Documentation: Backend & Frontend Vulnerability Testing with CodeQL

1. Objective

The objective of this task was to introduce **sample vulnerabilities** into both the backend and frontend of the **AutoAudit project**, then verify whether GitHub's **CodeQL analysis** and **CI/CD workflows** were able to detect them. This process simulates real-world scenarios where insecure coding practices could be introduced into financial/compliance systems like AutoAudit, which deal with **data processing, audits, and user inputs**.

---

2. Backend Testing

2.1 Vulnerabilities Added

Two vulnerabilities were added under backend/tests/ to reflect common risks in backend systems like AutoAudit:

1. **SQL Injection**
    - **Description**: SQL Injection occurs when user input is concatenated directly into SQL queries without sanitization, allowing attackers to manipulate queries and access unauthorized data.
    - **Why Chosen**: SQL Injection is one of the most common backend API vulnerabilities and directly applies to AutoAudit's backend where queries are processed.
    - **Dataset Reference**: Adapted from GitHub dataset Broken-Vulnerable-Code-Snippets – SQL Injection.
    - **Implementation**: Created a test file sql-injection.js under backend/tests/ with intentionally unsafe string concatenation queries.

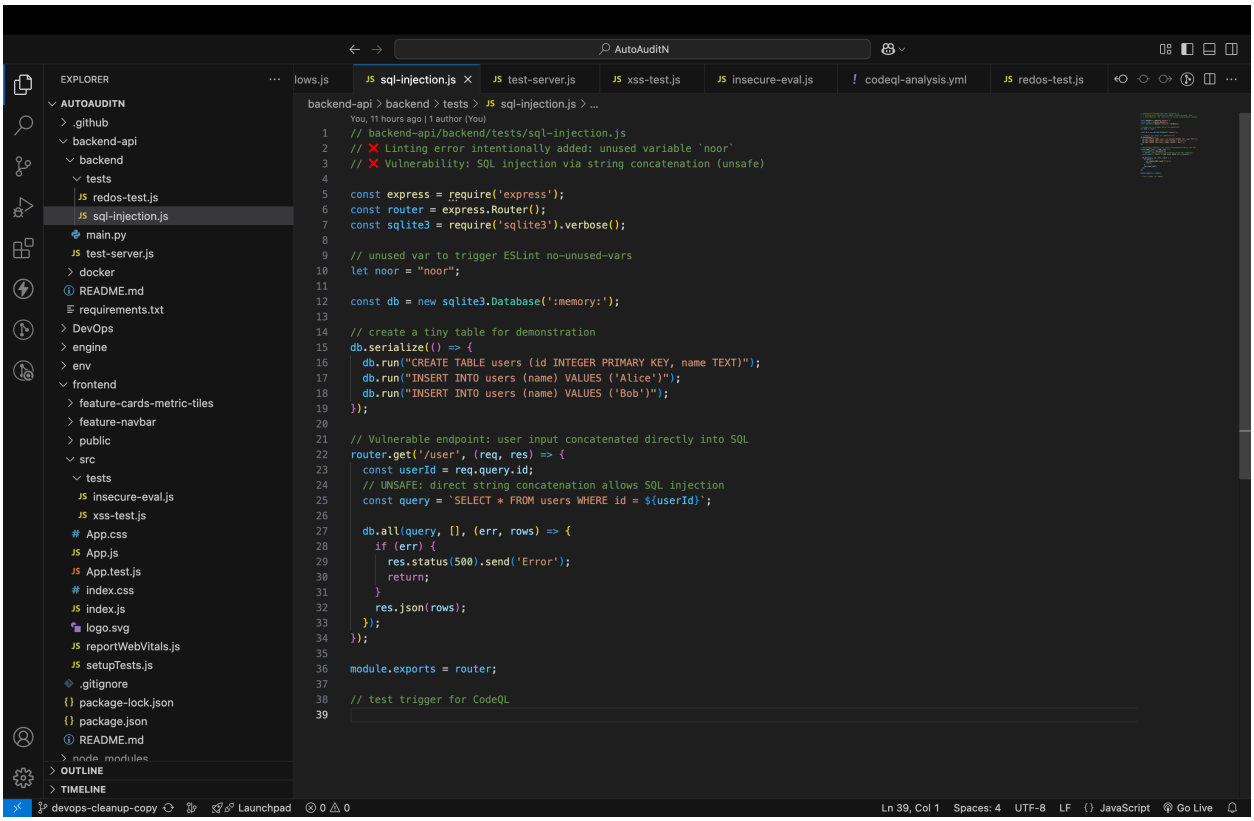

Figure 1: SQL-Injection Code

Commit Link -
https://github.com/nvirdi21/AutoAuditN/commit/900fc16ae1cf92509e21ecab9207c715926608d6

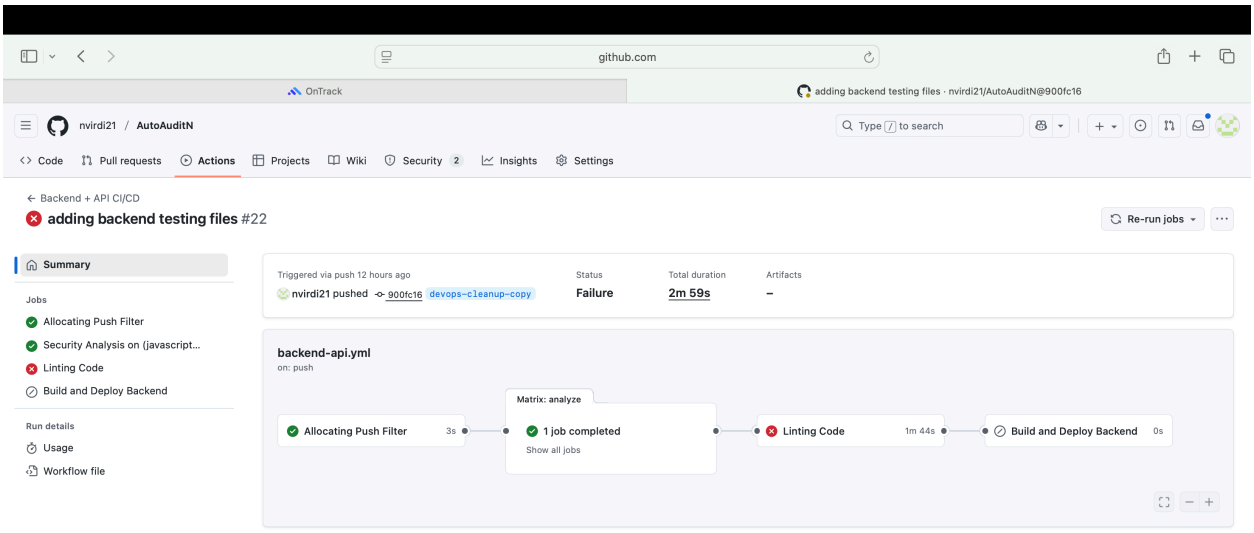

Figure 2: Backend CI — Linting failed (intentional lint error added for testing)

Link [Backend CI/CD] - https://github.com/nvirdi21/AutoAuditN/actions/runs/17849115964

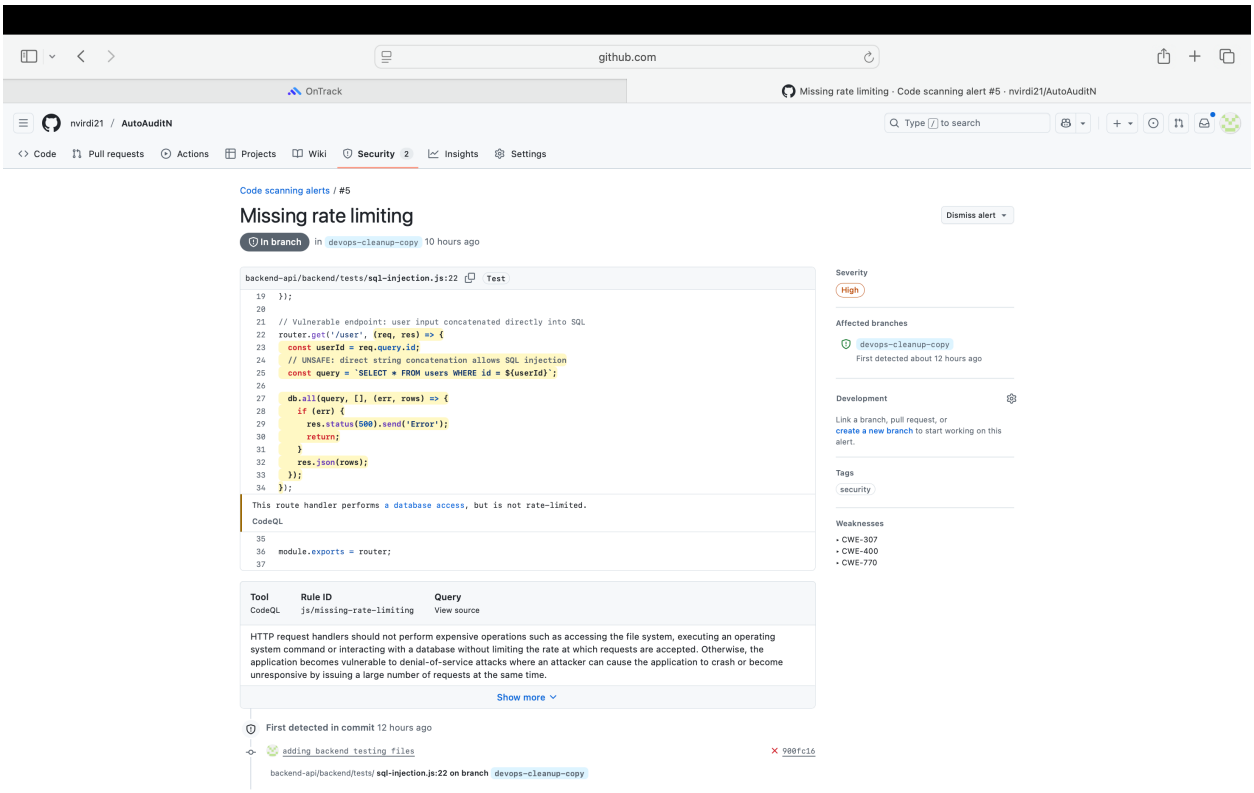

Figure 3: CodeQL security alert confirming SQL Injection Vulnerability

Link to the Security tab where CodeQl is flagging the security alerts - https://github.com/nvirdi21/AutoAuditN/security/code-scanning/5

2. **ReDoS (Regular Expression Denial of Service)**
    - **Description**: ReDoS exploits poorly written regex expressions that cause excessive backtracking, leading to denial of service by consuming high CPU.
    - **Why Chosen**: Regex is heavily used in backend validation and logging in AutoAudit, so this vulnerability is directly relevant.
    - **Dataset Reference**: Adapted from GitHub dataset [Broken-Vulnerable-Code-Snippets – ReDoS].
    - **Implementation**: Created a test file redos-test.js under backend/tests/ with a vulnerable regex /^(a+)+$/ that hangs on long malicious input.

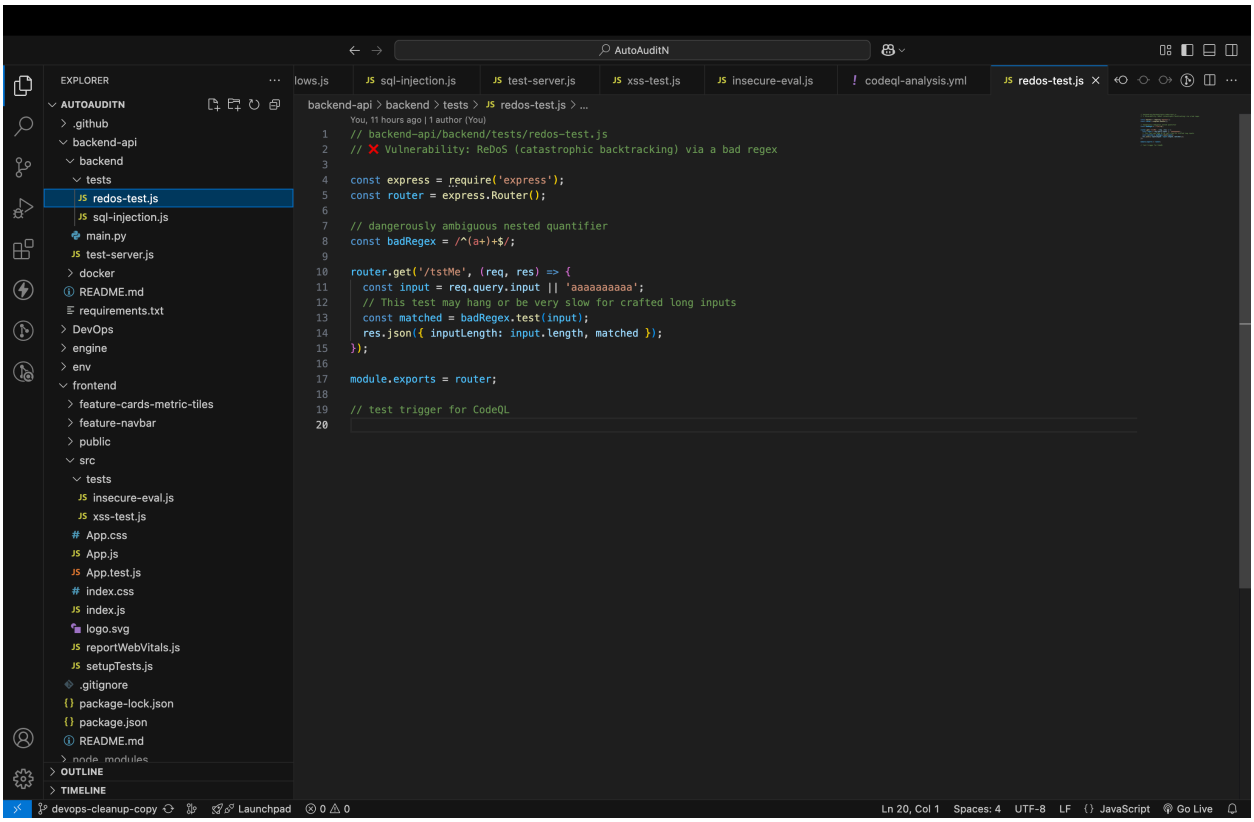

Figure 4: ReDos Code

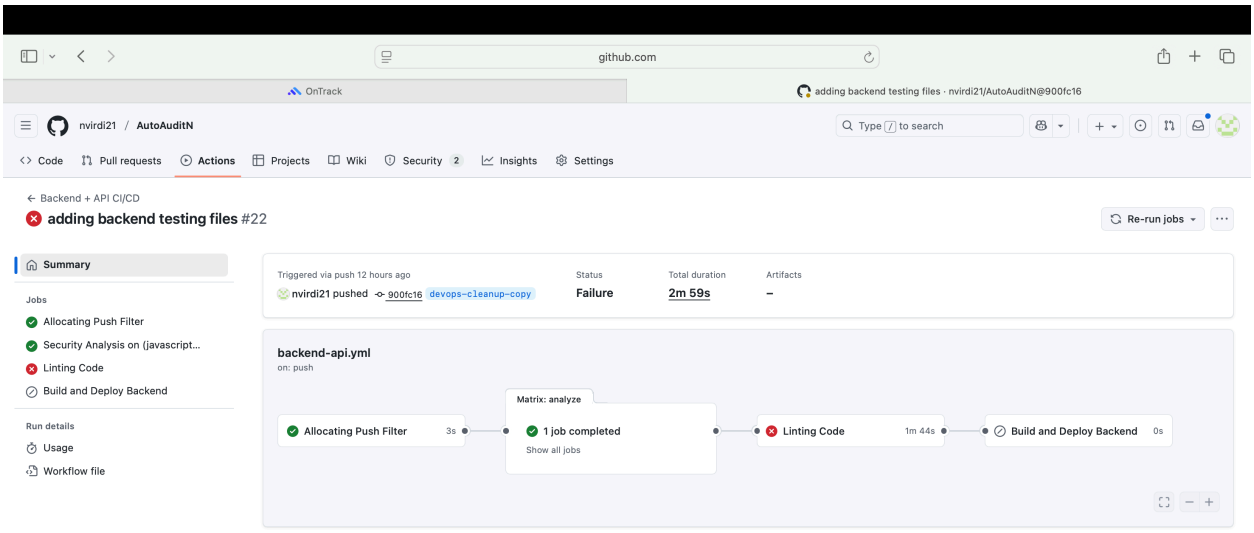

Figure 5: Backend CI — Linting failed (intentional lint error added for testing)

Link [Backend CI/CD] - https://github.com/nvirdi21/AutoAuditN/actions/runs/17849115964

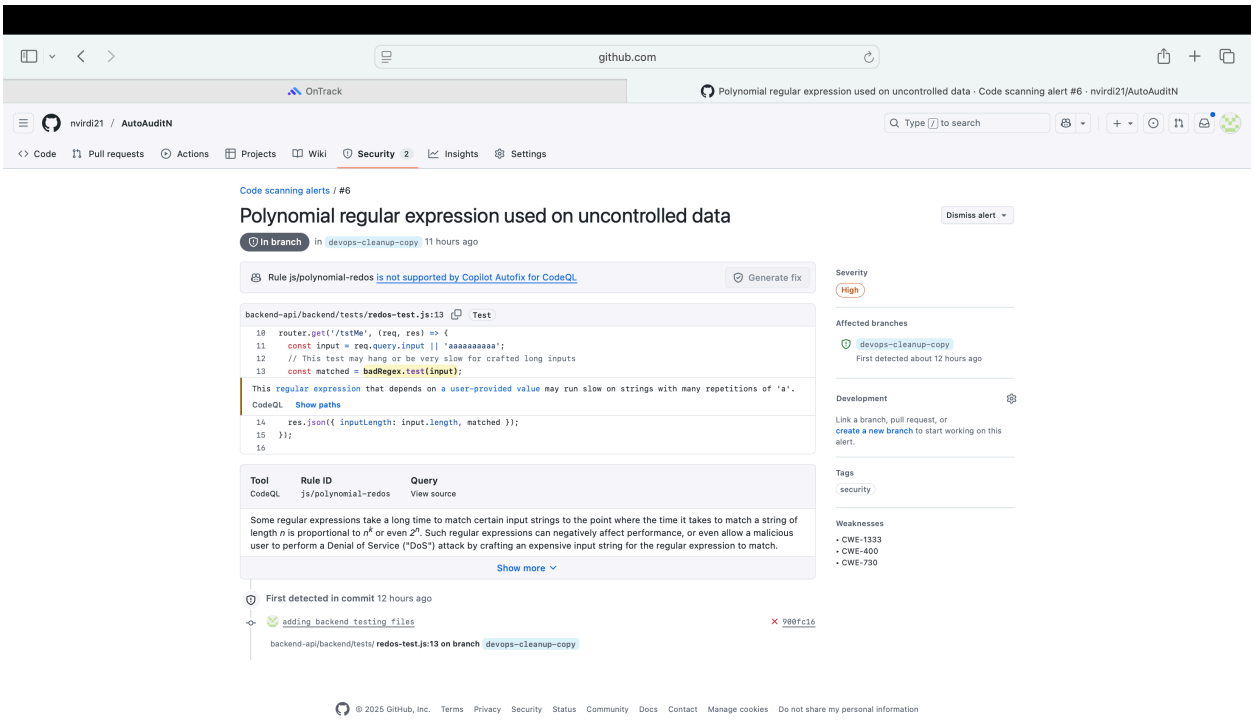

Figure 6: CodeQL security alert confirming ReDos Vulnerability

Link [Security Tab] - https://github.com/nvirdi21/AutoAuditN/security/code-scanning/6

2.2 Issues Faced & Fixes

- **Initial Problem:**
  - GitHub workflows flagged intentional **linting errors** (e.g., unused variables), but CodeQL was not flagging the vulnerabilities.
- **Fixes:**
  1. Added codeql-analysis.yml workflow file.

     - ***Why**: By introducing this workflow file, integrated CodeQL into the CI/CD pipeline, enabling automated security scans on each commit. This allowed the injected vulnerabilities (SQL Injection, ReDoS, XSS, etc.) to be properly detected and flagged under the repository's Security → Code scanning alerts tab.*

  2. Still no alerts visible in the **Security tab** initially.
  3. Root cause: Security tab was filtering **only main branch**.
  4. Switching the filter to my working branch (devops-cleanup-copy) showed the alerts correctly.

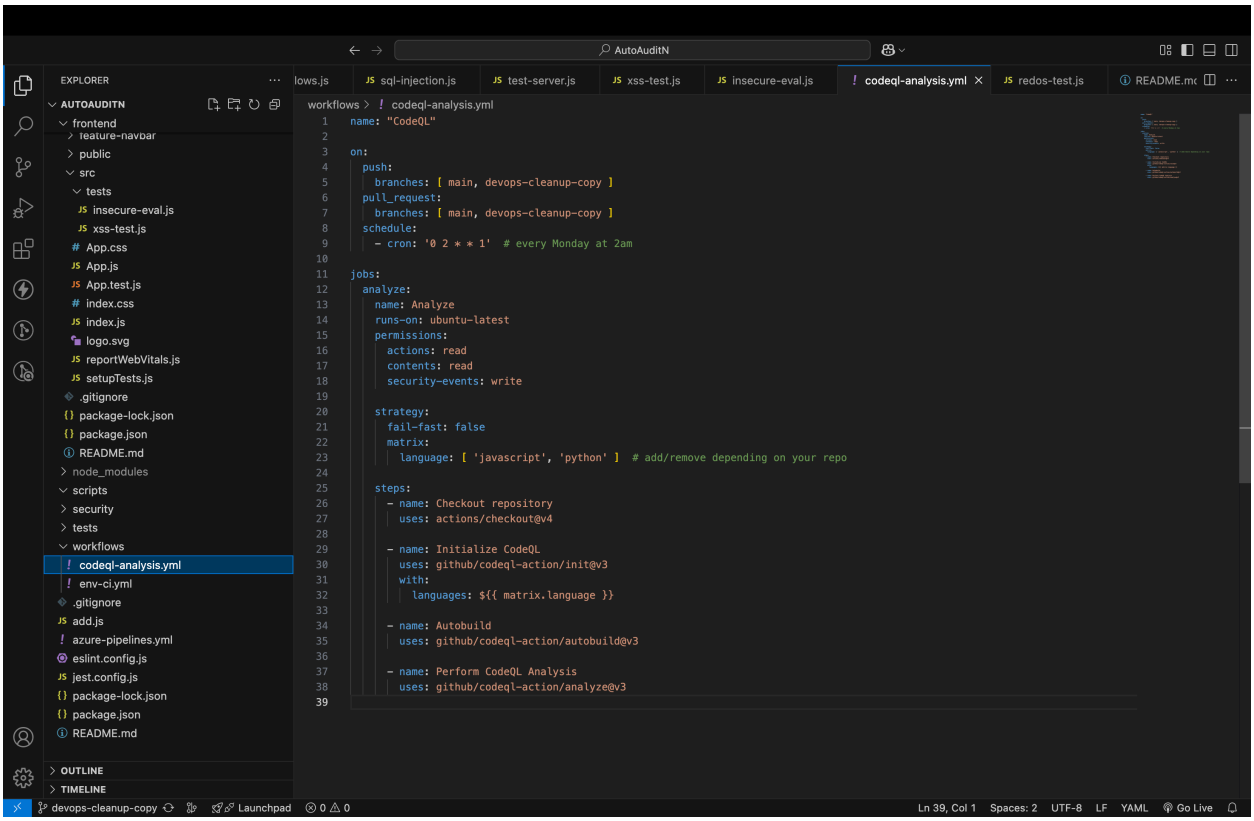

Figure 7: CodeQL Analysis Code

Commit Link - https://github.com/nvirdi21/AutoAuditN/commit/6c9c156238f76f5654fb46f44c2ae50b9118587e

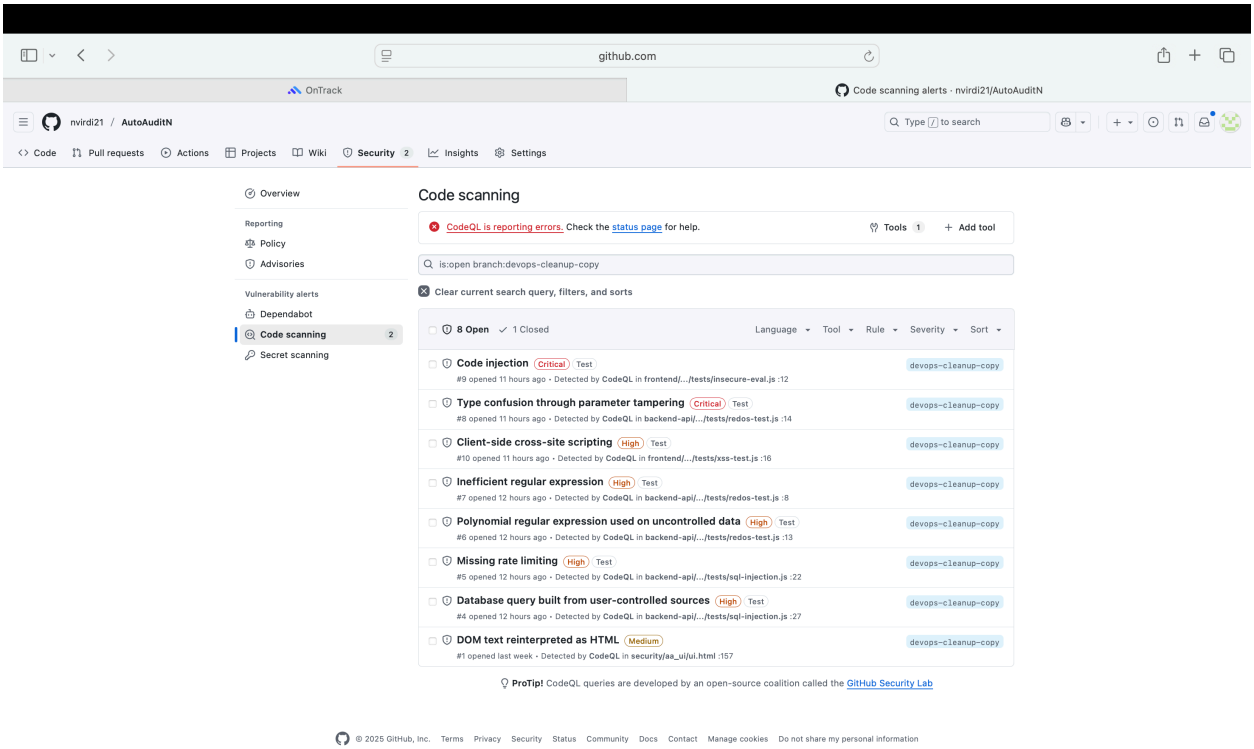

Figure 8: Selecting the correct flitter

3. Frontend Testing

3.1 Vulnerabilities Added

Two vulnerabilities were added under frontend/src/tests/ to reflect common risks in frontend systems like AutoAudit:

1. **Cross-Site Scripting (XSS)**
    - **Description**: XSS occurs when user input is rendered into the DOM without sanitization, allowing attackers to inject malicious scripts.
    - **Why Chosen**: XSS is a frequent vulnerability in frontend projects. AutoAudit's frontend (React-based) dynamically renders user-facing data, making it relevant.
    - **Dataset Reference**: Reviewed GitHub dataset Broken-Vulnerable-Code-Snippets – XSS.
    - **Note**: The dataset code does not directly match React (used in AutoAudit). The concept was adapted into a React component xss-test.js under frontend/src/tests/ to simulate unsafe rendering of user input using dangerouslySetInnerHTML.

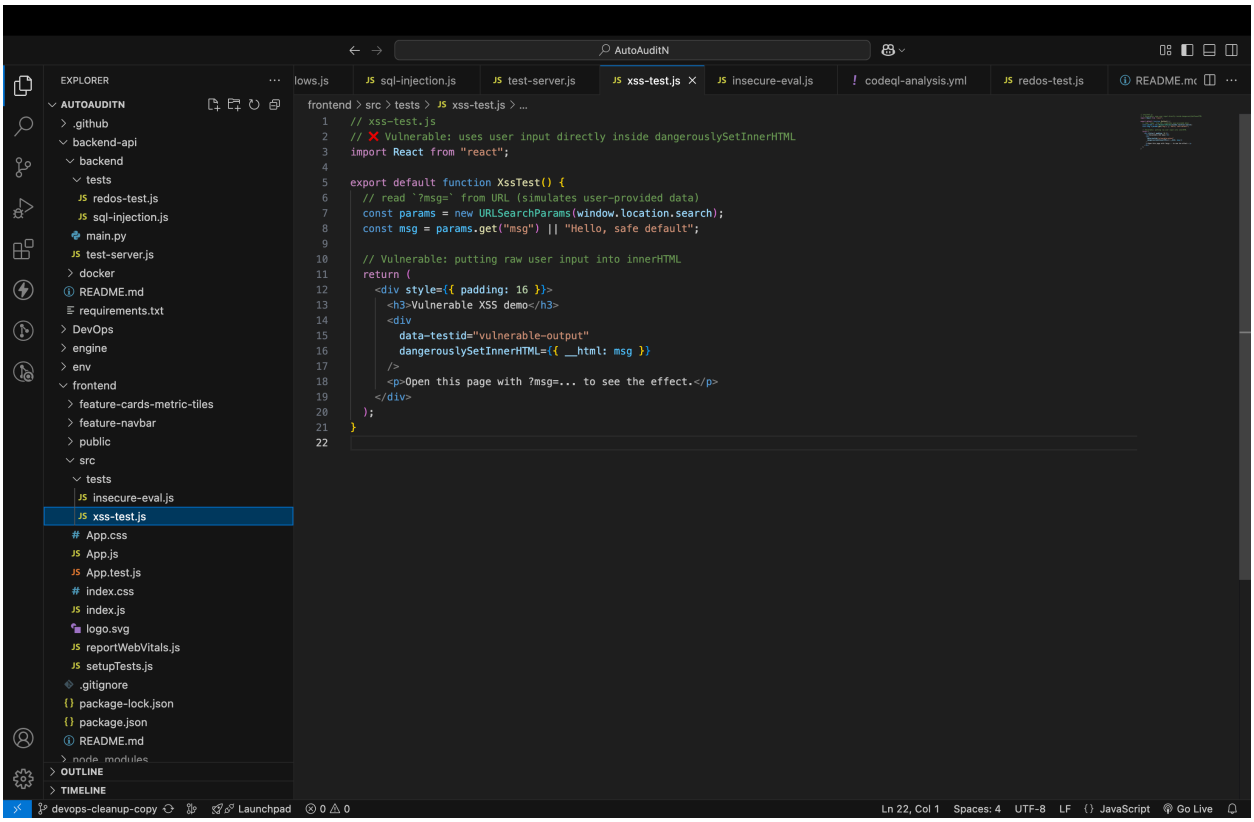

Figure 9: XSS Test Code

Commit Link -
https://github.com/nvirdi21/AutoAuditN/commit/8a533118fece4caa2d078f873649af272e468d4c

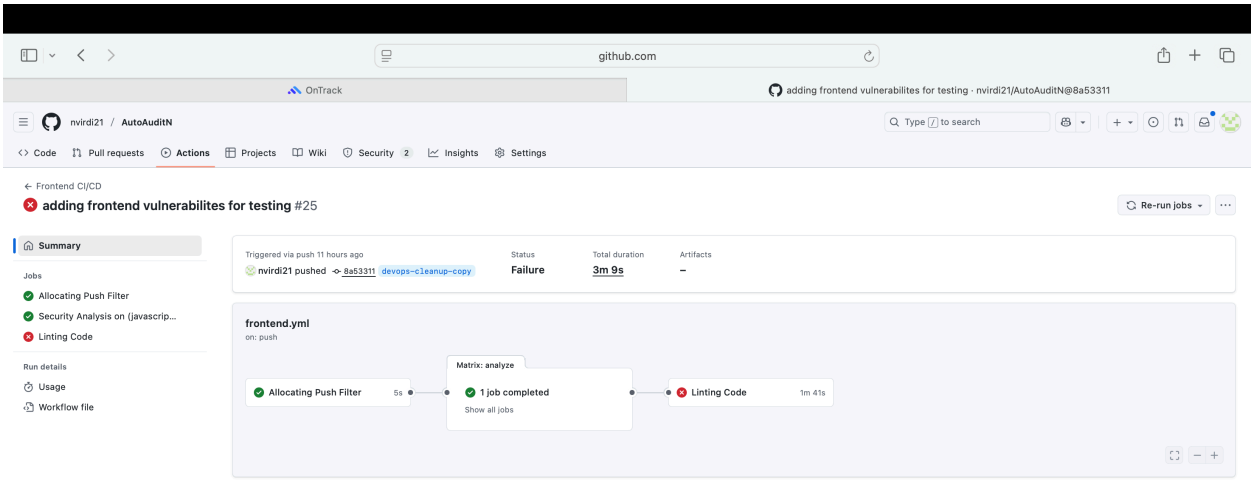

Figure 10: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Frontend CI/CD] - https://github.com/nvirdi21/AutoAuditN/actions/runs/17850315156

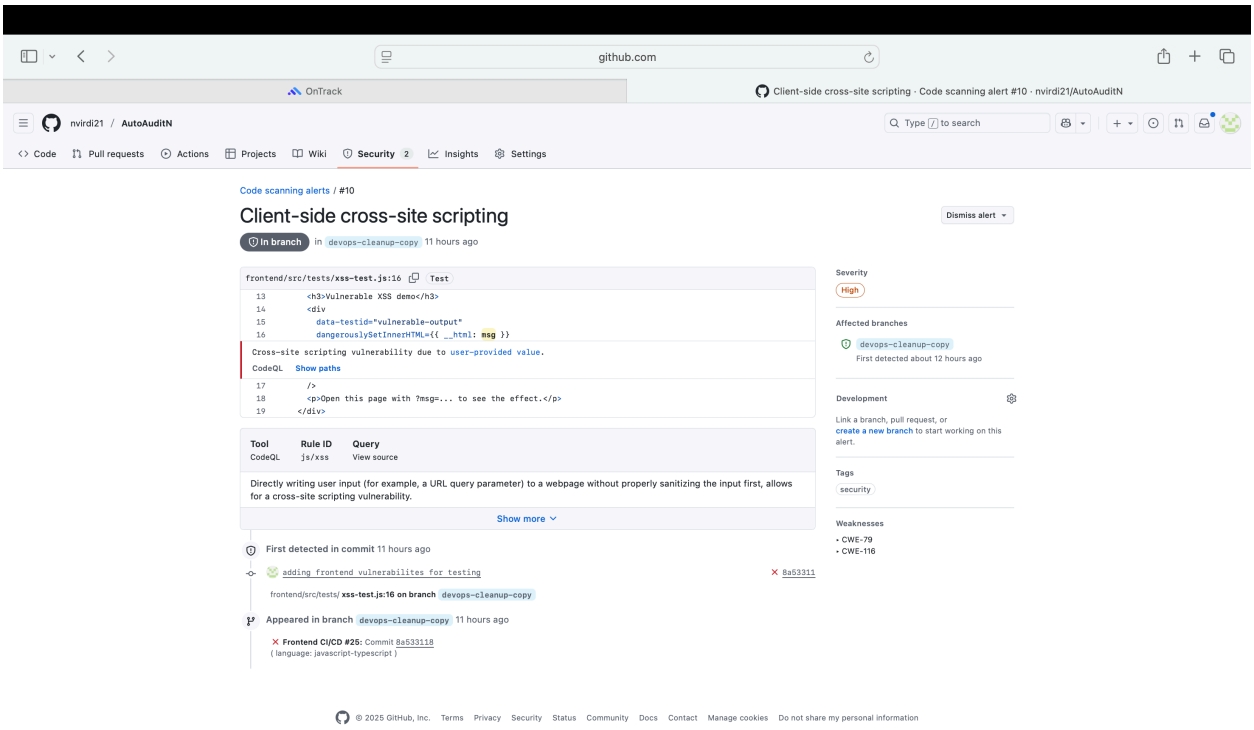

Figure 11: CodeQL security alert confirming XSS Vulnerability

Link [Security Tab] - https://github.com/nvirdi21/AutoAuditN/security/code-scanning/10

2. **Insecure eval Usage**
    - **Description**: Using eval() to execute user-provided input allows arbitrary code execution, making it a critical frontend security risk.
    - **Why Chosen**: Although React projects do not normally use eval(), this vulnerability was added to demonstrate detection of code execution risks in JavaScript projects.
    - **Dataset Reference**: Reviewed GitHub dataset [Broken-Vulnerable-Code-Snippets – Code Injection].
    - **Note**: The dataset uses PHP and NodeJS examples, which do not match React directly. The concept was adapted into a React file insecure-eval.js under frontend/src/tests/ where eval() executes unsanitized input.

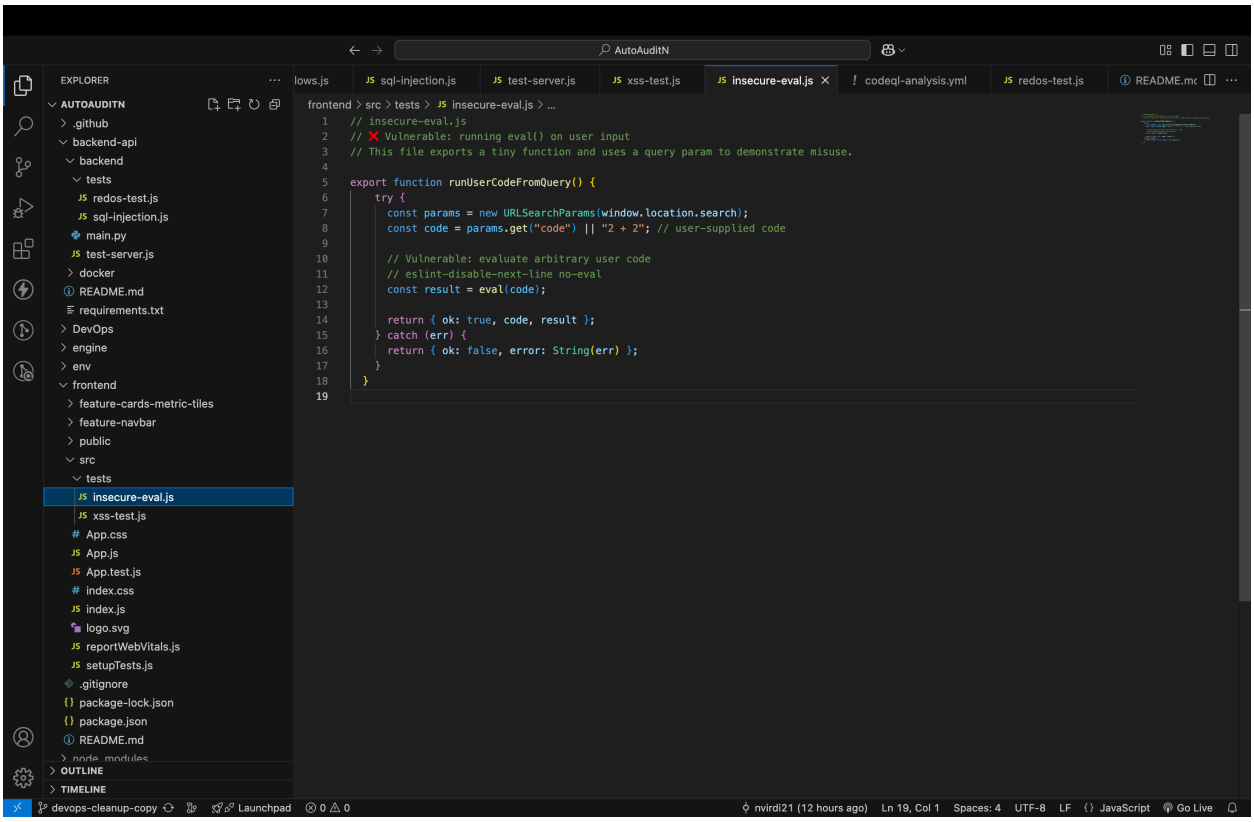

Figure 12: Insecure - eval Code

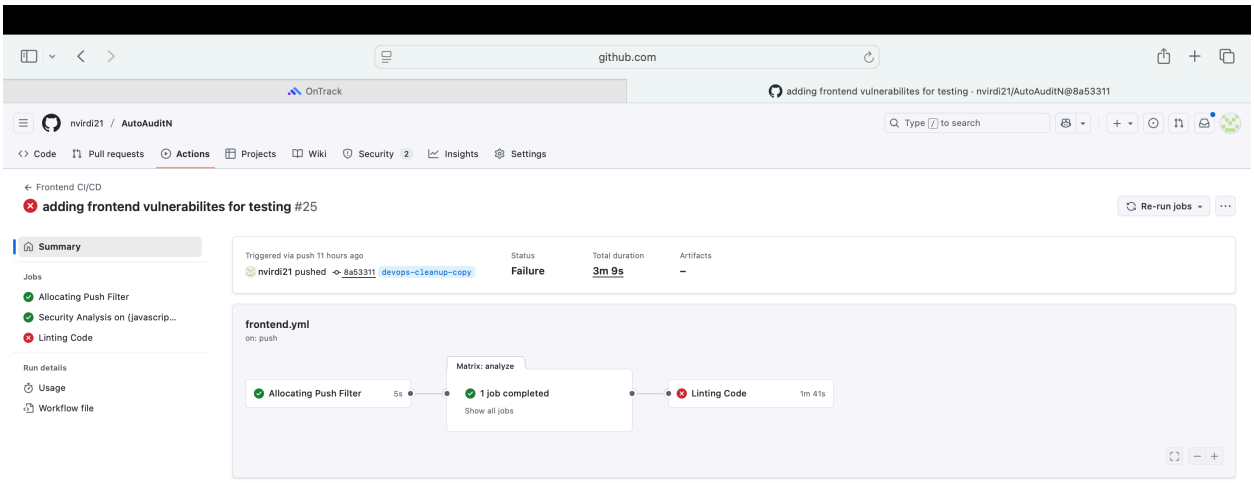

Figure 13: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Frontend CI/CD] - https://github.com/nvirdi21/AutoAuditN/actions/runs/17850315156

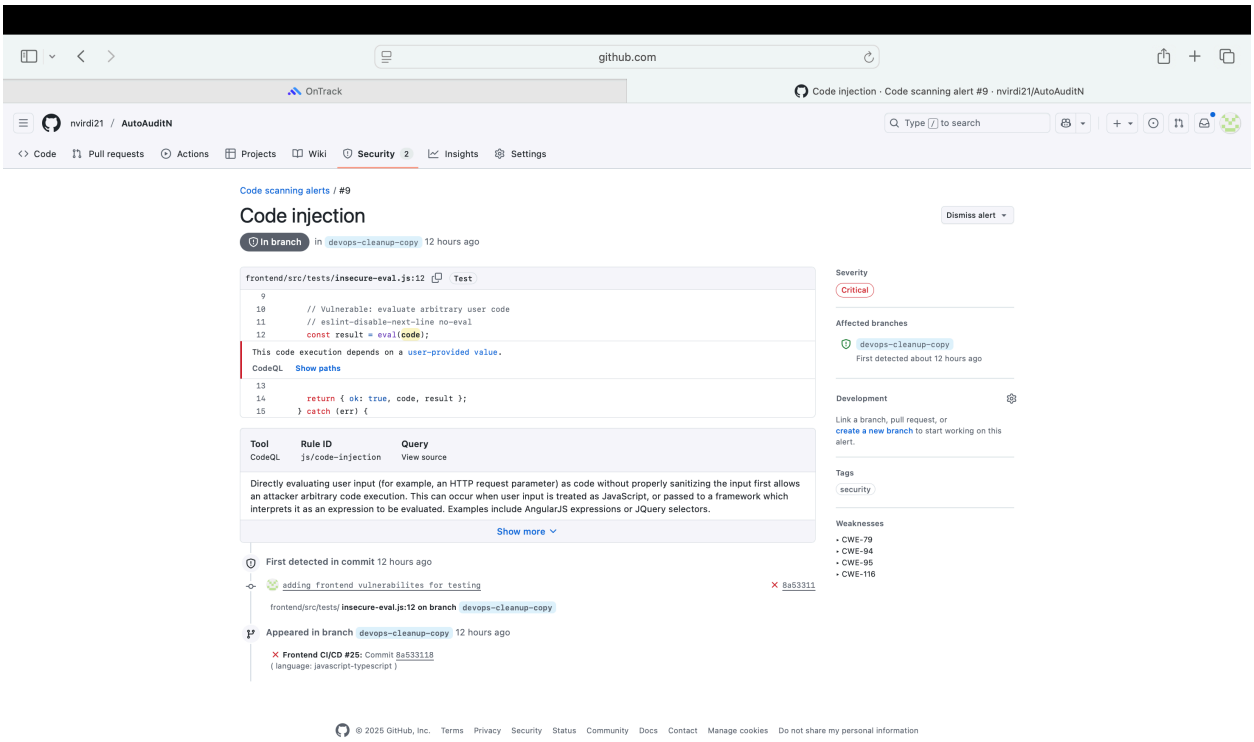

Figure 14: Frontend CI — Linting failed (intentional lint error added for testing)

Link [Security Tab] - https://github.com/nvirdi21/AutoAuditN/security/code-scanning/9

3.2 Issues Faced & Fixes

- Same CodeQL flagging issue as backend.
- Once vulnerabilities were pushed, CodeQL flagged the eval() injection as Critical and the XSS vulnerability as High under the *devops-cleanup-copy* branch filter.

---

4. Results

- **Backend:** SQL Injection + ReDoS vulnerabilities successfully added and flagged by CodeQL after branch filter adjustment.
- **Frontend:** Code Injection (eval) flagged as Critical; XSS vulnerability introduced for testing.
- Demonstrated full cycle: introduce vulnerabilities → commit → GitHub Actions workflow → CodeQL analysis → Security tab alerts.
