CS 405G Introduction to Database Systems

Fall 2013

Project Report

By

Feiyu Shi & Libby Ferland

12/1/2013

1. **Database Design**
2. ER Diagram

See Appendix.

1. Relational Schema Design, Functional Dependencies, and Normalization

**Entities:**

**Customer**(string:Email, string:Fname, string:LName, string:Password)

Email -> Fname, LName, Password

3NF

**Basket**(string:CEmail, int:BasketId, date: ShopDate)

CEmail, BasketId -> ShopDate

3NF

**Orders**(int:POrderID, string:Status)

POrderID -> Status

3NF

**Employee**(int:EId, string:EPassword, string:LastName, string:FirstName, Boolean:IsManager)

EId -> EPasword, LastName, FirstName, IsManager

3NF

**Item**(int:IId, string:IName, string:Category, string:Description, float:IPrice, int:Quantity, float:PromoPrice)

IId -> IName, Category, Description, IPrice, Quantity, PromoPrice

3NF

**ShipPrice**(string:ShipMethod, float:ShipRate)

ShipMethod -> ShipRate

3NF

**AddressBook**(int: AddrIndex, string: CEmail, string:AddrLine1, string:AddrLine2, string:City, string:State, int:Zip, Boolean: IsVisible)

AddrIndex, CEmail -> AddrLine1, AddrLine2, City, State, Zip, IsVisible

3NF

**PaymentMethods**(string:CEmail,int:CardNo,string:CHolderLastName,string:CHolderFirstName,date:CExpirDate, Boolean:IsVisible)

CEmail, CardNo -> CHolderLastName, CHolderFirstName, CExpirDate, IsVisible

3NF

**Relationships:**

**Purchase**(string:CEmail, int:InvoiceNo, int:PurchaseRating, string:Review, date:PurchaseDate)

CEmail, InvoiceNo -> PurchaseRating, Review, PurchaseDate

3NF

**Ship**(int:EId, int: OrderID, date:ShipDate)

EId, OrderID -> ShipDate

3NF

**Promote**(int:EId, int:IId, float:PromoRate, string:PStartDate, string:PEndDate)

EId, IId -> PromoRate, PStartDate, PEndDate

3NF

**BasketContains**(string:CEmail,int:BasketId, int:IId, int:BQuantity)

CEmail, BasktId, IId -> BQuantity

3NF

**OrderContains**(int: COrderID, int:IId, int:OQuantity)

COrderID, IId -> OQuantity

3NF

**BillingAddress**(string:CEmail, int:CardNo, int: AddrIndex, Boolean: IsVisible)

CEmail, CardNo, AddrIndex -> Isvisible

3NF

**ShippedTo**(int:OderID, string:CEmail, int:AddrIndex)

All attributes are primary keys, and they don’t depend on each other.

**PaidWith**(string:CEmail,int:CardNo, int:OrderID)

All attributes are primary keys, and they don’t depend on each other.

**ShippedBy**(int:OderID,string:ShipMethod)

All attributes are primary keys, and they don’t depend on each other.

1. **Description of Programs**

In designing this project, every effort was made to make it as modularized as possible. Each page visible to the user is contained in its own file, and some of the more complicated functionalities like placing an order were separated out as well. While this design may increase the number of separate files overall, the code itself is able to remain fairly simple, understandable, and easy to modify.

The codebase is divided loosely into sections based on functionality. There are groups of files to handle the following aspects of the project:

* Table creation and insertion of "dummy" data
* Customer login and home screen
* Employee login and home screen
* Customer address book and payment method management
* Employee inventory and pending orders management
* Manager sales management (statistics and promotional pricing)
* Main store and item displays, including search function
* Customer shopping basket and order placement

The files were named as transparently as possible. A complete list of all files in the project, including brief descriptions of their functions, can be found in Appendix B.

The major decision that determined the program flow was how to handle data storage and access. Tables and all data for this project are stored by a separate program called MAMP. The vast majority of program flow is controlled in the implementation instead of the data structuring; most of the program flow is controlled directly by the user through HTML commands. For example, users can access key functions such as item search, shopping basket management/order placement, and log out from nearly every page. Users are also able to return to the store home page and their account control panel from nearly every page. Since most of the actual program is controlled through HTML links to different PHP files, the program files themselves are fairly linear, modular, and easy to understand from a programmer's perspective. The decision to move program control to HTML allows a layer to be added between implementation and execution. The files themselves are compartmentalized and can be organized by functionality. The program itself presents a highly circular flow that was intended to make it much easier for users to navigate through the functions, just as they might in a functional online store. Relinquishing control to the implementation was done in the hope that it would make the program flow much simpler from the point of view of the users and the programmers.

Since the code itself is so highly modularized, no functions outside of PHP standard libraries were needed. There are several important design aspects that must be addressed, however, and most of them were made with the idea of keeping code as clear and simple as possible. First, sessions were used to store customer login information and allow for individual customer accounts to be logged in. Sessions were used in this case instead of cookies because while they store all the information a cookie does, they do not store it directly in a web browser. This allows for a slightly simpler method of data storage and, more importantly, made testing much easier since web browser data was not being modified as all information was being stored locally in the program. The only information that needed to be stored in the session array was customer email or employee ID (depending on who was logged in), so there was no need for a more complicated method of data storage.

The second design decision important to the project was the use of POST instead of GET to pass user input between pages. POST was used almost exclusively because the only data that might be called permanent was the login information; all other data came directly from user input and had a lifespan of a page or two (two or three files) at most. Since most data was extremely transient and would either be used to reference a table entry or create or update a table entry, it was not necessary to use GET and would not have made sense to do so. The use of GET or POST did introduce some difficulties when there were multiple input fields to pass to the next page. In this case, imploded arrays and hidden input HTML forms were used to pass multiple pieces of data as a single string.

The third design decision with the greatest impact on this project was how to handle security and data transparency. The decision to handle these two things at all adds an extra layer of complexity to the project. First, there is some input type safety checking in areas where customers and employees input data. For example, customers may not enter invalid expiration dates or card numbers. These checks were accomplished dually in PHP and HTML5. In PHP, if/else logical checks were used to find things like empty string or negative number entries in inappropriate places. In HTML5, regular expressions were used to find things like improperly formatted card numbers or zip codes through pattern matching. HTML was used in this case because although the regular expressions are the same as they might be in PHP, the logic was much simpler. Second, the program does provide some transparency to the user, just like a functional store might. For example, when a customer places an order, the program will echo all the order information back to them, or display an error message if appropriate. The data transparency was mostly important in areas where tables are being updated or new entries are being created, which allows the customer to ensure all data was properly processed and also makes the job of debugging slightly easier. The actual functions and workflow can be seen in Section 3.

The last design peculiarity that may require some justification is the use of a Boolean attribute to control visibility to the user. Due to the database design and its reliance on many foreign keys, allowing users to "delete" data is a complicated proposition. In order to accommodate this functionality, though not to necessarily make the program as secure as it would need to be to actually handle sensitive data, the Boolean "IsVisible" attribute was added to tables storing customer information. This attribute acts as a switch; the logic of checking for the Boolean bit every time information is displayed to the user is still much less complicated than checking for and removing foreign key dependencies to delete items from tables. The Boolean attribute occurs in multiple tables because different items might need to be hidden in different places. For example, all addresses are stored in an AddressBook table. If a customer deletes an address from their shipping address list, the Boolean switch is flipped on the AddressBook table entry. However, any payment methods that use the address for billing, or any past orders that were shipped to that address, can still reference the same table entry. This eliminates the need for a lot of redundancy or more complex data structuring, but the reasoning may not be entirely clear without explanation.

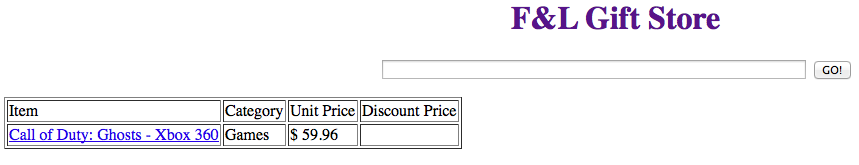
Please see Appendix B for more specific information regarding the codebase.

1. **Program Functions**
2. Search Engine

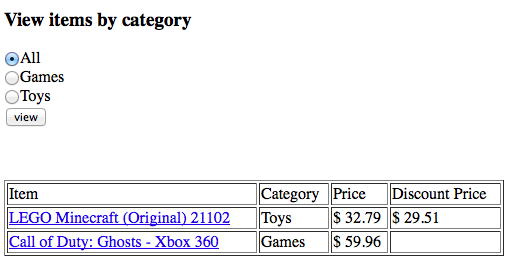
Input form:



Sample output:

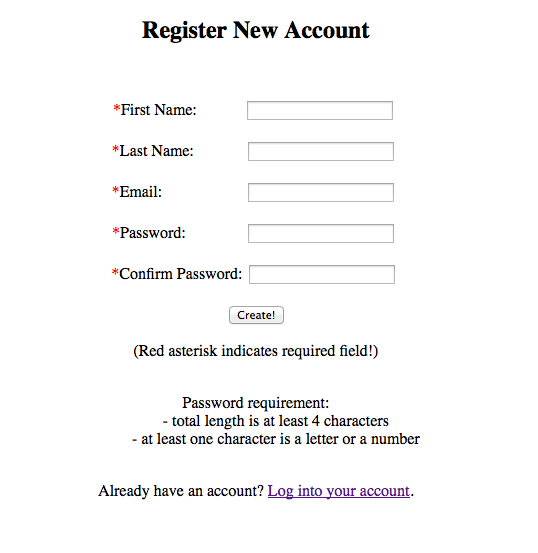


1. View by Category

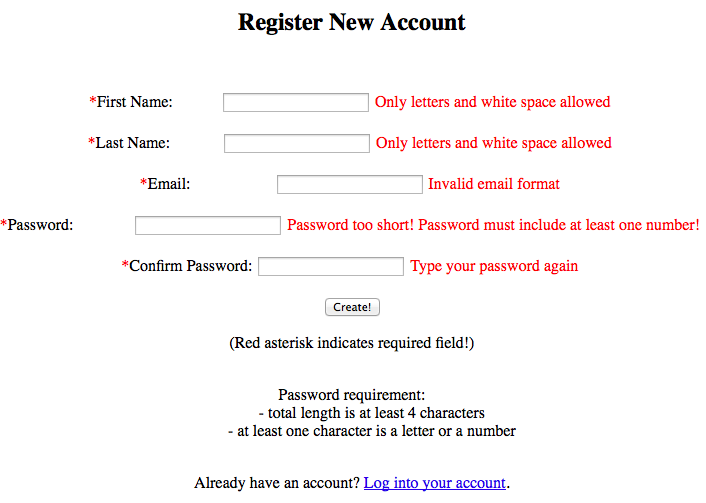


1. Customer Registration

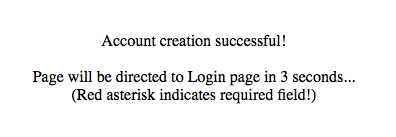
Input form:



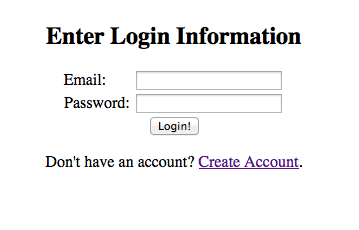
Output form 1 (errors):



Output form 2:

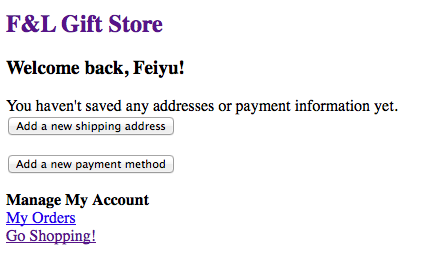


1. Customer Login

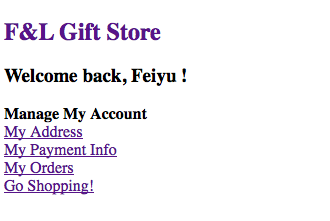


1. Customer Account

Page of new customer:

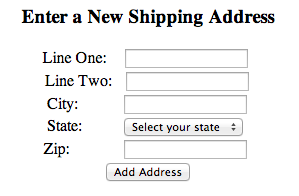


Page of old customer:



1. Address Book

Input form 1:



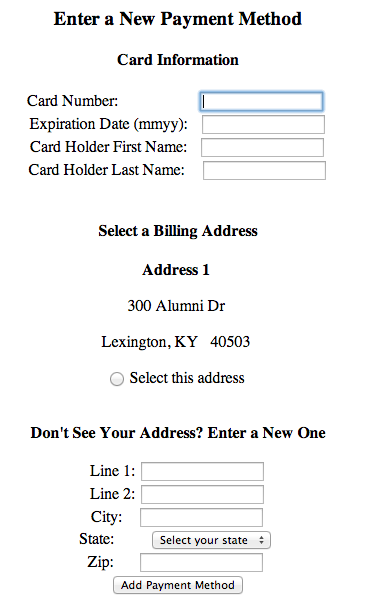
Input form 2:



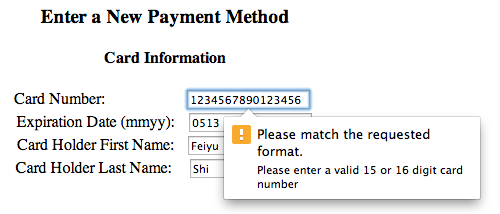


1. Payment Method

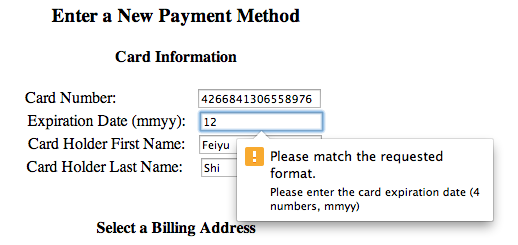
Input form 1:



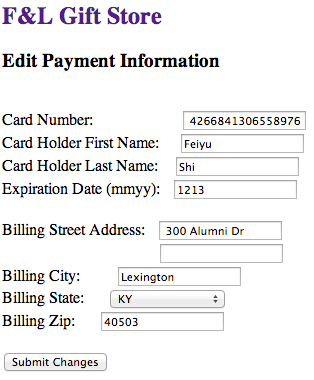
Input form 2 (error):



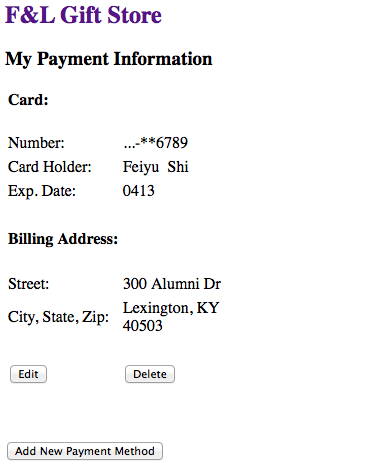
Input form 3 (error):



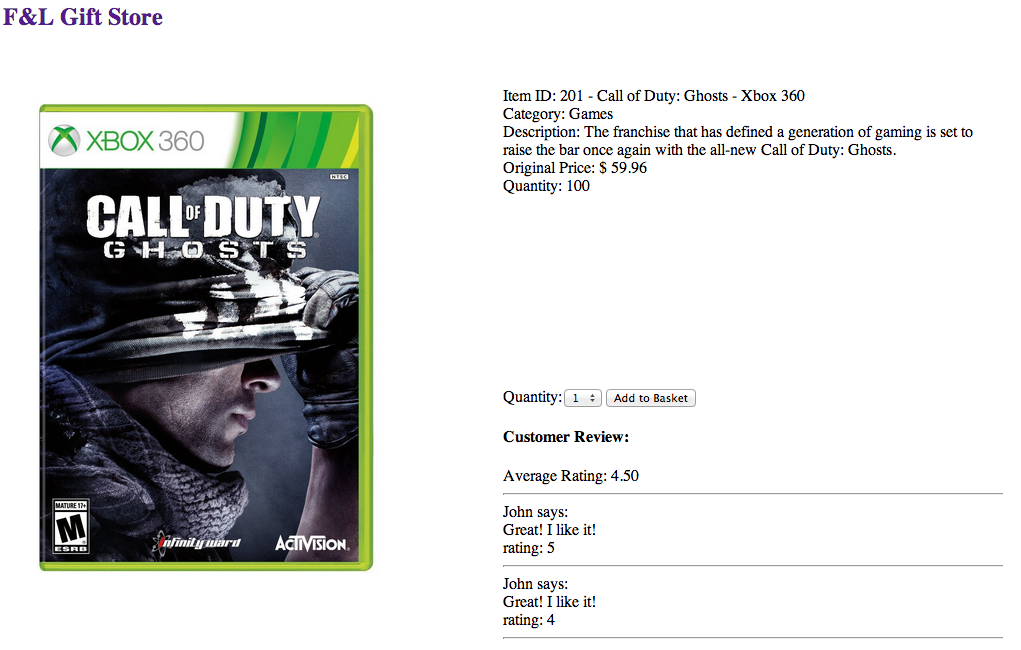
Input form 4:



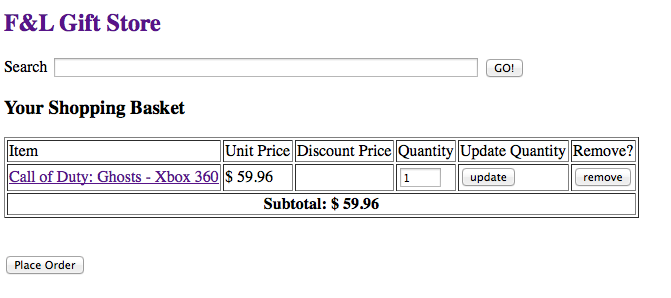
Output form:



1. Item Page

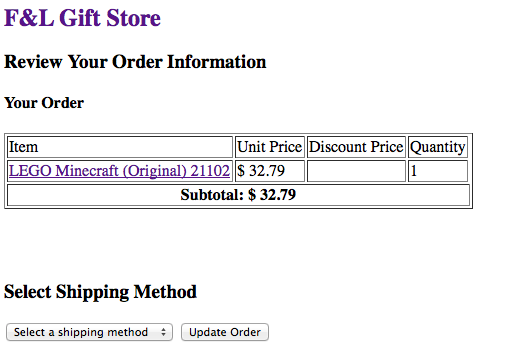


1. Customer Basket

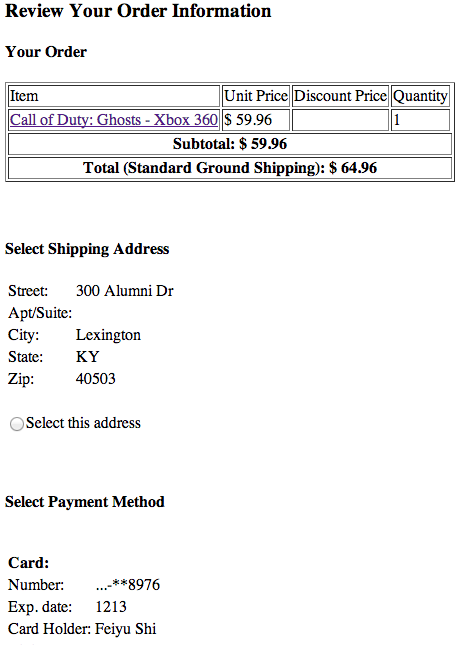


1. Order Placement

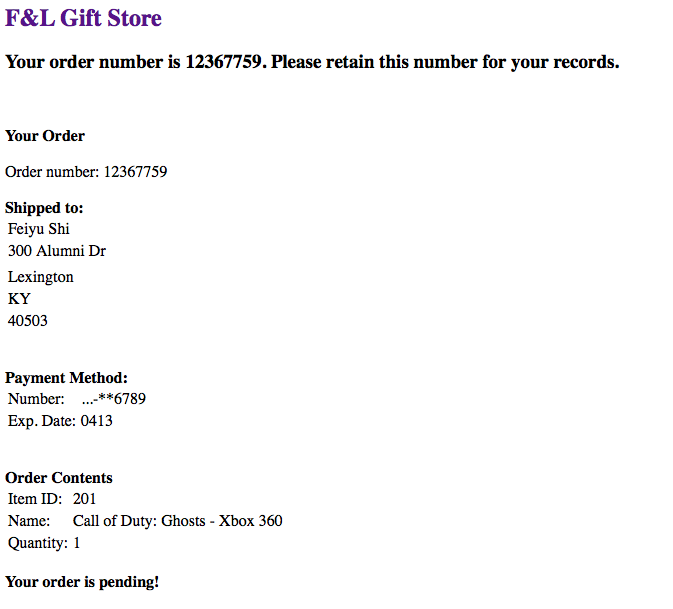
Output form 1:



Output form 2:

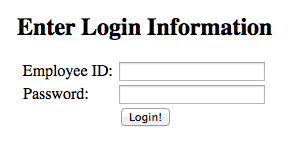


Output form 3:



1. Employee Login

Input form:

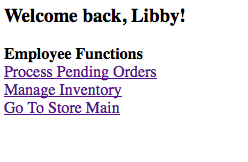


1. Employee Home

Output form 1:



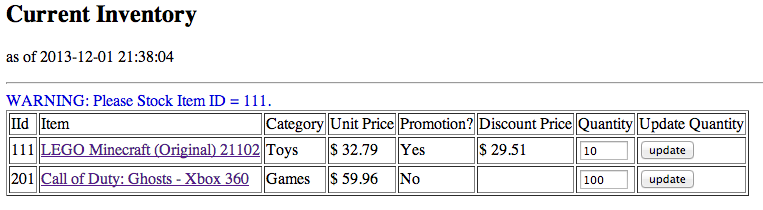
Output form2:



1. Pending Orders

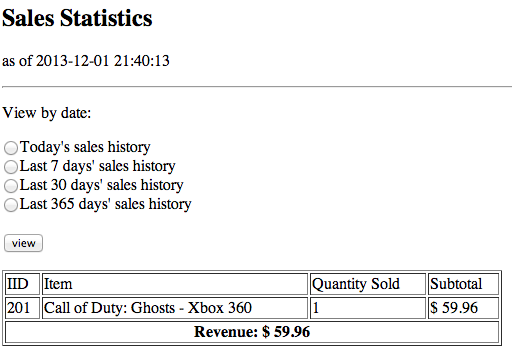


1. Inventory Management

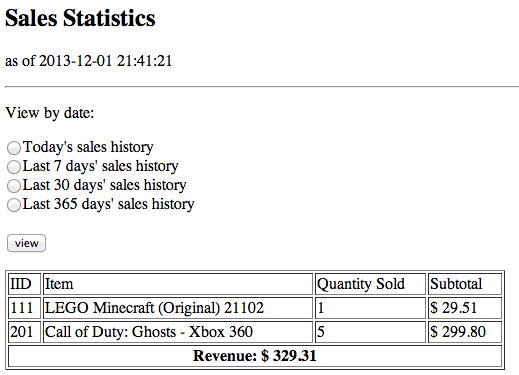


1. Manager Statistics

Output form 1 (sales, today):

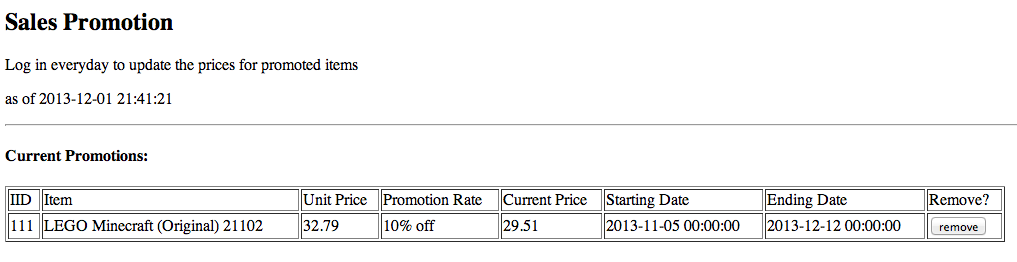


Output form 2 (sales, this year):

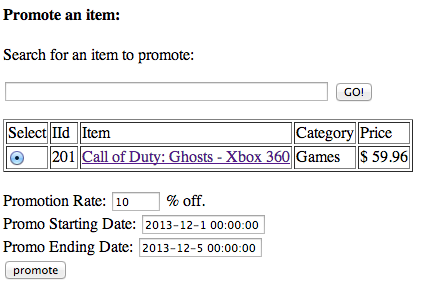


1. Manager Promotion

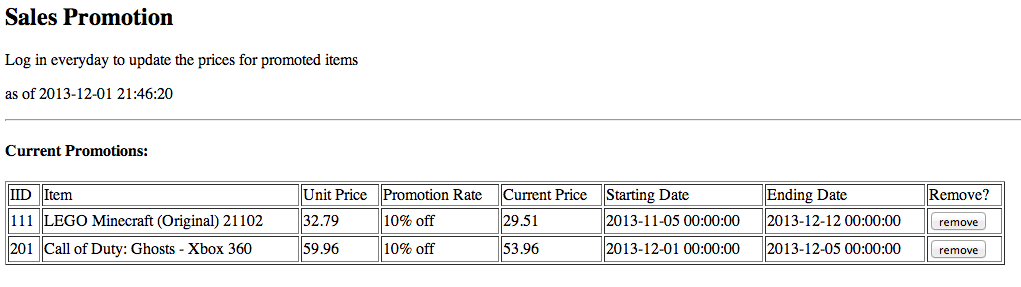
Output form 1:



Output form 2:



Output form 3:



1. **Testing**
2. Testing Process
3. Customer Account Registration

We test it by feeding in incomplete contents or contents with wrong format, and then all registration are blocked. If successful, we can observe the customer information in the database.

1. Customer Address Book and Payment Method

We test it by adding address and payment method and check them in the database. We can also edit that information and “delete” it. We actually can’t delete address and payment method from the database, because of the foreign key constraints, but we can prevent customer from seeing it in their account page.

1. Customer Basket

We add items from item page, and then check the basket. If it’s all right, it will display the item with correct quantity. We edit the quantity, if it’s negative or over the amount of stock, it will show error information. We can also remove the item. All corresponding information can be check in the database.

1. Customer Placing Order and Order History

When click place order button, we can first update shipping method and then select shipping address and payment method from address book and payment method storage. After all that, we can actually place the order and review the order information on the confirmation page. Order data could be seen in the database, customer order history page and the pending order page in the employee panel.

1. Employee Pending Orders

Employee should see all the pending orders. When click the ship button, corresponding inventory is updated and the order status is changed in customer order history.

1. Project Experience

This project was a huge learning experience for both of us. Neither of us had any previous experience with PHP, mySQL, or any sort of web programming, which at times made the project quite difficult. We found that sometimes problems that seem trivial, such as passing multiple data items through POST with one button click on an HTML form, can often be the most time consuming. However, one of the benefits of working on a team is that two heads really are much better than one; together we were able to solve every problem we found, no matter how frustrating.

We also learned quite a bit about the benefits of more strenuous design review. We were satisfied with our database design and ER diagram when we started coding; however, in the last week of the project we discovered a huge source of redundancy when we did normalization on the design. This required quite a bit of restructuring, recoding, and breaking things that were previously working. It was a frustrating experience but again we resolved the issue and learned a valuable lesson.

We both agree that perhaps the biggest lesson we learned, however, was to do with customer requirements and design specifications and how those translate into a project. Though the requirements may have looked very simple at times on paper, they often ended up being quite complex, moreso than we ever expected. For example, adding in the ability for customers to place orders was one of the more difficult bits of coding and required quite a bit of thought about the structuring of the database and the SQL queries. There were also features that seem incredibly simple when you use them, but are extraordinarily difficult to implement. Store search, for example, is just one simple feature but was probably the most complex bit of code that was written. On that note, one of the most valuable things we learned was not to get too ahead of ourselves. In theory the project was simple; in reality it was complex and quite large in scope. We often found, even faced with the huge task of just fulfilling the design requirements, that it was easy to get caught in the trap of feature creep, which turned out to be just as dangerous coming from the developers as it would be from a customer. This again was where we learned the value of having a programming team, because one was always able to check the other. We are satisfied that we have fulfilled the requirements and, despite time constraints, managed to add a few extra touches that make the program much easier to use and much more realistic.

Overall, we both found the project incredibly rewarding, though it was also quite time consuming and at times frustrating. Both of us learned some very valuable skills for the workplace in PHP/mySQL, have a better grasp on project design, management, and planning, and most and best of all, have a much better idea of what it means to work on a programming team and how it can be greatly beneficial to coding projects. Though we wish we had more time to spend on some things, such as the GUI for the website, we are both extremely satisfied with our results and glad to have had such a great opportunity to learn.