**Objective:**

* **Stage 1** focuses on creating the user interface (frontend) and implementing SQL injection detection mechanisms on a website using Flask, along with session management for secure access control.

**Sub-Stages:**

**1. Frontend (HTML Template) and Flask Setup:**

* We created the basic structure of the website with user interfaces for:
  + **Search**: Allows searching users by username.
  + **Signup**: Allows users to create new accounts.
  + **Login**: Allows users to authenticate and log in.
  + **Logout**: Allows users to log out of the system.

**2. SQL Injection Detection:**

* **Input Validation**:
  + Used regular expressions to validate usernames, email addresses, and passwords to avoid malicious inputs.
* **Suspicious Input Detection**:
  + Implemented a mechanism to detect SQL injection attempts by checking for common malicious SQL patterns (e.g., --, ;, ', etc.) in user inputs.
  + Used logging to record suspicious actions such as invalid username inputs, and login/signup attempts that might indicate SQL injection attempts.

**3. Database Setup and User Management:**

* **SQLite Database**:
  + Created a sample SQLite database to store user information (username, email, password).
  + Implemented user registration, login, and search functionalities, all with SQL injection prevention mechanisms.
  + Passwords were securely hashed using SHA-256 before being stored in the database.

**4. Session Management:**

* **Secure Sessions**:
  + Used Flask’s session management to securely handle user sessions.
  + Implemented session timeouts to log out users automatically after a certain period of inactivity.
  + Session-based access control was added to ensure only authenticated users can access specific sections.

**5. Logging:**

* **Logging Suspicious Activities**:
  + Used Python’s logging module to log suspicious activities, such as invalid inputs or login attempts, for future analysis and debugging.
  + Logging was used to track potentially malicious activities like SQL injection attempts, unsuccessful login attempts, etc.

**Final Structure:**

1. **Flask Application (app.py)**:
   * Routes to handle search, signup, login, and logout actions.
   * Integrated SQL injection detection in each of these routes.
   * Session management for user authentication.
2. **Frontend (index.html)**:
   * Created using HTML and Bootstrap for styling, with forms for user input in search, signup, login, and logout sections.
3. **Database (database.db)**:
   * Created a sample users table to store registered user data.
   * Data validation was implemented to avoid SQL injections and other suspicious inputs.
4. **Requirements (requirements.txt)**:
   * Included necessary packages like Flask, SQLite, etc., to run the application smoothly.

**Testing:**

* The implemented system was tested for:
  + **SQL Injection Detection**: Ensured that suspicious inputs such as ', --, ;, and others were properly flagged and logged.
  + **Frontend Functionality**: Ensured the forms for signup, login, and search worked correctly.
  + **Session Timeout and Access Control**: Verified that sessions timed out after the set period and that users could only access authorized parts of the site when logged in.