

Final Project Report

Overview

Our group has designed a full software system that services the orders of a high-end clothing store brand. The system is implemented as a web application with a front-end, back-end, and database.

Stack

- **Front-End**
 - HTML
 - CSS
 - JavaScript
- **Back-End**
 - PHP
- **Database**
 - PostgreSQL

HTML and CSS were used to implement the visual components of our design. This includes the web pages and some of the animations present.

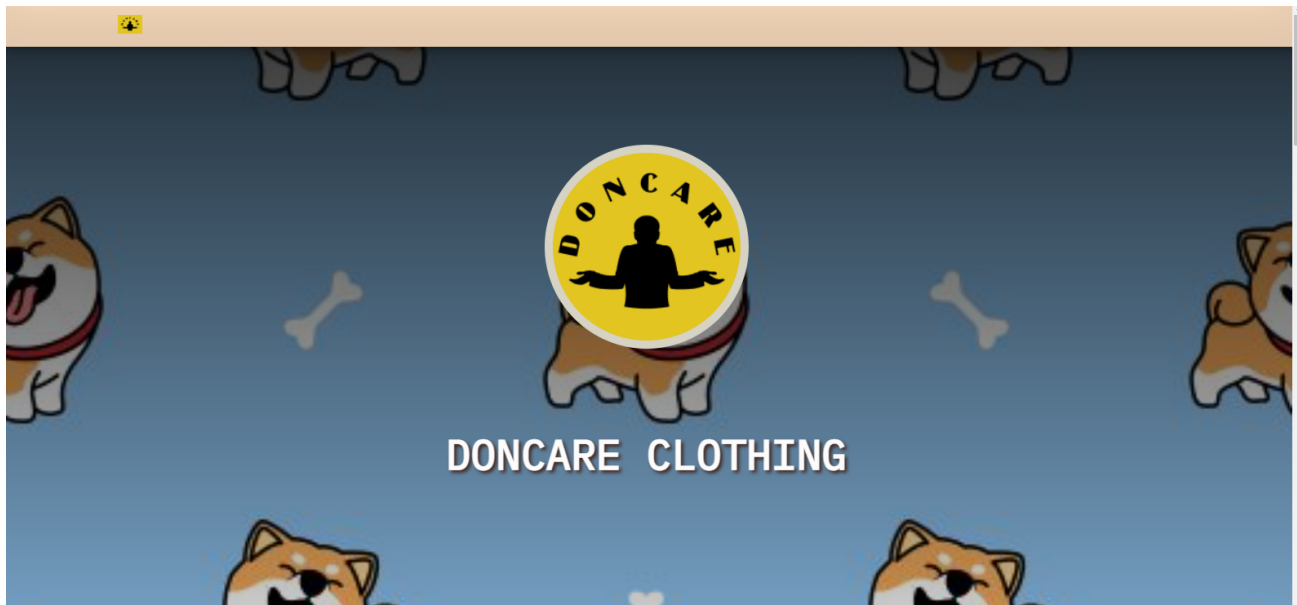
JavaScript was used to script functions client-side. This included verifying that certain form objects properly match and displaying any potential errors thrown in the back-end.

PHP was used to connect our database to our web application. The main application logic is processed in PHP. It facilitates the control structure of the program with branch statements and functions to determine what is stored in our database.

PostgreSQL was chosen as the database for the application due to its familiarity with our group.

Front-End Design

Designing a streetwear clothing store that is utilizing the HTML/ CSS and Javascript to create the front page of the clothing store. The overall design of the website is based on a care-free lifestyle named “Doncare” which is an abbreviation for the phrase “Don’t care”. For example, “Don’t care” has different layers of meaning such as don’t care about your failure, don’t care about the pain you’ve been through, don’t care about what other people say to you.



Main section:

```
<body data-bs-spy="scroll" data-bs-target="#navScrollspy">
  <nav class="navbar navbar-light navbar-expand-lg fixed-top" style='
    background-color: #2c3e50; color: white; padding: 5px 0;
  >
    <div class="container">
      <a href="#hero-section" class="navbar-brand">
        <div class="flip-container">
          <div class="flipper">
            <div class="front">
              
            <div class="back"></div>
          </div>
        </div>
      </a>
    </div>
  </nav>

  <section id="hero-section">
    <div class="avatar-container">
      <div class="img-container">
        <div id="profile-img">
          </div>
        </div>
        <h2 class="display-2 animate__animated animate__jackInTheBox">
        </div>
      </section>
```

Adding animations to the home page button and also the store logo



```
.flip-container {
  margin: 0 auto;
  perspective: 1000;
}

.flip-container:hover .flipper {
  transform: rotateY(180deg);
}

.flipper {
  transition: 0.8s;
  transform-style: preserve-3d;
}

.front, .back {
  backface-visibility: hidden;
  -webkit-backface-visibility: hidden;
  position: absolute;
}

.flip-container, .front, .back {
  width: 30px;
  height: 30px;
}

.front {
  z-index: 1;
  border-radius: 50%;
  top: -1px;
}

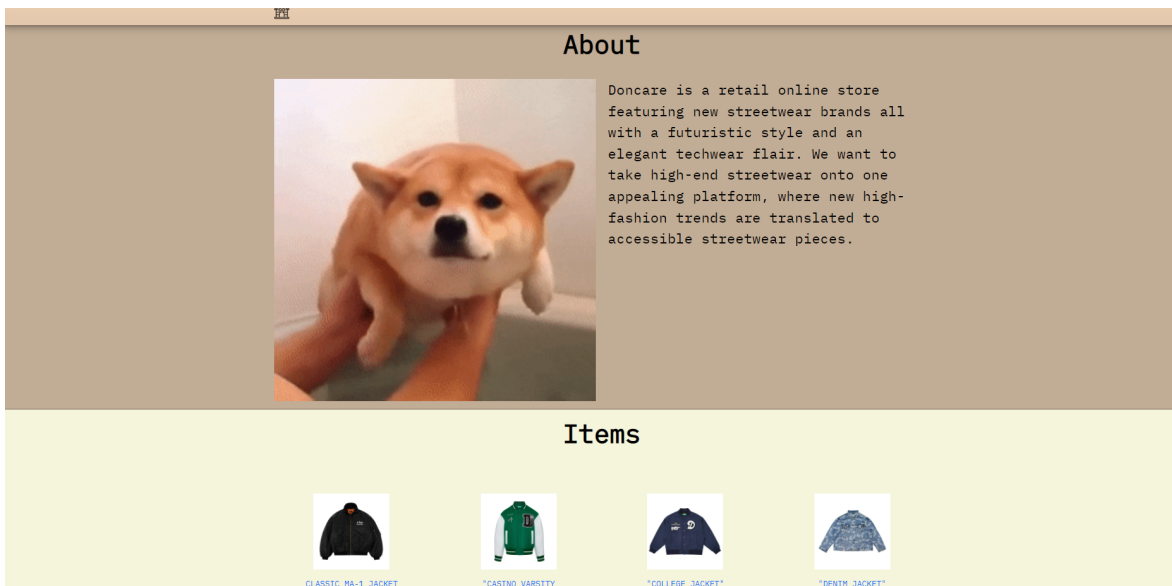
.front img {
  vertical-align: middle !important;
  color: #a2babe;
}

.back {
  border-radius: 50%;
  transform: rotateY(180deg);
}

.back img {
  width: 35px;
  height: auto;
  border-radius: 50%;
}
```

About section:

```
<section id="mission-section">
  <div class="container">
    <div class="row text-center">
      <div class="col-12">
        <div class="mission-container">
          <h3 class="display-2 text-center">About</h3>
        </div>
      </div>
    </div>
    <div class="row text-center">
      <div class="col-12 col-md-6">
        <div class="mission-container">
          
        <div class="mission-container">
          <p class="lead"> Doncare is a retail online store featur:
We want to take high-end streetwear onto one appealing platform, wher
          </p>
        </div>
      </div>
    </div>
  </div>
</section>
```



Used -col to separate between the gif and the store description

Item section:

```
<div class="row text-center">
  <h3 class="display-2 text-center" >Items</h3>
  <div class="col-12 col-md-6 col-lg-3">
    <div class="skill-container">
      </p>
      <p><a href="./checkout.php?order=1">CLASSIC MA-1 JACKET</a>
      <p>150$ </p>
    </div>
  </div>
  <div class="col-12 col-md-6 col-lg-3">
    <div class="skill-container">
      </p>
      <p><a href="./checkout.php?order=2">"CASINO VARSITY JACK
      <p>100$</p>
    </div>
  </div>
  <div class="col-12 col-md-6 col-lg-3">
    <div class="skill-container">
      </p>
      <p><a href="./checkout.php?order=3">"COLLEGE JACKET"</a><
      <p>200$</p>
    </div>
  </div>
  <div class="col-12 col-md-6 col-lg-3">
    <div class="skill-container">
      </p>
      <p><a href="./checkout.php?order=4">"DENIM JACKET"</a></p>
      <p>200$</p>
    </div>
  </div>
</div>
</div>
</section>
```

Separate each items with div to make them into 4 rows and assort the items into each rows

Carousel section:

```
<div class="space">
<section class="container text-center py-5" id="inspiration-section">
  <h3 class="display-4 text-center">Trending</h3>
  <div id="carouselExampleCaptions" class="carousel slide" data-
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleCapti
    <button type="button" data-bs-target="#carouselExampleCapti
    <button type="button" data-bs-target="#carouselExampleCapti
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active drk">
      

      </div>
    </div>
    <div class="carousel-item drk">
      

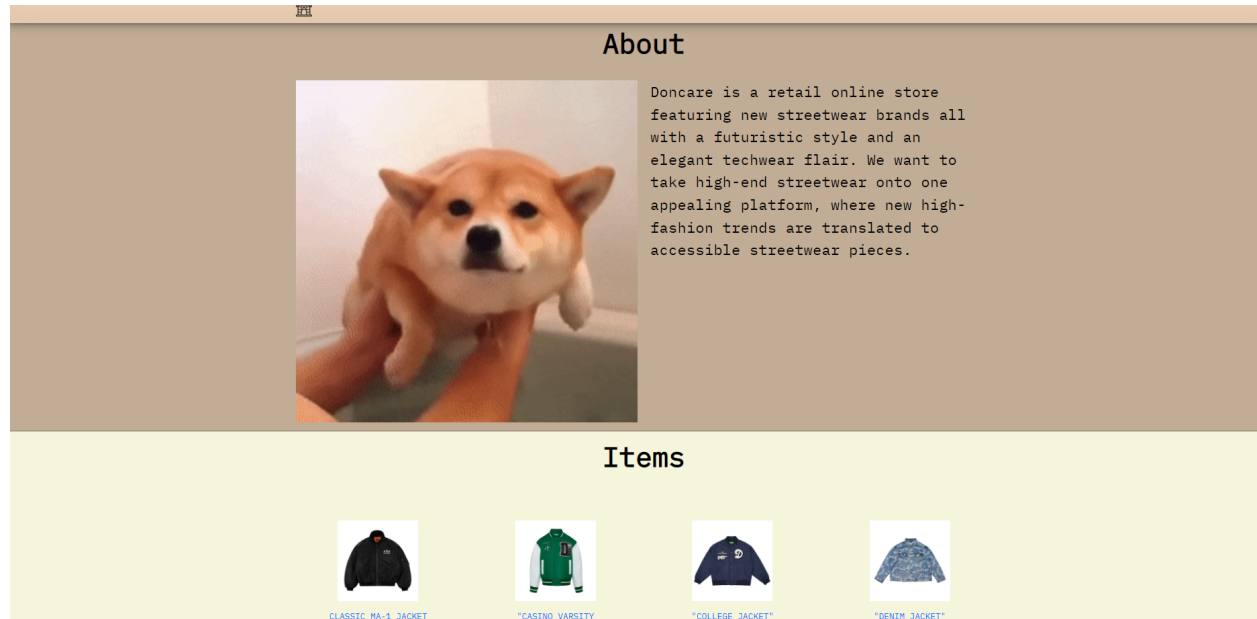
      </div>
    </div>
    <div class="carousel-item drk">
      

      </div>
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-1
    <span class="carousel-control-prev-icon" aria-hidden="true"
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-1
    <span class="carousel-control-next-icon" aria-hidden="true"
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

Include the motion to the carousel section with 3 pictures so that they can move automatically by themselves.

Back-End Design

Getting an Order:



Once a user is on the home screen of the website, a user can choose to order four items. These items are linked to the following checkout screen when they are clicked.

Checkout.php receives input from the URL to display the correct item in the checkout screen.

```
<div class="col-12 col-md-6 col-lg-3">
  <div class="skill-container">
    
    <p></p>
    <p><a href="./checkout.php?order=1">CLASSIC MA-1 JACKET</a></p>
    <p>150$ </p>
  </div>
</div>
<div class="col-12 col-md-6 col-lg-3">
  <div class="skill-container">
    
    <p></p>
    <p><a href="./checkout.php?order=2">"CASINO VARSITY JACKET"</a></p>
    <p>100$</p>
  </div>
</div>
<div class="col-12 col-md-6 col-lg-3">
  <div class="skill-container">
    
    <p></p>
    <p><a href="./checkout.php?order=3">"COLLEGE JACKET"</a></p>
    <p>200$</p>
  </div>
</div>
<div class="col-12 col-md-6 col-lg-3">
  <div class="skill-container">
    
    <p></p>
    <p><a href="./checkout.php?order=4">"DENIM JACKET"</a></p>
    <p>200$</p>
  </div>
</div>
```

Displaying Checkout Screen:

Doncare

Billing Address

Full Name
John More Doe

Email
john@example.com

Address
542 W. 15th Street

City
New York

State
NY

Zip
10001

Payment

Accepted Cards
Visa Mastercard Apple Pay Google Pay

Name on Card
John More Doe

Credit card number
1111-2222-3333-4444

Exp Month
September

Exp Year
2018

CVV
352

[Continue to checkout](#)

Already Have An Account?

[Login](#)

[Register](#)

Cart

CLASSIC MA-1 JACKET \$150

Total \$150

```
$name = @$_GET["name"];  
$email = @$_GET["email"];  
$address = @$_GET["address"];  
$city = @$_GET["city"];  
$state = @$_GET["state"];  
$zip = @$_GET["zip"];
```

The checkout screen can receive input from a query in the URL to set the values inside the text boxes. If no input for these variables are received, they are set to null.

```
<input type="text" id="name" name="name" placeholder="John More Doe" value="php if(is_null($name)) {echo "";} else {echo $name;} ?" required>  
<label for="email"><i class="fa fa-envelope"></i> Email</label>  
<input type="text" id="email" name="email" placeholder="john@example.com" value="php if(is_null($email)) {echo "";} else {echo $email;} ?" required>  
<label for="adr"><i class="fa fa-address-card-o"></i> Address</label>  
<input type="text" id="adr" name="address" placeholder="542 W. 15th Street" value="php if(is_null($address)) {echo "";} else {echo $address;} ?" required>  
<label for="city"><i class="fa fa-institution"></i> City</label>  
<input type="text" id="city" name="city" placeholder="New York" value="php if(is_null($city)) {echo "";} else {echo $city;} ?" required>  
<input type="hidden" name="orderId" value="php echo htmlspecialchars($orderId);?" />  
  
<div class="row">  
<div class="col-50">  
<label for="state">State</label>  
<input type="text" id="state" name="state" placeholder="NY" value="php if(is_null($state)) {echo "";} else {echo $state;} ?" required>  
</div>  
<div class="col-50">  
<label for="zip">Zip</label>  
<input type="text" id="zip" name="zip" placeholder="10001" value="php if(is_null($zip)) {echo "";} else {echo $zip;} ?" required>  
</div>  
</div>
```


The above code shows how these variables are used within the HTML. The value parameter for the input tag contains PHP code which checks to see if the variables are null. If they are null, then nothing is displayed within the textbox. If not, then whatever value that is present is displayed.

```
$error = @$_GET["error"];
if ($error == 1) {
    echo '<script type="text/javascript">
        |
        |         window.onload = function () { alert("User name or password is bad."); }
        |     </script>';
}

$orderId = $_GET["order"];
$cart = pg_query($dbconn, "SELECT productname, price FROM products WHERE productid='$orderId'");
$row = pg_fetch_row($cart);
if ($error == 2) {
    echo "<script type='text/javascript'>
        |
        |         window.onload = function () { alert('Item is out of stock or does not exist.')} }
        |     </script>";
}
```

Checkout.php also contains error functions that are related to the two possible errors when checking out.

The first error can occur if a user tries to login with a bad email or password. If this occurs, an alert is displayed stating that their username or password is bad.

The second error is for when the item that the user is trying to order is out of stock.

Both of these error functions are implemented with PHP but displayed with JavaScript.

```

<div class="col-50">
  <h3>Payment</h3>
  <label for="fname">Accepted Cards</label>
  <div class="icon-container">
    <i class="fa fa-cc-visa" style="color:navy;"></i>
    <i class="fa fa-cc-amex" style="color:blue;"></i>
    <i class="fa fa-cc-mastercard" style="color:red;"></i>
    <i class="fa fa-cc-discover" style="color:orange;"></i>
  </div>
  <label for="cname">Name on Card</label>
  <input type="text" id="cname" name="cardname" placeholder="John More Doe" required>
  <label for="ccnum">Credit card number</label>
  <input type="text" id="ccnum" name="cardnumber" placeholder="1111-2222-3333-4444" required>
  <label for="expmonth">Exp Month</label>
  <input type="text" id="expmonth" name="expmonth" placeholder="September" required>
  <div class="row">
    <div class="col-50">
      <label for="expyear">Exp Year</label>
      <input type="text" id="expyear" name="expyear" placeholder="2018" required>
    </div>
    <div class="col-50">
      <label for="cvv">CVV</label>
      <input type="text" id="cvv" name="cvv" placeholder="352" required>
    </div>
  </div>
</div>

```

The checkout screen also includes a form to include payment information, however this is not connected to our database. This is because in practice it would be connected to a cloud service payment processor to protect the information from being potentially compromised.

Adding a Registered User:

Doncare

Please fill in this form to create an account

Email

Enter Email

Password

Enter Password

Repeat Password

Repeat Password

Address

542 W. 15th Street

City

New York

State

NY

Zip

10001

Register

A user must fill in each input box to properly register. This is implemented using the input tag's required attribute.

```
<form onSubmit="return checkPassword(this)" action="./scripts/addUser.php" method="POST" >
  <div class="container">
    <a href="//localhost" style="color: black;"><h1>Doncare</h1> </a>
    <p>Please fill in this form to create an account.</p>
    <hr>

    <label for="email"><b>Email</b></label>
    <input type="text" placeholder="Enter Email" name="email" id="email" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="psw" id="psw" required>

    <label for="psw-repeat"><b>Repeat Password</b></label>
    <input type="password" placeholder="Repeat Password" name="psw-repeat" id="psw-repeat" required>
    <hr>
  </div>

  <div class="row">
    <div class="col-50">
      <label for="adr"><i class="fa fa-address-card-o"></i> <b>Address</b></label>
      <input type="text" id="adr" name="address" placeholder="542 W. 15th Street">
      <label for="city"><i class="fa fa-institution"></i> <b>City</b></label>
      <input type="text" id="city" name="city" placeholder="New York">

      <div class="row">
        <div class="col-50">
          <label for="state"><b>State</b></label>
          <input type="text" id="state" name="state" placeholder="NY">
        </div>
        <div class="col-50">
          <label for="zip"><b>Zip</b></label>
          <input type="text" id="zip" name="zip" placeholder="10001">
        </div>
      </div>
    </div>
  </div>
</form>
```

To reduce error when a user enters a password, a JavaScript function is created to check to ensure that the two password fields match.

This function returns once the form is submitted.

```
<script>
function checkPassword(form) {
  password1 = form.psw.value;
  password2 = form["psw-repeat"].value;

  // If Not same return False.
  if (password1 != password2) {
    alert ("\nPassword did not match: Please try again...")
    return false;
  }
}
</script>
```

Inside of the addUser.php file, we take input parameters and sanitize the input to prevent SQL injection attacks.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $email = test_input($_POST["email"]);  
    $password = test_input($_POST["psw"]);  
    $addressid = uniqid();  
    $address = test_input($_POST["address"]);  
    $city = test_input($_POST["city"]);  
    $state = test_input($_POST["state"]);  
    $zip = test_input($_POST["zip"]);  
}
```

The test_input function removes whitespace, slashes, and turns special characters into their HTML codes.

```
function test_input($data) {  
    $data = trim($data);  
    $data = stripslashes($data);  
    $data = htmlspecialchars($data);  
    return $data;  
}
```

We then check to see if the input email already exists within the user table.

If the user exists, we pass in an error value for it to be displayed to the register page.

```
7 $checkUser = pg_query($dbconn, "SELECT email FROM users WHERE email='$email'");  
8 $row = pg_fetch_row($checkUser);  
9 if ($row[0] == $email) {  
0     header("Location: //localhost/register.php?error=1");  
1     exit();  
2 }
```

The error is then displayed back on the register.php file.

```
if ($error == 1) {  
    echo '<script type="text/javascript">  
        window.onload = function () { alert("User already exists."); }  
    </script>';  
}  
>>
```

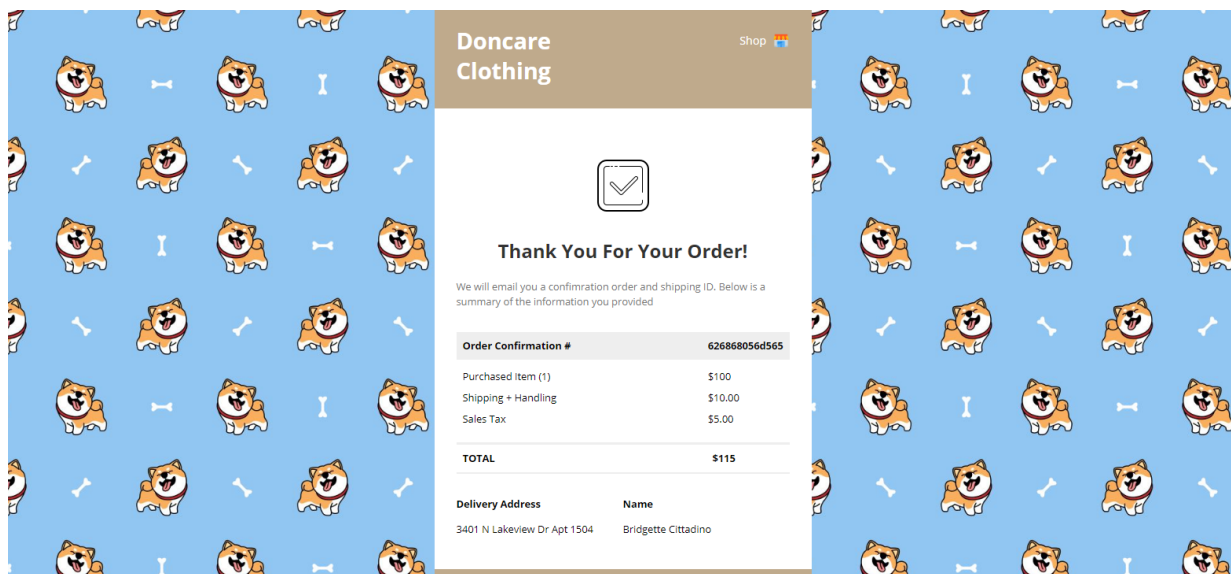
If the check passes, then we insert the input values into the appropriate tables. We finally return to the home page of the site.

```
$addAdd = "INSERT INTO addresses VALUES ('$addressid', '$city', '$state', '$zip', '$address')";
$addUser = "INSERT INTO users VALUES ('$email', '$password')";
$addHave = "INSERT INTO have VALUES ('$email', '$addressid')";

$resultAdd = pg_query($dbconn, $addAdd);
$resultUser = pg_query($dbconn, $addUser);
$resultHave = pg_query($dbconn, $addHave);

header('Location: //localhost');
```

Adding a Customer:



We perform the same process as registering a new user.

```
$name = test_input($_POST["name"]);
$email = test_input($_POST["email"]);
$addressid = uniqid();
$address = test_input($_POST["address"]);
$city = test_input($_POST["city"]);
$state = test_input($_POST["state"]);
$zip = test_input($_POST["zip"]);
```

We then check to ensure the item is in stock before ordering the product. If the item is in stock, then update the quantity of items and check to see if there are any items left. If there are no items left, we update the instock column to say 'no'.

If an item that is trying to be ordered is not in stock, then an error is thrown that will display on the checkout screen stating: 'Item is out of stock.'.

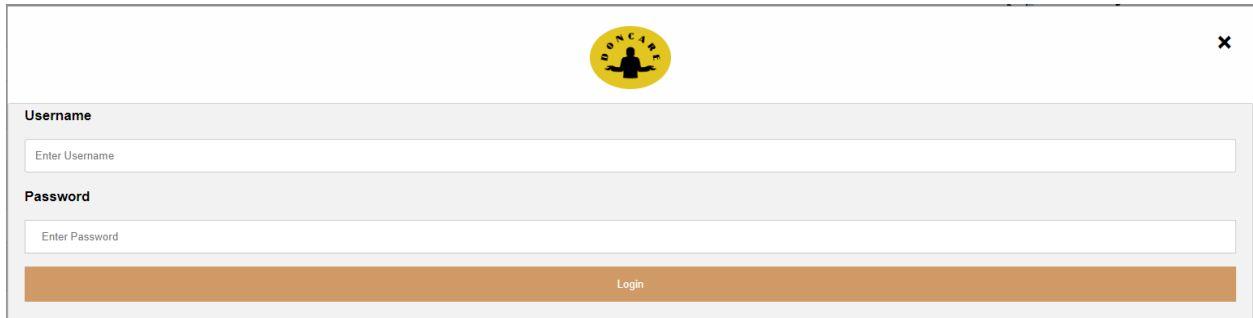
```
$prodName = pg_query($dbconn, "SELECT * FROM products WHERE productid='$orderId'");
$row = pg_fetch_row($prodName);

if ($row[4] == 'yes') {
    $updateOrd = "UPDATE products SET quantity=quantity-1 WHERE productid=$orderId";
    pg_query($dbconn, $updateOrd);
    $updateTest = pg_query($dbconn, "SELECT * FROM products WHERE productid='$orderId'");
    $row2 = pg_fetch_row($updateTest);
    if ($row2[3] == 0) {
        pg_query($dbconn, "UPDATE products SET instock='no' WHERE productid=$orderId");
    }
} else {
    header("Location: //localhost/checkout.php?order=$orderId&error=2");
    exit();
}
```

We finally add the address and customer as a part of our order to their respective tables.

```
$addAdd = "INSERT INTO addresses VALUES ('$addressid', '$city', '$state', '$zip', '$address')";
$addCust = "INSERT INTO customers VALUES ('$custId', '$name', '$email', '$orderId', '$addressid')";
$resultAdd = pg_query($dbconn, $addAdd);
$resultCust = pg_query($dbconn, $addCust);
```

Checking if a Customer is a User:



Once the login button has been clicked inside the checkout form, a user will input an already registered user into the Username and Password text boxes.

```
<button onclick="document.getElementById('id01').style.display='block'" style="width:auto; background-color: #CF9A66">Login</button>

<p><a href="./register.php">Register</a></p>

<div id="id01" class="modal">

  <form class="modal-content animate" action="./scripts/checkoutUser.php" method="post">
    <div class="imgcontainer">
      <span onclick="document.getElementById('id01').style.display='none'" class="close" title="Close Modal">&times;</span>
      
    </div>

    <div class="container">
      <label for="uname"><b>Username</b></label>
      <input type="text" placeholder="Enter Username" name="uname" required>

      <label for="psw"><b>Password</b></label>
      <input type="password" placeholder="Enter Password" name="psw" required>
      <input type="hidden" name="orderId" value="<?php echo htmlspecialchars($orderId);?>" />

      <button type="submit">Login</button>
    </div>
  </form>
</div>
```

The username, password, and order is then validated.

```
$user = test_input($_POST["uname"]);
$password = test_input($_POST["psw"]);
$order=test_input($_POST["orderId"]);
```

If the username is not in the users table or if the password inputted does not match the one stored in the table, an error is thrown. The user is returned back to the checkout page.

If the password and username are good, then the script grabs the address stored with the user in the have table and returns it back to the checkout page.

```
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}

$username = pg_query($dbconn, "SELECT * FROM users WHERE email='$user'"); //check if user exists for checkout
$row = pg_fetch_row($username);

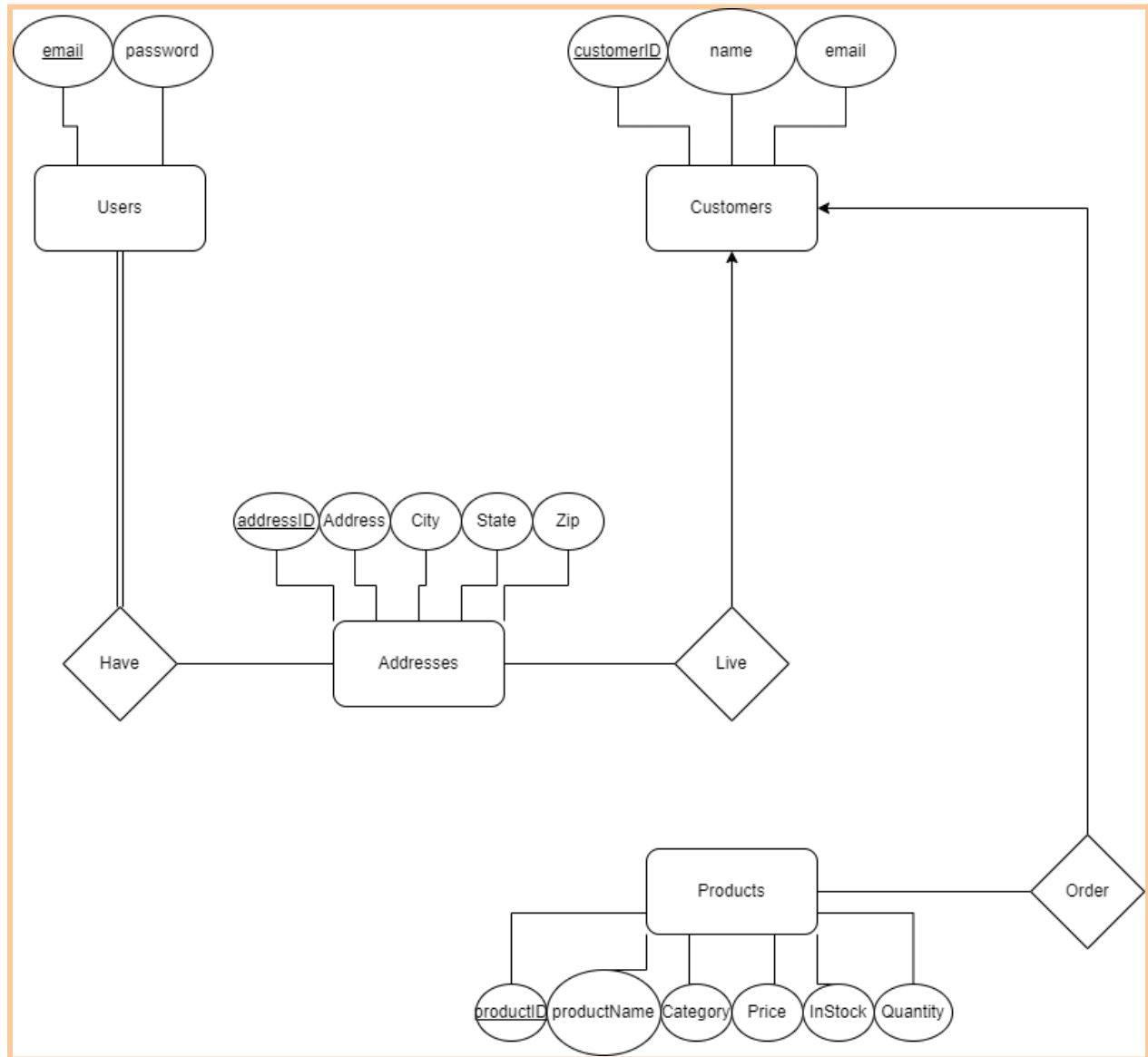
if ($row == false || $row[1] != $password) { //throw error if password is incorrect or user doesnt exist
    header("Location: //localhost/checkout.php?order=$order&error=1");
    exit();
}

$getAddID = pg_query($dbconn, "SELECT addressid FROM have WHERE email='$user'"); //grab address attached to user
$row = pg_fetch_row($getAddID);

$addQ = pg_query($dbconn, "SELECT * FROM addresses WHERE addressid='$row[0]'");
$row = pg_fetch_row($addQ);

header("Location: //localhost/checkout.php?order=$order&email=$user&city=$row[1]&state=$row[2]&zip=$row[3]&address=$row[4]"); //fill user info for checkout.php
exit();
}
```


Database Design



The final database design consists of five tables which helps the whole website work and create space to store users and products. I will attach screenshots below of each class and talk about their role in the body of this structure. Postgres was used to make the initial design of the database where everything was done through the terminal. After doing a dump, we were able to edit the database through vscode utilizing sql.

```

CREATE TABLE public.addresses (
    addressid character varying(13) NOT NULL,
    city character varying(50) NOT NULL,
    state character varying(50) NOT NULL,
    zip character varying(10) NOT NULL,
    address character varying(70)
);

ALTER TABLE public.addresses OWNER TO postgres;

--
-- Name: customers; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.customers (
    "custID" character varying(13) NOT NULL,
    name character varying(50) NOT NULL,
    email character varying(50) NOT NULL,
    "prodID" integer NOT NULL,
    addressid character varying(13)
);

ALTER TABLE public.customers OWNER TO postgres;

```

Our first table is a table called Customers which contains 3 elements: custID, name, and email. The primary key for this is custID. The elements productID and addressID are foreign keys from different tables which will be used for when a customer selects a product to purchase and is required to put in their address for shipping purposes. Customers are anyone who purchases anything off the website, so their information will be stored here.

Our next table is called Addresses which contains 5 elements: addressid, city, state, zip, and address. The primary key is addressid. This table consists of no foreign keys, however it has a foreign key for other tables. The purpose of this table is so we know

where to ship customer products. For users their address is already stored, however customers who are not users must enter their address while at the checkout screen.

```
CREATE TABLE public.have (  
    email character varying(120) NOT NULL,  
    addressid character varying(13) NOT NULL  
);  
  
ALTER TABLE public.have OWNER TO postgres;  
  
--  
-- Name: products; Type: TABLE; Schema: public; Owner: postgres  
--  
  
CREATE TABLE public.products (  
    productid integer NOT NULL,  
    productname character varying(50) NOT NULL,  
    category character varying(50) NOT NULL,  
    price real NOT NULL,  
    instock character varying(20) NOT NULL,  
    quantity integer NOT NULL  
);  
  
ALTER TABLE public.products OWNER TO postgres;
```

Our third table is called Have and it has 2 elements: email and addressid. These elements are not its own, they are foreign keys from table Users and Addresses respectfully. However, they are both primary keys for this table. The purpose of this table is that it connects the tables Users and Address together because all users have an address since it is part of signing up (in the front end).

Our fourth table is called Products and it has 6 elements: productID, productname, category, price, instock, and quantity. The primary key is productID. This table does not contain any foreign keys. The purpose of this table is that it contains all of the information concerning every product that we sell on our website.

```
CREATE TABLE public.users (
    email character varying(120) NOT NULL,
    password character varying(255) NOT NULL
);
```

Our last table is called Users and it consists of 2 elements: email and password. The primary key is email. This table does not have any foreign keys. A user is someone who creates an account with the website, all they need is an email and password. The password has been changed to be 255 characters long since we are incorporating encryption. A user also fills out their address on the front end.

Foreign Keys

Below are images of all the constraints talked about in the previous section.

```
ALTER TABLE ONLY public.customers
    ADD CONSTRAINT "customers2_prodID_fkey" FOREIGN KEY ("prodID") REFERENCES public.products(productid) NOT VALID;

--
-- Name: have fk2; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.have
    ADD CONSTRAINT fk2 FOREIGN KEY (email) REFERENCES public.users(email);

--
-- Name: have pk1; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.have
    ADD CONSTRAINT pk1 FOREIGN KEY (addressid) REFERENCES public.addresses(addressid);
```

```
ALTER TABLE ONLY public.addresses
    ADD CONSTRAINT addresses_pkey PRIMARY KEY (addressid);

--
-- Name: customers customers2_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.customers
    ADD CONSTRAINT customers2_pkey PRIMARY KEY ("custID");

--
-- Name: have have_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.have
    ADD CONSTRAINT have_pkey PRIMARY KEY (email, addressid);

--
-- Name: products products_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.products
    ADD CONSTRAINT products_pkey PRIMARY KEY (productid);

--
-- Name: users users_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_pkey PRIMARY KEY (email);

--
-- Name: customers customers2_addressid_fkey; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.customers
    ADD CONSTRAINT customers2_addressid_fkey FOREIGN KEY (addressid) REFERENCES public.addresses(addressid) NOT VALID;
```

Sources

- <https://bbbootstrap.com/snippets/order-confirmation-email-template-19073214>
 - HTML/CSS template for confirmation page
- https://www.w3schools.com/howto/howto_css_login_form.asp
 - HTML/CSS template for login button
- https://www.w3schools.com/howto/howto_css_register_form.asp
 - Used as the basis for register page
- https://www.w3schools.com/howto/howto_css_checkout_form.asp
 - HTML/CSS template for checkout form
- https://www.w3schools.com/php/php_form_validation.asp
 - Provided test_input() which helped validate GET and POST input
- <https://doncare-club.com/>
 - Used logo and name for rebranding of clothing store