CPS 630 - Web Applications
Smart Customer Services (SCS)

Technical Report for Project Iter I, II, III and IV

Team 37

Jenny Su, 500962385

Tiffany Tran, 500886609

Kevin Tran, 500967982

Michael Widianto, 501033366

Name	**Task**	**Percentage**
Jenny Su	Sign in, Cookies, Shopping cart process, Order history	25%
Tiffany Tran	Front-end, React, Homepage	25%
Kevin Tran	Maps, Service C	25%
Michael Widianto	Back-end, Review, DBMaintain	25%

Project Description

The objective of the project is to create a shopping website for clothing. Our goal is to create an interactive web application that allows users to: sign up and sign in to their own personal account, choose and add items to their shopping cart, submit their order and submit reviews.

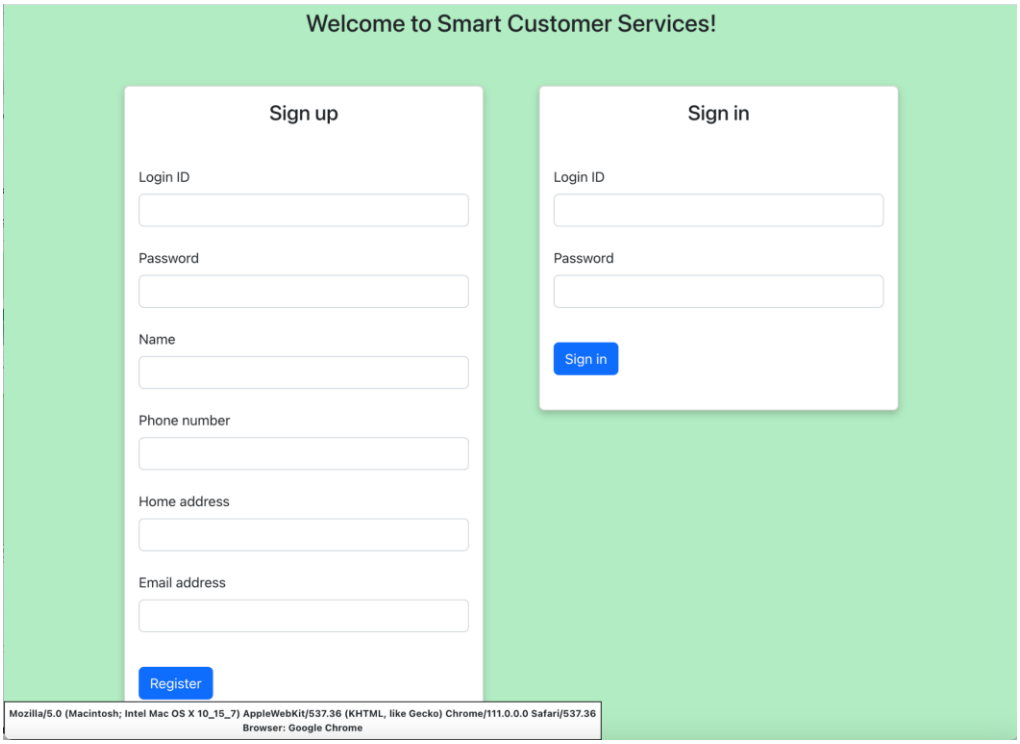
For the front-end, we used React to make our website based on Single-Page-Application (SPA) architecture. We used HTML and CSS with the Bootstrap framework and JavaScript to design and create responsive web pages.

For the back-end, we used XAMPP to create an Apache server and MySQL database to store our data. In addition to that, PHP was used for communication between the front-end and the back-end.

Design and Layout of Multi-pages and User Interfaces

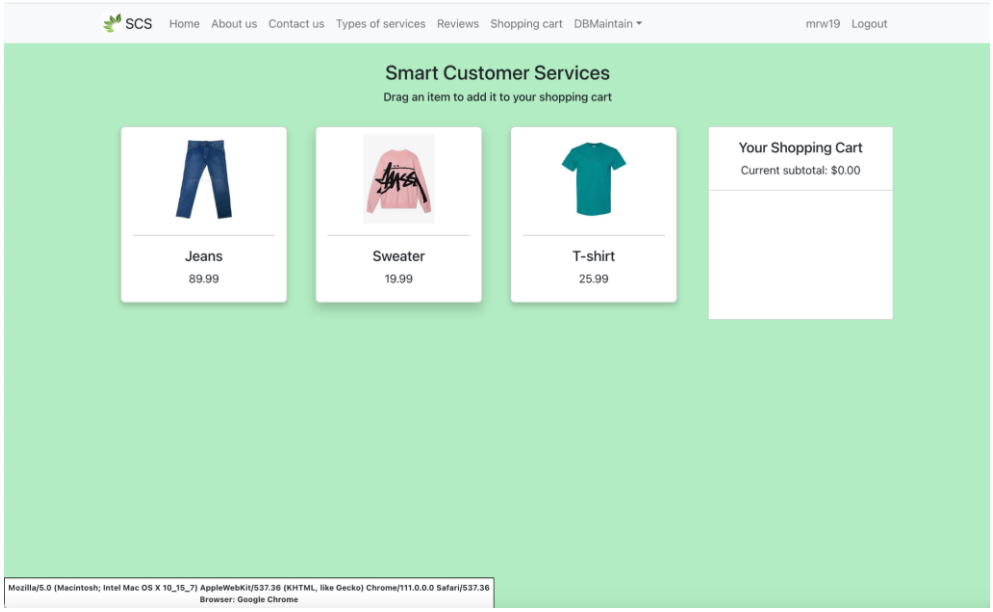
Sign in and Sign up - Launch page

The user first enters the web application through the sign in/sign up page. The user can choose the method of their choice though the respective forms. When signing up, the user will be returned to this page to sign in, and upon signing in, the user will be redirected to the home page. Each page after signing in has a navigation bar to move from one page to another, a title displaying which page the user is on and a short description describing what the page is.



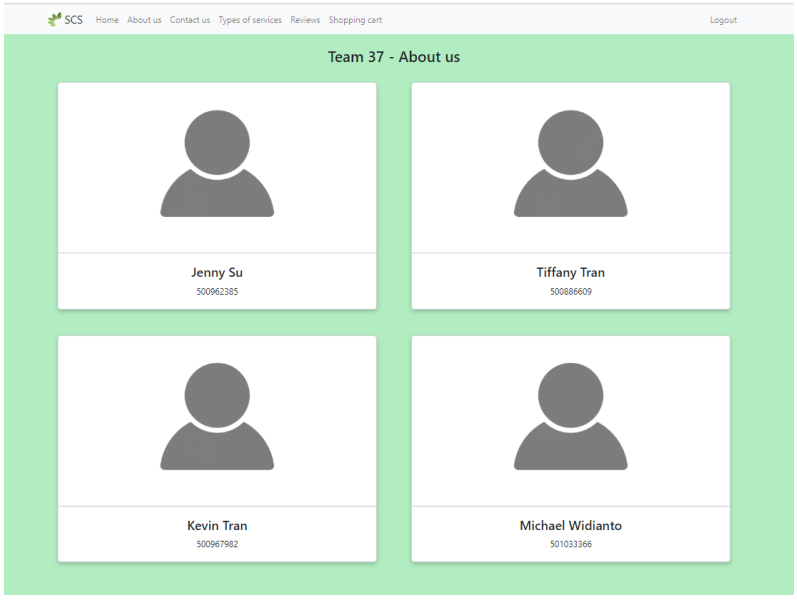
Home page

Users can add items to their cart from the home page. Users drag and drop the image icons of their desired item into the shopping cart container to add the item to their shopping cart. The subtotal and cart list will update after each item is added.



About us / Contact us

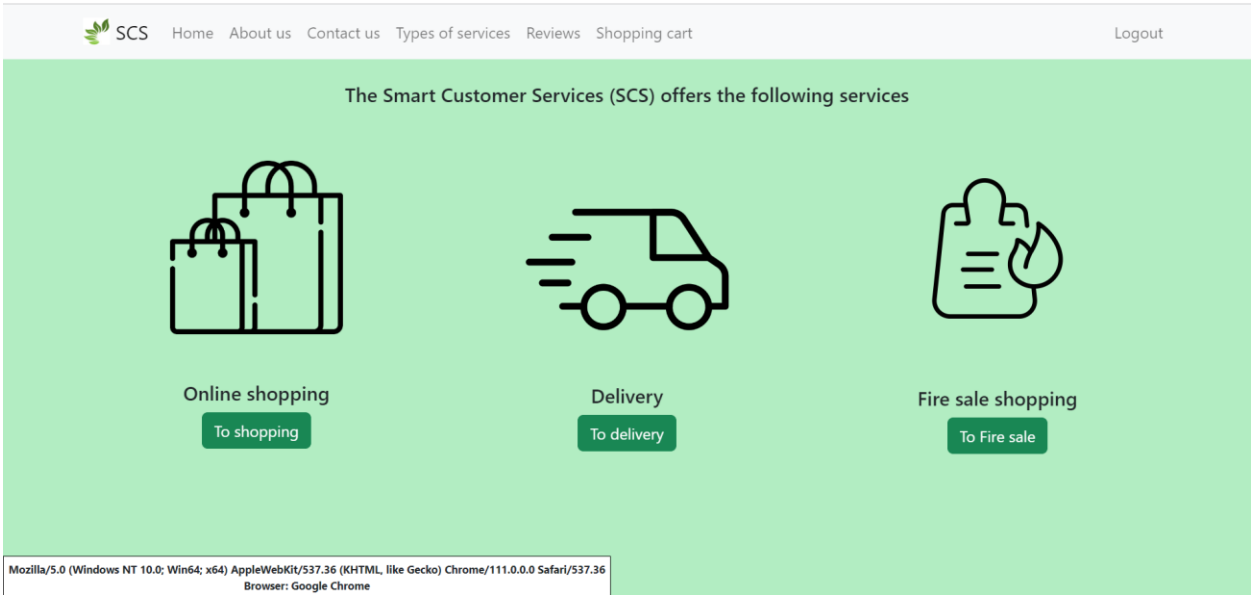
Two short pages that include information about the team members (name, student id, email)



Types of Services

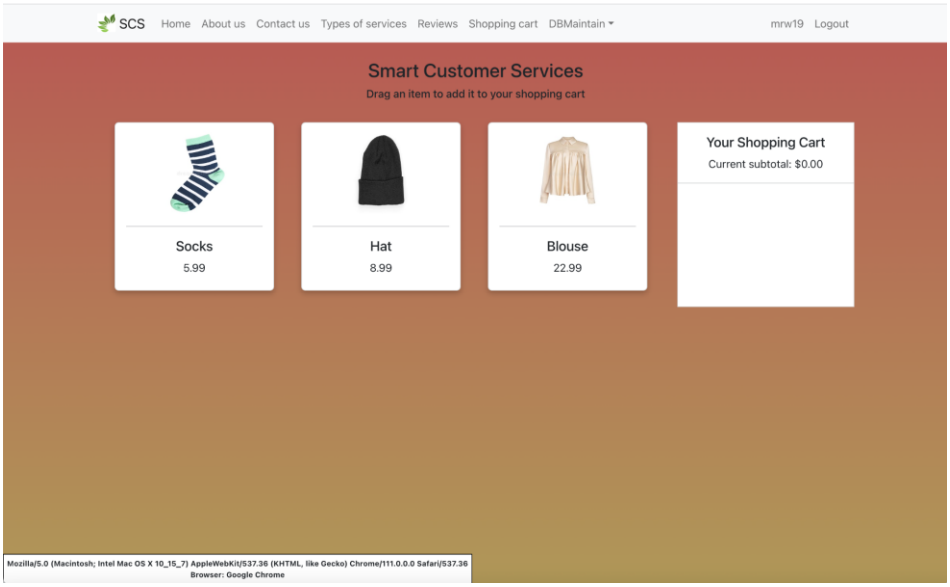
The user can select from three options what they would like to do:

a. Online shopping - takes the user to the home page to browse and add items to their shopping cart
b. Delivery - displays a Google map for the user to check the distance and route between their chosen delivery address and one of our shipping facilities
c. Fire sale shopping - takes the user to a special page to shop for items currently on sale



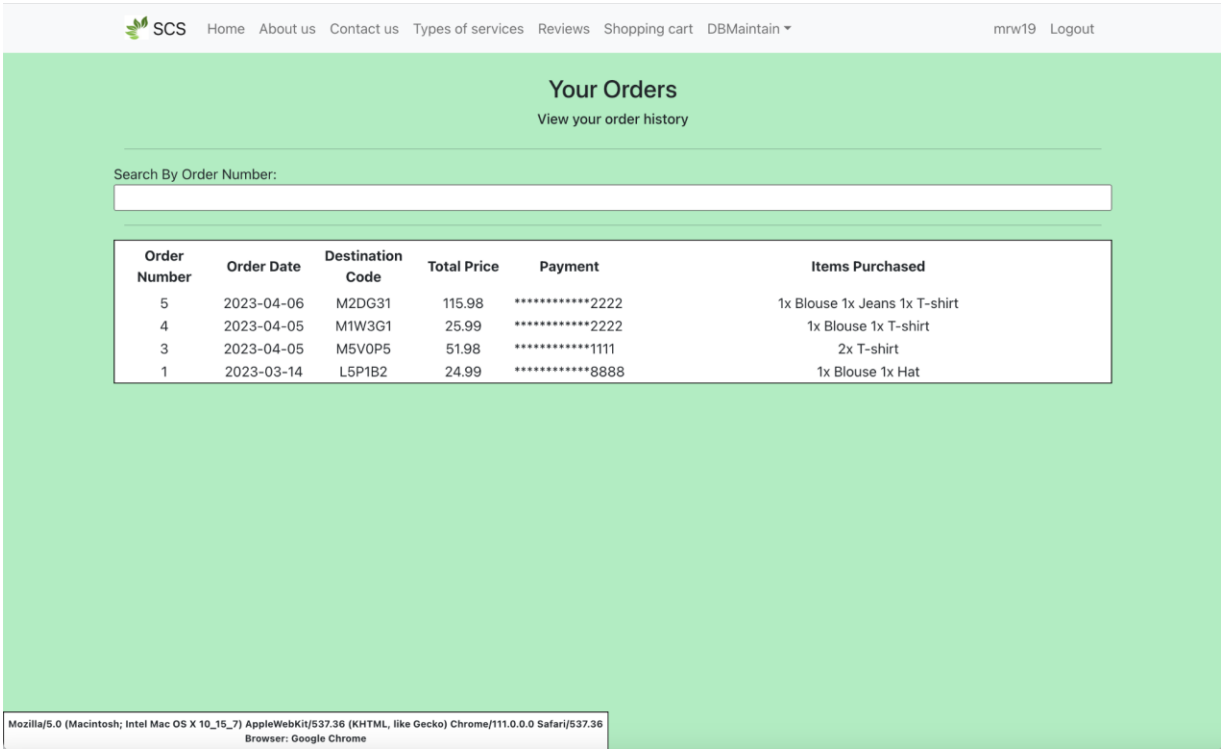
Fire Sale

The fire sale displays all items that are on sale (reduced price) and functions the same as the Home page: users drag the item into the shopping cart to add it to their cart.

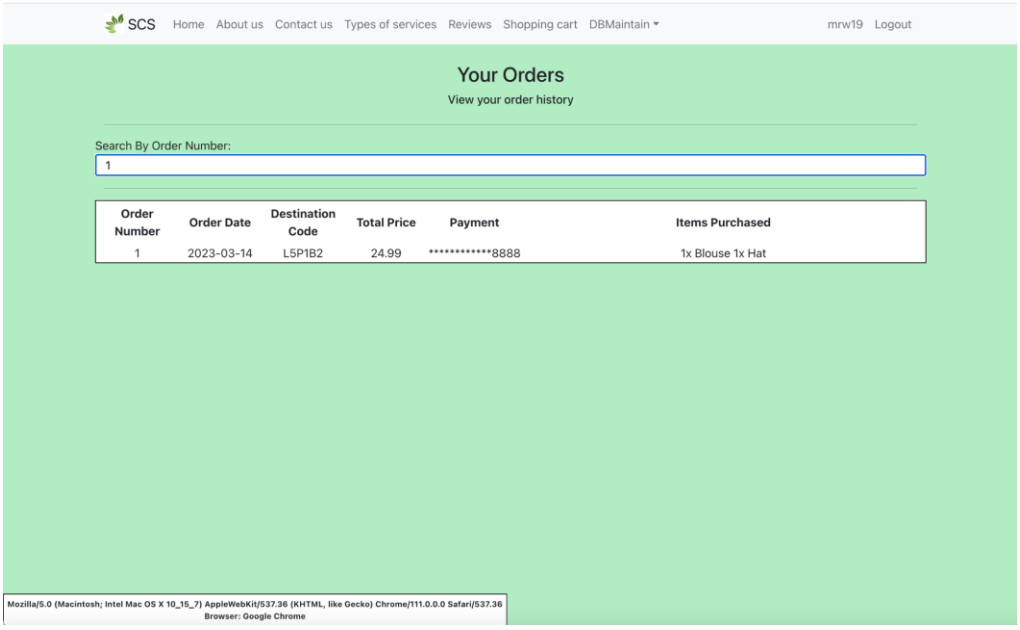


Order History

Users are able to view their order history via clicking their profile in the navigation bar. On the page, there is a search bar where users can search for a specific order by its order number and a table that contains all the orders (if the search bar is empty, it displays all the user's orders, otherwise, it contains the searched order if it exists).

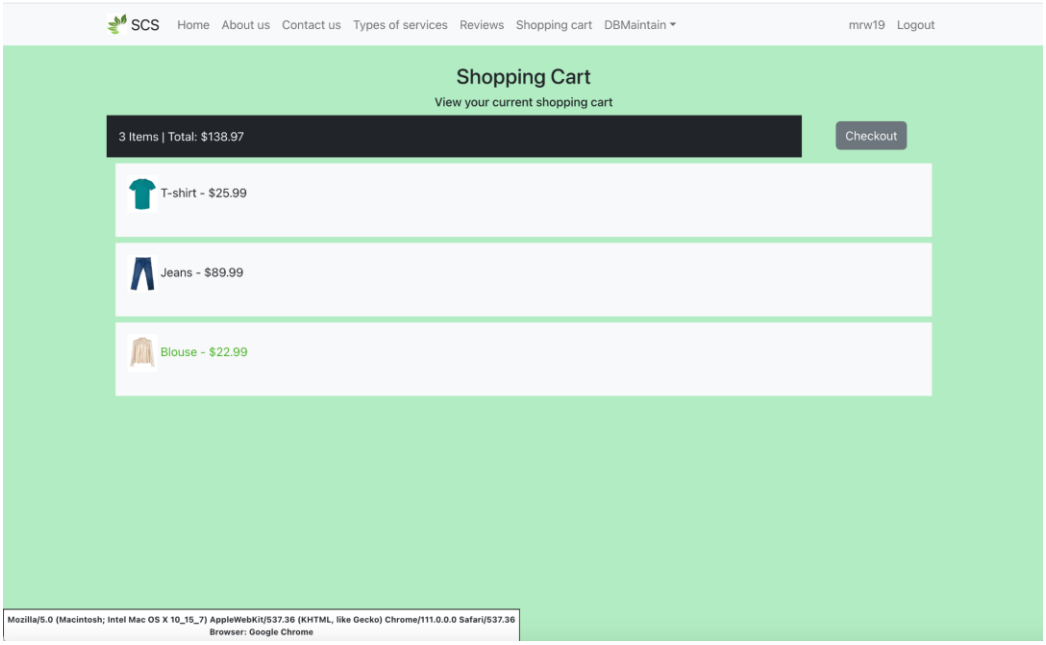


The table contains the order number, order date, destination code, total price, payment and items purchased.

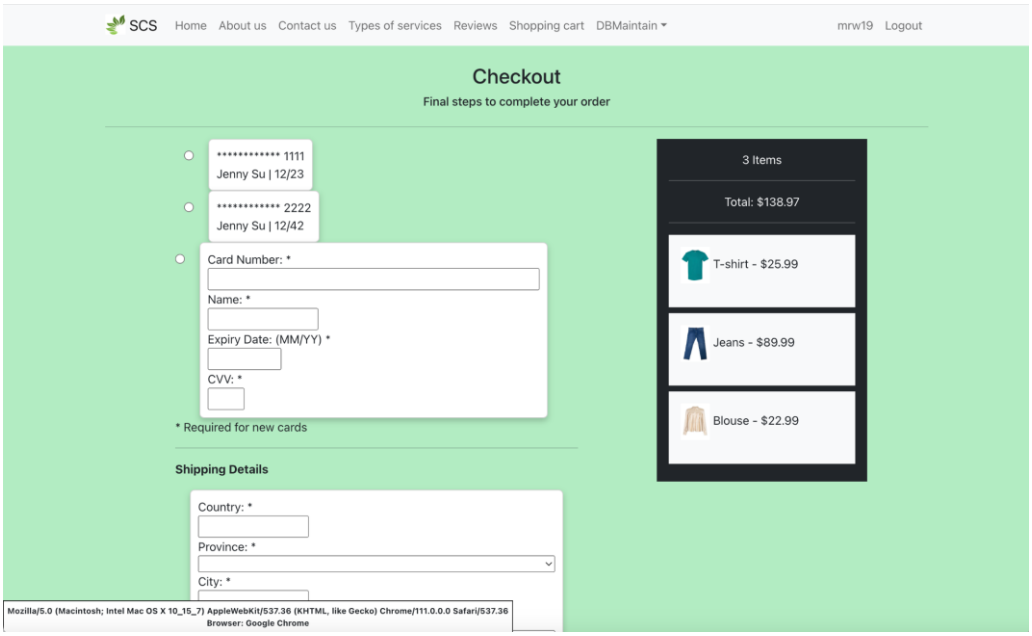


Shopping Cart

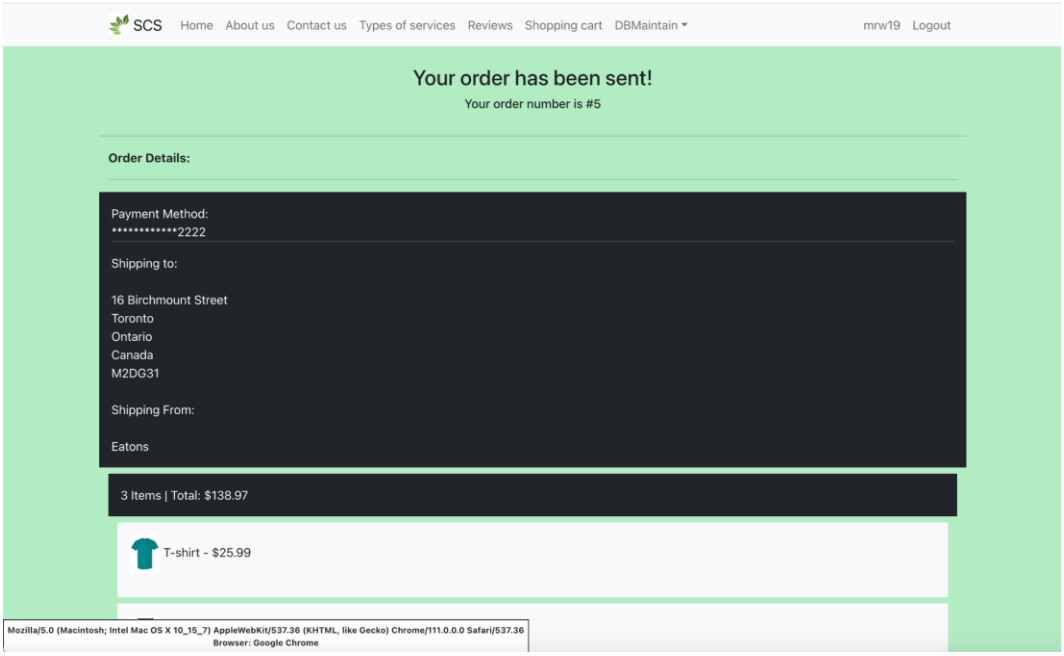
The user can view their cart, checkout and place their order from the shopping cart tab. The main shopping cart page lets the user view their cart - see the total price and number of items in their cart. When the user is ready, they can press the checkout button to begin the process of submitting their order.



The check out process begins with a form asking the user for which payment method they want to use and their shipping address, as well as view their cart on the right side of the screen and a map at the bottom of the page. For the payment method section, the user can choose a saved card or add a new card. Any new cards will be saved for quicker checkout later.

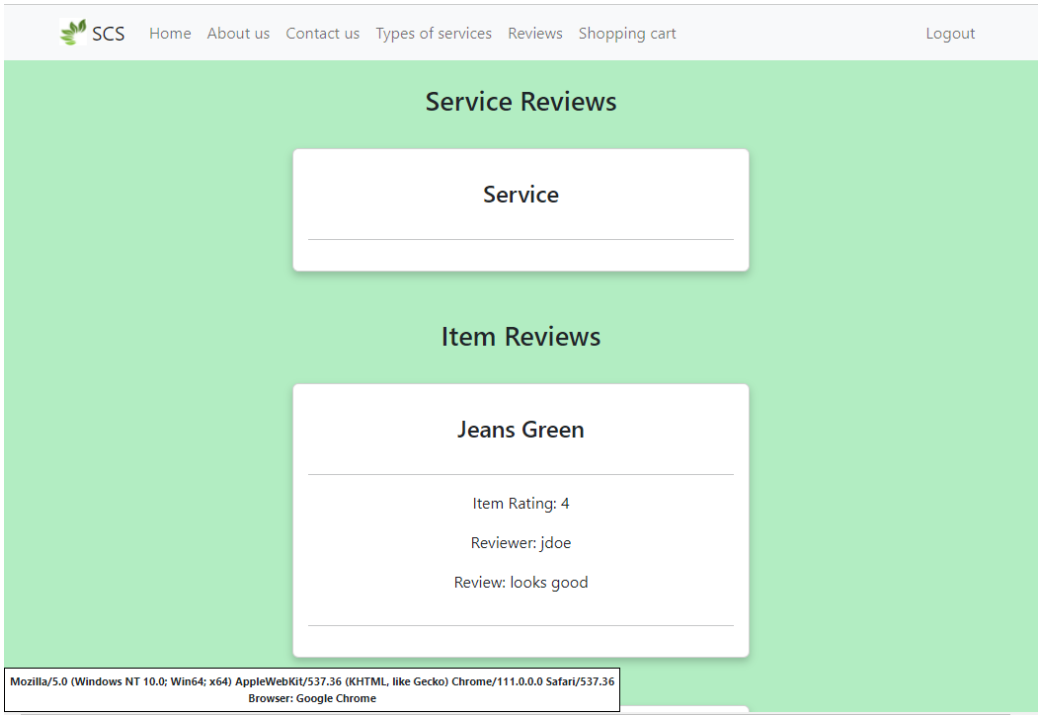


Upon placing the order, the user is taken to the review page which displays which payment method they used, where it is shipping, the items that they ordered and their order number. There is also a map on the bottom to display the shipping route.

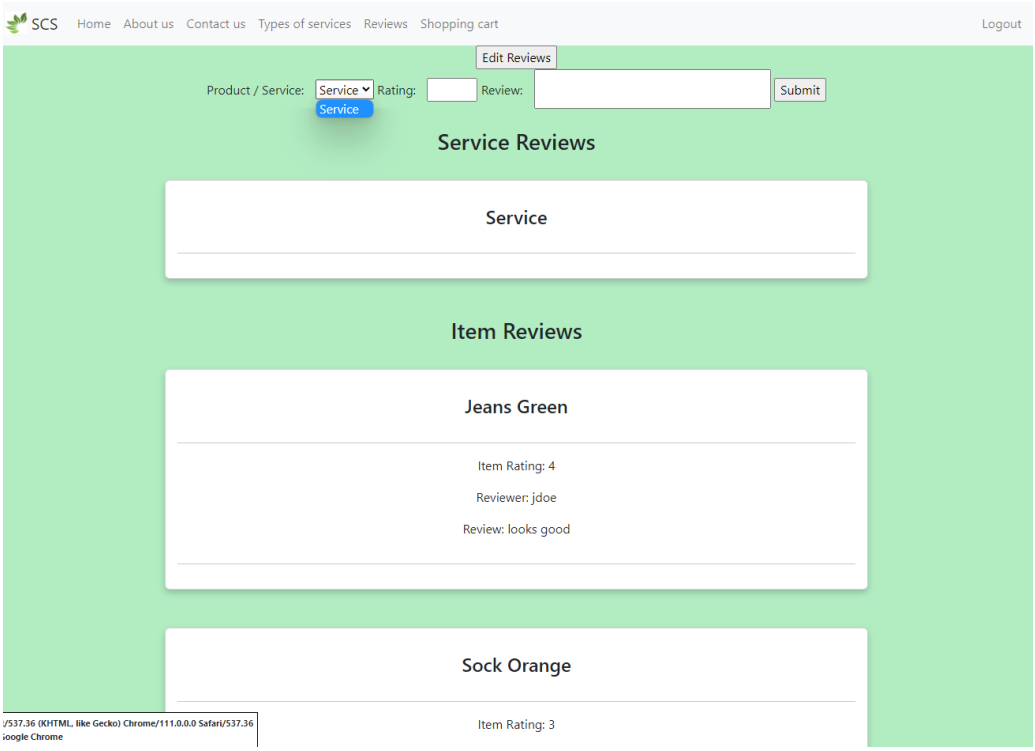


Reviews

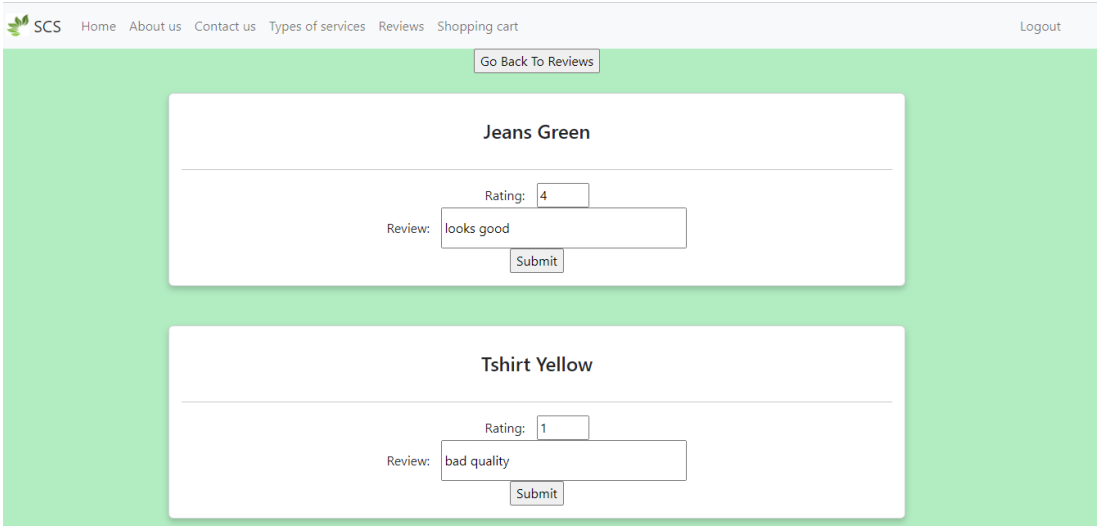
The user can view, create, and edit reviews on this page. If no user is logged in, the review page will only show reviews made by all existing users, but this is not possible as all users should login first to access the app.



If a user logged in, then the following page will be shown:



The options in the dropdown list are based on the items that the user has purchased and are not reviewed yet. Moreover, when a user signs up, the user can create a review on the service. For instance, in the image above and below, the user has purchased and reviewed “Jeans Green” and “Tshirt Yellow”, but the user has not create a review on the service. That’s why only the “Service” option appeared on the dropdown list.



The user can edit the review he made by clicking the “Edit Reviews” button, which will bring the user to a new page as shown above. On that page, the user can edit their input and click the “Submit” button. Everytime a user creates or updates a review, the app will bring the user to a page that responds to the user's feedback and provide a button to go back to the review or edit review page.

Design and Implementation of the Database

Database Tables

In our web application, we use ten tables and their relation between other tables can be seen in the Database Schema Diagram below. Each table with its fields and function will be explained below.

Shopping Table

Shopping table consists of the following fields: receipt_id (Primary Key), store_code, and total_price. Store_code is used to keep the store code of different store branches, and total_price is used to keep the total price of items purchased by a user. This table is referenced by the Order Table using foreign key.

Truck Table

Truck table consists of the following fields: truck_id (Primary Key), truck_code, and availability_code. Truck_code is used to represent different trucks for delivery, the availability of that truck is based on the value of availability_code where 1 means available, and 0 means otherwise. Moreover, truck_code is unique as it can't have two different availability codes at the same time. This table is referenced by the Trip Table through using foreign key.

Trip Table

Trip table consists of the following fields: trip_id (Primary Key), source_code, destination_code, distance, truck_id (Foreign Key), price. We use source_code and destination_code to represent postal code and distance to represent the distance between both postal codes in kilometers. Truck_id is used to reference the truck table and show which truck will be used for the delivery. Finally, the price field is used to show the delivery fee. This table is referenced by the Order Table using foreign key.

User Table

User table consists of the following fields: user_id (Primary Key), full_name, telephone, email, home_address, city_code, login_id, salt, user_password, balance. In this table, telephone, email, login_id, salt, and user_password must be unique. Before iteration IV, user_password stores the original password, but in iteration IV it stores the hashed version. This table is referenced by many tables such as Order Table, Review Table, Payment Table, and Purchased Item Table.

Item Table

Item table consists of the following fields: item_id (Primary Key), item_name, item_price, made_in, department_code, image_name. Image_name is used to keep the image file name for that item. This table used to keep track of all of our products (both non-Fire Sale and Fire Sale) and also referenced many tables such as Order Table, Review Table, Item Sale Table and Purchased Item Table.

Item Sale Table

Item sale table consists of the following fields: item_sale_id (Primary Key), item_id, sale_price, expiry_time. This table is used to store items that are on sale, it is shown in the Fire Sale Shopping Page (Service C). The sale_price field stores the new price (discount price) for the item that the item_id field references. The expiry_time field is used to store the date where this item will remain on sale.

Review Table

Review table consists of the following fields: review_id (Primary Key), item_id, user_id, RN, review. Item_id and user_id are foreign keys referring to the Item Table and User Table. The purpose of those fields is to show which user reviews which item. RN and review fields are the rating number and the review itself. The review field can be NULL as some users may prefer to only rate the item without making any comments. This table is used when showing the reviews in the review and edit review page.

Payment Table

Payment table consists of the following fields: payment_id (Primary Key), user_id, cardholder_name, card_number, cvv_code. This table essentially contains all information related to a credit card. This table references the User Table since a user can have multiple payment methods (n number of credit or debit card). This table is also being referenced by Order Table.

Order Table

Order table consists of the following fields: order_id, date_issued, date_received, total_price, payment_id, user_id, trip_id, receipt_id. The total_price field combines both the price for the items and delivery. This table references the Payment Table to show how the payment was made. It also references the User Table to show the user who made this order. The Trip Table is also referenced to show the trip to deliver the items to the user. Finally, it also references the Shopping Table to get the total price for the items.

Purchased Item Table

Purchased item table consists of the following fields: purchased_item_id (Primary Key), item_id, user_id, and order_id. This table basically stores the list of items in each order made by the user. The item_id and user_id when combined with the Review Table can create a list of reviewable items for the user. The item_id and order_id can also be used to show the list of items purchased in specific order, which is shown in the Order History page.

Using the Database in the Web Application

In our web application, each table has its own PHP class (located in Models.php) with its fields being the class property. Each property has its Getter and Setter methods except the primary key property which only has a get method. Each class implements an interface called Database which has the following function: insert(), delete(), update().

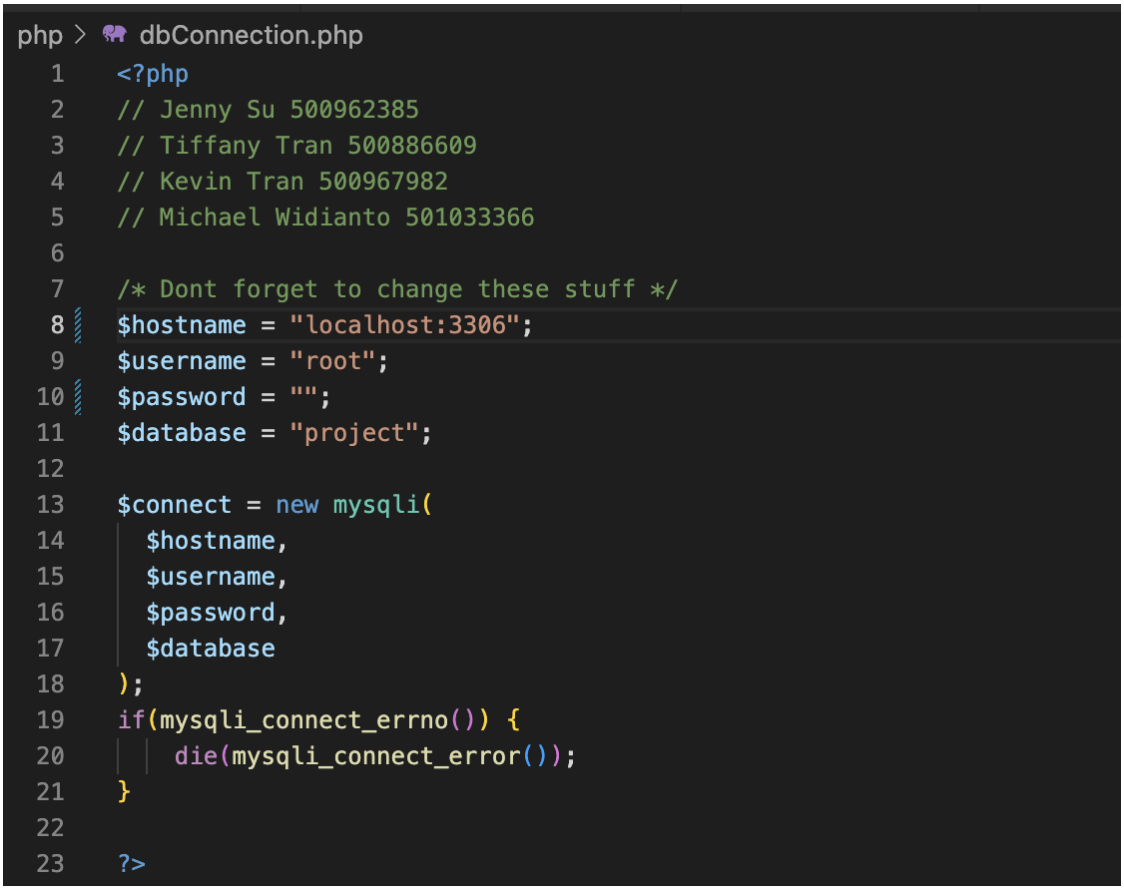
Each class has a minimum of two constructor methods, the first constructor (without primary key property parameter) is used when a user creates the object. For instance, when a user makes an order, an Order Object is created without its primary key value, then the insert() function is called to store it in the database. Another example is when an admin uses the Insert Page in DBMaintain to create new records. An object that is made using the first constructor can't run the delete() and update() method because technically it's not in the database yet.

The second constructor (with primary key property parameter) is used when we are getting records from the database to show data to the user. For instance, all of the records shown in Select Page of DBMaintain are transformed to their own Classes using the second constructor. Another example is when we show all of the reviews in the Review Page. An object that is made using the second constructor can't run the insert() page since it's already in the database. When updating a record, we only need to use the Setter methods to change the data, then call the update() function.

```php
/**
 * Enter 2 parameter ($store_code, $total_price) for creating the object from user input.
 * Enter 3 parameter ($receipt_id, $store_code, $total_price) for creating the object from database.
 */
public function __construct()
{
    $arguments = func_get_args();
    $numberOfArguments = func_num_args();

    if (method_exists($this, $function = '__construct' . $numberOfArguments)) {
        call_user_func_array(array($this, $function), $arguments);
    }
}

public function __construct2($store_code, $total_price)
{
    $this->store_code = $store_code;
    $this->total_price = $total_price;
}

public function __construct3($receipt_id, $store_code, $total_price)
{
    $this->receipt_id = $receipt_id;
    $this->store_code = $store_code;
    $this->total_price = $total_price;
}
```

Environment (XAMPP)

The XAMPP environment acts as a local host to run MySQL servers (our database), which is important in our web application to fetch and post data from and to the database.



```php
<?php
// Jenny Su 500962385
// Tiffany Tran 500886609
// Kevin Tran 500967982
// Michael Widianto 501033366

/* Dont forget to change these stuff */
$hostname = "localhost:3306";
$username = "root";
$password = "";
$database = "project";

$connect = new mysqli(
  $hostname,
  $username,
  $password,
  $database
);
if(mysqli_connect_errno()) {
    die(mysqli_connect_error());
}

?>
```

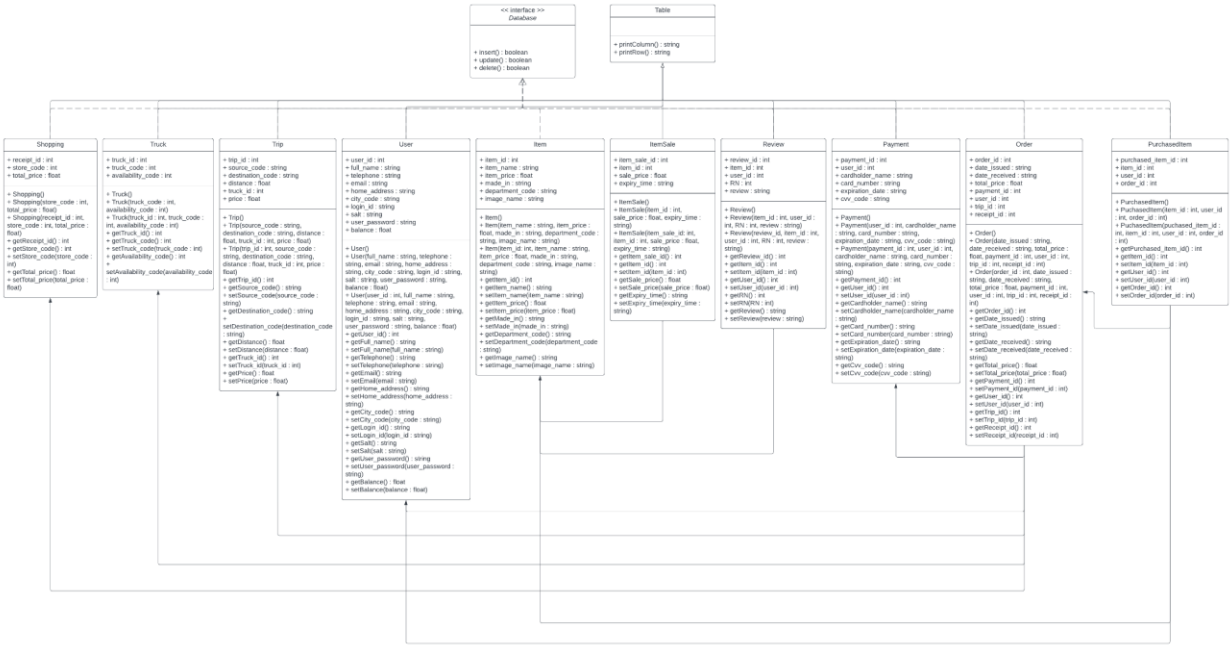
The XAMPP environment is declared in the DBConnection.php file. This file builds the connection from our React web application to the MySQL servers which are running on XAMPP.

Database Schema Diagram

Architecture of Application

Classes

We mapped all of our MySQL database tables to domain objects (models).



MVC Pattern

Our web application follows the model-view-controller architecture pattern. The model of the MVC pattern was implemented in php files such as in the files getCartItems.php or getItemID.php. These files dealt with the retrieval of data from the database and would pass it to the frontend via the controller. The database structure is defined in the Models.php file.

```php
<?php
header('Access-Control-Allow-Origin: *');
header("Access-Control-Allow-Headers: *");
include_once "submitQuery.php";
$item_firesale_id = json_decode($_GET['sale_item'], true);
$items = array();
for ($i = 0; $i < count($item_firesale_id); $i++){
    $item_id = $item_firesale_id[$i];
    $query = "SELECT itemsaletable.sale_price, itemtable.item_name, itemtable.item_price, itemtable.item_id, itemtable.image_name FROM itemsaletable INNER JOIN itemtable ON i
    $result = submitSelectQuery($query)[0];
    array_push($items, $result);
}
header('Content-Type: application/json; charset=utf-8');
echo json_encode($items);
?>
```

The controller can be seen in various parts of the code in .jsx files in the form of *useEffect axios.get* commands that retrieve data from the model layer and make that data accessible by the view. The data is then set and is used by the view to dictate what is presented to the user.

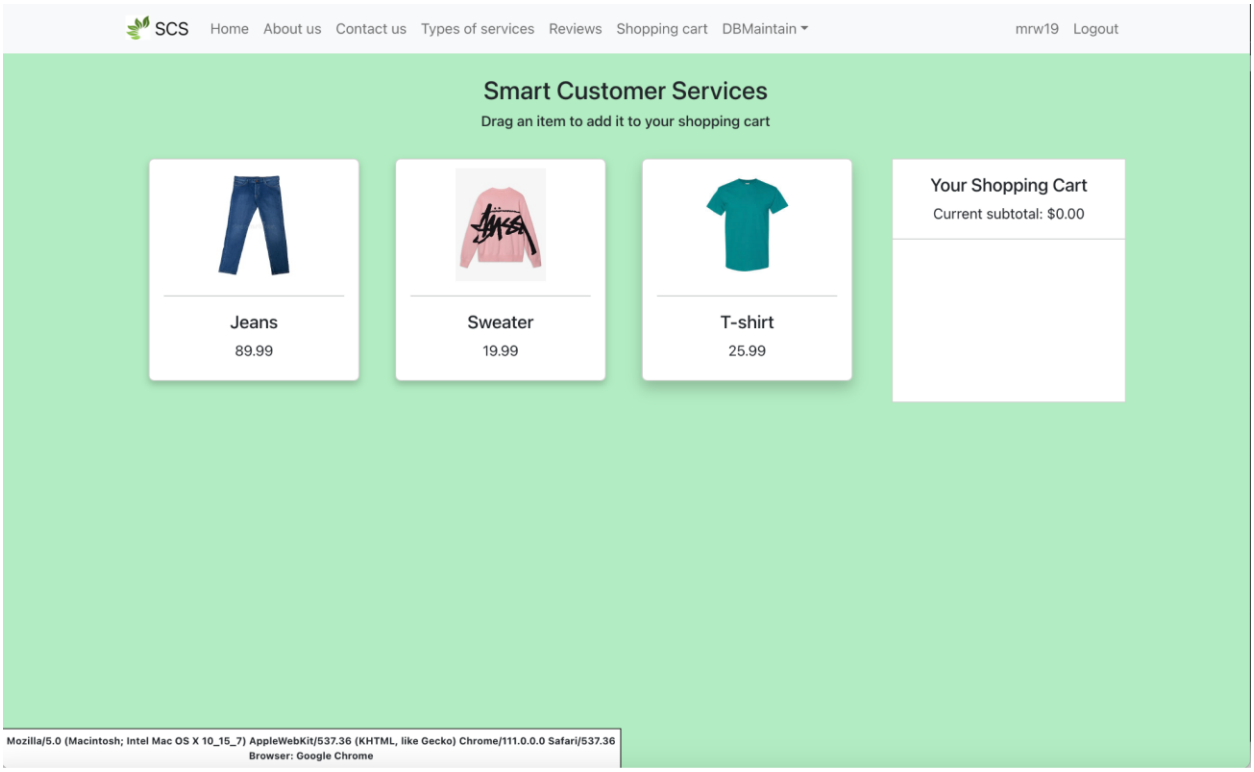
```
useEffect(() => {
    setCartItems([]);
    if (cookies.fire_items && cookies.fire_items.length > 0) {
        axios.get("http://localhost:8000/getItemForCart.php", {params: {sale_item:JSON.stringify(cookies.fire_items)}}).then((response) => {
            setCartItems(response.data);
        });
    }
}, []);
```

The view is defined in the various .jsx files in the src/components folder. Each .jsx file has a return statement that contains the HTML code used to construct a portion of the web application's user interface. JavaScript is also embedded in these files to handle user interactions with the user interface (e.g. drag and drop, clicking a button, etc). The file then ends with an export statement that allows App.js to import the element created.

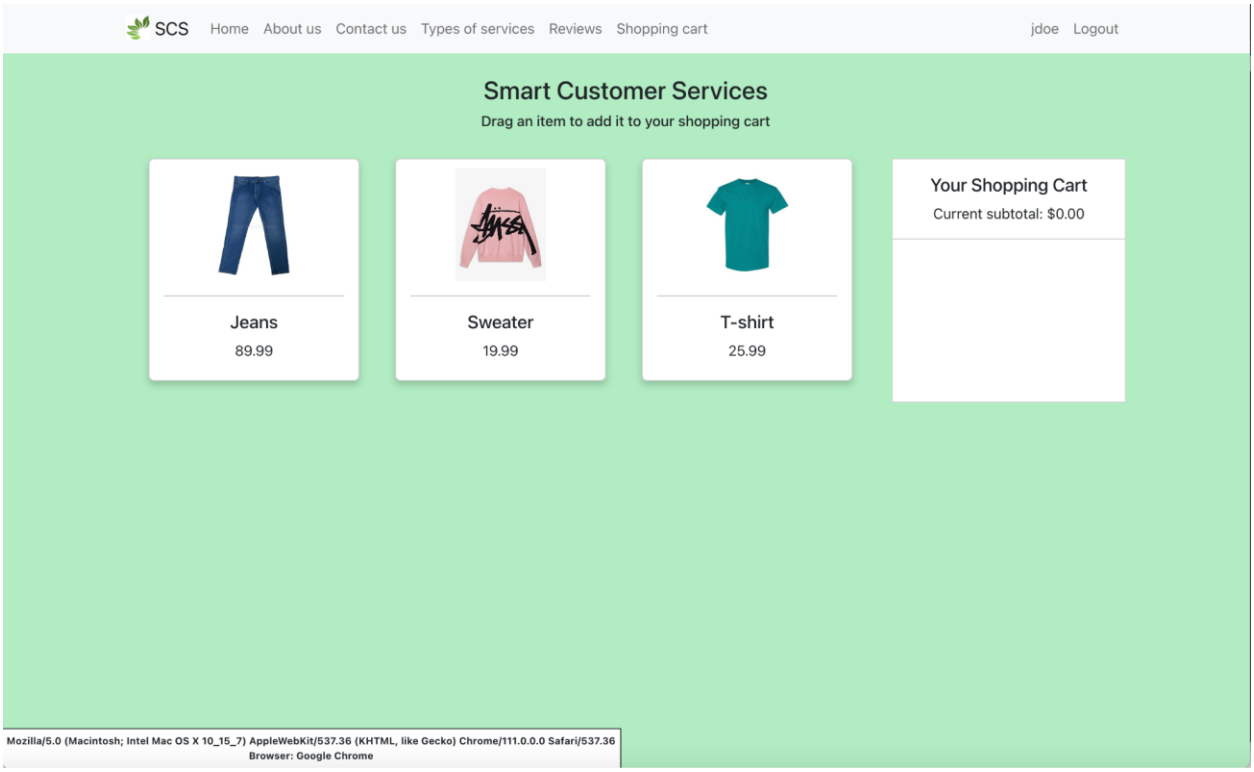
```
53
54      return (
55          <div id="browserDetection" style={divStyle}>
56              <p id="userAgent" style={divStyleP}></p>
57              <p id="browser" style={divStyleP}></p>
58          </div>
59      );
60  }
61
62  export default BrowserDetection;
```

Implementation of Two Modes of Operation

The web application sees two modes of operation: admin mode and normal mode.
The admin mode enables the visibility of the DBMaintain page. The DBMaintain page allows users to view and change the database directly by using special database functions. In this mode, the admin can: delete, update and insert rows in each table, and view the tables.
The admin mode is only accessible through authorized individuals, which the web application will identify via the user cookie. When a user is confirmed to be an admin, the web application will grant the user the admin privileges which they can navigate to through the navigation bar.



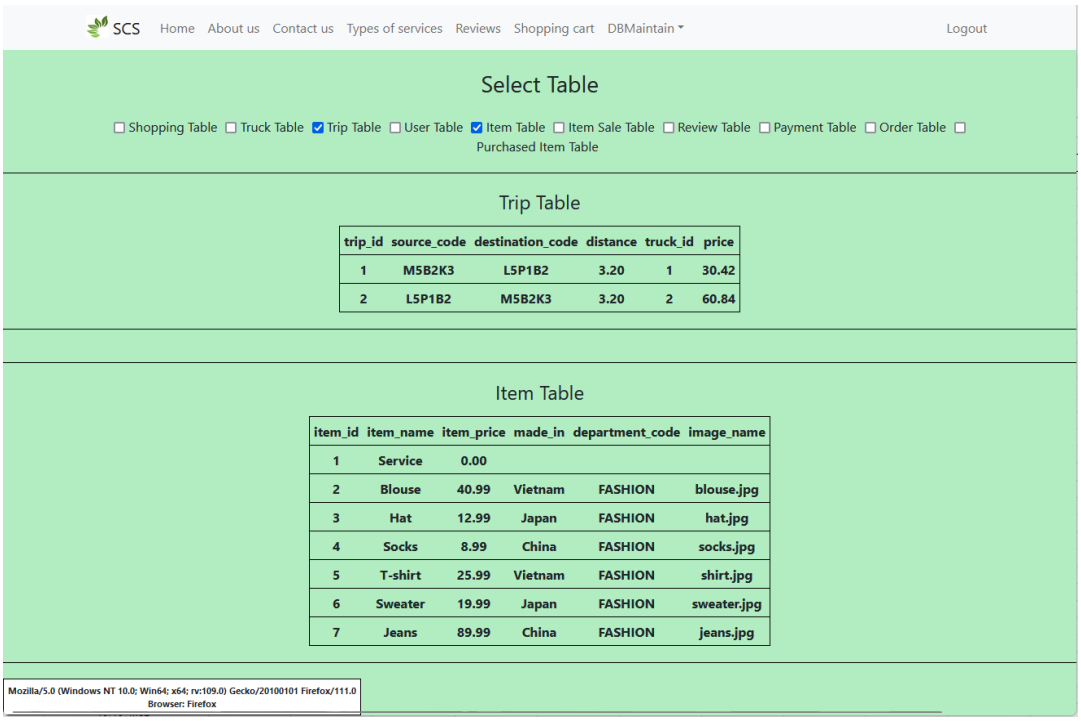
Admin Mode



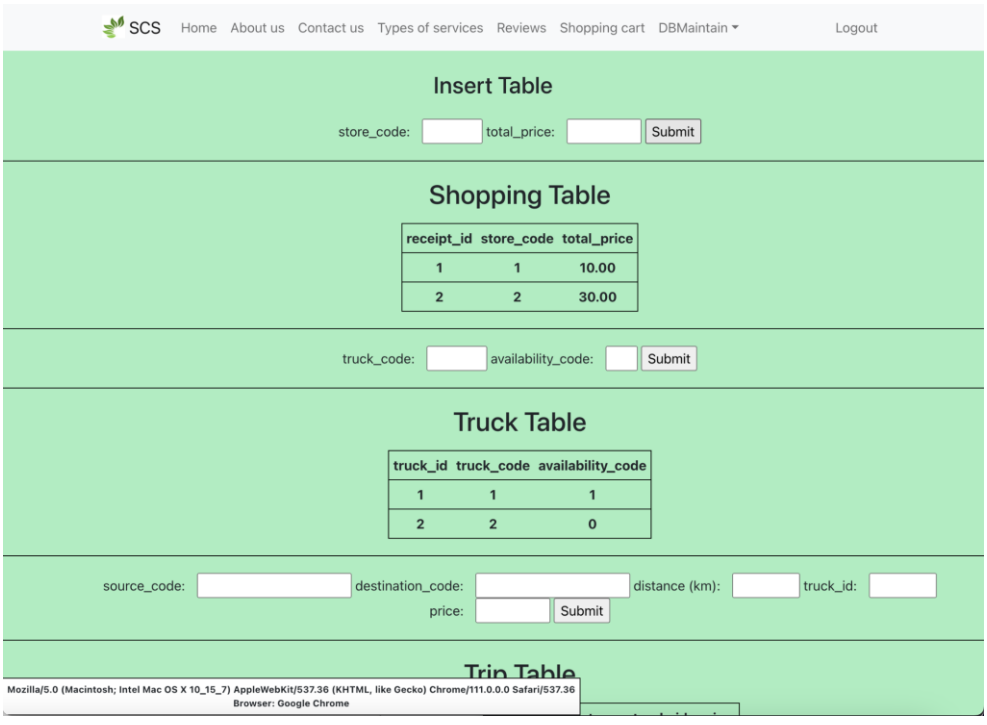
Normal Mode

Admin Functions (DBMaintain)

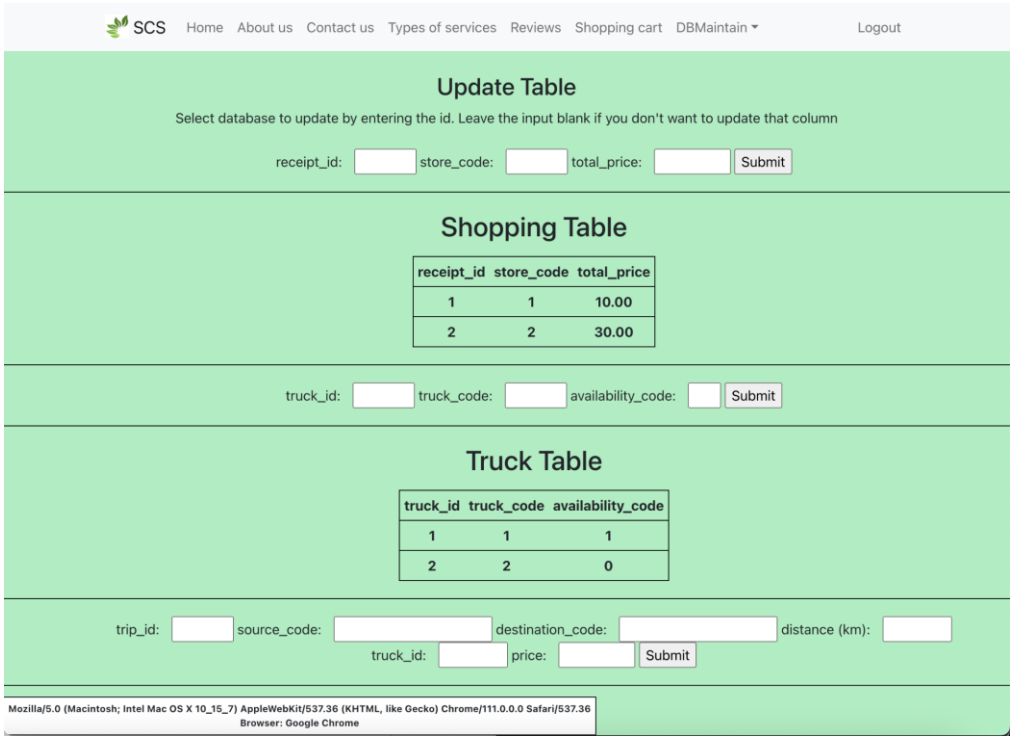
Select



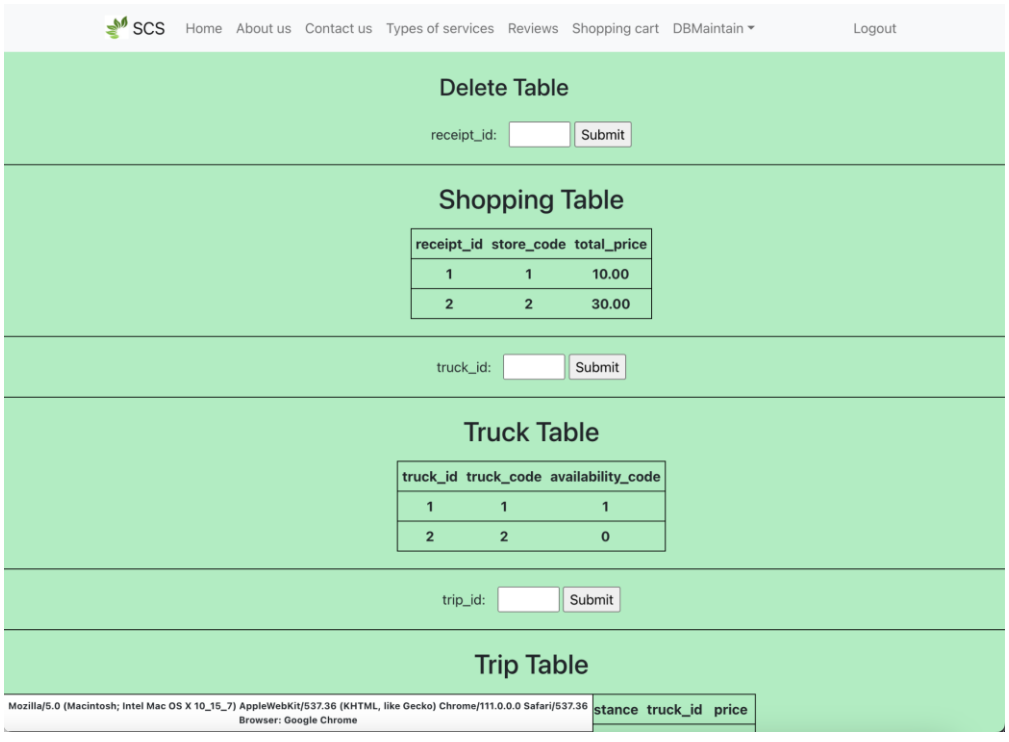
Insert



Update

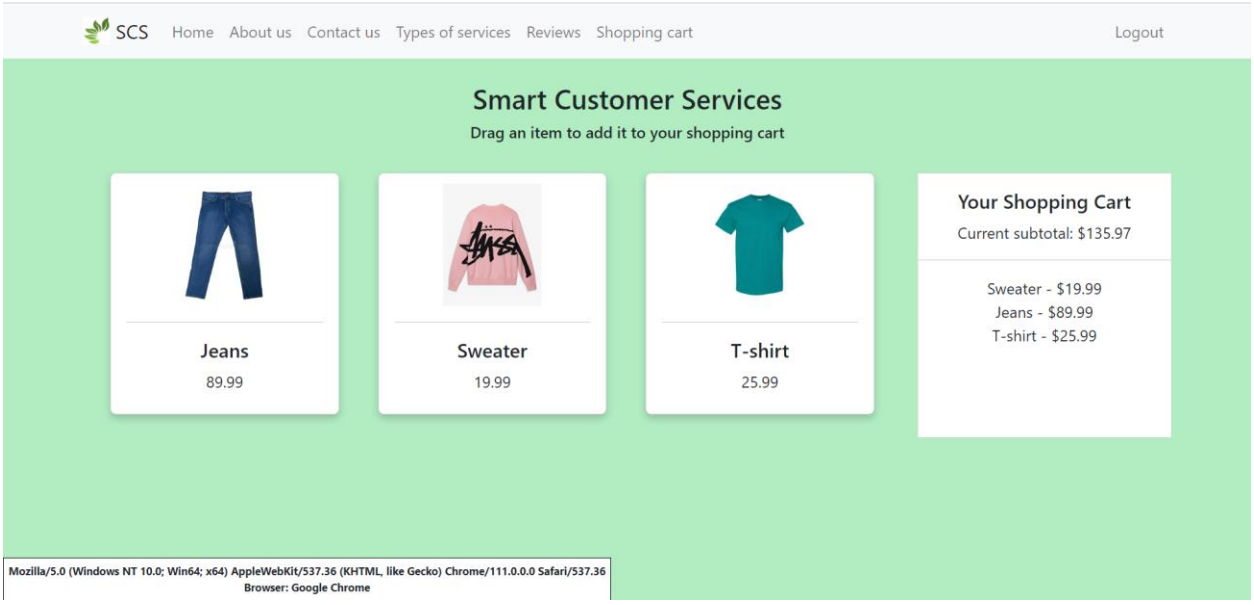

SCS Home About us Contact us Types of services Reviews Shopping cart DBMaintain Logout
Update Table
Select database to update by entering the id. Leave the input blank if you don't want to update that column
receipt_id: store_code: total_price: Submit
Shopping Table
receipt_id store_code total_price
1 1 10.00
2 2 30.00
truck_id: truck_code: availability_code: Submit
Truck Table
truck_id truck_code availability_code
1 1 1
2 2 0
trip_id: source_code: destination_code: distance (km):
truck_id: price: Submit
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36
Browser: Google Chrome

Delete

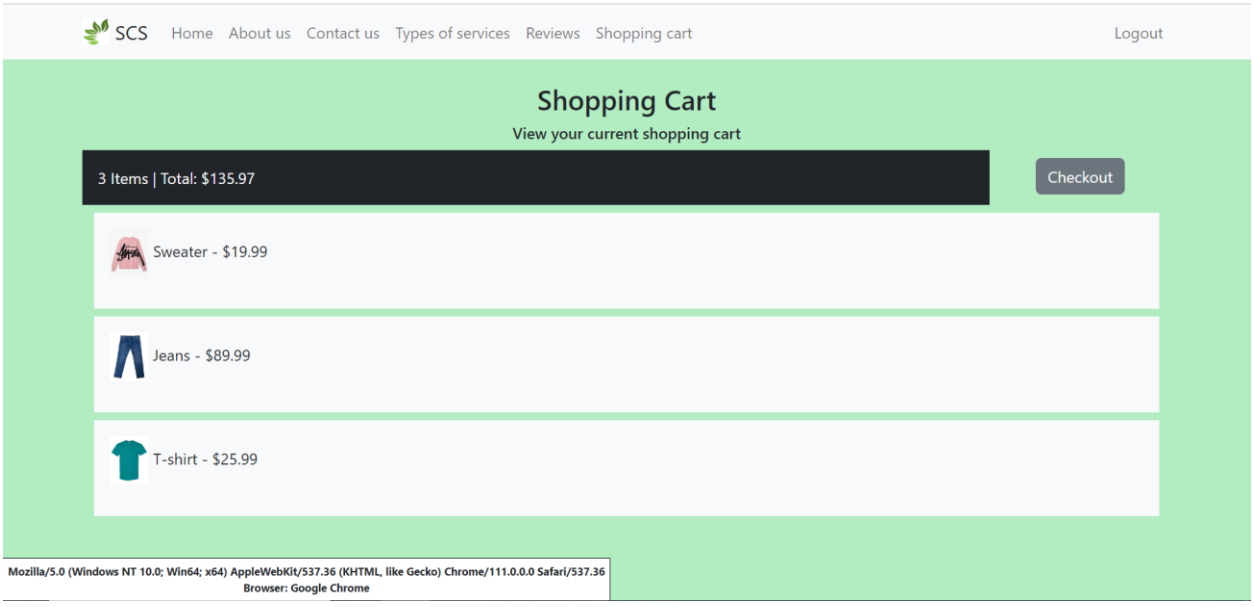

SCS Home About us Contact us Types of services Reviews Shopping cart DBMaintain Logout
Delete Table
receipt_id: Submit
Shopping Table
receipt_id store_code total_price
1 1 10.00
2 2 30.00
truck_id: Submit
Truck Table
truck_id truck_code availability_code
1 1 1
2 2 0
trip_id: Submit
Trip Table
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36
Browser: Google Chrome
stance truck_id price

Implementation of Service A and Service B

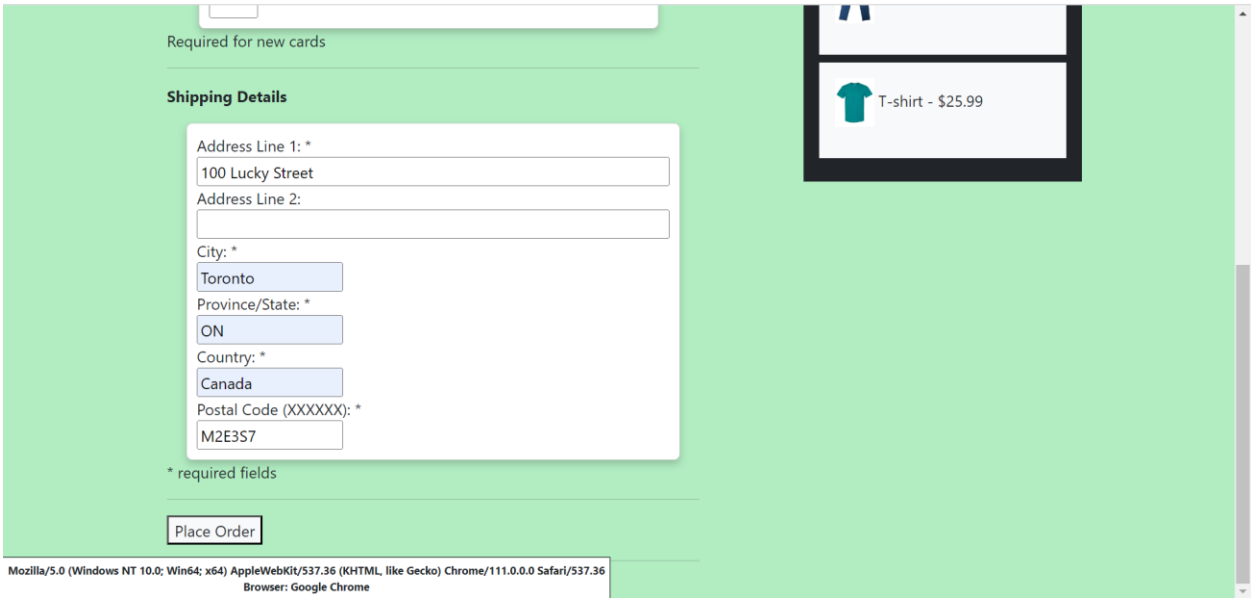
Service A allows the user to shop online for a selected department, in our case we developed the Clothing department. Service A was implemented using several screens, starting with the homepage where the user can browse clothing items for sale and add the items to their shopping cart, by dragging and dropping the item image into the cart on the right side of the screen.



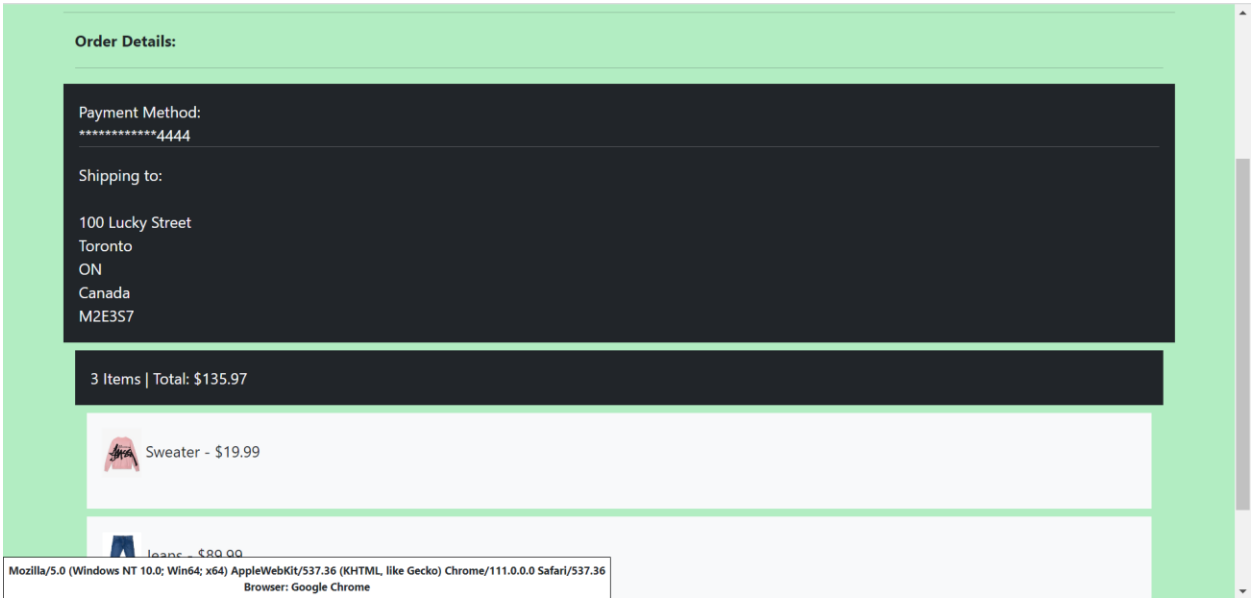
To checkout their items, the user navigates to the Shopping cart screen where they can review their order before clicking the “Checkout” button, which takes them to the screen to enter their payment information.



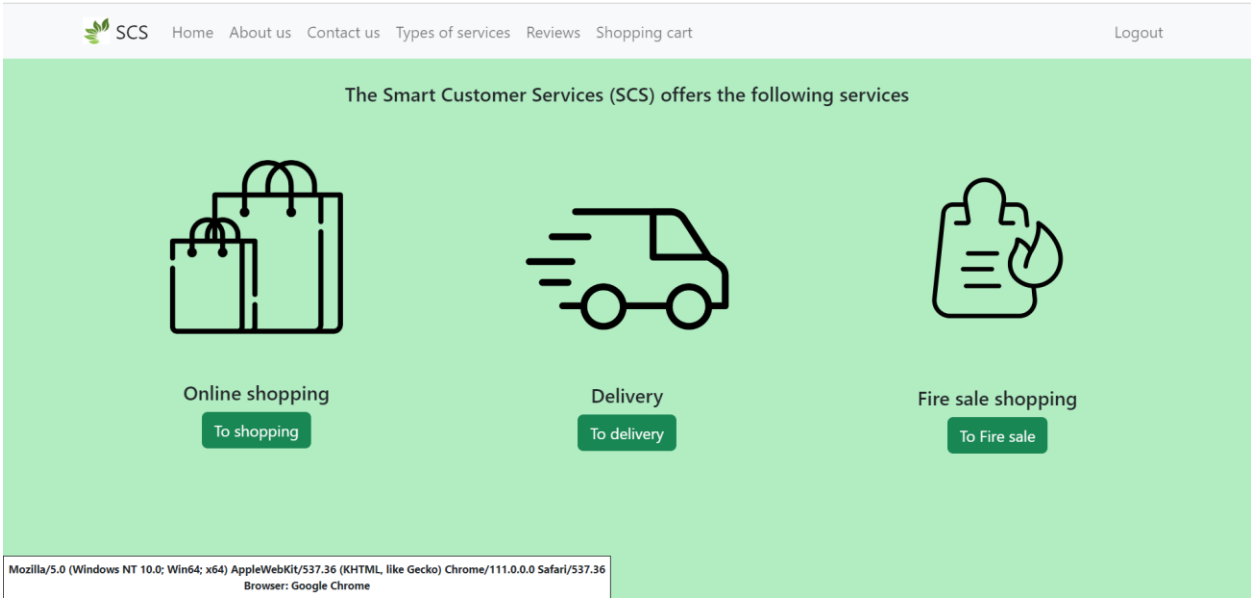
On this screen, there is an option to select a previously saved card to pay or to enter new payment information. After selecting a mode to pay and entering shipping details, the order is completed by clicking the “Place Order” button.



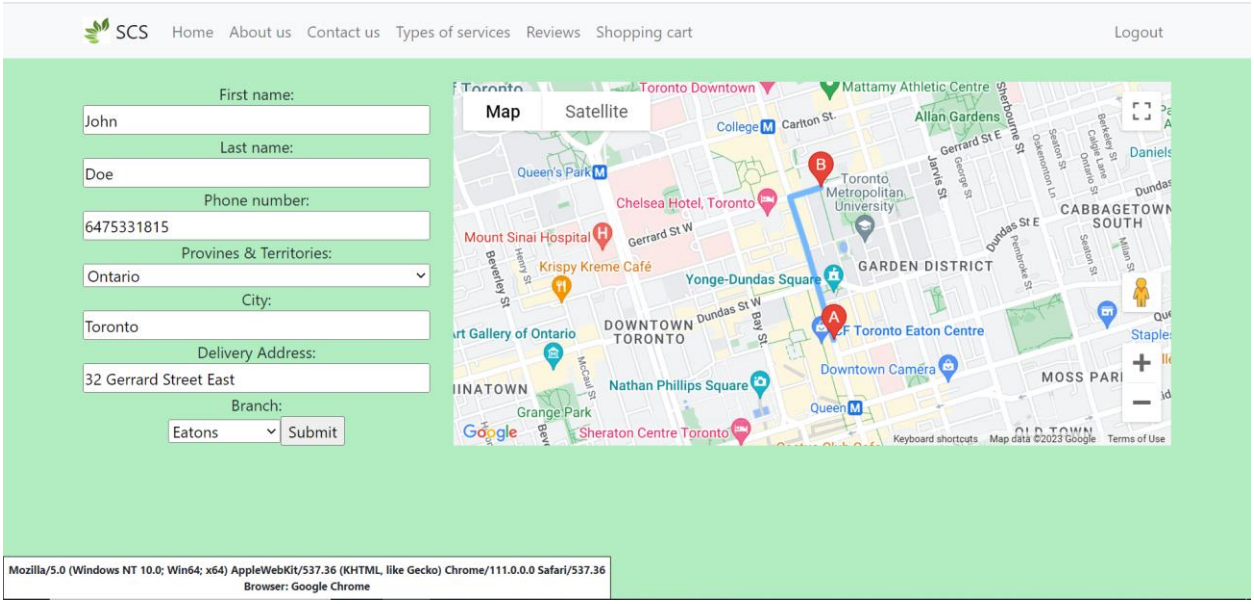
Finally, a confirmation screen displaying order details is displayed to the user. It shows the last four digits of the card used for payment, shipping information, and a list of the items purchased.



Service B displays the delivery route between an entered destination and a selected branch. To access Service B, the user will click on “Types of Services” on the navigation bar and click the “To delivery” button to access the maps screen.

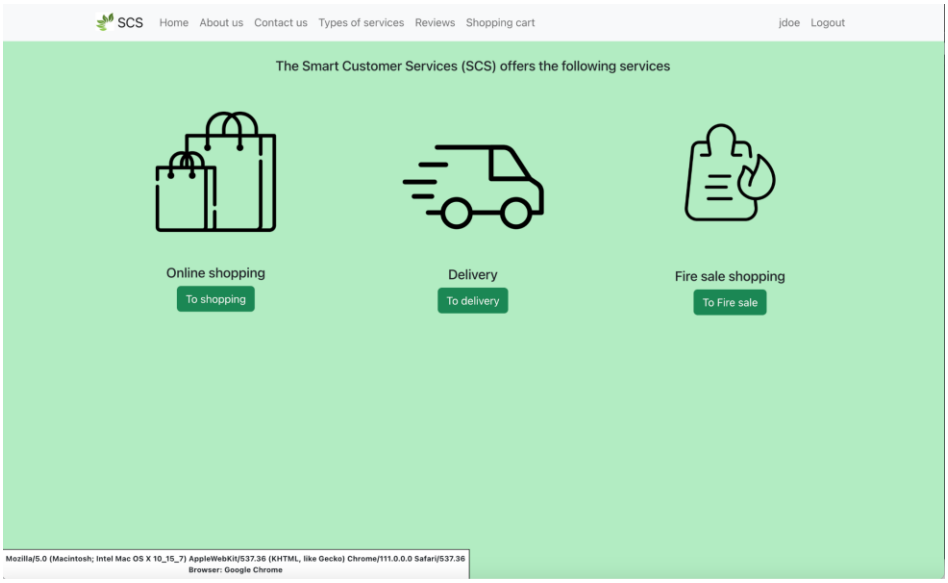


On the maps screen, the user will enter destination information, including their province/territory, city, and delivery address. They will then choose one of three available branches: Yorkdale, Eaton, and Dufferin Mall. Upon clicking the “Submit” button, a route from the chosen branch to the delivery address will be displayed on a Google maps view on the right side of the screen.

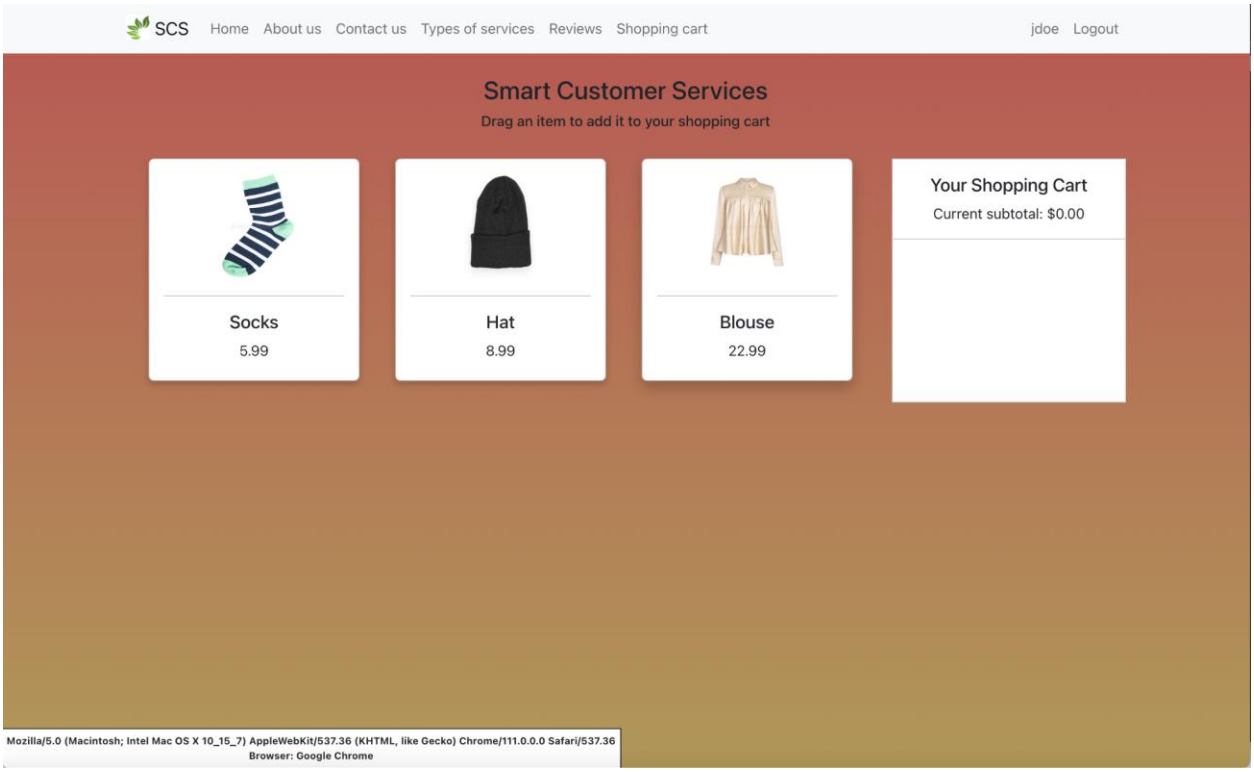


Implementation of Service C

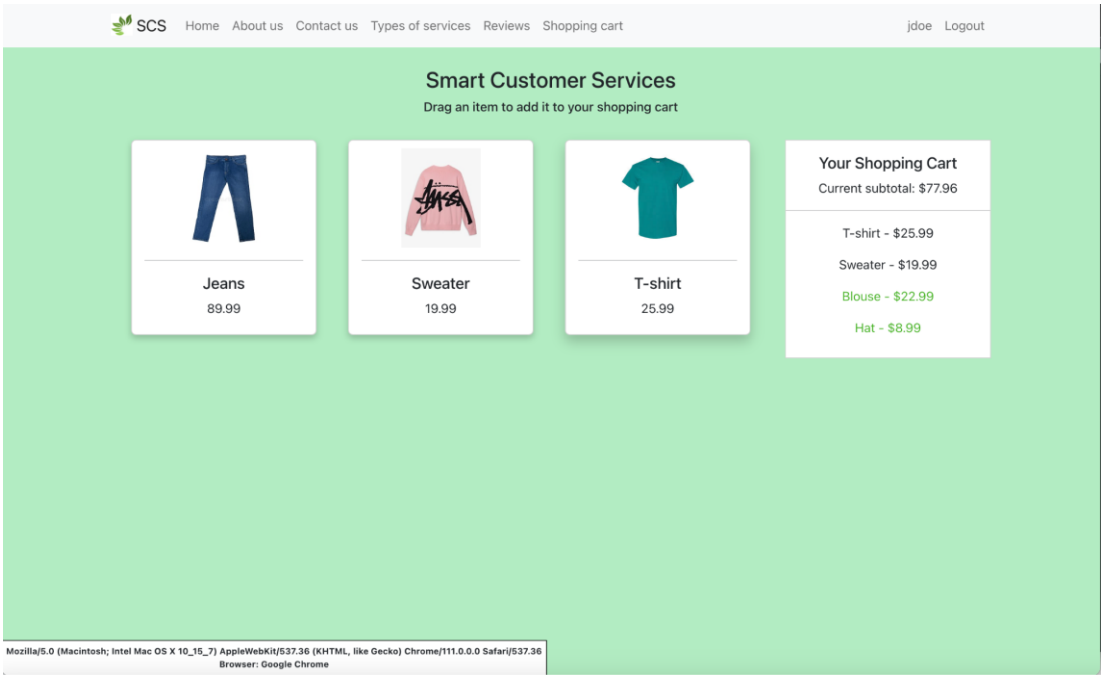
For Service C, we create an item sale page that displays all the items that are on sale. This page is accessible through the Types of Services page and clicking the to Fire Sale.



The interface for the fire sale page is the same as that of the home page which displays all items that are not on sale.



On sale items are differentiated from regular priced items via green text (which indicates that the item is on sale).



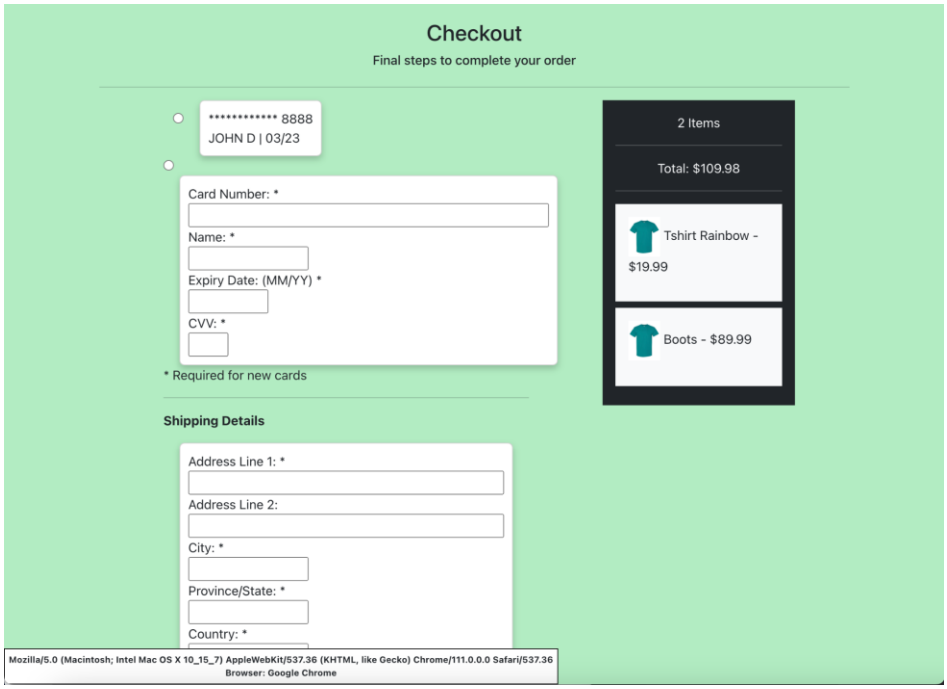
Shopping Cart Process

Saving selected items

When the user selects and adds an item to their cart, it is stored by a cookie that is visible to the entire web application. This cookie with the id: “items” is used throughout the shopping cart process: the shopping page, shopping cart, checkout and order review pages.

Placing the order

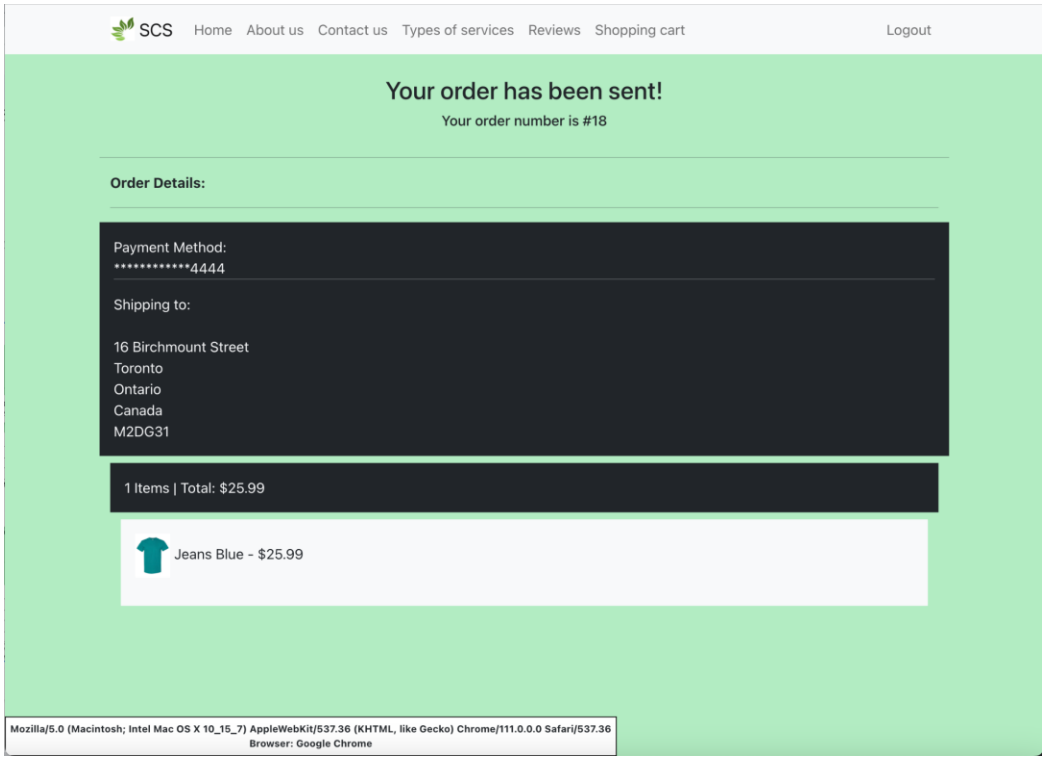
When the user places the order, all the items are sent to the back-end to store all the order information (date of order, total price, payment method, user who placed the order, trip information and receipt information).



The user must fill in all required fields to place their order. All shipping information must be filled in aside from the address line 2 and if the user chooses to use a new card, they are required to fill in the 4 card fields.

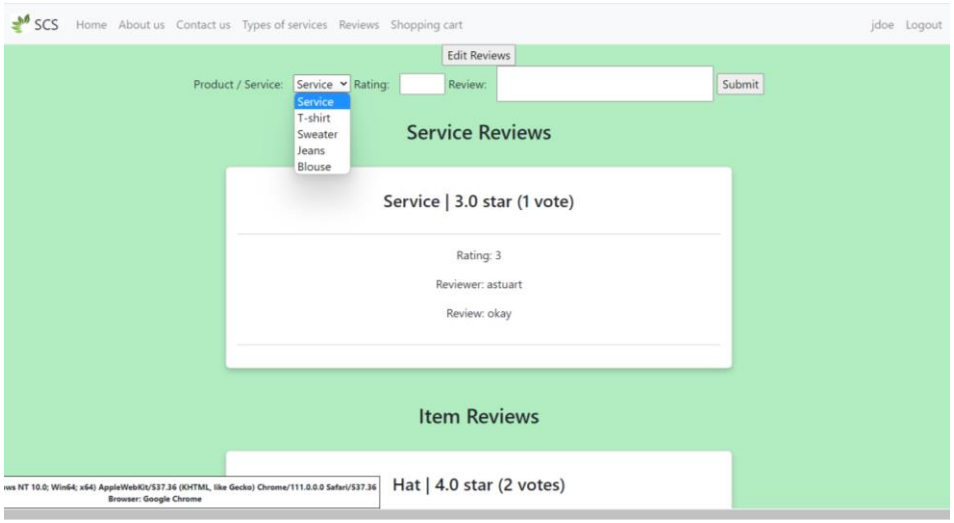
Invoice

The invoice contains order details: the order number, payment method, shipping address and items purchased. The layout is a simple page with all the information above in their own containers and a message letting the user know their order was sent.



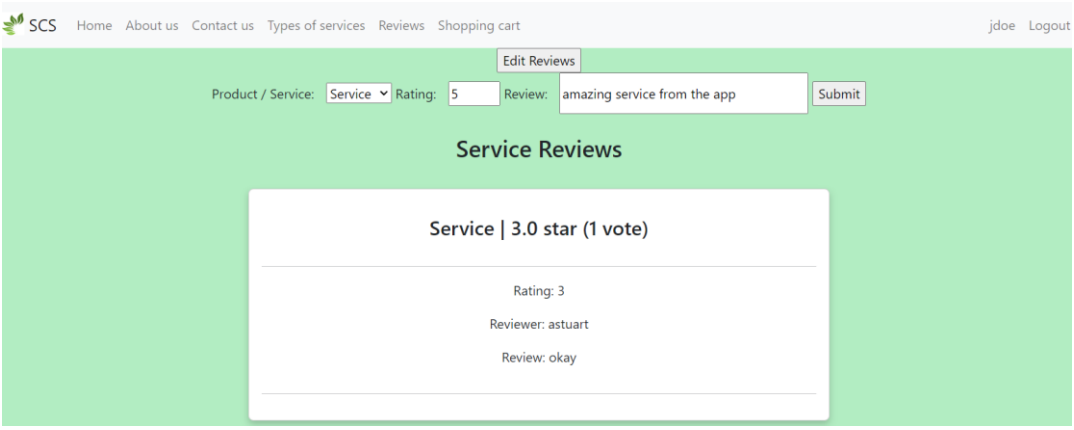
Review and Ranking System

A user can create a review when signing up or purchase a product. The list of items or services that appear in the dropdown box is based on the item that the user purchased and hasn't been reviewed yet.

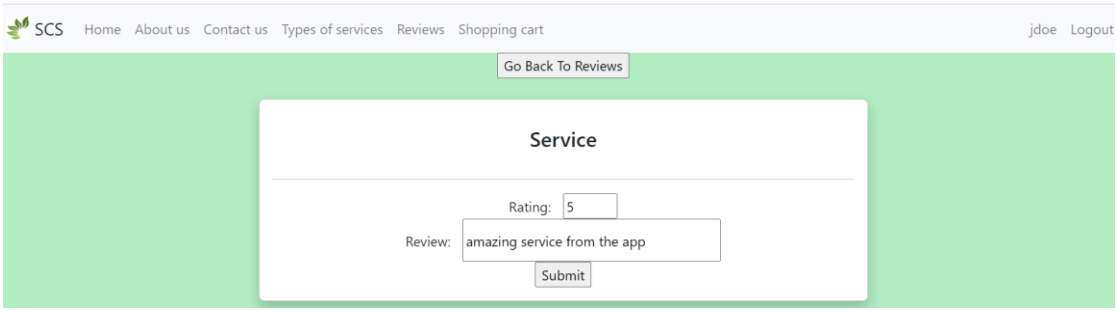


To create a review a user only needs to select the item to review, pick a rating from 1-5, write the review, and click the submit button. The user can also just rate it and not write a review. When a

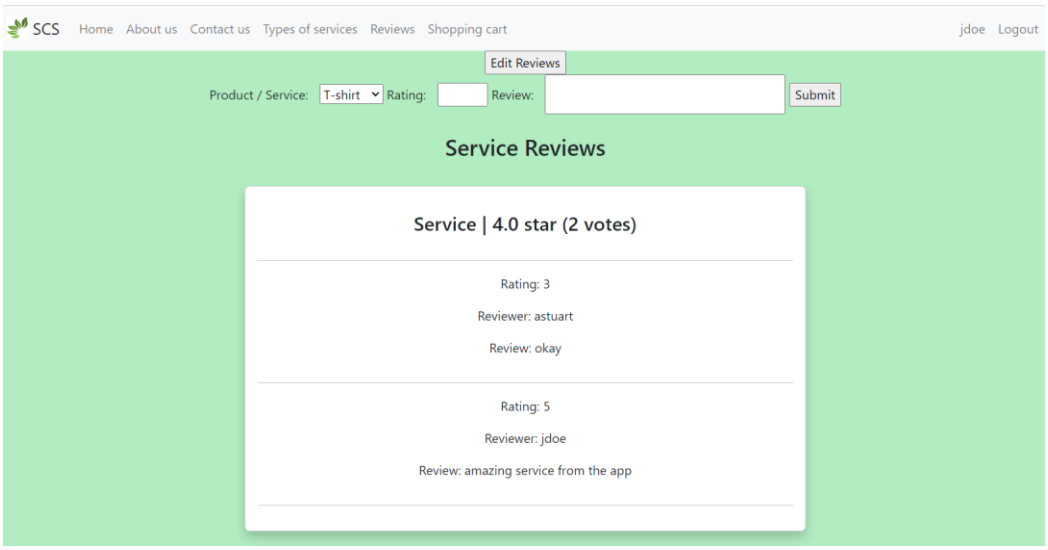
user submits the review, a Review Object will be made using the data given by the user, then the object calls the insert() function to submit the record to the database.



After the user creates a review, the user can also edit the review by clicking the “Edit Review”. In the edit review page, the user will be shown all of the reviews made by that user from the database. The way edit review works is similar to when the user creates a review, the only difference is that the object will call Setter methods to change the data, then call the update() function.



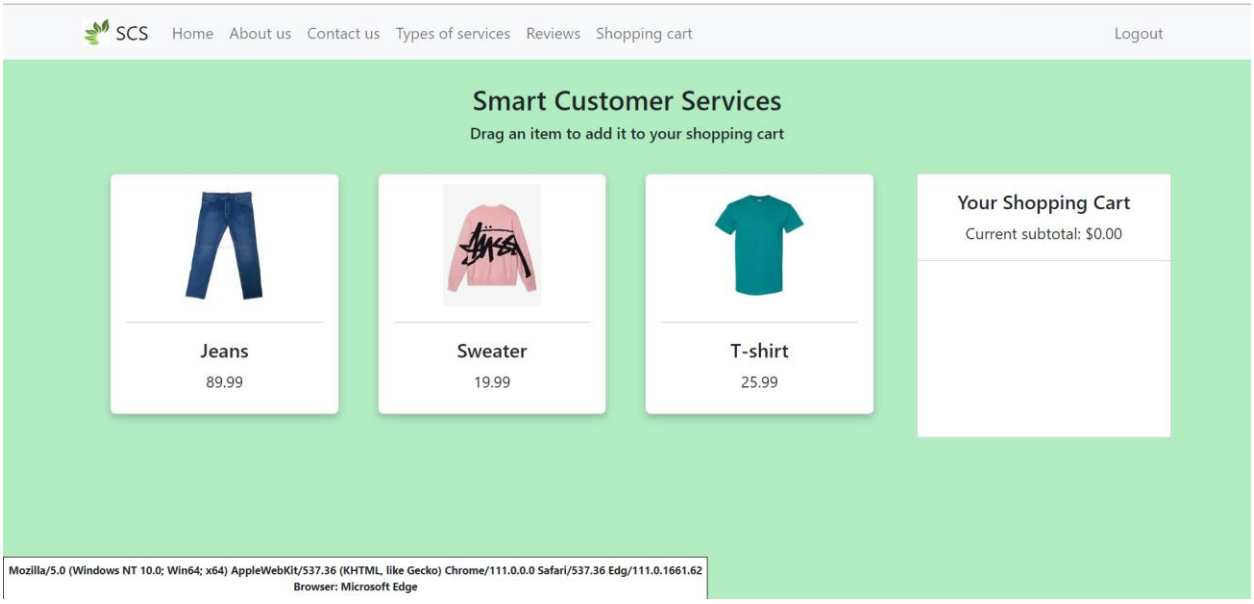
In the view page, our web application also shows all of the reviews made by every user. Moreover, we also evaluate all of the reviews and calculate the average for each item and show the user how much the user reviews that item.



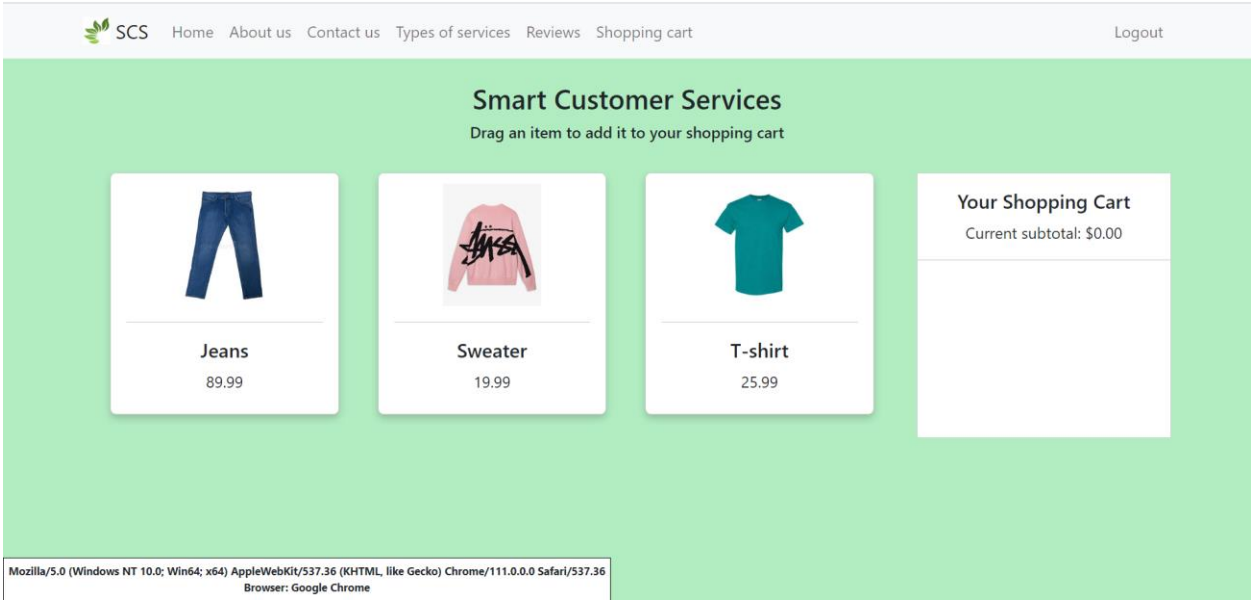
Browser Support

The web application supports and offers browser detection for the following three browsers: Microsoft Edge, Firefox, and Google Chrome. Browser detection information is displayed in a white box on the bottom left of the screen in our application.

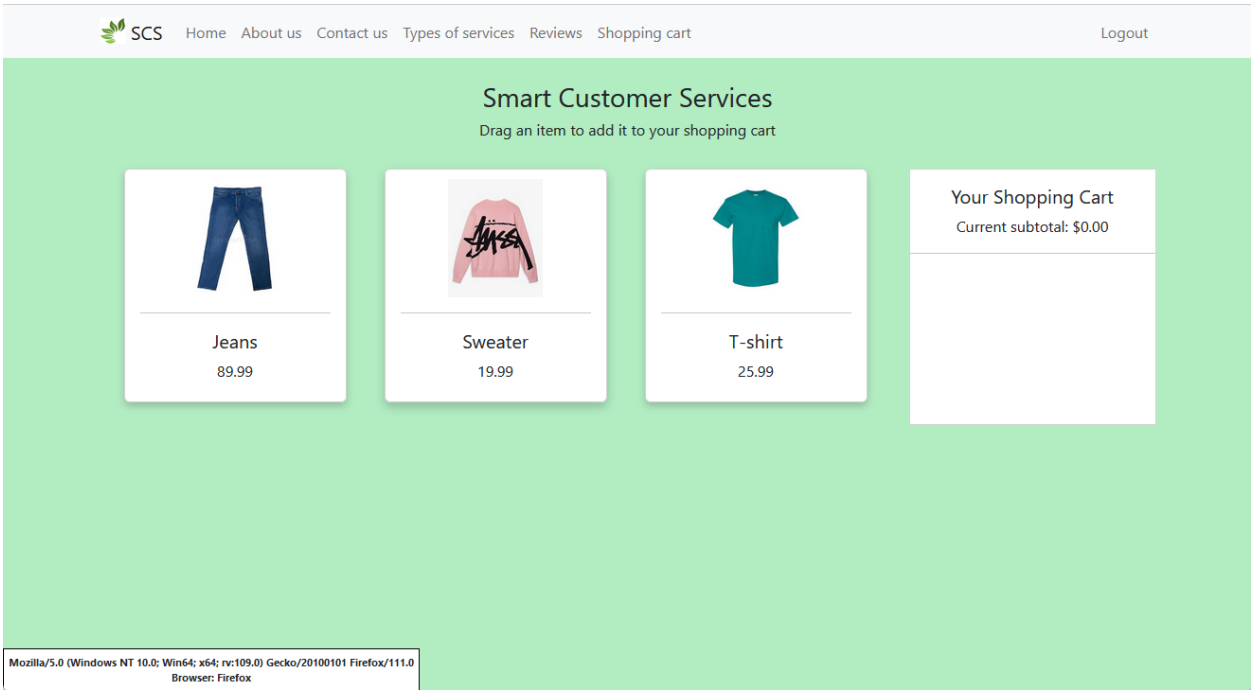
Microsoft Edge



Google Chrome

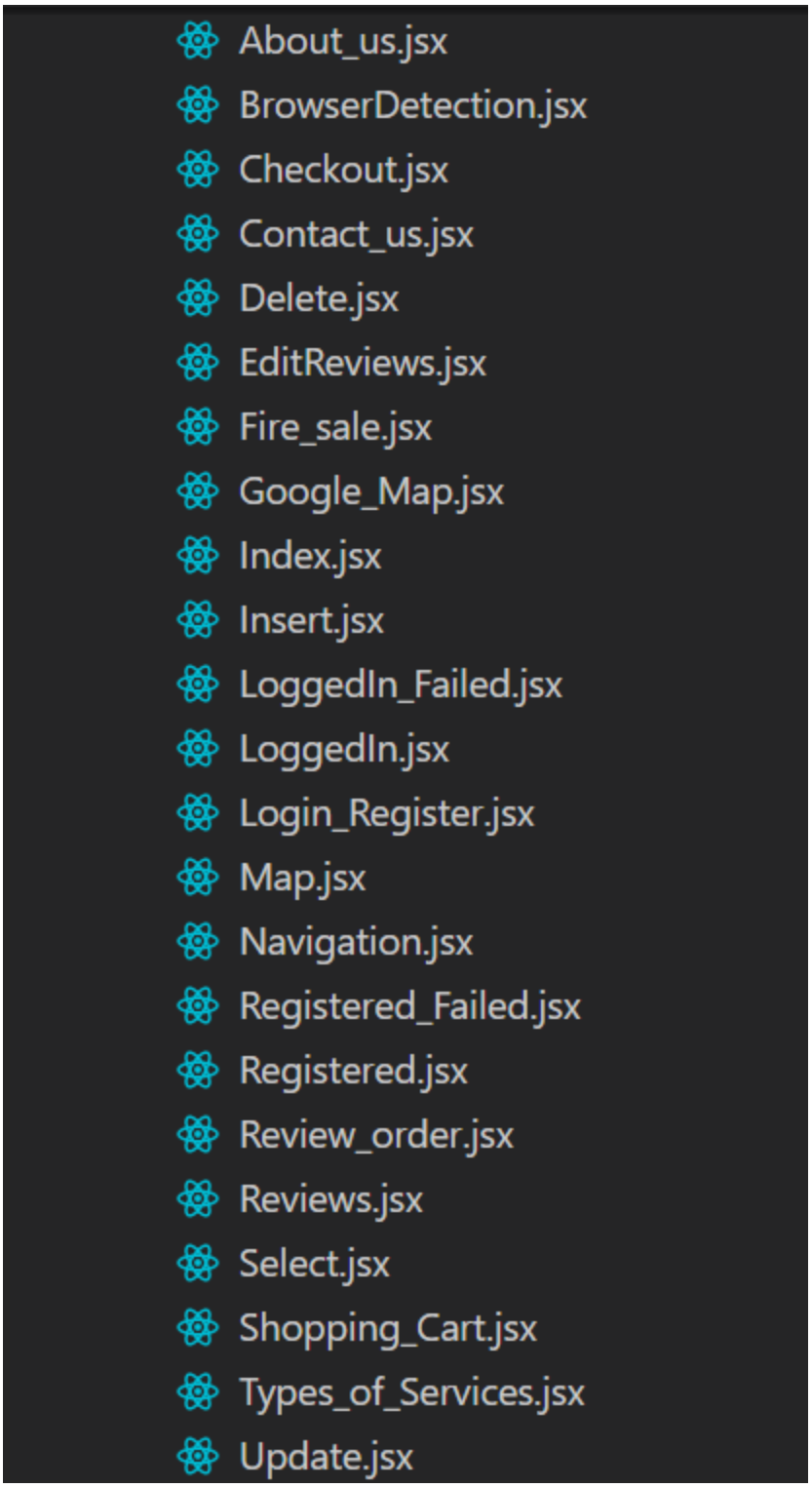


Firefox



SPA Design

Below is a list of the different web pages that are in our application (found in the react-app/src/components folder)



Our web application is a Single-Page Application (SPA) and was implemented using the React framework. We used BrowserRouter, Routes, and Route to define what component should be rendered onto the screen according to the current url path.

```
return (
  <div className="App">
    <BrowserRouter>
      <Routes>
        <Route element={<WithNav />}>
          <Route path="/home" element={<Index />}></Route>
          <Route path="/about_us" element={<About_us />}></Route>
          <Route path="/contact_us" element={<Contact_us />}></Route>
          <Route path="/types_of_services" element={<Types_of_Services />}></Route>
          <Route path="/edit_reviews" element={<EditReviews />}></Route>
          <Route path="/reviews" element={<Reviews />}></Route>
          <Route path="/shopping_cart" element={<Shopping_Cart />}></Route>
          <Route path="/map" element={<Map />}></Route>
          <Route path="/fire_sale" element={<Fire_Sale />}></Route>
          <Route path="/select" element={<Select />}></Route>
          <Route path="/insert" element={<Insert />}></Route>
          <Route path="/update" element={<Update />}></Route>
          <Route path="/delete" element={<Delete />}></Route>
          <Route path="/checkout" element={<Checkout />}></Route>
          <Route path="/review_order" element={<Review_order />}></Route>
        </Route>
        <Route element={<WithoutNav />}>
          <Route path="/" element={<Login_Register />}></Route>
          <Route path="/loggedIn" element={<LoggedIn />}></Route>
          <Route path="/login_fail" element={<LoggedIn_Failed />}></Route>
          <Route path="/registered" element={<Registered />}></Route>
          <Route path="/registered_fail" element={<Registered_Failed />}></Route>
        </Route>
      </Routes>
    </BrowserRouter>
    <BrowserDetection></BrowserDetection>
  </div>
);
```

The Route element <WithNav> contains the list of routes that should include the navigation bar at the top of the screen and the Route element <WithoutNav> contains the list of routes that do not display the navigation bar, such as the login screen. The <BrowserDetection> element is contained outside the BrowserRouter as it should always be rendered onto the page regardless of which component is currently being displayed.

Security Features

Our web application implements the password salting and hashing strategy to ensure security. If a user creates an account, the user's password will be concatenated with a random salt (using the

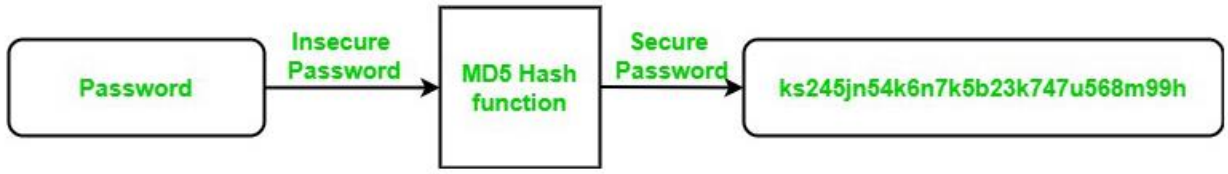
generateSalt() function), and then it will be hashed. The database only stores the salt and hashed password so that even the admin cannot know any user's original password.

```php
function generateSalt() {
    $length = random_bytes('8');
    return bin2hex($length);
}

function hashPassword($password) {
    return md5($password);
}
```

user_id	full_name	telephone	email	home_address	city_code	login_id	salt	user_password	balance
1	Michael Widianto	4169996666	michael.r.widianto@torontomu.ca	10 Yonge St	416	mrw19	85826b15dd2519f7	c434fb64677bcff7f7bc5cc0877c5517	0.00
2	John Doe	4163334444	johndoe@gmail.com	10 Adeline St	416	jdoe	f6be3922f21fb86b	5575eea42d4a4c617f947c0cd9c92dd0	500.00
3	Alex Stuart	7063334444	astuart@gmail.com	22 Yonge St	706	astuart	3bd49ff10b083902	50f7cd0bb98ed3860a6cf92c109e1ae1	200.00
4	Bill Clint	5121112222	bclint@gmail.com	45 Temprest St	512	blint	be920f453c0a5076	0cd39bf062838466ecf286b0d379537d	100.00

When a user signs in, we will take the user's salt from the database, append it with the password from the user, and hash it. Then, we will compare the hashed password in the database with the hashed password from the user.

Salting password refers to adding random data to the user's password before hashing it. Adding salt helps the hashing process to produce a unique output. On the other hand, hashing refers to the process of converting data of any size to a bit string of fixed size using a mathematical algorithm. Hashing is a one way mechanism, it is easy to compute the hash, but very difficult or impossible to regenerate the original input using only the hash value. There are some commonly used hashing algorithms such as Message Digest Algorithm (MD5) and Secure Hash Algorithm (SHA), but in our web application we utilize the MD5 algorithm.



MD5 Algorithm

The MD5 hashing algorithm starts with processing data in 512-bit strings, broken down to 16 words composed of 32 bits each, and outputs 128-bit message-digest value. The first stage of this algorithm is initializing the message-digest values using hexadecimal numerical values. Each stage uses four message-digest passes to manipulate values in the current data block and values

processed from the previous block. The final value computed from the last block becomes the MD5 digest for that block.

Through both salt and hashing passwords, it protects us from different attacks such as hash table attacks, as well as slowing down dictionary and brute-force offline attacks. Moreover, it also allows us to ensure confidentiality and data integrity.