

**INTERNSHIP REPORT**

**DevelopersHub Corporation (Cyber Security Intern)**

**Sarang Zameer (DHC-545)**

# Acknowledgment

I would like to sincerely thank **DevelopersHub Corporation** for giving me the opportunity to complete my internship with them. The experience provided me with valuable exposure to real-world web security practices and enhanced my ability to analyze, identify, and mitigate vulnerabilities in modern applications.

I am especially grateful to my supervisor at DevelopersHub Corporation for their continuous guidance, constructive feedback, and encouragement throughout the internship. Their mentorship not only improved my technical skills but also helped me develop a professional approach to problem-solving and secure application development.

I would also like to thank my institution for supporting me in this journey, as well as my peers and colleagues for their constant motivation and helpful insights. The combined support from all these individuals made it possible to successfully complete this internship project and the accompanying report.

# Abstract

This internship at **DevelopersHub Corporation** focused on strengthening the security of a vulnerable User Management System developed in Node.js and Express. The project simulated a real-world scenario where insecure systems are subjected to penetration testing and security enhancements. The work was carried out over three weeks, covering assessment, implementation, and validation phases.

In **Week 1**, the system was assessed for common vulnerabilities identified in the OWASP Top 10, including Cross-Site Scripting (XSS), SQL Injection, weak password storage, missing authorization, and absent security headers. Each vulnerability was documented with its potential impact and proof-of-concept examples.

In **Week 2**, security measures were applied using widely adopted Node.js libraries. Input validation was enforced through Validator.js, passwords were securely stored with Bcrypt, and JWT authentication was implemented to protect sensitive routes. Helmet.js was used to configure secure headers, while Winston handled logging of security-related events.

In **Week 3**, the system underwent validation through penetration testing and logging review. The fixes proved effective as XSS payloads were escaped, SQL Injection attempts were blocked, unauthorized access required valid tokens, and secure headers were present in HTTP responses. Logs captured authentication events, and Nmap confirmed only expected ports were open.

This internship project not only enhanced my technical skills but also gave me practical experience in applying security best practices to real-world applications. The knowledge gained during this period at DevelopersHub Corporation will serve as a strong foundation for future work in cybersecurity and secure application development.

Contents

[Acknowledgment 2](#_Toc207983718)

[Abstract 2](#_Toc207983719)

[Introduction 4](#_Toc207983720)

[Background and Literature Review 4](#_Toc207983721)

[Methodology and Setup 4](#_Toc207983722)

[Week 1 – Security Assessment 5](#_Toc207983723)

[Week 2 – Implementing Security Measures 11](#_Toc207983724)

[Week 3 – Validation and Testing 15](#_Toc207983725)

[Discussion and Recommendations 15](#_Toc207983726)

[Conclusion 16](#_Toc207983727)

[References 16](#_Toc207983728)

# Introduction

With the rise of internet-based systems, cybersecurity has become an essential field. Modern applications handle sensitive information such as personal details, login credentials, and financial records. Any breach or attack can result in serious consequences for both users and organizations. Building secure applications is no longer optional; it is a fundamental requirement.

This project simulated a real-world scenario where a vulnerable web application was provided for testing. The task was to identify flaws, apply fixes, and verify that vulnerabilities were mitigated. The main learning objectives included:

1. Understanding common vulnerabilities in web applications.
2. Learning to use cybersecurity tools such as OWASP ZAP and Nmap.
3. Applying practical fixes using Node.js libraries.
4. Developing a structured approach to security reporting.

# Background and Literature Review

The OWASP Top 10 project highlights the most critical risks to web applications. In this internship, the focus was on several of these risks:

**Cross-Site Scripting (XSS):** This occurs when untrusted data is included in web pages without validation or escaping. For example, inserting **<script>alert('Hacked')</script>** into a comment box can trigger a malicious script.

**Injection Attacks:** SQL Injection is a common example, where attackers manipulate queries by injecting code. For instance, using **admin' OR '1'='1** can bypass authentication.

**Broken Authentication:** Using plaintext passwords or weak session handling makes accounts vulnerable to takeover.

**Security Misconfigurations:** Missing HTTP headers, open ports, or default credentials expose the system to unnecessary risks.

These vulnerabilities are common in beginner applications and highlight why secure coding practices are critical.

# Methodology and Setup

The project was carried out in three phases. The application tested was a simple User Management System built with Node.js and Express. The following tools were used:

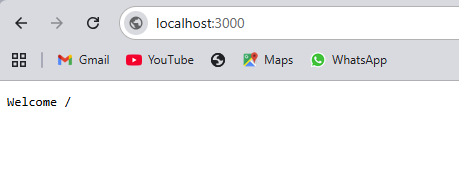
* OWASP ZAP for automated vulnerability scanning.
* Browser Developer Tools for manual input injection and inspection.
* Nmap for port scanning and network exploration.
* Postman for API testing.

The vulnerable application was set up locally using:

npm install

npm start

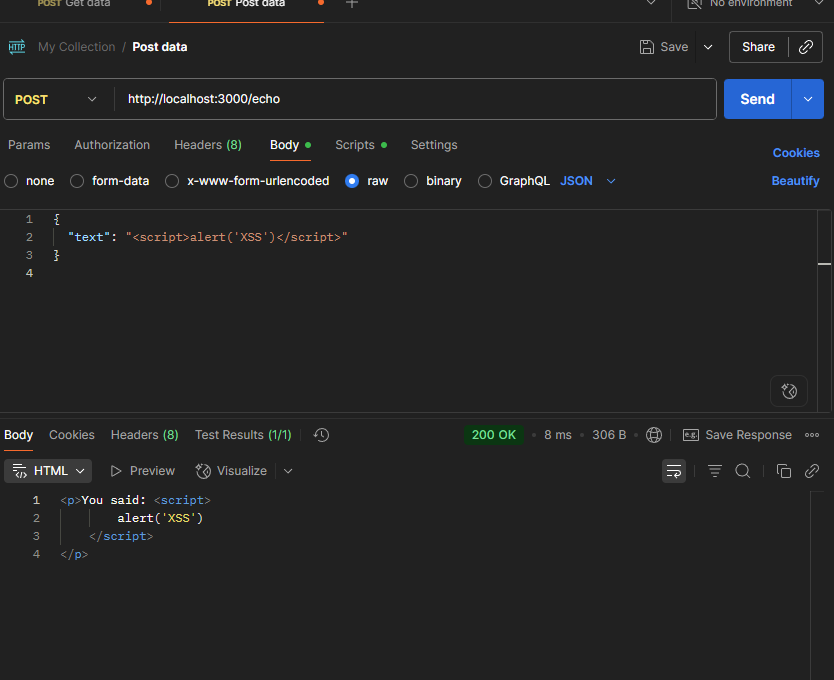
It was accessible at <http://localhost:3000>.

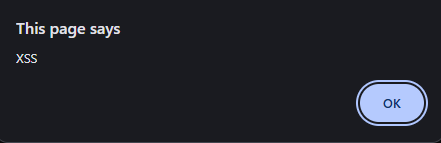


## Week 1 – Security Assessment

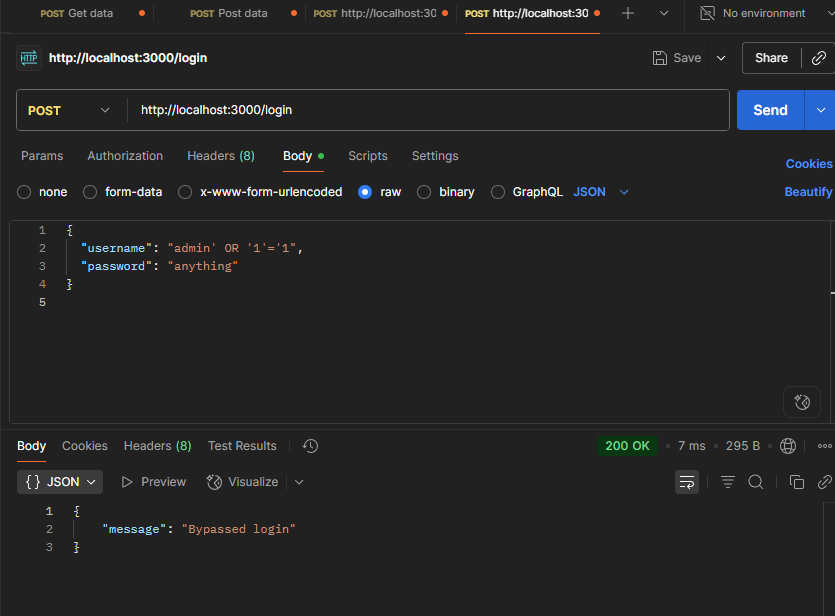
The first week focused on identifying vulnerabilities in the insecure application.

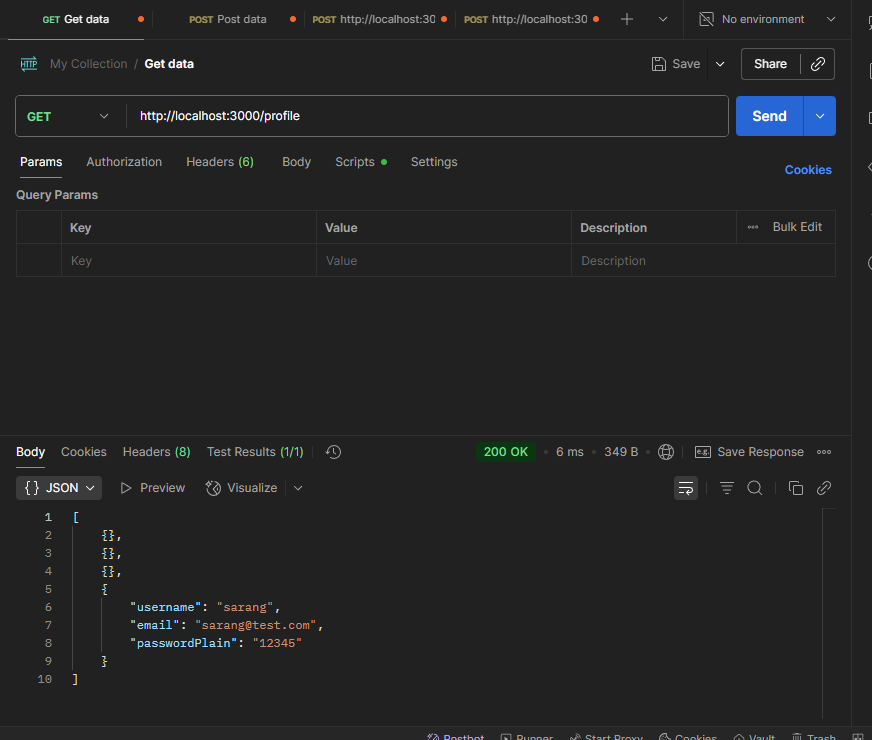
**Finding W1-1: Reflected XSS**  
Test: Sending **<script>alert('XSS')</script>** to the **/echo** endpoint resulted in a browser pop-up.  
Impact: Attackers could steal cookies or hijack sessions.  
Fix: Escape inputs using **validator.escape()**.





**Finding W1-2: Authentication Bypass**  
Test: Entering **admin' OR '1'='1** as the username bypassed authentication.  
Impact: Unauthorized access to sensitive accounts.  
Fix: Use parameterized queries and input validation.



**Finding W1-3: Missing Authorization**  
Test: Accessing **/profile** returned all user data without requiring authentication.  
Impact: Data leakage and privacy violations.  
Fix: Protect endpoints with authentication middleware.

**Finding W1-4: Missing Security Headers**  
OWASP ZAP flagged missing headers such as **X-Content-Type-Options** and **X-Frame-Options.**Impact: Increased risk of clickjacking and MIME sniffing.  
Fix: Apply Helmet.js to configure secure headers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Vulnerability | Location/Endpoint | Evidence | Impact/Risk | Suggested Fix |
| W1-1 | Reflected XSS | POST /echo | When submitting `<script>alert('XSS')</script>` in the request body, the application returned the script unescaped, resulting in a JavaScript alert pop-up. This confirmed that user-supplied input was directly reflected into the response without sanitization. | An attacker could exploit this to run malicious scripts in the victim’s browser, steal cookies, perform phishing, or deface the application. | Sanitize and escape all user inputs before reflecting them in the response. Libraries like validator.escape() should be used. |
| W1-2 | Authentication Bypass (SQLi-like) | POST /login | By entering `admin' OR '1'='1` as the username, the system responded with a successful login message despite invalid credentials. This simulated the behavior of SQL Injection where queries are concatenated without proper sanitization. | This vulnerability allows an attacker to bypass authentication and gain unauthorized access to protected accounts or administrative areas. | Always use parameterized queries and validate input. Never directly concatenate user input into SQL queries. |
| W1-3 | Missing Authorization Controls | GET /profile | Accessing the /profile endpoint without any authentication token returned the complete list of registered users, including sensitive details. This revealed that no authorization check was in place. | This leads to data leakage and exposure of sensitive information to unauthorized users. | Enforce proper authentication and authorization middleware on all protected endpoints. |
| W1-4 | Missing Security Headers | All Endpoints | OWASP ZAP flagged the absence of recommended HTTP headers such as X-Content-Type-Options, X-Frame-Options, and Strict-Transport-Security. | The lack of secure headers increases the attack surface, allowing risks such as clickjacking, MIME-type sniffing, and reduced browser protections. | Integrate Helmet.js to add standard secure HTTP headers. |

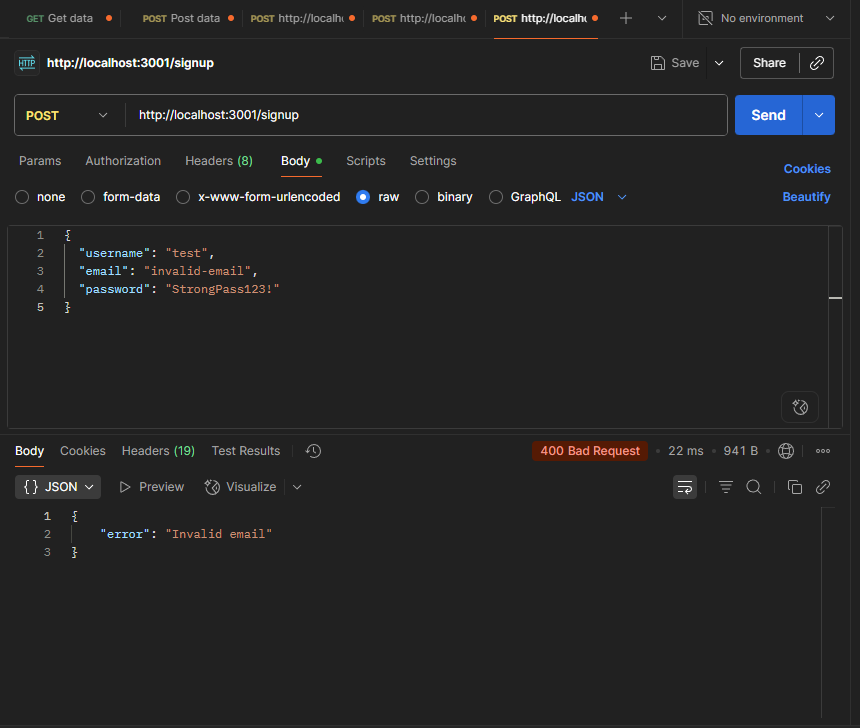
## Week 2 – Implementing Security Measures

The second week was dedicated to securing the application using well-known libraries.

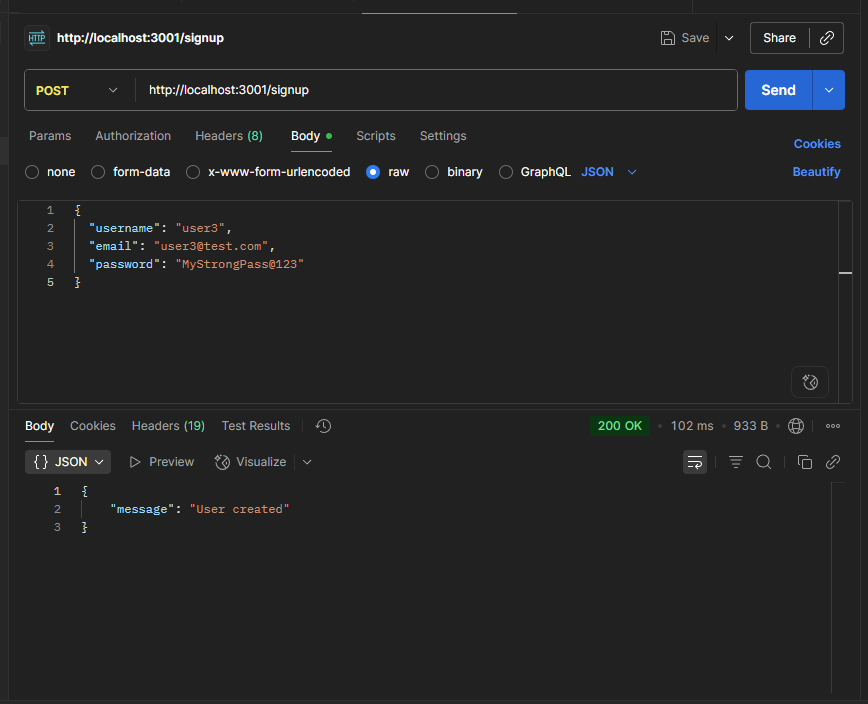
**Input Validation**  
Validator.js was used to ensure inputs, such as emails, were valid.

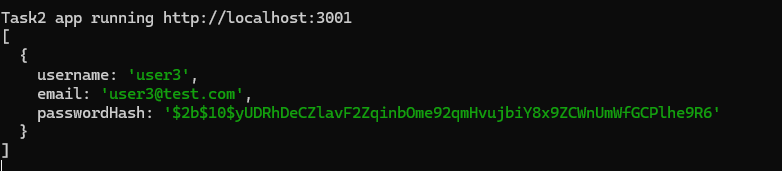
**if (!validator.isEmail(email)) {**

**return res.status(400).send('Invalid email');**

**}**

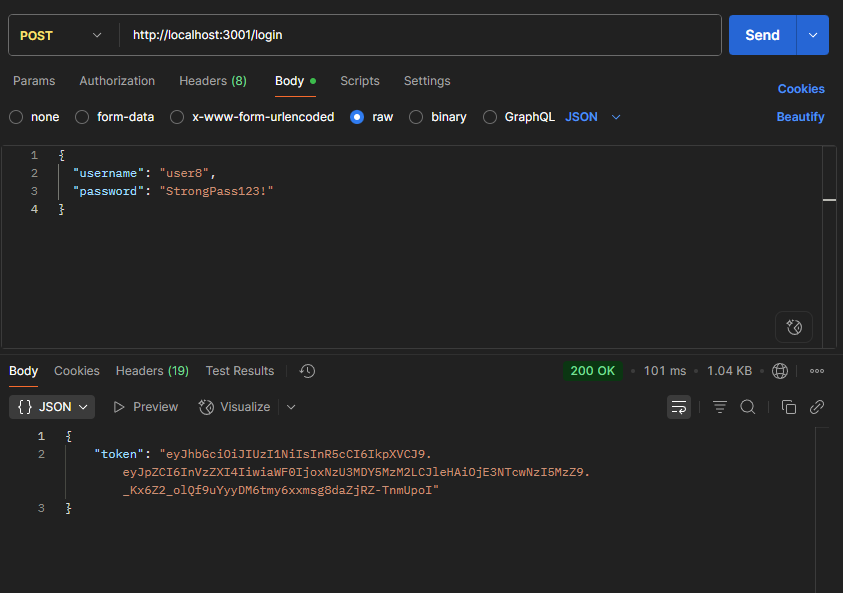
**Password Hashing**  
Passwords were hashed using bcrypt before storage.

**const hashedPassword = await bcrypt.hash(password, 10);**

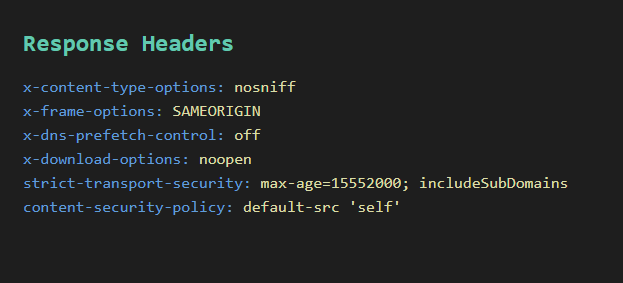
****

**JWT Authentication**  
jsonwebtoken was used to generate tokens required for protected routes.

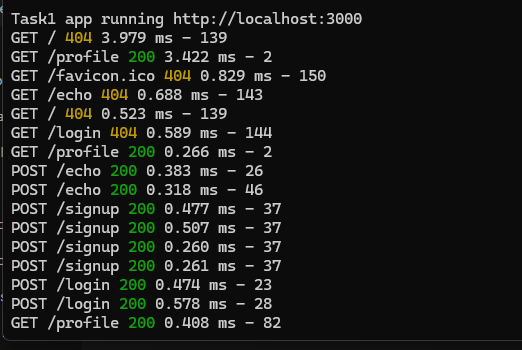
**const token = jwt.sign({ id: user.\_id }, 'secret', { expiresIn: '1h' });**

****

**Helmet.js Secure Headers**  
Helmet.js was used to add secure HTTP headers such as **X-Frame-Options** and **X-Content-Type-Options.**

****

**Logging with Winston**  
Winston was configured to log important security events. Logs were written both to the console and a file named **security.log.**

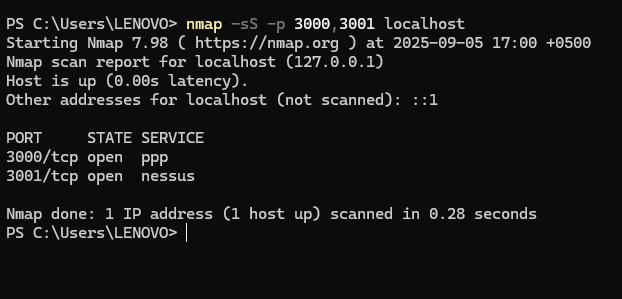


## 

## 

## Week 3 – Validation and Testing

The final week involved validating the fixes and ensuring the system resisted previous attacks.

* XSS attempts returned escaped HTML, no alert was triggered.
* SQL injection attempts were rejected with "Invalid credentials."
* The **/profile** route required a valid token and returned 401 Unauthorized without one.
* Security headers were visible in HTTP responses.
* Winston logs recorded login and signup attempts.
* Nmap confirmed only expected ports (3000, 3001) were open.

# Discussion and Recommendations

This project demonstrated how deliberate vulnerabilities can be exploited and then fixed through secure coding practices. While the fixes improved the application, additional measures should be considered:

1. Implement CSRF protection against cross-site request forgery.
2. Add rate limiting to prevent brute-force attacks.
3. Enforce account lockouts after repeated failed login attempts.
4. Use a production-grade database with parameterized queries.
5. Deploy the application over HTTPS with TLS certificates.

# Conclusion

The internship project successfully transformed a vulnerable web application into a more secure one. By going through the cycle of discovering, fixing, and validating vulnerabilities, I gained hands-on experience with practical security measures. These skills will be valuable in future projects where protecting user data is a priority.

# References

* OWASP Top 10: [https://owasp.org/www-project-top-ten/](https://owasp.org/www-project-top-ten/?utm_source=chatgpt.com)
* bcrypt.js Documentation
* validator.js Documentation
* Helmet.js Documentation
* Winston Logger Documentation
* OWASP ZAP Documentation
* Nmap Documentation