

## Digital Vault: Secure Web Application - Individual Report

Name: Jack Beans

Student ID: [Enter Your Student ID]

Module Code: ST6005CEM

Module Name: Security

Assignment Title: Individual Report (Secure Web Application)

---

### 1. Cover Page

Refer to the university template or branding guidelines. Include your name, student ID, course name, module code, and submission date.

---

### 1. Acknowledgment

I would like to express my gratitude to my module leader Arya Pokharel and Softwarica College for providing the opportunity to explore the field of web security. This project has significantly deepened my understanding of secure application development.

---

### 1. Table of Contents

(Generated automatically by Word/Docs based on styles)

---

### 1. Table of Figures

#### 2. Figure 1: Application Architecture

#### 3. Figure 2: MFA QR Code Setup

#### 4. Figure 3: Log Activity Dashboard

#### 5. Figure 4: Brute-force Lockout Flow

#### 6. Figure 5: Watermarked Image Upload

#### 7. Figure 6: Access Denied on Unauthorized Download

---

### 1. Table of Abbreviations

Abbreviation Meaning

MFA Multi-Factor Authentication RBAC Role-Based Access Control JWT JSON Web Token CORS Cross-Origin Resource Sharing PoC Proof of Concept DB Database

---

## 1. Abstract

This project report presents the design and implementation of Digital Vault, a secure web application focused on selling digital photos with strong security mechanisms. The system includes MFA, brute-force prevention, secure file uploads with watermarking, session management, and role-based access control. This report outlines the technologies, architecture, and security layers used, alongside audit logs and penetration testing. A proof-of-concept video demonstrates the final product.

Key Visuals: The visual walkthrough includes the MFA QR setup, watermark preview, role-based dashboard access, and audit log viewer interface.

---

## 1. Introduction

In today's data-driven world, users demand secure platforms to store, share, and trade digital assets. This report introduces Digital Vault, a secure web app designed to let creators sell their digital products (e.g., photos, art) with high privacy and transaction security. The application integrates strong authentication, watermark protection, secure file storage, and in-depth audit logging to prevent unauthorized access and data leaks ([OWASP, n.d.](#); [Lovell, n.d.](#)).

---

## 1. Frameworks and Languages Used During Development

### React.js

React.js was used to build the user interface of Digital Vault. Its component-based architecture and virtual DOM support dynamic rendering and responsive state management, essential for a secure and interactive UI ([Tailwind Labs, n.d.](#)).

### Tailwind CSS

Tailwind CSS enabled utility-first styling, providing responsive design out-of-the-box while allowing full control over the layout. It was used to maintain a modern and accessible UI across all pages ([Tailwind Labs, n.d.](#)).

### Node.js

Node.js powered the backend and API layer, enabling asynchronous request handling, scalable architecture, and seamless integration of Express-based middleware for security ([Express.js, n.d.](#)).

### MongoDB

MongoDB, with Mongoose ODM, was used to manage user, order, and upload records. Its schema flexibility enabled rapid prototyping, and indexes ensured efficient role-based queries ([MongoDB, n.d.](#)).

### Nodemailer

Nodemailer facilitated email verification, password reset, and activity notifications. It was integrated with Mailtrap during development and Gmail in production ([Nodemailer, n.d.](#); [Mailtrap, n.d.](#)).

---

## 1. System Design

The system follows a modular MERN stack architecture with secure client-server communication. Files are uploaded by verified creators, processed with Sharp for watermarking, and stored securely ([Lovell, n.d.](#)). Only customers with valid tokens and role-based access can view or download them.

### 9.1 Security by Design Principles

- Least Privilege: Users have limited access based on roles ([OWASP, n.d.](#)).
- MFA: Enabled using Speakeasy and QR code scanning ([Speakeasy, n.d.](#); [NPM, n.d.](#)).
- Password Policy: Enforced via regex and Zxcvbn password feedback ([Dropbox, n.d.](#)).
- Input Sanitization: Middleware like helmet, xss-clean, and mongo-sanitize are used ([Express.js, n.d.](#)).
- Account Lockout: Triggers after 5 failed logins.
- Audit Logging: Captures user activity for forensic and debugging purposes.

### 9.2 Core Code Examples

```
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W\s]).{8,64}$/;
```

```
if (user.failedLoginAttempts >= 5) { user.lockUntil = Date.now() + 30 * 60 * 1000; }
```







```
await logActivity(user._id, 'FILE_DOWNLOAD_SUCCESS', 'info', { method: req.method });
```

---

## 1. Proof of Concept

Video: [Insert Link to YouTube or Google Drive] GitHub Repository: [https://github.com/admiral41/ST6005CEM\\_Security\\_CW2](https://github.com/admiral41/ST6005CEM_Security_CW2)

Key Demonstrations:

-  Registration, email verification
-  MFA QR code setup and validation
-  Account lockout after failed logins
-  Secure image upload with watermarking
-  Download protection via RBAC
-  Activity logs with IP + timestamps

Suggested Visuals for Report

- Figure 2: MFA QR Code Setup Screenshot of the modal window showing MFA QR code and secret input field.
- Figure 3: Log Activity Dashboard Screenshot of admin panel showing recent log entries with user ID, IP, timestamp, and action.
- Figure 4: Brute-force Lockout Flow Screenshot showing lockout message after 5 failed login attempts.
- Figure 5: Watermarked Image Upload Preview of uploaded image displaying watermark text and styling.
- Figure 6: Access Denied on Unauthorized Download Alert or message shown to non-authenticated or unauthorized users trying to access a protected route.

---

## 1. Recommendations

- Encrypt audit logs at rest.
- Add CORS domain whitelisting for frontend-backend separation ([Express.js, n.d.](#)).
- Apply real-time file access monitoring.
- Integrate rate-limiting on all sensitive routes.
- Improve token revocation via Redis blacklist.

---

## 1. Conclusion

Digital Vault combines frontend usability with backend security to deliver a full-featured secure platform for creators and customers. Adhering to OWASP practices, it successfully implements MFA, account lockout, secure uploads, role-based access, and auditing. It stands as a viable and scalable proof of a secure digital product marketplace.

---

## 1. References (APA 7th Style)

• [OWASP. \(n.d.\). OWASP Top Ten.](#) • [Dropbox. \(n.d.\). Zxcvbn password strength estimator.](#) • [MongoDB. \(n.d.\). MongoDB Documentation.](#) • [Express.js. \(n.d.\). Helmet.js.](#) • [Speakeasy. \(n.d.\). TOTP authentication for Node.js.](#) • [NPM. \(n.d.\). qrcode.](#) • [Bcrypt. \(n.d.\). bcrypt for Node.js.](#) • [Tailwind Labs. \(n.d.\). Tailwind CSS Documentation.](#) • [Nodemailer. \(n.d.\). Nodemailer.](#) • [JWT.io. \(n.d.\). JSON Web Tokens Introduction.](#) • [Lovell, L. \(n.d.\). Sharp image processing.](#) • [Mailtrap. \(n.d.\). Email testing tool for developers.](#) • [Express.js. \(n.d.\). CORS Middleware.](#)

---

## 1. Appendix

• GitHub Repository: [https://github.com/admiral41/ST6005CEM\\_Security\\_CW2](https://github.com/admiral41/ST6005CEM_Security_CW2) • Proof-of-Concept Video: [Insert Link] • Screenshots: o QR Code modal view o Failed login lockout screen o Audit log table with filter UI o Image with visible watermark • Sample .env config (excluding secrets)