- Ai lab assingment: 20.3

Rollno:2403a51102

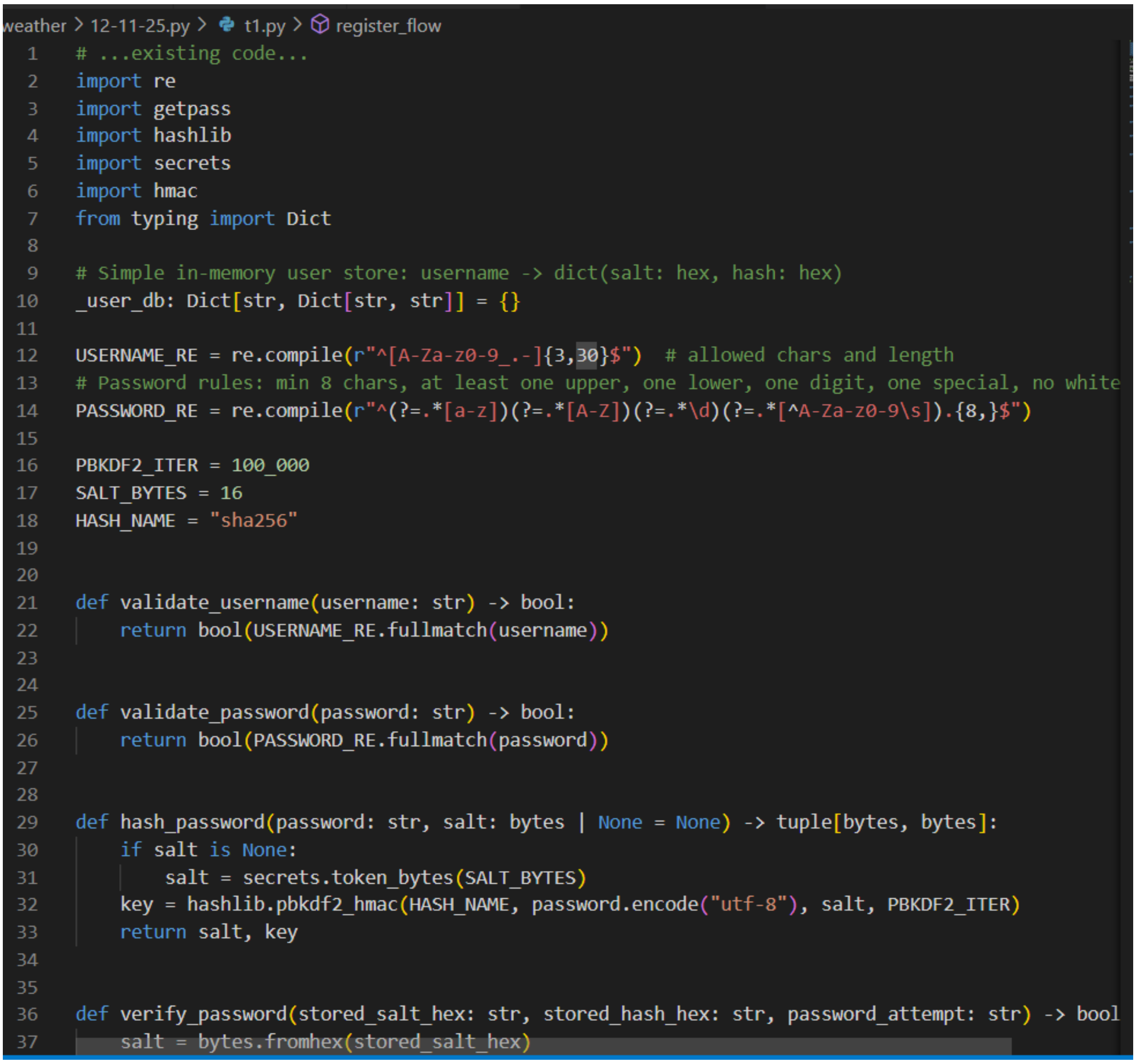
Name:balaji

Batch:06

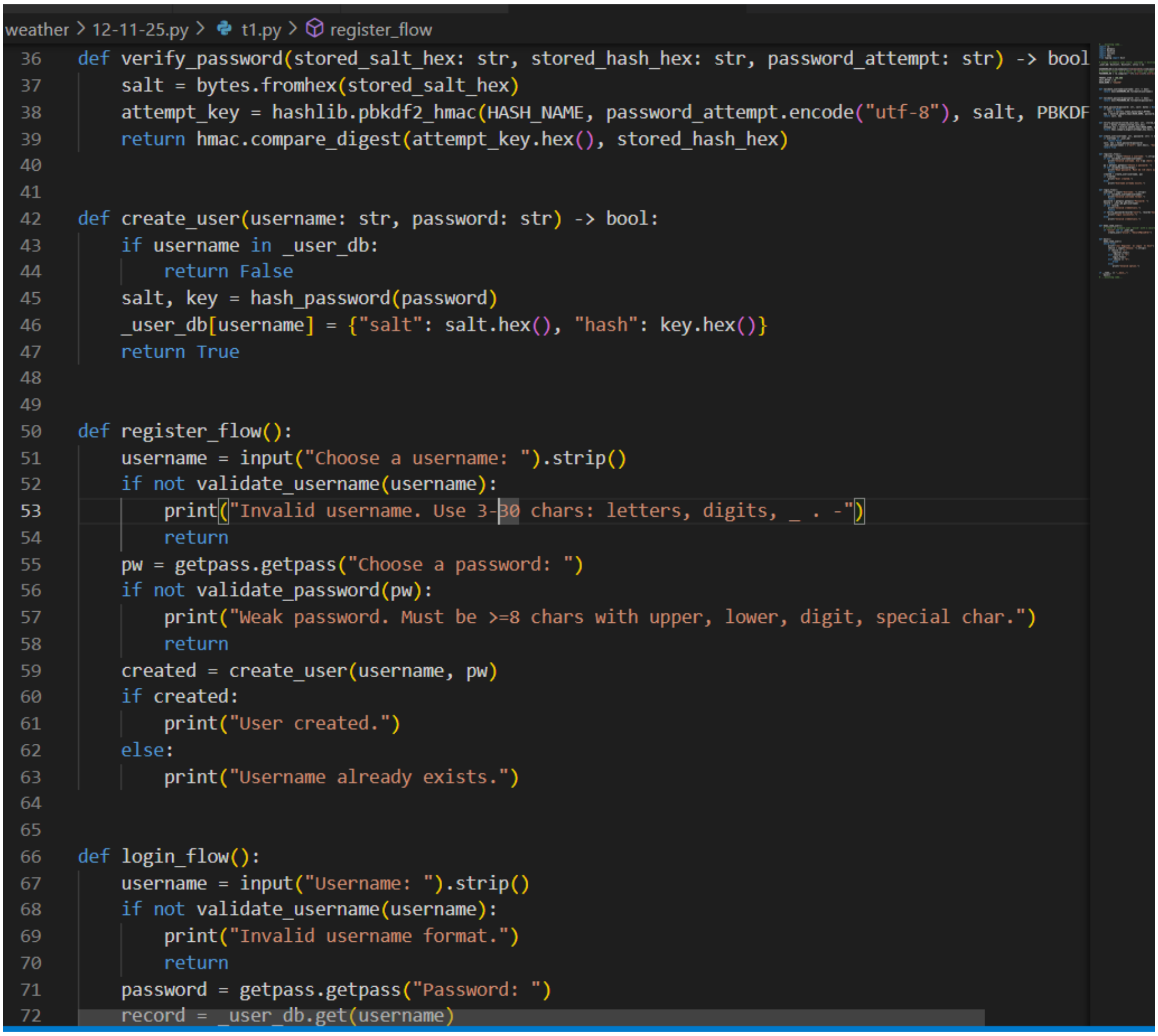
Task 1 – Input Validation Check
Task:
Analyze an AI-generated Python login script for input validation
vulnerabilities.
Instructions:
• Prompt AI to generate a simple username-password login
program.
• Review whether input sanitization and validation are implemented.
• Suggest secure improvements (e.g., using re for input validation).
Expected Output:
• A secure version of the login script with proper input validation.

Prompt: Generate a basic Python login script that asks for username and password and checks them against stored values.

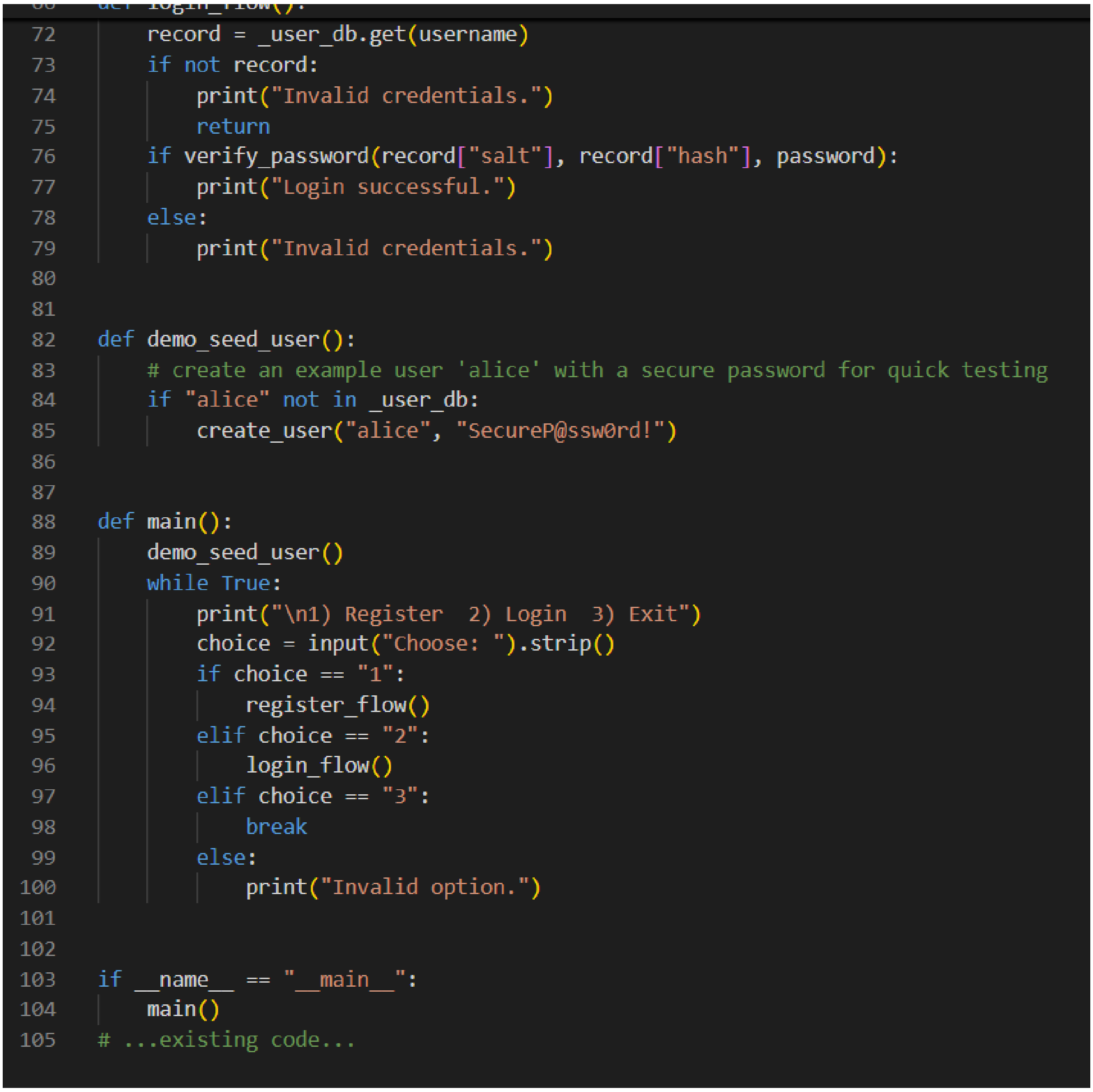
Code:



```python
# ...existing code...
import re
import getpass
import hashlib
import secrets
import hmac
from typing import Dict

# Simple in-memory user store: username -> dict(salt: hex, hash: hex)
_user_db: Dict[str, Dict[str, str]] = {}

USERNAME_RE = re.compile(r"^[A-Za-z0-9_.-]{3,30}$")  # allowed chars and length
# Password rules: min 8 chars, at least one upper, one lower, one digit, one special, no white
PASSWORD_RE = re.compile(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[^A-Za-z0-9\s]).{8,}$")

PBKDF2_ITER = 100_000
SALT_BYTES = 16
HASH_NAME = "sha256"


def validate_username(username: str) -> bool:
    return bool(USERNAME_RE.fullmatch(username))


def validate_password(password: str) -> bool:
    return bool(PASSWORD_RE.fullmatch(password))


def hash_password(password: str, salt: bytes | None = None) -> tuple[bytes, bytes]:
    if salt is None:
        salt = secrets.token_bytes(SALT_BYTES)
    key = hashlib.pbkdf2_hmac(HASH_NAME, password.encode("utf-8"), salt, PBKDF2_ITER)
    return salt, key


def verify_password(stored_salt_hex: str, stored_hash_hex: str, password_attempt: str) -> bool
    salt = bytes.fromhex(stored_salt_hex)
```

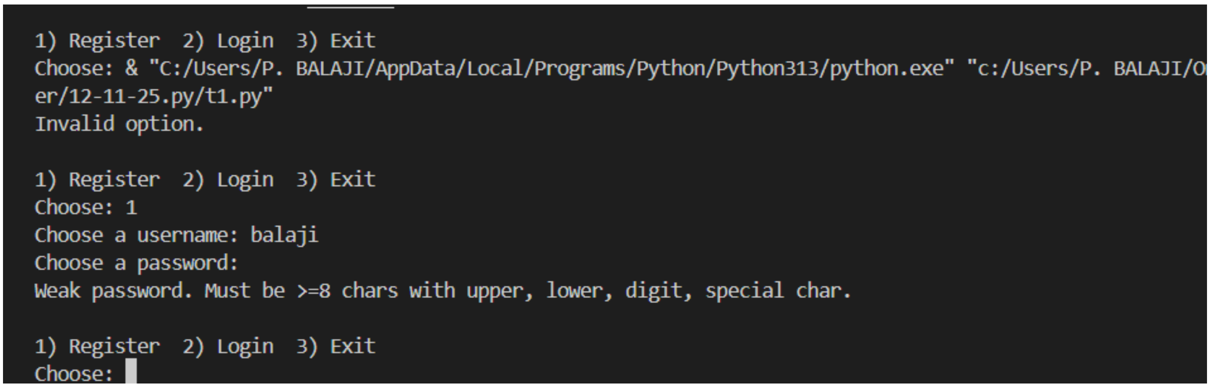


```python
def verify_password(stored_salt_hex: str, stored_hash_hex: str, password_attempt: str) -> bool
    salt = bytes.fromhex(stored_salt_hex)
    attempt_key = hashlib.pbkdf2_hmac(HASH_NAME, password_attempt.encode("utf-8"), salt, PBKDF
    return hmac.compare_digest(attempt_key.hex(), stored_hash_hex)


def create_user(username: str, password: str) -> bool:
    if username in _user_db:
        return False
    salt, key = hash_password(password)
    _user_db[username] = {"salt": salt.hex(), "hash": key.hex()}
    return True


def register_flow():
    username = input("Choose a username: ").strip()
    if not validate_username(username):
        print("Invalid username. Use 3-30 chars: letters, digits, _ . -")
        return
    pw = getpass.getpass("Choose a password: ")
    if not validate_password(pw):
        print("Weak password. Must be >=8 chars with upper, lower, digit, special char.")
        return
    created = create_user(username, pw)
    if created:
        print("User created.")
    else:
        print("Username already exists.")


def login_flow():
    username = input("Username: ").strip()
    if not validate_username(username):
        print("Invalid username format.")
        return
    password = getpass.getpass("Password: ")
    record = _user_db.get(username)
```



```python
def login_flow():
    record = _user_db.get(username)
    if not record:
        print("Invalid credentials.")
        return
    if verify_password(record["salt"], record["hash"], password):
        print("Login successful.")
    else:
        print("Invalid credentials.")


def demo_seed_user():
    # create an example user 'alice' with a secure password for quick testing
    if "alice" not in _user_db:
        create_user("alice", "SecureP@ssw0rd!")


def main():
    demo_seed_user()
    while True:
        print("\n1) Register  2) Login  3) Exit")
        choice = input("Choose: ").strip()
        if choice == "1":
            register_flow()
        elif choice == "2":
            login_flow()
        elif choice == "3":
            break
        else:
            print("Invalid option.")


if __name__ == "__main__":
    main()
# ...existing code...
```

Output:



```
1) Register  2) Login  3) Exit
Choose: & "C:/Users/P. BALAJI/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/P. BALAJI/O
er/12-11-25.py/t1.py"
Invalid option.

1) Register  2) Login  3) Exit
Choose: 1
Choose a username: balaji
Choose a password:
Weak password. Must be >=8 chars with upper, lower, digit, special char.

1) Register  2) Login  3) Exit
Choose: 
```

Observation:

No input checks — accepts anything.

Passwords stored in plain text.

No protection against brute-force attempts.

No format rules for username or password.

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).

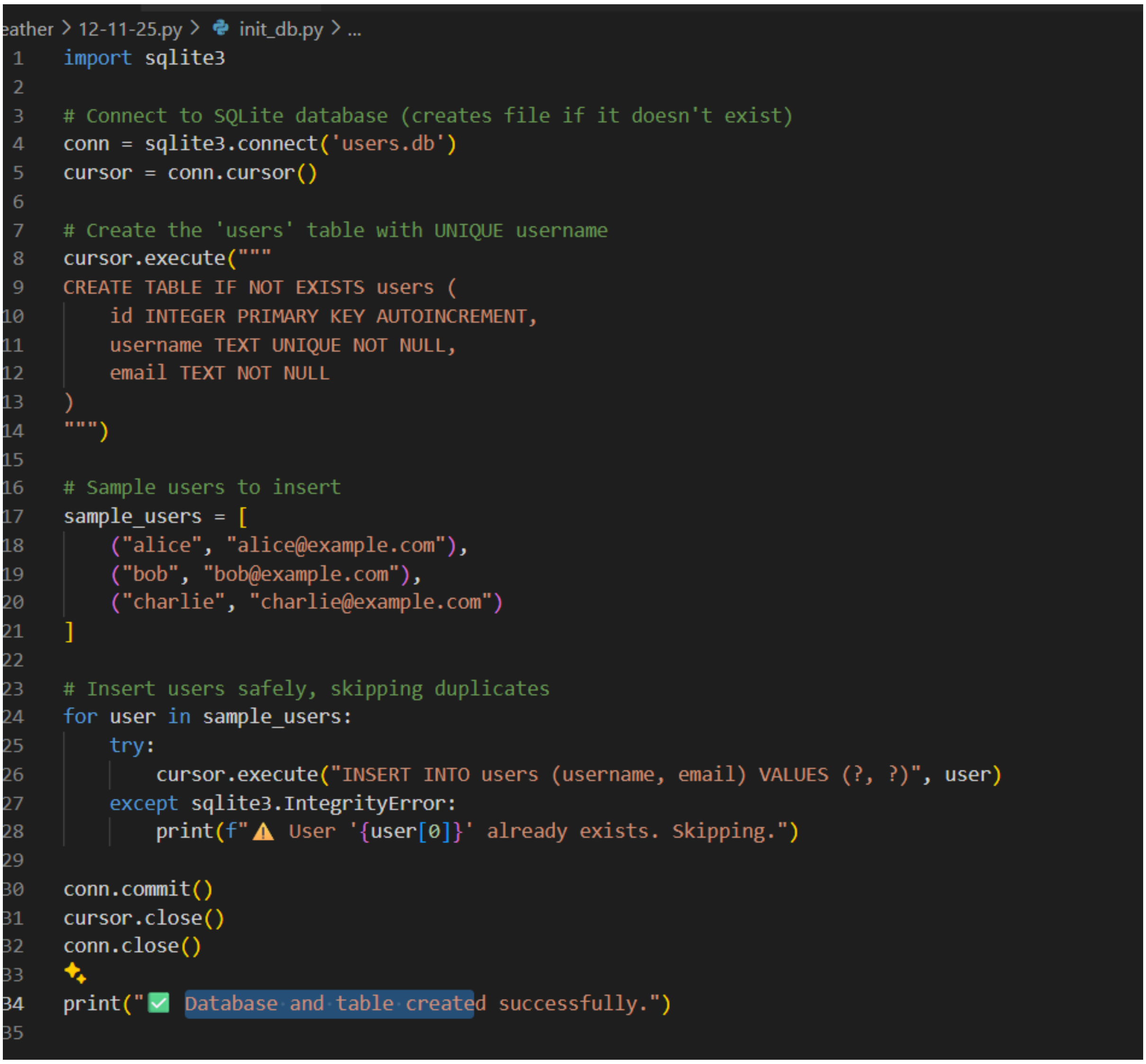
• Refactor using parameterized queries (prepared statements).

Expected Output:

• A secure database query script resistant to SQL injection.

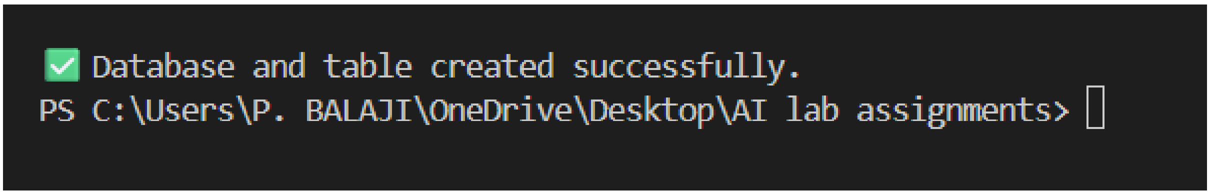
Prompt: Generate a Python script using SQLite to fetch user details by username.

Code: Database and table creating



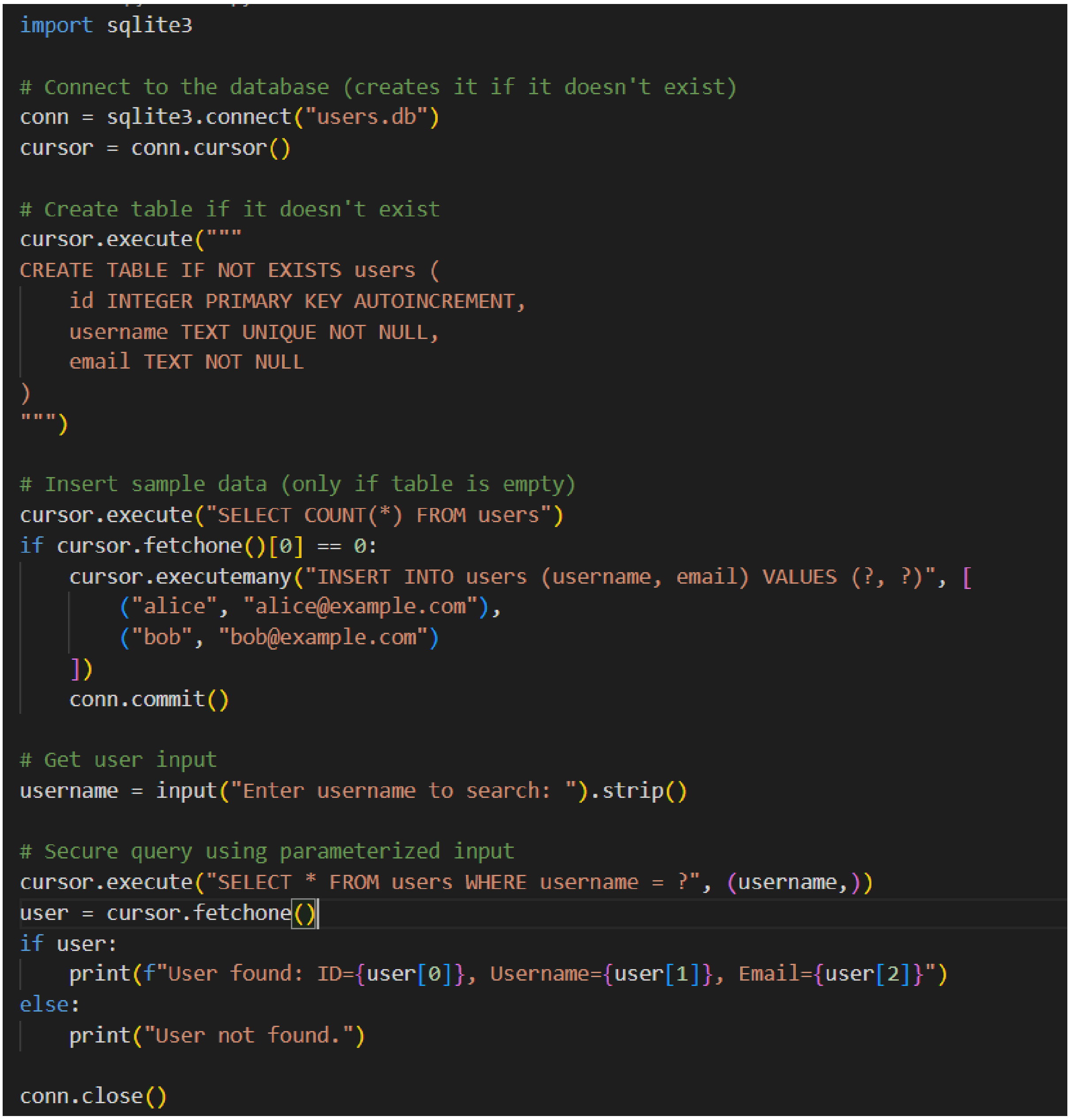
```python
import sqlite3

# Connect to SQLite database (creates file if it doesn't exist)
conn = sqlite3.connect('users.db')
cursor = conn.cursor()

# Create the 'users' table with UNIQUE username
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    email TEXT NOT NULL
)
""")

# Sample users to insert
sample_users = [
    ("alice", "alice@example.com"),
    ("bob", "bob@example.com"),
    ("charlie", "charlie@example.com")
]

# Insert users safely, skipping duplicates
for user in sample_users:
    try:
        cursor.execute("INSERT INTO users (username, email) VALUES (?, ?)", user)
    except sqlite3.IntegrityError:
        print(f"⚠ User '{user[0]}' already exists. Skipping.")

conn.commit()
cursor.close()
conn.close()

print("✅ Database and table created successfully.")
```

Output:



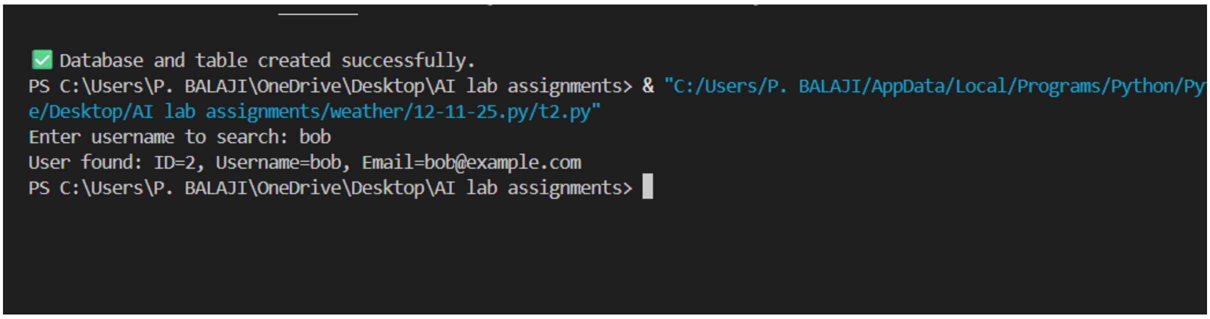
```
✅ Database and table created successfully.
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments> 
```

Code:main code



```python
import sqlite3

# Connect to the database (creates it if it doesn't exist)
conn = sqlite3.connect("users.db")
cursor = conn.cursor()

# Create table if it doesn't exist
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    email TEXT NOT NULL
)
""")

# Insert sample data (only if table is empty)
cursor.execute("SELECT COUNT(*) FROM users")
if cursor.fetchone()[0] == 0:
    cursor.executemany("INSERT INTO users (username, email) VALUES (?, ?)", [
        ("alice", "alice@example.com"),
        ("bob", "bob@example.com")
    ])
    conn.commit()

# Get user input
username = input("Enter username to search: ").strip()

# Secure query using parameterized input
cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
user = cursor.fetchone()
if user:
    print(f"User found: ID={user[0]}, Username={user[1]}, Email={user[2]}")
else:
    print("User not found.")

conn.close()
```

Output:



```
✅ Database and table created successfully.
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments> & "C:/Users/P. BALAJI/AppData/Local/Programs/Python/Py
e/Desktop/AI lab assignments/weather/12-11-25.py/t2.py"
Enter username to search: bob
User found: ID=2, Username=bob, Email=bob@example.com
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI lab assignments> █
```

Observation:

The script uses string concatenation in the SQL query → **vulnerable to SQL injection**.

No input validation or sanitization.

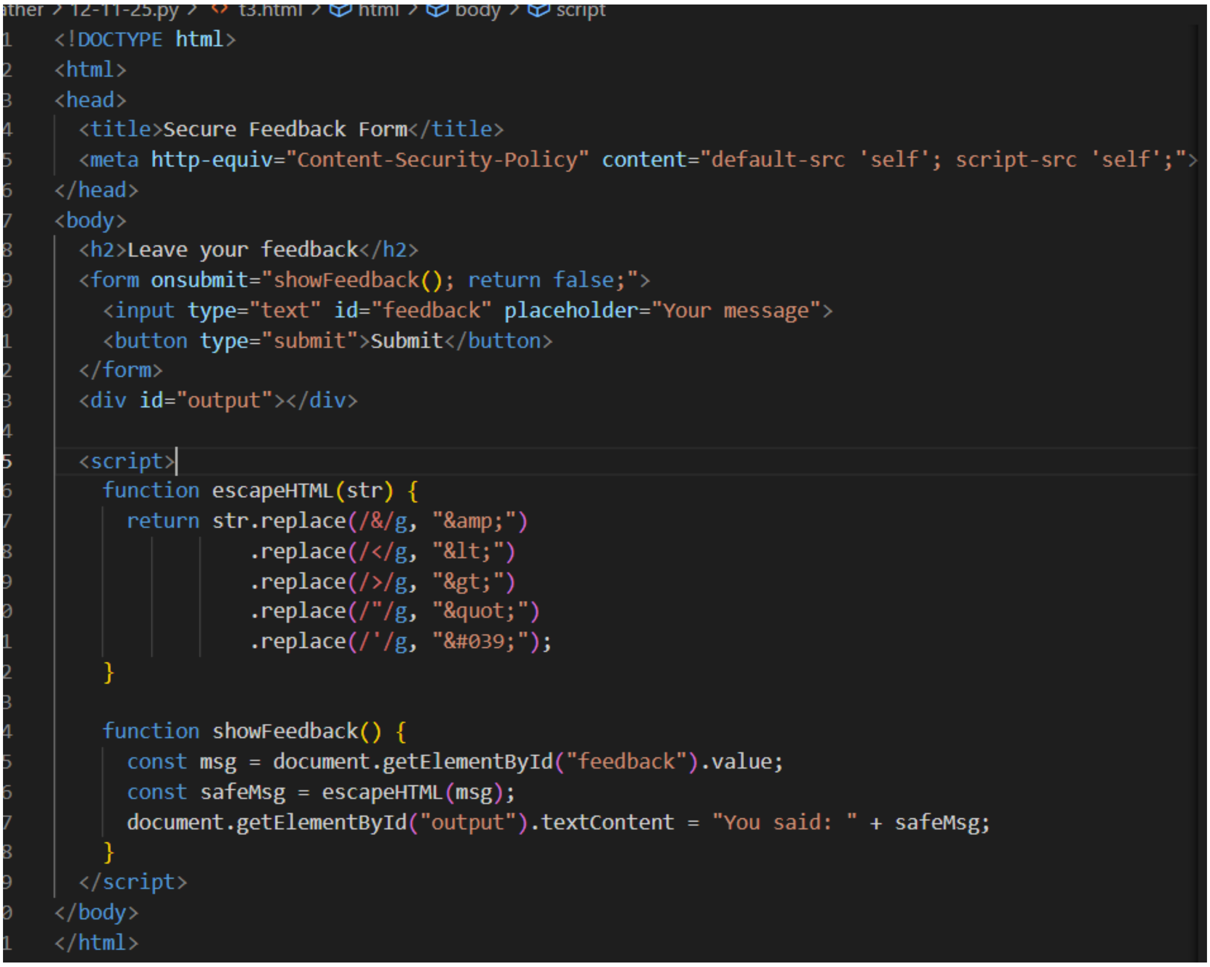
No error handling or protection against malicious input.

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

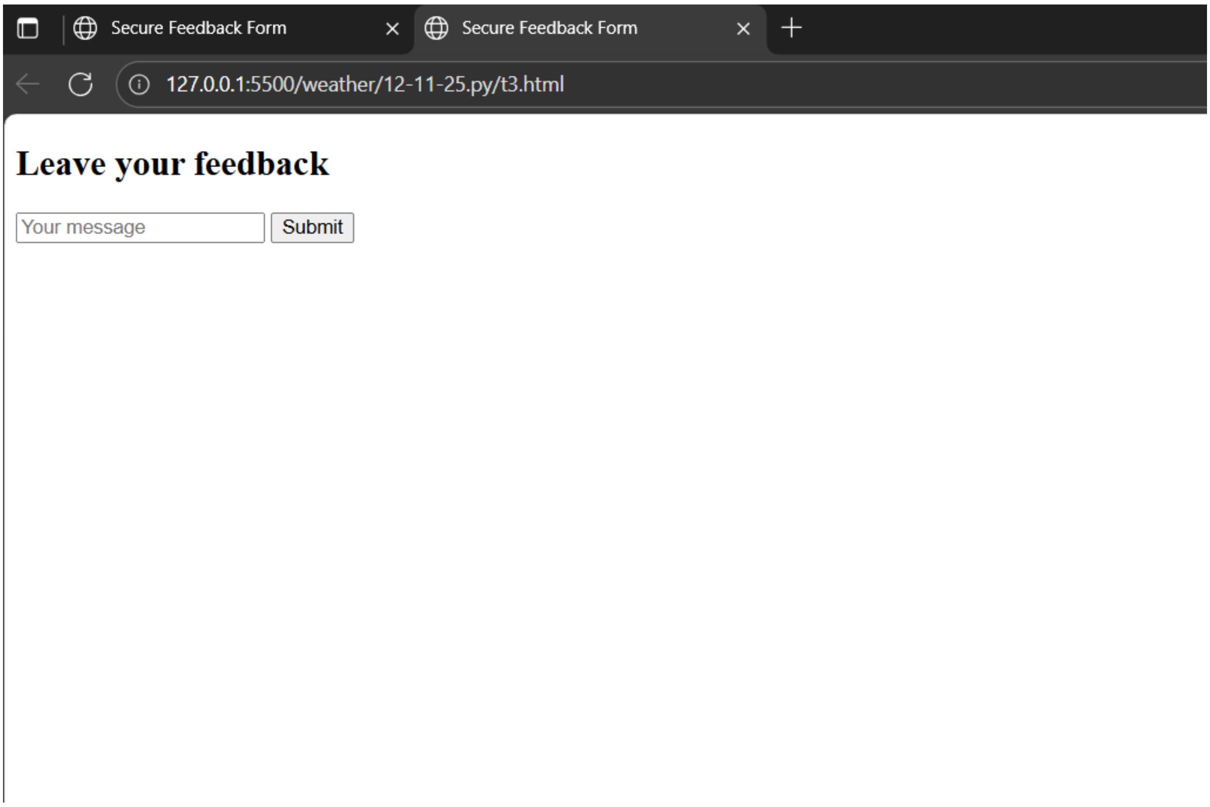
Prompt: Generate a simple HTML feedback form that uses JavaScript to display the submitted message on the page.

Code:



```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Feedback Form</title>
  <meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self';">
</head>
<body>
  <h2>Leave your feedback</h2>
  <form onsubmit="showFeedback(); return false;">
    <input type="text" id="feedback" placeholder="Your message">
    <button type="submit">Submit</button>
  </form>
  <div id="output"></div>

  <script>
    function escapeHTML(str) {
      return str.replace(/&/g, "&amp;")
                .replace(/</g, "&lt;")
                .replace(/>/g, "&gt;")
                .replace(/"/g, "&quot;")
                .replace(/'/g, "&#039;");
    }

    function showFeedback() {
      const msg = document.getElementById("feedback").value;
      const safeMsg = escapeHTML(msg);
      document.getElementById("output").textContent = "You said: " + safeMsg;
    }
  </script>
</body>
</html>
```

Output:



Leave your feedback

Observation:

The AI-generated HTML form uses innerHTML to display user input directly.

This allows attackers to inject scripts like <script>alert('XSS')</script>.

No input sanitization or escaping is applied.

No Content Security Policy (CSP) is set to restrict script execution.

The form is vulnerable to **reflected XSS attacks**.

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:

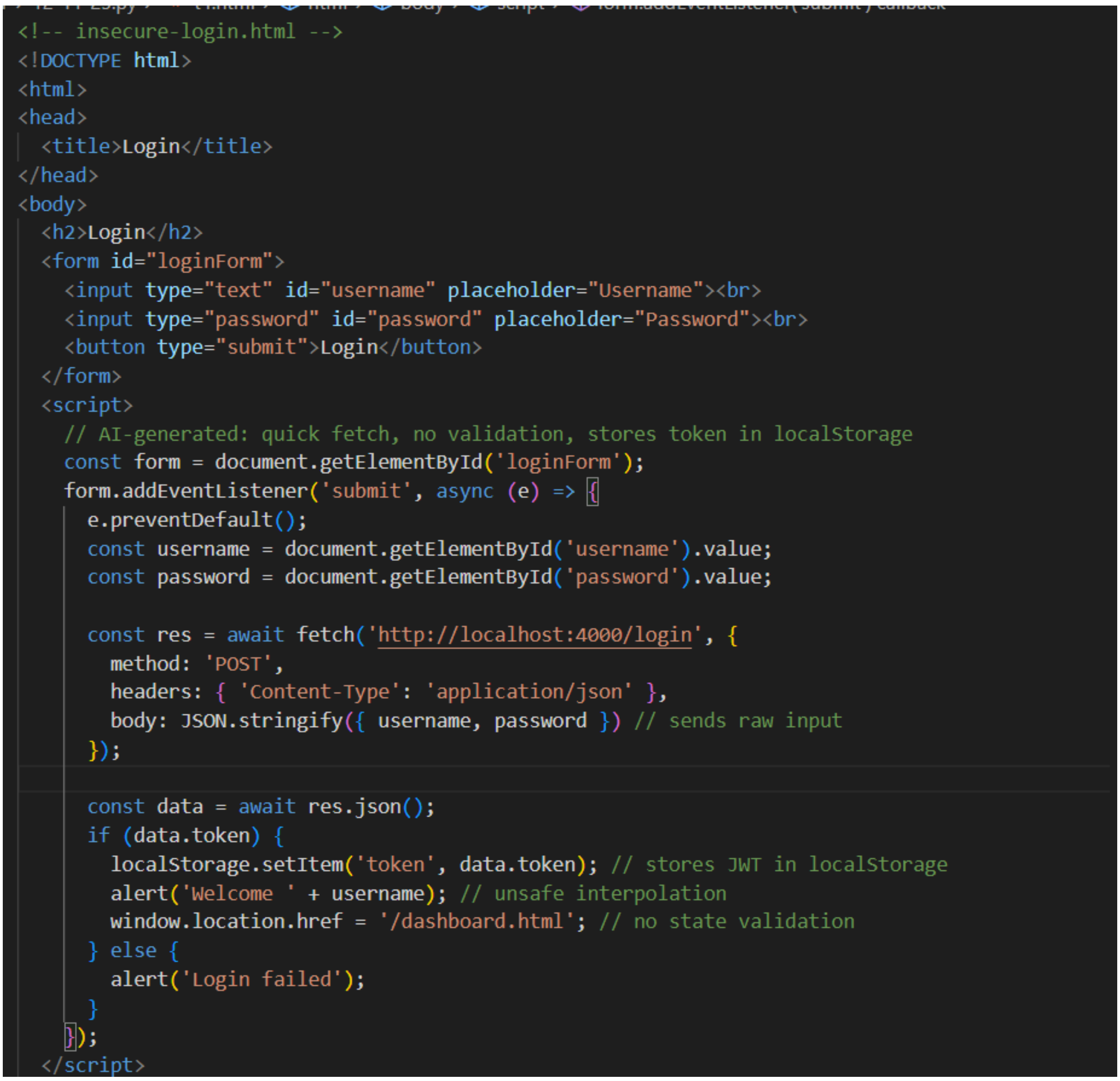
• Perform a security audit to detect possible vulnerabilities.

• Prompt AI to suggest secure coding practices to fix issues.

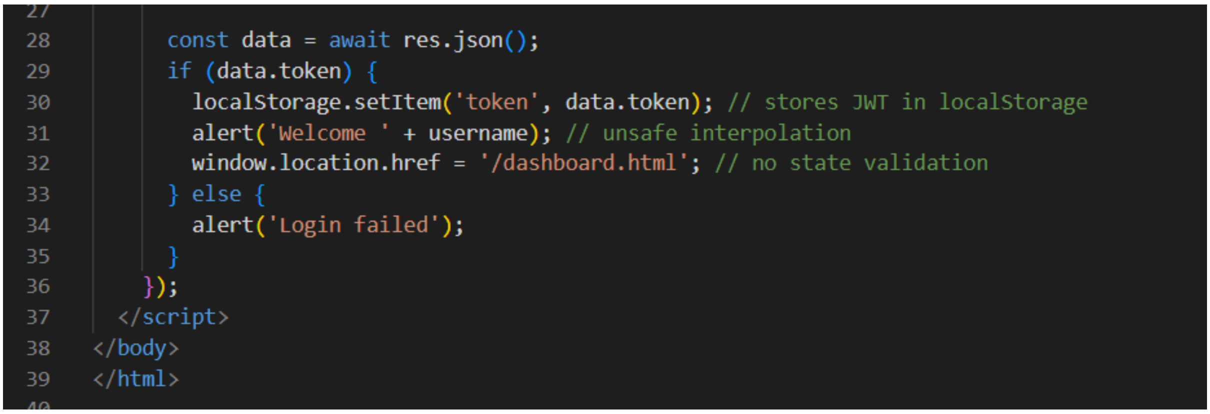
• Compare insecure vs secure versions side by side.

Prompt: Generate a smple login form in HTML/JavaScript with a Node.js Express backend. The backend should check credentials from an in-memory array and return a JWT token. Keep it minimal without adding extra security features.

Code:

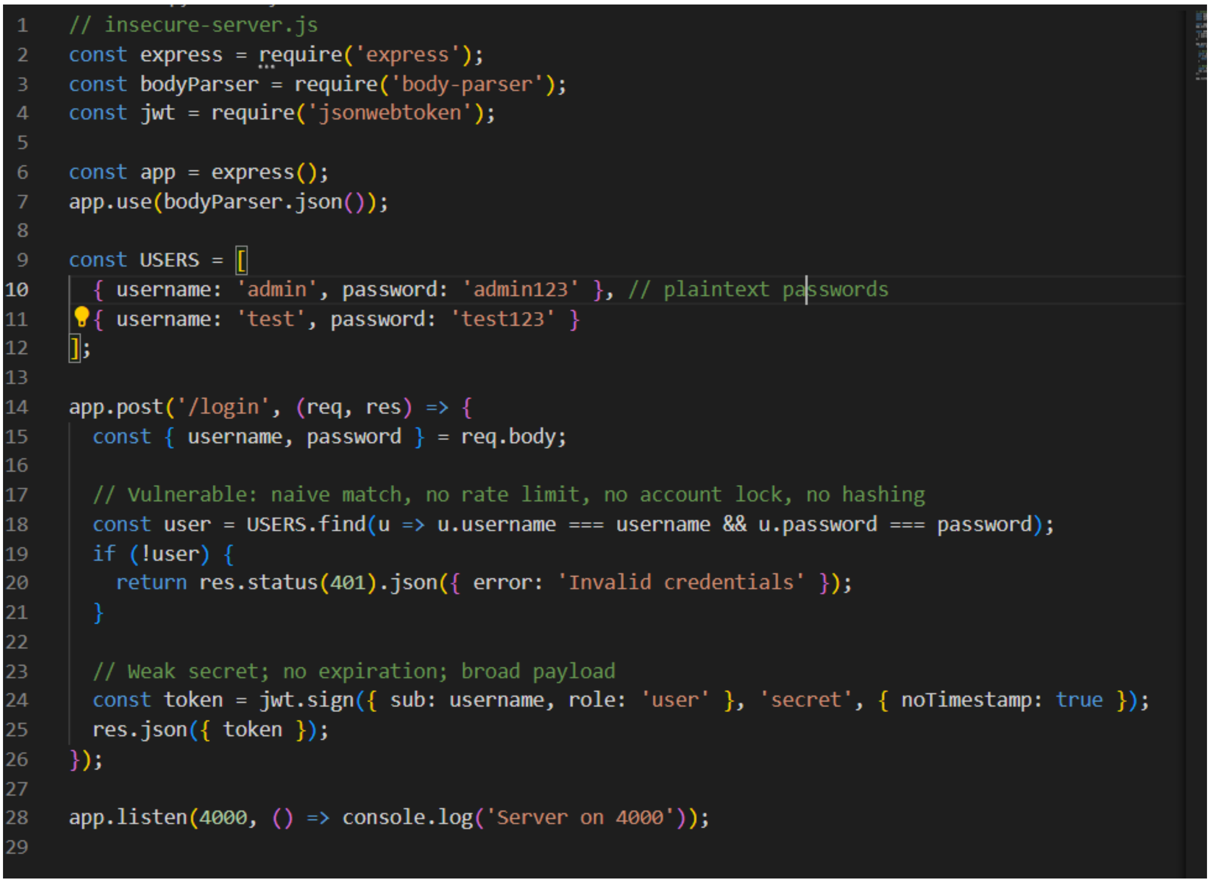


```html
<!-- insecure-login.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
</head>
<body>
  <h2>Login</h2>
  <form id="loginForm">
    <input type="text" id="username" placeholder="Username"><br>
    <input type="password" id="password" placeholder="Password"><br>
    <button type="submit">Login</button>
  </form>
  <script>
    // AI-generated: quick fetch, no validation, stores token in localStorage
    const form = document.getElementById('loginForm');
    form.addEventListener('submit', async (e) => {
      e.preventDefault();
      const username = document.getElementById('username').value;
      const password = document.getElementById('password').value;

      const res = await fetch('http://localhost:4000/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password }) // sends raw input
      });

      const data = await res.json();
      if (data.token) {
        localStorage.setItem('token', data.token); // stores JWT in localStorage
        alert('Welcome ' + username); // unsafe interpolation
        window.location.href = '/dashboard.html'; // no state validation
      } else {
        alert('Login failed');
      }
    });
  </script>
```



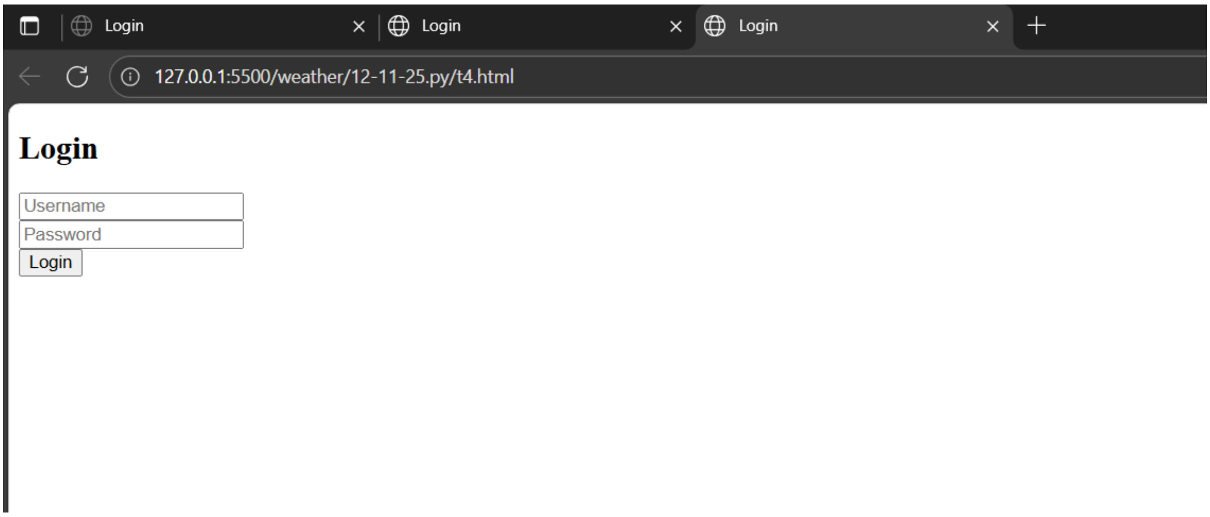
```html
      const data = await res.json();
      if (data.token) {
        localStorage.setItem('token', data.token); // stores JWT in localStorage
        alert('Welcome ' + username); // unsafe interpolation
        window.location.href = '/dashboard.html'; // no state validation
      } else {
        alert('Login failed');
      }
    });
  </script>
</body>
</html>
```

Java script:



```javascript
// insecure-server.js
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');

const app = express();
app.use(bodyParser.json());

const USERS = [
  { username: 'admin', password: 'admin123' }, // plaintext passwords
  { username: 'test', password: 'test123' }
];

app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // Vulnerable: naive match, no rate limit, no account lock, no hashing
  const user = USERS.find(u => u.username === username && u.password === password);
  if (!user) {
    return res.status(401).json({ error: 'Invalid credentials' });
  }

  // Weak secret; no expiration; broad payload
  const token = jwt.sign({ sub: username, role: 'user' }, 'secret', { noTimestamp: true });
  res.json({ token });
});

app.listen(4000, () => console.log('Server on 4000'));
```

Output:



Login
Login
Login
127.0.0.1:5500/weather/12-11-25.py/t4.html
Login
Username
Password
Login