Security report

Security breach	Covered?
Protection against malicious file uploads	Yes
Protection against Man-in-the-middle attacks	No
Protection against Link Injection Protection	No
Protection against Attribute autocomplete	No
Click hijacking protection	No
User input sanitization	Yes
Prepared statements	Yes
JWT session management	Yes

Protection against malicious file uploads

Regarding malicious file uploads, we have implemented many checks and methods to validate whether the file is not malicious. For the front-end, we only let the user upload a file one by one in order to not have to check all of the files simultaneously. The user is also only allowed to upload files with a .xlsx extension. For the back-end, many checks are done to validate whether the content of the xlsx file is not malicious. For example, the back-end checks once more with the help of the *FileMagic* library that it is an actual excel structured file. Then it checks whether it has the correct amount of rows and columns. Subsequently, before the data is inserted into the database, it is firstly sanitized to check for SQL injection.

User input sanitization

Most of the cross-site scripting (XSS) security vulnerabilities happened due to a lack of user input sanitization. The input sanitization plays an important role in preventing XSS attacks, and we've also implemented user input sanitization in our web application. As the user from the input, we only allow letters, numbers and "- ". Any other characters or symbols are not allowed; in this way, attackers cannot inject any script into our database to produce stored and reflected XSS attacks.

Prepared statements

A prepared statement is one of the best ways to prevent SQL injection. And that is what we used in our back end of the web application. All the queries that we send to the database are mostly prepared statements, so we only require some attributes from the user to fill in our prepared statement (all the user inputs need to pass the sanitization to be sent to the database) and return to the user what they've requested for. In this way, we've managed to prevent SQL injection. This is done as well within the querybuilder.

JWT session management

For session management, we've implemented JWT. The advantages of the JWT are that we do not need to store any of the tokens and easily distinguish between different users. As the user successfully logs in, a JWT token (which is valid for 30 minutes) is returned to the user. And this token is stored in the cookie of the user browser. Every request that the user sends to the back end will require the JWT token. Then the back end will check the validity of the token before it goes further.

Users in our system have different roles thus have different access levels, JWT token is also useful for us to distinguish between different kinds of user (with the field "audience" in the payload of the JWT token) so we can check if the user has the right to access a specific functionality.