

Principles of Web Application Architecture

Group Assignment

2017/2018

Members: Valerija Holomjova and Deborah Vella
Course Code: CIS1054-SEM2-A-1718

Table of Contents

Application Overview.....	3
Group Responsibilities	4
Testing and Requirements.....	5
Layout and Design.....	8
Home Page.....	9
About Us Page.....	10
Contact Us Page	11
Products Page (and Search Page)	12
Individual Product Page	13
Shopping Cart Page.....	14
Checkout Page	15
Databases.....	16
Employee Details – members.xml	16
Company Details – company.xml.....	17
Products Details - Products.xml	19
Website Pages – array.php	22
Code.....	23
Navigation Bar - nav.php.....	23
list.php	25
Home Page – index.php	27
Contact Us Page – contact_us.php	32
About Us Page – about_us.php.....	36
Product Page – products.php.....	38
Search Page – search.php	42
Header File – header.php.....	44
Footer File – footer.php	46
Shopping cart file - cart.php.....	47
Checkout Page – checkout.php.....	52

Application Overview

The website was created for a gaming company called SKULL Gaming, through which this company can sell their products such as gaming systems, consoles, peripherals and games. This application is created for the benefit video gaming enthusiasts who are keen on buying the latest gaming technologies out in the market.

The customers can access multiple pages which can be accessed from the navigation bar or other links found throughout the website. The pages are:

- Home page: Welcomes the user with a slideshow, description of company and links for products
- About Us page: gives a more in-depth description of the company and the team
- Contact Us page: a form is used to allow the user to send a message to SKULL Gaming company. At the bottom the company's address, emails and telephone numbers are printed.
- Products page: lists all products, provides links to different categories of products on the left-hand side of screen
- Product page: shows the details of the chosen product from the Products page. Displays a go back button, add to cart button and a field to specify the quantity in. this page is used also when customer searches for a specific item.
- Shopping Cart page: lists all products to be purchased together with their details. Allows the user to change quantity if need be. Buttons are provided to remove all items, remove one item and to checkout.
- Checkout page: lists all products to be purchased together with their details and total and overall prices. Text fields are outputted so that user enters his required personal data. Confirm order button and go back button are provided.

SKULL Gaming has access to the databases in which products' details, employees' details and company details are stored. As a result, SKULL Gaming can alter, add and deleted their information without the need of any knowledge about code. The website generates information dynamically and therefore, any changes made in the databases from the company side will result in a change of information on the client-side of the website. Also, with every purchase made, the company is notified with an email.

The 'DRY' principle has been implemented in the website by including the header file, footer file, navigation bar file and more instead of re-copying the code. These types of files can be found in the 'Includes' folder.

Group Responsibilities

Initially, the design and implementation of the website was split among both team members. However, as the website progressed into its dynamic design (by using PHP), both team members worked on the pages together. Any problems encountered in the pages were discussed and solved by both members through constant communication. Hence, both members contributed to each of the files in the website. Team members would work together at university during break hours or at home through online calls. Whenever issues, that could not be solved by the team members alone, came up, we made sure to attend the weekly Tuesday tutorial.

Testing and Requirements

The website was made using HTML, PHP, CSS and Javascript. MAMP, XAMPP and PhpStorm were used to host local servers for our webpage to run on, in order to parse the '.php' files. It has been tested on several browsers namely 'Google Chrome', 'Firefox', 'Safari' and 'Internet Explorer'. We have also configured XAMPP to be able to send emails to the company Gmail account. Please refer to the screenshots below illustrating the contact form and checkout form being filled out, followed by the mail received in the company Gmail account after the forms are submitted.

SKULL

Home About Us Products Contact Us Search...

Contact Us

Full Name:

E-mail:

Subject:

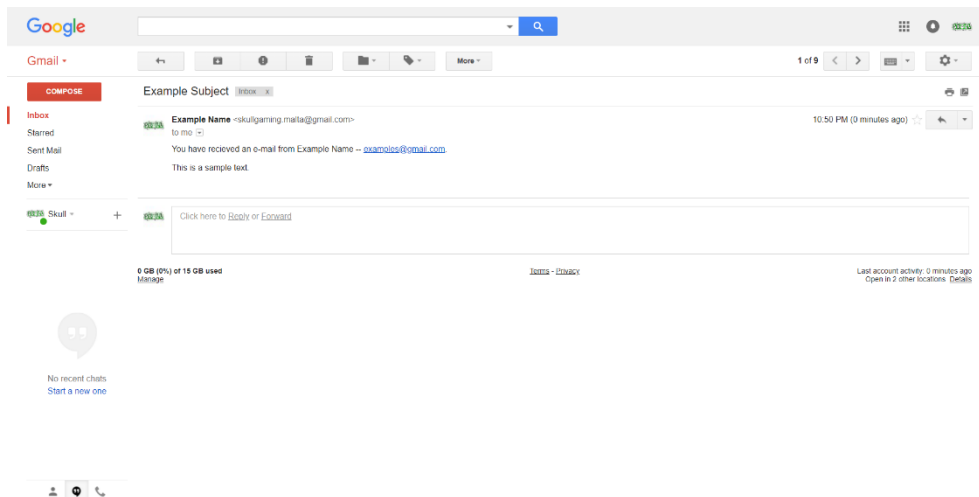
Contact Information

Address	E-mail	Phone
Address ICT Faculty, University of Malta Univeristy Ring Rd Msida MSD 2080 Malta	Owners valerija.holomjova.17@um.edu.mt deborah.velia.17@um.edu.mt	General Queries +356 2340 2340 +356 2340 4100
Opening Hours Mondays - 09:00 - 17:00 Tuesdays - 09:00 - 17:00 Wednesdays - 09:00 - 17:00 Thursdays - 09:00 - 17:00 Fridays - 09:00 - 17:00 Saturdays & Sundays - Closed	Company skullgaming.malta@gmail.com	Main Office +356 2340 2530

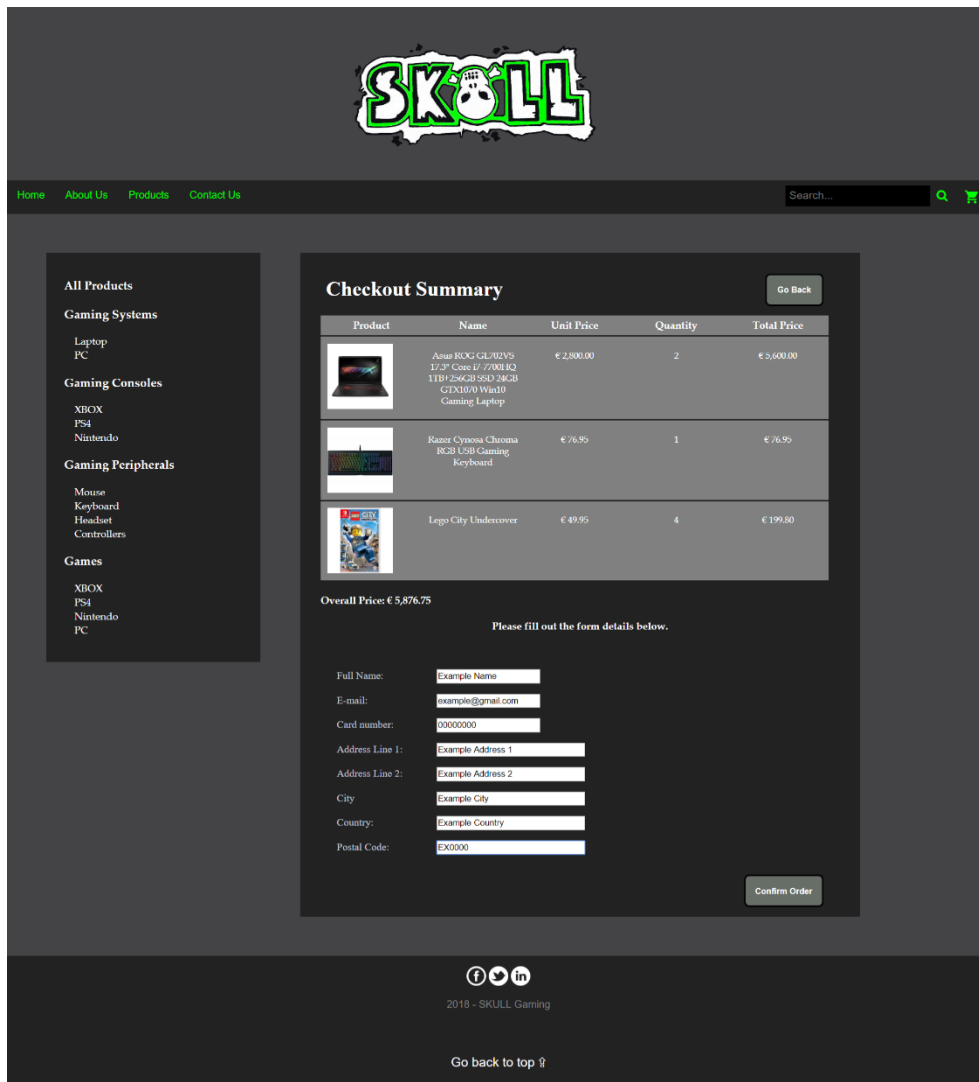
2018 - SKULL Gaming

[Go back to top](#)

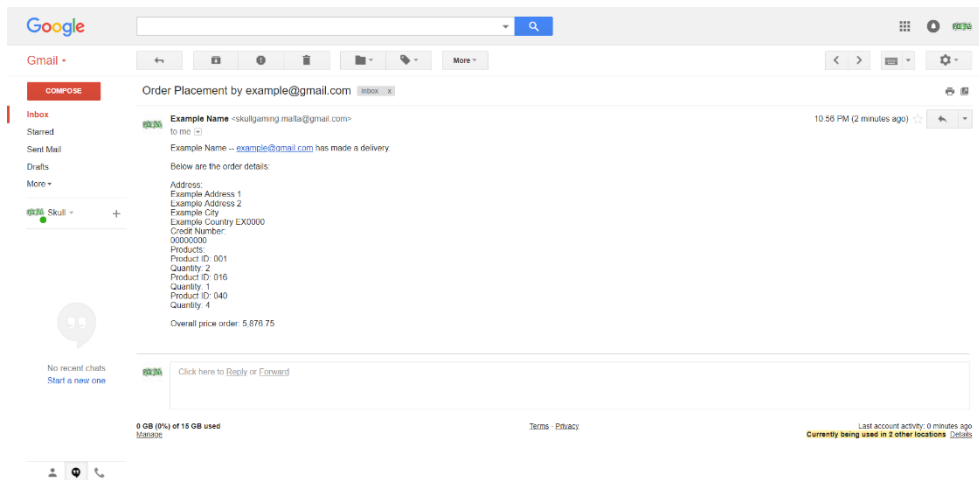
(Screenshot illustrating the contact form before it has been submitted.)



(Screenshot illustrating the mail received in the company Gmail account after the contact form is submitted.)



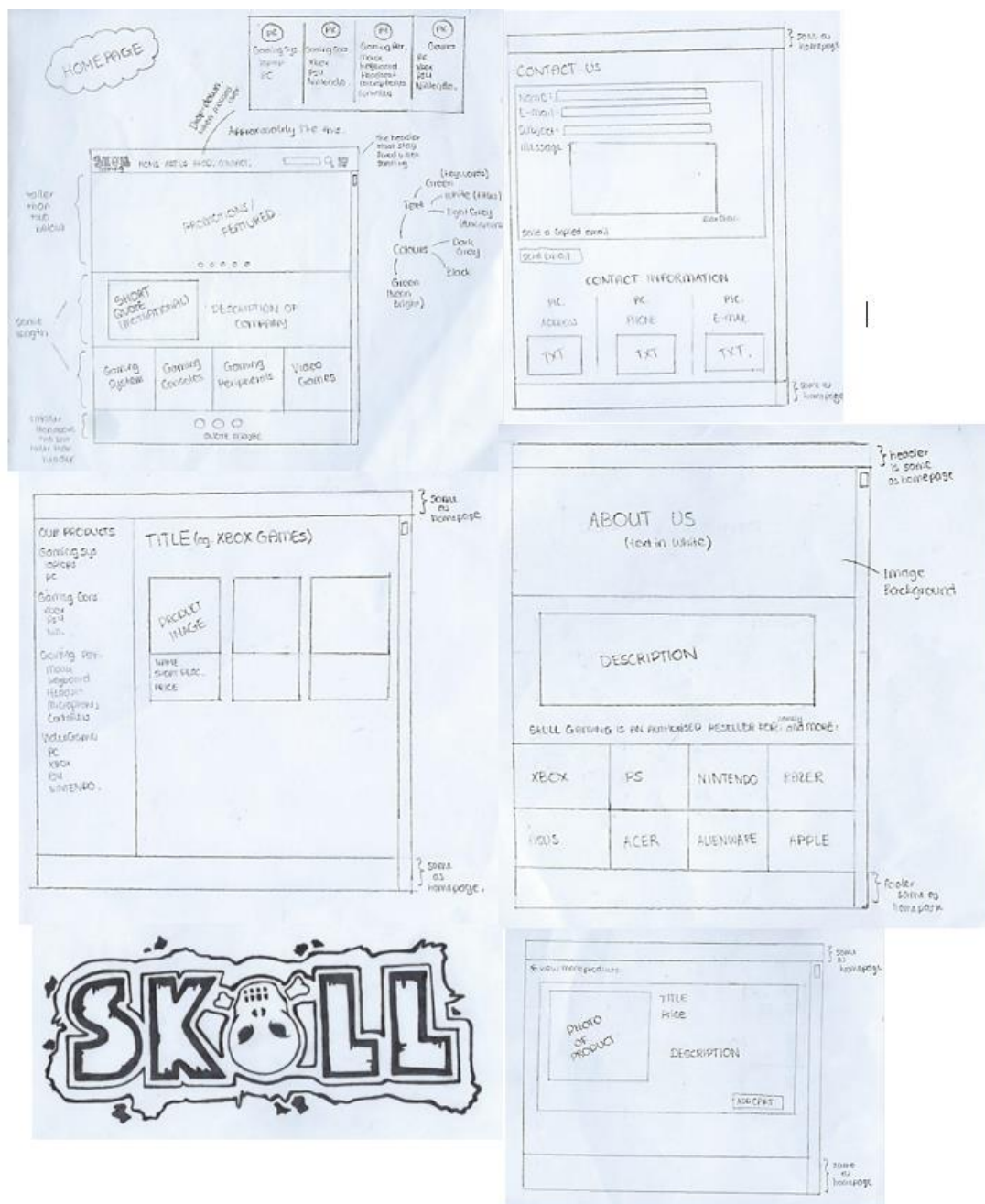
(Screenshot illustrating the checkout form before it has been submitted.)



(Screenshot illustrating the mail received in the company Gmail account after the checkout form is submitted.)

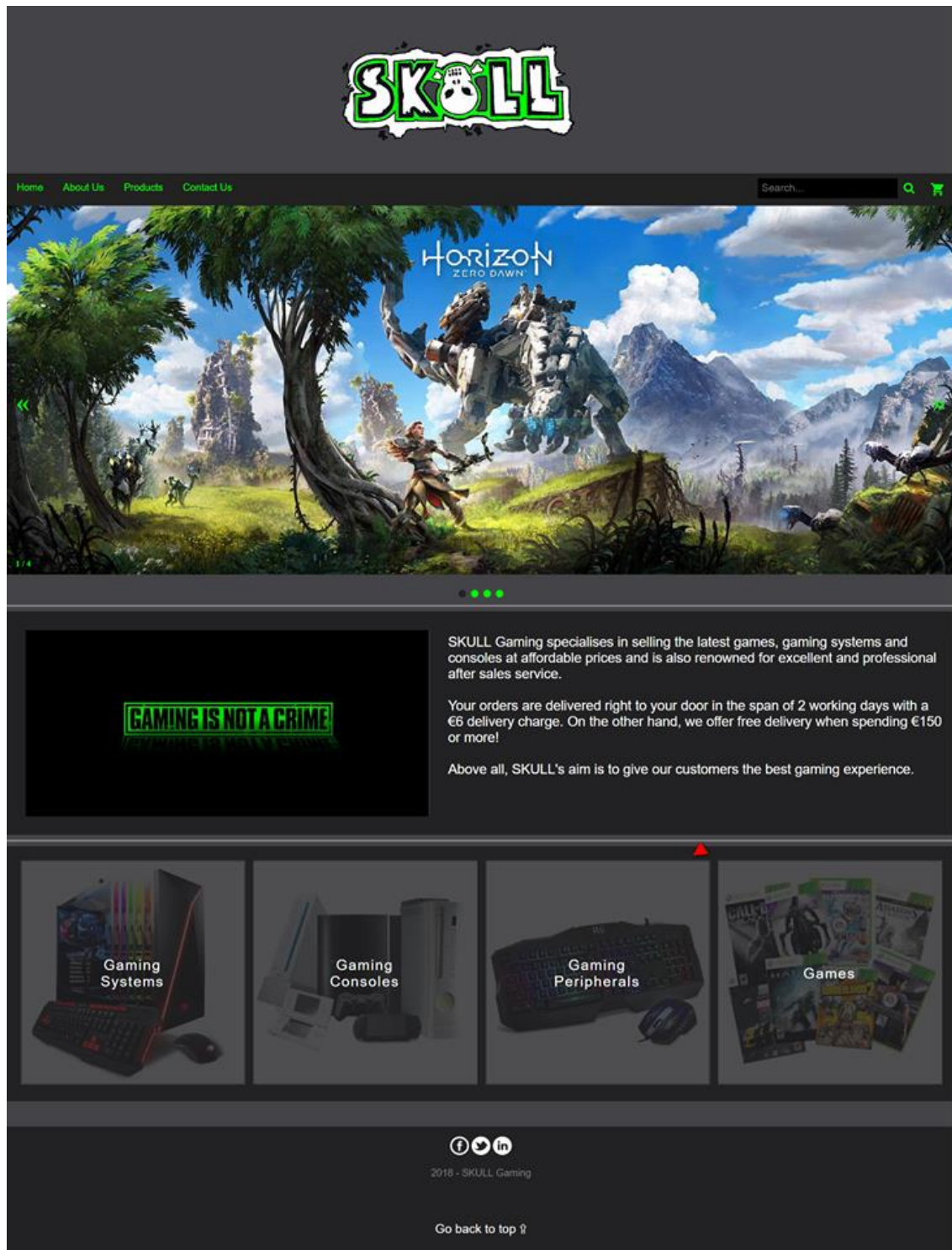
Layout and Design

For this assignment, both team members aimed to create a gaming retailer website that sells video games, gaming peripherals, consoles, laptops and other gaming systems. The design of the website has been discussed and planned together based on other gaming retail websites. The team members aimed to create a dark, modern and trendy design and colour scheme that would appeal towards gamers. Each page was sketched and planned beforehand as well as the company logo. Please refer to the images below for an outline of how the pages were planned. In the remainder of this section, the layout and purpose of each page will be displayed.



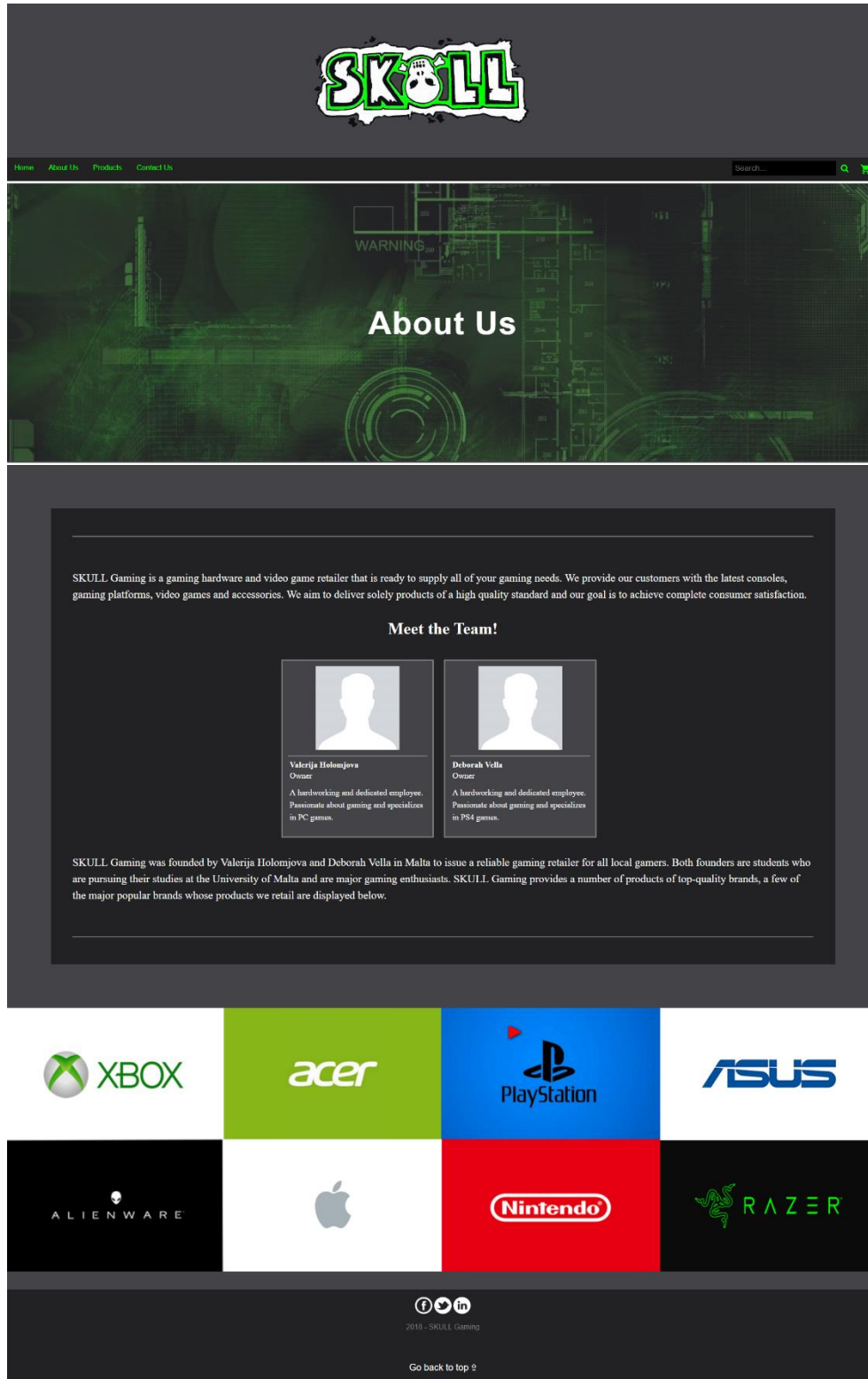
Home Page

This page welcomes the user into our website. It consists of a slideshow of pictures of videogames, a description of the company and links to the products based on their categories.




About Us Page

This page was created to display the aims and goals of the company as well as a description of the employees in the company. It also displays the companies whose products we used in our database.



Contact Us Page

This page was created so that the user could send an e-mail to the company or view details such as the address, phone numbers, and opening hours.



[Home](#) [About Us](#) [Products](#) [Contact Us](#)

Contact Us


Full Name:

E-mail:

Subject:

Please enter the message here...

Contact Information




Address

ICT Faculty, University of Malta
Univeristy Ring Rd
Msida MSD 2080
Malta

Opening Hours


Mondays - 09:00 - 17:00
Tuesdays - 09:00 - 17:00
Wednesdays - 09:00 - 17:00
Thursdays - 09:00 - 17:00
Fridays - 09:00 - 17:00
Saturdays & Sundays - Closed



E-mail

Owners
valerija.holomjova.17@um.edu.mt
deborah.velia.17@um.edu.mt




Company
skullgaming.malta@gmail.com



Phone

General Queries
+356 2340 2340
+356 2340 4100

Main Office
+356 2340 2530



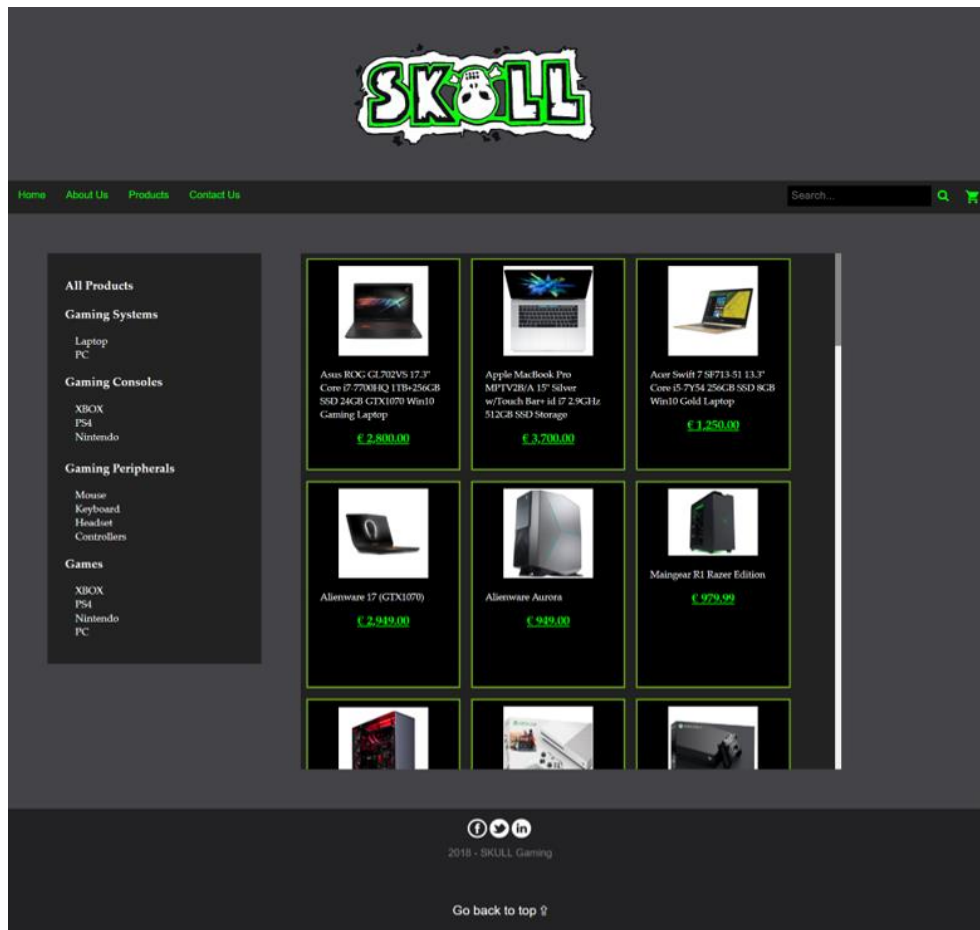
2018 - SKULL Gaming

[Go back to top](#)

11

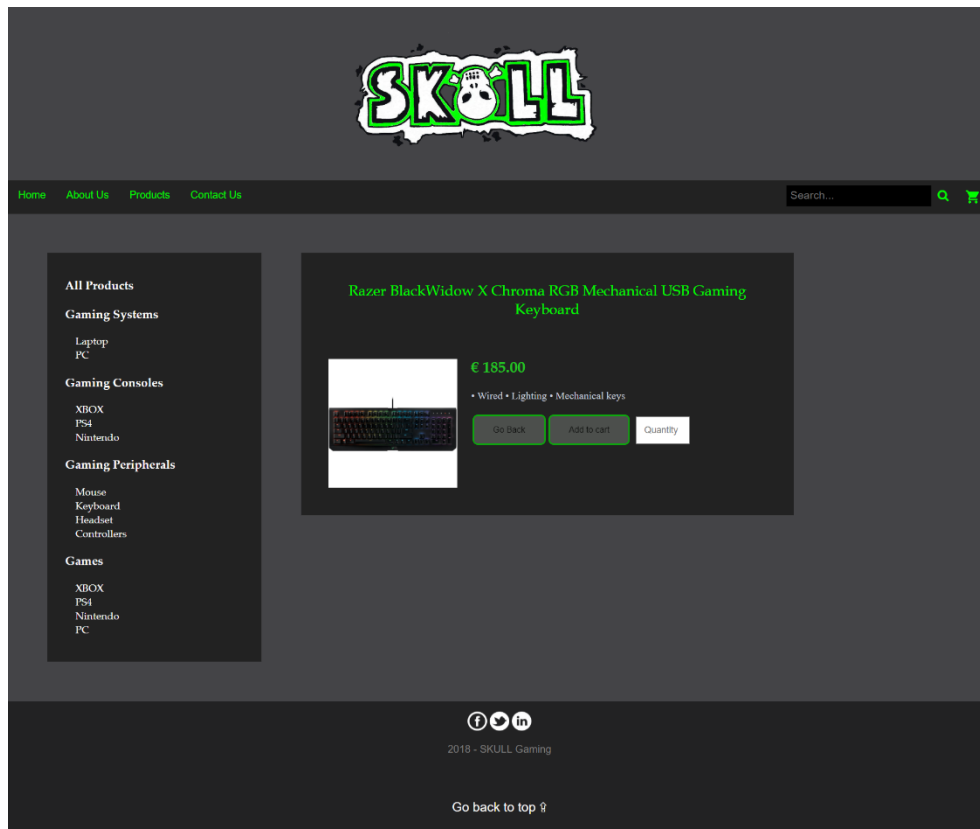
Products Page (and Search Page)

This page was created to display products of particular categories or subcategories. The layout of the search page is exactly the same. The search page displays products found based on the words inputted in the search field.



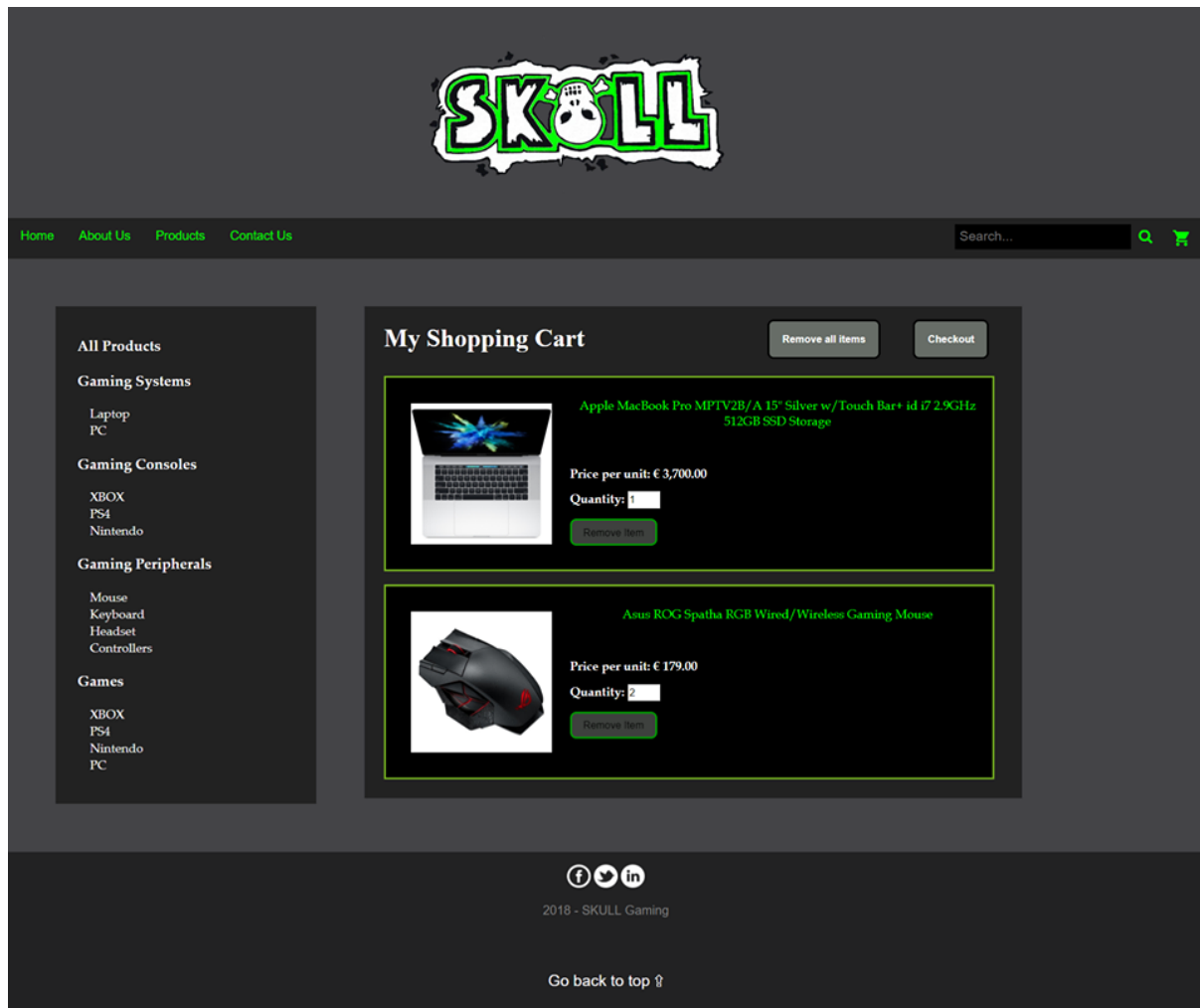
Individual Product Page

This page was created to display the details of individual products. It allows the user to enter a quantity of the product of their choice to add the product as well as its inputted quantity into their shopping cart.




Shopping Cart Page

The shopping cart page displays all the products added to cart by the user. It allows the user to remove specific items, retype the quantity, remove all items and to proceed to check out.





Checkout Page

The checkout page allows the user to see what s/he is about to buy and the total prices and the overall price. The user is required to enter his details before confirming the order. He also has the facility of going back to the shopping cart page instead of confirming order.



[Home](#) [About Us](#) [Products](#) [Contact Us](#)

Search...  

All Products

Gaming Systems

Laptop

PC

Gaming Consoles

XBOX

PS4

Nintendo

Gaming Peripherals

Mouse

Keyboard

Headset

Controllers

Games

XBOX



PS4

Nintendo

PC

Checkout Summary

Go Back

Product	Name	Unit Price	Quantity	Total Price
	Apple MacBook Pro MP1V2B/A 15" Silver w/ Touch Bar+ i4 i7 2.9GHz 512GB SSD Storage	€ 3,700.00	1	€ 3,700.00
	Anus ROG Spatha RGB Wired/Wireless Gaming Mouse	€ 179.00	2	€ 358.00

Overall Price: € 4,058.00

Please fill out the form details below.

Full Name:

E-mail:

Card number:

Address Line 1:




Address Line 2:

City:

Country:

Postal Code:

Confirm Order

2018 - SKULL Gaming

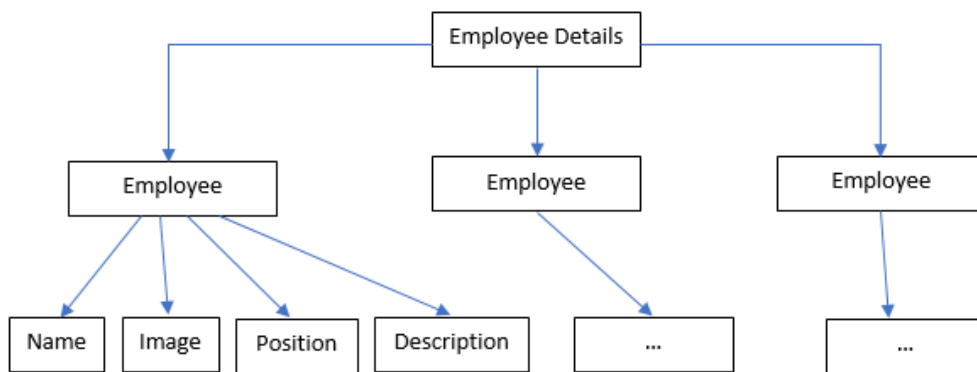
[Go back to top ↑](#)

Databases

This section will discuss the databases and data structures that were used to hold in order to build a dynamic website. This will allow certain aspects of the website such as the employee information or the products information to be modified without having to change the website page directly. All of the files described in the next section can be found in the 'Includes' folder.

Employee Details – members.xml

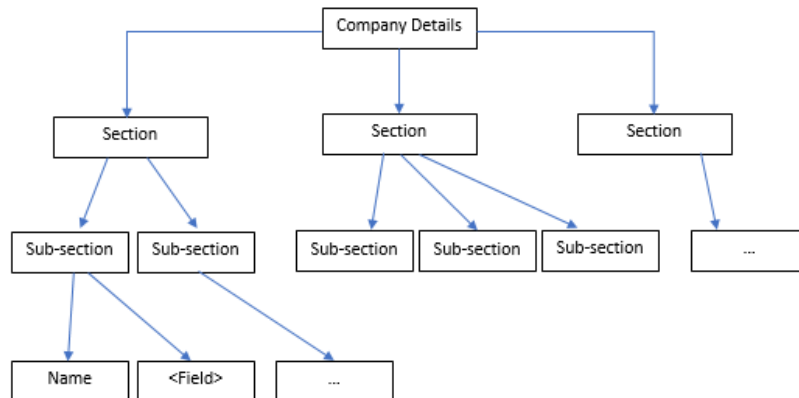
The database 'members.xml' holds the employee information such as their names, positions, image and descriptions. This is used in the "About Us" page. This database allows employees to be added or deleted without having to re-arrange the entire page format. The structure of the database can be displayed as the following:



```
<employees>
  <employee>
    <name>Valerija Holomjova</name>
    <img>Pictures/About_Us/default.jpeg</img>
    <position>Owner</position>
    <description>A hardworking and dedicated employee. Passionate about gaming
and specializes in PC games.</description>
  </employee>
  <employee>
    <name>Deborah Vella</name>
    <img>Pictures/About_Us/default.jpeg</img>
    <position>Owner</position>
    <description>A hardworking and dedicated employee. Passionate about gaming
and specializes in PS4 games.</description>
  </employee>
</employees>
```


Company Details – company.xml

The database ‘company.xml’ holds the company information, such as the opening hours, location, phone numbers and more. This is used in the “Contact Us” page. The structure of the database can be displayed as the following:



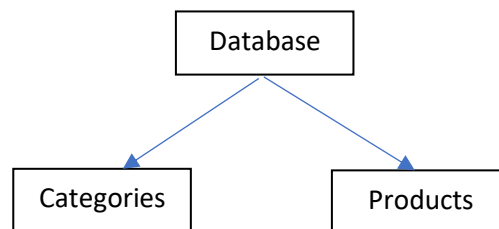
In our database, there are three sections: ‘phones’, ‘address’ and ‘e-mails’. Each section has a sub-section. For the ‘address’ section there is a ‘location’ and ‘hours’ sub-section. Then each sub-section can have a name attribute and an attribute where the content is displayed. For instance, for the ‘phones’ section, there is a ‘phone’ sub-section. One of the ‘phone’ sub-sections has a name called “General Queries” and a ‘number’ elements containing phone numbers corresponding to “General Queries”.

```
<company>
  <address>
    <location><![CDATA[
      ICT Faculty, University of Malta <br />
      Univeristy Ring Rd <br />
      Msida MSD 2080 <br />
      Malta]]>
    </location>
    <hours><![CDATA[
      Mondays - 09:00 - 17:00 <br />
      Tuesdays - 09:00 - 17:00 <br />
      Wednesdays - 09:00 - 17:00 <br />
      Thursdays - 09:00 - 17:00 <br />
      Fridays - 09:00 - 17:00 <br />
      Saturdays & Sundays - Closed ]]>
    </hours>
  </address>
  <emails>
    <email>
      <name>Owners</name>
      <contact><![CDATA[
        valerija.holomjova.17@um.edu.mt <br />
        deborah.vella.17@um.edu.mt]]>
    </contact>
    </email>
    <email>
      <name>Company</name>
      <contact>skullgaming.malta@gmail.com</contact>
    </email>
  </emails>
  <phones>
    <phone>
      <name>General Queries</name>
      <number><![CDATA[
```

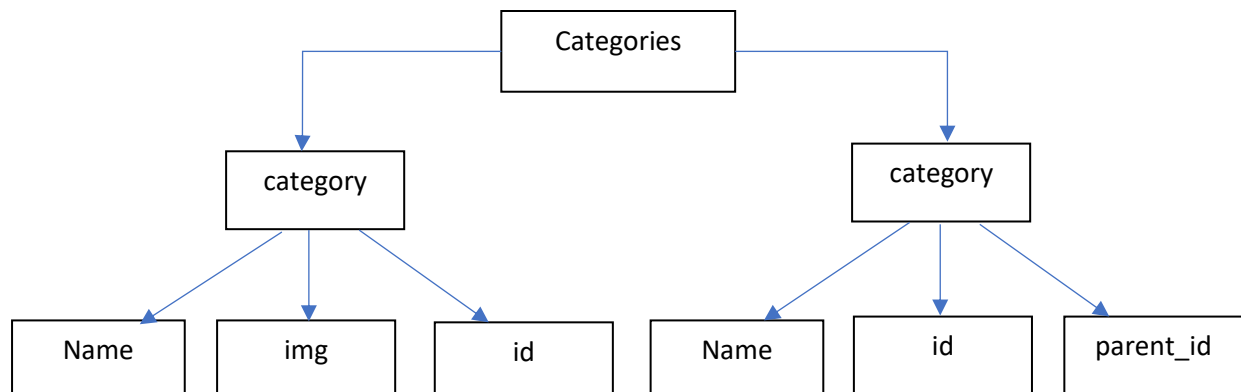
```
        +356 2340 2340 <br />
        +356 2340 4100]]>
    </number>
</phone>
<phone>
    <name>Main Office</name>
    <number>+356 2340 2530</number>
</phone>
</phones>
</company>
```

Products Details - Products.xml

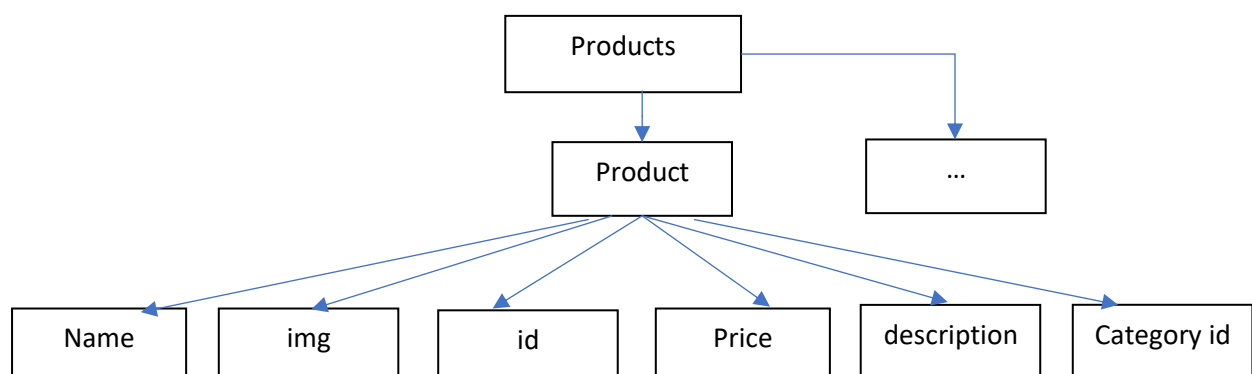
This xml file stores all information about the products and categories. The categories part of the xml contains all main categories and their sub categories. The database is split in two parts as shown below.



Each part is split into more attributes and sub categories. The structure of the categories part is shown in the diagram below.



The structure of the products part is shown in the diagram below.



The parent categories have the following attributes:

- id (stores category unique id)
- img (stores image)
- name (stores category name)

```
<categories>
  <!--Giving each category and id. the below have no parents-->
  <category>
    <id>01</id>
    <img>Pictures/Index/Categories/systems.jpg</img>
    <name>Gaming Systems</name>
  </category>

  <category>
    <id>02</id>
    <img>Pictures/Index/Categories/consoles2.jpg</img>
    <name>Gaming Consoles</name>
  </category>
```

The sub categories have the following attributes:

- id (category id)
- name(Subcategory name)
- parent_id

The parent_id automatically implies that the category is a child of another category. This id has to match the category id of its parent. In these xml snippet one can notice that Gaming systems and Gaming consoles are parent categories while Laptop and XBOX are their children respectively.

```
<!--The below are sub-categories of the above categories, and store their
corresponding parent id-->
<category>
  <id>05</id>
  <name>Laptop</name>
  <parent_id>01</parent_id>
</category>

<category>
  <id>07</id>
  <name>XBOX</name>
  <parent_id>02</parent_id>
</category>
```

After all categories, the products are stored in this xml file. Each product consists of the following attributes:

- name (product name)
- id (product id)
- img (image location)
- price
- description
- category_id (indicates which category it makes part of)

```
<products>

  <product>
    <name>Asus ROG GL702VS 17.3" Core i7-7700HQ 1TB+256GB SSD 24GB
    GTX1070 Win10 Gaming Laptop</name>
    <id>001</id>
    <img>Pictures/Product_Pics/Gaming_Systems/Laptops/asus.jpg</img>
    <price>2800.00</price>
    <description><![CDATA[• Intel Core i7-7700HQ 2.8 GHz<br />
    • 24GB DDR4<br />
    • 1TB+256GB<br />
    • 8 GB GeForce GTX 1070.]]>
    </description>
    <category_id>05</category_id>
  </product>
```

Website Pages – array.php

This file 'array.php' is an array for the pages in the website. Each element of the array is considered as one of the pages and structure-wise is another array which holds the name and link of the page. This file is used in displaying the links to the different pages in the navigation bar. This file can be modified if more pages need to be added or removed in the website.

```
<?php
//An array of the pages in the website.
$nav = array(
    array(
        name => "Home",
        link => "index.php"
    ),
    array(
        name => "About Us",
        link => "about_us.php"
    ),
    array(
        name => "Products",
        link => "products.php",
    ),
    array(
        name => "Contact Us",
        link => "contact_us.php"
    )
);
?>
```

Code

This section of the documentation will describe how our website functions by explaining the code that was used to construct the web pages. All of the files described in the next section excluding the header and footer files can be found in the main folder of the website. The header and footer files can be found in the 'Includes' folder.

Navigation Bar - nav.php

This page consists of the outputting of the navigation bar and linking its contents to their corresponding pages. To do this dynamically, the program makes use of an xml file called products.xml which stores all the information about the products and the categories each one falls under. The styling of the navigation bar is found in nav.css and is included at the beginning.

The headings and their links are stored in an array inside array.php. The program fetches the data from the array by looping through it, and checks if the link matches with products.php. If it does not match, the name is printed on screen and is linked to its respective page on click. On the other hand, if the data does match with products.php, the linking name(Products) together with a drop-down consisting of the categories are displayed.

To display the categories, for each fetched category, the program checks if it has a parent id or not. If it does not have, it shows that the category is the parent itself and is outputted. It then proceeds to print the parent's sub categories. For each subcategory, its parent id is compared with the parent category's id. If it matches it means that the subcategory belongs to the parent category printed earlier, therefore, the subcategory is displayed in the same column.

```
<?php include ('nav.css');?>

<div id="nav">
<ul>
    <!--Inputs every page as a link in the header-->
    <?php
        foreach($nav as $item){ //elements fetched from array.php
            //If the link is the product page -- creates dynamic drop down bar elements
            based on products in XML document.
            if($item[link] == "products.php"){

                echo "<li class=\"dropdown\"><a href=\"\"$item[link]\" \">$item[name]</a>";
                echo "<div class=\"dropdown-content\">";
                echo "<div class=\"row\">";

                foreach ($categories->category as $category){
                    if(!$category->parent_id){ //if the category does not have a parent
                    id, it means that it is the Parent.
                        echo "<div class=\"column\">";
                        echo "<a href=\"products.php?cat=$category->id\"><h3>$category-
>name</h3></a>"; //display the category name

                            foreach ($categories->category as $subcategory) {
                                if(strcmp(($subcategory->parent_id), $category->id) == 0) {
                                    //if the id in the sub category pointing to its parent, matches the id of the
                                    category display the subcategory's name
                                    echo "<a href=\"products.php?subcat=$subcategory-
>id&cat=$category->id\">$subcategory->name</a>";
                                }
                            } echo "</div>";
                        }
                    }
                }
            }
        }
    }
}
```

```

    }
    echo "</div>";
    echo "</div>";
    echo "</li>";
  } else {
    //Else displays each main web page in navigation bar from the
arrays.php file.
    echo "<li><a href=\"\$item[link]\">\$item[name]</a></li>";
  }
}

```

On the left hand-side the search bar is displayed accompanied with a search icon (using Font Awesome Web Application Icons) that directs you to search.php on click. Next to this, the cart icon is displayed (using Google Icons) that directs you to cart.php.

```

//Cart icon links to cart.php
echo "<div style=\"float:right\"><a href=\"cart.php\"><i class=\"material-
icons\" >&#xe8cc;</i></a></div>";

?>

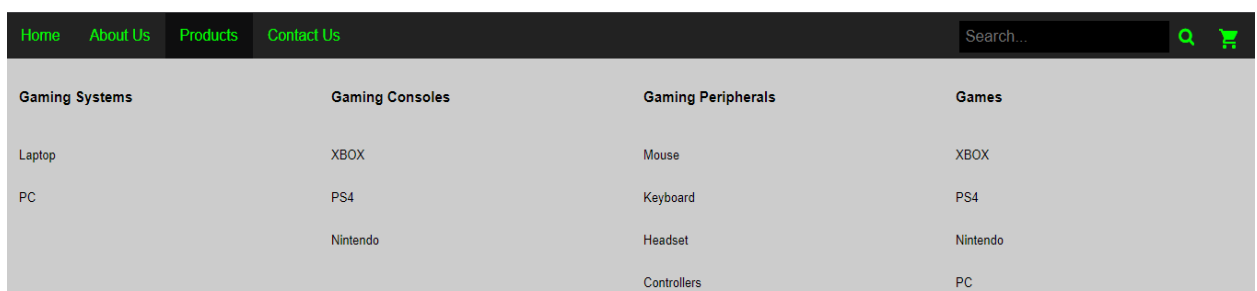
<!--Search Container links to search.php-->
<div class="search-container">
  <form method="post" action="search.php">
    <input type="text" placeholder="Search..." name="search" required>
    <button type="submit" name="search_submit"><i class="fa fa-
search"></i></button>
  </form>
</div>
</div>
</ul>
</div>

```

Please refer to the screenshots below for an example of how the navigation bar looks like in all possible cases.



(Navigation Bar having the About Us being hovered over.)



(Navigation Bar having the Products being hovered over. The drop down is displayed)



(The search bar is being used.)

list.php

This php file is used in the products page and for searching products. It loops throughout the categories inside the xml file, and on every iteration, it checks if the category has a parent id. When it results in not having a parent id it implies that that category is the parent itself. A link to products.php is used passing the category id in it.

```
<div id="list">

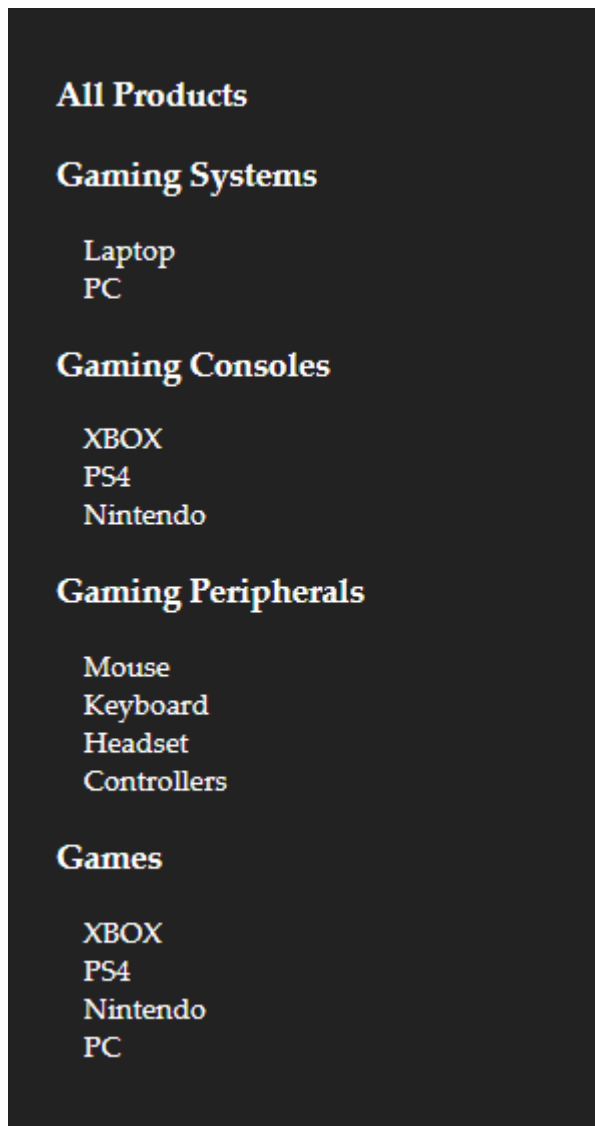
    <div class="list_items">
        <?php

            echo "<a href=\"products.php\" class=\"all\"><h3>All Products</h3></a>";
            foreach ($categories->category as $category) {
                //if category has no parent id - it is primary category
                if(!$category->parent_id){
                    echo "<a href=\"products.php?cat=$category->id\"
class=\"cat\"><h3>$category->name</h3></a>";
```

It continues by displaying the subcategories that are children of the parent category. For each subcategory, its parent id is compared with the parent category's id. If it matches it means that the subcategory belongs to that parent category printed earlier, therefore, the subcategory is displayed below it. This time the link links to the products.php and passes the subcategory id.

```
        //for that primary category - display all its subcategories
        foreach ($categories->category as $subcategory){
            if(strcmp($subcategory->parent_id, $category->id) == 0){
                echo "<a href=\"products.php?subcat=$subcategory->id&cat=$category-
>id\" class=\"subcat\">$subcategory->name</a><br>";
            }
        }
    }
}
?>
<br></div>
</div>
```

The following screenshot shows how the list looks like inside the products page.



Home Page – index.php

The header is included to import all css and functions needed. The code starts by creating a slideshow container in which pictures promoting games are displayed one after the other. For each image the path for access is defined and the text to be displayed on it is specified as well under the class numbers. Then the previous and next buttons are created. When these buttons are clicked the function `changeSlides()` is called passing 1 or -1 depending on what button is clicked.

```
<?php
define("TITLE", "Home | SKULL Gaming");
include('Includes/header.php');
?>

<div id="index">
    <div id="slideshow">
        <!--Creating a slideshow-->
        <!-- Slideshow container -->
        <div class="slideshow-container">
            <!-- Full-width images with number and caption text -->
            <div class="slides fade">
                <div class="numbers">1 / 4</div>
                
            </div>
            <div class="slides fade">
                <div class="numbers">2 / 4</div>
                
            </div>
            <div class="slides fade">
                <div class="numbers">3 / 4</div>
                
            </div>

            <div class="slides fade">
                <div class="numbers">4 / 4</div>
                
            </div>

            <!-- Next and previous buttons -->
            <a class="previous" onclick="changeSlides(-1)">&#171;</a> <!--passes -1 to
the function changeSlides-->
            <a class="next" onclick="changeSlides(1)">&#187;</a> <!--passes 1 to the
function changeSlides-->
        </div>
    <br>
</div>
```

Then, the four indicators, one for each picture are created and aligned in the centre of the screen. When these circles are clicked the function `thisSlide()` is called passing a number from 1 till 4 as parameter. The number passed depends on which of the four buttons are clicked.

```
<!-- The circles at the bottom of the slideshow -->
<div style="text-align:center">
    <span class="dot" onclick="thisSlide(1)"></span>
    <span class="dot" onclick="thisSlide(2)"></span>
    <span class="dot" onclick="thisSlide(3)"></span>
    <span class="dot" onclick="thisSlide(4)"></span>
</div>
```

Java script is used to control the slideshow and make it work. The script starts by setting slide index equal to 1 and calling the function `display_Slides_2` and passing `slideCounter` as parameter. The function `changeSlides` is then coded. This function takes in a 1 or -1 passed when one of the next

and previous arrows is clicked. The function `display_Slides_2()` is called passing the value of `slideCounter` when added to `num`.

```
<script>
    var slideCounter = 1;
    display_Slides_2(slideCounter);

    // Next/previous controls
    function changeSlides(num) {
        display_Slides_2(slideCounter += num);
    }

```

The function `thisSlide()` is later created and this time it expects a number from 1 to 4 according which indicator is selected. This method also calls the function `display_Slides_2()` but it now passes `slideCounter` which is equal to the number taken in as a parameter.

```
// Thumbnail image controls
function thisSlide(num) {
    display_Slides_2(slideCounter = num);
}

```

The function `display_Slides_2()` is coded taking in a number as parameter. It gets all elements that have a class of "slides" which are the pictures and their text. Afterwards it gets all elements that have a class dots which are the dots at the bottom of the slideshow.

```
//Changes slides when clicking the next and previous arrows
function display_Slides_2(num) {
    var i;
    var pic = document.getElementsByClassName("slides"); //get all elements that
    have a class of slides
    var circles = document.getElementsByClassName("dot");
}

```

If the number received from the parameter is greater than the length of the string `pic`, the `slideCounter` is set to 1. Then another if statement checks if `num` is less than 1. In this case the `slideCounter` is set to the length of the string `pic`. Then code proceeds with two `for()` loops. One of them loops through the length of slides and hiding the element at index `i` but keeps the same dimensions and position. The second `for()` loop iterates through the length of circles and activates the dots at index `i`. At last, the function displays the slides and activates the dots.

```
    if (num > pic.length)
    { slideCounter = 1 }
    //if the slide number is greater than the amount of slides, set slideCounter
    equal to 1 to start from 1st picture again.

    if (num < 1)
    { slideCounter = pic.length }
    //if slide number is less than 1 set slide index equal to the length of the
    string pic

    for (i = 0; i < pic.length; i++) { //loops through pictures
        pic[i].style.display = "none"; //hides entire element keeping same
        dimensions and position
    }

    for (i = 0; i < circles.length; i++) { //loops through dots
        circles[i].className = circles[i].className.replace(" active", "");
    }
    //Activating the dots

    pic[slideCounter - 1].style.display = "block"; //displaying the picture
    circles[slideCounter - 1].className += " active"; //activate dots
}

```

Another function called `display_Slides_1()` is coded so that the slides can change automatically on their own without any user interaction. In fact, this function takes no parameters. Before this method is accessed the `slideCounter` is set to 0 to begin from the start of the slides. This method contains all of the code explained above but this time using `slideCounter` only and not a variable passed like `num`. For the slides to change on their own, at the end the function `setTimeout()` is used in which the function `display_Slides_1()` is called again and setting the time for 3 seconds.

```
/*Slideshow changing slides by itself*/
var slideCounter = 0;
display_Slides_1();

function display_Slides_1() {
    var i;
    var pic = document.getElementsByClassName("slides");
    var circles = document.getElementsByClassName("dot");

    for (i = 0; i < pic.length; i++) {
        pic[i].style.display = "none";
    }

    for (i = 0; i < circles.length; i++) {
        circles[i].className = circles[i].className.replace(" active", "");
    }

    slideCounter++;
    if (slideCounter > pic.length) { slideCounter = 1 }
    circles[slideCounter - 1].className += " active";
    pic[slideCounter - 1].style.display = "block";
    setTimeout(display_Slides_1, 3000); // Change image every 3 seconds
}
</script>
</div>
<hr />
```

After the slideshow scripts are finished, a description of our company is typed inside a paragraph in html. A picture is outputted right next to it.

```
<!--Image and description next to each other-->
<div class="description">
    
    <p>
        SKULL Gaming specialises in selling the latest games, gaming systems and
        consoles at affordable prices and
        is also renowned for excellent and professional after sales service.<br />
    <br />
        Your orders are delivered right to your door in the span of 2 working days
        with a
        &euro;6 delivery charge. On the other hand, we offer free delivery when
        spending &euro;150 or more! <br /><br />
        Above all, SKULL's aim is to give our customers the best gaming experience.
    </p>
</div>
<hr />
```

A row of picture links generated dynamically is displayed at the bottom of the page. These pictures are retrieved from the `products.xml` file. Therefore, for each category that does not have a parent id, the image is outputted and the category name on top of it. Last but not least the footer is included.

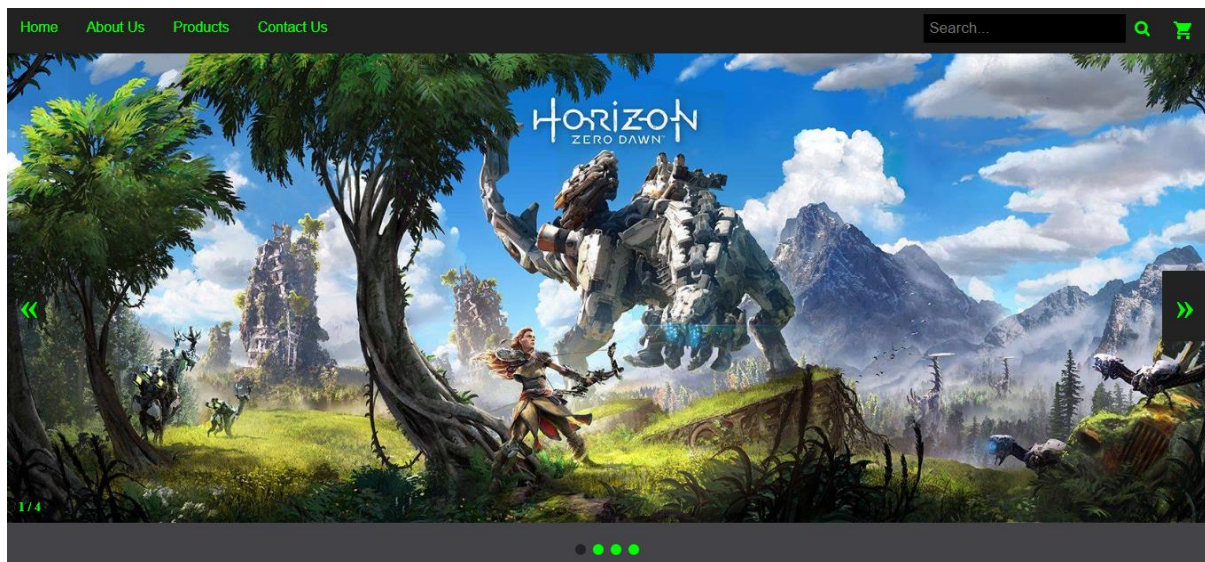
```

    <!--Displays the categories as picture links that direct the user to the
products that fall under the clicked category-->
    <div class="rowpic">
        <?php foreach ($categories->category as $category) {
            if (!$category->parent_id) {
                echo "<div class=\"row1\">
                    <a href='products.php?cat=$category->id'><img
src='$category->img'></a>
                    <div class=\"text\">$category->name</div>
                    </div>";
            }
        }
    </div>
</div>

<?php
include('Includes/footer.php');
?>

```

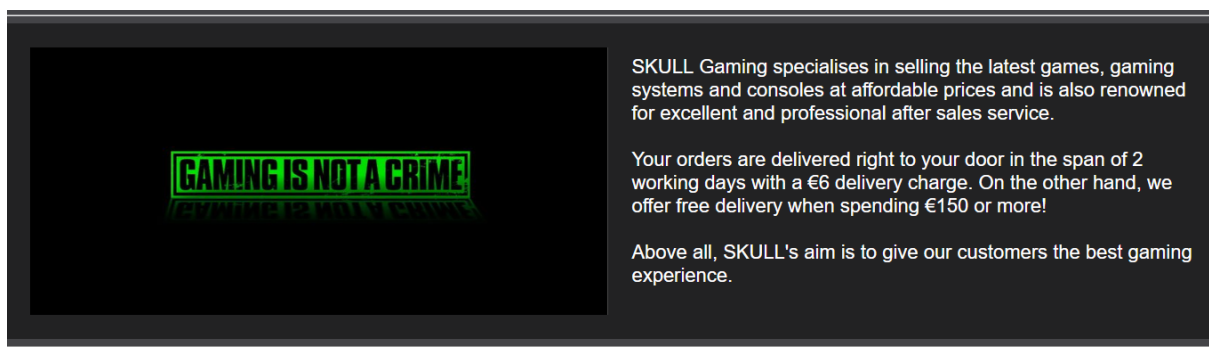
Please refer to the following screenshots to see how the home page is displayed.



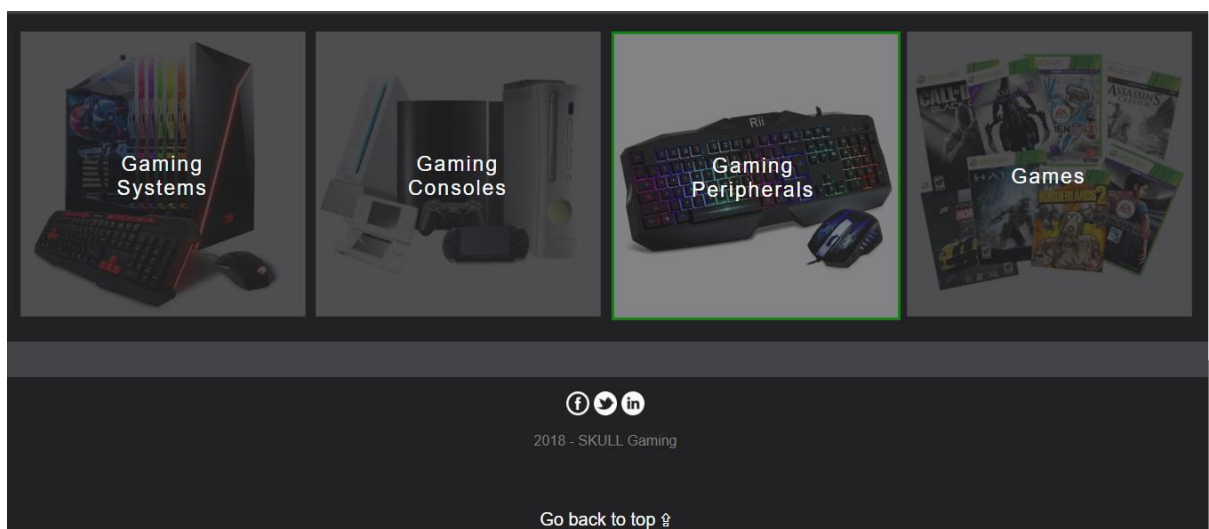
(The first picture of the slide show, the next arrow is being hovered upon and the 1st dot is activated)



(The slideshow when the 3rd dot is chosen or the slides where changed by the arrows, or automatically changed alone)



(A picture displayed next to the description of the company)



(Dynamically generated picture links. The Gaming Peripherals is being hovered over)

Contact Us Page – contact_us.php

At the start of the code, the title of the page is defined and the headers file is included. The 'company.xml' database is loaded so that the company's details can be accessed to be displayed later. If the database file fails to load, the 'exit' function outputs an error message and terminates the current script. A function called 'has_header_injection' is defined and will be used when processing the contact form input in order to prevent potential header injections.

```
<?php
//Title of the current page.
define("TITLE", "Contact Us | SKULL Gaming");
//Includes the header file.
include('Includes/header.php');
//Load's the company information database - 'company.xml'.
if(file_exists('Includes/company.xml')){
    $company = simplexml_load_file('Includes/company.xml');
} else {
    exit('File does not exist.');
```

Next, the code checks whether the form submit button has been clicked by the user. If the statement is true, the data from the HTML form are gathered and stored in appropriate variables. Then, the 'has_header_injection' function is used to determine whether there have been any header injections in the name, subject or e-mail fields. If a header injection is found, the 'die' function is used to exit the current script.

```
//Executes if the submit form button was pressed by the user.
if(isset($_POST['form_submit'])){
    //Gathering data from the user's HTML form and storing it in variables.
    $name = $_POST['name'];
    $subject = $_POST['subject'];
    $mail_from = $_POST['email'];
    $message = $_POST['message'];

    //If header injections are found kill the script.
    if(has_header_injection($name) || has_header_injection($mail_from) ||
has_header_injection($subject))
        die();
```

The code proceeds by declaring a variable containing the company e-mail and constructing the message that will be sent to the company e-mail by concatenating the variables containing the form details. The message will include the user's inputted name, e-mail and their message. The 'wordwrap' function is used to wrap the message to 150 characters per line to give it a well-ordered appearance. Subsequently, a header variable is declared containing information to be sent to the email programs such as its priority and its origin and destination. Afterwards, the e-mail is sent to the company using the 'mail' function containing the header variable, the message and the subject variable which was extracted as a string from the input of the user's 'subject' field. After the code described above, HTML is used to display a thank you message to user for filling it out the contact form as well as a button to go back.

```
//Declaring a variable containing the company e-mail.
$mail_to = "skullgaming.malta@gmail.com";

//Constructing the message that will be sent to the company e-mail with the form
```



```

details.
$txt = "You have recieved an e-mail from ".$name." --
".$mail_from.".\\n\\n".$message;
$txt = wordwrap($txt,150); //Wraps the message so that it has max 150 characters
per line.

//Constructing the header content for the e-mail.
$headers = "MIME-Version: 1.0\\r\\n";
$headers .= "Content-type: text/plain; charset=iso-8859-1\\r\\n";
$headers .= "From: " . $name . " <" . $mail_from . ">\\r\\n";
$headers .= "X-Priority: 1\\r\\n";
$headers .= "X-MSMail-Priority: High\\r\\n\\r\\n";

//Sends the e-mail to the company.
mail($mail_to, $subject, $txt, $headers);
?>
<!-- Displays message to user if form was submitted. -->
<p>Thank you for contacting us. We will be leaving a reply shortly...</p>
<button onclick="goBack()">Go Back</button>

```

If the contact form button was not submitted, the actual contact form is displayed using HTML. It is important to note that each input tag has a 'required' attribute which specifies that all of the input fields must be filled out before submitting the form.

```

}else{
    ?>
    <!-- ----- Contact Form ----- -->
    <form class="contact-form" action="" method="post">
    <p><label for="name">Full Name:</label>
    <input name="name" placeholder="Full name" type="text" required></p>
    <p><label for="email">E-mail:</label>
    <input name="email" placeholder="E-mail" type="email" required><br></p>
    <p><label for="subject">Subject:</label>
    <input name="subject" placeholder="Subject" type="text"
required><br></p>
    <p><textarea name="message" placeholder="Please enter the message here..."
required></textarea></p>
    <button type="submit" name="form_submit">Send</button>
    </form>
<?php }?>

```

Next, the code displays the company information in three different categories; 'Address', 'E-mail' and 'Phone number'. The information for each section is extracted from the 'company.xml' database file. For the 'Address' section, the location and opening hours are obtained by displaying the contents of the 'location' and 'hours' elements from the 'address' element in the XML file. For the 'E-mail' and 'Phone' section, a 'foreach' loop is used to display the names of types of e-mails and phone numbers as well as their contents. At the very end of the code a footer file is included.

```

</div>
<hr>
<br><br><br>
<div class="s2">
    <!-- ----- Company Information ----- -->
    <!-- Each section below acquires information about the company from the
company.xml database. -->
    <h2> Contact Information</h2><br>
    <div class="box"><br>
    <br>
    <h3>Address</h3>
    <div class="textbox">
    <!-- Displays the company address. -->
    <p><?php echo $company->{'address'}->location."<br>"?></p>
    <p class="boxh">Opening Hours</p>
    <!-- Displays the company opening hours. -->

```

```

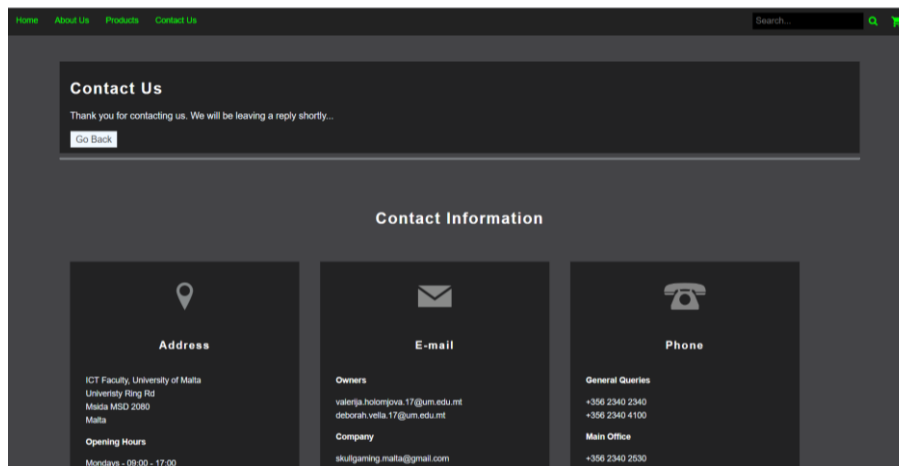
        <p><?php echo $company->{'address'}->hours?></p>
    </div><br>
</div>
<div class="box"><br>
    <br>
    <h3>E-mail</h3>
    <div class="textbox">
        <!-- Displays the company e-mails. -->
        <?php foreach($company->{'emails'}->email as $email){
            echo "<p class=\"boxh\">$email->name</p>";
            echo "<p>$email->contact</p>";
        }?>
    </div>
</div>
<div class="box"><br>
    <br>
    <h3>Phone</h3>
    <div class="textbox">
        <!-- Displays the company phone numbers. -->
        <?php foreach($company->{'phones'}->phone as $phone){
            echo "<p class=\"boxh\">\".$phone->name.\"</p>";
            echo "<p>$phone->number</p>";
        }?>
    </div>
</div>
<br><br>
</div>
<?php
//Includes the footer file.
include('Includes/footer.php');
?>

```

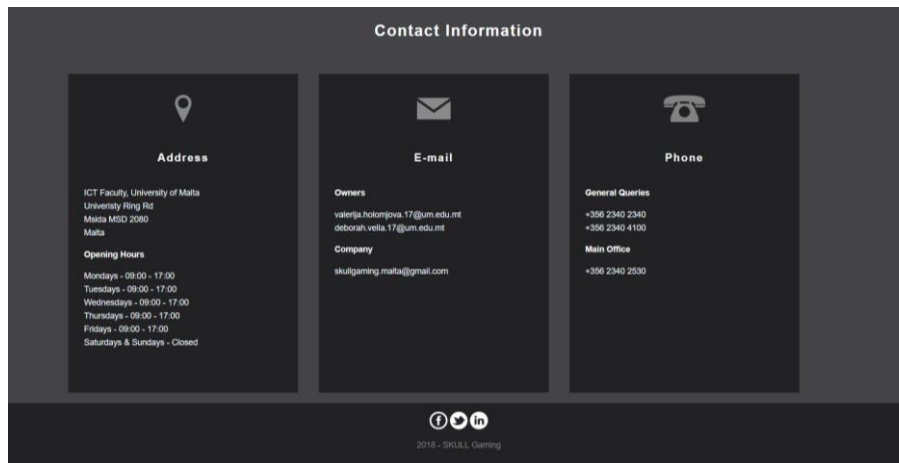
Please refer to the screenshots below and on the next page for an example of how the contact form looks before it is submitted and after, as well as the contact information.

The screenshot shows a web browser window with a dark theme. The browser's address bar displays 'http://localhost:8080/Contact_Us.php'. The page has a navigation bar with links: 'Home', 'About Us', 'Products', and 'Contact Us'. A search bar is located on the right side of the navigation bar. The main content area is titled 'Contact Us' and contains a form with the following fields: 'Full Name' (with a placeholder 'Full name'), 'E-mail' (with a placeholder 'E-mail'), 'Subject' (with a placeholder 'Subject'), and a large text area for the message (with a placeholder 'Please enter the message here...'). A 'Send' button is positioned at the bottom left of the form. Below the form, the text 'Contact Information' is visible. The browser's status bar at the bottom shows 'Contact_Us.php'.

(Screenshot illustrating the Contact Form before it has been submitted)



(Screenshot illustrating the Contact Form after it has been submitted)



(Screenshot illustrating the Contact Information Section)

About Us Page – about_us.php

At the start of the code, the title of the page is defined and the header file is included. The 'members.xml' database is loaded so that the company's employee details can be accessed to be displayed later. If the database file fails to load, the 'exit' function outputs an error message and terminates the current script.

```
<?php
//Title of the current page.
define("TITLE", "About Us | SKULL Gaming");
//Includes the header file.
include('Includes/header.php');
//Load's the employee information from database - 'members.xml '
if(file_exists('Includes/members.xml')){
    $employees = simplexml_load_file('Includes/members.xml');
} else {
    exit('File does not exist.');
```

The code proceeds by displaying an image with a heading at the top of the page using HTML for design purposes. Next, a description of the company is presented followed by a list of employees' in the company. The list contains the employee's names, positions and descriptions. These are extracted from the 'members.xml' database using a 'foreach' loop to access the contents of the name, position and description elements in the XML file. Finally, an image is displayed illustrating the logos of the companies whose products we have used in our website. A footer file is included at the end of the code.

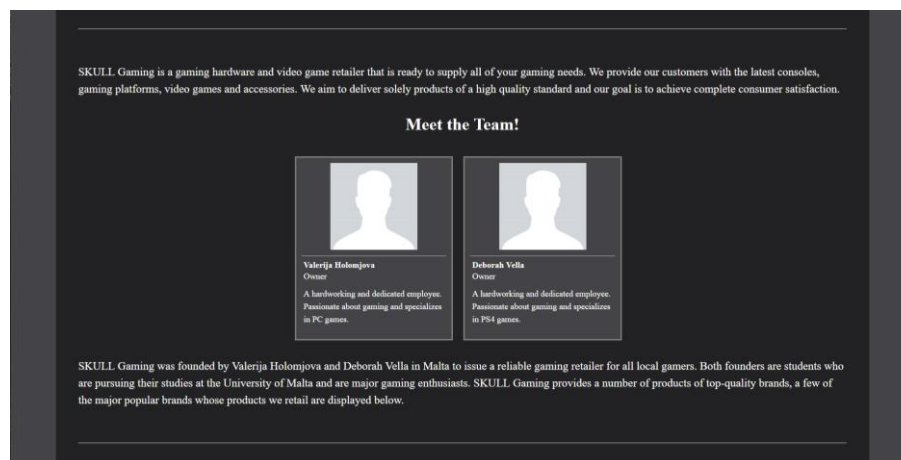
```
<div id="about">
    <!-- Image and Heading at the top of the page. -->
    <div class="top">
        
        <div class="t-text">
            About Us
        </div></div>
    <!-- ----- Description of Company ----- -->
    <div class="description">
        <hr><br>
        <p>
            SKULL Gaming is a gaming hardware and video game retailer that is ready
            to supply all of your gaming needs.
            We provide our customers with the latest consoles, gaming platforms,
            video games and accessories.
            We aim to deliver solely products of a high quality standard and our
            goal is to achieve complete consumer satisfaction.
        </p>
        <h2>Meet the Team!</h2>
        <!-- List of employees and their information extracted from the employee
        database. -->
        <div class='employees'>
            <?php
            foreach($employees as $employee){
                echo "<div class='employee'>
                    <img src='$employee->img'><hr>
                    <h3>$employee->name</h3>
                    <h4>$employee->position</h4>
                    <p>$employee->description</p></div>";
            } ?>
        </div><p>
            SKULL Gaming was founded by Valerija Holomjova and Deborah Vella in
            Malta to issue a reliable gaming retailer for all local gamers.
            Both founders are students who are pursuing their studies at the
            University of Malta and are major gaming enthusiasts.
```

```

        SKULL Gaming provides a number of products of top-quality brands, a few
of the major popular brands whose products we retail are displayed below.
        </p><br><hr>
    </div>
    <!-- Picture of company logos whose products we have used in our website. -->
    
</div>
<?php
//Includes the footer file.
include('Includes/footer.php');
?>

```

Please refer to the screenshot below for an example of how the list of employees has been displayed.



(Screenshot illustrating the list of employees in the company)

Product Page – products.php

At the start of the code, the title of the page is defined and the header file is included. The 'list.php' file is also included to display a list of product categories and subcategories that are linked to the current page.

```
<?php
//Title of the current page.
define("TITLE", "Products | SKULL Gaming");
//Includes the header file.
include 'Includes/header.php';
//Includes a list of links to product categories.
include 'Includes/list.php';
```

The code proceeds by checking the parameters that were passed into the product page, such as the value of 'subcat', 'cat' or 'prod'. If the 'subcat' variable appears as a parameter in the URL, the value of the variable is stripped of unwanted characters and stored in a variable. Next, the code accesses each product in the products databases. If the current product being accessed has the same category id as the one extracted from the 'subcat' variable, some of the details such as the name, image and price of that product is displayed. The link to its individual product page is also displayed. This process continues until all products of that subcategory are displayed.

```
//If 'subcat' appears in the link, displays all the products of the specific
subcategory.
if(isset($_GET['subcat'])) {
    echo"<div id=\"products\">";
    //Removes any unwanted characters and extracts the string after 'subcat' from
the link.
    $subcategory_choice = strip_bad_chars($_GET['subcat']);
    //Accessing each product in the products database.
    foreach($products->product as $product) {
        //If the product has the same category id from the link.
        if(strcmp($product->category_id, $subcategory_choice) == 0) {
            //Displays some of the details of the product.
            echo"<div class=\"item\">
                <a href='products.php?prod=$product->id'><img src='$product-
>img'></a><br>
                <h1>$product->name</h1>
                <h2>€ ".number_format((float)$product->price, 2, '.',
',')."</h2></div>";
        }
    } echo"</div>";
}
```

Next, an 'else if' statement checks whether the variable 'cat' appears as a parameter in the URL. If this condition is true, the value of the variable 'cat' is extracted, stripped from any unwanted characters and stored in another variable. Subsequently, each category in the products database is accessed. If the current category being accessed has a parent category id that is the same as the value extracted from the 'cat' variable it is considered a subcategory and the products of the subcategory is displayed. The products of each subcategory are then displayed in the same method as described in the 'subcat' section above. This process will repeat to display all products of the given category.

```
//If only 'cat' appears in the link, displays all the products of the specific
category.
} elseif (isset($_GET['cat'])) {
    echo"<div id=\"products\">";
    //Removes any unwanted characters and extracts the string after 'cat' from the
link.
    $category_choice = strip_bad_chars($_GET['cat']);
    //Accessing each category in the products database.
```

```

foreach($categories->category as $category){
    //If a subcategory has the same parent category id as the category id from
the link.
    if(strcmp(($category->parent_id), $category_choice) == 0){
        //Accessing each product in the database.
        foreach($products->product as $product){
            //Displays the products of that subcategory.
            if(strcmp(($product->category_id), $category->id) == 0){
                //Displays some of the details of the product.
                echo"<div class=\"item\">
                    <a href='products.php?prod=$product->id'><img
src='$product->img'></a><br>
                    <h1>$product->name</h1>
                    <h2>€ ".number_format((float)$product->price, 2, '.',
',')."</h2></div>";
            }
        }
    }
} echo"</div>";

```

Next an 'else if' statement checks whether the variable 'prod' appears as a parameter in the URL. If this condition is true, the value of the variable 'prod' is extracted, stripped from any unwanted characters and stored in another variable. Next, each product in the products database is accessed. If the current product being accessed has the same product id as the value of the 'prod' variable, then all of the details of the product such as the image, name, description and price will be displayed. A quantity input field and an "Add to Cart" button will be added so that the user could add the product of a specific quantity to their shopping cart session, as well as a "Go Back" button to resume looking for products. The quantity input field has been given a required attribute so the user must input a quantity before adding the product to the cart. The "Add to Cart" button will transfer the user to the 'cart.php' page. It is important to note that this will be the method in which individual products are displayed.

```

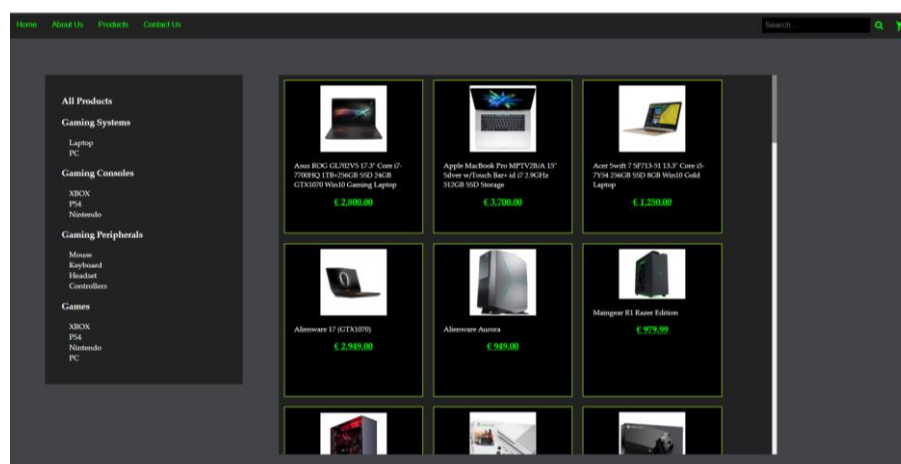
} elseif (isset($_GET['prod'])) {
    echo"<div id=\"product\">";
    //Removes any unwanted characters and extracts the string after 'prod' from the
link.
    $product_choice =strip_bad_chars($_GET['prod']);
    //Accessing each product in the products database.
    foreach($products->product as $product){
        //If the product has the same product id from the link.
        if(strcmp(($product->id), $product_choice) == 0){
            //Displays all of the details of the product.
            echo "<div class=\"item\">
                <h3>$product->name</h3><br>
                <img src='$product->img'>
                <h2>€ $product->price</h2>
                <p>$product->description</p>
                <button onclick=\"goBack()\">Go Back</button>
                <form class=\"cart_form\"
action=\"cart.php?action=add&prod=$product->id\" method=\"post\">
                    <input name=\"quantity\" placeholder=\"Quantity\"
type=\"text\" required>
                    <button type=\"submit\" name=\"cart_submit\">Add to
cart</button>
                </form></div>";
        }
    }echo"</div>";
}

```

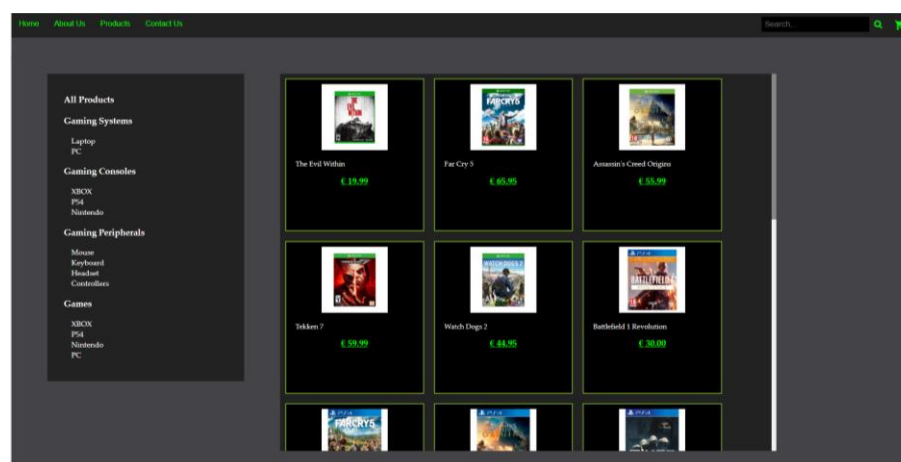
If neither of the 'if' statements mentioned above are satisfied, the page will display all of the products in the database. This is done by accessing each product in the database and displaying their details. At the end of the code, the footer file is included.

```
//Displays all the products in the database.
echo"<div id=\"products\">";
//Accessing each product in the products database.
foreach($products->product as $product){
    //Displays some of the details of the product.
    echo"<div class=\"item\">
        <a href='products.php?prod=$product->id'><img src='$product->
>img'></a><br>
        <h1>$product->name</h1>
        <h2>€ ".number_format((float)$product->price, 2, '.',
',')</h2></div>";
    }echo"</div>";
}
//Includes footer file.
include 'Includes/footer.php';
?>
```

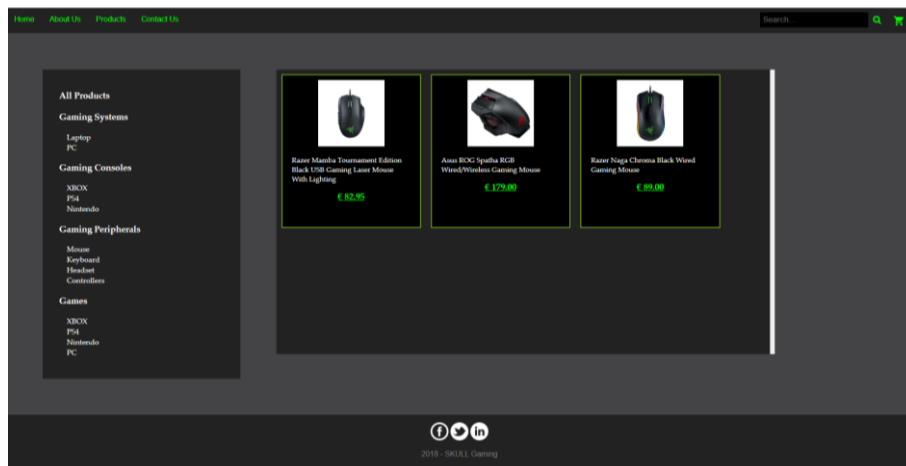
Please refer to the screenshot below for an example of the different methods in which the products can be displayed.



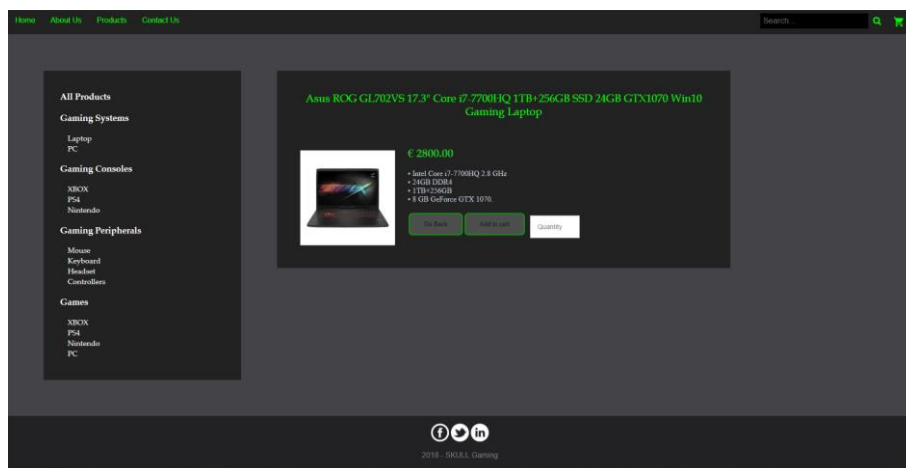
(Screenshot illustrating all the products from the database.)



(Screenshot illustrating all the products from 'Games' category.)



(Screenshot illustrating all the products from 'Mouse' subcategory.)



(Screenshot illustrating an individual laptop product.)

Search Page – search.php

At the start of the code, the title of the page is defined and the header file is included. The 'list.php' file is also included to display a list of product categories and subcategories that are linked to the current page.

```
<?php
//Title of the current page.
define("TITLE", "Products | SKULL Gaming");
//Includes the header file.
include 'Includes/header.php';
//Includes a list of links to product categories.
include 'Includes/list.php';
//If the search button has been successfully activated.
```

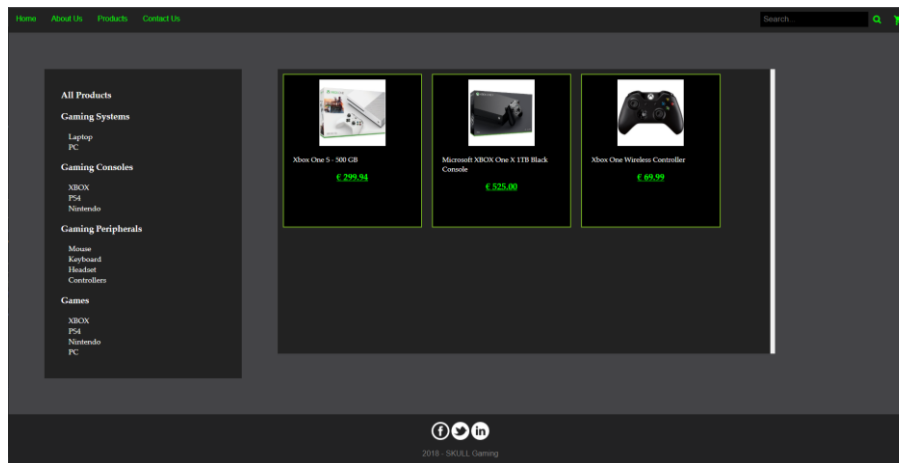
If the “Search” icon has been successfully activated, the value of the search input field is extracted and trimmed for unwanted characters. It is then stored in a variable called ‘search’. A variable called ‘found’ is declared and set to false and signifies whether a product similar to the ‘search’ variable is found. Next, each product in the database is accessed and checked whether the ‘search’ variable is a substring of any of the products names. If this condition is true, the ‘search’ variable is set to true and the details of the similar product is displayed. This process continues until all products that containing the ‘search’ variable substring in their name are displayed.

```
if (isset($_POST['search_submit'])) {
    //Extracts string from the search input field and removes whitespaces and other
    unnecessary characters.
    $search = trim($_POST['search']);
    //Variable is false if no such products exist.
    $found = false;
}
?>
<div id="products">
<?php
//Accessing each product in the database.
foreach ($products->product as $product) {
    //If the $search variable is a substring in the products name (case-
    insensitive).
    if (stripos($product->name, $search) !== false) {
        //Signifying at least one similar product has been found.
        $found = true;
        //Displays the details of the product.
        echo "<div class=\"item\">
            <a href='products.php?prod=$product->id'><img
src='$product->img'></a><br>
            <h1>$product->name</h1>
            <h2>€ ".number_format((float)$product->price, 2, '.',
',')."</h2>
            </div>";
    }
}
```

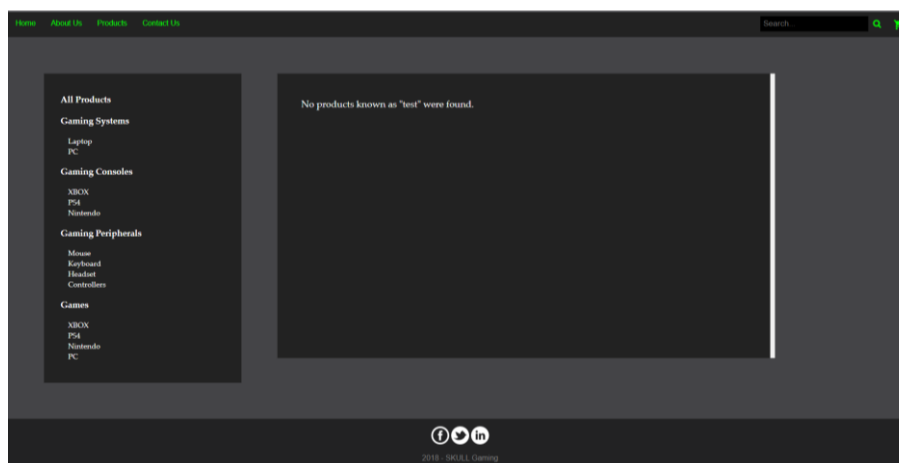
After the loop, an if statement checks whether the variable is ‘found’ is equivalent to false signifying that no similar products were found. If this statement holds, a message is displayed on the page that no such products were found. At the end of the code, a footer file is included.

```
//Executes no similar products were found.
if($found == false){
    //Displays a message that no such products were found on the page.
    echo "<h1 class='warning'>No products known as \"\$search\" were found.</h1>";
}
?> </div> <?php
include 'Includes/footer.php';
?>
```

Please refer to the screenshot below for an example the page when there are no products found and when there are similar products found.



(Screenshot illustrating all the products displayed after “xbox” was inputted in the search field.)



(Screenshot illustrating the page when no similar products were found.)

Header File – header.php

This file is included at the start of every page in the website and can be found in the 'Includes' folder. At the start of the code, a constant is defined and set to hold the company's name and the time zone is set. Both will be used in the footer for design purposes. Next, the file 'array.php' is included for the navigation bar. Subsequently, the products database is loaded and variables are set to represent the products and categories in the database so that they can later be easily accessed. A function called 'strip_bad_chars' is defined to be used to remove unwanted characters when extracting parameter values from the URL.

```
<?php
//To start a new session or resume an existing session.
session_start();

//Defining the company name as a constant for the footer.
define("COMPANY", "SKULL Gaming");
//Setting the timezone.
date_default_timezone_set('Malta/Europe');

//An array of website pages.
include('array.php');

//Loading the products database.
if(file_exists('Includes/products.xml')){
    $product_page = simplexml_load_file('Includes/products.xml');
} else {
    exit('File does not exist.');
```

The title of the page is set by echoing a constant called 'TITLE' which will be defined at the start of every page. Next, the stylesheets are included followed by a script containing a function called 'goBack' which will be used to return to the previous page for "Go Back" buttons. At the end of the code, the company logo and the navigation bar are included.

```
<!DOCTYPE html>
<html>
<head>
    <!-- Setting the title of the page. -->
    <title><?php echo TITLE; ?></title>
    <!-- ----- Stylesheets ----- -->
    <!--For Main-->
    <link rel="stylesheet" href="Stylesheet/style.css?">
    <!--For Index-->
    <link rel="stylesheet" href="Stylesheet/index.css?">
    <!--For Navigation Bar-->
    <link rel="stylesheet" href="Stylesheet/nav.css?">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <!--For google icons usage-->
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
    rel="stylesheet">
    <script>
        //Function to go back to previous page.
```

```

        function goBack() {
            window.history.back()
        }
    </script>
</head>
<body>
<!-- Company Logo -->
<div id="banner">

</div>
<!-- Navigation Bar -->
<?php include('Includes/nav.php') ?>

```

Please refer to the screenshot below for an example of how the company logo and navigation bar are displayed in every page.



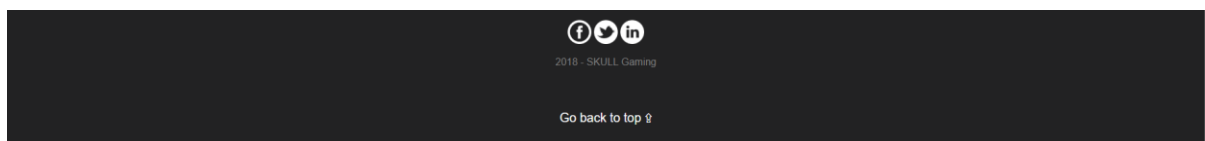
(Screenshot illustrating the Company Logo and Navigation Bar – the header of each page.)

Footer File – footer.php

This file is included at the end of every page in the website and can be found in the 'Includes' folder. At the start of the code, social media icons are displayed on the page. These were added for design purpose but in a real website these could be linked to the social media pages of the company. Next, the year and company name is displayed as well as a "Back to Top" bookmark so that the user could be redirected to the top of the page.

```
<br><br>
<div id="footer">
  <!-- Social Media Icons -->
  
  
  
  <!-- Company Name and Current Year -->
  <p><?php echo date('Y');?> - <?php echo COMPANY ?></p>
  <br><br>
  <!-- Back to Top -->
  <p><a href="#top">Go back to top &#8682;</a></p>
</div>
</body>
</html>
```

Please refer to the screenshot below for an example of how the footer file is displayed in every page.



(Screenshot illustrating the footer of each page.)

Shopping cart file - cart.php

This php file deals with the procedures of adding and removing products to and from the cart. It also handles the output of the contents in the cart. When the website is directed to cart.php the first thing the program does is check if any of the adding or removing buttons were clicked. This is done by the use of a switch case that contains three cases which are: add, remove and remove_all. If none of these cases are to be executed, the program outputs the products stored inside the cart. A Session is used to store the products added by the user.

The Switch

If the first case, add, is met, the program checks if the form was posted. It then gets the quantity entered by the user and stores it into the variable \$quantity. If the quantity is found to be greater than 20, the program notifies the user that the same product can be bought with a quantity of less than 20.

Then, when the Session is not empty, the program starts inputting the product in the cart, else it creates a session and adds the product. Before the product is added to cart, it first checks if the product already exists inside the cart. If it does exist, a variable called \$flag is set to 1. This variable is later used inside an if() condition which checks whether the value stored is 0 or 1. If it is 0, the function new_add_qty() is called passing \$item_id and \$item_qty as parameters, else, add_qty_cart() is called passing the same parameters.

```
case "add": //executes when user chooses add to cart
    //if the form was posted, execute the following code
    if(isset($_POST['cart_submit'])) {

        $item_id = strip_bad_chars($_GET['prod']); //removes unwanted characters in
        case someone tries to hack and alter the script. function found in header.php
        $item_qty = $_POST['quantity'];

        //does not let the user buy more than 20 products from the same item.
        if($item_qty > 20){
            echo "<div
            <div id= \"cart\">
            <div class= \"display\">
            <h1>Quantity cannot be greater than 20<h1>
            </div>
            </div>";
            goBack();
        }

        header('Location: cart.php');
        //changes the url to direct it to cart.php without any extra data so when
        it is refreshed it won't use the data passed through the add to cart.

        //if the session 'cart' isn't empty
        if (!empty($_SESSION['cart'])) {
            $flag = 0; //It will later stay set to 0 if item id is never found
            inside the cart

            //go through cart session
            $cart_size = count($_SESSION['cart']);
            for($counter = 0; $counter < $cart_size; $counter++) {
                if (!$_SESSION['cart'][$counter]["$item_id"]){
                    continue; //jump to iteration
                }
                $flag = 1; //set $flag equal to 1 when the item id already exists
```

```

in the cart
    }

    if($flag == 0){
        new_add_qty_cart($item_id, $item_qty); //calls the function that
        adds a product that was never added before
    } else {
        add_qty_cart($item_id, $item_qty); //calls the function that adds
        only the quantity to an already existing product inside the cart
    }

    } else {
        //else create an array for session cart and add the product
        $_SESSION['cart'] = array();
        new_add_qty_cart($item_id, $item_qty);
    }
}
break; //end of 1st case

```

The function's new_add_qty_cart() job is to add the item id and its respective quantity inside the cart Session. On the other hand, the method add_qty_cart() adds only the quantity to an already existing one. This is done by looping through the whole Session and keeps going until the place where the same item id is stored. The quantity passed as parameter is added with the stored quantity at the index found. The loop then terminates.

```

function new_add_qty_cart($item_id,$item_qty){ //called when the added item isn't
already in cart
//this function adds the item id and its corresponding quantity to the cart
Session
    array_push($_SESSION['cart'],array( $item_id => (int)$item_qty));
}

function add_qty_cart($item_id,$item_qty, $counter){
    $cart_size = count($_SESSION['cart']);

    for($counter =0; $counter<$cart_size; $counter++) {
        if(!$_SESSION['cart'][$counter][$item_id]){
            continue; //program jumps to iteration
        }
        $_SESSION['cart'][$counter][$item_id] += $item_qty; //adds the newly
        entered quantity to the last stored quantity of a particular product.
    }
}

```

If the second case, remove, is met, the program gets the product id and quantity and removes any characters that are extra. This is a safe procedure that prevents scripts from being hacked and changes by another individual. The cart size is found so that it will later be used inside a do...while() loop condition. The line: header('Location: cart.php'); is implemented so that when the cart page is refreshed the user is sent to cart.php but this time no variables are passed. This prevents products from being removed on refresh.

```

case "remove": //executes if user chooses to remove an item for the cart

    //gets passed product product id and quantity and removes extra characters for
    safe practices
    $item_id= strip_bad_chars($_GET['prod']);
    $item_qty= strip_bad_chars($_POST['qty']);

```



```

    $cart_size = sizeof($_SESSION['cart']);
    $counter=0;

    header('Location: cart.php'); //It redirects the user to cart.php page on
refresh

```

Inside the do...while() the program checks if the Session element pointed by a counter matches with item id. When they match, the product at that index is removed and the iteration is terminated. If they do not match, the counter is incremented and the loop continues.

```

do
{
    //find the array with that product id

    if($_SESSION['cart'][$counter][$item_id]) {
        array_splice($_SESSION['cart'],$counter,1); //Remove the elements from the
Session, found in the index pointed by $counter and removes 1 element
        break; //break the loop when element is removed
    }

    $counter++; //if the product id is not found increment the $counter to loop
through the rest of the elements
}while($counter<$cart_size); //loop as long as $counter is less than the
$cart_size
break; //end of 2nd case

```

The 3rd case simply deletes the whole Session. It first removes all the data from the variables and then destroys the entire Session. A brand-new Session is created ready to be used.

```

case "remove_all":
//removes all contents from the cart
session_unset($_SESSION);
session_destroy($_SESSION); //destroys the Session not just the data
$_SESSION['cart'] = array(); //recreate an empty Session

break; //end of 3rd case

```

Displaying Cart

If the shopping cart is not empty, all products added are displayed. At the top two buttons are displayed; one for proceeding to checkout and another to remove all items. For all products found the following are printed on screen: picture of product, name, price in euros up to 2 decimal places, a placeholder for quantity and a button that removes the product from cart. When the shopping cart is found to be empty a message "The shopping cart is empty" is printed to inform the user.

```

<!--DISPLAYS THE CART AND OPTIONS TO REMOVE OR SUBMIT-->
<div id="cart">
    <div class="display">
        <h1>My Shopping Cart</h1>
        <?php if($cart_size != 0){ ?>
            <form class="checkout_form" action="checkout.php" method="post">
                <!--directs the website to checkout.php if submit is clicked-->
                <button type="submit" name="checkout_submit" class="all"
>Checkout</button>
                <!--directs the website to cart.php if submit is clicked passing
remove_all which is then used inside a switch case-->
                <button onclick=
"window.location.href='cart.php?action=remove_all'" class="all" type="button"
>Remove all items</button>
            <?php
                for($counter=0; $counter<$cart_size; $counter++) { //loops through Session to
output contents

```

```

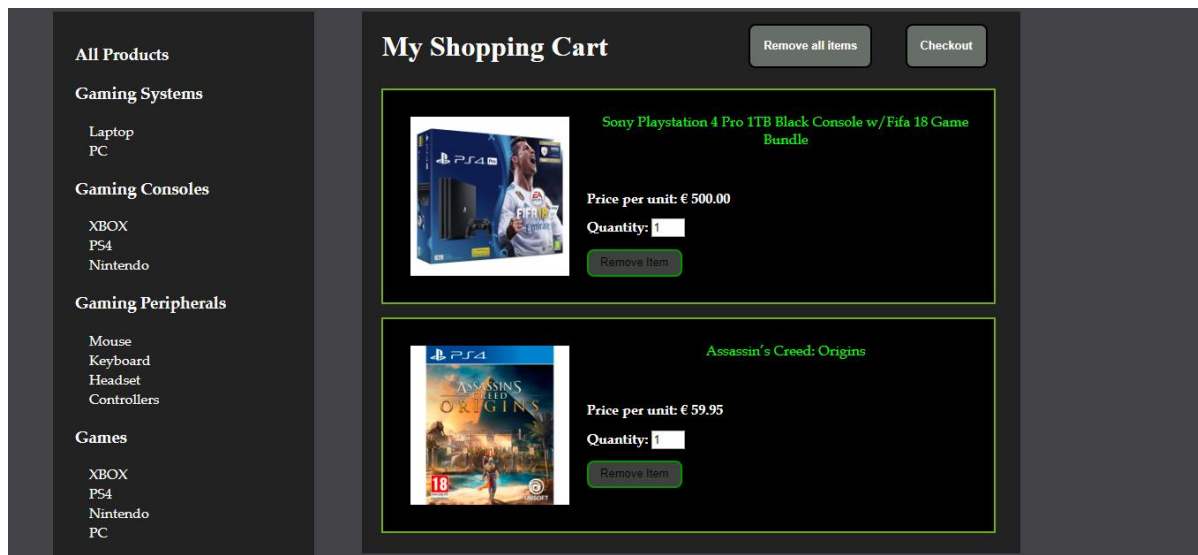
    foreach ($products->product as $product) {
        if(array_key_exists("$product->id" ,$_SESSION['cart'][$counter])) { //check
if the fetched product id exists inside the Session. If it exists, output its
details
            $quantity = $_SESSION['cart'][$counter]["$product->id"];
            echo "<div class=\"item\">
                <img src='$product->img'>
                <h3>$product->name</h3><br>
                <h2>Price per unit: € ".number_format((float)$product-
>price, 2, '.', ',')."</h2>
                <h2>Quantity: <input name=\"$product->id\"
value=\"$quantity\" type=\"text\"></h2>
                <button onclick=
\"window.location.href='cart.php?action=remove&prod=$product->id'\"
type=\"button\">Remove Item</button>
                </div>";
            }
        }
    }

?> </form>
    <?php }else{?>
    <br><br><br><br>
    <h4>The shopping cart is empty.</h4>    <!--If cart size is equal to 0
notify the user that the cart is empty-->
    <?php } ?>

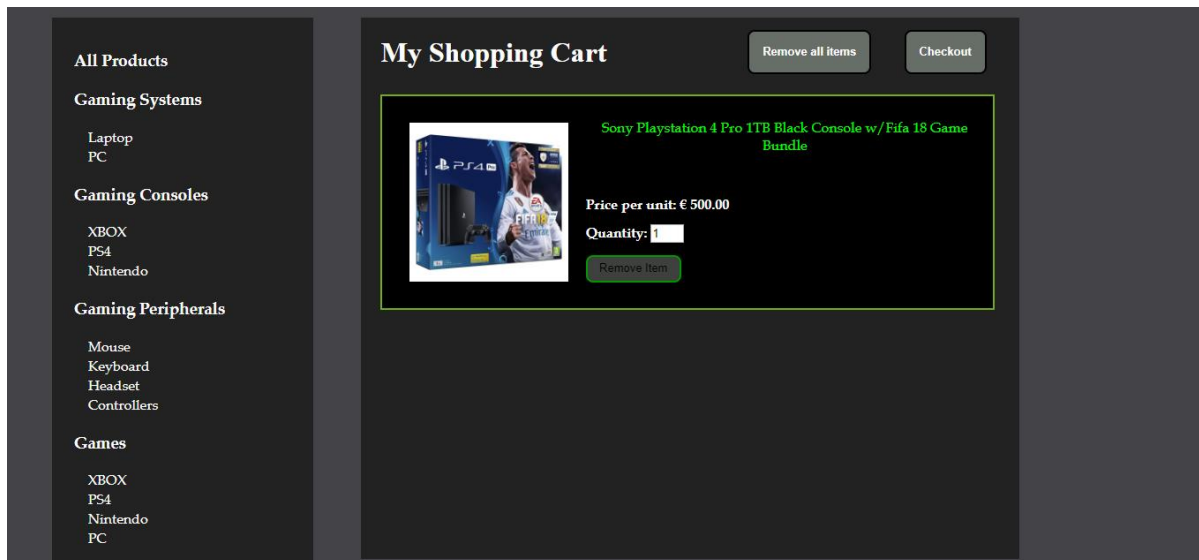
    </div>
</div>

```

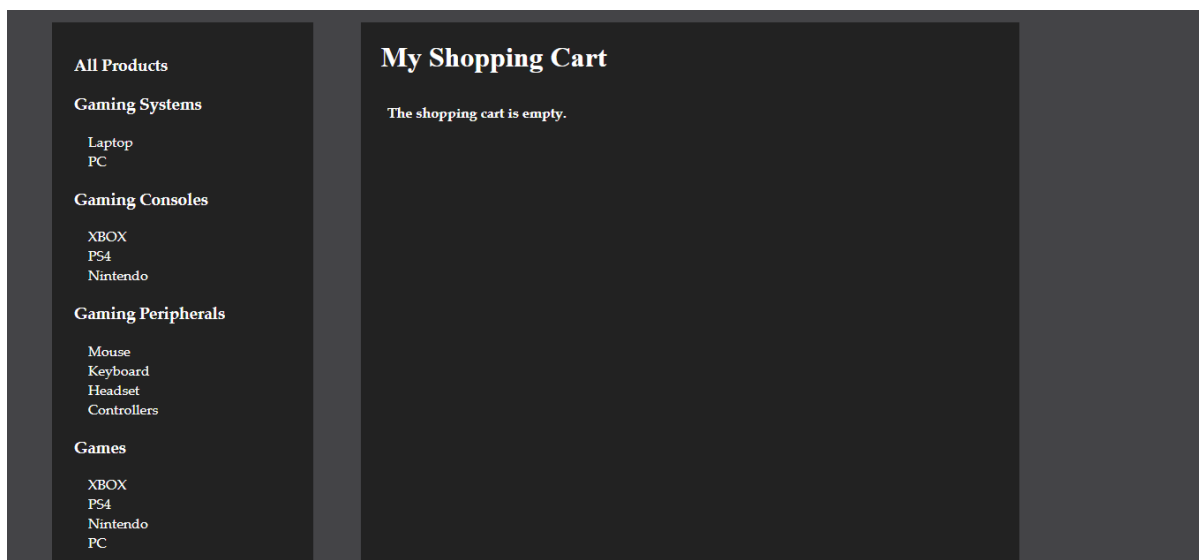
Please refer to the screenshots below for an example of how the cart looks like in all possible cases.



(Cart with Products in it)



(Cart after an item is removed)



(Cart after all items are removed. Cart is empty)

Checkout Page – checkout.php

This php file is split into two if statements; one caters for displaying the products to be bought, calculating the prices and takes input from the form to be filled by the customer (2nd if), while the other caters for sending the emails to the company email (1st if).

After including the header.php and the list.php, a function called `has_header_injection()` is coded. The role of this method is to check for any possible header injections that may be typed in by an attacker. This will be called later on inside the first `if()` statement.

```
<?php
define("TITLE", "Products | SKULL Gaming");
include 'Includes/header.php';
//Includes list of links to products on the left
include 'Includes/list.php';

function has_header_injection($str) {
    return preg_match("/[\r\n]/", $str); //in case an attacker injects other
headers into the message
}
```

The first if statement executes when the user chooses confirm other in the previous page, that is the shopping cart. It starts by getting the filled out form details and store each piece of information inside its respective variable. The information received are the name, email, credit number, full address and the total price. After all variables are set, the function `has_header_injection()` is called twice, passing the variables name and email one at a time. If any of these two results in having header injections, the script is killed.

```
if(isset($_POST['payment_submit'])){ //if user clicked on confirm order button
    //gets the customer details
    $name = $_POST['name'];
    $email = $_POST['email'];
    $credit_no = $_POST['card_no'];
    $address = $_POST['address_1']."\n".$_POST['address_2']."\n";
    $address .= $_POST['city']."\n".$_POST['country']." ".$_POST['post_code'];
    $total = $_POST['charge'];

    //If header injections are found kill the script.
    if(has_header_injection($name) || has_header_injection($email)){
        die();
    }
}
```

The code continues by storing the company's email address, the subject to be sent (contains the customer's email address), and the details (address and credit card number (for now)). As part of the details the products bought and their own details need to be stored as well. To do so, the code loops through the cart, and for each product, its product id is to be checked if it exists in the cart. If it does exist, the product id and quantity is concatenated with the other details initialised before. The same variable `$details` is then updated with the overall price correct to 2 decimal places.

```
//company email
$company_email = "skullgaming.malta@gmail.com";

$subject = "Order Placement by $email";

$details = "Below are the order details:\n\n"."Address:\n".$address."\nCredit
Number:\n".$credit_no."\nProducts:\n";
```

```

$cart_size=sizeof($_SESSION['cart']);

for($counter=0; $counter<$cart_size; $counter++) {
    foreach ($products->product as $product) {
        if(array_key_exists("$product->id" ,$_SESSION['cart'][$counter])) { //if
product id exists in the cart store the quantity inside the variable $quantity
            $quantity = $_SESSION['cart'][$counter]["$item_id"];

            $details .= "Product ID: " . $product->id. "\nQuantity:
".$quantity. "\n";
            //concatenate the product id and quantity with the details already
stored above.
        }
    }
}

$details .= "\nOverall price order: ".number_format((float)$total, 2, '.', ',');
//concatenate with the details, the total price correct to 2 decimal places

```

The message to be sent consists of the customer's name and email and the details found and stored above. When the message exceeds 200 characters, it is split into separate lines by using the `wordwrap()` in-built function. This is followed by custom built headers. The email is eventually sent using the function `mail()` which takes the company email as a receiver, the subject previously stored, the message to be sent and the headers. Last but not least the Session cart is destroyed and a new one is created. A thank you message is finally outputted on screen

```

$msg = $name." -- ".$email." has made a delivery.\n\n".$details; //msg using
customer's name and email. The details are outputted all
$msg = wordwrap($msg,200); //when it reaches the length of 200 split the msg in new
lines

//Custom Built Headers.
$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-type: text/plain; charset=iso-8859-1\r\n";
$headers .= "From: " . $name . " <" . $email . ">\r\n";
$headers .= "X-Priority: 1\r\n";
$headers .= "X-MSMail-Priority: High\r\n\r\n";

//Sends the e-mail - prebuilt function.
mail($company_email, $subject, $msg, $headers);
//sends it to the company with the previously defined subject and the message to be
sent

//removes all contents from the cart
session_unset($_SESSION);
session_destroy($_SESSION);
$_SESSION['cart'] = array(); //create new Session
?>
<div id="checkout">
    <div class="display">
        <br><br><h2>Thank you for shopping with us!</h2>
    </div>
</div>
<?php } //end of if

```

In the case where the condition of the 1st if statement is not met, the program checks the second if statement's condition. It executes when the user clicks on the checkout button inside the shopping cart page. A Go Back button is created to allow the user to go back to the shopping cart page. The checkout summary is displayed in the form of a table having the 1st column with products pictures,

2nd column with names, 3rd column with price per unit, 4th column with quantity and last column with the total price of that specific product.

```
else{
if(isset($_POST['checkout_submit'])){?> <!--If the user clicked checkout button in
the cart page-->
<div id="checkout">
    <div class="display">
        <h1>Checkout Summary</h1>
        <button onclick="goBack()" class="all">Go Back</button>
        <div class="grid-container">
            <!--Will output products in shopping cart in the form of a table. the
            Column headers are specified below-->
            <div class="grid-item"><h2>Product</h2></div>
            <div class="grid-item"><h2>Name</h2></div>
            <div class="grid-item"><h2>Unit Price</h2></div>
            <div class="grid-item"><h2>Quantity</h2></div>
            <div class="grid-item"><h2>Total Price</h2></div>
```

the cart size is then found and the variable \$overall_price is initialised to 0. A for loop is used to output the product/s to be purchased in the table. Inside the loop for each product, check if its id exists inside the Session and when it does, the total price is calculated (multiplying the price per unit with the quantity) and the new overall price is calculated as well (adds the total price found with the value of overall price). This is followed by the actual outputting of the details and using 2 decimal places whenever a price is displayed. The overall price is printed right after the details table.

```
<?php $cart_size=sizeof($_SESSION['cart']);
$overall_price = 0; //starting from 0 euros and accumulate price later

for($counter=0; $counter<$cart_size; $counter++) { //loop through cart
    foreach ($products->product as $product) { //for each product, if its id
    exists in the cart execute code below
        if(array_key_exists("$product->id" ,$_SESSION['cart'][$counter])) {

            $quantity = (float) $_POST["$product->id"];
            //update cart quantity for email
            $_SESSION['cart'][$counter]["$item_id"] = $quantity;
            $total_price = (float) ($quantity)*((float) $product->price);
            //multiplies quantity of product with its price to gain the total price for that
            product
            $overall_price += (float) $total_price; //add the last calculated
            price to the contents inside the overall price

            //output Product details
            echo "    <div class=\"grid-item\"><img src='$product->img'></div>
                <div class=\"grid-item\"><h3>$product->name</h3></div>
                <div class=\"grid-item\"><h3>€
                ".number_format((float)$product->price, 2, '.', ',')."</h3></div>
                <div class=\"grid-item\"><h3>$quantity</h3></div>
                <div class=\"grid-item\"><h3>€
                ".number_format((float)$total_price, 2, '.', ',')."</h3></div>";
        }
    }
}?>
</div>
```

Directly after, a form is created in which the customer has to enter his own personal details. The form caters for the following fields user details:

- Name
- Email
- Card number
- Address 1
- Address 2
- City
- Country
- Postal code

When the Confirm Order button is clicked, the overall price is passed and all entered fields are used to send the email. This section was discussed in the 1st if statement code.

```
<h4>Overall Price: € <?php echo number_format((float)$overall_price, 2, '.',  
,');?></h4>  
<h2>Please fill out the form details below.</h2>  
<!--The code below creates text boxes for the user to enter the required  
details-->  
<form class="payment_form" action="" method="post">  
<p><label for="name">Full Name:</label>  
<input name="name" type="text" required></p>  
<p><label for="email">E-mail:</label>  
<input name="email" type="email" required><br></p>  
<p><label for="card_no">Card number:</label>  
<input name="card_no" type="text" required><br></p>  
<div class="address">  
<p><label for="address_1">Address Line 1:</label>  
<input name="address_1" type="text" required><br></p>  
<p><label for="address_2">Address Line 2:</label>  
<input name="address_2" type="text" required><br></p>  
<p><label for="city">City</label>  
<input name="city" type="text" required><br></p>  
<p><label for="country">Country:</label>  
<input name="country" type="text" required><br></p>  
<p><label for="post_code">Postal Code:</label>  
<input name="post_code" type="text" required><br></p>  
</div>  
<input type="hidden" name="charge" value='<?php echo $overall_price;  
>?'>  
<button type="submit" name="payment_submit" class="all">Confirm  
Order</button>  
<!--The button activates the submitting payment and sending an email  
procedure-->  
</form>  
</div>  
</div>  
  
<?php }>  
include 'Includes/footer.php';  
>
```

Please refer to the following screenshots for a representation of how the checkout page looks like.



(The details of the products added to cart are shown)



(Shopping cart displayed when the Go Back button is clicked)

Overall Price: € 619.90

Please fill out the form details below.

Full Name:

E-mail:

Card number:

Address Line 1:

Address Line 2:

City:

Country:

Postal Code:

Confirm Order

(The form and its fields ready to be entered.)

Mouse
Keyboard
Headset
Controllers

Games

XBOX
PS4
Nintendo
PC

Overall Price: € 619.90

Please fill out the form details below.

Full Name:

E-mail: ⚠ Please fill out this field.

Card number:

Address Line 1:

Address Line 2:

City:

Country:

Postal Code:

[Confirm Order](#)

(When the Confirm order is clicked without entering any of the fields)

Keyboard
Headset
Controllers

Games

XBOX
PS4
Nintendo
PC

Please fill out the form details below.

Full Name:

E-mail:

Card number:

Address Line 1:

Address Line 2:

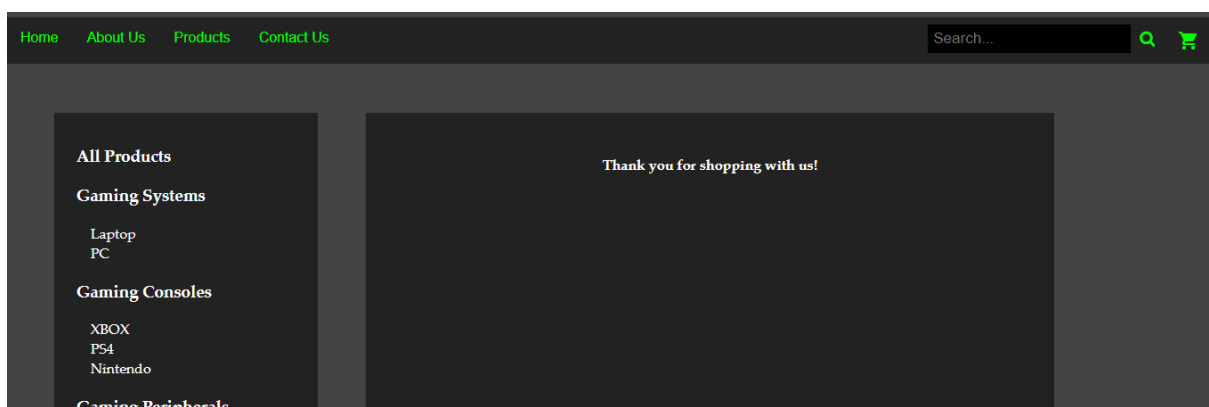
City:

Country:

Postal Code:

[Confirm Order](#)

(Fields all filled out)



(Confirm Order is clicked when all details are filled in)