Team Member Full Name	NetID
Kaia Damian	**kdamian**
Samara Jacobo	**sjacobo3**
Swindar Zhou	**kzhou3**

Features Implemented

- Feature 1.1: Create New User Profile
- Feature 1.2: Log-in
- Feature 1.3: Log-out
- Feature 2.1: Create Listings
- Feature 2.2: Update Listings
- Feature 2.3: Delete Listings
- Feature 3.1: View All Listings
- Feature 3.2: Search for Products
- Feature 3.3: Seller-Buyer Messaging
- Feature 4.1: Buying Daily Listings

Persistent Storage Design

We are using SQLite database to persist our data, located in the db.sqlite3 file in the outer course_project folder. Our database includes the 5 tables shown in Figure 1 (the yellow box is the key for acronyms used). The User table can be found as 'auth_user' in the db.sqlite3 file and is implemented using Django's built-in User class (from django.contrib.auth.models, documentation here: https://docs.djangoproject.com/en/5.1/ref/contrib/auth/).

We only use the attributes name, username, email, and password fields required for Feature 1.1. The 'first_name' built-in Django field acts as the 'name' field. Each user can have zero or many listings. The Listing table, which models each listing a user can make, can be found as 'campusmart_listing' in the db.sqlite3 file and is implemented using a custom model we made, which is a subclass of the Model class inside the models module from django. It includes the following attributes: created_by (a foreign key of the user who created the listing), title, description, price, condition (with choices New, Like New, Used, or Fair), photo, and status (with choices Available or Unavailable).

For Feature 3.3: Seller-Buyer Messaging, we have the additional Conversation and ConversationMessage tables. Each listing can have zero or many conversations, and each user can have zero or more conversations. The Conversation table, which models a conversation about a listing between two users, can be found as 'campusmart_conversation' in the db.sqlite3 file and is implemented using a custom model we made, which is a subclass of the Model class inside the models module from django. It includes the following attributes: listing (a foreign key of the listing the conversation is about), members (the users involved in the conversation), and

created_at (the date the conversation was created). Further, each conversation can have zero or more conversation messages and each user can have zero or more conversation messages. The ConversationMessage table, which models each individual message within a conversation, can be found as ‘campusmart_conversationmessage’ in the db.sqlite3 file and is implemented using a custom model we made, which is a subclass of the Model class inside the models module from django. It includes the following attributes: created_by (a foreign key of the user who sent the message), recipient (a foreign key of the user who received the message), conversation (a foreign key of the conversation the message is in), content (the content of the message), and created_at (the date the message was created).

Lastly, for Feature 4.1, we extended the built-in user model by implementing the Player model, which allows for the additional listings a user has purchased to persist past 24 hours. The Player model can be found as ‘campusmart_player’ in the db.sqlite3 file and is implemented using a custom model we made, which is a subclass of the Model class inside the models module from django. It includes the following attributes: user (the user the player is extending) and additional_listings (the number of additional listings a player has purchased).

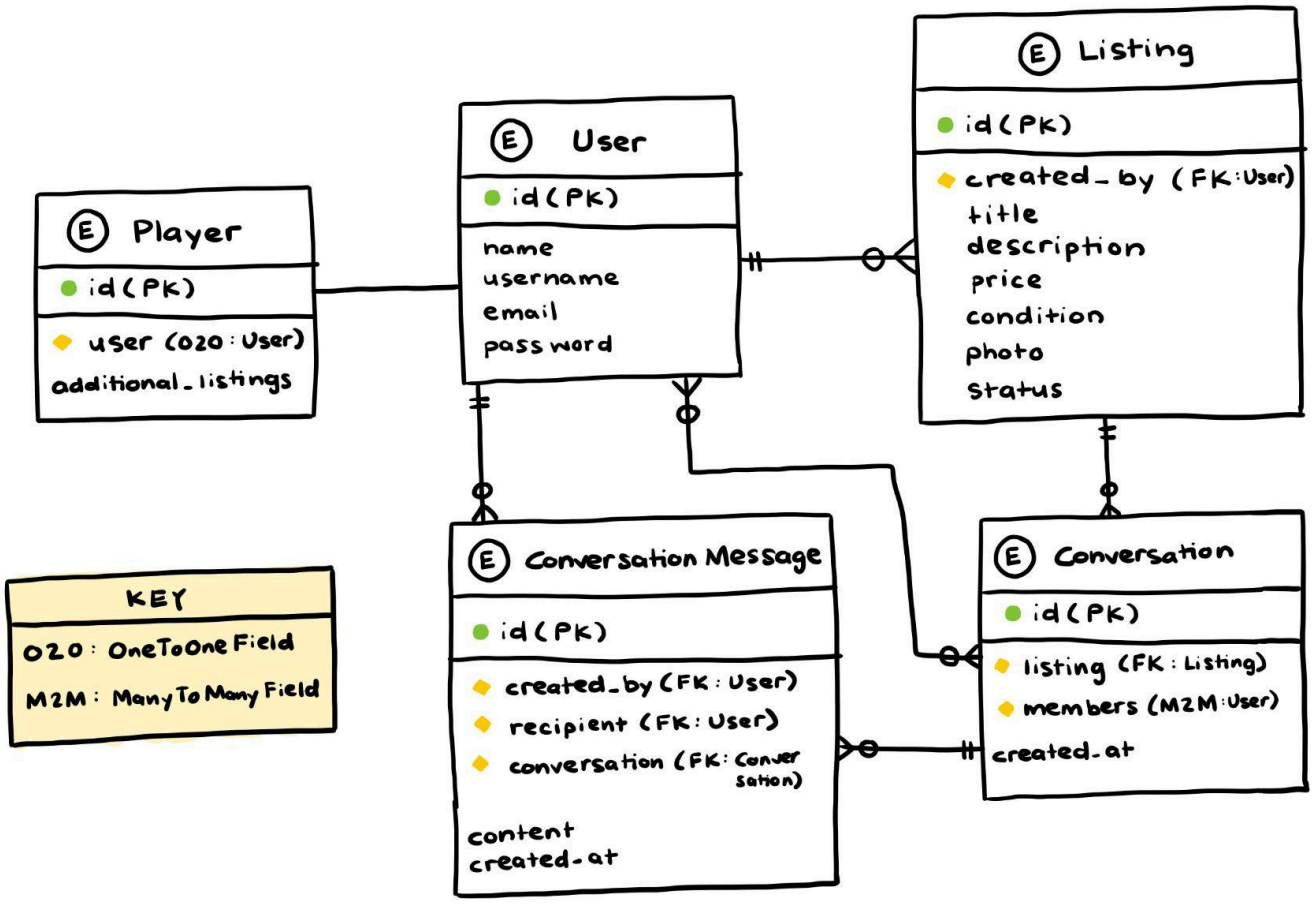

Figure 1. Database Schema with User, Listing, Conversation, ConversationMessage, and Player Tables

Demonstration of the Features

Feature 1.1 Create New User Profile

Figure 2 demonstrates the user Register page where new users can enter their name, email, username, and password to create an account with those specified credentials. You can reach this page by clicking 'Register' in the navigation bar. In our backend, we used `django.contrib.auth.models` API to create a user profile with their name, email, username, and password as displayed in Figure 2 for new users. When the user clicks the blue 'Register' button, a new user profile will be created and stored in the SQLite database, and the user is redirected to the 'Browse' page shown in Figure 17, where all of the available listings are shown.

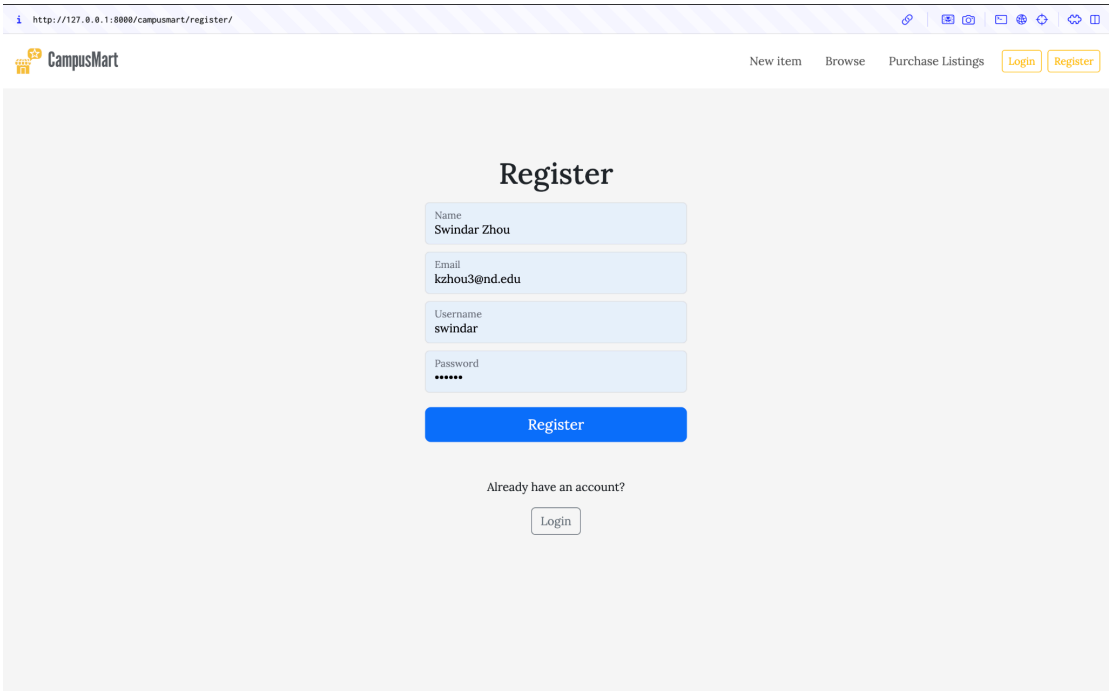

Figure 2. Screenshot for Feature 1.1 showing how to create a new user profile

Feature 1.2 Log-in

Figure 3 demonstrates the login page that a user is redirected to by clicking the yellow ‘Login’ button on the navigation bar. After the user inputs their username and password, when they click ‘Sign in’ the account information is stored in the SQLite database, and the user will be redirected to the ‘Browse’ page. User verification is done through ‘`from django.contrib.auth`’ functions. These functions are used to verify the username and password.

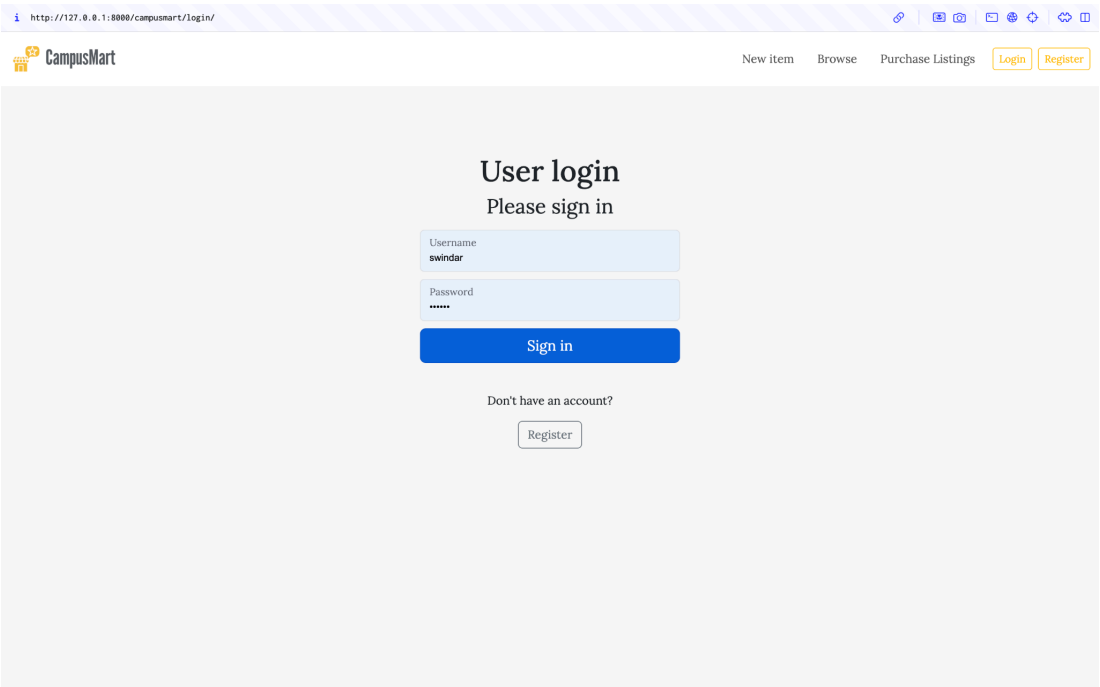

Figure 3. Screenshot for Feature 1.2 showing User Login page

Figure 4 shows the updated navigation bar after the user logs in, with different ‘Inbox’ and ‘Logout’ buttons. In addition, the logged in user’s username appears at the top right of the navigation bar.

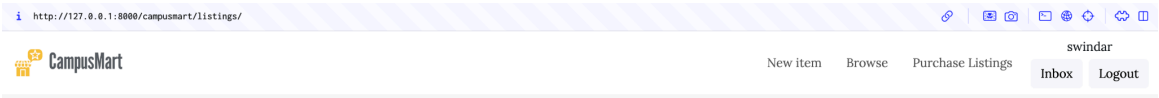

Figure 4. Screenshot for Feature 1.2 showing navigation bar after login

Figure 5 shows the error message that displays if the username and password pair is not valid and keeps the user at the ‘User login’ page to try again.

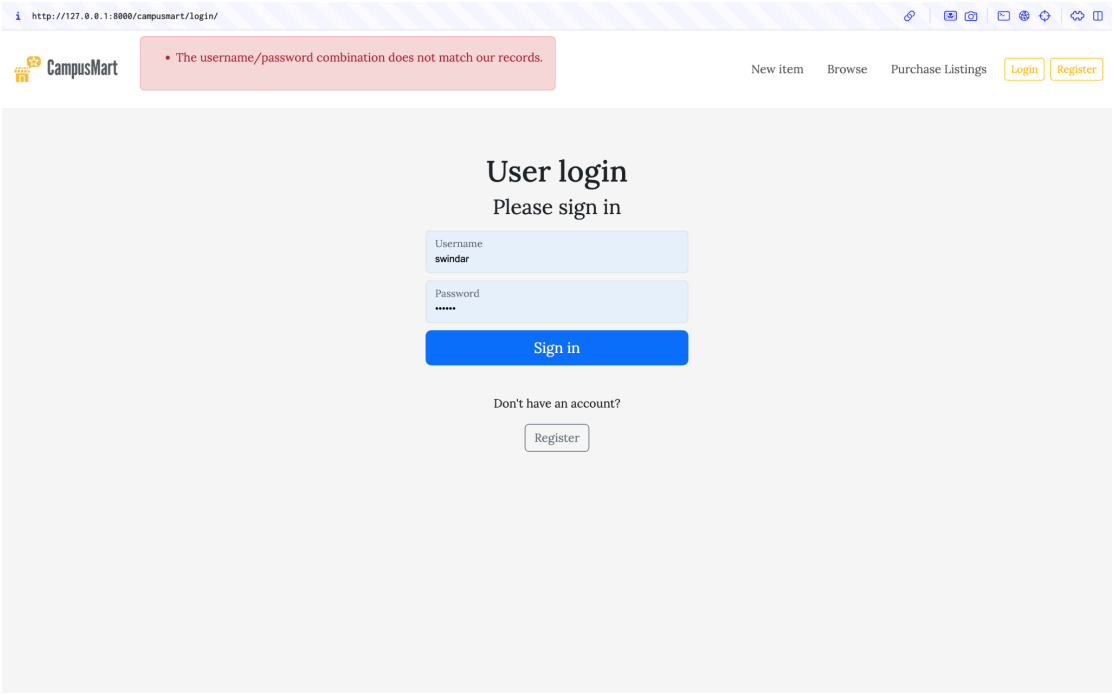

Figure 5. Screenshot for Feature 1.2 showing invalid user login

Feature 1.3 Log-out

Figure 6 shows the Logout button that is displayed to logged-in users, which they can click to log out of their account.

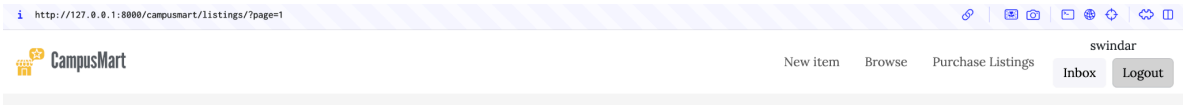

Figure 6. Screenshot for Feature 1.3 showing Logout button in the Navigation Bar

Figure 7 shows what happens after the user clicks the ‘Logout’ button shown in Figure 6, as it redirects the user to the index landing page of the website.

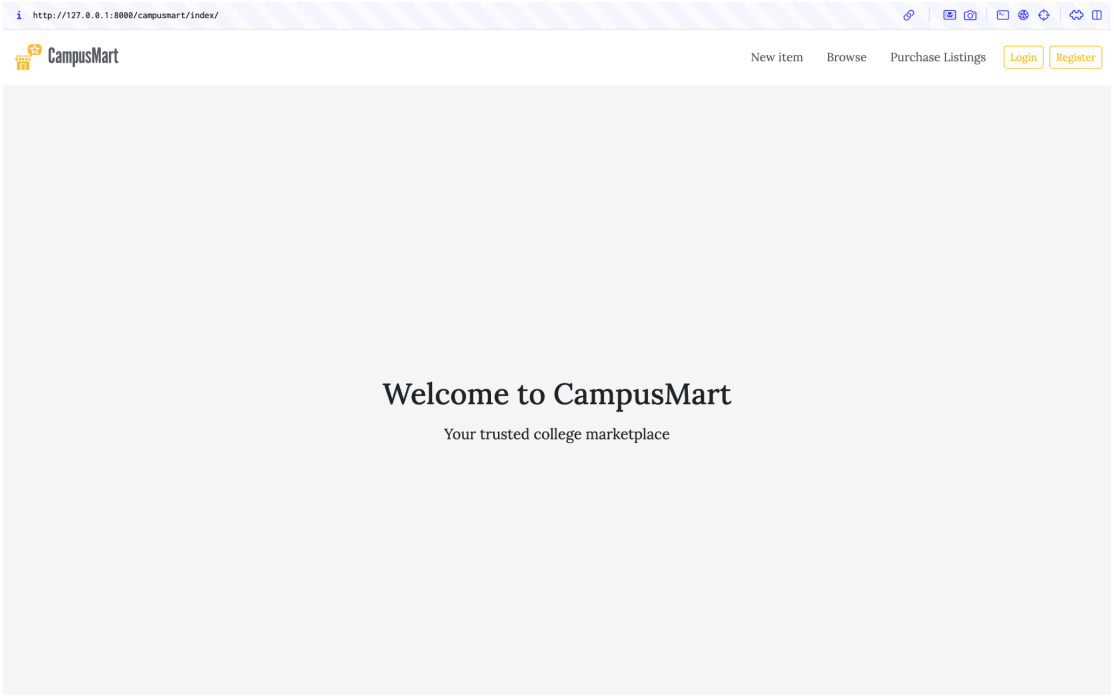

Figure 7. Screenshot for Feature 1.3 showing Redirect to Index Page after Logout

Feature 2.1 Create Listings

Figure 8 demonstrates how a logged-in user can create a listing on the ‘Create Listing’ page by clicking on the ‘New Item’ in the navigation bar. The user is required to enter a title, description of the listing product, price, and product condition (New, Like New, Used, Fair). The user is also required to upload an image file to display their product.

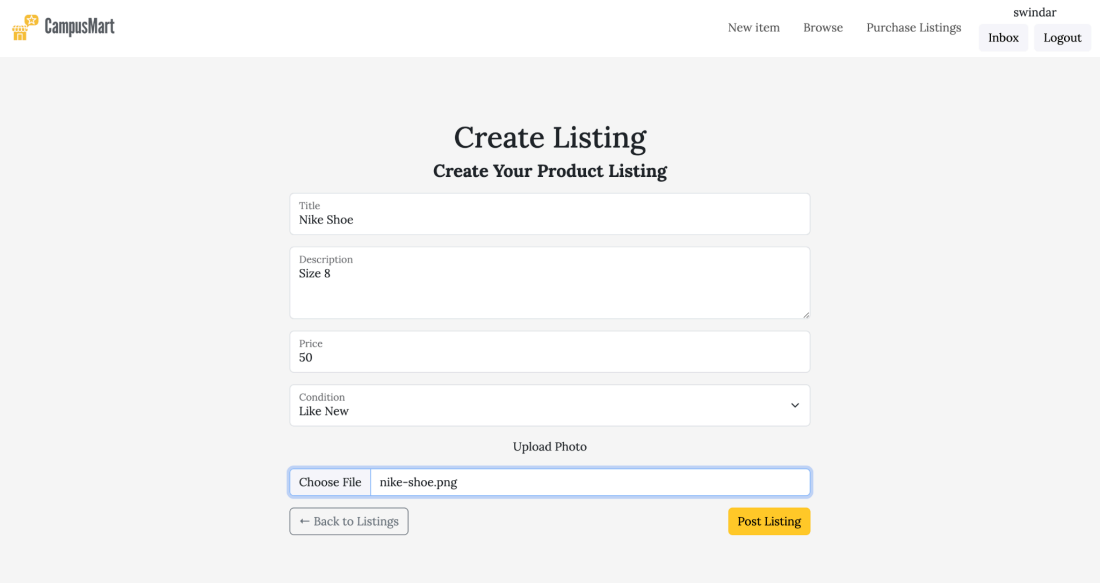

Figure 8. Screenshot for Feature 2.1 of Create Listing Form

Figure 9 shows that a new item ‘Nike Shoe’ has been added to the listing page. Since the item was created successfully, a pop-up message appears in green to remind the user that the new item has been added to the SQLite database.

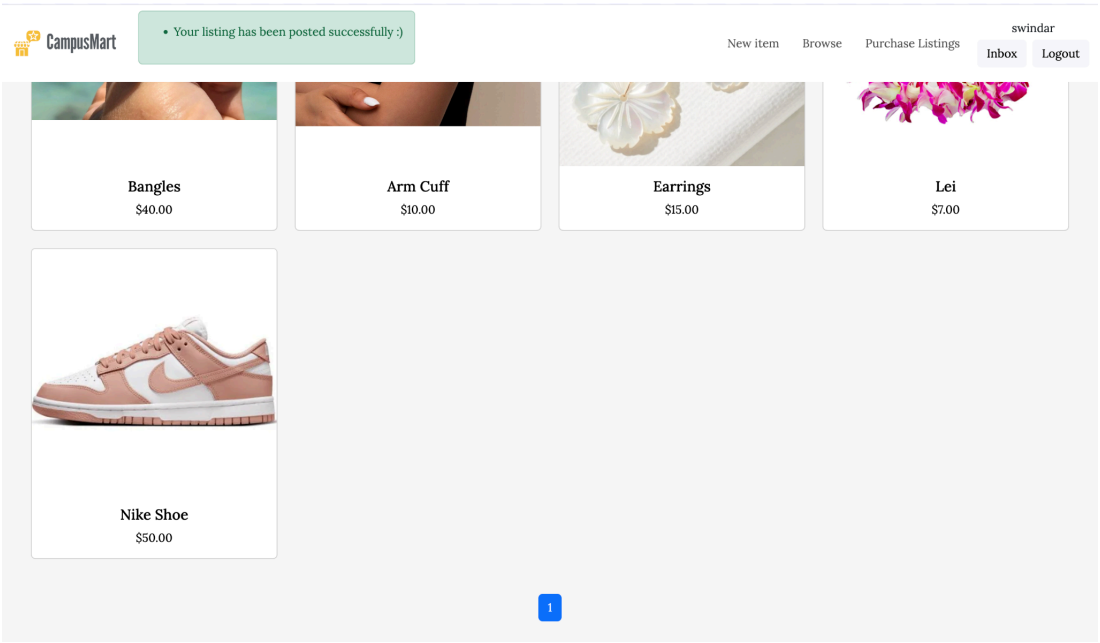

Figure 9. Screenshot for Feature 2.1 showing creation of new item ‘Nike Shoe’

Figure 10 shows what happens when the user attempts to create a listing when they have already reached their daily limit (3 listings max). A pop-up message in red indicates that the user needs to buy an extra listing, which redirects the user to the ‘checkout’ page that is implemented in Feature 4.1, explained later in this report.

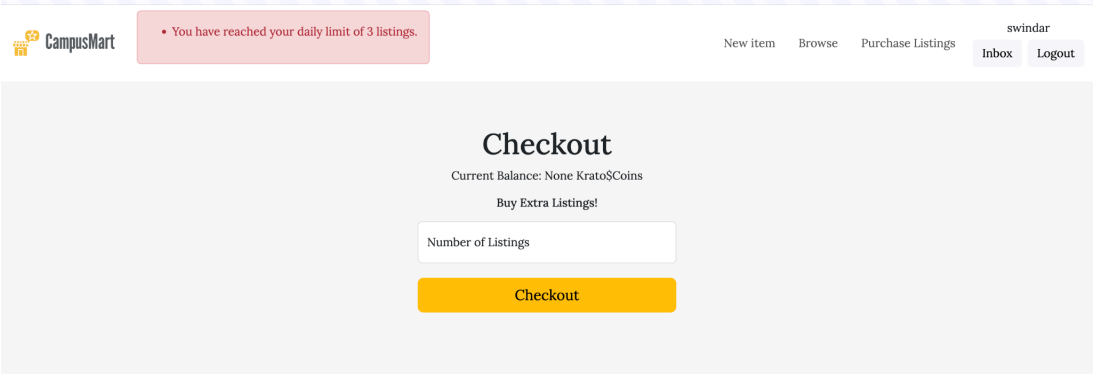

Figure 10. Screenshot for Feature 2.1 showing how daily listing limit is handled

Feature 2.2 Update Listings

A logged-in user who created the listing can edit their listing by clicking the specific item in the ‘Browse’ page, which will redirect them to the detailed page that displays the listing. Figure 11 shows this page, which has an ‘Update’ button in yellow that the user can click in order to edit the listing.

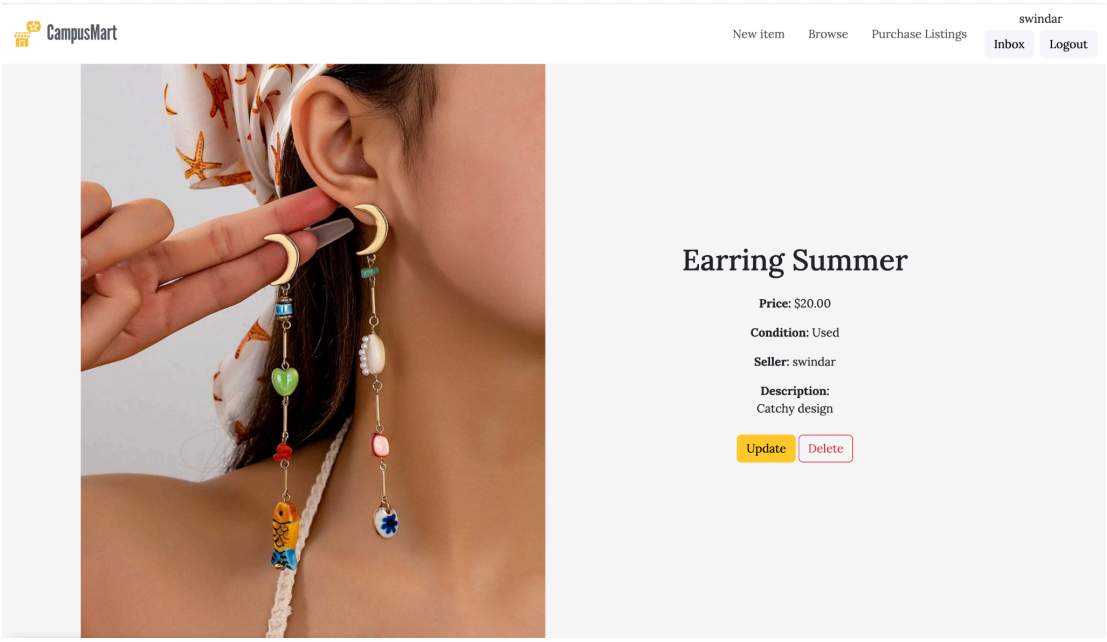

Figure 11. Screenshot for Feature 2.2 showing Update button

Figure 12 is the page the user is redirected to after they click the 'Update' button. This form allows them to update the title, description, price, condition, status, or photo.

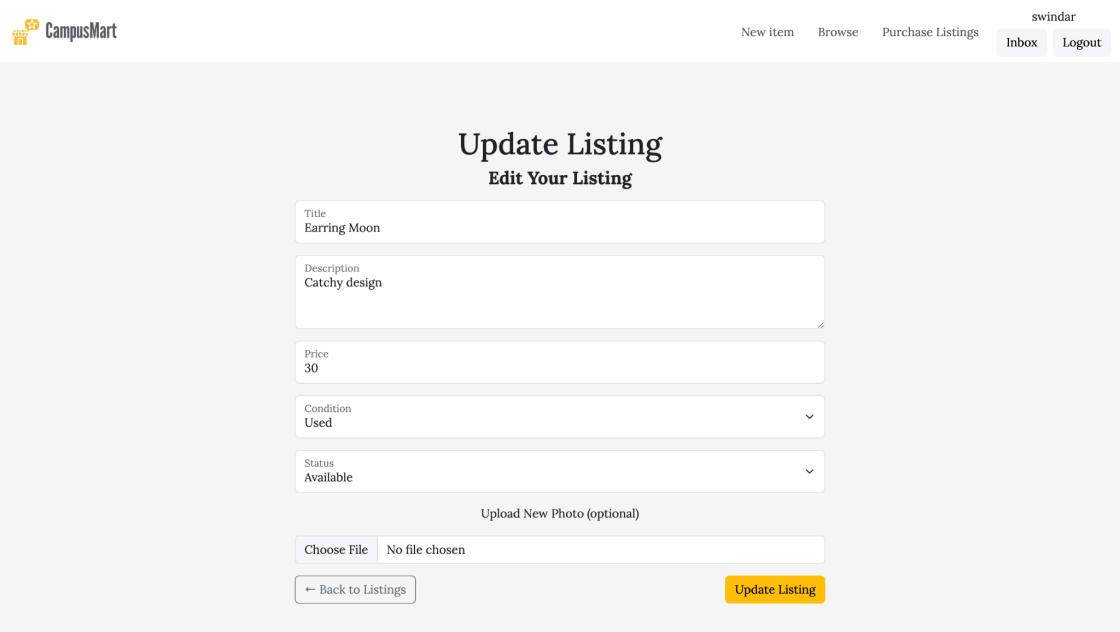

Figure 12. Screenshot for Feature 2.2 showing how to modify the listing through form

After the user submits their changes, they are redirected back to the Listing page which will show a pop-up message in green that says 'Listing updated successfully!' Figure 13 shows the updated listing item, which displays the new title 'Earring Moon' with the new price $30 from Figure 12.

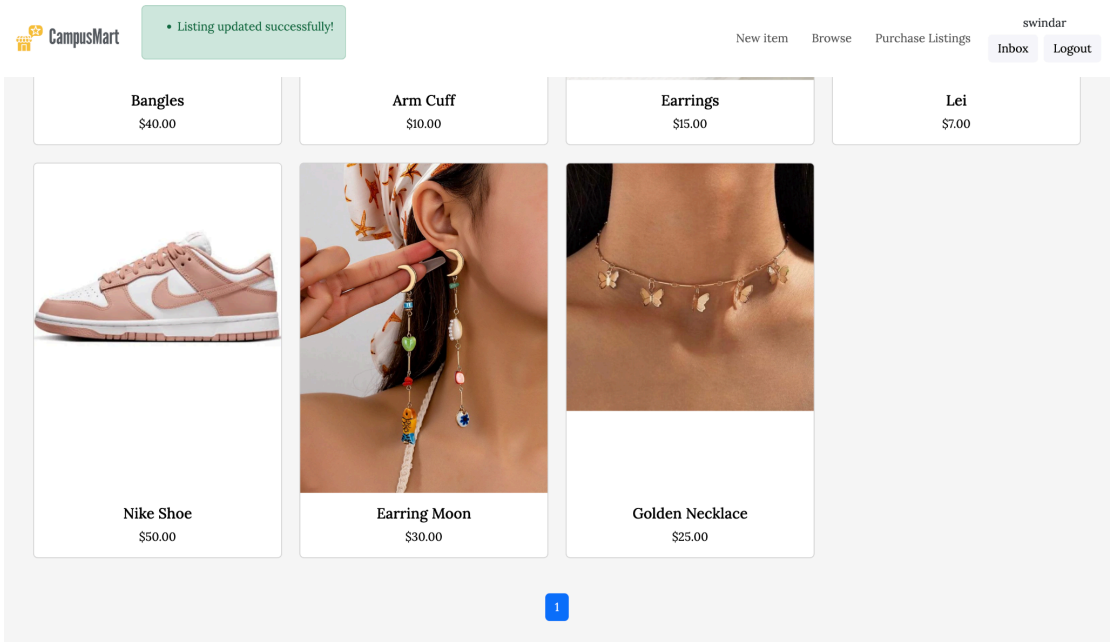

Figure 13. Screenshot for Feature 2.2 showing Updated Listing Page

Feature 2.3 Delete Listings

When the user clicks on the specific item that he/she added on the Listing page, the user can also choose to delete the item if they don't want to sell it, shown in Figure 14.

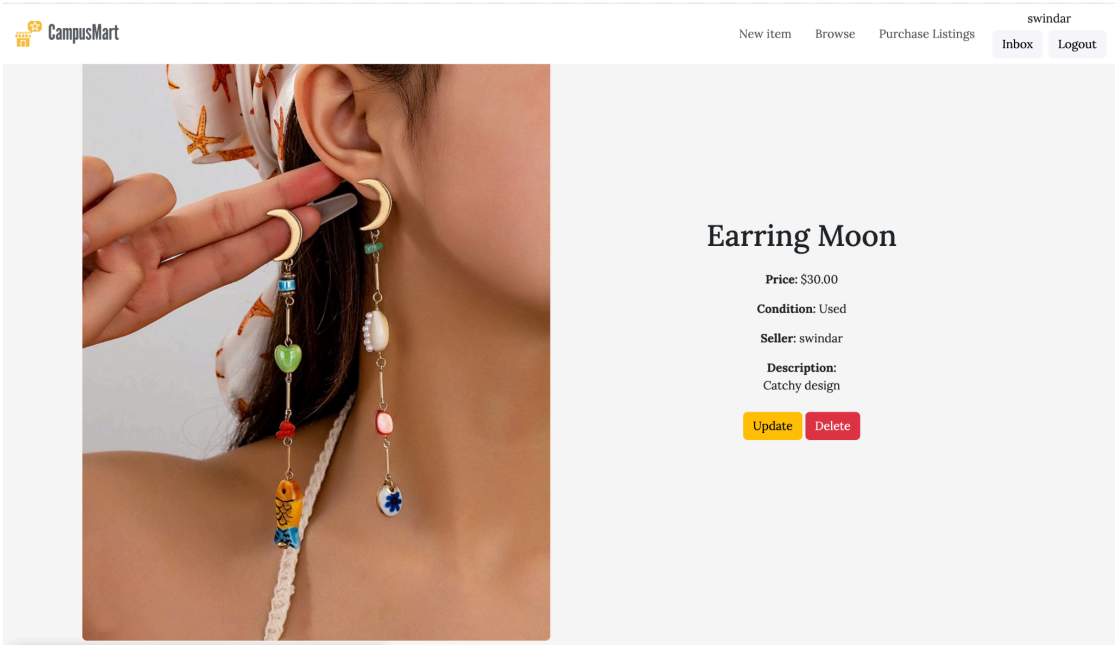

Figure 14. Screenshot for Feature 2.3 showing Delete button

If the user clicks the red 'Delete' button for the selected item shown in Figure 14, they are redirected to the page shown in Figure 15, which asks the user if they are sure about deleting the item. The user can then confirm the deletion by clicking the 'Yes, Delete' button. The user also has the ability to avoid the deletion by clicking the 'Cancel' button.

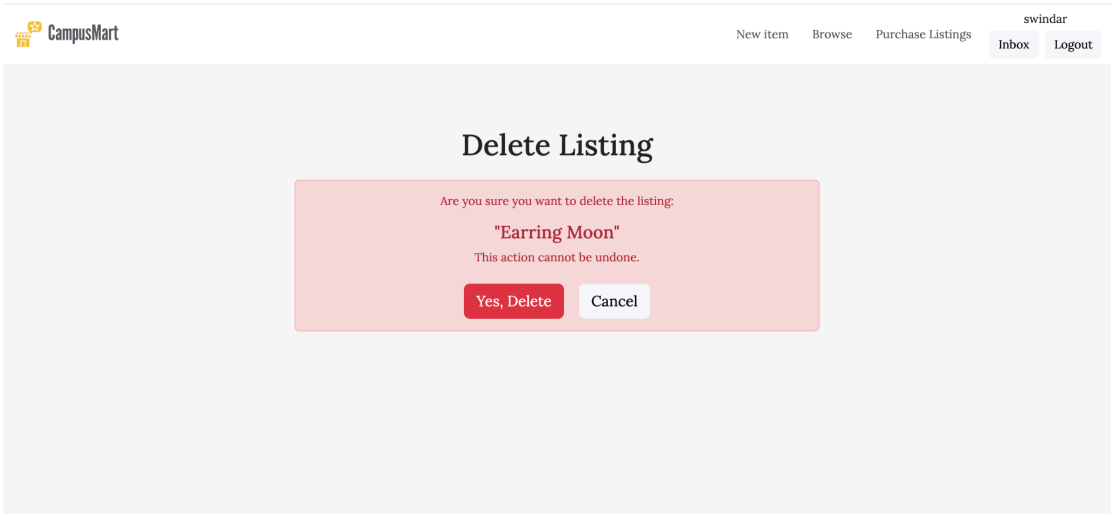

Figure 15. Screenshot for Feature 2.3 showing Confirmation page for Deleting item

If the user insists on deleting the selected and clicking the ‘Yes, Delete’ button in Figure 15, the user will be redirected to the Listing page. Figure 16 shows this page, with a pop-up message in green that says ‘Listing deleted successfully.’ The deleted item will also disappear from the Listing page and be deleted from the SQLite database.

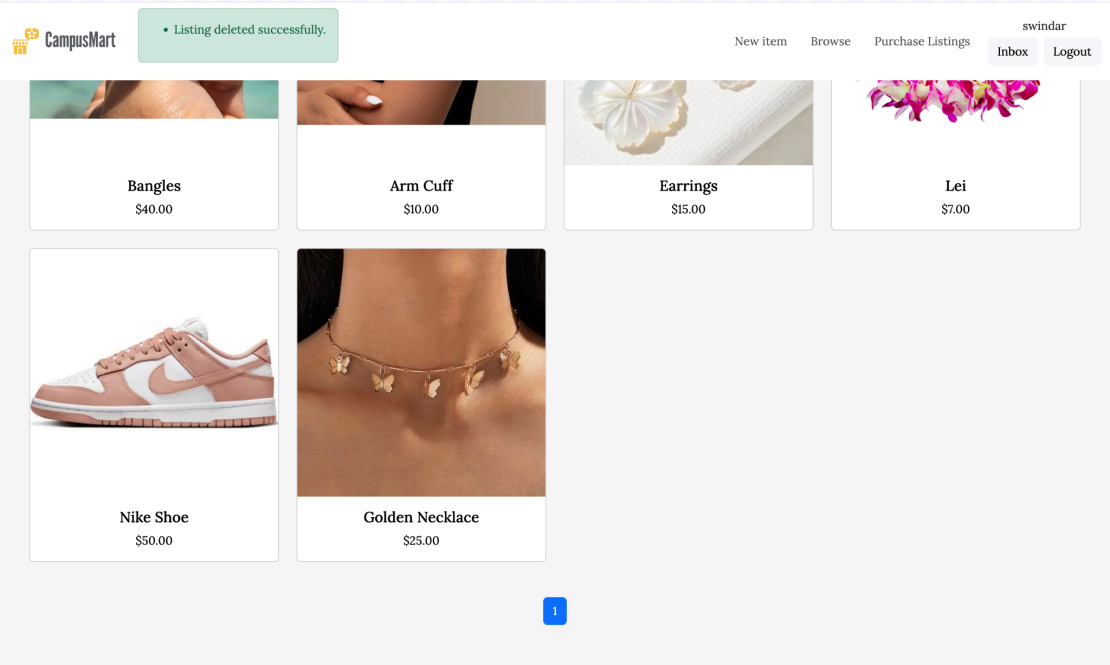

Figure 16. Screenshot for Feature 2.3 showing Redirect to Listing page after successful deletion

Feature 3.1 View All Listings

Figure 17 shows the implementation of Feature 3.1, showing how users can browse listings on the ‘Browse’ page. This feature takes all the listings with the status ‘Available’ and renders a snapshot of them to the user. The page shows the user the listing’s title, price, and thumbnail image.

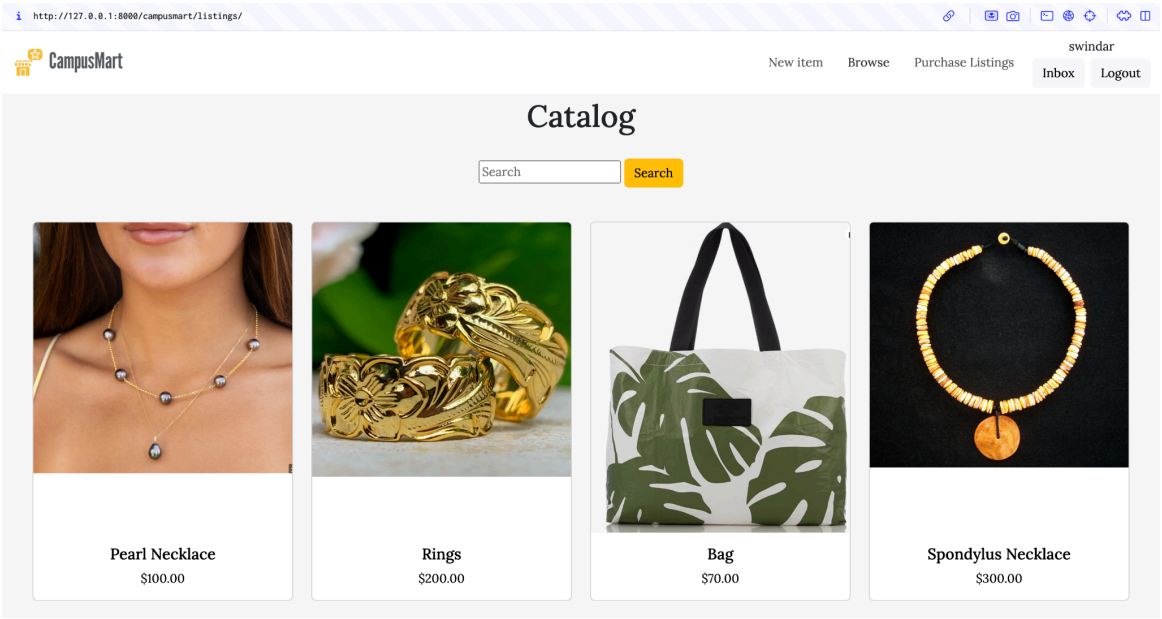

Figure 17. Screenshot for Feature 3.1 of all Listings Available

If the page has more than twenty listings, the user would be able to navigate through the pages by using ‘Next’ and ‘Previous’ buttons at the bottom of the screen, with a highlighted page number that the user is currently on, shown in Figure 18.

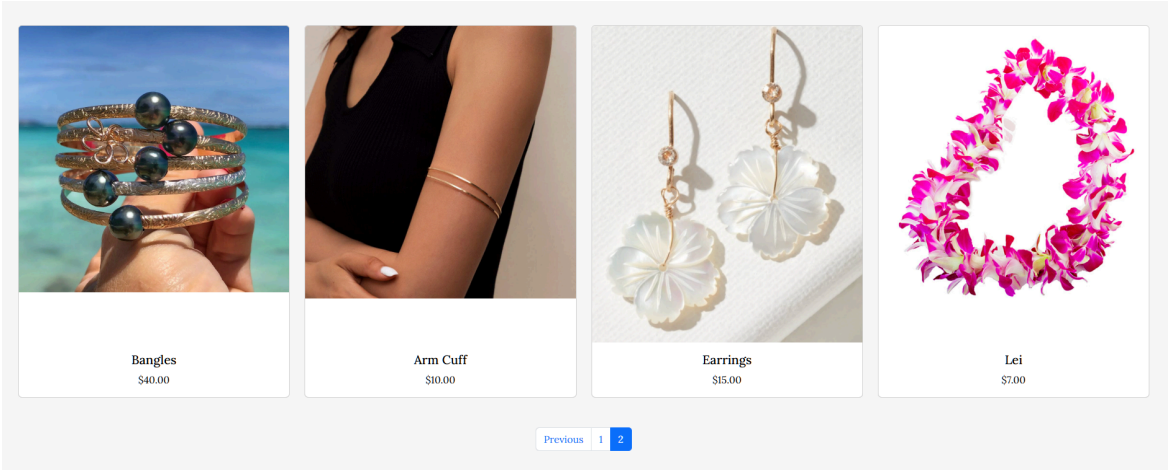

Figure 18. Screenshot for Feature 3.1 showing pagination

Figure 19 shows a detailed view of the listing with additional information like an item's description and condition. The user can reach this detailed view by clicking on the card of the item they are interested in from the Catalog shown in Figure 17. If the user did not create the item, they see the yellow 'Contact Seller' button shown in Figure 18. If not and are logged in, they see the 'Update' and 'Delete' buttons shown in Figure 11.

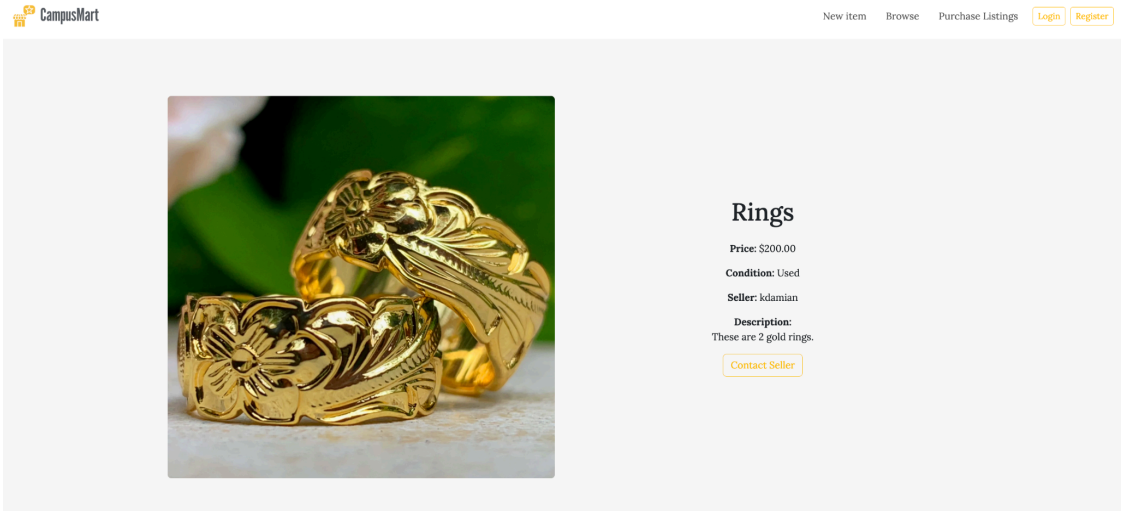

Figure 19. Screenshot forDetailed View of a Listing Titled "Rings"

Feature 3.2 Search For Products

On the same 'Browse' page, the user can search for items based on their title or description, by typing keywords into the 'Search' box and using the 'Search' button to render a filtered catalog view. Figure 20 shows that when the user inputs 'Earring' in the 'Search' box and clicks the 'Search' button, matching results that contain the word 'Earring' in the item's title or description will be displayed.

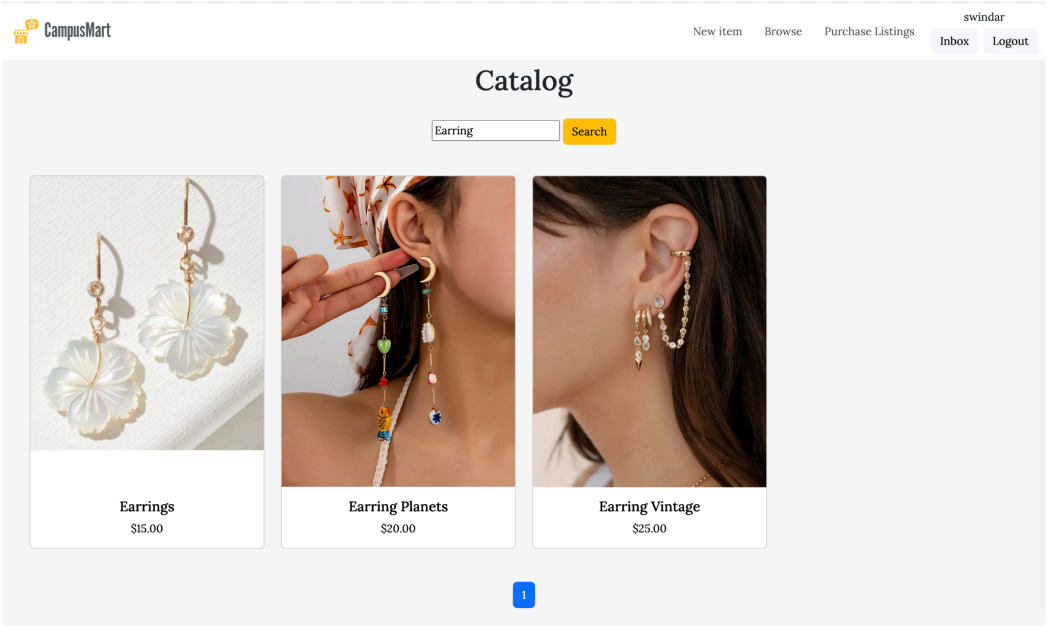

Figure 20. Screenshot for Feature 3.2 showing search results

Feature 3.3 Seller-Buyer Messaging

Logged-in users have the ability to message the seller if they are interested in buying the item or want more information. Figure 21 shows the page that users are taken to when clicking on the yellow ‘Contact Seller’ button shown in Figure 19. The user sends a message to the seller by typing into the ‘Your Message’ box and pressing the ‘Send’ button. The message is then rendered onto the page, showing the user who sent the message and a timestamp, along with the message.

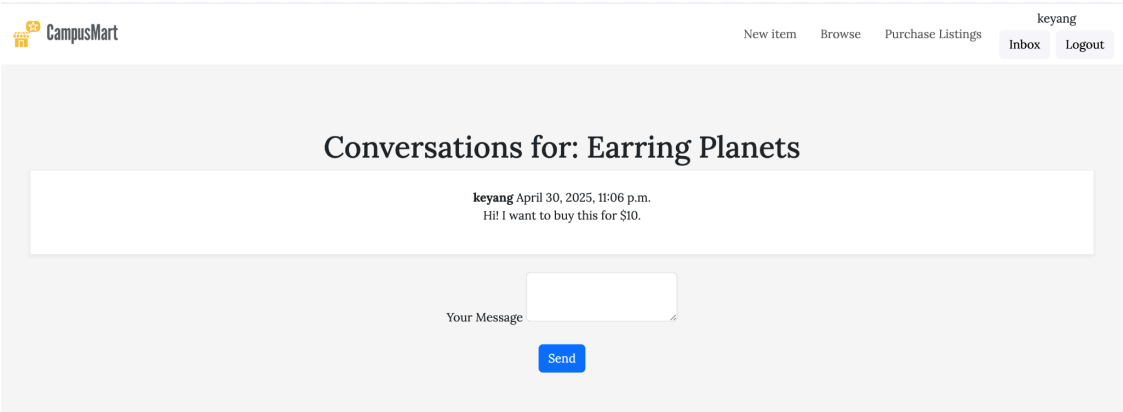

Figure 21. Screenshot of Feature 3.3 showing Conversation

Figure 22 shows all the messages the logged-in user has sent or received for products that have listed or are interested in. The ‘Inbox’ page can be reached by clicking the ‘Inbox’ button in the Navigation Bar. By clicking on the ‘View Conversation’ button, the user can view their message history shown in Figure 21.

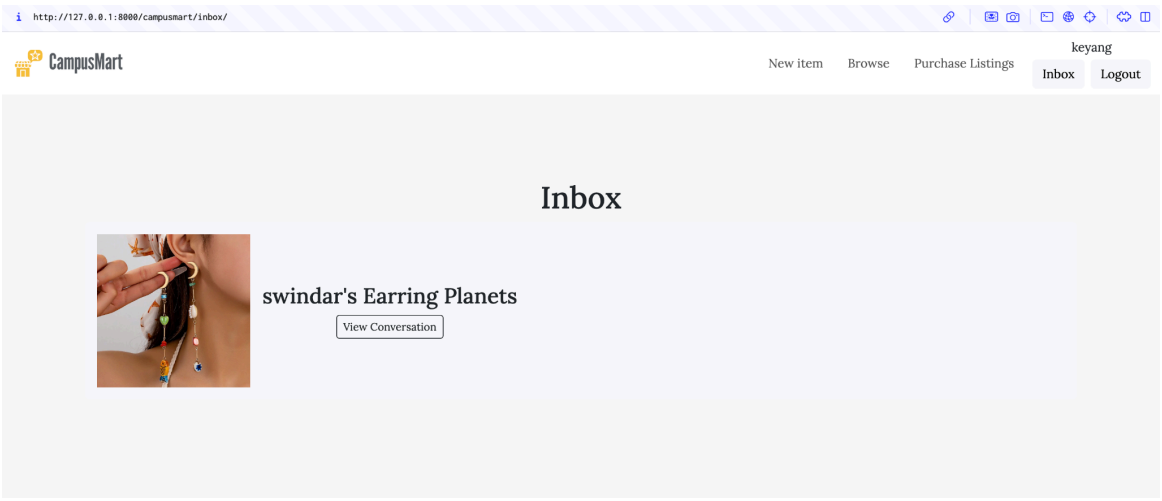

Figure 22. Screenshot of the logged-in users ‘Inbox’

Feature 4.1 Buying Daily Listings

Figure 23 shows the ‘Purchase Listings’ page, where the user (if he/she is registered as a Player and has coins) will be able to purchase additional listings that he/she can use after exceeding the daily limit. The user is presented with their current balance, and they can input how many additional listings they would like to purchase. This feature is implemented by making a request to the REST API specified in the project document.

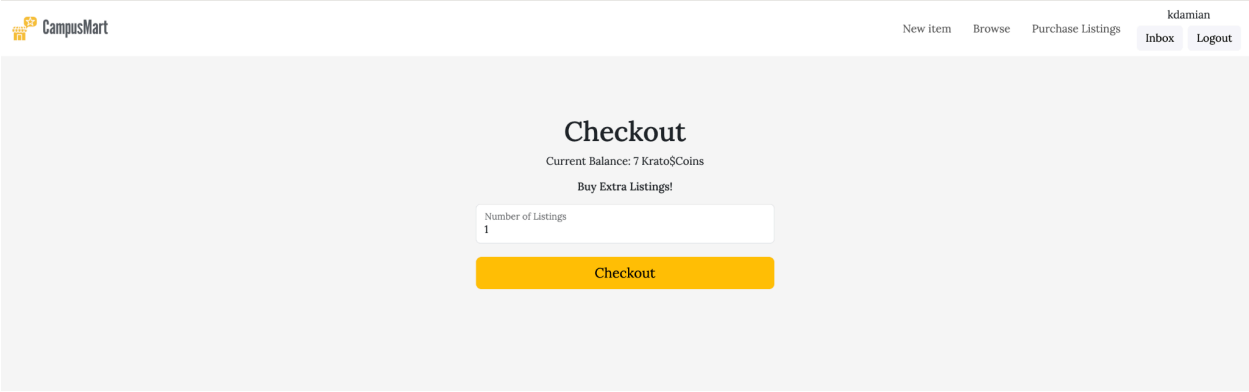

Figure 23. Screenshot for Feature 4.1 showing ‘Purchase Listings’ page

Figure 24 shows the message that is displayed after the user successfully purchases additional listings (one in this example). The current balance also decreases the appropriate amount after a successful purchase.

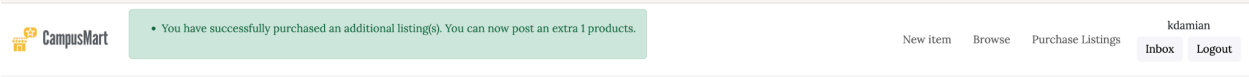

Figure 24. Screenshot for Feature 4.1 showing successful purchase message in navigation bar

Figure 25 shows the checkout page for a user who is not registered as a Player and does not have any coins, showing None as their current balance.

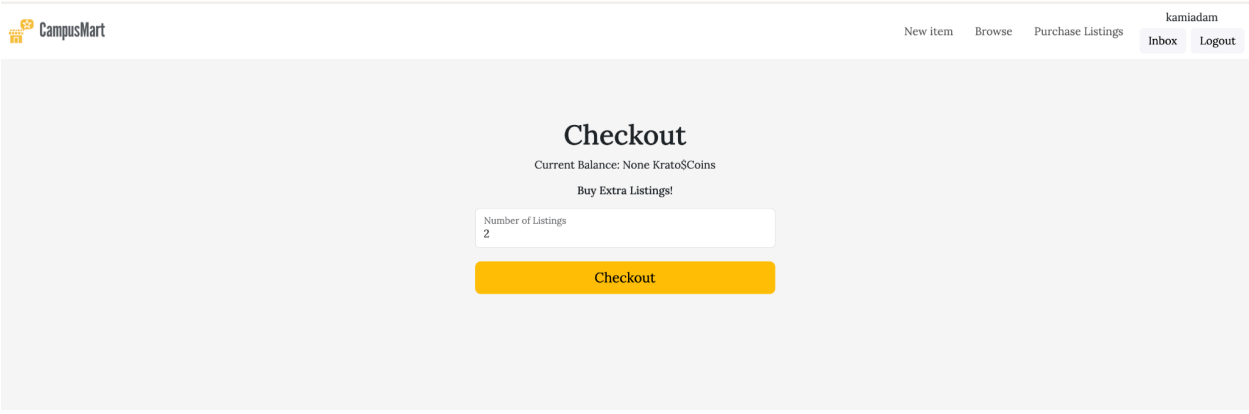

Figure 25. Screenshot for Feature 4.1 showing Checkout page for unregistered user

Figure 26 shows the error message that a user who is not registered as a Player and does not have any coins receives if he/she tries to purchase additional listings.

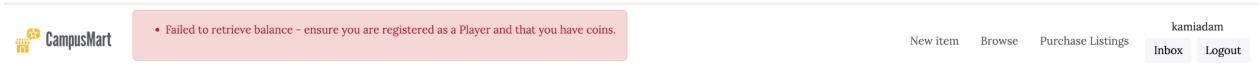

Figure 26. Screenshot for Feature 4.1 showing unsuccessful purchase message in navigation bar

Project's Learned Lessons

1. What programming paradigm(s) have you chosen to use and why? If you were to start the project from scratch now, would you make different choices? Do you think the paradigm(s) chosen helped (or not) in developing the project?
    a. We utilized the object-oriented programming paradigm in this project. It allowed us to abstract the problem in terms of classes (or models in this case) of objects. We saved these objects to our persistent sqlite database.
    b. We also utilized the imperative programming paradigm mainly within our view functions, as they have a linear flow, explicitly defining the control flow with many if statements.
    c. If we were to start the project from scratch, we would take the same approach since it is what makes the most sense to us, and as these paradigms are what we are most familiar with. Thus, using these well-known paradigms definitely helped us in developing our project.
2. What were the most intellectually challenging aspects of the project?
    a. One intellectually challenging aspect of the project was getting all of the html files, view functions, and url mappings to link up. The project features many forms, and we had to ensure the data was being created, updated, and deleted correctly.
    b. Another intellectually challenging aspect of the project was implementing the REST API in Feature 4.1. In this realistic example, it was hard to get our heads around the data flow and how everything was working.
3. What aspects of your process or your group's organization had the largest positive effect on the project's outcome?
    a. Our group was very good at communicating, as we have a group chat that we text on to share our progress and communicate meeting times. This openness and understanding definitely had the largest positive effect on our project's outcome.