Design Analysis PS2

Overview

Key design challenges

The main design issues I discovered are as follows:

- Properly counting the quantity of available items
- Sharing items among the store, shopping cart, and orders such that information is preserved properly
- Creating updatable order statuses

Details

Data representation

The schema differs from the object model in the way that items are represented. Each Store object has many instances of actual Item objects, however, each Cart object, Order object and Saved Object have many “mirror” item objects, called cart_items, order_items and saved items respectively. In the case of the cart and saved objects, they just contain the item_id as a means of referring to the Item object. Order items, however, contain a unique set of attributes such that they are a snapshot of the cart item at the time when it was ordered. This preserves the integrity of orders, as they will not reflect post-purchase changes to item price or name.

The Cart is checked out and subsequently transformed into an Order. All items in the Cart are duplicated as Order Items, to preserve their features. The Order itself belongs to the user who places it, but all storekeepers whose items were purchased within the order will also receive notification that an order was placed in their store.

Users, in my database, are named Storekeepers.

Key design decisions

I decided to not allow users to edit the quantity of items in the cart, mostly as a measure of controlling the availability of items. In my original design, I opted to decrement the item count whenever and item was added to a cart. However, this meant that items could be held indefinitely without purchase. My new design does not decrement the quantity of the item until a user has checked out and “purchased” the item. There will be some cases, now, when an item is no longer available after the user places it in their cart. I handle this in two ways. If an item’s quantity goes to zero while in the user cart, that item will be replaced with a message in either the cart or saved items stating that the item is no longer available. If an item’s quantity goes to zero when the user places their order (i.e. there was only one left, but the user had two in their cart), the checkout confirmation page will reflect that only one item was purchased, and apologize for any inconvenience.

I addressed my handling of separate types of items in the data representation section. Cart items and saved items should reflect changes in prices or name, while order items should preserve the state at the time of purchase.

In my design, order objects are owned by the customer, since an order with items from different stores is seamlessly integrated into one cart from the shopper perspective. Thus, updating the order status (from “Pending” to “Fulfilled”) is a challenge, since this change is the product of multiple storekeepers. As a result, I decided that each order item have a status, and that the order status itself is the sum of these statuses. When a shopkeeper fulfills a customer order, the result is that the items from their shop in the customer’s order change to status “Fulfilled.” The order status, what is displayed to the customer, only changes when all order item’s have a “Fulfilled” status.

I made the decision that all users must create an account and sign in to use the site. This allows for a more controlled experience and the ability to personalize in a better way. I also made the decision to have a dynamic item library, i.e. any Storekeeper can create any Item they wish. This design adds more flexibility to the user experience and allows for creative license among users in the community.

My original design separates the abstraction of shopper and storekeeper. The new iteration of my design abstracts this as one in the Storekeeper object. The Storekeeper, only once signed in, has the ability to upload items, place items in the cart, and review incoming orders. Once I began to implement the shopping cart, I found it made more sense to create the whole user experience under one login, so that anyone can both shop and sell.

Design Critique

Summary assessment from user’s perspective
Overall, the user interface works well. It is clear how to sell and purchase items, and the divide between the two actions. Having one user login eliminates any confusion as well, given that a user in my design has the capability to both buy and sell items. The navigation through deeper parts of the site is difficult at times, i.e. adding an item requires a long series of steps and clicks, and once a user gets to this point it is unclear how to backtrack. It would be nice to have some sort of indicator for the user to know where they are in terms of the overall page flow.

Summary assessment from developer’s perspective
My design allows for differentiations between orders and carts, via separate order items and shop items. This is a strong solution, since shop items are dynamic while order items are immutable. For example, when a storekeeper alters an item, if another storekeeper previously bought it, the item will not reflect these changes and instead show the information that was true at the time of purchase.

I think that incrementing the item count only at checkout is another good decision, as it solves the problem of items being placed in carts indefinitely. While there is no guarantee for the customer in terms of purchasing the item, this simple design avoids deadlocks.

Overall, my specs were good, as was the separation between model and controller; most methods were kept in the model. However, there are a number of variable misnomers that

can lead to some confusion. Shopkeeper, the name of my user model, is misleading since shopkeepers can not only sell but make purchases as well. Order items have an attribute called item_number, which refers to the id of the shop item it mirrors. This attribute has a misleading name, as it is unclear whether it refers to a quantity or id. A better name would have been original_id.

Most and least successful decisions
The best design decision I made was to clone items from the cart to the order. This is good because it allows for storekeepers to alter the original item without affecting user's who had already made the purchase, an especially important characteristic for shopping carts used in online stores.

[This problem has been solved in my new design] The worst design decision I made was to create and use pointers from the order items to the original store items. While my design allowed for the flexibility described above, my implementation revealed a number of inconsistencies, such that when an item was edited by the storekeeper, these edits were incorrectly reflected in previous orders of that item.

Analysis of design faults in terms of design principles
[This problem has been solved in my new design] Using pointers from order items to the original store items is a misrepresentation stemming from my original object model. My object model should have differentiated between the two, since they are in fact separate objects, but instead lumped all items into one object.

The mislabeling of item_number and certain methods was the result of not adapting names as the code evolved. The code is obviously dynamic and sometimes it is difficult to keep up with these changes and make sure attribute names properly reflect the underlying design. Finally, the very layered navigation is a result of using a number of different models and controllers. While this is not on its own a fault, there needs to be more transparency between the design and the user experience.

Priorities for improvement
[This problem has been solved in my new design] First priority is fixing the cloning from store item to order item so that more information is stored in the order item, as opposed to just a pointer back to the original.

The second is making the navigation through the site more clear, perhaps using some sort of header. The last is to fix attribute and method names to eliminate confusion.