Security setup:

The first page, the login page, requires the user to insert a password and username. A new user can be added via the create user page. When a new user is created the password is hashed, including a salt, using the argon2 algorithm. The input password is checked with the stored hashed password associated with the username, when they match, the login endpoint sends a response with a http-only cookie that holds a JWT. This token is generated using a HMAC254 algorithm with a secret key, it also includes a expiration date and a username claim.

With every request the user makes to the backend the cookie with the JWT is included in the http request. The JWT if verified and the username is retrieved to be checked if it exists in the database. If the user exists then the token is validated and the endpoint method is executed. If a request is denied with code 401 the user is automatically logged out.

Security scan:

We did a security scan using OWASP Zap which found two vulnerabilities:

1) Absence of Anti-CSRF Tokens
2) Cross-Domain JavaScript Source File Inclusion

The reason the first vulnerability is given is because the login form doesn't require a CSRF token. This makes our site vulnerable to CSRF attacks. A solution to that would be to send a CSRF token with the login page, which is required when the user sends a login request.
The second one is because we copied a javascript file from an online repository. To fix that would mean rewriting that script but that is a lot of work. Our cookies are http-only, so there shouldn't be any problems of javascript files editing the cookies. There is a possibility of a MITM attack, since we used HTTP instead of HTTPS.

Security against the most common attacks:

- SQL-Injection: There is very little that the user can edit in our site. Only the metadata and the file uploading can be exploited. All our queries use prepared statements so there shouldn't be a problem there. The files can be malicious, but the data is simply put into a stored procedure and sent to the database. The server never interacts with it.
- XSS: The frontend is not as secure as we would like it to be. There is a chance that XSS happens, but we kept user input to a minimum and changes in the server are not possible. The cookies cannot be requested or set using javascript, so the backend is secure
- CSRF: Our application is not secured very well against CSRF. We want to improve this by adding CSRF tokens to all of the server responses.