

SW Engineering CSC648/848 Spring 2022

GatorTrade

Team05

Milestone 4

14 May 2022

Members:

Team Lead: Kishan Patel - kpatel0@sfsu.edu

Front End Lead: Michael Cheung

Front End Member: Joanne Wong

Back End Lead: Faisal Zaheer

Back End Member: Juan Hernandez

Github Master: Ze Lei

History Table:

| Date Submitted | Date Revised |
|----------------|--------------|
| 14 May 2022 | |

Table of Content

| | |
|---|-----------|
| Product Summary | 3 |
| Usability Test Plan | 4 |
| QA Test Plans | 6 |
| Code Review | 8 |
| Self-Check on Best Practices for Security | 11 |
| Self-Check on the Adherence to Original Non-functional specs | 12 |

1. Product Summary:

a. Name of Product: GatorTrade

b. Description of Product:

Our application is tailored to the students, faculty, and staff at SFSU. Belongings they may not want anymore or think others can benefit from can be listed here. The platform will allow for SFSU students, faculty, and staff to buy, sell, or exchange items with one another. This means that the used textbooks, electronics, and other used or handcrafted goods can find new owners amongst the SFSU community. Users of this application can search and filter through categories for the goods they are looking for such as technology, books, or crafts. Also, this application is exclusively for the SF State community only. It can also be an outlet for artistic students to showcase their skills and sell handmade goods to fellow students and SFSU employees.

c. Itemized list of committed P1 functions:

i. The application allows users to:

1. register, login, and logout.
2. search and browse items.
3. sort items by their price and title.

ii. Registered users are allowed to:

1. post items for sale.
2. message other sellers.
3. view the posts they've made and messages sent to them.
4. sort messages received by item and date.
5. change their password.

iii. Admins are allowed to:

1. approve posts made by users or delete an item listing.

d. Uniqueness:

- i. GatorTrade allows for its users to experience exclusivity since it is limited to only SFSU personnel and students. Therefore, users can find goods and items within the SFSU community easily.

e. URL to product: [GatorTrade URL Link](#)

2. Usability Test Plan:

a. Test Objective:

The purpose of this test is to assess the usability of the post item for sale function.

b. Test Background and Setup:

- i. System Set-Up: First, have access to a device that can connect to the internet. Next, enter or click on the link that leads to the GatorTrade website application.
- ii. Starting Point: The starting point to test this function is the homepage of the GatorTrade website application.
- iii. Intended Users: The intended users of this function are the students, staff, and faculty of San Francisco State University.
- iv. URL of function to be tested: <http://3.84.15.167:3000/post>

c. Usability Task Description:

First, make sure you are on the GatorTrade homepage. Next, go to the Posts page. Then, enter relevant information regarding the item you want to list. If prompted to register or log in, please do so. Afterwards, submit your post and check the status of the post in the My Pages section.

d. Evaluation of Effectiveness:

Effectiveness shall be measured by whether a user is able to upload a post to the website. The user should be able to access the homepage, go to the post form page, and enter relevant information to be uploaded for sale or exchange. This shall be measured by the number of successful completion of the function out of the total number of tests that are to be performed. Errors and bugs detected shall also be taken into account for the effectiveness of the function.

e. Evaluation of Efficiency:

Efficiency shall be measured by the amount of effort and time it takes the user to perform the function. The effort would be calculated from the number of webpages required to be accessed before being able to complete the task. The amount of time would be determined from the average time it takes each

successful completion of the task out of the average time of all tasks, completed or uncompleted.

f. Evaluation of User Satisfaction:

i. 1. It was easy to reach the form to upload an item to the website.

| Strongly Disagree | Disagree | Somewhat Disagree | Neutral | Somewhat Agree | Agree | Strongly Agree |
|-------------------|----------|-------------------|---------|----------------|-------|----------------|
| | | | | | ✓ | |

ii. 2. I was able to upload information about the item I wanted to list.

| Strongly Disagree | Disagree | Somewhat Disagree | Neutral | Somewhat Agree | Agree | Strongly Agree |
|-------------------|----------|-------------------|---------|----------------|-------|----------------|
| | | | | | | ✓ |

iii. 3. My experience with the user interface/design was pleasant.

| Strongly Disagree | Disagree | Somewhat Disagree | Neutral | Somewhat Agree | Agree | Strongly Agree |
|-------------------|----------|-------------------|---------|----------------|-------|----------------|
| | | ✓ | | | | |

3. QA Test Plans:


- a. Test Objectives:
 - i. The post item function is to be tested for any bugs.
- b. Hardware and Software Setup:
 - i. HW: A computer or smartphone that has a connection to the internet.
 - ii. SW: The test will start at this url: <http://3.84.15.167:3000/post> while the user is logged in or not logged in.
- c. Feature to be Tested:
 - i. Post functionality
- d. QA Test Plan - Table Format:

| Test No. | Test Title | Description | Test Input | Expected Output | Results: (Chrome) PASS/FAIL | Results: (MS Edge) PASS/FAIL |
|----------|-----------------------------|--|--|---|--------------------------------|---------------------------------|
| 1 | Post with user logged in | Test post function with title, category, price, description, and image field filled out and user logged in. | Title: iPhone 10 Category: Electronics Price: 300 Description: Used iPhone 10 for sale. Image: 1952 x 1384 png file | The Items table in the database should have a new row with all columns populated respectively and approved field 0 for false. | PASS | PASS |
| 2 | Post without user logged in | Test post function with title, category, price, description, and image fields filled out and user not logged in. | Title: iPhone 10 Category: Electronics Price: 300 Description: Used iPhone 10 for sale. Image: 1952 x 1384 png file | Users get redirected to login/register, but all fields data are saved for when they go back to the post page. | FAIL | FAIL |

| | | | | | | |
|---|-----------------|--|--|--|------|------|
| 3 | Size validation | Test title, price, and description fields for size validation. | Title: type a title with characters > 45. Category: Electronics Price: 2147483648 Description: type a description with characters > 1024. | Title only allows max 45 characters. Category will have Electronics. Price will not allow users to input numbers greater than 2147483647 or less than 0. Description will not allow characters > 1024 | FAIL | FAIL |
|---|-----------------|--|--|--|------|------|

4. Code Review:

Sent Emails



Faisal Zaheer

To: Kishan Ripalkumar Patel

Wed 5/11/2022 11:52 AM


Hi Kishan,

Here is the post feature code of the GatorTrade application code that needs to be code reviewed. I've attached a screenshot of the entire section of code that needs to be reviewed. This portion of the code is from the milestone4 branch on Github. The two files needing reviewed are the posts.js file, located in ./application/routes/posts.js for reference, and Posts.js, located in ./application/models/Posts.js, and the code is from lines 19 - 97 in the ./routes/posts.js file mentioned, and lines 10 - 21 in the ./models/Posts.js file mentioned. My suggestion for code review would be to include comments with the preface of KP for your name (for example: // KP: this is a comment from Kishan). I am ok with another form of written review, so long as you let me know how and where to find the suggestions/comments. Thank you very much Kishan.

Regards,

Faisal

Faisal Zaheer (Back End Lead)
fzaheer@mail.sfsu.edu



Kishan Ripalkumar Patel

To: Faisal Zaheer

Fri 5/13/2022 7:23 AM

Hey Faisal,

I've reviewed the code and pushed the commented files on the milestone4 branch.

I've added all my comments at the bottom of both files and added line numbers where changes are required. I think this would make it easier for you as you won't need to track each comment separately.

Here's my summary:

1. Header comments are present for each file.
2. In line comments are ample and explains code sufficiently.
3. Method and variable names are consistent. They properly define the role of the method or variable.
4. Not a lot of coding errors overall. I've addressed the few that are in my comments in the code.

Thanks a lot Faisal.

Best,
Kishan (Team Lead)

Reviewed Code


```

routes > # posts.js > router.post('/createPost') callback > then() callback
19 // Creates storage in public/images folder to store user uploaded images.
20 var storage = multer.diskStorage({
21   destination: function(req, file, cb) {
22     cb(null, "public/images");
23   },
24   filename: function(req, file, cb) {
25     let fileExt = file.mimetype.split('/')[1];
26     let randomName = crypto.randomBytes(22).toString("hex");
27     cb(null, `${randomName}.${fileExt}`);
28   }
29 });
30
31 // Connects established multer storage to upload files.
32 var uploader = multer({storage: storage});
33
34 // CREATE POST:
35 router.post('/createPost', uploader.single("itemImage"), (req, res, next) => {
36   // Uploaded File + Created Thumbnail paths
37   let fileUploaded = req.file.path;
38   let filePath = "images/" + req.file.filename;
39   let fileAsThumbnail = `thumbnail-${req.file.filename}`;
40   let destinationOfThumbnail = req.file.destination + "/thumbnails/" + fileAsThumbnail;
41   let thumbnailPath = "images/thumbnails/" + fileAsThumbnail;
42
43   // Targeted Post Item form input data:
44   let title = req.body.itemTitle;
45   let category = req.body.category;
46   let categoryId = parseInt(category);
47   let price = req.body.itemPrice;
48   let description = req.body.itemDesc;
49
50   // Gets seller's user id from their logged in session:
51   let seller = req.session.userId;
52
53   if(seller == undefined){
54     res.redirect("/login");
55   }
56
57   // Sharp resizes uploaded image to a thumbnail version of the image
58   // with a 200x200 size, then exports to destinationOfThumbnail,
59   // in thumbnails folder.
60   else{
61     sharp(fileUploaded)
62       .png()
63       .resize(200, 200, {
64         // Sets image to fit the 200x200 dimensions as much as possible while prioritizing its aspect ratio.
65         fit: 'contain',
66         background: { r: 255, g: 255, b: 255, alpha: 0 } // Empty space is transparent.
67       })
68       .toFile(destinationOfThumbnail)
69       .then(() => {
70         // Sends user inputted post info to the post model to enter into the DB.
71         return PostModel.create(title, categoryId, description, filePath, thumbnailPath, price, seller)
72       })
73       .then((postWasCreated) => {
74         if(postWasCreated) {
75           console.log("Your post was created successfully!");
76           req.flash('success', "Your post was created successfully!");
77           // Redirects to home page after successful post creation.
78           req.session.save( err => {
79             res.redirect('/');
80           });
81         } else {
82           console.log("Your post was NOT created.");
83           throw new PostError('Post could not be created!!', '/post', 200);
84         }
85       })
86       .catch((err) => {
87         if(err instanceof PostError) {
88           errorPrint(err.getMessage());
89           req.flash('error', err.getMessage());
90           res.status(err.getStatus());
91           res.redirect(err.getRedirectURL());
92         } else {
93           next(err);
94         }
95       });
96   }
97 });

```

```

models > Posts.js > create > then() callback
9
10 // Handles sending the Post info to the database to create a new Post record.
11 PostModel.create = (title, category, description, photopath, thumbnail, price, seller) => {
12   let baseSQL = 'INSERT INTO GatorTrade.Items (title, category, description, photopath, thumbnail, price, seller) VALUES (?, ?, ?, ?, ?, ?, ?)';
13   console.log("Parameters:", title, category, description, photopath, thumbnail, price, seller);
14   return db
15     .execute(baseSQL, [title, category, description, photopath, thumbnail, price, seller])
16     .then((results, fields) => {
17       // Returns whether the Promise statement of creating a new post has succeeded or not.
18       return Promise.resolve(results && results.affectedRows);
19     })
20     .catch((err) => Promise.reject(err));
21 };

```

Code Review Feedback

Feedback for ./application/routes/posts.js

```

// KP: Code Review by Kishan Patel for Milestone 4
// 1. Header and in-line comments are properly used. Provide all the information required and makes understanding the code really easy.
// 2. Method and variable names are consistent. The names clearly defines the role of the method or variable.
// 3. Line 60. Try to put the else part directly below if. Nothing in between.
// 4. Line 53. Not a big issue. I would just recommend using === instead of ==

```

Feedback for ./application/models/Posts.js

```

// KP: Code Review by Kishan Patel for Milestone 4
// 1. Header and in-line comments are properly used. Provide all the information required and makes understanding the code really easy.
// 2. Method and variable names are consistent. The names clearly defines the role of the method or variable.
// 3. In the header comment for this file, name of the author is missing.
// 4. Line 12. There is a console log present. Remove it if not required.
// 5. Line 13. Not a big issue. I think it would be better if the .execute will either be on the same line, directly following
// db or indented directly below db instead of return.( Just a suggestion )

```

5. Self-Check on Best Practices for Security:

| Asset to be Protected | Types of Possible/Expected Attacks | Strategy to Mitigate/Protect Asset |
|-----------------------|------------------------------------|---|
| User Passwords | Database Breach | Password Encryption (bcrypt) |
| Input Validation | Invalid input -> app crash | Try catch blocks, regex validation on client and server |
| Product Information | Invalid input | Try catch blocks, client and server javascript |
| User email | Invalid email | Regex on client and server checking for sfsu.edu |
| Search Bar Input | Invalid input | Try catch blocks, client and server regex |
| Search Bar Input | SQL injection | Use of ? in SQL query |

6. Self-Check on the 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.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.ON TRACK
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.DONE
7. The language used shall be English (no localization needed).DONE
8. Application shall be very easy to use and intuitive.DONE
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.DONE
15. Media formats shall be standard as used in the market today.DONE
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, Spring 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