Team Member Full Name	NetID
Kam Williams	**kwilli33**

Persistent Storage Design

We are using a **SQLite database** to persist all application data. The database includes five main tables: CustomUser, Listing, Message, PurchaseHistory, and Auth. These tables collectively manage user accounts, product listings, direct messages, and track purchases of additional listings.

- The CustomUser table extends Django's default user model to include an extra_listings field, which keeps track of how many additional daily listings a user has purchased.
- The Listing table stores data for each product posted by users, including fields such as title, description, price, condition, status (Available/Unavailable), and a photo.
- The Message table logs buyer-seller communications, recording sender, receiver, message content, and timestamps.
- The PurchaseHistory table records when users buy additional listings, storing transaction metadata for auditing purposes.

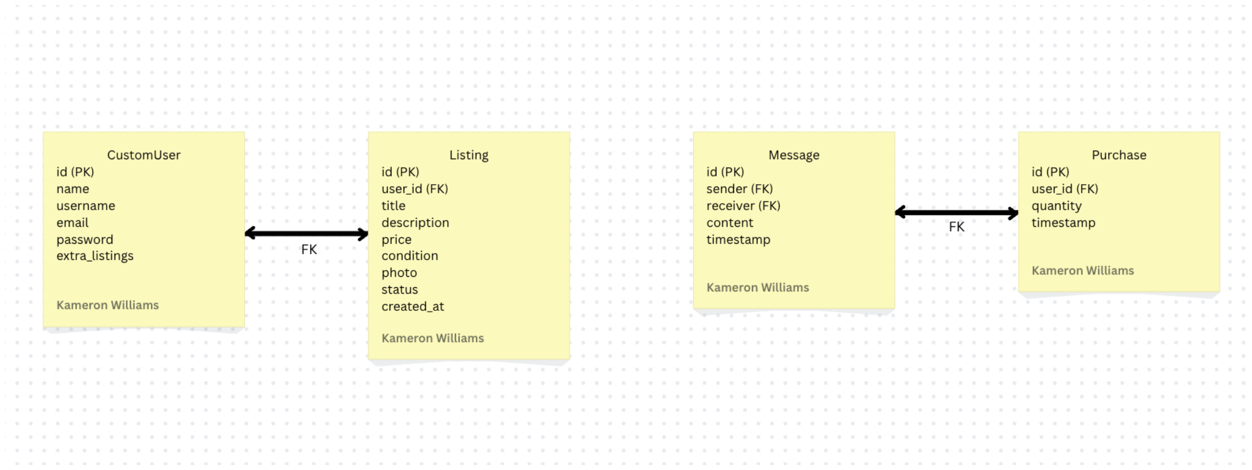
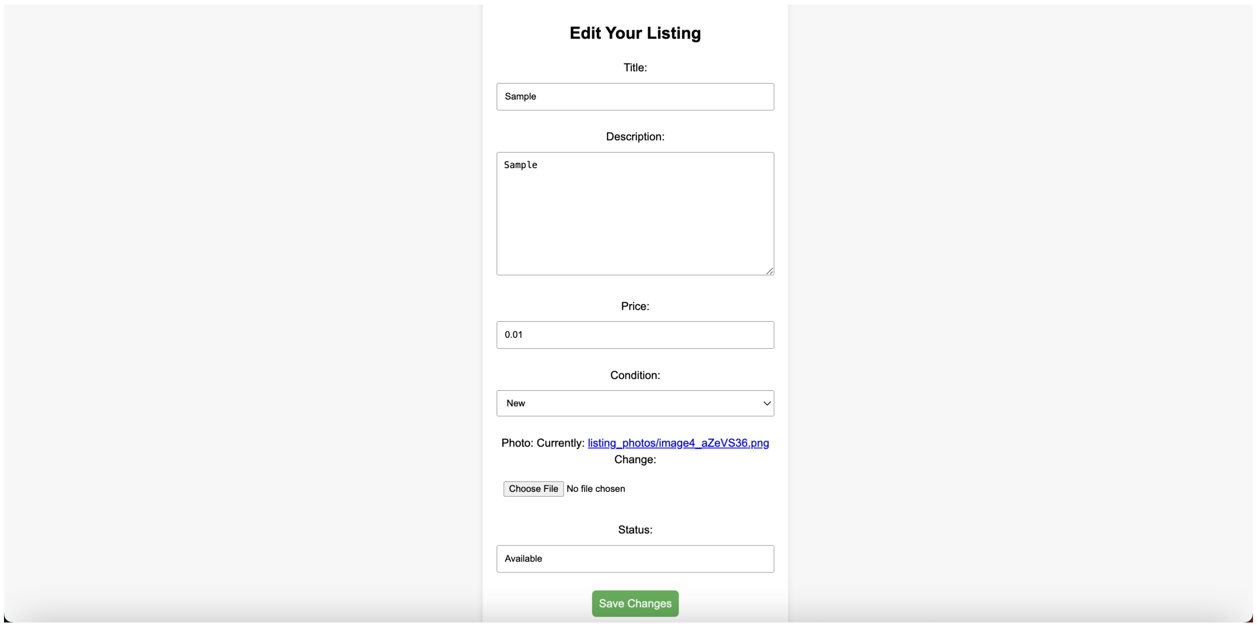


Figure 1 database schema

Demonstration of the Features

Feature 2.2 – Update Listings

Figure 2 shows the **Edit Listing** page, where users can modify their own listings. This feature allows users to change the title, description, price, condition, and update the product's status between **Available** and **Unavailable**. It enforces permissions so only the original owner of a listing can edit it.



Edit Your Listing

Title: Sample

Description: Sample

Price: 0.01

Condition: New

Photo: Currently: listing_photos/image4_aZeVS36.png

Change: Choose File No file chosen

Status: Available

Figure 2 Screenshot for feature 2.2

Feature 2.3 – Delete Listings

Figure 3 shows the **Delete Listing** confirmation page. Users are asked to confirm before deleting a product listing. This ensures that users do not accidentally remove listings. Like editing, delete functionality is permission-restricted: only the creator of the listing can delete it.

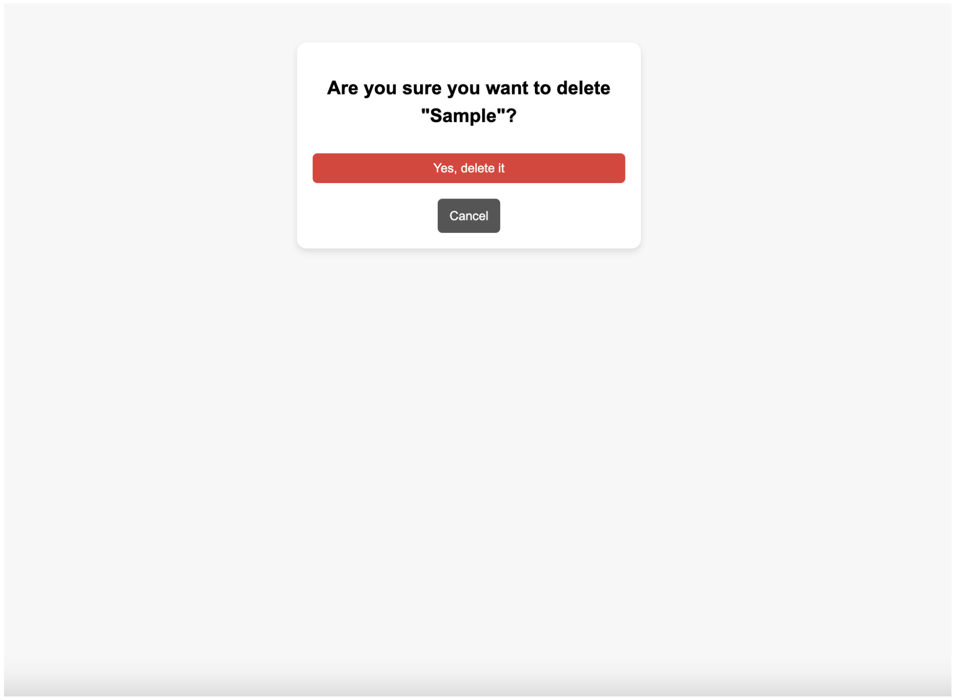


Figure 3 screenshot for feature 2.3

Feature 3.1 – View All Listings

Figure 4 shows the **Home Page**, where all available listings are displayed in a grid format. Each card shows the product image, title, and price at a glance. Listings are paginated to show 20 per page, with navigation links for "Previous" and "Next."

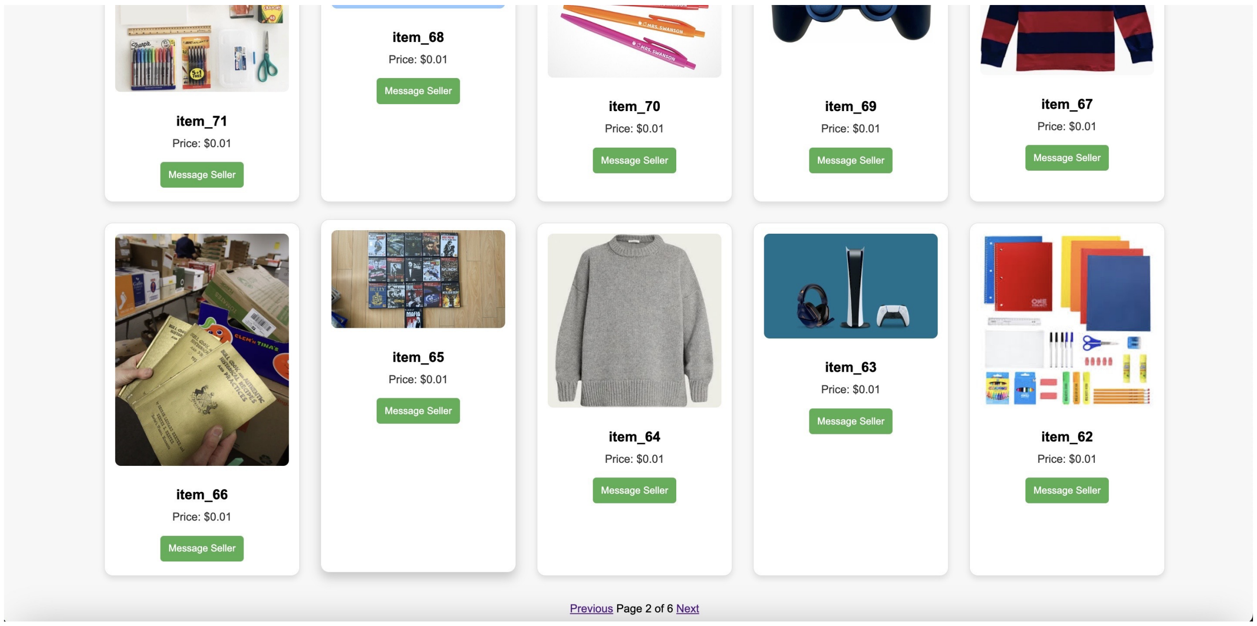


Figure 4 screenshot for feature 3.1

Feature 3.2 – Search Listings

Figure 5 shows the **Search Bar** at the top of the home page. Users can type keywords that match the title or description of a product to filter results. This feature helps buyers quickly find what they are looking for within the marketplace.

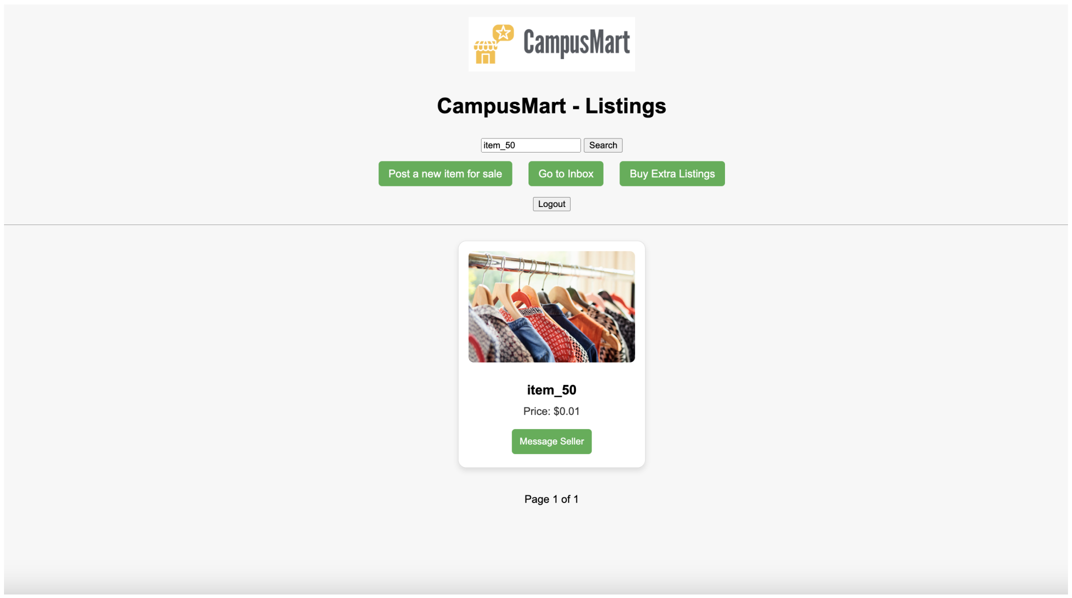
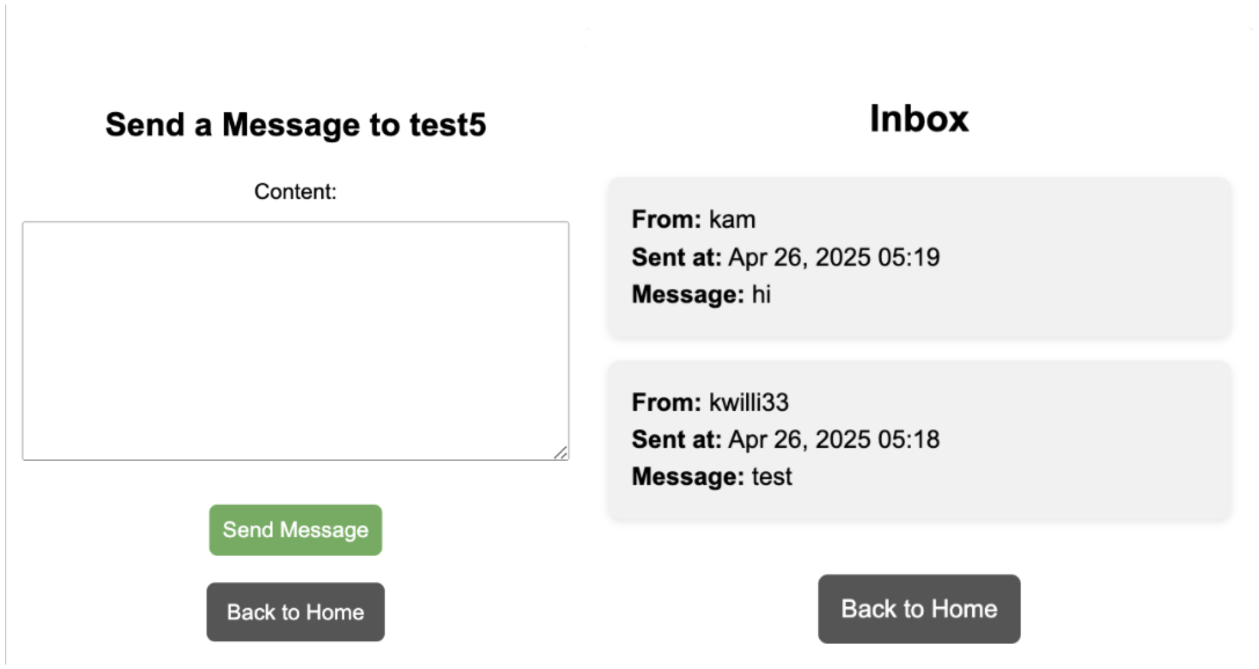


Figure 5 screenshot for feature 3.2

Feature 3.3 – Seller-Buyer Messaging
Figure 6 shows the **Inbox** and **Send Message** screens. Buyers can click "Message Seller" on a listing card to send a direct message to the seller. Messages are loaded when the inbox is accessed (no live chat required), and both sender and receiver can view message history.



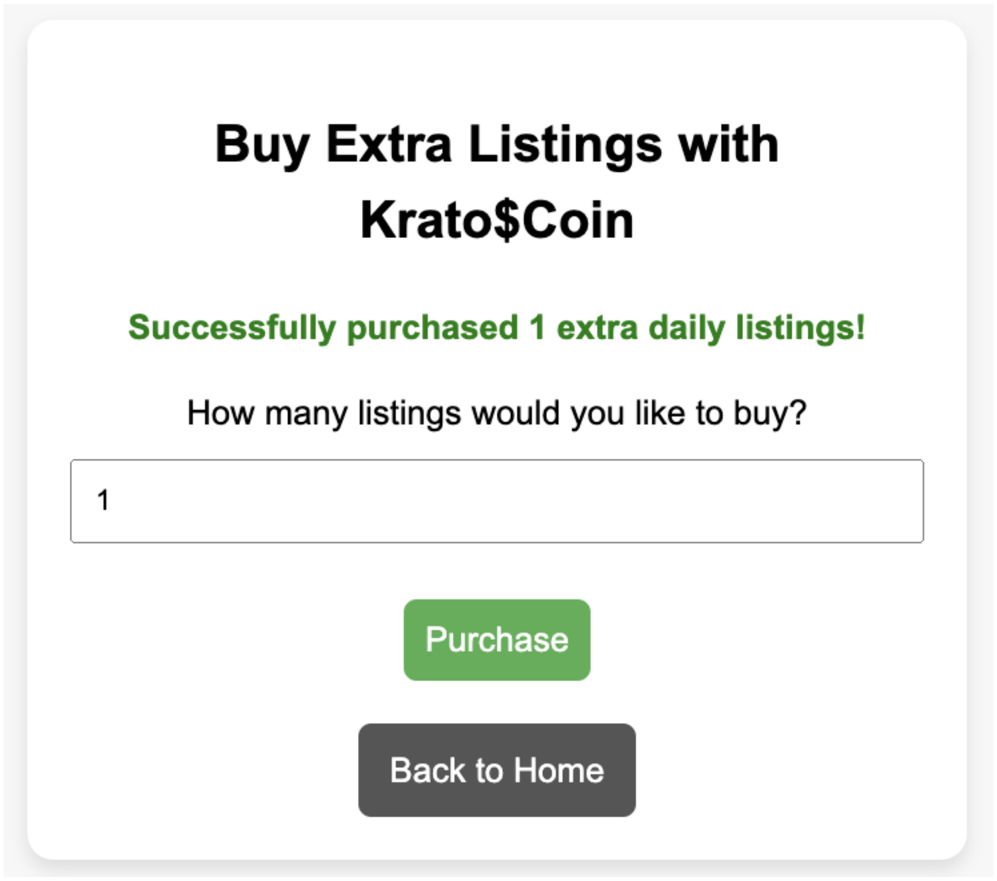
From: kam
Sent at: Apr 26, 2025 05:19
Message: hi

From: kwilli33
Sent at: Apr 26, 2025 05:18
Message: test

Figure 6 screenshot for feature 3.3

Feature 4.1 – Buying Extra Listings
Figure 7 shows the **Buy Extra Listings** page. Users can purchase additional daily listings using Krato$Coin. The page displays any success or error messages after attempting a purchase. Upon success, the system increases the user’s extra_listings

count, and a request is made to the external REST API to deduct the appropriate number of coins from the user's balance.


Buy Extra Listings with Krato$Coin

Successfully purchased 1 extra daily listings!

How many listings would you like to buy?

1

Figure 7 screenshot for feature 4.1

Project's Learned Lessons

In this section, you will reflect on the group's activities and performance through the entire project, and capture your reflections, observations and thoughts. It should address the following items, but feel free to expand on these in light of your group's unique experiences.

1. **What programming paradigm(s) have you chosen to use and why? Would you make different choices if starting over? Did the paradigm(s) help?**

   I used Object-Oriented Programming (OOP) through Django's models and procedural programming for view logic. Django's OOP structure made it straightforward to manage users, listings, and messages with clear relationships. If starting over, I would keep the same approach because it fit well and helped keep things organized and modular.

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

The most challenging part was integrating with the external REST API for purchasing extra listings. Handling token authentication and syncing the external coin balance with my app’s extra_listings field required careful debugging. Ensuring proper permissions—so only listing owners could edit/delete—was also tricky to get right.

3. **What aspects of your process had the largest positive effect on the project’s outcome?**

   Working incrementally and focusing on one feature at a time helped me stay organized. Using GitHub for version control and testing each feature thoroughly before moving on also ensured stability and a smoother development process.