**For security purposes, I implemented the following details:**
-Parametization of the SQL Queries to protect against string injection in the SQL forms. Tried to inject in all possible fields and didn't work!

-Using JWT encoding and decoding for the tokens. This helps with authenticating the user sessions and the account they are logged into.

- Hashed passwords are stored instead of plain text ones, so even if an attacker managed to get the database, it wouldn't be able to identify the passwords used. When logging in, you compare the hash stored with the hash generated for the given password.

- Alert messages to users doing incorrect behaviour (wrong credentials, user already exists, posting regular texts instead of a URL/link, trying to delete a URL that they didn't post)

-Users can delete their own URL but not anyone else's. This is true for all but the admin user, who can delete anyone's.

**What I didn't have time to implement:**

**-**I tried researching how to limit CORS access, but couldn't figure it out in the allotted time. I also don't use security headers.

-I also don't use locked versions of the packages I installed, so if one of them gets modified with malicious behaviour, then it would affect my code as well.

-My errors are all displayed through console logs. After reading the OWASP file, it seems it would have been a good idea to use console errors instead, as we don't want "silent logs". If I had a bit more time, I would have tackled this first.

-Finally, the last thing I can think of is a more extended error system; right now, there are a lot of weird, very unlikely cases in which the code could fail (error in the database, networks are congested, internal errors). These aren't handled. The way I implemented I don't believe that it would crash, but an input might not get handled, and the user wouldn't be notified of it.