UMass Lost and Found Milestone 6

4/25/2025
Owen,Jacob,Roberto,Nathan

UMass-Lost-and-Found Project Vision

Problem: It is common for UMass students to misplace important belongings like UCards, water bottles, and academic utensils. There currently doesn't exist an organized place for students to communicate what items they may have lost or found.

Solution: UMass Lost and Found intends to be the first and best place for students to go to communicate with each other items they are looking for or have found. Our web app is organized and easy to use and provides users with all the tools and information they need to find lost items or share items they found.

UMass-Lost-and-Found Project Vision Continued

Key Features:

- Home page containing all found item posts
- Posting features such as images and GPS location to help users locate items
- Sorting, filtering, and searching options to narrow down posts
- Messaging system for communication between posters and users

Repository: https://github.com/realraft/UMass-Lost-and-Found

Milestone #5: https://github.com/realraft/UMass-Lost-and-Found/milestone/4

UMass-Lost-and-Found Builders

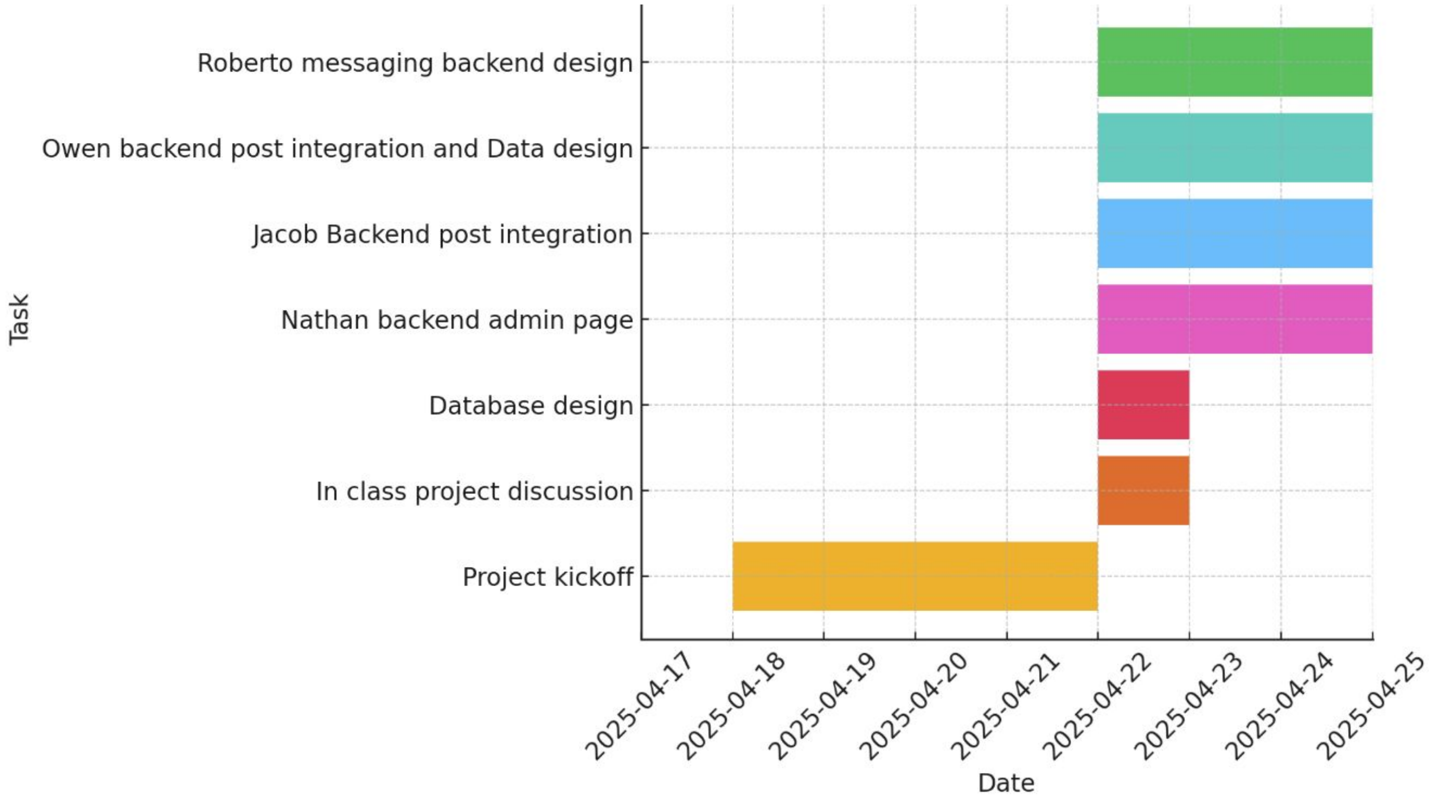
Owen Raftery
Role(s): Project-Manager
Role(s) Description: Will be leading the team during meetings and give instructions on what to code.
Issues from current milestone: Allow backend to sent front end to user. Implement post manager page that allows users to edit and delete their own posts.

Nathan Dennis
Role(s): Notetaker, Front-end developer
Role(s) Description: Will be taking down notes, and recording what my teammates do in regards to the website. Will also offer suggestions and implement them when necessary. Will mainly work on the front-end.
Issues from current milestone:

UMass-Lost-and-Found Builders

Jacob Taylor
Role(s): Implement backend functions in models and controller, integrate that with the frontend, and update the css styling in postItemPage
Role(s) Description: Implement key features so the backend and frontend are compatible. In specific to post creation. Will implement update and create crud operations for postings.
Issues from current milestone: Linking front-end and back-end together as we had to make sure everything worked respectively.
Roberto Rubio
Role(s): Communicator
Role(s) Description: Will make sure the Backend and Frontend are syncronized and will code for both.
Issues from current milestone:

Historical Timeline For Milestone 6



Jacob Taylor Work Summary

I worked on implementing the backend for the posts, primarily on the creation of the posts serverside. I edited files in backend/routes/postRoutes.js, and backend/controller/postController.js, and backend/models/index.js I also worked on linking the frontend to the backend for posting, and updating the user interface on our postItemPage.

**PRS: https://github.com/realraft/UMass-Lost-and-Found/pull/169,
https://github.com/realraft/UMass-Lost-and-Found/pull/165,
https://github.com/realraft/UMass-Lost-and-Found/pull/166,**

https://github.com/realraft/UMass-Lost-and-Found/pull/162

**Issues closed: https://github.com/realraft/UMass-Lost-and-Found/issues/167 ,
https://github.com/realraft/UMass-Lost-and-Found/issues/163 ,
https://github.com/realraft/UMass-Lost-and-Found/issues/150 ,
https://github.com/realraft/UMass-Lost-and-Found/issues/142**

Feature Demonstration Jacob Taylor

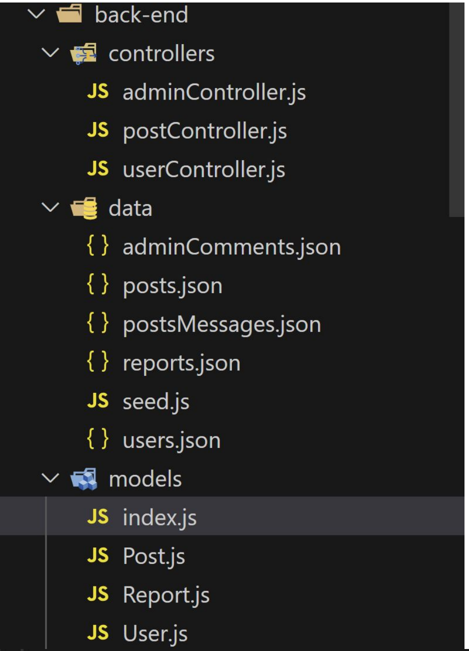
Backend branch: feature-post-frontend-integration, and 167-database-bug-fix

Frontend Branch: `feature-post-frontend-integration`

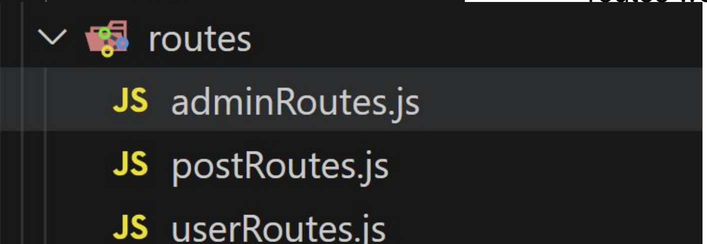
My features were implementing the post creation backend integration. Integrating that with the fronten when a user creates a new post from our postItemPage. I also changed the css styling in our postItemPage to make it more modern and cleaner. I also implemented an image upload feature to our posts for users to try out. The frontend and backend features are both functional as they were required for other team members to complete their work.

To be specific I implemented the create and contributed to the update, part of the posts in respect to th CRUD operations.

Code Structure and Organization Jacob Taylor



```
back-end
  controllers
    adminController.js
    postController.js
    userController.js
  data
    adminComments.json
    posts.json
    postsMessages.json
    reports.json
    seed.js
    users.json
  models
    index.js
    Post.js
    Report.js
    User.js
```



```
routes
  adminRoutes.js
  postRoutes.js
  userRoutes.js
```

In this project the **front-end** and **back-end** are in completely separate folders, with no overlapping responsibilities.

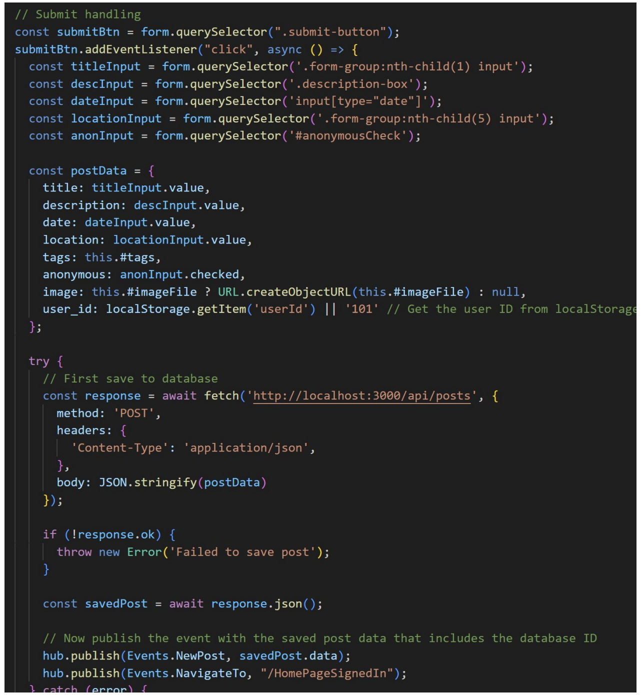
1. **Routes** look at the URL and HTTP method and grab any parameters included

2. **Controllers** take that data, and call any of our methods we defined, for example getAllPosts, or getPostById.

3. **Models** are the only code that ever touches our data folder, they directly access it.
4. **server.js** ties everything together: it starts Express.js, and initializes all our routes we defined in our routes folder.

Our code is easy to maintain and read with this split up approach. As our frontend files do not touch the backend folder directly, as the frontend fetches data from the backend using our routes we defined.

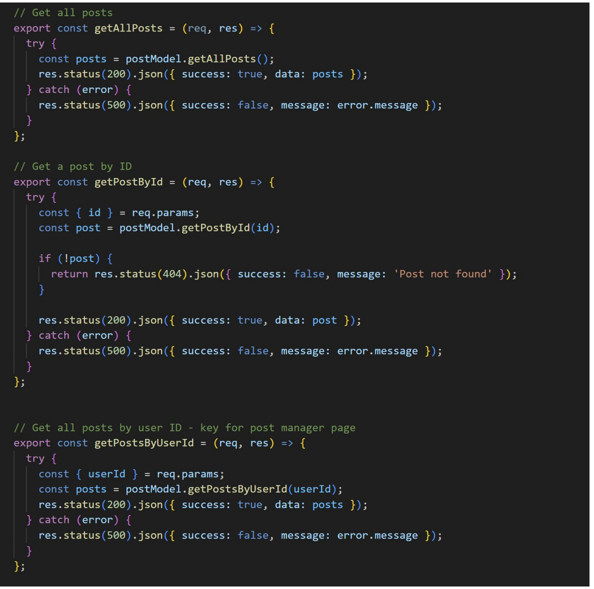
Frontend Implementation Jacob Taylor



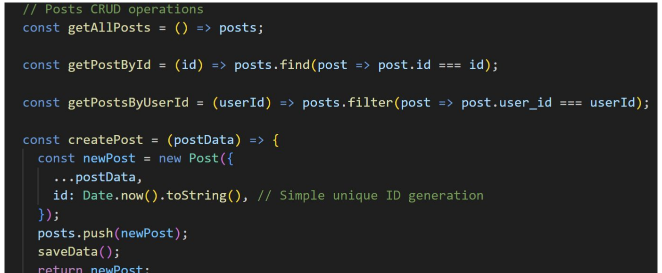
```
// Submit handling
const submitBtn = form.querySelector(".submit-button");
submitBtn.addEventListener("click", async () => {
  const titleInput = form.querySelector('.form-group:nth-child(1) input');
  const descInput = form.querySelector('.description-box');
  const dateInput = form.querySelector('input[type="date"]');
  const locationInput = form.querySelector('.form-group:nth-child(5) input');
  const anonInput = form.querySelector('#anonymousCheck');

  const postData = {
    title: titleInput.value,
    description: descInput.value,
    date: dateInput.value,
    location: locationInput.value,
    tags: this.#tags,
    anonymous: anonInput.checked,
    image: this.#imageFile ? URL.createObjectURL(this.#imageFile) : null,
    user_id: localStorage.getItem('userId') || '101' // Get the user ID from localStorage
  };

  try {
    // First save to database
    const response = await fetch('http://localhost:3000/api/posts', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(postData)
    });

    if (!response.ok) {
      throw new Error('Failed to save post');
    }

    const savedPost = await response.json();

    // Now publish the event with the saved post data that includes the database ID
    hub.publish(Events.NewPost, savedPost.data);
    hub.publish(Events.NavigateTo, "/HomePageSignedIn");
  } catch (error) {
```

This specific part of code is when a user creates a new post on our postItemPage. It integrates with the backend, as it makes a post request to: 'http://localhost:3000/api/posts' endpoint we defined. The components is used in the frontend as we parse the forms for the post data. For example the 'description-box' is the input field for grabbing the description of the post. I faced challenges in this feature as the post was getting updated in the frontend, but was not getting updated in our database. The solution was I was only publishing the new event to the frontend but not handling it server-side.

Backend implementation Jacob Taylor



```javascript
// Get all posts
export const getAllPosts = (req, res) => {
  try {
    const posts = postModel.getAllPosts();
    res.status(200).json({ success: true, data: posts });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

// Get a post by ID
export const getPostById = (req, res) => {
  try {
    const { id } = req.params;
    const post = postModel.getPostById(id);

    if (!post) {
      return res.status(404).json({ success: false, message: 'Post not found' });
    }

    res.status(200).json({ success: true, data: post });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};


// Get all posts by user ID - key for post manager page
export const getPostsByUserId = (req, res) => {
  try {
    const { userId } = req.params;
    const posts = postModel.getPostsByUserId(userId);
    res.status(200).json({ success: true, data: posts });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};
```



```javascript
// Posts CRUD operations
const getAllPosts = () => posts;

const getPostById = (id) => posts.find(post => post.id === id);

const getPostsByUserId = (userId) => posts.filter(post => post.user_id === userId);

const createPost = (postData) => {
  const newPost = new Post({
    ...postData,
    id: Date.now().toString(), // Simple unique ID generation
  });
  posts.push(newPost);
  saveData();
  return newPost;
```

This (top image) snippet is located in backend/controllers/postController These define the post operations used when the frontend make a request to the specified post endpoint. Particularly used in the frontend to display the posts in our homepage. The bottom snippet is located in backend/models/index.js and is used in the postController file. A challenge I faced was making sure everything was working property before moving on to something else. I learned I need to test thoroughly before moving onto another part of the project, as post controller relies on the model/index.js file.

Challenges and insights Jacob Taylor

I learned that in order for our team to work productively everyone needs to be on the same page. As my section is required for everything else's part to work correctly. If I didn't finish my work on time or it didn't work, everyone else's part wouldn't have worked either. I also learned that asking for help is important as if you don't understand another group members code. Just ask as its better than not knowing how it may integrate in your part. A key takeaway from working in a collaborative environment is communication is key when working in a big codebase. Everyone needs to be on the same page when working on a website together.

Future Improvements and next steps Jacob Taylor

1. Make sure public users cannot access our administration page, we do not want users hacking into our website and deleting posts or worse.
   https://github.com/realraft/UMass-Lost-and-Found/issues/116?issue=realraft%7CUMass-Lost-and-Found%7C172
2. Test anonymous listing feature, as we want users to feel safe making listing in case they don't want to reveal their name.

   https://github.com/realraft/UMass-Lost-and-Found/issues/116?issue=realraft%7CUMass-Lost-and-Found%7C171

3. When we implement the google maps api, our backend server may break, we need to integrate it properly do nothing breaks.

   https://github.com/realraft/UMass-Lost-and-Found/issues/116?issue=realraft%7CUMass-Lost-and-Found%7C170

4. Reduce code duplication, and create helper functions for better readability and maintainability. This is important if a bug may arise and we need to look through our codebase to identify it.

   https://github.com/realraft/UMass-Lost-and-Found/issues/116?issue=realraft%7CUMass-Lost-and-Found%7C173 .

Owen Raftery Work Summary

Details: I implemented the post manager page and also the ability for the backend to send the front end to the user. The post manager page uses the user id stored in the local storage to run a GET request to the backend and get the user's posts. The page allows the user to edit or delete them with PUT and DELETE requests and then those posts are dynamically updated across the web app. Jacob contributed to the editing portion here as well. I had a lot of pathing to fix to get the backend server to send the front end to the user with correct css.

PRs:

- https://github.com/realraft/UMass-Lost-and-Found/pull/158
- https://github.com/realraft/UMass-Lost-and-Found/pull/161
- https://github.com/realraft/UMass-Lost-and-Found/pull/164
- https://github.com/realraft/UMass-Lost-and-Found/pull/174
- https://github.com/realraft/UMass-Lost-and-Found/pull/175

Owen Raftery Feature Demonstration

Post Manager Page:

- Reads browser local browser storage for a user id
- Uses GET to pull the user specific posts
- Uses DELETE to delete a users post
- Uses EventHub to dynamically update posts after an edit or a delete

Edit Post Page:

- Fills Post Item Page with the posts information from the backend using a GET
- Uses PUT to edit a user’s post

Owen Raftery Front End

I implemented the Post manager page and edit post page. Edit post page is the same as the post item page with different functionality to edit a post rather than update it. This now allows users to update their posts and delete them as needed.

```
export class PostManagerPage extends BasePage {
  #container = null;
  #listingContainer= null;
  #userId = null;

  constructor() {
    super();
    this.loadCSS("pages/PostManagerPage", "PostManagerPage");
    this.#userId = localStorage.getItem('userId') || '101';
  }

  render() {
    document.body.className = 'post-manager-page';

    if (this.#container) {
      setTimeout(() => this.#checkForSearchQuery(), 0);
      return this.#container;
    }

    this.#container = document.createElement("div");
    this.#container.className = "page-container";

    this.#setupContainerContent();
    this.#attachEventListeners();
    this.#loadData();

    return this.#container;
  }

#createListingElement(post, addToBeginning = false) {
    if (!this.#listingContainer) return;

    const listing = document.createElement("div");
    listing.classList.add("listing");
    listing.id = post.id;
    listing.style.cursor = 'pointer';

    listing.addEventListener('click', (e) => {
      if (!e.target.classList.contains('edit-button') &&
          !e.target.classList.contains('delete-button')) {
        EventHub.getEventHubInstance().publish(Events.ViewPost, post);
      }
    });
```

Owen Raftery Back End

I implemented the backends ability to send the front end html, js pages, and CSS to the user. This links backend functionality with the front end. I also implemented the CRUD api points to delete a post, get a post by user id, and contributed to updating a post. This allows posts to be customizable to the user.

```
export const getPostsByUserId= (req, res) => {
  try {
    const { userId } = req.params;
    const posts = postModel.getPostsByUserId(userId);
    res.status(200).json({ success: true, data: posts });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const updatePost = (req, res) => {
  try {
    const { id } = req.params;
    const postData = req.body;

    const updatedPost = postModel.updatePost(id, postData);

    if (!updatedPost) {
      return res.status(404).json({ success: false, message: 'Post not found' });
    }

    res.status(200).json({ success: true, data: updatedPost });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const deletePost = (req, res) => {
  try {
    const { id } = req.params;
    const deletedPost = postModel.deletePost(id);

    if (!deletedPost) {
      return res.status(404).json({ success: false, message: 'Post not found' });
    }

    res.status(200).json({ success: true, data: deletedPost });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};
```

Owen Raftery Challenges and Insights

The main challenge I faced was linking the backend with the front end. I had a lot of issues with pathing and it was hard to find the static path to get the backen to send the front end. After that was working it was really hard to get the CSS to now work with the front end. That was fixed through further pathing issues with the Base Page.

Owen Raftery Future Improvements and Next Steps

- Make sure filters reapply after dynamic updates
- Synthesize the data requirements for posts (make them better and make some fields required)
- Implement authentication and logging in
- https://github.com/realraft/UMass-Lost-and-Found/issues/178
- https://github.com/realraft/UMass-Lost-and-Found/issues/179
- https://github.com/realraft/UMass-Lost-and-Found/issues/180

Roberto’s Work Summary

I worked on the following tasks:

Routes, controller, and model for the messaging page back-end integration.

Modified the front-end to use fetch calls when needing to add, get, or send messages to other users.

In the back-end is mostly CRUD functions for the messages.

I’ll have to reopen some of the issues once I figure out a way to fix the messaging page.

Issues, branches and PR:

- https://github.com/realraft/UMass-Lost-and-Found/issues/123
- https://github.com/realraft/UMass-Lost-and-Found/issues/138
- Messagingroutesandcontroller
- Messagesfrontandback
- https://github.com/realraft/UMass-Lost-and-Found/pull/177

Roberto’s Feature Demonstration

There isn’t really a feature demonstration. The code tries to send messages to the .json files in the data folder, acts a server. The main issue happens when creating a new conversation and the messaging page doesn’t load when accessed through the drop down menu.

If it worked, what it would do is the following: create conversations between users for different posts. Send and get messages from the server to render the conversations in the messaging page. I believe the problem arises from the inexistence of users at the moment.

The code would also dynamically change the messages once a new one is sent, as implemented in the previous milestone.

Roberto’s Code Structure and Organization

Most of the new code is in the back-end folder. Here there are files in the routes, models, data, and controllers folders. All give way to the CRUD implementation of the messaging page of the website.

In the front-end, some of the changes are to MessagingService. I’ve come to the conclusion that IndexedDB is not really needed anymore. All storing will be done with the server and calls to fetch.

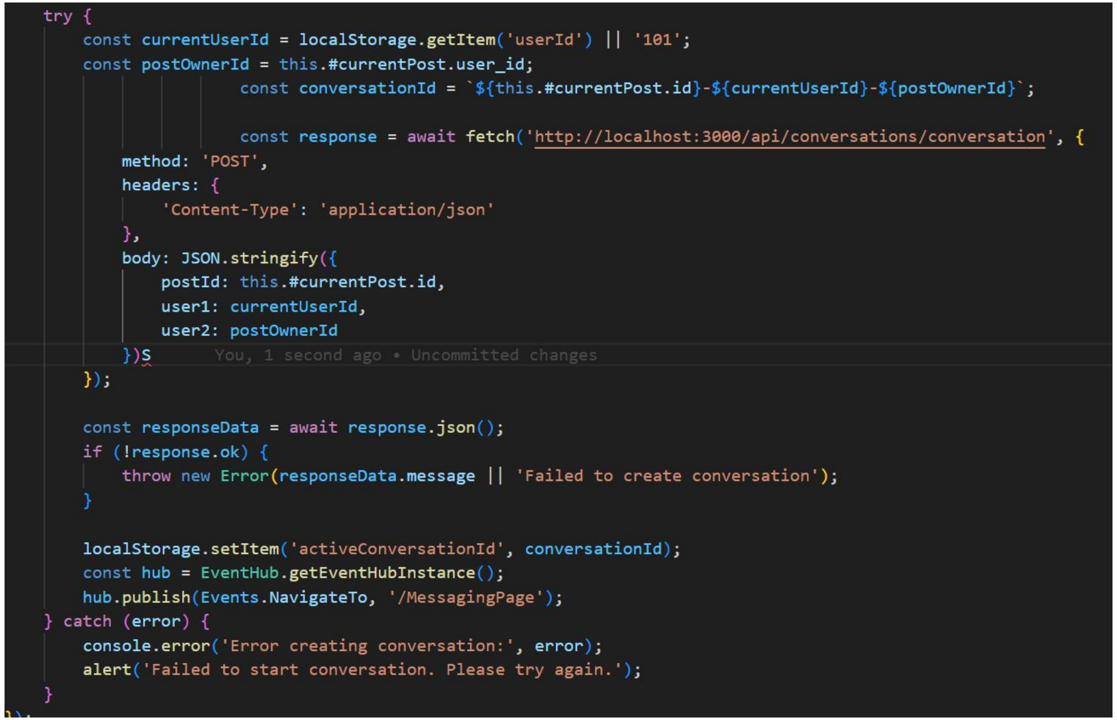
In the front-end the file organization wasn’t changed, but some of the functions in the index.js file were modified to work along with the back-end.

Roberto’s Front-end Implementation

In front-end is where I added the fetch calls to a few files to handle the retrieval and storage of messages per post.

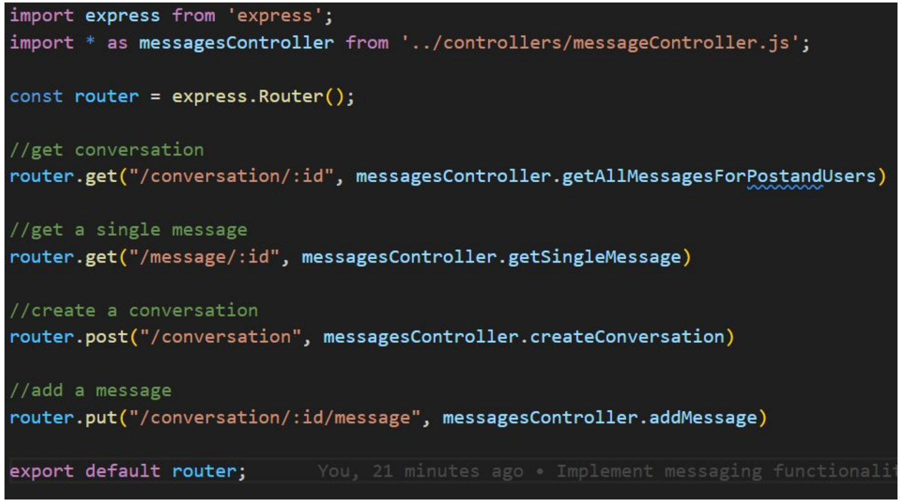
I also had to modify a few files, including the CSS for better styling.

Also, I added a lot of error handling to figure out bugs that were causing the messaging page to fail.



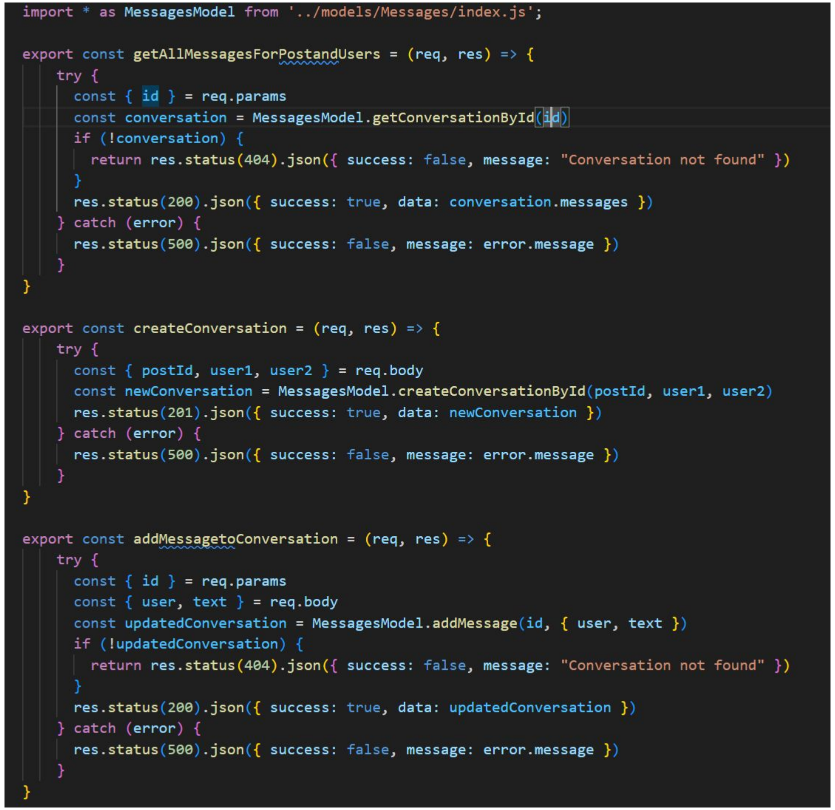
```
try {
    const currentUserId = localStorage.getItem('userId') || '101';
    const postOwnerId = this.#currentPost.user_id;
                const conversationId = `${this.#currentPost.id}-${currentUserId}-${postOwnerId}`;

                const response = await fetch('http://localhost:3000/api/conversations/conversation', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            postId: this.#currentPost.id,
            user1: currentUserId,
            user2: postOwnerId
        })S      You, 1 second ago • Uncommitted changes
    });

    const responseData = await response.json();
    if (!response.ok) {
        throw new Error(responseData.message || 'Failed to create conversation');
    }

    localStorage.setItem('activeConversationId', conversationId);
    const hub = EventHub.getEventHubInstance();
    hub.publish(Events.NavigateTo, '/MessagingPage');
} catch (error) {
    console.error('Error creating conversation:', error);
    alert('Failed to start conversation. Please try again.');
}
```

Roberto’s Back-end Implementation



```
import express from 'express';
import * as messagesController from '../controllers/messageController.js';

const router = express.Router();

//get conversation
router.get("/conversation/:id", messagesController.getAllMessagesForPostandUsers)

//get a single message
router.get("/message/:id", messagesController.getSingleMessage)

//create a conversation
router.post("/conversation", messagesController.createConversation)

//add a message
router.put("/conversation/:id/message", messagesController.addMessage)

export default router;        You, 21 minutes ago • Implement messaging functionalit
```

I implemented the Back-end for the messaging page, routes to redirect fetches, and functions to interact with the conversation and message classes.



```
import * as MessagesModel from '../models/Messages/index.js';

export const getAllMessagesForPostandUsers = (req, res) => {
    try {
        const { id } = req.params
        const conversation = MessagesModel.getConversationById(id)
        if (!conversation) {
            return res.status(404).json({ success: false, message: "Conversation not found" })
        }
        res.status(200).json({ success: true, data: conversation.messages })
    } catch (error) {
        res.status(500).json({ success: false, message: error.message })
    }
}

export const createConversation = (req, res) => {
    try {
        const { postId, user1, user2 } = req.body
        const newConversation = MessagesModel.createConversationById(postId, user1, user2)
        res.status(201).json({ success: true, data: newConversation })
    } catch (error) {
        res.status(500).json({ success: false, message: error.message })
    }
}

export const addMessagetoConversation = (req, res) => {
    try {
        const { id } = req.params
        const { user, text } = req.body
        const updatedConversation = MessagesModel.addMessage(id, { user, text })
        if (!updatedConversation) {
            return res.status(404).json({ success: false, message: "Conversation not found" })
        }
        res.status(200).json({ success: true, data: updatedConversation })
    } catch (error) {
        res.status(500).json({ success: false, message: error.message })
    }
}
```

Roberto’s Challenges and Insights

One challenge I faced during this milestone was connecting the frontend and backend through the fetch routes because the conversations weren’t loading properly. This is why I had to change a great amount of the frontend’s code for the messaging page.

Another challenge I faced was finishing in time because I had to catch up on the Backends course material and learn as many techniques as possible to replicate for our website.

I found solution to my problems by talking with my teammates and separating all the features; to not cause any conflicts between the different pages of the website.

Finally, I had so much trouble getting the messaging page. For some reason, I couldn’t figure out, the messaging page stopped working once I tried to join the backend with the frontend.

Roberto’s Future Improvements and Next Steps

First, I need to get the messaging page working again. Time wasn’t the issue, I just couldn’t understand what was going on. After so many changes the messaging page stopped working in every aspect.

In the future, the messaging system will be optimized. This means I’ll try to figure out better algorithms for saving some messages locally, and I’ll most likely add more features to the messaging system, which users will use to better interact with others.

The next step for backend is to upgrade to an actual database, try to create users in different computers, and get them to interact with each other. This would close the loop on backend and frontend for the messaging page because messages will be sent, received, stored, and dynamically displayed.

Nathan Dennis' Work Summary

I worked on the following:

Created adminController https://github.com/realraft/UMass-Lost-and-Found/issues/160

Made real-time changes to HomePageSignedIn using AdminPage https://github.com/realraft/UMass-Lost-and-Found/issues/130

Created new route for admin back end https://github.com/realraft/UMass-Lost-and-Found/issues/154

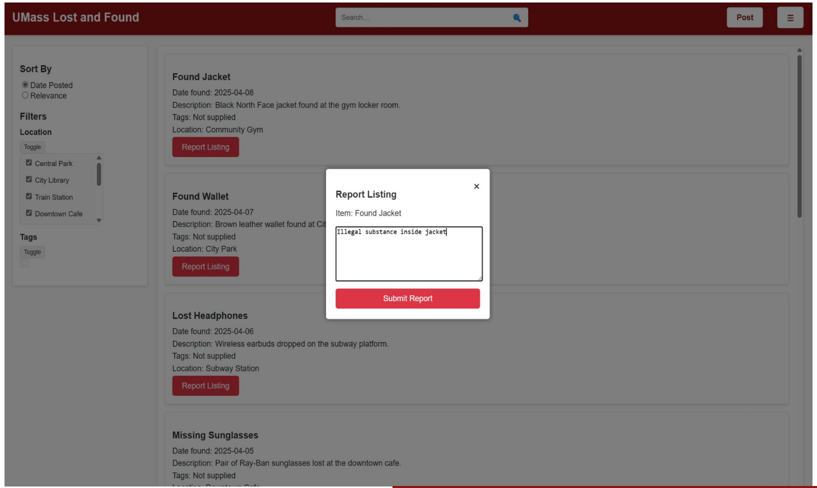
Interacting with Express.js for interacting with server https://github.com/realraft/UMass-Lost-and-Found/issues/140

I worked on using CRUD with the back end server to dynamically review, keep, and delete posts that are in the home page, from the admin page. Posts also keep and retain administrator comments in the back end server for each post. A post that is kept does not update the home page, but a post that is deleted updates the home page by deleting the reported post.

Nathan Dennis’ Feature Demonstration

- Usage of CREATE to add new comments to posts to upload to server
- Usage of READ to read and display reported posts
- Usage of UPDATE to edit existing comments in the posts, as well as mark posts for being kept
- Usage of DELETE to remove posts from both the listings page and the administrator page
- Real-time update of the Administrator page when a post is marked as reported
- Real-time update of the listings page when a post is deleted from the Administrator page
- Comments in reported posts remain persistent between sessions

Nathan Dennis’ Code Explanation (Screenshots of site)



UMass Lost and Found

Search...

Post

Sort By

(●) Date Posted
( ) Relevance

Filters

Location

Toggle

- ☑ Central Park
- ☑ City Library
- ☑ Train Station
- ☑ Downtown Cafe

Tags

Toggle

Found Jacket
Date found: 2025-04-08
Description: Black North Face jacket found at the gym locker room.
Tags: Not supplied
Location: Community Gym
Report Listing

Found Wallet
Date found: 2025-04-07
Description: Brown leather wallet found at Cit
Tags: Not supplied
Location: City Park
Report Listing

Lost Headphones
Date found: 2025-04-06
Description: Wireless earbuds dropped on the subway platform.
Tags: Not supplied
Location: Subway Station
Report Listing

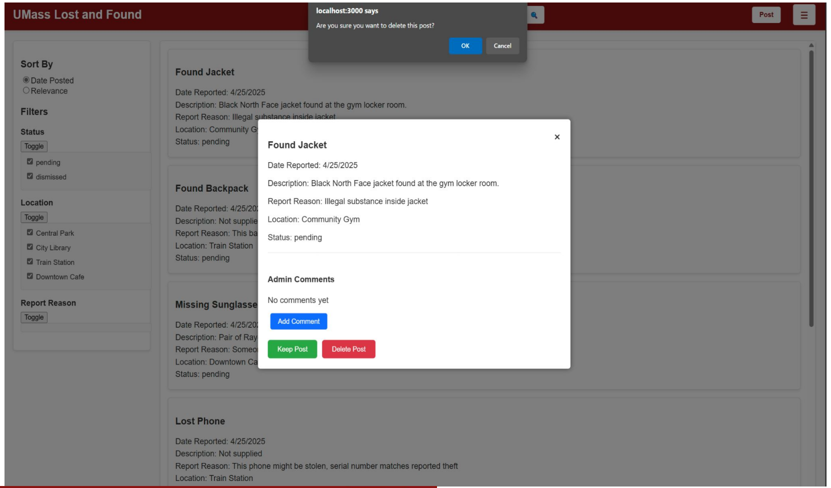
Missing Sunglasses
Date found: 2025-04-05
Description: Pair of Ray-Ban sunglasses lost at the downtown cafe.
Tags: Not supplied

Report Listing ×

Item: Found Jacket

```
Illegal substance inside jacket
```

Submit Report



UMass Lost and Found

Post

localhost:3000 says
Are you sure you want to delete this post?
OK Cancel

Sort By

(●) Date Posted
( ) Relevance

Filters

Status

Toggle

- ☑ pending
- ☑ dismissed

Location

Toggle

- ☑ Central Park
- ☑ City Library
- ☑ Train Station
- ☑ Downtown Cafe

Report Reason

Toggle

Found Jacket
Date Reported: 4/25/2025
Description: Black North Face jacket found at the gym locker room.
Report Reason: Illegal substance inside jacket
Location: Community G
Status: pending

Found Backpack
Date Reported: 4/25/20
Description: Not supplie
Report Reason: This ba
Location: Train Station
Status: pending

Missing Sunglasse
Date Reported: 4/25/20
Description: Pair of Ray
Report Reason: Someo
Location: Downtown Ca
Status: pending

Lost Phone
Date Reported: 4/25/2025
Description: Not supplied
Report Reason: This phone might be stolen, serial number matches reported theft
Location: Train Station

Found Jacket ×

Date Reported: 4/25/2025

Description: Black North Face jacket found at the gym locker room.

Report Reason: Illegal substance inside jacket

Location: Community Gym

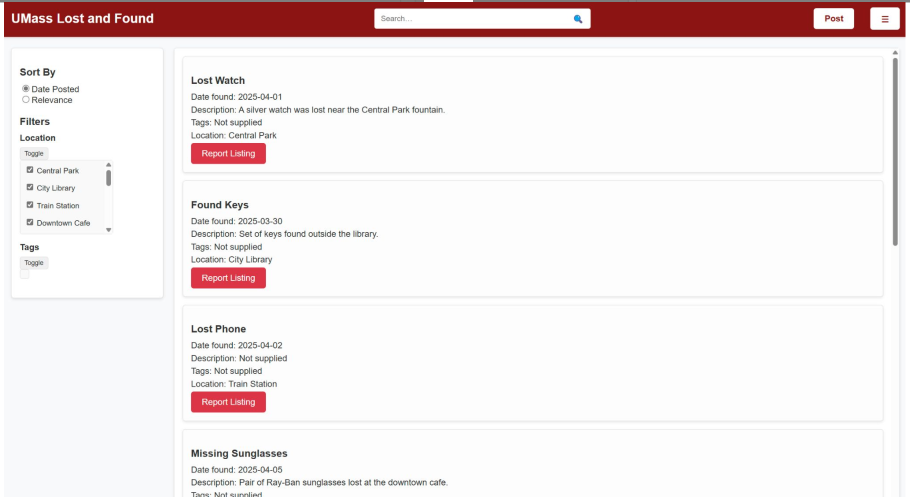
Status: pending

Admin Comments

No comments yet

Add Comment

Keep Post Delete Post



UMass Lost and Found

Search...

Post

Sort By

(●) Date Posted
( ) Relevance

Filters

Location

Toggle

- ☑ Central Park
- ☑ City Library
- ☑ Train Station
- ☑ Downtown Cafe

Tags

Toggle

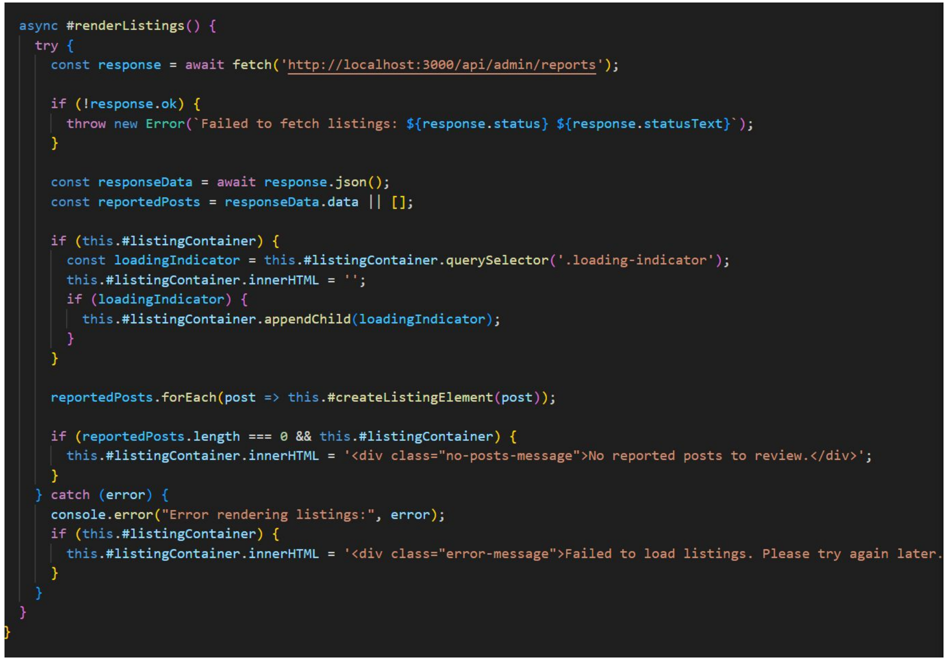
Lost Watch
Date found: 2025-04-01
Description: A silver watch was lost near the Central Park fountain.
Tags: Not supplied
Location: Central Park
Report Listing

Found Keys
Date found: 2025-03-30
Description: Set of keys found outside the library.
Tags: Not supplied
Location: City Library
Report Listing

Lost Phone
Date found: 2025-04-02
Description: Not supplied
Tags: Not supplied
Location: Train Station
Report Listing

Missing Sunglasses
Date found: 2025-04-05
Description: Pair of Ray-Ban sunglasses lost at the downtown cafe.
Tags: Not supplied

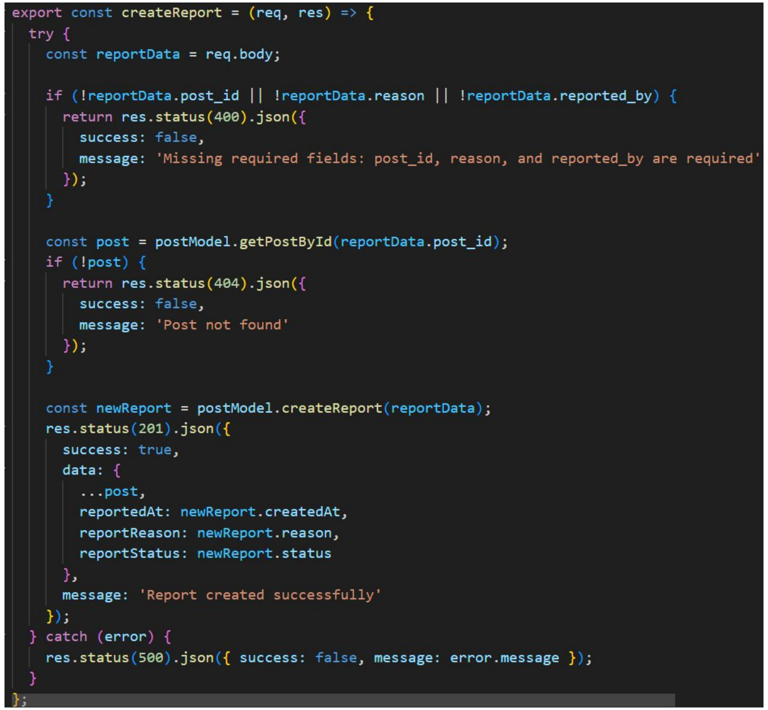
Nathan Dennis’ Code Explanation (Front End)



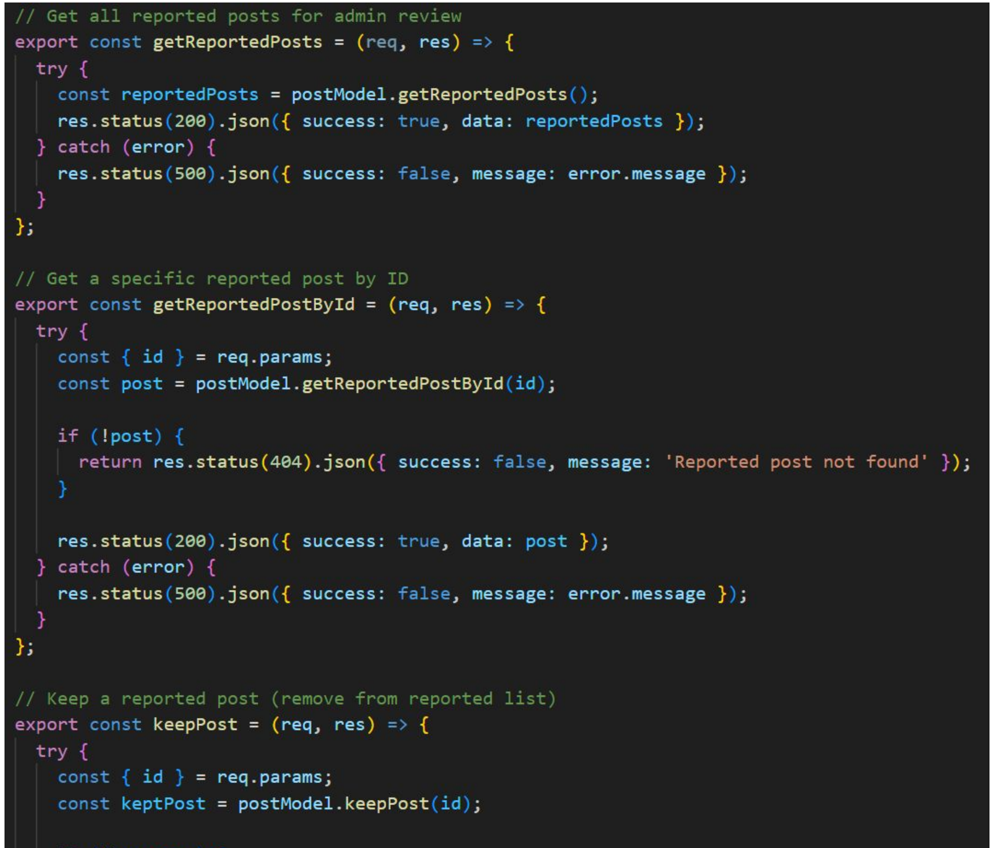
```
  async #renderListings() {
    try {
      const response = await fetch('http://localhost:3000/api/admin/reports');

      if (!response.ok) {
        throw new Error(`Failed to fetch listings: ${response.status} ${response.statusText}`);
      }

      const responseData = await response.json();
      const reportedPosts = responseData.data || [];

      if (this.#listingContainer) {
        const loadingIndicator = this.#listingContainer.querySelector('.loading-indicator');
        this.#listingContainer.innerHTML = '';
        if (loadingIndicator) {
          this.#listingContainer.appendChild(loadingIndicator);
        }
      }

      reportedPosts.forEach(post => this.#createListingElement(post));

      if (reportedPosts.length === 0 && this.#listingContainer) {
        this.#listingContainer.innerHTML = '<div class="no-posts-message">No reported posts to review.</div>';
      }
    } catch (error) {
      console.error("Error rendering listings:", error);
      if (this.#listingContainer) {
        this.#listingContainer.innerHTML = '<div class="error-message">Failed to load listings. Please try again later.
      }
    }
  }
}
```

This snippet of the code allows for posts that are reported to be read from the server. The posts have the same data as the listings page, and there is basic error messages for failing to retrieve posts

Nathan Dennis’ Code Screenshots (Back End)



```javascript
export const createReport = (req, res) => {
  try {
    const reportData = req.body;

    if (!reportData.post_id || !reportData.reason || !reportData.reported_by) {
      return res.status(400).json({
        success: false,
        message: 'Missing required fields: post_id, reason, and reported_by are required'
      });
    }

    const post = postModel.getPostById(reportData.post_id);
    if (!post) {
      return res.status(404).json({
        success: false,
        message: 'Post not found'
      });
    }

    const newReport = postModel.createReport(reportData);
    res.status(201).json({
      success: true,
      data: {
        ...post,
        reportedAt: newReport.createdAt,
        reportReason: newReport.reason,
        reportStatus: newReport.status
      },
      message: 'Report created successfully'
    });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};
```



```javascript
// Get all reported posts for admin review
export const getReportedPosts = (req, res) => {
  try {
    const reportedPosts = postModel.getReportedPosts();
    res.status(200).json({ success: true, data: reportedPosts });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

// Get a specific reported post by ID
export const getReportedPostById = (req, res) => {
  try {
    const { id } = req.params;
    const post = postModel.getReportedPostById(id);

    if (!post) {
      return res.status(404).json({ success: false, message: 'Reported post not found' });
    }

    res.status(200).json({ success: true, data: post });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

// Keep a reported post (remove from reported list)
export const keepPost = (req, res) => {
  try {
    const { id } = req.params;
    const keptPost = postModel.keepPost(id);
```

Nathan Dennis’ Code Explanation (Back End)

adminController.JS checks and validates that a reported post has a post ID, a reason for reporting. If checked, a new data object is created containing all of the properties of the original post, and the user is informed of the post being successful. postModel is then called which fetches the reports and produces an error if there is an internal server error. It then validates the ID. The final part of the code snippet checks the id of the post and returns an error if it cannot find the post.

Nathan Dennis’ challenges and insights

The most significant challenge of this project were my attempts at updating the Admin Page in real time with reported posts. I didn’t realize at the time that it was due to the page being refreshed which updated it, which was then promptly fixed. Having a back-end server for persistence is a better way to store comments in dynamic objects as well than attempting to used IndexedDB, which ended in failure during the previous milestone.

Nathan Dennis’ Future improvements

Optimization of reports page for easier readability and reliability

https://github.com/realraft/UMass-Lost-and-Found/issues/183

Utilization of SQLite and Sequelize into current implementation

https://github.com/realraft/UMass-Lost-and-Found/issues/185

Checking numerous reported posts at once to be kept or deleted

https://github.com/realraft/UMass-Lost-and-Found/issues/187

Send admin comments to user page for item

https://github.com/realraft/UMass-Lost-and-Found/issues/184