For this assignment, I implemented the customer side of a website for selling video games. `home.php` is the starting/home page of the site. At the top are links to different product categories that the user can browse through and purchase items. The home page lists only the current special sales items. The content is generated by doing a joined query into my `SpecialSales`, `Products`, and `ProductCategory` tables and printing out the returned data. I chose to have a product image represented by a simple file URL within a folder called “pimg” and the files named according to their `productID` number. This is a faster and easier process for the database to perform rather than storing all the image data as binary within the database itself. Each item on the page is represented by its box picture on the left, the productName in the
middle, the productDescription beneath it, the productPrice (multiplied by its percent off) on the right, and an “Add to Cart” button beneath it. All items across my website are represented in this format through database queries.

A user can add products to their cart regardless of whether or not they are logged in. Since a customer would have the realistic expectation that if they add items to their shopping cart and return later it should still be there, I did not think that only storing order items within a session cookie would be good enough. Instead, whenever a user clicks on “Add to Cart”, an order item is inserted into the database where it can be selected later without worry that the user may have timed out. I created a new OrderItems table which includes fields for the ProductID, price, and quantity. The Add button will invoke AJAX to send the id and price of whatever the user clicked on to a script (editOrder.php) which inserts the data, giving it a default quantity of 1 and an auto incremented orderItemID. Furthermore, whenever a user adds an item, their overall order information has to be updated as well in the table “Orders”. This table includes the fields of an orderID, to match with an OrderItem, a customerID, to match with whose order this is, an orderDate that is auto set to the current timestamp on every update, and a total orderPrice.

Using this structure of database tables while still allowing a user to not be logged in required some extra work to ensure a shopping cart could be properly saved for later shipping. First was to realize that even if a user just wants to browse the website, and ignore adding items to their cart, a proper ‘OrderID’ would need to be generated even if only to be informative that their cart is empty. Therefore, upon a user’s first click on “Add to Cart”, or alternatively if they visit the Checkout page, if the customer has not already been given an OrderID, we insert a new empty Order into the Orders table, do a query into the last row that was just created, and place the created OrderID into the customer’s session variable as $_SESSION['orderID’]. This way wherever the user is on the site their cart info can still be matched with them. The other issue was that a customer must have a CustomerID matched with their Order (saved in $_SESSION['customerID’), and again I solved this by giving all customers a CustomerID upon entering the site even if not logged in.

This opens up the problem of that if a customer creates a shopping cart and then leaves forever without buying anything, there is wasted space in the database. To solve this I added a special field in my Customer table called “temp”, which is assigned a ‘n’ for customers that have
sign in accounts already, and a ‘y’ if the customer is a “temporary customer” who has not signed up yet. If a temporary customer signs up, and is therefore given a new real CustomerID, I update the Orders table with the new CustomerID corresponding to the $_SESSION['orderID'].

With the way my database is setup, this is the only difference between a real and temporary customer. The OrderItems table only corresponds to an orderID, not a customerID, and does not need to be updated. Finally, if this site were to be used by real customers, I or an administrator could periodically delete temporary customers. This would only require a query to the Customer table for customers with temp='y' and deleting that customer, the Order associated with their customerID, and any OrderItems associated with that OrderID.

The other fields in my Customer table are the CustomerID, creditCard number, creditExpirationDate, creditType (Visa, MasterCard, American Express, or Other), and both a shippingAddress and billingAddressID, corresponding to another table I created, called Address, that has an AddressID, first name, last name, country, city, street, zip, and state. A shipping address and billing address can be the same, in which case the billingAddressID and shippingAddressID will be the same number, or they can be different. These fields can be null in the case of a temporary customer, or are filled in through the customer signing up.

The Sign Up button, on the top right corner of my home page, links the user to the page signup/signup.php. This takes the user to a page designed to complete their username and password to be used to login to the site and keeping their information, as well as the described billing and shipping information. By default shipping and billing addresses are the same and only occupy 1 row in the database, but the user can click a checkbox to fill out a separate billing address. When they are done filling out the form it is shown in the signup/addProfile.php page where the user can review their information or go back and change it. If they submit the form there is an AJAX call to the script signup/addCustomer.php, an all purpose script for dealing with customer information in the database. The user information is sent to the script, parsed with htmlspecialchars() to increase security, and inserted in the database. Afterwards the user is taken back to the home page, which now seeing that they have been assigned a $_SESSION['userID'], display the Logout, Orders, and Profile options instead of the signup button.
Profile links to `signup/editProfile.php` and displays a summary of the same information the customer uses in the Sign Up form. They can choose to edit any of their information, after inputting a password, and then submitting it. They have one last chance to review their information in the `signup/updateProfile.php` page, and then submit the information to the database again through `addCustomer.php`, this time doing an UPDATE instead of an INSERT.

At the top of all customer pages are links to the various game console categories that they can shop through. Rather than create separate pages for each category I realized that all pages should look basically identical except for what category my database lookup is going to. I created one file `category.php`. When the user clicks on a category button, the CategoryID is passed as a GET to `category.php?cid=#`. The category page first looks up the newest sale for an item in that category and displays it at the top. Then there is another lookup for all items in the category (except for the one already displayed on sale) and listed down the page.

When the customer is done adding items to their cart and click go to checkout they can review their order. Their items are shown down the page and they can choose to change the quantity of the item, delete the item, or delete the entire cart. If they choose to do any of these options `editOrder.php` is called. If the quantity is changed I must first delete the previous quantity*price from the total price in the Order table. Then I can add the new quantity*price to the total price. If it is a delete I just remove the current quantity*price from the total price, and then delete the orderitem itself. Finally, if it is a delete cart, the Order and all OrderItems in that Order are deleted, and the user is given a new Order.

When they are done, and are logged in, the order is complete and they are returned to the home page. I created another field in the Order table called “toShip” (‘y’ or ‘n’), meaning that the order is complete and ready for shipping. This way it is easy to differentiate between shopping carts that are in progress or have already been completed. On the home page, the Orders button displays a summary of all previous orders for the customer, in `orders.php`. If the user has not purchased anything yet, then nothing is displayed, but if they have all rows from the Order table associated with their CustomerID with a toShip value of (y) are displayed. Clicking on the green arrow next to an Order will then display all OrderItem information associated with that Order next to it.
Reports of all orders can be seen on the employee/manager.php page after logging in through the home page as a Manager. Clicking on the Order selection shows all orders and what was in them. The manager can also choose whether or not they only want to see purchases that were on Special Sale, search between two Order dates, look for a specific ProductName, or select a specific Category.