|  |
| --- |
|  |
| Dynamic Database Driven Website for Local Art Company |
|  |
|  |

**HIT326 Database driven web applications**

2020

By: Alexander Lay

S253460

Dynamic Database Driven Website for Local Art Company

Table of Contents

[Overview 2](#_Toc42185252)

[About the project 2](#_Toc42185253)

[Business Requirements 2](#_Toc42185254)

[Project Approach 3](#_Toc42185255)

[Strategy 4](#_Toc42185256)

[The prototype and features 4](#_Toc42185257)

[Installation instructions 5](#_Toc42185258)

[Uninstalling 5](#_Toc42185259)

[Database Tests 6](#_Toc42185260)

[Application Tests 6](#_Toc42185261)

[Working code description 10](#_Toc42185262)

[Application.php 11](#_Toc42185263)

[Index.php 16](#_Toc42185264)

[Model.php 21](#_Toc42185265)

[Views 27](#_Toc42185266)

[Security 31](#_Toc42185267)

[Licensing and copyright 31](#_Toc42185268)

[Lessons Learned 32](#_Toc42185269)

[Conclusion 32](#_Toc42185270)

[References 34](#_Toc42185271)

[Appendices 34](#_Toc42185272)

[Appendix A 35](#_Toc42185273)

[Appendix B 36](#_Toc42185274)

# Overview

### About the project

A small local art company currently uses a static website which primarily uses text and images to display their products for sale. There is currently no transaction system and/or a system to create and place an order. This causes a problem as this system is slow and unorthodox in today’s standard of online shopping. Purchase transactions currently occur outside their current website whether it be in person or another third party website (like eBay). Customers do not want to go through the hassle of a slow and frustrating process. This is a huge issue in their business model and could deter potential customers from purchasing and hence loss of profits. According to Sleeknote 2020, 75% of people will buy something online at least once a month. This shows the increasing popularity of online shopping and the importance of a dynamic website for any company wishing to do business online.

The company would like it so that a customer can view a product on their website and place an order for that particular product(s). An automated system then sends an email to the company with the order to be fulfilled. The company has included a set amount of key functionalities which they would like their website to perform which are described below.

The overall goal for this project is to create a dynamic website to help the local art company potentially improve business.

### Business Requirements

All the business requirements are as below:

* A customer should be able to buy/place an order from the website using a shopping cart system
* When a customer completes a purchase, a record of the order is stored in a database.
* A copy of the order details should be sent to the customer after purchase.
* A copy of the order details should be sent to the company after purchase for fulfilment.

The Business Rules are outlined as follows:

* + Customer is always identified by email address.
  + Only one phone number is held for a customer.
  + A customer can make many purchases and a purchase generally will contain more than one product.
  + Each new purchase is identified by an incrementing number.
  + There is no need to keep track of how many products are in stock as products can be made when the order is received.
  + If a product is no longer available, then it will be removed from the front end of the website.

Some additional features the company would like are:

* The ability for the company to post items on the home page, where only recent items are shown.
* Add customer testimonial functionalities, where a customer can leave a review on a product they have purchased.
* Add security and mobile compatibility, so that the website is secure from cyber-attacks and it can be viewed from a mobile device.

Considering these business requirements, it was not feasible to implement all these functionalities. Therefore, only the functionalities which are deemed most important have been implemented. These inclusions and exclusions are listed below.

#### Inclusions

* Establish database.
* Create appropriate tables.
* Customer sign-up and sign-in functionalities.
* Display all products which are currently for sale.
* Shopping cart system.
* Store order in database

#### Exclusions

* Add a customer review section.
* A home page with the most recent product.
* If a product is no longer available, then it will be removed from the front end of the website.
* An automated email sent to customer and business owner.
* Mobile compatibility.

# Project Approach

The project has been completed by utilizing the popular WAMP (Windows, Apache, MySQL, PHP) stack technology. These technologies have been proven to be very efficient and effective in creating small projects that require a database and dynamic interactivity. The WAMP technology is best described by a Model, View, Controller (MVC) structure. Ideally a team member would work on a separate functionality individually. This would speed up the process, as each team member can focus on that specific part which they may also excel in. Although for this case, the project has been undertaken individually by myself, where I have completed all parts of the project.

REST stands for Representational State Transfer. REST is a web based architecture which implements/responds to CRUD (Create, Read, Update, Delete) operations between the client and server through HTTP requests. REST is stateless, which means the server does not store any information about the client’s requests. REST has become increasingly popular because it is easy to create and manage because of the separation of client and server services. This also means that changes made on the client side does not affect servers side code.

The project follows the rules of REST because all GET requests are not stored and all session data is destroyed when a customer is logged out. The project also has CRUD capabilities these are as follows:

**Create:** A customer can create an account. Anyone can also add a product for sale.

**Read:** Anyone can view all the products which are stored in the database. A customer can view all their details from the “my account” page.

**Update:** A customer can update their details of their account (only using SQL scripts at this time).

**Delete:** A customer can delete their account (only using SQL scripts at this time).

The project code follows the MVC structure which is later described in detail.

# Strategy

As mentioned before, the project utilizes a MVC pattern to help organize the logic of the code into more readable and practical use. This is done in order to DRY (Don’t Repeat Yourself) up the code for reusability. Code/functions are stored in the “model.php” file where code/files that display certain logic are stored in the views folder. These MVC functionalities are separated as such:

Model: model.php

View: views folder

Controller: index.php

# The prototype and features

The project has not been deployed to any sort of server as per request. To use the project application, it must be setup locally on a personal machine.

The project application has not been considered for mobile compatibility as it was a feature that was excluded. For future purposes it would be optimal that the application was to work flawlessly on mobile as the majority of users are on mobile according to an article by Broadbandsearch 2020.

The features which have been implemented are:

* sign up
* sign in
* sign out
* add product
* display products for sale
* add products to cart
* place an order
* store the order in the database

To see more details on how these features work, there are instructions that detail how they should work further in the report.

# Installation instructions

The project was completed with:

* PHP version 7.3.12
* Apache version 2.4.41
* MySQL version 5.0.12
* Windows 10 home edition

This was checked by using the in-built phpinfo() function and launching it on a localhost.

To install/run the application you need to install WAMP (check appendix for website). We then need to configure some files within the apache configuration files. The steps to setup the project are as below:

1. Download and install WAMP from their website (see appendix).
2. Setup httpd-vhost.config to same path as where the project application is stored.
3. Start all services of the WAMP server
4. Login to phpmyadmin from a browser and create a database called “art\_db”.

(user name: root, password: hit325)(localhost:3308)

1. Run the create.sql file which will create all tables (found at /lib/sql/)
2. Run the load.sql file which will load some data into the tables for later use (found at /lib/sql/).
3. Type “localhost” into the address bar in your browser.
4. The application is ready to use.

# Uninstalling

To uninstall the project just simply,

1. Delete the database to delete all tables associated within phpmyadmin.
2. Stop all WAMP server services.
3. Uninstall all related WAMP program files.

# Database Tests

In order to test the database for CRUD operations, there are various scripts included which will test its validity. These scripts contain comments within them explaining what each script does.

Firstly, the create.sql should be executed which will create all the tables (/lib/sql/). The test scripts are found within the “sql test scripts” folder (/lib/sql/sql test scripts).

The create.sql script has been created to create the associated tables in the database.

The load.sql script will load some data into the database.

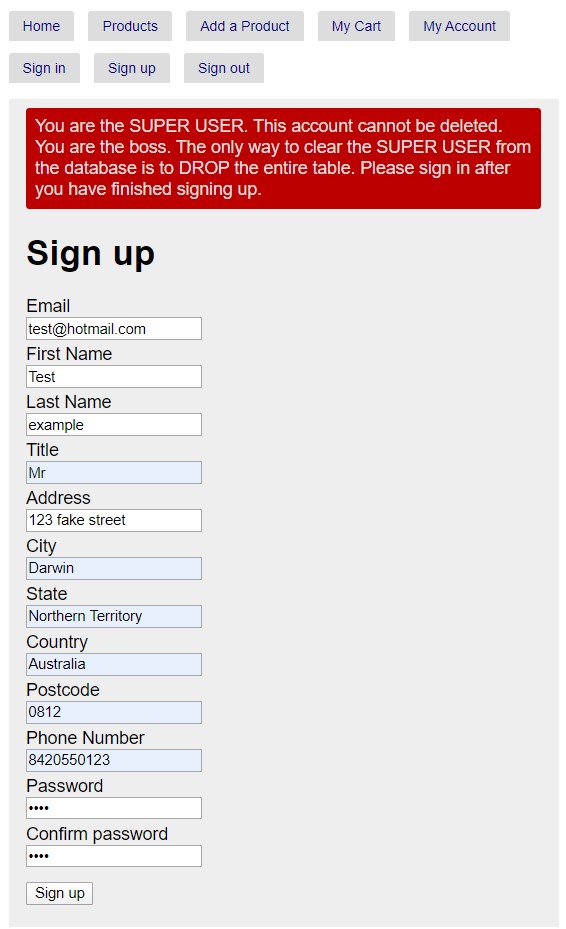
The update.sql script will update some of the values which have been previously entered.

The delete.sql script will delete some of the data which have been previously entered.

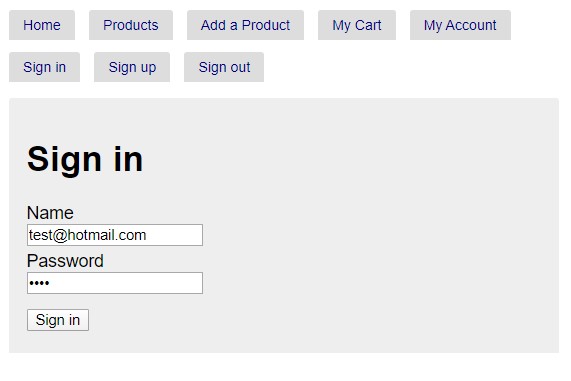
# Application Tests

To verify the project application is setup and working correctly, there are some tests which should be carried out to verify all features are functioning as they should. The accompanying screen captures show what the expected outcome is to be.

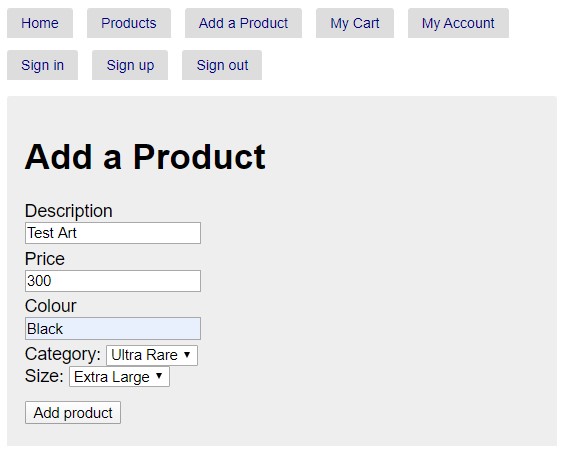
1. Create an account (sign up) by clicking on the sign up button and enter your necessary details. After you have entered your details click sign up on the bottom.



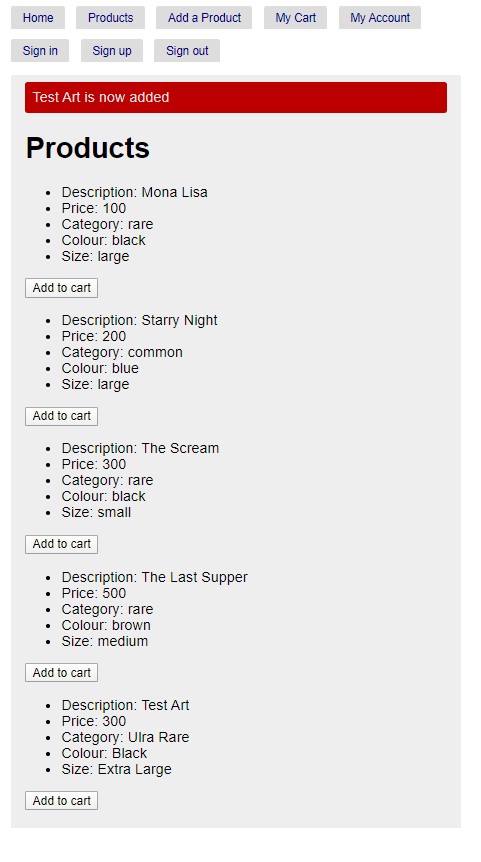
1. Sign in with the account you just created by clicking on the sign in button.



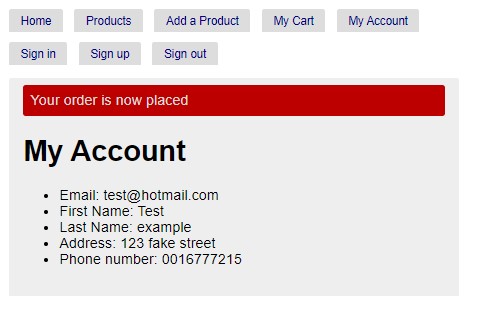
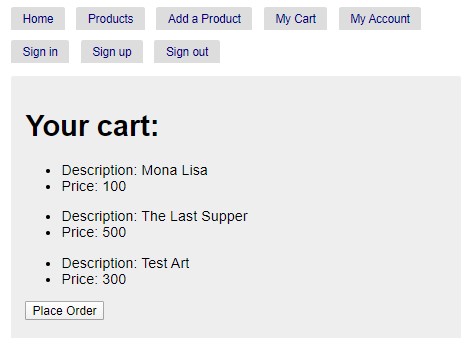
1. Add a product to the website for sale by clicking the add product button. Click add product when complete.



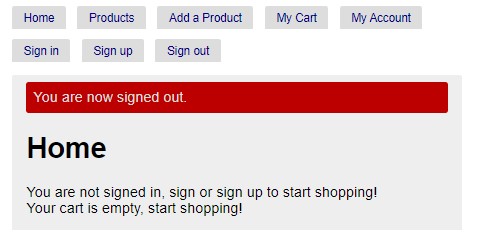
1. Go to the products page and add as many products you like to your cart.



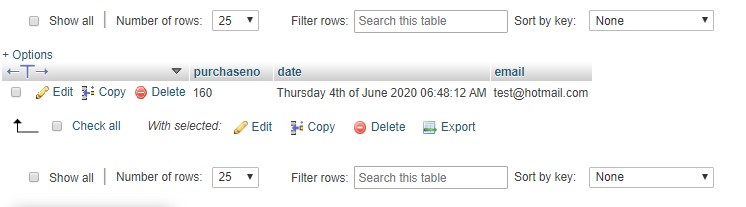
1. Go to the cart page and click “place order”



1. Sign out of your account.



1. Check phpmyadmin if orders have been correctly stored in the database.



# 

# Working code description

The project application contains many functionalities which in-turn work together to produce the final product. These functionalities are written in PHP, HTML, CSS and database queries are written in SQL. The project used a combination of MYSQL, prepared statements and PDO.

With all the files working together, they must use a database and server such as MYSQL database and Apache server. Although the project is not deployed online, it utilizes a virtual environment when run locally.

With the document ROOT in mind, the project application folder structure can be viewed as the following:

* css (folder)
* lib (folder)
* .htaccess
* index.php

**CSS Folder**

Cascading Style Sheets (CSS), is used to style particular pages. CSS is most commonly used with HTML and each HTML file must reference a CSS file in order to use its styling properties. In this application this is done a little differently where the CSS file is referenced from the index.php file. This is done to stop repeat code and creates a more cohesive project. This could also be seen as a way to reduce file size by not repeating the same code.

This folder contains the standard CSS file called “standard.css”. This file contains all the necessary code to style the whole application. If certain pages or elements within the views require a different set of CSS styling, another CSS file can be created and referenced accordingly.

**Lib Folder**

The lib folder when opened contains an “sql”and “views” folder along with files called application.php and model.php. This folder contains the majority of the applications code.

**Views Folder**

All files within the views folder act as the “views” as part of the MVC pattern. Within the “views” folder contains all the php files which function as the “views” of the application. The purpose of these files is to display necessary information on the website. As per the MVC pattern, the views are controlled/dependent on the model. The code within model.php will ultimately determine what is displayed with the views files.

The folder also contains a partials folder. This contains small pieces of code (usually 1-2 lines) which can be used in the main views files when referenced accordingly. Again this is to reduce repeat code.

**SQL Folder**

This folder contains all the SQL scripts which create the associated tables for the database. The folder also contains the SQL scripts which will load data into the tables for initial use. The SQL files are:

* create.sql
* load.sql
* sql test scripts (folder)

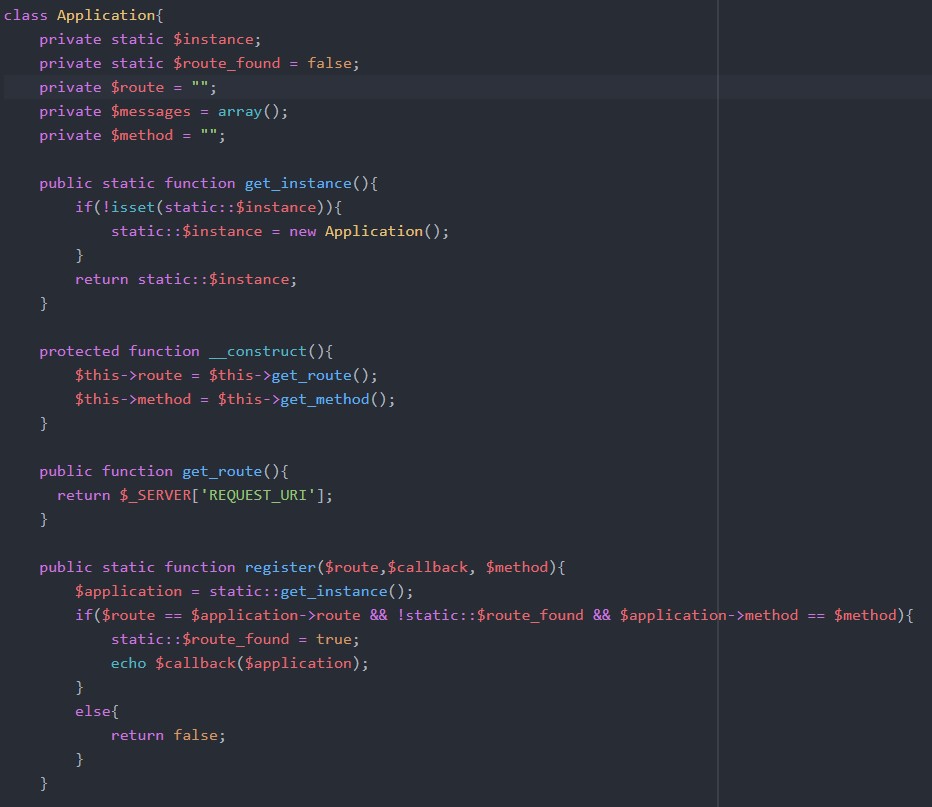
### Application.php

This php file contains all the essential functions which are used across the index.php and model.php files.

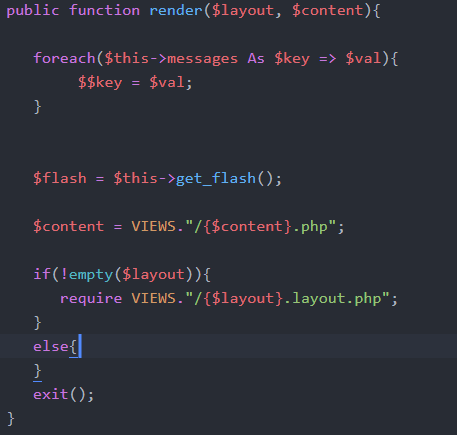
The following functions define the essential request that follow CRUD. If none of the routes match, then a 404 error would occur.



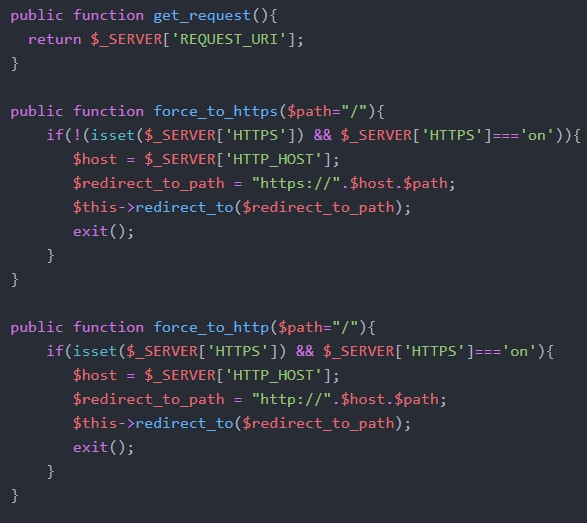
The following code defines the application class, which is used throughout the index.php file.



The render function handles what will be displayed as well as what CSS file it will use. $layout and $content are parameters which are given within the index.php file.



These functions are used to direct a page to either use HTTP or HTTPS.



This get\_method() handles various types of form requests when “\_method” is referenced to in the HTML/php form. The redirect function redirects the specified page when called.



There is a function which is used to set a session message and a function to retrieve that message for later use with in the view pages. The flash message is a “temporary” message which is displayed when a certain action is complete. These actions are usually POST requests.



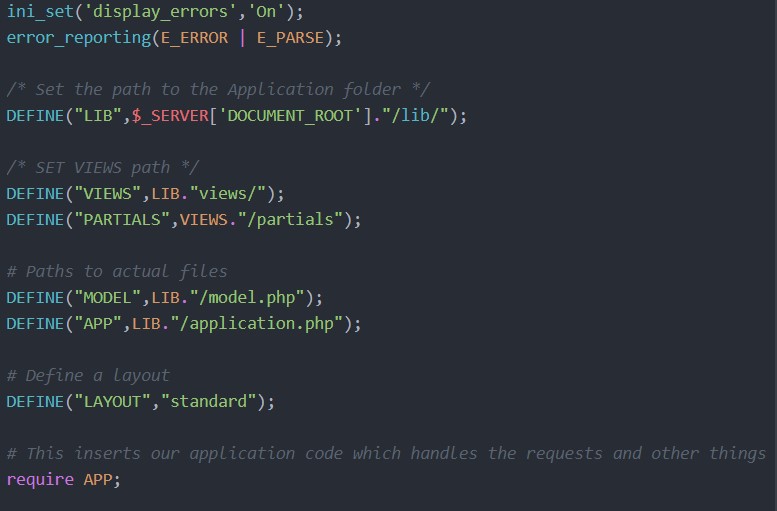
The resolve function displays a 404 error if the $route\_found is not found.



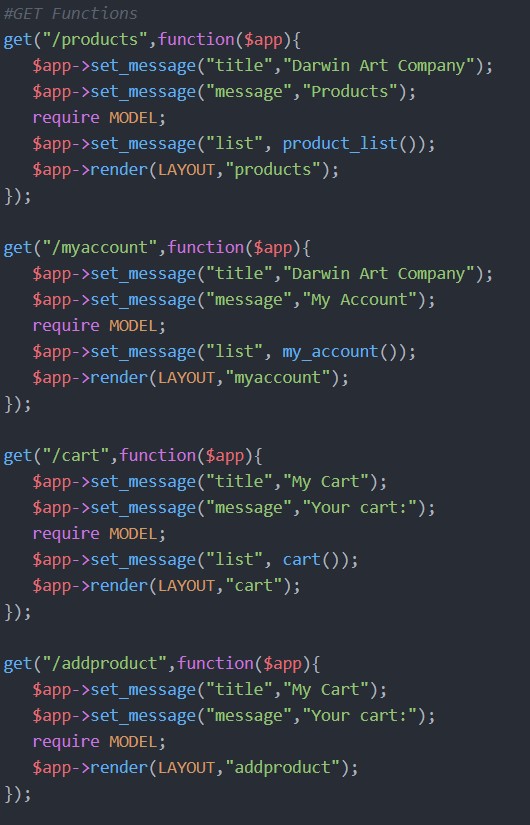
### Index.php

The index.php file acts as a controller for the whole application in terms of the MVC pattern. The index.php contains many lines of code which manipulates the model. The index.php file is explained as below:

At the start of the file are all the necessary paths. These paths can be referenced when used in other functions or pieces of code. The default layout has been set to “standard” which means it utilizes the “standard.css” file unless otherwise specified. The other paths can be seen as below:



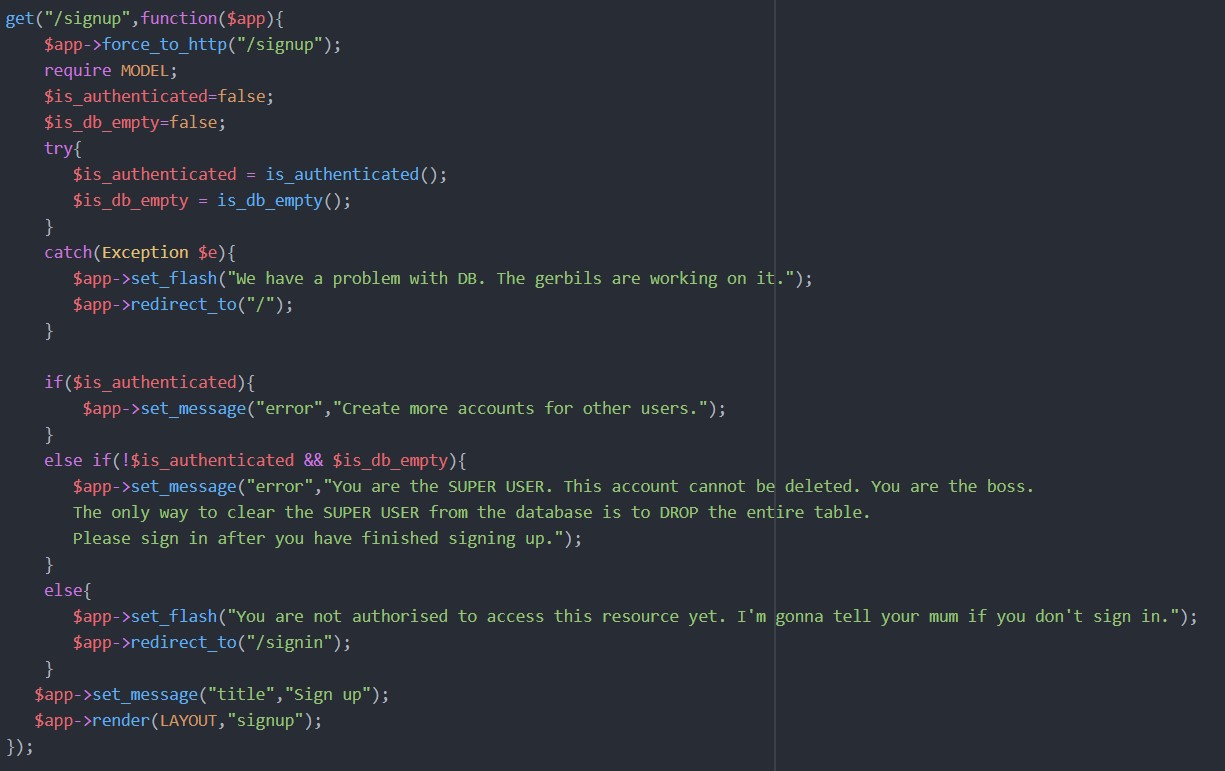
These functions below handle the get requests which display the associated data. As you can see the structure of these functions are very similar, where they use the “set\_message” function from the application.php. All functions contain “require MODEL”. This is because they require code/functions within the model.php file. It can also be seen that the functions utilize the “render” function to decide what views file will be displayed.



The sign-in GET request function utilizes the “is\_authenticated” function within the model.php file. The purpose of this is to identify whether the customer is already signed in or not and if so a message will be displayed depending if they are signed in or not. The request uses the “signin” view file.



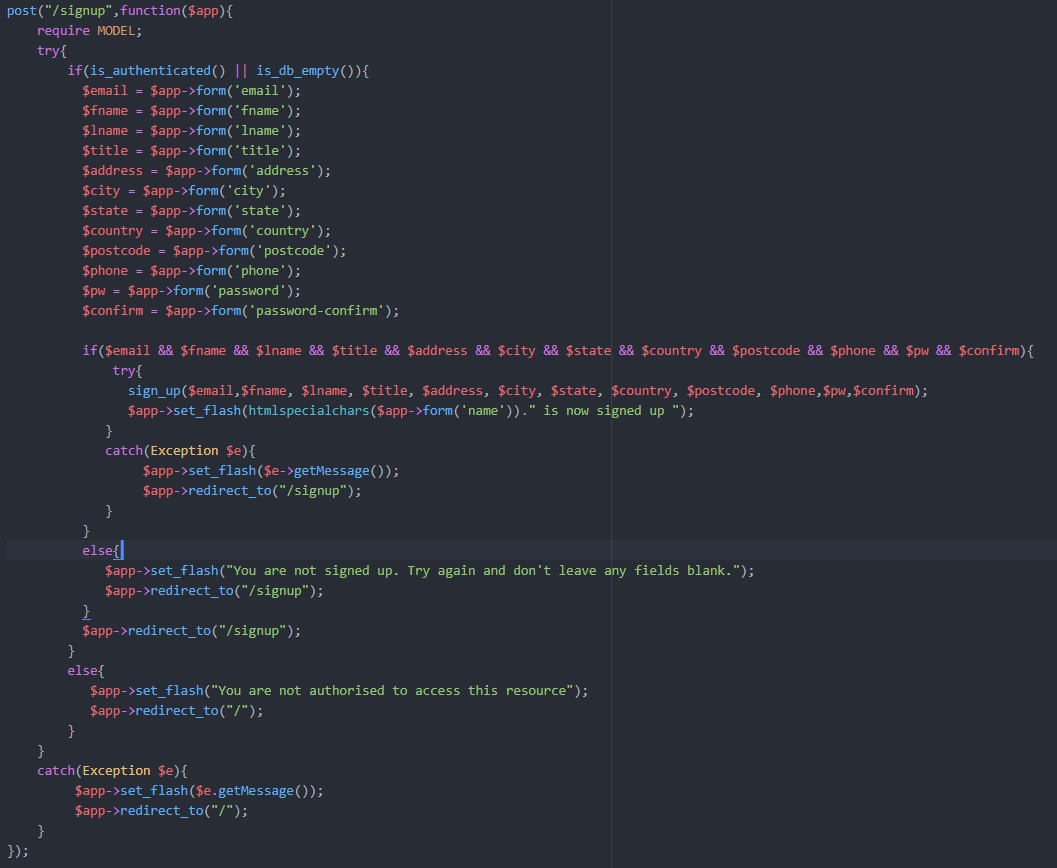
The sign-up GET request function works similarly to the previous one, although it has extra messages for the various different situations. It can also be seen that the “set\_flash” function is used to set a temporary message when an action is completed. It also utilizes the “redirect” function to redirect the page after an action is complete.



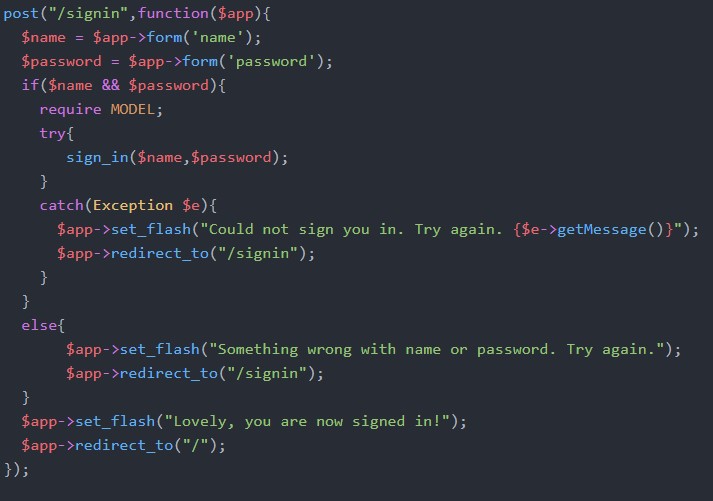
The sign-out request works similarly to the previous requests.



The sign-up post request takes in values from the “signup” views form. These values are then passed into the “sign\_up” function where they will ultimately be stored into the customer database.



The sign-in post request works similarly where it takes values from a form and uses them for the “sign\_in” function within the models.php file. It first checks whether the name and password have been entered before attempting to sign in.



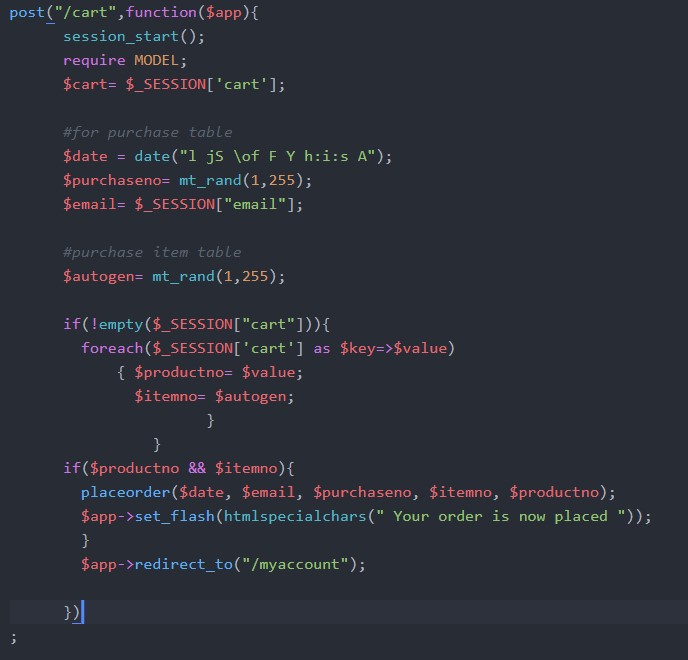
The addproduct post request utilizes the same principles as the sign up request. This request adds a new product to the database. This uses the “addproduct” function.



The cart post request handles the placing an order with all items in the session cart. A session cart is instantiated.

For the “purchase” table, todays date will be stored, a purchase number will be generated between 1-255 and their customers email will be stored.

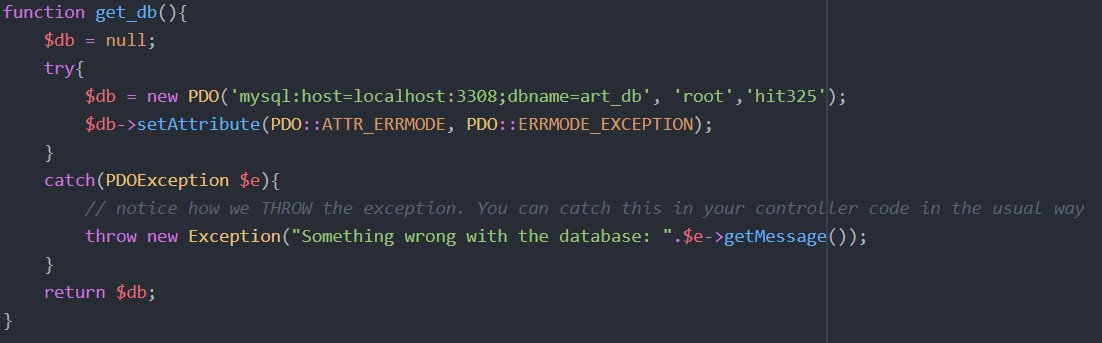
For the “purchaseitem” table, an item number is generated. There is a loop which iterates through the session cart array and then assigns those values to $productno. The “placeorder” function is then called to utilize these values and store them into the database.



### Model.php

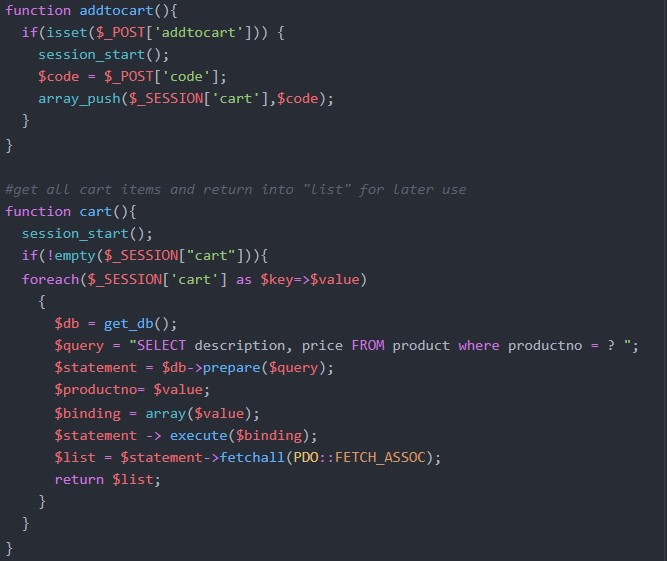
The model.php files acts as the model for the whole application in terms of the MVC pattern. The model.php contains many lines of code which manipulates the views. The model.php file is explained as below:

The “get\_db” function is used to establish a database connection. It utilizes PDO. It can be seen it connects to mysql with the host being localhost on port 3308. The database it uses is “art\_db” with username “root” and password “hit325”. This function is utilized by many other functions throughout the model.php file to establish connections to the database and to either retrieve or store data. By creating a function for this connection we can stop the amount of repeat code, because we constantly need to make this connection.

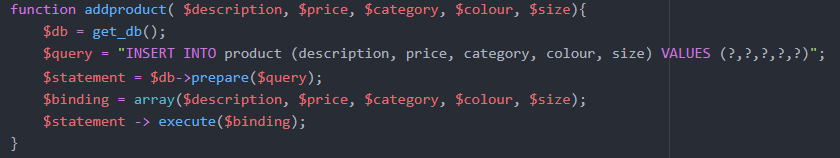


The add to cart function works by pushing the cart item into the session cart array. When a button is pushed within the products view page, this function is activated. The product number is added to the cart array.

The cart function works by iterating through the cart array. It then matches the cart array product numbers to their associated description and price from the database. It then returns $list which is used to display the description and price in the cart page.



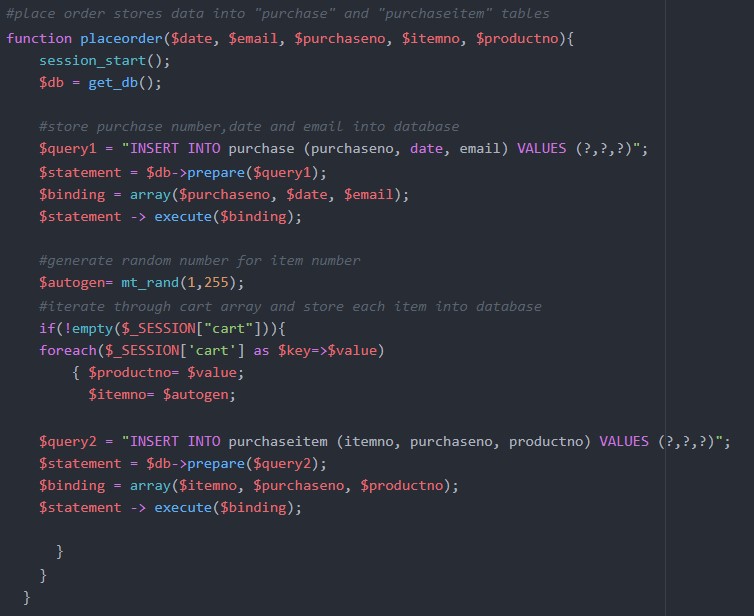
The add product takes in 5 values from a form which is then inserted in the product table of the database.



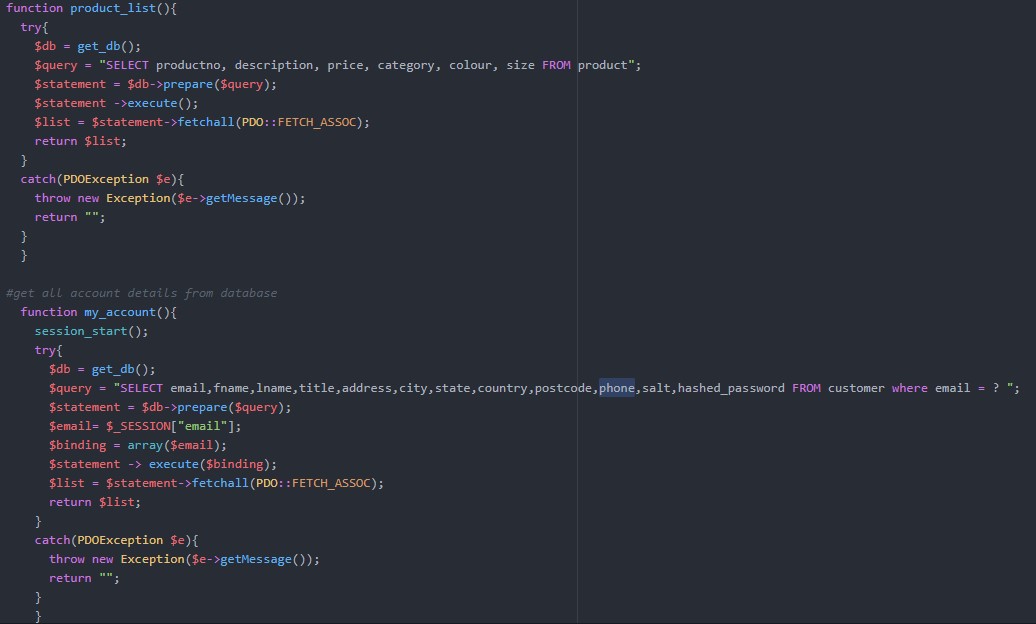
The placeorder function takes in 5 values which will be used to insert in the “purchase” table and the “purchaseitem” table.

“query1” handles the database query for the purchase table. The purchase number, date and customer email are inserted into the purchase table in the database

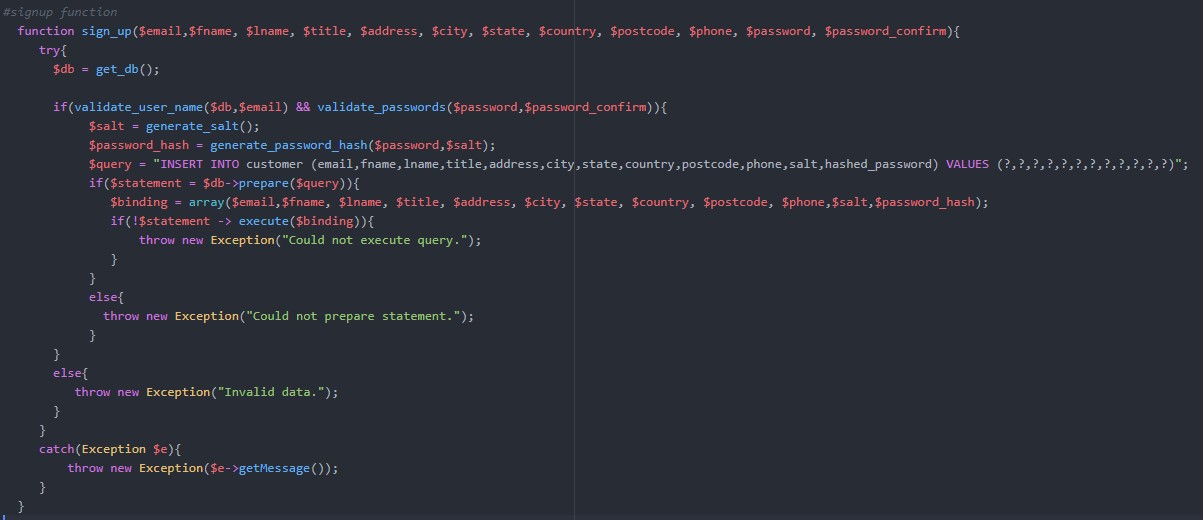
“query2” handles the database query for the purchaseitem table. Firstly, an item number is randomly generated. Then the session cart array it iterated through and each value is assigned to $productno. The purchase number and item number will be the same for every purchased item per order.



The “product list” and “my account” functions work very similarly where they both retrieve data from the database and then then they are displayed through the views.



The signup function takes in 12 values which are stored into the customer table in the database. The function utilizes the “salt” and “generate\_password\_hash” functions which help with security. These 2 functions will be later explained.



The sign in function requires an email and password. The function then searches through the database to find a match with the email and password. If the password is not found an error message will appear as: “account does not exist”. If it is successful in finding a match. An authenticated session is set with the customer’s email and a cart array is instantiated.

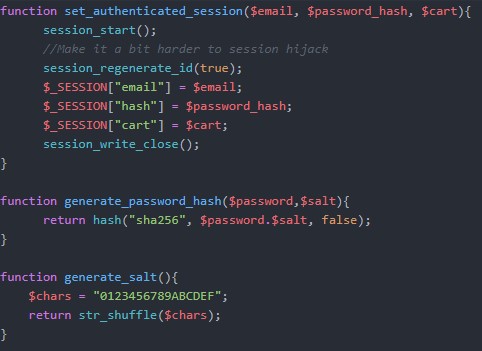


When the sign out function is called, the email and array is emptied and the session\_destroy function is called.



The set authenticated session function stores the session details.

The generate\_salt and generate\_password\_hash are used to “encrypt” the password when stored in the database. This done for security reasons which is explained in detail later.



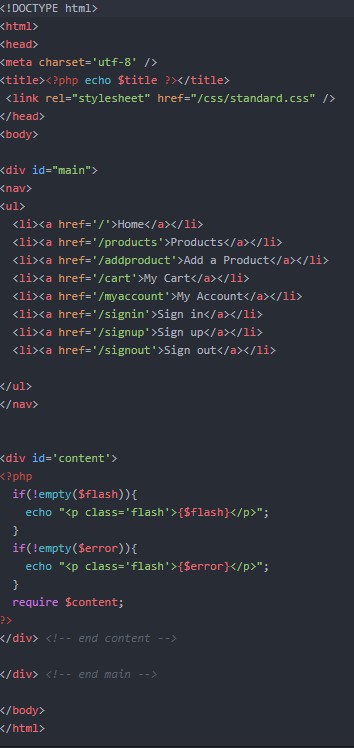
Is authenticated determines whether the customer sign in is authenticated or not.



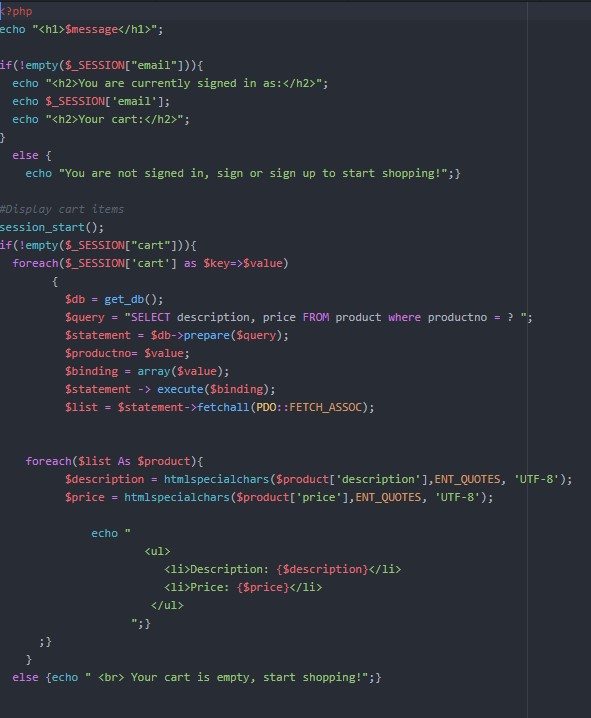
### Views

All files contained in the views folder handle the display of all the data.

The standardlayout.php contains the main navigation written in HTML. It also contains where the flash message will be displayed.



The home page displays a message showing what email you have signed in as. The page will also show all the current cart items if there have been any added. If not, it will show a message that says the cart it empty.



My account page and product page works similarly where it uses model functions to pass in data from them to display. There is a loop with iterates through the list and displays all the associated data.





Sign up and add product work the same where they take inputted values from the user and then those values are stored into the database. However, the sign-up page uses “partials” which are lines of code stored in the partials folder for reusability.





Signin form uses partials identified in the partials folder.



Cart page shows all current cart items along with a place order button. This is used in conjunction with other model and controller functions.



# Security

The project application uses a MySQL database which utilizes PDO. The application does not currently use HTTPS but it should be implemented in the future as it adds increased security for the user when browsing according to Love C 2019.

The database requires a login to access the data. This done to add security to database due to the sensitive information stored. The following credentials to login and access the database are:

**Username:** root

**Password:** hit325

Many functions also use the “try” and “catch” method to catch any errors that many be present.

Passwords use the salt and hashed password functions. This is to stop sensitive information being viewed by employees when they view the database. Therefore, not even employees who view the database will know what the passwords are.

# Licensing and copyright

The application will be free to use and the code will be open-source which is available on GitHub.com (see appendix).

# Lessons Learned

During the progress report, it was identified that I was not able to complete certain features due to my lack of coding experience. At first I intended to complete all business requirements and features, but I later realized it would not be feasible to do so, therefore I added more features to the exclusion list. Although this was the case I managed to complete all the main features which I deemed to be the most important.

The feature to send an email to the customer and art store owner was difficult to implement. After some research it seems as though for the feature to work on a local machine, an email server must be used and additional settings must be altered with in the apache configuration files. With all this in mind I had decided I was not able to get this feature to work, therefore all of the attempted code for this feature was removed

The updated proposed timeline indicated that all features were to be completed by the end of week 12. This was not the case as I was still trying to get the orders to store in the database during week 13. Also as mentioned before it was proposed that I complete the email feature, but that was not the case.

If I were to do this project again, I would look at all the features/requirements and research what is necessary for each feature to work before I state what I will be including in the project. This way the client will not be surprised when features they were expecting are not included. I would also work in a group as there would be more tasks evenly assigned amongst the group. This would greatly help with timeline issues and the lack of coding experience.

# Conclusion

The project to create a dynamic website for a local art company has been challenging in many ways. The project was completed using WAMP technology along with some light HTML and CSS. The main features which were implemented are:

* sign up
* sign in
* sign out
* add product for sale
* display products for sale
* display account information
* display cart on cart page and home page
* add products to cart
* place an order
* store the order in the database

These are the main features which I deemed most important, therefore I focused on implementing them. The feature to send an automated email to the customer and art company owner is very important but was not implemented due to the complexity involved.

With limited coding experience and website project management I have found completing it to be difficult. By completing this project, it is evident that it would have been more feasible to work in a group. Doing so may have resulted in more features being implemented and a better proposed timeline.

# References

Broadbandsearch 2020, “Mobile Vs. Desktop Usage (Lastest 2020 Data)”, <https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics#:~:text=53%20percent%20of%20web%20traffic,56.7%20percent%20from%20desktop%20users.>

Kristensen, E 2020, “15 Eye-Opening Online Shopping Statistics for 2020” <https://sleeknote.com/blog/online-shopping-statistics>.

Love, C 2019, “How HTTPS Works to Keep You Secure”, <https://love2dev.com/blog/how-https-works/>.

# Appendix

WAMP Download: <https://wampserver-64bit.en.softonic.com/>

GitHub Repository: <https://github.com/alexander253/HIT326-Project>