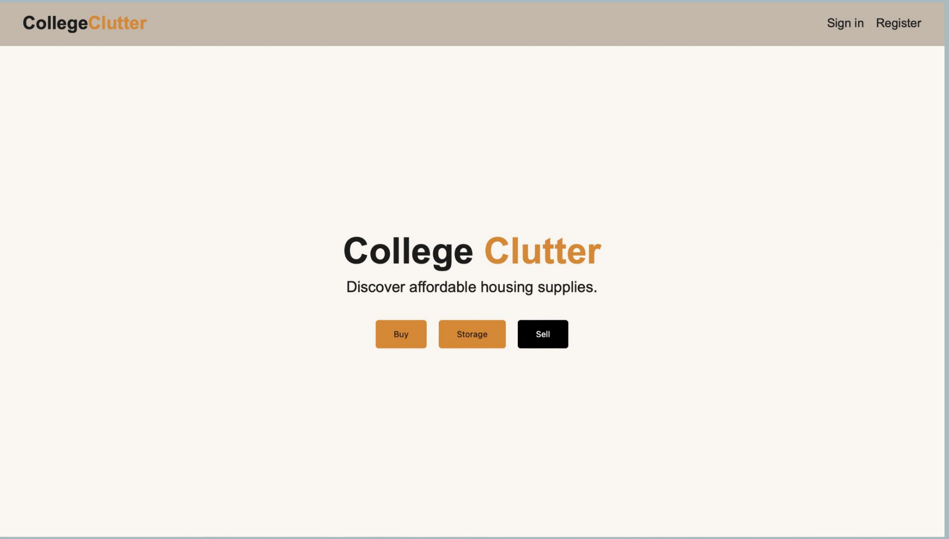
College Clutter


CollegeClutter

Sign in Register

College Clutter

Discover affordable housing supplies.

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

Milestone #5

CS 326 SPRING 2025

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: #6
Issues on GitHub:
https://github.com/umass-byrneb/CS326-Group-36/issues?q=is%3Aissue%20state%3Aopen%20milestone%3A%22Milestone%20%236%22

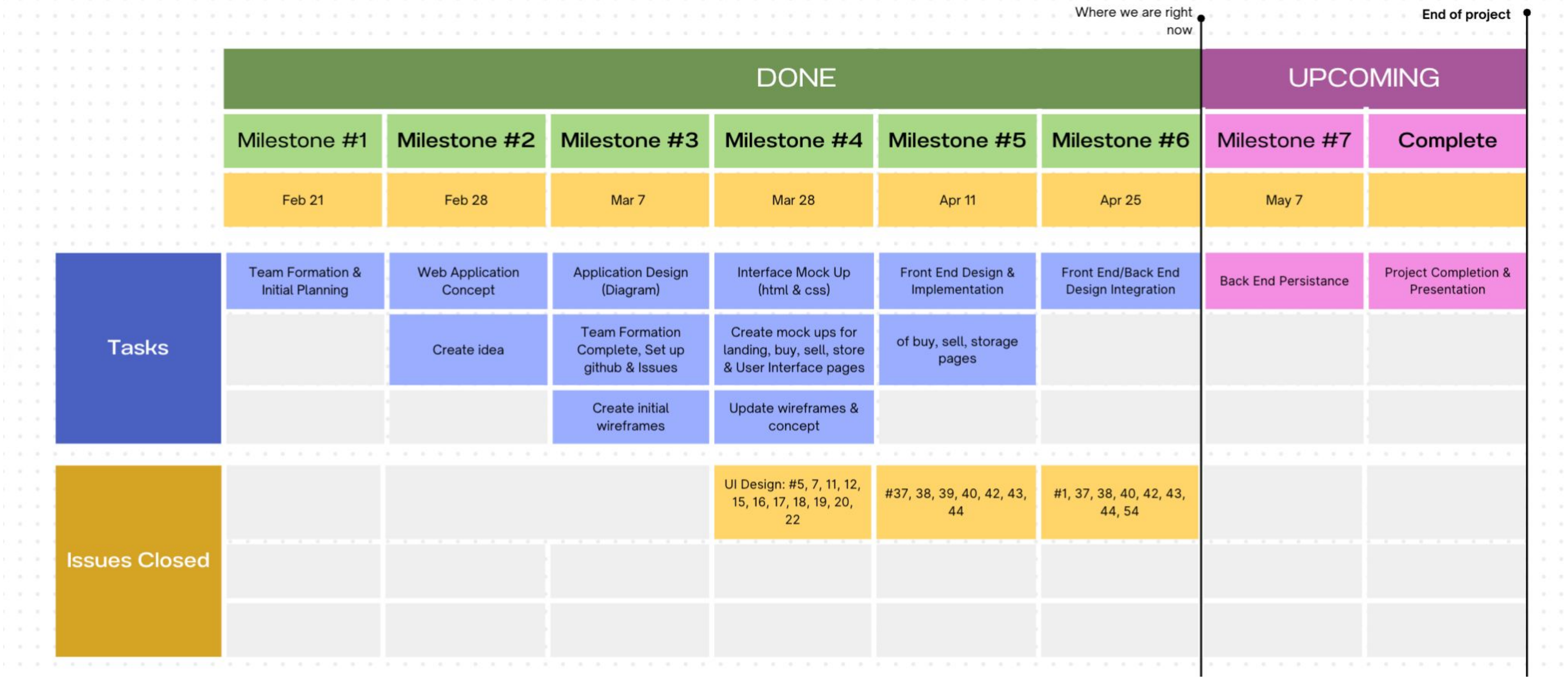
The builders.

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.



Where we are right now

End of project

	DONE						UPCOMING	
	Milestone #1	Milestone #2	Milestone #3	Milestone #4	Milestone #5	Milestone #6	Milestone #7	Complete
	Feb 21	Feb 28	Mar 7	Mar 28	Apr 11	Apr 25	May 7	
Tasks	Team Formation & Initial Planning	Web Application Concept	Application Design (Diagram)	Interface Mock Up (html & css)	Front End Design & Implementation	Front End/Back End Design Integration	Back End Persistance	Project Completion & Presentation
		Create idea	Team Formation Complete, Set up github & Issues	Create mock ups for landing, buy, sell, store & User Interface pages	of buy, sell, storage pages			
			Create initial wireframes	Update wireframes & concept				
Issues Closed				UI Design: #5, 7, 11, 12, 15, 16, 17, 18, 19, 20, 22	#37, 38, 39, 40, 42, 43, 44	#1, 37, 38, 40, 42, 43, 44, 54		

Brandon Byrne

Assigned Work Summary

Assigned Issues: #43, #44
Commits: login.js, RegistrationComponent.js, User.js, back end files, usertask

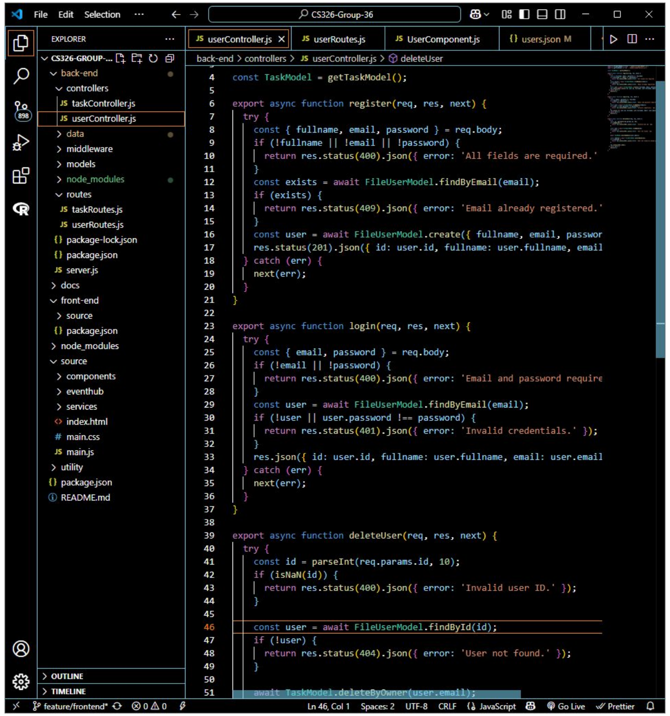
Tasks completed: Back-end integration: Added POST /v1/users/register and POST /v1/users/login endpoints. Wiredup models/fileUserModel.js, controllers/userController.js, routes/userRoutes.js. Ensured new users are persisted in data/users.json

Front-end wiring: Extended RegisterComponent to call /v1/users/register, handle “email already exists,” redirect to login. Extended LoginComponent to call /v1/users/login, store currentUser in localStorage, redirect to dashboardUpdated nav and route-guards so only logged-in users see Sell/User pages

- Links to PRs closed on GitHub:
- https://github.com/umass-byrneb/CS326-Group-36/pull/61
- https://github.com/umass-byrneb/CS326-Group-36/pull/59
- https://github.com/umass-byrneb/CS326-Group-36/pull/60

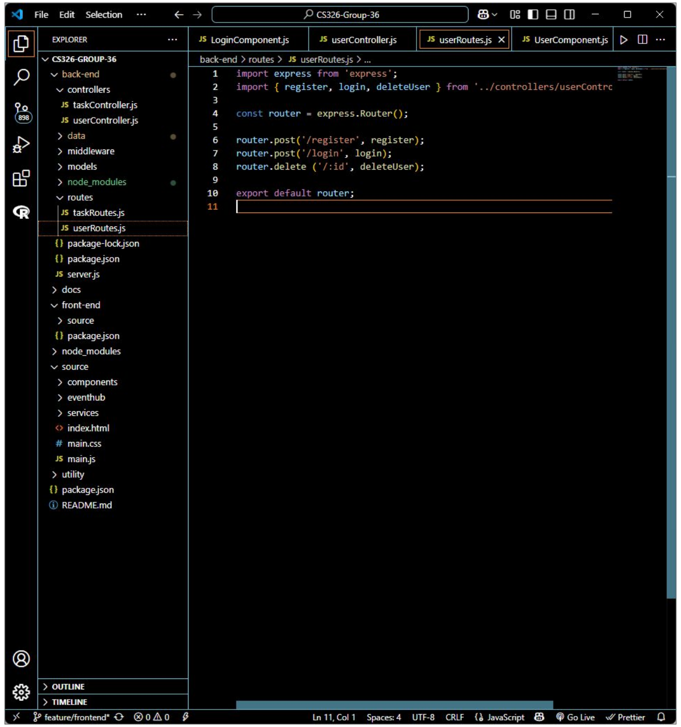
Brandon Byrne

Screenshots and demonstration



```
const TaskModel = getTaskModel();

export async function register(req, res, next) {
  try {
    const { fullname, email, password } = req.body;
    if (!fullname || !email || !password) {
      return res.status(400).json({ error: 'All fields are required.'
    }
    const exists = await FileUserModel.findByEmail(email);
    if (exists) {
      return res.status(409).json({ error: 'Email already registered.'
    }
    const user = await FileUserModel.create({ fullname, email, passwor
    res.status(201).json({ id: user.id, fullname: user.fullname, email
  } catch (err) {
    next(err);
  }
}

export async function login(req, res, next) {
  try {
    const { email, password } = req.body;
    if (!email || !password) {
      return res.status(400).json({ error: 'Email and password require
    }
    const user = await FileUserModel.findByEmail(email);
    if (!user || user.password !== password) {
      return res.status(401).json({ error: 'Invalid credentials.' });
    }
    res.json({ id: user.id, fullname: user.fullname, email: user.email
  } catch (err) {
    next(err);
  }
}

export async function deleteUser(req, res, next) {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
      return res.status(400).json({ error: 'Invalid user ID.' });
    }

    const user = await FileUserModel.findById(id);
    if (!user) {
      return res.status(404).json({ error: 'User not found.' });
    }

    await TaskModel.deleteByOwner(user.email);
```

filterUsermodel



```
import express from 'express';
import { register, login, deleteUser } from '../controllers/userContrc

const router = express.Router();

router.post('/register', register);
router.post('/login', login);
router.delete ('/:id', deleteUser);

export default router;
```

UserRoutes

College Clutter

Brandon Byrne

Feature demonstration and code explanation

During this milestone I extended our back-end to fully support user authentication by adding two new endpoints, `/v1/users/register` and `/v1/users/login`. In the data layer I created and wired up `FileUserModel` methods to read, write, and query `data/users.json`, and in `userController.js` I implemented a `register` action that validates incoming fullname, email, and password fields, checks for duplicate emails, and persists new users; likewise, the `login` action looks up credentials, returns a 401 on failure, or the user profile on success. I then exposed these actions in `userRoutes.js` under POST routes, and updated our Express server to serve them alongside our task API. On the front-end I enhanced `RegisterComponent` so it captures form input, submits it as JSON to `/v1/users/register`, handles errors like "email already exists," and, on success, resets the form and navigates to the login page. I also updated `LoginComponent` to POST credentials to `/v1/users/login`, parse the JSON response, store the returned user object in `localStorage.currentUser`, and redirect to the user dashboard, while displaying prompts or hiding protected pages when no user is logged in. Overall, these tasks have fully knit together our registration and login flows, laying a solid foundation for persistent, secure user sessions across the application.

Brandon Byrne

Challenges and Insights

Obstacles:
During this milestone I ran into several obstacles that, while frustrating at the time, taught me valuable lessons about building a robust web application. Early on, I kept getting 404 errors or HTML responses instead of JSON because my Express route mounts didn't match my router.post() definitions—this drove home the importance of having a clear, shared API contract and double-checking base paths versus endpoint paths. I also hit "Cannot use import statement outside a module" errors when mixing CommonJS and ES modules, which led me to learn that adding "type": "module" to package.json is essential for consistency across our Node environment. Finally, managing authentication state in localStorage.currentUser worked for now, but the repeated JSON parsing and manual checks at each entry point revealed the limitations of simple client-side state and underscored the future need for a token-based approach. Together, these hurdles reinforced the importance of precise configuration, consistent conventions, and thinking ahead about state management.

Lessons Acquired:
Working within our team environment surfaced several key takeaways about collaboration. First, drafting even a one-page API spec early prevented wasted effort and misaligned assumptions—front-end and back-end developers could work in parallel without stepping on each other's toes. Second, our modular folder structure (separating components, services, controllers, and models) allowed each teammate to own distinct areas of the codebase, reducing merge conflicts and clarifying responsibilities. Finally, instituting a pull-request and code-review process fostered knowledge sharing and caught both logic errors and stylistic inconsistencies before they reached staging, which boosted code quality and helped everyone learn best practices more quickly.

Brandon Byrne

Future improvements & next steps

Future Improvements:
As we solidified the core flows, we also identified several areas of technical debt and opportunities for optimization. Storing plaintext passwords in users.json is a critical security risk that must be addressed by integrating a hashing library.. Relying on localStorage for session state lacks server-side validation and timeout controls, suggesting a migration to JWT-based authentication. Our file-based JSON storage is vulnerable to race conditions under concurrent writes and won't scale well—moving to a real database (even a lightweight one) would improve both performance and reliability. Form validation currently depends on alert() pop-ups; a better user experience would use inline validation messages and disable the submit button until fields are valid.

https://github.com/umass-byrneb/CS326-Group-36/issues/62
https://github.com/umass-byrneb/CS326-Group-36/issues/63

Sargam Nohria

Assigned Work Summary

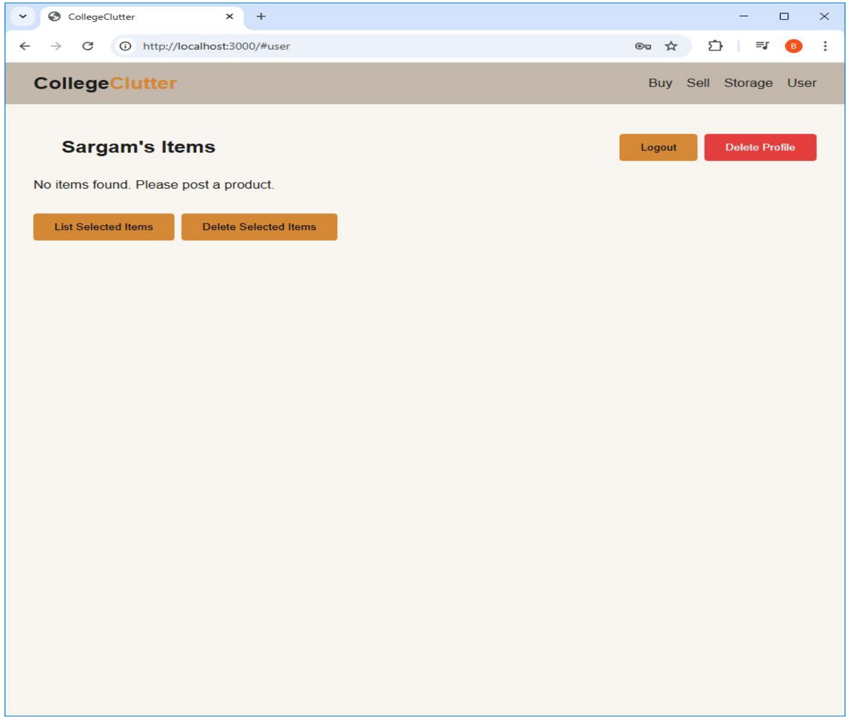
Assigned Issues: #40, #42
Commits: UserPage, BuyPage, Backend File restructure

Tasks completed:On the Buy page I wired up GET /v1/tasks, filtered only listed: true items from all users, and rebuilt our dynamic UI. Together these changes close the loop: Sell → User dashboard (draft/unlisted) → List for sale → Buy page (public listing).

- Links to PRs closed on GitHub:
- https://github.com/umass-byrneb/CS326-Group-36/pull/58
-

Sargam Nohria

Screenshots & Demonstration



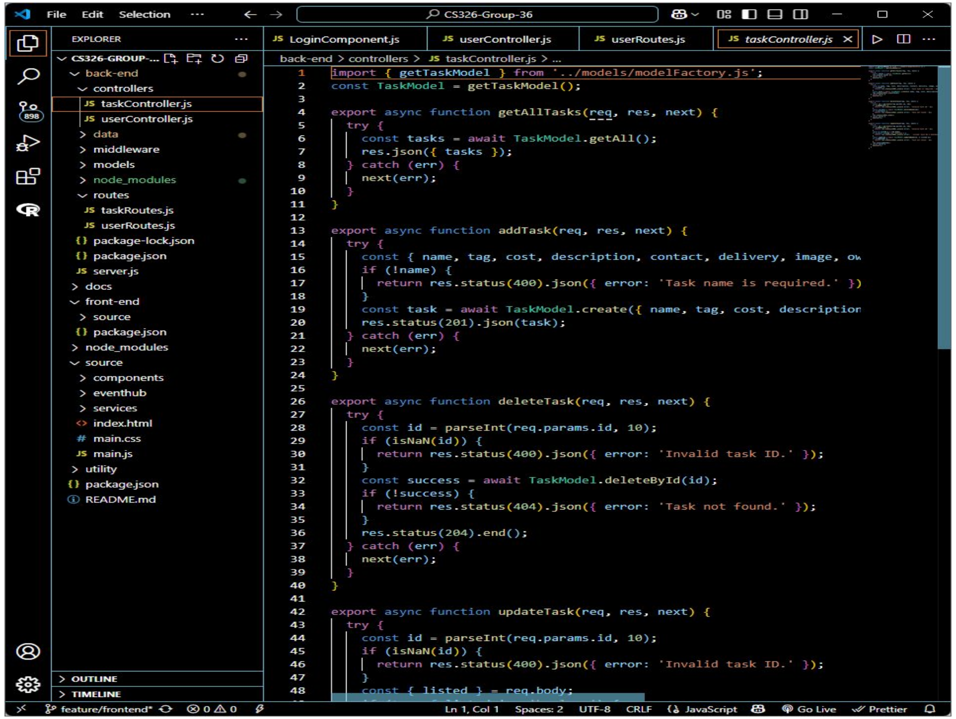
Sargam's Items

No items found. Please post a product.

Revamped user page

Sargam Nohria

Screenshots & Demonstration



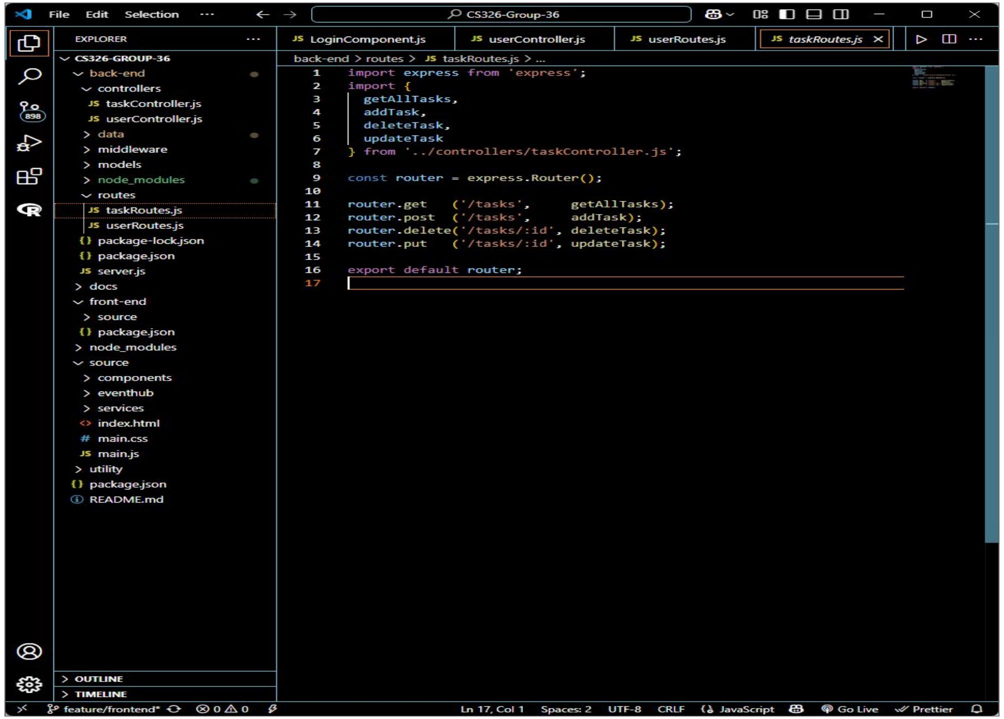
```javascript
import { getTaskModel } from '../models/modelFactory.js';
const TaskModel = getTaskModel();

export async function getAllTasks(req, res, next) {
  try {
    const tasks = await TaskModel.getAll();
    res.json({ tasks });
  } catch (err) {
    next(err);
  }
}

export async function addTask(req, res, next) {
  try {
    const { name, tag, cost, description, contact, delivery, image, ow
    if (!name) {
      return res.status(400).json({ error: 'Task name is required.' })
    }
    const task = await TaskModel.create({ name, tag, cost, description
    res.status(201).json(task);
  } catch (err) {
    next(err);
  }
}

export async function deleteTask(req, res, next) {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
      return res.status(400).json({ error: 'Invalid task ID.' });
    }
    const success = await TaskModel.deleteById(id);
    if (!success) {
      return res.status(404).json({ error: 'Task not found.' });
    }
    res.status(204).end();
  } catch (err) {
    next(err);
  }
}

export async function updateTask(req, res, next) {
  try {
    const id = parseInt(req.params.id, 10);
    if (isNaN(id)) {
      return res.status(400).json({ error: 'Invalid task ID.' });
    }
    const { listed } = req.body;
```

taskController

College Clutter

Sargam Nohria

Screenshots & Demonstration



```
import express from 'express';
import {
  getAllTasks,
  addTask,
  deleteTask,
  updateTask
} from '../controllers/taskController.js';

const router = express.Router();

router.get   ('/tasks',     getAllTasks);
router.post  ('/tasks',     addTask);
router.delete('/tasks/:id', deleteTask);
router.put   ('/tasks/:id', updateTask);

export default router;
```

taskRoutes

Sargam Nohria

Code and UI explanation

Over this milestone I completed the integration of our back-end with both the User and Buy pages. On the User page, I replaced the previous in-memory mock data with a real GET /v1/tasks call, filtered those tasks by the logged-in user’s email, and rendered each item along with checkboxes, a “List Selected Items” button that sets a listed flag via PUT /v1/tasks/:id, and a red “Delete Profile” button that cascades to delete both the user and all their items via DELETE /v1/users/:id. On the Buy page, I wired up a GET /v1/tasks request that pulls in every item marked listed: true, built a cost slider whose range is computed from the data, and implemented a responsive, case-insensitive prefix search bar, dynamic tag and delivery filters (complete with removable pill tags), and three sort buttons (“New”, “Price ascending”, “Price descending”). Together, these changes close the loop from drafting an item on Sell → managing listings on your dashboard → public sale on the Buy page.

Sargam Nohria

Challenges and Insights

- Along the way, several obstacles taught me important lessons. I spent time debugging 404s and HTML-in-JSON errors because my Express route mounts didn't match my router definitions, which reinforced the need for a precise, shared API contract. I also ran into Node's "import… outside a module" error until I learned to enable ES modules via "type": "module" in package.json. Managing front-end state in localStorage.currentUser worked but required repeated JSON parsing and careful null checks, highlighting both the power and pitfalls of simple client-side authentication. Finally, orchestrating the cascade-delete flow (delete user → delete all tasks → confirm) underscored the importance of operation ordering and error handling in server logic.

  Working on this with my teammates underscored the value of collaboration practices. Drafting even a minimal API spec early meant front-end and back-end developers could code in parallel with confidence that our endpoints would align. Our modular folder structure—separating components, controllers, models, and routes—let each person own specific parts of the codebase without stepping on one another's toes. And instituting a pull-request review process caught subtle bugs (from filter logic to edge-case error handling) before merging, which not only improved quality but served as an ongoing learning opportunity for the whole team.

Sargam Nohria

Future improvements & next steps

- Fetching all tasks and filtering in the browser won't scale once we have hundreds of listings—server-side filtering, pagination, or virtual scrolling will be required. We currently store plaintext passwords and use localStorage for session state; we must integrate hashing and migrate for secure, verifiable authentication. Our file-based JSON storage can suffer race conditions under concurrent writes and isn't suitable for production; migrating to a lightweight database would improve reliability.

- https://github.com/umass-byrneb/CS326-Group-36/issues/64
- https://github.com/umass-byrneb/CS326-Group-36/issues/65

Simran Lekhwani

Assigned Work Summary

Assigned Issues: #54 (Storage Page)
Commits: Back-end integration (routes, controller, model, data structure); front-end updates (SellComponent, UserComponent, Storage Service and Storage Remote Service)

Tasks completed:
Completed the back-end integration for the Storage page. Added GET /v1/storage/listings and POST /v1/storage/listings endpoints to add and retrieve storage items from the in-memory data structure for storage (storage.json). Also added a PUT /v1/storage/listings/:id endpoint to update a listing when it gets listed by the user. Added respected routes (storageRoutes), controller (storageController), model (fileStorageModel) and in-memory data structure (storage.json) for the storage page.
Updated the front-end, mainly the user and sell pages, to redirect storage space related posts to the storage listing endpoint (POST and PUT /v1/storage/listing). In addition, instead of loading from a fake service (from milestone 5), the storage page loads the current storage listings using a GET request to the in-memory data structure on the back end.

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

Simran Lekhwani

Screenshots and Demonstration

Simran Lekhwani

Code structure and Organization:

For modularity, the back-end is structured in different “modules” or directories: controllers, data, middleware, model, routes, and server.js

Middleware does error handling.

Under routes, there is a specific file dedicated for Storage page routes, which perform actions based on asynchronous functions imported from the Storage Controller in the controllers dir. These functions rely on the fileStorageModel to perform CRUD operations (in this case, create, read and update) for the in-memory data structure in the data dir

For isolation and modularity, continuing from the previous milestones, the front-end is broken down into source, eventhub and services.

Updates were made to the Sell page to allow users to post/sell storage spaces. Consequently, the user page was reflected with the new storage spaces

Simran Lekhwani

Front-end Implementation:

Back-end Implementation:

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