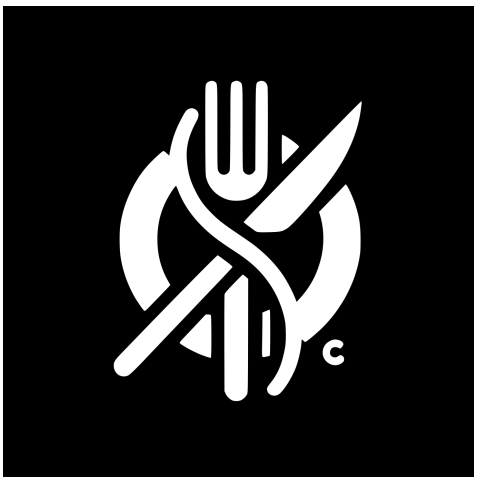
Security Document

Dine Master Pro



Canim

Date	15/01/2025
Version	1.0
State	Complete
Author	Samuil Kozarov - product developer

Table of Contents

Introduction

This report evaluates the security of my project against the **OWASP Top 10 (2021)** vulnerabilities. The analysis aims to identify potential risks, assess their probability and impact, and outline strategies for mitigation.

Risk is calculated as:
Risk = Probability × Impact

Higher-risk problems are prioritized for immediate action. The purpose of this document is to increase awareness of security vulnerabilities and provide actionable insights to improve application security.

OWASP Top 10 Risk Analysis Table

OWASP Category	Description of Risk	Likelihood (High/ Medium/ Low)	Impact (Critical/ Moderate/ Low)	Risk (Low/ Moderate/ High)	Actions Required	Status (Fixed/ Planned/ In progress/ Not Fixed)	Comments/Reasoning
A01: Broken Access Control	Unauthorized access to resources.	Medium (in general) Low (for the application)	Critical	Low	Implement strict role-based access control (RBAC).	Fixed	Added access control for all endpoints. ● As a customer, I can see and manage only my information. ● As a delivery person I can see the information of all employees but manage only my personal information ● As an admin I can see information of all users, but manage only mine
A02: Cryptographic Failures	Insecure handling of sensitive data (password).	Medium (in general) Low (for the application)	Critical	Low	Use strong encryption for passwords and sensitive data.	Fixed	Currently encrypting passwords with BCrypt.

A03: Injection	SQL injection via user input.	Medium (in general) Low (for the application)	Critical	Low	Use parameterized queries and input validation.	Fixed	All queries are parameterized, which means that the value is escaped before included in the sql statement (:value).
A04: Insecure Design	Weak design choices leading to vulnerabilities.	Low (in general) Low (for the application)	Moderate	Low	Perform threat modeling and follow secure design principles.	Fixed	Combines all security measures (Authentication, Authorization, RBA, input validations etc.).
A05: Security Misconfiguration	Misconfigured servers or frameworks.	Medium (in general) Low (for the application)	Moderate	Moderate	Code reviews by other teammates (in a team setting) or in an individual setting (by the developer). Regular configuration reviews and trying to use the default configurations and settings as little as possible.	Fixed	The modern frameworks provide such support and reviews (sonarqube warned me for a security hotspot about an API key that was not hidden). Remove unnecessary services and default configs (ex., many admin panels can be accessed with username: admin and password admin/password (if connected to the same network) which might be dangerous if a malicious user accesses my network). Good error handling mechanism is also part of this.

A06: Vulnerable and Outdated Components	Outdated libraries or dependencies.	High (in general) Medium (for the application)	Critical	Moderate	Regular dependency updates and vulnerability scans.	Fixed	Sonarqube does this and scans the whole code for vulnerabilities and other smells.
A07: Identification and Authentication Failures	Weak authentication mechanisms.	High (in general) Low (for the application)	Critical	Moderate	Enforce strong password policies and refresh tokens with short expiration time.	Fixed	Added JWT-based authentication with access and refresh token both of which have a very short expiration time. Refresh Token Automatic Reuse Detection mechanism is also implemented.
A08: Software and Data Integrity Failures	Lack of integrity checks in updates or inputs.	Low (in general) Medium (for the application)	Critical	Low	Secured CI/CD pipeline and good dependency scanning tool.	Fixed	Secret api keys should be kept as environment variables. Strict input validations are also added both in front and back end (in front end with react hook form and in back end with DTO classes).
A09: Security Logging and Monitoring Failures	Insufficient logging of security events.	Medium (in general) Medium (for the application)	Moderate	Moderate	Implement centralized logging with monitoring tools.	Not fixed	Monitoring tools have not been implemented (Datadog or APM). Logging of the errors is only added to the console in the backend

A10: Server-Side Request Forgery (SSRF)	Attackers sending unauthorized requests.	High (in general) Low (for the application)	Moderate	Moderate	Validate and restrict external requests. (role based access)	Fixed	Role based access is a security measure against such attacks (both for public host and local host). White list of ip addresses that can have access to admin panels, or similar important pages.

Reasoning for Security Risks

A01: Broken Access Control

- **Explanation**: Broken access control could allow unauthorized users to access restricted resources.
- **Likelihood**: Medium, because strict role based access control is implemented as well as some other role-based constraints. (refresh token, access token, a customer can only access and change his/her own information)
- **Impact**: Critical, as it could lead to unauthorized actions.
- **Actions**: Enforced RBAC with clear user roles and strict endpoint access control.

A02: Cryptographic Failures

- **Explanation**: Cryptographic failures often result from improper encryption practices or insecure algorithms.
- **Likelihood**: Medium, as sensitive data like passwords is encrypted before stored and password checks are one way (when I check if a given password is correct, I do not decrypt the already existing one, but encrypt the given one and check if matching).

- **Impact**: Critical, as stolen credentials or unencrypted sensitive data could severely harm users and the system.
- **Actions**: Implemented password hashing using BCrypt.

A03: Injection

- **Explanation**: User inputs can be malicious and directly affect the database, resulting in compromised data integrity or system access.
- **Likelihood**: Medium, as all queries are parameterized, which means that the value is escaped before included in the sql statement (:value)
- **Impact**: Critical, as it could compromise sensitive data or crash the application.
- **Actions**: Replaced raw SQL queries with parameterized statements and implemented strict input validation.

A04: Insecure Design

- **Explanation**: Weak design patterns, such as inadequate threat modeling, can lead to exploitable vulnerabilities.
- **Likelihood**: Medium, as the project follows secure coding practices.
- **Impact**: Moderate, as flaws in design could be exploited over time.
- **Actions**: Future implementation of threat modeling and adherence to secure design principles.

A05: Security Misconfiguration

- **Explanation**: Misconfigured servers, frameworks, or applications can expose the system to attacks.
- **Likelihood**: Medium, as small misconfigurations can easily occur.
- **Impact**: Moderate, as attackers can exploit exposed endpoints or unnecessary services.
- **Actions**: Regular configuration reviews, removal of unnecessary services, and proper error handling were implemented.

A06: Vulnerable and Outdated Components

- **Explanation**: Outdated libraries or dependencies can introduce known vulnerabilities into the system.
- **Likelihood**: High, as dependency vulnerabilities are common.
- **Impact**: Critical, as attackers can exploit these vulnerabilities to compromise the system.
- **Actions**: Might implement automated dependency scans tools like OWASP Dependency-Check; Sonarqube is already implemented and checks for smells and vulnerabilities.

A07: Identification and Authentication Failures

- **Explanation**: Weak authentication mechanisms can allow attackers to impersonate users.
- **Likelihood**: High, as authentication is a frequent target for attackers.
- **Impact**: Critical, as it could allow unauthorized access to sensitive user data.
- **Actions**: Added JWT-based authentication with access and refresh token both of which have a very short expiration time. Refresh Token Automatic Reuse Detection mechanism is also implemented.

A08: Software and Data Integrity Failures

- **Explanation**: Lack of integrity checks for updates or user inputs can lead to unauthorized modifications.
- **Likelihood**: Low, as the system doesn't frequently handle untrusted updates.
- **Impact**: Critical, as compromised updates or inputs can affect the application's integrity.
- **Actions**: Secret api keys should be kept as environment variables, Dependency scanning should be added as a stage in the pipeline and if a stage fails, the pipeline should not move further to the next stagesSecret api keys should be kept as

environment variables, Dependency scanning should be added as a stage in the pipeline and if a stage fails, the pipeline should not move further to the next stages. Strict input validations are also added both in front and back end (in front end with react hook form and in back end with DTO classes).

A09: Security Logging and Monitoring Failures

- **Explanation**: Insufficient logging can delay the detection of attacks.
- **Likelihood**: Medium, as proper logging requires dedicated infrastructure and processes.
- **Impact**: Moderate, as delayed detection can increase the damage caused by a breach.
- **Actions**: Error handling is added in both the front end and back end and logging is added in the console of the backend.

A10: Server-Side Request Forgery (SSRF)

- **Explanation**: SSRF vulnerabilities allow attackers to send unauthorized requests to internal or external services.
- **Likelihood**: Low, as the system has limited exposure to external resources.
- **Impact**: Moderate, as it can be used to access internal networks or sensitive services.
- **Actions**: Role based access is a security measure against such attacks (both for public host and local host). White list of ip addresses that can have access to admin panels, or similar important pages.

Conclusion

In conclusion, I think that my project respects most of the OWASP Top 10 Security principles, which talks about a well secured application. All critical principles have been covered, those that are not covered, have lower risk. The application should stay stable during malicious users attacks.