

**CATSHOP**

## **COMP3340 - Final Project**

*CatShop - Documentation Manual*

Joshua Lumb (103255040)  
Jonah Moses (101726496)  
Dany Sanioura (110009461)  
Mustafa Fawaz (103184737)  
COMP-3340  
Instructor: Paul Preney  
March 2020

## **TABLE OF CONTENTS**

### [1.0 OVERVIEW](#)

#### [1.1 WEBSITE FEATURES & USERS](#)

### [2.0 INSTALLATION](#)

#### [2.1 FILE STRUCTURE](#)

#### [2.2 DATABASE SETUP](#)

#### [2.3 HARD-CODED ELEMENTS](#)

### [3.0 FEATURES](#)

#### [3.1 HTML & CSS](#)

##### [3.1.1 Layout](#)

##### [3.1.2 Header and Nav-Bar](#)

##### [3.1.3 Main Body](#)

##### [3.1.4 Footer](#)

##### [3.1.5 Shop, Contact Us, Sign-Up and Others](#)

#### [3.2 JAVASCRIPT AND AJAX](#)

#### [3.3 PHP & MySQL](#)

#### [3.4 SECURITY](#)

### [4.0 EXTRAS](#)

#### [4.1 ADDITIONAL FEATURES](#)

##### [4.1.1 CSS @Media Tags](#)

# 1.0 OVERVIEW

## 1.1 WEBSITE FEATURES & USERS

For our project we designed an online store for a fictional company that sells cats and cat related accessories.

When the user arrives at the website, they are greeted by an appealing and modern design.

Guest users (those who are not logged in), are able to view the about page, contact us page, store page, and sign up page. Store functions cannot be accessed without logging in, however store content is anyone.

A guest user may use the sign up page to register for the website, but is not allowed to sign up with a username that is already being used or with invalid credentials.

All users, including guests, may access the contact us page. The contact us page allows anyone to contact a representative of CATSHOP for any reason. All they need to do is provide a name, email, and a message and that message will be forwarded to the CATSHOP admin staff.

All users, including guests, may access the about us page. The about us page is one way to help customers get to know CATSHOP and share the latest information regarding CATSHOP. Although all users can view this page, only the admin user can create, remove, and update the information on this page.

Once a user has registered for an account, he or she may log in to the website using their new credentials. Logging in allows a user to access store content, add store products (cats) to a cart, and check out (note: checkout is not currently implemented as no group members are willing to sell our cats).

Logged in users also gain access to the "My Account" page where they may create, remove, and update their delivery address. This address information remains persistent across individual user sessions/logins.

Navigation to the sign up page is hidden to logged in users, but if they decide to go to that page by trying to access the signup.php resource, the content of the page is hidden and they are told that they are already registered users.

We hope you enjoy testing our website as much as we enjoyed making it.

## 2.0 INSTALLATION

### 2.1 FILE STRUCTURE

All files are kept in the /public\_html directory of the domain, however not all of those files are directly accessible by users. For example, users are prohibited from accessing files in the /css, /internal and /images directories. These items can only be accessed via the web pages that incorporate them in their code.

The root directory contains all pages that will be accessed directly by the user including index.php, signup.php, etc.

The /css directory contains style sheets for each page.

The /images directory contains all image files that are used on the web page.

The /internal directory contains .php and .js files that will not be accessed directly by users. These files will be accessed in the background by pages that are visible to the user in order to accomplish various tasks such as connecting pages to the sql databases, event handling, and generally controlling page behaviour.

### 2.2 DATABASE SETUP

You will find our existing database credentials in the databasehandler.internal.php file. These can be used to access an existing database containing the website data.

Alternatively, if one wishes to test the website on their own database, one can be created and those credentials can be substituted into the databasehandler.internal.php file.

Database creation can be done using the DirectAdmin control panel under Account Manager->MySQL Management. When the database is created, a user of the database must also be created.

In phpMyAdmin, simply run the provided script **createtables.sql** to populate your newly created database with our sample data including paths to images, product data, user table with predefined admin account, and “about us” data.

Fig 2.3 - users table

```
/* registered users table */  
CREATE TABLE users (  
  
    usernum int(11) NOT NULL AUTO_INCREMENT,  
    username varchar(50) NOT NULL,  
    password varchar(50) NOT NULL,  
    address text(6500),  
    PRIMARY KEY (usernum)  
  
);
```

The users table contains all user data and is mainly used to register and login users. This data is divided into 3 attributes: usernum (int, primary key), username (varchar(50)), password (varchar(50)), and address (text(6500)).

Fig 2.2 - products table

```
/* Products Table */  
CREATE TABLE products (  
  
    id int(11) NOT NULL,  
    nam varchar(255) NOT NULL,  
    img varchar(255) NOT NULL,  
    price float NOT NULL,  
    PRIMARY KEY (id)  
  
);
```

The products table contains all product data and is used to store and access shop items. This data is divided into 4 attributes: id (int, primary key), nam (varchar(255)), img (varchar(255)), and price (float).

Fig 2.3 - about\_us table

```
/* About us table for admin modification */  
CREATE TABLE about_us (  
  
    id int(11) NOT NULL,  
    about text(6500),  
    PRIMARY KEY (id)  
  
);
```

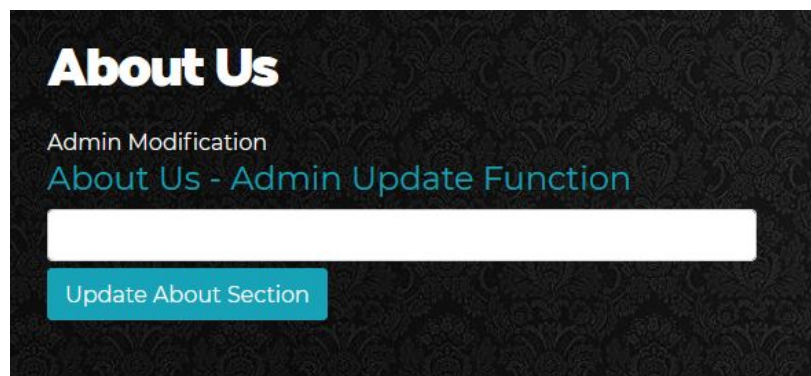
The about\_us table contains all modular/editable data for the site to display. This data is divided into 2 attributes: id (int, primary key) and about (text(6500)).

## 2.3 HARD-CODED ELEMENTS

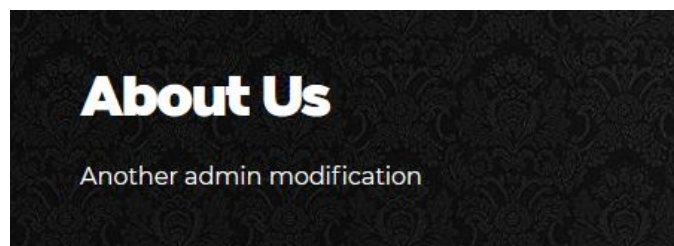
Only a user signed in with **admin** credentials (username: admin / password: admin) can update the text on the “about us” page (see figures 2.4 - 2.5). Only a user who is logged in can access cart functions. These functionalities are hard coded into their respective pages.

The products table in the sql database also contains various hard coded products and these can be seen by looking at the “insert into” statements in createtables.sql.

*Fig 2.4 - Before Admin Modification*

A screenshot of a web application interface for updating the 'About Us' page. The background is dark with a subtle pattern. At the top, the text 'About Us' is displayed in a large, bold, white font. Below it, the text 'Admin Modification' appears in a smaller white font, followed by 'About Us - Admin Update Function' in a teal color. A white text input field is positioned below the text. At the bottom, there is a teal button with the text 'Update About Section' in white.

*Fig 2.5 - After Admin Modification*



## 3.0 FEATURES

### 3.1 HTML & CSS

*\*For this section, we encourage users to have the webpage open to try all features. Only important code-snippets and web-page snapshots will be shown\**

#### 3.1.1 Layout

Early on in the development of our website, we created a skeleton that allowed us to frame the pages moving forward. This skeleton (Fig 3.1) became the final design as we began styling the front-end and coding all back-end features. Each page will be briefly described below outlining the overall HTML/CSS features on our webpage.

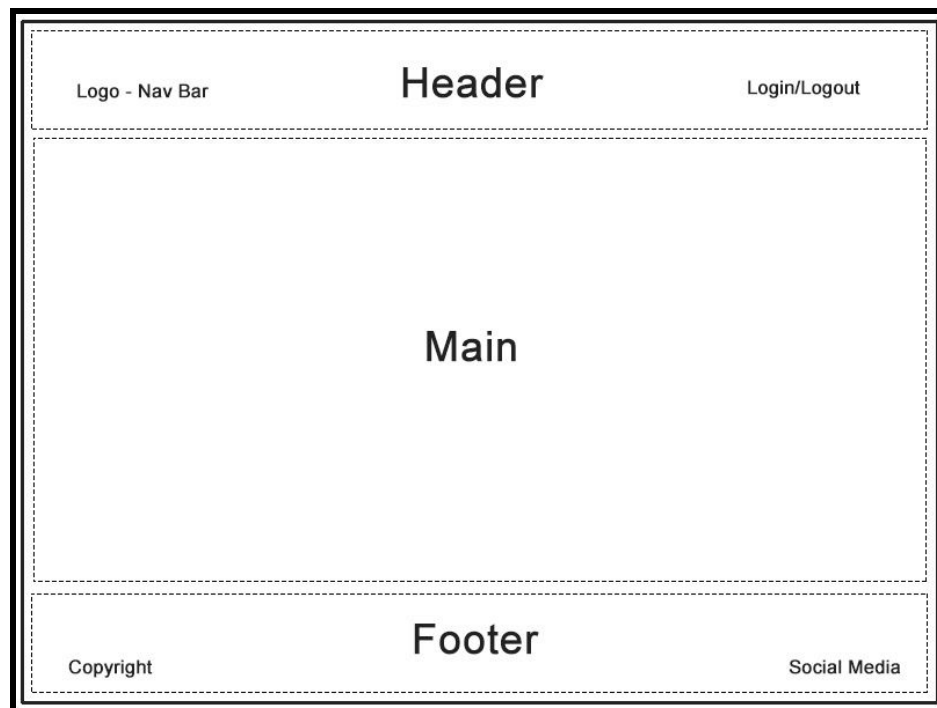


Fig 3.1

Our web page consists of 5 key pages that follow this design: Home, About, Shop, Contact Us and Sign-up (all .php files). Each .php page consists of HTML <link> tags that call on their own unique CSS stylesheets. We have 5 individual stylesheets total. The page wrapper is within the <body> tags with each element (Header, Main, Footer) being ordered to fit the skeleton above utilizing the *flex-direction: column* feature (Fig 3.2). Each page consists of the static header and

static footer with the content between the <main> tags varying. All pages utilize *border-box* *box-sizing* and CSS Flex-Box. This layers the page elements as demonstrated.

### 3.1.2 Header and Nav-Bar

```
/* ENCOMPASSES WHOLE PAGE */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

/* is the full wrapper */
body {
  font-family: "Montserrat", sans-serif;
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
```

Our header is of order: 0 (top) and utilizes the *flex-wrap: wrap* for full responsiveness on smaller windows. This wraps the elements of the navigation bar when needed.

The navigation bar consists of an unordered list with the dot-style removed, all aligned to center. Each item in the list consists of a link to another page on our website. We have also included a login form/button, logout button and a sign-up link. Utilizing PHP and sessions, our logout button is only presented when a user is currently logged in and the login form is only available if

no user is logged in.

Fig 3.2

### 3.1.3 Main Body

The <main> *body* is where the uniqueness of each page is presented. The <main> body also uses flex-box features to ensure responsiveness (Fig 3.3). Our *Home* and *About* pages make use of a background which sits flush beneath the header. Our Shop, Contact and Sign-Up forms utilize the <main> extensively as detailed in section 3.1.5. Within the body, we have some text (heading and paragraph) to fill the void. When the admin is logged in, the *About* page text is modifiable utilizing an input form which posts to the website (see PHP section).

```
/* BODY STARTS HERE */
main {
  display: flex;
  flex-direction: column;
  justify-content: center;
  flex-grow: 1;
  min-height: 350px;
  background: url(cat1080full.jpg) no-repeat
    center 0px;
  width: 100%;
  height: 100%;
  background-blend-mode: luminosity;
  color: white;
}
```

Fig 3.3

### 3.1.4 Footer

The <footer> on our pages sits as the last element in the flex column. The footer consists of an unordered list (similar to the nav-bar) which has images linking social media websites (these are not hard-coded). These social media icons are located in the *images* folder and can be edited as needed. The footer is also responsive and flexes when the browser window size is changed.



### 3.1.5 Shop, Contact Us, Sign-Up and Others

Our *Shop* (*store.php*) page makes use of the `<main>` body extensively. The main is broken into a *products-container* section which consists of a few `<div>` classes that create each *product-card*. All products sit within the *products-container* which houses the image, information and add-to-cart button. This information is pulled from the SQL database using PHP. Should a new item be inserted into the database, the page will update to include it. These items all make use of CSS flex-box (fig 3.4 - 3.5) and are responsive to browser changes (try it!). Fig 3.6 is an example of a product-card with our favourite actor, the *Spooked Cat*.

The *Shop* also consists of a Shopping Cart which utilizes HTML tables. The cart is populated when an item is added to the cart using PHP. Once an item is added, buttons appear to either *Remove* the item or *Checkout* (not hard-coded). This table is also flex compatible and responsive to browser changes.

Fig 3.4 - 3.5 - Store Container

```
.products-container {
  order: 0;
  display: flex;
  flex-wrap: wrap;
}

.product-card {
  display: flex;
  flex-direction: column;
  padding: 1%;
  flex: 1 16%;
  box-shadow: 0px 0px 2px 0px rgba(0,0,0,0.25);
}

.product-info {
  order: 1;
  margin-top: auto;
  text-align: center;
  padding-top: 20px;
  bottom: 0;
  left: 0;
  color: black;
}
```

The *Sign Up* and *Contact Us* pages are very simple in nature. The `<main>` body consists of flexible input forms with CSS styling applied. When an input form is clicked, a focus border appears to point the users attention. This is shown in Fig 3.7 (Sign up form is similar). We have also included an HTML text-editor (*TinyMCE*) for the message input (shown in Fig 3.7). Depending on the page, formatted messages may appear through the use of PHP, AJAX and JavaScript for varying inputs. These messages have very little CSS styling applied. In addition, each of the buttons are formatted to match the look of the webpage.

Some extra CSS features are explained in section 4.1 *Additional Features*.

Fig 3.6 - Product Card

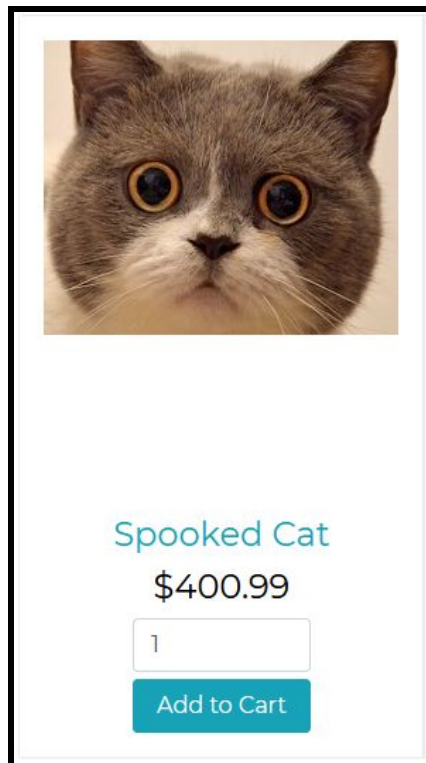


Fig 3.7 - Contact Form

## Contact Us

Name:

Email Address:

Message:

File Edit View Format

↶ ↷ Paragraph ▼ **B** *I* ...

p

POWERED BY TINY

Submit

Fig 3.8 - Flexed Shopping Cart

Cart Details				
Product Name	Quantity	Price	Total	Action
Spooked Cat	1	400.99	400.99	<a href="#">Remove</a>
Total			400.99	
<a href="#">Checkout</a>				

## 3.2 JAVASCRIPT AND AJAX

The Javascript file (*contact.js*) is used for the 'Contact Us' page to compel the user to enter their name, a valid email address and a message. `$('#btn-default').click(function(e)` method triggers the click event when the user clicks the submit button. Inside this method, the input values of the fields name, email, and message are assigned to the variables name, email, and message respectively. If the name variable is an empty string or if the user didn't enter their, an error message is displayed on the screen saying that the user must enter their name (Fig 3.9). Similarly with email and message..

Fig 3.9

```
$(document).ready(function() {
    var delay = 1000;
    $('#btn-default').click(function(e){

        var name = $('#name').val();
        if(name == ''){
            $('#message_box').html(
                '<span style="color:red;">Enter Your Name!</span>'
            );
            $('#name').focus();
            return false;
        }
    })
});
```

After the user has filled the text fields, the `ajax()` method is used to perform an AJAX (asynchronous HTTP) request. `beforeSend` method is used so the loading gif is inserted before an Ajax request is started. Then, file `ajax.php` is called to display the "Thank you" note on the screen(Fig 3.10 - 3.11)

Fig 3.10

```
$.ajax
({
    type: "POST",
    url: "internal/ajax.php",
    data: "name="+name+"&email="+email+"&message="+message,
    beforeSend: function() {
        $('#message_box').html(
            ''
        );
    },
    success: function(data)
    {
        setTimeout(function() {
            $('#message_box').html(data);
        }, delay);
    }
});
```

Fig 3.11

```
if ( ($_POST['name']!="") && ($_POST['email']!="")){
$name = $_POST['name'];
$email = $_POST['email'];
$message = $_POST['message'];

echo "<span style='color:green; font-weight:bold;'>
    Thank you for contacting us, we will get back to you shortly.
</span>";
```

### 3.3 PHP & MySQL

PHP is predominantly used to track user session variables across our website, interface with the database, and dynamically access or display content. PHP is divided into two types: inline PHP , which functions alongside existing html code and standalone PHP which provides backend utility.

Standalone php is demonstrated in the databasehandler.internal.php file. This script is called in several places through our site and provides a connection to the mysql database (or errors out if unable). Factoring this code out allowed us to write DRY, clean code and keep setup to a minimum for new hosts.

Inline php is demonstrated in store.php which will dynamically hide / display content based on the user's session variables (see section 1.1).

The Contact Us page makes use of PHP. Contact.php is used so that the message the user sends is emailed to the admin user(Fig 3.12). The email shows the name of the user, their email, the message and the text "You have received a feedback" as the subject.

Fig 3.12

```
if(isset($_POST['submit'])){
    sleep(4);
    $name = $_POST['name'];
    $subject = "You have received a feedback";
    $mailFrom = $_POST['email'];
    $message = $_POST['message'];
    $mailTo = "dan_san@live.com";
    $headers = "From: ".$mailFrom;
    $txt = "You have received an e-mail from ".$name.".\\n\\n".$message;

    mail($mailTo, $subject, $txt, $headers);
    header("Location: contactus.php?mailsend");
```

PHP is also used on the registration page and for the user authentication portions of the website. For registration, the PHP interfaces with the MySQL database using the standalone PHP file databasehandler.internal.php and signup.internal.php. The latter of those pages uses various if-else statements to check if a user already exists in the database or if the form is filled out incorrectly (Fig. 3.13). The inline php code on signup.php will dynamically serve the user with error or success messages by finding \$\_GET variables in the url bar that are supplied to the browser by the standalone php file signup.internal.php (Fig. 3.14).

Fig 3.13

```
if(empty($username) || empty($password) || empty($passwordVerify))
{
    header("Location: ../signup.php?error=emptyfield&username=".$username);
    exit();
}
else if($password != $passwordVerify)
{
    header("Location: ../signup.php?error=passwordmismatch&username=".$username);
    exit();
}
else
{
    $sql = "SELECT * FROM `users` WHERE `username` LIKE '$username'";
    $result = mysqli_query($connection, $sql); //Result of the above query
    $row = mysqli_num_rows($result); //Number of rows from result, used to check :
    if(!$result)
    {
```

Fig 3.14

```
<?php
if(isset($_GET['error']))
{
    if($_GET['error'] == "emptyfield")
    {
        echo '<p class="signuperror">Please fill in all fields</p>';
    }
    else if($_GET['error'] == "passwordmismatch")
    {
        echo '<p class="signuperror">Your passwords must match</p>';
    }
    else if($_GET['error'] == "userexists")
    {
        echo '<p class="signuperror">That username is taken - please choose a different one</p>';
    }
}
else if($_GET['signup'] == "successful")
{
    echo '<p class="signupsuccess">You have successfully signed up for CatShop!</p>';
}

if(isset($_SESSION['uname']))
{
    echo '<p>Hi ' . $_SESSION['uname'] . ', you already have an account!</p>';
}
```

The login mechanism on each page works in a similar way to the registration page. The login button will call standalone PHP code login.internal.php and this code will connect to the

database using the standalone database handler PHP code and run through various if-else statements to determine if the user entered correct login information. If so, the PHP code will start a user session, if not an appropriate error message will be supplied.

Once the user has logged in, inline PHP code on the nav bar of each page reveals the My Account page which allows the user to create, update, and delete a delivery address where orders will be delivered. Delivery information is stored in the sql database and is tied to a user's individual account and is persistent across user sessions/logins.

The logout functionality is very simple and is made with the standalone PHP code `logout.internal.php` that ends the user session and navigates to the index page (fig. 3.15).

`session_unset()` frees all session variables currently registered and `session_destroy()` destroys all data registered to a session.

*Fig 3.15*



```
1<?php
2    session_start();
3    session_unset();
4    session_destroy();
5    header("Location: ../index.php");
6?>
```

A MySQL database is used to store data on users, products, and modular information the site will render dynamically. Please see section 2.2 Database Setup for more information about the MySQL database.



## 3.4 SECURITY

The file databasehandler.internal.php has been made hidden from users as it contains private database credentials and is only accessed by pages on the backend in order to connect them with the MySQL database. The public will have no need to access this file directly in order for the database functions on the website to function correctly.

Injection.php is used to prevent HTML/SQL attacks. When the user fills any text field, injection.php uses the print\_r() method to print the text and places it between <pre> tags that represent preformatted text which is to be presented exactly as written. Then, foreach (\$\_POST as \$k => \$v) is used to loop over each character and, by using htmlspecialchars(\$v), check if the character is a special character, it converts it to HTML entity (Fig 3.16) ( i.e. & (ampersand) becomes &amp;, ' (single quote) becomes &#039;, < (less than) becomes &lt; (greater than) becomes &gt;).

Fig 3.16

```
<?php
if (isset($_POST['submit']))
{
    echo '<pre>';
    print_r($_POST);
    echo '</pre>';
    foreach ($_POST as $k => $v)
    {
        echo '<p>'.htmlspecialchars($v).'</p>';
    }
    echo '<hr />';
}
?>
```

The real\_escape\_string mysqli object function is used to sanitize user input before applying updates or allowing inserts to the database (Fig 3.17).

Fig 3.17

```
$newText = filter_input(INPUT_POST, 'address');
$newText = $connection->real_escape_string($newText);
$query = 'UPDATE users SET address="' . $newText . '"WHERE users.username = "' . $_SESSION['uname'] . '"';
```

## 4.0 EXTRAS

### 4.1 ADDITIONAL FEATURES

Each section below details the additional features we added into our web page.

#### 4.1.1 CSS @Media Tags

Across our pages, we have included some CSS formatting for specific sized mobile devices. This allows flexibility and responsiveness for mobile browsers. When the browser detects a mobile device of a specific minimum width, the following classes and CSS elements are implemented and/or modified (specific to our Shop product layout). See figures 4.1 - 4.3 below.

*Fig 4.1*

```
@media screen and (min-width: 550px) {  
  .page-footer {  
    flex-direction: row;  
    justify-content: space-between;  
    align-items: center;  
  }  
  .page-footer ul {  
    margin-bottom: 0;  
  }  
}
```

*Fig 4.2*

```
@media ( max-width: 920px ) {  
  .product-card {  
    flex: 1 21%;  
  }  
  
  .products .product-card:first-child,  
  .products .product-card:nth-child(2) {  
    flex: 2 46%;  
  }  
}
```

```
@media ( max-width: 600px ) {  
  .product-card {  
    flex: 1 46%;  
  }  
}  
  
@media ( max-width: 480px ) {  
  h1 {  
    margin-bottom: 20px;  
  }  
}
```

*Fig 4.3*