Documentación de Inventario de aplicaciones

La aplicación se divide en db\_funcions.py donde se creó la clase Controller.

class DataBase:

    def \_\_init\_\_(self):

        self.connection = None

        try:

            self.connect()

            self.make\_table()

            messagebox.showinfo("CONEXION", "Base de Datos Creada exitosamente")

        except:

            print("Error al crear la base de datos")

    def connect(self):

        self.connection = sqlite3.connect("mibase.db")

    def make\_table(self):

        self.connect()

        cursor = self.connection.cursor()

        sql = """CREATE TABLE aplicaciones

                (id INTEGER PRIMARY KEY AUTOINCREMENT,

                nombre varchar(100) NOT NULL,

                tipo varchar(20) NOT NULL,

                nivel int NOT NULL,

                ruta varchar(200) NOT NULL,

                descripcion varchar(500) NOT NULL)

        """

        cursor.execute(sql)

        self.connection.commit()

Inventario\_aplicaciones.py es el main de la aplicación.

class Controller:

    def \_\_init\_\_(self, root ):

        self.root\_controller = root

        self.vista\_inventario = VistaInventario(self.root\_controller)

if \_\_name\_\_ == '\_\_main\_\_':

    root = Tk()

    Controller(root)

    root.mainloop()

el modelo.py en donde se crea la clase Abmc. Dentro de ella encontraran que se instancia db.

class Abmc:

    def \_\_init\_\_(self):

        self.db = DataBase()

en dicha clase están los métodos Alta(), consultar(), limpiar(),modificar(),borrar(), actualizar\_treeview() y buscar\_una\_app()

alta()

def alta(self, id, nombre: str, tipo: str, nivel, ruta: str, descripcion: str, tree, boton\_alta):

        errores: str = ''

        #Llamamos a los metodos estaticos

        errores += Utils.valid\_string('Nombre de la Aplicación', nombre)

        errores += Utils.valid\_int('Nivel de Riesgo', str(nivel))

        if errores == '':

            #Hacer el Alta o la Modificación

            if boton\_alta['text'] == "Actualizar":

                cursor = self.db.connection.cursor()

                data = ( nombre, tipo, nivel, ruta, descripcion, id)

                sql = "UPDATE aplicaciones SET nombre = ?, tipo = ? , nivel = ?, ruta = ?, descripcion = ? WHERE id = ?"

                cursor.execute(sql, data)

                self.db.connection.commit()

                messagebox.showinfo("Modificación", "Se actualizó exitosamente")

                self.actualizar\_treeview(tree)

            else:

                cursor = self.db.connection.cursor()

                data = (nombre, tipo, nivel, ruta, descripcion)

                sql = "INSERT INTO aplicaciones (nombre, tipo, nivel, ruta, descripcion) VALUES(?, ?, ?, ?, ?)"

                cursor.execute(sql, data)

                self.db.connection.commit()

                messagebox.showinfo("Alta", "Se guardó exitosamente")

                self.actualizar\_treeview(tree)

        else:

            messagebox.showinfo("Alta", errores)

consultar()

 def consultar(self, tree):

        self.actualizar\_treeview(tree)

limpiar()

def limpiar(self, tree, boton\_alta, id\_val, nombre\_val, tipo\_val, nivel\_val, ruta\_val, descripcion\_val):

        id\_val.set(0)

        nombre\_val.set('')

        tipo\_val.set('')

        nivel\_val.set('')

        ruta\_val.set('')

        descripcion\_val.set('')

        boton\_alta['text'] = "Agregar"

modificar()

def modificar(self, tree, boton\_alta, id\_val, nombre\_val, tipo\_val, nivel\_val, ruta\_val, descripcion\_val):

        valor = tree.selection()

        item = tree.item(valor)

        my\_id = item['text']

        app\_sel = self.buscar\_una\_app(my\_id)

        id\_val.set(my\_id)

        nombre\_val.set(app\_sel[1])

        tipo\_val.set(app\_sel[2])

        nivel\_val.set(app\_sel[3])

        ruta\_val.set(app\_sel[4])

        descripcion\_val.set(app\_sel[5])

        boton\_alta['text'] = "Actualizar"

Se crea la Clase Abmc con su constructor

class Abmc:

    def \_\_init\_\_(self):

        self.db = DataBase()

borrar()

def borrar(self, tree):

        valor = tree.selection()

        item = tree.item(valor)

        mi\_id = item['text']

        if messagebox.askokcancel("Atencion", f"Está seguro que desea eliminar la Aplicacion con ID: {mi\_id}"):

            cursor = self.db.connection.cursor()

            # mi\_id = int(mi\_id)

            data = (mi\_id,)

            sql = "DELETE FROM aplicaciones WHERE id = ?;"

            cursor.execute(sql, data)

            self.db.connection.commit()

            tree.delete(valor)

            messagebox.showinfo("Atención", f"La aplicación {mi\_id} se eliminó correctamente")

actualizar\_treeview()

  def actualizar\_treeview(self, mitreview):

        records = mitreview.get\_children()

        for element in records:

            mitreview.delete(element)

        sql = "SELECT \* FROM aplicaciones ORDER BY id ASC"

        cursor = self.db.connection.cursor()

        datos = cursor.execute(sql)

        resultado = datos.fetchall()

        for fila in resultado:

            mitreview.insert("", 0, text=fila[0], values=(fila[1], fila[2], fila[5]))

buscar\_una\_app()

    def buscar\_una\_app( self, id ):

        sql = f"SELECT \* FROM aplicaciones WHERE id = {id}"

        cursor = self.db.connection.cursor()

        datos = cursor.execute(sql)

        resultado = datos.fetchone()

        return resultado

Se crea la clase Utils.py con tres métodos estáticos de validación

class Utils:

    #Se agrega como metodo estatico ya que no será necesario crear un objeto, sino que usamos directamente el metodo.

    @staticmethod

    def valid\_string(nombre\_campo: str, cadena: str) -> str:

        patron\_string = "^[A-Za-záéíóú]\*$"

        if not re.match(patron\_string, cadena):

            return f"El {nombre\_campo} tiene caracteres inválidos. Solo se pueden ingresar letras de la a la z \n"

        else:

            return ''

    @staticmethod

    def valid\_int(nombre\_campo: str, cadena: str) -> str:

        patron\_num = '^([0-9])\*$'

        if not re.match(patron\_num, cadena):

            return f"El {nombre\_campo} tiene caracteres inválidos. Solo se pueden números \n"

        else:

            return ''

    @staticmethod

    def validate\_entry(text, new\_text):

        # Primero chequear que el contenido total no exceda los dos caracteres.

        if len(new\_text) > 2:

            return False

        # Luego, si la validación anterior no falló, chequear que el texto solo contenga nros

        return text.isdecimal()

Se crea la vista.py

class VistaInventario:

    def \_\_init\_\_(self, window):

        self.root = window

        # Creo el objeto ABMC

        self.abmc = Abmc()

# Título, tamaño y color de la ventana principal

        self.root.title("Inventario de Aplicaciones")

        self.root.geometry("560x485")

        self.root.config(bg="#FFFDFD")

        # Título del cuadro principal

        titulo = Label(self.root, text="Ingrese los datos de la Aplicación", bg="#76B6E3", fg="White", height=1,

                    width=80)

        # Título del cuadro detalle

        titulo\_detalle = Label(self.root, text="Listado de Aplicaciones", bg="#76B6E3", fg="White", height=1, width=80)

        # Defino variables para tomar valores de campos de entrada

        id\_val, nombre\_val, tipo\_val = IntVar(), StringVar(), StringVar()

        nivel\_val, ruta\_val, descripcion\_val = IntVar(), StringVar(), StringVar()

        w\_ancho = 50

        # Etiquetas / Labels:

        id = Label(self.root, text="ID", bg="#FFFDFD")

        nombre = Label(self.root, text="Nombre", bg="#FFFDFD")

        tipo = Label(self.root, text="Tipo", bg="#FFFDFD")

        nivel = Label(self.root, text="Nivel de Riesgo", bg="#FFFDFD")

        ruta = Label(self.root, text="Ruta del Log", bg="#FFFDFD")

        descripcion = Label(self.root, text="Descripción", bg="#FFFDFD")

        # Campos de ingreso de información / Entries:

        entrada\_id = Entry(self.root, textvariable=id\_val, state="readonly", width=10)

        entrada\_nombre = Entry(self.root, textvariable=nombre\_val, width=w\_ancho)

        entrada\_tipo = Entry(self.root, textvariable=tipo\_val, width=w\_ancho)

        #utlizamos el metodo estatico validate\_entry para realizar las validaciones de regex

        entrada\_nivel = Entry(self.root, textvariable=nivel\_val, width=w\_ancho, validate="key",

                            validatecommand=(self.root.register(Utils.validate\_entry), "%S", "%P")

                            )

        entrada\_ruta = Entry(self.root, textvariable=ruta\_val, width=w\_ancho)

        entrada\_descripcion = Entry(self.root, textvariable=descripcion\_val, width=w\_ancho)

        # Botones / Buttons:

        boton\_alta = Button(

            self.root,

            text="Agregar",

            command=lambda: self.abmc.alta(

                id\_val.get(),

                nombre\_val.get(),

                tipo\_val.get(),

                nivel\_val.get(),

                ruta\_val.get(),

                descripcion\_val.get(),

                tree, boton\_alta)

        )

        boton\_consulta = Button(self.root, text="Consultar", command=lambda: self.abmc.consultar(tree))

        boton\_borrar = Button(self.root, text="Borrar", command=lambda: self.abmc.borrar(tree))

        boton\_modificar = Button(self.root, text="Modificar",

                                command=lambda: self.abmc.modificar(tree, boton\_alta, id\_val, nombre\_val, tipo\_val,

                                                                    nivel\_val, ruta\_val, descripcion\_val))

        boton\_limpiar = Button(self.root, text="Limpiar campos",

                            command=lambda: self.abmc.limpiar(tree, boton\_alta, id\_val, nombre\_val, tipo\_val,

                                                                nivel\_val, ruta\_val, descripcion\_val))

        boton\_salir = Button(self.root, text="Salir", command=self.root.quit)

Creación del treeview para el formulario de Inventario.

tree = ttk.Treeview(self.root)

        # Posición de etiquetas, campos de entrada y botones /Labels, Entries, Buttons

        # Título

        titulo.grid(row=0, column=0, columnspan=6, padx=1, pady=1, sticky=W + E)

        # Labels

        id.grid(row=1, column=1)

        nombre.grid(row=2, column=1, sticky=E)

        tipo.grid(row=3, column=1, sticky=E)

        nivel.grid(row=4, column=1, sticky=E)

        ruta.grid(row=5, column=1, sticky=E)

        descripcion.grid(row=6, column=1, sticky=E)

        # Entries

        entrada\_id.grid(row=1, column=2)

        entrada\_nombre.grid(row=2, column=2)

        entrada\_tipo.grid(row=3, column=2)

        entrada\_nivel.grid(row=4, column=2)

        entrada\_ruta.grid(row=5, column=2)

        entrada\_descripcion.grid(row=6, column=2)

        # Buttons

        boton\_alta.grid(row=7, column=2, sticky=EW)

        boton\_limpiar.grid(row=3, column=3, sticky=EW)

        boton\_borrar.grid(row=15, column=1, sticky=EW)

        boton\_modificar.grid(row=15, column=2, sticky=EW)

        boton\_salir.grid(row=15, column=4, sticky=EW)

        # Título Detalle

        titulo\_detalle.grid(row=8, column=0, columnspan=6, padx=1, pady=1, sticky=W + E)

        boton\_consulta.grid(row=9, columnspan=1, column=1, sticky=EW)

        # Árbol de información

        tree.grid(row=10, column=1, columnspan=4)

        tree["columns"] = ("col1", "col2", "col3")

        tree.column("#0", width=90, minwidth=50, anchor=W)

        tree.column("col1", width=100, minwidth=80)

        tree.column("col2", width=50, minwidth=80)

        tree.column("col3", width=250, minwidth=80)

        tree.heading("#0", text="ID")

        tree.heading("col1", text="Nombre")

        tree.heading("col2", text="Tipo")

        tree.heading("col3", text="Descripción")