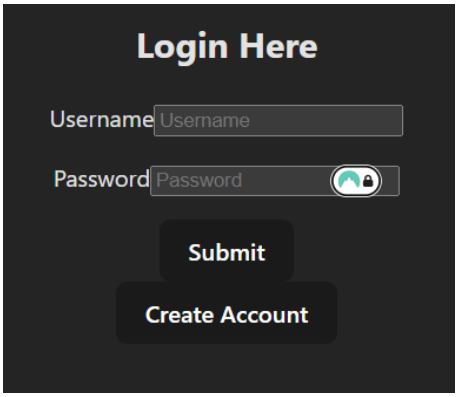
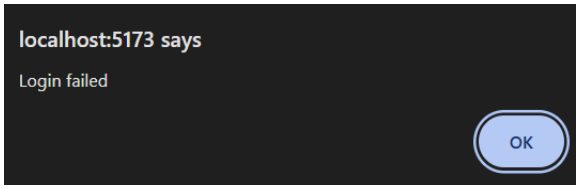
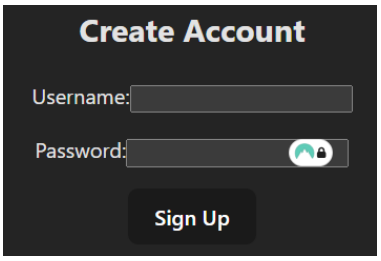
Implemented

Login Page: The user can log in with a username and password. If the user does not have an account, they will be notified with an alert when they try to log in. Then, they can click Create Account to create an account. They then can log in. There is an admin account with the username “admin” and the password “admin.”

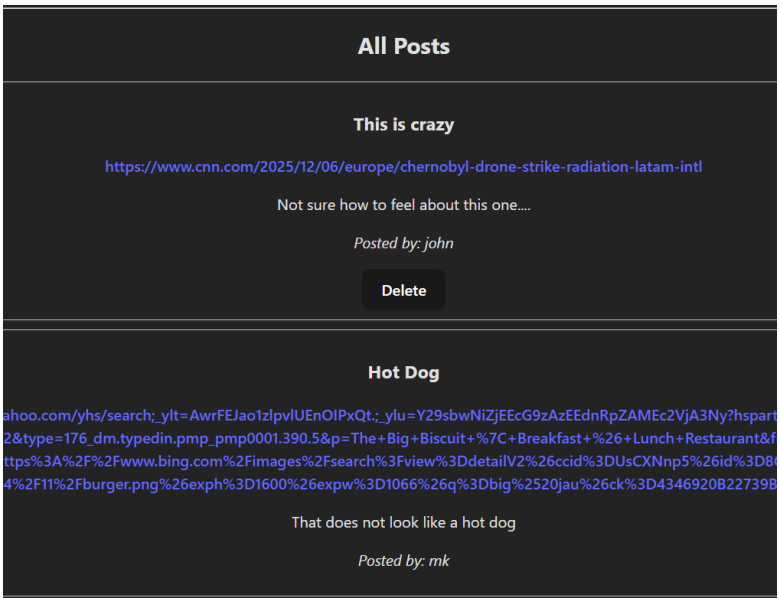




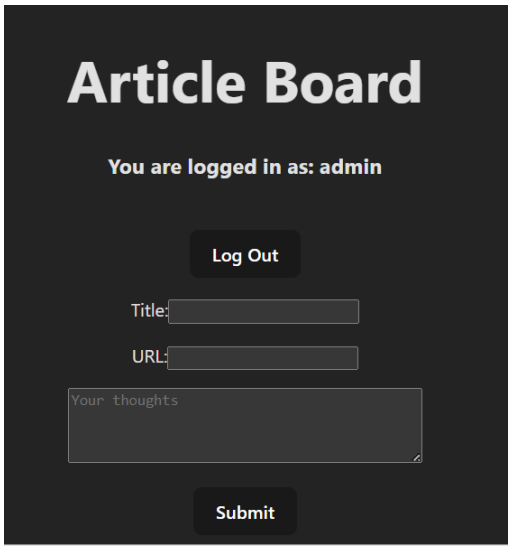


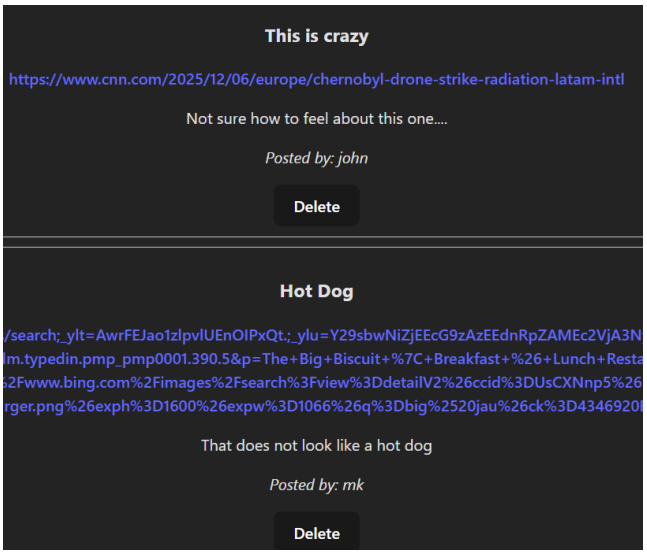
—----------

Board Page: Once logged in, a user can create a post. For our posts, we require a Title and a section to write out your thoughts. Below you can see an example of all the posts. We are logged in as john, so john can only delete his posts.



Admin Account: If a user is logged in as the admin, they can delete all posts on the message board.





Parameterized Queries: We handle SQL Injections by using Parameterized Queries. This prevents A05: Injection in the OWASP Top Ten.

```
// -------------------- LOGIN --------------------
app.post("/login", async (req, res) => {
    const { username, pw } = req.body;

    const user = await db.get(
        "SELECT * FROM users WHERE username = ? AND password = ?",
        [username, pw]
    );
```

Not Implemented

Because of our unfamiliarity with React and Node JS, we did not get as much implemented as we would have liked. Within the 20-hour time limit, we were only able to get a bare bones Web App working properly. Because of this, there were many security features that we would want to implement in the future that we were unable to.

Password Hashing: As of now we just store the password into our DB as plaintext. In the future we would want to add some hashing to this to align with Cryptography standards.

Web Tokens: Since we wanted to get our app functioning first, we did not have time to try and implement JWT. In the future this is something we could implement for stateless authentication.

Admin Handling: Our admin account is just a user with “admin” as the username and password. This is extremely insecure. In the future we would change this to be something very complex. Also, our admin status checking is just an else if statement in our backend code, which literally just checks username === "admin" && pw === "admin." It seems like this would be very insecure. We’d probably want to involve some sort of role in our database to handle some of these problems.

Error Handling: We would like to add more robust error handling to let the server know if issues are occurring. If a user has tried to log in 40 times in a row, perhaps they should be timed out. While we use try/catches, we don’t have any sort of console.error() or res.status() messages.