## Choosing a Language and Application

**Language:** Python

**Application:** A simple web application that allows users to sign up, log in, and view their profile information.

from flask import Flask, render\_template, request, redirect, url\_for

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/signup', methods=['GET', 'POST'])

def signup():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

# Store user data in a database

return redirect(url\_for('index'))

return render\_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

# Verify user credentials and log in

return redirect(url\_for('profile'))

return render\_template('login.html')

@app.route('/profile')

def profile():

# Retrieve user profile information from the database

return render\_template('profile.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Potential Vulnerabilities and Recommendations:**

**Insecure Direct Object References (IDOR):**

* 1. **Vulnerability:** If user-controlled input is directly used to access resources without proper validation, it could lead to unauthorized access.
  2. **Recommendation:** Use parameterized queries or prepared statements to prevent SQL injection.
  3. **Example:** Instead of SELECT \* FROM users WHERE id = request.args.get('id'), use parameterized queries.

**Cross-Site Scripting (XSS):**

* 1. **Vulnerability:** If user-controlled input is not properly sanitized, it could be injected into the web page, leading to malicious code execution.
  2. **Recommendation:** Use output encoding or templating engines to sanitize user-provided content.
  3. **Example:** Use template filters or escape() functions to escape user input before rendering it in HTML.

**SQL Injection:**

* 1. **Vulnerability:** If user-controlled input is directly used in SQL queries without proper validation, it could lead to unauthorized access or data manipulation.
  2. **Recommendation:** Use parameterized queries or prepared statements to prevent SQL injection.
  3. **Example:** Instead of SELECT \* FROM users WHERE username = '%s' % username, use parameterized queries.

**Password Storage:**

* 1. **Vulnerability:** Storing passwords in plain text is a security risk.
  2. **Recommendation:** Use strong hashing algorithms (e.g., bcrypt, argon2) to store passwords securely.
  3. **Example:** Use a password hashing library like bcrypt to hash passwords before storing them.

**Session Management:**

* 1. **Vulnerability:** Improper session management can lead to session hijacking.
  2. **Recommendation:** Use secure session cookies, implement HTTP Strict Transport Security (HSTS), and regularly rotate session IDs.