



Software Engineering Internship Web Application

Security Assessment Report

April 27th, 2023

Security Assessment – S.E. Internship Repository Application

Table of Contents

1. Summary	3
1. Assessment Scope	
Tools, Platform and Software Used For Testing:	3
2. Summary of Findings	3
3. Summary of Recommendations	5
2. Goals, Findings, and Recommendations	5
1. Assessment Goals	5
2. Detailed Findings	5
3. Recommendations	5
3. Methodology for the Security Control Assessment	5
4. Figures and Code	8
4.1.1 Process or Data flow of System (this one just describes the process for requesting), use-cases, security checklist, graphs, etc.	8
4.1.2 Other figure of code	
(see public repository here: https://github.com/DevByDJ/Database-Engineering-Project)	
10	
5. Works Cited	14

1. Summary

Executive Summary Here: Describe the overall goal, method, and major findings/recommendations here. (it's the TLDR)

1. Assessment Scope

Tools, Platform and Software Used For Testing:

- VS Code
- Javascript
- Supertest
- Jest (unit tests)
- Windows 11/

2. Summary of Findings

Of the findings discovered during our assessment, 4 were considered High risks, 5 Moderate risks, 5 Low, and 0 Informational risks. The SWOT used for planning the assessment are broken down as shown in Figure 2.

Severity	Probability ----->				
	Frequent	Probable	Likely	Possible	Rare
Emergency	Not Verifying a User Session	Verifying that a user is signing up and not a fraudulent software	Not limiting the amount of log-in attempts being made	Not securing the HTML, CSS and JavaScript	
Major	Hashing the password being used within code to access the database	Not securing the GitHub Repo	Not hashing passwords before saving them into a database	Backing up source code outside of a cloud storage.	
Moderate	Not requiring passwords to be reset every 3 months	Reducing the amount of unused dependencies in the project	Securing website access based off authority	Requiring only read access to users to prevent data leaking	
Minor	Securing workstations with passwords	Remote access to developing not requiring two-step authentication			
Negatable					
V					

Figure 1. Findings by Risk Level

Security Assessment – S.E. Internship Repository Application



Figure 2. SWOT

Strengths: Provides user authentication against an online database such as MongoDB, coupled with O-Auth to provide some levels of approved authorization into the app. Also uses GitHub as its own version control that alerts the dev of any potential danger relating to the dependencies in the project.

Security Assessment – S.E. Internship Repository Application

Opportunities: JSON Web Tokens. Could allow for a more robust authentication system that'll allow for more secure user sessions.

Threats: DDOS Attacks and Database attacks/breaches using SQL injectors.

Weaknesses: Currently no user session authentication. Unsecured HTML and CSS code in the front end.

3. Summary of Recommendations

2. Goals, Findings, and Recommendations

1. Assessment Goals

The purpose of this assessment was to do the following:

- Ensure the confidentiality, integrity, and availability of an internship repository.
- Identify potential vulnerabilities in the application's design, implementation, and deployment.
- Assess the effectiveness of user authentication and authorization mechanisms.
- Analyze the security of data storage and transmission.
- Provide recommendations for mitigating risks and improving the application's overall security standing.

2. Detailed Findings

3. User Authentication and Authorization:

Strengths: Implementation of secure user registration and sign-in processes, utilizing strong hashing algorithms for password storage (for example, bcrypt) and protection against brute force attacks.

Weaknesses: Lack of multi-factor authentication, and potential vulnerability to Cross-Site Request Forgery attacks.

4. Application Security:

Strengths: Use of secure API endpoints with proper validation and sanitization of user inputs and protection against Cross-Site Scripting attacks.

Weaknesses: Potential vulnerability to SQL Injection attacks,

5. Data Storage and Transmission:

Strengths: Encrypted connections using HTTPS, secure storage of sensitive user data.

Weaknesses: Insecure storage of sensitive data.

Security Assessment – S.E. Internship Repository Application

6. Deployment and Infrastructure:

Strengths: Use of secure hosting platforms with strong access controls and regular security updates

Weaknesses: Inadequate firewall configurations.

7. Retirement

Strengths: Project is already using version control software.

Weaknesses: Repository is made public.

8. Cross-Platform Use

Strengths: Uses Chrome based foundation, that comes with a supported library of internet defenses

Weaknesses: Exploits already found with the platform may affect the new web application.

1. Recommendations

As referred to in Figure 2. I recommend the usage of **JWT** to help with authentication and authorization. This can help validate user sessions to help prevent malicious attacks by outside users. I also recommend establishing defense measures to protect the web application against **SQL injections** to keep the database's data safe from data breaches, especially since it'll be handling user sensitive data.

7. Methodology for the Security Control Assessment

3.1.1 Risk Level Assessment (delete this text: you don't have to change 3.1.1)

Each Business Risk has been assigned a Risk Level value of High, Moderate, or Low. The rating is, in actuality, an assessment of the priority with which each Business Risk will be viewed. The definitions in Table 1 apply to risk level assessment values (based on probability and severity of risk). While Table 2 describes the estimation values used for a risk's "ease-of-fix".

Table 1 - Risk Values

Rating	Definition of Risk Rating
High Risk	Not Verifying User Session
Moderate Risk	Not Limiting the amount of log-in attempts
Low Risk	Not Securing the HTML, CSS and Javascript

Security Assessment – S.E. Internship Repository Application

Rating	Definition of Risk Rating
Informational	Requiring only read access to users to prevent data leaking
Observations	Backing up source code outside of a cloud storage

Table 2 - Ease of Fix Definitions

Rating	Definition of Risk Rating
Easy	Add expirations on JWT
Moderately Difficult	<ul style="list-style-type: none">• Securing API key using Github Secrets
Very Difficult	<ul style="list-style-type: none">• Create Unit Tests that will test the rigidity of the code I am implementing
No Known Fix	Preventing SQL Injection for the web application to fortify my databases defenses.

3.1.2 Tests and Analyses

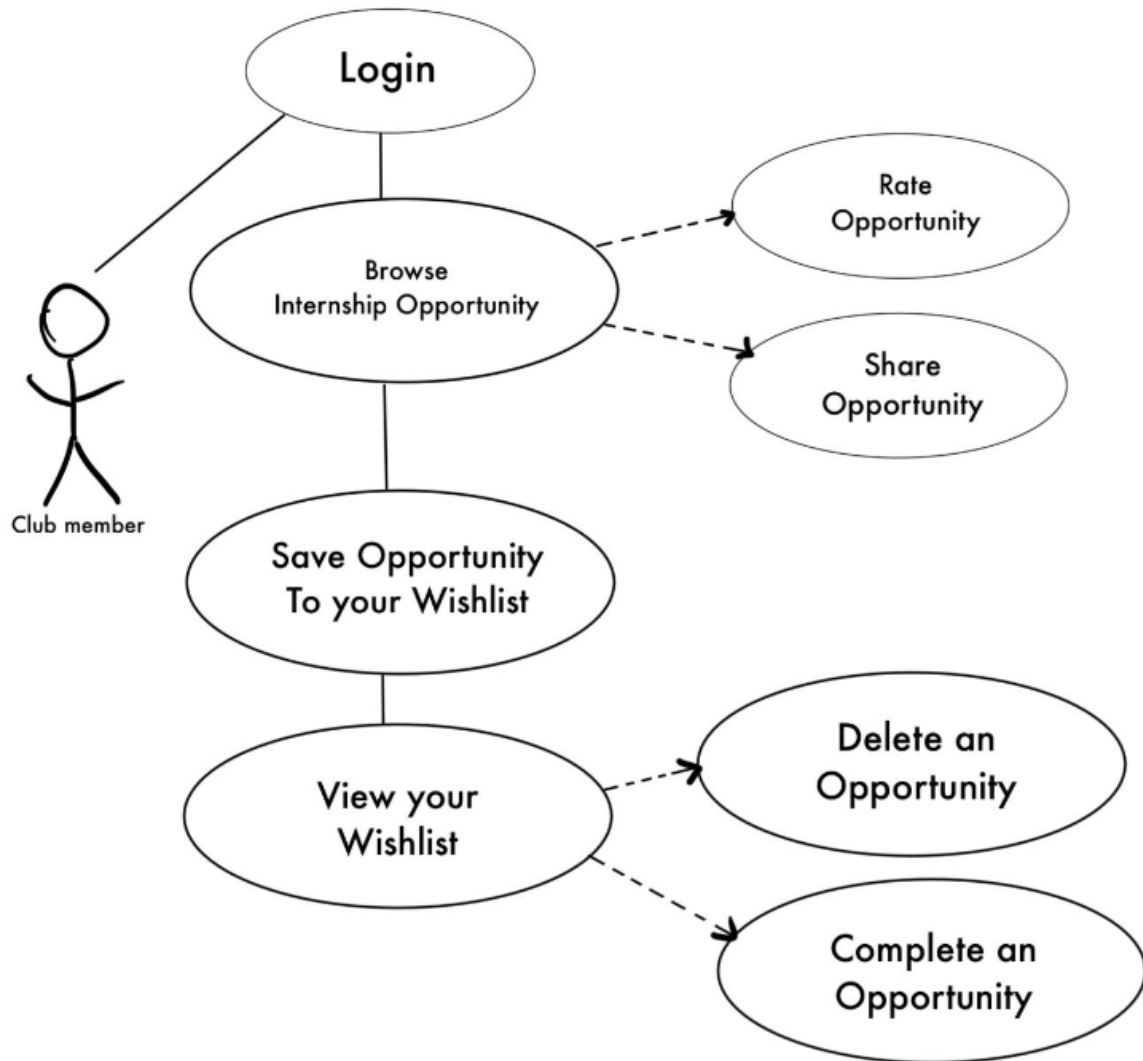
- Added authentication to a web application, I utilized JSON Web Tokens (JWT), Express.js, and Node.js. The login page collects the user's login credentials, which are then verified against a database. Upon successful verification, a JWT is generated and sent back to the client as a response. The JWT contains user data, which is used for subsequent requests to authorized routes. Middleware is used to verify the JWT and authorize access to protected routes.
- I created unit tests for the MongoDB database connection, login, register, and dashboard pages of a web application, as well as the new authentication that uses JWT, using a testing framework Jest and Supertest. Within the framework, I wrote test cases that simulate user interactions with the various pages and verify the expected outcomes. For the MongoDB connection, I wrote a test that the connection is successful and that the expected data can be retrieved from the database. For the authentication, I wrote a test that a JWT is generated and decoded correctly, and that access to protected pages is correctly restricted based on the presence of a valid token.

3.1.3 Tools

- Supertest
- Jest (unit tests)
- Postman

8. Figures and Code

8.1.1 Process or Data flow of System (this one just describes the process for requesting), use-cases, security checklist, graphs, etc.



- ☐ Club members will login (After registering with their school email)
- ☐ They will then be brought to their Dashboard to browse the catalog of available Internship Opportunities.
- ☐ (Option 1) They will be able to rate an Internship Opportunity of their choosing.
- ☐ (Option 2) They can share the Internship Opportunity with other members of the club.

Security Assessment – S.E. Internship Repository Application

- ☐ Next they can save the Opportunity to their own personal wishlist.
- ☐ Finally they can view their personalized wishlist
- ☐ (Option 1) They can maintain their list by deleting any unwanted Opportunities.
- ☐ (Option 2) They can complete the opportunity by applying to the Internship.

Security Assessment – S.E. Internship Repository Application

8.1.2 Other figure of code

(see public repository here:

<https://github.com/DevByDJ/Database-Engineering-Project>)

Security Assessment – S.E. Internship Repository Application

PULL REQUESTS:

The screenshot shows a GitHub pull request interface for the repository 'DevByDJ / Node-Server-Jest'. The pull request is titled 'FrontEnd UI, Backend API, Mongo Database and Unit Tests #4' and is in a 'Merged' state. It shows 13 commits merged into the 'main' branch from the 'test-branch' on January 3. The pull request is categorized as an 'enhancement' and has 10,119 views and 3,414 reactions.

The pull request description is divided into four sections:

- Front End UI**
 - Added login, register and a dashboard page.
 - Register and login page have symbol blocking password inputs
 - Dashboard is using imported PNG files (subject to change)
- Backend API's**
 - Added functional API endpoints using Express JS
 - Provided structured file system to reduce redundant scripting and organize workflow
 - Each page supported (dashboard, login, register) can support GET, POST methods
- Mongo Database**
 - Integrated MongoDB to handle the database of user information we'll be collecting
 - Required User Fields to be submitted before new documents can be added to a collection
 - Time stamped entries to keep track of recent activity
- Unit Testing**
 - Provided Tests for all the features listed above.

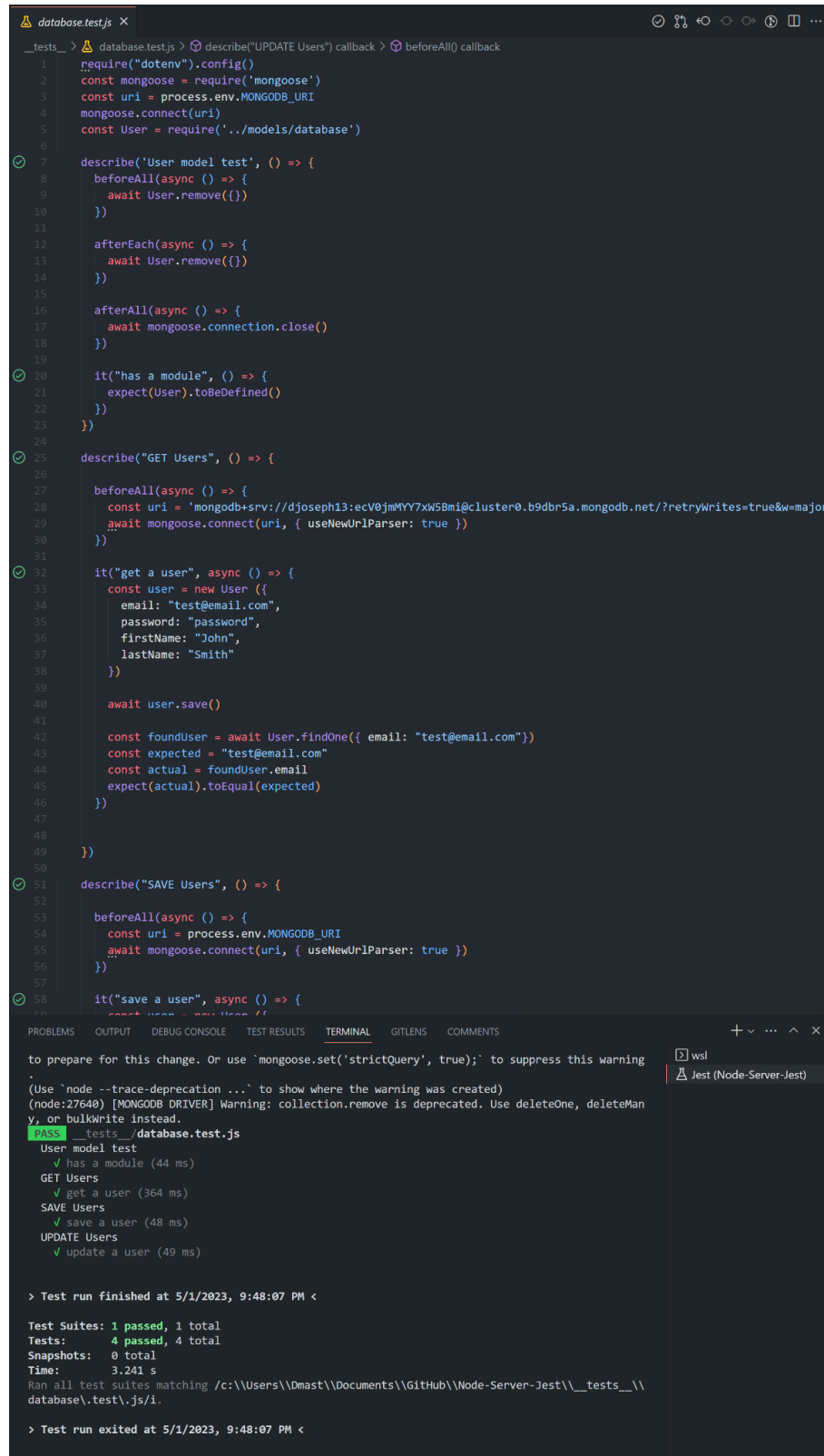
The commit history shows 13 commits by DevByDJ and others, including updates to app.js, nodemon, code refactoring, and integration of a new DB. The pull request was closed and then reopened by DevByDJ on January 3. It was merged into the main branch on January 3, with 0 of 2 checks passed.

The pull request also includes a 'RESULTS' section stating 'Code runs well!' and a 'Food for thought' section noting that the CI/CD pipeline isn't working optimally due to MongoDB server requirements.

The bottom of the screenshot shows a comment box with a 'Comment' button and a footer with GitHub's terms, privacy, and security information.

Security Assessment – S.E. Internship Repository Application

Unit Tests:



```
database.test.js
__tests__ > database.test.js > describe("UPDATE Users") callback > beforeAll() callback
1 require("dotenv").config()
2 const mongoose = require('mongoose')
3 const uri = process.env.MONGODB_URI
4 mongoose.connect(uri)
5 const User = require('../models/database')
6
7 describe('User model test', () => {
8   beforeAll(async () => {
9     await User.remove({})
10  })
11
12   afterEach(async () => {
13     await User.remove({})
14  })
15
16   afterAll(async () => {
17     await mongoose.connection.close()
18  })
19
20   it("has a module", () => {
21     expect(User).toBeDefined()
22   })
23 })
24
25 describe("GET Users", () => {
26
27   beforeAll(async () => {
28     const uri = 'mongodb+srv://djoeph13:ecV0jmYY7xW5Bmi@cluster0.b9dbr5a.mongodb.net/?retryWrites=true&w=majority'
29     await mongoose.connect(uri, { useNewUrlParser: true })
30   })
31
32   it("get a user", async () => {
33     const user = new User({
34       email: "test@email.com",
35       password: "password",
36       firstName: "John",
37       lastName: "Smith"
38     })
39
40     await user.save()
41
42     const foundUser = await User.findOne({ email: "test@email.com" })
43     const expected = "test@email.com"
44     const actual = foundUser.email
45     expect(actual).toEqual(expected)
46   })
47
48 })
49
50
51 describe("SAVE Users", () => {
52
53   beforeAll(async () => {
54     const uri = process.env.MONGODB_URI
55     await mongoose.connect(uri, { useNewUrlParser: true })
56   })
57
58   it("save a user", async () => {
59     const user = new User({
60       email: "test@email.com",
61       password: "password",
62       firstName: "John",
63       lastName: "Smith"
64     })
65     await user.save()
66   })
67 })
68
69 describe("UPDATE Users", () => {
70   beforeAll(async () => {
71     const uri = process.env.MONGODB_URI
72     await mongoose.connect(uri, { useNewUrlParser: true })
73   })
74
75   it("update a user", async () => {
76     const user = new User({
77       email: "test@email.com",
78       password: "password",
79       firstName: "John",
80       lastName: "Smith"
81     })
82     await user.save()
83
84     const foundUser = await User.findOne({ email: "test@email.com" })
85     const expected = "test@email.com"
86     const actual = foundUser.email
87     expect(actual).toEqual(expected)
88   })
89 })
90
91 })
```

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL GITLENS COMMENTS

to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:27640) [MONGODB DRIVER] Warning: collection.remove is deprecated. Use deleteOne, deleteMany, or bulkWrite instead.

PASS __tests__/database.test.js

User model test

- ✓ has a module (44 ms)

GET Users

- ✓ get a user (364 ms)

SAVE Users

- ✓ save a user (48 ms)

UPDATE Users

- ✓ update a user (49 ms)

> Test run finished at 5/1/2023, 9:48:07 PM <

Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 3.241 s
Ran all test suites matching /c:\Users\Omast\Documents\GitHub\Node-Server-Jest__tests__\database.test.js/i.

> Test run exited at 5/1/2023, 9:48:07 PM <

Security Assessment – S.E. Internship Repository Application

API TESTING:

The screenshot shows a REST client interface with the following details:

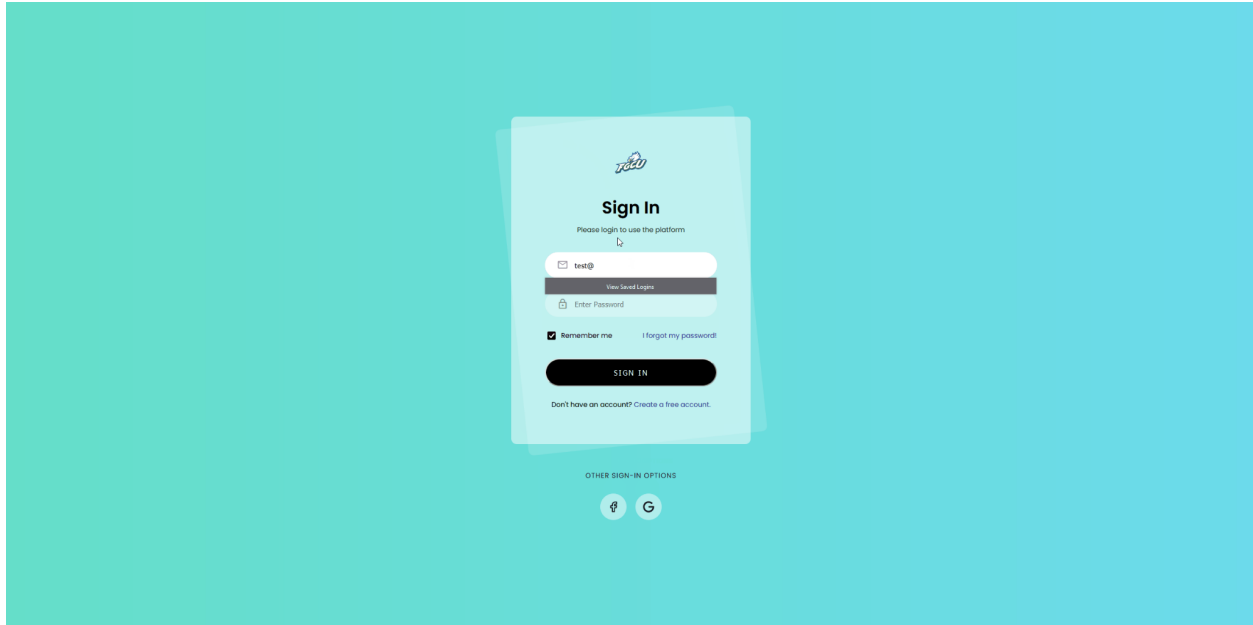
- URL:** localhost:3000/table
- Method:** GET
- Status:** 200 OK
- Time:** 68 ms
- Size:** 742 B

The response body is a JSON array with one object:

```
[{"student_id": 4, "username": "djoseph13", "grade_level": "Junior ", "major": "Software Engineering", "school": "FGCU", "email": "djosephNP@gmail.com", "job_id": 12345, "position_name": "Software Developer", "position_type": "Internship", "location": "Cupertino, CA", "start_date": "2023-05-15T04:00:00.000Z", "end_date": "2023-08-05T04:00:00.000Z", "semester": "Summer", "company_id": 12, "company_name": "Apple", "industry": "Technology", "company_location": "Cupertino, CA", "tag_id": 2, "tag_description": "Data Science", "survey_id": 1}]
```

UI TEST (Functional Testing):

Security Assessment – S.E. Internship Repository Application



9. Works Cited

Joseph, Danny. “Home.” *GitHub*, <https://github.com/DevByDJ/Database-Engineering-Project>.

Accessed 1 May 2023.