****

Academic year 2025-2026

**20CYS401**

**Secure Software Engineering**

Lab Record

*Submitted by*

P Thanusree

CH.EN.U4CYS22038

**Amrita Vishwa Vidyapeetham Chennai – 601 103, Tamil Nadu, India.**

**October 2025**

**Question:For the given application complete full secure devops lifecycle within single lab session**

**Application name: Feeddback submission portal**

**Suggested technology stack:Node.js+mongoDB**

**Focus area/expected output:cross site scripting prevention**

**5.Test Case Generation**

Write 3 test cases (functional, security, negative).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC ID | Type | Title | Preconditions | Test Steps (Numbered) | Test Data / Input | Expected Result |
| TC-F-01 | Functional | Submit feedback successfully | User is authenticated (or anonymous flow allowed); feedback page loads. | 1. Open Feedback page. 2. Fill required fields. 3. Click Submit. 4. Refresh or navigate to admin/view page to view stored feedback. | **Name:** "Asha" **Email:** "asha@example.com" **Rating:** 4 **Comment:** "Great service." | 1. Submission returns HTTP 200/201. 2. UI shows success message. 3. Feedback appears in the database and on view page with content exactly as entered (properly encoded). |
| TC-S-01 | Security | Stored XSS attempt via feedback comment (sanitization check) | Tester has an account or can submit feedback; viewing page shows stored comments without additional sanitization. | 1. Open Feedback page. 2. In comment field, submit payload. 3. Click Submit. 4. As a different user (or admin), open the page that lists stored feedback. 5. Observe if script executes or is shown escaped. | **Payload:** <script>alert('XSS')</script> | Expected: Payload is not executed. The UI must display an escaped/sanitized version or strip dangerous tags. No JS alert, no cookie/session leakage. Server logs the attempt. HTTP response should be 200 but stored value must be safe. |
| TC-N-01 | Negative | Reject overly long / malformed input (input validation) | Feedback form available; server enforces max lengths. | 1. Open Feedback page. 2. Enter required fields; in comment field input a string > 10,000 chars or invalid email in email field. 3. Click Submit. 4. Observe response and any error messages. | **Very long comment:** 20,000 'a' characters **Email:** not-an-email | Expected: Server-side rejects or truncates with clear validation error (HTTP 400). UI shows readable error messages (e.g., "Comment too long" or "Invalid email"). No crash, no 500 error, and nothing is stored in DB. |

Test case1 :

Functional:

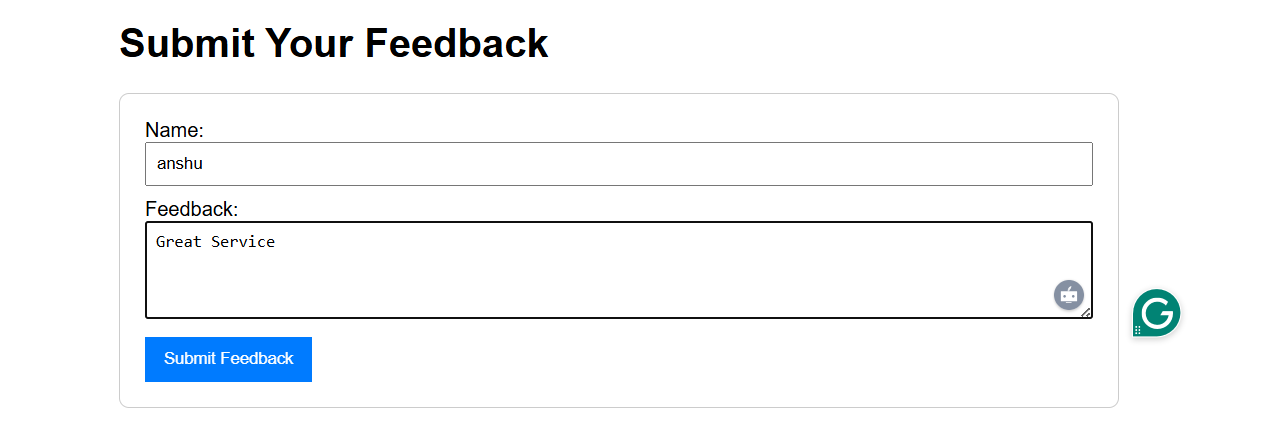
**Test 1: TC-F-01 (Functional)**

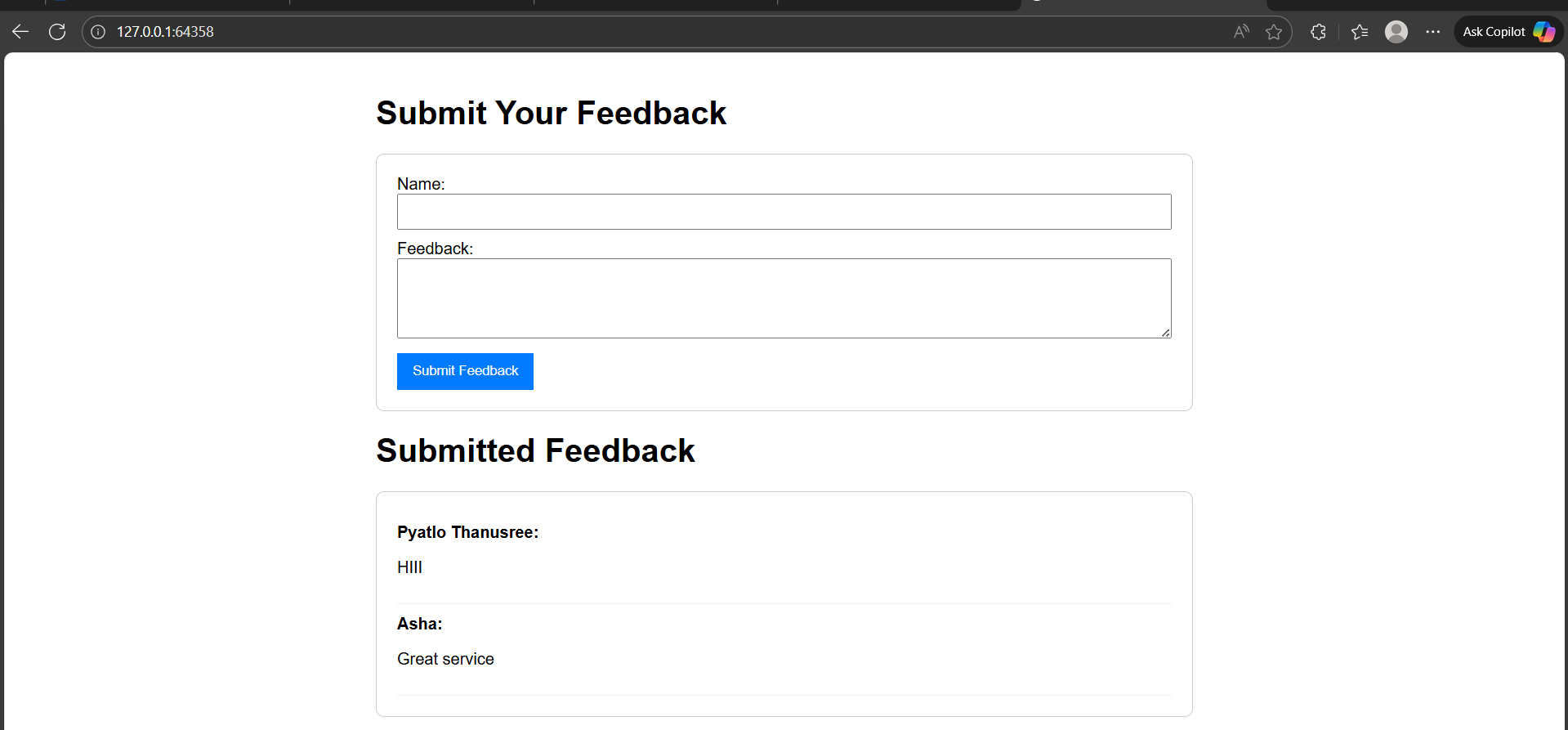
**Goal:** To verify that the application works correctly for a normal user submitting valid feedback.

**How to Execute:**

1. **Step 1 (Open Page):** You have already done this by running the minikube service command.
2. **Step 2 (Fill Form):** In the web page, fill in the form fields with the "Test Data / Input" from your document.
   * **Name:** Asha
   * **Feedback:** Great service. *(Note: The other fields like Email and Rating are not in our simple application, so we only test the ones we have).*
3. **Step 3 (Click Submit):** Click the **"Submit Feedback"** button.

Output:





2. **Test 2: TC-S-01 (Security)**

**Goal:** To verify that the application correctly handles a malicious script submission, preventing an XSS attack. The expected result depends on which version of server.js you are currently running.

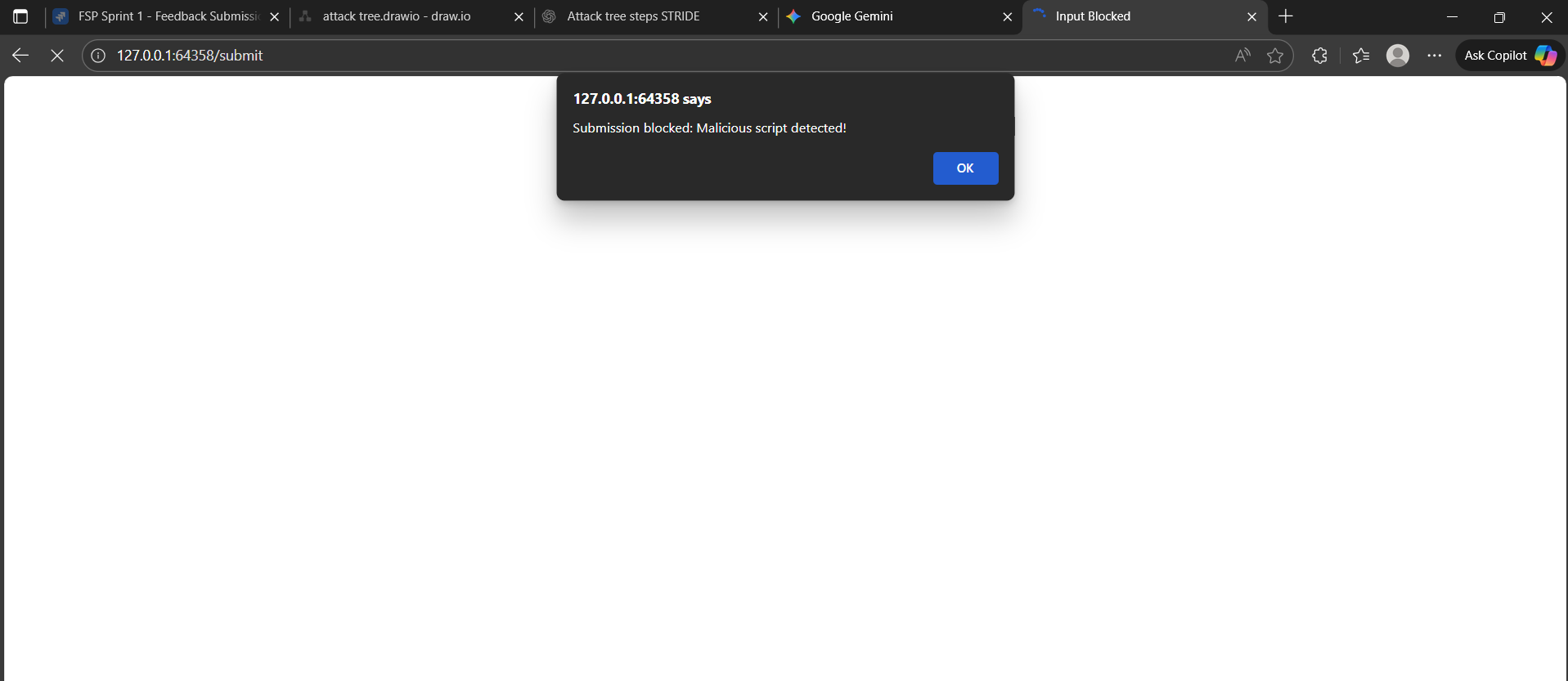
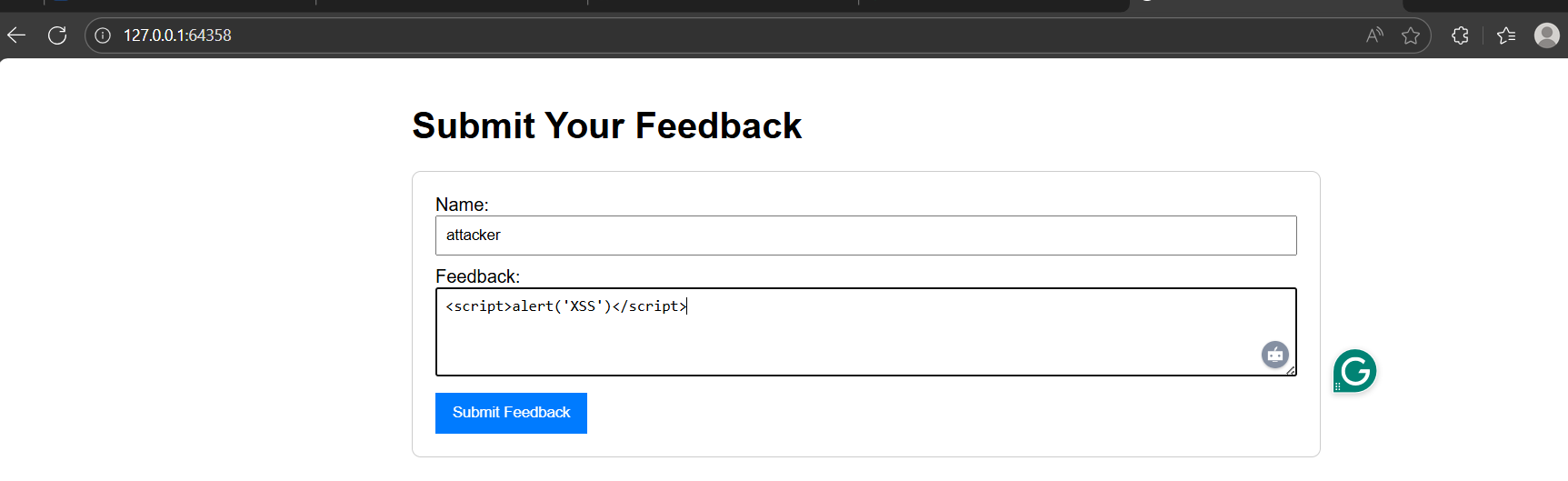
* **If you are using the "Output Encoding" version (the first one):** The script will be displayed safely as text.
* **If you are using the "Input Validation" version (the second one):** The script will be blocked, and a pop-up will appear.

I'll describe how to test the **Input Validation** version, as that's the one you wanted to implement.

**How to Execute:**

1. **Step 1 (Open Page):** Make sure you are on the main application page.
2. **Step 2 (Enter Payload):** In the feedback form, enter the malicious payload.
   * **Name:** Attacker
   * **Feedback:** <script>alert('XSS')</script>
3. **Step 3 (Click Submit):** Click the **"Submit Feedback"** button.

Output:



**Test 3: TC-N-01 (Negative)**

**Goal:** To test how the application handles invalid input. In our application, the simplest "invalid input" is an empty field.

**How to Execute:**

1. **Step 1 (Open Page):** Go to the main application page.
2. **Step 2 (Enter Incomplete Data):** Fill out one field but leave another one empty.
   * **Name:** Test User
   * **Feedback:** *(Leave this field completely blank)*
3. **Step 3 (Click Submit):** Click the **"Submit Feedback"** button.

Output:

