# UMass Lost and Found Milestone 7

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 #7:
https://github.com/realraft/UMass-Lost-and-Found/releases/tag/milestone-6-submission

# UMass-Lost-and-Found Builders

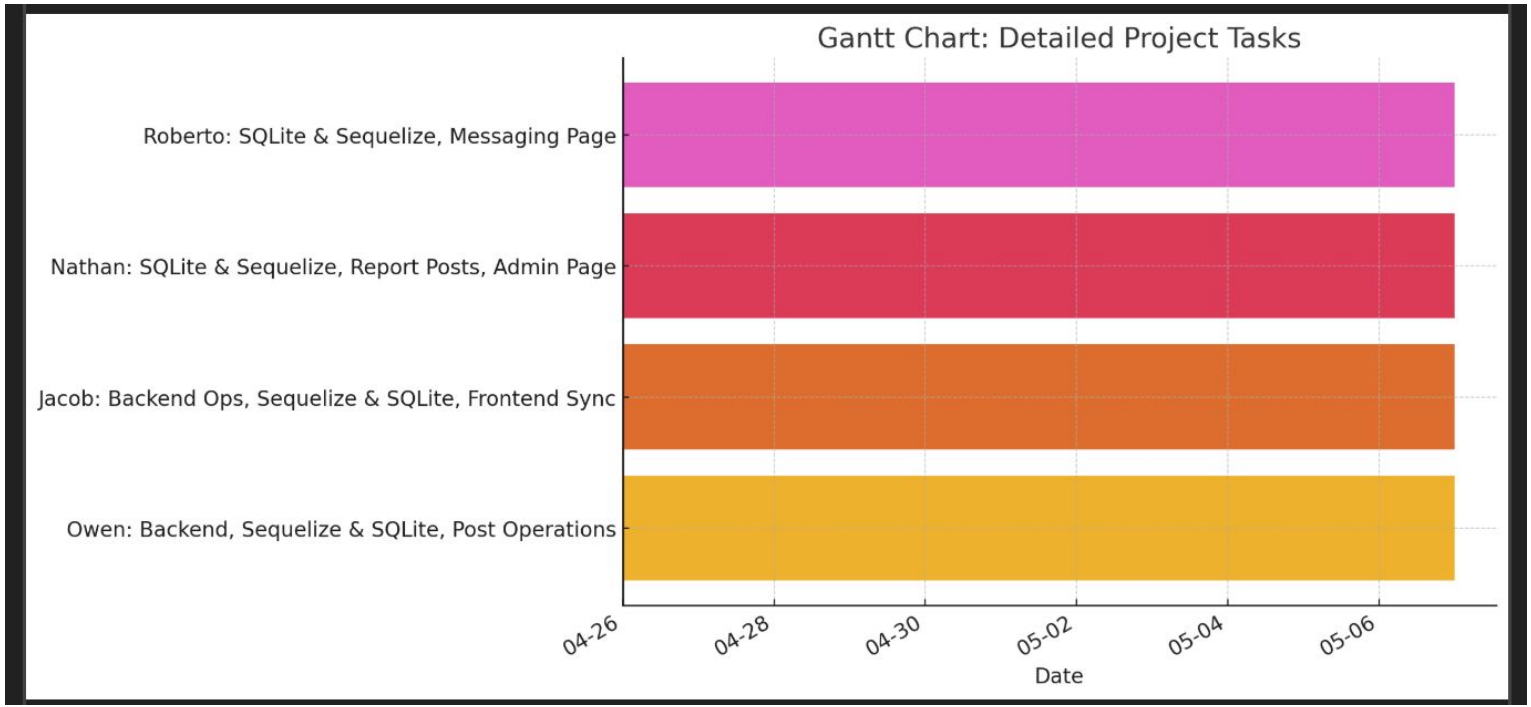
Owen Raftery
Role(s): Project-Manager
Role(s) Description: Will be leading the team during meetings and give instructions on what to code.
Issues from current milestone: Implement basic SQLite and post CRUD operations.

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): Time manager
Role(s) Description: Synced the frontend with backend through creating post operations for deletion of posts using sequelize and integrating these changes for out frontend to work with SQLite
Roberto Rubiot
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 7



# Jacob Taylor Work Summary

**Issues:**

https://github.com/realraft/UMass-Lost-and-Found/issues/218

https://github.com/realraft/UMass-Lost-and-Found/issues/216

https://github.com/realraft/UMass-Lost-and-Found/issues/195

**PR's:**https://github.com/realraft/UMass-Lost-and-Found/pull/204

https://github.com/realraft/UMass-Lost-and-Found/pull/223

https://github.com/realraft/UMass-Lost-and-Found/pull/217

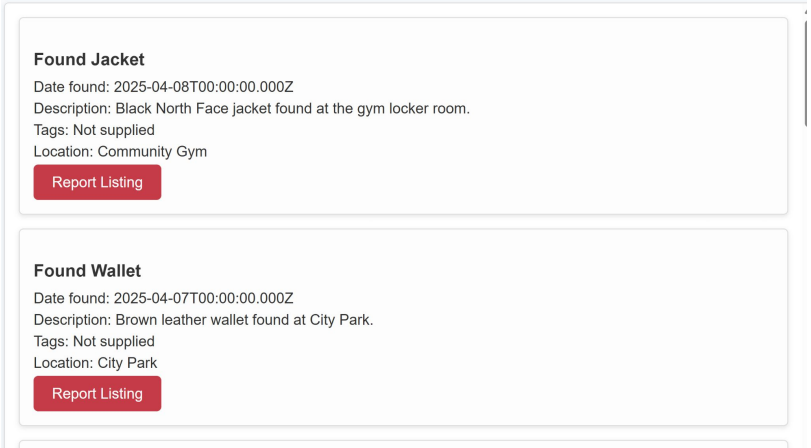
In this milestone I worked in  216-fix-bug-issue-in-new-posts,218-report-page-bug-fix, and 195hompagesignedin-to-use-sqlite-and-sequelize. I worked on implementing the frontend to interact with the new changes made with the backend. I worked on the delete operations in postOperations.js using sequelize and used operations that were implemented by others to sync with the frontend. I also worked on some bug issues related to a creation of a new post, where the database was not creating usersids for different users, so the creation of new posts was failing.

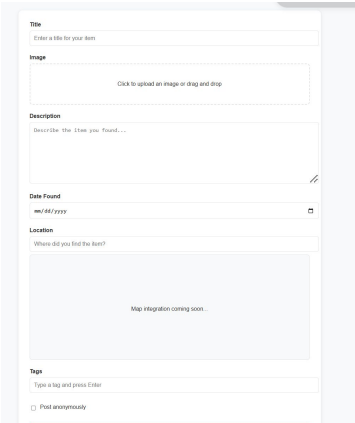
# Feature Demonstration Jacob Taylor

**Backend branch: 195hompagesignedin-to-use-sqlite-and-sequelize,**

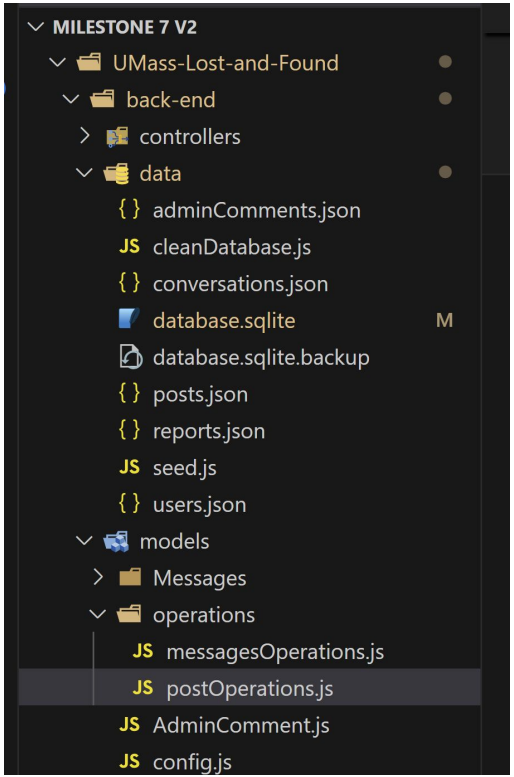
**Frontend Branch:** `218-report-page-bug-fix`

**My features were implementing the frontend integration of the implemented backend sequelize and sqlite operations. I made it so the frontend end was compatible with these new features. I fetched to the backend to update the the homepagesigned with listings, and creation of a new post, by when a user clicks creates a new post it, it creates a new user id, and populates the database with the data.**



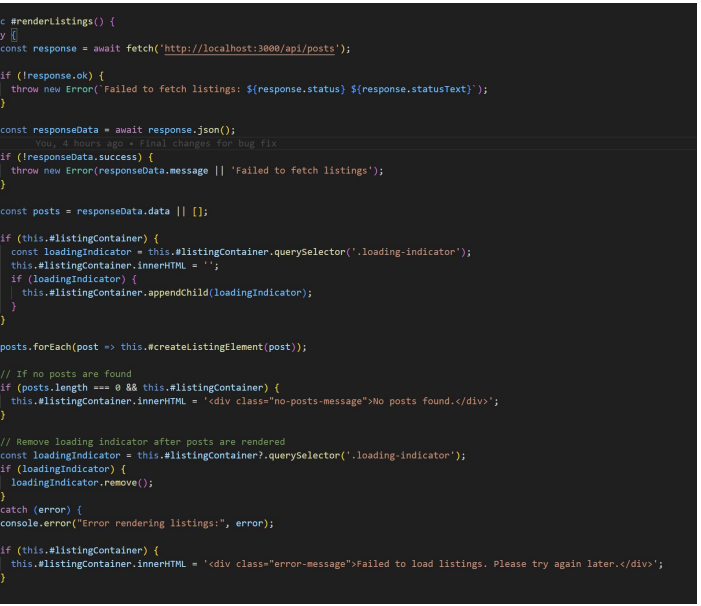
# Code Structure and Organization Jacob Taylor




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

Our code is easy to maintain and read with this split up approach. As our frontend files do not touch the backend folder directly. I was tasked with making the delete operations for the database using sequelize in backend/models/operations/postOperations.js. The operations are defined in separate files for maintainability.

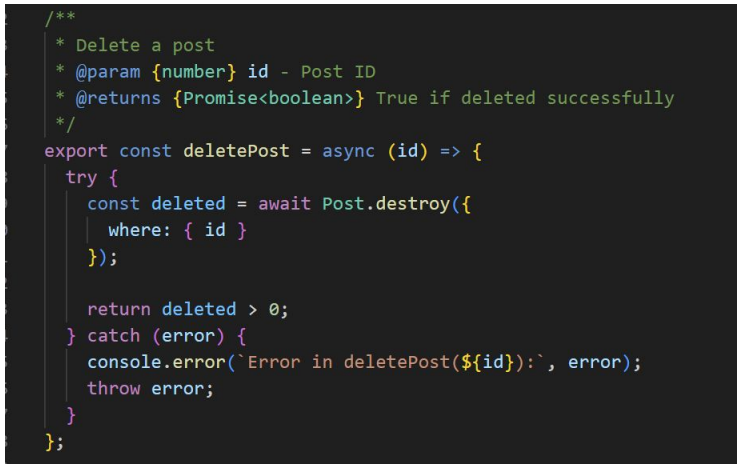
# Frontend Implementation Jacob Taylor



```
c #renderListings() {
y {
const response = await fetch('http://localhost:3000/api/posts');

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

const responseData = await response.json();
      You, 4 hours ago • Final changes for bug fix
if (!responseData.success) {
  throw new Error(responseData.message || 'Failed to fetch listings');
}

const posts = responseData.data || [];

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

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

// If no posts are found
if (posts.length === 0 && this.#listingContainer) {
  this.#listingContainer.innerHTML = '<div class="no-posts-message">No posts found.</div>';
}

// Remove loading indicator after posts are rendered
const loadingIndicator = this.#listingContainer?.querySelector('.loading-indicator');
if (loadingIndicator) {
  loadingIndicator.remove();
}
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.</div>';
}
```

Frontend implementation for rendering the listings on homepagesignedin, it fetches from the backend and loads the post for the user to see. Uses error messages when something is wrong from the backend.

# Backend implementation Jacob Taylor



```
/**
 * Delete a post
 * @param {number} id - Post ID
 * @returns {Promise<boolean>} True if deleted successfully
 */
export const deletePost = async (id) => {
  try {
    const deleted = await Post.destroy({
      where: { id }
    });

    return deleted > 0;
  } catch (error) {
    console.error(`Error in deletePost(${id}):`, error);
    throw error;
  }
};
```

Using Sequelize implemented the delete operation. It finds the post by its ID and removes it from the database if successful. On error, its logs to the console and throws an error message for information about the error.

# Challenges and insights Jacob Taylor

The biggest issue we faced as a group was handling the userId for creating posts. It was a huge obstacle because we had numerous errors related to the userId. I also learned that we need to document our code, as other group members had trouble understanding what some group mates had done.  Overall I learned how to communicate effectively with my team members, and learned how to work together on a large codebase and project.

# Future Improvements and next steps Jacob Taylor

1. Implement the  google maps api and research the costs and benefits that the feature could add for user preferences

   https://github.com/realraft/UMass-Lost-and-Found/issues/116?issue=realraft%7CUMass-Lost-and-Found%7C118

2. Implement a video upload feature, so users can send a video of a possible lost item

   https://github.com/realraft/UMass-Lost-and-Found/issues/236

3. Implement authentication so only UMass students can use our site.

   https://github.com/realraft/UMass-Lost-and-Found/issues/237

# Owen Raftery Work Summary

Details: I implemented the basic structure of SQLite for the website. This included creating the database, installing dependencies, creating the DB structure for every datatype we need, and defining FK and PK constraints. I then implemented the CRUD operations for post related tasks.

Issues:

- post CRUD · Issue #234 · realraft/UMass-Lost-and-Found
- Swap Post manager and home page signed in to use SQLite and Sequelize · Issue #194 · realraft/UMass-Lost-and-Found
- Create SQLite database · Issue #190 · realraft/UMass-Lost-and-Found

PRs:

- added basic/general sqlite implementation by realraft · Pull Request #196 · realraft/UMass-Lost-and-Found
- implemented posts CRUD by realraft · Pull Request #197 · realraft/UMass-Lost-and-Found
- added default userId by realraft · Pull Request #233 · realraft/UMass-Lost-and-Found

# Owen Raftery Feature Demonstration

- SQLite Implementation
  - Dependencies installed
  - Database created
  - FK and PK constraints defined
  - Backend DB usage created
- Post CRUD operations
  - Defined functions that perform crud operations for the posts table in the DB

# Owen Raftery Front End

N/A

# Owen Raftery Back End

implemented the basic structure of SQLite for the website. This included creating the database, installing dependencies, creating the DB structure for every datatype we need, and defining FK and PK constraints. I then implemented the CRUD operations for post related tasks.

# Owen Raftery Challenges and Insights

# Owen Raftery Future Improvements and Next Steps

- Map flag implementation
  - Fix potential bug issues that may arise when we implement the google maps api. · Issue #170 · realraft/UMass-Lost-and-Found

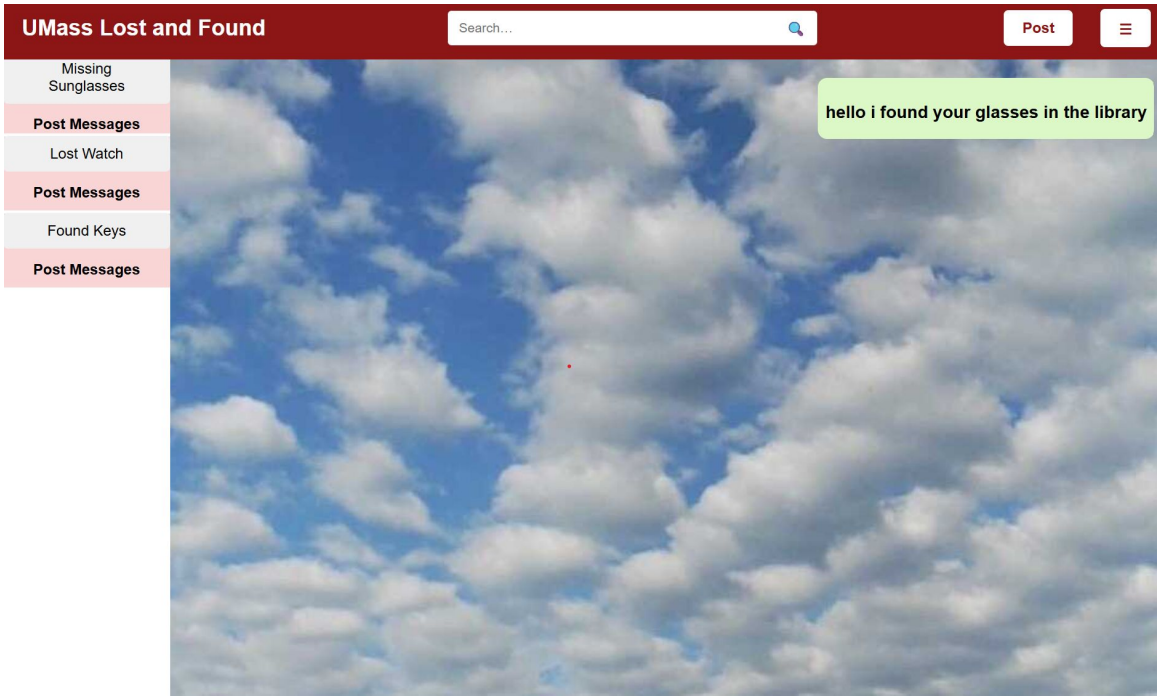
# Roberto’s Work Summary

| | |
|---|---|
| What I worked on this milestone is transition the backend to use sequelize and the sqlite database.<br><br>I changed all files related to messages in the backend and the index.js for messaging page and posteditempage in the frontend.<br><br>I had to change some errors I hadn’t changed from previous milestones.<br><br>I have around 12 PR for this milestone, some are not very relevant cause only one file was barely changed.<br><br>I worked on two branches: messagesFix and messagingPagePersistance | Issues, branches and PR:<br><br>• https://github.com/realraft/UMass-Lost-and-Found/issues/149<br>• https://github.com/realraft/UMass-Lost-and-Found/issues/124<br><br>• https://github.com/realraft/UMass-Lost-and-Found/tree/messagingPagePersistance<br>• https://github.com/realraft/UMass-Lost-and-Found/tree/messagesFix<br><br>• https://github.com/realraft/UMass-Lost-and-Found/pull/224<br>• https://github.com/realraft/UMass-Lost-and-Found/pull/226 |

# Roberto’s Feature Demonstration

The current user is able to create new conversations and send messages to other users.

The messages are stored in the database, so closing and refreshing the page doesn’t delete these objects.

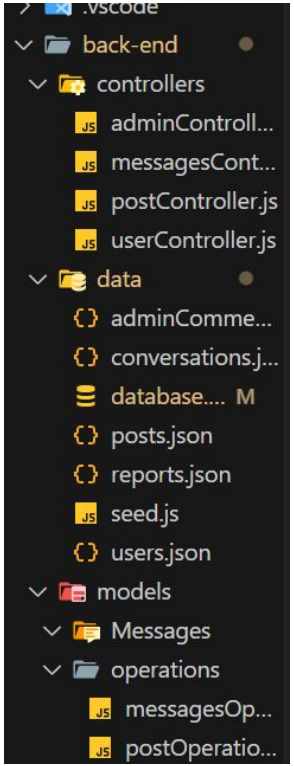


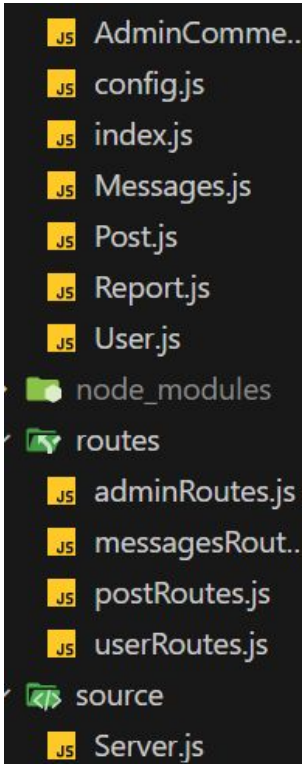
# Roberto’s Code Structure and Organization

The files in the backend are stored in folders separating data, controllers, operations, models, routes, and server.

The frontend is separated by pages because of the multiview structure.

Most of the files I changed are in backend and these are named with words related to messages.



# Roberto’s Front-end Implementation

Frontend implementation for creating conversations with other users when clicking contact finder on a posted item page.

The fetch sends a request to create a new conversation and assigns it the postid and the two user ids.

```
contactButton.addEventListener('click', async () => {
    try {
        const response = await fetch(`/api/conversation/ids/${this.#currentPost.id}/${2}/${this.#currentPost.user_id}`,
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
        });

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

This a sequelize function for getting all conversations for a specific user. Conversations belong to two users, so it searches for both when the first or the second user is the current.

It also choose what attributes to include and return, for both the conversation and for each message.

The other functions are to create a conversation and to add a message.

```
/**
 * Get all conversations for a user.
 * @param {number} id - conversation ID
 * @returns {Promise<Object[]>} Conversation object[]
 */
const getAllConversationsforUserId = async (userId) => {
    try {
        console.log('Fetching conversations for userId:', userId);
        const conversations = await Conversation.findAll({
            where: {
                [Op.or]: [
                    { user1_id: userId },
                    { user2_id: userId }
                ]
            },
            include: [
                {
                    model: User,
                    as: 'User1',
                    attributes: ['id', 'username', 'email']
                },
                {
                    model: User,
                    as: 'User2',
                    attributes: ['id', 'username', 'email']
                },
                {
                    model: Message,
                    as: 'messages',
                    attributes: ['id', 'user_id', 'text', 'createdAt'],
                    include: [
                        {
                            model: User,
                            as: 'sender',
                            attributes: ['id', 'username', 'email']
                        }
                    ]
                }
            ],
            order: [
                ['updatedAt', 'DESC'],
                [{ model: Message, as: 'messages' }, 'createdAt', 'ASC']
            ]
        });
        return conversations;
    } catch (error) {
        throw new Error(`Error getting the conversation for user ${userId}: ${error.message}`);
    }
};
```

# Roberto’s Challenges and Insights

This milestone I had to overcome many obstacles. I had so many mistakes to fix, and I had to go one by one coding all these again. To tackle these problems I used console errors and logs to figure out what functions were causing the messaging page to crash.

After a while, it started to work bit by bit. Sometimes when fixing a programming error I did in a previous milestone I encountered even more problems because this new code wasn’t well connected to the old code.

One of my biggest challenges was the routes because I misunderstood what to name these.

One problem I had, was setting everything to belong to a specific user because we don’t have any authentication.

# Roberto’s Future Improvements and Next Steps

Some key future improvements are to code more functions related to conversations and messages. Like for the admin to delete conversations or if a post is taken down all conversations related to it disappear.

Also, with authentication many users can log onto the website and actually communicate with each other. As long as the database isn’t overridden the messages are stored somewhere for all users to access.

Finally, the messaging service should be optimized because if a huge amount of users had conversations, the fetches will get extremely expensive and the server would block requests.

https://github.com/realraft/UMass-Lost-and-Found/issues/178

https://github.com/realraft/UMass-Lost-and-Found/issues/232

# Nathan Dennis' Work Summary

I worked on the following:

Utilize SQLite and Sequelize for changing the back-end functionality for Admin and Reporting

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

Optimized Report Page

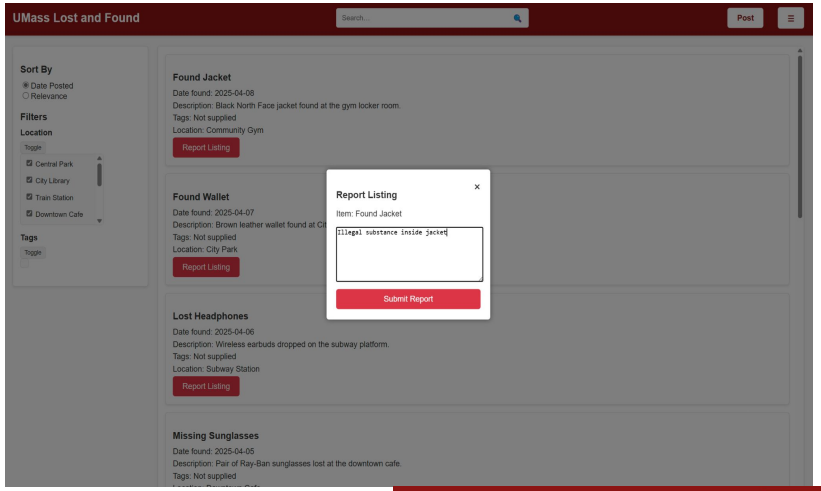
https://github.com/realraft/UMass-Lost-and-Found/issues/183

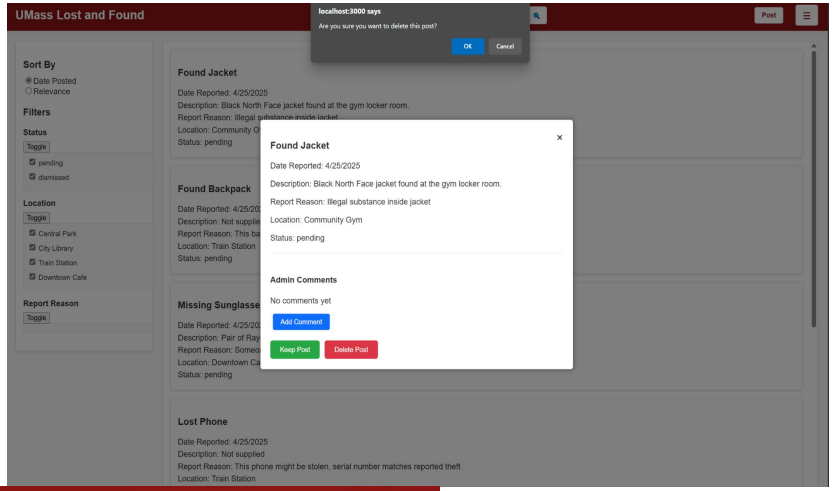
Not much has changed since the previous milestone. All which was done was converting the code implementation from json to SQLite And Sequelize, with the same functionality as what was done with the .json server. Errors with how SQL was handling prevented further functionality for us than what was already implemented, other than converting it to SQLite and Sequelize.

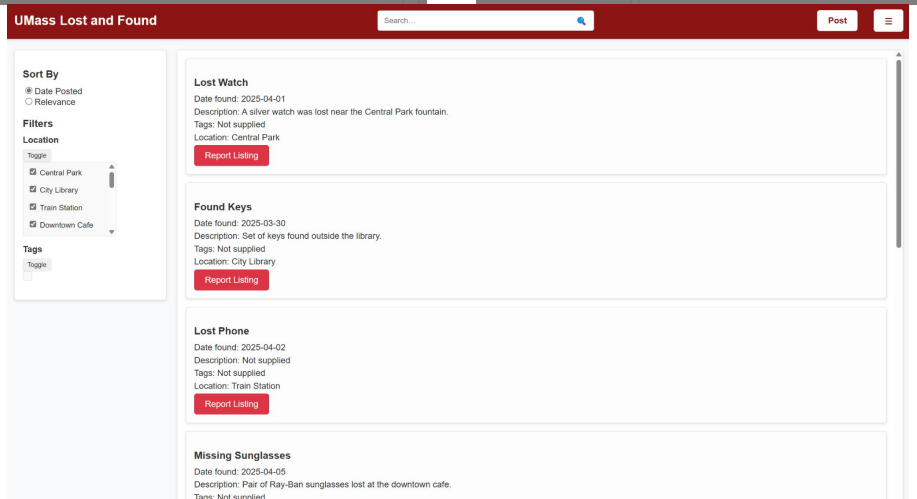
# Nathan Dennis’ Feature Demonstration

- Usage of SQLite and Sequelize for posts being reported to and from a server
- 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

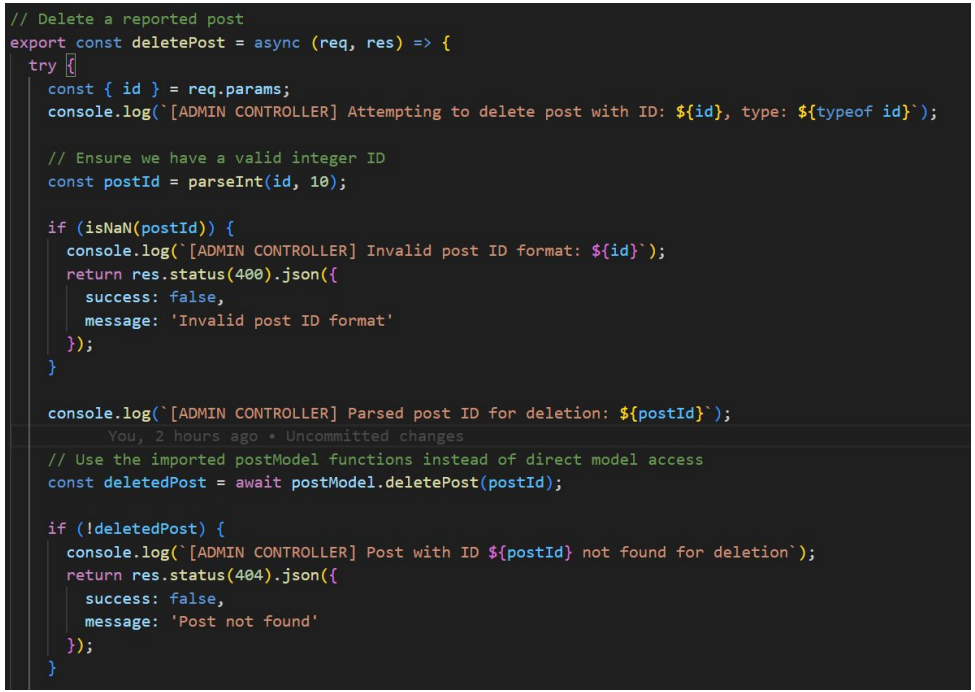
# Nathan Dennis' Code Explanation (Screenshots of site)



# Nathan Dennis' Code Explanation (Back End)



```
// Delete a reported post
export const deletePost = async (req, res) => {
  try {
    const { id } = req.params;
    console.log(`[ADMIN CONTROLLER] Attempting to delete post with ID: ${id}, type: ${typeof id}`);

    // Ensure we have a valid integer ID
    const postId = parseInt(id, 10);

    if (isNaN(postId)) {
      console.log(`[ADMIN CONTROLLER] Invalid post ID format: ${id}`);
      return res.status(400).json({
        success: false,
        message: 'Invalid post ID format'
      });
    }

    console.log(`[ADMIN CONTROLLER] Parsed post ID for deletion: ${postId}`);
          You, 2 hours ago • Uncommitted changes
    // Use the imported postModel functions instead of direct model access
    const deletedPost = await postModel.deletePost(postId);

    if (!deletedPost) {
      console.log(`[ADMIN CONTROLLER] Post with ID ${postId} not found for deletion`);
      return res.status(404).json({
        success: false,
        message: 'Post not found'
      });
    }
```

Requires validation for a post ID which is required for primary and foreign keys. Usage of imported model calls for deletion of posts. Ensuring posts and reports are Deleted in proper order for proper SQLite integrity

# Nathan Dennis’ challenges and insights

The most challenging part of this were minor issues, mainly regarding user ID, which was resolved in a console on the user’s end, when functionality failed otherwise. Wasted a lot of time trying to figure out that issue and not knowing about it. Otherwise, it was more or less straightforward conversion from utilization of a json server to SQLite and Sequelize

# Nathan Dennis’ Future improvements

Authentication to check if it was an administrator (cancelled, but will consider doing)

https://github.com/realraft/UMass-Lost-and-Found/issues/186

Select and check numerous posts at the same time
https://github.com/realraft/UMass-Lost-and-Found/issues/187

Send comments to user page for reported items

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