Design

Overview

Purpose & Goals

The purpose of this assignment is to create a shopping cart with different interfaces for a customer and for a seller. The customer should be able to add and remove items from the shopping cart, view their shopping cart, and place an order (without implementation of a payment transaction). The seller should be able to view and edit their listing of products, and view/edit orders from their customers.

The main goal of this assignment is mainly to understand how user authentication and user accounts would work in a rails application, as well as security issues that come up because of user accounts. A sub goal would be to further understand how interactions between different models and databases work, and this will be built upon knowledge obtained from project 1.

There are currently many implementations of shopping carts--any online shopping system will have one. Many of these are very well done, with smart interfaces that allow easy navigation and order placement for users. Because an online shopping cart will have direct implications on a store's profit margin, it is a crucial part of the shopping experience, and This project does not seek to improve on the shopping cart system, but rather serves as a mean to understand the goals outlined above.

Context diagram

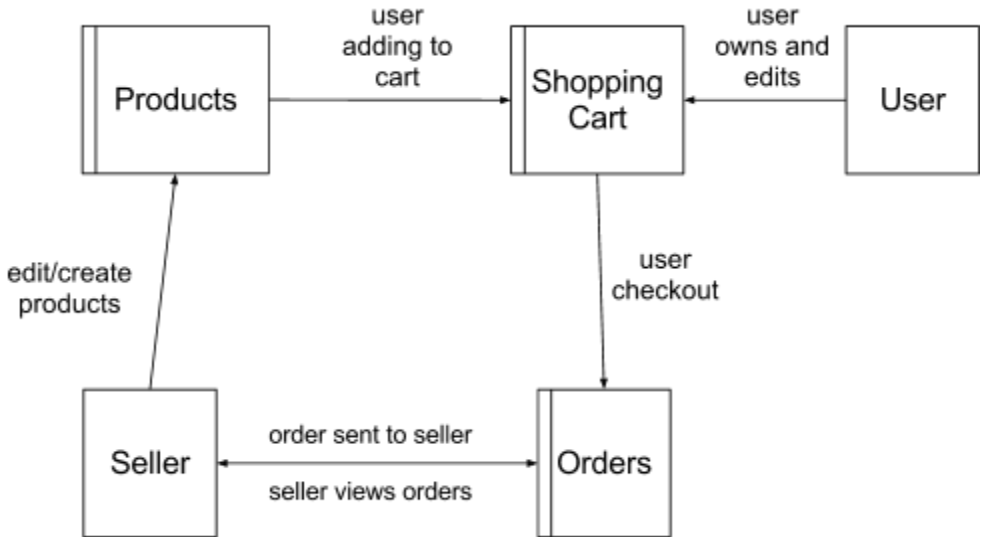


Figure 1. Context Diagram for shopping cart

Figure 1 above shows a simple context diagram for the shopping cart. There are two main external elements: the seller and the user. The seller can edit and create products. They can

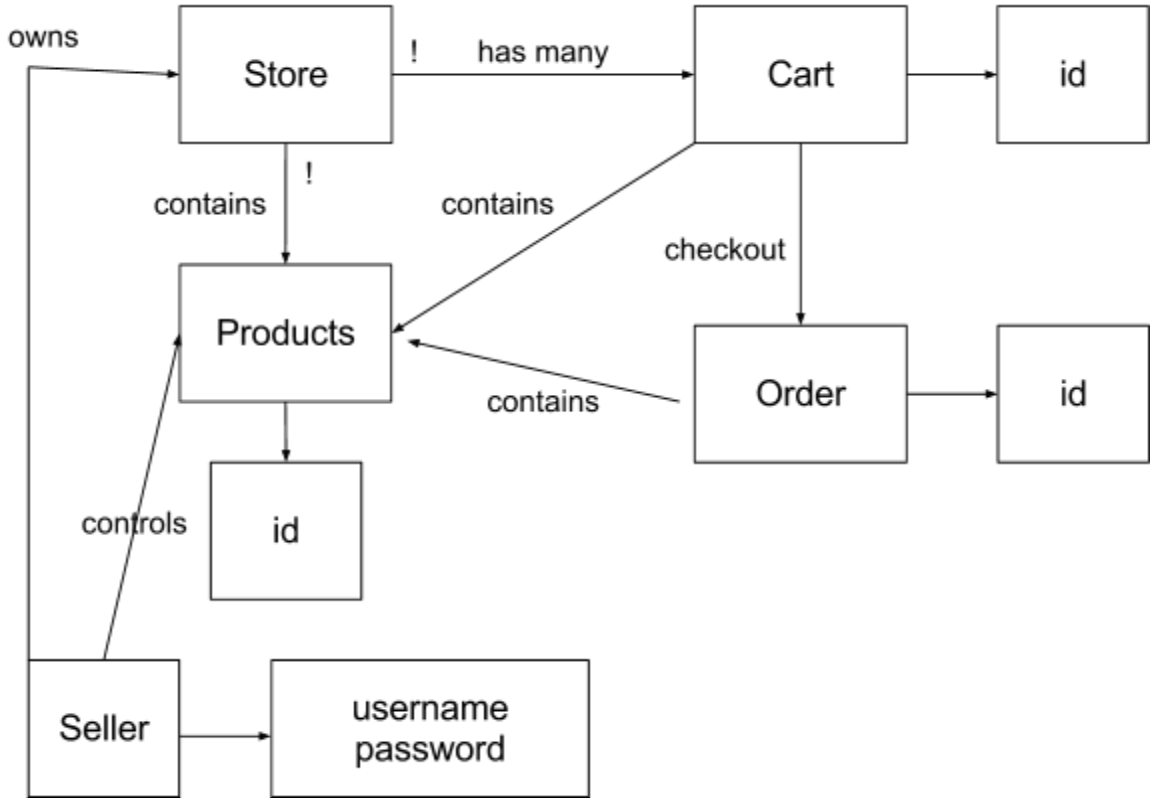
also view orders and process orders. The user owns a shopping cart, and edits it by adding or removing products from it. The internal system consists of a index display of available products, as well as a shopping cart that is specific to a user. The shopping cart translates to an order when the user checks out.

Concepts

Key concepts

1. A *product* is some sort of good that the seller wants displayed to the customer via a product index
2. A *cart* is a collection of products that the customer has chosen and wants to purchase.
3. An *order* is the final collection of products in the cart; it belongs to a customer, and is displayed to the seller.
4. The *seller* is the user that controls the product listings. The seller is able to access the list of orders submitted by the use of authentication
5. *Authentication* serves to identify the user and is used to limit access to certain parts of the website appropriately. In this project, user accounts will serve as the method for authentication.
6. *User accounts* is the method through which the website can identify the type of user (customer or seller). This will allow the site to authorize the seller. The seller will be able to login and logout; only logged in users will be able to access the list of orders.
7. *Sessions* are used to identify that the same customer is on the site while browsing separate pages. This is crucial when implementing a shopping cart as the cart must be the current representation of the customer's shopping cart. This may be done using browser cookies or using server side sessions.

Object model



Behavior

Feature descriptions

Customer

Be able to browse all products available. Customer will be able to add and remove items to and from the shopping cart. Shopping cart allows checkout, which brings the customer to a form to fill in shipping information. This is then submitted as an order.
Customer should be able to see the total price, change quantity, and empty the entire cart if needed.

Seller

Be able to view a list of orders that have been submitted to the website. This list of orders has to be private, and so can only be accessed via authentication. This will be implemented with a login/logout system. Seller should also be able to create/remove/edit the list of products available.

Security concerns

Security requirements:

1. Non-seller cannot edit/create products, and cannot access order listing
2. Cookies should be cleared after exiting window to minimize ‘sharing’ sessions between one customer to another
3. Seller can only access order listing using a username and a password
4. If the seller forgets to log out, the site should automatically log the seller out, so that the next user at the computer will not have access to orders list
5. Only admin user can create more users

Possible attacks:

1. Session hijacking by seizing cookie used. To minimize security risks from thise, large objects will not be stored in the session (only the relevant ids will). Cookie storage can also be secured using a secret.
2. Session Fixation, where the attacker may fix a session id known to him. In order to counter this, the site must issue a new session identifier and declare the old one as invalid, and this is done using reset_session method in rails.
3. Sessions that don’t expire are exposed to attacks identified above, so the best thing to counter this is to expire sessions on the server, and delete sessions that were created a long time ago.
4. Mass Assignment in model increases vulnerability for the database, and any relations extend the vulnerability by exposing the data columns. This can be mitigated with attr_protected method.

User Interface

Customer:

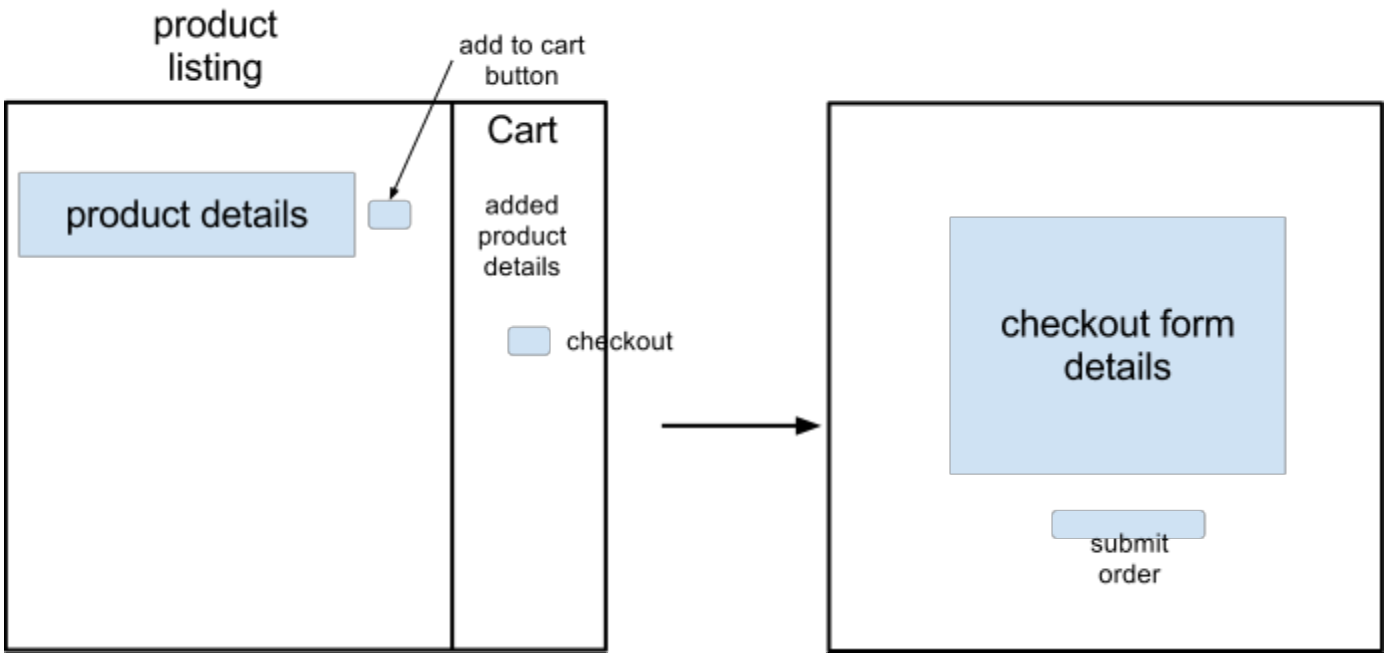


Figure 2: Customer flow

Figure 2 above shows the customer flow. As seen, the cart is implemented as a partial template, which means that any addition or removal from the cart does not require the page to refresh, which translates to a smoother shopping experience for the customer. The only page change occurs when a) the customer looks at the next page of products or b) the user checks out and submits an order.

If the cart is empty, the partial template for the cart will display a message of some sort (eg. “Your cart is empty!”)

Seller:

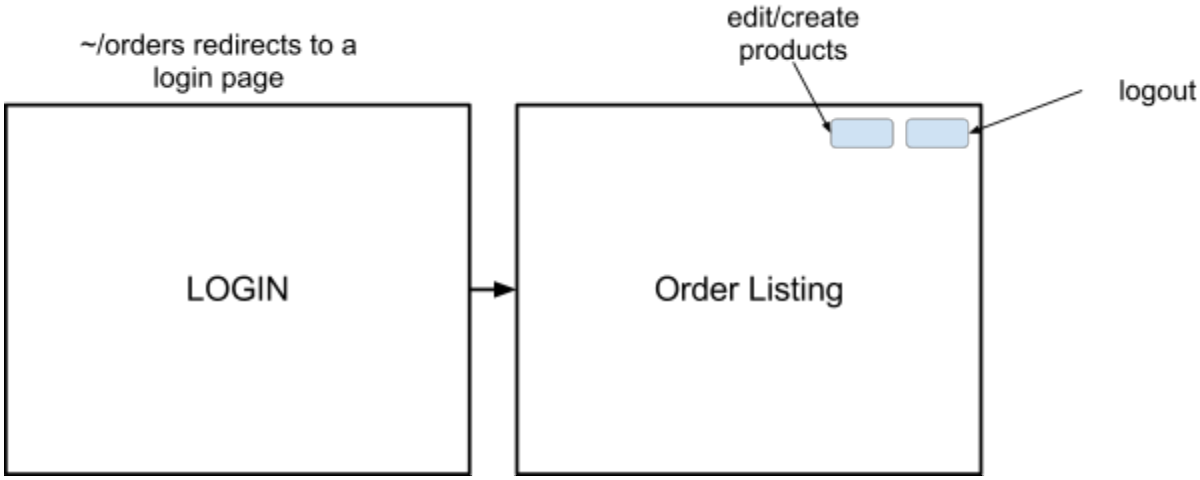


Figure 3: Seller flow

Figure 3 above shows the flow of the site for a seller. In order to access the order page, the seller must first login. After logging in, the orders are listed, with two links to a) edit/create products and b) log out. If window is close without logging out, the site should automatically log out for the seller to maximize site security.

Design challenges

Challenge 1: Seller authentication and authorization

A seller needs to be able to securely access the list of products and be able to edit and add to them, as well as view and process orders. This requires authentication: identifying that the user is a seller, and authorization: giving the user the ability to do the above.

The best and most common option is to use user accounts, via a login and logout system. Only sellers will be able to access certain parts of the sites when they are logged in. This will be implemented as a database of users, with a default user already created in the database. Once logged in, the seller will be able to change product listings as necessary, view the orders, and create new user accounts for other administrators. Each time a seller logs in, he/she will create a new user session. This user session will be implemented server side.

Challenge 2: Shopping cart that persists through website browsing

Customer must be able to see all the current contents of his cart while browsing through all the pages. This cart must persist when multiple pages are opened.

One way of implementing this is to use cookies to store information. This means that if the user were to access the site on a different page, the information will be available for use. This is a common implementation, and is the optimal choice for a shopping cart. However, cookies are sent with every request between the browser and the server, and if the shopping cart is large with a lot of data, this would mean a lot of data being sent between. This is avoided by the

creation of a cart model, so the only data being sent is the model id.

Using cookies opens up concerns in terms of security. This is because the cookie is used as a temporary authorization for a session--if someone hijacks your cookie, they thus have hijacked your session. In the context of our project however, the security holes cookies open will have minimal impact. A customer session can be defined as the time from browsing and adding to the shopping cart, to submitting an order. There is no confidential customer or seller information being released or stored in the cooke, as all it contains are the items in that cart. The cookies will be used to identify the unique cart of a customer, and when the customer submits an order, the cookies will then be destroyed.

Challenge 3: Prevention of accidental duplicate orders
There is a high probability that an impatient user will click the “submit order” button multiple times, and thus submit a duplicate order. This is obviously not ideal for both seller and customer. There are several ways that this can be avoided.
1. We can simply add a warning message that tells the customer to not double click the button. While this may deter the bulk of duplicate orders, it is heavily reliant on a) the customer reading instructions and b) the customer following instructions. It is thus a very unreliable fix, and should be used in conjunction with something more robust.
2. Disable the button once it has been clicked for a certain period of time (say 5 seconds). This would indicate to the customers that the order has been submitted, and remove the ability for customers to submit duplicate orders. This is a good fix as it a) updates the customer on the status and b) is pretty robust. This will probably be the implemented solution.
3. An additional way of ensuring there are no duplicate orders is to have a helper method on the server side that checks the orders in the database. If all fields are identical, it would raise an error flag and delete the duplicate. This may be useful as an additional check, but is not completely necessary.

Implemented: When an order is submitted, the cart is immediately cleared. Thus, if a customer presses the button multiple times, the additional entries are submitted as blank order forms. What this means is that rather than checking for duplicate entries, all the database needs to check for are empty entries, where there are no products.

Challenge 4: Adding multiples of the same product
When customers are adding items to the shopping cart, they shouldn’t have to physically add an item twice; rather, they should be able to add an item once, with a quantity of two. Similarly, the order should display the product and its quantity as opposed to two listings of the same product.

This will be done by adding quantity as an attribute and incrementing this as necessary. While this challenge is not necessarily a hard fix, I brought it up as it may be a feature that is overlooked. For example, removing a product completely from the shopping cart is different from reducing the quantity of the product, and there needs to be the appropriate methods to handle this difference.

Challenge 5: Exiting site before checking out
When customers exit (and by exit, we mean closing the window) before checking out, the site has to handle the current shopping cart. The best way to deal with the cart would be to remove all items from the cart. Another option is to save the cart; however, this may not be the most optimal as many customers may share the machine.

The model would thus define a method that clears the cookies when the customer exits the site, so that the next time the site is visited, the cart will be empty. While it is not necessary in this project to delete the cart from the cart database, it might be more optimal spacewise to do so, especially if we were to actually deploy this shopping cart to production, where there may be millions of shopping carts.

Code Design

Scaffolds: User, Products, Cart, Order
Controller: Store

Store
Only used to display products, an optional controller that may be useful in customizing the look of the shopping site.

User (Used for seller access)
id:integer (auto generated)
username:string (validate -> longer than 6 characters)
password:string (validate -> longer than 6, upper and lower case letters)

Session (Used for logging in and logging out)
session_id

Products
id:integer (auto generated)
name:string
description:text
picture:string (validate -> url to picture ending in jpeg, png or tiff; optional)
price:double (validate -> has to be greater than $0.01, scale to 2 decimal places)

Cart
id:integer (auto generated)
product_id (id of product added to cart)
quantity (quantity of product)

Order
Basically stores cart items into an orders database. This is the database that would be viewed by the seller; contains all the product information.

An order has many ‘carts’ (I put this in quotations because it only has carts that have been submitted); carts have many products.
A customer owns a cart, and when submits a cart, owns an order.
The seller owns the products, and has many orders.

Line items
an order has many “line_items”, which are essentially just lines in the database when you add a product to the cart. This is segmented from the Order, as it makes more sense relationship wise → an order consists of many line items. This also simplifies methods such as adding or decreasing the quantity of a product.