

Anexo B

Documentación del Sistema de Gestión de Inventario en Python

1. Introducción

El código presentado implementa un sistema de gestión de inventario utilizando Python. El programa utiliza Tkinter para crear la interfaz gráfica de usuario, MySQL para la base de datos y openpyxl para la manipulación de archivos Excel. A continuación, se describen detalladamente cada sección y función del código.

2. Importación de librerías

Primero, se importan las librerías necesarias para el funcionamiento del programa.

```
1 import json
2 import os
3 from tkinter import *
4 from tkinter import ttk, messagebox,
5 filedialog from datetime import datetime
6 import openpyxl
7 import mysql.connector
```

- json y os son módulos utilizados para manejar archivos y datos de configuración.
- tkinter, ttk, messagebox y filedialog permiten crear la interfaz gráfica.
- datetime es usado para obtener la fecha actual.
- openpyxl permite trabajar con archivos de Excel.
- mysql.connector se utiliza para conectar a la base de datos MySQL.



3. Creación y carga de la configuración

El programa necesita un archivo de configuración (config.json) para almacenar la información de conexión a la base de datos.

```

1  # Funci n para crear el archivo de configuraci n si no
   existe
2  def crear_configuracion():
3      config_file = 'config.json'
4
5      if not os.path.exists(config_file):
6          host = input("Ingrese el host de MySQL (por defecto
   'localhost'): ") or 'localhost'
7          user = input("Ingrese el usuario de MySQL: ")
8          password = input("Ingrese la contrase a de MySQL: "
   )
9          database = input("Ingrese el nombre de la base de
   datos: ")
10
11         config = {
12             "host": host,
13             "user": user,
14             "password": password,
15             "database": database
16         }
17
18         with open(config_file, 'w') as f:
19             json.dump(config, f, indent=4)
20
21         print(f"Archivo de configuraci n '{config_file}'
   creado con xito .")
22     else:
23         print(f"El archivo de configuraci n '{config_file}'
   ya existe.")
24
25 # Funci n para cargar la configuraci n desde el archivo
26 def cargar_configuracion():
27     with open('config.json', 'r') as f:
28         return json.load(f)
29
30 # Llamar a la funci n de configuraci n al inicio
31 crear_configuracion()
32 config = cargar_configuracion()

```

- La función `crear-configuración()` verifica si existe el archivo de configuración. Si no existe, solicita al usuario los detalles de la base de datos y los guarda en `config.json`.
- La función `cargar configuración()` lee el archivo `config.json` para cargar la configuración.
- Se llama a `crear configuración()` al inicio para asegurarse de que el archivo esté disponible.

4. Conexión a la base de datos

La función de conexión utiliza la configuración cargada para conectarse a la base de datos.

```

1 def connection():
2     return mysql.connector.connect(
3         host=config["host"],
4         user=config["user"],
5         password=config["password"],
6         database=config["database"]
7     )

```

`connection()` establece una conexión con la base de datos MySQL utilizando los detalles en el archivo de configuración.

5. Configuración de la interfaz grafica

Se crea la ventana principal y se configuran los elementos de la interfaz.



```

1 # Crear la ventana principal
2 window = Tk()
3 window.title("Inventory Management System")
4 window.geometry("900x640")
5 window.configure(bg="#f0f0f0") # Fondo de la ventana
6
7 # Crear el panel dividido
8 paned_window = PanedWindow(window, orient=VERTICAL)
9 paned_window.pack(fill=BOTH, expand=True)
10
11 # Crear el frame para la entrada de datos
12 input_frame = Frame(paned_window, bg="#d0e7ff", relief="
    groove", bd=2)
13 paned_window.add(input_frame, minsize=200)
14
15 # Crear el frame para mostrar la tabla
16 table_frame = Frame(paned_window, bg="#f0f0f0")
17 paned_window.add(table_frame)

```

- Se configura la ventana principal con un título, tamaño y color de fondo.
- Se añade un panel de división con dos **frames**: uno para la entrada de datos y otro para la tabla de resultados.

6. Creacion de la tabla y configuraciones adicionales

El código configura la tabla de datos utilizando Treeview de la librería **ttk**.



```

1 # Crear barras de desplazamiento para la tabla
2 vertical_scrollbar = Scrollbar(table_frame, orient="vertical
    ")
3 vertical_scrollbar.pack(side=RIGHT, fill=Y)
4
5 horizontal_scrollbar = Scrollbar(table_frame, orient="
    horizontal")
6 horizontal_scrollbar.pack(side=BOTTOM, fill=X)
7
8 # Estilos de la tabla
9 my_tree = ttk.Treeview(table_frame, show='headings', height
    =20, yscrollcommand=vertical_scrollbar.set,
    xscrollcommand=horizontal_scrollbar.set)
10 my_tree.pack(side=TOP, fill=X)
11
12 vertical_scrollbar.config(command=my_tree.yview)
13 horizontal_scrollbar.config(command=my_tree.xview)
14
15 # Definir columnas de la tabla, agregando la columna "
    Hoja_de_vida"
16 my_tree['columns'] = ("ID", "Cantidad", "Descripcion", "
    Marca", "Fecha", "Observaciones", "Lugar", "Condicion", "
    Proveedor", "A os_de_servicio", "Valor_Compra", "
    Hoja_de_vida")
17 for col in my_tree['columns']:
18     my_tree.heading(col, text=col)
19     my_tree.column(col, anchor=W)

```



- Se crean barras de desplazamiento para la tabla.
- Se configura la tabla para mostrar los encabezados y se definen las columnas.

7. Variables de entrada y funciones del sistema

Se declaran variables para almacenar datos del usuario y se crean funciones para las operaciones de consulta, inserción, actualización, eliminación, entre otras.

```

1 # Variables de entrada
2 itemIdVar = StringVar()
3 cantidadVar = StringVar()
4 descripcionVar = StringVar()
5 marcaVar = StringVar()
6 observacionesVar = StringVar()
7 lugarVar = StringVar()
8 condicionVar = StringVar()
9 proveedorVar = StringVar()
10 anosServicioVar = StringVar()

11 valorCompraVar = StringVar()
12 hojaVidaPathVar = StringVar() # Nueva variable para la
    direcci n de la hoja de vida

```



- Las variables permiten capturar la información ingresada por el usuario en la interfaz.

8. Botones y Funciones del Sistema de Gestión de Inventarios

En esta sección se describen los botones de la interfaz gráfica de la aplicación, así como sus funciones correspondientes en el código. A continuación, se detalla la funcionalidad de cada uno de los botones:

8.1 Botón de Buscar

El botón de **Buscar** permite consultar artículos en la base de datos en función de criterios de búsqueda. La función asociada a este botón es **query_article()**, la cual construye una consulta **SQL** con los valores proporcionados en los campos de entrada.

```

1 # Funci n para consultar un art culo por ID o campos
  parciales
2 def query_article():
3     itemId = itemIdVar.get()
4     cantidad = cantidadVar.get()
5     descripcion = descripcionVar.get()
6     marca = marcaVar.get()
7     lugar = lugarVar.get()
8     condicion = condicionVar.get()
9
10    query = "SELECT * FROM articulos WHERE 1=1"
11    if itemId:
12        query += f" AND ID LIKE '%{itemId}%'"
13    if cantidad:
14        query += f" AND Cantidad LIKE '%{cantidad}%'"
15    if descripcion:
16        query += f" AND Descripcion LIKE '%{descripcion}%'"
17    if marca:
18        query += f" AND Marca LIKE '%{marca}%'"
19    if lugar:
20        query += f" AND Lugar LIKE '%{lugar}%'"
21    if condicion:
22        query += f" AND Condicion LIKE '%{condicion}%'"
23

```



A continuación, se muestra el código para la función buscar:

```

24     conn = connection()
25     cursor = conn.cursor()
26     cursor.execute(query)
27     results = cursor.fetchall()
28
29     my_tree.delete(*my_tree.get_children()) # Limpiar la
        tabla
30
31     if results:
32         # Mostrar los resultados en la tabla y llenar los
            campos si hay solo un resultado
33         for result in results:
34             my_tree.insert('', 'end', values=result)
35
36         # Si se encuentra un nico resultado, llenar los
            campos autom ticamente
37         if len(results) == 1:
38             result = results[0]
39             itemIdVar.set(result[0])
40             cantidadVar.set(result[1])
41             descripcionVar.set(result[2])
42             marcaVar.set(result[3])
43             observacionesVar.set(result[5])
44             lugarVar.set(result[6])
45             condicionVar.set(result[7])
46             proveedorVar.set(result[8])
47             anosServicioVar.set(result[9])
48             valorCompraVar.set(result[10])
49             hojaVidaPathVar.set(result[11]) # Cargar la
                ruta de hoja de vida en el campo
                    correspondiente
50     else:
51         messagebox.showwarning("No Result", "No articles
            found with those criteria.")
52
53     conn.close()

```

8.2 Botón de Agregar/Actualizar

El botón de **Agregar/Actualizar** permite añadir un nuevo artículo a la base de datos o actualizar uno existente. La función asociada es `add_update_article()`. A continuación, se muestra el código para la función agregar o actualizar un artículo:


```

1 # Funci n para agregar o actualizar un art culo
2 def add_update_article():
3     itemId = itemIdVar.get()
4     cantidad = cantidadVar.get()
5     descripcion = descripcionVar.get()
6     marca = marcaVar.get()
7     observaciones = observacionesVar.get()
8     lugar = lugarVar.get()
9     condicion = condicionVar.get()
10    proveedor = proveedorVar.get()
11    anosServicio = anosServicioVar.get()
12    valorCompra = valorCompraVar.get()
13    hojaVidaPath = hojaVidaPathVar.get() # Obtener la ruta
        de la hoja de vida
14
15    if all([itemId, cantidad, descripcion, marca,
16           observaciones, lugar, condicion, proveedor,
17           anosServicio, valorCompra]):
18        try:
19            conn = connection()
20            cursor = conn.cursor()
21
22            # Verificar si el art culo ya existe
23            cursor.execute(f"SELECT * FROM articulos WHERE
24                           ID = '{itemId}'")
25            if cursor.fetchone():
26                # Si existe, actualiza
27                sql = f"""
28                    UPDATE articulos SET Cantidad='{cantidad}
29                        ', Descripcion='{descripcion}',
30                        Marca='{marca}',
31                        Observaciones='{observaciones}', Lugar
32                       ='{lugar}', Condicion='{condicion}',
33                        Proveedor='{proveedor}',
34                        A os_de_servicio={anosServicio},
35                        Valor_Compra={valorCompra},
36                        Hoja_de_vida='{hojaVidaPath}'
37                    WHERE ID = '{itemId}'
38                """
39                messagebox.showinfo("Success", "Article
40                                     updated successfully")
41            else:
42                # Si no existe, inserta
43                sql = f"""
44                    INSERT INTO articulos (ID, Cantidad,
45                        Descripcion, Marca, Fecha,
46                        Observaciones, Lugar, Condicion,
47                        Proveedor, A os_de_servicio,
48                        Valor_Compra, Hoja_de_vida)
49                    VALUES ('{itemId}', '{cantidad}', '{
50                        descripcion}', '{marca}',
51                        CURRENT_DATE, '{observaciones}', '{
52                        lugar}', '{condicion}', '{proveedor}
53                        ', {anosServicio}, {valorCompra}, '{
54                        hojaVidaPath}')

```



```

36         """
37         messagebox.showinfo("Success", "Article
38             added successfully")
39
40         cursor.execute(sql)
41         conn.commit()
42         conn.close()
43         query_article() # Refresh the table to show the
44             new entry
45     except Exception as e:
46         messagebox.showerror("Error", f"Error while
47             adding/updating article: {str(e)}")
48     else:
49         messagebox.showwarning("Input Error", "Please fill
50             all fields")

```

8.3 Botón de Limpiar Todo

El botón de **Limpiar Todo** borra el contenido de los campos de entrada y limpia la tabla de resultados. La función asociada es **clear_fields()**. A continuación, se muestra el código para esa función:

```

1 # Funci n para limpiar los campos
2 def clear_fields():
3     itemIdVar.set('')
4     cantidadVar.set('')
5     descripcionVar.set('')
6     marcaVar.set('')
7     observacionesVar.set('')
8     lugarVar.set('')
9     condicionVar.set('')
10    proveedorVar.set('')
11    anosServicioVar.set('')
12    valorCompraVar.set('')
13    hojaVidaPathVar.set('')
14    # Limpiar la tabla de resultados (Treeview)
15    my_tree.delete(*my_tree.get_children())

```

8.4 Botón de Eliminar

El botón de **Eliminar** permite borrar un artículo seleccionado en la tabla. La función asociada es **delete_article()**.

```

1 # Funci n para eliminar un art culo seleccionado
2 def delete_article():
3     selected_item = my_tree.selection() # Obtener la
4     selecci n actual
5     if selected_item:
6         item_id = my_tree.item(selected_item)['values'][0]
7         # Obtener el ID del art culo
8         confirmar_eliminar = messagebox.askyesno("Confirmar
9         Eliminaci n", f" Est seguro de que desea
10        eliminar el art culo con ID {item_id}?")
11
12    if confirmar_eliminar:
13        try:
14            conn = connection()
15            cursor = conn.cursor()
16            # Eliminar el art culo de la base de datos
17            cursor.execute(f"DELETE FROM articulos WHERE
18            ID = '{item_id}'")
19            conn.commit()
20            conn.close()
21
22            # Eliminar el art culo de la tabla visual
23            my_tree.delete(selected_item)
24            messagebox.showinfo(" xito ", f"Art culo
25            con ID {item_id} eliminado exitosamente."
26            )
27        except Exception as e:
28            messagebox.showerror("Error", f"No se pudo
29            eliminar el art culo: {str(e)}")
30    else:
31        messagebox.showwarning("Selecci n Vac a", "Por
32        favor, seleccione un art culo de la tabla para
33        eliminarlo.")

```

8.5 Botón de Salir

El botón de **Salir** cierra la aplicación después de mostrar una ventana de confirmación.

La función asociada es `exit_app()`.

```

1 # Funci n para salir de la aplicaci n
2 def exit_app():
3     # Mostrar el di logo de confirmaci n
4     confirmar_salida = messagebox.askyesno("Confirmar salida
5         ", " Ests seguro de que deseas salir?")
6
7     if confirmar_salida: # Si el usuario elige "S "
8         window.quit() # Cerrar la ventana principal

```

8.6 Abrir Hoja de Vida al Hacer Doble Clic

La aplicación permite abrir el archivo de hoja de vida al hacer doble clic en un artículo dentro de la tabla. Esta funcionalidad se implementa con la función `abrir_hoja_vida()`.

```

1 # Funci n para abrir archivo Excel desde la tabla
2 def abrir_hoja_vida(event):
3     selected_item = my_tree.selection() # Obtener la
4     selecci n actual
5     if selected_item:
6         hoja_vida_path = my_tree.item(selected_item)['values
7             '][11] # Obtener la ruta de la hoja de vida
8
9         if os.path.exists(hoja_vida_path): # Verificar si
10             el archivo existe
11             os.startfile(hoja_vida_path) # Abrir el archivo
12             Excel
13     else:
14         messagebox.showerror("Archivo no encontrado", f"
15             No se encontr la hoja de vida en: {
16             hoja_vida_path}")

```

Para que esta función se ejecute cuando el usuario haga doble clic en un artículo de la tabla, se añade el siguiente evento:

```

1 # Asignar el evento de doble clic para abrir la hoja de vida
2 my_tree.bind("<Double-1>", abrir_hoja_vida)

```

Esta configuración permite al usuario abrir un archivo Excel asociado al artículo haciendo doble clic en la fila correspondiente.

9. Resumen de los botones



Se han implementado los botones de la interfaz gráfica para las operaciones de búsqueda, agregar/actualizar, limpiar campos, eliminar y salir. Además, se ha añadido la funcionalidad de abrir el archivo Excel al hacer doble clic en un artículo.