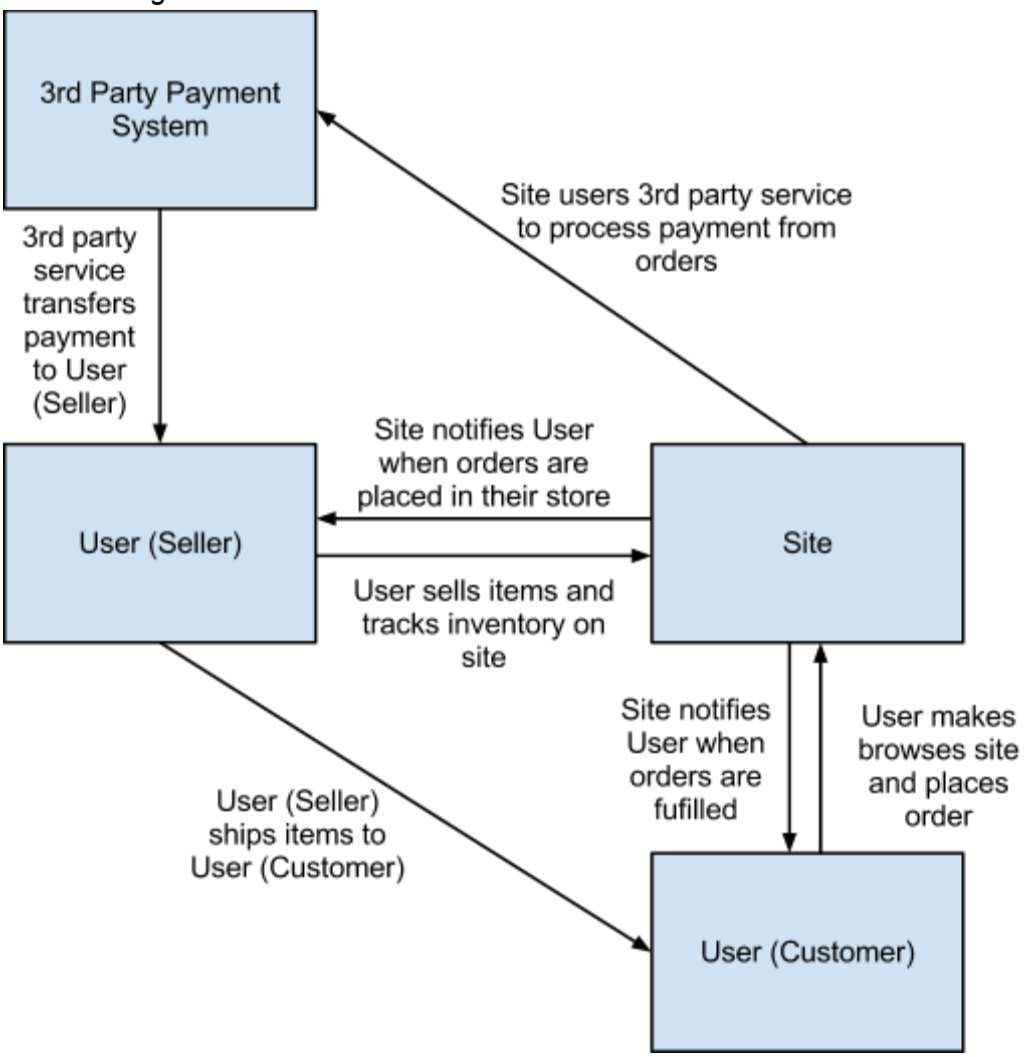
Problem Analysis PS2

Overview

Purpose and goals

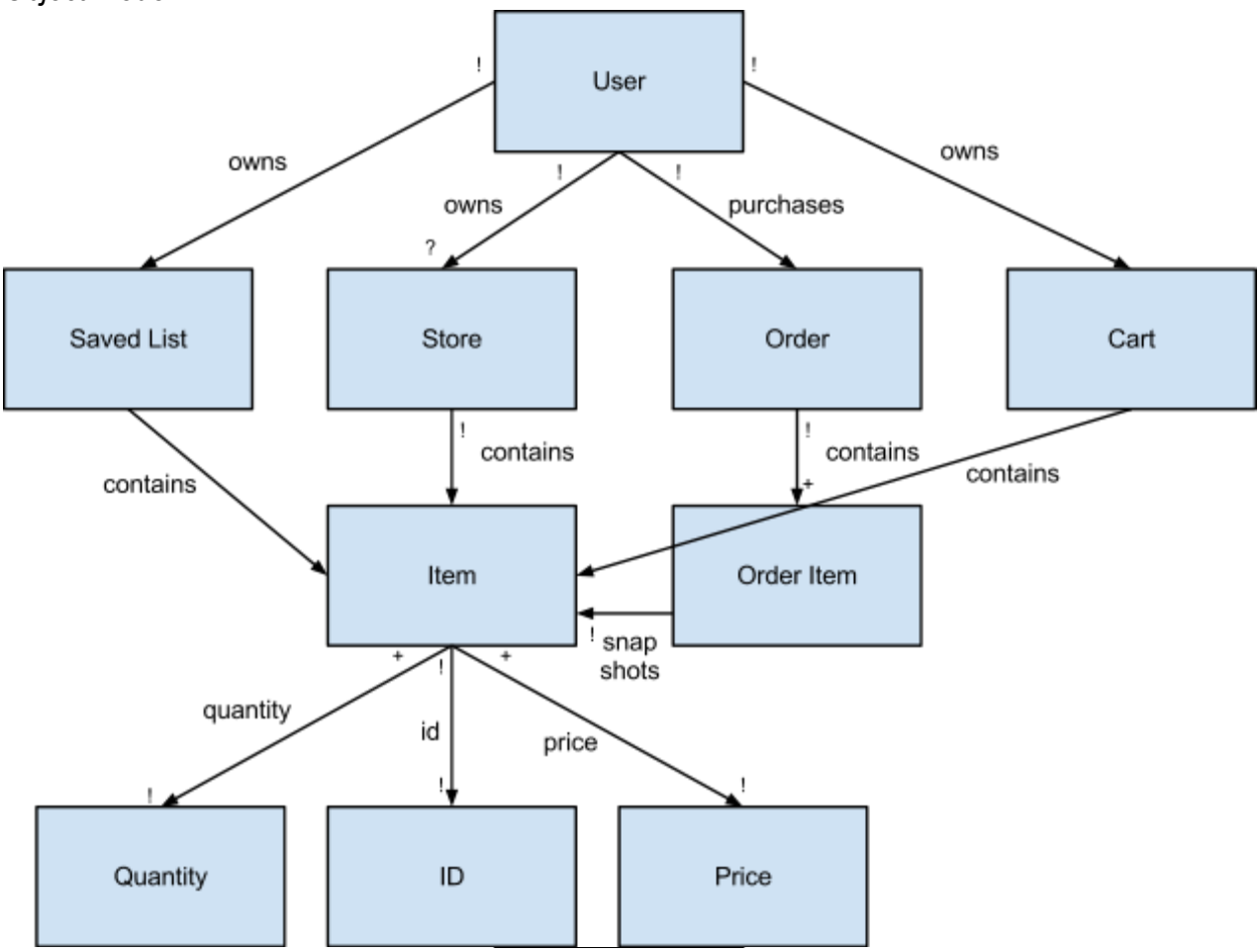
The goal of this product is to create an intimate community for users to both purchase merchandise from their online “neighbors” and curate their own storefront. The product will make it easy for customers to browse and shop products from multiple vendors, but still checkout and process payments as if buying from only one storefront (think Amazon). The interface is simple and intuitive, better promoting interactions between customer and seller. Existing solutions aren’t geared towards a community of customers and storekeepers, so this product will lower the barrier separating the two parties.

Context diagram



Domain

Object model



- When a user places an order, this order contains order items, which are snapshots that will preserve the original characteristics of the purchased store item, even if they change over time.

Event model

User Login
User Logout
User Create Store
User Add Item
User Edit Item: edit the price or quantity of an item
User View All Stores: view all stores, whether items are available to purchase or not
User View Store: view all items in a particular store available to purchase
User View Profile: user profile contains the fol information regarding your store, orders you've placed, and orders placed within your store
User Fulfill Order: mark items from a customer order as "Fulfilled", i.e. as the storekeeper you have shipped the products

User Add to Cart: add item from particular store to the cart
User Checkout: places the order; can only be placed if items are in cart and items available
User Save Item: save item from cart for later in the saved list
User Unsave Item: unsave item from saved list and add it back to the cart

user ::= loggedOut (register | login) loggedIn (shoppingEvents | storekeeperEvents)* logout
Any user must login in order to interact with the site.

shoppingEvents ::= (viewStore (addCartItem viewCart (saveItem unsaveItem?)*) checkout?) | viewProfile
Users can browse stores, adding items to their cart and only after items are in the cart can a user save and unsave items. Users may or may not check out. Separately, users may view their profile and see orders they've placed in the past.

storekeeperEvents ::= viewProfile createStore (addItem editItem*)* fulfillOrders?
Users may create a store and view it on their profile, at which point they can add items and edit items if wanted. After users create and stock their store, they may potentially fulfill placed within their store.

Behavior
Feature descriptions

- User login: To access the site's features, all users must register and sign in. Once logged in, users are automatically able to shop and they may also create a store and sell items if they wish.
- Shopping cart: All users own a unique cart that tracks items across sessions. Shoppers can add items from any store into one cart, view the cart and proceed to checkout seamlessly, as if all the items were contained in one store.
- Saved List: Shoppers can save items from their cart into a saved list for later, and proceed with checkout minus the items in their saved list.
- Shopkeeper Homepage: Users who choose to create a store can easily view the status of incoming orders, fulfill orders, update the status of pending order, edit current merchandise, and add new items.
- Shopper Homepage: Users can view all merchandise from all shopkeepers organized by stores. They can also see the status of the orders they have placed, along with a list of all past orders.
- Checkout: Users can checkout when there is at least one item in the cart, and its quantity is greater than zero. There is no 3rd party payment system is linked at the moment, so all orders that go through checkout are purchased.
- Search: Users can search for items by name on the homepage, which leads them to the appropriate storefront.

Security concerns

One sensitive part of this project is the payment procedure. In this case, all payments will be handled by a third party (think PayPal), which eliminates the sites responsibility for this risk.

Other concerns are users trying to access the order information of other users, users trying to administer stores that don't belong to them, and users infiltrating the database.

At all sensitive pages, there are checks to ensure that the user is both logged in and the "correct user," i.e. the owner of the store if editing store items or the owner of the cart if proceeding to checkout. This means that even if a user types in the appropriate URL to edit a store, they will only have permissions to view and edit if they are the store owner. Sessions are hashed and also renewed at each login, to prevent attacks on this front.

I also made use of attr_accessible in my models and all queries to the database are made in ways that prevent SQL-Injections. To protect from CSRF, I added the protect_from_forgery line. The authlogic gem salts user passwords, so they are not stored in the database as they are typed by a user. Even if a user infiltrated the database for passwords, the password data stored would be useless.

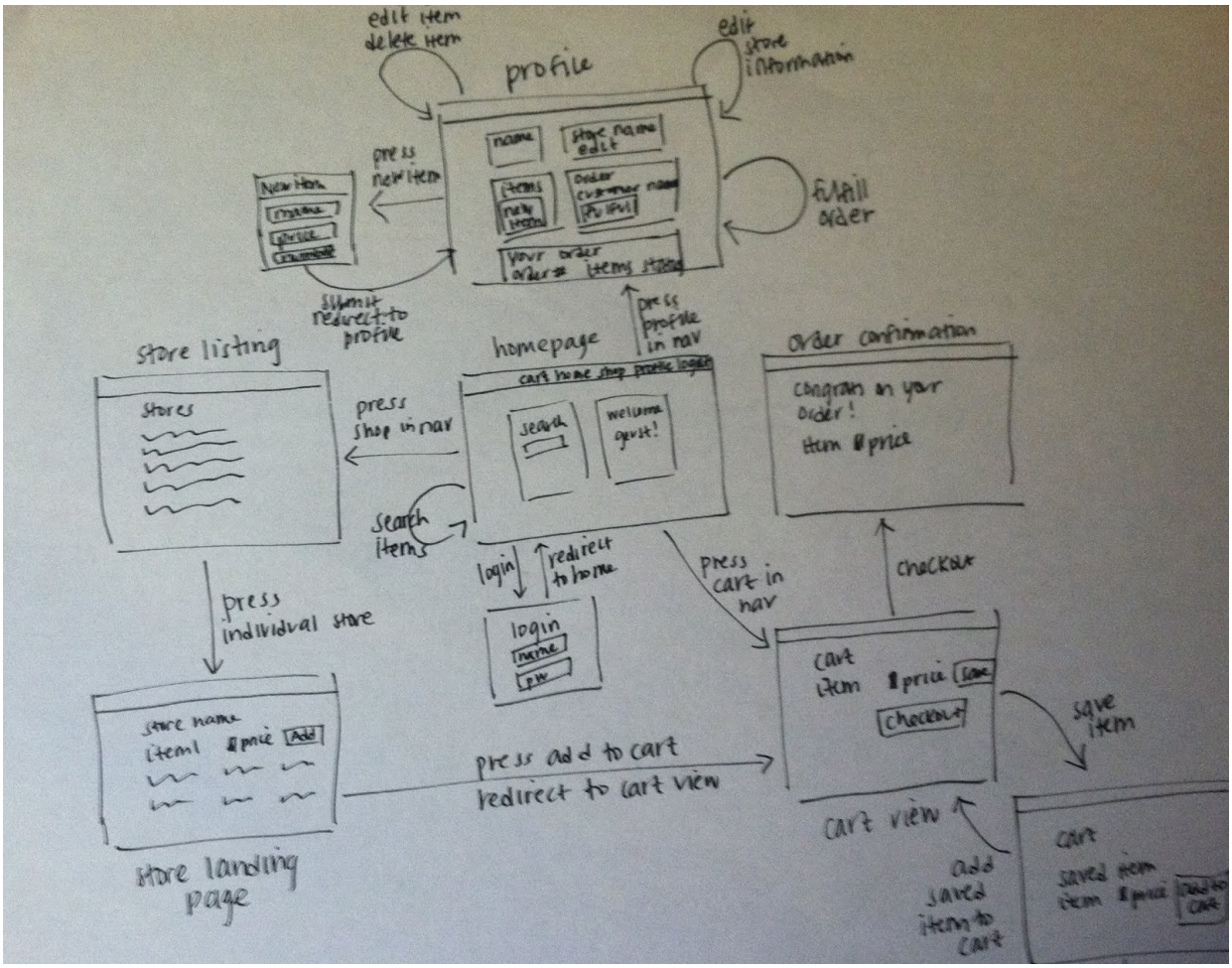
Operations

- GET /
    - Effects: returns home page prompting sign in or registration.
- POST /signin
    - Requires: a valid username and password.
    - Modifies: session.
    - Effects: authenticates a user with a username and password and creates a new session if successful.
- DELETE /signout
    - Requires: user is logged in.
    - Modifies: session.
    - Effects: destroys the session.
- GET /signup
    - Effects: returns a form prompting the creation of a username and password.
- POST /signup
    - Requires: a valid username and password.
    - Modifies: users table.
    - Effects: creates new account with username and password and creates a new session for the user.
- GET /stores
    - Requires: user is logged in.
    - Effects: returns a list of all stores
- GET /stores/new
    - Requires: user is logged in and does not already have store.
    - Effects: returns a form prompting for name of new store.
- POST /stores

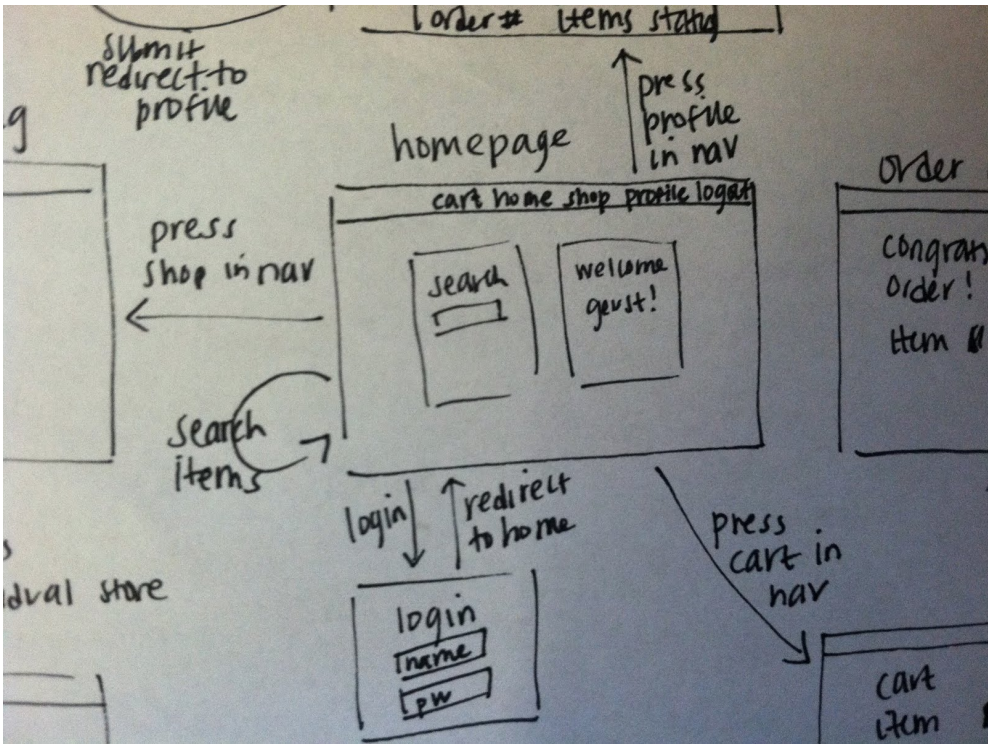
    - Requires: user is logged in and does not already have store. A store name.
    - Modifies: stores table.
    - Effects: creates new store that belongs to the current user.
- GET /stores/store_id/shop
    - Requires: user is logged in and store with store_id exists.
    - Effects: returns list of products within the given store. If the current user doesn't own the store, products are displayed on if they have a quantity greater than zero.
- GET /stores/store_id/items
    - Requires: user is logged in and owns store that holds item with given item_id.
    - Effects: returns a form prompting for name, description, price and quantity of new item.
- POST /items
    - Requires: user is logged in.
    - Modifies: items table.
    - Effects: creates new item with given name, description, price and quantity belonging to the given store.
- GET /items/item_id/edit
    - Requires: user is logged in and owns store that holds item with given item_id.
    - Effects: returns a form prompting for a new name, description, price or quantity of the specified item.
- PUT /items/item_id
    - Requires: user is logged in and owns store that holds item with given item_id.
    - Modifies: items table.
    - Effects: updates attributes of the item with given item_id.
- DELETE /items/item_id
    - Requires: user is logged in and owns store that holds item with given item_id.
    - Modifies: items table.
    - Effects: designated item is destroyed.
- GET users/user_id
    - Requires: user is logged in and id matches given user_id.
    - Effects: returns profile information including user's store information and order history.
- GET /users/user_id/cart
    - Requires: user is logged in and id matches given user_id.
    - Effects: returns list of all items within the cart and within the saved list.
- POST /users/user_id/basket
    - Requires: user is logged in and id matches given user_id.
    - Modifies: cart_items table.
    - Effects: creates new cart_item that belongs in the cart of the current user with a pointer to the item with the given item_id.
- POST /cartitem/cartitem_id/save
    - Requires: user is logged in and cart item exists.
    - Modifies: cart_items table and saved_items table.

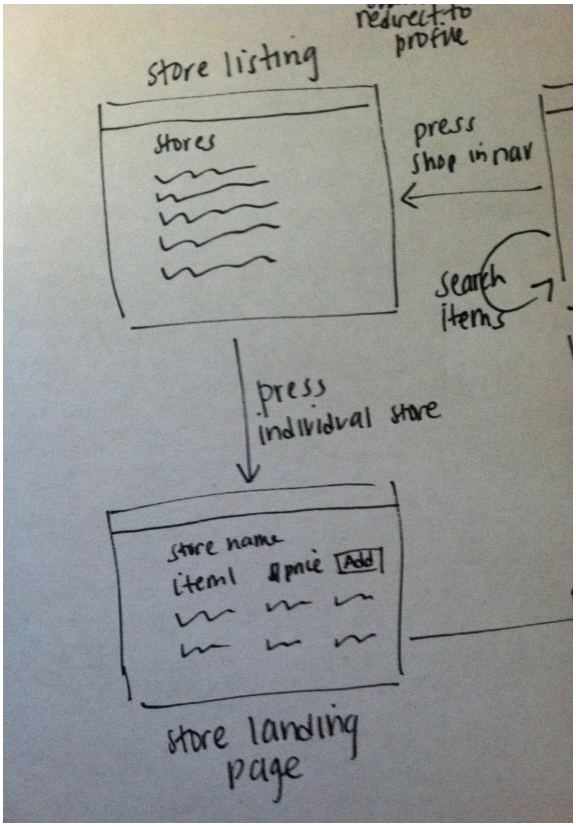
    - Effects: removes item from cart_items and creates a new saved_item with pointer to the same item_id as the cart_item had.
- DELETE /cartitem/cartitem_id/destroy
    - Requires: user is logged in and id matches given user_id.
    - Modifies: cart_items table.
    - Effects: removes item from cart_items.
- POST /cartitem/cartitem_id/addcart
    - Requires: user is logged in and saved item exists.
    - Modifies: cart_items table and saved_items table.
    - Effects: removes item from saved_items and creates a new cart_item with pointer to the same item_id as the saved_item had.
- POST /users/user_id/checkout
    - Requires: user is logged in and id matches given user_id. At least one item in the cart has quantity greater than zero.
    - Modifies: cart_items table, items table, orders table and order_items table.
    - Effects: new order belonging to current user is created.If item to which cart_item points to has a quantity greater than zero, a corresponding order_item is created and added to the new order. The cart_items are deleted and the item quantity decremented.
- POST /users/user_id/fulfill
    - Requires: user is logged in and id matches given user_id. Orders have been placed in the user's store and have a status "Unfulfilled."
    - Modifies: order_items table.
    - Effects: changes in the status of the order_items from "Unfulfilled" to "Fulfilled."

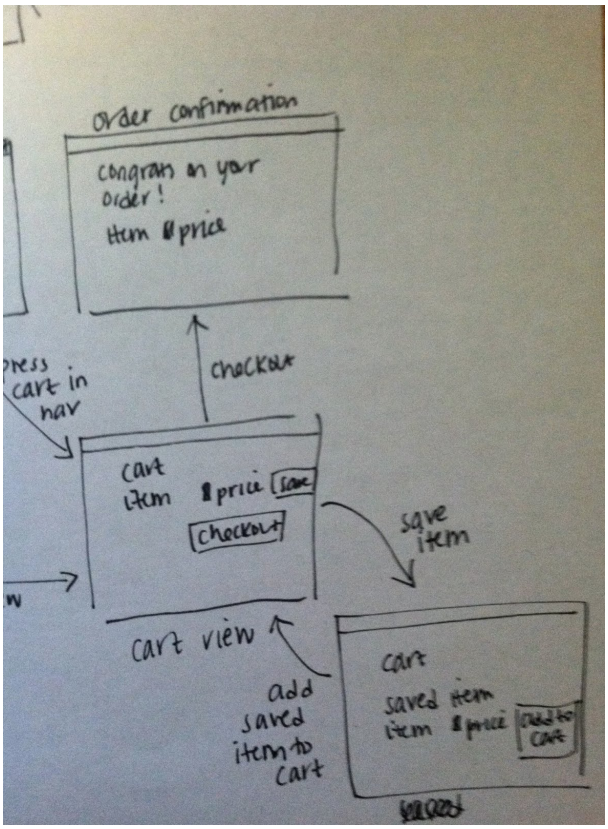
User interface

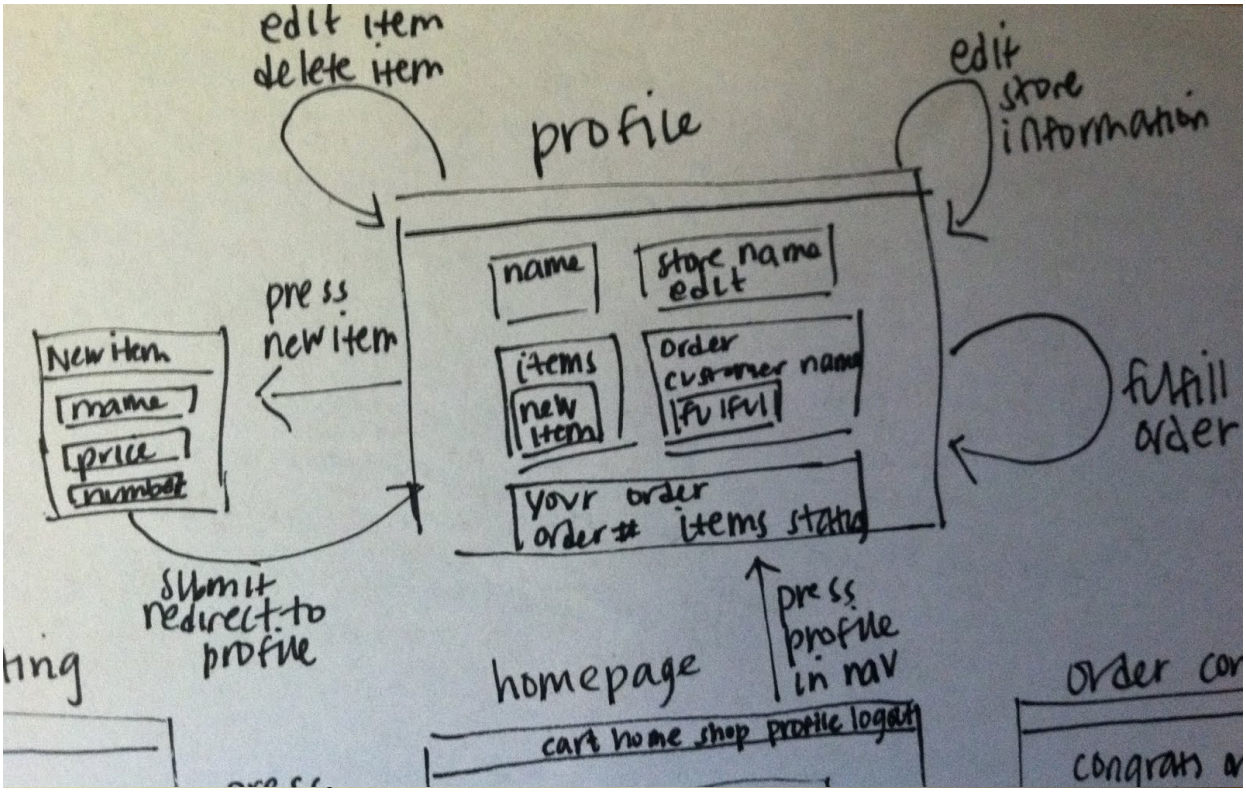


Details follow


submit
redirect to
profile
order # items stating
press
profile
in nav
homepage
order
cart home shop profile logout
press
shop in nav
search
welcome
guest!
congrats
order!
item
search
items
login
redirect
to home
press
cart in
nav
idual store
login
name
pw
cart
item


redirect to
profile
store listing
stores
press
shop in nav
search
items
press
individual store
store name
item
price
Add
store landing
page


order confirmation
congrats on your order!
item $price
press cart in nav
checkout
cart
item $price
checkout
save item
cart view
add saved item to cart
cart
saved item
item $price
add to cart


edit item
delete item
profile
edit store information
name
store name
edit
press new item
New item
name
price
number
items
new item
order
customer name
fulfil
fulfil order
your order
order # items status
submit
redirect to profile
press profile in nav
homepage
cart home shop profile logout