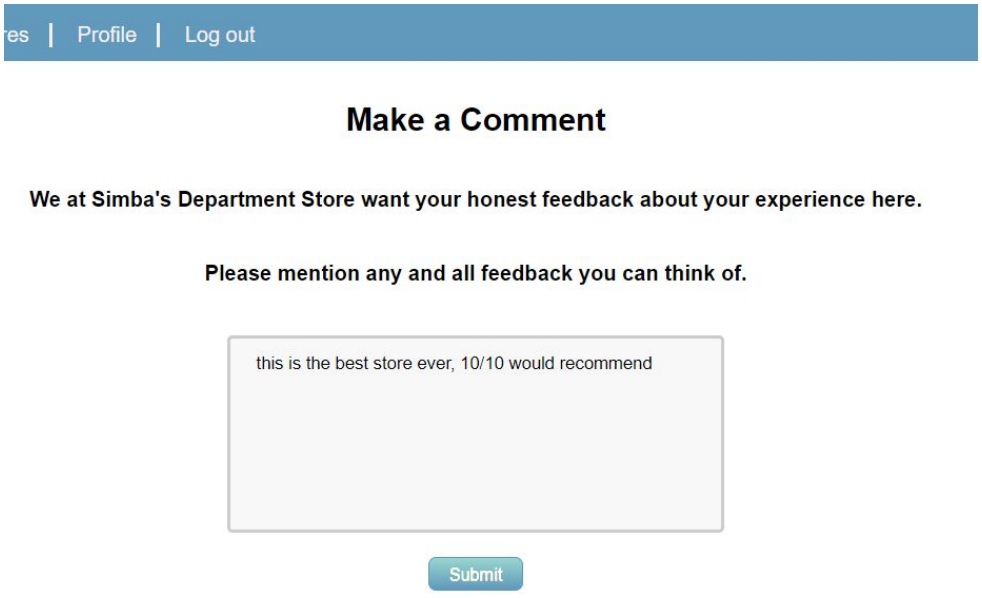
UPC	Size	Color	Department	Description	Brand	Price	Store	Qty
10001	7	Silver	Jewelry	Heart Necklace	Pandora	$399.94	103	9
10001	7	Silver	Jewelry	Heart Necklace	Pandora	$399.94	102	11
10001	7	Silver	Jewelry	Heart Necklace	Pandora	$399.94	101	9
10002	12	Silver	Jewelry	Ring	Pandora	$399.99	103	2
10002	12	Silver	Jewelry	Ring	Pandora	$399.99	101	2
10002	12	Silver	Jewelry	Ring	Pandora	$399.99	102	4
10003	7	Gold	Jewelry	Ring	Pandora	$399.99	102	7
10003	7	Gold	Jewelry	Ring	Pandora	$399.99	101	2

Back to Top

Item Catalog after Search:



Comment:



Back to Top

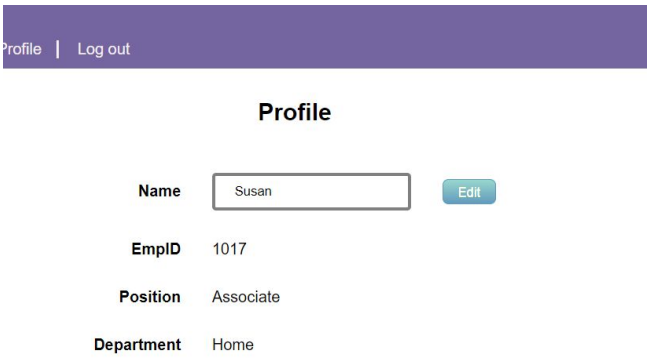
Employee View

Login is the same as Customer

Home Page:

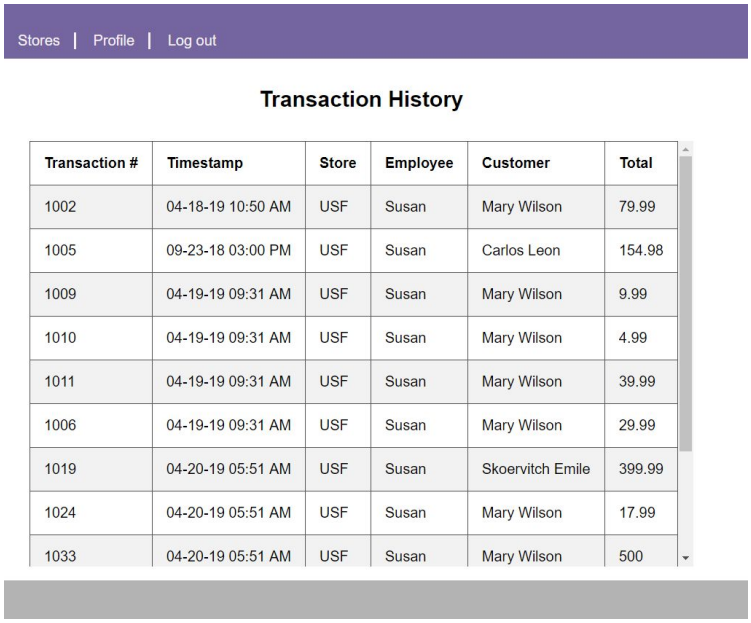
About and Store Search are the same as Customer except header is the purple shown above.

Profile:

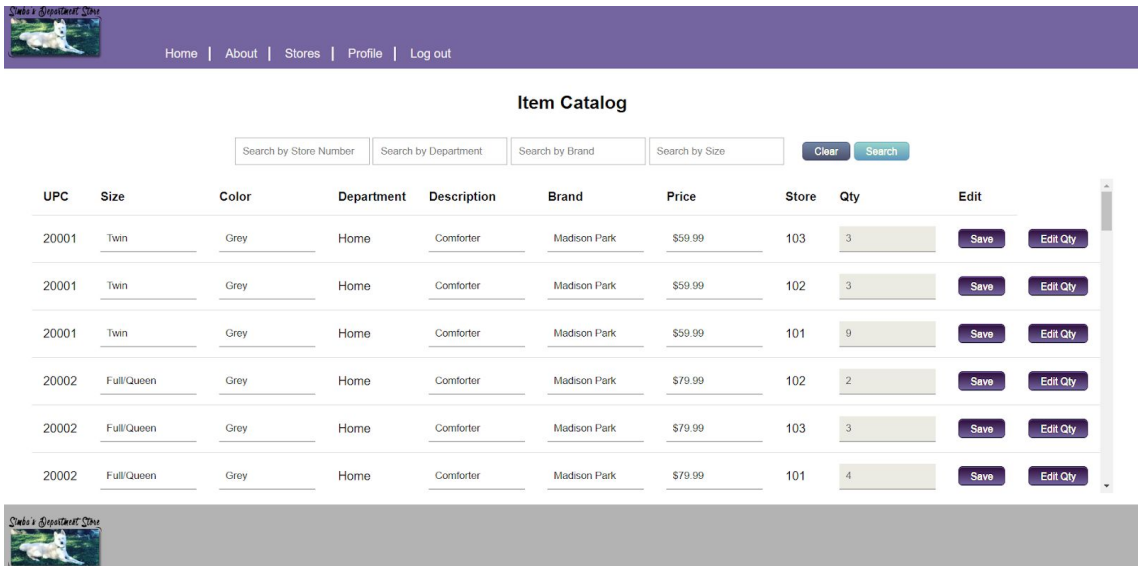


Back to Top

Transaction History:

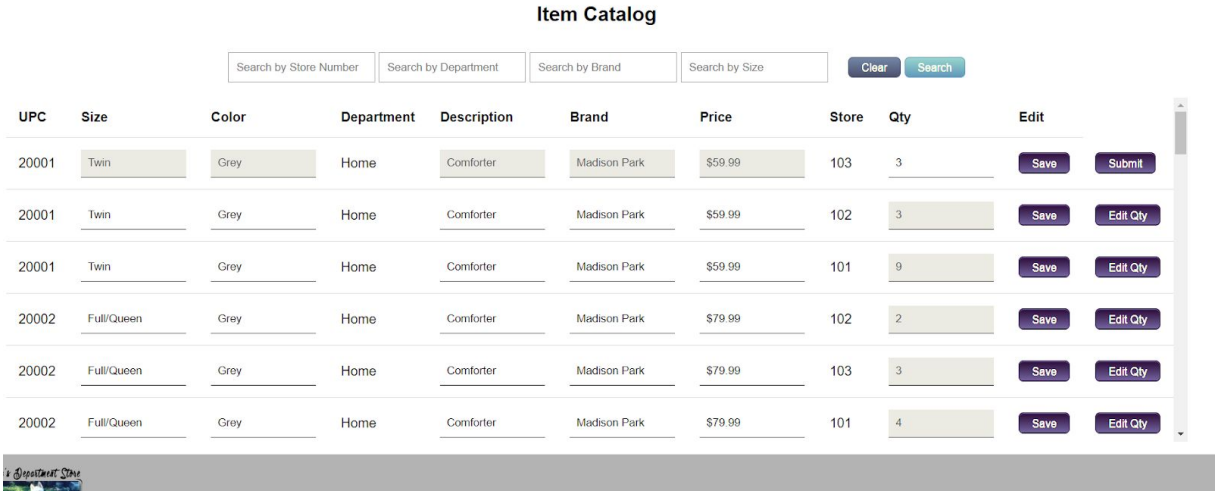


Item Catalog: (At this point the underlined areas are the editable regions, saved with the save button.)

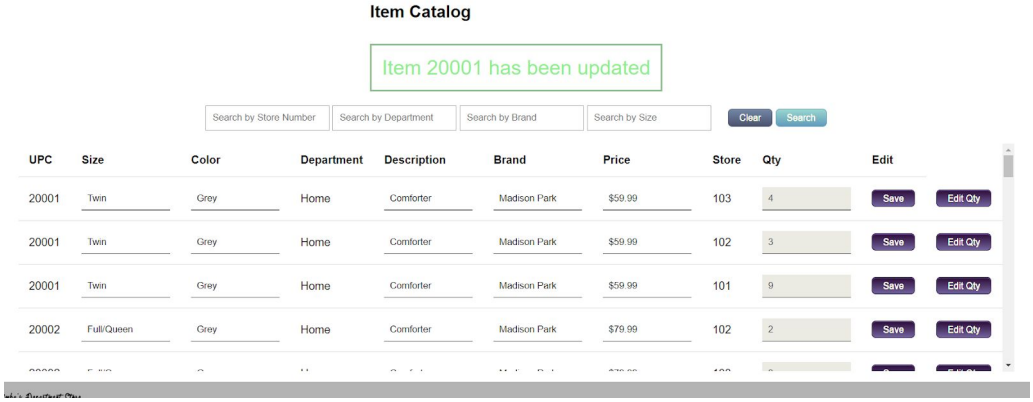


Back to Top

Item Catalog Qty Edit: After they click Edit Qty, the rest of the fields will disable leaving you to edit the Qty for that item in that store. To save new Qty, click submit.



Item Catalog Qty Saving:

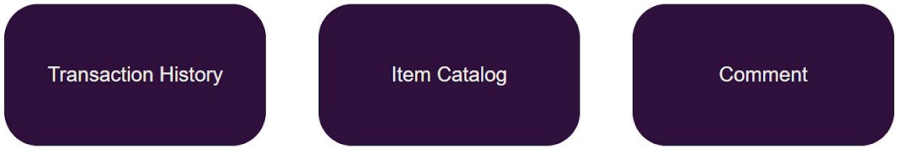


Back to Top

Manager View

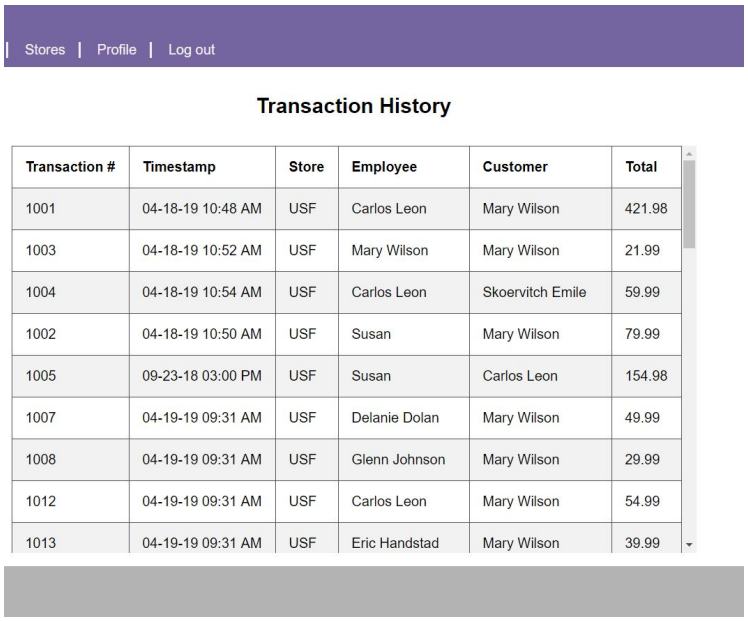
Login is the same as Customer and Employee

Home Page:



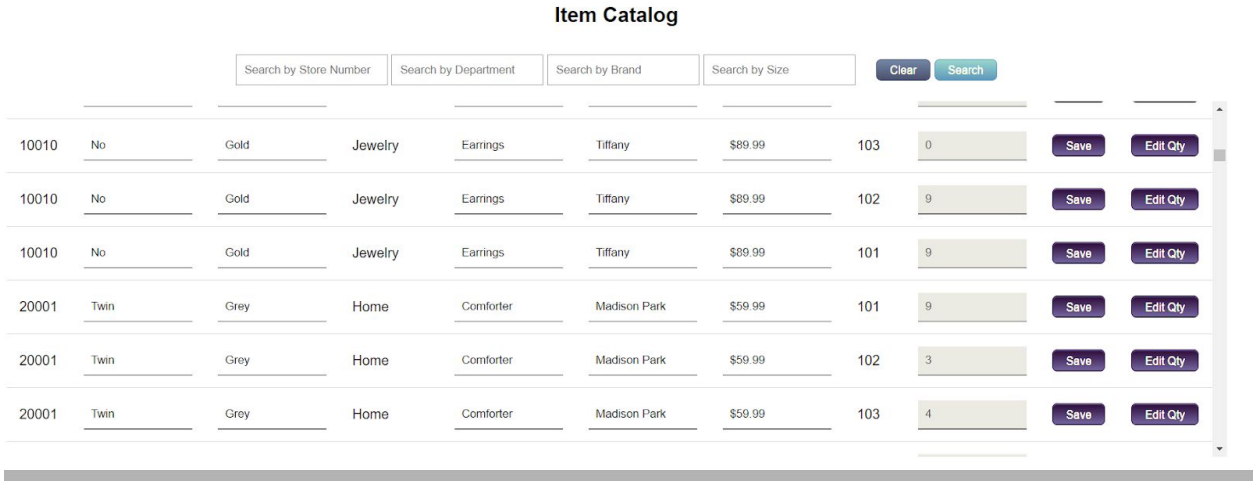
About, Store Search, and Profile are all the same as Employees.

Transaction History: (See all transactions, not just their own)

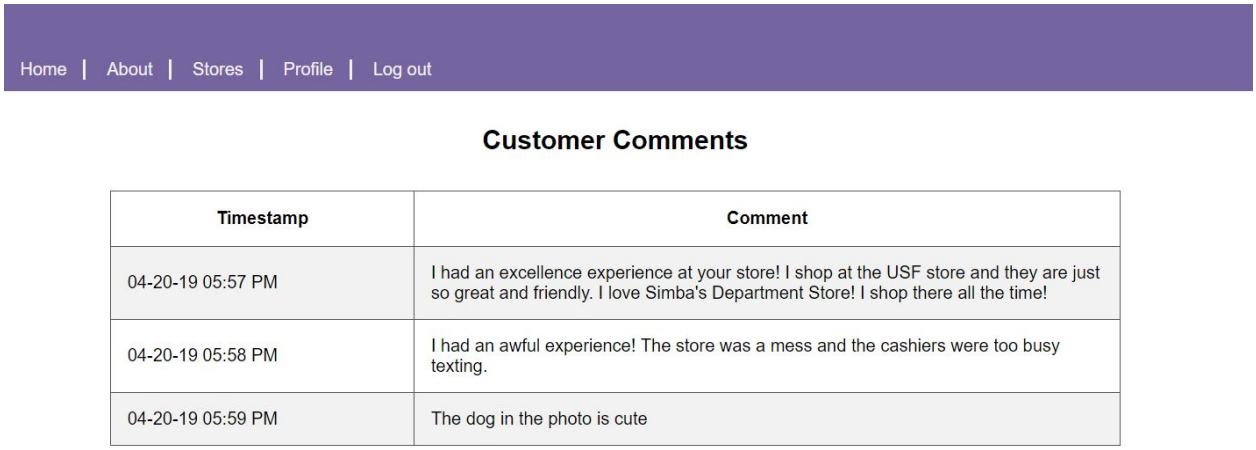


Back to Top

Item Catalog: (See all items, not just their department, and can edit all)



Comments:



Back to Top