Team Member Full Name	NetID
Ethan Little	elittle2
Derick Shi	dshi2
Michael Egan	megan3
Aaron Wang	awang27

Persistent Storage Design

We are using SQLite database to persist our data. Our database includes the tables shown in *Figure 1*. Since we are utilizing the default Django authentication system, there are many additional tables and attributes that our program does not use, so those are not displayed. For our purposes, the database contains 4 tables:

1. **auth_user** – the built-in table provided by Django to store user information.
2. **campusmart_listingcounter** – tracks how many listings each user posts per day.
3. **campusmart_listing** – stores individual listings. Each listing is linked to a user via a foreign key.
4. **campusmart_listingimage** – stores images associated with listings. Each image is linked to a specific listing.
5. **campusmart_message** – stores messages title, body and the user and listing ids associated with each message.
6. **campusmart_purchasedlistingcounter** – stores the count of extra listings each person has (the listings they have purchased and not used).

Instead of creating separate tables per user or listing, Django uses foreign key relationships to associate data.

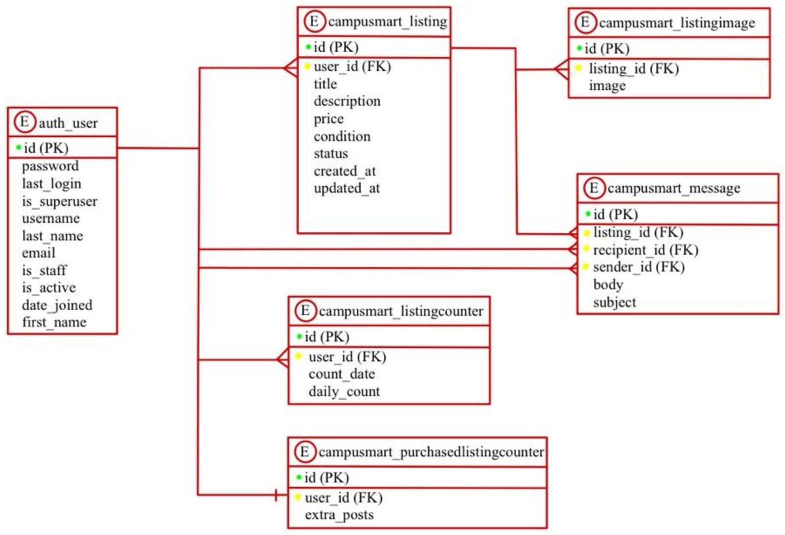


Figure 1 database schema

Demonstration of the Features Implemented for Phase 1

Sign up (Feature 1.1)

Figure 2 shows a screenshot for the sign-up page. When the user clicks the “Sign Up” button, a new user is created in the database based on the fields that are input. It utilizes the default Django authentication system, which automatically checks for invalid input such as usernames already in the database, weak passwords, conflicting passwords, and empty fields.

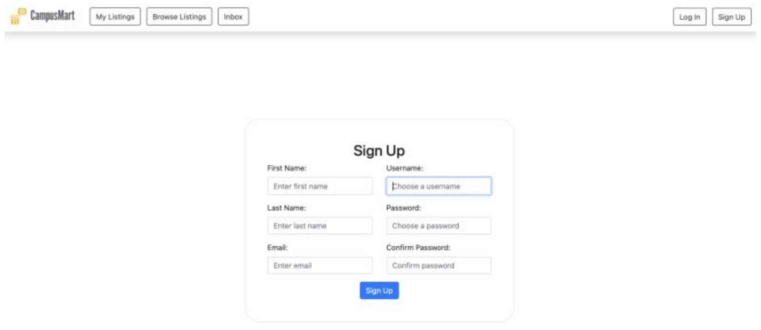


Figure 2 screenshot for feature 1.1

Log in (Feature 1.2)

Figure 3 shows a screenshot for the log-in page. When the user clicks the “Log In” button, the input fields are checked to see if they match the username and password of any user in the database. If they do, the user is authenticated and the navbar displays a “Hello, <username>!” message next to the “Log Out” button (as show later in *Figure 4*) and the user is redirected to a temporary “Home” page. Once again, this utilizes the default Django authentication system which checks for invalid password/username and displays an appropriate message.

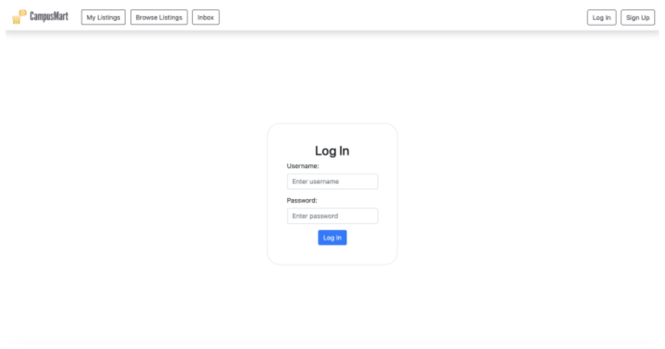


Figure 3 screenshot for feature 1.2

Log out (Feature 1.3)

Figure 4 shows a screenshot for our “Log Out” feature. It is a button in the top right corner of the screen that is only displayed when a user is logged in. When clicked, a post request is sent which logs user out and redirects back to the Home Page. Meanwhile the “Log In” and “Sign Up” buttons are displayed again on the navigation bar.

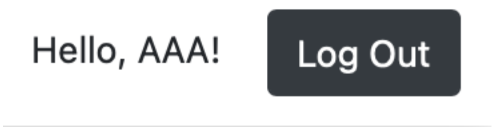
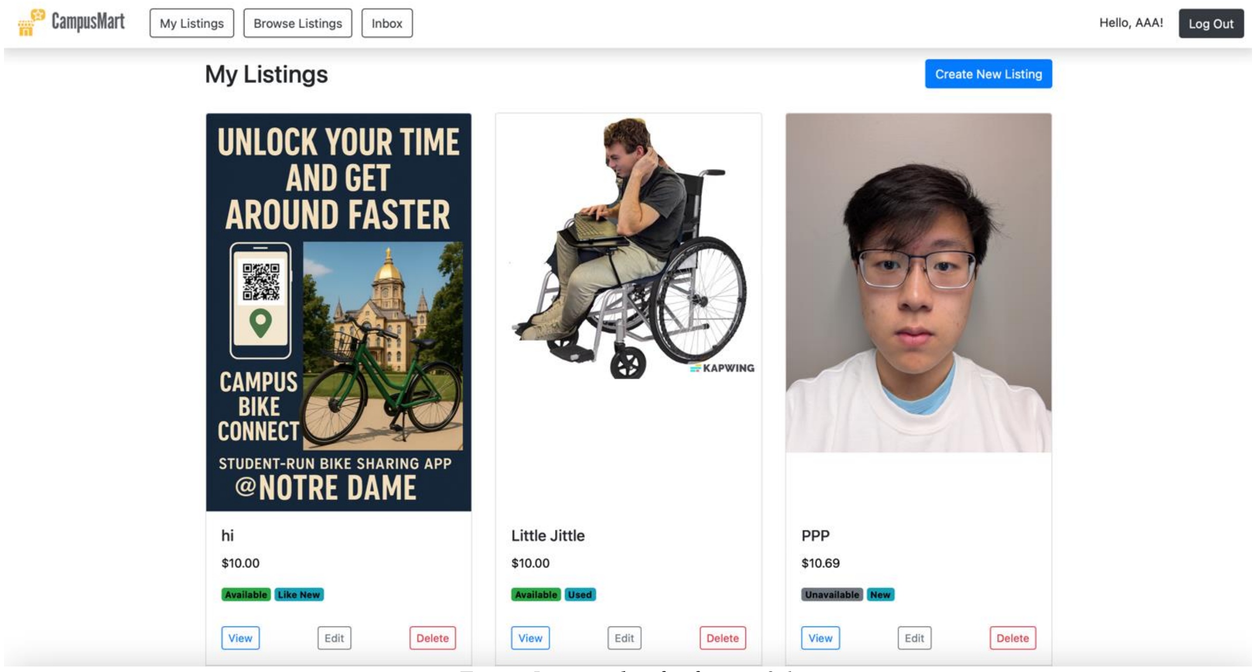


Figure 4 screenshot for feature 1.3

Create Listings (Feature 2.1)

Figure 5 shows a screenshot for the My Listings page. This shows the created listings for our user in the marketplace as well as the main attributes of its name, price, and its image. It also shows the corresponding buttons to bring up other pages to change the attributes for each listing like to view, edit, or delete.

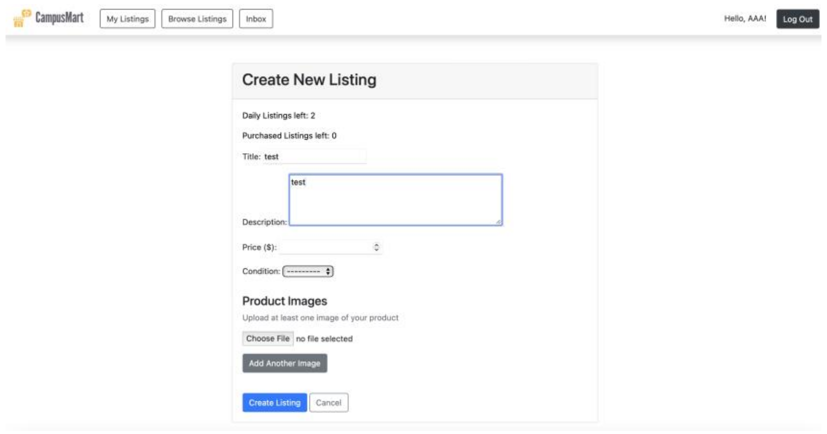


My Listings

hi	Little Jittle	PPP
$10.00	$10.00	$10.69
Available Like New	Available Used	Unavailable New

Figure 5 screenshot for feature 2.1

If a user clicks the "Create New Listing" button, it will bring up the listing form template and prompt them to enter the appropriate information to create another item to sell. This would create another entry in our **campusmart_listing** table in our database that stores the information needed for the listing. This could be seen in *Figure 6*.



Create New Listing

Daily Listings left: 2

Purchased Listings left: 0

Title: test

Description: test

Price ($):

Condition: ---------

Product Images

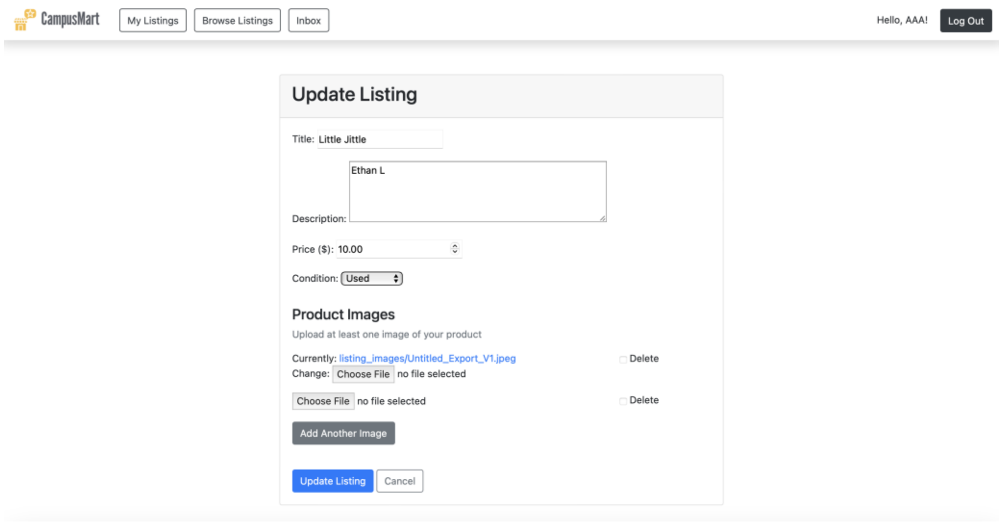
Upload at least one image of your product

no file selected

Figure 6 screenshot for feature 2.1

We also implemented the necessary safeguard to make sure that all items created would have the necessary information and you could only create 3 listings per day for a user or it prevents you from creating more. Feature 4.1 will go more in detail about this. *Figure 6* shows a “fill out this field” notification.

Update Listings (Feature 2.2)
If a user clicks the “Edit” button from *figure 5* on an item that they have created a listing for, it will bring up the listing form template and prompt them to edit the appropriate information the item. This edits the entry in our **campusmart_listing** table in our database that stores the information needed for the listing. *Figure 7* shows a screenshot for our update form using the same template as the create listing form.



Update Listing

Title: Little Jittle

Description: Ethan L

Price ($): 10.00

Condition: Used

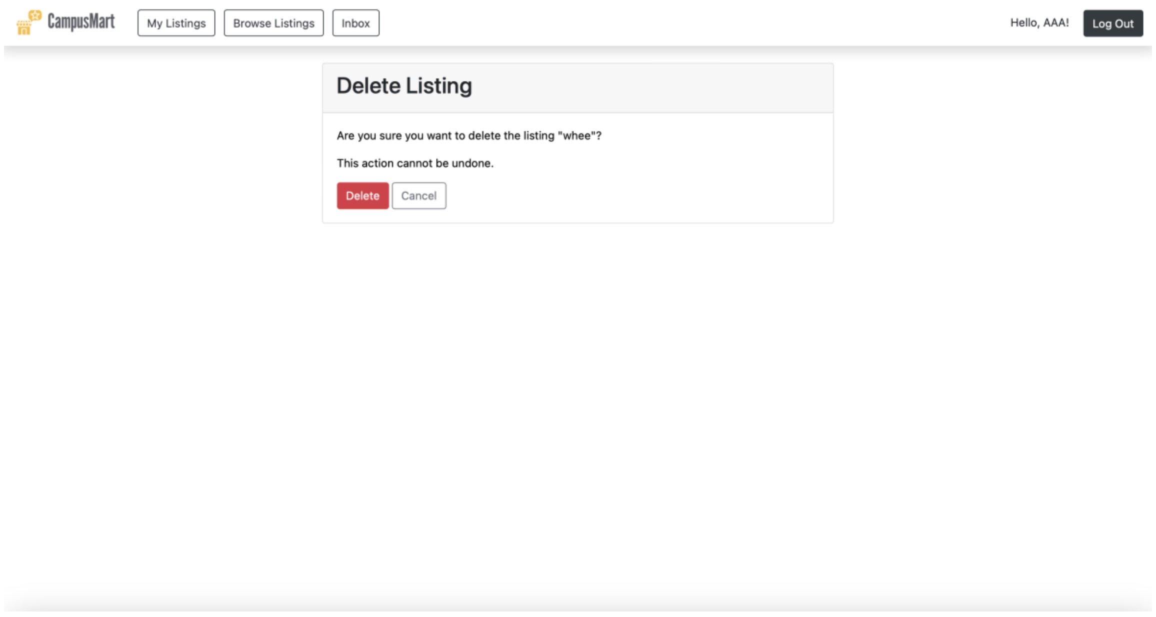
Product Images
Upload at least one image of your product

Currently: listing_images/Untitled_Export_V1.jpeg Delete
Change: Choose File no file selected

Choose File no file selected Delete

Figure 7 screenshot for feature 2.2

Deleting Listings (Feature 2.3)
We also have a “delete” button from *Figure 5*. When clicking this button, a confirmation of deletion pops up as shown in *Figure 8*. If cancelled, nothing happens. If you confirm deletion, it removes the listing from **campusmart_listing** database.



Delete Listing

Are you sure you want to delete the listing "whee"?

This action cannot be undone.

Figure 8 screenshot for feature 2.3

View all listings (Feature 3.1)
Figure 9 shows a screenshot of our market page. This page shows all available listings from other users in a grid format. It also shows a button to view details of the listing. Additionally, we have implemented pagination so that only the 20 most recent products are shown on the first page. *Figure 10* shows how we implement pagination. After reaching 20 items, there are buttons at the bottom to navigate pagination.

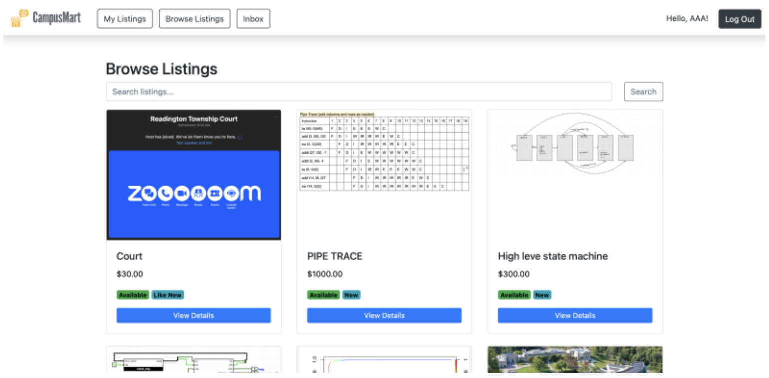


Figure 9 screenshot for feature 3.1

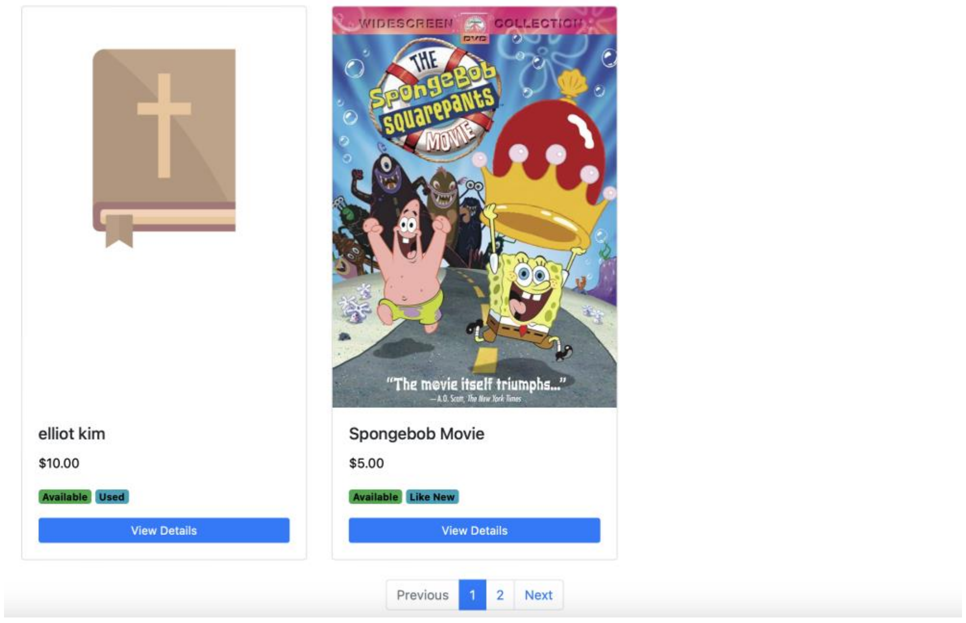


Figure 10 screenshot for feature 3.1

Searching for Products (Feature 3.2)

Figure 11 shows a screenshot of our search bar at work. Our search bar works by creating a case insensitive query for contains in both the title and descriptions of each listing. It then returns all items that the query finds. Below is an example that finds items based on both Title and description.

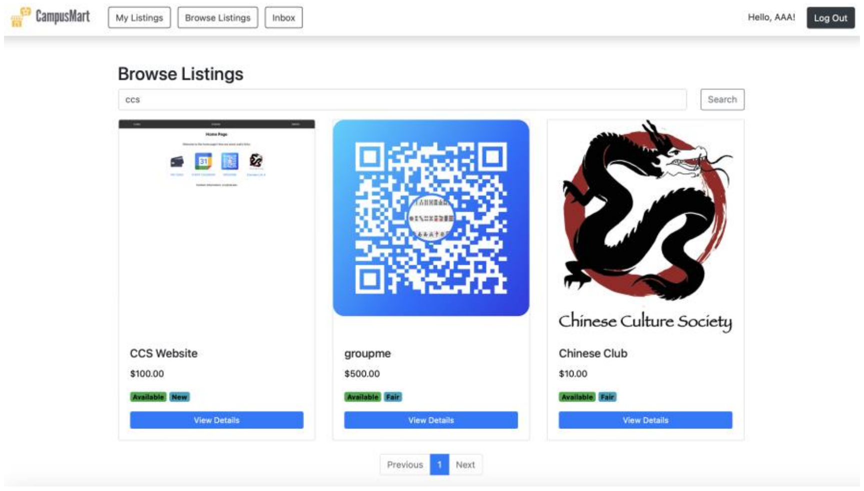
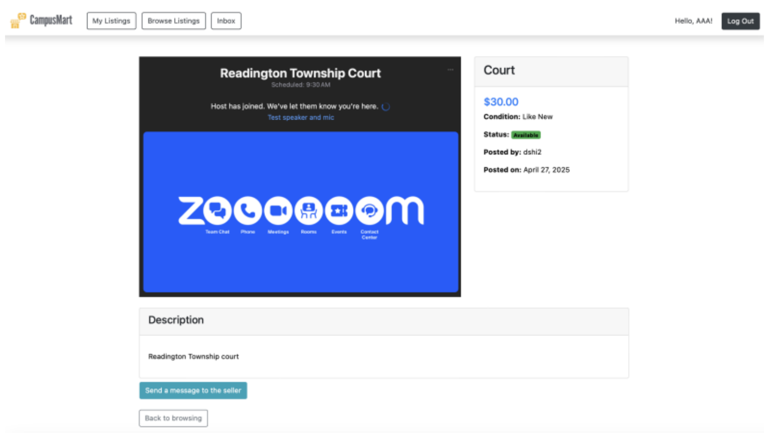


Figure 11 screenshot for feature 3.1

Seller-Buyer Messaging (Feature 3.3)

We have implemented a feature to allow users to send and receive messages about individual listings. The View Details buttons shown in *Figures 9-11* redirect you to a page with a detailed description of each product as well as a carousel for images as shown in *Figure 12*.



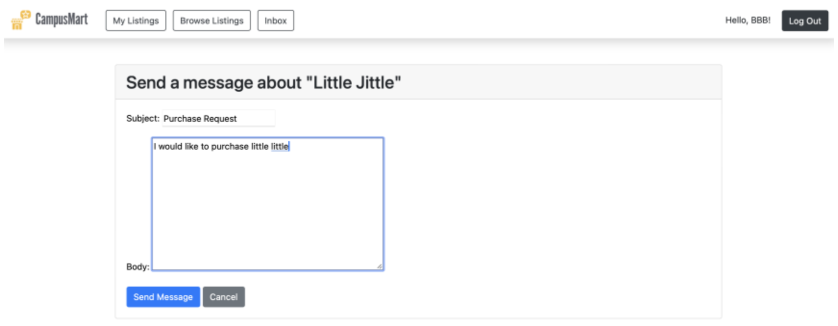
Readington Township Court
Scheduled: 9:30 AM
Host has joined. We've let them know you're here.
Test speaker and mic

Court
$30.00
Condition: Like New
Status: Available
Posted by: dshi2
Posted on: April 27, 2025

Description
Readington Township court

Figure 12 screenshot showing listing detail

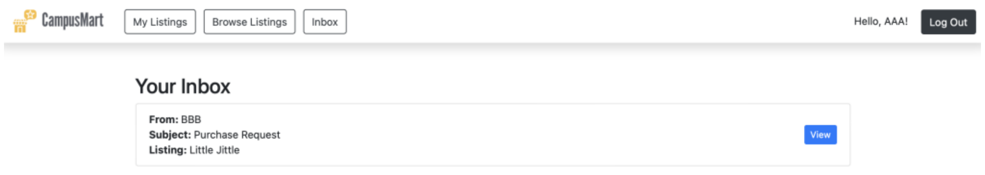
The Listing detail page allows you to send a message to the seller. Upon clicking the Send Message button, the user is redirected to a form to draft a message as shown in *Figure 13*.



Send a message about "Little Jittle"
Subject: Purchase Request
Body: I would like to purchase little little

Figure 13 screenshot showing send message form

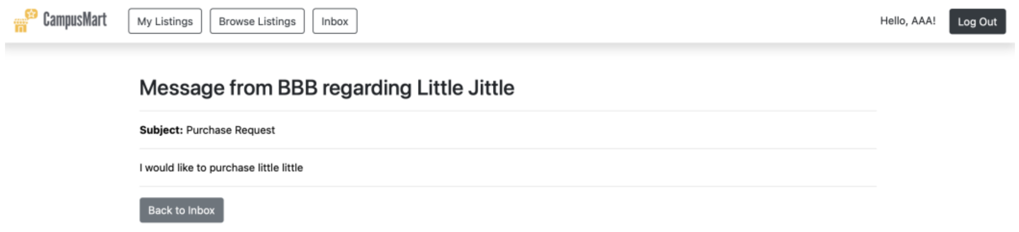
When a message is sent, a database entry is created inside **campusmart_message** with all the information as shown in the UML Diagram (*Figure 1*). Looking to the seller side, a user can see all their messages in the inbox as shown in *Figure 14*. These messages are found with a Django query to match the recipient id with the user.



Your Inbox
From: BBB
Subject: Purchase Request
Listing: Little Jittle

Figure 14 screenshot showing inbox

On this page, you can see all the messages and view them in more detail with the "View" button as shown in *Figure 15* which shows the full message.



Message from BBB regarding Little Jittle

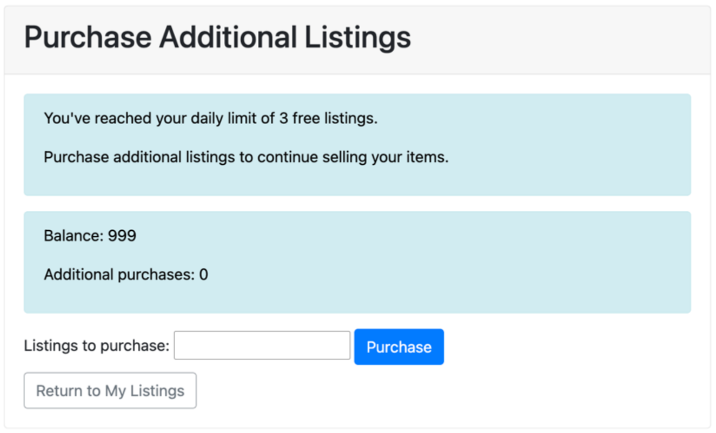
Subject: Purchase Request

I would like to purchase little little

Figure 15 screenshot showing view message

Buying More Listings (Feature 4.1)

This feature implements the purchasing of more listings on top of the daily listing. When creating a listing, first the daily listings are used, and then the purchased listings when there are no daily listings left. When using daily listings, the **campusmart_listingcounter** is updated and when using purchased listings, the **campusmart_purchasedlistingcounter** is updated. When the user has 0 daily listings and 0 purchased listings left, they are redirected to a purchase listings page which gives them the option to purchase listings using Kratos coins as shown in *Figure 16*.



Purchase Additional Listings

You've reached your daily limit of 3 free listings.

Purchase additional listings to continue selling your items.

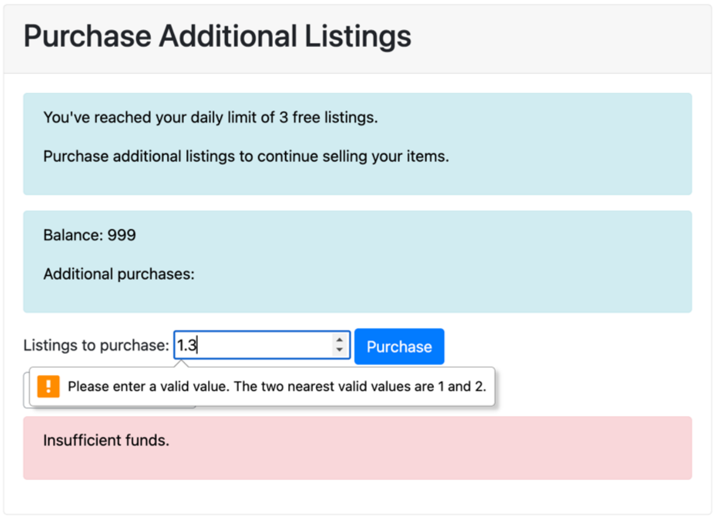
Balance: 999

Additional purchases: 0

Listings to purchase:

Figure 16 screenshot showing purchase listings form

This form verifies that the user is trying to buy a valid amount of listings (a positive integer). An error showing an invalid number is shown in *Figure 17*. Upon clicking purchase, the form sends a post request to the Django server. The server then calls the API and tries to make the payment for that many purchases. If the payment is successful, the purchased listings will be added. Otherwise, a corresponding error message will be displayed as shown in *Figure 17*.


Purchase Additional Listings

You've reached your daily limit of 3 free listings.

Purchase additional listings to continue selling your items.

Balance: 999

Additional purchases:

Listings to purchase: 1.3

Please enter a valid value. The two nearest valid values are 1 and 2.

Insufficient funds.

Figure 17 screenshot showing purchase listings errors

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?***

In using Django, we have used a lot of object-oriented programming as the main paradigm. I think that this is a good way to have created the project because each item and instance in our project can be represented well as an object as shown in the UML diagram. If we were to start this project from scratch now, we likely would have chosen the same paradigms as it encourages a model to reuse code we've already written with objects and templates instead of completely rewriting the code for every page. These paradigms chosen were helpful in developing the project as it modularized every single aspect of the project helping us as a team stay organized as different team members could work on different parts without worrying about breaking each others code.

2. ***What were the most intellectually challenging aspects of the project?***

One of the most challenging aspects of the project was designing the data models and understanding how they would interact. Conceptualizing the relationships and dependencies between different models required careful thought, especially to ensure that they accurately represented the application's logic and user flows. For instance, we wanted to make sure that the listings created for one user doesn't show up in the browse listing pagefor the same user.

3. ***What aspects of your process or your group's organization had the largest positive effect on the project's outcome?***

Our group had excellent communication and organization of our different roles which contributed to a streamlined and efficient process. We split up the different technical challenges quite fairly and communicated whenever we ran into an issue and tackled those together. This helped lead our group to finish the features effectively and ahead of schedule.