# A close-up of a logo Description automatically generated

# Security Report: OWASP Top 10 Security Risks

### This report provides an analysis of how our application deals with the OWASP Top 10 security risks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Security Risk | Likelihood | Impact | Risk | Current Mitigation Actions |
| A01:  Broken Access Control | Very Likely | Severe | High | Implemented CORS, Authentication via Outlook, and AuthProvider (checks for token identity) |
| A02:  Cryptographic Failures | Likely | Severe | Moderate | Oulook API for the user account data, we don’t make use of HTTPS |
| A03:  Injection | Unlikely | Severe | Low | No-SQL database (MongoDB) |
| A04:  Insecure Design | Likely | Severe | Moderate | Unit Test and Integration tests implemented, stick to SOLID, Outlook Authentication |
| A05:  Security Misconfiguration | Unlikely | Moderate | Low | Express and React libraries, don’t send security headers, no use of deprecated features |
| A06:  Vulnerable and Outdated Components | Unlikely | Moderate | Low | No use of deprecated features and NPM as module manager |
| A07: Identification and Authentication Failures | Likely | Severe | Moderate | Outlook API for authentication (don’t use sessions, routes not protected), token is stored in local storage and currently we don’t have roles |
| A08:  Software and Data Integrity Failures | Likely | Severe | Moderate to High | NPM is consuming trusted repositories (Open-Sourced packages), no use of CI/CD pipeline |
| A09:  Security Logging and Monitoring Failures | Very Likely | Severe | High | Risk taken due to limited technical knowledge |
| A10:  Server-Side Request Forgery | Likely | Severe | Moderate | React framework takes care of HTTP redirect, no encrypted responses, use of CORS for localhost |

# Explanation

* Overall our application is secure because we made use of Outlook Authentication and API, NO-SQL Database, Express and React libraries, NPM and Open Source packages, CORS configuration, Unit and Integration tests implemented (End to End test following) and we are currently working on making the routes secure;   
  Under there’s an explanation of the risks that have been identified has more likely to occur in our project
* A01 – We considered risk high mainly because we didn’t make use of the HTTPS protocol and for this risk it makes it very likely to be exposed but we did out best and implemented features such as Outlook Authentication that lower a bit the likelihood
* A02 – In this risk on one hand we have implemented the features to make it secure but don’t have the knowledge to prevent it at the best, so we considered it to be moderate risk. We made use of the Outlook API for the user account data to be protected and GDPR compliant
* A04 – Despite having Unit and Integration tests our design can still have some insecurities that make this risk likely to happen
* A07 – We prevented this risk by implementing the Outlook API and using it for authentication and we also store the token on the local storage instead of being in the session, but we did not make use of roles or use sessions, so it also made the risk moderate to happen
* A08 – This risk we prevented it by making sure that NPM always uses Open-Sourced packages that are trusted but we did not made use of a CI/CD pipeline, and this makes the risk more exposed and more likely to happen
* A010 – In this risk we made it secure by using the React Framework that took care of the HTTP redirect and using CORS configuration for the localhost but we did not have encrypted responses so the risk will be likely to occur

# References

<https://owasp.org/Top10/>