Milestone 4

SW Engineering CSC648/848 Fall 2022

GaterTrader

2022-12-09

Team 7

Name	Role	Email
Gineton Alencar	Team Lead	galencar@mail.sfsu.edu
Eddie Fu	Front-end Lead	efu1@sfsu.edu
Duccio Rocca	Back-end Lead	drocca@sfsu.edu
Yoshimasa Iwano	GitHub Master, Engineer	yiwano@sfsu.edu
Dominique Henry	Document Master, Engineer	dhenry3@mail.sfsu.edu
Rai'd S. Muhammad	Software Engineer	rmuhammad@mail.sfsu.edu
Kobe D. Shelby	Software Engineer	kshelby@mail.sfsu.edu

Table of Contents

1. Product Summary

1.1 Name: GaterTrader

1.2 Product Description: GaterTrader is a web platform exclusive for San Francisco State University (SFSU) students, staff, and faculty to sell and share content that may be useful to all. It was built by SFSU students for the SFSU community. Students and faculty will immediately see a benefit from the school-friendly categorization of media by course and field of study. Professors will be able to publish their syllabi, notes, and handouts in one place. Students, for example, will have the opportunity to be rewarded for their creativity by facilitating buying and selling of media.

1.3 URL: http://52.9.59.15:1234/

1.4 Priority 1 Functions

1. **unregistered-users**
    a. shall be able to browse posts of items for sale.
    b. shall be able to register if they meet registration requirements.
    a. shall be able to login if they have previously created a registered-user account.
    b. shall be required to register or login in order to access media from posted items in full resolution.
    c. shall be required to register or login in order to contact another registered-user who has posted an item.
2. **registered-users**
    a. shall have all the requirements of unregistered-users.
    b. shall be able to post digital media for sale or for free.
    c. shall be able to receive messages from other registered-users interested in posts.
    d. shall be able to reply to messages from other registered-users who have messaged them regarding .
3. **admin**
    a. admin shall have all requirements of unregistered-users and registered-users.
    b. shall be required to approve each media item before it can be publicly accessed in order to verify that it conforms to GaterTrader terms of use and content policies.
    c. shall be required to delete posts that violate the terms of use and content policy.
    d. shall be required to ban registered-users who violate the terms of use or content policies of GaterTrader.

2. Usability Test Plan

2.1 Test objectives:

In this test plan the uploading and posting of content will be tested. The purpose of this project is to allow SFSU sellers and buyers the opportunity to publish or purchase content within SFSU. Furthermore, uploading and posting will be the largest functionality that will allow both users access to what they need.

2.2 Test background and setup:

The setup of GaterTrader makes the usability for the seller painless. When first getting access to the site above will be the dashboard which clearly gives the option of logging in or making a post which are relevant for the seller in this case scenario. For GaterTrader in order to make a post you do need to have a registered account. Once there is an account and a post is ready to be made, the seller must fill out required cells and submit for review. Once reviewed and approved it will be posted on the website and ready for viewing.

Starting point once an account is created is filling out the content form. The required fields are the name of the content, the price, the subject and description. The name allows buyers to look up via name if they know it. The subject is another way to look up what content there is under a specific subject. Lastly, the description which allows the buyer to write down additional information about the content that they are selling.

The intended users for this specific function would be for the sellers. Uploading and posting gives sellers the opportunity to upload the content of their choice to GaterTrader. Once the post has been approved and uploaded, it will then allow the consumer user to look up, view, and purchase the content of their choice.

2.3 Usability Task description:

Tasks needed in order to make a post will be to:

Create an account on GaterTrader, once the account has been created and you are now logged in, create a post. Once a post has been created, it will be submitted to the administrator so they can review and approve it for posting. Once approved your content will be posted to GaterTrader and ready for users to potentially purchase the product.

2.4 Evaluation of Effectiveness:

The effectiveness that I'm evaluating on this functionality is that users are able to navigate the homepage as this is what sets the tone for the entire website. On the homepage making sure everything is visible (the users don't have to look around for anything) and easy to use.

Users should be able to create accounts with ease. Once that step is done, the process of creating a post should be as simple as possible. The effectiveness of the make a post form is the required field asking of too much or just enough from the user. Overall, the whole process of creating a post should be a smooth and timeless process.

2.5 Evaluation of Efficiency:

Some key points of efficiency would be the whole process of making an account and creating a post. Are there alot of questions that are required for the user to answer? The number of buttons that need to be pushed in order to get those tasks done. Also, once a post has been submitted for review in what time frame it will take to get approved. Is there somewhere on the make a post page that shows the approximate wait time for approval?

2.6 Evaluation of user satisfaction:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Navigating through GaterTrader was self explanatory.					
It was easy to create a post on GaterTrader.					
The overall process of creating an account was intuitive.					

Comments:

3. QA Test Plan

3.1 Test objectives

Test search functionality with queries that match and do not match items currently in the database. Will verify that results output matching queries when appropriate or no results (only suggestions) when appropriate.

3.2 Hardware/Software

- Laptop PC.
- Windows 10.
- URL: http://52.9.59.15:1234/

3.3 Browser: Mozilla Firefox v. 107.0.1

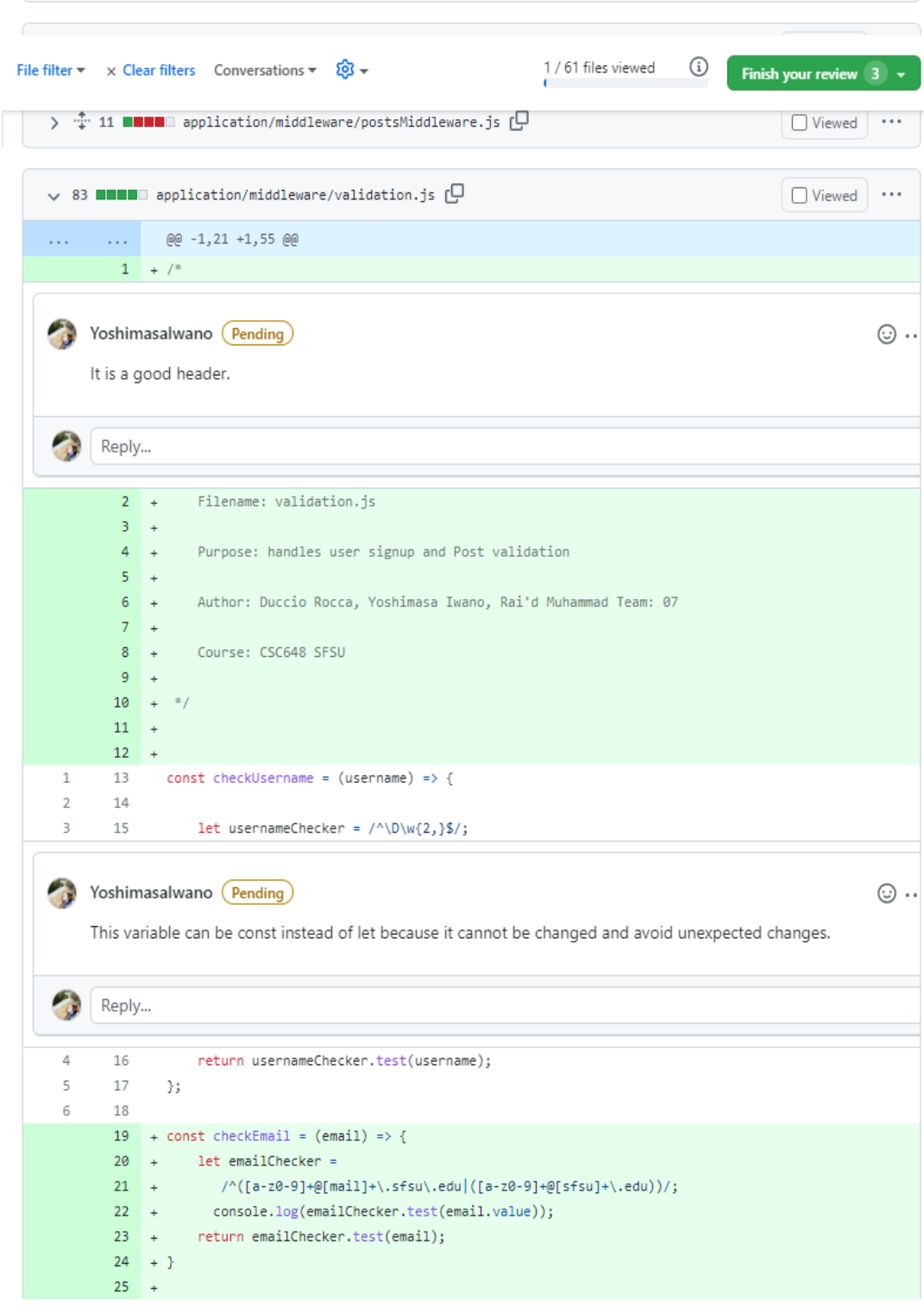
#	Test Title	Description	Test Input	Expected Output	Results
1	Single result	Perform a search with a query which has one match in the database.	“waterfall”	Waterfall	PASS
2	No result	Perform a search with a query which has no matches in the database.	“keyword”	“No results found”	PASS
3	Multiple results	Perform a search with a query which has four matches in the database.	“title”	test-Title1, test-Title2, test-Title3, test-Title4	PASS

3.4 Browser: Google Chrome v. 108.0.5359.96

#	Test Title	Description	Test Input	Expected Output	Results
1	Single result	Perform a search with a query which has one match in the database.	“waterfall”	Waterfall	PASS
2	No result	Perform a search with a query which has no matches in the database.	“keyword”	“No results found”	PASS
3	Multiple results	Perform a search with a query which has four matches in the database.	“title”	test-Title1, test-Title2, test-Title3, test-Title4	PASS

4. Code Review

This is a screen shot capture of commented reviewed code. The coder is Rai'd S. Muhammad and the reviewer is Yoshimasa Iwano.



```
 7  26  const checkPassword = (password) => {
 8  27      let passwordChecker = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[A-Za-z\d]{6,}$/;
 9  28      return passwordChecker.test(password);
10  29  };
11  30
    31  + const emailValidation = (req, res, next) => {
```

Yoshimasalwano **Pending**

The order of emailvalidation and usernamevalidation can be switched for better readability.

Reply...

```
    32  +     let email = req.body.email;
    33  +       if (!checkEmail(email)) {
    34  +
    35  +           req.flash('error', 'Invalid email; Minimum length 6 and it must end with @mail.sfsu.edu
    36  +           req.session.save(err => {
    37  +
    38  +               res.redirect("/signup");
    39  +           });
    40  +       } else {
    41  +           next();
    42  +       }
    43  + }
    44  +
12  45  const usernameValidation = (req, res, next) => {
13  46      let username = req.body.username;
14  47      if (!checkUsername(username)) {
15  48
16  49          req.flash('error', 'Invalid username; Minimum length 2 and it must start with an alphabet
17  50          req.session.save(err => {
18      -
    51  +           if(err) throw err
    52  +           console.log("there was an error with the login")
19  53              res.redirect("/signup");
20  54          });
21  55      } else {
22  56          next();
23  57      }
24  58  };
25  59
26  60  const passwordValidation = (req, res, next) => {
27  61
28  62      let password = req.body.password;
29  63      let confirmPass = req.body.matchPassword;
30  64
31  65      if (!checkPassword(password)) {
32  66
33  67          req.flash('error', 'Invalid password');
34  68          req.session.save(err => {
35  69
36  70              res.redirect("/signup");
37  71          });
38  72      } else {
39  73          if (password === confirmPass) {
40  74              next();
41  75          } else {
42  76              req.flash('error', "Passwords don't match");
43  77              req.session.save(err => {
```

```
                    res.redirect("/signup");
                });
            }

        }
    };

+ const postValidation = (req, res, next) => {
+
+     let title = req.body.PostTitle;
+     let description = req.body.PostDescription;
+     let fk_userId = req.session.userId;
+     let fileUploaded = req.file.path;
+
+     if (fileUploaded == null) {
+         req.flash('error', 'Invalid File');
+         req.session.save(err => {
+
+             res.redirect("/make-post");
+         });
+     } else {
+
+
+         if (title == null) {
+             req.flash('error', 'Invalid Title');
+             req.session.save(err => {
+
+                 res.redirect("/make-post");
+             });
+         } else {
+             if (description == null) {
+                 req.flash('error', 'Invalid Description');
+                 req.session.save(err => {
+
+                     res.redirect("/make-post");
+                 });
+             } else {
+                 if (fk_userId == null) {
+                     req.flash('error', 'Invalid User');
+                     req.session.save(err => {
+
+                         res.redirect("/make-post");
+                     });
+                 } else {
+                     next();
+                 }
+             }
+
+         }
+     }
+ };
+

- module.exports = {usernameValidation, passwordValidation};
+ module.exports = {usernameValidation, passwordValidation, postValidation, emailValidation};
```

5. Security Best Practices Self-check

Asset to be protected	**Types of possible/expected attacks**	**Your strategy to mitigate/protect the asset**
User's images	Malware and Phishing	Back-up and encrypt the data
User's email	Username enumeration	Making identical server responses (returning identical responses)
User's passwords	SQL injection and XSS	Uses or prepared statements and allowing-list input validation and encoding. Also providing secure input handling and have preparation if input handling fails
User's content uploaded	MiTM attack	Having secure connections, multi-factor authentication, and Endpoint security
Server's Database	Database DoS, packet sniffing, query string manipulation and SQL injection	Patching to prevent sticking to one authentication method. Set minimum amount of roles and using SSL and TLS encryption to prevent packet sniffing. Also creating a dedicated firewall

6. Non-functional Specs Self-check

#	Non-functional Requirement	Status
1	Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0.	DONE
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	All or selected application functions must render well on mobile devices.	ON TRACK
4	Data shall be stored in the database on the team's deployment server.	DONE
5	No more than 50 concurrent users shall be accessing the application at any time.	ON TRACK
6	Privacy of users shall be protected.	ON TRACK
7	The language used shall be English (no localization needed).	DONE
8	Application shall be very easy to use and intuitive.	ON TRACK
9	Application should follow established architecture patterns.	DONE
10	Application code and its repository shall be easy to inspect and maintain.	DONE
11	Google analytics shall be used.	ON TRACK
12	No email clients shall be allowed. Interested users can only message to sellers via in-site messaging. One round of messaging (from user to seller) is enough for this application.	DONE
13	Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.	DONE
14	Site security: basic best practices shall be applied (as covered in the class) for main data items.	ON TRACK
15	Media formats shall be standard as used in the market today.	ON TRACK
16	Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development.	DONE
17	The application UI (WWW and mobile) shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Fall 2022. For Demonstration Only" at the top of the WWW page nav bar. (Important so as to not confuse this with a real application).	DONE