

Nombre: Andy Ezequiel Sanic Tiul

Carnet: 202006699

Link: <https://github.com/AndyST70/Practicas-SA-A-202006699/tree/main/Tareas%20vacas/tarea%20extra>

CASO DE USO VIBE CODING : APLICACIÓN AUTENTICACIÓN TIENDA EN LINEA

Se desarrollo una aplicación de autenticación para un sistema de tienda en línea el sistema permite lo siguiente:

- Se registran nuevos usuarios con nombre, correo y contraseña
- Inician sesión con email y contraseña

Se protegen por ultimo las contraseñas con SHA256, se izo uso de JWT para la autenticación, el frontend usa react, mui, para una interfaz moderna.

BACKEND

Quiero que construyas una aplicación de autenticación básica para una tienda en línea usando las siguientes herramientas y condiciones:

- Backend: Flask en Python (sin usar Blueprints).
- Base de datos: MySQL (usa mysql-connector-python).
- Seguridad: Las contraseñas deben ser hasheadas usando hashlib.sha256.
- Frontend: React con Material-UI (MUI).
- La aplicación debe incluir: Un formulario de registro de usuario (nombre, correo, contraseña). Un formulario de inicio de sesión.
- Validación en ambos formularios (campos vacíos, correos repetidos). Manejo de sesiones simples (usando session de Flask). Interfaz clara y responsiva con Material UI.

Restricciones: No se permite usar Blueprints de Flask.

El código debe estar separado pero en un solo archivo para backend (app.py por ejemplo). Todo debe estar bien documentado con comentarios. crea el script de la base de datos y haz uso de JWT Objetivo final: Que un usuario pueda registrarse y luego iniciar sesión en una tienda. Si el login es exitoso, que se le muestre un mensaje de bienvenida con su nombre. Si falla, un mensaje de error.

FRONTEND

Necesito que construyas el frontend en React con Material UI (MUI) para una aplicación de autenticación que ya tiene su backend funcionando con Flask y MySQL.

La aplicación debe tener:

- Un formulario de registro con los campos: nombre, email, password.
- Un formulario de inicio de sesión con los campos: email, password.
- Un componente de recuperación de contraseña opcional (como Recuperar.jsx).
- Requisitos: Usa useState para manejar el formulario.
- Usa FormData para enviar datos al backend.

- Usa fetch o funciones importadas desde endpoints/ para conectarse con el backend (por ejemplo: registrar(formData), login(formData)).
- Muestra mensajes de éxito o error con <Typography> usando color verde (success.main) o rojo (error).

HERRAMIENTA DE VIBE CODING : APLICACIÓN DE AUTENTICACIÓN

Se hizo uso de Chat-Gpt como llma y Github Copilot como vibe coding para este trabajo

Salida que emitió para el backend

BACKEND

```
✂ GitHub Copilot

e detalle a continuación los archivos y scripts necesarios para cumplir con tu requerimiento. Incluyo:

• Script SQL para la base de datos MySQL.
• Backend en Flask (app.py) con JWT, manejo de sesiones y hash de contraseñas.
• Frontend en React con Material-UI, con formularios de registro e inicio de sesión, validaciones y mensajes.

Script SQL para la base de datos

-- script_db.sql
CREATE DATABASE IF NOT EXISTS tienda_online CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE tienda_online;

CREATE TABLE IF NOT EXISTS usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  correo VARCHAR(100) NOT NULL UNIQUE,
  password_hash VARCHAR(64) NOT NULL
);
```

```

# Ruta de registro
@app.route('/api/register', methods=['POST'])
def register():
    data = request.json
    nombre = data.get('nombre', '').strip()
    correo = data.get('correo', '').strip().lower()
    password = data.get('password', '')

    # Validaciones
    if not nombre or not correo or not password:
        return jsonify({'error': 'Todos los campos son obligatorios.'}), 400

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT id FROM usuarios WHERE correo = %s", (correo,))
    if cursor.fetchone():
        cursor.close()
        conn.close()
        return jsonify({'error': 'El correo ya está registrado.'}), 400

    password_hash = hashlib.sha256(password.encode()).hexdigest()
    cursor.execute("INSERT INTO usuarios (nombre, correo, password_hash) VALUES (%s, %s, %s)",
                  (nombre, correo, password_hash))
    conn.commit()
    cursor.close()
    conn.close()
    return jsonify({'message': 'Usuario registrado correctamente.'}), 201

# Ruta de login

```

Agregar contexto...

```

# Ruta de login
@app.route('/api/login', methods=['POST'])
def login():
    data = request.json
    correo = data.get('correo', '').strip().lower()
    password = data.get('password', '')

    if not correo or not password:
        return jsonify({'error': 'Todos los campos son obligatorios.'}), 400

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT id, nombre, password_hash FROM usuarios WHERE correo = %s", (correo,))
    user = cursor.fetchone()
    cursor.close()
    conn.close()

    if not user or user['password_hash'] != hashlib.sha256(password.encode()).hexdigest():
        return jsonify({'error': 'Correo o contraseña incorrectos.'}), 401

    # Generar JWT
    token = jwt.encode({
        'user_id': user['id'],
        'nombre': user['nombre'],
        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=2)
    }, JWT_SECRET, algorithm='HS256')

    session['user_id'] = user['id']
    session['nombre'] = user['nombre']

```

Agregar contexto...

```
# Ruta protegida de ejemplo
@app.route('/api/profile', methods=['GET'])
def profile():
    token = request.headers.get('Authorization', '').replace('Bearer ', '')
    if not token:
        return jsonify({'error': 'Token requerido.'}), 401
    try:
        data = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
        return jsonify({'nombre': data['nombre']}), 200
    except jwt.ExpiredSignatureError:
        return jsonify({'error': 'Token expirado.'}), 401
    except jwt.InvalidTokenError:
        return jsonify({'error': 'Token inválido.'}), 401

if __name__ == '__main__':
    app.run(debug=True)
```

2 vulnerabilidades ->

FRONTEND

Cambia 'TU_PASSWORD' por la contraseña real de tu usuario MySQL.

3. Frontend React (estructura y componentes principales)

Estructura recomendada

```
/src
  /components
    LoginForm.jsx
    RegisterForm.jsx
    Welcome.jsx
  App.jsx
  index.js
```

Ejemplo de RegisterForm.jsx

```
import React, { useState } from 'react';
```

4. Notas de instalación

- Instala dependencias backend:

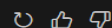
```
pip install flask flask-cors mysql-connector-python pyjwt
```

- Instala dependencias frontend:

```
npx create-react-app tienda-frontend
cd tienda-frontend
npm install @mui/material @emotion/react @emotion/styled
```

¿Quieres que genere los archivos y carpetas automáticamente para ti? ¿O necesitas ayuda con la configuración de entorno?

Código similar encontrado con 1 tipo de licencia - [Ver coincidencias](#)



Agregar contexto...


Ejemplo de RegisterForm.jsx

```
import React, { useState } from 'react';
import { TextField, Button, Box, Typography, Alert } from '@mui/material';

export default function RegisterForm({ onRegisterSuccess }) {
  const [nombre, setNombre] = useState('');
  const [correo, setCorreo] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [success, setSuccess] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setSuccess('');
    if (!nombre || !correo || !password) {
      setError('Todos los campos son obligatorios.');
```

```
      return;
    }
    const res = await fetch('http://localhost:5000/api/register', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ nombre, correo, password }),
      credentials: 'include'
    });
    const data = await res.json();
    if (res.ok) {
```

 Agregar contexto...

Edición de archivos en el área de trabajo en modo agente

CHAT COMO LLMA

```
const fd = new FormData();
fd.append('name', name);
fd.append('email', email);
fd.append('password', password);

const res = await registrar(fd);
setMsg({
  text: res.message || res.error || 'Error desconocido',
  error: !res.message,
});

return (
  <Box mt={4}>
    <Typography variant="h6">Registro</Typography>
    <TextField label="Nombre" name="name" fullWidth margin="normal" onChange={handleChange} />
    <TextField label="Correo" name="email" fullWidth margin="normal" onChange={handleChange} />
    <TextField label="Contraseña" name="password" type="password" fullWidth margin="normal" onCh
    <Button variant="contained" onClick={handleSubmit}>Registrarse</Button>
    {msg.text && (
      <Typography mt={2} color={msg.error ? 'error.main' : 'success.main'}>
        {msg.text}
      </Typography>
    )}
  </Box>
```

src/endpoints/index.js

```
js

export const url = "http://localhost:5000";

export const registrar = async (formData) => {
  return fetch(url + "/register", {
    method: "POST",
    body: formData,
    credentials: "include",
  }).then(res => res.json());
};

export const login = async (formData) => {
  return fetch(url + "/login", {
    method: "POST",
    body: formData,
    credentials: "include",
  }).then(res => res.json());
};
```

```

const fd = new FormData();
fd.append('email', email);
fd.append('password', password);

const res = await login(fd);
if (res.token) {
  localStorage.setItem('token', res.token);
  onLogin(res.name);
  setMsg({ text: `Bienvenido ${res.name}`, error: false });
} else {
  setMsg({ text: res.error || 'Error en login', error: true });
}
};

return (
  <Box mt={4}>
    <Typography variant="h6">Iniciar Sesión</Typography>
    <TextField label="Correo" name="email" fullWidth margin="normal" onChange={handleChange} />
    <TextField label="Contraseña" name="password" type="password" fullWidth margin="normal" onChange={handleChange} />
    <Button variant="contained" onClick={handleSubmit}>Entrar</Button>
    {msg.text && (
      <Typography mt={2} color={msg.error ? 'error.main' : 'success.main'}>
        {msg.text}
      </Typography>
    )}
  </Box>
);

```

src/Login.jsx

```

import React, { useState } from 'react';
import { TextField, Button, Typography, Box } from '@mui/material';
import { login } from './endpoints';

export default function Login({ onLogin }) {
  const [form, setForm] = useState({ email: '', password: '' });
  const [msg, setMsg] = useState({ text: '', error: false });

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async () => {
    const { email, password } = form;
    if (!email || !password) {
      setMsg({ text: 'Todos los campos son obligatorios', error: true });
      return;
    }

    const fd = new FormData();

```

AJUSTES Y MEJORAS PARA BACKEND

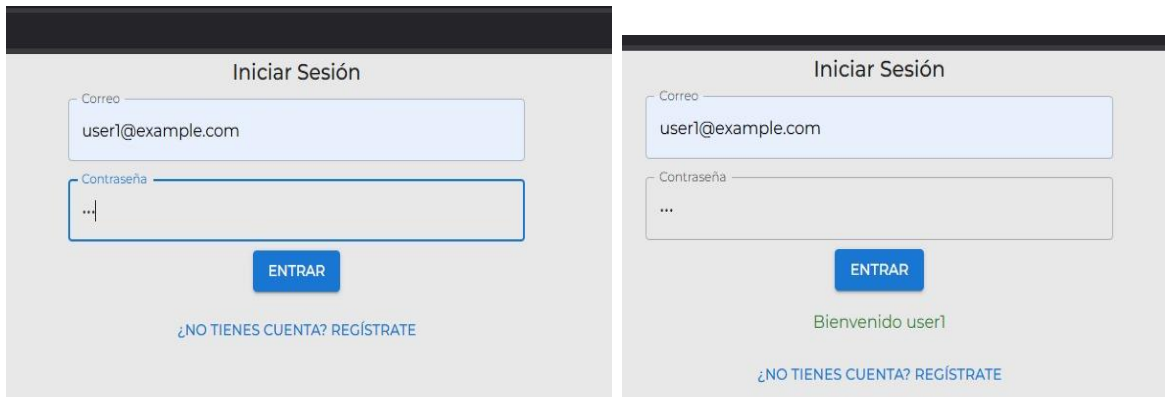
Cabe resaltar que el backend tuvo un par de complicaciones que se arreglaron manualmente, entre ellas la exposición de contraseñas críticas, y un mal manejo al cerrar y abrir las conexiones, pero se arreglo de forma manual, al igual que el manejo de CORS, pero se reestructuro el código en ese sentido y quedó funcional.

AJUSTES Y MEJORAS PARA FRONT

En el frontend ocurrió algo similar: el código generó algunos problemas relacionados con dimensiones, navegación entre páginas y la lógica de estructura, como hacer la transacción entre páginas, dimensiones de textos, de hecho, toco pedirle la misma documentación una ocasión pero se pudo resolver, genero una página muy bien estructurada, si se tiene un formato inicial y se agrega, se puede optimizar el tiempo de gestión y como se desarrolla nuevas páginas

CAPTURAS DE PRODUCTO FINAL

INICIAR SESION



Two screenshots of the login page. The left screenshot shows the login form with fields for "Correo" (user1@example.com) and "Contraseña" (masked with dots), an "ENTRAR" button, and a link "¿NO TIENES CUENTA? REGÍSTRATE". The right screenshot shows the same form after successful login, with a green message "Bienvenido user1" and the same link.

REGISTRAR USUARIO

Inicia sesión'. The right screenshot shows the same form after successful registration, with a green message 'Usuario registrado exitosamente' and the same link." data-bbox="135 731 849 906"/>

Two screenshots of the registration page. The left screenshot shows the registration form with fields for "Nombre" (User2), "Correo" (practica2calificacion@gmail.com), and "Contraseña" (masked with dots), a "REGISTRARSE" button, and a link "Ya tienes cuenta? [Inicia sesión](#)". The right screenshot shows the same form after successful registration, with a green message "Usuario registrado exitosamente" and the same link.

USUARIO REGISTRADO

Result Grid				
		Filter Rows:	Edit:	
		Export/Import:		Wrap Cell
	id	name	email	password
▶	1	user1	user1@example.com	a665a45920422f9d417e4867efdc4fb8a04a1f3f...
	2	User2	practica2calificacion@gmail.com	a665a45920422f9d417e4867efdc4fb8a04a1f3f...
✱	NULL	NULL	NULL	NULL

CONCLUSION

El uso de Vibe Coding permitió acelerar el desarrollo de una aplicación de autenticación básica, resolviendo tareas repetitivas y permitiendo enfocarse en ajustes críticos y experiencia de usuario. Aunque fue necesario intervenir manualmente en varios puntos (como seguridad, CORS y navegación), el flujo base generado fue útil y adaptable.