**IS 491 Senior Project I**

**Online Supermarket**

**Xhoni Robo, 100097396**

**Author:**   **, Date:**

*signature*

**Supervisor:**   **, Date:**

*signature*

**Table of Contents**

[Abstract: 3](#_Toc469397212)

[1.1 Introduction 4](#_Toc469397213)

[2.1 Analysis: Functional Requirements 6](#_Toc469397214)

[2.2 Analysis: Non-functional requirements 7](#_Toc469397215)

[3.1 Design: Basics 8](#_Toc469397216)

[3.2 Design: Pages In-Depth 15](#_Toc469397217)

[3.3 Design: Database 17](#_Toc469397218)

[4.1 Implementation 19](#_Toc469397219)

[5.1 Results and Conclusion 29](#_Toc469397220)

[6.1 References 33](#_Toc469397221)

# Abstract:

In the chaotic times that we live in today, most people do not have that much free time. For them, every second of a day counts and they have to spend it wisely. This online supermarket is aimed at helping these exact individuals take one chore out of their list. Once complete, the online supermarket will be a website that people can use to order common everyday items online, and have them delivered home. This would work similarly to, for example, a pizza place would.

The website is going to be built from scratch using HTML or HTML5, with great care that it displays as intended on all everyday computer browsers (Chrome, Firefox, Opera, I.E.) as well as mobile phones. It will be connected to a database and will have queries for items, orders, and more, all created using MySQL. The scripts that will make the website work will be done in PHP, a very well known scripting language that is perfect for websites, and relatively simple to use.

The basic functionalities of the website include an Admin and User portal. The admin can check the database for items and orders, as well as update it if need be. The user will simply be able to browse the contents of the online supermarket and place orders. More functionalities will be added based on time constraints, like orders through SMS, a Bulgarian version of the website (with the help of a native speaker), etc.

**Declaration of authorship:**

“The Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of prof. Stoyan Bonev. All sources, used in development, are cited in the text and in the Reference section.”

Author:

# 1.1 Introduction

In this day and age time is something that for most is a luxury. People have difficulties managing between harsh work hours and simple things like buying groceries. As such, this need leads to the possibility of a quality of life improvement that can turn people's lives around. This is where the idea of an online supermarket is born. Not the first initiative ever, as the original idea belongs to South Koreans who used something similar in the form of online food dispensers. The way the website works is that it allows users to log in and browse the items that are in store. After picking what they like and adding it to their carts, they can finalize the purchase by paying.

The user is first presented with the home page. On the top right corner there are links for navigating back and forth between the different web pages. The first link in order of importance is the Sign Up link. Using this link, a user will be able to have his own username and password credentials to log into the website. This will help him in various ways, like keeping his settings and allowing him to make orders of multiple items. The new user info is stored in the database. For security purposes, the password uses a hashing function, which in this case is sha1.

Secondly, there's the Log In link. Of course, after signing up this is the next logical step into accessing the users own information. The login form will allow the user to enter their username and password. Both will be checked for correct input and whether they exist or not in the database. The way the password is checked keeps the user secure. Again, the input is hashed using sha1 and matched upon the pre-existing password in the database. Should the passwords be a match, the user is allowed to log in.

An important aspect of the log in is that there is a special account already in the database. That is the admin account. The admin is supposed to be, in this case, the supervisor of the database and items and products. After a successful login with the username "admin" and password "password", the admin will be presented with a table of all the products currently being offered. From there, he is given the ability to manipulate the data to his liking.

Examples include being able to add a record. This is done through the help of a php file that uses a webform to insert the data. The data is first checked for correctness, and then inserted into the database. A similar principle is used by the update links on each item. They use the GET method to get the ID of the item that is to be modified, while the rest of the information is supplied through a webform.

To top it off, the admin can also delete products from the database. The link that does this uses the exact same principle that the update php file does. We use a GET method for the id, match it using a query to the existing one in the database, and then execute the statement to delete it before returning back to the beginning and allowing the admin to do something else.

The average user however will not be able to access the database directly. Instead, all he can do is browse the items and add them or remove them back and forth from the cart. Both the products and the cart pages use clever looping to display the items in a professional manner so that the user interaction with the pages is a pleasant one. The database is accessed only through php files, at least under normal circumstances. That way, it is protected by unwanted visitors, as php code does not get executed in the browser, but in the server. This adds an extra layer of safety for the customer and for the database as well.

Most, if not all, files are php files. They were written using the program PHPStorm as well as Notepad++ whenever some quick editing was needed. All of the code is handwritten, with a couple of html snippets being repeated from page to page. An example of this would be the styling declarations.

For styling the web pages, the bootstrap libraries were used. They have convenient built in classes that offer more options to the designer in terms of styling, and allow far more flexibility than conventional fluid or fixed layouts. The images used in the project are all gotten from Google image search. I do not claim ownership of the images. For the project, HTML5 was chosen instead of the more conventional XHTML due to the extra objects that it allows to use, like for example the navigation bar.

The way the code is, it allows for easy modification and future implementations. Should more functionality be needed, there is little to no need at all to change the existing code. This is due to the high cohesion and low coupling of each file. At most, the most changes are going to be applied to the html, due to how difficult it is to keep it in the intended place after the php code changes.

# 2.1 Analysis: Functional Requirements

The functional requirements set as a goal in the beginning of the project, before the development stage, are as follows:

1. The solution must be in the form of a web page, so that everyone with internet can access it.
2. All links have to work
3. A user can create an account in order to use the website
4. The user is able to login into his account after creation
5. The user can select the items he wants and put them in his cart
6. The user can choose to remove items from the cart if he changes his mind
7. After feeling satisfied with his choices, the user can finally pay for his items, thus emptying the cart
8. User, product, cart and all info is to be stored in a database
9. The admin is a user with special priviliges. He is allowed to change the products that the company offers through a special page that only he is allowed to see.
10. The admin can view all the products without needing to access the database, as well as update or delete records
11. Input has to be verified before being submitted to the database

# 2.2 Analysis: Non-functional requirements

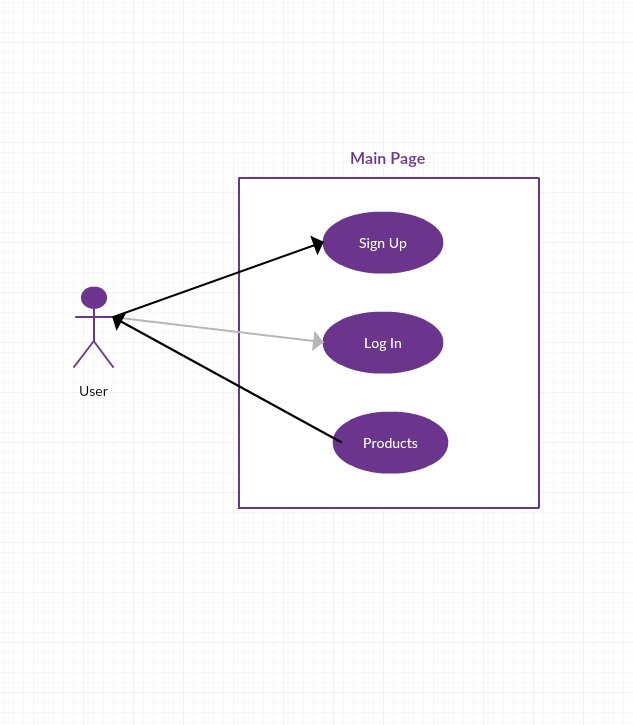
The following are a set of nonfunctional requirements:

1. Website must look professional
2. User must not have direct access to the database
3. Website must be easy to maintain, i.e. the code needs to be neat and organized with low coupling and high cohesion
4. Website must work on all main browsers (Chrome, Firefox, I.E.)

Also, there is a minimalistic constraint placed on the programmer, that forces him to make sure all files, except for the database, are on the same machine. This is to help with portability in cases where such a thing may be needed.

# 3.1 Design: Basics

When first opening the website, the user is presented with the home page, or rather the main web page of the project. From the user has some choices. He could browse the products, sign up a new account, or login into an already existing account.

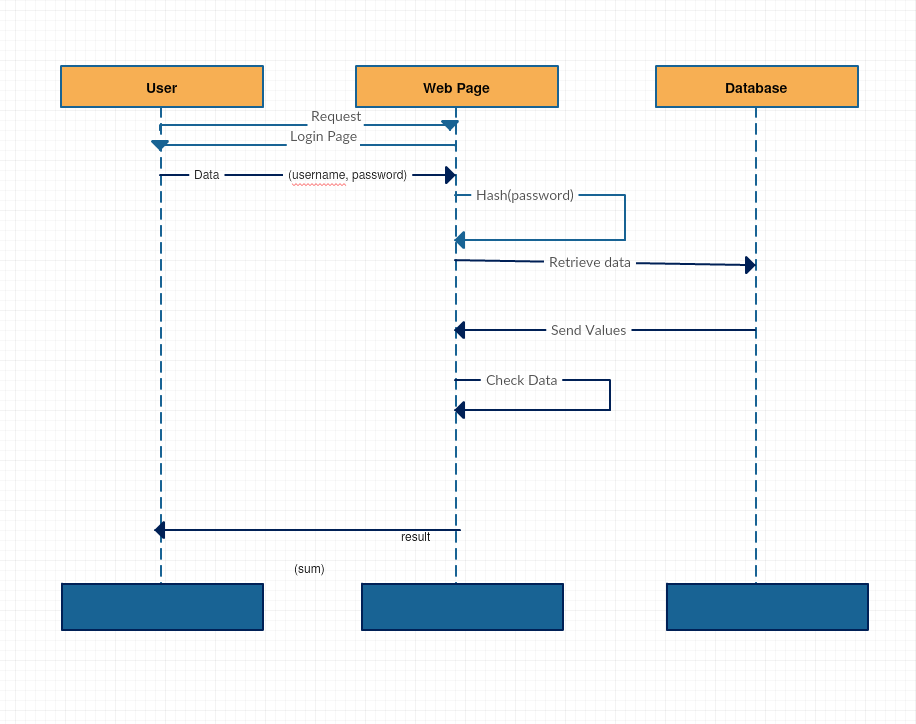


Beginning user case diagram (Use Case Diagram)

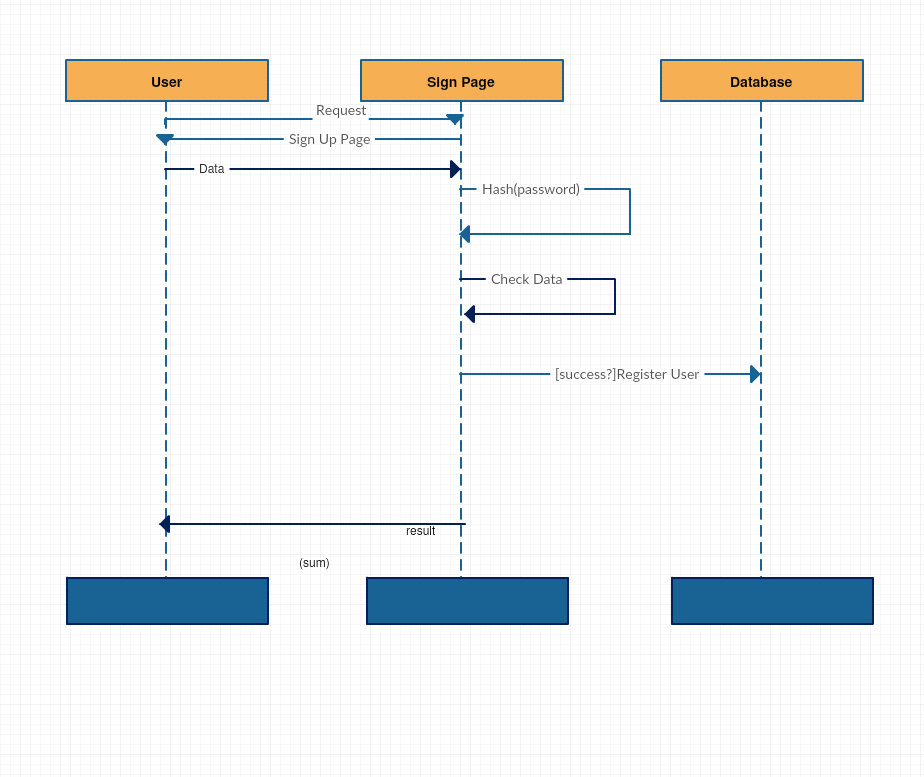
We also have a special kind of user, the admin. Since the admin account is preset, he doesn't need to sign up.



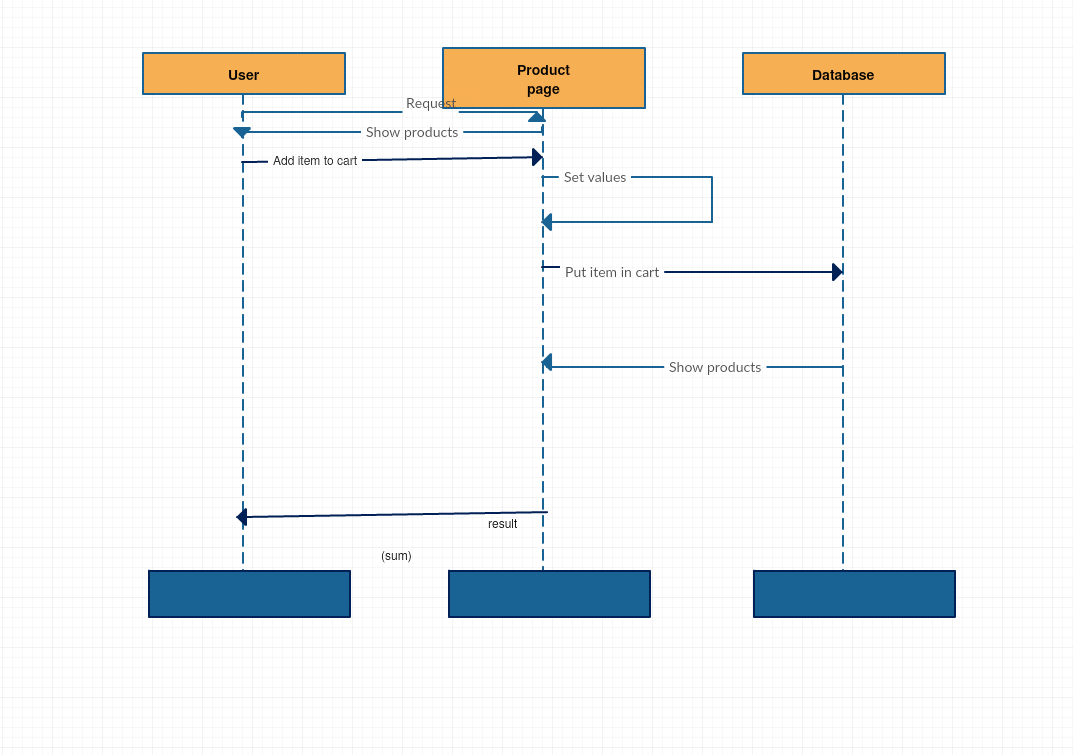
Full use case diagram at end of design stage (Use Case diagram)



Login process (Sequence Diagram)



Sign Up Process (Sequence Diagram)



User views products and adds them to cart (Sequence Diagram)

As we can see, the user never accesses the database directly, and the same thing goes for the admin. All interactions are done through the main web pages which then manipulate the database. Of course, before any actual operations are done the data needs to be checked. In the case of the login and sign up, the password needs to be hashed before being sent to the database, in order to protect users from any possible security breaches.

The web pages are connected to each other using links mostly. However, behind the scenes there's many other php files that manipulate data before redirecting the user back to the page they just were in. This happens so fast that the user doesn't even have time to realize it.



File Location/Solution Explorer

As we can see in the picture above, we have many more files than actual pages. A lot of operations call other php files to access the code inside of them. The php file manipulates the data and returns the user back to the starting page. Because all the paths are relative, as long as the file structure is kept the same. This means that the website can be moved from machine to machine as long as the database remains the same. This added flexibility really helps in cases of server failures.

# 3.2 Design: Pages In-Depth

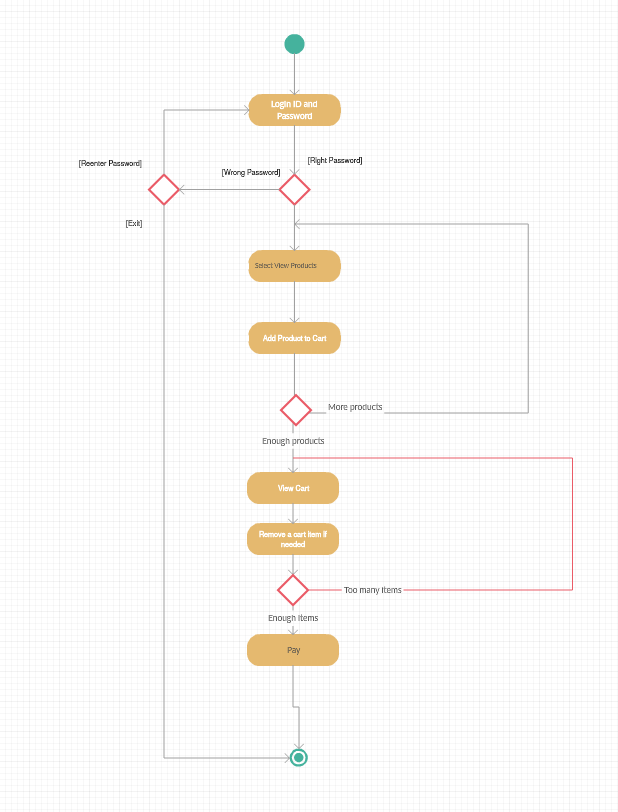
Disregarding the home page, which exists mostly due to aesthetic reasons and provides no actual noteworthy code, the main web pages that the user has access to are the Log In, Sign Up, Products and Cart.

The login.php file takes care of the login page. Its task is to access the database and check that the username the user just gave exists in the database. If it does, then it will perform another check to see if the username is admin. Like mentioned before, the admin has special privileges and gets redirected to a different page than the user. After the username check is performed, next up is the password.

Firstly, the password is hashed using sha1 inbuilt encryption in php. We do this in order to be able to compare it to the passwords in the database. All passwords in this project will use that hash code. This way, whoever sees the database cannot access the individual user accounts. Again, before an actual connection to the database is made, the information is checked for incorrect input. Leaving it in the user's hands could cause problems in the database that could undermine the website's stability. The error messages pop up using the javascript alert() function. This way the user has immediate feedback on the situation of his account. The login page only needs two text fields, the one for username and one for password. In addition to that, it has a button and a link to the new user page.

In order for the user to be able to login he needs to make an account. The only account which stays in the database is the admin account, therefore everyone else has to make their own. This is taken care of by the newuser.php page. The page uses an isset() function to make sure that this is the first time the user sees the page. In that case, the form is displayed and the user can begin to enter his data. Each of the fields has a corresponding error associated to it. If any of the inputs are incorrect, the user will be able to fix it, and the database stays safe. A function to clean up the input is used. This function trims, strips slashes and removes special characters from the input, allowing for an easy time testing it. There are two tests per field, the generic required field test and the input check test. For example, username can only be alphanumeric characters and the email needs to be of the format .

After making an account the user can browse the products and add them to his cart. The cart will keep a list of all the products that the user added. The products can be added and removed at will, whatever the user desires. Once completed, the user can click the Pay button to officially order and pay for all the items.



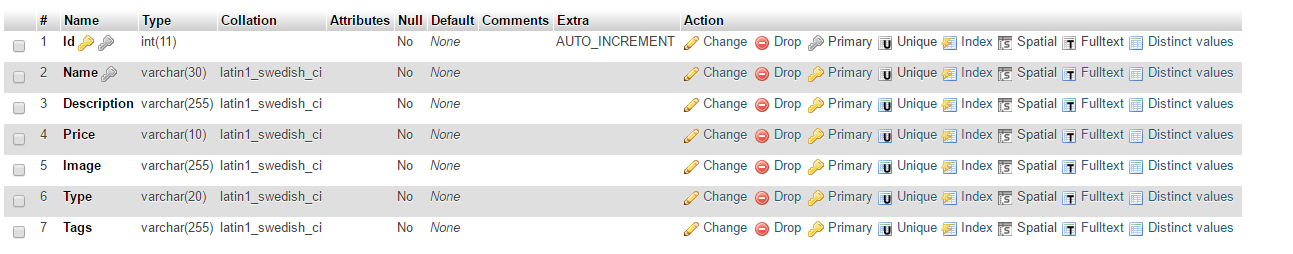
User Activity Diagram

# 3.3 Design: Database

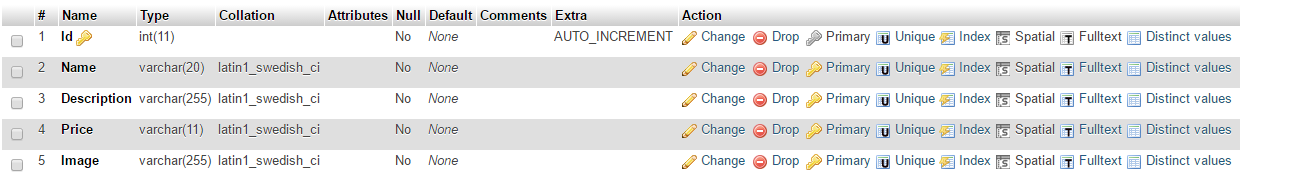
The database holds 3 main tables, as well as some auxiliary tables necessary for extra features that would've been added had there been more time. The main tables are User, Cart, Product. The users table hold the records of all users that currently have accounts in the database. Its primary key is ID, with Username being a Unique attribute. That is because only one username can be for each account. It also has attributes for password, which is a hashed value of the real password, as well as email to be used for password changes in the future.

The products are all held in the database as well. Currently they have attributes such as ID, Image, Name, Description and Price. Those are the main attributes, but there are also Type and Tags. The last two are used by the admin to search for products on his control panel.

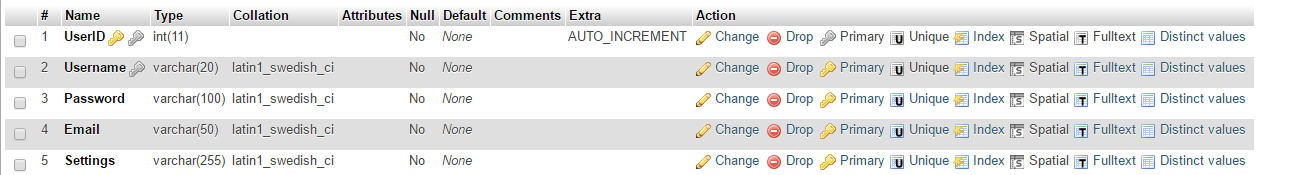
The user picks his desired products and then they are added to the cart. The cart holds all the items that the user wants to buy. They are linked to the products by the Name. The cart is almost similar to the products, except for the User field. The user field holds the value of the username of the user that owns the current cart. As long as the user doesn't pay, his cart will stay. The user can decide to remove or add items as he sees fit, but once he is done and clicks the pay button, all items to his name will be removed and the transaction considered complete.



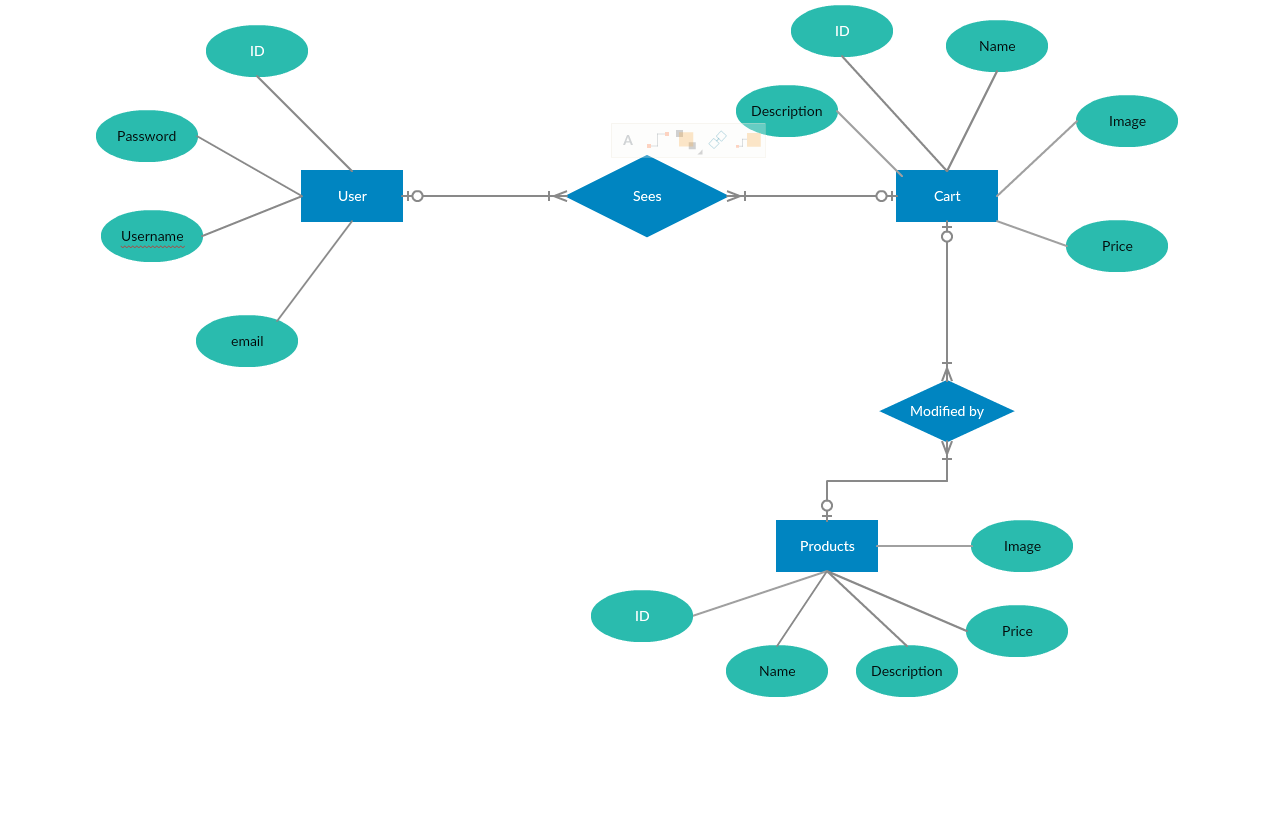
Product table



Cart Table



Users Table

****

Database Diagram

# 4.1 Implementation

The language used was PHP. In the beginning of the project, I opted to use ASP.Net due to how seemingly easy it seemed at first glance. However, in the beginning of the development process it was clear that my master of this language was lacking. Simply put, I got stuck on the database implementation. Therefore, one month into the senior project I had to start over completely from scratch and begin using PHP.

The pros of using PHP revolve simply around the fact that there are more options. However, this is easily offset by the fact that php and html in the same file are very difficult to balance. This is why, in order for the website to look alright I resorted to using bootstrap. Bootstrap is a set of css libraries and javascript scripts that make handling html items much easier. The problems with the alternative methods, which is using a fluid or fixed layout, is that they only work in either one situation or the other. Fluid for example would look awful on mobile.

Not only does bootstrap provide more control, but it also provides some neat features that go very well with HTML5, as well as extra classes for containers.

<?php

//login.php

$database = "teat";

$user = "root";

$pass = "";

if (isset($\_POST['username'])){

$username = $\_POST['username'];

$password = $\_POST['password'];

$tName = "users";

$connection = new mysqli("localhost", $user, $pass, $database);

if ($connection->connect\_error) {

echo "<p>Could not connect to the " .

"database: " . $connection->connect\_error . "</p>\n";

exit;

}

$sql = "SELECT Username, Password FROM $tName

WHERE Username='$username'";

$result = $connection->query($sql);

if(empty($result)) {

echo '<script language="javascript">';

echo 'alert("Username or Password is incorrect.")';

echo '</script>';

}

else {

$row = $result->fetch\_row();

$password = sha1($password);

if ($username == "admin" && $password == $row[1]) {

header("Location: admin.php");

die();

}

else if ($password == $row[1]) {

header("Location: testUIforTEAT.html");

die();

}

else {

echo '<script language="javascript">';

echo 'alert("Username or Password is incorrect.")';

echo '</script>';

}

}

}

?>

<?php

//newuser.php

if (isset($\_POST['Submit'])) {

$user = $\_POST["user"];

$pass = $\_POST["pass"];

$pass2 = $\_POST["pass2"];

$email = $\_POST["email"];

$errorCount = 0;

$userError = "";

$password1Error = "";

$password2Error = "";

$emailError = "";

//Function to trim and clean data. Used for testing later

function test\_input($data)

{

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

return $data;

}

if (empty($\_POST["user"])) {

$userError = "Username is required";

$errorCount++;

} else {

$user = test\_input(($\_POST["user"]));

// check name only contains letters and whitespace

if (!preg\_match("/^[a-zA-Z' -]\*$/", $user)) {

$userError = "Only letters and white space allowed";

$errorCount++;

}

}

if (empty($pass)) {

$password1Error = "Password is required";

$errorCount++;

}

if (empty($pass2)) {

$password2Error = "Repeated Password is required";

$errorCount++;

}

if ($pass != $pass2) {

$password2Error = "Passwords don't match";

$errorCount++;

}

if (empty($\_POST["email"])) {

$emailError = "Email is required";

$errorCount++;

} else {

$email = test\_input($\_POST["email"]);

// check if e-mail address syntax is valid or not

if (!preg\_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/", $email)) {

$emailError = "Invalid email format";

$errorCount++;

}

}

if ($errorCount > 0) {

echo '<script language="javascript">';

?>alert("The following errors occured:\n" \* <?php echo($userError); ?>); <?php

echo '</script>';

}

else {

// Database values

$database = "teat"; // WAMP database was used

$userdb = "root";

$passdb = "";

// Try to connect

$conn = new mysqli("localhost", $userdb, $passdb, $database);

if ($conn->connect\_error) {

echo "<p>Could not connect to the $database " .

"database: " . $conn->connect\_error . "</p>\n";

exit;

}

$tName = "users";

$pass = sha1($pass);

$sql = "INSERT INTO $tName (Username, Password, Email, Settings)

VALUES ('$user', '$pass', '$email', 'read')";

if ($result = $conn->query($sql)) {

$conn->close();

header("Location: login.php");

die();

} else {

$conn->close();

echo "<p>Unable to complete sign up process.</p>";

}

}

}

?>

<?php

//teatproducts.php

//html formatting of database items

while($row = $result->fetch\_row()) {

?> <div class="col-sm-4 col-md-4 col-lg-4 col-xs-6">

<div class="thumbnail"> <img src="<?php echo $row[0]; ?>" alt="Thumbnail Image 1" class="img-responsive imgitem">

<div class="caption">

<h3><?php echo $row[1]; ?></h3>

<p>Price: $<?php echo $row[2]; ?></p>

<p><a href="teataddcart.php?id=<?php echo $row[3];?>&user=buck" class="btn btn-primary" role="button"><span class="glyphicon glyphicon-shopping-cart" aria-hidden="true"></span> Add to Cart</a></p>

</div>

</div>

</div>

<?php

}

?>

<?php

//admin.php

// Set variables used to access the database. In this case, WAMP was used with phpMyAdmin.

$database = "teat";

$user = "root";

$pass = "";

// Attempt connection, if error print the error out.

$connection = new mysqli("localhost", $user, $pass, $database);

if ($connection->connect\_error) {

echo "<p>Could not connect to the " .

"database: " . $connection->connect\_error . "</p>\n";

exit;

}

//Set value of name of the table.

$tName = "product";

if (!isset($\_GET['name']) || ($\_GET['name'] == "")) {

//Select all records from the table.

$sql = "SELECT \* FROM $tName";

$result = $connection->query($sql);

echo "<table border='1'>"; //Echoes table so that it is used by html

for ($i=0; $i < $result->field\_count; $i++){

echo "<th>"; //Same thing with the header of the table

$fields = $result->fetch\_field();

echo $fields->name;

echo"</th>";

}

echo "<th>";

echo "";

echo"</th>";

//Prints records

$i = $result->field\_count;

while($row = $result->fetch\_row()) {

echo "<tr>";

for ($j=0; $j<$i; $j++) {

if ($j == 4) {

echo "<td>";

?> <img src="<?php echo $row[4];?>" alt="Image" height="42" width="42"><?php

echo "</td>";

}

else {

echo "<td>";

echo $row[$j];

echo "</td>";

}

}

echo "<td><a href='admindelete.php?id=" . $row[0] . "'>Delete</a>&nbsp;<a href='adminupdate.php?id=" . $row[0] . "'>Update</a></td>";

echo "</tr>";

}

echo "</table>";

?>

<?php

//admindelete.php

$id = $\_GET['id'];

// Database values

$database = "teat"; // WAMP was used for the database

$user = "root";

$pass = "";

// Try to connect

$conn = new mysqli("localhost", $user, $pass, $database);

if ($conn->connect\_error) {

echo "<p>Could not connect to the $database " .

"database: " . $conn->connect\_error . "</p>\n";

exit;

}

$tName = "product";

// Add a DELETE SQL command using data sent by user

$sql = "DELETE FROM $tName

WHERE Id=$id";

if ($result = $conn->query($sql)) {

$conn->close();

header("Location: admin.php");

die();

} else {

$conn->close();

echo "<p>Unable to delete the record</p>";

echo "<p><a href='admin.php'>Go Back</a> </p>";

}

?>

<?php

//admininsert.php

if (isset($\_POST['Submit'])) {

$user = $\_POST["user"];

$pass = $\_POST["pass"];

$pass2 = $\_POST["pass2"];

$email = $\_POST["email"];

$errorCount = 0;

$userError = "";

$password1Error = "";

$password2Error = "";

$emailError = "";

//Function to trim and clean data. Used for testing later

function test\_input($data)

{

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

return $data;

}

if (empty($\_POST["user"])) {

$userError = "Username is required";

$errorCount++;

} else {

$user = test\_input(($\_POST["user"]));

// check name only contains letters and whitespace

if (!preg\_match("/^[a-zA-Z' -]\*$/", $user)) {

$userError = "Only letters and white space allowed";

$errorCount++;

}

}

if (empty($pass)) {

$password1Error = "Password is required";

$errorCount++;

}

if (empty($pass2)) {

$password2Error = "Repeated Password is required";

$errorCount++;

}

if ($pass != $pass2) {

$password2Error = "Passwords don't match";

$errorCount++;

}

if (empty($\_POST["email"])) {

$emailError = "Email is required";

$errorCount++;

} else {

$email = test\_input($\_POST["email"]);

// check if e-mail address syntax is valid or not

if (!preg\_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/", $email)) {

$emailError = "Invalid email format";

$errorCount++;

}

}

if ($errorCount > 0) {

echo '<script language="javascript">';

?>alert("The following errors occured:\n" \* <?php echo($userError); ?>); <?php

echo '</script>';

}

else {

// Database values

$database = "teat"; // WAMP database was used

$userdb = "root";

$passdb = "";

// Try to connect

$conn = new mysqli("localhost", $userdb, $passdb, $database);

if ($conn->connect\_error) {

echo "<p>Could not connect to the $database " .

"database: " . $conn->connect\_error . "</p>\n";

exit;

}

$tName = "users";

$pass = sha1($pass);

$sql = "INSERT INTO $tName (Username, Password, Email, Settings)

VALUES ('$user', '$pass', '$email', 'read')";

if ($result = $conn->query($sql)) {

$conn->close();

header("Location: login.php");

die();

} else {

$conn->close();

echo "<p>Unable to insert the record.</p>";

}

}

}

?>

<?php

//teatcart.php

<?php

while($row = $result->fetch\_row()) {

?> <div class="col-sm-4 col-md-4 col-lg-4 col-xs-6">

<div class="thumbnail"> <img src="<?php echo $row[0]; ?>" alt="Thumbnail Image 1" class="img-responsive imgitem">

<div class="caption">

<h3><?php echo $row[1]; ?></h3>

<p>Price: $<?php echo $row[2]; ?></p>

<p><a href="teatremovecart.php?id=<?php echo $row[3]; ?>" class="btn btn-primary" role="button"><span class="glyphicon glyphicon-shopping-cart" aria-hidden="true"></span> Remove from Cart</a></p>

</div>

</div>

</div>

<?php

}?>

<?php

//teataddcart.php

//Set value of name of the table.

$tName = "product";

$t2Name = "cart";

$sql = "INSERT INTO $t2Name (Name, Description, Price, Image)

SELECT Name, Description, Price, Image

FROM $tName

WHERE Id=$id";

$connection->query($sql);

$connection->close();

header("Location: teatproducts.php");

die(); ?>

<?php

//removecart.php

$id = $\_GET['id'];

// Add your database name (username), username and password

$database = "teat"; // The database name is your username

$user = "root";

$pass = "";

// Try to connect

$conn = new mysqli("localhost", $user, $pass, $database);

if ($conn->connect\_error) {

echo "<p>Could not connect to the $database " .

"database: " . $conn->connect\_error . "</p>\n";

exit;

}

$tName = "cart";

// Add an INSERT SQL command using data sent by user

$sql = "DELETE FROM $tName

WHERE Id=$id";

$conn->query($sql);

$conn->close();

header("Location: teatcart.php");

die();

?>

<?php

//pay.php

$id = $\_GET['id'];

// Add your database name (username), username and password

$database = "teat"; // The database name is your username

$user = "root";

$pass = "";

// Try to connect

$conn = new mysqli("localhost", $user, $pass, $database);

if ($conn->connect\_error) {

echo "<p>Could not connect to the $database " .

"database: " . $conn->connect\_error . "</p>\n";

exit;

}

$tName = "cart";

// Add an INSERT SQL command using data sent by user

$sql = "DELETE FROM $tName";

$conn->query($sql);

$conn->close();

header("Location: teatcart.php");

die();

?>

# 5.1 Results and Conclusion

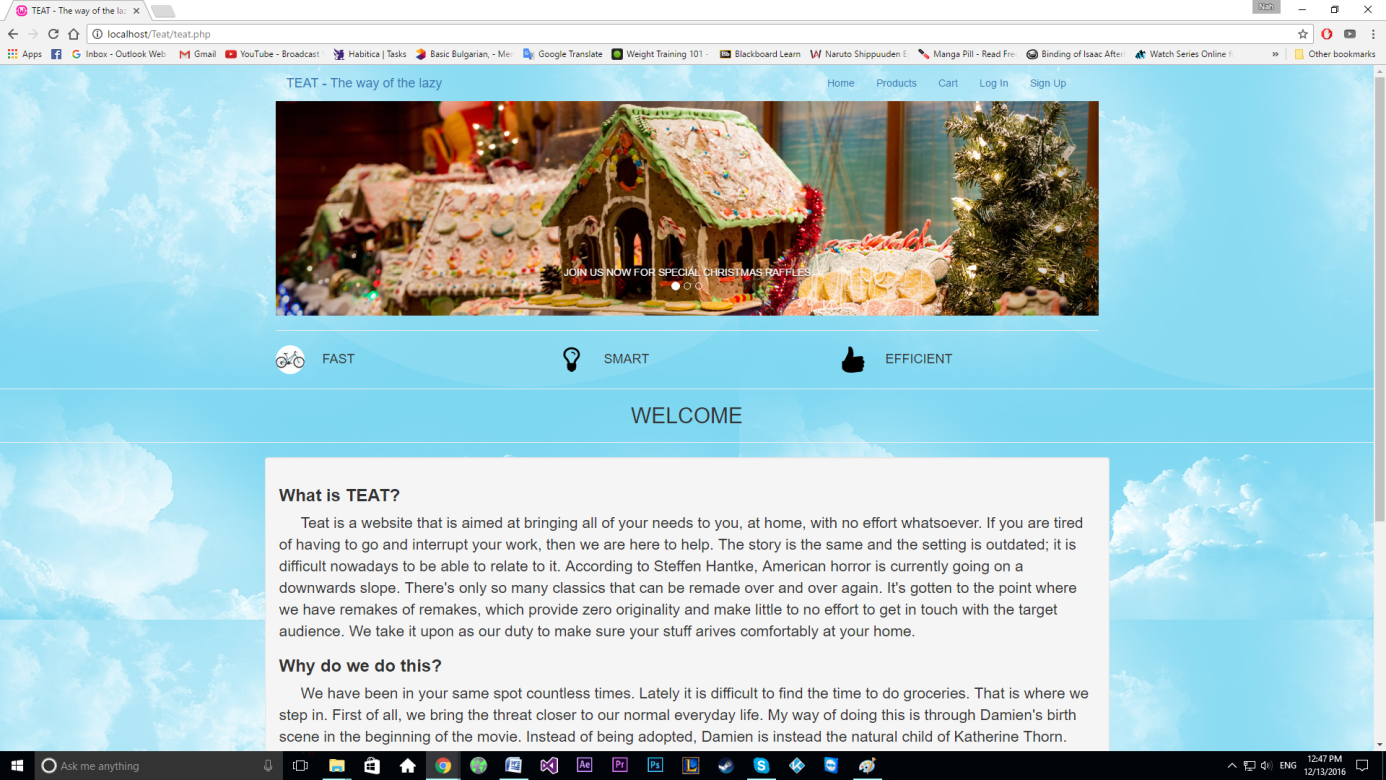
All in all, the webpage came out looking decent, with a lot of functionalities that an average site should have. If I had not foolishly wasted time with ASP.Net, there could have been even more features that would make it actually competitive in the market.

However, all the initial features were implemented in time, and the website works well enough. The biggest difficulty in making this project was the HTML part of it. Simply put, HTML is difficult to control especially due to the inclusion of the database. Formatting it at first was a challenge and required a lot of knowledge and internet searching.

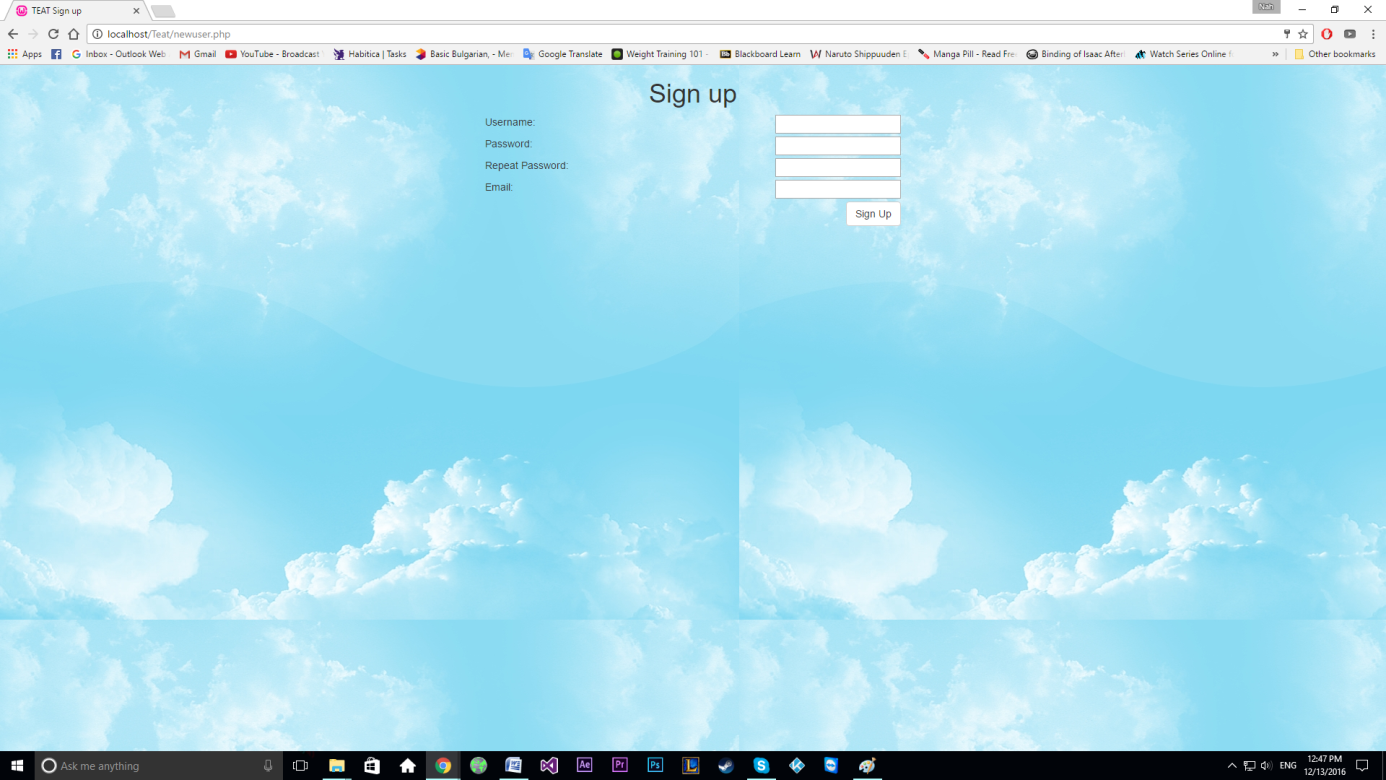
What I could have done better might have been a more distinct layered architecture. The current one is visible, but the layers are not placed in different folders. If I had more time, I would simply refactor a lot of the code as well.

In the end, my knowledge of both HTML and PHP increased. Despite not having taken a PHP course before, it somehow worked out.

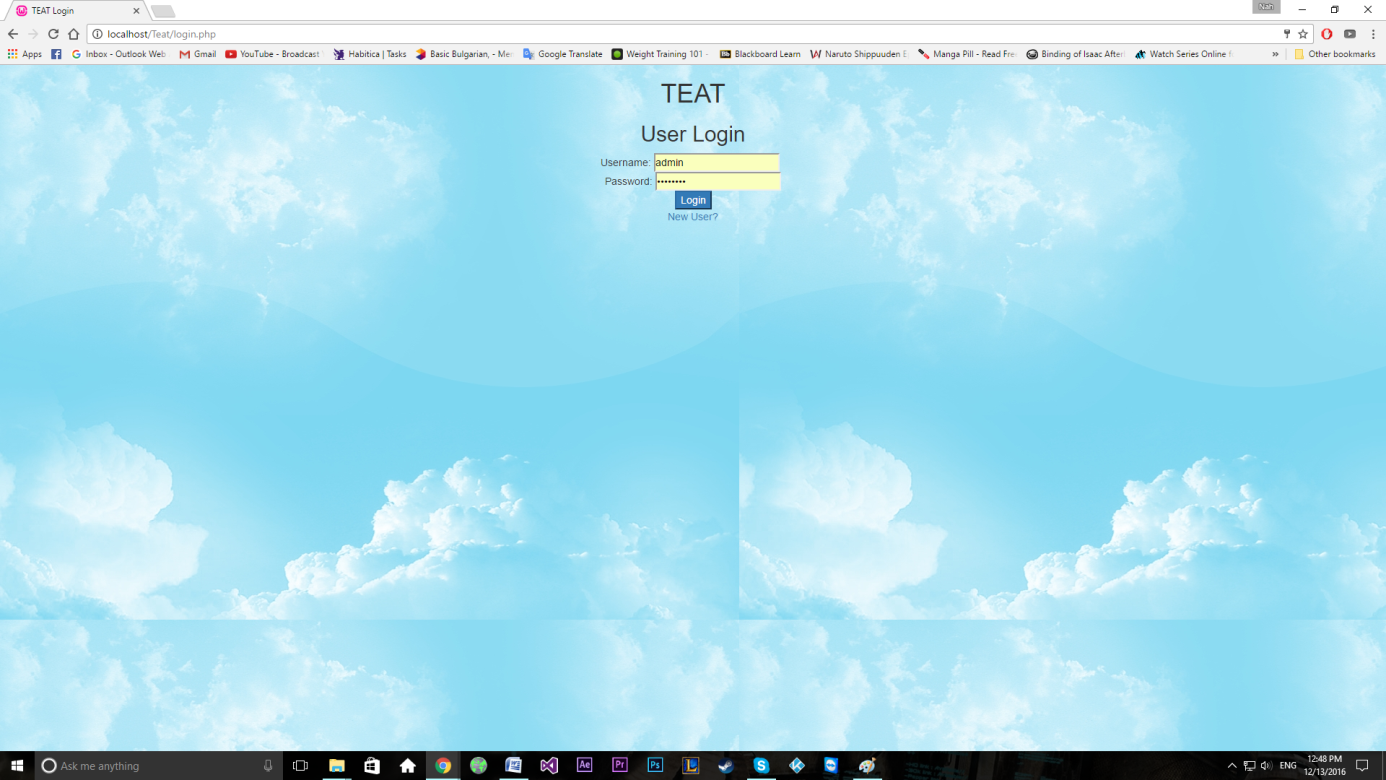
**5.2 Website Samples**



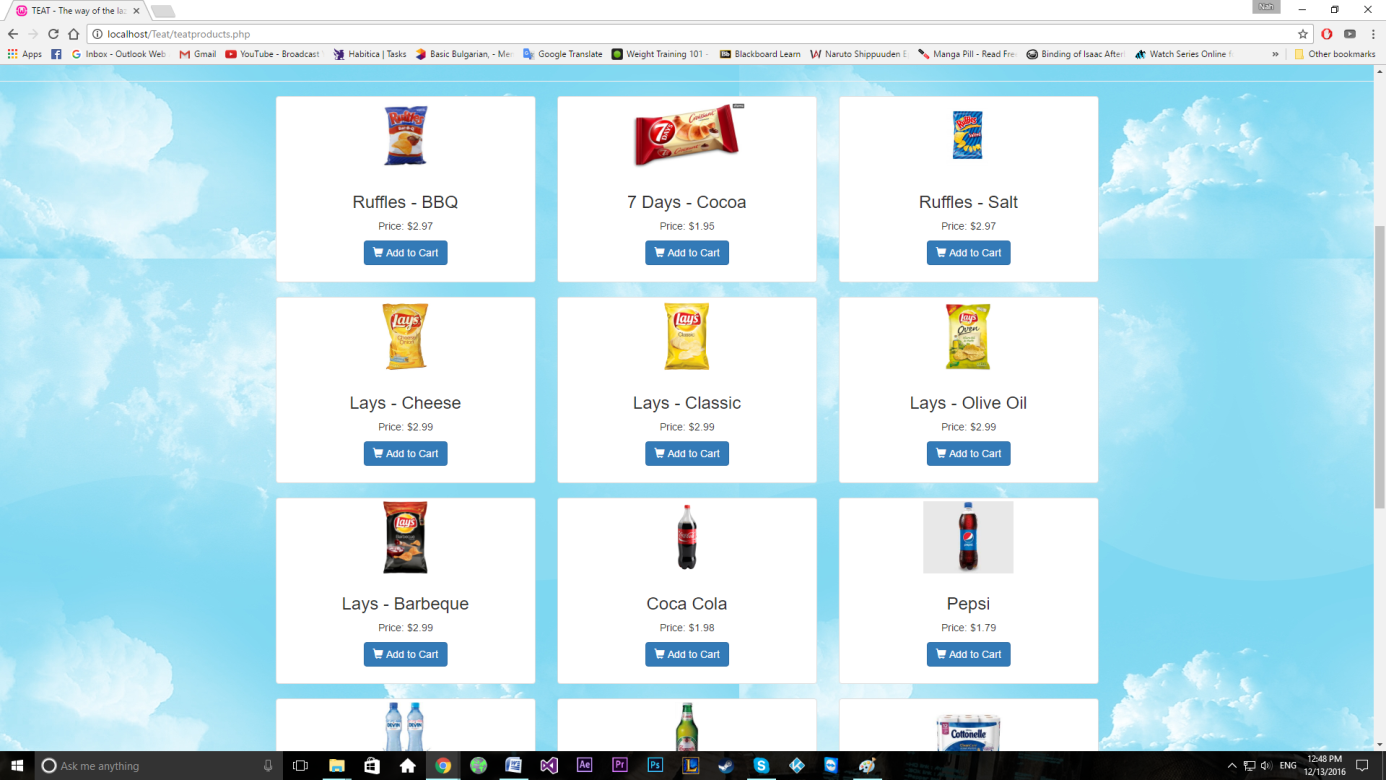
Home page

****

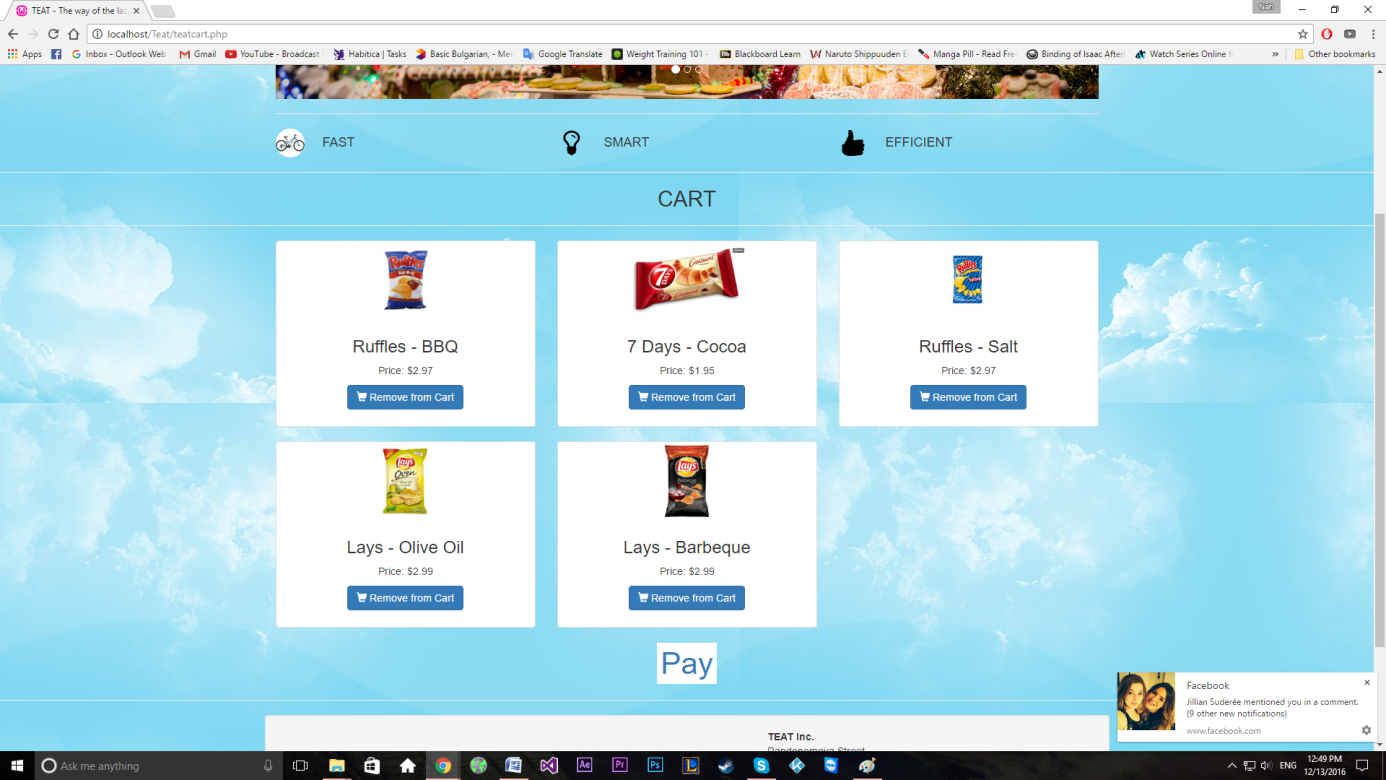
Sign Up Page



Login Page



Products



Cart

# 

# 6.1 References

* W3Schools for information about layouts and bootstrap
* INF335 for PHP knowledge
* Google for Images