



# SW ENGINEERING CSC648/848

## FALL 2019

### Gator Barter

Section 01

Team # 04

Aditya Bodi, Team Lead | abodi@mail.sfsu.edu

Alan Ng, Frontend Lead

Philip Yu, Frontend Engineer

Deming Yan, Full Stack Engineer

Akshay Kasar, Git Master- Backend Engineer

Alex Kohanim, Backend Lead, Full Stack Engineer

Tejasvi Belsare, Document Master- Backend Engineer

“Milestone 4”

Date : 12/11/2019

Version	Date Submitted	Owner	Description
1.0	December 4 <sup>th</sup> , 2019	Tejasvi	Initial draft with all sections and content
1.1	December 10 <sup>th</sup> , 2019	Alan, Deming, Akshay, Alex	Updated Usability, QA tests, Code Review
1.2	December 11 <sup>th</sup> , 2019	Aditya	Document review and formatting
1.3	December 15 <sup>th</sup> , 2019	Tejasvi	Changes as per Professor's feedback and freeze

## Table of Contents

1. Product summary	2
2. Usability test plan	3
3. QA test plan	6
4. Code Review	8
5. Self-check on best practices for security	10
6. Self-check: Adherence to original Non-functional specs	11

## 1. Product summary

Name of our product – Gator Barter

Below is the list of all major committed functions for Gator Barter:

1. Home Page
2. Search results (with search text validation)
3. Filter search by category
4. Sort the search results
5. Item details on separate page including information like images, description, price and seller
6. User Login
7. User Registration (including lazy registration)
8. Item posting (with all fields validation)
9. Messaging from a user to another user
10. Item trading (unique feature)
11. Seller dashboard (including seller items and messages)
12. Admin dashboard (to approve or reject the items posted for selling)

Gator Barter is a buying and selling website specially designed for San Francisco State University students. Gator Barter will be marketed as a website created by SFSU students for SFSU students. This website would serve as a platform for students to sell their items, class materials, etc. as well as buy essential items at a cheaper rate compared to the market price that too within campus. Students can also give away items free of cost on the site or trade their items with other items. It will provide an affordable and safe marketplace for students to exchange their belongings.

Moreover, this website builds direct communication between buyer and seller. This unique approach will eliminate long wait times and no shipping costs would be involved. The basic attraction of our website is its simplicity and user-friendliness. Also, the good design of Gator Barter can be eventually used as a template for other universities looking to design their own buy and sell website.

URL to our website: <https://gatorbarter.store/>

## 2. Usability test plan

### Test Objective:

The goal of usability testing is to identify any usability problems, collect quantitative data on participants' performance as well as determine user satisfaction for the website. If a website is difficult to navigate or doesn't clearly articulate a purpose, users will leave it. Ideally, usability testing is done before a site launches, and then many times after. Currently, our website is in its development and testing phase. Hence, usability testing will help us to find out the flaws or weaknesses in the usability flow at an early stage. User comments or reviews will help us to judge and improve our website in terms of usability.

We will be conducting a usability test for searching item(s) on our website. This will allow the user to navigate through a series of usage flows touching on the main components of the website. Also, test users will be asked to fill a survey consisting of a Lickert scale. Success factors would then be determined based on feedback from the test users.

### Test Background and Setup:

#### **Purpose:**

The purpose of this test is to identify any usability problems within the search functionality of our website.

#### **System setup:**

A laptop or a mobile with an internet connection is required and access to web browsers like chrome (v77), firefox (v69). We will be testing our website on the mentioned two versions and the latest versions of these browsers on laptops and mobile in order to check the consistency while using our website on different browsers.

#### **Starting point:**

The starting point will be the homepage of our website located on URL: <https://gatorbarter.store/>

#### **Intended Users:**

The intended users would be SFSU students.

#### **URL for the system/function for Testing:**

The URL for the search function is same as the home page/starting point of our website: <https://gatorbarter.store/>

## Task Description:

There are different tasks that the user needs to go through in order to successfully searching an item and then view more details of that item. This consists of searching and selecting the item they are interested in buying.

Task	Machine State	Successful completion Criteria	Benchmark
Open Gator Barter website on browser	Loaded URL	Homepage of the website displayed	
Choose an appropriate category if required	Appropriate category is highlighted in the 'categories drop-down'	Category Selected	
Searching an item	Loaded the results page	Filled out the search field and searched for an item for purchase	

## Measure effectiveness:

1. Check that the task is able to be performed and completed
2. Measure the number of people who completed the tasks
3. One suggested way to implement:

Test/use case	% completed	errors	comments
Search	100	0	

## Measure efficiency:

1. Test Efficiency in time: average time to complete the task, the average time of those who completed the task/av. time of all users
2. Test Efficiency in the effort: number of clicks
3. Test Efficiency in content/design: number of screens, Number of clicks, and number of pages of instructions

## Completion Criteria:

User has successfully completed the following tasks: -

1. Launched the website URL on the browser.
2. Found the Login button and Logged in successfully.
3. Found the search button and searched for an item successfully.

## User Satisfaction Questionnaire:

**Website user survey. Select one option for below questions –**

<b>Task</b>	<b>Strongly Agree</b>	<b>Agree</b>	<b>Neutral</b>	<b>Disagree</b>	<b>Strongly Disagree</b>
The GUI is simple and user friendly.	✓				
It was easy to search an item for sale on Gator Barter.			✓		
It was easy to locate the desired item from searching		✓			
Item details were easy to understand when displayed in more details.			✓		

### **General Comments/Feedback:**

The website is easy to understand and easy to use

### 3. QA test plan

#### Test Objective:

Quality assurance (QA) is a systematic process of verifying whether a software meets required specifications and customer expectations. The objective of below QA tests would be to test the functionality of searching for an item and viewing it on Gator Barter website, find bugs and resolve them prior to product release.

#### HW and SW setup:

1. A laptop with an internet connection is required and access to any of the web browsers among Google Chrome, Mozilla Firefox.
2. Browse to the website using the following URL. This will direct to the homepage of our website.
3. URL: <https://gatorbarter.store/>
4. Test the following cases for searching items.

**Feature to be tested:** Search an item for different categories.

#### Test Plans:

Test 1:

Input: Chose “Furniture” in the Category drop down menu then, enter “couch” in the search text field.

Output: Check the results consist of exactly 2 couch results

Test 2:

Input: Keep “All” in the Category drop down menu then, enter “mouse” in the search text field.

Output: Check the results consist of exactly one mouse result

Test 3:

Input: Chose any category from Category drop down menu and search for an item that does not exist in the database

Output: Feedback saying no results, but suggest other products from the website

Firefox				
Number	Description	Test input	Expected output	Pass/Fail
1	test % like in search for name field in specific category	Chose “Furniture” in the Category drop down menu then, enter “couch” in the search text field.	get 2 results, all have “couch” in name field	Pass

2	test % like in search for name field	Keep “All” in the Category drop down menu then, enter “mouse” in the search text field.	get 1 result with “mouse” in name field	Pass
3	test search for item not in database	Chose any category from Category drop down menu and enter “dog” in the search text field.	0 results from query, but get suggestions.	Pass

### Google Chrome

Number	Description	Test input	Expected output	Pass/Fail
1	test % like in search for name field in specific category	Chose “Furniture” in the Category drop down menu then, enter “couch” in the search text field.	get 2 results, all have “couch” in name field	Pass
2	test % like in search for name field	Keep “All” in the Category drop down menu then, enter “mouse” in the search text field.	get 1 result with “mouse” in name field	Pass
3	test search for item not in database	Chose any category from Category drop down menu and enter “dog” in the search text field.	0 results from the query but, get suggestions.	Pass



## 4. Code Review

### Coding Style:

- Use of PEP8 python coding style has been targeted.
- MVC architecture has been followed in the project
- Implementation of flask-blueprints framework had also been used to make the code modular and reusable.

### Code Review Example:

Code Review screenshot of an email:

@AlexKohanim requested changes on this pull request.

Please fix these issues before we can merge with full-stack

---

In [application/helperFunctions.py](#):

```
> @@ -0,0 +1,133 @@
+##This module contains or the utility/helper functions which can be reused later.

+from dbCursor import getCursor
+from queries import query
+import gatorProduct as product
+
+from flask import session
+import time
+import os
+from werkzeug.utils import secure_filename # for input picture loading
+
+
+db = getCursor()[0]
+
+ALLOWED_EXTENSIONS = set(['pdf', 'png', 'jpg', 'jpeg'])
```

Please remove pdf as we only accept image files.

---

In [application/dbCursor.py](#):

```
> @@ -0,0 +1,5 @@

+##This module contains functions for initializing connection to the MySQL database
+import pymysql
+db = pymysql.connect('0.0.0.0', 'root', None, 'gatorbarter')
```

Instead of literal values for DB config, read from another class or from a read-only file


---


You are receiving this because you authored the thread.

Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).

---

Following are the same steps on the GitHub review request page:



**AlexKohanim** requested changes 1 hour ago


[View changes](#)

AlexKohanim left a comment


Please fix these issues before we can merge with full-stack

application/helperFunctions.py Outdated


```
10 +
11 + db = getCursor()[0]
12 +
13 + ALLOWED_EXTENSIONS = set(['pdf', 'png', 'jpg', 'jpeg'])
```

**AlexKohanim** 1 hour ago

Please remove pdf as we only accept image files.

**akshaypkasar** 2 minutes ago Author

Made the change and pushed the fix!




Reply...


Resolve conversation

application/dbCursor.py Outdated


```
... .. @@ -0,0 +1,5 @@
1 + import pymysql
2 + db = pymysql.connect('0.0.0.0', 'root', None, 'gatorbarter')
```

**AlexKohanim** 1 hour ago

Instead of literal values for DB config, read from another class or from a read-only file

**akshaypkasar** 1 minute ago Author

Made the change and pushed the fix!



Reply...

## 5. Self-check on best practices for security

### List of Major Assets:

1. User Database
2. System Data
3. Images
4. AWS Server
5. Sales data

### Protection Description:

1. User Database - We are sanitizing the user's input for the items to be fetched from the database. This prevents the SQL injection.
2. System Data- We are using 'https' protocol to have secure requests and protect our system data.
3. Images - We are accepting only valid image types and we have a system in place to never execute the images
4. AWS System - We are using cronjobs to monitor various system errors and executing appropriate services.
5. The password of a user account is hashed/encrypted before updating the user information in the database.
6. Any input data is validated to prevent any SQL injection attacks. We have used flask-bleach in the backend to validate any input data.
7. Any input data validation to prevent cross browser scripting for the search field.

## 6. Self-check: Adherence to original Non-functional specs

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team, but all tools and servers have to be approved by class CTO). - ON TRACK
2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers - DONE
3. Selected application functions must render well on mobile devices - ON TRACK
4. Data shall be stored in the team's chosen database technology on the team's deployment server. - DONE
5. No more than 50 concurrent users shall be accessing the application at any time - DONE
6. Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users. - DONE
7. The language used shall be English. - DONE
8. Application shall be very easy to use and intuitive. - DONE
9. Google analytics shall be added - DONE
10. No e-mail clients shall be allowed - DONE
11. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. - DONE
12. Site security: basic best practices shall be applied (as covered in the class) - DONE
13. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development - DONE
14. The website shall prominently display the following exact text on all pages *"SFSU Software Engineering Project CSC 648-848, Fall 2019. For Demonstration Only"* at the top of the WWW page. (Important so as to not confuse this with a real application). - DONE