



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FCFM

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Desarrollo de Software Seguro

Actividad práctica (asíncrona): Almacenamiento de contraseñas

Estudiante: Osmar Abelardo Bustos Vázquez

Carrera: Lic. en Seguridad de Tecnologías de Información

Matrícula: 1912361

Grupo: 064

Docente: M.C. Romeo Alfonso Sánchez López

12 de noviembre 2023

AGOSTO-DICIEMBRE-2023

Almacenamiento de contraseñas

Enlace de página en Github: <https://github.com/osm4r/DSS-Almacenamiento-de-contrasenas>

Aprendizajes:

- Uso de la librería argon2-cffi
- Reforzamiento de conocimiento de SQLite
- Implementación de sal
- Implementación de pimienta
- Manera de realizar un login más seguro sin la necesidad de administrar contraseñas
- Aplicación de verificaciones de seguridad adicionales
- Importancia del almacenamiento seguro de contraseñas

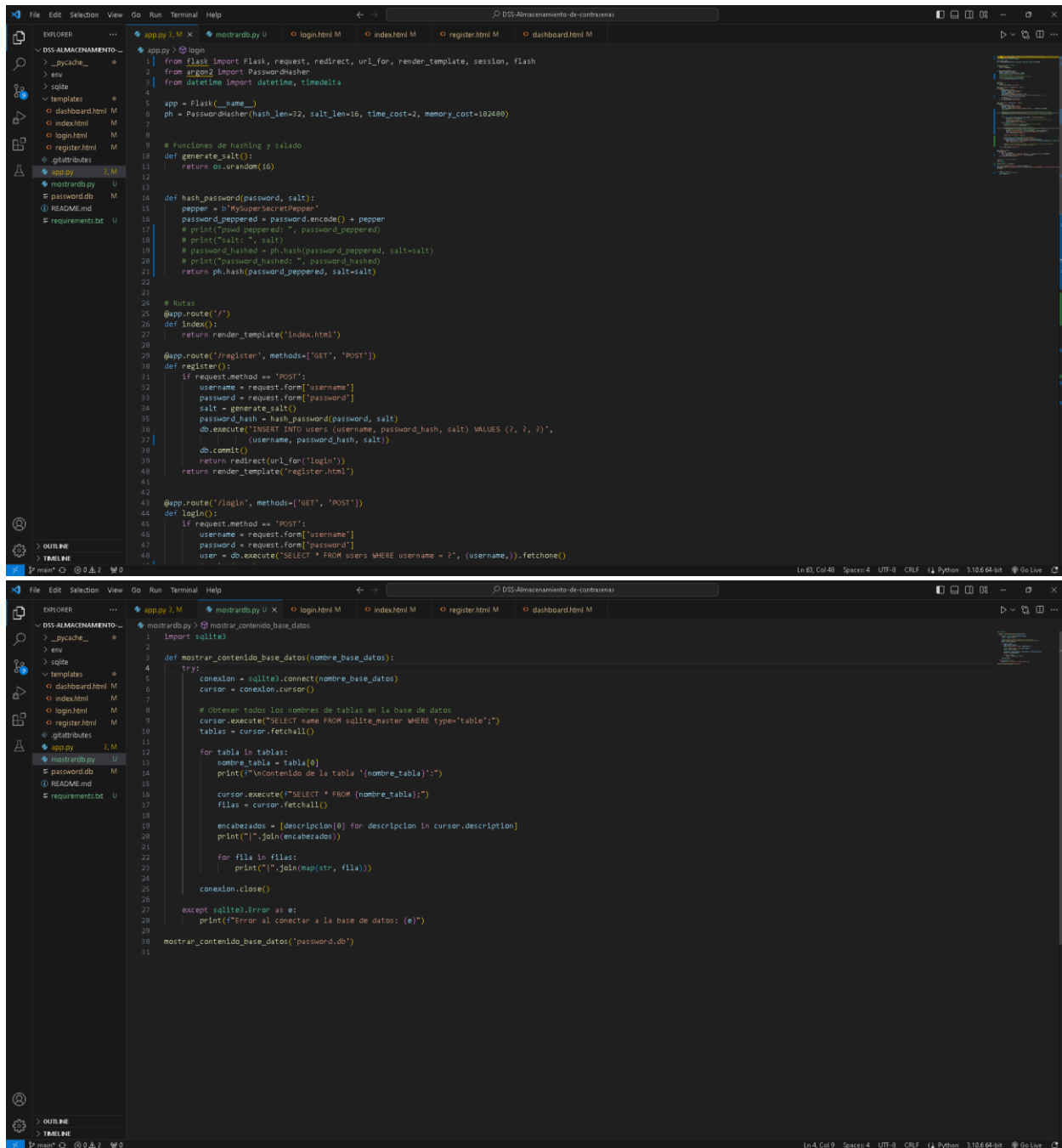
Dificultades:

Tuve varias dificultades ya que en el código había unos pequeños errores, los cuales fueron:

- Colocar “app.secret_key = os.urandom(16)” fuera del if para iniciar el programa ya que no lo tomaba en cuenta.
- Utilizar de esta manera username=user[0] en vez de username=user[“id”] ya que es una user es una tupla, no un diccionario como se muestra en el código.

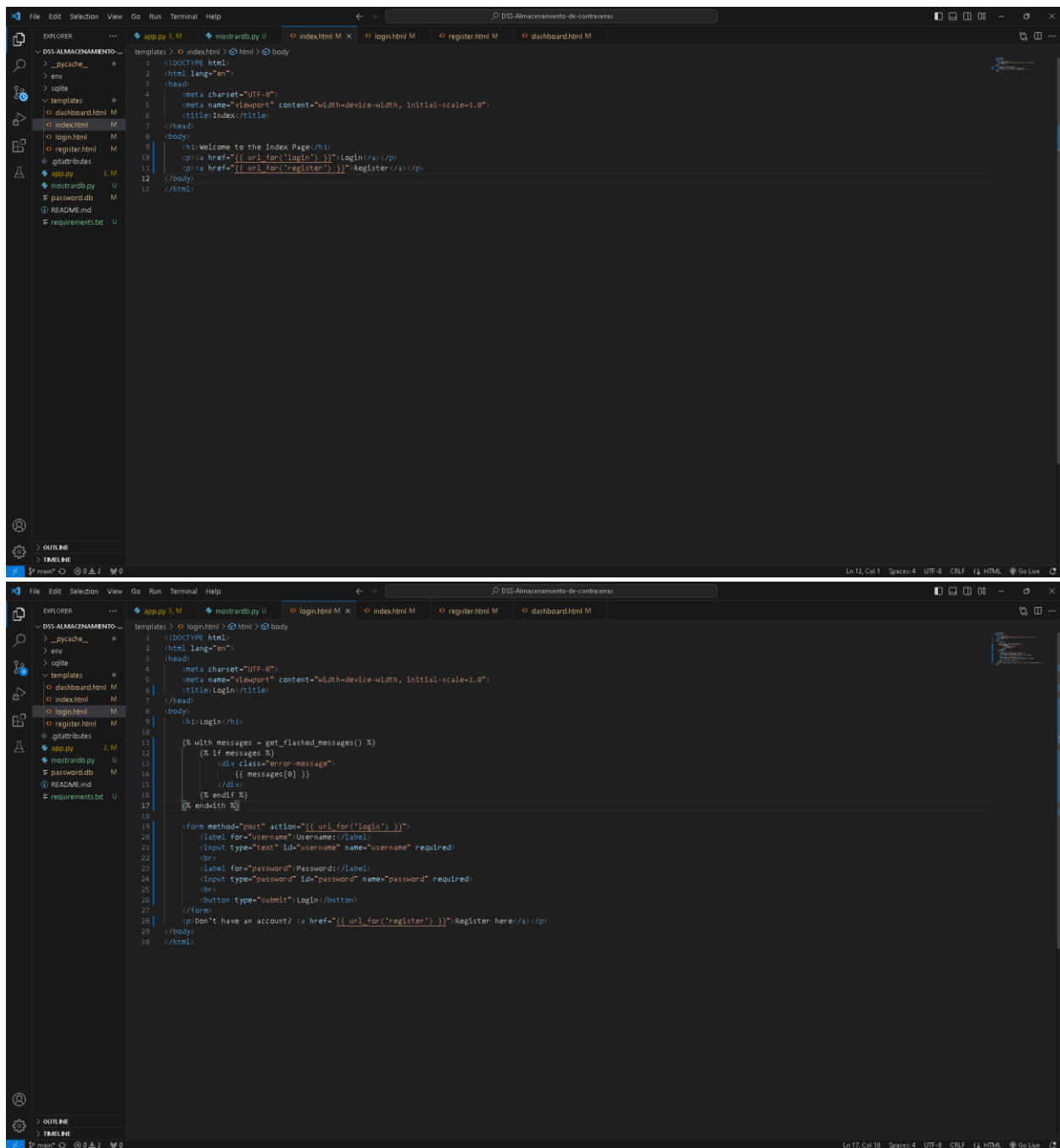
También tuve ciertas dificultades generales para poder realizar la actividad, sin embargo, fueron resueltas mediante la investigación. Entre ellas estaba la no inicialización de ciertas funciones.

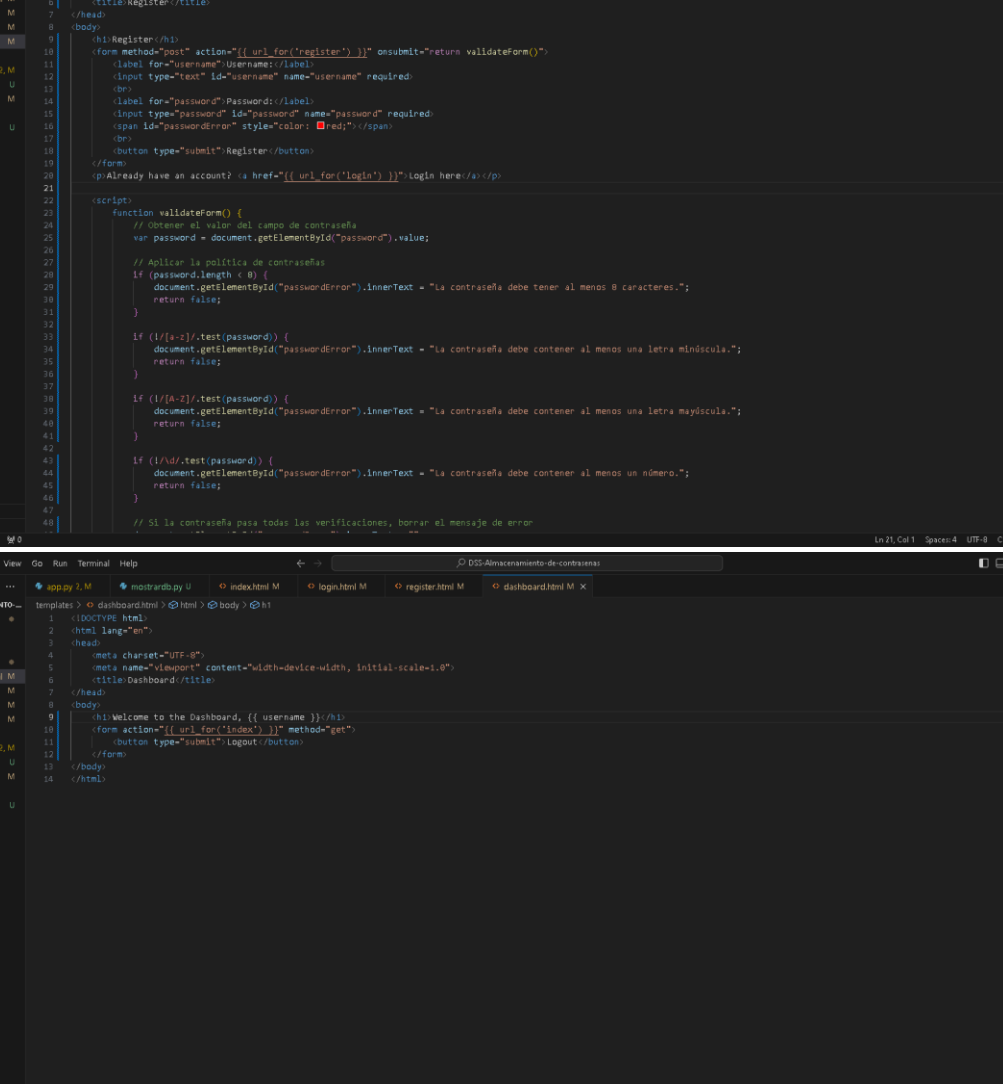
Pruebas y capturas de pantalla:



```
1 from flask import Flask, request, redirect, url_for, render_template, session, flash
2 from argon2 import PasswordHasher
3 from datetime import datetime, timedelta
4
5 app = Flask(__name__)
6 ph = PasswordHasher(hash_len=32, salt_len=16, time_cost=2, memory_cost=102400)
7
8 # Funciones de hashing y salado
9
10 def generate_salt():
11     return os.urandom(16)
12
13
14 def hash_password(password, salt):
15     pepper = b'MySuperSecretPepper'
16     password_peppered = password.encode() + pepper
17     # print("Password peppered: ", password_peppered)
18     # print("Salt: ", salt)
19     password_hashed = ph.hash(password_peppered, salt=salt)
20     # print("Password hashed: ", password_hashed)
21     return ph.hash(password_peppered, salt=salt)
22
23
24 # Rutas
25 @app.route('/')
26 def index():
27     return render_template('index.html')
28
29 @app.route('/register', methods=['GET', 'POST'])
30 def register():
31     if request.method == 'POST':
32         username = request.form['username']
33         password = request.form['password']
34         salt = generate_salt()
35         password_hash = hash_password(password, salt)
36         db.execute("INSERT INTO users (username, password_hash, salt) VALUES (?, ?, ?)",
37                 (username, password_hash, salt))
38         db.commit()
39         return redirect(url_for('login'))
40     return render_template('register.html')
41
42
43 @app.route('/login', methods=['GET', 'POST'])
44 def login():
45     if request.method == 'POST':
46         username = request.form['username']
47         password = request.form['password']
48         user = db.execute("SELECT * FROM users WHERE username = ?", (username,)).fetchone()
```

```
1 mostrar_contenido_base_datos
2 import sqlite3
3
4 def mostrar_contenido_base_datos(nombre_base_datos):
5     try:
6         conexion = sqlite3.connect(nombre_base_datos)
7         cursor = conexion.cursor()
8
9         # Obtener todos los nombres de tablas en la base de datos
10        cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
11        tablas = cursor.fetchall()
12
13        for tabla in tablas:
14            nombre_tabla = tabla[0]
15            print(f"Contenido de la tabla '{nombre_tabla}':")
16            cursor.execute(f"SELECT * FROM {nombre_tabla};")
17            filas = cursor.fetchall()
18
19            encabezados = [description[0] for description in cursor.description]
20            print("\n".join(encabezados))
21
22            for fila in filas:
23                print("\n".join(map(str, fila)))
24
25        conexion.close()
26
27    except sqlite3.Error as e:
28        print(f"Error al conectar a la base de datos: {e}")
29
30 mostrar_contenido_base_datos("password.db")
31
```



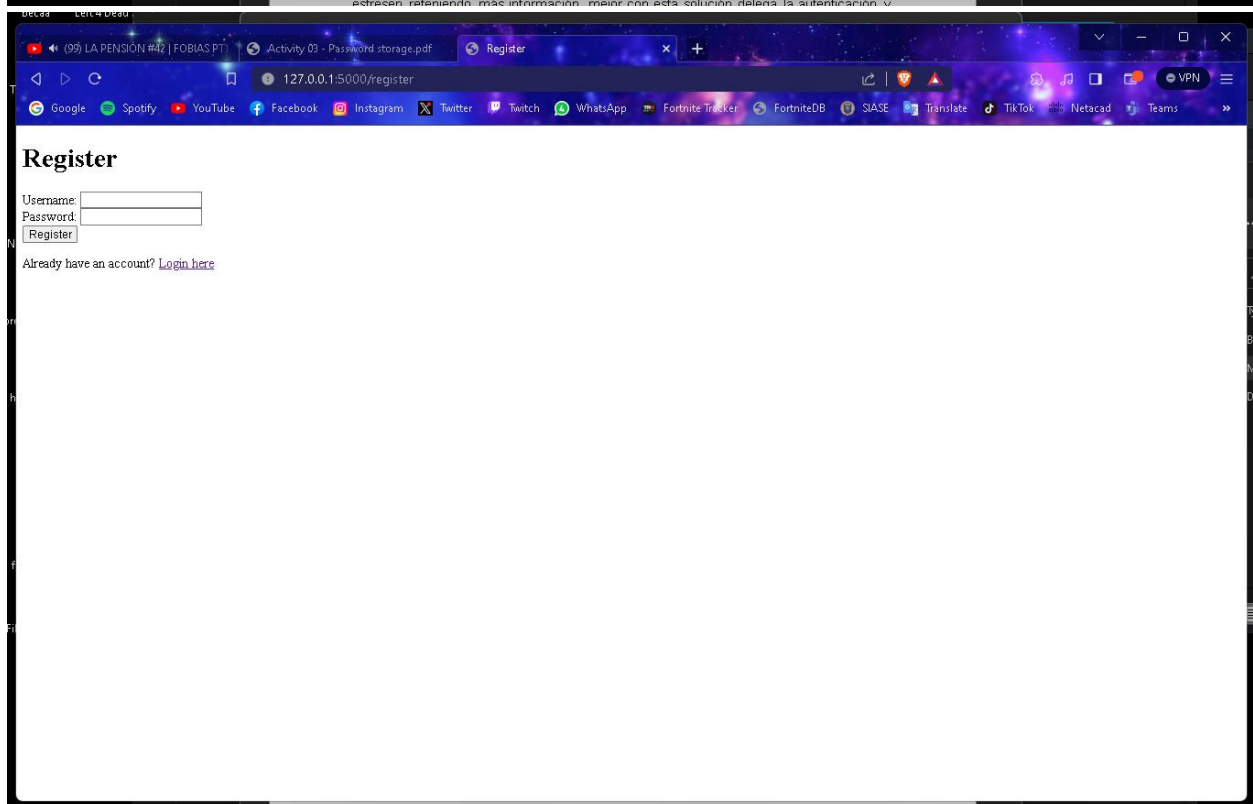
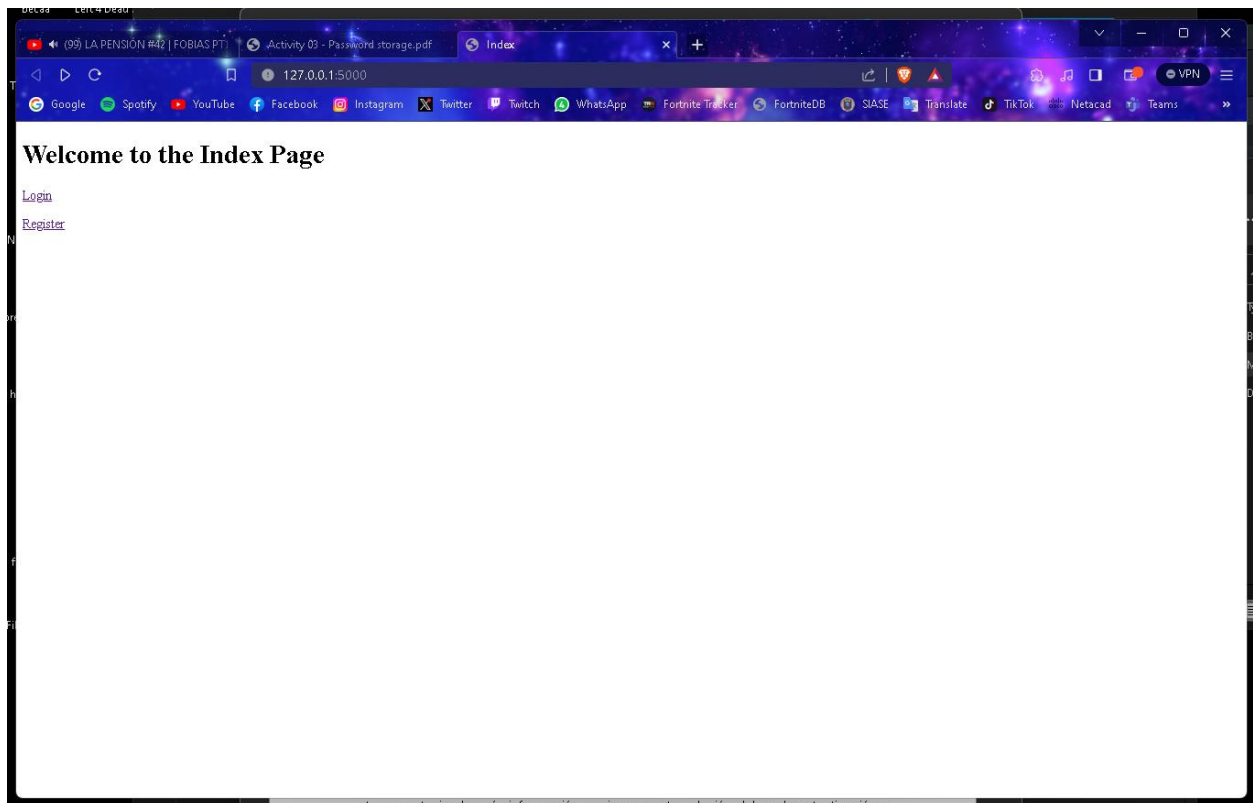


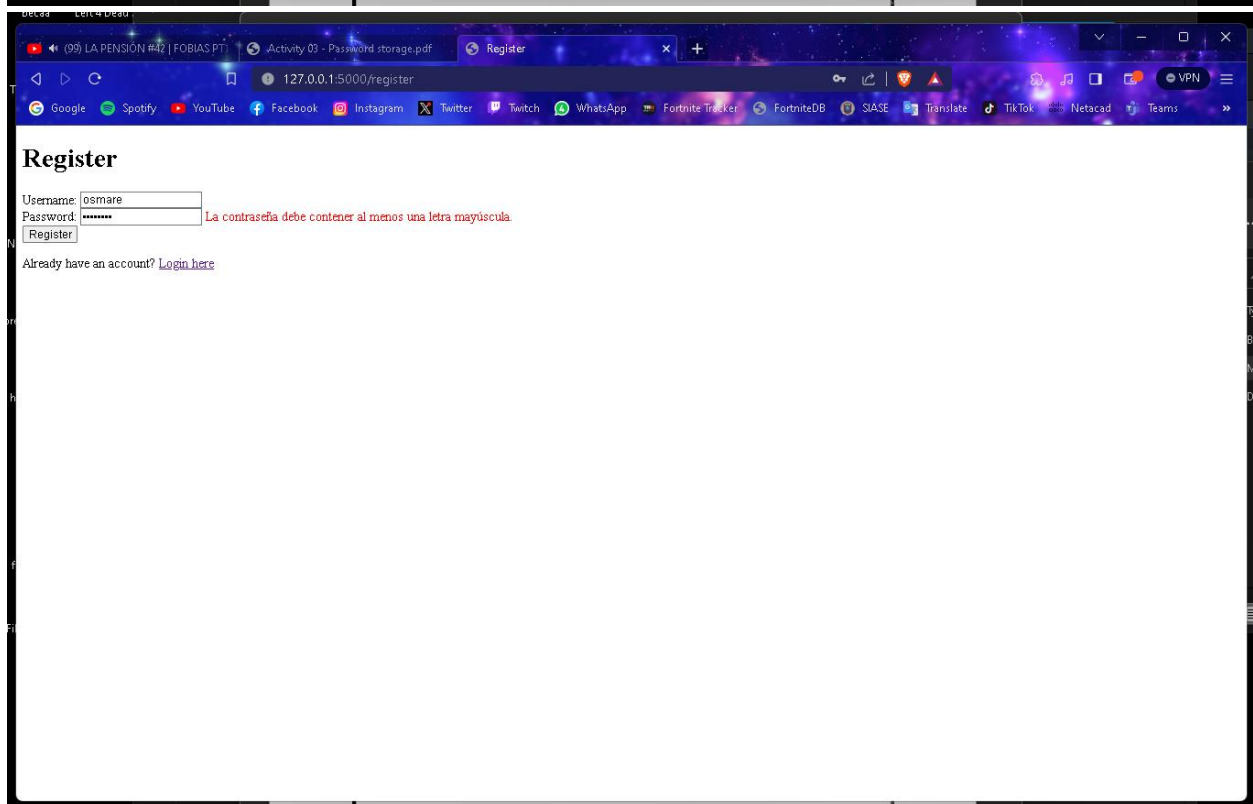
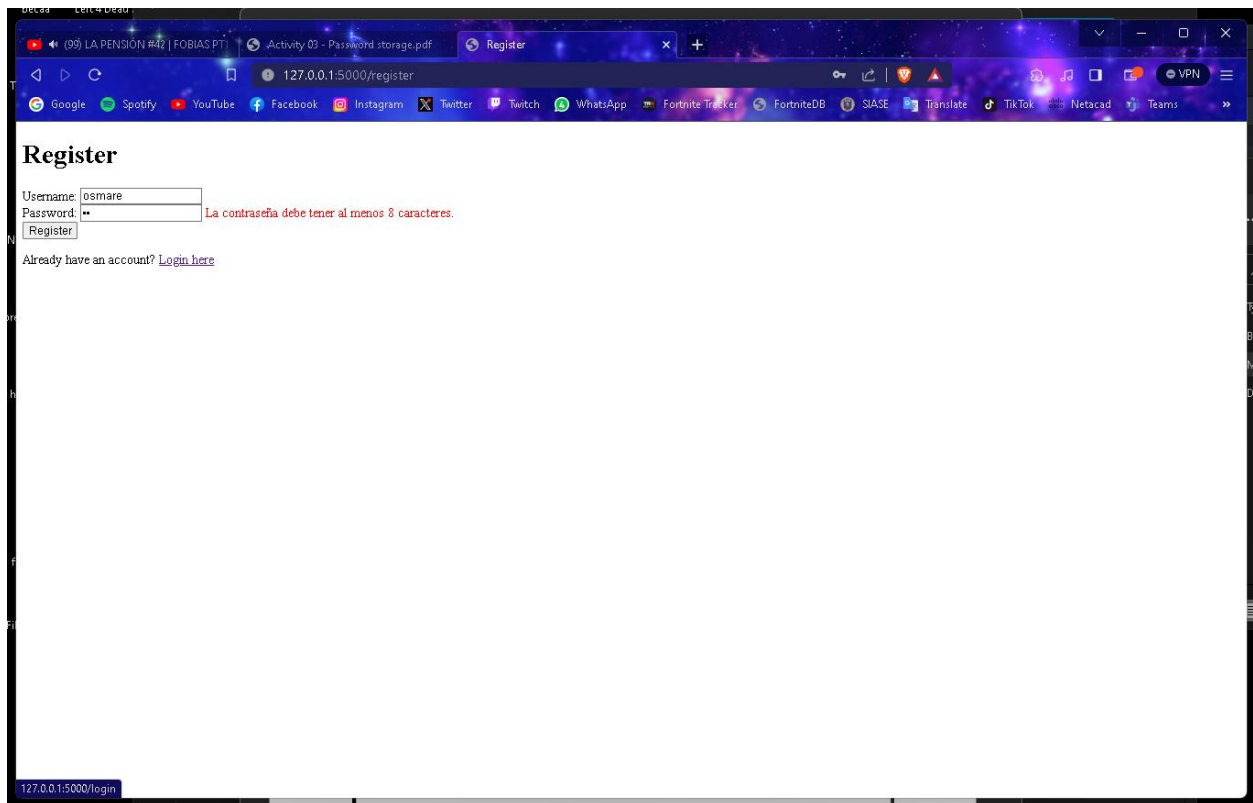
The image displays two screenshots of a web development project in progress, showing the code for a password management system.

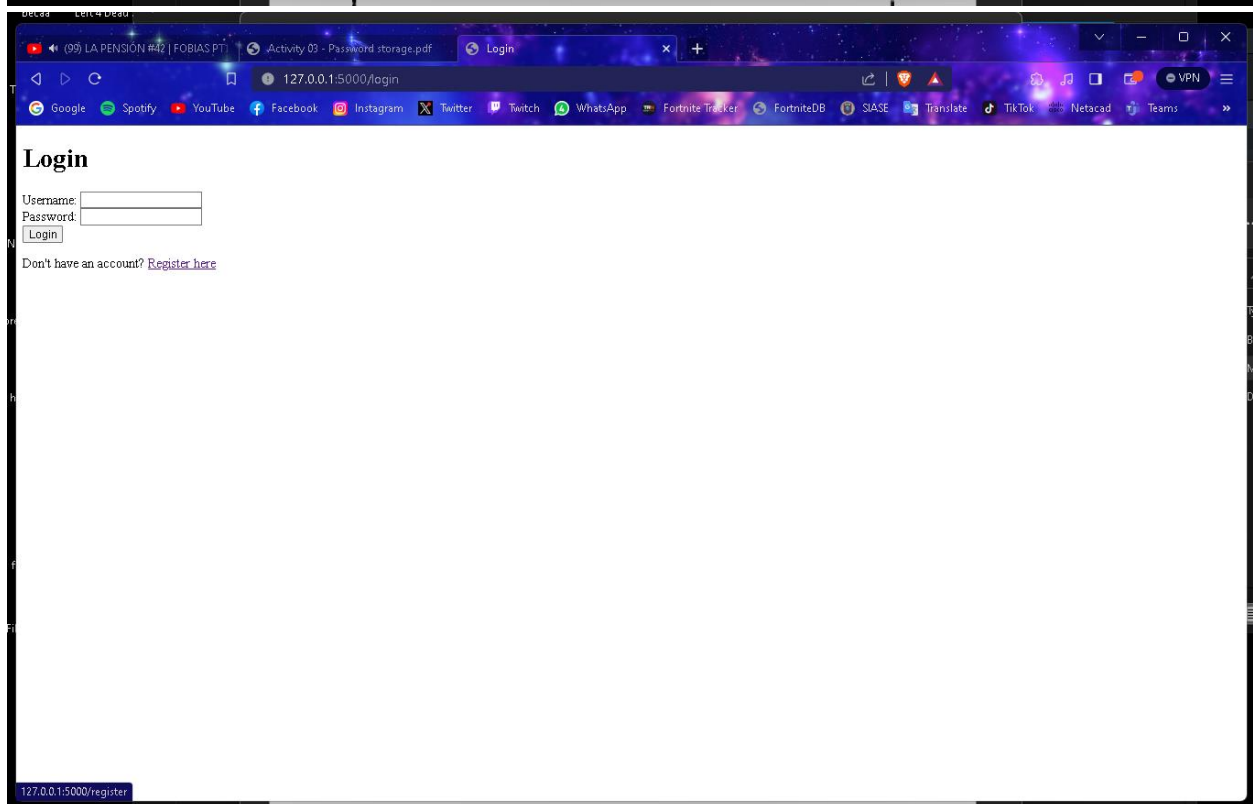
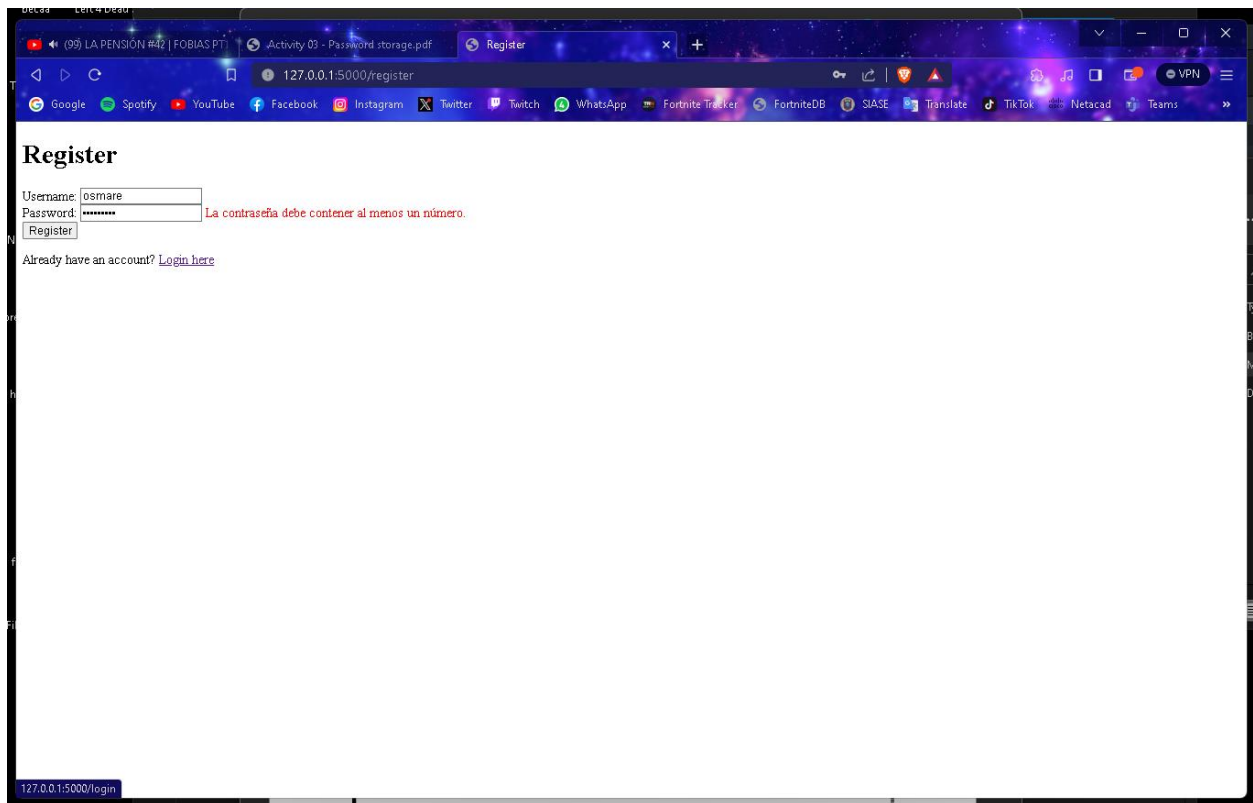
Top Screenshot: register.html

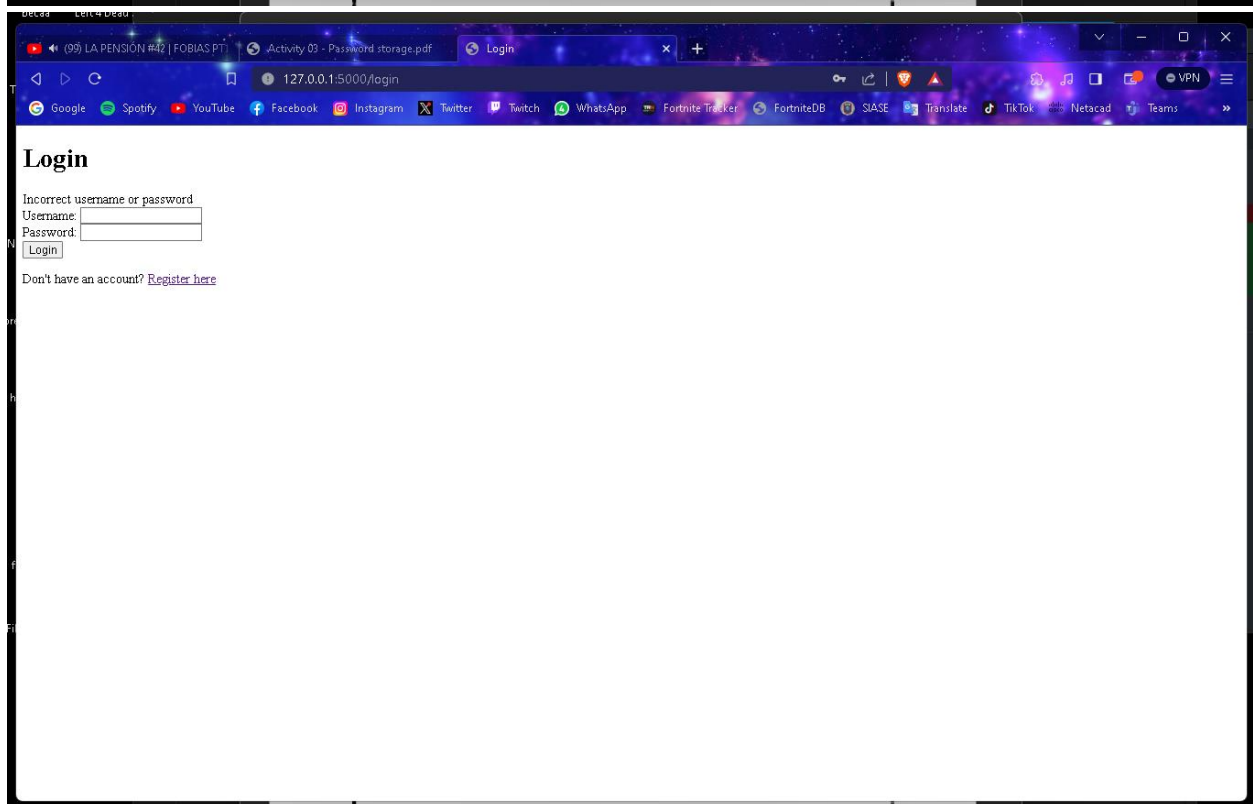
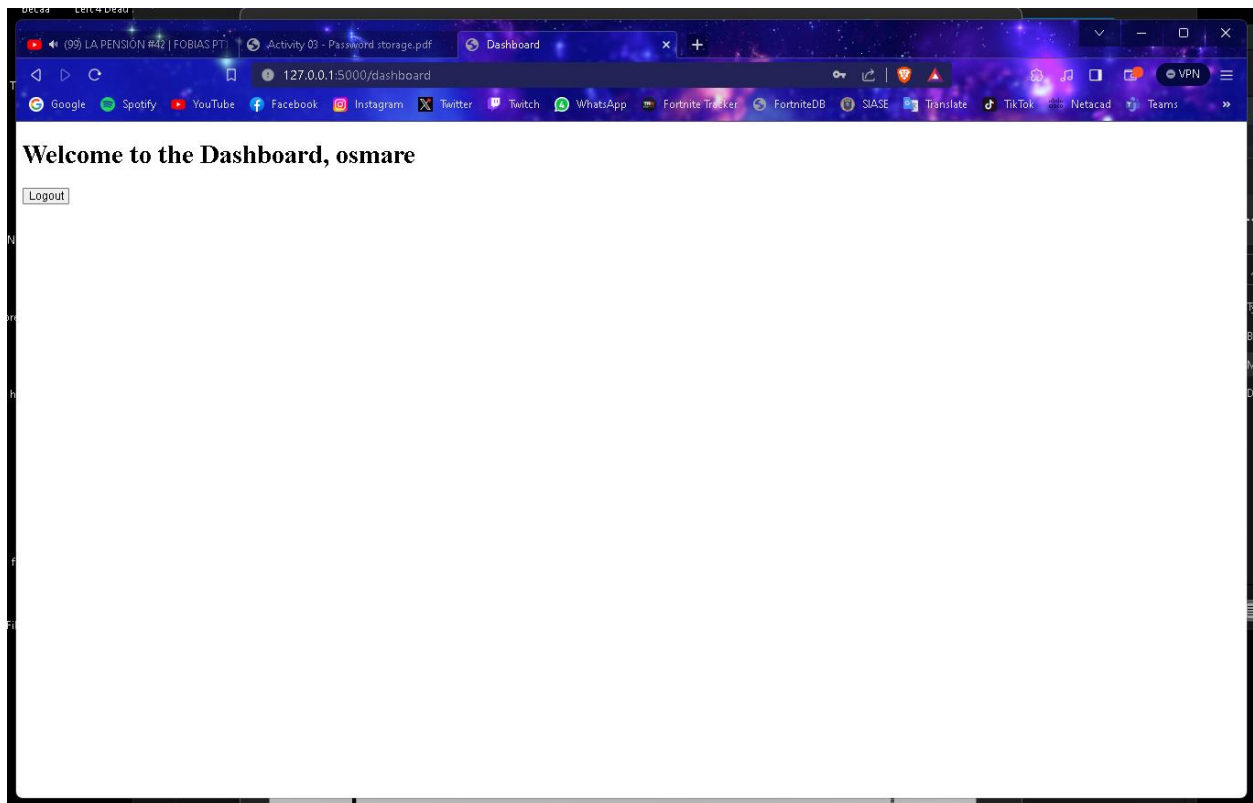
The top screenshot shows the `register.html` file in a code editor. The file is part of a project named `DSS-ALMACENAMIENTO`. The code defines a registration form with the following structure:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Register</title>
7 </head>
8 <body>
9   <h1>Register</h1>
10  <form method="post" action="{{ url_for('register') }}" onsubmit="return validateForm()">
11    <label for="username">Username:</label>
12    <input type="text" id="username" name="username" required>
13    <br>
14    <label for="password">Password:</label>
15    <input type="password" id="password" name="password" required>
16    <span id="passwordError" style="color: red;"></span>
17    <br>
18    <button type="submit">Register</button>
19  </form>
20  <p>Already have an account? <a href="{{ url_for('login') }}">Login here</a></p>
21
22  <script>
23    function validateForm() {
24      // Obtener el valor del campo de contraseña
25      var password = document.getElementById("password").value;
26
27      // Aplicar la política de contraseñas
28      if (password.length < 8) {
29        document.getElementById("passwordError").innerText = "La contraseña debe tener al menos 8 caracteres.";
30        return false;
31      }
32
33      if (!/[a-z]/.test(password)) {
34        document.getElementById("passwordError").innerText = "La contraseña debe contener al menos una letra minúscula.";
35        return false;
36      }
37
38      if (!/[A-Z]/.test(password)) {
39        document.getElementById("passwordError").innerText = "La contraseña debe contener al menos una letra mayúscula.";
40        return false;
41      }
42
43      if (!/\d/.test(password)) {
44        document.getElementById("passwordError").innerText = "La contraseña debe contener al menos un número.";
45        return false;
46      }
47
48      // Si la contraseña pasa todas las verificaciones, borrar el mensaje de error
49    }
50  </script>
```









Preguntas:

1. ¿Por qué es importante utilizar técnicas de almacenamiento seguro de contraseñas en las aplicaciones web?

Es muy importante utilizar técnicas de almacenamiento seguro de contraseñas en las aplicaciones web porque las contraseñas son una parte fundamental de la autenticación de usuarios. Si las contraseñas se almacenan de manera insegura los usuarios pueden enfrentar riesgos de seguridad, como acceso no autorizado a sus cuentas.

2. ¿Qué es la "salt" en el contexto del hashing de contraseñas y por qué es importante?

Se refiere a agregar datos aleatorios a la entrada de una función para garantizar una salida única, el hash, incluso cuando las entradas son las mismas. En la protección por contraseña, salt es una cadena de datos aleatoria que se utiliza para modificar un hash de contraseña.

3. ¿Qué es la "pimienta" en el contexto del hashing de contraseñas y por qué es importante?

La pimienta es un valor secreto adicional que se agrega a las contraseñas antes de aplicar el hash, similar a la "salt". La diferencia clave es que la "pimienta" es un valor global y compartido, mientras que cada usuario tiene su propia "salt". La "pimienta" añade una capa adicional de seguridad, especialmente útil en el caso de una base de datos comprometida, ya que el atacante necesitaría conocer tanto la "pimienta" como la contraseña para descifrar las contraseñas.

4. ¿Qué es Argon2 y cómo se utiliza para almacenar contraseñas de forma segura?

Argon2 es una función de hash diseñada específicamente para el almacenamiento seguro de contraseñas. Utiliza técnicas avanzadas para resistir ataques de fuerza bruta y ataques de diccionario. Argon2 permite ajustar la cantidad de tiempo y memoria necesarios para calcular el hash, lo que hace que sea más costoso y lento para los atacantes realizar ataques.

5. ¿Cuál es la diferencia entre un ataque de fuerza bruta y un ataque de diccionario, y cómo se pueden prevenir en la autenticación de contraseñas en una aplicación web?

Un ataque de fuerza bruta consiste en probar todas las combinaciones posibles de contraseñas, mientras que un ataque de diccionario conlleva probar palabras comunes o combinaciones de palabras que podrían estar en una lista que se haga.

Reflexión final (conclusión):

En conclusión, esta actividad me gustó mucho ya que me dio una cierta experiencia en el desarrollo de aplicaciones web seguras, principalmente centrándose en el almacenamiento y autenticación de contraseñas.

Pienso que el tanto como el uso de la librería argon2-cffi de contraseñas, el reforzamiento de conocimiento de SQLite y la implementación de salt y pimienta fueron indispensables para mejorar la manera de realizar un login más seguro ya que así reconocí la importancia del almacenamiento seguro de contraseñas en las aplicaciones de hoy en día.

En cuanto a las dificultades encontradas, se identificaron errores en el código, como la ubicación incorrecta de ciertas líneas y el uso de sintaxis inadecuada para acceder a elementos de datos. Estos desafíos fueron superados mediante una investigación y el analizado del código.

En esta actividad me gustó que no se nos proporcionó tanto código o ayuda ya que yo mismo tuve que realizar los templates de index, login, register y dashboard que fueron utilizados para el correcto funcionamiento del programa para modificar la base de datos.

También realice un código para que mostrara los registros guardados en la base de datos para visualizar que si este funcionando correctamente y que si se agregan los nuevos usuarios.

En general, fue una práctica muy completa, sin embargo, me hubiera gustado que no tuviera los errores de lógica en el código, esto me hubiera ahorrado algo de tiempo en la realización de la actividad.

.

Bibliografía:

Argon2-cffi. (n.d.). PyPI. Retrieved November 12, 2023, from

<https://pypi.org/project/argon2-cffi/>

Infante, D. C. H. (2022, June 3). *Seguridad en aplicaciones web: Qué es, cómo funciona y los mejores servicios*. Tutoriales Hostinger.

<https://www.hostinger.mx/tutoriales/seguridad-en-aplicaciones-web>

Rodríguez, D. (2022, March 25). Ventajas de guardar las contraseñas con sal y pimienta. *Analytics Lane*. <https://www.analyticslane.com/2022/03/25/ventajas-de-guardar-las-contrasenas-con-sal-y-pimienta/>

Mazara, K. (2021, June 21). *¿Qué es salting? - Recursos de arquitectura y diseño seguros (CompTIA Security+ SY0-601)*.