

SW Engineering CSC 648-848 Summer 2021
dropsell.gq, Jose's Angels

Team 03

Name	Email	Roles
Mitchel Baker	mbaker3@mail.sfsu.edu	Team lead
Krina Bharatbhai Patel	kpatel11@sfsu.edu	Frontend lead
Charmaine Eusebio	ceusebio1@mail.sfsu.edu	Frontend engineer
Rowena Elaine Echevarria	rechevarria@mail.sfsu.edu	Frontend engineer
Michael Schroeder	mschroeder@mail.sfsu.edu	Backend lead, Github master
Kenneth N Chuson	kchuson@mail.sfsu.edu	Backend engineer
Jamie Dominic Walker	jwalker5@mail.sfsu.edu	Backend engineer

Milestone 4
July 30, 2021

History Table

Version	Date	Notes
M4V1	07/30/2021	
M3V1	07/22/2021	
M2V2	07/20/2021	
M2V1	07/08/2021	
M1V2	07/02/2021	
M1V1	06/22/2021	

Table of Contents

Product Summary	3
Itemized list of functions	4
Usability Test Plans	6
Creating a new product	6
Creating an Auction Product	10
Adding products to cart / Checkout process	14
Price Matching	18
Seller/Buyer scheduling	22
QA Test Plan	25
Code Review	27
Self-Check on best practices for security	34
Self-Check: Adherence to original non-functional specs	37
List of Contributions	49

1. Product Summary

- Name of the product: Dropsell
- URL to the product accessible on deployment server: <http://dropsell.gg>
- How we plan to market and sell our product is two fold. First we will need some online advertisements to spread the name of our product. The top two methods of spreading our name brand online are pay-per-click (PPC), and paid social. PPC is one of the top methods for advertising your brand or product, but this involves competing with other big names for key search words. While this approach will work long term once we have established our name, it is not best for us at first due to the smaller purse and the fact that we are still trying to recuperate start-up costs. The second strategy of paid social is much more aligned with our goals. How paid social ads work is creating ads for social media platforms. This method allows us to target certain audiences that we feel our product applies to and since social media is one of the common pastimes of adults and teens this ensures we reach the most amount of people.
- The second method to build a strong user base is word of mouth. Once we get client traffic on the website we are confident that users will want to come back again and again. As users recognize the simplicity and value of Dropsell they will quickly recommend our page to their friends and family.
- What is unique about our web application drop-sell, is that we are developed from the ground up to be user friendly. Most if not all big time competitors simply added selling functionality onto their existing website. Dropsell is designed for the modern age for usability and comfort. What our web application does differently than others is that we provide tools to take the stress out of buying or selling a product. For example if you are unsure how much to charge for an item you can use our built in tool to quickly and accurately compare your product to what has sold previously or what is currently listed for sale. This takes the stress of guessing a price and underselling the product.

Itemized list of functions

1. Users for this application will be able to search for products.
2. A person that wishes to sell an item can choose to list that item directly to the market place or post it as an auction.
3. A user will be able to purchase an item by auction or buy it now style options, whichever the seller enables.
4. A user will be able to sort and filter search results to best suit their needs or interests.
5. A user will be able to see details related to their purchases in their account information.
6. If a user opted for delivery consignment upon purchase of an item, then they will be able to see tracking information in their account.
7. Instead of face to face deliver options, users can elect to have items delivered via consignment options(AKA shipping via ups for example)
8. A user shall be able to use website tools to price match items they wish to sell or purchase.
9. Any user must agree to the website's terms and conditions.
10. A user can message sellers with questions related to their products.
11. Users can add items to their shopping carts.
12. A user's shopping cart will update as the user adds or removes items from the cart.
13. A user shall be able to return back to shopping to further fill up their cart if they are midway through the check out process.
14. A user can choose to cancel their order or modify it appropriately.
15. A user can have details related to the purchase sent to their accounts for review at a later date.
16. When a user is checking out they can choose to add any additional information they want
17. When a user checks out they can choose different forms of payments.
18. A user shall receive a receipt/invoice after purchasing their item that will include all information required(ie, purchase price, name of seller, contact info, etc).
19. Users shall agree to terms and conditions prior to be granted access to

20. Each user shall agree to a seller's fee that will be deducted once that item sells.
21. A user will be able list products for sale and will list any pertinent information.
22. A user shall be able to edit a post that they made if they deem it appropriate.
23. If a user wishes to sell more of a certain item they will be able to adjust the quantity on their post.
24. A user shall be able to delete or remove an item if they no longer wish to sell it.
25. A user will be able to check all items they have for sale in their profile/account page.
26. A user will have different options for selling products(IE a user can choose the starting bid for an item, a user can choose if that item should last for 1-3 days, etc).
27. A user can set an item's duration on the website for a minimum of 1 hour, to a max of 30 days.
28. A user will be able to see pertinent information related to an item they are interested in purchasing(IE a user can see the current bid on an item, a user can see time remaining on an item, etc.).
29. A user shall be able to bid on an item with a single click.
30. A user shall see an accurate count down of time remaining on an item's listing time.
31. Users shall be able to contact other users.

2. Usability Test Plans

Creating a new product

- Test objectives
 - For this usability test plan we will focus on adding a product to the Dropsell web application for sale. This type of sale will cover simple purchasing from another user for the listed price. Auction type sales will not be included in this test, but will be included in its own test later. Testing this feature is important because besides auction type sales this is the only way for users to purchase items, and if you can't list an item for sale, then no one can purchase it.
 - Like the auction sale type mentioned later, being able to list products for purchase is the main reason for this application. Testing this feature will allow us to learn more about the user's experience in the most critical feature of this web application and the more we learn about the user experience the more we can shape it to be the best experience possible.
- Test description
 - The system was set up using a Linux Ubuntu system. I used multiple browsers, ranging from Brave browser, Firefox, to Google Chrome. When testing between browsers, there were no conflicts here when testing the creation of a new product.
 - The best starting point for testing the creation of a new product would be after logging in. Once the user is redirected to their Profile page, they should then click on seller settings which is where they will find the "Products" button which will display the component for creating a new product.
- Usability Task description
 - When thinking about the intended user base who would gravitate towards creating new products, people who are looking to clear out their garage, or sell old clothing for some extra cash come to mind. I had my girlfriend use the site to create a new product since she likes to post her used clothes online for people to purchase. She was a good candidate choice because she points out areas of the project which are lacking in UI/UX expertise. She is also a marketing major so she was able to point out details in the application related to advertising towards sellers.
- URL of the System to be Tested
 - The URL of the system which is used for testing the creation of a new product is on <http://dropsell.gq/seller-settings>, since the creation of a new product is located in the seller settings.

- Usability Test Table

Test/Use Case	% Completed	Errors	Comments
Locating Products in Seller Settings	80%	No errors found	Locating the "Products" in seller settings was difficult. These settings were too tucked away.
Inputting product data into the form	100%	N/A	Inputting data for the product title, description, category, price, and image were straight forward and intuitive to do.
Form validation when inputting invalid data	0%	Our nonfunctional requirements specify that product titles should not exceed 80 characters in length, while product descriptions should not exceed 500 characters in length. When testing for these system constraints, the application did not notify the user that the data inputted was incorrect.	N/A
Submit product	100%	N/A	Submitting a new product was simple. The submission button was easy to find, and I was notified when the product was successfully added.

- Questionnaires

I was able to find the Products section in seller settings with ease

☐ Strongly Agree ☐ Agree ☐ Neutral ☒ Disagree ☐ Strongly Disagree

Comments

I logged in and was redirected to the Profile page. I saw the seller settings option at the top but did not know at first that this was where creating a new product was located.

Added the data for creating a new product was straight forward to do.

☐ Strongly Agree ☒ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

Inputting data into the inputs was easy for me to do. I had no problem navigating between the title, description, category, price, and image inputs.

The submission button for creating a product was easy to find, and I was notified that my product was created successfully

☐

Strongly Agree

☒

Agree

☐

Neutral

☐

Disagree

☐

Strongly Disagree

Comments

After inputting data into all the required fields, the submission button was easy to find. I was also notified that my product was created successfully after I clicked submit.

Creating an Auction Product

- Test Objectives
 - For this test plan, the feature in question is the process of creating an auction on the site. An auction differs from a regular listing, in that it allows potential buyers to submit a price point and compete with other buyers until the time limit expires. This is beneficial to a buyer, due to potential cost savings over a regular listing. This feature is important to test, as it can be one of the main selling points of the site to users. We want to be able to test the frontend and backend features of the site to determine that the auction listing is being created, with all of the necessary information being added to the mysql database.
 - Understanding the functionality of creating this auction is very important to understanding how the company can innovate going forward. There are other features of the site that allow us to pick up items from customers to sell ourselves, to alleviate the stress and frustration of selling items online. Allowing this system to flourish under these testing conditions, then we can assess the impact that this will have on our growth as a company.
- Test Description
 - This system was set up using the Windows 10 operating system. I used several different browsers throughout this test, with minimal conflicts. I tested this feature using Chrome, Firefox, and Microsoft Edge. There were zero issues with these browsers and operating system combination.
 - A good starting point for this testing environment would be right after login, at profile settings. We want to be able to switch over to the seller-settings page from the profile page after logging in. Once we are there, we can determine what kind of listing we want to have. Is this an instant purchase listing, or an auction style listing?
- Usability Task Description
 - The intended users of the platform are anyone looking to lighten the load or do some sort of spring cleaning. The intended users for the testing environment are the same folks. I used my wife (who is not very technically savvy) to try and attempt creating an auction page. She was a perfect candidate as she is a professional at constantly finding errors or flaws in my designs. I believe she represents what most people look for in a testing candidate, someone to come in and try and break what you think works fine.
- URL of the System to be Tested
 - The URL of the system to be tested will be at <http://dropsell.gg/seller-settings> , the creating a listing in auction format is the feature to be measured. We are also testing for specific parts of the auction for usability, such as the duration, the length, and the ability to place bids and outbid other users.

- Usability test table

Test/Use Case	% Completed	Errors	Comments
Create product auction	100%	No errors in creating a product listing	Creates listing but without all features
Set duration	75%	Specified minute duration not available	Exact start date and end date are implemented
Set minimum bid	95%	Bidding feature not implemented	N/A
Auction is found under search	50%	Specific auction listings are not available yet	N/A

- Questionnaires

I felt that creating an auction was an easy experience

☐ Strongly Agree ☒ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

Creating an auction is not very different from creating the regular product listing

!

I had complete control over the auction duration

☒ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

It is clear to me how to view who is the top bidder

☐ Strongly Agree ☐ Agree ☒ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

Adding products to cart / Checkout process

- Test objectives
 - For this testing process, there are two functional requirements being tested here. First, we want to test the ability to add to the user's shopping cart. The second functional requirement involves completing the checkout process of said item. We want our test to be able to acknowledge that an item can be added to the cart, and remain in the cart for several different webpages. When the user is ready, the checkout process can begin. This test should acknowledge that the product's persistence remains through the cart, to the completion of the transaction. Once the transaction is completed, the item is then removed from the cart, and from the rest of the website as well.
 - Understanding the functionality of creating this auction is very important to understanding how the company can innovate going forward. Being able to have a fluid cart and checkout process is the backbone of our company's entire product. Allowing this system to flourish under these testing conditions, then we can assess the impact that this will have on our growth as a company, as well as determine the stability and usability for our testing group, or the average user.
- Test description
 - This system was set up using the Windows 10 operating system. I used several different browsers throughout this test, with minimal conflicts. I tested this feature using Chrome, Firefox, and Microsoft Edge. There were zero issues with these browsers and operating system combination.
 - I decided that the best starting point is the home page. Let the user add the item to the cart, move around different web pages to signify the item is still within the cart, and then begin the checkout process whenever they are ready. Once that process had started, I wanted to be able to test that the entire checkout functionality was tested. Entering payment information, reading over the invoice, and hitting submit order. All of these may seem minor to the user, but being able to test each one individually is crucial to the usability of our product.
 - The intended users of this product are the same as the intended users from the usability testing of both creating the listing, and creating an auction. These are people who want to purchase an item that they are selling, whether it is to finally grab that Xbox that is sold out elsewhere, or to bring to fruition the idea of, another man's trash is another man's treasure.
- URL of the system to be tested and what is to be measured
 - The URL of the system to be tested will just be the home page as a good starting point. (<http://dropsell.gg>). The system that is to be tested and measured, will be the checkout process. Adding an item to the cart will not change the URL, unless the site is redirected to a shopping cart screen. Once on this screen, the user can be given the option of completing checkout. If the customer proceeds with completing checkout then the process of entering payment info is to be tested. Persistence of payment information and items added to cart are to be tested by the user as well.

- Usability test table

Test/Use Case	% Completed	Errors	Comments
Persistent shopping cart	100%	Shopping cart remains full across pages	This done by adding product to the mysql database
Purchase product before opening shopping cart	75%	Application will error out if product is not purchased first before opening up the shopping cart from the hamburger menu	Workaround to this issue to click purchase button to add into the cart
Add payment information	90%	Payment information stored in checkout process, but not persistent in account management	N/A
Submit order	100%	N/A	N/A

- Questionnaires

It was easy to add an item to the shopping cart

☐

Strongly Agree

☒


Agree


☐


Neutral☐☐


Comments


Adding payment information was intuitive

Strongly Agree

Agree


Neutral


Disagree


Strongly Disagree


Comments


Purchasing the item I wanted went without a hitch

Strongly Agree

Agree

Neutral

Disagree

Strongly Disagree

Comments

It was unclear whether purchase and add to cart
were the same option

Price Matching

- Test objectives
 - The objectives of this test are to determine whether the ability to price match using web scraping tools is efficient and functional. Our main objective is not only usability, ease of use, but also functionality. We want to make sure that we are able to determine the prices of competitors so that our potential users can make informed decisions, which ultimately lead them to use our site over the competition.
 - Our price matching setup currently scrapes prices from www.amazon.com, but will be added to support more websites as well. We want our tests to show to the potential user that it is easy to use, and works as intended. This feature is not only important to the regular everyday user, but also for people looking to sell as well. Being able to see what a competitor is offering on the same product, and being able to slightly undercut them to achieve that sale, is what using our website should be all about. Having all of these features baked into the core product is very important.
- Test description
 - There will be several tests done for price matching. The first test conducted will be to determine whether or not figuring out how to price match was user friendly. Was it an easy task to figure out how to do? Were the users able to complete a sale without even realizing it was a feature? These are all considerations being taken to understand how this will be accomplished. There will be another test to determine whether there was more than one site. We also want to find out if the design is simple enough for the average user to figure out how to price match against more than just Amazon.
 - It is very important as well for the buyer to be able to determine how price matching works on the buyer's side. Do they have access to the same functionality outside of seller settings? These are forms that will be tested, as it is very important to our core mission that we want it to be as easy as possible for both buyers and sellers to make the most informed decision that they can on this subject. Being able to determine if an item is cheaper on our website, will help secure the sale for both the seller and the buyer, at a lower rate than shopping at the big box stores and websites.
- URL of the system to be tested and what is to be measured
 - The URL of the system to be tested will be at <http://dropsell.gq/seller-settings>, and as well as the product listing page itself. What will be measured is the ease of use and accessibility of the price matching feature itself. Does the link to Amazon bring up the relevant product, or a similar product in price/scale? We want to make sure that the feature makes sense from a front end standpoint as well as a back end standpoint. Is it possible for it to point to more than one competitor, and how well it does the job at scraping the data from those websites to determine the accuracy of the product being searched.

- Usability test table

Test/Use Case	% Completed	Errors	Comments
Price matches to same product	95%	N/A	Sometimes product isn't always exact
Price matches to similar product	100%	N/A	Product will always be matched to similar if not the same exact product
Price matching on more than one website	0%	Price matching only currently occurs on Amazon.com	N/A
Buyer is able to price match from product page	100%	N/A	N/A

- Questionnaires

It was easy to price match a product as a buyer

☐ Strongly Agree ☒ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

Could be clearer, but overall very efficient

I was able to successfully find a better price using the price matching feature

☒ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☐ Strongly Disagree

Comments

The feature worked exactly as intended

I was able to price check from several different online retailers

☐ Strongly Agree ☐ Agree ☐ Neutral ☐ Disagree ☒ Strongly Disagree

Comments

Currently I am only able to price check from Amazon

Seller/Buyer scheduling

- Test Objectives
 - The seller/buyer schedule calendar feature is to ability and organize meetup times between the buyers and sellers. This can help to improve organizing and managing plans, meetups, and work times. Sellers should be able to have this feature because it helps them to be able to track their present and future tasks. This can also improve business to view the history between sellers and buyers meetup ups and making sure to keep track if there are economic issues. Also, the schedule calendar meetup notes are able to record how the sellers set up their good meet up times performance.
 - This feature is important for business management because it can analyze buyers and sellers meetup times and records the performance for the business. Also, ability to store the meetup notes data onto the database whenever sellers and/or buyers put their meetup notes on the schedule calendar feature.

- Test Description
 - This system was set up using the Windows 10 operating system. I used several different browsers throughout this test, with minimal conflicts. I tested this feature using Chrome and Safari. There were zero issues with these browsers and operating system combination.
 - When you do not have an account for the dropsell application, you need to register an account, then login in. Then, it goes directly to your profile page. If you are a seller, you need to go to the seller settings and click "Set Up Schedule". After that, you are able to fill up your title, start meeting time, end meeting time, and location. Once you click submit, it goes to the calendar and records your meetup note and you are able to cancel the meetup note if you want. Also, making sure that the library is working and can handle the feature component properly and handling inputs for displaying in front of the calendar schedule.
 - The difficult part of testing this feature is when a seller decides to cancel the meeting, then the buyer does not want to let the seller cancel the meeting, and there might be a time conflict issue. The important testing is whenever a seller or buyer logs out from an application, it should record and not automatically delete their meeting notes.

- URL of the system to be tested and what is to be measured
 - The URL of the system to be tested will be at <http://dropsell.gq/set-workSchedule>, on the title, start time, end time, and location inputs are to test up if the meeting notes are actually displayed on the calendar.

- Questionnaires

Test/Use Case	% Completed	Errors	Comments
Add Meetup Note	100%	No errors in add meetup note	Creates the list of meetup notes
Display Meetup Note	80%	Handling start and end meeting up error	Making sure the start time and end time have the correct order
Delete Meetup Note	100%	No errors in delete meetup note	N/A
Edit Meetup Note	50%	Must edit the meetup notes	Possibly can change the meetup notes, but not editing the title, start time, end time, and location.

- Testing Table

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Library	Test the calendar react library	Install and get Calendar react library	Display calendar react on the set up schedule component	Pass
2	URL	Check if the calendar react is working when hosting	Display calendar react while hosting	Set up schedule link route	Pass
3	Database	Records meeting notes	mySQL data	Prints and records the meeting notes	Fail
4	Functioning Calendar schedule	Ability to add, edit, and delete meeting notes.	User meeting notes inputs	Change meeting notes	Pass
5	Socket	Records meeting notes between the buyers and sellers	Add meeting notes and tag a buyer/seller	Should be able to see the meeting notes between a buyer and a seller	Fail

3. QA Test Plan

- Test Objectives
 - Our objective with these tests is to verify that Dropsell's superior feature - price matching is handled properly or not. Due to the fact that this feature is our main focus as it is not implemented by our competitor sites, we need to ensure that this functionality is working reliably. We will be targeting specific tests such as verifying if the puppeteer library is scraping the similar product data from amazon. Additionally, we will also test our ability to store and retrieve product information on the database, which will be stored in a separate table.
- HW and SW Setup
 - For all features, the hardware needed is a computer or laptop with power, running on Windows, Ubuntu or MacOS operating system which is connected to a stable internet connection. Users will need a browser such as Chrome or Firefox installed on their computer. Last setup requirement is to navigate to a browser and go to Dropsell via <http://dropsell.qq/>
- Features to be Tested
 1. Library
 - 1.1. Test the high-level browser automation library used for Price Matching
 2. URL
 - 2.1. Check if amazon's url is given to create http header
 3. Parsing
 - 3.1. Check how the data extracted from web scraping is converted into structured format
 4. Database
 - 4.1. Check if the extracted data is getting stored in mysql table 'puppeteer_data'
 5. Internal Comparison
 - 5.1. Check if Price Matching is applied to compare price with other products of Dropsell

- QA Test Plan

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Library	Test the high-level browser automation library used for Price Matching	Product name	Prices of the same product that are for sale on Amazon	Pass
2	URL	Check the http header	Send http request to Amazon API gateway	Amazon URL	Pass
3	Parsing	Check if the scraped data is converted into structured format	Product title, image, price, seller	JSON format of Product title, image, price, seller	Pass
4	Database	Check if the extracted data is getting stored in database	JSON data extracted from web scrapping	new row in 'puppeteer_data' table	Pass
5	Internal Comparison	Check if Price Matching is applied to compare price with other products of Dropsell	Browse a Product	Price matching results with Dropsell products	Fail

4. Code Review

a) Coding style

- i) The coding style we chose is to have separation code for the front end which is in the client directory and back end for the server directory. For the front end, we decided to have pages and the separation of components. The pages are responsible for handling components when rendering an application. The components have different features of each page such as settings, landing page, about, etc. The module's component is to have different partial features for the pages in the client side such as navigation bar, search bar, menu, and etc. So, these partial features are the children for the pages and the partial features have their own functionalities in the entire application.
- ii) When it comes to handling user inputs, buttons, checkboxes, and etc. These features are where actions and reducers are responsible for keeping track of. Inside the redux directory, we have actions and reducer directories. Actions is where getting the user actions input data from the feature components. Then, using axios, is where to work with an API and be able to communicate between frontend and backend for requesting and responding for the user input data.

b) Example of our application code style

So let's start with our front end first. The client directory is where we handle most of the feature components. Let's say we use registration of how this page works for both front end and back end as an example.

```
JS Register.js 1 ●
application > client > src > pages > JS Register.js > ...
1  import React from 'react';
2  import { Redirect, NavLink } from 'react-router-dom';
3  import TOS from '../components/TOS';
4  import { connect, useDispatch, useSelector } from 'react-redux';
5  import {
6      setFirstname,
7      setLastname,
8      setEmail,
9      setUsername,
10     setPassword,
11     setConfirmPassword,
12     setTOS,
13     createUser,
14     setDriversLicense
15 } from '../redux/actions/registerActions';
16 import NavBar from '../components/Modules/NavBar';
17 import Footer from '../components/Modules/Footer';
18
19
20 const Register = (props) => {
21
22     // dispatch state data back to redux
23     const dispatch = useDispatch();
24
25     const registerFirstname = useSelector((state) => state.registerReducer.firstname);
26     const registerLastname = useSelector((state) => state.registerReducer.lastname);
27     const registerEmail = useSelector((state) => state.registerReducer.email);
28     const registerUsername = useSelector((state) => state.registerReducer.username);
29     const registerPassword = useSelector((state) => state.registerReducer.password);
30     const registerConfirmPassword = useSelector((state) => state.registerReducer.confirmPasswor
31     const registerDriversLicense = useSelector((state) => state.registerReducer.driversLicense)
32     const termsOfServices = useSelector((state) => state.registerReducer.termsOfServices);
33
34     const submitHandler = () => {
35         dispatch(createUser());
36     };
37
38 }
```

We made a functional base which is called “Register” and in line 6 through 14 is where the variables that we need to use for register redux action. Also, in line 25 through 32 is where we use for redux actions and reducer that can handle for user inputs, checkboxes, and buttons.

```

72     <div className="register-wrapper-first-last">
73       <div>
74         <p>First Name</p>
75         <input className="register-firstname"
76           type="text" placeholder="First name"
77           autoComplete="First Name"
78           value={registerFirstname}
79           onChange={(e) => dispatch(setFirstname(e.target.value))}
80           required/>
81       </div>
82       <div>
83         <p>Last Name</p>
84         <input className="register-lastname"
85           type="text" placeholder="Last name"
86           autoComplete="Last Name" value={registerLastname}
87           onChange={(e) => dispatch(setLastname(e.target.value))}
88           required/>
89       </div>

```

In the JSX, in the attribute of onChange, dispatch is coming from react hooks and it is important working with reducer then takes the action functions such as “setFirstname”, “setLastname”, and etc.

```

JS Register.js 1
application > client > src > pages > JS Register.js > [0] Register
93     </div>
94     <div className="register-wrapper-confPassword">
95       <p>Confirm Password</p>
96       <input className="register-confPassword" type="password" autoComplete="new-password" placeholder="Confirm Password" value={regi
97     </div>
98     <div className="dl-wrapper">
99       <p>Only enter if signing up as a Seller</p>
100      <input className="register-dl" type="text" autoComplete="DL#" placeholder="Driver's License ID" value={registerDriversLicense}
101    </div>
102    </label>
103
104    <p> <b>Terms of Services and Privacy Agreement</b></p>
105    <TOS/>
106
107    <div className="tos-checkbox">
108      <span className="tos-checkbox-text"> By checking the box, you agree to Dropsell's Terms of Services and Privacy Agreement. </span>
109    </div>
110
111    <button className="submit-signup" onClick={submitHandler}> Register </button>
112  </div>
113
114  <Footer/>
115 </div>
116 );
117 }
118 };
119
120 function mapStateToProps(state) {
121   return { registered: state.registerReducer.registered };
122 }
123
124 export default connect(mapStateToProps)(Register);

```

We use mapStateToProps for extracting data that takes the state of the register reducer.

```

JS registerActions.js 1 X
application > client > src > redux > actions > JS registerActions.js > [e] setEmail
1  import axios from 'axios';
2
3  export const setFirstname = (firstname) => ({
4    type: 'USER_SET_FIRSTNAME',
5    firstname
6  });
7
8  export const setLastname = (lastname) => ({
9    type: 'USER_SET_LASTNAME',
10   lastname
11 });
12
13 export const setEmail = (email) => ({
14   type: 'USER_SET_EMAIL',
15   email
16 });
17
18 export const setUsername = (username) => ({
19   type: 'USER_SET_USERNAME',
20   username
21 });
22
23 export const setPassword = (password) => ({
24   type: 'USER_SET_PASSWORD',
25   password
26 });
27
28 export const setConfirmPassword = (confirmPassword) => ({
29   type: 'USER_SET_CONFIRM_PASSWORD',
30   confirmPassword
31 });
32
33 export const setTOS = (TOS) => ({
34   type: 'USER_SET_TOS',
35   TOS
36 });
37
38 export const setDriversLicense = (driversLicense) => ({

```

Now inside the action directory, we have the list of set action functions for handling from register inputs, checkboxes, and buttons events.

```

43 export const createUser = () => {
44   return (dispatch, getState) => {
45     const userData = {
46       firstname: getState().registerReducer.firstname,
47       lastname: getState().registerReducer.lastname,
48       email: getState().registerReducer.email,
49       username: getState().registerReducer.username,
50       password: getState().registerReducer.password,
51       confirmPassword: getState().registerReducer.confirmPassword,
52       driversLicense: getState().registerReducer.driversLicense
53     };
54
55     console.log(userData);
56
57     axios.post('/api/register', userData)
58       .then((res) => {
59         console.log(res);
60         if(res.status === 201) {
61           dispatch(redirectUser(true));
62         }
63       })
64       .catch((err) => {
65         console.log(err);
66       });
67   };
68 };
69
70
71 export const redirectUser = (registered) => ({
72   type: "USER_IS_REGISTERED",
73   registered
74 });

```

When the user is done signing up and clicks the register button, it goes to the action createUser function and takes the firstName, lastName, and all register inputs data. Then, using axios to go to the api route and respond to the back-end server.

```
JS registerReducer.js 1 X
application > client > src > redux > reducers > JS registerReducer.js > ...
1  const INITIAL_REGISTER_STATE = {
2    username: '',
3    password: '',
4    confirmPassword: '',
5    registered: false,
6    termsOfServices: false,
7  };
8
9  const registerReducer = (state = INITIAL_REGISTER_STATE, action) => {
10
11    switch(action.type) {
12      case 'USER_SET_USERNAME':
13        return {
14          ...state,
15          username: action.username,
16        };
17
18      case 'USER_SET_PASSWORD':
19        return {
20          ...state,
21          password: action.password,
22        };
23
24      case 'USER_SET_CONFIRM_PASSWORD':
25        return {
26          ...state,
27          confirmPassword: action.confirmPassword,
28        };
29
30      case 'USER_IS_REGISTERED':
31        return {
32          ...state,
33          registered: action.registered,
34        };
35    }
36  }
```

The registration reducer takes from the registration input handler events by using mapStateToProps.

```

101 app.post('/api/register', (req, res, next) => {
102   const { username, password, confirmPassword } = req.body;
103
104   if (username && password && confirmPassword) {
105     console.log(username + " " + password + " " + confirmPassword);
106     next();
107   }
108   else {
109     res.status(400).send({
110       error: "The payload is wrong!"
111     });
112   }
113 },
114 (req, res) => {
115   store
116     .createUser(req.body.username, req.body.password)
117     .then((createdUser) => {
118       console.log(createdUser);
119       res.status(201).send(createdUser);
120     })
121     .catch(error => {
122       console.log(error);
123       res.status(500).send({ error: "Unable to insert the user" });
124     });
125 });

```

In the server directory in index.js, this function will take or is requesting the username and password of an existing user for our application for the login feature.

```

60 async function createUser(username, password) {
61
62   const encPassword = await bcrypt.hash(password, saltRounds);
63   console.log(encPassword);
64
65   const date = new Date();
66
67   const result = await pool.query(
68     "INSERT INTO users SET username = ?, password = ?, created_at = ?, is_active = ?, is_b
69     [username, encPassword, date, 1, 1, 0]
70   );
71   if (result[0].length < 1) {
72     throw new Error(
73       `Failed to create a new user ${username}`
74     );
75   }
76   return getUserById(result[0].insertId);
77 }

```

In the store.js, will take the mysql query that inserts the username and password to the database when creating a user from a registration.



Inside the mySQL workbench is where the users attribute data is located.

5. Self-Check on best practices for security

1. Major assets being protected
 - a. Registered user account information
 - i. Password
 - b. AWS EC2 instance
 - i. SSH URL
 - ii. SSH username
 - iii. SSH RSA Key
 - c. RDS MySQL database
 - i. Database username
 - ii. Database password
 - iii. Database name

When a user creates a new account, the account information they've inputted is sent securely to the `/api/register` route in our Node.js backend. The user's data is first processed in the first body of the `/api/register` route. For validating user input in the register route, we are using the validator npm library. To validate user registration data, we test to determine if the email is the correct format, if the first name/last name/username/password/confirm password are filled in or not (if these inputs are empty then they are not valid), and if the password and confirm password inputs are equal to one another. If all these tests pass for user registration, then this data is passed as parameters into the .createUser() function written in store.js.

```

182 app.post('/api/register', (req, res, next) => {
183   const { firstname, lastname, email, username, password, confirmPassword, driverLicense } = req.body;
184
185   if (validator.isEmail(email) &&
186       !(validator.isEmpty(firstname)) &&
187       !(validator.isEmpty(lastname)) &&
188       !(validator.isEmpty(username)) &&
189       !(validator.isEmpty(password)) &&
190       !(validator.isEmpty(confirmPassword)) &&
191       validator.equals(password, confirmPassword)) {
192
193     console.log("User email of: " + email + " is valid.\n");
194     console.log("User first name of: " + firstname + " and last name of: " + lastname + " are valid.\n");
195     console.log("User password of: " + password + " and confirm password of: " + confirmPassword + " are valid and are matching.\n");
196
197     next();
198   }
199   else {
200
201     if(!(validator.isEmail(email))) {
202
203       res.status(400).send({
204         error: "Enter a valid email"
205       });
206     }
207
208     if(validator.isEmpty(firstname) ||
209        validator.isEmpty(lastname) ||
210        validator.isEmpty(username) ||
211        validator.isEmpty(password) ||
212        validator.isEmpty(confirmPassword)) {
213
214       res.status(400).send({
215         error: "One or many of the inputs you wrote are empty, please try again"
216       });
217     }
218
219     if(!(validator.equals(password, confirmPassword))) {
220       res.status(400).send({
221         error: "Your password and confirm password need to match"
222       });
223     }
224   }
225 },
226 (req, res) => {
227   store
228     .createUser(req.body.firstname, req.body.lastname, req.body.email, req.body.username, req.body.password)
229     .then((createdUser) => {
230       console.log(createdUser);
231
232       store.createUserCart(createdUser.user_id)
233         .then((userCart) => {
234           console.log(userCart);
235         })
236         .catch(error => {
237           console.log(error);
238           res.status(500).send({ error: "Unabe to create user shopping cart" });
239         });
240
241       res.status(201).send(createdUser);
242     })
243     .catch(error => {
244       console.log(error);
245       res.status(500).send({ error: "Unable to insert the user" });
246     });
247   });

```

Then, within `.createUser()` in `store.js`, we encrypt the password which was passed in as a parameter, write our SQL statement involving `'INSERT'` syntax, and then apply the data we'd like to add into the database with the parameters we've provided `.createUser()`. The encryption process is done on line 88 using the `bcrypt` library. All we have to do is specify the number of salt rounds for our encryption. `'saltRounds'` is just a variable which is set equal to 10. Then, we pass our password parameter and the number of salt rounds into `bcrypt's .hash()` function, which will create our encrypted password. At this point, we have validated the user registration data which is why we move towards inserting the data into our users table. In regards to error handling, if an error occurs with the insertion of a new user, we are notified of this if the length of the `result[0]` object is less than one. If the `result[0]` object is less than one, then this means there was not a new row created in the users table.

```
86 | async function createUser(firstname, lastname, email, username, password) {
87 |
88 |   const encPassword = await bcrypt.hash(password, saltRounds);
89 |
90 |   const date = new Date();
91 |
92 |   const result = await pool.query(
93 |     "INSERT INTO users SET first_name = ?, last_name = ?, email = ?, username = ?, password = ?, created_at = ?, is_active = ?, is_buyer = ?, is_seller = ?",
94 |     [firstname, lastname, email, username, encPassword, date, 1, 1, 0]
95 |   );
96 |   if (result[0].length < 1) {
97 |     throw new Error(
98 |       'Failed to create a new user ${username}'
99 |     );
100 |   }
101 |   return getUserById(result[0].insertId);
102 | }
```

6. Self-Check: Adherence to original non-functional specs

Non-functional Requirements Specifications	Status	Comment
1. There should be WebSocket functionality for displaying the most recently created posts, the hottest daily pics, and other displays of item data. a. WebSocket functionality should be used for displaying user's notifications, messages, and for auctions.	Issues	Low priority in relation to implementing our superior feature price matching
2. SSL certificate, which enables websites to move from HTTP to HTTPS, for better website security.	On track	
3. Warning/caution list page for what sellers/buyers should look out for. A standard list providing details and warnings for potential scammers or bots.	On track	
4. Submit a ticket. After flagging an item since it seems sketchy, or an item might be fake, the user could continue the process by filing a ticket and sending a message to us to follow through further with reporting suspicious behavior.	On track	
5. Limited login attempts for security purposes. For example, if the user enters the wrong password 3 times, they will be locked out of trying to log in to their account for 30 minutes.	Issues	Low priority over implementing input validation for user login
6. Captcha to fight against bots. Captcha provides an extra level of security for users.	On track	
7. Email verification for both seller and buyer. Verification ensures that there is a valid user behind each seller and buyer account.	Issues	Low priority over implementing functional requirements
8. Two factor authentication for added security measures. The user will be required to authenticate their account on two different devices to help prevent a breach in security.	Issues	Low priority over implementing functional requirements
9. Use of sessions on the backend to securely keep track of the user's shopping cart information once they redirect from	Done	

the main shopping site.		
<p>10. Users should be able to view and update their personal information settings securely.</p> <p>a. Users can update their account information, change their full name, update their email, change their password. Update their payment information or see their complete payment history in a secure fashion.</p>	Done	
<p>11. Password recovery for all users when they need it. The recovery process will involve having an email sent to the user to the email address associated with their account, where they would have the opportunity to change their password.</p>	Issues	Low priority over implementing functional requirements
<p>12. Username recovery for all users when they need it. Similar to the password recovery process, where the user will get an email to recover their username.</p>	Issues	Low priority over implementing functional requirements
<p>13. Load balancers or distributed microservices on the backend so that we don't have just only servers doing all of the backend work? We don't want the backend to go down and have the whole server unavailable to all users.</p>	Issues	Low priority over implementing functional requirements
<p>14. Keeping track of unaccept/accepted usernames and write reviews.</p> <p>a. They must be appropriate and cannot contain duplicate usernames.</p>	Issues	Low priority over implementing functional requirements
<p>15. If the user does not do anything while logging in for like 15 minutes or more, then it automatically logs out the user.</p>	Done	Done but user session logs out after 30 minutes
<p>16. Correct login information entered. Username and password matching correctly.</p>	Done	
<p>17. Valid post information when a seller creates a new post.</p>	Done	
<p>18. The user's shopping cart should be updated with the total amount of the user's cart and their selected items FROM THE SERVER. User shopping cart data cannot be handled by the client, the client should only be used to display the data sent from the server.</p>	Done	

19. Username should be at least 8 characters in length. The username shall have a minimum length for security reasons.	On track	
20. User password should contain uppercase/lowercase and numbers with a minimum length of 8 characters. This password requirement ensures an extra layer of security.	Issues	Low priority over implementing functional requirements
21. When a user inputs their email into the registration form, a valid email should have @ along with a valid ending. This requirement checks if the email is in the right format.	Done	
22. New post titles should not exceed 80 characters in length.	On track	
23. New post description should not exceed 500 characters in length.	On track	
24. A minimum of 10 salt should be used when hashing a user's password.	Done	
25. Perform password hashing when a user creates a new account. This will guarantee us that passwords saved in the database are secure.	Done	
26. Bcrypt library should be used for hashing a user's newly created password.	Done	
27. A user's session should be serialized after logging in, and deserialized when a user leaves the site.	Done	
28. Stripe.js or paypal payment SDK should be used for creating and validating payments from users.	Done	
29. Seller analytics page (demographics of customers, what is sold more frequently, etc.) Analytics will provide the seller with information to help them better their product listings.	Issues	UI implemented but functionality is low priority
30. Passport library should be used to authenticate an existing user's requests.	Issues	Low priority in relation to functional requirements
31. Encapsulate any SQL statements so that SQL injections are not possible.	Done	
32. The table for seller posts should have	Done	

the following constraints for its ID column: integer type, auto increment, and public key.		
33. Each post should have a foreign key tied back to the user's id who created the post.	Done	
34. Each post should have columns for the posts' title, description, price, photo, the time when the post was created, if the post is still active on the app or whether it has expired.	Done	
35. Post comments should be represented in a comments table. Each comment's id should be of integer type, auto incremented, and the public key.	On track	The table for product comments is created but product comments are not finished
36. Each comment should also have a foreign key tied back to the creator of the comment, and a foreign key tied back to the post the comment was for. Each comment should also have the time that it was created.	Done	
37. Session data for each user should be stored in the database. If their session is still valid, allow them to log in directly into the app.	On track	
38. Redux should be used to manage the state for the react client. Redux will be essential for keeping track of user's shopping cart information.	Done	
39. Use of redux mapStateToProps() for updating user's shopping cart information, losing user account details. a. The mapStateToProps() function will be used to load state data, such as a user's shopping cart items as properties passed into React components so that the data can be displayed to the user.	Done	
40. Multer library should be used for uploading images to the server/database.	Done	
41. Browser support for IE, Chrome, Firefox, Brave, etc. Cross-browser support for all users to be able to access.	Done	
42. By default, users who are not logged in should still be able to access the	Done	

marketplace. It will show up on the top right corner that they are not logged in/login is shown instead of logout.		
43. When fetching data from api routes, there should be a maximum response time of 1s. Anything more than this will impede upon user experience.	Done	
44. When a user sends an axios request, the server should be able to parse through urlencoded or Json data.	Done	
45. React components should be kept simple. Design a React component so that it accomplishes one job for the user. If a component is more complex, then use multiple react components with parent/child relationships.	Done	
46. An object titled INITIAL_STATE should be used in each Redux reducer to hold each reducer's state data.	Done	
47. Redux actions should start with the name of the reducer in capitals, an underscore, plus whatever the action is doing, an underscore, plus whatever the action is changing. Example: USER_UPDATE_POST, USER_SEND_MESSAGE, etc.	Done	
48. If a team member has to create a new React route, then do so in App.js. This is where all of the Routing takes place.	Done	
49. Variable names should not be redundant, but instead describe what role the variable has. Thinking of good naming conventions is important so everyone else on the team knows why and how the variable was used a. By default use camel case for variable names.	Done	
50. Commenting is essential. Before jumping into code, every team member should write some form of pseudocode describing what it is they're going to code before actually doing so. a. Outside of pseudocode, commenting should be used everywhere to describe how features or architecture works.	Done	

51. At least 3 images of products required on listing. Having more than 1 image of the product provides customers with more information of how the product will look.	Done	
52. Minimum/maximum size of images on product page. An appropriate size of the images (perhaps 600x600 or 2x2) will help users with product visibility.	Done	
53. At least a 10 word description of the product being sold. The seller will provide adequate details on the product listing.	Done	
54. Buyer protections provide buyers with peace of mind when using the app.(Paypal has 180 days for users to be refunded if there is an issue with the purchased item).	Issue	Low priority compared to implementing functional requirements
55. At least a 5 word title of the product being sold. The seller will provide an adequate title on the product listing.	Done	
56. Once an item is sold, the listing is automatically updated to reflect as such. This will notify users that the listing is no longer available/has been sold, or that the number of products available is now different.	Issue	Low priority compared to implementing functional requirements
57. Products must be concrete, not abstract (no services, only physical items.) Restrictions for what can be bought and sold on the app should be made clear.	Done	
58. Maximum number of listings per seller- 100? 1000? To keep balance of how we are trying to cater to the smaller businesses.	Done	
59. Offer commitment on each listing that has the offer option. Once the buyer makes an offer or bid and wins, a transaction automatically occurs and the buyer gets their account charged (automatic withdrawal.)	Issue	Low priority compared to implementing functional requirements
60. Product page must include how much of each item is available. The number of products available lets the buyer know how much is in stock.	Done	
61. Product page has an expiration of 60 days. The expiration ensures that the	Issue	Low priority compared to implementing functional

product listings are current and that sellers are active.		requirements
62. Sellers should not be able to reject buyer's meetup time within 2 days, unless it is an emergency case.	Issue	Low priority compared to implementing functional requirements
63. Product page must include a place of origin. This requirement will let the buyer know where the product is coming from, and perhaps what to expect when it comes to shipping time.	On track	
64. Product page must include if the item is new or used. This detail is essential to let the buyer know the condition of the product being sold.	On track	
65. 24 hour window to modify order; after the 24 hours, buyers will be committed to their order.	Issue	Low priority compared to implementing functional requirements
66. Must show the seller's name of a selling item.	Done	
67. Must show the seller's rating selling items.	Issues	Low priority compared to implementing functional requirements
68. Stay connected- follow us on social media. This will provide the app with cross platform exposure and potential business.	Issues	Low priority compared to implementing functional requirements
69. App news/announcements. Users will have the option to subscribe to the app to receive newsletters through email.	Issues	Low priority compared to implementing functional requirements
70. Paid ads throughout the site. Advertisements will be another source of revenue for the app. Perhaps the end of the page will include a link titled "advertise with us" to attract potential business.	Issues	Low priority compared to implementing functional requirements
71. FAQ page that highlights all the questions and answers that users generally have when using the app.	On track	
72. Coupon/promo code expiration. Every code will have an expiration date to keep discounts as special.	Issues	Low priority over implementing functional requirements
73. Recently sold items on the home page. This will show users what has recently sold, and for how much it has sold for.	Issues	Low priority compared to implementing functional requirements

74. Team members should use pm2 as their process manager when running the project locally. 'npm start' is NOT necessary for running the project. Only 'pm2 start process.config.js' should be used.	Done	
75. Freenom website should be used to configure a free domain name for our app.	Done	
76. If the AWS EC2 instance is stopped and then reset, the public ipv4 address will have changed. This means the app's public IP address must be changed in /credentials, and it must be updated in freenom's domain name configuration.	Done	
77. Team members should use the 'systemctl status nginx' command to test the status of the nginx web server.	Done	
78. If necessary, team members should only update the nginx server blocks contained in /etc/nginx/sites-available/default. No where else.	Done	
79. In order to verify any errors related to nginx, always use the command 'sudo nginx -t'	Done	
80. If team members need to reload the nginx server, then run 'sudo systemctl reload nginx'.	Done	
81. If team members pull changes to a remote branch, make sure to 'npm install' so you have any new packages/libraries installed from other people's commits.	Done	
82. Team members should always create new branches from the development branch.	Done	
83. After you are done with your work, create a merge request from your branch back into the development branch so we can compare the differences between the two.	Done	
84. If you ever have a question about any new code pushed to github, then team members must ask about what they're unsure on in the discord #help channel.	Done	

85. When team members encounter a bug in the code, team members should write down the steps in order to duplicate the bug, then create a pull request labeled as 'bug' so that the rest of the time can replicate the bug.	Done	
86. If team members need to add additional documentation for a feature they've created, they should add the documentation label onto their pull request.	Done	
87. If a team member is creating a new feature, then they should mark their pull request with the enhancement label.	Done	
88. If a team member has a question about a portion of code they are writing, then they should create a pull request from their branch back into development with the question label. This way, the rest of the team can review what the question is to resolve it.	Done	
89. Code should be merged from the development branch into master branch only when all merge requests for the current build have been agreed upon and added to the current git history.	Done	
90. Whenever a milestone is close to submission, a thorough check of the /credentials folder should be done to ensure everyone has the most recent updates.	Done	
91. SSH should always be used when logging into the remote EC2 instance.	Done	
92. Whenever logging into the EC2 instance, make sure to check for updates: use the following commands: 'sudo apt update', 'sudo apt upgrade' or 'sudo apt full-upgrade', and 'sudo apt autoremove'. If you are prompted to reboot, then reboot the instance.	Done	
93. Create a CI/CD pipeline for deploying code to the remote EC2 instance.	Issues	Low priority over implementing functional requirements
94. The Node api server should have access to react build files so it works correctly on the remote EC2 instance.	Done	

95. In order to login to EC2 instance correctly, team members should ensure that the csc648.cer file in their local github repository's / credential folder has file permission 600.	Done	
96. CSS styling to create a box-shadow around an item when the user hovers over it. This will allow the user to view the item clearly.	Done	
97. High contrast between text and background for ease of use. This will make the overall appearance of the app easy to view.	Done	
98. The sidenav should be implemented with the React transition group library.	Done	Finished but implemented without the react library since this was not needed
99. All Contact, About, Careers, company information, etc. should be stored in the footer of the app.	Done	
100. All Login/Logout, Shopping Cart, Notification, Auction pages should be stored in the sidenav menu.	Done	
101. @media queries should be used for making the app usable on mobile devices.	On track	
102. In order to satisfy requirements for screen sizes <= iPhone 5, the minimum width for @media should be 300px.	Done	
103. In order to satisfy requirements for screen sizes <= iPhone 6, the minimum width for @media should be 360px.	Done	
104. In order to satisfy requirements for screen sizes <= iPad, the minimum width for @media should be 720px.	Done	
105. In order to satisfy requirements for screen sizes <= desktop, the minimum width for @media should be 1280px.	Done	
106. By default, buyers will not have their shipment information added. If they choose delivery, then they must be prompted for their address information.	Done	
107. Users should be able to choose what page route that he or she needs to go to such as Home, Notification, About,	Done	

Contact, Login, and Logout.		
108. A hamburger menu should be used to represent the navigation bar, this menu should have an animation which opens and closes it.	Done	
109. Users should be able to interact with a “See More” at the end of the top 50 posts, which when clicked will load another 50 posts to view on the Home page.	Issues	Low priority compared to implementing functional requirements
110. When users click the hamburger menu, there should be a navigation bar component which is triggered in addition to the side nav which should pop out from the side.	Done	
111. Navigation bar should have an animation triggered when it is opened or closed. a. This task should be implemented with some sort of CSS transform or the use of a React component to represent the navigation bar.	Done	
112. When a user clicks the Notifications tab in the navigation bar, a dropdown menu should appear. a. A list of the user’s notifications ordered by most recent to least recent is a requirement.	Issues	Low priority compared to implementing functional requirements
113. When a user clicks on a notification from another user, they should be redirected to a page displaying their conversation history.	Issues	Low priority compared to implementing functional requirements
114. If a user is not logged in, views an item, and then clicks the button to message the seller, the user should be prompted to either login with their username/password or to create an account with a button.	Issues	Low priority compared to implementing functional requirements
115. Lock password input in login from being updated or clicked on after the user clicks again.	Issues	Low priority compared to implementing functional requirements
116. If a user attempts to login and they use the wrong credentials, they should be redirected back to /login route.	Done	
117. Once a user logs in with the correct	Done	

username/password, redirect them to the home page at route '/' to start interacting with our app's features.		
118. Users can switch to dark or white background mode for the app.	Issues	Low priority compared to implementing functional requirements
119. Category menu (beauty/cosmetics, home, appliances, tech gadgets/computers, athletic tools, clothing). This will allow searching items faster, and to keep all the products organized.	On track	
120. Photo slider which has left/right arrows allowing the buyer to navigate through the pictures of an item posted by a seller. This will allow the buyer to see details on the design of the product.	Issues	We had time for implementing one image, multiple images became a priority 2 which is why the photo slider has not been finished.
121. Have an option to slide through the posts or go to the next page.	Issues	Low priority compared to implementing functional requirements
122. Display seller's contact information.	Done	
123. Shopping cart with number of items icon. The buyer will be able to view the total items in their cart and the total cost of the items.	On track	
124. Option to drag item(s) to the shopping cart button. This will provide shopping ease for buyers.	Issues	Low priority compared to implementing functional requirements

7. List of Contributions

<u>Student Name</u>	<u>Contributions</u>
Mitchel Baker	Mitchel was responsible for writing the usability test for the creation of new marketplace products. He was also the main contributor for the self-check on best practices for security section, while also working with Rowena to finalize the adherence to nonfunctional requirements. In addition to the Milestone 4 document, he also continued implementation of our team's functional requirements for the application.
Charmaine Eusebio	Charmaine was one of the main contributors for the QA test plan section. She collaborated extensively with Krina to perform valuable QA testing for our product's main feature.
Kenneth N Chuson	Kenneth was responsible for writing up the code review section of the Milestone 4 document. After he finished writing the code review, he had Mitchel review the work he had submitted for constructive feedback. He also wrote the usability test plan for buyer/seller scheduling.
Krina Bharatbhai Patel	Krina was one of the main contributors for the QA test plan section. She collaborated with Charmaine to write up a test plan which played an important role in breaking apart the main section of our product's main feature. Her work helped our backend team in coming up with an efficient plan for implementing our product.
Michael Schroeder	Michael played a key role in the following usability test plans: creation of an auction product, adding products to cart, and for price matching. He created the usability test and Lickert Scale questionnaire templates for our team which was

	extremely helpful for everyone else writing usability test plans.
Rowena Elaine Echevarria	Rowena played a key role by creating the table for the self-check on adhering to our original nonfunctional specifications. She communicated effectively on the requirements she was not sure about, and collaborated with Mitchel by working together on finalizing which requirements were done, on track, or had issues.
Jamie Dominic Walker	Jamie was the lead contributor to our product summary section in the Milestone 4 document. He applied precision and care by taking note of our main priority one requirements, while rewording each function in a way that was coherent and readable. He communicated to the rest of the team by requesting feedback when finished, which demonstrates his consideration for the rest of his team members.