**Design document**

***CareNest***

*People for People NGO*

A logo of hands and flowers

Description automatically generated

|  |
| --- |
| **Date : 20.12.2024** |
| **Version : 1.0** |
| **State : Beginning** |
| **Author : Bianca Cristea** |

Table of Contents

[1. Introduction 3](#_Toc185596196)

[2. OWASP top ten 4](#_Toc185596197)

[3. Project Analysis 5](#_Toc185596198)

[4. Reasoning 7](#_Toc185596199)

[5. Conclusion 9](#_Toc185596200)

# Introduction

In today’s world, online security is one of the most important aspect a user is looking for when using an application, needing to have their data protected and trust the party they engage with.

*CareNest*, a web platform connecting elderly and sick individuals with caretakers, handles sensitive information such as personal details, medical data, and service requests. This makes the platform a critical target for potential cyber threats.

The Open Web Application Security Project (OWASP) provides a globally recognized source of information for identifying and mitigating current security risks that people commonly confront. With a view to this guideline, this report evaluates the level of security in *CareNest* application, highlighting vulnerabilities, analysing current implementations and prospects of improvement.

# OWASP top ten

According to the movements and evolutions of technologies that our world faces, it is obvious that risks our applications are prone to, can also change in time. As for now, OWASP has channelled their efforts into keeping this ranking of online dangers up to date, and this is a list of their last study:

1. Broken-Access control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

# Project Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Likelihood | Impact | Risk | Actions possible | Planned |
| Broken-Access control | Low | High | Moderate | Continue monitoring for possible misconfigurations, ensuring all roles are permitted only designated actions | Verify permissions in WebSecurityConfig file, as well as roles permitted to access endpoints |
| Cryptographic Failures | Low | High | Moderate | HTTPS can be configured by obtaining an SSL/TLS certificate and updating the server to enforce HTTPS for all traffic. | Only if time permits |
| Injection | Low | High | Moderate | Regularly review input validation rules and ensure queries use parameters correctly. | Double check that all validations are correct |
| Insecure Design | Low | High | Moderate | Define secure coding standards and regularly adapt SonarQube to include coverage for OWASP security risks | No |
| Security misconfiguration | Low | Medium | Moderate | Move sensitive variables to secure secrets management | No, risk accepted |
| Vulnerable and Outdated Components | High | High | Critical | Regularly update dependencies, scan for vulnerabilities in libraries and frameworks. | Yes |
| Identification and Authentication Failures | Medium | High | High | Monitor authentication and password reset for abuse and enforce 2 factor authentication. | No, risk accepted |
| Software and Data Integrity Failures | Low | High | Moderate | Ensure all third-party dependencies are verified, use integrity checks for file uploads | Partially |
| Security Logging and Monitoring Failures | Medium | High | High | Log all security-relevant events, monitor logs for suspicious activities, and ensure log integrity. | No, risk accepted |
| Server-side Request Forgery | Low | High | Moderate | Validate and whitelist URLs for outgoing requests, restrict access to internal resources. | No, actions have already been taken |

# Reasoning

Broken Access Control

Every endpoint is restricted in the backend to users that are permitted to perform specific actions, and the frontend dynamically changes according to the role of the logged in user. Checks are also made, so that sensitive data that belongs to a user cannot be displayed for another user to see.

Cryptographic Failures

Currently, no sensitive data is stored in the database as plain text, and no data is decrypted during validation checks. However, communication between browser and API remains unencrypted, making it susceptible to interceptions from malicious users in a production environment. Enabling HTTPS would ensure a safe communication with the intended server but given that the application is still in a developmental phase, this is not yet a priority. If time permits, actions will be taken in this sense.

Injections

All user inputs are validated before being processed, but human error is possible, and certain aspects might have been overlooked. This is why I plan on running tests, trying to break the application and ensure that no malicious data is sent to the system I developed.

Insecure Design

The application is already designed in a layered structure (controller, service and persistence layers), and includes tests that cover at least 80% of the business layer, as well as tests on controller layer (using authentication). A pipeline that has SonarQube as a step is configured for this project so that the application cannot be deployed toa production environment before it meets the set standards of the quality gate. For the level of this application, I believe that enough measures have been taken to prevent this risk from happening.

Security misconfiguration

My current configurations are not necessarily vulnerable, but they can be improved. Even though actions are restricted, additional measures could be taken to align with best practices in the industry.

Vulnerable and outdated components

This risk is ranked as critical because my application relies on a considerable number of dependencies that need to be kept updated, which implies regular maintenance: any delay in that sense can make the application a target for attacks. To lower this risk, a dependency vulnerability scanner can be implemented in the CI pipeline for both frontend and backend (the backend already uses SonarQube as a tool that detects vulnerabilities).

Identification and Authentication Failures

Although I have implemented a mechanism to reset the password using a generated reset code with an expiry time, this is still vulnerable, even if it is a from of mitigation. An attacker might gain access to a user’s email address, and therefore lead to account takeover. At the same time, log in attempts could be monitored so that unusual activity is detected and blocked.

Software and Data Integrity Failures

The current state of the application includes security measures like JWT tokens and SonarQube checks in the pipeline, but it could benefit from improvements like third party dependency checks and integrity checks on uploaded files (a validation that I plan on implementing more strictly, as now it only verifies that only supported file forms are uploaded).

# Conclusion

The *CareNest* application has committed to providing its users with a reliable platform that securely serves their needs by implementing measures such as role-based access control, JWT token authentication, password reset mechanisms and SonarQube integration in the CI pipeline. These efforts demonstrate that the application was designed with a view to the quality of a user’ s experience, trying to avoid vulnerable gateways.

For CareNest to remain a trusted platform during development and after deploying production as well, further actions need to be taken in order to strengthen security and limit the risks. Mitigation, prevention and continuous monitoring will ensure that cyber threats remain an abstract notion for this project.