SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week10 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:20.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 20 – Security Testing: Identifying Vulnerabilities in AI-Generated Code** **Lab Objectives:** • Understand how to test AI-generated code for common security vulnerabilities. • Learn to apply secure coding principles while analyzing AI	Week10 - Monday

outputs.

- Practice detecting risks such as **SQL injection, XSS, hardcoded credentials, and weak encryption**.
- Enhance code reliability and safety by using AI for secure refactoring.

Task 1 – Input Validation Check

Task:

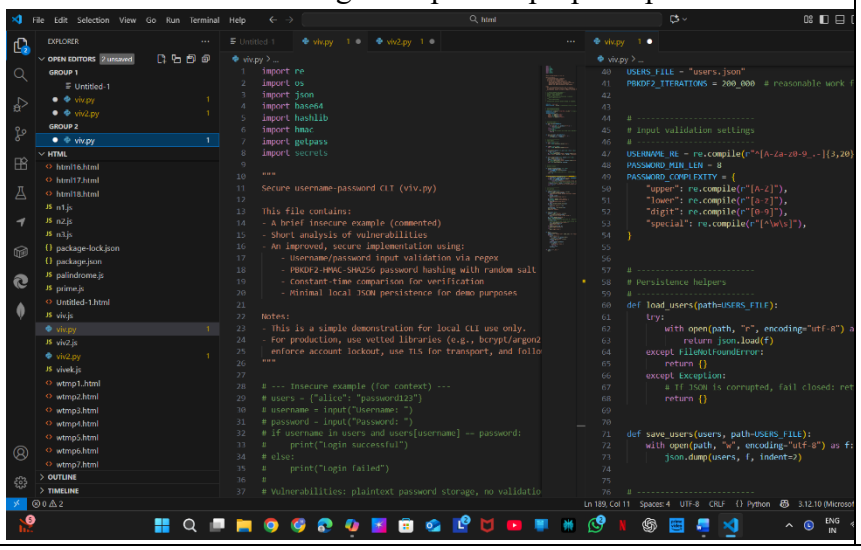
Analyze an AI-generated **Python login script** for input validation vulnerabilities.

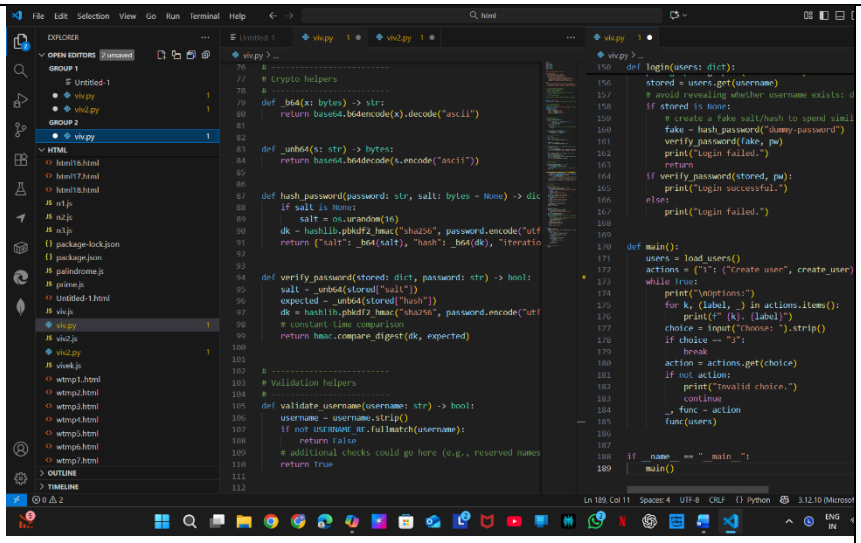
Instructions:

- Prompt AI to generate a simple username-password login program.
- Review whether input sanitization and validation are implemented.
- Suggest secure improvements (e.g., using re for input validation).

Expected Output:

- A secure version of the login script with proper input validation.
-





- Explanation: CLI for local user management: create accounts and log in, storing data in users.json.
- Username validated with a regex; passwords must meet length and complexity rules and have no control/leading-trailing whitespace.
- Passwords are hashed with PBKDF2-HMAC-SHA256 using a random 16-byte salt and 200,000 iterations; salt and hash are base64-stored.
- Verification uses hmac.compare_digest for constant-time comparison and performs a fake verify on missing users to reduce timing leaks.
- Exposes create_user, login, and a menu loop; includes comments advising stronger production practices (bcrypt/argon2, TLS, account lockout).
-

Task 2 – SQL Injection Prevention

Task:

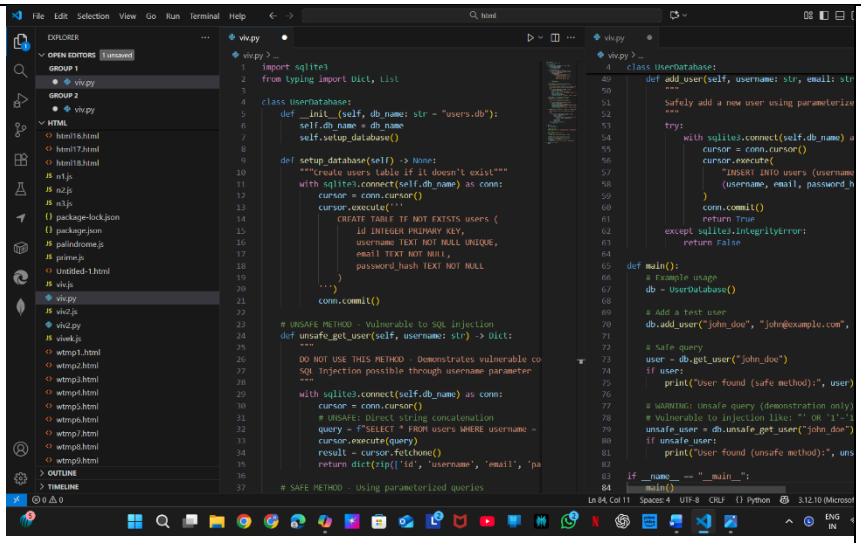
Test an AI-generated script that performs SQL queries on a database.

Instructions:

- Ask AI to generate a Python script using SQLite/MySQL to fetch user details.
- Identify if the code is vulnerable to **SQL injection** (e.g., using string concatenation in queries).
- Refactor using **parameterized queries (prepared statements)**.

Expected Output:

- A secure database query script resistant to SQL injection.



- Explanation: Manages an SQLite users database and ensures the users table exists via setup_database().
- unsafe_get_user shows an insecure string-built query vulnerable to SQL injection (do not use).
- get_user uses parameterized queries to safely fetch a user by username.
- add_user inserts a new user with parameterized queries and returns False on IntegrityError (e.g., duplicate username).
- main demonstrates usage: create DB, add a test user, then fetch with the safe method and (for demonstration) the unsafe method.
-

Task 3 – Cross-Site Scripting (XSS) Check

Task:

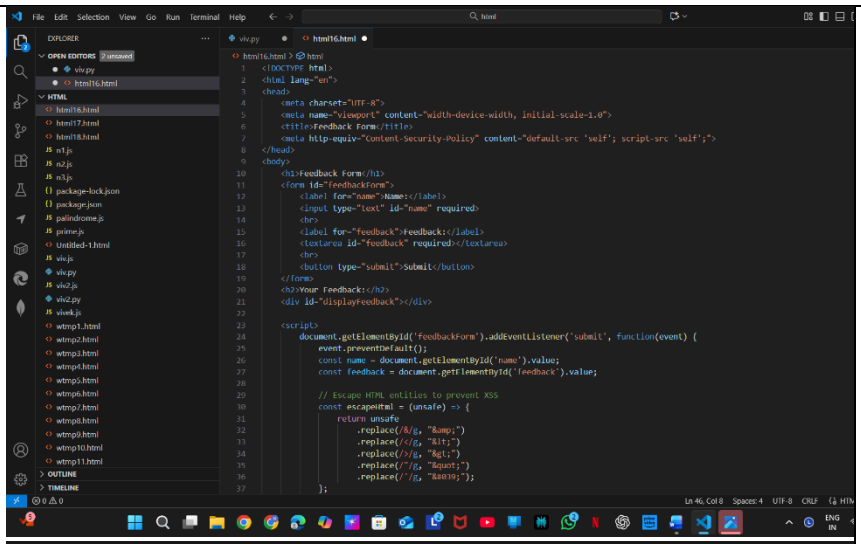
Evaluate an AI-generated **HTML form with JavaScript** for XSS vulnerabilities.

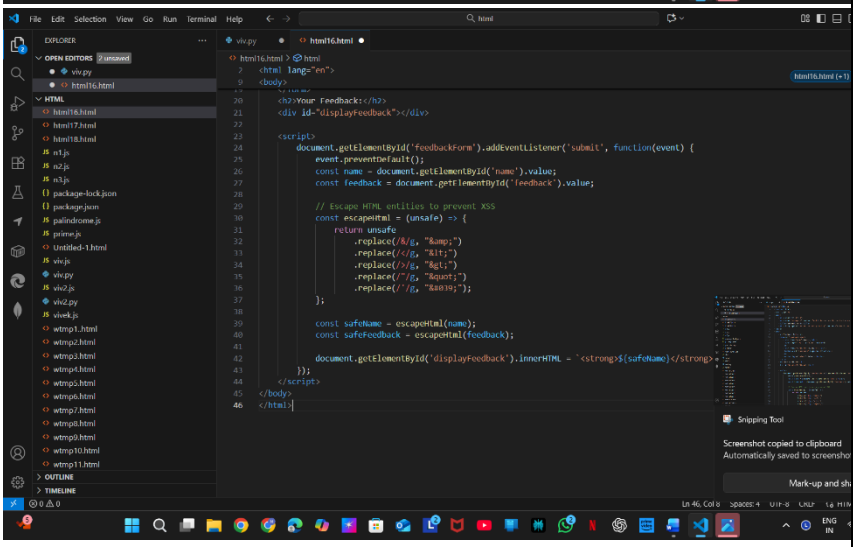
Instructions:

- Ask AI to generate a feedback form with JavaScript-based output.
- Test whether untrusted inputs are directly rendered without escaping.
- Implement secure measures (e.g., escaping HTML entities, using CSP).

Expected Output:

- A secure form that prevents XSS attacks.

- 

- 

- Explanation: Simple client-side feedback form with a Content-Security-Policy limiting scripts to 'self'.
- JavaScript intercepts the form submit, prevents default behavior, and reads the name and feedback inputs.
- escapeHtml replaces &, <, >, ", and ' with HTML entities to mitigate XSS.
- The code injects the escaped name and feedback into the page using innerHTML (name wrapped in).
- No server persistence — data stays in the browser; escaping + CSP reduce XSS risk but server-side handling would be needed for storage.
-

Task 4 – Real-Time Application: Security Audit of AI-Generated Code

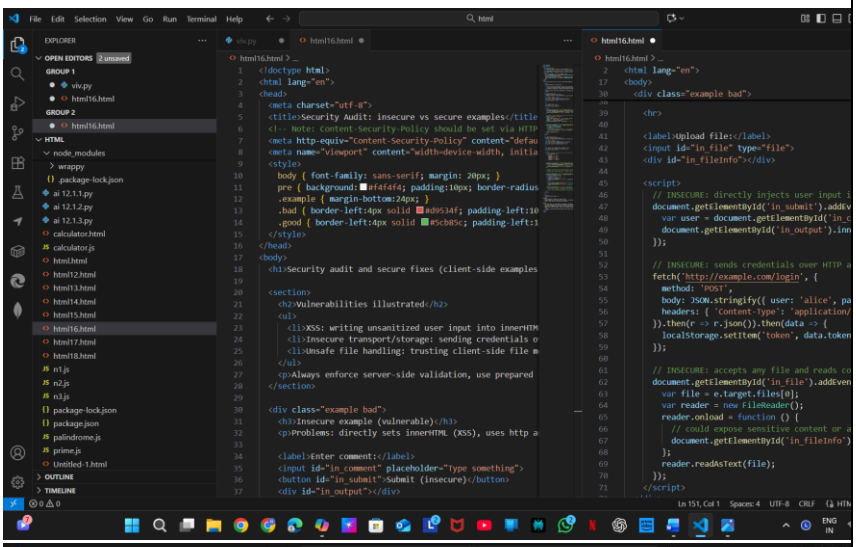
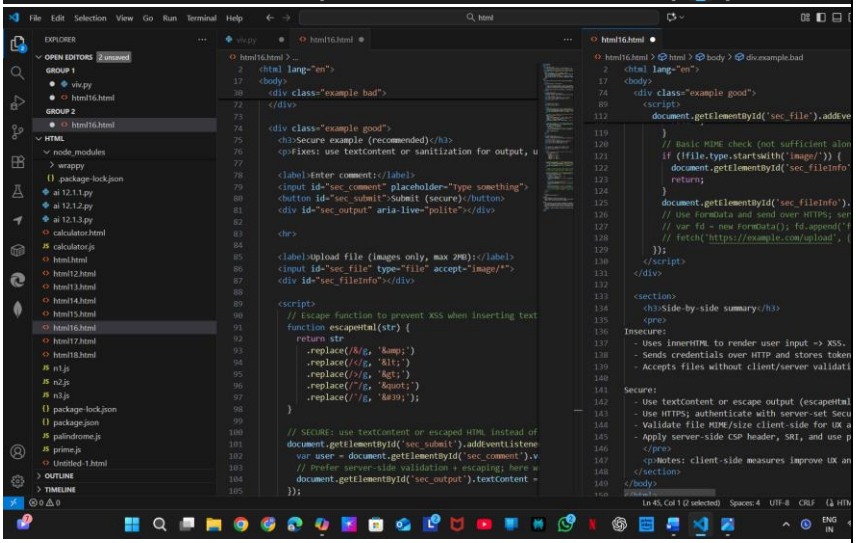
Scenario:

Students pick an **AI-generated project snippet** (e.g., login form, API integration, or file upload).

Instructions:

- Perform a security audit to detect possible vulnerabilities.
- Prompt AI to suggest **secure coding practices** to fix issues.
- Compare insecure vs secure versions side by side.

Expected Output:

- A security-audited code snippet with documented vulnerabilities and fixes.
- 
- 
- Explanation: HTML page demonstrating client-side insecure vs secure patterns and includes a meta Content-Security-Policy note.
- Insecure example: injects unsanitized input via innerHTML (XSS), posts credentials over HTTP and stores tokens in localStorage, and reads uploaded files with no validation.

	• Secure example: escapes input (escapeHtml) or uses textContent to prevent XSS, recommends HTTPS and server-set Secure/HttpOnly cookies instead of localStorage, and performs basic client-side file type/size checks. • File handling: secure code restricts to images and max 2MB for UX, but explicitly notes the server must re-validate, scan, and enforce stricter checks. • Page includes a side-by-side summary and emphasizes that client-side measures help UX but all security-critical checks must be implemented on the server (CSP headers, prepared statements, HTTPS, etc.). • ✅ Deliverables (For All Tasks) 1. AI-generated prompts for code and test case generation. 2. At least 3 assert test cases for each task. 3. AI-generated initial code and execution screenshots. 4. Analysis of whether code passes all tests. 5. Improved final version with inline comments and explanation. 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.	