

Study on Campus

CS 326 Group 11

Milestone #5 | April 11th, 2025

[Presentation link](#)

Project Overview

Working with a study partner or group can provide increased understanding, accountability, routine, camaraderie, and productivity. However, finding an effective one can be difficult to students for a multitude of reasons: large classes, social anxiety, mismatched goals, unclear expectations, and lacking a method of contact. **As a solution, we propose a web application that offers a platform to help students more easily connect with each other, find study locations that promote productivity, and organize or find study groups.**

Key Features

- **Study group posting:** Students can post open invites for study groups, along with their goals, preferences, and location. Users can browse and filter for groups that match their needs.
- **Communication between users:** Users can discuss meeting details and coordinate study group plans through public comments on invite posts and/or private messages.
- **User-reported location crowding score:** Users can rate how crowded popular study spots are to help others make more informed decisions.

Team

Erika Elston (Project Manager)

Primary Contributions: [User profile screen](#) UI implementation, [location browsing screen](#) frontend feature implementation

Julia Farber (Time Keeper)

Primary Contributions: [Post browsing screen](#), [post viewing screen](#) frontend feature implementation

Anastasia Isakov (Project Organizer/Documentation Lead)

Primary Contributions: [Post creation screen](#), [login/signup screens](#) frontend feature implementation

Ashley Kang (Quality Control)

Primary Contributions: [Settings screen](#) frontend feature implementation

M	T	W	Th	F
<div> <div>Historical Development Timeline</div> <div>previous Milestone timeline</div> </div>		3/26	3/27	3/28 <div> Deadline Milestone #4 (Interface Mockup) Development All initial interface mockups merged to main </div>
3/31 <div> Team meeting Discuss Milestone #5 Development Began location browsing page UI implementation </div>	4/1	4/2	4/3 <div> Team meeting Discuss UI standardization, site structure/navigation Development Began user profile page UI implementation </div>	4/4 <div> Team meeting Stand-up #3, discuss <u>features and complexity</u> </div>
4/7 <div> Team meeting Communicated goals and drafted feature points/descriptions Development Updated HTML/CSS, began implementing post creation functionality. Updated css/html for post browsing and post viewing. </div>	4/8	4/9 <div> Development Began implementing settings functionality. Began implementing post browsing rendering and search bar with filter. </div>	4/10 <div> Development Updated HTML/CSS, began implementing login/signup functionality </div>	4/11 <div> Deadline Milestone #5 (Front-End Design & Implementation) Team meeting Stand-up #4 Development Opened PRs for post creation, login, signup, settings, post browsing, post viewing, location browsing, and user profile </div>



Erika Elston

Work Summary

Key Issues:

- [Location Browsing and Crowding Score Reporting Screen](#) (UI)
- [Location and Crowding Information Data Structure](#) (Data)
- [Crowding Score Report Data Structure](#) (Data)
- [User Profile Screen](#) (UI)
- [User Profile Data Structure](#) (Data)

Key Commits:

- [b031720](#) (3/30): Initial setup for Milestone 5. Add frontend subfolders, organize Milestone 4 pages, & init package.json.
- [4d10f3a](#) (4/3): Add initial User Profile data structure and initialize Post data structure.
- [1814604](#) (4/3): Initial UI implementation for User Profile screen with UserProfileComponent.js and UserProfileComponent.css. Aimed to replicate Milestone 4 mockup (i.e., no additional feature implementation, yet).
- [e5b34ba](#) (4/11): Initial UI implementation for Location Browsing Screen with LocationBrowsingComponent.js, LocationCardComponent.js, and corresponding CSS files. Implemented fake fetch() for location data retrieval.
- [4548edb](#) (4/12): Implementation of preliminary features for Location Browsing Screen: card filtering (event handling, DOM), card expansion (event handling, multi-view UI).

Pull Requests:

- [#40](#) (3/30): Add frontend subfolders, organize pages from Milestone 4, and init package.json.
- [#45](#) (4/3): Add initial User Profile data structure, so team can start with same basic User structure. Init Post data structure.
- [#62](#) (4/11): Implemented UI implementation for user profile screen frontend.
- [#63](#) (4/12): Implemented functionality for location browsing screen frontend.

Feature Demo: Location Browsing

Branch: [12-location-browsing-and-crowding-score-reporting-screen](#)

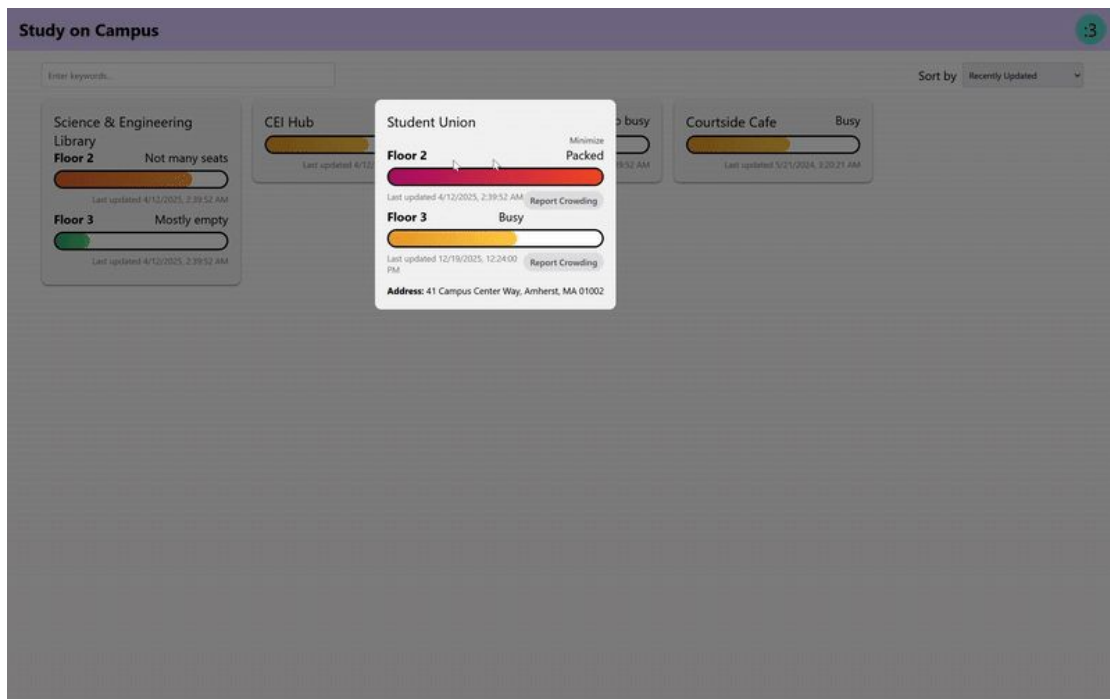
Advanced (est. 4pts): JS + A (DOM), C (Event Handling, User Interaction), D (Multi-view UI), F (Asynchronous Data Handling)

This page allows users to view study locations and their corresponding crowding scores. Users can filter using keywords or one of three sort options. Clicking on a location card presents users with additional location details and an option to report the crowding score for the location they selected.

Sub-features: (1pt) Location data retrieval using `fetch()`, (1pt) location card sorting, and (2pts) expanded card views

Next Steps: The final sub-feature for this page will enable users to contribute to the crowding score report via the “Report Crowding” button that appears on the expanded card view. This will involve additional event handling, DOM, and modification of corresponding location data.

I would also like to refine the filtering functionality, as the keyword filtering and dropdown option filtering do not yet work in conjunction.



Code

(1pt) Location retrieval using fetch()

I implemented **basic asynchronous data handling** by using `fetch()` to retrieve location data. For this milestone, I retrieved the data from a mock JSON file. For future milestones, this feature may be modified to rely on our backend database.

The `LocationBrowsingComponent` includes an asynchronous function to retrieve the location data. Once the location data is saved, each location card is rendered with `LocationCardComponent` using the data retrieved.

Challenges:


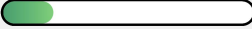
- **Data representation:** I initially had difficulty determining an efficient and intuitive way to represent single-floor and multi-floor locations, because the HTML structures varied. I decided to use a "type" attribute to differentiate between the two, and simply utilize conditionals to render each card appropriately.
- **Fake fetch():** I had some issues figuring out the fake `fetch()` function, but eventually figured out a suitable implementation.

```
// fetch location data from mock server (local JSON file)
async #getLocations() {
  try {
    const response = await fetch("http://localhost:3000/lib/data/MockLocations.json");
    if (response.ok) {
      const json = await response.json();
      const data = await JSON.parse(json);


      if (Array.isArray(data) && data.length > 0) { // check data format / if location data exists
        this.#locationsData = data; // save location data
        return data;
      } else {
        throw new Error("no locations found");
      }
    } else {
      throw new Error(response.status);
    }
  } catch (error) { // error handling
    console.log(`Error fetching location data: ${error}`);
  }
}
```

```
ok: true,
status: 200,
statusText: "okay",
url: "http://localhost:3000/lib/data/MockLocations.json",
json: async () => JSON.stringify(MockLocations), // temp solution to json fetching
text: async () => "This is a mock response",
};
```

```
{
  "name": "Science & Engineering Library",
  "address": "740 N Pleasant St #273, Amherst, MA 01003",
  "type": "Multi-Floor",
  "floors": [
    {
      "name": "2",
      "reports": [
        {
          user_id: "user123",
          score: 4,
          timestamp: Date.now()
        }
      ]
    },
    {
      "name": "3",
      "reports": [
        {
          user_id: "user123",
          score: 1,
          timestamp: Date.now()
        }
      ]
    }
  ]
},
```

Science & Engineering
Library
Floor 2 Not many seats

Last updated 4/12/2025, 3:41:48 AM
Floor 3 Mostly empty

Last updated 4/12/2025, 3:41:48 AM

```
{
  "name": "CEI Hub",
  "address": "100 Natural Resources Rd, Amherst, MA 01003",
  "type": "Single-Floor",
  "reports": [
    {
      user_id: "user789",
      score: 3,
      timestamp: Date.now()
    },
    {
      user_id: "user789",
      score: 3,
      timestamp: Date.now()
    },
    {
      user_id: "user123",
      score: 2,
      timestamp: Date.parse(new Date("April 10, 2025 12:34:05"))
    }
  ]
},
```

CEI Hub Busy

Last updated 4/12/2025, 3:41:48 AM

Code

(1pt) Location card sorting & (2pts) expanded card views

I used **dynamic content updates** and **event handling** to implement location card sorting. When a user types in the keyword field or selects a sorting option, the location browsing container is re-rendered accordingly. This provides users with a way to more easily find a study location that suits their needs.

I used **event handling** and **multi-view UI** techniques to implement the expanded post views. Clicking a location card brings the card to the center of the window and dims the rest of the page. The expanded card displays additional location data and will allow the user to report on the location crowding.

Challenges:

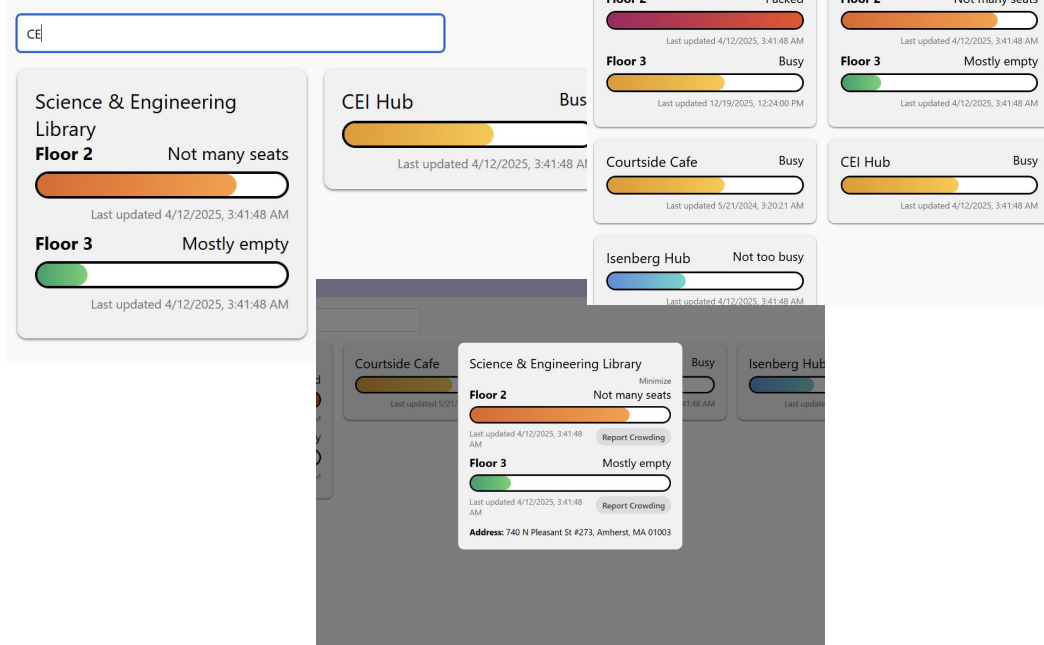
- **Sort by options:** Pulling the needed information to sort the cards was a bit tedious, because the location data for Single-Floor and Multi-Floor buildings are structured differently. I am considering making adjustments to the renderCards method for readability.
- **Expanded card view:** As I didn't mockup the expanded card view in Milestone 4, I came across some minor CSS issues that I will need to address: "Minimize" button appears below the location title for multi-floor buildings; card width affects some of the crowding score bars.

```
// filter and re-render location cards based on given query (string)
#filterLocationCardsByQuery(query) {
  const key = query.toLowerCase(); // convert search query to lowercase

  // TODO: determine if filter may benefit from extra keyword matches (other than just title)
  const matches = this.#locationsData.filter(location => { // filter locations by title match
    return location.name.toLowerCase().includes(key);
  })

  this.#renderCards(matches); // render filtered cards only
}

// TODO: account for filter by query AND sort option -> may need to rework both functions a bit
#sortLocationCards(sortOption) {
  console.log(sortOption);
  this.#renderCards(this.#locationsData, sortOption);
}
```



Challenges and Insights

Obstacles: Early on, I had trouble determining the “correct” way to represent certain data structures, especially without the guidance of having a backend implemented. Getting stuck on the details made it difficult for me to feel like I could progress with a couple of the UI components I was working with.

In hindsight, I think if I had stepped back from the multiple, complicated solutions I was trying—and moving forward the best I could with something simpler—, I could have been somewhat more productive this milestone. The data structures I ended up with weren’t quite as difficult as I originally thought the implementation would’ve required for something functional. There are probably more efficient ways to represent single- and multi-floor buildings, for example, but I still have an implementation that works and is somewhat easy to understand regardless.

Insights: I appreciate being able to meet regularly with the team, as well as their communication and patience! I think it’s been great to see the web app come together over the past few weeks, and I’m excited to learn more as we move forward to implementing the backend.



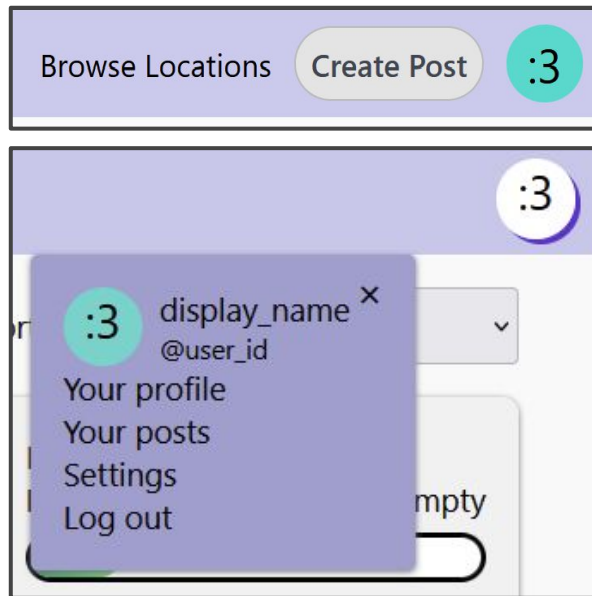
Future Improvements and Next Steps

Location Browsing and Crowding Score Reporting Screen (#12): I'd like to clean up my code, adjust CSS styling details, and complete the final sub-feature for this page (crowding score reporting) as we begin working on the backend implementation. Given time, I'd also like to implement the Google Maps API embed for the expanded post view (#42).

User Profile Screen (#6): I replicated the profile mockup from Milestone 4, but would ideally like to implement some minor profile editing functionality from this page. This would involve multi-view UI techniques and modifying the stored user settings.

Navbar Dropdown Menu (#47): I've begun working on the dropdown menu and styling for the navigation bar elements. I'd like to complete these elements to enable additional connections across the web app.

Visual Preferences and Interface Customization (#36): While Milestone 6 does not involve authentication, I think that some of the visual preference functionality could be implemented with IndexedDB. Given time, I'd like to figure out some of the customization options.





Anastasia Isakov

Work Summary

Issues worked on in this Milestone:

- [Post Creation Screen](#) (UI)
- [Login/Signup Screens](#) (UI)

Commits:

- [dcb00e0](#) (4/11): Changed font-size from *{} to html{} in main.css.
- [4d01afc](#) (4/11): Changed to single index.html with multi-view for login/signup page, added JavaScript functionality.
- [f436a71](#) (4/11): Changed tag menu to allow users to input their own tags which render on screen.
- [830315d](#) (4/11): Added JavaScript and IndexedDB functionality for post submission.
- [1ee7bbc](#) (4/11): Added period to post title error message.
- [16b3aa9](#) (4/11): Fixed redirect bug for sign up.

Pull Requests:

- [#58](#) (3/28): Improved HTML/CSS Implementation of Post Creation with added JavaScript Functionality.
- [#60](#) (3/28): Improved HTML/CSS Implementation of Login/Signup with added JavaScript Functionality.

Feature Demo: Post Creation

Branch: [19-post-creation-screen](#)

Basic/Advanced (4pts): JS + DOM, Form Validation and User Feedback, Event Handling and User Interaction, IndexedDB and Persistence

Description: This feature enables users to create and submit a post by entering a post title (required), post location, and post description. Users can also click an “Open Tagging Menu” button that opens a dynamic tagging menu. Within this menu, users can input tags, which are rendered in real-time onto the page. Upon form submission, all post content—including the title, location, description, and tags—is saved to IndexedDB, and the user is redirected to the Post Browsing Screen whether they click post or cancel.

Reason: This feature uses DOM manipulation for real-time tag management and rendering. The user is provided real-time feedback if the Post Title is missing, preventing form submission. Event listeners are used to handle form submission, tagging menu toggling, and tag removal. Finally, on post submission, posts are saved to IndexedDB, allowing for data persistence.

Key Code: Tagging Menu

The tagging logic for post creation is encapsulated within its own section, responding to user events such as clicking a button or hitting enter.

Adding and removing tags updates the DOM in real time, and the menu toggle allows for a cleaner UI.

Challenges included general formatting so that the tagging menu wouldn't interfere with the other post content, as well as figuring out how to remove tags.

Study on Campus

Close Tag Menu

Cancel

Post

UI Demo

```
const tags = [];  
  
// Toggle the tag input view  
addTagsBtn.addEventListener('click', () => {  
  const isHidden = tagInputView.classList.toggle('hidden');  
  addTagsBtn.textContent = isHidden ? 'Open Tag Menu' : 'Close Tag Menu';  
  tagInput.focus();  
});  
  
// Add tags  
tagInput.addEventListener('keydown', (e) => {  
  if (e.key === 'Enter' && tagInput.value.trim() !== '') {  
    e.preventDefault();  
  
    const tagText = tagInput.value.trim();  
    const tag = document.createElement('div');  
    tag.classList.add('tag');  
  
    const tagLabel = document.createElement('span');  
    tagLabel.textContent = tagText;  
  
    const removeBtn = document.createElement('span');  
    removeBtn.textContent = 'x';  
    removeBtn.classList.add('remove-tag');  
    removeBtn.title = 'Remove tag';  
  
    tag.appendChild(removeBtn);  
    tag.appendChild(tagLabel);  
  
    tagList.appendChild(tag);  
    tagInput.value = '';  
  
    tags.push(tagText);  
  }  
});  
  
// Remove tags  
tagList.addEventListener('click', (e) => {  
  if (e.target.classList.contains('remove-tag')) {  
    const tag = e.target.parentElement;  
    const tagText = tag.querySelector('span').textContent;  
  
    const index = tags.indexOf(tagText);  
    if (index !== -1) {  
      tags.splice(index, 1);  
    }  
  
    tag.remove();  
  }  
});
```

Feature Demo: Login/Signup

Branch: [22-log-in-and-sign-up-screens](#)

Basic/Advanced (4pts): JS + Form Validation and User Feedback, Event Handling and User Interaction, Multi-view, Asynchronous Data Handling

Description: This feature allows users to either log in or sign up by toggling between two views—Login and Signup. Both views require users to enter a valid email and a password with a minimum length of 8 characters. As users interact with the input fields, real-time validation provides immediate feedback when criteria are not met. Upon login, user credentials are validated against a local JSON file acting as a mock server. If credentials are valid, the user is redirected to the Post Browsing Screen; if not, an error message appears. During signup, the system checks if the email already exists, displaying an error message if so. If the signup is successful, the user is redirected to the Profile Screen.

Reason: This feature uses real-time form validation to provide immediate feedback on email format and password length, preventing form submission until inputs meet required criteria. Event listeners are used to handle form submissions, toggle between login and signup views, and monitor input changes for validation. It uses a multi-view UI structure by dynamically switching between login and signup screens without a page reload. Finally, it performs asynchronous data handling by fetching user data from a JSON file acting as a server and routing the user based on server response.

Key Code: Multi-View

The first couple of blocks handle the multi-view, similarly to the code in class. The additional `clearInputFields` function is used to ensure that the error messages are cleared, as well as the current input.

The last couple of blocks handle real-time user feedback displaying the appropriate message and removing it once the input is valid.

Study on Campus 3

Log In

Email:

*Must be a valid email.

Password:

Log In

New User? Sign Up Here!

UI Demo

Challenges included restructuring parts of the HTML to allow for cross-view functionality.

```
//multi-view functionality
function navigate(viewId){
  document.querySelectorAll(".view").forEach((view) => {
    view.style.display = "none";
  });
  document.getElementById(viewId).style.display = "block";
  clearInputFields(viewId);
}

//clears inputs and error messages when switching between pages
function clearInputFields(viewId){
  const emailInput = document.querySelector(`#${viewId} input[type="text"]`);
  const passwordInput = document.querySelector(`#${viewId} input[type="password"]`);
  const emailErrorSpan = document.querySelector(`#${viewId} .email-error-message`);
  const passwordErrorSpan = document.querySelector(`#${viewId} .password-error-message`);
  const enterErrorSpan = document.querySelector(`#${viewId} .enter-error-message`);

  if (emailInput) emailInput.value = "";
  if (passwordInput) passwordInput.value = "";
  if (emailErrorSpan) emailErrorSpan.style.display = "none";
  if (passwordErrorSpan) passwordErrorSpan.style.display = "none";
  if (enterErrorSpan) enterErrorSpan.style.display = "none";
}

//event-handlers to switch between login and signup pages
document.getElementById("new-user").addEventListener("click", (e) => {
  e.preventDefault();
  navigate("signup");
});

document.getElementById("old-user").addEventListener("click", (e) => {
  e.preventDefault();
  navigate("login");
});

navigate("login");

//validating email format
function validateEmail(email){
  const emailRegex = /^[^\\s@]+@[^\\s@]+\\.([^\\s@]{3,})$/;
  return emailRegex.test(email);
}

//show invalid email/password messages
function showMessage(inputElement, messageClass, isEmail){
  const messageElement = inputElement.parentNode.querySelector(`.${messageClass}`);
  if (isEmail){
    messageElement.style.display = validateEmail(inputElement.value) ? "none" : "block";
  }
  else {
    messageElement.style.display = inputElement.value.length >= 8 ? "none" : "block";
  }
}
```

Challenges and Insights

An unexpected challenge this week was that my laptop turned off and wouldn't turn back on again, and so I had to get it repaired. I hadn't pushed any of my code from my local repository, so I had to essentially start from scratch on a new computer or lose time. This definitely made me realize the importance of periodically pushing to the branch on Github, as I only really do that once my code is done. In terms of the code itself implementing the mock fetch with the JSON file acting as a server was confusing and difficult. However, I feel like I understand more of how the different components interact with each other now, and I gained more insight into how to use the eventHub.

Working with my team for this milestone was productive. We met fairly regularly within the two weeks and discussed any questions on Discord.




Future Improvements and Next Steps

#32 Password Reset Screen(s): Turned the Login and SignUp screens into a multi-view, creating the basic format for handling a future password reset screen. Login also successfully accesses a user through a mock fetch and json file, so implementing the password reset itself will work based on that.

#35 UI Standardization: We've made progress on our UI Standardization, with a single main.css that is imported into different styles.css files. With added javascript implementation, though, there are more event handlers and service files that could possibly be combined.

#19 Post Creation Screen: Currently all posts are sent to IndexedDb. However, with the backend implemented, they will be redirected there. The current implementation of IndexedDb, though, can possibly be adjusted to allow users to save post drafts. This would require some tweaking of the Post Creation Screen as well.





Julia Farber

Work Summary

Issues:

- [Time Keeper](#) (Team)
- [Post Browsing and Search Screen](#) (UI)
- [Post Viewing Screen](#) (UI)
- [Post Data Structure](#) (Data)

Commits:

- [3b0a3d7](#), [4d659fe](#) (4/7) – Post browsing– changing some styling to match rest of the team. Also updated css media to better fit. Experimented with dropdown options, until recalled team agreed on a click dropdown (that requires javascript).
- [e84cec7](#), [79849c2](#), [499660c](#) (4/7) – Post viewing – stylization fixes based on team interfaces, similar to above.
- [Cb8b744](#), [de09fa8](#), [0feb4be](#), [fb43755](#), [7de195e](#), [6ad59ab](#) (4/10 and 4/11) – Details in commit messages and in pull request below.

Pull Requests:

- [#57](#) – Post Browsing and Post Viewing features both on this branch (rendering these pages individually, search feature functionality, and commenting).

Features Demo:

- 1) Adding Comments Under Posts (2 pt)
- 2) Retrieving Post Content from IndexedDB for Post Viewing (1 pt)

Retrieving Post Content from IndexedDB for Post Viewing (1 pt): Rendering page entirely based on mock feed data, passed into database (in future may come from server with fetch() in next milestone). Uses creating the HTML through javascript to properly format and create varied posts with ease. Uses indexedDB for post data retrieval. Enables primary function of communicating on this application through the indexedDB database on posts.

Commenting under Posts explained on slide 25.

Study On Campus

[Browse Locations](#)[+ Create a Post](#)

:3

[← Back to All Posts](#)**Melissa** (she/her)

9:30 A.M 9/26/2026

Study buds for discrete exam coming up...

CS 250

Comp Sci

Midterm

Floor 2

Starts At: 9:00 A.M, Tomorrow**Location:** Campus Library

Hi everyone! My names Melissa and I am a cs major! I am looking to study with my fellow classmates for this upcoming CS 250 midterm. I am feeling honestly rilly nervous and even with the SI sessions I'm still stressed. I am going to be at the library (floor 2) tomorrow at 9 am and gonna spend the entire day there studying up! Please join at any time even if you don't feel super ready (b/c im not either!). I'd love to meet some of my classmates! Anyway good luck to everyone else taking this exam in two days. :P

:U

Udella (she/her) 11:32 A.M., 9/26/2026

Omg I'll show up! At 11 am tho prolly. See u then!

:J

Jessie (She/her) 3:29 A.M., 9/26/2026

I'll be joining!

Features Demo:

- Search Bar with Filter Functionality (2pt)
- Retrieving Posts from IndexedDB for Post Browsing Page (1 pt)

Retrieving Posts from IndexedDB for Post Browsing Page (1 point): Rendering page entirely based on mock feed data, passed into database (in future may come from server with fetch()) in next milestone). Uses creating the HTML through javascript to properly format and create varied posts with ease. Uses indexedDB for post data retrieval. Enables primary function of communicating on this application through the indexedDB database on posts.

Search Bar with Filter Functionality explained on next slide.

Study buds for discrete exam co...

CS 250

Comp Sci

Midterm

Floor

Hi everyone! My names Melissa and I am a cs major! I am looking to study with my fellow classmates for this upcoming CS 250 midterm. I am feeling honestly rly nervous and even with the SI sessions I'm still stressed. I am going to be at the library (floor 2) tomorrow at 9 am and gonna spend the entire day there studying up! Please join at any time even if you don't feel super ready (b/c im not either!). I'd love to meet some of...

Campus Library at 9:00 A.M, Tomorrow

Title Goes Here...

Tags Go Here

Some text goes here. A lot of varied text too. Of all various unique sizes. Maybe some expressions too as well. Agreed upon here.

Location here at Time, Date

Title Goes Here...

Tags Go Here

Some text goes here. A lot of varied text too. Of all various unique sizes. Maybe some expressions too as well. Agreed upon here.

Title Goes Here...

Tags Go Here

Some text goes here. A lot of varied text too. Of all various unique sizes. Maybe some expressions too as well. Agreed upon here.

Code Snippet of Search Bar with Filter Functionality

Branch: indexedDB-load-all-posts

Basic (2 pts): JS + A (DOM), E (IndexedDB, Persistence)

Description: A search bar that allows users to filter posts on the feed to the keywords they want, displaying posts that match. The search bar dynamically updates based on user input, filtering keywords against title, tags, location, and description. Data is retrieved from IndexedDB as well. Responsive to user input (DOM) and to data (IndexedDB). See slide 23 for gif demo.

UI Impact: Users are searching for study partners of specific criteria, which will ease the search process.

Implemented with IndexedDB, using a mock feed of posts fed into the posts database. Used event listeners on the search that then rendered the post feed differently. With the filter being called with each keystroke, making it dynamic. Fully functional and complete with messages to remove any user confusion.

Challenges: Rendering fully depending on indexedDB. Had to figure out passing in the indexedDB service in constructor for desired rendering of all posts in the database.

```
filterPosts(searchQuery) {  
  const filteredPosts = this.#allPosts.filter(post => {  
    const sQLower = searchQuery.toLowerCase();  
    return (post.title.toLowerCase().includes(sQLower) ||  
      post.description.toLowerCase().includes(sQLower) ||  
      post.tags.some(tag => tag.tag.toLowerCase().includes(sQLower)) ||  
      post.location.toLowerCase().includes(sQLower));  
  }); //next milestone will use isExpired as well to check if post is too old.  
  this.#renderPosts(filteredPosts);  
}
```

```
#searchQueryListener() {  
  document.getElementById("search-bar").addEventListener("input", (search) => {  
    this.filterPosts(search.target.value)  
  })  
}
```

```
#renderPosts(posts) {  
  this.#container.innerHTML = '';  
  if (posts.length === 0) {  
    const msg = document.createElement('p');  
    msg.classList.add("empty");  
    msg.textContent = "No posts found. 😞 Maybe make a post..?";  
    this.#container.appendChild(msg);  
  } // else :  
  posts.forEach(post => {  
    const postView = document.createElement('div');  
    postView.classList.add('post', 'can-click');  
    this.#container.appendChild(postView);  
    postView.innerHTML = `<div>${post.title}</div>`;  
    postView.appendChild(postView);  
  });  
}
```


Code Snippet of Commenting Under Posts

Branch: indexedDB-load-all-posts

Basic (2 pts): JS + A (DOM), E (IndexedDB, Persistence)

Description: Commenting functionality enabled by a comment box under posts that uses an event listener (DOM) of a button click ("Post Comment") that will post a comment (must have at least one character, no empty comments). Comments stored and retrieved from database (indexedDB) and dynamically update on the page for the user as they are posted. See slide 22 for gif demo.

Not fully finished for 3 points (but finished for 2 points), as comments would benefit from displaying real-time updates and loading the comments dynamically (not only from the user themselves, but from other users/commenters too). Will require figuring out of fetch and mock fetch, which has been challenging and confusing to understand (from class examples as well.)

UI Impact: Will allow students to coordinate, change plans, ask questions, and interact in order to find real study partners. Communication seen by everyone held on the server, beneficial.

```
submitC.addEventListener("click", () => {  
  if (box.value.length > 0) {  
    const newComment = {  
      userId: "1111111",  
      name: "You",  
      message: box.value,  
      timeStamp: {time: "6:47 P.M.", date: "9/26/2026"},  
      iconContent: ":Y",  
      iconBgColor: "#5ad8cc",  
      pronouns: "they/them"  
    }  
    this.#post.postComments.push(newComment);  
    this.#service.updatePost(this.#post.postId, newComment)  
    this.#renderComments();  
    box.textContent = '';  
  }  
})
```

Creates new comment object that updates data in the database (could also update the server json with patch()). Also re-renders comments for dynamic update on screen with no refresh.

Challenges and Insights

This milestone has been the most challenging thus far since we could not use a backend and had to simulate one through mock fetching and indexedDB. It was challenging attempting mock fetch and I became confused on how we are supposed to pass in mock JSON data into a fake fetch. Lecture had left me confused and I realized that I had to continue with indexedDB, despite fetch being a better implementation method for my features. It was also challenging to begin, as the structure felt overwhelming –, from the events, services, and components. After analyzing the tasks_v(1-3).zip from class, I began to understand better how to approach this milestone. Trial and error has been helpful extremely during this milestone, as the terminal errors and live server refreshes made me better understand the structure of the front-end and how my code was working. A key takeaway from working in a collaborative team environment, is that it is really important to keep each other updated on which features everyone is working on, as collaboration on confusing aspects or similar features can help accelerate the working process. Also, keeping each other updated ensures that nobody is left behind and holds everyone accountable for their self-assigned work. Working on a team on a fullstack project makes me realize that we can all focus on specific features that will tie in together over time.



Future Improvements and Next Steps

It feels like a lot of technical debt may have been accrued this milestone, at least from the code that I worked on. The front-end is much more complex than the previous milestone, and due to my understanding gaps, I may have missed on some vital front-end eventbus updating/subscribing. These will have to be discussed, implemented or fixed, alongside with the work of the backend of the next milestone.

Future Task: <https://github.com/eelston/326Group11/issues/61> . Addressing issue of messy code, and potential bugs. Additionally, the commenting being dynamically updated from other users will be something important too.





Ashley Kang

Work Summary

Issues:

- [Settings Screen Page](#) (UI)
- [Course Catalog Data Structure](#) (Data)
- [Quality Control](#) (Team)

Commits:

- [0a93c41](#) (4/11): Implemented password visibility, adding classes/dropdown menu for classes, and saving preferences to IndexedDB
- [3e5be14](#), [D9c6934](#) (4/11): Edited class dropdown menu to choose classes based on course subject and number, added Course.ts for course structure

Pull Requests:

- [#59](#) (4/11): Added functionality to settings page

Feature Demo: Adding classes dropdown menu

Branch: 18-settings-screen

Basic/Advanced (2pts): JS + A (DOM), C (Event Handling, User Interaction), E (IndexedDB, Persistence)

Study On Campus ⌵

Account

User ID

EmailEdit

PasswordEdit

☐ Show Password

Save changes

Preferences

Display major on posts☐

Display pronouns on posts☐

Receive email notifications☐

Save changes

User Profile


Display name

Pronouns

Major

Bio

Current classes



Save changes

The dropdown menu is a two-tier course selection system that allows users to:

- Select a course subject from the first dropdown
- Choose a specific course number from the dynamically populated second dropdown
- Add selected courses to their current classes list
- Save selected courses to IndexedDB

While it's functional, some potential improvements are adding a removal option, preventing duplicate classes, and adding the ability to search classes.

Code

The drop down menu allows users to select the course subject and course number and add it to their current classes. It provides visual feedbacks and maintains selected courses when reloading page.

Some challenges I faced while implementing this feature is creating the two dropdown options and preventing the user from choosing the course number without choosing the course subject.

```
subjectSelect?.addEventListener("change", (event) => {
  console.log("Subject changed:", event.target.value);
  const subject = event.target.value;

  courseNumberSelect.innerHTML = '<option value="">-- Select a number --</option>';

  if (subject) {
    console.log("Filtering courses for subject:", subject);
    const subjectCourses = MockCourses.filter(course => course.course_subject === subject);
    console.log("Found courses:", subjectCourses);

    if (subjectCourses.length > 0) {
      subjectCourses.forEach(course => {
        const option = document.createElement("option");
        option.value = course.course_number;
        option.textContent = `${course.course_number} - ${course.course_name}`;
        courseNumberSelect.appendChild(option);
      });
      courseNumberSelect.disabled = false;
    } else {
      console.log("No courses found for subject:", subject);
      courseNumberSelect.disabled = true;
    }
  } else {
    courseNumberSelect.disabled = true;
  }
  dropdownAddBtn.disabled = true;
});

courseNumberSelect?.addEventListener("change", (event) => {
  dropdownAddBtn.disabled = !event.target.value;
});

dropdownAddBtn.addEventListener("click", () => {
  const subject = subjectSelect.value;
  const number = courseNumberSelect.value;
  if (!subject || !number) return;

  const course = MockCourses.find(c =>
    c.course_subject === subject && c.course_number === number
  );

  if (course) {
    const input = document.createElement("input");
    input.type = "text";
    input.value = `${course.course_name} (${course.course_subject} ${course.course_number})`;
    input.disabled = true;

    currentClasses.insertBefore(input, currentClasses.querySelector(".add-btn-container"));

    subjectSelect.value = "";
    courseNumberSelect.innerHTML = '<option value="">-- Select a number --</option>';
    courseNumberSelect.disabled = true;
    dropdownAddBtn.disabled = true;
    dropdownMenu.style.display = "none";

    saveClassToDB(course.course_name);
  }
});
```

Challenges and Insights

For this milestone, I faced several challenges when adding JavaScript to my existing HTML and CSS files. One big issue was getting the dropdown menu to work properly. I needed it so that users could only pick a course number after choosing a course subject, and the list of options came from a mock data file. Making this work smoothly took some time and testing. Another challenge was using IndexedDB to save user preferences and profile updates. Since it works differently from regular storage and runs in the background, it took some effort to make sure the data saved correctly and stayed there after refreshing the page.



Future Improvements and Next Steps

- Add a remove/edit option for current classes
- Add a search option when adding a class
- Change updated information confirmation with a message/banner
- Potentially reconfigure settings page layout using multiview UI to prevent overcrowding of information on the page and make room for adding more settings sections in the future

