

SISTEMA DE GESTIÓN DE TAREAS EFICIENTE

Laboratorio de GSI

Javier Cuartero Corredor & Juan de Dios Carreara Zazo

Nuestro sistema de gestión de tareas eficiente es una aplicación de escritorio que hemos desarrollado y que su principal objetivo facilitar a los usuarios su organización controlando la gestión del tiempo.

Contenido

1. Explicación del código	2
2. LAB BOOK	6
Hito 1 ->.....	6
Hito 2 ->.....	6
Hito 3 ->.....	6
3. Resultado final	7

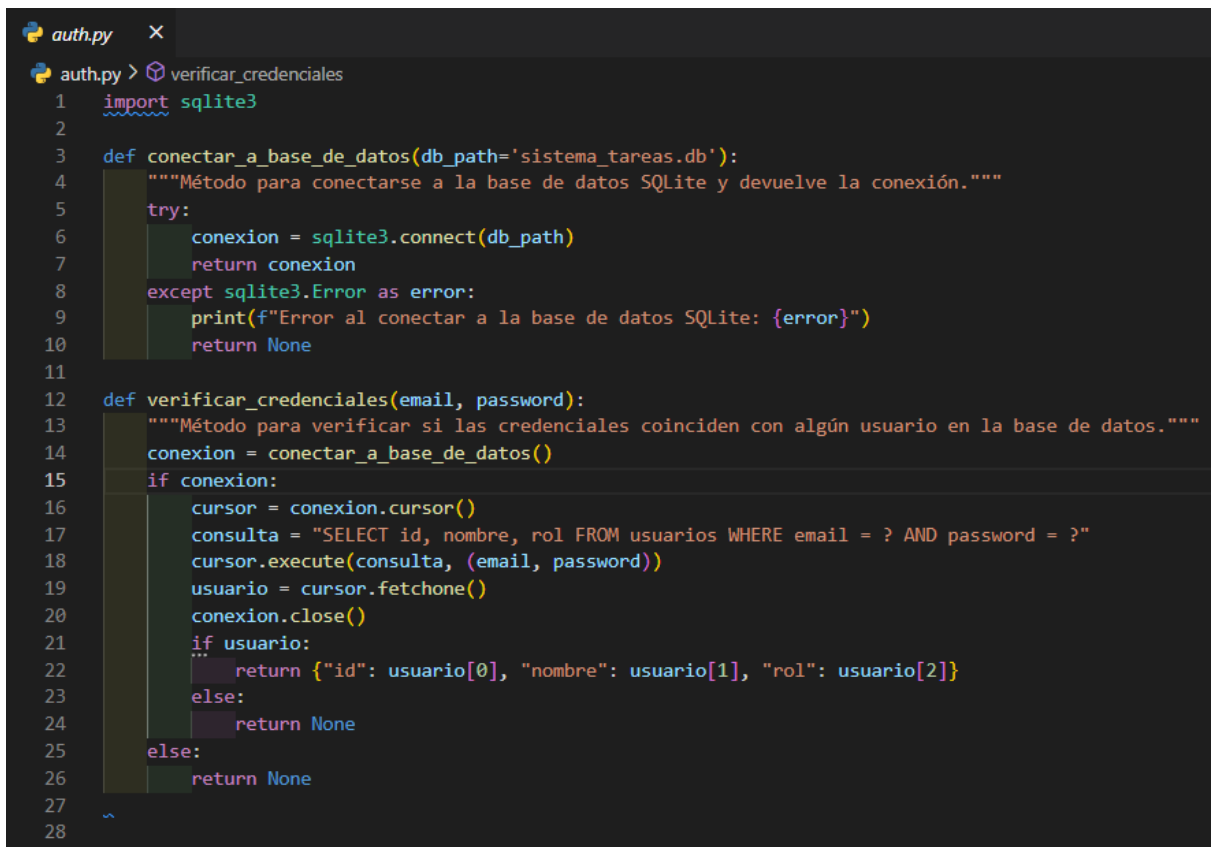
1. Explicación del código

Para este proyecto hemos hecho una aplicación sobre la gestión del tiempo. Para esto hemos realizado una aplicación para poder gestionar los diferentes eventos que tendrá una persona, ya sea el trabajo o sus tareas de casa, incluso sus eventos de ocio como quedar con tus amigos o un partido de pádel. Para esto hemos utilizado Python como lenguaje de programación.

Hemos utilizado distintos archivos:

auth.py: Este archivo contiene funciones relacionadas con la manipulación de una base de datos SQLite y la verificación de credenciales de usuarios.

1. `conectar_a_base_de_datos()`: Esta función se encarga de establecer una conexión a una base de datos SQLite.
2. `verificar_credenciales()`: Esta función verifica si las credenciales proporcionadas (correo electrónico y contraseña) coinciden con algún usuario en la base de datos.



```
auth.py x
auth.py > verificar_credenciales
1 import sqlite3
2
3 def conectar_a_base_de_datos(db_path='sistema_tareas.db'):
4     """Método para conectarse a la base de datos SQLite y devuelve la conexión."""
5     try:
6         conexion = sqlite3.connect(db_path)
7         return conexion
8     except sqlite3.Error as error:
9         print(f"Error al conectar a la base de datos SQLite: {error}")
10        return None
11
12 def verificar_credenciales(email, password):
13     """Método para verificar si las credenciales coinciden con algún usuario en la base de datos."""
14     conexion = conectar_a_base_de_datos()
15     if conexion:
16         cursor = conexion.cursor()
17         consulta = "SELECT id, nombre, rol FROM usuarios WHERE email = ? AND password = ?"
18         cursor.execute(consulta, (email, password))
19         usuario = cursor.fetchone()
20         conexion.close()
21         if usuario:
22             return {"id": usuario[0], "nombre": usuario[1], "rol": usuario[2]}
23         else:
24             return None
25     else:
26         return None
27
28
```

db.py: Este archivo Python contiene una serie de funciones que interactúan con una base de datos SQLite llamada "sistema_tareas.db". Aquí hay una descripción de las principales funciones y su propósito:

1. `conectar_a_base_de_datos(path='sistema_tareas.db')`: Esta función se utiliza para establecer una conexión con la base de datos SQLite. Si la base de datos especificada en el camino (path) no existe, se creará una nueva. Retorna el objeto de conexión SQLite o None si hay algún error.

2. `crear_tablas(conexion)`: Esta función se encarga de crear las tablas necesarias en la base de datos si no existen. Define dos tablas: usuarios y tareas.
3. `insertar_usuario(conexion, nombre, email, password, rol)`: Inserta un nuevo usuario en la tabla de usuarios con los detalles proporcionados como argumentos.
4. `obtener_usuarios(conexion)`: Obtiene todos los usuarios de la base de datos y devuelve una lista de diccionarios, donde cada diccionario representa un usuario con su ID, nombre, correo electrónico y rol.
5. `obtener_tareas_de_usuario(conexion, usuario_id)`: Obtiene todas las tareas asignadas a un usuario específico utilizando su ID. Las tareas se ordenan por fecha de entrega y luego por prioridad.
6. `insertar_tarea(conexion, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea, usuario_id)`: Inserta una nueva tarea en la tabla de tareas con los detalles proporcionados.
7. `eliminar_tarea(conexion, tarea_id)`: Elimina una tarea de la base de datos utilizando su ID.
8. `marcar_tarea_como_hecha(conexion, tarea_id)`: Marca una tarea como completada (en este caso, la elimina) utilizando su ID.
9. `obtener_tarea_por_id(conexion, tarea_id)`: Obtiene una tarea específica de la base de datos utilizando su ID.
10. `actualizar_tarea(conexion, tarea_id, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea)`: Actualiza los detalles de una tarea existente en la base de datos utilizando su ID.

```

db.py 9+
db.py > insertar_tarea
1  import sqlite3
2  from sqlite3 import Error
3
4  def conectar_a_base_de_datos(path='sistema_tareas.db'):
5      """Método para conectar la base de datos, si no existe la creamos"""
6      conexion = None
7      try:
8          conexion = sqlite3.connect(path)
9          print("Conexión exitosa a SQLite")
10     except Error as e:
11         print(f"Error al conectar a SQLite: {e}")
12     return conexion
13
14 def crear_tablas(conexion):
15     """Método para crear las tablas en la base de datos si no existen."""
16     crear_tabla_usuarios = """
17     CREATE TABLE IF NOT EXISTS usuarios (
18         id INTEGER PRIMARY KEY AUTOINCREMENT,
19         nombre TEXT NOT NULL,
20         email TEXT NOT NULL UNIQUE,
21         password TEXT NOT NULL,
22         rol TEXT NOT NULL CHECK (rol IN ('empleado', 'jefe'))
23     );
24     """
25     crear_tabla_tareas = """
26     CREATE TABLE IF NOT EXISTS tareas (
27         id INTEGER PRIMARY KEY AUTOINCREMENT,
28         titulo TEXT NOT NULL,
29         descripcion TEXT,
30         fecha_entrega TEXT NOT NULL,
31         prioridad INTEGER NOT NULL,
32         tipo_tarea TEXT NOT NULL,
33         usuario_id INTEGER NOT NULL,
34         FOREIGN KEY (usuario_id) REFERENCES usuarios (id)
35     );
36     """
37     try:
38         cursor = conexion.cursor()
39         cursor.execute(crear_tabla_usuarios)

```

```

db.py 9+
db.py > insertar_tarea
14 def crear_tablas(conexion):
40     cursor.execute(crear_tabla_tareas)
41     conexion.commit()
42     except Error as e:
43         print(f"Error al crear tablas: {e}")
44
45 def insertar_usuario(conexion, nombre, email, password, rol):
46     """Método para insertar un nuevo usuario en la base de datos."""
47     sql = '''INSERT INTO usuarios (nombre, email, password, rol) VALUES (?, ?, ?, ?);'''
48     try:
49         "Bucle para insertar usuarios en la base de datos."
50         cursor = conexion.cursor()
51         cursor.execute(sql, (nombre, email, password, rol))
52         conexion.commit()
53         return cursor.lastrowid
54     except Error as e:
55         print(f"Error al insertar usuario: {e}")
56         return None
57
58 def obtener_usuarios(conexion):
59     """Método para obtener todos los usuarios de la base de datos."""
60     sql = '''SELECT id, nombre, email, rol FROM usuarios;'''
61     try:
62         cursor = conexion.cursor()
63         cursor.execute("SELECT id, nombre, email, rol FROM usuarios")
64         usuarios = [{"id": row[0], "nombre": row[1], "email": row[2], "rol": row[3]} for row in cursor.fetchall()]
65         return usuarios
66     except Error as e:
67         print(f"Error al obtener usuarios: {e}")
68         return []
69
70 def obtener_tareas_de_usuario(conexion, usuario_id):
71     """Método para obtener todas las tareas asignadas a un usuario específico."""
72     sql = '''SELECT id, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea FROM tareas WHERE usuario_id = ? ORDER BY fecha_entrega, prioridad DESC;'''
73     cursor = conexion.cursor()
74     cursor.execute(sql, (usuario_id,))
75     tareas = [{"id": row[0], "titulo": row[1], "descripcion": row[2], "fecha_entrega": row[3], "prioridad": row[4], "tipo_tarea": row[5]} for row in cursor.fetchall()]
76     return tareas

```

```

db.py 9+
db.py > insertar_tarea

78 def insertar_tarea(conexion, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea, usuario_id):
79     """Método para insertar una nueva tarea en la base de datos."""
80     sql = 'INSERT INTO tareas (titulo, descripcion, fecha_entrega, prioridad, tipo_tarea, usuario_id) VALUES (?, ?, ?, ?, ?, ?)'
81     cursor = conexion.cursor()
82     cursor.execute(sql, (titulo, descripcion, fecha_entrega, prioridad, tipo_tarea, usuario_id))
83     conexion.commit()
84     return cursor.lastrowid
85
86 def eliminar_tarea(conexion, tarea_id):
87     """Eliminar una tarea por su ID."""
88     sql = 'DELETE FROM tareas WHERE id = ?;'
89     cursor = conexion.cursor()
90     cursor.execute(sql, (tarea_id,))
91     conexion.commit()
92
93 def marcar_tarea_como_hecha(conexion, tarea_id):
94     """Marcar una tarea como hecha (en este caso la eliminamos) por su ID. Es igual que eliminar_tarea."""
95     eliminar_tarea(conexion, tarea_id)
96
97 def obtener_tarea_por_id(conexion, tarea_id):
98     """Obtener una tarea por su ID."""
99     sql = 'SELECT id, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea FROM tareas WHERE id = ?;'
100    cursor = conexion.cursor()
101    cursor.execute(sql, (tarea_id,))
102    tarea = cursor.fetchone()
103    return {"id": tarea[0], "titulo": tarea[1], "descripcion": tarea[2],
104           "fecha_entrega": tarea[3], "prioridad": tarea[4], "tipo_tarea": tarea[5]}
105
106 def actualizar_tarea(conexion, tarea_id, titulo, descripcion, fecha_entrega, prioridad, tipo_tarea):
107     """Actualizar una tarea existente en la base de datos."""
108     sql = 'UPDATE tareas SET titulo = ?, descripcion = ?, fecha_entrega = ?, prioridad = ?, tipo_tarea = ? WHERE id = ?;'
109     cursor = conexion.cursor()
110     cursor.execute(sql, (titulo, descripcion, fecha_entrega, prioridad, tipo_tarea, tarea_id))
111     conexion.commit()
112

```

models.py: crea los objetos Usuario y Tarea:

1. La clase Usuario representa a un usuario del sistema y tiene los siguientes atributos: id_usuario, nombre, email, password y rol.
2. La clase Tarea representa una tarea asignada a un usuario y tiene los siguientes atributos: id_tarea, titulo, descripcion, fecha_entrega, prioridad, usuario_id y tipo_tarea.

```

models.py X
models.py > Usuario > _init_

1 class Usuario:
2     def __init__(self, id_usuario, nombre, email, password, rol):
3         self.id_usuario = id_usuario
4         self.nombre = nombre
5         self.email = email
6         self.password = password
7         self.rol = rol
8
9     def __repr__(self):
10        return f"Usuario(id={self.id_usuario}, nombre={self.nombre}, email={self.email}, rol={self.rol})"
11
12 class Tarea:
13     def __init__(self, id_tarea, titulo, descripcion, fecha_entrega, prioridad, usuario_id, tipo_tarea):
14         self.id_tarea = id_tarea
15         self.titulo = titulo
16         self.descripcion = descripcion
17         self.fecha_entrega = fecha_entrega
18         self.prioridad = prioridad
19         self.usuario_id = usuario_id
20         self.tipo_tarea = tipo_tarea
21
22     def __repr__(self):
23        return f"Tarea(id={self.id_tarea}, titulo={self.titulo}, fecha_entrega={self.fecha_entrega}, prioridad={self.prioridad}, tipo_tarea={self.tipo_tarea}, usua

```

main.py: Este script implementa un sistema de gestión de tareas con una interfaz gráfica usando Tkinter. Aquí hay un resumen de lo que hace:

- **Inicio de Sesión:** Cuando se ejecuta el programa, se muestra una ventana para que el usuario inicie sesión. Si las credenciales son correctas, se abre la ventana principal del sistema.
- **Ventana Principal:** La ventana principal muestra un saludo al usuario y proporciona opciones para ver, agregar, editar y eliminar tareas. Además, los usuarios con el rol de "jefe" tienen la capacidad adicional de asignar tareas a otros usuarios.
- **Calendario:** Se incluye un calendario que muestra los días en los que hay tareas. Los días con tareas se marcan en azul claro.
- **Funcionalidades de Tareas:**
 - Se pueden ver las tareas para un día específico haciendo clic en el botón "Mostrar tareas".
 - Se pueden agregar nuevas tareas utilizando el botón "Agregar Tarea".
 - Las tareas se muestran en una lista ordenada por tipo de tarea y prioridad.
 - Se pueden editar y eliminar tareas haciendo clic derecho en ellas.
- **Funcionalidades adicionales para el rol de Jefe:**
 - Los usuarios con el rol de "jefe" pueden asignar tareas a otros usuarios utilizando el botón "Asignar Tarea a Empleado".

2. LAB BOOK

Hito 1 -> En la primera entrega (las dos primeras semanas del proyecto) lo que hicimos fue ver cómo plantear el laboratorio, es decir, qué usarlo, cómo usarlo, repartirnos las tareas, primeros bocetos de la interfaz gráfica...

Hito 2 -> En la segunda entrega (semana 3 y 4) creamos el repositorio GitHub ,definimos cuáles iban a ser las funciones principales del código y empezamos a desarrollarlas, además creamos la base de datos.

Hito 3 -> En las dos últimas semanas nos hemos dedicado más a lo que es “picar” código y refinar detalles y cosas que iban surgiendo o se nos iban ocurriendo

3. Resultado final

Como resultado final tenemos una aplicación de escritorio en la que podemos iniciar sesión.



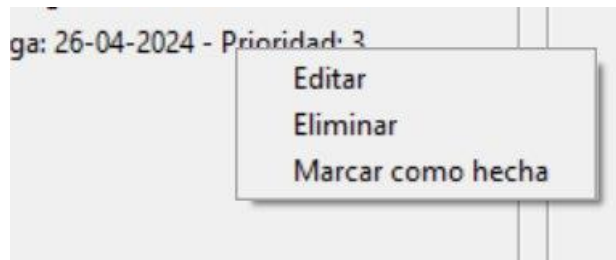
Ventana Inicio de Sesión

Una vez tenemos la sesión iniciada accedemos a la pantalla principal de la aplicación, donde de forma bastante intuitiva podemos añadir, eliminar, modificar y gestionar nuestras tareas.



Ventana pantalla principal

También podemos observar en la imagen anterior, que los días en los que hay alguna tarea para hacer salen marcados en azul clarito en el calendario.



Menú de acciones para cada tarea

A screenshot of a 'Agregar tarea' (Add task) window. The window has a title bar with a pencil icon and the text 'A...'. It contains several input fields: 'Título:', 'Descripción:', 'Fecha de entrega (DD-MM-AAAA):' (with the value '20-04-2024'), 'Prioridad (número):', and 'Tipo de tarea:' (a dropdown menu). At the bottom is an 'Agregar' button.

Ventana agregar tarea

El usuario que esté dado de alta en la base de datos como ‘Jefe’ tiene la misma pantalla y las mismas funcionalidades que un usuario normal, pero este, además puede asignar tareas a sus empleados

Sistema de Gestión de Tareas

¡Bienvenido Julián Ruiz - jefe!

Cerrar sesión

	lun	mar	mié	jue	vie	sáb	dom
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30	1	2	3	4	5
19	6	7	8	9	10	11	12

TRABAJO
Corregir exámenes GSI - Fecha de entrega: 25-05-2024 - Prioridad: 2

OCIO

COTIDIANA

Mostrar tareas
Agregar Tarea
Asignar Tarea a Empleado

Asignar Tarea a Empleado

Empleado:

Javier Cuartero

Juan Perez

Javier Cuartero

Juande de Dios Carrera

Diego López

Mario Serna

Carlos Pérez

Fecha de entrega (dd-mm-aaaa):

Prioridad (número):

Tipo de tarea (trabajo, cotidiana u ocio):

TRABAJO

Asignar Tarea

Ventana asignar tarea a empleados

Un jefe solo puede asignar tareas de tipo trabajo a sus empleados, es decir, no puede asignarles tareas de ocio y/cotidianas.