Security Analysis

Passwords

All passwords are securely saved in the database. When the server receives the password as plain text I hashes it with salt. Both the hashed password and the salt are base64 encoded before being added to the database. The only risk with them is if there is person-in-the-middle attack. They could see the password and the username or email being sent as plain text and gain access this way.

XSS attacks

To protect the server from XSS attacks we sanitize all user input. We do this with an NPM package called DOMPurify. The package has a page to test what attacks can be negated but here is a short example:

```
DOMPurify.sanitize('<img src=x onerror=alert(1)//>');
//becomes <img src="x">

DOMPurify.sanitize('<svg><g/onload=alert(2)//<p>');
//becomes <svg><g></g></svg>

DOMPurify.sanitize('<p>abc<iframe//src=jAva&Tab;script:alert(3)
>def</p>');
//becomes <p>abc</p>

DOMPurify.sanitize('<math><mi//xlink:href="data:x,<script>alert
(4)</script>">');
//becomes <math><mi></mi></math>

DOMPurify.sanitize('<TABLE><tr><td>HELLO</tr></TABL>');
//becomes <table><tbody><tr><td>HELLO</td></tr></tbody></table>

DOMPurify.sanitize('<UL><li><A HREF=//google.com>click</UL>');
//becomes <ul><li><a href="//google.com">click</a></li></ul>
```

SQL injections

To protect the database from SQL injections we use the above mentioned DOMPurify package and Prepared Statements. The prepared statements turn all external parameters to plain SQL strings.