SW Engineering CSC648/848 Fall 2022
Gatormmunity

Team 7

Anthony Zhang (Team Lead),
azhang12@mail.sfsu.edu

Marwan Alnounou

Mohamed Sharif

Jose Lopez

Florian Cartozo

Milestone 4
12/14/2022

History Table

Version	Submission Date
M4V1	12/01/2022
M4V2	12/14/2022

Table of Contents

1) Product Summary

Name of the Product: Gatormmunity

All Major Committed Functions:

1. All users shall be able to contact the developers via the Contact Us page.
2. All users shall be able to view the home page.
3. All users shall be able to view the about page to learn about Gatormmunity.
4. A guest user shall be able to view the registration page.
5. A guest user shall be able to view the login page.
6. A guest user shall be able to register for an account.
7. An approved user shall be able to log in to their account.
8. An approved user shall be able to reset their password by contacting the admins via the Contact Us page.
9. An approved user shall be able to report other users via the Contact Us page.
10. An approved user shall be able to sort forum threads.
11. An approved user shall be able to create a forum thread.
12. An approved user shall be able to assign their forum thread to a category.
13. An approved user shall be able to create a forum post.
14. An approved user shall be able to attach an image when creating a thread. The image will be displayed in the first post of the thread.
15. An approved user shall have a profile page.
16. An approved user's profile page shall show their details, e.g. their full name.
17. An approved user's profile page shall show their recent activities, e.g. their recently created listings or threads.
18. An approved user shall be able to view a user's profile page.
19. An approved user shall have a profile picture.
20. An approved user shall be able to change their profile picture.
21. An approved user shall be able to create a user group.
22. An approved user shall be able to join a user group after being invited to it, via the link provided in the invite message.
23. An approved user shall be able to send direct messages to other users.
24. An approved user shall be able to receive direct messages from other approved users.
25. An approved user shall be able to send messages in Gator Chat.
26. An approved user shall be able to see marketplace listings in the marketplace.
27. An approved user shall be able to select marketplace listings in the marketplace.
28. An approved user shall be able to report marketplace listings via the Contact Us page.
29. An approved user shall be able to buy items from the marketplace by contacting a listing's seller.
30. An approved user shall be able to sell items on the marketplace by creating a listing.
31. An approved user who is selling an item shall be able to assign their listing to a category.
32. An approved user shall be able to upload a picture of an item they are selling.

33. An approved user shall be able to delete their own listings from the marketplace.
34. An approved user shall be able to search for users.
35. An approved user shall be able to search for marketplace listings.
36. An approved user shall be able to search for forum threads.
37. An approved user shall be able to apply filters to their user search results.
38. An approved user shall be able to apply filters to their marketplace search results.
39. An approved user shall be able to apply filters to their forum thread search results.
40. A moderator shall be able to approve unapproved users via the Admin page.
41. A moderator shall be able to reject unapproved users via the Admin page.
42. A moderator shall be able to ban unapproved and approved users from Gatormmunity via the Admin page.
43. A moderator shall be able to delete forum threads.
44. A moderator shall be able to delete forum posts.
45. A moderator shall be able to delete listings in the marketplace.
46. An administrator shall be able to appoint moderators via the Admin page.
47. An administrator shall be able to unappoint moderators via the Admin page.
48. An administrator shall be able to ban moderators from Gatormmunity after first unappointing the moderator via the Admin page.
49. An administrator shall be able to change a user’s password via the Admin page.
50. A group member shall be able to invite other users to their group.
51. A group member shall be able to leave their group.
52. A group member shall be able to create forum threads in their group’s forum.
53. A group member shall be able to create forum posts in their group’s forum.
54. A group member shall be able to send messages in their group’s chat.
55. A group moderator shall be able to kick group members from their group via the Group Members page.
56. A group moderator shall be able to delete forum threads in their group’s forum.
57. A group moderator shall be able to delete forum posts in their group’s forum.
58. A group administrator shall be able to delete their group.
59. A group administrator shall be able to appoint group moderators via the Group Members page.
60. A group administrator shall be able to unappoint group moderators via the Group Members page.
61. A group administrator shall be able to kick group moderators from the group after first unappointing the moderator via the Group Members page.
62. A group administrator shall be able to edit the announcement on their group’s home page.

What is Unique about our Project:

Gatormmunity distinguishes itself from its competitors like Craigslist and OfferUp by having mandatory identity verification by requiring a picture of one's SFSU ID card in order to register. This makes our platform safer because all registered users will know every other user is a member of SFSU and has been vetted by Gatormmunity's moderators. Also, we have a Forums page that allows members of Gatormmunity to talk to each other and plan events.
Furthermore, users can form groups with others and have their own private group chat and forum. This lets friends or people who share interests talk to each other in private. Gatormmunity also has a live chat (Gator Chat) for all of its members, where users can talk to one another in real time.

Product URL: http://54.241.101.69/

Since new accounts must be approved by a moderator or administrator in order to log in, we provided some credentials to some already approved accounts at the bottom of the Credentials README on GitHub. There is one account for each permission level: one for Approved Users, one for Moderators, and one for Administrators.

2) Usability Test Plan

Usability Test Plans for each of the 6 Test/Use Cases:

Test/Use Case #1: Create a Forum Thread

Test Objectives:
We are testing if the user can find the Gatormmunity Forums page, click on the “Create Thread” button, fill out the form on the Create Thread page, and optionally, if they are able to use the resources in the Helpful Links section of the Create Thread page. We are also testing if the user is able to recognize that their forum thread was successfully created, since we received feedback that sometimes it was not clear if an action was successful.

We want the user to be able to find the Gatormmunity Forums page easily because it is a key part of our website, and we need to see if the user can quickly locate the Create Thread button since that is how users will post events and questions to the Gatormmunity Forums. We note whether the user clicks on a link in the Helpful Links section because we feel that the helpful links are hard to notice. One of our users in Milestone 3 thought uploading an image for a thread was mandatory when it was optional, so we will observe whether or not a user tries to upload an image.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
The user should begin the test from the dashboard page after logging in because this is the first test, and presumably the user will not yet have logged into the test account provided for them. Ultimately, it does not make a difference where the user starts because the navigation bar which houses the Forums button is accessible on every page. However, the user must be logged in to see the Forums button in the navigation bar.

Intended Users:
This test is intended for every registered user because the Forums page is one of our core functionalities, and should be easy to access for users of almost all skill levels. The user should at least have some prior experience with using websites because our website requires that the user know that the navigation bar has clickable links. This may be a concern for some students and faculty because not every member of SFSU is familiar with websites.

URLs to be Tested:
Gatormmunity Forums Page (accessible from the navbar): http://54.241.101.69/forums
Create Thread Page (accessible from the Forums page): http://54.241.101.69/create-thread
View Thread Page (the user is redirected here after creating a thread): http://54.241.101.69/thread/N, where N is an integer representing the id of the thread.

What is Being Measured:
We are measuring if the user is able to rapidly find the Forums button in the navigation bar, if the user is able to find the Create Thread button in the Gatormmunity Forums page, if the user is able to fill out the form without getting confused, and if the user is able to recognize that their thread was successfully created after pressing the Submit button on the Create Thread page.

Test/Use Case #2: Post in a Forum Thread

Test Objectives:
We are testing if the user is able to recognize what a thread looks like on the Gatormmunity Forums page, if the user will know that a thread is clickable on the Gatormmunity Forums page, if the user will know where to find the Create Post form, if the user knows how to submit the form, and if the user will recognize that their post was successfully submitted. We are also testing if the user can find the "Back to Forums" button in the top left in case they want to find some other thread to post in.

We are testing the user's ability to recognize threads on the Forums page because being able to view other people's threads and the posts within them is one of the ways in which a user can interact with Gatormmunity's community. We added a darken on hover effect to the threads on the Forums page so that the user can recognize that the threads are clickable, which we are testing for. Since posting in a thread is how you can reply to other users in a thread, we want to ensure that the process is easy and intuitive for our users. Again, we want to test if the user knows their post was successfully posted because we received feedback saying it was not so obvious in Milestone 3.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
The user will begin the test from the View Thread page (i.e. they are already viewing a thread) because this test will take place immediately after the user created a thread from Test #1. Since it would be too simple for the user to post in their newly created thread, we ask that the user find some other interesting thread in the Forums page to post in. We want the user to experience the process of

posting to a thread from the very beginning, and we need to test if the user is capable of doing this without support.

Intended Users:
This test is intended for every registered user because being able to post in a thread is one way users can communicate with one another on Gatormmunity. The forums feature is one of our key features, so posting to a thread should be easy for users of almost all skill levels. The user should at least have some prior experience with using websites because posting to a thread requires that the user know that something darkening on hover and the cursor changing to a hand means that the item being hovered over is clickable.

URLs to be Tested:
View Thread Page (the user starts on this page due to the test starting immediately after Test #1): http://54.241.101.69/thread/N, where N is an integer representing the id of the thread.
Gatormmunity Forums Page (the user gets here via the "Back to Forums" button or from the Forums button in the navbar): http://54.241.101.69/forums
View Thread Page (this is a different thread than the one the user started on. The user gets here by clicking on a thread from the Gatormmunity Forums page): http://54.241.101.69/thread/M, where M is an integer representing the id of a different thread from the one the user started the test on.

What is Being Measured:
We are measuring if the user can find out how to go back to the Forums page via either the Back to Forums button or navigation bar, if the user can find a thread they want to post to in the Forums page, if they can find where to type text into to post to the thread, and if they are able to recognize that their post was successful.

Test/Use Case #3: Sell an Item

Test Objectives:
We are testing if the user knows that the "Market" button in the navigation bar leads to the marketplace page where users can sell things, if the user knows that creating a listing means that they are selling an item, if the user knows how to fill out the Create Listing form, and if the user is able to see their listing after submitting the create listing form.

We test the user's ability to find the "Market" button in the navigation bar because the marketplace is another one of our key functionalities, so it should be easily findable by Gatormmunity's users. We are checking if the term "Create Listing" is clear to users that it is how they can sell items and services on Gatormmunity. We ask that the user fill out the Create Listing form because we want to test if it is clear what the user should enter in each of the text fields. We also want to see how the user reacts to learning that a listing photo is required to create their listing. We test if the user can see their listing since we want to ensure that the user knows that their listing creation was successful.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
Since the test occurs after Test #2, the user will likely start from the View Thread page, though it does not matter where they start because the navigation bar is available on every page. Naturally, the user must be logged in to access the marketplace page.

Intended Users:
This test is intended for every registered user because we want everyone to know how to create a listing in case they would ever want to do so in the future. The ideal user for this feature should be aware of the risks of meeting up with another person to trade because even though the risk of an incident is less with Gatormmunity compared to its competitors, the risk is still there.

URLs to be Tested:
Marketplace Page (accessible from the navbar): http://54.241.101.69/marketplace
Create Listing Page (accessible from the marketplace page): http://54.241.101.69/create-listing
View Listing Page (the user is redirected here after creating a listing): http://54.241.101.69/listing/N, where N is an integer representing the id of the listing.

What is Being Measured:
We are measuring if the user can find the Marketplace page without incident, if the user recognizes that the Create Listing button is what they need to press to sell an item, if the user correctly fills out the required fields in the form, and if the user successfully creates the listing. We observe the reaction of the user after being redirected to the View Listing page showing their newly created listing, in case there is any confusion about what they are seeing.

Test/Use Case #4: Create a User Group

Test Objectives:
We are testing if the user is able to successfully find the Create Group button on the Users Groups page accessible from the navigation bar. We are testing if the user understands what "User Group" means because it might not be the best terminology for referring to groups that users can form with their own private forums and chat rooms. We test if the user can successfully fill out the Create Group form, and if the user is satisfied with the appearance of the Group Home page.

We test the user on their ability to find the Groups page because this is how users can view the list of groups they are in, so being able to find the Groups page without trouble is important. We test that users can successfully create a group because at first, new users will not be in any groups, and will want to either create their own group or join one. Since any approved user can create a group, we believe many users will want to create one to see what being in a group is like, therefore we want to ensure that creating a group is a pain free process for the user. Finally, we ask that the user give their thoughts on the appearance of the group home page because we feel that the group home page could benefit from having more content on it, but we also do not wish to overwhelm the user.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
Since the test occurs after Test #3, the user will likely start from the View Listing page, though it does not matter where they start because the navigation bar is available on every page. The user must be logged in to access the Users Groups page.

Intended Users:
This test is intended for every registered user since we believe that being able to join groups just to see what groups are like is something many users will try at least once. Anybody who wants to create a group must go through the process the tester goes through, therefore it is important that this feature works as expected.

URLs to be Tested:
Users Groups Page (accessible from the navbar): http://54.241.101.69/groups
Create Group Page (accessible from the users groups page): http://54.241.101.69/create-group
Group Home Page (the user is redirected here after creating a group): http://54.241.101.69/group/N, where N is an integer representing the id of the group.

What is Being Measured:
We are measuring if the user can find the Users Groups page without problems, if the user recognizes that the Create Group button is what they need to press to “Create a User Group”, if the user correctly fills out the required fields in the form, if the user successfully creates the group, and the user’s reaction to the new group’s home page shown upon creating a group.

Test/Use Case #5: Send Messages in Group Chat

Test Objectives:
We are testing if the user is able to find the Group Chats page, if the user is able to find the group chat room they are looking for on the Group Chats page, if the user can see the chat messages from other group members, if the user is able to send a message in the group chat, if the user can see their sent chat message, and if the chat messages successfully update in real time.

We test if the user can find the Group Chats page since group chats are a significant feature of Gatormmunity, and we want the chat room to be easy to find and get to. We test if the user can see chat messages from other group members because we want to make sure that fetching the messages from the database works without problems. We test if the user can send messages because we want to see if the user has any difficulties or complaints about the message sending process. Lastly, we test that the user can see their message show up in the chat log because if a user cannot see their messages in the chat log, then they will not know if their message sending was successful.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
Since the test occurs after Test #4, the user will likely start from the Group Home page, though it does not matter where they start because the navigation bar is accessible on every page. The user must be logged in to access the Group Chats page.

Intended Users:
This test is intended for every registered user. If the group chat function works, then Gator Chat also works, and all approved users have access to Gator Chat whether they are in a group or not. As a result, we must ensure that the Group Chats page is easy and intuitive to find for all of our registered users.

URL to be Tested:
Group Chat Page (accessible from the navbar): http://54.241.101.69/chat
Different group chats are accessible from the same URL by clicking on the group chat rows on the left column of the Group Chat page.

What is Being Measured:
We are measuring how quickly the user can locate the Group Chats page, if they are able to find a particular group to send chat messages in, if they are able to see the chat messages for a particular group on the screen, how quickly the chat messages load, if the user is able to successfully send a message, and how long it takes for the sent message to appear in the chat message log.

Test/Use Case #6: Search

There are 6 tests rather than 5 since it was said in class that 5 is the minimum number of tests we must have rather than the exact amount we must have.

Test Objectives:
We are testing that the search bar in the navigation bar successfully takes user input if any was provided, if the search bar takes the user to the search results page, if the search page shows the expected results, if the search filters work, if the search results are able to be clicked on, and that the clicked on search result takes the user to the correct page.

We test the search bar because virtually every user will use it, hence it must be good enough such that users are satisfied with it. We want to ensure that the search message and search results do not confuse the user because we do not want the user to avoid using the search bar, we want them to use it as much as possible if it will help them find what they want faster. We test the search filters since they allow a user to refine their search results and change what they are searching for. We test the search results are laid out correctly because styling and formatting has been a recurring problem, and we test that the search results take the user to the correct page because we want the user to be able to see more about the item they are searching for, e.g. the profile page of the user, the listing details and contact details for a listing, or a thread’s posts.

Test Description

System Setup:
The user will be performing this test on their computer or laptop using a supported operating system (Windows 10, Windows 11, or macOS Monterey 12.5/12.6) with a supported browser (Google Chrome 105, Microsoft Edge 105, or Safari 15.5). The user must have reliable internet access.

Starting Point:
Since the test occurs after Test #5, the user will likely start from the Group Chat page, though it does not matter where they start because the search bar and the navigation bar are accessible on every page. The user does not need to be logged in to access the search page, but to be able to search for listings and threads requires a logged in user.

Intended Users:
This test is intended for logged in users, but non-logged in users can use the search bar too, just with less functionality such as not being able to search listings and threads, and not being able to see profile pages. Every logged in user will be using the search bar, so every one of our testers must interact with the search bar so that we can determine its problems.

URL to be Tested:
Search Page (accessible from the search bar in the navbar): http://54.241.101.69/search OR http://54.241.101.69/search/X, where X is the user's search terms, if any were entered into the search bar. We allow no input into the search bar, which acts as a search for all items regardless of their name or title, although other filters such as category and price can still be applied.

What is Being Measured:
We are measuring how quickly the user is able to find what they are looking for, if they were confused on any part of the searching process, if the search was able to display recommendations on a search that matches nothing, how quickly the search results load, and how well the search results look on the tester's computer, since screen sizes have an effect on how many columns there are or how wide a search result is, for instance.

Usability Test Table for Measuring Effectiveness and Efficiency:

Test/Use Case	% Completed	Errors	Comments	% in Time Completed
Create a Forum Thread	100%	None	One tester was not aware they could click on the text in the navbar. Most testers did not click on the helpful links. Most testers typed only a few words or a sentence in their thread body, despite the massive text field.	100%
Post in a Forum Thread	100%	None	One tester did not see the "Back to Forums" button at the top of the View Thread page; we made the button text larger as a result. One tester suggested that we use a rich text editor instead of a simple plaintext input form. No testers made use of the category filter on the Forums page.	100%

			Asking the tester to post in a different forum thread than the one they created in Test #1 made the test more confusing for them.	
Sell An Item	100%	One tester tried to upload a PDF as the listing's image, which caused an alert message to show telling the user to upload an image with one of the accepted image formats. One tester did not upload an image, which caused the browser to tell the tester: "Please select a file."	One tester did not have any images on their computer to upload for the listing's image, so they had to find an image online, save it, then upload it. We should consider making images optional to increase the task's efficiency. One tester said we should add more categories for our listings. One tester tried to use AirDrop to upload a file. It did not work, since we do not support AirDrop.	75%
Create a User Group	100%	One tester did not upload an image, which caused the browser to tell the tester: "Please select a file."	One tester said that it does not look too good to have the group's name and "Announcement" be centered, while the group's description and announcement text are left-aligned. One tester said that we should round the profile picture thumbnail in the navbar because images look better rounded, such as in the group home page.	100%
Send Messages in Group Chat	100%	None	One tester asked how come they could not see any messages in their group chat. The chat log was empty because their group was newly created and thus did not have any messages sent in it yet. We should have sent some messages beforehand in the groups the test user was in. One tester said the message input box covered up some of the chat messages in the message log. This issue could not be reproduced on the observer's computer despite resizing the window, zooming in/out, etc.	100%

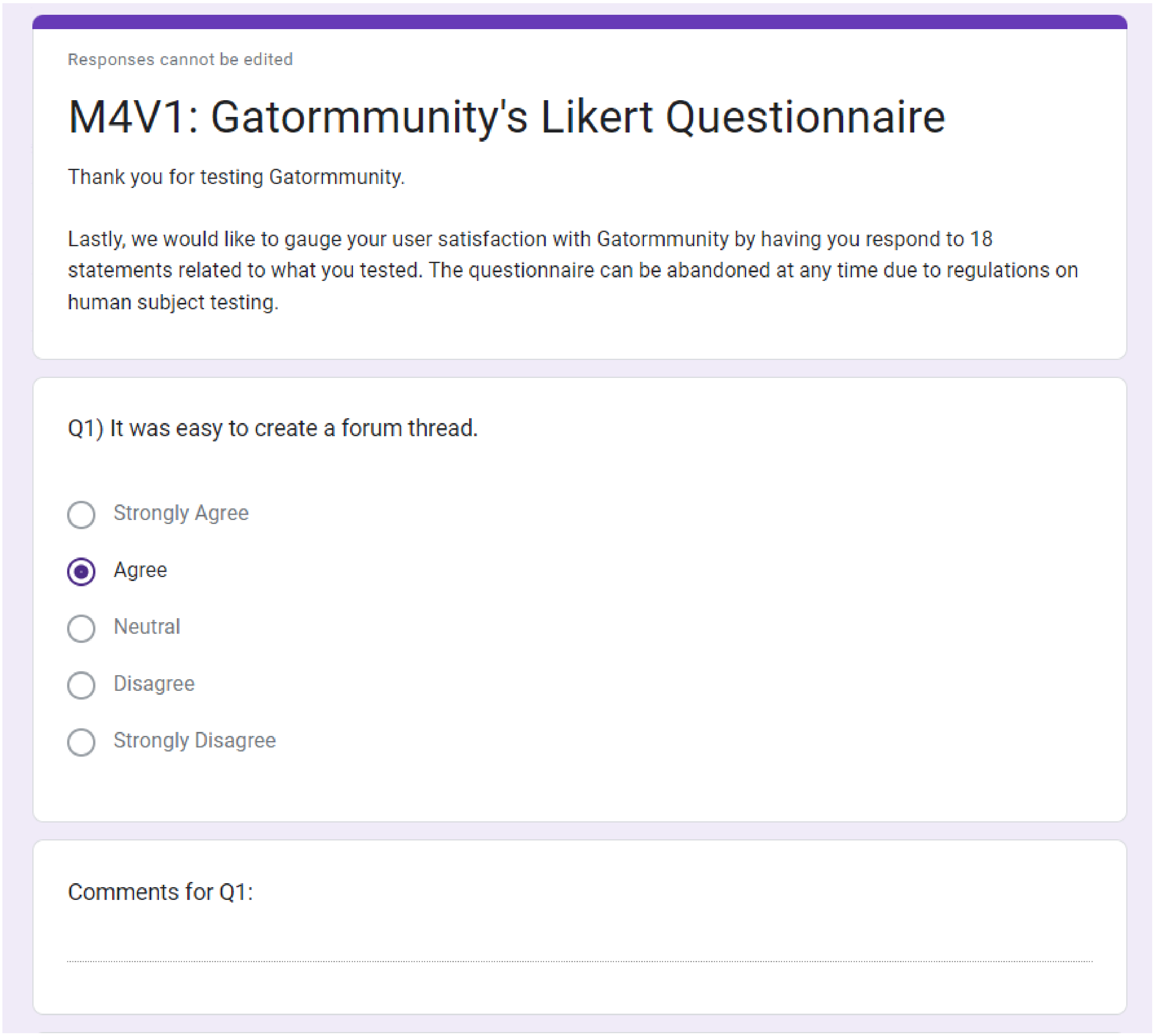
Search	66%	None	The search page works but the testers were not sure how to change the search category from users to listings and asked the observer how to change the search category. Multiple testers did not like how Users was the default search category because they were expecting to see listings but saw users instead, which confused them. We should make it more obvious that the user can change their search category in the left column where the search filters are. Alternatively, we could consider having the search page search for users, listings, and threads at the same time and have all of the matching items show in the same search results page. Then, the user can narrow down their search category with the search filters. Or, we can add a message to the top of the search results page telling the user they can switch their search category with the search filters on the left. We will need to revise the search page so that nobody struggles with finding what they are looking for (listings, more often than not).	0%

User Satisfaction using a Likert Questionnaire:

We used Google Forms to create a Likert Questionnaire, which is accessible here: https://forms.gle/GvjsXjuFmMoBQXjbA

There are 18 different statements, 3 for each function tested.

Sample Response:



Responses cannot be edited

M4V1: Gatormmunity's Likert Questionnaire

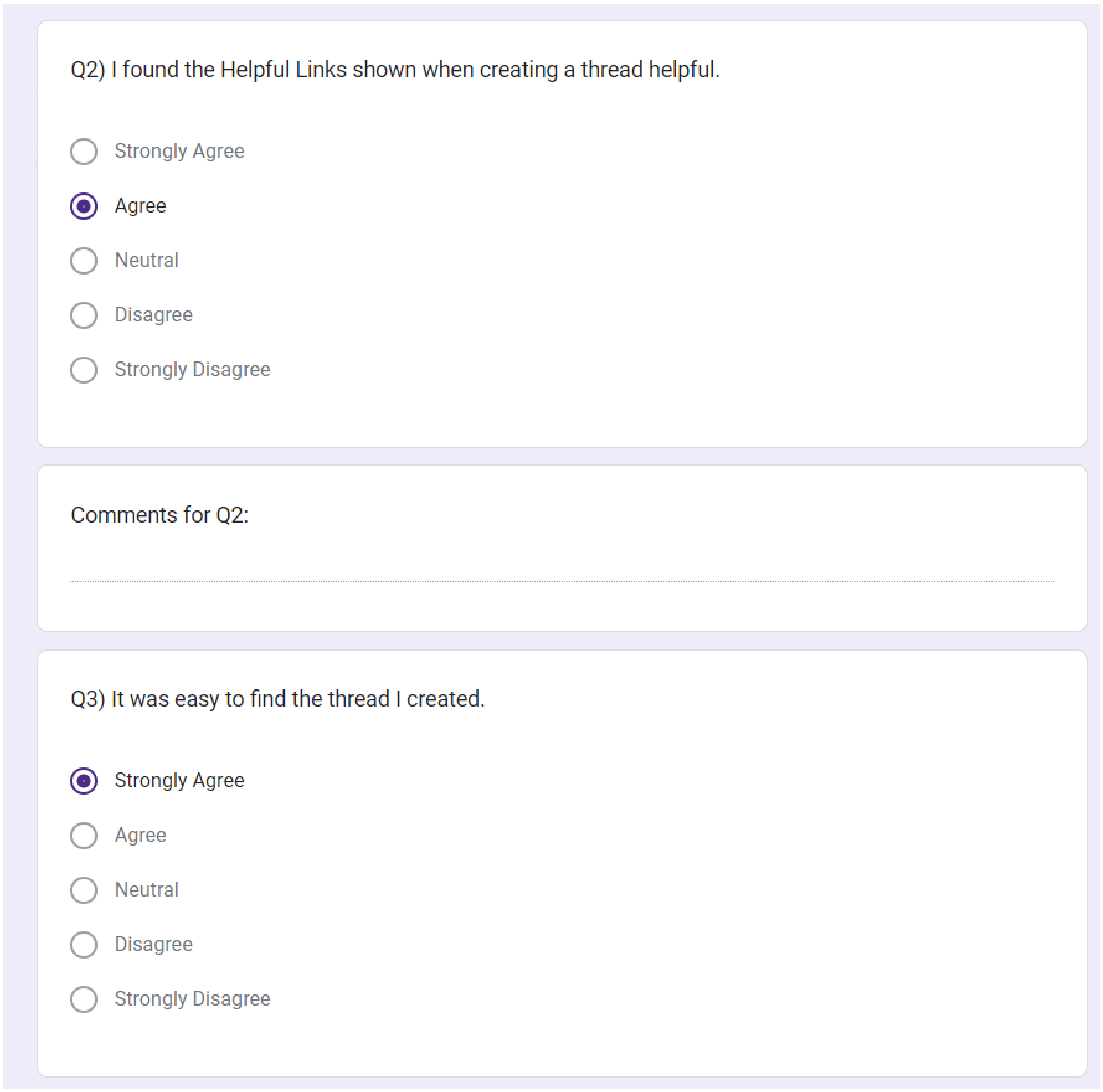
Thank you for testing Gatormmunity.

Lastly, we would like to gauge your user satisfaction with Gatormmunity by having you respond to 18 statements related to what you tested. The questionnaire can be abandoned at any time due to regulations on human subject testing.

Q1) It was easy to create a forum thread.

Strongly Agree [✓ Agree] Neutral Disagree Strongly Disagree

Comments for Q1:



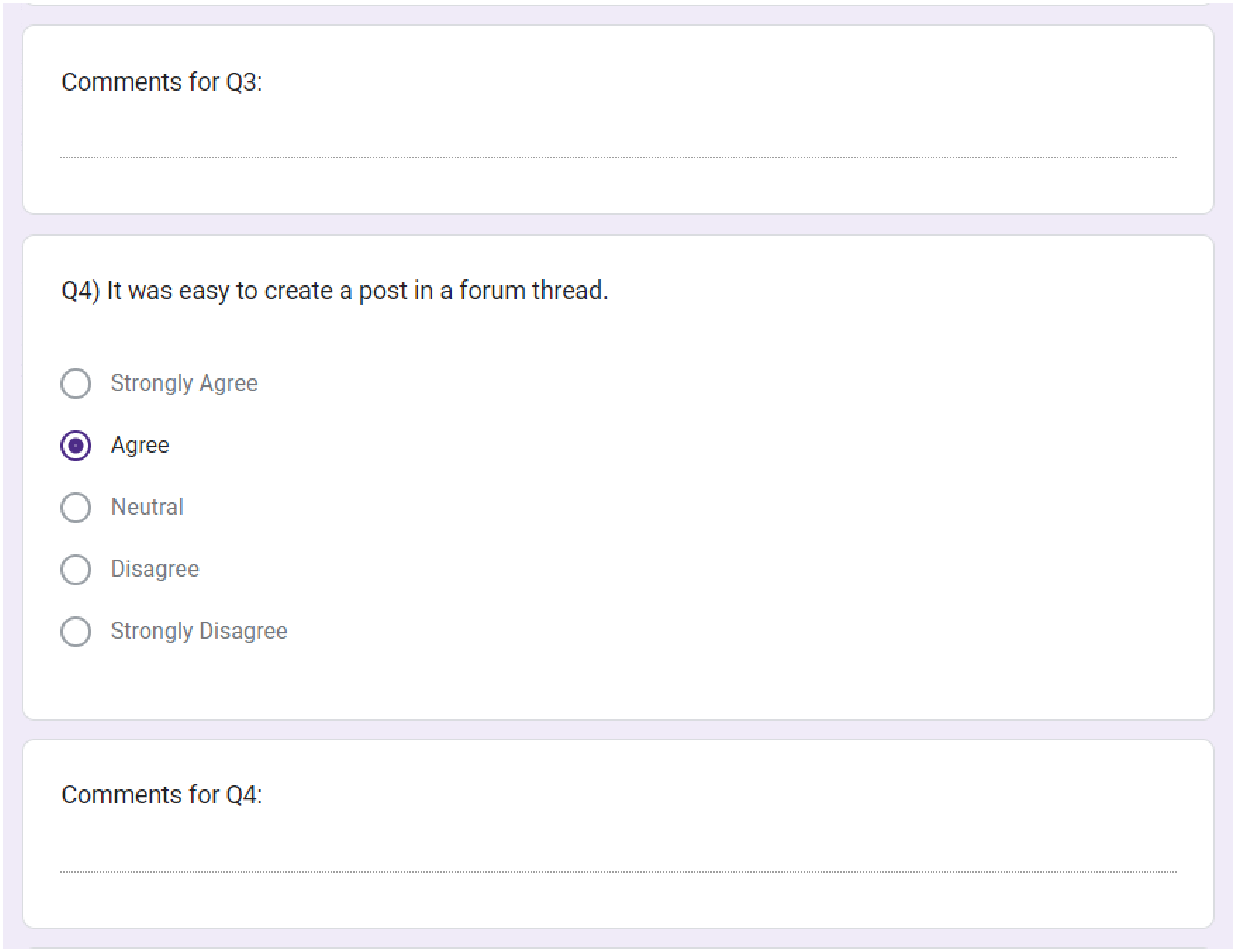
Q2) I found the Helpful Links shown when creating a thread helpful.

- ○ Strongly Agree
- [✓ Agree]
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

Comments for Q2:

Q3) It was easy to find the thread I created.

- [✓ Strongly Agree]
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

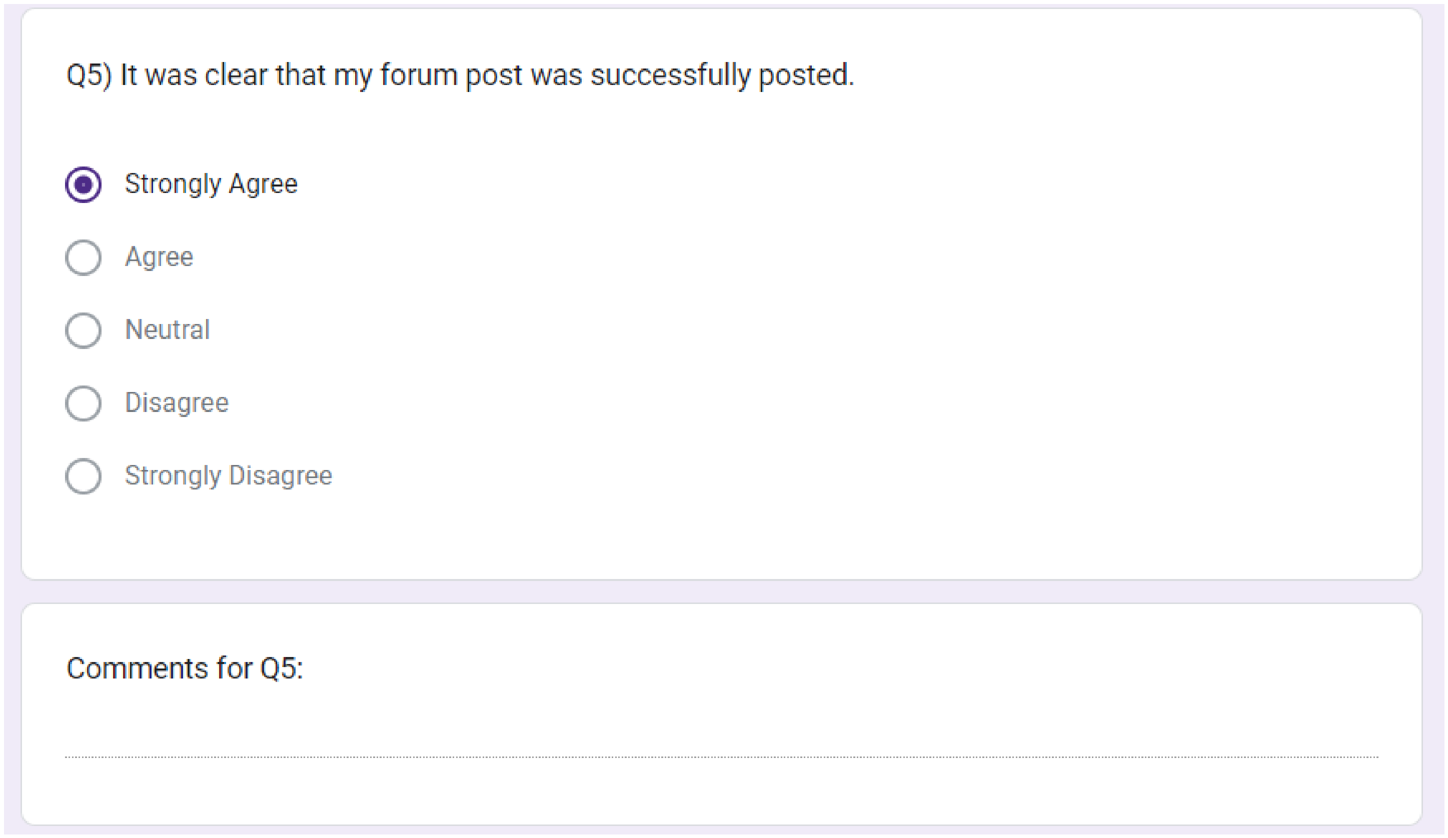


Comments for Q3:

Q4) It was easy to create a post in a forum thread.

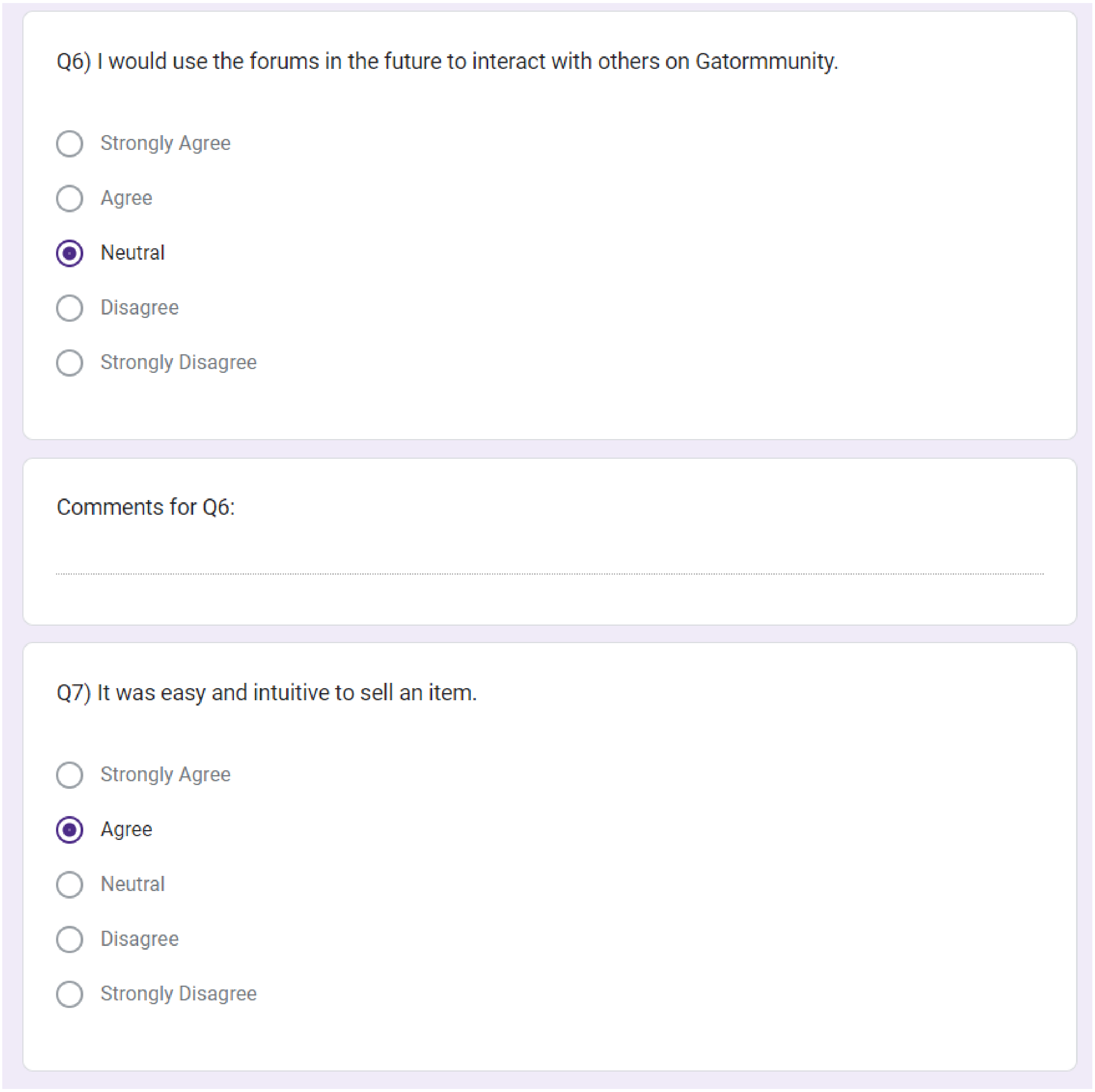
Strongly Agree [✓ Agree] Neutral Disagree Strongly Disagree

Comments for Q4:


Q5) It was clear that my forum post was successfully posted.

[✓ Strongly Agree]
Agree
Neutral
Disagree
Strongly Disagree

Comments for Q5:



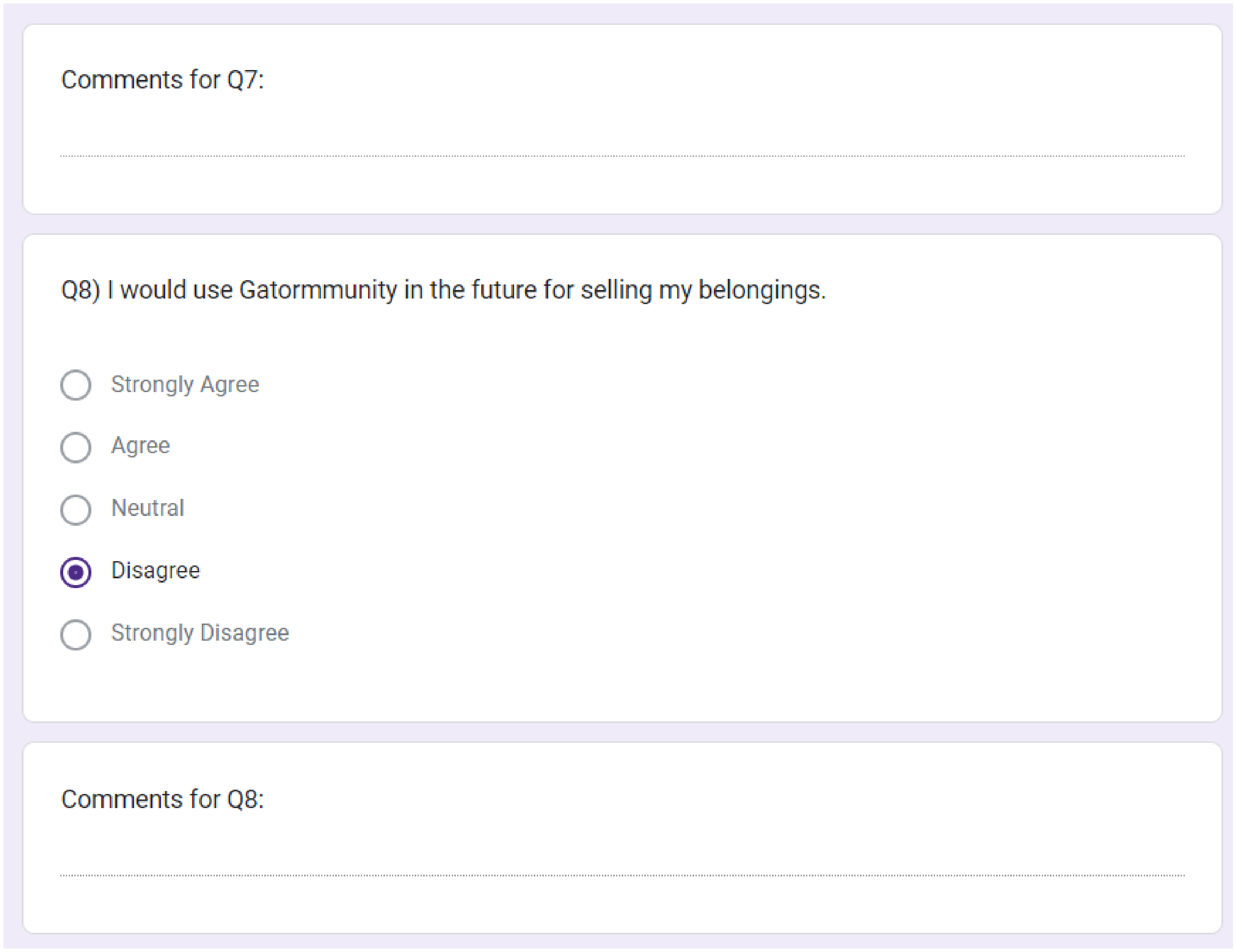
Q6) I would use the forums in the future to interact with others on Gatormmunity.

- Strongly Agree
- Agree
- [✓ Neutral]
- Disagree
- Strongly Disagree

Comments for Q6:

Q7) It was easy and intuitive to sell an item.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

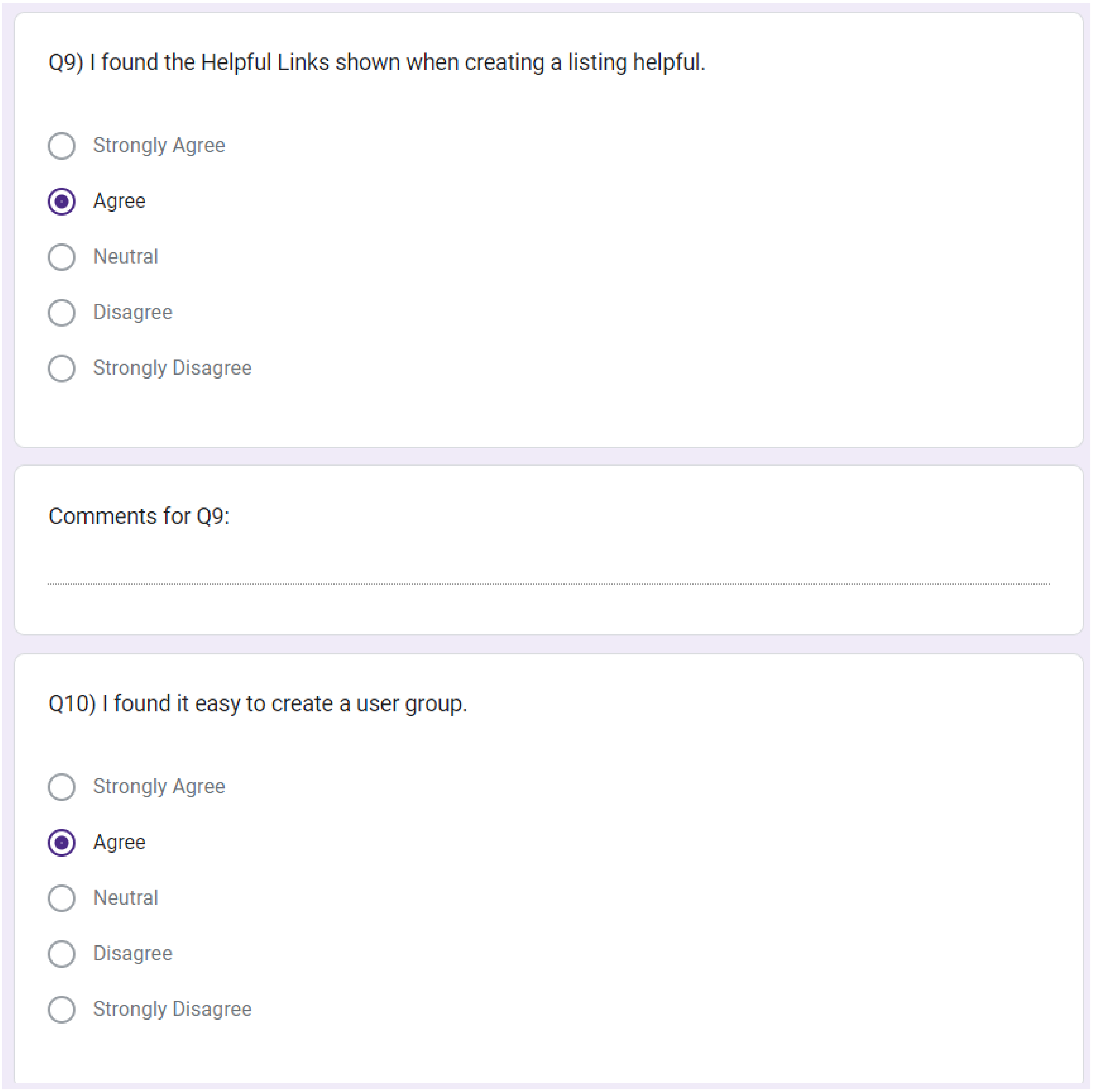


Comments for Q7:

Q8) I would use Gatormmunity in the future for selling my belongings.

Strongly Agree
Agree
Neutral
[✓ Disagree]
Strongly Disagree

Comments for Q8:



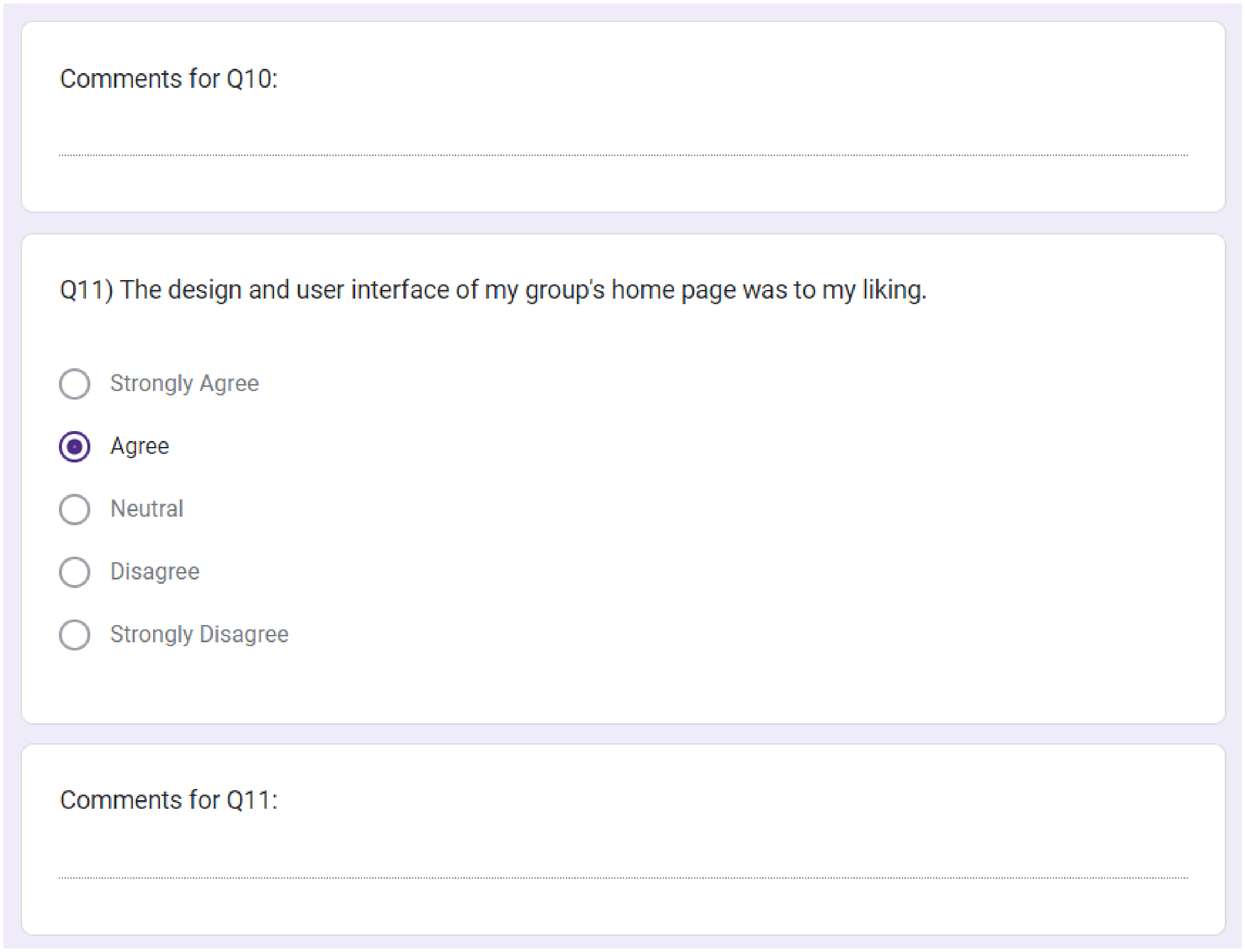
Q9) I found the Helpful Links shown when creating a listing helpful.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

Comments for Q9:

Q10) I found it easy to create a user group.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

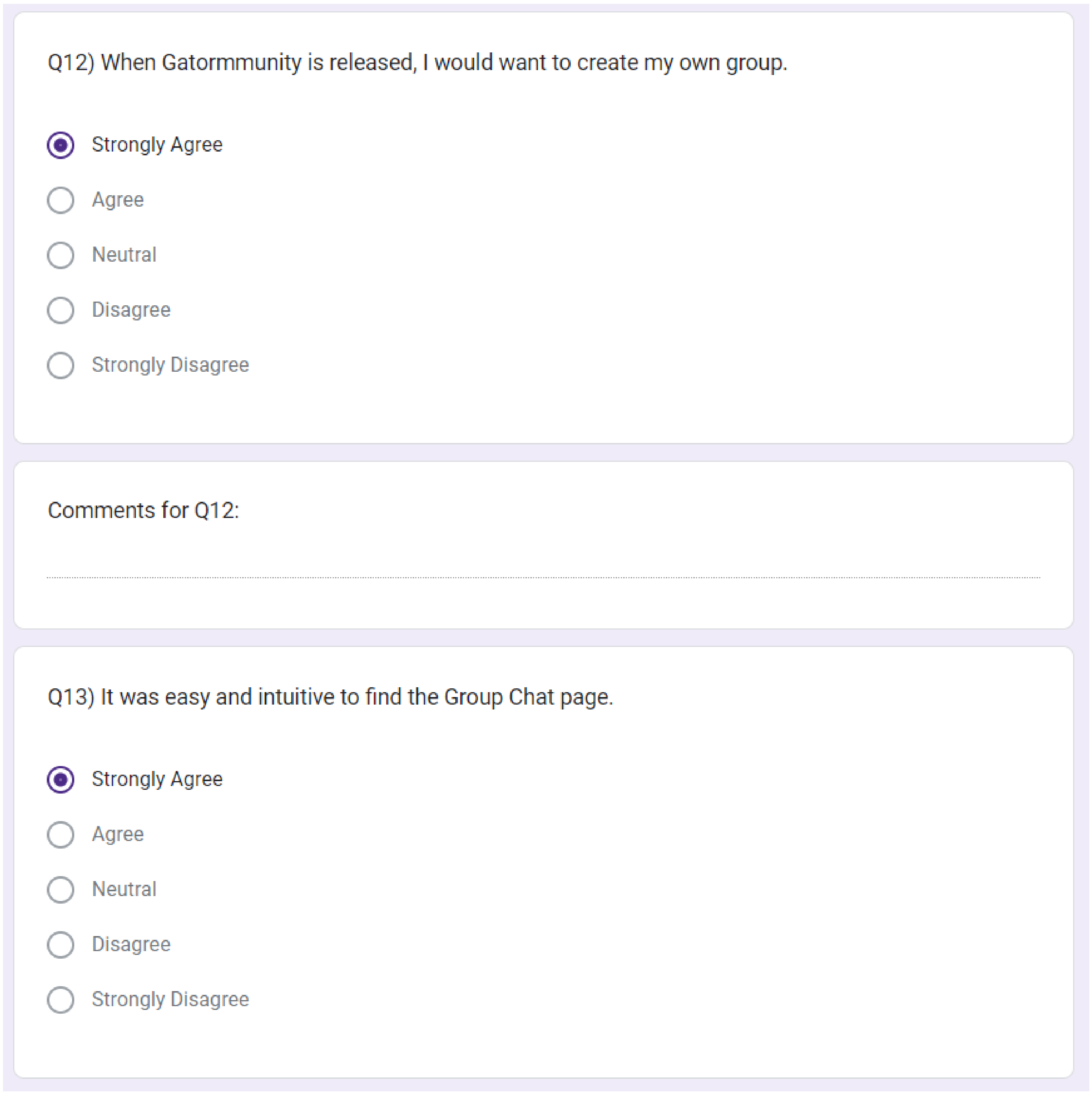


Comments for Q10:

Q11) The design and user interface of my group's home page was to my liking.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

Comments for Q11:



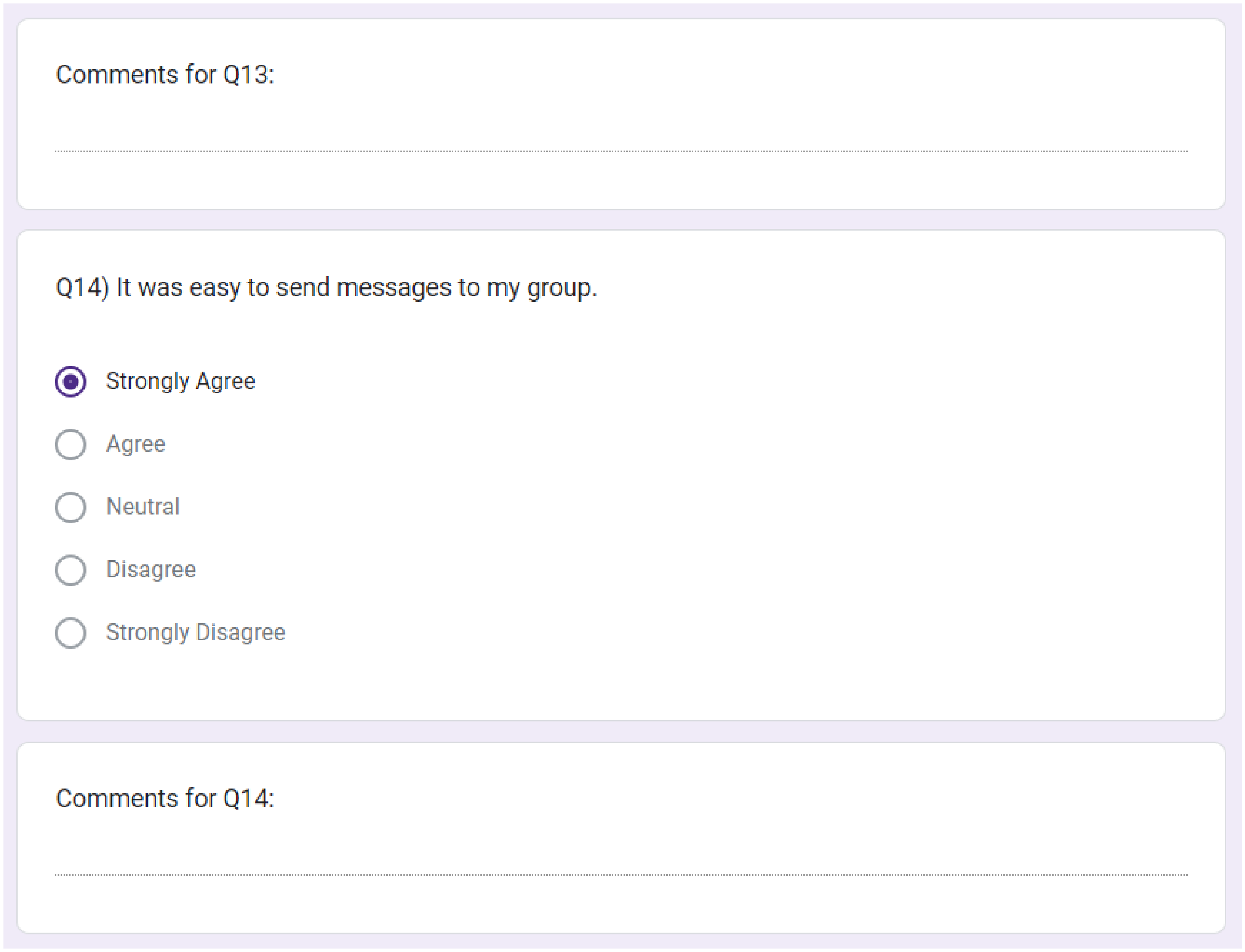
Q12) When Gatormmunity is released, I would want to create my own group.

[✓ Strongly Agree]
Agree
Neutral
Disagree
Strongly Disagree

Comments for Q12:

Q13) It was easy and intuitive to find the Group Chat page.

[✓ Strongly Agree]
Agree
Neutral
Disagree
Strongly Disagree

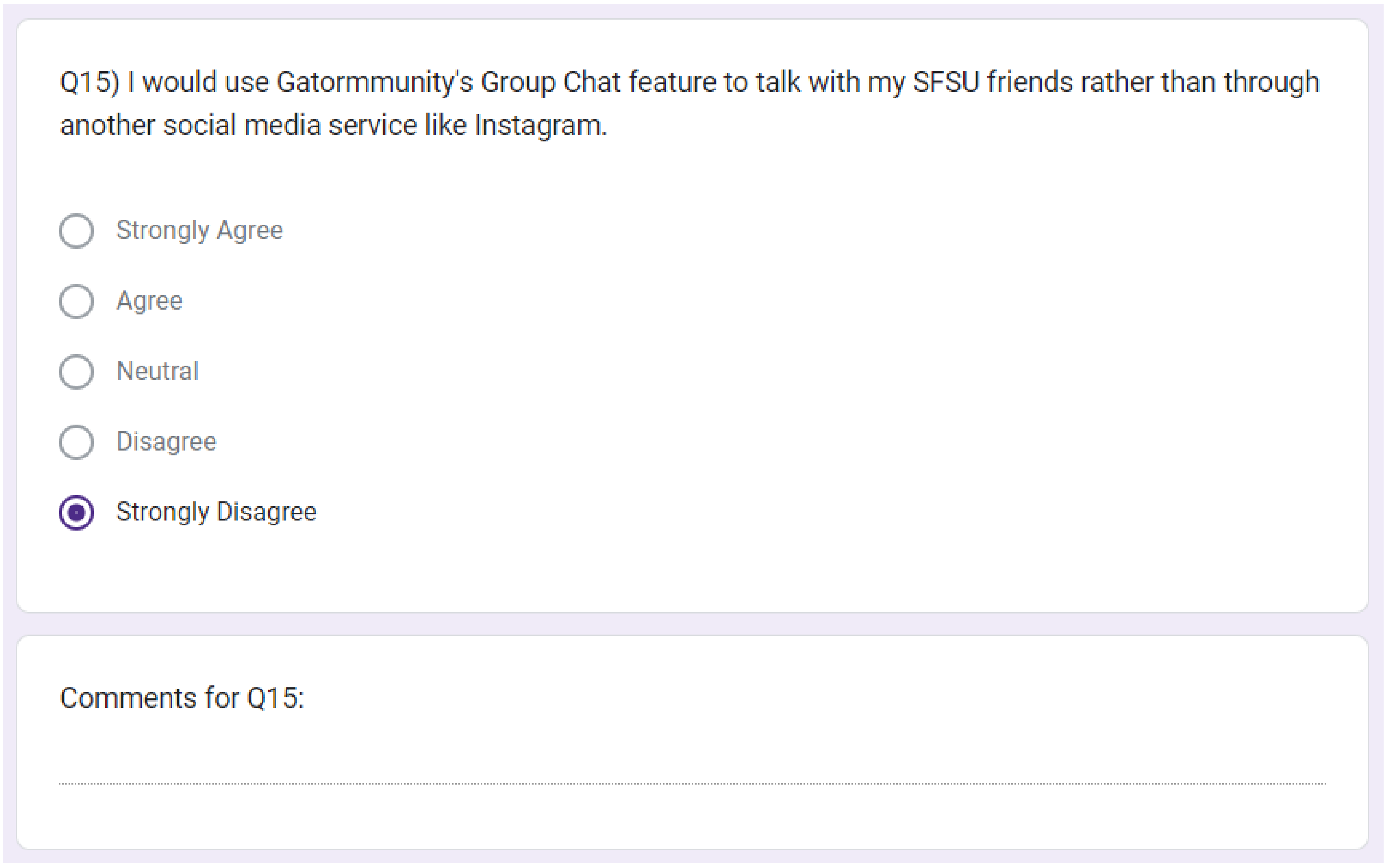


Comments for Q13:

Q14) It was easy to send messages to my group.

[✓ Strongly Agree]
Agree
Neutral
Disagree
Strongly Disagree

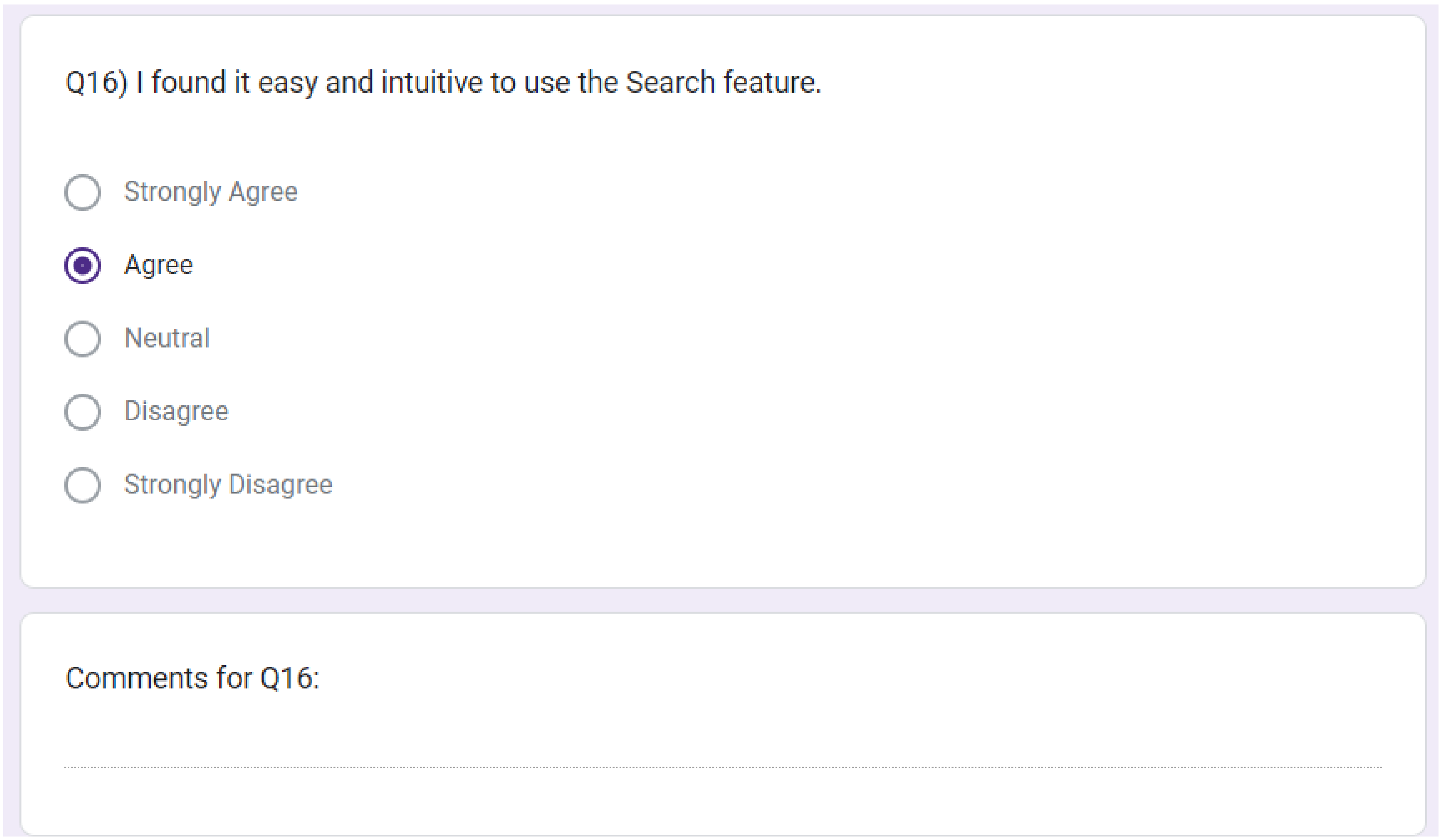
Comments for Q14:



Q15) I would use Gatormmunity's Group Chat feature to talk with my SFSU friends rather than through another social media service like Instagram.

- Strongly Agree
- Agree
- Neutral
- Disagree
- [✓ Strongly Disagree]

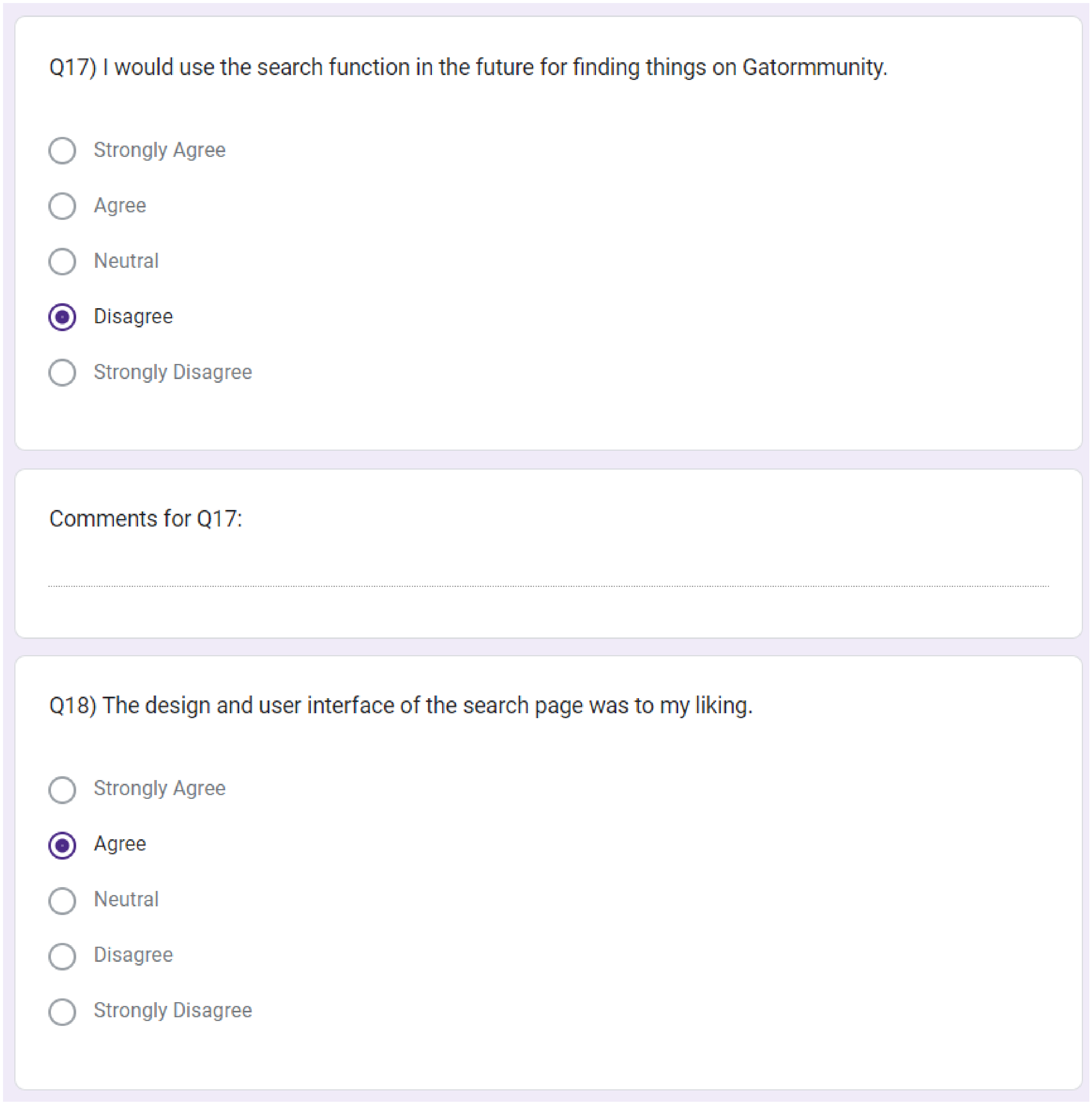
Comments for Q15:



Q16) I found it easy and intuitive to use the Search feature.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

Comments for Q16:



Q17) I would use the search function in the future for finding things on Gatormmunity.

- Strongly Agree
- Agree
- Neutral
- [✓ Disagree]
- Strongly Disagree

Comments for Q17:

Q18) The design and user interface of the search page was to my liking.

- Strongly Agree
- [✓ Agree]
- Neutral
- Disagree
- Strongly Disagree

Comments for Q18:

3) QA Test Plan

The five non-functional requirements to test:

3.3. Passwords shall be hashed before being stored in the database. In other words, there shall be no plaintext passwords stored in the database.
6.3. Approved users shall only upload files less than or equal to 5 MB in size.
7.2. The application shall use the personal data of its users for verifying their identity.
7.3. The application shall store and use its users' credentials for logging them in.
8.2. The application shall support the Google Chrome 105, Microsoft Edge 105, and Safari 15.5 web browsers.

Test #1: Password Hashing

3.3. Passwords shall be hashed before being stored in the database. In other words, there shall be no plaintext passwords stored in the database.

Test Objectives:
We are testing if the passwords are being hashed before being inserted into the database. The only instance where we insert a password into the database is when someone registers for an account, so we will be registering for accounts with passwords of various lengths and key combinations. In all successful registration cases, we are expecting our account table to have a new entry in it with the hashed password.

HW and SW Setup:
We will test on a computer or laptop using a supported operating system and a supported web browser. We will have MySQL Workbench open so that we can check what we just inserted into the database. The URL of the page to do the testing is that of the registration page:
http://54.241.101.69/register

Feature to be Tested:
We are testing the database hashing code on the back end. Since we are registering for an account, this test also tests our registration feature's front end and back end.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Alphabetical Password	Tests password hashing with a password with only alphabetical characters.	A new user with the password: testing	The account table should have a new row containing the new user's hashed password.	Pass
2	Alphanumeric Password	Tests password hashing with a password with alphabetical and numeric characters.	A new user with the password: 123testing123	The account table should have a new row containing the new user's hashed password.	Pass
3	Complex Password	Tests password hashing with a password with alphabetical, numeric, and special characters.	A new user with the password: ?123!testing&123#	The account table should have a new row containing the new user's hashed password.	Pass

Test #2: File Input Validation

6.3. Approved users shall only upload files less than or equal to 5 MB in size.

Test Objectives:
We will be testing if our file input validation works. Our file input validation is supposed to only allow users to upload files that are at most 5 MB in size and the file must be an image of type JPEG, PNG, WebP, GIF, or AVIF. We reuse our file input validation function since our application mandates the same file size and type requirements for all forms that allow file input, therefore we can choose any form that allows for file input to test.

HW and SW Setup:
We will test on a computer or laptop using a supported operating system and a supported web browser. We will have MySQL Workbench open so that we can get the filename of the file we just inserted into the database. We will have the marketplace page open so that we can see if the file and listing was successfully uploaded.
The URL of the page to do the testing is that of the create listing page:
http://54.241.101.69/create-listing

Feature to be Tested:
We are testing the file input validation on the Create Listing page. We will be uploading files of various sizes and types, and verifying that our file input validation meets the non-functional requirement.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Upload Valid Image File	Create a listing with an image that is of an accepted file type and file size.	A new listing with an image that shows a picture of tea. The file type is JPEG, and the file size is 148 KB.	The listing should be successfully created, and we should be able to see the file in the public/listing_photos folder.	Pass
2	Upload .txt File	Create a listing with a file that is not of an accepted file type (.txt) but with a valid file size.	A new listing with a text file with this text in it: “Hello”. The file type is TXT, and the file size is 5 bytes.	The listing should NOT be created, and the user should see an alert message saying the file is of an invalid type.	Pass
3	Upload Invalid Image File	Create a listing with an image that is of an accepted file type but a file size that is too large.	A new listing with an image that shows a picture of a fridge. The file type is PNG, and the file size is 9.30 MB.	The listing should NOT be created, and the user should see an alert message saying the file cannot exceed 5 MB.	Pass

<u>Test #3: Checking an Unapproved User’s personal data to verify their identity</u>

7.2. The application shall use the personal data of its users for verifying their identity.

Test Objectives:
We will be testing if the user data that is provided at registration is being stored in the database and that this data will be visible to moderators and administrators for the purpose of verifying the new user’s identity. In particular, the mods/admins should be able to see an unapproved user’s user ID, full name, email, SFSU ID number, SFSU ID picture, and registration date.

HW and SW Setup:
We will test on a computer or laptop using a supported operating system and a supported web browser. We will have the administration page open so that we can see the approval requests from unapproved users. In the approval requests tab is where mods/admins can verify users' identities and make unapproved users into approved users.
The URL of the page to do the testing is that of the administration page, which is only accessible to users with a moderator or administrator account: http://54.241.101.69/admin
To register for the accounts, the URL is that of the registration page: http://54.241.101.69/register

Feature to be Tested:
We are testing the approval requests section of the Administration page. We will be registering for accounts with different SFSU ID pictures, names, SFSU ID numbers, and so on, and we will be verifying that we are able to see this information in the approval requests section of the Administration page.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Unapproved User with Legitimate Credentials	Register for an account with credentials that look like a real SFSU student's account. The image should actually be an SFSU ID photo.	A new account with a real name, SFSU email, SFSU ID number, password, and genuine SFSU ID picture.	The new user's personal data (except the password) should be visible in the Approval Requests section of the Administration page.	Pass
2	Unapproved User with Fake Credentials	Register for an account with credentials that are clearly fake. The image should NOT be an SFSU ID photo.	A new account with a name composed of random characters, an illegitimate SFSU email, a random string of numbers for the SFSU ID number, password, and a picture that does not show an SFSU ID.	The new user's personal data (except the password) should be visible in the Approval Requests section of the Administration page.	Pass
3	Unapproved User with Legitimate Credentials but Wrong	Register for an account with credentials that look like a real SFSU student's account. The image should NOT be an SFSU ID photo.	A new account with a real name, SFSU email, SFSU ID number, password, but not an SFSU ID	The new user's personal data (except the password) should be visible in the	Pass

	Image		picture.	Approval Requests section of the Administration page.	

Test #4: Log In

7.3. The application shall store and use its users' credentials for logging them in.

Test Objectives:
We will be testing if a user's SFSU ID number and hashed password are correctly inserted into the database, and that a user is able to log in (after being approved). By extension, we are testing that the registration and approval requests work correctly. We will check the user and account table to verify that the user was successfully created.

HW and SW Setup:
We will test on a computer or laptop using a supported operating system and a supported web browser. We will have the administration page open so that we can see the approval requests from the newly created unapproved users. We will have the registration page open on another tab in order to create new users. In this same tab, we will log in after being approved.
We will have MySQL Workbench open to check the user and account table to verify successful insertion into the database.
The URL of the page we are testing is that of the login page, which is accessible only if you are not logged in: http://54.241.101.69/login
The URL of the page to approve unapproved users is that of the administration page, which is only accessible to users with a moderator or administrator account: http://54.241.101.69/admin
To register for the accounts, the URL is that of the registration page: http://54.241.101.69/register

Feature to be Tested:
We are testing the login feature. Since a new user has to be approved after registering before they can log in, we are testing the registration and approval features as well.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Register and Login After Being Approved	Register for an account and get approved by a moderator or administrator. We will check the database tables to see if the correct credentials were inserted. Then, we will	Register for a new account with the SFSU ID number: 913911011 and the password: CSC680. The new	The new user's SFSU ID number should be present in the \`user\` table, a hashed password should be present in	Pass

		log in after being approved.	user should be approved.	the `account` table with the new user's user_id, and the user should be able to login with the provided test input credentials.	
2	Attempt to Login with an Unapproved Account	Try to login with an account that has not yet been approved. The credentials should be correct.	Register for a new account with the SFSU ID number: 812310552 and the password: CSC317. The new user will not be approved.	The new user's SFSU ID number should be present in the `user` table, a hashed password should be present in the `account` table with the new user's user_id, and the user should NOT be able to login with the provided test input credentials.	Pass
3	Attempt to Login with Incorrect Credentials	Try to login with invalid credentials.	Login with the SFSU ID Number: 413413413 and the password: hello	The user should see an alert message telling them their credentials are incorrect.	Pass

<u>Test #5: Web Browser Support</u>

8.2. The application shall support the Google Chrome 105, Microsoft Edge 105, and Safari 15.5 web browsers.

Test Objectives:
We will be testing if our application runs fine and looks the same on the three web browsers in the non-functional requirement. We will check the Forums page, the Marketplace page, and the Chat page and see if there are any visual glitches or inconsistencies.

HW and SW Setup:
We will test on a computer or laptop using a supported operating system. We will test on each of the three supported browsers. For this test, we used two computers: a Windows 10 computer for Google Chrome 105 and Microsoft Edge 105 testing, and a Mac for Safari 15.5.
The first URL to test is the forums page: http://54.241.101.69/forums
The second URL is the marketplace page: http://54.241.101.69/marketplace
The third URL is the chat page: http://54.241.101.69/chat

All three URLs require the user to be logged in.

Feature to be Tested:
We will be testing the Forums page, the Marketplace page, and the Chat page, and perhaps some other random pages and buttons. The features we are testing are the appearance of the pages, their speed, and whether the pages perform as expected or not.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Test on Google Chrome 105	Use the application with Google Chrome 105 and see if everything performs as expected.	Use Google Chrome 105 and go to the Forums, Marketplace, and Chat page. Press buttons and click on links.	The pages should look the same as it does on every other supported browser, and the functions should be the same too. Nothing should be broken.	Pass
2	Test on Microsoft Edge 105	Use the application with Microsoft Edge 105 and see if everything performs as expected.	Use Microsoft Edge 105 and go to the Forums, Marketplace, and Chat page. Press buttons and click on links.	The pages should look the same as it does on every other supported browser, and the functions should be the same too. Nothing should be broken.	Pass
3	Test on Safari 15.5	Use the application with Safari 15.5 and see if everything performs as expected.	Use Safari 15.5 and go to the Forums, Marketplace, and Chat page. Press buttons and click on links.	The pages should look the same as it does on every other supported browser, and the functions should be the same too. Nothing should be broken.	Pass

b) We have performed the tests and verified that everything works as expected. All tests have passed, and the results of our application match up with the Expected Correct Output. We used various browsers and everything appears to function as expected.

4) Code Review

a)
The coding style we chose has: a maximum of 130 characters per line (with a few exceptions, such as with npm scripts), the use of semicolons in JavaScript, and our code is indented with tabs with a tab size of 4 spaces.

We try to reduce the number of lines in our code by using single-line if statements if it does not impact readability too much, and we use spaces in our code to make it more readable, e.g. we do `{ title, description, price }` rather than `{title,description,price}`.

Our JavaScript variables are written in camelCase, while the attributes in our database are in snake_case. Most of our functions should have a JSDoc comment that explains what the function does, written above the function's signature. They are formatted as follows: /** comment */
Most of our files should have a header comment that explains what the file is for.

b)
1. Anthony Zhang submitted code to the members in his team (Team 7) for peer review. It is a 2 page snippet of the Create Listing page's front end code. It includes the React states and the fetch() function used to send the request to the back end.

2. Anthony Zhang submitted code to Team 5's Team Lead, Zhenyu Lin, and to Team 6's Team Lead, Alex Sanchez, for peer review. It is a 2 page snippet of the Create Thread page's back end code. In particular, it is the controller function which handles the back end logic, and one of the model functions which handles the database querying.

We peer reviewed Team 5's code since they peer reviewed our code.
We received Team 6's peer review and their own code on 12/1, so we did not have too much time to discuss the feedback they left us and peer review their code.

3. **Code submitted to the team for review in Section 1:**
We chose this code because it shows how we use React states, how we document our code with JSDoc and in-line comments, how the front end uses fetch() to communicate with the back end, and how we handle the data that is returned by the back end. The Create X pages we have (Create Thread, Create Listing, Create Group) are all very similar code-wise, so the peer review we do for the Create Listing page will benefit all of our Create X pages.

```
/* This file handles the display of the Create Listing page for the Gatormmunity Marketplace.
 * The page takes user input for creating the listing, which is sent to the back end for validation
and listing creation. */

import { useContext, useEffect, useRef, useState } from "react";
import { Button, Col, Form, Container, Row } from "react-bootstrap";
import { UserContext } from '../App.js';
import { Navigate, useNavigate } from 'react-router-dom';
import HelpfulLinksList from "../components/help/HelpfulLinksList";
import * as questions from '../components/help/Questions';
import { ERROR_STATUS, LISTING_CATEGORIES, SUCCESS_STATUS } from "../components/Constants.js";

export default function CreateListing() {
    const userSession = useContext(UserContext); // the user's session data
    const navigate = useNavigate();
    const itemPhotoInput = useRef();

    /** The links that will be shown in the helpful links list. */
    const links = [questions.meetingBuyerSeller, questions.reportUser];

    // contains the form's data
    const [form, setForm] = useState({
        listingTitle: "",
        listingDescription: "",
        price: "",
        category: LISTING_CATEGORIES[0] // the default category
    });

    const [itemPhoto, setItemPhoto] = useState(null); // stores the photo of the item
    const [returnData, setReturnData] = useState(null); // stores the data sent back from the back
end

    /** Updates the create listing form's state. */
    function updateForm(value) {
        return setForm((prev) => {
            return { ...prev, ...value };
        });
    }

    /**
     * Sends the form to the back end to create the listing.
     *
     * The front end sends:
     * listingTitle: {string} The listing's title. Required; must be 1-255 characters.
     * listingDescription: {string} The listing's description. Required; must be 1-2'500 characters.
     * price: {string} The item's price. Required; must be in a valid currency format, e.g. 21.55 or
21; must be non-negative.
     * category: {string} The item's category. Required; must be one of the predefined category
options.
     * sellerId: {int} The seller's user id. Required; must be a positive integer.
     * itemPhoto: {File} The item's photo. Required; must be JPEG, PNG, WebP, GIF, or AVIF; cannot
exceed 5 MB.
     */
    async function createListing(e) {
        e.preventDefault();

        /** The form that we send to the backend containing the listing's data. */
        const formData = new FormData();

        // append the form's content to `formData`
        for (const index in form) formData.append(index, form[index]);
        formData.append("sellerId", userSession.user_id);
        formData.append("itemPhoto", itemPhoto);

        await fetch("/api/listings/create-listing", {
            method: "POST",
            body: formData
        })
            .then((res) => res.json())
            .then((data) => setReturnData(data))
            .catch(console.log());
    }

    /* Redirects to the newly created listing's page upon successful listing creation. Otherwise,
show an error. */
    useEffect(() => {
        if (returnData?.status === SUCCESS_STATUS) navigate(`/listing/${returnData.listingId}`);
        else if (returnData?.status === ERROR_STATUS) alert(returnData.message);
    }, [returnData]); // eslint-disable-line react-hooks/exhaustive-deps
```

Code submitted to Team 5 and 6 for review in Section 2:
We chose this code from our Create Thread page's back end because its structure is similar to the rest of our back end code. It shows the JSDoc comment describing what the back end sends to the front end, how we initialize our variables from what the front end sends with req.body, and the .then() promise chain which has us perform multiple database queries to insert the thread, post, and attachment into the database, if an image was provided. Also, we show our error-handling code.

```
/**
 * Creates a thread either for Gatormmunity or for a group, depending on what groupId's value is.
 * The thread's original post and optionally, its attachment, is created too.
 *
 * The back end sends:
 * On successful thread creation:
 *     status: {string} "success",
 *     threadId: {int} The id of the new thread.
 *
 * On failure:
 *     status: {string} "error",
 *     message: {string} The error message to display the user.
 */
exports.createThread = (req, res) => {
    let returnData = {}; // holds the data that will be sent to the front end
    const { threadTitle, threadBody, category, groupId, creatorId } = req.body;

    /* The thread image is optional, thus we check if `req.file` exists first before assigning these
variables a value. */
    const threadImagePath = req.file ? req.file.path : null;
    const threadImageThumbnailPath = req.file ? path.join(req.file.destination,
`tn-${req.file.filename}`) : null;
    const threadImageOriginalFilename = req.file ? req.file.originalname : null;

    createThumbnail(req.file, threadImagePath, threadImageThumbnailPath, 250, 250)
        .then(() => { // if the thumbnail was successfully created, create the thread
            return ThreadModel.createThread(threadTitle, category, groupId, creatorId);
        })
        .then((threadId) => {
            if (threadId < 0) throw new Error("Error with createThread().");

            returnData.threadId = threadId;
            return PostModel.createPost(threadBody, true, threadId, creatorId); // create the
thread's original post
        })
        .then((postId) => {
            if (postId < 0) throw new Error("Error with createPost().");

            // If a thread image was provided, attach it to the post, otherwise, we are done and can
send back `returnData`.
            if (threadImagePath) {
                return PostModel.createAttachment(threadImageOriginalFilename, threadImagePath,
                                                  threadImageThumbnailPath, postId);
            } else {
                returnData.status = SUCCESS_STATUS;
                res.json(returnData); // send the success to the front end

                return Promise.reject("SKIP"); // break out of the .then() promise chain early
            }
        })
        .then(attachmentId => {
            if (attachmentId < 0) throw new Error("Error with createAttachment().");

            returnData.status = SUCCESS_STATUS;
            res.json(returnData); // send the success to the front end
        })
        .catch((err) => {
            if (err === "SKIP") return; // if returnData has already been sent, do nothing

            returnData = { status: ERROR_STATUS };
            returnData.message = "An error occurred while creating your thread.";

            // delete the uploaded files on failed thread creation, if they exist
            if (threadImagePath) {
                fs.unlink(threadImagePath, () => { });
                fs.unlink(threadImageThumbnailPath, () => { });
            }

            console.log(err);
            res.json(returnData); // tell the front end of the failure
        })
};
```

Code showing how we query the MySQL database using JavaScript in PostModel.createPost():

```
/**
 * Inserts a new post into the database.
 *
 * @param {string} body The body of the thread.
 * @param {boolean} is_original_post Whether the post is the original post, i.e. the post created
when the thread is created.
 * @param {int} thread_id The id of the thread the post belongs to.
 * @param {int} author_id The id of the user who created the post.
 * @returns On success, the new post's post_id. On failure, -1.
 */
PostModel.createPost = (body, is_original_post, thread_id, author_id) => {
    const insertSQL = `INSERT INTO post (body, is_original_post, thread_id, author_id)
                       VALUES (?, ?, ?, ?);`;

    return database
        .query(insertSQL, [body, is_original_post, thread_id, author_id])
        .then(([results]) => {
            if (results.affectedRows) return results.insertId;
            else return -1;
        })
        .catch((err) => Promise.reject(err));
};
```

Discussion about the Review for Section 1:

The code we sent to our own team in Section 1 received this feedback:
We do not have comments indicating where a function starts and ends. A client who does not know much about coding will be confused by our code. We should use React Class components rather than React Functional components. Our code and comments should be written such that non-computer scientists will be able to understand it.

In our discussion about the review, we thought that our files are becoming rather long and we should try to split up the larger files into smaller files so that it is easier to navigate through the code. This is especially true for the code we have in our back end.
We could benefit from having more in-line comments and more header comments that describe the purpose of the file, and some of our functions are missing JSDoc comments that describe the function, its parameters, and its return values.

Discussion about the Review for Section 2:

The code we sent to Team 5 in Section 2 received this feedback:
The reviewer was not sure what an original post was; we should elaborate on what it is in a comment. It is not clear what an attachment is: is it any kind of file, a text file, an image file? We should state what it is more explicitly. Someone unfamiliar with our website might think ‘thread’ refers to process threads, rather than a forum thread.

In our discussion about the review, we felt that the feedback was fair. While our milestone document clarifies what an original post is, it might help to have it in a code comment too for easy reference. The same goes for attachments: the milestone documents say it is an image, but the code that we sent to Team 5 did not explicitly say as such.
We coined the term “attachment” back in the early stages of development when we would allow users to upload any type of file. Now, we only allow images for the sake of simplicity, but we kept the term “attachment” since it was what we were used to. We might have to change the name of the table and the code to make it more clear to any new developers who would work on the codebase.

Team 6 left us this feedback about our code in Section 2:
We should have a header comment for our file with threadController in it. We should not display the original filename of the thread’s image because it could be inappropriate and get us in trouble.

In our discussion about the review, we felt that the feedback was fair. We should spend more time writing comments in our code, and that is what we will be focusing on in the coming days. All P1s are implemented, so all that is left to do is to make small improvements. We have moderators and admins who can delete threads and ban any user who tries to post inappropriate things, but if we feel that the potential for abuse is too high, we will hide the original filename.

5) Self-Check on Best Practices for Security

Major Assets being Protected:

- Passwords are hashed before being stored in the database. Thus, nobody can know the password of another user, not even the developers who have access to the database.

- Only approved users (users who were manually approved by a moderator after registering) can login and use Gatormmunity. Unapproved users and people not logged in cannot view most of the pages on Gatormmunity, including the Forums, Marketplace, Groups, Chat, and Inbox page. However, everyone can access the Terms of Service and Contact Us page, for example.

- Only moderators and administrators can access the Administration page. On the Administration page, moderators can approve/reject unapproved users and ban approved users. Administrators can do that, as well as appoint and unappoint moderators.

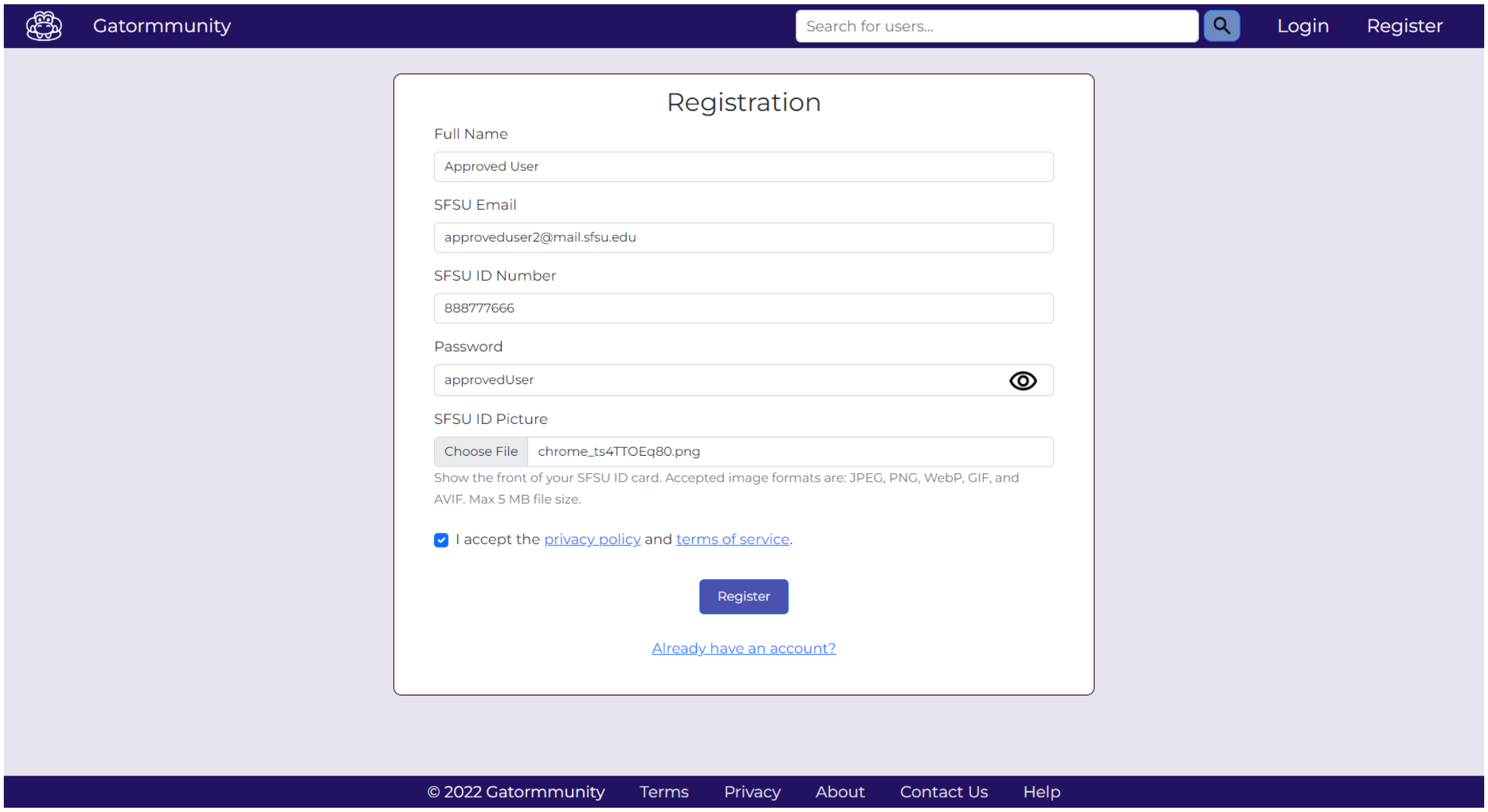
- Only moderators and administrators can view the SFSU ID number and SFSU ID card picture of Gatormmunity's users. This is necessary because moderators need to be able to see these credentials in order to approve/reject unapproved users.

- Only a group's members can see the threads and posts in that group's forums as well as send and receive messages in that group's chat.

Password Encryption Process:
Passwords are hashed before being stored in the database. When a user registers for an account, we take their plaintext password and use the bcrypt npm library's hash function to hash that plaintext password with 10 rounds. That hashed password is then inserted into the account table in our database. We store the user's password in a separate table from the rest of the user's data.

We show real examples (screenshots) below.

First, we register for a new account with these credentials. We can show/hide the password by pressing the eye icon in the password field.



Registration

Full Name: Approved User

SFSU Email: approveduser2@mail.sfsu.edu

SFSU ID Number: 888777666

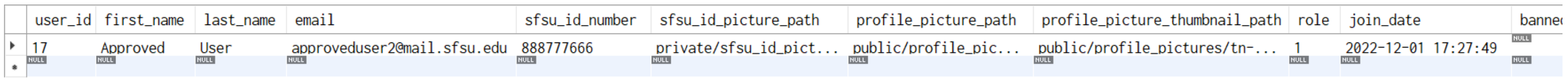
Password: approvedUser

SFSU ID Picture: Choose File chrome_ts4TTOEq80.png

Show the front of your SFSU ID card. Accepted image formats are: JPEG, PNG, WebP, GIF, and AVIF. Max 5 MB file size.

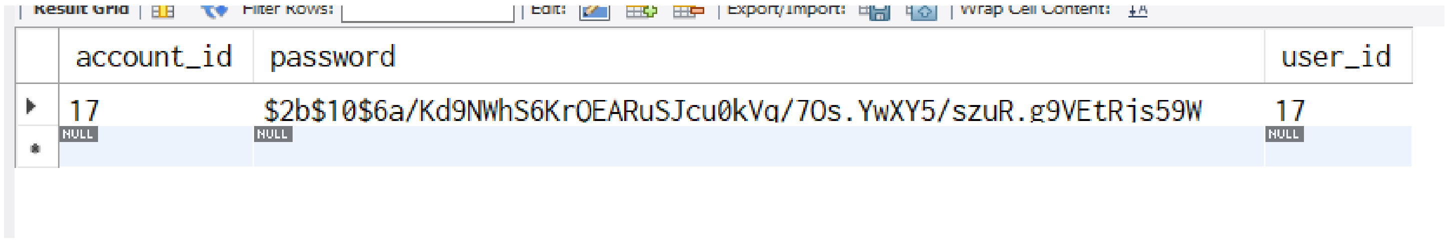
✓ I accept the privacy policy and terms of service.

Here is the newly created user's entry in the 'user' table of our database. We do not store the password in the 'user' table. As you can see, they have a user_id of 17.



user_id	first_name	last_name	email	sfsu_id_number	sfsu_id_picture_path	profile_picture_path	profile_picture_thumbnail_path	role	join_date	banne
17	Approved	User	approveduser2@mail.sfsu.edu	888777666	private/sfsu_id_pict...	public/profile_pic...	public/profile_pictures/tn-...	1	2022-12-01 17:27:49	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Here is the newly created user's entry in the 'account' table of our database, which stores the user's password. The user_id foreign key is 17, the same as the user_id from the 'user' table. As you can see, the password is hashed and is NOT stored in plaintext.



account_id	password	user_id
17	$2b$10$6a/Kd9NWhS6KrOEARuSJcu0kVq/7Os.YwXY5/szuR.g9VEtRjs59W	17
NULL	NULL	NULL

Input Data Validation:

We validate input on both the client and server-side. The client-side performs basic validation such as requiring required fields, only allowing certain types of input (e.g. numbers), and checking the email format. The server-side performs thorough validation.

We validate:

- **Images:** All images must be either JPEG, PNG, WebP, GIF, or AVIF file types, and they must be within 5 MB.

- **Direct Messages:** The message body must be between 1-5000 characters, and the provided conversation_id and sender_id must exist, be positive, and be integers.

- **Groups:** An image must be uploaded. The group's name must be between 1-255 characters, the group's description must be at most 5000 characters, the provided admin_id must exist, be positive, and be an integer.

- **Chat Messages:** The message body must be between 1-5000 characters, and the provided group_id and sender_id must exist and be integers. sender_id must be positive.

- **Group Announcements:** The announcement must be at most 5000 characters, and the provided group_id and user_id must exist, be positive, and and be integers.

- **Listings:** An image must be uploaded. The listing title must be 1-255 characters, the description 1-2500 characters, the price must be a decimal with at most 2 decimal places, the price must be from 0.00 to 99'999.99, the category must be one of those in the dropdown, and the seller_id must exist, be an integer, and be positive.

- **Forum Threads:** The thread title must be between 1-255 characters, the body must be between 1-10'000 characters, the category must be one of those in the dropdown, and the creator_id must exist, be an integer, and be positive.

- **Forum Posts:** The post body must be between 1-10'000 characters, the thread_id and author_id must exist, be an integer, and be positive.

- **Registration:** An SFSU ID picture must be uploaded. The full name must be between 1-100 characters, and a first and last name must be provided. The email must have an @ symbol, must end with "sfsu.edu", and must be between 1-255 characters. The SFSU ID number must be 9 digits long, be an integer, and be positive. The password must be 6-64 characters long.

- **Listing Search Input:** The category filter must be one of those in the dropdown, and the max price filter must be a decimal with at most 2 decimal places.

- **Thread Search Input:** The category filter must be one of those in the dropdown.

- **User Search Input:** The role filter must be one of those in the dropdown.

The code that we use to validate user input is stored in the directory: ‘server/middleware’. We use the yup npm library to validate the forms, while we use regular JavaScript to validate the image file type and file size.

6) Self-Check: Adherence to Original Non-Functional Specs

1. System Requirements
    1.1. The AWS server's region shall be North California, United States.
        DONE
    1.2. The database shall be stored on the AWS server.
        DONE

2. Performance Requirements
    2.1. The application should be available for at least 23 hours a day.
        DONE
    2.2. The application's homepage should load in at least 7 seconds.
        DONE
    2.3. The application should be accessible from anywhere in the world.
        DONE

3. Storage, Security, and Environmental Requirements
    3.1. The server's storage shall be a 30 GiB volume.
        DONE
    3.2. The database shall be secured with a password.
        DONE
    3.3. Passwords shall be hashed before being stored in the database. In other words, there shall be no plaintext passwords stored in the database.
        DONE

4. Usability Requirements
    4.1. Guest users shall accept the privacy policy and terms of service before they can register for an account.
        DONE
    4.2. Approved users should abide by the terms of service.
        DONE

5. Marketing and Legal Requirements
    5.1. The application shall be known as Gatormmunity in its marketing.
        DONE
    5.2. The application shall be known as Gatormmunity in its licensing.
        DONE
    5.3. The application shall have a logo.
        DONE
    5.4. The application shall have a privacy policy.
        DONE
    5.5. The application shall have terms of service.
        DONE

6. Content Requirements
    6.1. The application shall support the English language.
        DONE
    6.2. The application shall support US Dollars as currency.
        DONE
    6.3. Approved users shall only upload files less than or equal to 5 MB in size.
        DONE

7. Privacy Requirements
    7.1. The application shall collect and store personal data from its users.
        DONE
    7.2. The application shall use the personal data of its users for verifying their identity.
        DONE
    7.3. The application shall store and use its users' credentials for logging them in.
        DONE
    7.4. The application shall not reveal any of a user's private data to any other user.
        DONE

8. Compatibility Requirements
    8.1. The application shall support the Windows 10, Windows 11, and macOS Monterey 12.5 and 12.6 operating systems.
        DONE
    8.2. The application shall support the Google Chrome 105, Microsoft Edge 105, and Safari 15.5 web browsers.
        DONE
    8.3. The application should not be designed for mobile devices.
        DONE
    8.4. The application should be designed for and work on laptops.
        DONE
    8.5. The application should be designed for and work on desktop computers.
        DONE

9. Organizational Requirements
    9.1. The AWS server should not incur any costs to the developers.
        DONE
    9.2. The developers should use semicolons in their JavaScript code.
        DONE
    9.3. The developers should use tabs in their code, and not spaces.
        DONE
    9.4. The developers should not have lines of code that exceed 130 characters in length.
        DONE

9.5. The developers should document their code with comments.
DONE

9.6. The application's code should not be messy.
DONE

9.7. The application shall use working and tested code. In other words, faulty and untested code shall not be served from the remote server.
DONE

9.8. The master branch of the application's GitHub repository shall contain only working and tested code.
DONE

9.9. The documentation and technical reports shall have a table of contents.
DONE

9.10. The documentation and technical reports shall have page numbers.
DONE

9.11. The documentation and technical reports shall start each section on a new page.
DONE

7) List of Contributions to the Document and Contribution Scores

Anthony Zhang:

- Was an active participant in the meetings and team Discord server
- Sent the email to Ortiz for M4V1 Section 7
- Updated M3V2
- Wrote Sections 1, 2, 3, 4, 5, 7 of M4V1: Product Summary, Usability Test Plan, QA Test Plan, Code Review, Self-Check on Best Practices for Security, List of Contributions
- Helped peer review Team 5's and Team 6's code
- Helped test our live website by creating forum threads, creating forum posts, creating listings, sending chat messages, creating groups, joining groups, approving users, and got a tester to test our app and fill out the questionnaire
- Added Leave Group button for group members, and a Delete Group button for group admins
- Added Delete Thread, Delete Post, and Delete Listing buttons for moderators; sellers can delete their listings too
- Added show password eye button to the Login page
- Wrote the front end fetch functions for and worked on these pages: Administration, Chat, Create Thread, Dashboard, Gatormmunity Forums, Group Forums, Group Home, Group Members, Inbox, Join Group, Search, User Profile, View Thread
- Worked on the back end for the components: Change Profile Picture
- Wrote the back end code for the pages: Administration, Create Listing, Create Thread, Group Create Thread, Group Forums, Group Members, Inbox, Join Group, Login, Search, User Profile, View Thread

Marwan Alnounou:

- Was an active participant in the meetings and team Discord server
- Wrote Section 6 of M4V1: Adherence to Original Non-Functional Specs
- Sent the email to Ortiz for M4V1 Section 7
- Helped test our live website by sending chat messages
- Wrote some rules for Gatormmunity in the Terms of Service
- Added a Group Forums button for each of the groups in the Users Groups page
- Worked on the front end for the components: Group Change Announcement, Group Invite, Navbar
- Wrote the front end fetch functions for and worked on these pages: Group Members, Marketplace, View Listing

Mohamed Sharif:

- Was an active participant in the meetings and team Discord server
- Helped peer review our own code and Team 5's code
- Worked on the back end for the components: Group Invite
- Wrote the back end code for the pages: Group Home, User Profile, Users Groups, View Listing

Jose Lopez:

- Was an active participant in the meetings
- Sent the email to Ortiz for M4V1 Section 7
- Helped peer review Team 5's code
- Helped test our live website by creating forum threads, creating listings, sending chat messages, and banning users
- Helped make the styling more consistent for the Create pages (Create Thread, Create Group, etc) by having the pages share one CSS file
- Moved the login page's error messages from inside the page to an alert message box
- Wrote the front end fetch functions for and worked on these pages: Create Group, Create Listing, Group Create Thread, Group Home, Users Groups

Florian Cartozo:

- Sent the email to Ortiz for M4V1 Section 7
- Helped peer review Team 5's code
- Helped test our live website by creating forum posts, joining groups, sending chat messages, sending emails via the contact us page, and got a tester to test our app and fill out the questionnaire
- Created M4's EER Model and Database
- Added a message is currently sending indicator for the Contact Us page
- Worked on the back end for the components: Group Change Announcement
- Wrote the back end code for the pages: Chat, Create Group, Dashboard, Gatormmunity Forums, Marketplace

Contribution Scores:

Anthony Zhang: 10
Marwan Alnounou: 10
Mohamed Sharif: 8
Jose Lopez: 9
Florian Cartozo: 10