**Individual Project**

**Security Report**

**Software Engineering**

**beer.io**

**Fontys**

**University of Applied Sciences**

Prepared by:

Kristian Savov

Contents

[1. Analysis 3](#_Toc92707266)

# Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Likelihood | Impact | Risk | Actions possible | Planned |
| A1:  Broken Access Control | High | High | High | Role-based authorization  Not allow access to private resources without logging in | Yes |
| A2: Cryptographic Failures | Low | High | Medium | Encrypt passwords when storing in DB | Yes |
| A3:  Injection | Low | High | Medium | Use ORM  Validate user input on the server side | Yes |
| A4:  Insecure Design | Medium | Medium | Medium | Write unit and integration tests  Segregate tier layers | Yes |
| A5:  Security Misconfiguration | Medium | High | High | Remove or do not install unused features and frameworks | Yes |
| A6:  Vulnerable and Outdated Components | Medium | Low | Low | Remove unused dependencies, unnecessary features, components, files, documentation | No |
| A7:  Identification and Authentication Failures | Low | High | Medium | Do not ship or deploy with any default credentials, esp. for admin users | Yes |
| A8:  Software and Data Integrity Failures | Low | High | Medium | CI/CD pipeline has proper segregation | Yes |
| A9:  Security Logging and Monitoring Failures | Low | Medium | Low | Ensure that log data is encoded correctly to prevent injections or attacks on the logging | No |
| A10:  Server-Side Request Forgery | Low | Medium | Low | Log all accepted and blocked network flows on firewalls  Validate and sanitize all client-supplied input data | No |

# Reasoning

## 2.1 A1: Broken Access Control

Could occur when role-based authorization is not set up, or it is set up incorrectly. It is highly likely to happen because there are many things that must be configured, so it is possible that something is missed. Because that will allow user with the incorrect role to access private information, this has very high impact on the security of the application. Role-based authorization is implemented but is not incorporated in the application, meaning that right now all roles have access to the same content.

## A2: Cryptographic Failures

There should not be any sensitive information stored in plain text in the database. That is the reason why it is encrypted. The sensitive information that is used in the application is the user’s password and the JWT, generated when logging in the application. Due to them being encrypted and the JWT having an encryption secret, it is harder for anyone that gets hold of them to read them.

## A3: Injection

For preventing Injection into the application, I have used Hibernate ORM in order to interpret the SQL queries and not allow the user to have any access to writing SQL queries. The application is not completely secure, however, because the records that can be made in the case of an SQL injection have not been limited.

## A4: Insecure Design

In order to make sure the design of the application is secure enough, it is recommended to segregate the tier layers on the system depending on the exposure. This way it is sure that the user will not have access to any vulnerable data through the frontend. SonarCloud is also used to alarm about any security vulnerabilities.

## A5: Security Misconfiguration

There are not any unnecessary features that are enabled and the application does not ship with any default accounts. One thing, however, that is possible to occur is the configuration of Spring Security, because there are a big number of features that have to be set up. Therefore it is possible that some of them have been missed.

## A6: Vulnerable and Outdated Components

Due to the project being used as a learning source, there have been a lot of testing with dependencies, especially in the frontend, until the optimal ones are found. Therefore there is high chance that there are unused dependencies, which can be a vulnerability.

Because the initial idea of the project was different from the final one, there are some components in the backend, that pose a security threat, due to them not being in use.

## A7: Identification and Authentication Failures

In order to prevent identification and authentication failures the application is not shipped with any default credentials already included. The user’s input is validated from the frontend whether it meets simple criteria for security, e.g. password length. Users are not allowed to register with emails that already exist.

## A8: Software and Data Integrity Failures

CI/CD is applied in GitHub Actions. The project is built and tested, then it is being evaluated by SonarCloud for any security vulnerabilities. Only after passing the security checks then the Docker images are built and deployed to Docker Hub.

## A9: Security Logging and Monitoring Failures

No data is logged from the application. Later on it is possible to be made that a log is generated by the application with all the events that are happening, mainly user interactions.

## A10: Server-Side Request Forgery (SSRF)

Data that has been input from the user is not sanitized in any way. No network flows are logged. In the future “deny by default” firewall policies or network access control could be implemented to allow only the essential traffic.

# Conclusion

Overall the security of the application is on a good level, considering the goal of the project. The roles for authorization, as well as the correct configuration of Spring Security, would be a vital part of improving the security of the application.