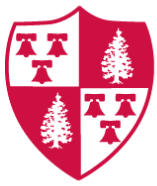


MONTCLAIR STATE UNIVERSITY

Department of Computer Science

Software Engineering II Project Report

Arthur Levitsky
Nachi Patil
Christian Kohlmayer
Daisy Gonzalez
Hope Diamantopoulos
Joseph Jeetan
Lauren Economy

CSIT 415 - Software Engineering II

Instructor: Dr. Kazi Zakia Sultana

14, May, 2021

Table of Contents:

Functional Requirements

Login Component

Admin Login

User Login

User Signup

Order Component

View Products

User Cart

Place Order and Payment API

Admin View and Process Order

User View Order

Cancel Order

Managerial Component

Admin Product Entry, Update, Delete

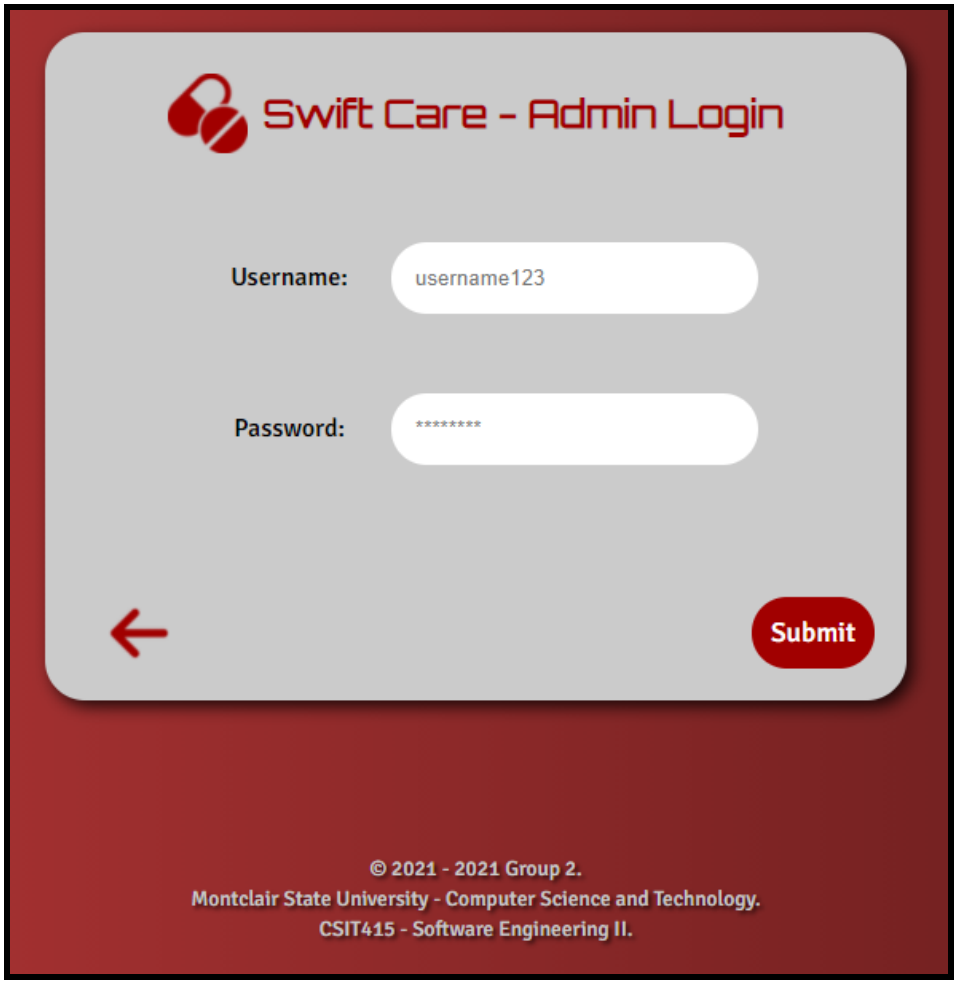
Admin View User Details

Payment Component

Payment API

Admin Login (Front-End):

The admin login page features a red gradient background and a container that holds a form inside. This container was provided with a border radius so that the corners would be round to provide aesthetic flow with the entire page. The container itself is gray and has a shadow casting behind it so that it stands out from the background. The contents inside the container is a form with labels stating *Username* and *Password*. There are input boxes to the right of it that have also been heavily modified to stand out from the container so that the user would know where to type and also hold the aesthetic flow of the entire page. At the bottom of the container, there is an arrow pointing to the left. This icon is used to describe to the user that they can go back a page. To the bottom right, we have a button that states *Submit* and has a background color and round corners to simulate an actual button that the user needs to press. At the bottom of the page, we have a footer that provides a copyright for the website page, details about the university, and the course provided in white text to easily stand out from the background. This was created with HTML and CSS.



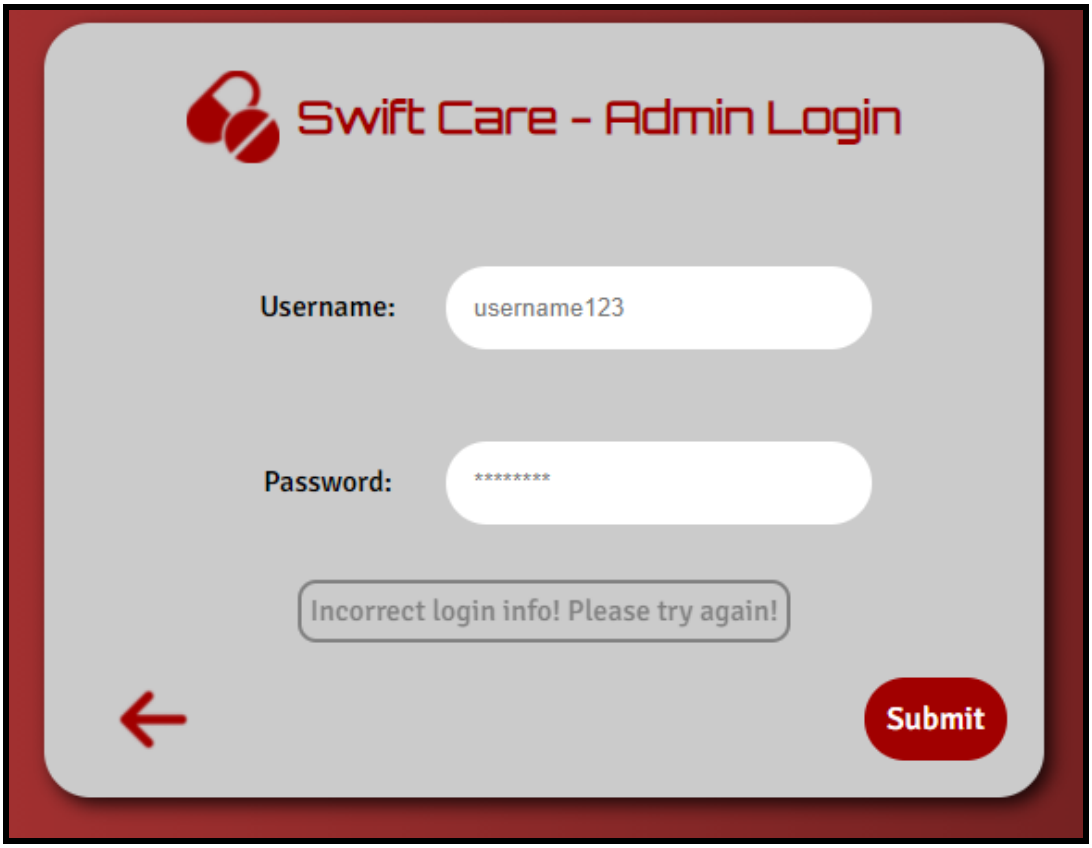
Admin interface for the Admin Login Page

Admin Login (Back-End):

The Login process was implemented using sessions in php in the backend. In order to ensure that only an admin could access the admin panel or any of the pages restricted only to admin access if statement checks were implemented. For example, an Admin account is not able to login to a user account because they are in different tables and there are checks to ensure that two users cannot be logged in at once (Especially not two different types of users). The admin details entered into the Admin Login page, are checked against the Admin user information in the database (The password information in the database is hashed and salted for security measures). If there is a corresponding Admin account, then they are taken to the Admin Panel. Otherwise, they are rerouted back to the same page and a message saying "Incorrect login info! Please try again!" Echo notice was used to display error messages.

id	username	password
20	super23	$2y$10$4e5B6hMpQJCmwJTE/Xy.4.gcsRTb16lZvg1/adyEaqF...

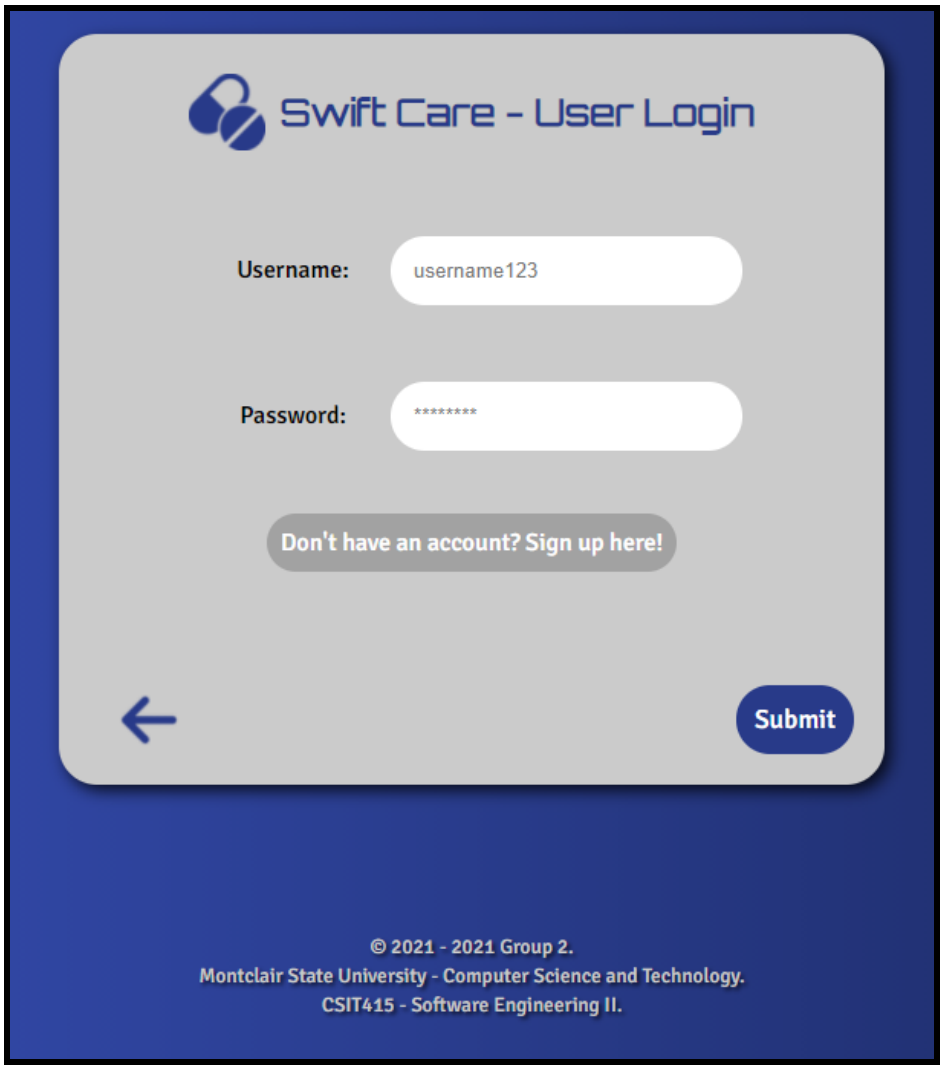
Phpmyadmin displaying 'admin' table with encrypted admin password.



When an admin attempts to log and provides the wrong information, he is presented with a notice.

User Login (Front-End):

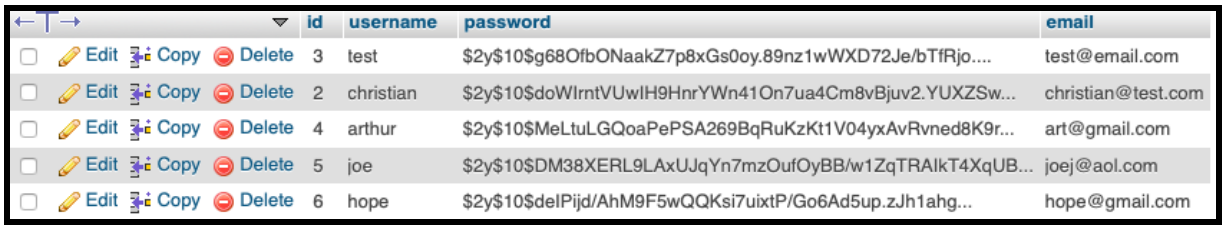
Similar to the admin login page, the user login page also features a gradient background that starts with a lighter blue from the left side of the screen to a darker blue to the right side of the screen. The title in this form holds similar characteristics to the admin login page, but instead of *Swift Care - Admin Login*, it has been renamed to *Swift Care - User Login* to signify that this is the place where customers have to login. The container is the same color and has the same rounded corners as the admin login page. The difference is that this container has a larger height (longer) to fit more elements properly. The form labels and input fields are exactly the same. The *Submit* button is similar but just holds a blue background color instead. The back arrow is blue instead of red. However, the major difference is that we have a rounded gray link that is below the input fields which asks whether the user needs to create an account or not. When clicking on that link, the user will be redirected to the sign up page. This was created with HTML and CSS.



User interface for the User Login Page

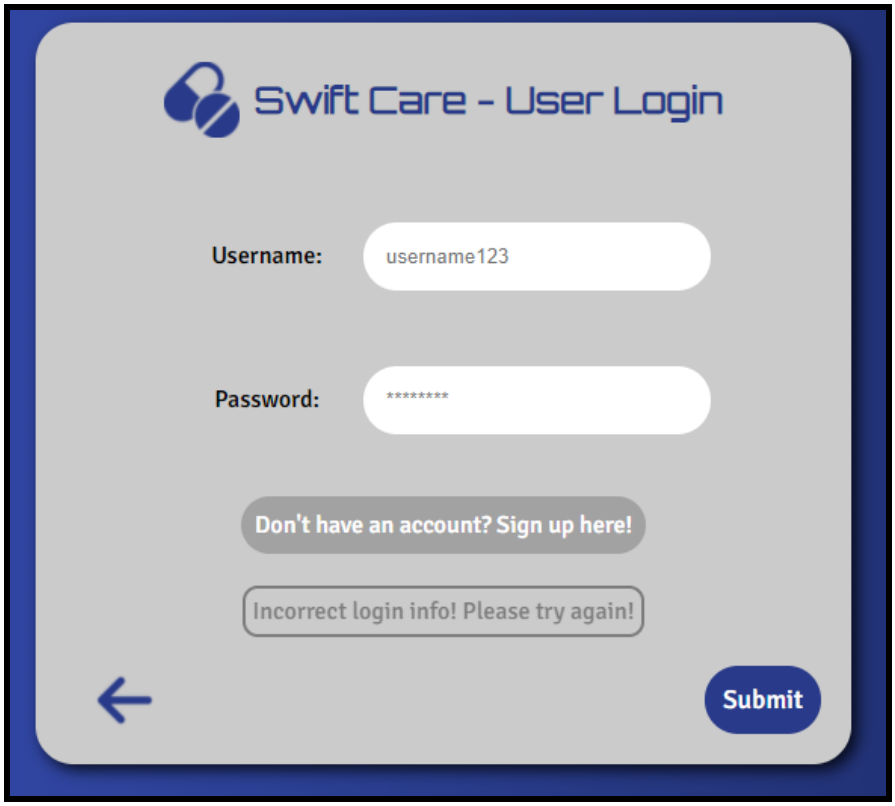
User Login (Back-End):

Just like for Admin login, the Login process for User was implemented using sessions in php in the backend. In order to ensure that only the specified user could access their user specific pages and account, if statement checks were implemented. An Admin account is not able to login to a user account because they are in different tables and there are checks to ensure that two users cannot be logged in at once (Especially not two different types of users). The user details entered into the User Login page, are checked against the Users table which contains user information in the database (The password information in the database is hashed and salted for security measures). If there is a corresponding User account, then they are taken to the User Panel. Otherwise, they are rerouted back to the same page and a message saying "Incorrect login info! Please try again!" is displayed. Echo notice was used to display error messages.



	id	username	password	email
Edit Copy Delete	3	test	$2y$10$g68OfbONaakZ7p8xGs0oy.89nz1wWXD72Je/bTfRjo....	test@email.com
Edit Copy Delete	2	christian	$2y$10$doWIrntVUwIH9HnrYWn41On7ua4Cm8vBjuv2.YUXZSw...	christian@test.com
Edit Copy Delete	4	arthur	$2y$10$MeLtuLGQoaPePSA269BqRuKzKt1V04yxAvRvned8K9r...	art@gmail.com
Edit Copy Delete	5	joe	$2y$10$DM38XERL9LAxUJqYn7mzOufOyBB/w1ZqTRAIkT4XqUB...	joej@aol.com
Edit Copy Delete	6	hope	$2y$10$deIPijd/AhM9F5wQQKsi7uixtP/Go6Ad5up.zJh1ahg...	hope@gmail.com

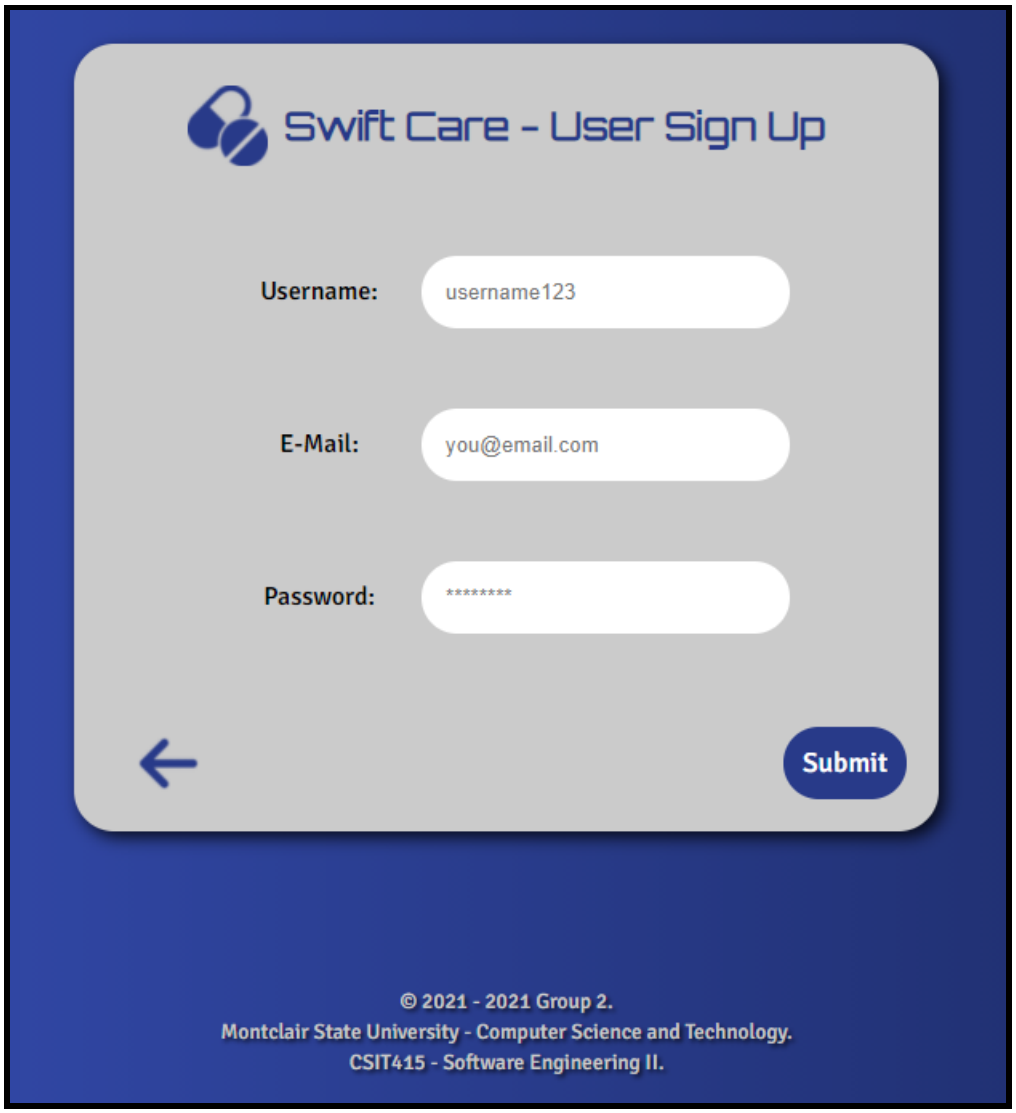
Phpmyadmin displaying 'user' table and encrypted passwords for the accounts.



When a user inputs wrong login information, he is presented with a notice.

User Signup (Front-End):

Holding similarity to the user login page, the user sign up page holds all the same attributes other than the title which was changed to *Swift Care - User Sign Up*. This was done so that the user would understand that this is the form where they would be required to sign up. Unlike the user login and admin login pages that only have *Username* and *Password*, the user signup page holds an additional label and input field where the user is asked to provide an email address. It's common among many websites to ask for users to provide an email address so that emails can be sent in terms of order details, deals, and prevention of duplicated accounts. This was created with HTML and CSS. The placeholders show the user an example of what they can write in the input field to avoid confusion.



User interface for the User Signup Page

User Signup (Back-End):

User Signup was also implemented using sessions in php in the backend. If statement checks were implemented to ensure that an admin would not be creating a user account, a user with the same address does not already exist and that certain limitations on creation of an account are not ignored. If there was an issue with the user signing up they are rerouted back to the same page and a message saying either "Username or email already taken" or "An error occurred while signing up" is displayed depending on the issue. The error messages are purposely not too specific for security reasons. Echo notice was used to display error messages.

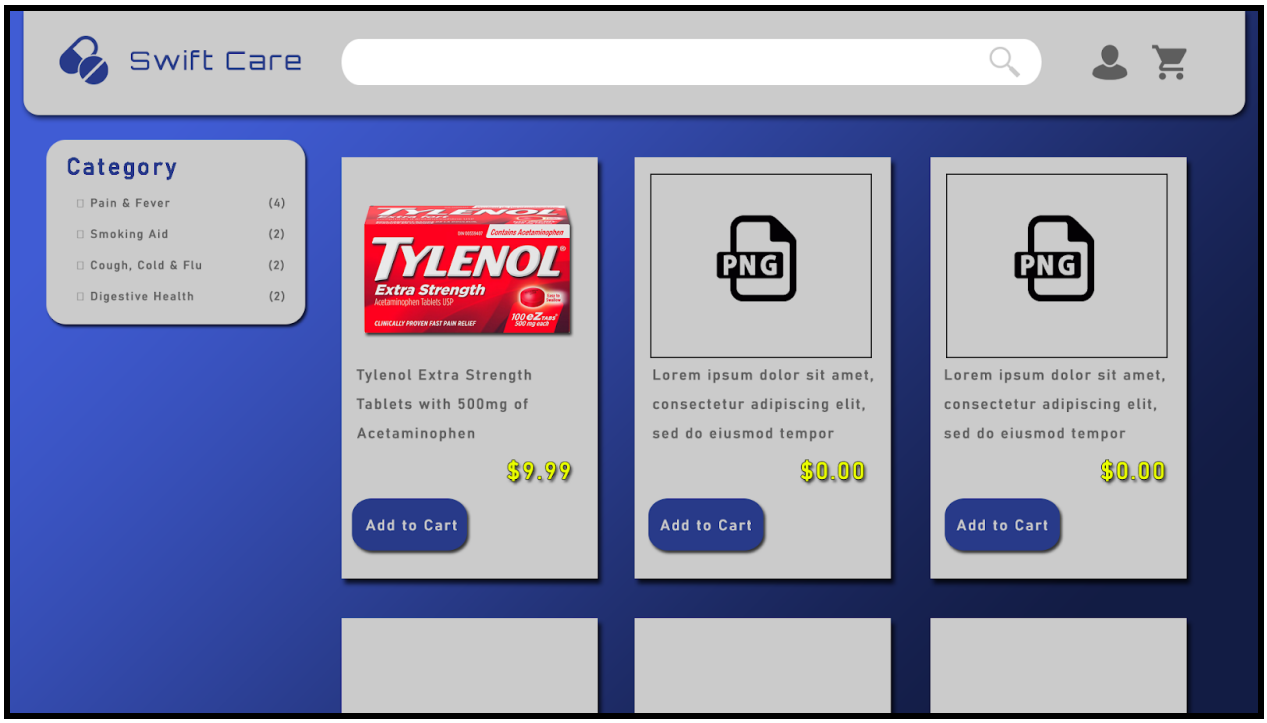
	id	username	password	email
Edit Copy Delete	3	test	$2y$10$g68OfbONaakZ7p8xGs0oy.89nz1wWXD72Je/bTfRjo....	test@email.com
Edit Copy Delete	2	christian	$2y$10$doWIrntVUwIH9HnrYWn41On7ua4Cm8vBjuv2.YUXZSw...	christian@test.com
Edit Copy Delete	4	arthur	$2y$10$MeLtuLGQoaPePSA269BqRuKzKt1V04yxAvRvned8K9r...	art@gmail.com
Edit Copy Delete	5	joe	$2y$10$DM38XERL9LAxUJqYn7mzOufOyBB/w1ZqTRAIkT4XqUB...	joej@aol.com
Edit Copy Delete	6	hope	$2y$10$deIPijd/AhM9F5wQQKsi7uixtP/Go6Ad5up.zJh1ahg...	hope@gmail.com

Phpmyadmin displaying 'user' table which is compared to see if the email address already exists.

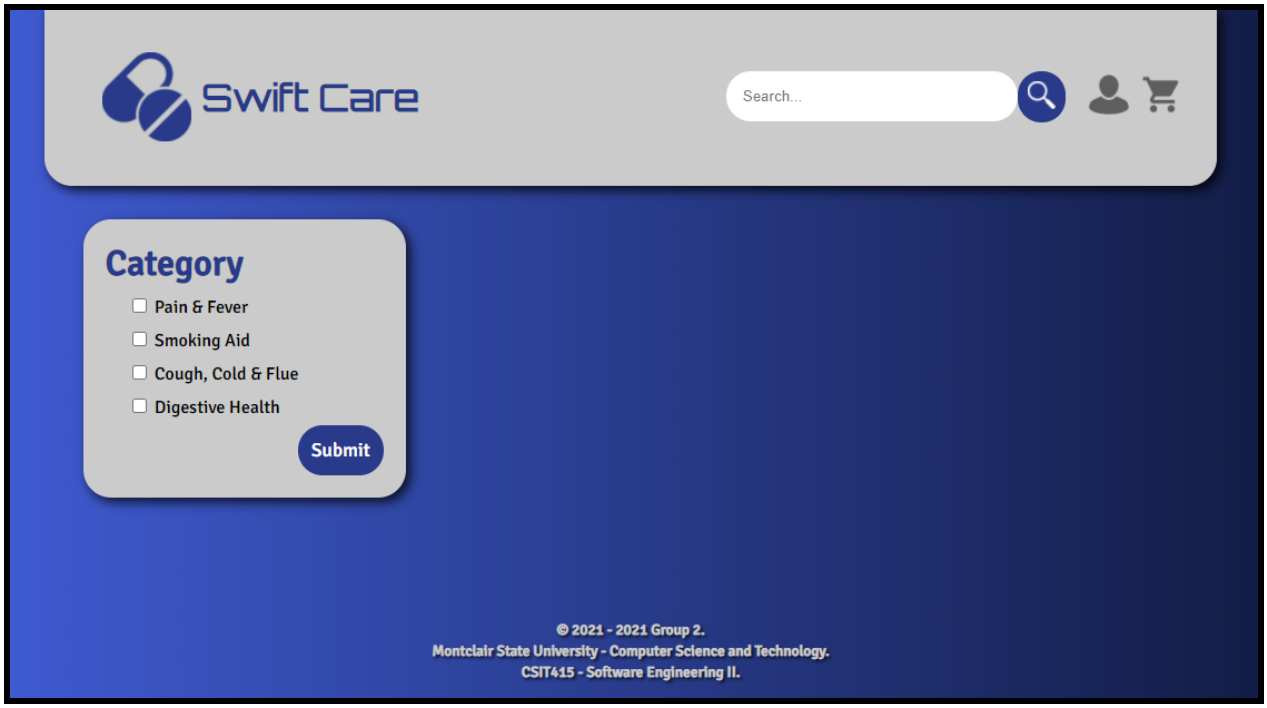
When a user inputs a username or email that is taken, he is presented with a notice.

View Products (Front-End):

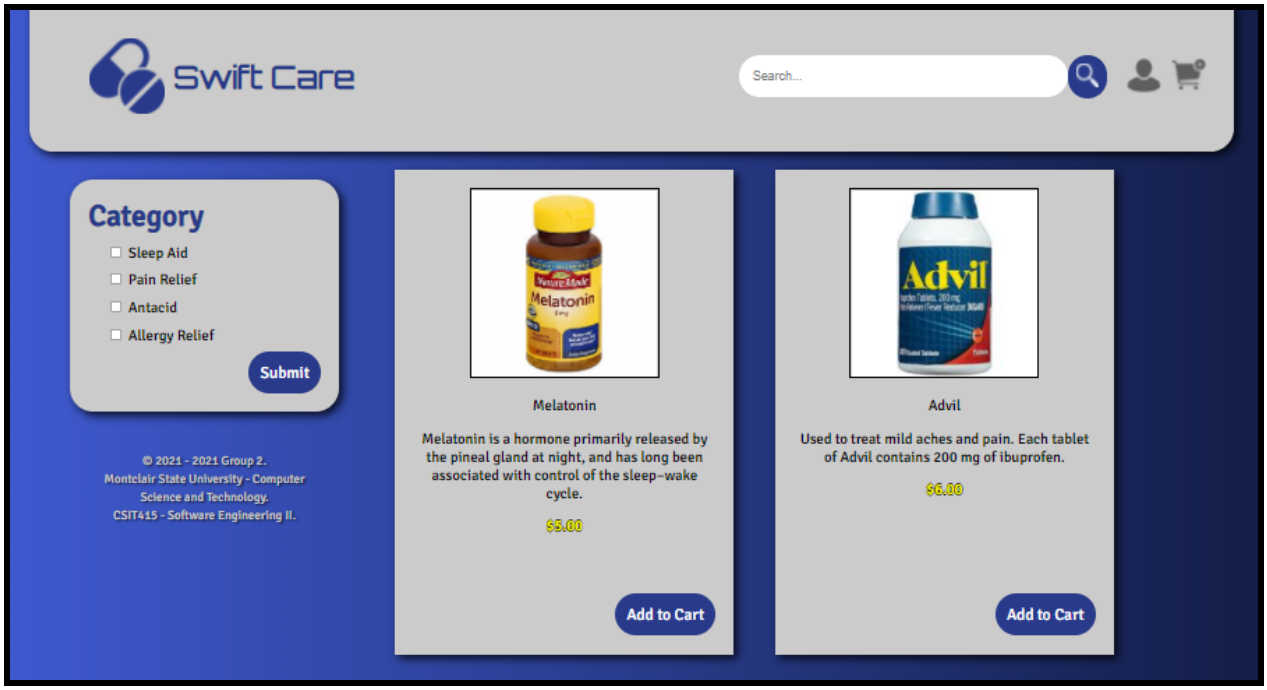
This is the first and main page that was designed for the medicine website project. The original implementation of the website was to hold a gray container that stretches at the top portion of the screen to show the user where they can type into the search bar, a user icon link that redirects the user to the user login page, cart icon that redirects the user to the cart page, as well as a category container that holds a form where it allows a user to select the check marks boxes that are labeled with a specific category of medicine. The *Submit* button has been reused from similar pages like the user login page and user signup page. Lastly, we have containers for each product displayed that displays three products per row while in fullscreen (1920 x 1080). Each container displays an image of the product, the title of the product, the description of the product, the price that is highlighted in yellow to make it stand out for the user when browsing, as well as an *Add to Cart* button.



Wireframe Design of the Swift Care Landing Page (Created with Adobe Photoshop)



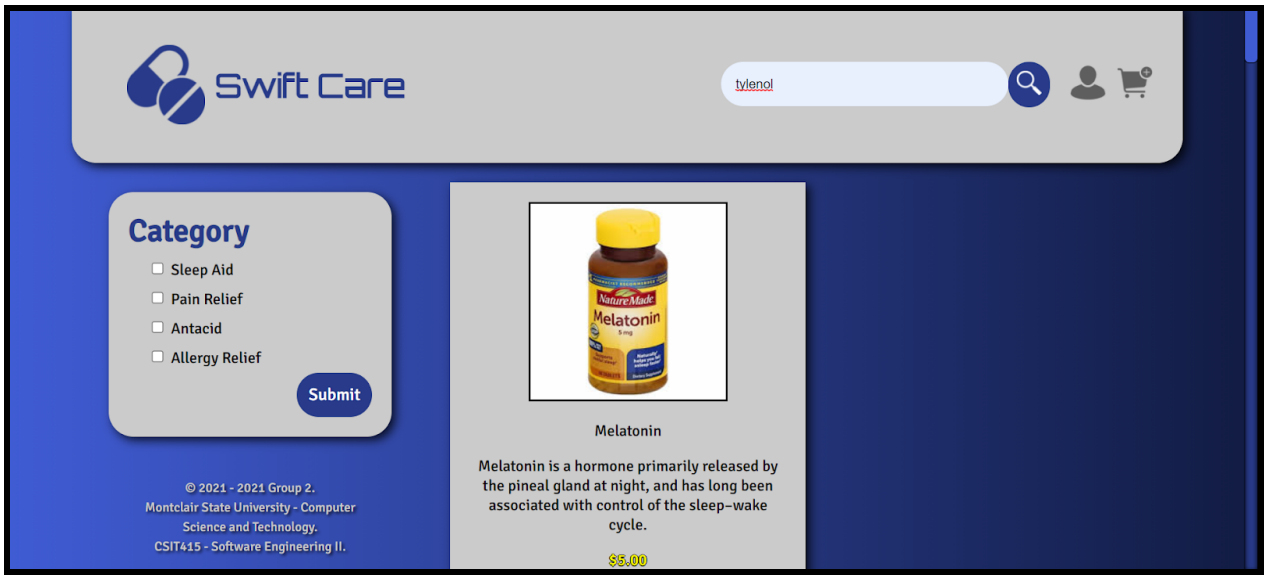
First Implementation of the Swift Care Landing Page (Version 1: HTML and CSS only)



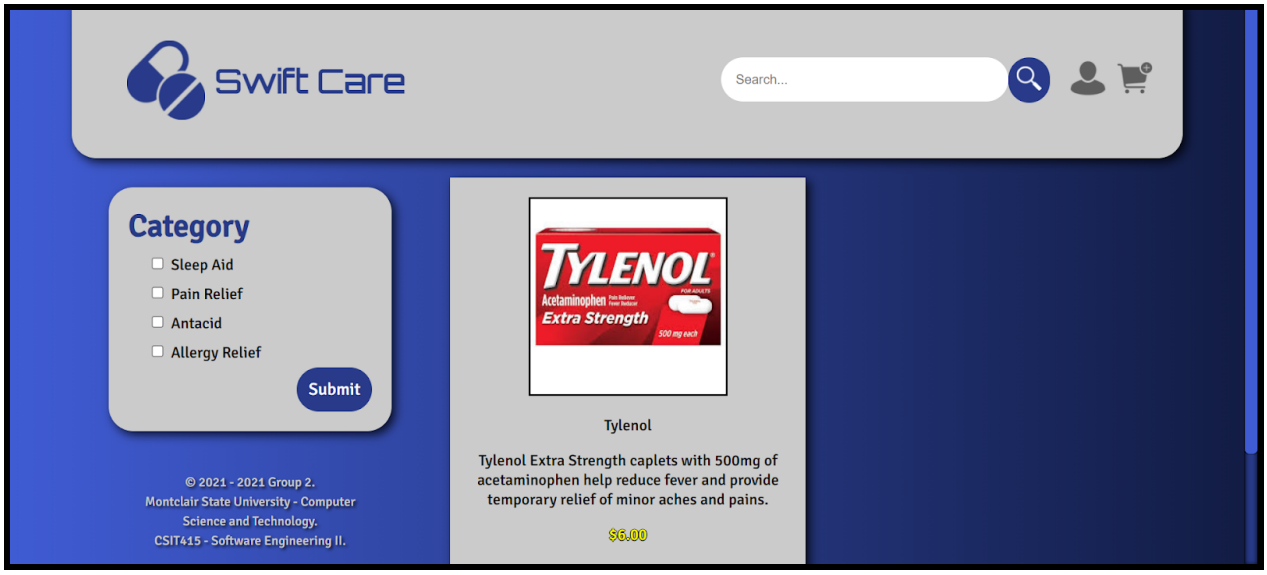
Final Implementation of the Swift Care Landing Page (Version 7: HTML, CSS, and PHP included)
(Note: Page displays two products because window is not fullscreen)

View Products (Back-End):

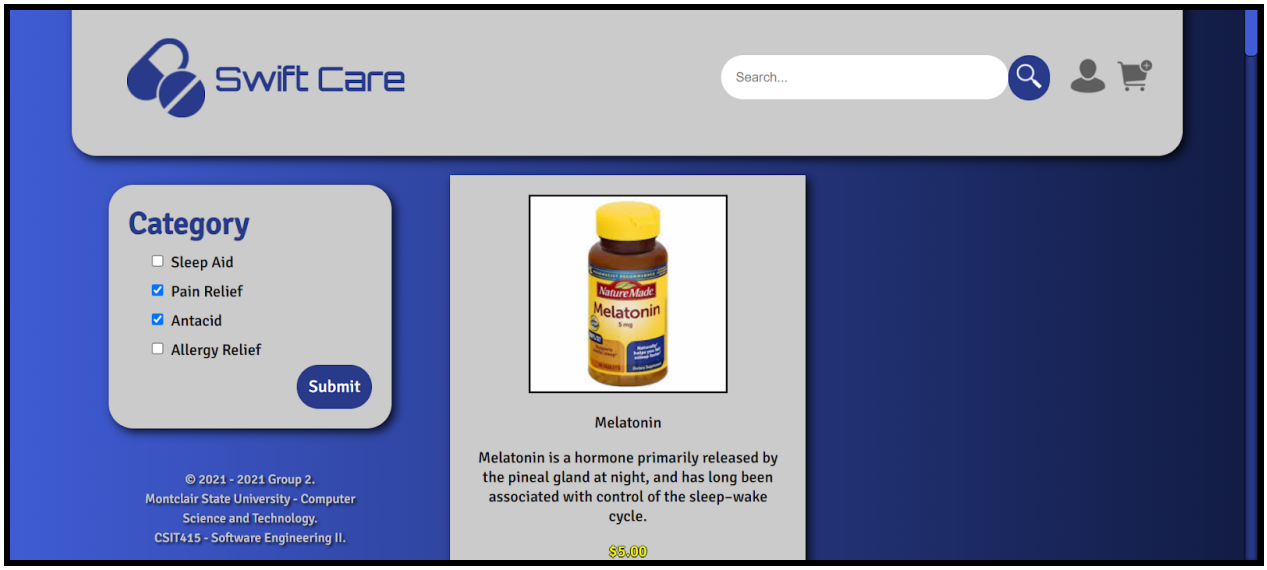
Items are displayed on the homepage. By default, all items available are displayed. The user is able to search items by name and by category. For searching items by name, a keyword is taken as input and all products whose name contain it are output. For category, the user checks one of more types of products available, namely Sleep Aid, Pain Relief, Antacid, and Allergy Relief. All products of the selected category/ies are displayed.The search function was implemented using a MySQL table to store product information. The product table holds a product's ID, name, category, description, price, and a link to an image. On the homepage, all but the product's ID and category are output. Database querying was done through PHP and SQL, and input was taken through HTML/CSS and PHP. Output was given also through HTML/CSS and PHP.



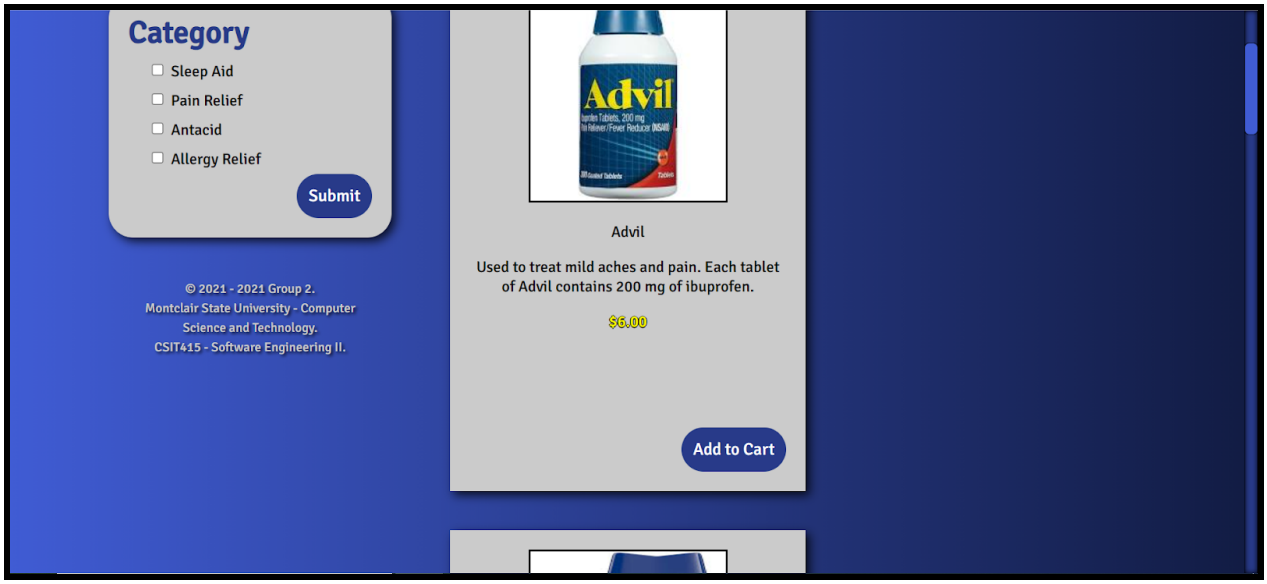
Query being typed into the search bar.



Results of the query "tylenol"



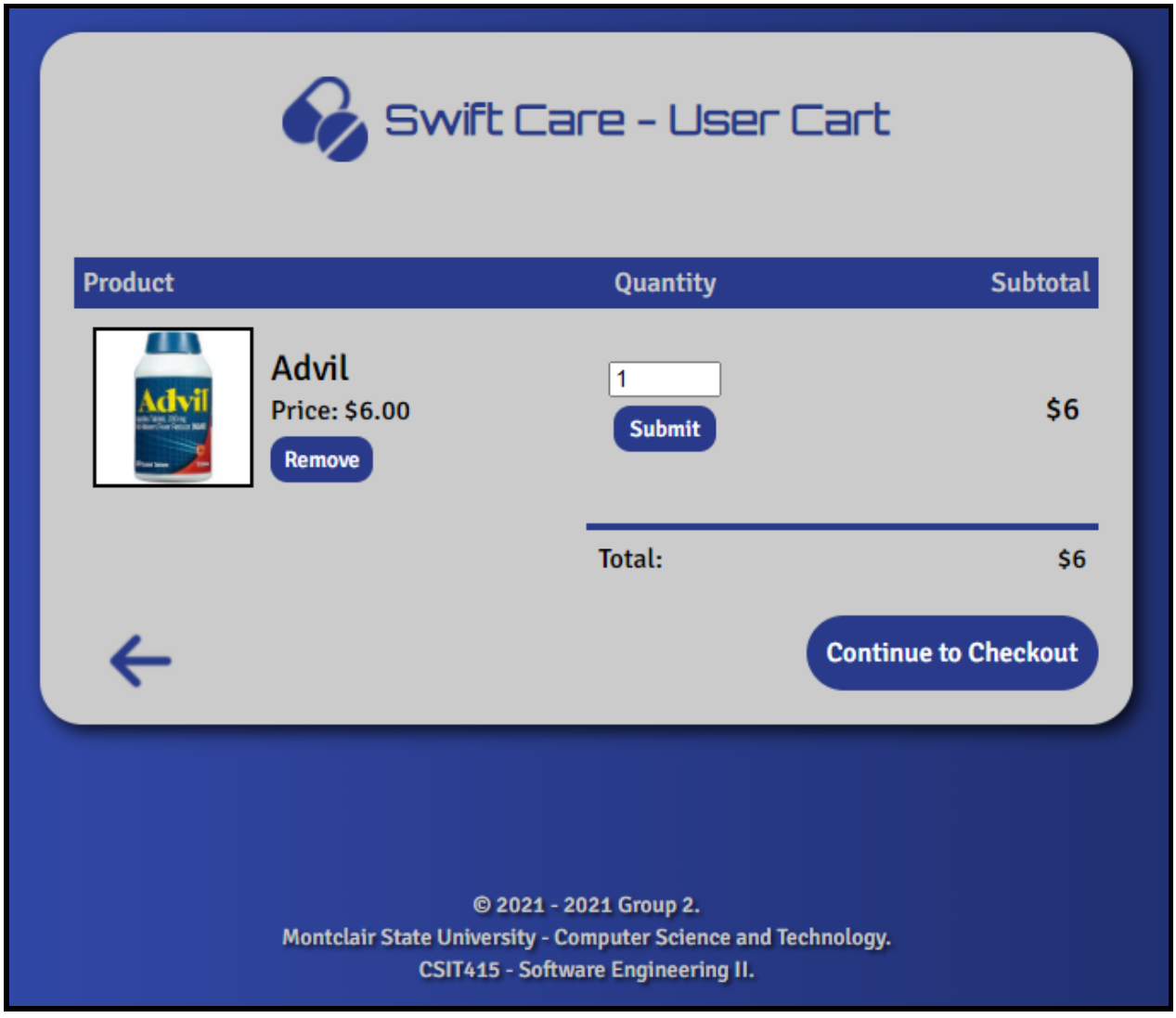
Categories being selected



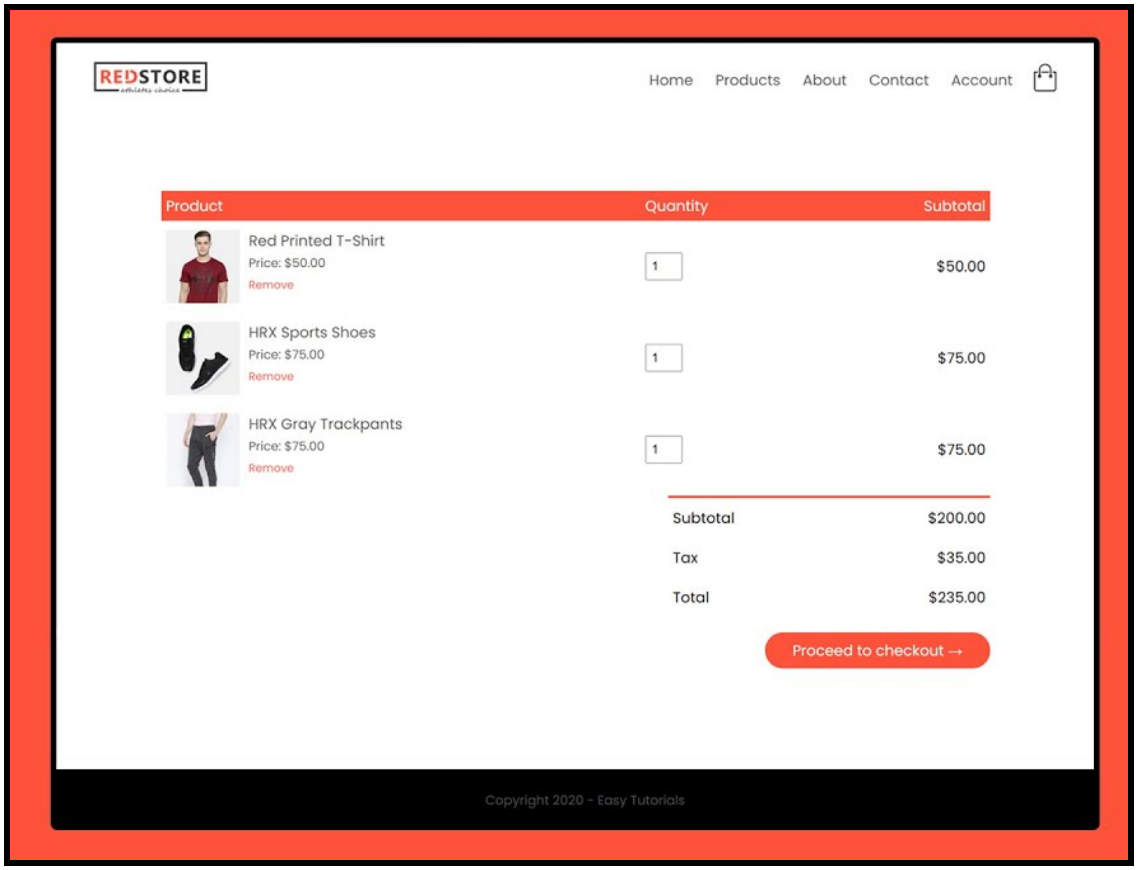
Results of searching by Pain Relief and Antacid categories (Note: most are below the fold)

User Cart (Front-End):

The front-end for the *User Cart* has been provided with specific care. The only similarities that this page has with other pages is that it retains the background color, the text color, the blue arrow to return back a page, and the "Continue to Checkout" button. But the actual design of the *User Cart* was done using two tables. The top table displays the headers "Product", "Quantity", and "Subtotal". The Product column displays a small picture of the product, the name of the product, the price, and a remove button. The Quantity column displays a numerical box that accepts a number and can change the quantity after hitting the submit button below it. The Subtotal box displays the cost of the specific product after taking quantity into account. The bottom table displays the "Total" row header and the total amount to the right side.



The User Cart Page, showing Advil product before checking out.



User Cart Inspiration. Provided by Youtube Channel - Easy Tutorials (https://www.youtube.com/watch?v=oXrlgOEiy6o)

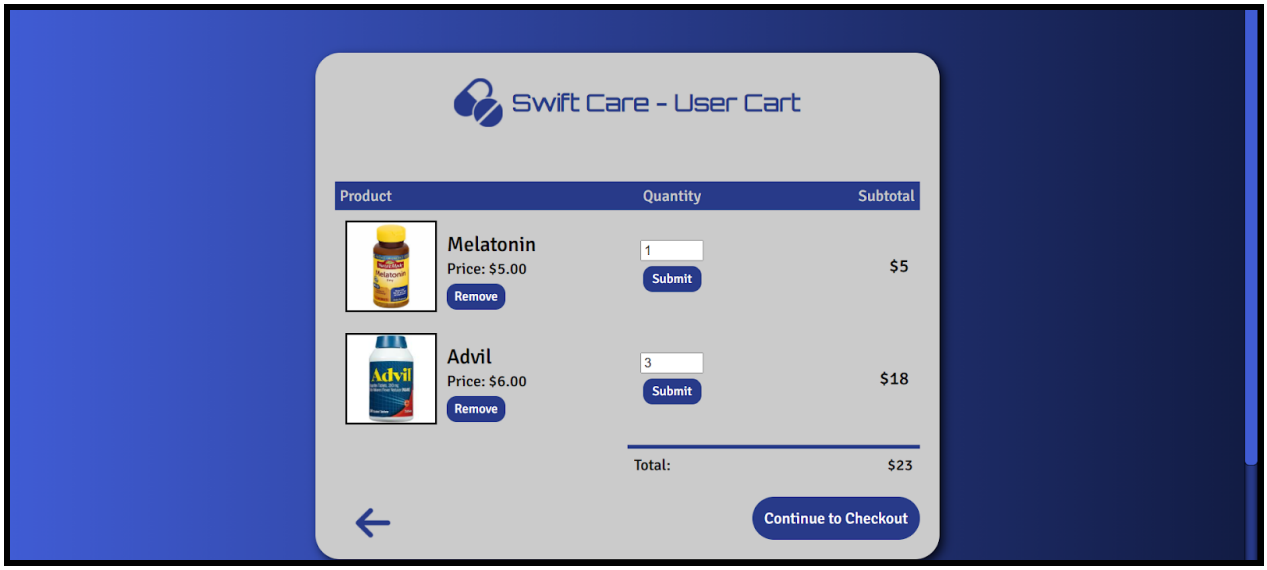
User Cart (Back-End):

The cart was implemented with a MySQL database table that holds a user ID, product ID, quantity, and a timestamp of when the entry was added. Whenever a user clicks the add to cart button on a product on the homepage, the product's ID and price are taken from the products table and stored in the cart table along with the user's ID. If there is already an entry in the cart with the same product ID and user ID, then the quantity of the entry is simply incremented by 1. If not, a new entry is made with the user's ID, product ID, and a quantity of 1. The table automatically assigns a timestamp to the entry of when it was added.

When the user clicks the cart icon from the homepage, they are taken to their cart page. If there are no items in their cart, it simply displays a notice informing them that the cart is empty. If there are items in the cart, the name, image, and price of each item is displayed, along with the amount of each item in the cart. The total cost of each item is also displayed. The user is also given the option to change the number of an item in the cart or remove an item from the cart. Finally, the user can proceed to checkout with the items they have in their cart.

Items being added and removed to and from the cart were implemented through PHP and SQL statements. Items in the cart being displayed with their picture, names, prices, and quantity was implemented through HTML/CSS and PHP. Input taken from the user for adding/removing/updating items in the cart was implemented through HTML/CSS and PHP as well.

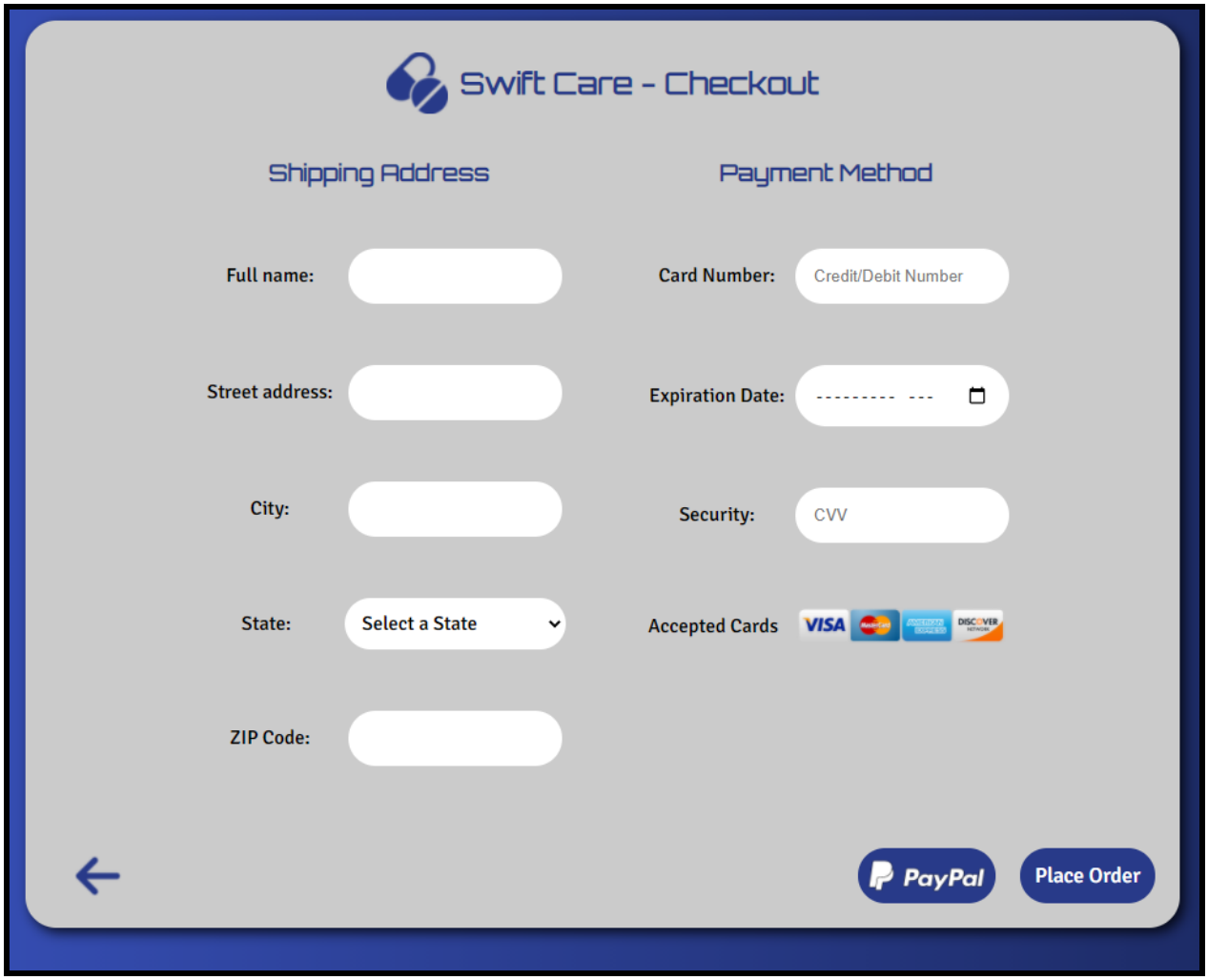
An empty cart.



A cart with items

Place Order and Payment API (Front-End):

The place order page is divided into two tables, shipping address and payment method. Using input fields, the shipping address will take in the user's full name, street address, city, state, zip code, and email address. The payment method has inputs for the user's card number, card expiration date, the card's security CVV number. It also displays the accepted cards such as Visa, Master Card, American Express, and discover. The input fields were implemented using different input types and attributes. For example, the card's expiration date has type "month" to display the calendar feature to allow the user to pick the date. The interface of this checkout page is very user friendly and aesthetically pleasing. The page is implemented using the blue CSS pages to indicate it is on the user side. The page also utilizes the main CSS page so that the styling is consistent with the other pages.

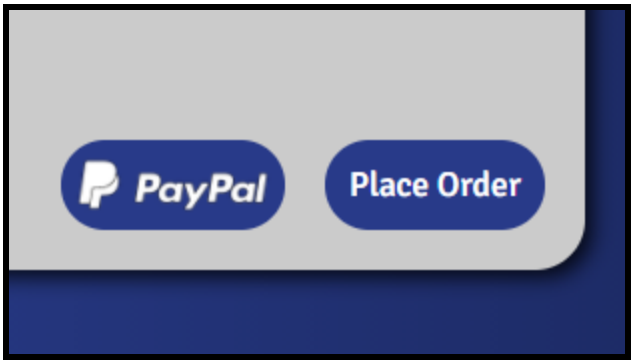


The Checkout Page, showcasing input fields required to allow the user to place their order.

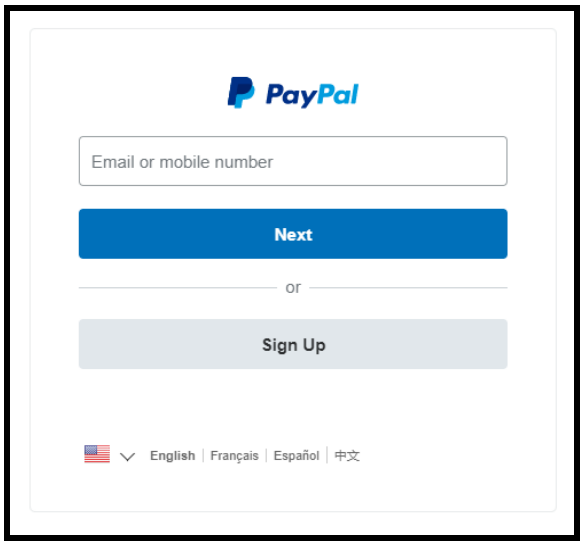
Place Order and Payment API (Back-End):

In terms of the *Payment API*, we have two buttons at the bottom of the page that allow the user to "Place Order" or access "PayPal". The PayPal button is essentially a representation of how a fake Payment API would supposedly work. When the user presses on the "PayPal" button, it redirects the user to the login page of the PayPal website. In theory, this would allow the user to log into their PayPal and send money to *Swift Care* using money from their PayPal account.

The "Place Order" button is a bit more functional in the sense that it actually interacts with our database. When the user clicks this button, an SQL statement is used to select all items in the cart table with a matching user_id. The total price, card information, and address provided by the user are then stored in variables and provided to an SQL statement that inserts them into a new row in the orders table. Following this, another two SQL statements are used to again select the items from the cart table with a matching user_id, and then insert each row into a new row of the orderitems table. This is performed separately and stored in two different tables (orders and orderitems) so that the details of each order can be displayed more easily later. After all of these statements have been executed, one last SQL statement is used to delete the rows of the cart table with a user_id that matches the session ID variable.



PayPal button and Place Order button.



PayPal Login screen after hitting "PayPal" button.

Admin View and Process Order (Front-End):

The front-end portion of the *Admin Process Order* page is created by reusing the gray container, background color and title text color from other admin pages. The things that are significantly different in this page compared to the other admin pages is that the title icon has been changed and the title name has also been changed to “Admin - Unprocessed Order” and we also have a second container that is renamed to “Admin - Processed Order”. Both containers hold tables of an OrderID, Total Price, User ID, Timestamp, Authorized By, and Order Details. Authorized By and Order Details feature buttons with a red background and rounded edges to simulate the look and feel of a button. When there are no unprocessed orders or processed orders, a gray message will be displayed instead of a table. Red arrows facing leftwards are meant to bring the admin back to the *Admin Panel* screen.

Admin View Order Page displays two containers which hold two tables for Unprocessed and Processed Orders.

For the front-end of the *Admin View Orders*, the page was also created from an existing admin page holding a gray container and a title with an icon. Similar to the *Admin Process Orders* page, this page holds a table but with three headers that state Product, Price, and Quantity. The title is also named "Admin - Order Details - #29", where #29 is dynamically displayed to signify to the admin that he is viewing the order components of Order #29. When the admin moves to a different page, a different number will display.

Admin View Orders Page that dynamically displays order # in the title and the order components in a table format.

Admin View and Process Order (Back-End):

Orders placed by users are stored in a MySQL database table with an order ID, total price, ID of the user who placed it, a timestamp for when it was placed, and the ID of the admin who authorized it. When first placed by the user, the admin ID field is left blank. In the admin section of the website, admins can view all orders that have been placed. The order ID, total price of the order, ID of the user who placed the order, timestamp of when the order was placed, and the ID of the admin who authorized the order are displayed. An option is also given to view details of an order, which redirects the admin to a page which displays the name, price, and quantity of each item in an order. This information is taken from another table in the MySQL database which stores the product ID, ID of order it is in, and quantity of each item ordered. The product ID from this table is used in the products table to get the name and price of the item.

Orders with the authorizing admin ID field left blank are considered unauthorized. Unauthorized orders are given priority and output on a table above the authorized orders. The admin has the option to authorize the order. Upon choosing to do so, the order entry will be updated to have the current admin's ID in the authorizing admin ID field.

Implementation of viewing and processing orders was done by storing order information in MySQL database tables. Accessing and changing the information of the table entries was done through PHP and SQL. Input and output of the table entries was done through HTML/CSS and PHP.

User View Order (Front-End):

The front-end portion of *User View Order* follows the same style format as the rest of the user end website. As seen the same blue background is shown along with the grey container that holds all functionality and text other than the footer on the bottom of the page. The blue user icon is shown on top along with the same style format title with the blue text. Specific to this page are the functional blue "View Details" buttons that bring the customer to the *User Cancel Order* page for the specific order they selected. Text providing the order information is black to ensure clear readings on the user's order information which include the "Order ID", "Total Price", and "Timestamp" (which is when the user finalized the order). The blue arrow is positioned in the bottom left corner so that the user may return to the previous page which is the *User Control Panel*.

User View Order (Back-End):

Logged in users are able to view all of their past orders by navigating to user_vieworder.php. The back-end for this file first checks to ensure that a user is logged in to an account stored in the users database table. Any user not currently logged in will automatically be redirected to the user_login.php.

To display a logged in user's order history, an SQL statement is used to select all orders from the orders database table that have a UID which matches the users session id variable. The orderID, totalPrice, and timeStamp are then simply echoed in table format.

For additional order details, users may then click the "View Details" button to navigate to user_order_details.php. This file is a bit more complex as it displays all of the information stored in the database for that particular order. This task is accomplished by a series of SQL statements. The first statement selects the rows from orders that have an orderID which matches the order selected on the previous page. This variable is accessible through a post on user_vieworder.php. A foreach statement is then used to display the address and cardInfo provided by the user when they placed the order. Following this, a far more complex SQL statement is utilized on line 70 to select all rows from orderitems that contain an orderID that matches an ID in the products table as well as an orderID that matches the posted order_id variable. Once this statement is successfully executed, another foreach statement is utilized to echo a table containing the name, price, and quantity of each individual item in the particular order.

```php
//checks if user is logged in; if not, redirects to user login page
if (!isset($_SESSION['logged_in']) && $_SESSION['logged_in'] == true) {
    header('Location: user_login.php');
}
```

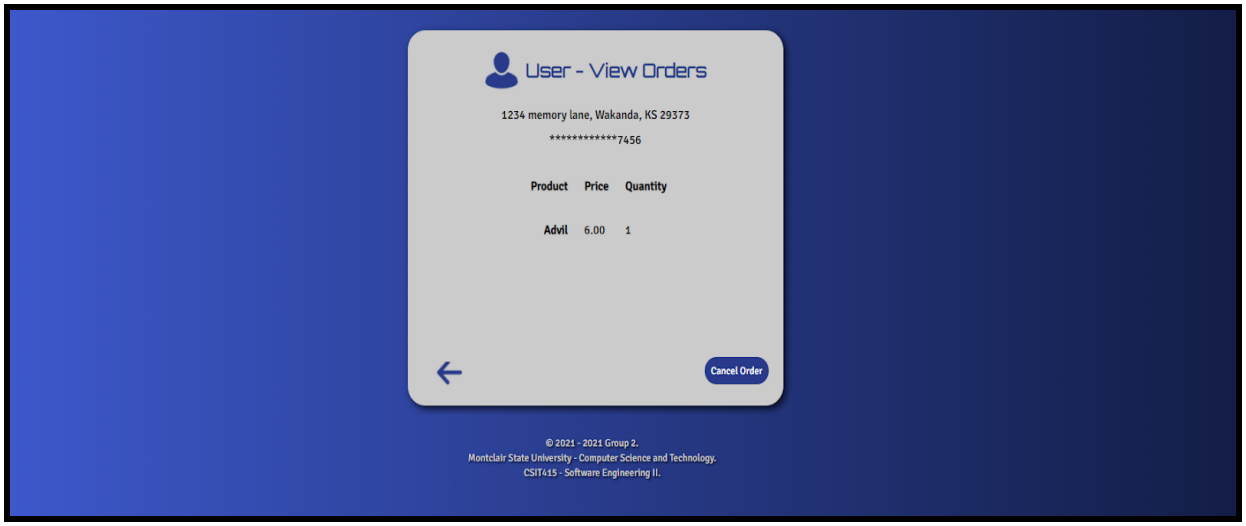
```php
try {
    // SQL statement to select all orders from orders table with a UID that matches the logged in users id
    $stmt = $db->prepare("SELECT * FROM orders WHERE UID=?");
    $stmt->execute([$_SESSION['id']]);
    $results = $stmt->fetchAll();
```

```php
try {
    $stmt = $db->prepare("SELECT * FROM orders WHERE orderID=?");
    $stmt->execute([$_POST['order_id']]);
    $results = $stmt->fetchAll();
```

```php
// SQL statement to display all orderitems rows with an orderID that matches the orderID in the orders table
try {
    $stmt = $db->prepare("SELECT * FROM orderitems OI, products P WHERE OI.IID=P.ID AND OI.orderID=?");
    $stmt->execute([$_POST['order_id']]);
    $results = $stmt->fetchAll();
```

Cancel Order (Front-End):

For the front-end of *Cancel Order* on the User end the same style as the rest of the user front-end website is used with the familiar blue background and grey container. Within the grey container holds all the functional buttons and informational text (other than the footer) for a simple, easy to use experience. Specifically talking about this page the blue "Cancel Order" button (blue represents user) will immediately cancel and delete the order from the *User View Order* page. In the center of the panel is the black text that shows the order's specific details such as the address that the order is shipping to, the last 4 of the credit card number, the name of the product, the price of the product, and the quantity. The familiar arrow pointing left at the bottom left of the page will bring the user back to the *User View Order* page.



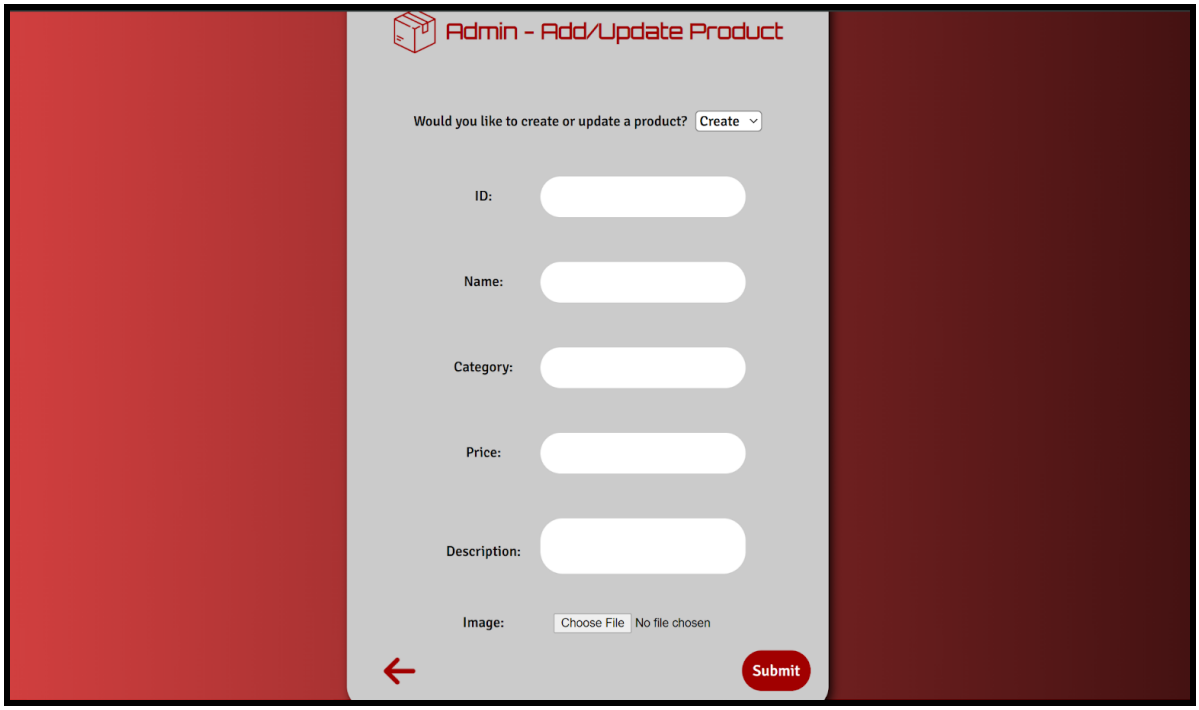
Cancel Order (Back-End):

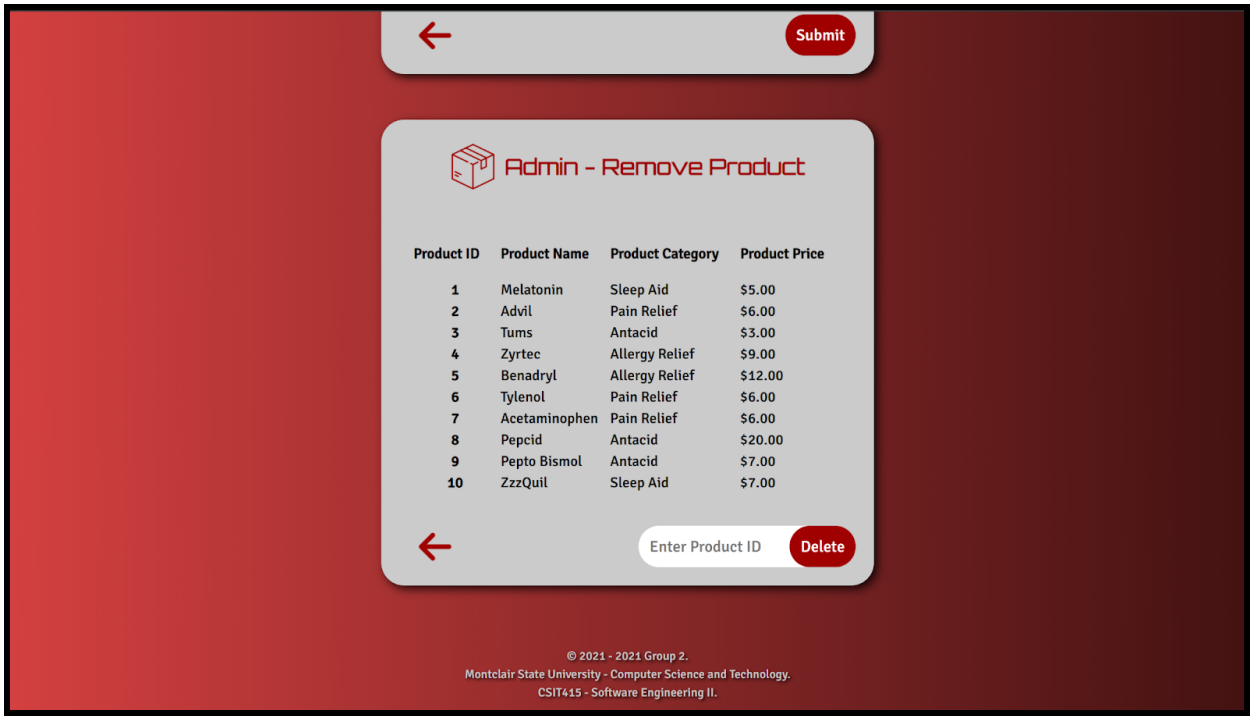
Users are given the capability to cancel their past orders before they're approved and shipped by an admin.This functionality was built into the user_order_details.php file under the details of the selected order. When the cancel order button is clicked on the page, it will post to two different SQL statements that delete all rows in the orders and orderitems database tables with the corresponding orderID.

```php
// SQL statements to delete order contents from orders and orderitems tables
if (isset($_POST['options'])) {
    $stmt = $db->prepare("DELETE FROM orders WHERE (orderID=?)");
    $stmt->execute([$_POST['options']]);
    $stmt = $db->prepare("DELETE FROM orderitems WHERE (orderID=?)");
    $stmt->execute([$_POST['options']]);
```

Admin Product Entry, Update, Delete (Front-End):

The front-end portion of the *Admin Product Entry, Update, Delete* page is created by reusing the gray container, background color and title text color from other admin pages. The things that changed in this page in comparison to the rest is that there are two different containers, one that allows the admin to add products and one which displays what products are currently there. There is also a delete button which also allows the admin to delete a product they have based on the ID they are given when adding the product. The first container allows input from the admin for the product's ID, Name, Category, Price, Description and Image. There is also an option which allows the admin to update a current product. The second container contains the product's ID, name, category and price. Admin Product Entry, Update, Delete feature buttons with a red background and rounded edges to simulate the look and feel of a button. Red arrows facing leftwards are meant to bring the admin back to the *Admin Panel* screen.




Submit
Admin - Remove Product
Product ID Product Name Product Category Product Price
1 Melatonin Sleep Aid $5.00
2 Advil Pain Relief $6.00
3 Tums Antacid $3.00
4 Zyrtec Allergy Relief $9.00
5 Benadryl Allergy Relief $12.00
6 Tylenol Pain Relief $6.00
7 Acetaminophen Pain Relief $6.00
8 Pepcid Antacid $20.00
9 Pepto Bismol Antacid $7.00
10 ZzzQuil Sleep Aid $7.00
Enter Product ID
Delete
© 2021 - 2021 Group 2.
Montclair State University - Computer Science and Technology.
CSIT415 - Software Engineering II.

Admin Product Entry, Update, Delete (Back-End):

The admin product entry page back-end is designed to provide logged in admins with a simple control page in which they can add new products, update existing ones, or delete ones that are no longer available. To access manage_product.php, the user must be logged in as an admin, or else they'll be redirected to the admin login page. This security measure was implemented using a php session variable and our admins database table.

Interacting with the back-end of this page is rather simple to avoid user error. Admins must simply select whether they would like to create or update a product, then fill in the required fields and upload an image. All inputs are then assigned to a variable and input into the products database table through various SQL statements. In addition to this, the provided image file is sent through multiple checks to ensure it is a real image and that it doesn't already exist on the server. Following this, the image is then formatted properly and sent to the directory containing all pre-existing image files.

The delete product functionality works very simplistically as well. As the front end of the page displays all the current products in the products table, all the admin must do is input the ID of the product they wish to delete. Once the delete button is clicked, an SQL statement is used to delete the row with the matching Product ID. An if statement is also utilized to delete the product's image from the server's directory.

```php
if ($_POST['options'] == 'create') {
    // SQL statement to add a new row to the products database
    $stmt = $db->prepare("INSERT INTO products (ID, name, category, description, price, image) VALUES (?, ?, ?, ?, ?, ?)");
    $stmt->execute([$ID, $name, $category, $description, $price, $target_file]);
```

```php
} elseif ($_POST['options'] == 'update') {
    //SQL statement to change attributes of product with matching ID
    $stmt = $db->prepare("UPDATE products SET name=?, category=?, description=?, price=?,  image=? WHERE ID=?");
    $stmt->execute([$name, $category, $description, $price, $target_file, $ID]);
```

```php
if ($_POST['options'] == 'delete') {
    // SQL DELETE products where user input credentials match
    $stmt = $db->prepare("SELECT * FROM products WHERE (id=?)");
    $stmt->execute([$_POST['ID']]);
    $results = $stmt->fetch(PDO::FETCH_ASSOC);
```

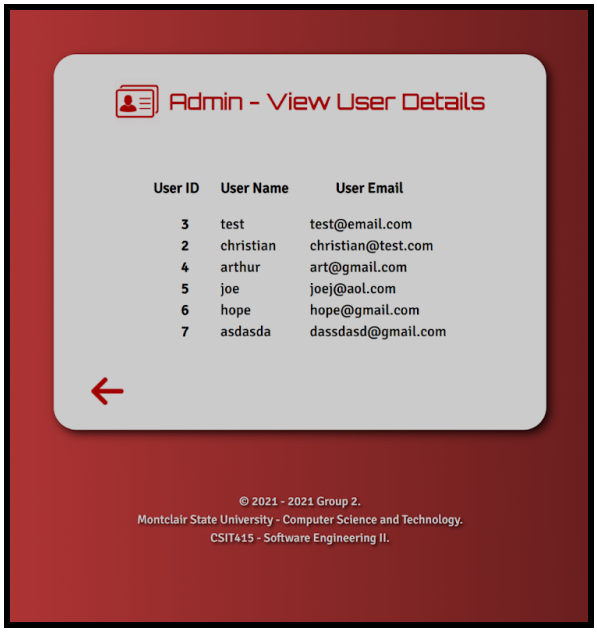
Admin View User Details (Front-End):

The front-end portion of the *Admin View User details* page is created by reusing the gray container, background color and title text color from other admin pages. The things that changed in this page in comparison to the rest is that there is now one container which displays the User ID, User Name, and User email. Admin View User Details features red arrows facing leftwards are meant to bring the admin back to the *Admin Panel* screen.

Admin View User Details (Back-End):

Interacting Since we are storing our user information in the “users” table we just simply connect to the database and fetch the information. This is done by selecting all the data from the users table and querying to make the information available to use. This way the information can be shown easily on the front end by echoing the results.

```php
<?php

//includes db connection
require_once 'database_connection.php';

//general search statement for if no search was performed
$stmt = 'SELECT * FROM `users`';

//preparese db query
$query = $db->prepare($stmt);

//runs query and gets results
$query->execute();
$results = $query->fetchAll();

?>
```



Interface Requirements

Application Type
Front-end Software
Back-end Software
Operating System
Supported Browsers
Editors and Compilers Used

- **Application Type:**
    - *Swift Care* is a **web-based application**. It can be accessed through supported web browsers and searched by users looking to purchase from a medicine website.
- **Front-end Software:**
    - The software that was largely used to develop the front-end were applications such as **Notepad**, **Sublime Text**, and **Visual Studio**.
    - The front-end was written solely in **HTML** and **CSS**.
- **Back-end Software:**
    - The software that was largely used to develop the back-end was applications such as **phpmyadmin** for database management and **MySQL** for the actual database.
    - The website was also created on a real public server which was setup with **LAMP software.**
    - Individual local testing of the database files and web-page files were done using **XAMPP software**.
    - The back-end was written solely in **PHP**.
- **Operating System:**
    - The website is operational on **Windows, Linux**, and **MacOS**.
- **Supported Browsers:**
    - The website is supported on browsers such as **Brave, Chrome,** and **Firefox.**
- **Editor and Compiler Used:**
    - **No compilers** were used in the production of the *Swift Care* medicine website.

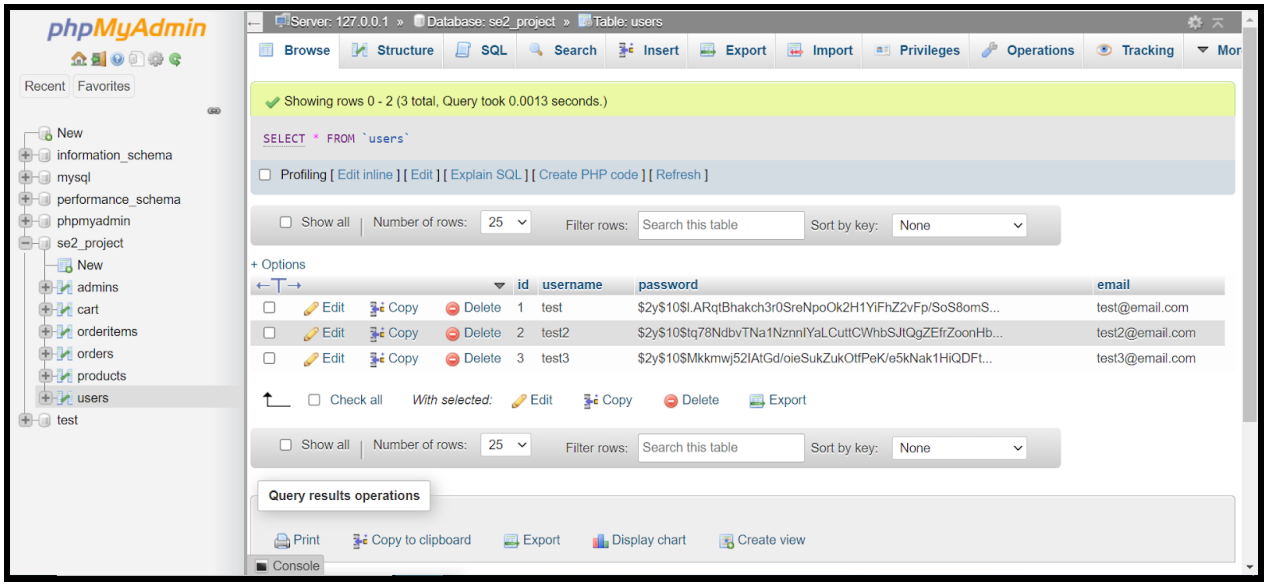
Non-Functional Requirements

Security
Usability

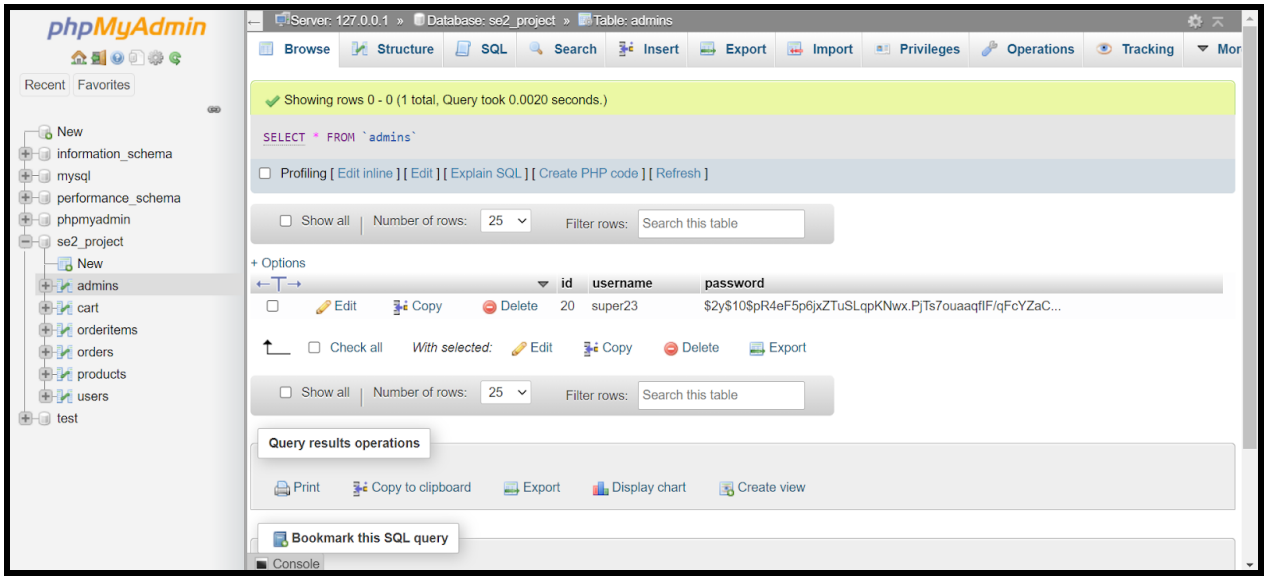
Security:

Security was implemented by storing all passwords in databases with a hash function. A PDO object was also used whenever database queries were made in order to prevent SQL injections. Notifications which were output for users when incorrect information was input during logging in and signing up was left intentionally vague in order to reduce the risk of hacking. I.e., if an incorrect password is given while logging in, informing the user the username *or* password is incorrect as opposed to just the password. This is also done if the username input does not exist. When signing up, if an existing email is input, the user is notified that either the email *or* the username being entered are already taken. This is also the case for if the user tries to sign up with an already existing username.

Another security feature does not allow an admin to be logged in at the same time as a user on the same device. If an admin is logged in and a user logs in on the same device, the admin is logged off. The opposite happens if an admin logs in while a user is logged in on the same device.

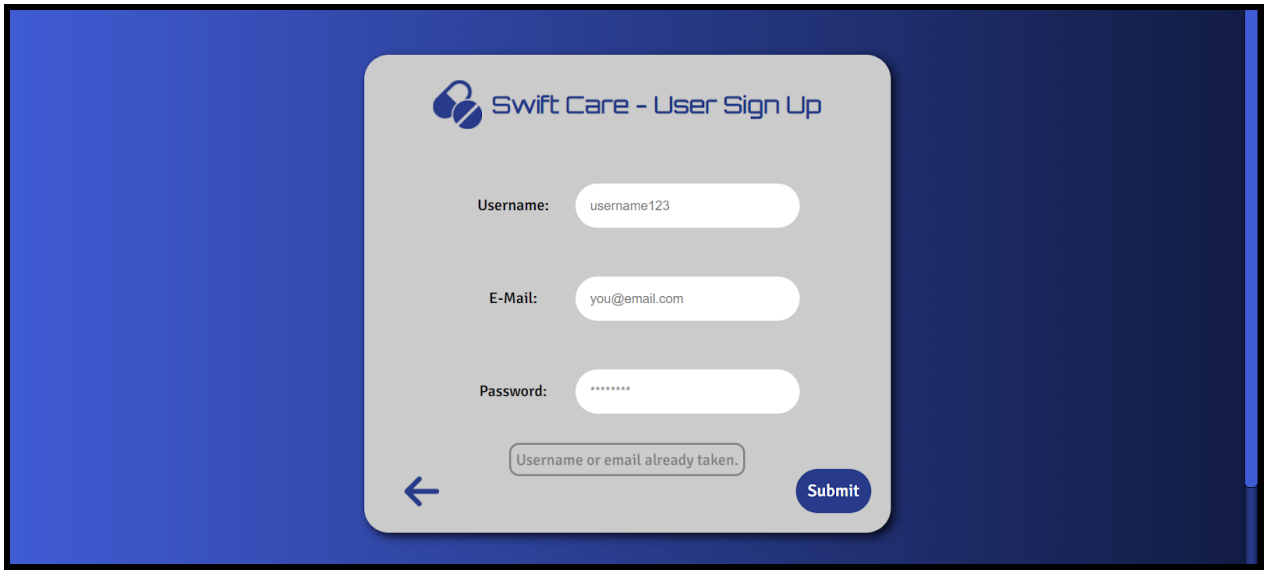


The users table with hashed passwords



The admins table with hashed passwords

Notice output if incorrect username or password are entered for login.

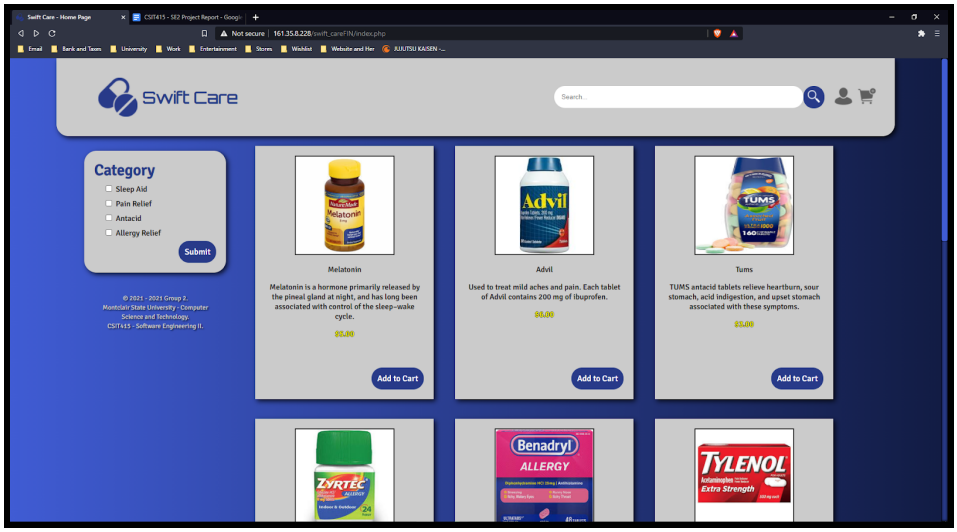


Notice output if already existing username or email are entered for sign up.

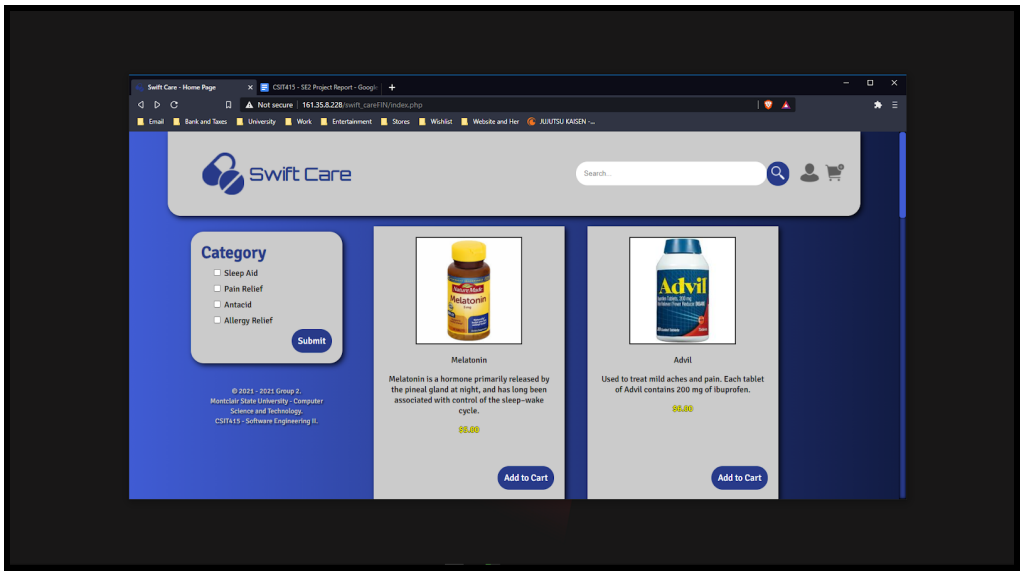
Usability:

Usability was a very important factor in designing *Swift Care* website. The HTML and CSS and the structure of the website was done in a matter that was simplistic, comfortable, understandable, and aesthetically pleasing. Our purpose was to make users want to use our websites over competitors. Here are the following features to describe Usability:

- When users resize the window screen, the website fixes and centers itself based on the current size of that screen. The purpose is to allow users with any size screen to use the website comfortably.



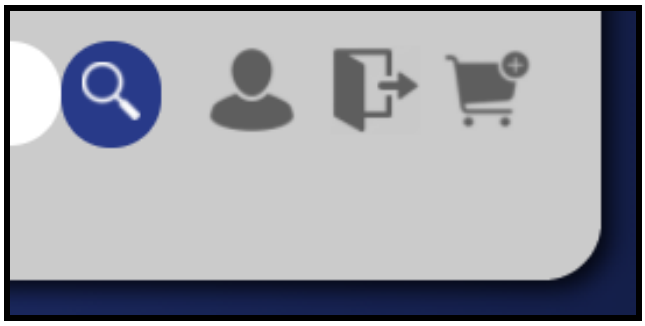
Display of the website in fullscreen (notice three products per row).



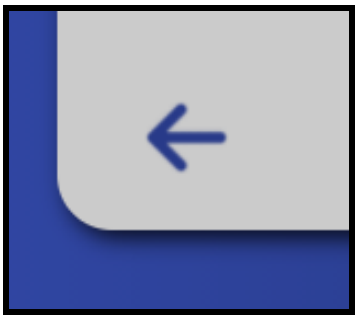
Display of the website not in fullscreen
(notice two products are displayed)

- The overall design of the website is very simplistic as well as aesthetically more universal. Meaning that more people wouldn't be discouraged from using the website based on how it looks.

- The use of icons is heavily prevalent in *Swift Care*. The use of icons provide interface metaphors so that when users enter the website, they will not be required to learn anything, rather just be required to recognize what the icons mean in order to operate the website. A huge example is the cart icon. In our modern society, almost every person knows what a shopping cart looks like. They take that shopping cart to stores such as Shoprite or Stop and Shop. When a user enters our website and sees the cart icon, they will know immediately where their products are being stored when they hit the "Add to Cart" button. Other examples of interface metaphors are the user icon, the back arrow icon, and logout icon (while the user is logged in).



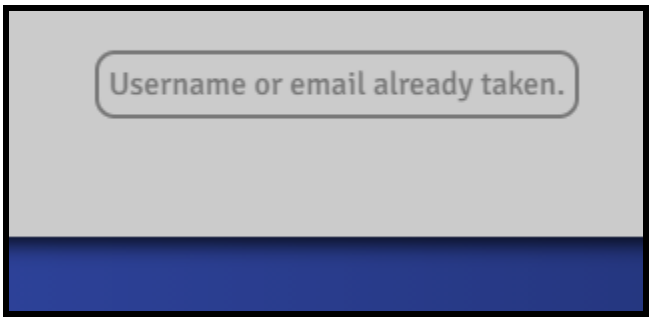
Representation of Search Icon, User Icon Logout Icon, and Cart Icon.



Back Arrow icon

- The use of displaying messages back to the user is also an extremely important concept to usability within our website. In Human Computer Interaction, these terms are described with the concept of gulf of execution and gulf of evaluation. For gulf of execution, a user is trying to understand how to use the system and what they do with it (example: a user trying to log in). The gulf of evaluation is what the user interprets after they completed their goal (Have I logged in? Did I create an account? What happened?). We want to minimize the gulf of execution and gulf of evaluation. So to do this, when a user, for example, creates an account in the website, they will be prompted with an error message in case something goes wrong. This helps the user understand what to do next which would be to retry and use a new username or email. Without these messages displayed back to the user, the user will get confused if they tried to sign up and failed but no message was displayed.

"Signup successful!" message after user creates an account.



When a user attempts to create an account and fails, the message "Username or email already taken." must be displayed to notify them.

Roles of the Team Members:

Nachi:

Set up the server to host and deploy the website, set up product search (by name and by category), set up cart and cart functionality (add/remove items, update quantity to items), set up admin view of orders placed and admin ability to authorize orders.

Arthur:

Largely responsible for the creation of the front-end portion of the website. Created the user interface for *Admin Login* page, *User Login* page, *User Signup* page, *View Products* page, *User View Order* page, *Admin Process Order* page. Provided assistance to the back-end of the *User Login* functionalities. Responsible for creating and modularizing most of the CSS for easy reusability of styling in all the front-end modules. Provided assistance to the rest of the front-end modules. Created Google Sheets module schedule.

Christian:

Organized and deployed initial database tables, in addition to creating the back end files for Admins to execute *Product Entry*, *Updates*, and *Deletion*, as well as User functionality for *Checkout*, *Viewing Order Details*, and *Canceling Orders*.

Hope:

Completed the backend for *Admin Login*, *User Login*, *User Signup* and Admin: *View User Details*. Set up most of the zoom meetings outside of class and helped troubleshoot other backend modules.

Lauren:

Using HTML and CSS, created front-end page *Place Order* for users and *View Order* for admin side. Utilized the reusable CSS to maintain continuity in the pages I worked on.

Joseph:

Created front-end pages for *Admin View and Process Order*, *User View Order*, and *Cancel Order* for User. Was done using HTML and CSS. Also recommended some style changes for simplicity reasons.

Daisy:

Created front-end pages for *Admin: View User Details* and *Product Entry, Update and Delete* using HTML and CSS. Was done using reusable CSS so that the pages would have a cohesive design.

	Module	Sub-Module	Front-End Completion Date	Back-end Completion Date
Timeline ↓	Server	Create Web Server		Nachi - 3/2/21
		Integrate Database with Server		Nachi - 3/5/21
	UX/UI	Design Wireframe (Sketch of Main Page)	Arthur - 3/7/21	
	Admin	Login	Arthur - 4/15/21	Hope - 4/18/2021
	User	Login	Arthur - 4/15/21	Hope + Arthur + Nachi - 5/6/2021
		Sign Up	Arthur - 4/1/21	Hope + Nachi - 4/30/2021
	Admin	Product Entry, Update, Delete	Daisy - 5/7/21	Christian - 4/29/2021
		View User Details	Daisy - 5/7/21	Hope - 5/7/21
	User	View Products	Arthur - 3/16/21	Nachi - 4/18/2021
		Add or Remove from Cart	Arthur - 4/29/2021	Nachi - 4/30/2021
	Payment	Payment API (Fake)	Arthur - 5/4/2021	N/A
	User	Pay using Card (Place Order)	Lauren - 5/2/21	Christian - 5/10/2021
		View Order	Joe - 5/7/21	Christian - 5/12/2021
	Admin	View Order	Lauren - 5/7/21	Nachi - 5/6/2021
		Process Order	Arthur + Joe - 5/7/21	Nachi - 5/6/2021
	User	Cancel Order	Joe - 5/7/21	Christian - 5/12/2021

The Google Sheets "CSIT415 - Group 2 Module Schedule" Completed.

Project Experience:

As a result of the pandemic, not everyone was able to meet up in person. Much of the work had to be done remotely. This made sharing code and integrating modules difficult. Also, everyone having different levels of knowledge when it came to front-end and back-end development made it difficult to figure out who should complete each section. This led to group members having to learn new features they hadn't worked with previously and therefore encouraged pair programming.

Organization also played a much bigger role in the project than expected, and more time should have been put into it before starting development. A uniform naming convention was not used at first, which made it difficult to integrate some pages and functions with each other. Methods of testing and location for storing and sharing code could have been made more clear in the beginning as well. These factors should be considered very early on upon starting another group project.