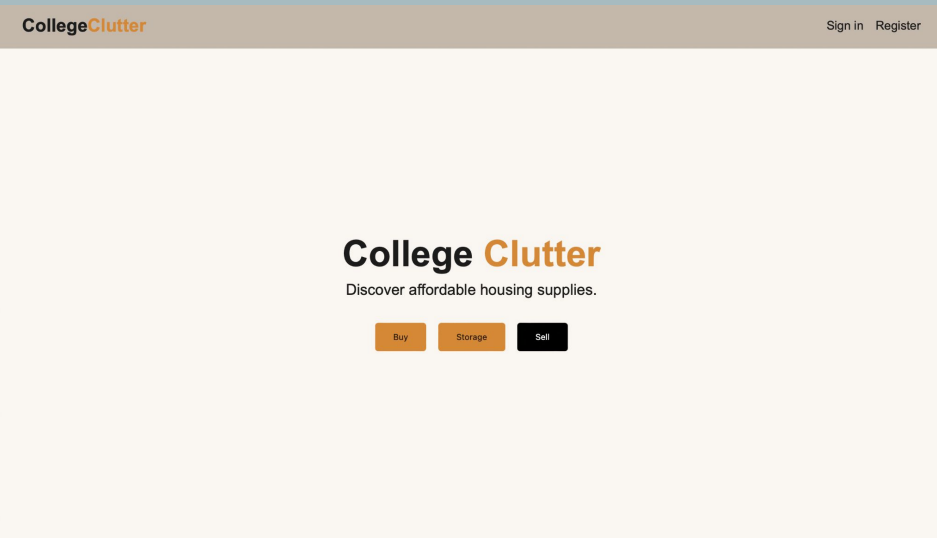


College Clutter



By:
Brandon Byrne
Sargam Nohria
Simran Lekhwani
(GROUP 36)

The project's vision.

Project's name: College Clutter

Description: Every year students buy new furniture every year to offset costs of storage and leading to tons of waste at the end of every semester. College Clutter is a website designed to connect students to other students and local storage facilities to collectively create optimal storage/selling opportunities. Waste and cost of living are both increasing at remarkable rate for students at Amherst. By offering a reasonable storage option we can give students an option to reduce costs between semesters and save on massive waste while improving local business outcomes during the down season.

Key functionalities: The website will be a lightweight platform that will use collective data from user inputs to generate optimal storage options with other users and local storage facilities. The user will have a simple drop-down interface to use commonplace data on furniture sets for ease of use and fast adoption. Users can list housing items for sale, buy items, or find storage for their items using the simple interface.

Tagged repository: <https://github.com/umass-byrneb/CS326-Group-36>

Milestone: #5

Issues on GitHub: <https://github.com/umass-byrneb/CS326-Group-36/issues>

The builders.

Brandon Byrne

Senior Computer Science Major | bbyrne@umass.edu

Background knowledge: C,C++, Python, Java, js and React

Other Interests: Hiking(46er) with my dogs 🐕🐕🐕

Reflection: I believe in this website as I personally have had these problems in the past and think it would improve student life at the end/beginning of the semester and decrease waste on campus

Role: backend development, front end development

Sargam Nohria

Senior Computer Science & Anthropology Major | snohria@umass.edu

Background knowledge: C, Python, Java, HTML, CSS

Other Interests: running, rock climbing, baking

Reflection: I am interested in increasing students' accessibility to affordable housing items and a more efficient exchange of existing resources. I believe this website will help people live well.

Role: wireframes/UI, front end developer

Simran Lekhwani

Senior Computer Science Major | slekhwani@umass.edu

Background Knowledge: C, C++, Python, Java, Go, JS, SQL

Other Interests: Dancing, Rock climbing, Reading

Reflection: As a student who has lived off campus and is graduating as well, this application would be very helpful to reduce some stress associated with moving and relocating by introducing a platform that allows one to communicate with other individuals who are interested in selling or buying furniture.

Role: Backend development, front-end development

Historical timeline.

Historical Timeline

Map out the timeline of your team's project



Brandon Byrne

Assigned Work Summary

Assigned Issues: #37, #40, #44

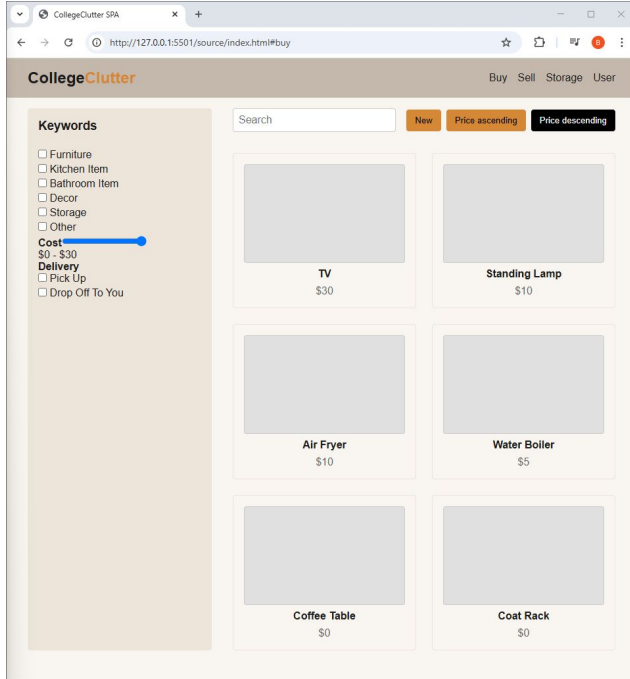
Commits: BuyComponent.js, RegisterComponent.js, LandingComponent.js, LoginComponent.js, SellComponent.js

Tasks completed: I transitioned the Registration and buy Component from a static single-page form to a multi-step interactive form with client-side validation, IndexedDB persistence, and simulated asynchronous server-side submissions.

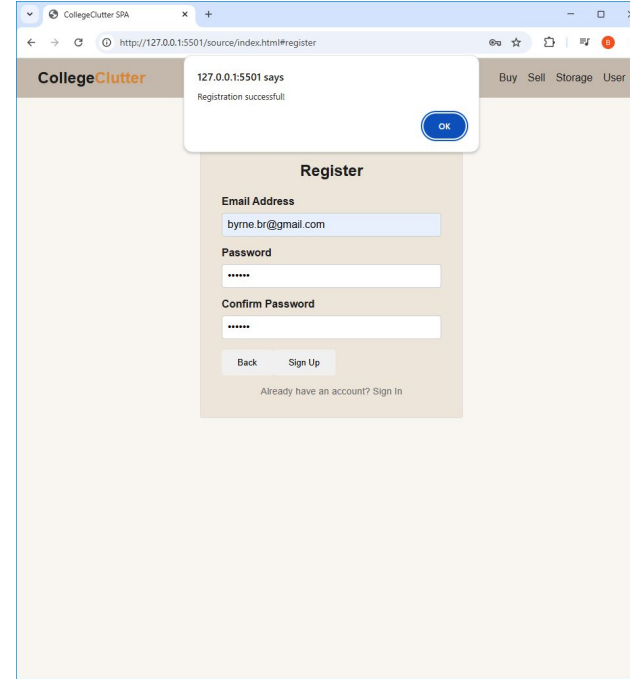
- Links to PRs closed on GitHub:
- <https://github.com/umass-byrneb/CS326-Group-36/pull/50>
- <https://github.com/umass-byrneb/CS326-Group-36/pull/49>
- <https://github.com/umass-byrneb/CS326-Group-36/pull/46>

Brandon Byrne

Screenshots and demonstration



Buy page



Registration Page

Brandon Byrne

Feature demonstration and code explanation

RegistrationComponent.js- The form is rendered in two steps, with conditional rendering based on the `currentStep` property in the component. Users move forward and backward through the form using “Next” and “Back” buttons. IndexedDB is utilized to persist user progress after each step, allowing form state to be maintained even after a page refresh. Upon component initialization, the previously saved data is retrieved and rep Each step has client-side validation to ensure data integrity. Upon successful validation, user input data is stored, and on the final submission, a fake asynchronous request simulates server interaction.

BuyComponent.js- The Buy Component was significantly enhanced with Dynamic filtering using category and delivery checkboxes. Interactive tag management corresponding to checkbox states. Real-time cost slider filtering with a visible price range indicator. Enhanced search functionality with intuitive partial-word matching. Comprehensive sorting options including newest postings and price sorting.

Brandon Byrne

Challenges and Insights

Obstacles:

Synchronizing checkbox states with dynamic tags presented initial complications, particularly regarding event handling loops. Implementing an intuitive partial-match search required careful iteration, balancing performance and user expectation. Handling combined filtering logic (cost, search, tags, and delivery) without significant performance degradation was complex.

Lessons Acquired:

Developed a strong understanding of event-driven programming, particularly how DOM events propagate and how to manage potential recursion or infinite loops. Learned to clearly separate filtering logic from rendering logic, leading to cleaner and maintainable code structures. Gained proficiency in creating intuitive UI interactions, understanding deeply the importance of user-friendly search and filtering mechanics.

Brandon Byrne

Future improvements & next steps

Future Improvements:

The current filtering and sorting mechanisms, while functional, might benefit from implementing more efficient data structures or algorithms, especially as the product list grows. There is potential to further abstract reusable logic into separate utility modules or classes, improving the maintainability and readability of the component. Learned effective use of IndexedDB for local state persistence, enhancing user experience significantly. Gained insights into asynchronous programming patterns and best practices, particularly in error management and user feedback mechanisms.

Collaborative development highlighted the importance of clear naming conventions and consistency in event handling strategies. Regular stand-ups and discussions helped in aligning the team on filtering logic and user interaction expectations. Continuous integration of team input throughout development cycles significantly improved the product's overall usability and robustness.

<https://github.com/umass-byrneb/CS326-Group-36/issues/52>

<https://github.com/umass-byrneb/CS326-Group-36/issues/51>

Sargam Nohria

Assigned Work Summary

Assigned Issues: #38 (sell page)

Commits: SellComponent.js, SellComponent.css

Tasks completed: front end work for the Sell page: creating a form that dynamically creates a preview to show the user what their sell post would look like as they type their data into the form, connecting this form to IndexedDB, creating an upload image function and adding that to the preview, ensuring a clean look to the webpage

- Links to PRs closed on GitHub:
- <https://github.com/umass-byrneb/CS326-Group-36/pull/47>
- <https://github.com/umass-byrneb/CS326-Group-36/pull/48>


Sargam Nohria

Screenshots & Demonstration

← → ↺ 127.0.0.1:5500/source/#sell ☆ 0 ⋮

CollegeClutter

Buy Sell Storage User



Choose File | natural-polib...-64_600.avif

Product Name

Tag (type of product you are selling)

Cost

Contact

Delivery


Product Description

Hello, this is the description.

Post

Preview your selling post:

Product Details



Coffee Table

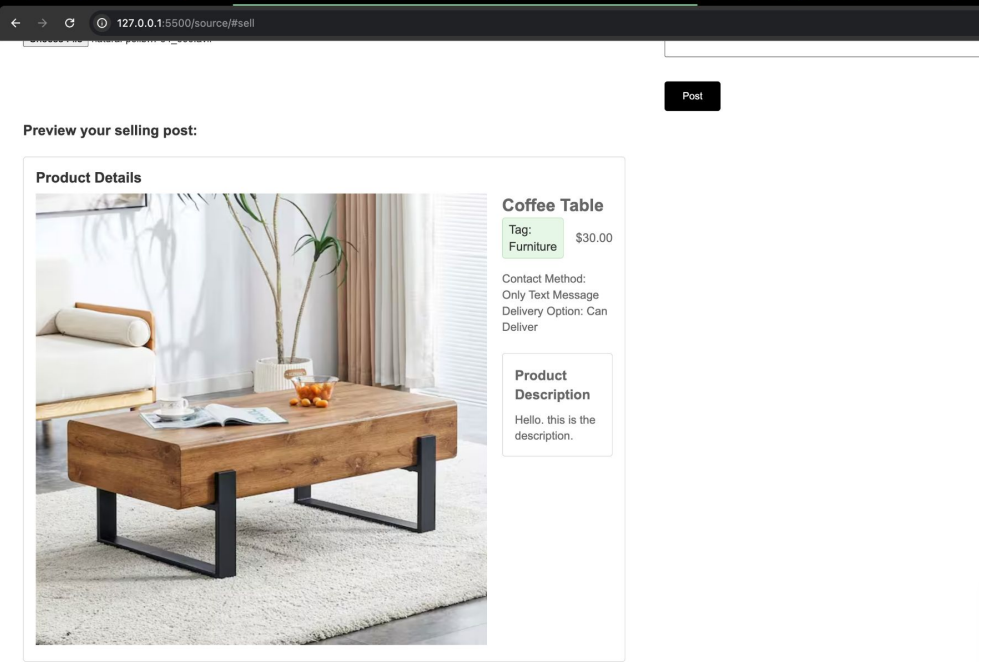
Tag: Furniture \$30.00

Contact Method: Only Text Message
Delivery Option: Can

Sell page part 1, form content dynamically updates in the preview portion

Sargam Nohria

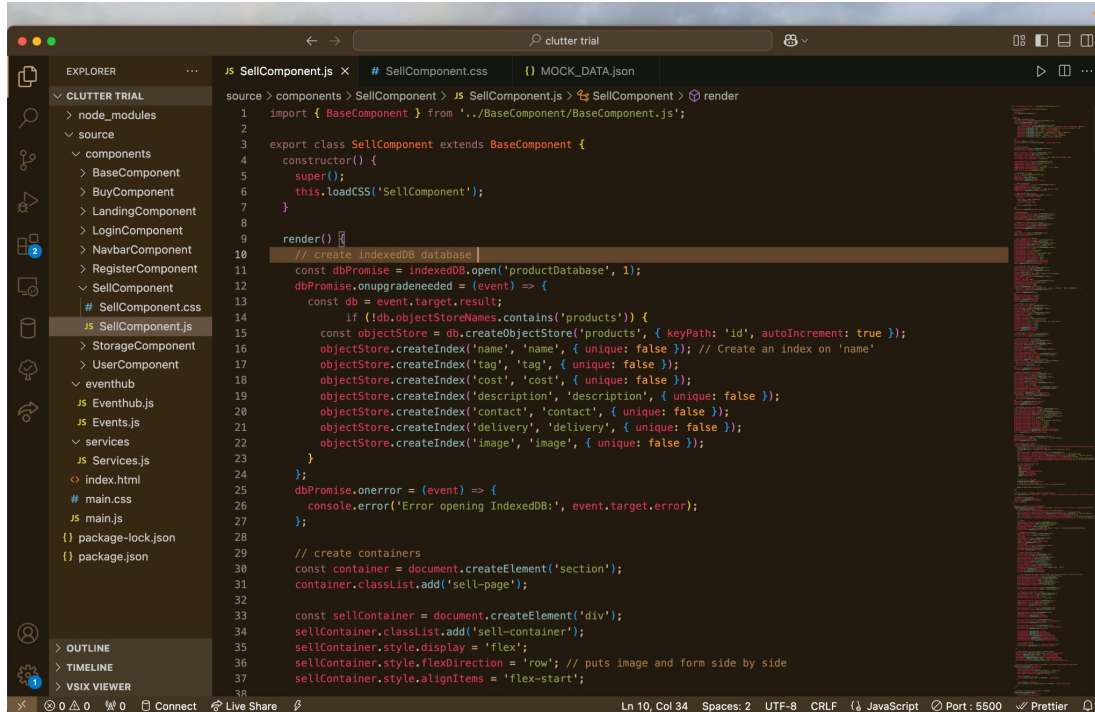
Screenshots & Demonstration



Sell page part 2, the preview part of the webpage

Sargam Nohria

Screenshots & Demonstration



```
source > components > SellComponent > JS SellComponent.js > SellComponent > render
1  import { BaseComponent } from '../BaseComponent/BaseComponent.js';
2
3  export class SellComponent extends BaseComponent {
4    constructor() {
5      super();
6      this.loadCSS('SellComponent');
7    }
8
9    render() {
10     // create indexedDB database
11     const dbPromise = indexedDB.open('productDatabase', 1);
12     dbPromise.onupgradeneeded = (event) => {
13       const db = event.target.result;
14       if (!db.objectStoreNames.contains('products')) {
15         const objectStore = db.createObjectStore('products', { keyPath: 'id', autoIncrement: true });
16         objectStore.createIndex('name', 'name', { unique: false }); // Create an index on 'name'
17         objectStore.createIndex('tag', 'tag', { unique: false });
18         objectStore.createIndex('cost', 'cost', { unique: false });
19         objectStore.createIndex('description', 'description', { unique: false });
20         objectStore.createIndex('contact', 'contact', { unique: false });
21         objectStore.createIndex('delivery', 'delivery', { unique: false });
22         objectStore.createIndex('image', 'image', { unique: false });
23       }
24     };
25     dbPromise.onerror = (event) => {
26       console.error('Error opening IndexedDB:', event.target.error);
27     };
28
29     // create containers
30     const container = document.createElement('section');
31     container.classList.add('sell-page');
32
33     const sellContainer = document.createElement('div');
34     sellContainer.classList.add('sell-container');
35     sellContainer.style.display = 'flex';
36     sellContainer.style.flexDirection = 'row'; // puts image and form side by side
37     sellContainer.style.alignItems = 'flex-start';
38
```

Code snippet

Sargam Nohria

Code and UI explanation

I created the front-end features for the 'Sell' page and functionality of the website. This included allowing the user to upload an image to the webpage to show what they are selling, creating a form to allow the user to input product details. These pieces of data get dynamically updated and displayed in a "preview your sell post" section of the website that appears below the form. The data also gets saved as an object into an IndexedDB database, and the sell page itself is a part of a multiview UI.

Sargam Nohria

Challenges and Insights

- Figuring out in what way it makes sense to implement IndexedDB on the webpage, and implementing it
- Figuring out how to properly organize elements on the page with flex, so that the preview section looks clean
- Better accustomed to using GitHub

Sargam Nohria

Future improvements & next steps

- Connecting the User page to a backend database to store user information
 - Allows the sell and store buttons to have functionality
- Same with using a backend database to store and pull product items to sell, and connecting that to the sell page

Simran Lekhwani

Assigned Work Summary

Assigned Issues: #39 (Storage Page), #43 (Login Page)

Commits: Restructured the code base based on course requirements and added a color palette, StorageComponent.js, StorageComponent.css, Landing.js Events.js, Service.js, StorageListingService.js, StorageListingFakeService.js, utility/fetch.js, mock Storage listings data

Tasks completed:

Implemented the storage page which renders the current listings of spaces for sale with the addition of IndexedDB persistence and user interactivensess through mock backend server.

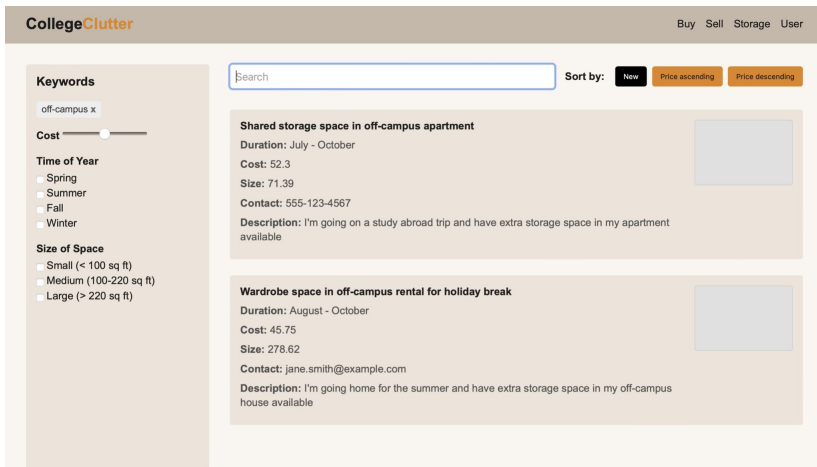
The user can filter through the listing using the search bar which generates tags (visible on the side), through the filters on the sidebar or through the buttons next to the search bar (New, Price Ascending, Price Descending). The addition of the user input enforces dynamic content updates where the data is being fetched from IndexedDB or the server (further explained in later slides).

- Links to PRs closed on GitHub: <https://github.com/umass-byrneb/CS326-Group-36/pull/53>

Tasks Remaining: The Login page had a lot of the features implemented from the previous milestone. Since the focus of this milestone was not to implement authentication, this task didn't take much priority. Few features such as user input validation could have been added, but again this was not the focus of this milestone, and such features can be implemented in a more efficient manner with an actual server.

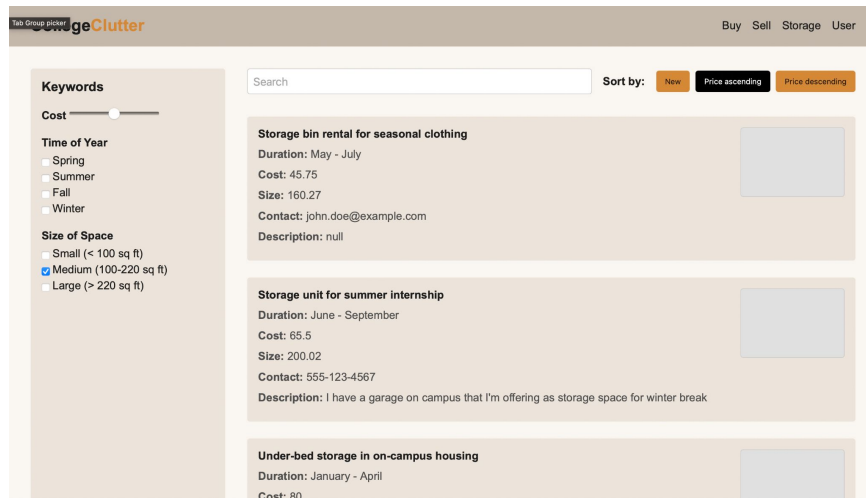
Simran Lekhwani

Screenshots



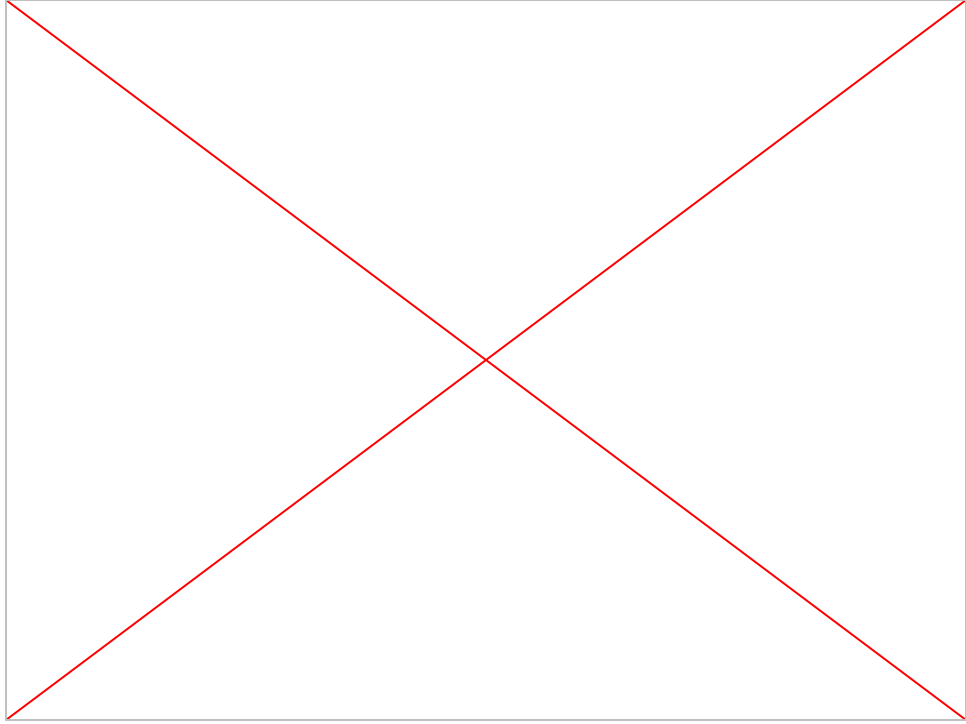
Search is filtered based on an 'off-campus' tag

Search is filtered based price (ascending order as indicated by the highlighted button) and storage space size (between 100-220 sq ft range)



Simran Lekhwani

Demo



Simran Lekhwani

Code & UI explanation

Data collection:

Data from the mock server is fetched and stored in IndexedDB under a table StorageListings. This was done to ensure user can access storage listings offline and filter through them, but also to avoid numerous API calls for each user request and the possible consequence of overflowing the network.

User Input/Interaction:

The user could sort the listing based on how new they were and their prices, using the buttons at the top of the listings. As suggested this required sorting of the arrays. Any activity through the filters on the side bar are enforced by fetching the current listing in the DB and filtering them based on the selected filter.

The user could also search using keywords which could be entered through the search bar. As the user enters each keyword, it dynamically appears on the page while in the backend, it fetches from the listings stored in the DB. If a tag (or keyword) is added, it filters the current listing stored in the DB. If a tag is removed, it fetches from the server to get the previously unfiltered list.

Simran Lekhwani

Challenges and Insights

- I think this milestone was a great exercise for asynchronous programming. Having to deal with the mock server, fetching data from it and storing it in the DB was a very helpful practice as I encountered various roadblocks with resolving promises and displaying the correct input.
- Another challenging aspect was curating the mock data itself and accommodating for human errors in user input. There can be a lot of variability in the way users can post their storage space or in their input in the search bar (for example, the cost can be in term of per month or the total cost, or there can be a spelling mistake in their keyword), which can make filtering challenging. This is something I believe I would like to continue working on during the remaining milestones, as it is very probable challenge in real-life.

Simran Lekhwani

Future improvements & next steps

Data architecture:

<https://github.com/umass-byrneb/CS326-Group-36/issues/55>

- Look into a feasible storage data type
- Make an implementation decision about flexibility in user input (any restrictions in user input - such as input type - to aid in filtering)

Accommodate for user error in search bars and enhance search algorithm:

<https://github.com/umass-byrneb/CS326-Group-36/issues/56>

- Would be a great feature to work on to ensure full functionality of the website
- Control user error through input restriction or accommodate them in the code
- Increase the efficiency of the filtering algorithm (especially for filtering through tags/keywords)

User interactive and input validation for the login page:

<https://github.com/umass-byrneb/CS326-Group-36/issues/57>

- Check for incorrect username and password once an authentication system has been established
- Forget password functionality - another feature