Security Implementation Documentation

This document outlines the security mechanisms implemented in the application and the planned security testing required for comprehensive validation.

Authentication & Access Control

Feature	Description	Implementation Detail
JWT (JSON Web Tokens)	Used for stateless authentication. It allows the server to verify the user's identity without storing session data on the server side.	Stored in **Cookies** with appropriate flags.
RBAC (Role-Based Access Control)	Regulates access to resources based on user roles (e.g., Admin vs. Regular User).	Role checks are enforced on the backend before resource access is granted.
Protected Frontend Routes	UI logic to prevent unauthorized users from accessing specific pages (like a Dashboard or Admin panel) before logging in.	Implemented using route guards/private routes in the frontend framework.

Data Protection & Cryptography

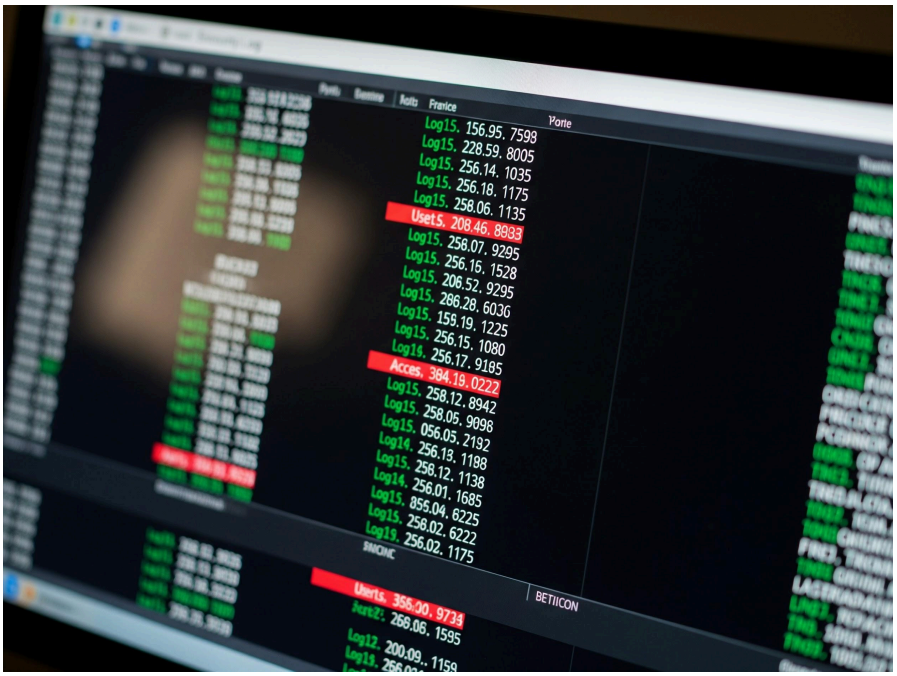
Feature	Description
Bcrypt	Used for **password hashing**. Ensures that even if the database is leaked, the actual passwords remain unreadable.
Encryption	Protecting sensitive data at rest to maintain confidentiality.
Environment Variables (.env)	Keeping sensitive keys (like Database URIs and Secret Keys) out of the source code to prevent them from being exposed on GitHub.
HTTPS Server	Ensures all data transmitted between the client and the server is encrypted (**Data in Transit**).

Input Validation & Attack Prevention

Feature	Description	Tool/Library Used
Validation & Sanitization	Ensures data matches the required schema, preventing malicious or malformed data.	**Joi (Backend)** and **Yup (Frontend)**
Helmet	Middleware that sets various **HTTP headers** to help protect the app from well-known web vulnerabilities (like XSS and Clickjacking).	Helmet middleware
CORS (Cross-Origin Resource Sharing)	Configured to restrict which domains are allowed to interact with the API.	Express/Server configuration
Express Rate Limiter	Protects the app from **Brute Force** or **DoS** attacks by limiting the number of requests a single IP can make within a specific timeframe.	Express middleware

Monitoring & Storage

Feature	Description	Purpose
Winston (Logfile)	A logging library used to record system events and errors.	Crucial for **Security Auditing** and tracking suspicious activity.
Cookies	Used to store tokens securely.	Implemented using `HttpOnly` and `Secure` flags to prevent theft via client-side scripts.



Security Scans to Perform

To complete the security documentation, the following mandatory security scans must be executed, and the results/screenshots included in the final report.

A. Dependency Scan (SCA - Software Composition Analysis)

This scan checks for vulnerabilities within the third-party libraries used in the project.

- **Quick Scan:** Run `npm audit` in the terminal.
- **Advanced Tool: Snyk**. This tool provides a detailed report of vulnerable libraries and recommended fixes. The report will be attached as File.

B. Static Application Security Testing (SAST)

SAST analyzes the application's source code for security flaws without actually executing the application.

- **Recommended Tool: SonarQube** or the **Snyk Security** extension in VS Code.
- **Focus Areas:** The scan should prioritize identifying:
  - Hardcoded secret keys.
  - Potential SQL injection risks.
  - Usage of weak cryptographic algorithms.

C. Dynamic Application Security Testing (DAST)

DAST tests the running application by simulating attacks from an external perspective.

- **Recommended Tool: OWASP ZAP** (highly recommended) or **Burp Suite**.
- **Action:** Point the tool at the running website URL and initiate an "Automated Scan."
- **Output:** The generated report will detail runtime vulnerabilities such as Cross-Site Scripting (XSS) or missing security headers.

Please schedule the time to run these tests by Date and include the results in a new folder labeled "Security Audit Artifacts" at File. We will review the findings in a follow-up meeting on Date. The calendar invitation for the review session is Calendar event.

External Certificate Validation

The application utilizes industry-standard TLS protocols to secure data transmission. External validation confirms the following security standards are met:

Security Element	Specification	Importance
Certificate Issuer	DigiCert Global G2 TLS RSA SHA256 2020 CAI	Establishes the trust chain for the website's identity.
Protocol	TLS 1.3	Ensures the use of the latest, most secure version of the Transport Layer Security protocol.
Encryption/Authentication	AES 128 GCM	Provides high-speed, authenticated encryption for data in transit.
Key Exchange	X25519	A secure, modern elliptic curve Diffie-Hellman function used to securely establish a session key.

STRIDE Threat Modeling Report

Application Name: Secure Web Application

Objective: Identify and mitigate security threats using the STRIDE methodology.

The table below outlines potential threats, their examples within the application, and the corresponding mitigation strategies.

Category	Description	Example in our App	Mitigation Strategy
Spoofing	An attack targeting the impersonation of a person or a device.	An attacker logs into the system using a stolen password or fake identity.	Implement secure Password Hashing with bcrypt.
Tampering	Changing or modifying data without authorization.	An attacker intercepts an HTTP request and modifies the role from "User" to "Admin".	Use HTTPS (TLS), Input Validation, and Signed JWT Tokens to ensure data integrity.
Repudiation	Performing an action and denying it due to a lack of system records.	A user changes sensitive data and denies it because there are no logs proving their identity.	Enable Detailed Logging.
Information Disclosure	Disclosure of sensitive information to unauthorized individuals.	A vulnerability in the API exposes sensitive user fields like email or ID to unauthorized viewers.	Implement Encryption in transit, and use generic error messages.
Denial of Service	Attacks aiming to stop or slow down the system service.	An attacker sends 100,000 requests per second to the website to crash the server.	Implement Rate Limiting.
Elevation of Privilege	An attacker with limited privileges gaining higher unauthorized access.	A regular user exploits a vulnerability like SQL Injection to become an Admin.	Apply the Principle of Least Privilege and strict Role-Based Access Control (RBAC).

DREAD Risk Assessment Report

Scoring Legend: Each category is rated 1 (Lowest) to 10 (Highest).

The DREAD model is used to calculate a quantitative risk score for identified threats based on five factors: Damage, Reproducibility, Exploitability, Affected Users, and Discoverability.

THREAT	(D)	(R)	(E)	(A)	(D)	TOTAL SCORE	AVERAGE (DREAD SCORE)	RISK LEVEL
SESSION HIJACKING	7	7	6	8	7	35	7.0	High
XSS	6	5	5	7	6	29	5.8	Medium
BRUTE FORCE	4	3	2	4	3	16	3.2	Low

Technology Stack & Dependencies

The project is built using a modern secure stack (Node.js, Express, React, and MongoDB). Below is a breakdown of the core libraries used in both the Frontend and Backend.

Frontend Dependencies (React)

These libraries handle the user interface, routing, and client-side data validation.

Library	Version	Purpose
React / React-dom	^19.2.0	Core framework for building the user interface.
React-router-dom	^7.11.0	Handles client-side routing and **Protected Routes**.
Axios	^1.13.2	Secure HTTP client for making API requests to the backend.
Formik	^2.4.9	Manages form state and submission logic.
Yup	^1.7.1	**Security:** Client-side schema validation and data sanitization.

8.2 Backend Dependencies (Node.js & Express)

These libraries are the core of the application's security and server-side logic.

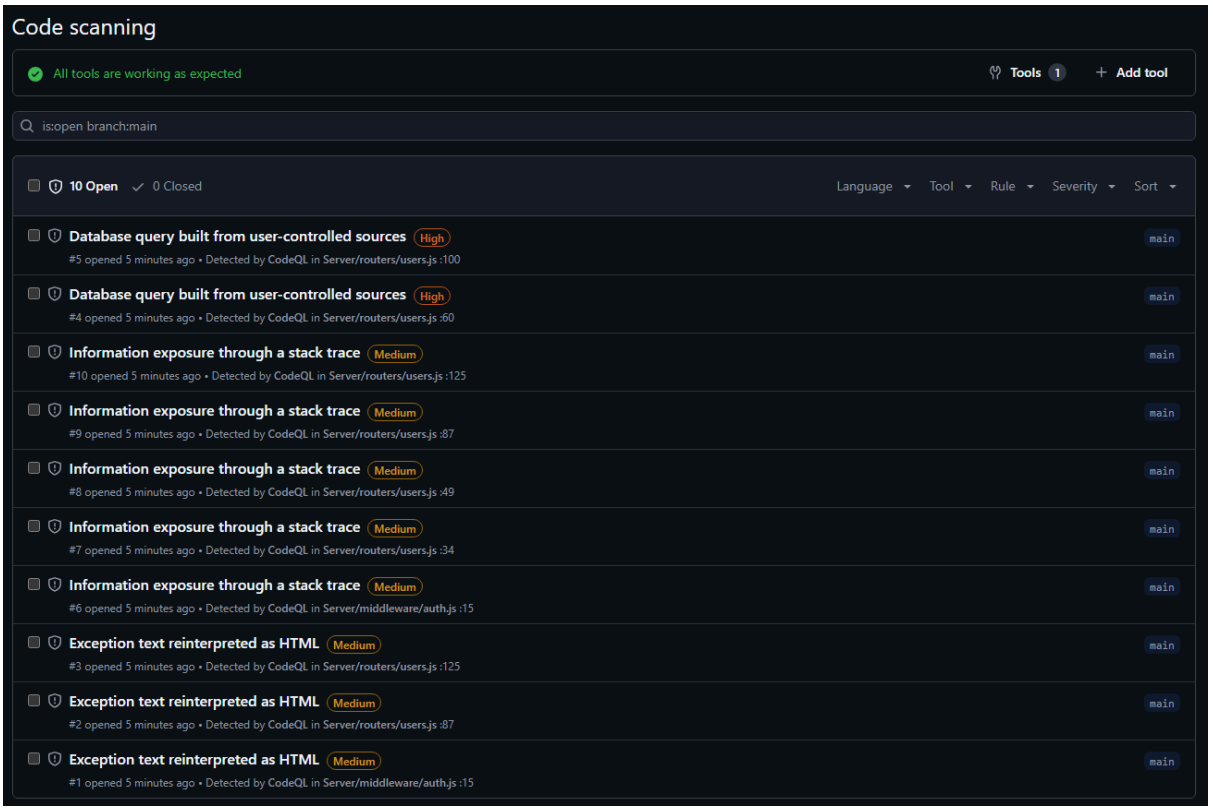
Library	Version	Security/Functional Purpose
Express	^5.2.1	Web framework used to build the REST API.
Mongoose	^9.0.2	Object Data Modeling (ODM) for secure MongoDB interactions.
Bcrypt	^6.0.0	**Security:** Strong password hashing (Work factor included).
Jsonwebtoken	^9.0.3	**Security:** Implements JWT for stateless authentication.
Helmet	^8.1.0	**Security:** Sets secure HTTP headers (XSS, Clickjacking protection).
Express-rate-limit	^8.2.1	**Security:** Protects against Brute Force and DoS attacks.
Joi	^18.0.2	**Security:** Strict backend schema validation (Prevents NoSQL Injection).
Winston	^3.19.0	**Security:** Professional logging for audit trails and error tracking.
Cookie-parser	^1.4.7	Handles secure, HttpOnly cookie management.
Cors	^2.8.5	**Security:** Manages Cross-Origin Resource Sharing policies.
Dotenv	^17.2.3	**Security:** Manages sensitive environment variables (API keys).
express-validator	^7.3.1	**Security:** Validates and sanitizes incoming server-side request data (body, params, query).

Security Scanning

5.1 Tool 1: GitHub CodeQL (SAST Scan)

Description: A Static Application Security Testing (SAST) scan was performed using GitHub CodeQL Advanced to analyze the source code for security vulnerabilities, including injection risks and data exposure.

Scan Evidence:



5.2 Tool 2: Snyk

Description: A software composition analysis (SCA) was conducted to identify vulnerabilities in third-party libraries and dependencies.
Scan Evidence:

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
638

```
30    if (users.length === 0) return res.status(404).send("No users found");
31
32    return res.status(200).send(users);
33  } catch (err) {
34    return res.status(500).send(err);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/routers/**users.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
638

```
46    return res.status(201).send(user);
47  } catch (err) {
48    console.log(err);
49    return res.status(500).send(err);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/routers/**users.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
638

```
83        .cookie("token", token, cookieOptions)
84        .send(response);
85    } catch (err) {
86      console.log(err);
87      return res.status(500).send(err);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/routers/**users.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
638

```
121         .cookie("token", token, cookieOptions)
122         .send(response);
123       } catch (err) {
124         console.log(err);
125         return res.status(500).send(err);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/routers/**users.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
638

```
134         .status(200)
135         .send({ message: "Logged out successfully" });
136     } catch (error) {
137       console.log(err);
138       return res.status(500).send(err);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/routers/**users.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE
588

```
11      const decoded = jwt.verify(token, process.env.JWT_SECRET);
12      req.user = decoded;
13      next();
14    } catch (error) {
15      return res.status(400).send(error);
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/middleware/**auth.js** 3 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Information Exposure - Server Error Message

SNYK CODE | CWE-200 + 1 MORE

SCORE 588

```
3   const { error } = schema.validate(req.body, { abortEarly: false });
4
5   if (error) {
6     const errorMessage = error.details.map(detail => detail.message).join(', ');
7     return res.status(400).send({
```

An error object flows to *send* and is leaked to the attacker. This may disclose important information about the application to an attacker.

Server/middleware/**joiValidation.js** 12 steps in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details

Cross-Site Request Forgery (CSRF)

SNYK CODE | CWE-352

SCORE 313

```
9  import { globalLimit } from "./middleware/limiter.js";
10
11 dotenv.config();
12
13 const app = express();
```

CSRF protection is disabled for your *Express app*. This allows the attackers to execute requests on a user's behalf.

Server/**index.js** 1 step in 1 file

Learn about this type of vulnerability and how to fix it

Ignore across repository | View details