Project Definition Review:

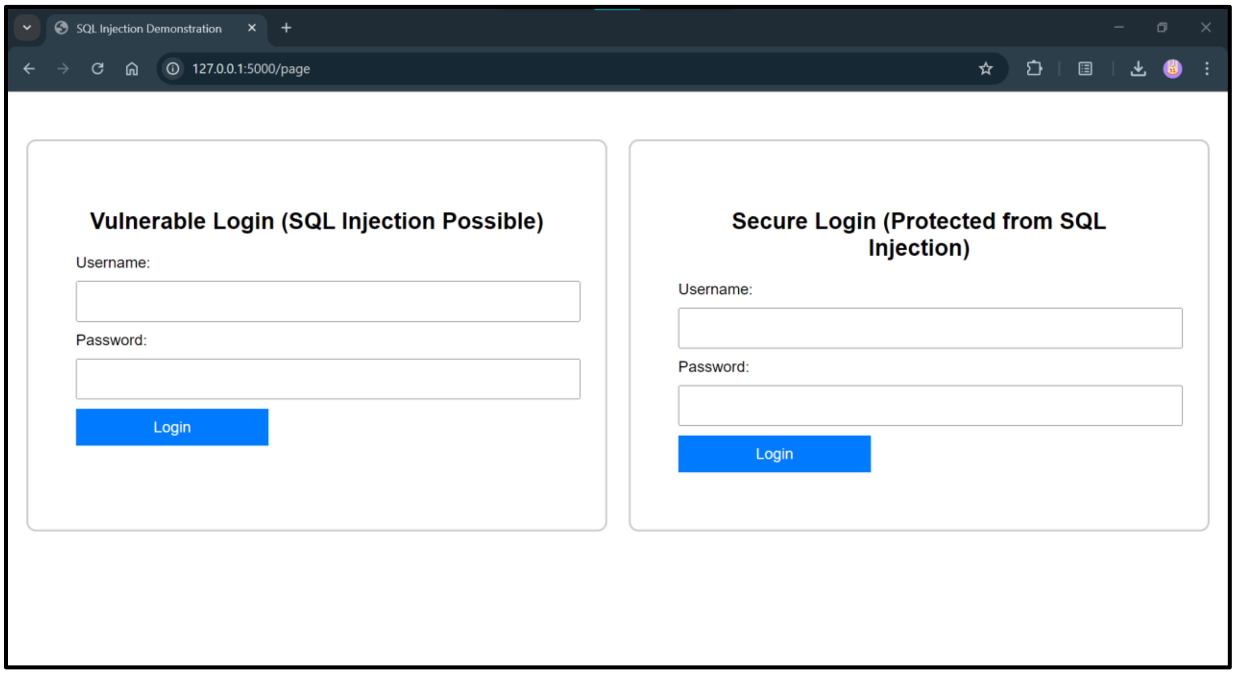
The project requires creating a **vulnerable database-driven web application** that demonstrates an **SQL injection attack** and then applying the necessary **prevention techniques** to thwart the attack. The goal is to **combine both the vulnerable and secure versions** of the application so that you can demonstrate both scenarios side by side.

Methodology:

The project is designed to demonstrate both a **vulnerable web application** susceptible to SQL injection and a **secure version** that prevents such attacks using best practices. Below are the steps taken to achieve this:

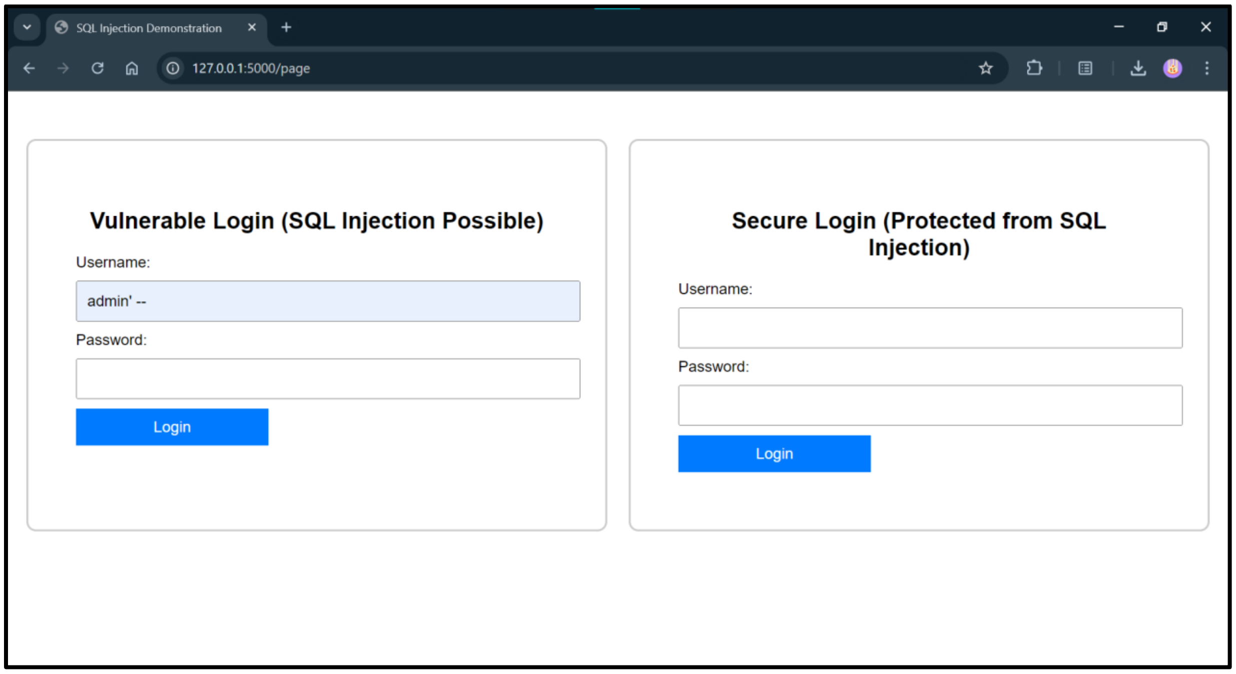
1. Setting Up the Web Application:

- A **Flask** web application is developed to handle user authentication.
- **SQLite** is used as the database to store user credentials.
- The web interface consists of two login forms: one that is intentionally vulnerable to SQL injection and another that is secured against it.


SQL Injection Demonstration
127.0.0.1:5000/page
Vulnerable Login (SQL Injection Possible)
Username:
Password:
Login
Secure Login (Protected from SQL Injection)
Username:
Password:
Login

2. Vulnerable Login Demonstration:

- The first form (Vulnerable Login) is designed to show how SQL injection can be exploited.
- The user input is directly concatenated into the SQL query string without parameterization, allowing attackers to modify the query through malicious input.
- The input `admin' --` in the username field effectively demonstrates bypassing the authentication mechanism, simulating an SQL injection attack.



Vulnerable Login (SQL Injection Possible)

Username: admin' --

Password:

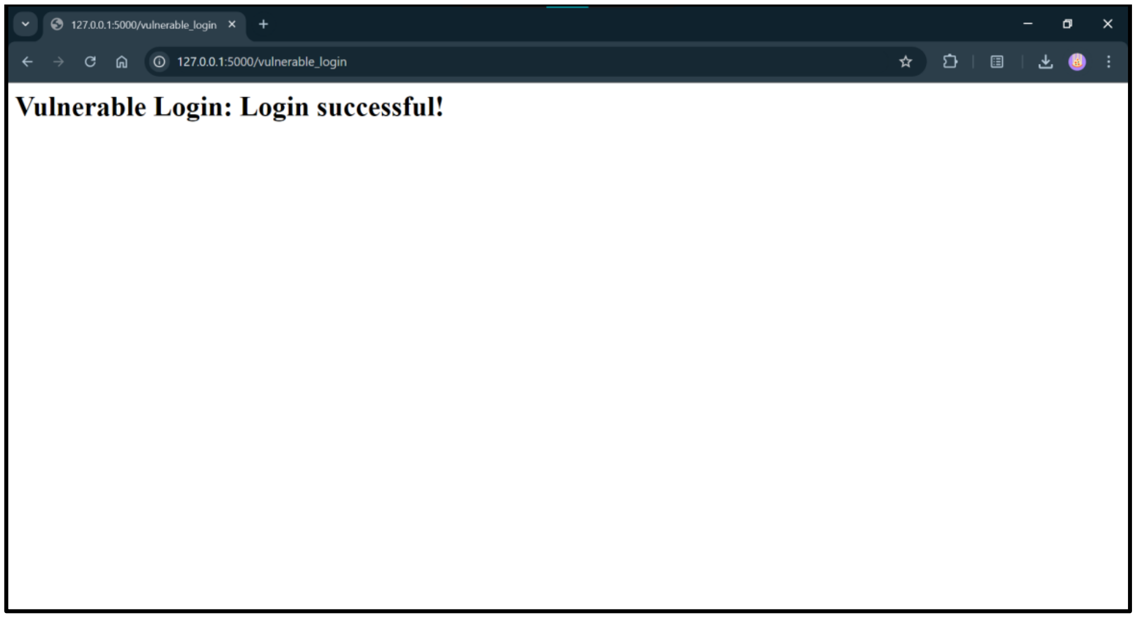
Login

Secure Login (Protected from SQL Injection)

Username:

Password:

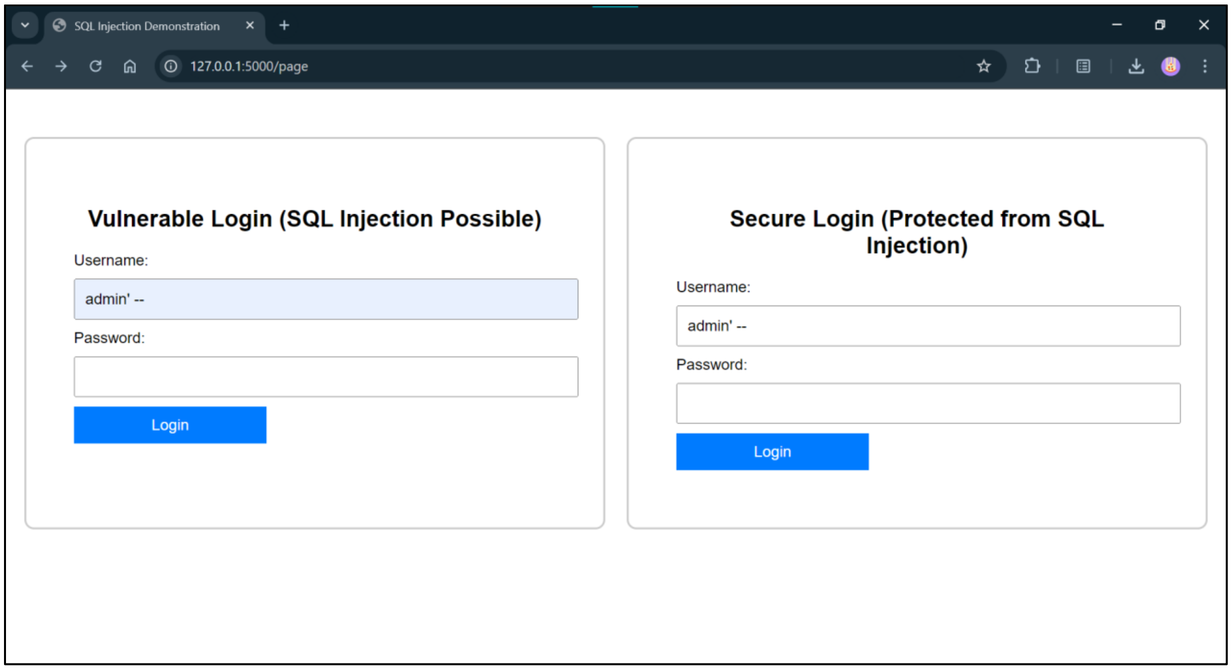
Login


127.0.0.1:5000/vulnerable_login
127.0.0.1:5000/vulnerable_login

Vulnerable Login: Login successful!

3. Secured Login Implementation:

- The second form (Secure Login) demonstrates how to properly defend against SQL injection using **parameterized queries** (prepared statements).
- This version separates the SQL logic from the user input, ensuring that user-supplied data is treated strictly as input and not executable code.
- The same attack attempt on this form returns an "Invalid credentials!" message, showing that the SQL injection is thwarted.



Vulnerable Login (SQL Injection Possible)

Username: admin' --

Password:

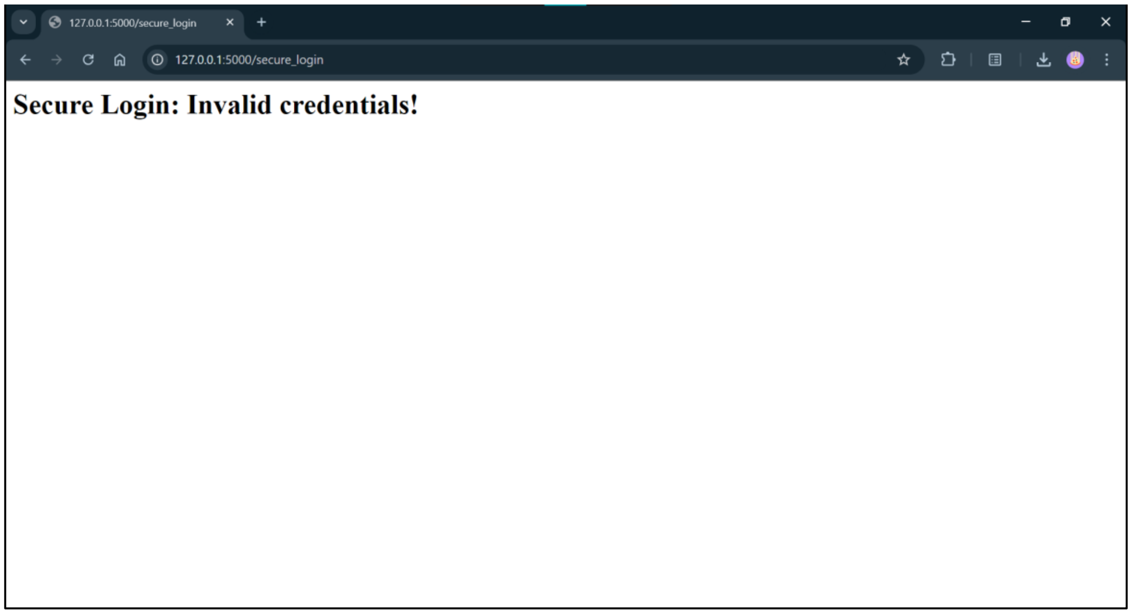
Login

Secure Login (Protected from SQL Injection)

Username: admin' --

Password:

Login



Secure Login: Invalid credentials!

Key Components of the Project:

- **Flask**: The web framework used to handle routes and display the forms.
- **SQLite**: The database used to store user credentials (demonstrating how SQL queries are executed).
- **Forms**: One vulnerable form and one secure form to demonstrate the contrast between SQL injection and its prevention.
- **Parameterized Queries**: Used in the secure login form to prevent SQL injection.

In Summary:

This project successfully demonstrates the concept of SQL injection and prevention within a combined web application. You can show both the **vulnerability** and the **protection** side by side without changing tabs, which aligns perfectly with your project definition.