Manual técnico del aplicativo sena.exe
Realizado por Iván Casallas y Alexandra Martinez

Fase preliminar, instalaciones requeridas en el sistema.

El aplicativo SENA.EXE el cual corre el software que controla datos de contratistas del SENA está escrito en las tecnologías de Python y toda la biblioteca tkinter que corresponde a la biblioteca propia para el desarrollo de GUI, siendo esta esencial para que corran los aplicativos de escritorio.

1. Para hacer uso del codigo es necesario la instalación de Python en el equipo y en el IDLE correspondiente (Visual Code Studio u otro programa).
Para ello, se requiere abrir Microsoft Store y buscar Python en su versión más nueva para proceder a su instalación.

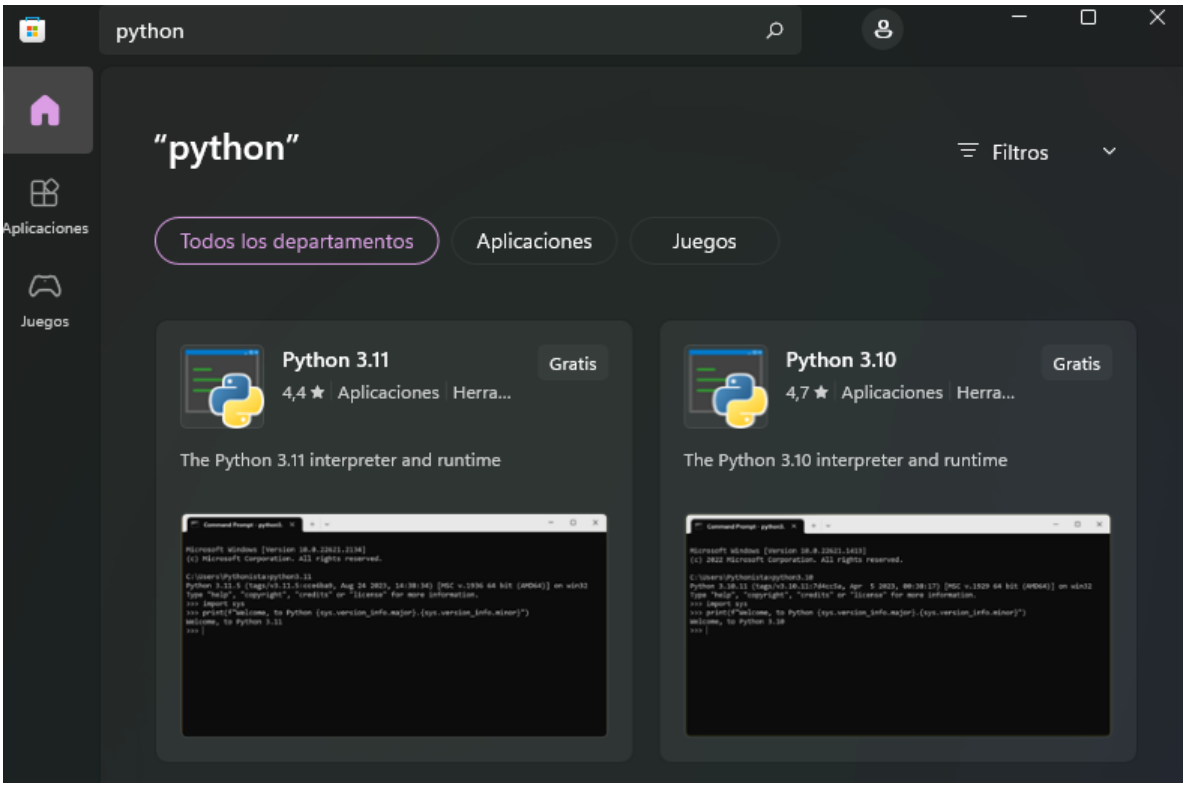


Imagen 1. Búsqueda de Python en la Store de Microsoft. Se requiere instalar

Python 3.11 en este caso. Seguir las instrucciones de instalación hasta finalizar el proceso.

A continuación, descargar el IDLE de Python para el Visual Code Studio. En la barra lateral de Iconos elegir el ícono de extensión con el siguiente dibujo

Imagen 2. Ícono de extensiones en la que se puede elegir el IDLE de Python.

O escribir en la barra superior de búsqueda el idle necesario.

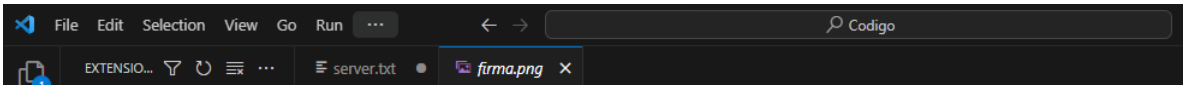


Imagen 3. Barra de búsqueda para encontrar el IDLE de Python.

Seguido de eso instalar el IDLE que aparece enseguida

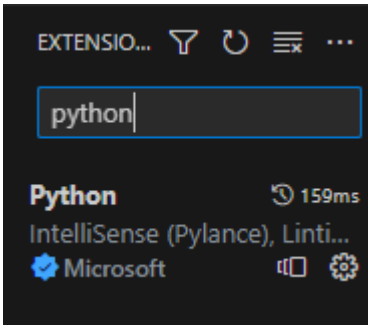


Imagen 4. IDLE de Python, instalar.

2. Tras la instalación de Python se requiere hacer correr las bibliotecas en el sistema, para ello es necesario abrir la terminar del Sistema.
Esto se realiza oprimiendo las techas [INICIO] + [R] y escribir en la ventana emergente la siguiente palabra

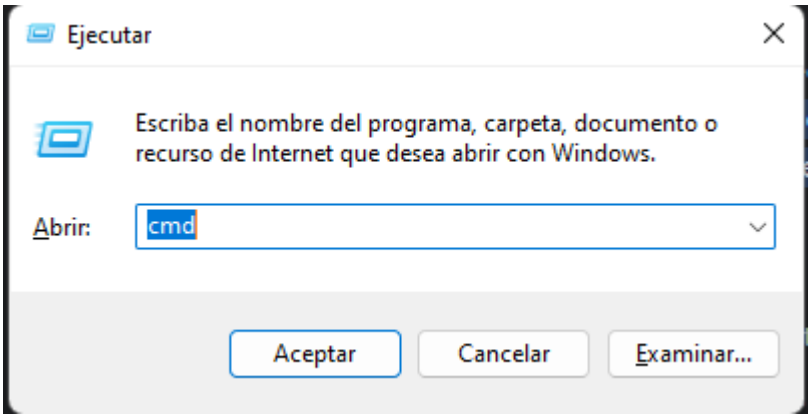


Imagen5. Ejecutor de Windows, Escribir CMD y presionar Aceptar.

Al abrir la terminar de Windows se requiere instalar las librerías que aparecen en el código fuente del aplicativo. En la carpeta Código, en el archivo Programa.py.

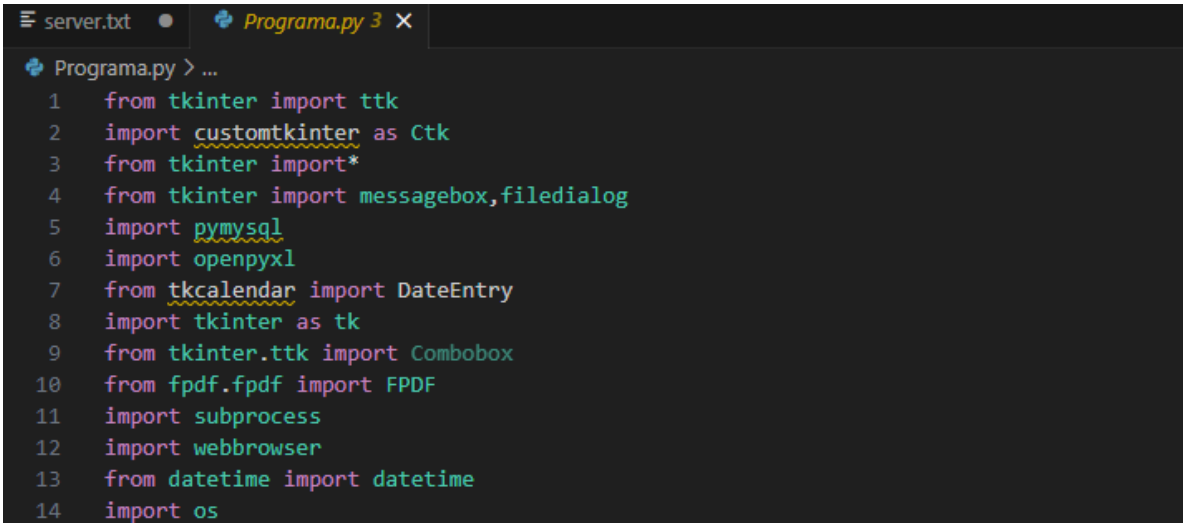


Imagen 6. Lista de librerías utilizadas en el archivo .exe

La instalación se realiza en la terminal escribiendo PIP INSTALL [nombre de la biblioteca] y la tecla enter para hacer correr la orden. Un ejemplo de instalación es

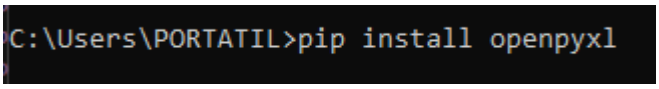


Imagen 7. Ejemplo de instalación de la librería openpyxl.

Tras poner enter, se debe iniciar el proceso de instalación de la librería, si ya se encuentra instalada debe correr el sistema de instalación y luego decir que se completó la instalación o la actualización del módulo como se muestra a

continuación.

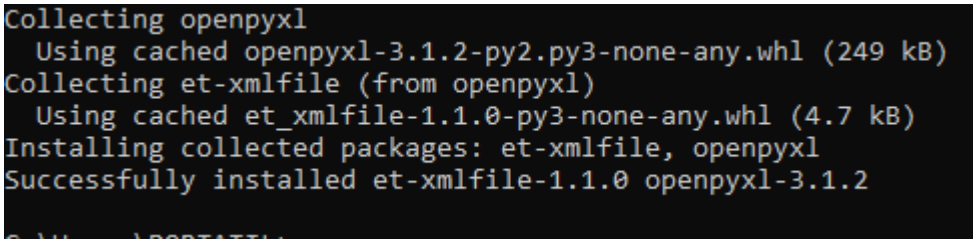


Imagen 8. Código de instalación de la librería openpyxl finalizado.

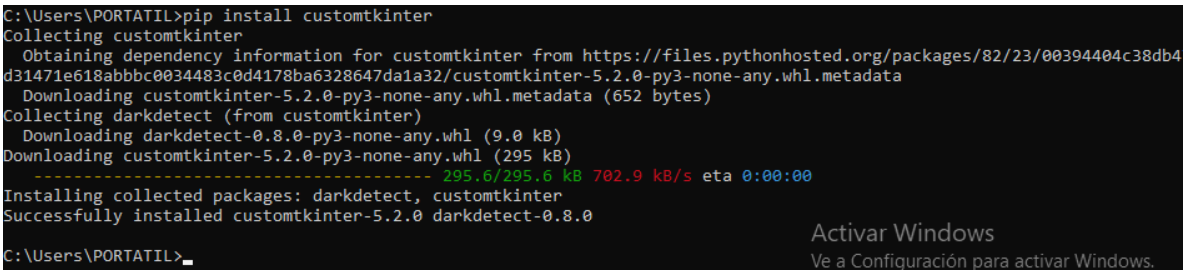


Imagen 9. Instalación de Librería Customtkinter. Inicia con pip install Customtkinter y se ejecuta hasta finalizar, se ve mensaje final que dice "successfully installed…"

Ordenar la instalación de las siguientes librerías en el orden requerido:

1. pip install customtkinter
2. pip install pymysql
3. pip install openpyxl
4. pip install tkacalendar
5. pip install FPDF
6. pip install datetime

Fase de explicación carpetas y código del aplicativo.

El programa se encuentra contenido en una carpeta de nombre Sena, en ella se encuentran 6 subcarpetas y 2 archivos.

Subcarpetas

1. **__pycache__**

   Es un directorio que Python crea en tu proyecto cuando ejecutas un script. Esta carpeta contiene archivos .pyc , que son versiones compiladas de tus

scripts de Python. No se debe eliminar ni alterar nada en esta carpeta, ya que ella compila el funcionamiento interno de todo Script creado a lo largo del código.

2. **Babel**

   Es otro directorio en el que se establecen datos locales, mensajes y protocolos de cualquier script, se encripta la librería utilizada y requerimientos técnicos, es una carpeta parecida a __pycache__ y en ella se encuentra una subcarpeta de __pycache__. No se debe eliminar ni alterar ningún archivo de esta carpeta.

3. **Certificados**

   Carpeta que contiene los certificados descargados de los contratistas desde el funcionamiento del aplicativo, en estado inicial se encuentra vacío. A medida que se descarguen certificados estos quedan guardados en la carpeta con el número de documento del contratista.

4. **Fpdf**

   Esta carpeta contiene archivos y subcarpetas que se generan automáticamente al instalar la librería y hacer correr la aplicación con el fin de generar archivos .pdf

   __**init**__**.py** es el archivo que contiene el arranque de la librería y otros scripts. **(NO BORRAR NADA NI ALTERAR NADA)**

   Fonts.py contiene las características de fuentes y tipado para la generación de archivos .pdf. **(NO BORRAR NADA NI ALTERAR NADA)**

   fpdf.py este archivo contiene la versión y la información total de la librería, hace correr los fonts y el init. **(NO BORRAR NADA NI ALTERAR NADA)**

   html.py contiene la funcionalidad de posición y protocolos de scripts propios de la librería. **(NO BORRAR NADA NI ALTERAR NADA)**

   Php.py contiene los archivos que permiten codificar posición y protocolos para la creación de archivos .pdf para lenguajes php y la interpretación**(NO BORRAR NADA NI ALTERAR NADA)**

   Py3k.py Contiene el módulo que difiere entre diferentes situaciones de protocolos de Python importando partes de sistema y generando los

aciertos a errores y codificados. **(NO BORRAR NADA NI ALTERAR NADA)**

Temprate.py Archivo que sustenta y ayuda a la librería Fpdf.
(NO BORRAR NADA NI ALTERAR NADA)

Ttfonts.py en este archivo se contiene diferentes fuentes que interactúan en la librería **(NO BORRAR NADA NI ALTERAR NADA)**

5. **Img**

   En esta subcarpeta se contiene las imágenes que operan en la función de creación de los archivos .pdf, estas firmas pueden ser modificables según los nuevos requerimientos del sistema. **ADVERTENCIA:** Modificar puede alterar el resultado de los archivos .pdf

   Adicional a las firmas e imágenes necesarias para la creación de archivos .pdf contiene dos archivos .ico que funcionan como íconos del sistema. Uno de ellos es del ejecutable .EXE este corresponde al de SENA.ico

6. **Server**

   Server.txt es un archivo de protocolo que está emparentado para hacer correr la configuración de IP del Servidor, Usuario, contraseña y Base de Datos que requiere el sistema **(NO BORRAR NADA NI ALTERAR NADA, UNICAMENTE SI DESEA MODIFICAR LA IP QUE ES LA QUE SE CONECTA AL SERVIDOR PARA USAR LA BASE DE DATOS.)**

Archivos .py que se encuentran fuera de las carpetas

Programa.py

En este archivo se encuentra toda la operatividad del aplicativo después de iniciar sesión para que corra, este archivo contiene 955 líneas de código.

```
Programa.py > Contratista > __init__
1   from tkinter import ttk
2   import customtkinter as Ctk
3   from tkinter import*
4   from tkinter import messagebox,filedialog
5   import pymysql
6   import openpyxl
7   from tkcalendar import DateEntry
8   import tkinter as tk
9   from tkinter.ttk import Combobox
10  from fpdf.fpdf import FPDF
11  import subprocess
12  import webbrowser
13  from datetime import datetime
14  import os
```

las primeras 14 líneas son parte de los procesos e importaciones que requiere el programa de las librerías internas e instalables para que opere

```
15  Ctk.set_appearance_mode("system")  # Modes: system (default), light, dark
16  Ctk.set_default_color_theme("blue")  # Themes: blue (default), dark-blue, green
17  current_date1 = datetime.now().strftime("%d")
18  current_date2 = datetime.now().strftime("%m")
19  current_date3 = datetime.now().strftime("%Y")
20
```

las siguientes líneas permiten la creación de características de color y que corra la librería datatime que permite ver la fecha y hora.

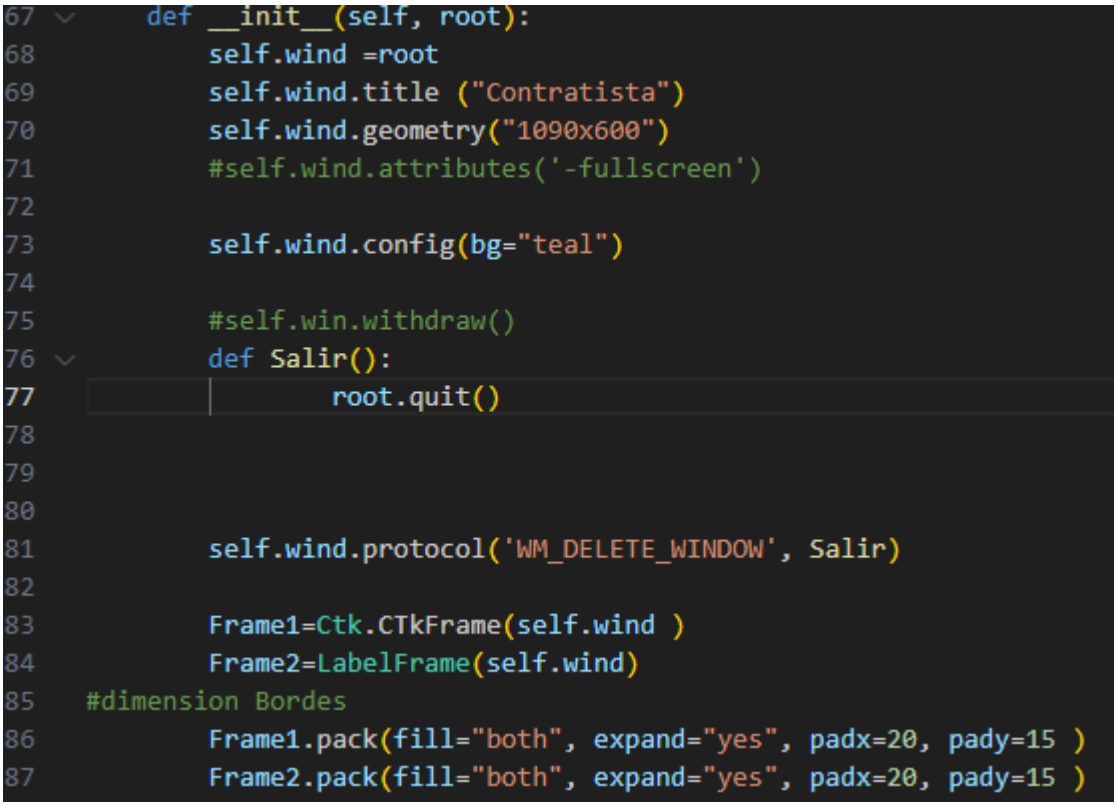
```
class PDF(FPDF):

    def header(self):
        # Logo
        self.set_font('Arial', '', 10)
        self.ln(3)
        self.multi_cell(align="J",h=15,w=0,txt="Certificación")
        self.image('./img/logo.jpg', x = 100, y = 10, w = 15, h = 15)
        self.ln(3)
```

En la línea 23 del código se inicia la clase PDF en la que se establecen los limites y todo lo que debe aparecer en los archivos .pdf que son creados por el aplicativo según el contratista requerido. Esta clase contiene los headers, en el que está el

logo, el footer, en la línea 44.

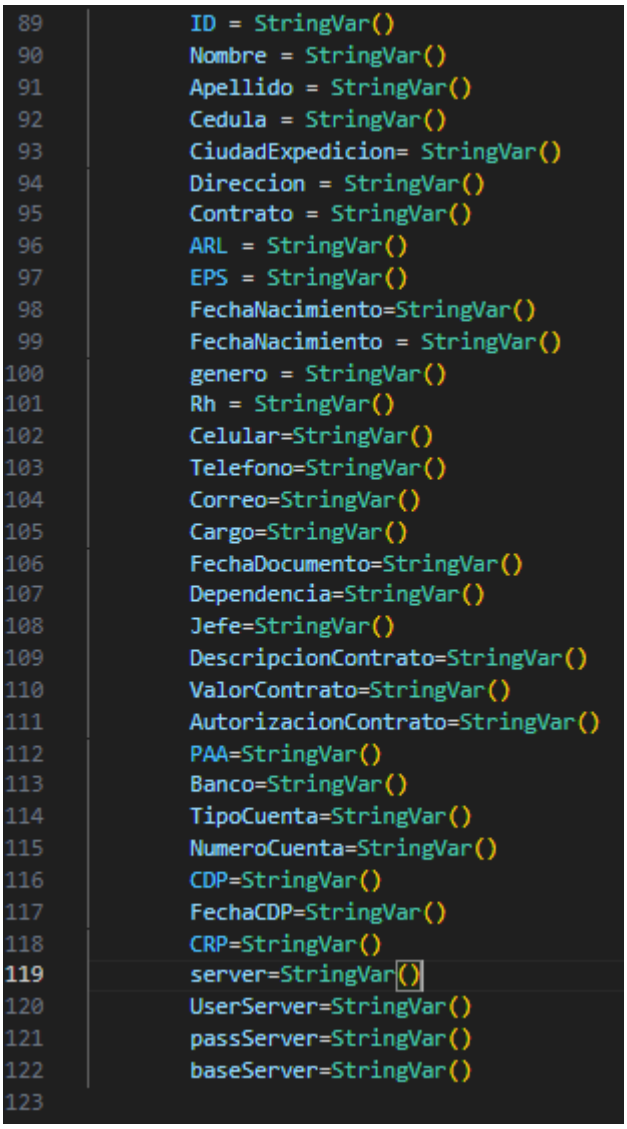
```
class Contratista:

    def __init__(self, root):
        self.wind =root
        self.wind.title ("Contratista")
        self.wind.geometry("1090x600")
        #self.wind.attributes('-fullscreen')

        self.wind.config(bg="teal")

        #self.win.withdraw()
        def Salir():
                root.quit()



        self.wind.protocol('WM_DELETE_WINDOW', Salir)

        Frame1=Ctk.CTkFrame(self.wind )
        Frame2=LabelFrame(self.wind)
```

en la línea 65 se crea la clase Contratista que contiene los principales módulos del aplicativo. Esta clase puede ser tomada como la clase principal y está el funcionamiento particular de todo el proceso interno del aplicativo. Las siguientes líneas de código serán explicadas para entender la clase Contratistas en su totalidad.



```
67      def __init__(self, root):
68          self.wind =root
69          self.wind.title ("Contratista")
70          self.wind.geometry("1090x600")
71          #self.wind.attributes('-fullscreen')
72
73          self.wind.config(bg="teal")
74
75          #self.win.withdraw()
76          def Salir():
77                  root.quit()
78
79
80
81          self.wind.protocol('WM_DELETE_WINDOW', Salir)
82
83          Frame1=Ctk.CTkFrame(self.wind )
84          Frame2=LabelFrame(self.wind)
85  #dimension Bordes
86          Frame1.pack(fill="both", expand="yes", padx=20, pady=15 )
87          Frame2.pack(fill="both", expand="yes", padx=20, pady=15 )
```

Crea la ventana y permite correr toda la ventana así como sus dimensiones y el cierre necesario.



```
89          ID = StringVar()
90          Nombre = StringVar()
91          Apellido = StringVar()
92          Cedula = StringVar()
93          CiudadExpedicion= StringVar()
94          Direccion = StringVar()
95          Contrato = StringVar()
96          ARL = StringVar()
97          EPS = StringVar()
98          FechaNacimiento=StringVar()
99          FechaNacimiento = StringVar()
100         genero = StringVar()
101         Rh = StringVar()
102         Celular=StringVar()
103         Telefono=StringVar()
104         Correo=StringVar()
105         Cargo=StringVar()
106         FechaDocumento=StringVar()
107         Dependencia=StringVar()
108         Jefe=StringVar()
109         DescripcionContrato=StringVar()
110         ValorContrato=StringVar()
111         AutorizacionContrato=StringVar()
112         PAA=StringVar()
113         Banco=StringVar()
114         TipoCuenta=StringVar()
115         NumeroCuenta=StringVar()
116         CDP=StringVar()
117         FechaCDP=StringVar()
118         CRP=StringVar()
119         server=StringVar()
120         UserServer=StringVar()
121         passServer=StringVar()
122         baseServer=StringVar()
123
```

Desde la línea 89 y hasta la 122 se definen las variables que operan dentro de los campos del formulario en el aplicativo, todas estas variables funcionan a lo largo del código para realizar los llamados y las interacciones.

```
129   ##funcion para llamar IP conexion varios equipos REEMPLAZAR localhost
130
131
132
133           def cargarServer():
134                   if os.path.isfile("./server/server.txt"):
135                           fileser=open("./server/server.txt","r")
136                           contador = 0
137                           for linea in fileser:
138                                   if contador ==0:
139                                           server.set(linea.strip())
140                                           print(server.get())
141                                   elif contador ==1:
142                                           UserServer.set(linea.strip())
143                                           print(UserServer.get())
144                                   elif contador ==2:
145                                           passServer.set(linea.strip())
146                                           print(passServer.get())
147                                   elif contador ==3:
148                                           baseServer.set(linea.strip())
149                                           print(baseServer.get())
150                                   elif contador >= 4 :
151                                           break
152                                   contador += 1
```

Esta parte del código hace correr el Server definido en la subcarpeta Server de la carpeta principal, este protocolo lo que hace es leer la información del archivo y procesarla para que pase oculta y no sea visible para toda persona.

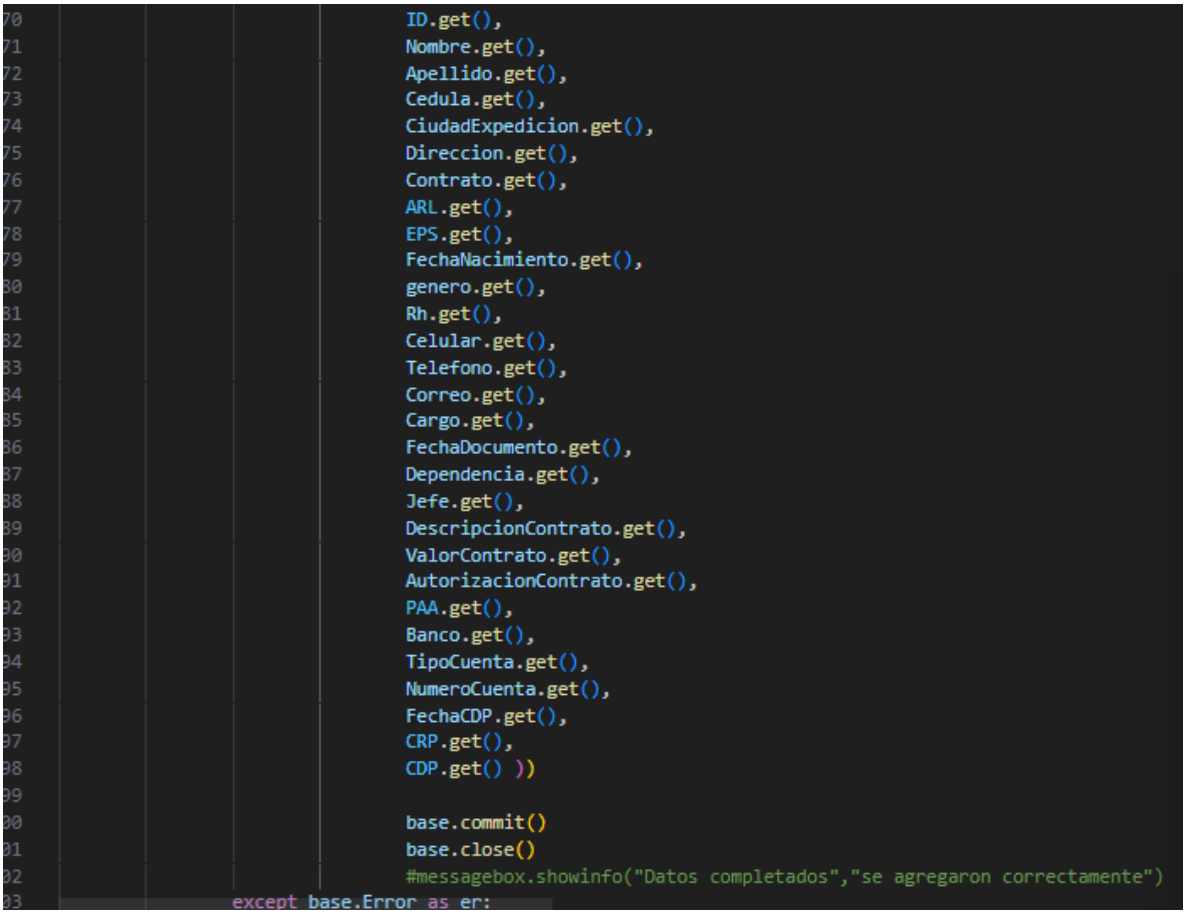
```
154   # FUNCIONES DE LOS BOTONEaS
155
156       def Agregar():
157               try:
158
159
160                   if ID.get() == "" or Nombre.get() == "" or Apellido.get() == "" or Cedula.get() == ""or CiudadExpedicion.get() == ""or Direccion.get()=="" or Contrato.ge
161                           messagebox.showerror("Por favor", "Ingresa la información correcta")
162
163
164                   else:
165
166                           base= pymysql.connect(host=server.get(),user=UserServer.get(), password=passServer.get(), database=baseServer.get())
167                           cursor =base.cursor()
168                           cursor.execute("insert into cliente values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",(
169
```

la línea 156 inicia la función agregar del CRUD.

en esta se inicia por llamar los campos requeridos en la línea 160. E insertar los valores establecidos en la función, realizando la unión con la base de datos. Esto se encuentra entre las líneas 166 a 169. En la línea 169 se debe tener siembre los valores a insertar iguales a las funciones llamadas y a modificar dentro de la base de datos. Si por ejemplo se tiene, 3 variables llamadas se debe modificar 3 valores también.

```
168                   cursor.execute("insert into cliente values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",(
```

En esta imagen se ve como se tiene 27 valores a modificar y 27 variables llamadas en las siguientes líneas de código de la función agregar.



```
                    ID.get(),
                    Nombre.get(),
                    Apellido.get(),
                    Cedula.get(),
                    CiudadExpedicion.get(),
                    Direccion.get(),
                    Contrato.get(),
                    ARL.get(),
                    EPS.get(),
                    FechaNacimiento.get(),
                    genero.get(),
                    Rh.get(),
                    Celular.get(),
                    Telefono.get(),
                    Correo.get(),
                    Cargo.get(),
                    FechaDocumento.get(),
                    Dependencia.get(),
                    Jefe.get(),
                    DescripcionContrato.get(),
                    ValorContrato.get(),
                    AutorizacionContrato.get(),
                    PAA.get(),
                    Banco.get(),
                    TipoCuenta.get(),
                    NumeroCuenta.get(),
                    FechaCDP.get(),
                    CRP.get(),
                    CDP.get() ))

                    base.commit()
                    base.close()
                    #messagebox.showinfo("Datos completados","se agregaron correctamente")
            except base.Error as er:
```

Llamado de las funciones a introducirle datos. Parte de la función de agregar.

```
211         def Limpiar():
212                 self.entID.delete(0, END)
213                 self.entNombre.delete(0, END)
214                 self.entApellido.delete(0, END)
215                 self.entCedula.delete(0,END)
216                 self.entCiudadExpedicion.delete(0,END)
217                 self.entDireccion.delete(0, END)
218                 self.entContrato.delete(0, END)
219                 self.entARL.delete(0, END)
220                 self.entEPS.delete(0, END)
221                 self.entFechaNacimiento.delete(0,END)
222                 self.entgenero.delete(0,END)
223                 self.entRh.delete(0,END)
224                 self.entCelular.delete(0,END)
225                 self.entTelefono.delete(0,END)
226                 self.entCorreo.delete(0,END)
227                 self.entCargo.delete(0,END)
228                 self.entFechaDocumento.delete(0,END)
229                 self.entDependencia.delete(0,END)
230                 self.entJefe.delete(0,END)
231                 self.entDescripcionContrato.delete(0,END)
232                 self.entValorContrato.delete(0,END)
233                 self.entAutorizacionContrato.delete(0,END)
234                 self.entPAA.delete(0,END)
235                 self.entBanco.delete(0,END)
236                 self.entTipoCuenta.delete(0,END)
237                 self.entNumeroCuenta.delete(0,END)
238                 self.entCDP.delete(0,END)
239                 self.entFechaCDP.delete(0,END)
240                 self.entCRP.delete(0,END)
```

La función limpiar permite limpiar los campos de el formulario. Esta función está relacionada con las cajas de texto que se encuentran desde la línea 672 más abajo en el código, en cada línea de texto de cada label existente por variable, debe estar relacionada una línea de código de la función limpiar para que esta misma pueda correr.

El ejemplo de concordancia que debe existir se muestra en la siguiente imagen tomando de ejemplo el label nombre y la variable nombre.

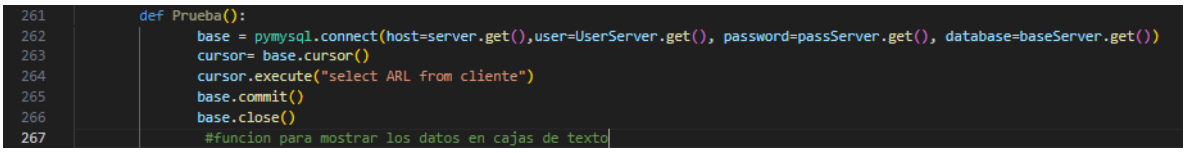
```
679         lbl2 = Ctk.CTkLabel(Frame1, text= "Nombre", width=20)
680         lbl2.grid(row=1, column=0,padx=5, pady=3,sticky="e")
681         self.entNombre = Ctk.CTkEntry(Frame1,border_color="teal", textvariable=Nombre)
682         self.entNombre.grid(row=1,column=1, padx=5, pady=3)
683         self.entNombre.bind("<Return>", lambda event: BuscarNombre(1))
684
```

La función de la linea 213 es llamada en las líneas 681 hasta la 683 para que la función tenga efecto y limpie los campos del formulario que estén llenos de información. Así respectivamente con cada label de las cajas de texto que esté

relacionado con una variable y con una función de limpiar.

```
def Mostrar():
    try:
        base = pymysql.connect(host=server.get(),user=UserServer.get(), password=passServer.get(), database=baseServer.get())
        cursor= base.cursor()
        cursor.execute("select * from cliente")
        result = cursor.fetchall()

        if len(result) !=0:
            self.trv.delete(*self.trv.get_children())
            for row in result:
                self.trv.insert('',END,values= row)
        base.commit()
        base.close()
    except pymysql.Connection.Error as er:
        codigo_error_extendido = er.args[0]
        messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )

```

La función mostrar permite traer la información de la base de datos y mostrarla en el lugar especificado para tal fin. En el aplicativo, la información traída se ve en la parte inferior de la aplicación.



```
    def Prueba():
        base = pymysql.connect(host=server.get(),user=UserServer.get(), password=passServer.get(), database=baseServer.get())
        cursor= base.cursor()
        cursor.execute("select ARL from cliente")
        base.commit()
        base.close()
         #funcion para mostrar los datos en cajas de texto
```

la función prueba permite hacer pruebas en los datos de las cajas de label. Se tiene por si uno busca errores y facilita la operatividad del código sin modificar

muchos campos.

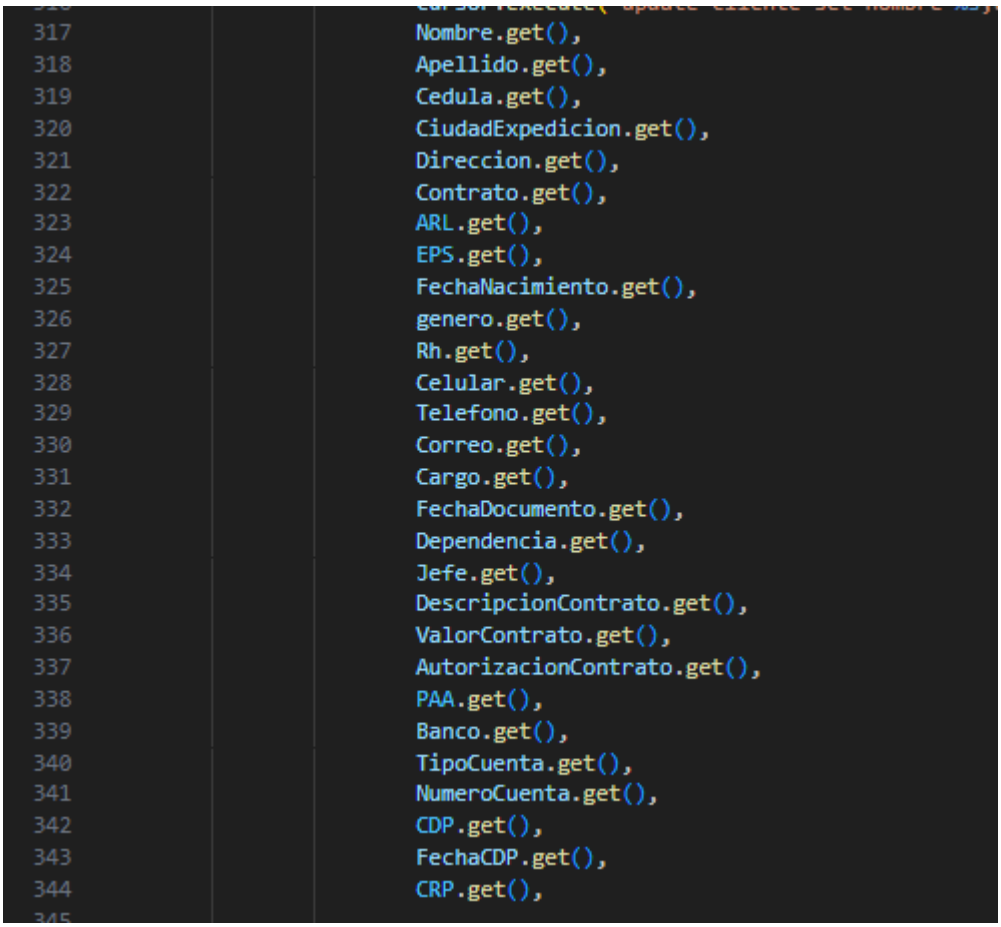
```
270         def traineeInfo(ev):
271                 viewInfo = self.trv.focus()
272                 learnerData = self.trv.item(viewInfo)
273                 row = learnerData['values']
274                 ID.set(row[0])
275                 Nombre.set(row[1])
276                 Apellido.set(row[2])
277                 Cedula.set(row[3])
278                 CiudadExpedicion.set(row[4])
279                 Direccion.set(row[5])
280                 Contrato.set(row[6])
281                 ARL.set(row[7])
282                 EPS.set(row[8])
283                 FechaNacimiento.set(row[9])
284                 genero.set(row[10])
285                 Rh.set(row[11])
286                 Celular.set(row[12])
287                 Telefono.set(row[13])
288                 Correo.set(row[14])
289                 Cargo.set(row[15])
290                 FechaDocumento.set(row[16])
291                 Dependencia.set(row[17])
292                 Jefe.set(row[18])
293                 DescripcionContrato.set(row[19])
294                 ValorContrato.set(row[20])
295                 AutorizacionContrato.set(row[21])
296                 PAA.set(row[22])
297                 Banco.set(row[23])
298                 TipoCuenta.set(row[24])
299                 NumeroCuenta.set(row[25])
300                 CDP.set(row[26])
301                 FechaCDP.set(row[27])
302                 CRP.set(row[28])
303 
304 
```

La función TraerInfo permite llenar los campos del formulario con la información que se encuentra en la base de datos. esta función facilita ver toda la información de un contratista en los formularios para modificarla, eliminarla o agregar nuevos

campos.

La función actualizar es parte fundamental del CRUD y opera de forma parecida a la función Agregar. En e IF se busca establecer la relación de las variables a modificar y en el ELSE se busca actualizar los campos solicitados en la base de datos.

debe haber una concordancia entre las variables llamadas y los datos a modificar entre la línea 315 y la 345



```
365         def Eliminar():
366                 delete=messagebox.askquestion("Eliminar","Realmente Deseas Eliminar el Registro")
367                 if delete == "yes":
368                         base = pymysql.connect(host=server.get(), user="admin", password="admin", database="base")
369                         cursor =base.cursor()
370                         cursor.execute("delete from cliente where id=%s",ID.get())
371                         base.commit()
372                         Mostrar()
373                         base.close()
374                         Limpiar()
375                         messagebox.showinfo("La informacion ha sido eliminada","El Registro Se ha Eliminado Correctamente")
376
377
```

La variable Eliminar elimina todo un registro y los datos de un contratista que se hallan elegido en el aplicativo.

```
378         def crearpdf():
379         #import plantillapdf
380         #from plantillapdf import PDF
381         # Instantiation of inherited class
382                 pdf = PDF(orientation="portrait",format="A4")
383                 pdf.alias_nb_pages()
384                 pdf.add_page()
385                 pdf.set_font('Arial', 'B', 9)
```

La variable crearpdf es la que crea toda la plantilla y establece las variables llamadas de la base de datos al documento .pdf a generar

```
393             pdf.set_font('Arial', '', 9)
394             pdf.multi_cell(w = 0, h = 7, txt = 'Que la señor(a) '+Nombre.get()+" "+Apellido.get()+', identificado con cédula de ciudadanía No. '+Cedula.get()+' de  '+CiudadE
395             pdf.ln(h=2)
396             pdf.multi_cell(w=0,h=7,txt="1.Número de Contrato:  "+Contrato.get(), align='J')
397             pdf.ln(h=2)
398             pdf.multi_cell(w=0,h=7,txt="Objeto: Prestar servicios de apoyo de carácter temporal a la gestión en los procesos académicos administrativos de la Coordinación Ac
399             pdf.ln(h=2)
400             pdf.multi_cell(w=0,h=7,txt="Plazo de ejecución: 1 de febrero de 2023 hasta el 28 de diciembre de 2023", align='J')
401             pdf.ln(h=2)
402             pdf.multi_cell(w=0,h=7,txt="Fecha de Inicio de Ejecución: 1 de febrero de 2023", align='J')
403             pdf.ln(h=2)
404             pdf.multi_cell(w=0,h=7,txt="Fecha de Terminación de Contrato: 28 de diciembre de 2023", align='J')
405             pdf.ln(h=2)
406             pdf.multi_cell(w=0,h=7,txt="Término de Ejecución: El término real ejecutado por la contratista es de Diez (10) meses y Dieciseis (16) días, con una prórroga de D
407             pdf.ln(h=2)
408             pdf.multi_cell(w=0,h=7,txt="Prorroga: Doce (12) días.", align='J')
409             pdf.ln(h=2)
410             pdf.multi_cell(w=0,h=7,txt="Adición: Hasta el 28 de diciembre de 2023", align='J')
411             pdf.ln(h=2)
412             pdf.multi_cell(w=0,h=7,txt="Valor: El valor del contrato para todos los efectos legales y fiscales se fijó en la suma de VEINTICINCO MILLONES DOSCIENTOS OCHENTA
413             pdf.add_page()
414             pdf.multi_cell(w=0,h=7,txt="Obligaciones Específicas del Contrato: 1. Apoyar a la Coordinación académica en lo relacionado con el sistema de gestión documental y
415
416             pdf.multi_cell(w=0,h=7,txt="Se expide a solicitud del interesado, de acuerdo con la información registrada en el sistema ON BASE del SENA y Secop II, a los  "+sti
```

en estas líneas se realizan las concatenaciones de los datos, y se ve reflejado en el documento final. Si se requiere agregar o modificar un dato del pdf, se puede modificar parte de este código para que el archivo .pdf que se requiere salga según los requerimientos.

```
418     #pdf.ln(h=2) salto de linea
419             pdf.image('./img/firmapablo.jpg',
420             x = 110, y = 240,
421              w = 10, h = 10)#, link = url
422
423             pdf.image('./img/firmadagny.jpg',
424              x = 110, y = 250,
425             w = 10, h = 10)#, link = url
426
427             pdf.image('./img/firma.png',
428              x = 100, y = 200,
429             w = 25, h = 25)#, link = url
430
431             pdf.ln(h=10)
432             pdf.set_font('Arial', 'B', 9)
433
434             pdf.multi_cell(w=0,h=7,txt="Diego Armando Cristancho Cristancho ", align='C')
435             pdf.multi_cell(w=0,h=7,txt="Subdirector (E) Centro para la Industria de la Comunicación ", align='C')
436             pdf.multi_cell(w=0,h=7,txt="Gráfica Servicio Nacional de Aprendizaje SENA", align='C')
437             pdf.ln(h=5)
438
439             pdf.multi_cell(w=0,h=7,txt="Proyectó: Juan Pablo Alvarez Sarmiento; Apoyo Administrativo ", align='J')
440
441             pdf.set_font('Arial', '', 9)
442
443             pdf.multi_cell(w=0,h=7,txt="Revisó: Dagny Galindo; Profesional en Contratación", align='J')
444
```

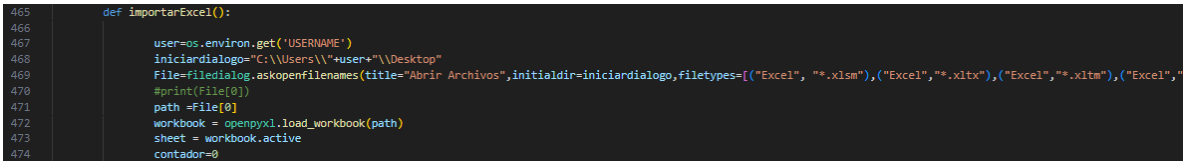
en las líneas 418 hasta la 419 se establecen las relaciones con las firmas e imágenes necesarias para el documento .pdf a realizar, si se requiere cambiar las firmas se debe cambiar el path de la imagen a establecer en esa área del documento.

Las líneas 434 a la 443 establecen las alineaciones respectivas a esas firmas en el documento.

```
445             """((((((((((((((((((((((((((CREAR CARPETA Y ABRIR PDF ))))))))))))))))))))))))) """
446             if os.path.exists('./certificados/'):
447                     next
448             else:
449                     subprocess.call("cmd /c mkdir certificados")
450             pdf.output("./certificados/"+Cedula.get()+'.pdf')
451             #ruta_pdf = os.path.join(os.path.dirname(os.path.abspath(__file__)),"certificados",Cedula.get()+'.pdf')
452
453             ruta = os.path.abspath("./certificados/"+Cedula.get()+'.pdf')
454             print(ruta)
455             webbrowser.open_new (ruta)
456             subprocess.call([ruta],shell=True)
457             """((((((((((((((((((((((((((FIN CREAR CARPETA Y ABRIR PDF ))))))))))))))))))))))))) """
458             Frame2=Label(self.wind,bg="teal", text=ruta, font=("calibri", 14 )).pack()
459
460             #subprocess.call([path],shell=True)
461         #subprocess.run([path],shell=True)
462             #webbrowser.WindowsDefault(path)
463             #os.system(path)
464      #crearpdf()
```

Esta parte del código establece el guardado de los certificados en la carpeta Certificados e indica que se guarden con la concatenación de la variable cédula como se ve en la línea 453. Si se requiere, este espacio se puede modificar para

renombrar el tipo de guardado o el lugar.



```
def importarExcel():

    user=os.environ.get('USERNAME')
    iniciardialogo="C:\\Users\\"+user+"\\Desktop"
    File=filedialog.askopenfilenames(title="Abrir Archivos",initialdir=iniciardialogo,filetypes=[("Excel", "*.xlsm"),("Excel","*.xltx"),("Excel","*.xltm"),("Excel","
    #print(File[0])
    path =File[0]
    workbook = openpyxl.load_workbook(path)
    sheet = workbook.active
    contador=0
```

La función ImportarExcel permite llenar la base de datos con un archivo CSV o xlsm o formatos reconocidos por Excel. Para ello, el Excel debe contener las columnas exactas que concuerden con la información de la base de datos a modificar.

```
            list_values = list(sheet.values)
            for row in list_values[1:]:
                contador+=1
                ID.set(row[0])
                Nombre.set(row[1])
                Apellido.set(row[2])
                Cedula.set(row[3])
                CiudadExpedicion.set(row[4])
                Direccion.set(row[5])
                Contrato.set(row[6])
                ARL.set(row[7])
                EPS.set(row[8])
                FechaNacimiento.set(row[9])
                genero.set(row[10])
                Rh.set(row[11])
                Celular.set(row[12])
                Telefono.set(row[13])
                Correo.set(row[14])
                Cargo.set(row[15])
                FechaDocumento.set(row[16])
                Dependencia.set(row[17])
                Jefe.set(row[18])
                DescripcionContrato.set(row[19])
                ValorContrato.set(row[20])
                AutorizacionContrato.set(row[21])
                PAA.set(row[22])
                Banco.set(row[23])
                TipoCuenta.set(row[24])
                NumeroCuenta.set(row[25])
                CDP.set(row[26])
                FechaCDP.set(row[27])
                CRP.set(row[28])
                Agregar()
                Limpiar()
```

El código corre cada una de las variables y al final agrega y limpia los campos. Permitiendo que quede en la base de datos la información.

```
511     def BuscarCodigo(event):
512         cargarServer()
513         #print(server.get()+"  " +UserServer.get()+"    " +passServer.get()+"    "+baseServer.get())
514         try:
515             self.trv.delete(*self.trv.get_children())
516             base = pymysql.connect(host=server.get(), user="root", password="", database="base")
517             cursor= base.cursor()
518             cursor.execute('select * from cliente WHERE id LIKE "%'+ID.get()+'%"')
519             #micursor.execute('SELECT * FROM "tblestudiantes" WHERE Nom_Estudiante LIKE "%'+VarNombre.get()+'%"')
520             result = cursor.fetchall()
521 
522             if len(result) !=0:
523                 for row in result:
524                     self.trv.insert(parent='',index=row[0],iid=row[0],values= row)
525                 self.trv.selection_set(row[0])
526                 self.trv.see(row[0])
527             base.commit()
528             base.close()
529         except pymysql.Connection.Error as er:
530             codigo_error_extendido = er.args[1]
531             messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )
532 
```

La función buscarcódigo permite traer la información del contratista a través de un ID en específico.

```
533     def BuscarNombre(event):
534         cargarServer()
535         #print(server.get()+"  " +UserServer.get()+"    " +passServer.get()+"    "+baseServer.get())
536         try:
537             self.trv.delete(*self.trv.get_children())
538             base = pymysql.connect(host=server.get(), user="root", password="", database="base")
539             cursor= base.cursor()
540             cursor.execute('select * from cliente WHERE Nombre LIKE "%'+Nombre.get()+'%"')
541             #micursor.execute('SELECT * FROM "tblestudiantes" WHERE Nom_Estudiante LIKE "%'+VarNombre.get()+'%"')
542             result = cursor.fetchall()
543 
544             if len(result) !=0:
545                 for row in result:
546                     self.trv.insert(parent='',index=row[0],iid=row[0],values= row)
547                 self.trv.selection_set(row[0])
548                 self.trv.see(row[0])
549             base.commit()
550             base.close()
551         except pymysql.Connection.Error as er:
552             codigo_error_extendido = er.args[1]
553             messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )
554 
```

La función BuscarNombre permite traer la información del contratista a través de un nombre en específico.

```
555     def BuscarCedula(event):
556         cargarServer()
557         #print(server.get()+"  " +UserServer.get()+"    " +passServer.get()+"    "+baseServer.get())
558         try:
559             self.trv.delete(*self.trv.get_children())
560             base = pymysql.connect(host=server.get(), user="root", password="", database="base")
561             cursor= base.cursor()
562             cursor.execute('select * from cliente WHERE Cedula LIKE "%'+Cedula.get()+'%"')
563             #micursor.execute('SELECT * FROM "tblestudiantes" WHERE Nom_Estudiante LIKE "%'+VarNombre.get()+'%"')
564             result = cursor.fetchall()
565 
566             if len(result) !=0:
567                 for row in result:
568                     self.trv.insert(parent='',index=row[0],iid=row[0],values= row)
569                 self.trv.selection_set(row[0])
570                 self.trv.see(row[0])
571             base.commit()
572             base.close()
573         except pymysql.Connection.Error as er:
574             codigo_error_extendido = er.args[1]
575             messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )
576 
577     def BuscarCorreo(event):
```


La función BuscarCedula permite traer la información del contratista a través de una Cedula en específico

```
577     def BuscarCorreo(event):
578             cargarServer()
579             #print(server.get()+"  " +UserServer.get()+"   " +passServer.get()+"   "+baseServer.get())
580             try:
581                     self.trv.delete(*self.trv.get_children())
582                     base = pymysql.connect(host=server.get(), user="root", password="", database="base")
583                     cursor= base.cursor()
584                     cursor.execute('select * from cliente WHERE Correo LIKE "%'+Correo.get()+'%"')
585                     #micursor.execute('SELECT * FROM "tblestudiantes" WHERE Nom_Estudiante LIKE "%'+VarNombre.get()+'%"')
586                     result = cursor.fetchall()
587
588                     if len(result) !=0:
589                             for row in result:
590                                     self.trv.insert(parent='',index=row[0],iid=row[0],values= row)
591                             self.trv.selection_set(row[0])
592                             self.trv.see(row[0])
593                     base.commit()
594                     base.close()
595             except pymysql.Connection.Error as er:
596                     codigo_error_extendido = er.args[1]
597                     messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )
598
```

La función Buscarcorreo permite mostrar la información del contratista a través de un correo en específico

```
599     def BuscarContrato(event):
600             cargarServer()
601             #print(server.get()+"  " +UserServer.get()+"   " +passServer.get()+"   "+baseServer.get())
602             try:
603                     self.trv.delete(*self.trv.get_children())
604                     base = pymysql.connect(host=server.get(), user="root", password="", database="base")
605                     cursor= base.cursor()
606                     cursor.execute('select * from cliente WHERE Contrato LIKE "%'+Contrato.get()+'%"')
607                     #micursor.execute('SELECT * FROM "tblestudiantes" WHERE Nom_Estudiante LIKE "%'+VarNombre.get()+'%"')
608                     result = cursor.fetchall()
609
610                     if len(result) !=0:
611                             for row in result:
612                                     self.trv.insert(parent='',index=row[0],iid=row[0],values= row)
613                             self.trv.selection_set(row[0])
614                             self.trv.see(row[0])
615                     base.commit()
616                     base.close()
617             except pymysql.Connection.Error as er:
618                     codigo_error_extendido = er.args[1]
619                     messagebox.showwarning("Error de Conexion ","Error "+str({codigo_error_extendido})+" No se ha podido conectar al Servidor " +server.get() )
620
621
622
623
```

La función BuscarContrato permite mostrar la información del contrastista digitando el número del contrato en específico

```
627  # BARRA MENU
628
629          barraMenu=tk.Menu(root,foreground="orange",activebackground="blue")
630          root.config(menu=barraMenu, width=300, height=300,cursor="heart",highlightbackground="yellow")
631          menuconect=tk.Menu(barraMenu, tearoff=0,activebackground ="orange",activeforeground="red",)
632          menuconect.add_command(label="Conectar",activebackground ="green")
633          menuconect.add_command(label="Salir",activebackground ="green")
634          barraMenu.add_cascade(label="Inicio",menu=menuconect,activebackground ="orange")
635          bbddMenu=tk.Menu(barraMenu, tearoff=0,activebackground ="green",activeforeground="red",)
636          bbddMenu.add_command(label="Agregar",activebackground ="green",command=Agregar)
637          bbddMenu.add_command(label="Actualizar",activebackground ="green",command=Actualizar)
638          bbddMenu.add_command(label="Nuevo",activebackground ="green",command=Limpiar)
639          bbddMenu.add_command(label="Crear PDF",activebackground ="green",command=crearpdf)
640          bbddMenu.add_command(label="Importar Excel",activebackground ="green",command=importarExcel)
641          bbddMenu.add_command(label="Eliminar",activebackground ="green",command=Eliminar)
642          barraMenu.add_cascade(label="Registro",menu=bbddMenu,activebackground ="Orange")
643
644          menuventanas=tk.Menu(barraMenu, tearoff=0,activebackground ="green",activeforeground="red",)
645          menuventanas.add_command(label="Primero",activebackground ="green")
646          menuventanas.add_command(label="Siguiente",activebackground ="green")
647          menuventanas.add_command(label="Anterior",activebackground ="green")
648          menuventanas.add_command(label="Ultimo",activebackground ="green")
649          barraMenu.add_cascade(label="Movimientos",menu=menuventanas,activebackground ="orange")
650
651          menubuscar=tk.Menu(barraMenu, tearoff=0,activebackground ="green",activeforeground="red",)
652          menubuscar.add_command(label="Todos",activebackground ="green")
653          menubuscar.add_separator()
654          menubuscar.add_command(label="Por Email",activebackground ="green",command=lambda :BuscarCorreo(2))
655          menubuscar.add_separator()
656          menubuscar.add_command(label="Por Codigo",activebackground ="green",command=lambda :BuscarCodigo(2))
657          menubuscar.add_separator()
658          menubuscar.add_command(label="Por Contrato",activebackground ="green",command=lambda :BuscarContrato(2))
659          menubuscar.add_separator()
```

```
662          menubuscar.add_command(label="Por Nombre",activebackground ="green",command=lambda :BuscarNombre(2))
663          menubuscar.add_separator()
664          menubuscar.add_command(label="Por Cedula",activebackground ="green",command=lambda :BuscarCedula(2))
665          menubuscar.add_separator()
666          menubuscar.add_command(label="Vaciar Tabla",activebackground ="green")
667          barraMenu.add_cascade(label="Busqueda",menu=menubuscar,activebackground ="orange")
668          menuinfo=tk.Menu(barraMenu, tearoff=0,activebackground ="green",activeforeground="red",)
669          menuinfo.add_command(label="Acerca de...",activebackground ="green")
670          barraMenu.add_cascade(label="Ayuda...",menu=menuinfo,activebackground ="orange")
671          #FIN BARRA MENU
```

desde la línea 627 hasta la 670 se establece todo el tob barra menú en el que se organiza el funcionamiento de la clase TK kinter que permite correr la ventana en un orden específico, opera de forma homologa a un CSS y un HTML que operan juntos, pero contenida en la información de la librería tk kinter para aplicaciones de escritorio, en este espacio se le da formato a los botones y otras funciones.

```
679         lbl1 = Ctk.CTkLabel(Frame1, text= "ID", width=20)
680         lbl1.grid(row=0, column=0,padx=5, pady=3,sticky="e")
681         self.entID=Ctk.CTkEntry(Frame1,border_color="teal",textvariable=ID)
682         self.entID.grid(row=0,column=1, padx=5, pady=3)
683         self.entID.bind("<Return>", lambda event: BuscarCodigo(1))
684 
685         lbl2 = Ctk.CTkLabel(Frame1, text= "Nombre", width=20)
686         lbl2.grid(row=1, column=0,padx=5, pady=3,sticky="e")
687         self.entNombre = Ctk.CTkEntry(Frame1,border_color="teal", textvariable=Nombre)
688         self.entNombre.grid(row=1,column=1, padx=5, pady=3)
689         self.entNombre.bind("<Return>", lambda event: BuscarNombre(1))
690 
691         lbl3 = Ctk.CTkLabel(Frame1, text= "Apellido", width=20)
692         lbl3.grid(row=2, column=0,padx=5, pady=3,sticky="e")
693         self.entApellido = Ctk.CTkEntry(Frame1,border_color="teal", textvariable=Apellido)
694         self.entApellido.grid(row=2,column=1, padx=5, pady=3)
695 
696         lbl4 = Ctk.CTkLabel(Frame1, text= "Cedula", width=20)
697         lbl4.grid(row=3, column=0,padx=5, pady=3,sticky="e")
698         self.entCedula = Ctk.CTkEntry(Frame1,border_color="teal", textvariable=Cedula)
699         self.entCedula.grid(row=3,column=1, padx=5, pady=3)
700         self.entCedula.bind("<Return>", lambda event: BuscarCedula(1))
701 
702         lbl5 = Ctk.CTkLabel(Frame1, text= "Ciudad Expedicion", width=20)
703         lbl5.grid(row=4, column=0,padx=5, pady=3,sticky="e")
704         self.entCiudadExpedicion= Ctk.CTkEntry(Frame1,border_color="teal", textvariable=Ciudad
705         self.entCiudadExpedicion.grid(row=4,column=1, padx=5, pady=3)
706 
707         lbl6 = Ctk.CTkLabel(Frame1, text= "Dirección", width=20)
708         lbl6.grid(row=5, column=0,padx=5, pady=3,sticky="e")
```

Desde la línea 679 hasta la línea 865 corresponde a los label que representan las etiquetas y los Entry corresponde a cajas de texto donde se agrega la información requerida estas, se encuentran distribuidas en 4 filas y cada una con su dimensión correspondiente.

```
872         btn1 =Ctk.CTkButton(Frame1,fg_color="teal" ,text="Agregar", width=12, height=2, command= Agregar)
873         btn1.grid(row=11, column=0, padx=10, pady=10)
874         btn1.place(rely=0.86,relx=0.27,relheight=0.11,relwidth=0.10)
875 
876         btn2 =Ctk.CTkButton(Frame1,fg_color="teal" , text="Eliminar", width=12, height=2, command=Eliminar)
877         btn2.grid(row=11, column=1, padx=10, pady=10)
878         btn2.place(rely=0.86,relx=0.75,relheight=0.11,relwidth=0.10)
879 
880         btn3 =Ctk.CTkButton(Frame1,fg_color="teal" , text="Actualizar", width=12, height=2,command= Actualizar)
881         btn3.grid(row=11, column=2, padx=10, pady=10)
882         btn3.place(rely=0.86,relx=0.39,relheight=0.11,relwidth=0.10)
883 
884         btn4 =Ctk.CTkButton(Frame1,fg_color="teal" , text="Monitor", width=12, height=2, command= Mostrar)
885         btn4.grid(row=11, column=3, padx=10, pady=10)
886         btn4.place(rely=0.86,relx=0.51,relheight=0.11,relwidth=0.10)
887 
888         btn5 =Ctk.CTkButton(Frame1,fg_color="teal" , text="Limpiar", width=12, height=2, command=Limpiar)
889         btn5.grid(row=11, column=4, padx=10, pady=10)
890         btn5.place(rely=0.86,relx=0.63,relheight=0.11,relwidth=0.10)
891 
892         btn6 =Ctk.CTkButton(Frame1,fg_color="teal", text="expide tu certificado", width=20, height=2, command=crearpdf)
893         btn6.grid(row=11, column=5, padx=1, pady=10)
```

Desde la línea 872 hasta la línea 898 corresponde a los botones creados en la clase contratista para realizar las consultas requeridas dando Click sobre ellas realizando las funciones de agregar, actualizar, mostrar en monitor, limpiar las cajas de texto, expedir certificado e importar Excel.

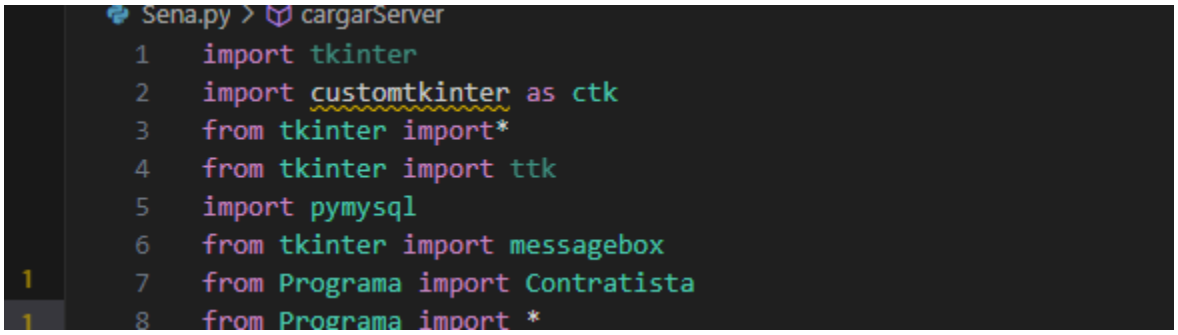
```
900         treeScroll = ttk.Scrollbar(Frame2,orient="vertical")
901         treeScroll.pack(side="right",fill="y")
902
903         treeScrollx = ttk.Scrollbar(Frame2,orient="horizontal")
904         treeScrollx.pack(side= "bottom",fill="x")
```

Desde la línea 900 hasta la línea 904 corresponde al código realizado para adicionarle los Scroll en la ventana los cuales nos permiten desplazarnos de izquierda a derecha, de arriba a bajo dentro de la pantalla.

```
908         self.trv = ttk.Treeview(Frame2, columns=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
909         self.trv.pack()
910 
911         self.trv.heading(1, text="ID")
912         self.trv.heading(2, text="Nombre")
913         self.trv.heading(3, text="Apellido")
914         self.trv.heading(4, text="Cedula")
915         self.trv.heading(5, text="CiudadExpedicion")
916         self.trv.heading(6, text="Direccion")
917         self.trv.heading(7, text="Contrato")
918         self.trv.heading(8, text="ARL")
919         self.trv.heading(9, text="EPS")
920         self.trv.heading(10, text="Fecha de nacimiento")
921         self.trv.heading(11, text="Género")
922         self.trv.heading(12, text="RH")
923         self.trv.heading(13, text="Celular")
924         self.trv.heading(14, text="Telefono")
925         self.trv.heading(15, text="Correo")
926         self.trv.heading(16, text="Cargo")
927         self.trv.heading(17, text="Fecha de Documento")
928         self.trv.heading(18, text="Dependencia")
929         self.trv.heading(19, text="Jefe")
930         self.trv.heading(20, text="DescripcionContrato")
931         self.trv.heading(21, text="ValorContrato")
932         self.trv.heading(22, text="AutorizacionContrato")
933         self.trv.heading(23, text="PAA")
934         self.trv.heading(24, text="Banco")
935         self.trv.heading(25, text="TipoCuenta")
936         self.trv.heading(26, text="NumeroCuenta")
```

Desde a línea 908 hasta la línea 942 corresponde a los treeview los cuales se encuentran representados de forma jerárquica y ubicados dentro del aplicativo en columnas.

SENA.PY



```
Sena.py > cargarServer
1   import tkinter
2   import customtkinter as ctk
3   from tkinter import*
4   from tkinter import ttk
5   import pymysql
6   from tkinter import messagebox
7   from Programa import Contratista
8   from Programa import *
```

De la línea 1 a la línea 8 corresponde a la importación de las bibliotecas para el correcto funcionamiento de la aplicación de escritorio.

```
9   ctk.set_appearance_mode("System")  # Modes: system (default), light, dark
10  ctk.set_default_color_theme("blue")  # Themes: blue (default), dark-blue, green
11  #si desea colocar icono descomentar numeral
12  #pantalla.iconbimap("nombre icono")
13  global pantalla1
14  pantalla1=ctk.CTk()
```

De la línea 9 hasta la línea 14 corresponde al tema del aplicativo donde se le puede cambiar de tema y realiza llamado a la pantalla principal.

```
20  def cargarServer():
21      if os.path.isfile("./server/server.txt"):
22          fileser=open("./server/server.txt","r")
23          contador = 0
24          for linea in fileser:
25              if contador ==0:
26                  server.set(linea.strip())
27                  print(server.get())
28              elif contador ==1:
29                  UserServer.set(linea.strip())
30                  print(UserServer.get())
31              elif contador ==2:
32                  passServer.set(linea.strip())
33                  print(passServer.get())
34              elif contador ==3:
35                  baseServer.set(linea.strip())
36                  print(baseServer.get())
37              elif contador >= 4 :
```

Desde la línea 20 hasta la línea 40Esta parte del código hace correr el Server definido en la subcarpeta Server de la carpeta principal, este protocolo lo que hace es leer la información del archivo y procesarla para que pase oculta y no sea visible para toda persona.

```
49  def abrir_ventana_principal():
50
51      global Contratista
52
53
54      root = Toplevel(pantalla1)
55      Contratista =Contratista(root)
56      pantalla1.withdraw()
57
58
59
```

De la línea 49 hasta la línea 56 corresponde a la función abrir la ventana principal y contine la función withdraw para ocultar la ventana.

```
def inicio_sesion():


    pantalla1.geometry("290x400")
    pantalla1.title("Inicio De Sesion")
    pantalla1.resizable(False,0)
    pantalla1.wm_attributes("-topmost", 0)
    def Salir():

        pantalla1.quit()
    pantalla1.protocol('WM_DELETE_WINDOW', Salir)
    Label(pantalla1, text="Por favor ingrese su \n usuario y Contraseña",bg="Green",fg="w

    Label(pantalla1, text="").pack()

    global nombreusuario_verify
    global contraseñausuario_verify

    nombreusuario_verify=StringVar()
    contraseñausuario_verify=StringVar()

    global nombre_usuario_entry
    global contraseña_usuario_entry

    Label(pantalla1, text="usuario").pack()
    nombre_usuario_entry =ctk.CTkEntry(pantalla1, textvariable=nombreusuario_verify)
    nombre_usuario_entry.pack()
    Label(pantalla1).pack()
    Label(pantalla1, text="contraseña").pack()
    contraseña_usuario_entry = ctk.CTkEntry(pantalla1, show="*", textvariable=contraseñau
```

desde la línea 64 hasta la línea 71 corresponde a la dimensión geométrica de la ventana inicio de sesión, y la función salir.

```
88      Label(pantalla1, text="usuario").pack()
89      nombre_usuario_entry =ctk.CTkEntry(pantalla1, textvariable=nombreusuario_verify)
90      nombre_usuario_entry.pack()
91      Label(pantalla1).pack()
92      Label(pantalla1, text="contraseña").pack()
93      contraseña_usuario_entry = ctk.CTkEntry(pantalla1, show="*", textvariable=contraseñausu
94      contraseña_usuario_entry.pack()
95      Label(pantalla1).pack()
96      ctk.CTkButton(pantalla1,width=140,height=45,fg_color="teal" ,text="Iniciar Sesion",  co
97      pantalla1.mainloop()
98
```

desde la línea 88 hasta la línea 96 corresponde a los label que son las etiquetas y los entry correspondientes a las cajas de texto donde se requiere usuario y contraseña del inicio de sesión.

```
103 def validacion_datos():
104     if nombreusuario_verify.get()=="" or contraseñausuario_verify.get()=="":
105         messagebox.showwarning("Error ","El usuario ni la contraseña pueden estar Vacios ")
106     else:
107 
108         bd = pymysql.connect(host=server.get(),user=UserServer.get(), password=passServer.get(), database=baseServer.get(
109 
110         fcursor=bd.cursor()
111 
112         fcursor.execute("SELECT contraseña FROM login WhERE usuario='"+nombreusuario_verify.get()+"' and contraseña='"+co
113 
114         if fcursor.fetchall():
115             print(fcursor)
116             #messagebox.showinfo(title="Inicio de Sesion correcto", message="Usuario y contraseña correcta")
117             bd.close()
118 
119             abrir_ventana_principal()
120 
```

Desde la línea 103 hasta la línea 127 corresponde a La función validar datos con sus respectivas condiciones para hacer el ingreso si es válido la información o contiene error y de esta manera dirigirlo a la pagina principal.

USUARIO: ADMIN

CLAVE:1023