# **Week 4 - Lab Assignment**

**Name: Aryan Bhagat**

**Net ID: sl5310**

**Name: Moksha Shah**

**Net ID: bp4199**

**Course: CS-671 - Cyber Security**

**PROBLEM STATEMENT:**

1. Write any Python or Java programming web or cloud software application of your choice (similar to MyCSUEB). The application should include database connectivity and web application procedures. Save the document as Wk4Lab.doc. (5 points)
2. Explain the function of the web application. Append this explanation to the document saved as Wk4Lab.doc. (5 points)
3. Create a flowchart for your project's web programming application. Append this flowchart to the document saved as Wk4Lab.doc. (10 points)
4. Identify the possible cybersecurity threats, risks, and vulnerabilities associated with this web application. Append this information to the document saved as Wk4Lab.doc. (10 points)
5. Design, develop, and test the web application online or on a local host. Save all (20 points)

**Flow:**

Start

|

Open Browser → Navigate to Homepage (Login Page)

|

Enter Net ID and email → Click 'Login'

|

Validate Credentials (Database Query)

|

+ → [Valid] → Show Dashboard (Display Grades)

|

[Invalid] → Display "Login Failed" Message

|

End

**app.py:**

|  |
| --- |
| **from flask import Flask, render\_template, request, redirect, url\_for, session import random import smtplib import os  app = Flask(\_\_name\_\_) app.secret\_key = 'supersecretkey'  def send\_otp(email, otp):  try:  sender\_email = "aryanbhagat5702@gmail.com"   sender\_password = "logi cwoj uywq lrvj"     server = smtplib.SMTP('smtp.gmail.com', 587)  server.starttls()  server.login(sender\_email, sender\_password)   subject = "Your OTP Code"  body = f"Your OTP code is {otp}"  message = f"Subject: {subject}\n\n{body}"   server.sendmail(sender\_email, email, message)  server.quit()  except Exception as e:  print(f"Failed to send email: {e}")   @app.route('/') def login\_page():  return render\_template('login.html')  @app.route('/send\_otp', methods=['POST']) def send\_otp\_route():  netid = request.form['netid']  email = request.form['email']    otp = random.randint(100000, 999999)  session['otp'] = otp  session['email'] = email   send\_otp(email, otp)    return redirect(url\_for('verify\_otp')) @app.route('/verify\_otp', methods=['GET', 'POST']) def verify\_otp():  if request.method == 'POST':  user\_otp = request.form['otp']  if int(user\_otp) == session['otp']:  return "OTP Verified! Welcome to your dashboard."  else:  return "Invalid OTP. Please try again."   return '''  <h2>Enter the OTP sent to your email</h2>  <form method="POST">  <label for="otp">OTP:</label>  <input type="text" name="otp" required>  <button type="submit">Verify OTP</button>  </form>  '''  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True)** |

**login.html:**

|  |
| --- |
| <!DOCTYPE html> <**html** lang="en">  <**head**>  <**meta** charset="UTF-8">  <**meta** name="viewport" content="width=device-width, initial-scale=1.0">  <**title**>CSUEB Login</**title**>  <**style**>  body {  font-family: Arial, sans-serif;  background-color: #000000;  color: #ffffff;  }   .login-container {  width: 300px;  margin: 100px auto;  background-color: #1c1c1c;  padding: 20px;  border-radius: 10px;  box-shadow: 0 0 10px rgba(255, 0, 0, 0.5);  text-align: center;  }   .login-container h2 {  margin-bottom: 20px;  }   .login-container img {  width: 150px;  margin-bottom: 20px;  }   .login-container input[type="text"],  input[type="email"] {  width: 100%;  padding: 10px;  margin: 10px 0;  border-radius: 5px;  border: 1px solid #555;  background-color: #333;  color: #fff;  }   .login-container button {  width: 100%;  padding: 10px;  background-color: #ff0000;  color: white;  border: none;  border-radius: 5px;  cursor: pointer;  }   .login-container button: hover {  background-color: #cc0000;  }  </**style**> </**head**>  <**body**>  <**div** class="login-container">  <**img** src="https://www.csueastbay.edu/\_global/images/header/new-csueb-logo/csueb-logo.png" alt="CSUEB Logo">  <**h2**>CSUEB Login</**h2**>  <**form** action="/send\_otp" method="post">  <**label** for="netid">NetID</**label**>  <**input** type="text" id="netid" name="netid" placeholder="Enter your NetID" required>  <**label** for="email">Email</**label**>  <**input** type="email" id="email" name="email" placeholder="Enter your email" required>  <**button** type="submit">Send OTP</**button**>  </**form**>  </**div**> </**body**> </**html**> |

* **Application Function:**

This web application is a simple student portal similar to MyCSUEB, where students can log in using their credentials to access their grades and schedules.

1. Login: The user enters their email and Net ID. If the credentials match the database records, the user is redirected to the dashboard. Otherwise, they receive a "Login Failed" message.
2. Dashboard: Once logged in, students can view their details, such as grades and other information stored in the database.

* **Cybersecurity threats, risks, and vulnerabilities:**

**1. Session Hijacking:**

Session hijacking occurs when an attacker gains unauthorized access to a user's session by stealing or manipulating session cookies. This can allow attackers to impersonate legitimate users and gain access to their accounts.

**Resolution:**

* Use HttpOnly and Secure flags for cookies.
* Implement session expiration and idle timeouts.
* Regenerate session IDs on important events.

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

Cross-Site Scripting (XSS) attacks occur when an attacker injects malicious scripts into web pages viewed by other users. This can lead to data theft, session hijacking, or other malicious activities.

**Resolution:**

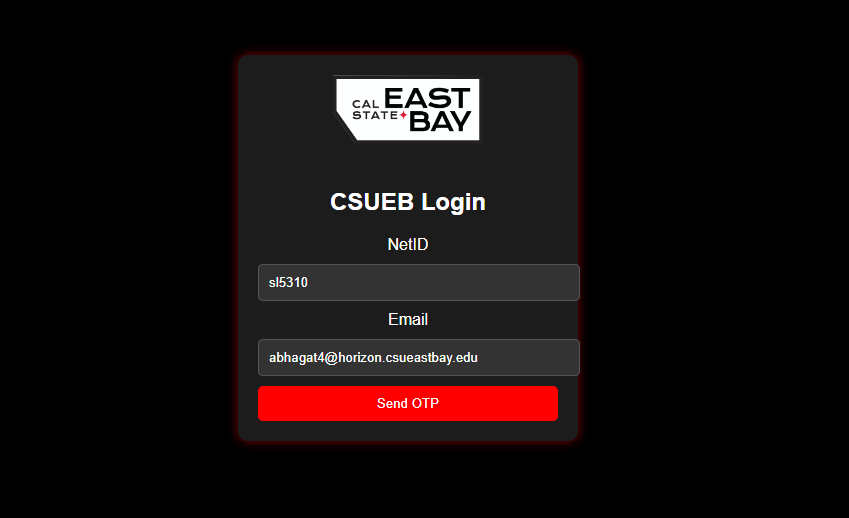
* Sanitize and validate user inputs.
* Use Flask’s Jinja2 templating with automatic escaping.
* Implement a Content Security Policy (CSP).

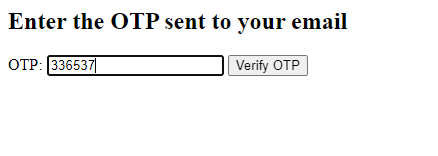
**3. Insecure Password Storage:**

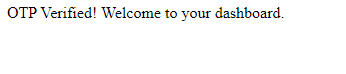
Insecure password storage involves using weak or outdated hashing algorithms to store user passwords. This can make it easier for attackers to crack passwords and gain unauthorized access.

**Resolution:**

* Use strong hashing algorithms like bcrypt.
* Use unique salts for each password.
* Regularly update security practices.
* **Application view:**

****

****

****