|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Likelihood | Impact | Risk | Actions possible | Planned |
| A1: Broken Access Control | High | Severe | High | N/A, fixed | N/A |
| A2:Cryptographic Failures | Very unlikely | Severe | Low | No passwords or user data used | N/A |
| A3: Injection | Unlikely | Severe | Low | Prevented through ORM and input validation | N/A |
| A4: Insecure Design | Moderate | Moderate | Moderate | N/A, mitigated through Agile threat modeling | N/A |
| A5: Security Misconfiguration | Moderate | Severe | Moderate | Secured via proper configurations | N/A |
| A6: Vulnerable and Outdated Components | Low | High | Moderate | Managed via Maven/Gradle dependency updates | No, risk accepted |
| A7: Identification and Authentication Failures | Low | Severe | Moderate | Robust authentication mechanisms implemented | N/A |
| A8: Software and Data Integrity Failures | Low | Severe | Low | CI/CD pipeline includes automated checks | N/A |
| A9: Security Logging and Monitoring Failures | High | Severe | High | Logging and alert mechanisms under development | N/A |
| A10: Server-Side Request Forgery (SSRF) | High | Moderate | Moderate | Improve framework implementation | No, risk accepted |

Security Report

# FitQuest

**1. Broken Access Control**

FitQuest implements role-based access control (RBAC) using Spring Security to ensure strict permissions for data access and modification. The implementation includes:

* JWT tokens to validate user identity and roles.
* API-level access checks to enforce ownership of resources.

**A screen shot of a computer code

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

**2. Cryptographic Failures**

FitQuest employs encryption for sensitive data:

* Passwords are hashed using bcrypt before storage.
* Sensitive data like JWTs is signed with a strong secret key and uses secure algorithms like HMAC-SHA256.

A screen shot of a computer

Description automatically generated

**A computer screen with text

Description automatically generated**

**3. Injection**

To mitigate SQL Injection and other injection risks:

* FitQuest relies on Spring Data JPA, which automatically parameterizes queries.
* User inputs are validated and sanitized. (Code example provided earlier).

**A screen shot of a computer code

Description automatically generated**

**A computer screen with white text

Description automatically generated**

**4. Insecure Design**

Design principles prioritize security:

* Real-time updates and access controls are designed with scalability and safety in mind.
* Regular threat modeling during Agile sprints helps identify and mitigate risks.

**5. Security Misconfiguration**

FitQuest ensures secure defaults:

* Unnecessary endpoints are disabled.
* Spring Security is configured to reject insecure requests by default.

**A black background with white text

Description automatically generated**

**6. Vulnerable and Outdated Components**

Dependencies are managed with tools like Maven or Gradle:

* Avoiding unsupported versions and patching critical vulnerabilities promptly.

**7. Identification and Authentication Failures**

FitQuest enforces strong authentication:

* The use of JWT tokens ensures secure and scalable user session management.

**8. Software and Data Integrity Failures**

A robust CI/CD pipeline includes:

* Automated security testing to prevent malicious code deployment.
* Automated code quality checking to prevent uncovered code.