**Final Projects Writeup**

**Individual – Time Tracker**

**Individual – Store Front**

Jimmy Munoz

6/11/2023

CIS-17B

1. **Introduction**

The projects that I decided to work on were the Time Tracker and Store Front. With the group project, we had Yahtzee. I will have a section talking about each of these three projects. All accounts will be shown at the end of each section regarding the project itself.

1. **How To Install and Run**
2. Import the SQL dump files into your MYSQL server using MYSQL Workbench. Run the files to create the tables.
3. To run the files, place the project folders into your “htdocs” folder. This folder is found in your “xampp” folder.
4. With the folder now in htdocs, open it in Netbeans and run the program.
5. **Time Tracker**
   1. **The Basics**

I first wanted to create the basics of the program, so I first decided to do the home screen. Like the one that I made in C++ previously, I made the user interface for the main menu very similar for the web. The one thing I did not add, however, was the option to exit program.

* 1. **User Signup**

For the user signup page, I created a form element in HTML which accepts a username, password, and a password confirmation. Once the form is submitted, it is sent to the same page to a PHP section, which will verify the inputs. There is a verification whether all fields were entered, another for the password length if it is not a minimum of 8 characters long, and one more for a duplicate username. If these errors do not exist, then the new user will be saved in the database using MYSQL statements in the PHP section of the page. These are the **form validations** that I used for this portion of the project. The following screenshot is the User Signup page.

A screenshot of a computer

Description automatically generated with medium confidence

* 1. **User Login**

For the user login page, I once again created a form element which only accepts a username and password from the user. Once submitted, the data from the form will be sent to the PHP section of this page and will verify if the data entered has a match in the database. I created a UserInfo class in PHP which will carry on the information of the user if there is a successful login. If there exists a login, a new UserInfo object will be created, and will store the information of the user, which are username, hours, minutes, seconds. Once completing this, I created a new session variable to **serialize** the user object, in which the object will be stored in the session so that the information will be retrieved in the user’s page upon successful login. The user will then be redirected to their respective page. The following image is the User Login page.

A screenshot of a login screen

Description automatically generated with medium confidence

* 1. **User Page**

The user’s page has the same setup as to what it was in the C++ version. Before loading the page however, I have a PHP section of this page which will retrieve the **serialized** data that was sent from the user login page. This data is then sent to the JavaScript portion of this page, which will be used to display the user’s information. Now, with the information being **transferred from PHP object to a JavaScript object**, we can now view our time. There are three sections to the user page: view your time, track new time, and log out. In the view time function, and using the new JavaScript object, the time recorded for this user will be displayed on the screen.A screenshot of a computer

Description automatically generated with medium confidence The track new time option will display a small menu asking the user if they wish to track a new time. There are a set of directions for the user to read before beginning. These directions are the same as the C++ version. Once the user clicks on the button to start a new time, I have a variable which holds the current time upon clicking on the button. When the user decides to click on the button again to stop recording a new time, another variable is created such that it holds another time in which the button was clicked. After getting these variables, I subtract the one from the end by the one from the beginning to get the total time recorded for this run. That time will then be sent to the PHP portion of the page to be processed. This process is the same as the C++ version using the same algorithm for calculating the hours, minutes, and seconds. Upon getting the new values, a MYSQL statement will be executed which will update the current user’s time in the database. A screenshot of a computer

Description automatically generatedFinally, a message will appear which will display the time recorded for this run. The user can then view their new time in the view time section of the page.

A screenshot of a computer

Description automatically generated with medium confidence

* 1. **Admin Login**

This page is the same as the user login page, only this time it deals with the administrator. The only difference between these two is that instead of searching in the user table from PHP myadmin, it checks the admin table. Other than that, it functions the exact same as the user page. Upon successful login, the admin will be redirected to the admin page.

* 1. **Admin Page**

In the admin page, there are two options, the view users option and the logout option.A screenshot of a computer

Description automatically generated with medium confidence

By clicking on the view users option, the admin will be able to see all the users that are registered in the database along with their times recorded. The logout option simply logs out the administrator.

* 1. **View Users**

As mentioned before in 1.6, the view users page allows the administrator to view all users currently in the database along with their times. In this page, a tale will be displayed that shows all users. For each user, it will show their username, hours, minutes, and seconds. Along with that, two buttons will be available for each user. One of these buttons is the edit button, which will allow the administrator to modify a user’s time, and a delete button, which will allow the administrator to delete the user.

A screenshot of a computer

Description automatically generated with medium confidence

Upon clicking the edit button, a small menu will appear which will allow the admin to modify the respective user’s time. There are three inputs, one for the hours, minutes, and seconds. This function works the same as the C++ version of this project, where the admin can enter a negative number to take away time and a positive number to add time. There will be two buttons, one to exit the menu and another to submit the changes. When clicking on the exit button, the changes will not be made, and the menu will close. When clicking on the submit button, the data entered will be submitted to a PHP section of the code and process it. As mentioned before, this process is the exact same as the C++ version. One more thing, each input for the time is set to 0 by default.A screenshot of a computer

Description automatically generated with medium confidence

When clicking on the delete button for the user, a small popup will be displayed with a message for the admin, warning them that all existing data for that user will be lost upon deletion. If the admin chooses to delete the user, a form will be executed, sending the id of the user. The id is used to search for the user in the database, and then delete it permanently. A screenshot of a computer

Description automatically generated with medium confidence

* 1. **Final Notes (Which requirements were used and how)**

From the requirements provided, there were 3 topics that I used for this project. The first one was form validation, as mentioned in Section 1.2. Form validations were used in login and signup pages. I evaluated each input to make sure that nothing went wrong, such as finding a duplicate username to comparing password. Another topic that was used was serialization, as mentioned in Section 1.3. The way I did this is that I created a PHP object which contained the information of the user. This object was stored in a session variable and serialized, which will then be received again in a PHP section from the next page. When receiving the data from the PHP section on the new page, that data is then transferred to a JavaScript object. In this JavaScript object, the data is used to display the user’s information to the screen. This also refers to the next and final topic that I used, which was passing information PHP objects to JavaScript objects.

* 1. **Admin and User Information**

This section contains the information about the admin and users available.

Admin Username: **TestAdmin**

Admin Password: **123456Qw**

First User Username: **User1**

First User Password: **123456Po**

Second User Username: **Username1**

Second User Password: **123456Po**

1. **Store Front**
   1. **The Basics**

I used the same layout as the one I had for the C++ version of this project, but I wanted to improve on it more. I kept the same flow of the program, but I just wanted to improve more on the user interface. So, I decided to focus more on the CSS to make it more user friendly.

* 1. **The Home Page**

This page has a very similar feel to mostly all retail websites on the internet. This page contains all the items that the retailer has to offer, with each of the items having a button to add to cart. Before going any further, I wanted to evaluate the PHP I used for this page. Upon loading onto the page, a block of code will be executed, verifying to see if a user is logged in by searching via session variable. If there is not a user logged in, then at the top right of the screen, there will be two options that will allow the user to signup or login, if there ss a user logged in, then those two options will be switched to view the user’s cart and to logout. This will be further explained later. For now, I will be explaining this page assuming there is not a user logged in. Each item will be contained in their own respective grid, making the page look more organized. Each of these grids contains the item’s name, price, stock amount, description, and a button to add to the cart. The items will be displayed in columns of three. If the user clicks on the button to add to the cart, they will be redirected to the login page to login to their account before adding an item to their cart. I have a separate class in PHP called Admin, which has all admin functions. These functions range from getting all items from the catalog, updating an item, adding a new item, and deleting a new item. There are plenty more functions that I will explain later. For now, in the PHP section of this page, I declared a variable which will receive the information sent from the “getItems” function from the Admin class. This function will return all the items found in the “enum\_items” table in the PHP myadmin page. The value returned is a JSON array of objects, with each object containing the item’s information as stated before. This JSON array is then sent to set to a variable located in JavaScript. I also created a class called User, which contains all the functions for the user. These range from displaying all items from the retailer to displaying the user’s cart. More functions are in this class and will soon be explained later. So, the variable that contains the JSON array that was sent from PHP is being passed as a parameter to the User object constructor. Then, I displayed all items using the “displayItems” function located in the User object. This simply displays all items available. I used another function called “verifyUser” from the User object to verify if a user is logged in. As stated before, this function is used to determine what should be displayed on the top right corner of the screen. The “attemptAddCart” function, also a method of the User object, is used to attempt to add an item to your cart. As stated before, this determines whether an item should be added to your cart or redirect you to the user login page. We will return to this page later when a user is logged in. The following screenshot is how it looks when a user is not logged in. A screenshot of a computer

Description automatically generated with medium confidence

* 1. **User Signup Page**

The user signup page operates similarly as the one used in the Time Tracker project. During the time of creating this project, I realized that you have to secure each form by using special functions, such as “htmlspecialchars” which will prevent JavaScript injections. So, for each form executed in this project, I used this function and more functions to improve the security of my pages. This is the main difference between the two projects. Also, I added an email section to the user signup. Upon entering data to the form, the “verifyNewUser” function from the PHP admin object will be called. The parameters of this function are the username, email, password, and password confirmation that was entered in the form. In this function, I have it set to return all errors that were found in the data entered. First, I called another function from the admin object called “testInput” which will test the input to prevent any injections. I call this function on each input that was entered. Once all data has been tested and returned, the next step is to search for any errors. I have a series of possible errors that are possible, and these are the same series that I used in the previous project. One new thing I added though was a function called “checkDuplicate” that is in the admin class. This function accepts two parameters, the name you want to check to be duplicated, and the field of the name. The first field is for the username, which will search for duplicates for the username only. The second field is for the email address, which will search for duplicates for the email. The third and final field is for the item name, which will search for duplicates for the item’s name. This function will return the number of rows found for the name requested. I used this function for both the username and the email to make sure there are no duplicates. After verifying all possible errors, the function will return all errors found. If the number of errors is greater than zero, that means that the signup was not successful, and the errors will be displayed on the screen. If there were no errors sent back, the “addUser” function will be called from the admin class. This function accepts three parameters which are the username, email address, and password. In this function, the data that was sent from the form will now be stored in the database, which means that the signup was successful. The user will then be redirected back to the home page which contains all items in the catalog. A screen shot of a computer

Description automatically generated with medium confidence

* 1. **User Login Page**

Using the same form validation as mentioned previously, the user login page operates the same as the one created from the Time Tracker project. In this page, the user can enter their account information to login. Upon submitting the data in the form, in the same page, a block of PHP code will execute. In this code, another function from the admin class called “verifyLogin: will be called. This accepts parameters of the username entered, password entered, and a level. The level will represent who is logging in. In this function, the inputs from the form will be tested using the “testInput” function mentioned previously. Then, by using the level that was passed as a parameter, we use it to verify who is trying to login. If we were to send 0 for the level, this means that a MYSQL statement will be prepared to get all data from the “entity\_users” table. If we entered 1 for the level, then a MYSQL statement will be prepared to get the data from “entity\_admin” table. In this case, since we are sending a 0 as the level, we are preparing a MYSQL statement that will receive all the users’ data from the table. In this statement, we are selecting all data from the table that contains the username entered. If the username entered was not found, that means that it does not exist, which will return the error. If the username entered was found, there is one more stage to complete the login. By using the password verify function that is built in PHP, we can use it to verify the password entered with the password that is stored in the database. If the function returns true, then a message will be returned saying “Success” which is used for conditional purposes when returning to the main page. If the function returned false, then an error will be returned to the page. The function returns the errors found in this set of verifications. Back to the main page, if the received is equal to “Success”, then that means the login was successful and a new session variable will be created, which will store the username of the name entered. They will then be redirected to the catalog page that contains all the items. If the login was not successful, the errors will be displayed.

A screenshot of a computer

Description automatically generated

* 1. **Home Page (with a user logged in)**

Now that we have a user logged in, we can display now display something new in the top right corner. As mentioned previously in Section 2.2, if there is a user logged in, then there will be new options on the top right corner of the screen. So, instead of displaying a login and signup option, it has now turned into view cart and logout. Along with having these new functions, we can now store items on our cart. So now when we click add to cart, a message will appear saying that the item was added to our cart successfully. For the cart, I am using cookies that store all items of the user. This cookie will be used to display the items in the cart when clicking on the view cart option. The view cart option will take the user to a new page that shows their cart with more functions that I will explain in the next section. The logout option will simply logout the user.

A screenshot of a computer

Description automatically generated

* 1. **View Cart Page**

As mentioned in the previous section, the view cart page will display the cart of the user via cookies. In this page, there are two new options that appear on the top right corner of the screen. One being to view your order history and the other one returns you back to the home page. In this page though, you have two more functions. One is the remove from cart button, which will remove an item from your cart. The other function is to purchase items from the cart, which will purchase all the items from your cart. If the user decides to remove an item from their cart, a function from the user class known as “removeItem” will be called. This function accepts the index of that item from the JSON array as the parameter. Inside this function, the item that was requested to be deleted will be deleted from the array, and the new array will overwrite the cookie. If the item deleted was the last item from your cart, the cookie will be deleted. Upon completion of this, the “updateCart” function will be called, which will display the current cart. If the user decides to purchase all items from the cart, a block of PHP code will be executed from the same page which will then call the “storeToDB” function from the admin class. The parameters passed through are the username of the user and the items that were in the user’s cart. First, I search for the user’s id by using their username. When getting the id of the user, I store information to the “entity\_orders” table, which contains the id of the user and id of the item they bought. This will be explained further in the final section of this project. After that, I store the order id and the user id to a table called “xref\_orders\_users” and I store another one containing the order id and the item id to a table called “xref\_orders\_items”. These tables will be further explained in the last section of this project. Once all the data has been stored, I update the stock of each item bought. I also deleted the cookie of the user’s cart.

A screenshot of a computer

Description automatically generated with medium confidence

* 1. **View Order History**

This page will allow the user to view their order history. If the user has not ordered anything, a message will appear saying that you have not ordered anything yet. In the PHP section, I call a function from the admin class called “getUserOrderHistory”, which will return all orders from the user. By using OpenOffice, I was able to create a query that uses five tables, entity\_users, entity\_orders, enum\_orders, xref\_orders\_users, and xref\_orders\_items. The cross-reference tables are used to relate the respective id of each the id listed in its table. By using this, I can display information about a certain user and the items they bought. So, by using this, this page displays all the orders that user has ordered.

A screenshot of a computer

Description automatically generated with medium confidence

* 1. **Admin Login Page**

This page is no different than the user login page that was explained from this project, the only difference is that it is testing for admin login verification. Nothing has changed other than that.

* 1. **Admin Page**

In this page, the admin has a few functions. First of all, you can view all the items that are on the website, and you can modify or delete each of them. On the top right corner of the screen, there are two new options. One is to check all orders and the other is to logout. The admin also has the option to add a new item.

A screenshot of a computer

Description automatically generated

When clicking on the edit button for any item, a small menu will appear that will allow the admin to modify an item’s information. The values that are left blank means that there has been no modifications. When clicking on the confirm changes button, a function from the admin class will execute called “modifyItem”. This function will update the existing item that was requested to be modified. If the admin clicks on the delete button and confirms it, another function will be called known as “deleteItem”, which will simply delete the item from the database.

A screenshot of a computer

Description automatically generated

The add new item button will display a small menu which will allow the admin to add a new item to the store. After entering the new item’s information, the “verifyNewItem” function from the admin class will be called. In this function, the information entered will be verified to make sure there are no errors. If there are no errors, then the “addNewItem” will be called which will add the item entered into the database.

A screenshot of a computer

Description automatically generated

* 1. **View All Orders**

In this page, all the orders that all users have placed will be shown here. By using the “getAllOrders” function from the admin class, and by using the OpenOffice method I stated before, I can display all items ordered from their respective user.

A screenshot of a computer

Description automatically generated with medium confidence

* 1. **Final Notes (Which Topics Were Used and How)**

I managed to fit all topics in this project, and I will elaborate on it now. For the MVC design requirement, I used four different classes to help implement this. The Model portion of the MVC design contains two classes, the PHP class known as “Admin”, and the JavaScript class known as “UserModel”. The PHP class deals with interacting with the database, such as retrieving, storing, and updating data from the database. The JavaScript class mainly deals with the logic of the user’s cart, such as attempting to add an item to their cart and attempting to remove an item. These different type of functions will return a certain message which would be used to display a specific interface on the screen. Speaking of retrieving that message, the one who uses that message is the View portion of the MVC design, a JavaScript class known as “UserView”. This class specializes in displaying a certain interface on the screen, depending on the code received from the Model class. The functions in this class ranges from displaying the user’s cart to displaying all order history from all users. The Controller portion of the MVC design is the class known as “UserController”. This class contains both the Model and View class. This class is used for requesting something from the server, such as displaying all items and attempting to store an item to a user’s cart. The second part was related to passing information with JavaScript objects to PHP with cookies. The way I did this was using cookies as the carts. When I want to add an item to the cart, a cookie is either created or updated containing that item. Then when I want to view that item or items in my cart, the cookie is called from PHP to see if it exists or not, and it executes its respective function involving the cookie. The third part involves JSON and how it was used. Although I did not use it for local storage, I did use JSON for cookies and receiving information from the database, such as retrieving the information of an item. The fourth part involved using databases using SQL using entities, cross-referencing, and enumeration tables. For the entities, I had the admin, users, and orders table be the entities. For the cross-referencing, I used two of these that cross referenced the orders and users, and another that cross referenced the orders and items. The enumeration table was used for the items. By using these tables, I was able to relate the ids of the items, orders, and users to display the correct information of each order. The fifth part involves form validation, which I mentioned previously back in Section 2.2 and the final section of Section 1. To summarize though, I created functions that validate and test the user’s input. The sixth part involves the user-admin login system. I created certain pages that are only for the users and some that are only for the admin. These are not the same since they each have their own respective functions. For the seventh and final topic, this involves using cookies and sessions. As mentioned previously, I used cookies to hold the items acting as the user’s cart. For sessions, I used a session variable that holds the username of the user or admin. These variables are used for specific purposes, such as what to display on screen if the user is not logged in.

* 1. **Admin and User information**

This is the information of the admin and user.

Admin Username: **TestAdmin123**

Admin Password: **123456Qw**

First User’s Username: **User1**

First User’s Password: **123456Po**

Second User’s Username: **Username2**

Second User's Password: **123456Po**

Third User's Username: **Username3**

Third User's Password: **123456Po**