Lista 9

Gerado com IA

Exercício: Implementando Cadastro com Criptografia de Senhas e Banco de Dados SQLite

Objetivo: Modificar a aplicação para criptografar as senhas dos usuários no momento do cadastro e realizar a validação ao fazer login, utilizando o pacote `werkzeug.security`.

Passo 1: Preparação do Ambiente

1. **Instale o pacote necessário (se ainda não instalado):**
   - Execute:

```
pip install flask werkzeug
```

2. **Atualize as importações no arquivo `app.py`:**
   - Adicione o seguinte:

```
from werkzeug.security import generate_password_hash,
check_password_hash
```

3. **Objetivo deste passo:** Garantir que a aplicação tenha o pacote correto para realizar a criptografia.

Passo 2: Configurando o Banco de Dados

1. **Atualize o banco de dados:**
   - No início de `app.py`, adicione ou mantenha a configuração do banco:

```
import sqlite3

DATABASE = 'usuarios.db'

def conectar_banco():
    return sqlite3.connect(DATABASE)

def criar_tabela():
    conn = conectar_banco()
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS usuarios (
```

```
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            nome TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

criar_tabela()
```

2. **Objetivo deste passo:** Garantir que o banco de dados esteja pronto para armazenar os usuários.

Passo 3: Implementando o Cadastro com Criptografia

1. **Atualize a rota `/cadastro`:**
   - Modifique o código para criptografar as senhas antes de salvar no banco:

```
@app.route('/cadastro', methods=['GET', 'POST'])
def cadastro():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        nome = request.form['nome']

        # Criptografa a senha antes de salvar
        senha_criptografada = generate_password_hash(password)

        try:
            conn = conectar_banco()
            cursor = conn.cursor()
            cursor.execute('''
                INSERT INTO usuarios (username, password, nome)
                VALUES (?, ?, ?)
            ''', (username, senha_criptografada, nome))
            conn.commit()
            conn.close()
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            return render_template('cadastro.html', erro='Usuário já
cadastrado.')

    return render_template('cadastro.html')
```

2. **Objetivo deste passo:** Garantir que as senhas sejam armazenadas de forma segura.

Passo 4: Atualizando o Login

1. **Atualize a rota `/login`:**
    - Modifique a validação para comparar senhas utilizando `check_password_hash`:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = conectar_banco()
        cursor = conn.cursor()
        cursor.execute('''
            SELECT password FROM usuarios WHERE username = ?
        ''', (username,))
        usuario = cursor.fetchone()
        conn.close()

        if usuario and check_password_hash(usuario[0], password):
            session['username'] = username
            resposta = make_response(redirect(url_for('dashboard')))
            resposta.set_cookie('username', username, max_age=60*60*24)
            return resposta

        return render_template('login.html', erro='Usuário ou senha
inválidos.')

    if 'username' in session:
        return redirect(url_for('dashboard'))

    return render_template('login.html')
```

2. **Objetivo deste passo:** Validar a senha criptografada corretamente durante o login.

Passo 5: Listando Usuários

1. **Atualize ou mantenha a rota `/usuarios`:**
    - A lógica continua acessando diretamente os dados do banco:

```
@app.route('/usuarios')
def listar_usuarios():
    if 'username' not in session:
        return redirect(url_for('login'))

    conn = conectar_banco()
    cursor = conn.cursor()
    cursor.execute('SELECT nome, username FROM usuarios')
    usuarios = cursor.fetchall()
    conn.close()
```

```
    return render_template('usuarios.html', usuarios=usuarios)
```

2. **Objetivo deste passo:** Permitir que somente usuários logados visualizem a lista de usuários.

Passo 6: Testando a Aplicação

1. **Inicie a aplicação:**
   - Execute o servidor:

```
python app.py
```

2. **Siga os passos:**
   - Acesse `http://127.0.0.1:5000/cadastro` para registrar novos usuários.
   - Faça login com um dos usuários cadastrados.
   - Acesse `http://127.0.0.1:5000/usuarios` para ver a lista de usuários cadastrados.
3. **Teste de proteção:**
   - Confirme que `/usuarios` só pode ser acessado após login.

Conclusão do Exercício

Neste exercício, você:

1. Implementou a criptografia de senhas usando o pacote `werkzeug.security`.
2. Atualizou o fluxo de cadastro e login para manipular senhas criptografadas.
3. Manteve os dados persistentes em um banco de dados SQLite.

Desafio Extra:

1. Adicione uma funcionalidade para redefinir a senha de um usuário.
2. Implemente a validação de força da senha no momento do cadastro.