**Introduction**

This is a project of a file explorer that is coded using both python and MySQL. The goal of this project was to create a useful utility while also making it look minimalistic and also modern. This file explorer was created in order to fill in some features that windows file explorer lacks. This program is coded and designed to run on any windows machine, but it is has been better optimized to run on Windows 10 machines. The program is also capable of running on other platforms such as linux, Mac OS and even android with some slight tweaks to the file accessing and permission system. But some features may not work properly.

As stated before the main goal of this file explorer was to offer some features that Windows File Explorer lacks. Some of the features that have been added are…

* The ability to copy or cut files from multiple directories to a single location in one operation.
* Quick actions that are easily accessible with a single click away.
* The ability to have a username and a password to lock the file explorer from unwanted access.
* A clean user interface that follows Google Material Design 2 standards with nice smooth animations and transitions.
* A very simple and modern way to interact with various files present on your system

Along with these extra features, all the normal features of any ordinary file explorer is also present. This would include the ability to browse, cut, copy, and delete files and directories. Also unlimited undo and redo history is also available.

In this program, python is used to handle all the various functions of the program. MySQL tables are used to store the user actions for undo and redo history. MySQL tables are also used for managing the selected files before an action is executed. It is also used to handle the login credentials. Kivy is used to create the user interface and is coded in a separate “KV” file that is called and executed by python on program execution. This is the basic structure of the file explorer.

**Modules Used:**

There are a various number of modules used in order for the file explorer to run. They are primarily divided into three sections...

1. Modules That Handle The User Interface
   * Kivy: This module handles all the logic required to organize the user interface. It controls at what time what screen must be shown and links all the screens together.
   * Kivy MD: This module is a modified version of the Kivy module with better looking user interface elements. All the user interface elements in this module follow Google material Design 2 guidelines. This module has been used for creating the actual user interface. It has been slightly modified to fit with the aesthetics of our project.
2. Modules That Handle File Operations:

* Os: It is used for various file operations such as selecting all folders within a specified directory.
* Shutil: This module is used for copy and cut operations on files.
* Nt: This module is used for other miscellaneous file operations such as getting the name of a file.
* Win32api,Win32file : Both these modules are used to get a device partition info
* Getpass: This module is used to obtain the current user id name.

1. Modules Related to Mysql:

* Mysql\_connector: It is used to connect python with a MySQL database.
* Json: It enables the ability to add json objects to MySQL database in order to get varying individually addressable entries for a single row in a single column. This is used for storing multiple user actions in one single session i.e.one user action entry in the user action table

**Hardware and Software Requirements:**

**Hardware Requirements:**

* A relatively modern System is preferred.
* A minimum of 4 GB of RAM.
* The minimum processor required is anything above an Intel Pentium or an AMD athalon series.
* The program requires a minimum of 200mb of hard disk space to be installed.

**Software Requirements:**

* The system can be running any operating system, but it is preferred that the system is running Windows 10 with all latest updates for optimal performance.
* MySQL server must also be installed on the system for the program to be used.

**Existing System and Proposed System:**

This file explorer was designed keeping in mind the requirements that was not met by default file explorers such as windows file explorer and also mac finder.

Some features that these software’s lacked on and that are present in this file explorer are…

* The ability to copy and cut files from multiple different locations at one time to a single location. As of now both windows file explorer and Finder does not allow you to copy multiple files that are all stored in different locations to a single location or multiple locations. This is achieved in our file explorer using a file cart system making it more efficient and faster for a user to sort a large number of files.
* Windows file explorer and Finder also do not allow a user to lock his/her file explorer. This is a primary feature of this file explorer as the privacy of your files and their content is very important.
* This file explorer also has a quick command menu allowing an user to execute commands directly on the screen very quickly without needing to be familiar or memorize and shortcut keys. This is better than the current system in other explorers that require you to memorize certain short cut keys in order for increasing the speed of your work.
* Most importantly, the file explorer flaunts a beautiful, modern, minimalistic design that is based on Google Material Design 2. The user interface pulls a lot of reference to the sleek and beautiful look of android 10 devices while running on a windows machine. This is far better than the somewhat outdated and messy looking interface of windows file explorer and Finder.
* This File explorer also has a full on properly designed dark theme to prevent eye strain when working in dim lighting conditions. The dark theme is designed to cover every part of the user interface unlike Windows explorer’s half designed dark theme and Finder’s outdated style.

**Coding**

* The program is split into two files. One file contains the python program which is attached first. The second file contains the KV file which contains majority of the user interface code. Both codes can be executed directly from the python console and must be within the same folder. The Kivy parser that is called by the Kivy module will automatically load the file and convert it into the user interface.

Python:

    #### File Explorer Code ####

#### Created By: Gokul, Shashti, Guhan ####

#Import Kivy Modules

**from** kivymd.app **import** MDApp

**from** kivy.lang **import** Builder

**from** kivy.uix.screenmanager **import** ScreenManager, Screen

**from** kivymd.uix.dialog **import** MDDialog

**from** kivymd.uix.list **import** ThreeLineIconListItem,IconLeftWidget,ThreeLineAvatarListItem

**from** kivy.core.window **import** Window

**from** kivymd.uix.button **import** MDFlatButton

**from** kivymd.uix.textfield **import** MDTextField

#Import File Managing modules

**import** ntpath

**import** os

**import** shutil

**import** json

**import** win32api

**import** win32file

**import** getpass

#Import MySQL Module

**import** mysql.connector

#Initialize MySQL connection and prepare cursor

mydb = mysql.connector.connect(

    host = "localhost",

    user = "root",

    passwd = "Guhan",

    database = "file\_explorer"

    )

mycursor = mydb.cursor()

#Define Core MySQL Formulas to be used later in the program

select\_check = "SELECT EXISTS(SELECT \* FROM selection WHERE path=%s)"

insert\_select = "INSERT INTO selection VALUES ( %s, %s, %s, %s)"

exist\_check = "SELECT EXISTS(SELECT \* FROM selection)"

user\_actions = "INSERT INTO user\_action(data) VALUES (%s)"

action\_check = "SELECT EXISTS(SELECT \* FROM user\_action)"

undo\_formula = "SELECT \* FROM user\_action ORDER BY id DESC LIMIT 1"

redo\_insert = "INSERT into undo\_history(data) VALUES(%s)"

redo\_check = "SELECT EXISTS(SELECT \* FROM undo\_history)"

redo\_formula = "SELECT \* FROM undo\_history ORDER BY id DESC LIMIT 1"

####ALL THE FUNCTIONS THE PROGRAM RECQUIRES TO RUN###

#Define Login Class for login functionality

**class** Login():

**def** loginop(self, userid, passwd):

        mycursor.execute("SELECT \* FROM login\_info")

        self.auth\_data = mycursor.fetchone()

        self.auth\_id = str((self.auth\_data[0]))

        self.auth\_passwd = str((self.auth\_data[1]))

**if** self.auth\_id == userid **and** self.auth\_passwd == passwd:

            #This updates the global variable of sm that tells the screen

#Manager to display the main page

            sm.current = "main\_screen"

**else**:

            Errors.passwd\_not\_crct(self)

#Define Select\_functions class that when called takes the selected file data and updates the

#selection table to be used by file cart

**class** Select\_Functions():

    #the copy\_select function is called when an entry is to be copied

**def** copy\_select(self, selected\_entry):

        #First we check to see if the user has highlighted any entry in the

# file explorer

        #If true then we execute as planned or else we display an error to se

# lect an entry

**if** len(selected\_entry) != 0:

            self.selected\_entry\_name = ntpath.basename(selected\_entry[0])

            self.entry\_path = selected\_entry[0]

            ##Now we check if the selected file is a drive. If it is we cancel

#rest of function and ask to select a file or a directory

**if** os.path.ismount(self.entry\_path):

**return** Errors().file\_not\_selected()

**else**:

            #Now we check to see if the selected item is a file or a directory

# and we assign a name for it

            #if it is neither a file or a directory we display an internal error m #essage

**if** os.path.isfile(self.entry\_path):

                    self.entry\_type = "File"

**elif** os.path.isdir(self.entry\_path):

                    self.entry\_type = "Directory"

**else**:

**print**('The selected element is not valid or no longer exists')

              #Now we prepare the tuple for being added to the selection table

                mycursor.execute(select\_check, (self.entry\_path,))

                self.check\_value = mycursor.fetchone()

                self.selection\_data = (str(self.selected\_entry\_name),self.entry\_path, self.entry\_type, 1)

               #Check to see if the selected file is already in the selection table

**if** self.check\_value[0] == 0:

                    mycursor.execute(insert\_select, self.selection\_data)

                    mydb.commit()

**else**:

**return** Errors().file\_already\_exists()

                #Resest all variables to nothing to prevent the program from

# crashing if user executes other function in between

                self.selected\_entry\_name = None

                self.entry\_path = None

                self.entry\_type = None

**else**:

**return** Errors().file\_not\_selected()

        FileCart.FileCart\_Update(self)

    #the cut\_select function is called when an entry is to be cut

**def** cut\_select(self, selected\_entry):

         #First we check to see if the user has highlighted any entry in the

# file explorer

        #If true then we execute as planned or else we display an error to

# select an entry

**if** len(selected\_entry) != 0:

            self.selected\_entry\_name = ntpath.basename(selected\_entry[0])

            self.entry\_path = selected\_entry[0]

         #Now we check if the selected file is a drive. If it is we cancel rest

# of function and ask to select a file or a directory

**if** os.path.ismount(self.entry\_path):

**return** Errors().file\_not\_selected()

**else**:

                #Now we check to see if the selected item is a file or a directory

# and we assign a name for it

                #if it is neither a file or a directory we display an internal error

# message

**if** os.path.isfile(self.entry\_path):

                    self.entry\_type = "File"

**elif** os.path.isdir(self.entry\_path):

                    self.entry\_type = "Directory"

**else**:

**print**('The selected element is not valid or no longer exists')

              #Now we prepare the tuple for being added to the selection table

                mycursor.execute(select\_check, (self.entry\_path,))

                self.check\_value = mycursor.fetchone()

                self.selection\_data = (str(self.selected\_entry\_name),self.entry\_path, self.entry\_type, 2)

               #Check to see if the selected file is already in the selection table

**if** self.check\_value[0] == 0:

                    mycursor.execute(insert\_select, self.selection\_data)

                    mydb.commit()

**else**:

**return** Errors().file\_already\_exists()

                #Resest all varibales to nothing to prevent the program from cr

# ashing if user executes other function in between

                self.selected\_entry\_name = None

                self.entry\_path = None

                self.entry\_type = None

**else**:

**return** Errors().file\_not\_selected()

        FileCart.FileCart\_Update(self)

    #This function is used to mark an entry for moving to the recycle bin.  #There is no way to direct delete an entry

**def** delete\_select(self,selected\_entry):

        #First we check to see if the user has highlighted any entry in the

# file explorer

        #If true then we execute as planned or else we display an error to se

# lect an entry

**if** len(selected\_entry) != 0:

            self.selected\_entry\_name = ntpath.basename(selected\_entry[0])

            self.entry\_path = selected\_entry[0]

         #Now we check if the selected file is a drive. If it is we cancel rest

#of function and ask to select a file or a directory

**if** os.path.ismount(self.entry\_path):

**return** Errors().file\_not\_selected()

**else**:

                #Now we check to see if the selected item is a file or a directory

# and we assign a name for it

                #if it is neither a file or a directory we display an internal error

# message

**if** os.path.isfile(self.entry\_path):

                    self.entry\_type = "File"

**elif** os.path.isdir(self.entry\_path):

                    self.entry\_type = "Directory"

**else**:

**print**('The selected element is not valid or no longer exists')

              #Now we prepare the tuple for being added to the selection table

                mycursor.execute(select\_check, (self.entry\_path,))

                self.check\_value = mycursor.fetchone()

                self.selection\_data = (str(self.selected\_entry\_name),self.entry\_path, self.entry\_type, 3)

               #Check to see if the selected file is already in the selection table

**if** self.check\_value[0] == 0:

                    mycursor.execute(insert\_select, self.selection\_data)

                    mydb.commit()

**else**:

**return** Errors().file\_already\_exists()

                #Resest all varibales to nothing to prevent the program from cr

# ashing if user executes other function in between

                self.selected\_entry\_name = None

                self.entry\_path = None

                self.entry\_type = None

**else**:

**return** Errors().file\_not\_selected()

        FileCart.FileCart\_Update(self)

#This class is called when the selections in the selection table have to be pasted

**class** Execute\_Functions():

    #This function is called when the selections in the selection table have

# to be pasted

**def** execute(self, current\_path):

        #First we must check that there are selections in the selection table

#If not display error popup

        mycursor.execute(exist\_check)

        self.check\_exists = mycursor.fetchone()

**if** self.check\_exists[0] == 0:

**return** Errors().no\_file\_in\_cart()

**else**:

            #Go on with the normal execute function

            mycursor.execute("SELECT \* FROM selection")

            self.execute\_data = mycursor.fetchall()#get all of the recquired operations to be execute

            self.execute\_data\_length = len(self.execute\_data) #along with the amount of operations to be executed

            MainScreenvar = sm.get\_screen('main\_screen') # Assign a variable that will call the main screen whenever the file browser view needs to be updated

            #Here we are creating a list so that we can later append all the

# executed actions within a single token to this list.

            #This list is then converted into a json string and inserted into

# the database

            list1 = []

            #Initiate the loop that will execute all the recquired amount of op

#erations

**for** i **in** range(0, self.execute\_data\_length):

**if** self.execute\_data[i][3] == '1': #Check if it is a copy operation that must be executed

**if** self.execute\_data[i][2] == "File" : #Check if it is a file

**try**:#We try to copy or else we raise an error message

                            shutil.copy2(self.execute\_data[i][1], current\_path)

                            MainScreenvar.ids.filechooser\_icon.\_update\_files()#Updates the file chooser view

                            #Prepare a tuple for insertion into list

                            pre\_tuple = (1, "File", self.execute\_data[i][1], os.path.join(current\_path, str(self.execute\_data[i][0])))

                            list1.append(pre\_tuple)#Append the tuple to the list of all executed operations

**except** shutil.SameFileError: #This error is raised if a file already exists with the same name

                           dialog = MDDialog(text = self.execute\_data[i][0]+ " was not coppied from " + self.execute\_data[i][1] +

                                                       ' to ' +  current\_path + ' as there exists a file already in that name ',

                                             radius = (30,30,30,30))

                           dialog.open()

**else**:#If it is a folder

                        self.dir\_src = os.path.join(current\_path, str(self.execute\_data[i][0])) #Executed if it is a directory

**try**:

                            shutil.copytree(self.execute\_data[i][1], self.dir\_src)

                            MainScreenvar.ids.filechooser\_icon.\_update\_files()

                            pre\_tuple = (1, "Directory", self.execute\_data[i][1], self.dir\_src)

                            list1.append(pre\_tuple)

**except** shutil.SameFileError:

                           dialog = MDDialog(text = self.execute\_data[i][0]+ " was not copied from " + self.execute\_data[i][1] +   ' to ' +  current\_path + ' as there exists a folder already in that name ',

                                             radius = (30,30,30,30))

                           dialog.open()

**elif** self.execute\_data[i][3] == '2':#Executed if the operation is a cut operation

                #Here once again we check if it is a file or a directory. This is

# not necessary as both files and directories have same move

                # command but still we do it for sake of it.

**if** self.execute\_data[i][2] == 'File':

**try**: # Try to move or else we raise and error

                            shutil.move(self.execute\_data[i][1], current\_path)

                            MainScreenvar.ids.filechooser\_icon.\_update\_files()

                            pre\_tuple = (2, "File", self.execute\_data[i][1], os.path.join(current\_path, str(self.execute\_data[i][0])))

                            list1.append(pre\_tuple)

**except**:

                           dialog = MDDialog(text = self.execute\_data[i][0]+ " was not cut from " + self.execute\_data[i][1] +

                                                       ' to ' +  current\_path + ' as there exists a file already in that name ',

                                             radius = (30,30,30,30))

                           dialog.open()

**else**:

**try**: #Try to move or else raise and error

                            shutil.move(self.execute\_data[i][1], current\_path)

                            MainScreenvar.ids.filechooser\_icon.\_update\_files()

                            pre\_tuple = (2, "Directory", self.execute\_data[i][1], os.path.join(current\_path, str(self.execute\_data[i][0])))

                            list1.append(pre\_tuple)

**except**:

                           dialog = MDDialog(text = self.execute\_data[i][0]+ " was not cut from " + self.execute\_data[i][1] +

                                                       ' to ' +  current\_path + ' as there exists a folder already in that name ',

                                             radius = (30,30,30,30))

                           dialog.open()

**else**: #Executed if the operation is a delete operation

**try**:

                        shutil.move(self.execute\_data[i][1],'C:\Recycle Bin' )

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (3, "both", self.execute\_data[i][1],os.path.join("C:\Recycle Bin", self.execute\_data[i][0]))

                        list1.append(pre\_tuple)

**except**:

                        files = os.listdir('C:\Recycle Bin')#Get all files in recyclebin

                        files.sort(reverse=True) # We sort the list in reverse order so we can extrapolate the correct number to be added

                        key = 0 #Assign the key as simply zero

**for** a **in** range(0,len(files)): #We then traverse across the list of files

                            #Assign variables for the new element name and its extension

                            newelement\_name, newelement\_ext= os.path.splitext(self.execute\_data[i][0])

                            #Assign variables for the existing element name and its extension

                            exisiting\_name, exisiting\_ext= os.path.splitext(files[a])

                            #Check to see if the new element and the old element have the same extension

**if** newelement\_ext == exisiting\_ext:

                                #If the name of the new element name is same as the old element we assign key as zero

**if** newelement\_name == exisiting\_name:

**print**("hello")

                                    key = -1

**break**

                                #Else if the new elemenet is merely a substring of the old element we get the index of the highest element

**elif** newelement\_name **in** exisiting\_name:

                                    key = files.index(files[a])

**break**

**print**(key)

**if** key == -1:

                            new\_number = 1 #We assign the new elements respective number

**else**:# Or else we assign a number higher than the previous number

                            existing\_number = os.path.splitext(files[key])[0]

                            existing\_number = existing\_number[-1]

                            new\_number = int(existing\_number) + 1

                        #We then rename the element its new name and then mo

# ve it to the recycle bin

                        deleting\_element = os.path.split(self.execute\_data[i][1])[0] + "\\" + newelement\_name + str(new\_number) + newelement\_ext

                        os.rename(self.execute\_data[i][1], deleting\_element)

                        shutil.move(deleting\_element, 'C:/Recycle Bin')

                        #Then the usual of updating the file viewer and also addi

# ng the tuple to user\_Action table

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (3, "both", deleting\_element,os.path.join("C:\Recycle Bin", newelement\_name + str(new\_number)+ newelement\_ext))

                        list1.append(pre\_tuple)

            #Resets the selection table for the next round

            mycursor.execute("DELETE FROM selection")

            mydb.commit()

            #First we take the list of all the executed function and then conve

# rt it into a JSON string

            #We then take this string and then add to a tuple to be inserted

# into user action table

**if** len(list1) > 0:

                json\_list = json.dumps(list1)

                user\_actions\_data = (json\_list,)

                mycursor.execute(user\_actions, user\_actions\_data)

                mydb.commit()

**else**:

**pass**

        FileCart.FileCart\_Update(self) #Update the fileCart view

        mycursor.execute("DELETE FROM undo\_history") # Reset the undo history table to prevent collapsation

#This is the class that contains both the undo and redo functions

#P.S It is some very complicated code logic wise :)

**class** undo\_redo\_functions():

    #This is the undo function and is called whenever an undo operation must be executed

**def** undo(self):

        #First check to see if there is any undo history to undo... or else sho

#w error message

        mycursor.execute(action\_check)

        action\_check\_var = mycursor.fetchone()

**if** action\_check\_var[0] == 1:

            #Take the last entered action from the user\_action table and extr

# act it from the tuple

            #Then convert the operations that where executed back into a list

#from a JSON string

            #Also get the the amount of operations to be executed in one go

            mycursor.execute(undo\_formula)

            self.undo\_data\_tuple= mycursor.fetchone()

            self.undo\_data = json.loads(self.undo\_data\_tuple[1])

            self.undo\_operations\_amount = len(self.undo\_data)

            MainScreenvar = sm.get\_screen('main\_screen')

**for** i **in** range (0, self.undo\_operations\_amount):

                #Check to see if operation is a copy operation.If so then we che

#ck to see if it is a file or a directory we then execute

                # a delete operation on the newly copied file

**if** self.undo\_data[i][0] == 1:

**if** self.undo\_data[i][1] == 'File':

                        os.remove(self.undo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

**else**:

                        shutil.rmtree(self.undo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                #If the operation is a cut operation we check here for saftey.In

# case some other value is inserted by mistake.If it is a cut

                # operation we just move the entry back

                #to the original file location

**elif** self.undo\_data[i][0] == 2:

**if** self.undo\_data[i][1] == 'File':

                        shutil.move(self.undo\_data[i][3], self.undo\_data[i][2])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

**else**:

                        shutil.move(self.undo\_data[i][3], self.undo\_data[i][2])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                #If the operation is a delete operation we restore from the recyc

#le bin

**elif** self.undo\_data[i][0] == 3:

                    shutil.move(self.undo\_data[i][3],self.undo\_data[i][2])

                    MainScreenvar.ids.filechooser\_icon.\_update\_files()

**else**:

**return** Errors().there\_was\_a\_problem()

**else**:

**return** Errors().no\_undo\_history()

        mycursor.execute(redo\_insert,(self.undo\_data\_tuple[1],))

        mydb.commit()

        mycursor.execute('DELETE FROM user\_action ORDER BY id DESC LIMIT 1')

        mydb.commit()

    #This is the redo function..The most complicated part of the program logic wise :)

**def** redo(self):

        #First we check if there is anything to redo by checking database

        mycursor.execute(redo\_check)

        undo\_check\_tuple = mycursor.fetchone()

        #If so go on with redo function

        #Get redo operation and check how many functions to execute to redo

**if** undo\_check\_tuple[0] == 1:

            mycursor.execute(redo\_formula)

            self.redo\_data\_tuple = mycursor.fetchone()

            self.redo\_data = json.loads(self.redo\_data\_tuple[1])

            self.redo\_operations\_amount = len(self.redo\_data)

            MainScreenvar = sm.get\_screen('main\_screen')

            #Prepare a list for later insertion into user\_action

            list2 = []

**for** i **in** range (0, self.redo\_operations\_amount):

**if** self.redo\_data[i][0] == 1:#If it is a copy function to redo

                #Simply redo the copy and add the actions to the afor

# mentioned list

**if** self.redo\_data[i][1] == 'File':

                        shutil.copy2(self.redo\_data[i][2], self.redo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (1, 'File', self.redo\_data[i][2], self.redo\_data[i][3])

                        list2.append(pre\_tuple)

**else**:

                        shutil.copytree(self.redo\_data[i][2], self.redo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (1, 'Directory', self.redo\_data[i][2], self.redo\_data[i][3])

                        list2.append(pre\_tuple)

**elif** self.redo\_data[i][0] == 2:# if it is a cut function

**if** self.redo\_data[i][1] == 'File':

                        shutil.move(self.redo\_data[i][2], self.redo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (2, 'File', self.redo\_data[i][2], self.redo\_data[i][3])

                        list2.append(pre\_tuple)

**else**:

                        shutil.move(self.redo\_data[i][2], self.redo\_data[i][3])

                        MainScreenvar.ids.filechooser\_icon.\_update\_files()

                        pre\_tuple = (2, 'Directory', self.redo\_data[i][2], self.redo\_data[i][3])

                        list2.append(pre\_tuple)

**else**: # If it is a delete operation

                    shutil.move(self.redo\_data[i][2], self.redo\_data[i][3])

                    MainScreenvar.ids.filechooser\_icon.\_update\_files()

                    pre\_tuple = (3, 'both', self.redo\_data[i][2], self.redo\_data[i][3])

                    list2.append(pre\_tuple)

            #Convert the list into a json string to re inserted into user actions

            json\_list = json.dumps(list2)

            user\_actions\_data = (json\_list,)

            mycursor.execute(user\_actions, user\_actions\_data)

            mydb.commit()

            #We then delete the respective token from the undo\_history

            mycursor.execute("DELETE FROM undo\_history ORDER BY id DESC LIMIT 1 ")

            mydb.commit()

###This class handles updating the user interface with the correct amount of icons and relevant data

###for the filecart

**class** FileCart():

### This function is called whenever the file cart has to be updated.It rea###ds the selection table and generates the required elements to be displ###ayed in the list

**def** FileCart\_Update(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        #We then clear the container of all widgets for a fresh batch of list it

#ems

        MainScreenvar.ids.back\_layer.ids.container.clear\_widgets()

        #First we get the amount of rows in the selection tables to see how

much times we must itterate

        mycursor.execute("SELECT COUNT(name) FROM selection")

        self.cart\_icons\_amount  = mycursor.fetchone()

        #Then we get the actual data that we must use to create the elemen

#ts

        mycursor.execute("SELECT \* FROM selection")

        self.filecart\_data = mycursor.fetchall()

**for** i **in** range (self.cart\_icons\_amount[0]):

            #Check to see if the current element is a file or a directory

**if** self.filecart\_data[i][2] == "File":

                #Check too see if a copy operation was executed on the file

**if** self.filecart\_data[i][3] == '1':

                    #Create the list item to be insert into scroll view

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                   secondary\_text = self.filecart\_data[i][1],

                                                   tertiary\_text = "Copied",

                                                   )

                    #Create the icon

                    self.item2 = IconLeftWidget(icon = 'file',

                                                pos= self.pos,

                                                size = self.size,

                                                )

                    self.item.bind(on\_release = self.customcallback\_function)

                    #Add icon to list item

                    self.item.add\_widget(self.item2)

                    #Add total list item with icon to the scroll view

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

                #Check too see if a cut operation was executed on the file

**elif** self.filecart\_data[i][3] == '2':

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                      secondary\_text = self.filecart\_data[i][1],

                                                      tertiary\_text = "Cut",

                                                      )

                    self.item2 = IconLeftWidget(icon = 'file',

                                                pos= self.pos,

                                                size = self.size

                                                )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item.add\_widget(self.item2)

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

                #Check to see if a delete operation was executed on the file

**elif** self.filecart\_data[i][3] == '3':

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                   secondary\_text = self.filecart\_data[i][1],

                                                   tertiary\_text = "Recycled",

                                                   )

                    self.item2 = IconLeftWidget(icon = 'file',

                                                pos= self.pos,

                                                size = self.size

                                                )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item.add\_widget(self.item2)

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

**else**:

**if** self.filecart\_data[i][3] == '1':

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                       secondary\_text = self.filecart\_data[i][1],

                                                       tertiary\_text = "Copied",

                                                       )

                    self.item2 = IconLeftWidget(icon = 'folder',

                                                pos= self.pos,

                                                size = self.size

                                                )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item.add\_widget(self.item2)

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

**elif** self.filecart\_data[i][3] == '2':

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                      secondary\_text = self.filecart\_data[i][1],

                                                      tertiary\_text = "Cut",

                                                      )

                    self.item2 = IconLeftWidget(icon = 'folder',

                                                pos= self.pos,

                                                size = self.size

                                               )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item.add\_widget(self.item2)

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

**elif** self.filecart\_data[i][3] == '3':

                    self.item = ThreeLineIconListItem(text = str(self.filecart\_data[i][0]),

                                                      secondary\_text = self.filecart\_data[i][1],

                                                      tertiary\_text = "Recycled",

                                                      )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item2 = IconLeftWidget(icon = 'folder',

                                                pos= self.pos,

                                                size = self.size

                                                )

                    self.item.bind(on\_release = self.customcallback\_function)

                    self.item.add\_widget(self.item2)

                    MainScreenvar.ids.back\_layer.ids.container.add\_widget(self.item)

###Define a class for all the error handling and displaying popup error messages

**class** Errors(MDDialog):

    #This error message is displayed when no valid selection is selected bu#t the copy or cut function has been called

    #Also called when a user selects a drive which is not a valid selection

**def** passwd\_not\_crct(self):

        self.pnc\_error = MDDialog(

                        text="Hey you got the password or the userid wrong...try again",

                        radius=[30, 30, 30, 30],)

        self.pnc\_error.open()

**def** file\_not\_selected(self):

        self.fns\_error = MDDialog(

                        text="Please Select a file or a folder to move to the filecart",

                        radius=[30, 30, 30, 30],)

        self.fns\_error.open()

    #This error message is displayed when a selection is already present in the filecart but user is attempting to add again

**def** file\_already\_exists(self):

        self.fae\_error = MDDialog(

                        text="This file/Directory has already been added to the Filecart",

                        radius=[30, 30, 30, 30], )

        self.fae\_error.open()

    #This error mesage is displayed when the paste function is called without any entries present in the filecart

**def** no\_file\_in\_cart(self):

        self.nfinc\_error = MDDialog(

                        text="There are no files in the filecart, please select a file",

                        radius=[30, 30, 30, 30], )

        self.nfinc\_error.open()

    #This error is displayed if any internal error occurs where an unknown value is read from the database

**def** there\_was\_a\_problem(self):

        self.twap\_error = MDDialog(

                        text="There is a problem. Please close and open the program again",

                        radius=[30, 30, 30, 30],)

        self.twap\_error.open()

    #This error is displayed when there is nothing left for the undo operati#on

**def** no\_undo\_history(self):

        self.nuh\_error = MDDialog(

                        text="There is nothing left to undo",

                        radius=[30, 30, 30, 30],)

        self.nuh\_error.open()

    #This error is displayed when there is an error in the creation of a fold#er

**def** folder\_cannot\_be\_created(self):

        self.fcbc\_error = MDDialog(

                        text="The Folder Could Not Be Created",

                        radius=[30, 30, 30, 30],)

        self.fcbc\_error.open()

    #This error is displayed when a user must select something to rename

**def** select\_to\_rename(self):

        self.str\_error = MDDialog(

                        text="Select Something to Rename it",

                        radius=[30, 30, 30, 30],)

        self.str\_error.open()

    #This error is displayed when the file cannot be renamed

**def** element\_cannot\_be\_renamed(self):

        self.ecbc\_error = MDDialog(

                        text="The selected item cannot be renamed",

                        radius=[30, 30, 30, 30],)

        self.ecbc\_error.open()

###This class deals with searching and filtering

**class** search():

    #The function that is called whenever the user hits ctrl + s and it searches for the required file

**def** filters(self, text):

        MainScreenvar =sm.get\_screen('main\_screen')

        MainScreenvar.ids.filechooser\_icon.filters.clear()

        #Just in case nothing is searched for we need to return default view

**if** text == '':

            MainScreenvar.ids.filechooser\_icon.filters.clear()

            MainScreenvar.ids.filechooser\_icon.\_update\_files()

**else**:

            MainScreenvar.ids.filechooser\_icon.filters.append("\*" + text)

###This class contains the code that changes the path of the main icon fi###lechooser based on what is clicked in the list file chooser

**class** new\_path\_filechooser():

**def** update\_filechooser(self,newpath):

       MainScreenvar =sm.get\_screen('main\_screen')

       MainScreenvar.ids.filechooser\_icon.path = newpath[0]

###This class deals with loading the machines drives into the drive bay

**class** drivebay():

    #This function is called on execution to load all drives in system into t#he drive bay

**def** load\_drivebay(self):

        #get the names of the drives in the system

        self.drives = win32api.GetLogicalDriveStrings()

        self.drives = self.drives.split('\000')[:-1]

**for** i **in** self.drives:#iterate over no of drives

            self.disk\_type = win32file.GetDriveType(i)#get the drive type

**if** self.disk\_type == 3:#If hard disk

                self.disk\_usage = round(shutil.disk\_usage(i)[2]/1024/1024/1024, 2)

                self.item = ThreeLineAvatarListItem(text=i,

                                                secondary\_text = win32api.GetVolumeInformation(i)[0],

                                                tertiary\_text = str(self.disk\_usage) + "Gb is left")

                self.item2 = IconLeftWidget(icon = 'harddisk', pos=self.pos,size=self.size)

                self.item.add\_widget(self.item2)

                self.item.bind(on\_release = self.callback\_function)

                self.ids.drivers.ids.drivebay.add\_widget(self.item)

**elif** self.disk\_type == 2:# If pendrive or removable medium

                self.disk\_usage = round(shutil.disk\_usage(i)[2]/1024/1024/1024, 2)

                self.item = ThreeLineAvatarListItem(text=i,

                                                secondary\_text = win32api.GetVolumeInformation(i)[0],

                                                tertiary\_text = str(self.disk\_usage) + "Gb is left")

                self.item2 = IconLeftWidget(icon = 'usb-flash-drive', pos=self.pos,size=self.size)

                self.item.add\_widget(self.item2)

                self.item.bind(on\_release = self.callback\_function)

                self.ids.drivers.ids.drivebay.add\_widget(self.item)

**elif** self.disk\_type == 3: #If cd drive

                self.disk\_usage = round(shutil.disk\_usage(i)[2]/1024/1024/1024, 2)

                self.item = ThreeLineAvatarListItem(text=i,

                                                secondary\_text = win32api.GetVolumeInformation(i)[0],

                                                tertiary\_text = str(self.disk\_usage) + "Gb is left")

                self.item2 = IconLeftWidget(icon = 'album', pos=self.pos,size=self.size)

                self.item.add\_widget(self.item2)

                self.item.bind(on\_release = self.callback\_function)

                self.ids.drivers.ids.drivebay.add\_widget(self.item)

**elif** self.disk\_type == 0: #If drive is corrupt

                self.disk\_usage = round(shutil.disk\_usage(i)[2]/1024/1024/1024, 2)

                self.item = ThreeLineAvatarListItem(text=i,

                                                secondary\_text = win32api.GetVolumeInformation(i)[0],

                                                tertiary\_text = "Something is wrong with this drive")

                self.item2 = IconLeftWidget(icon = 'alert-circle', pos=self.pos,size=self.size)

                self.item.add\_widget(self.item2)

                self.item.bind(on\_release = self.callback\_function)

                self.ids.drivers.ids.drivebay.add\_widget(self.item)

###This class deals with displaying the correct folder from shortcuts tab

**class** Shortcuts():

    #Go to Desktop(get the current user id and go to that respective folders)

**def** desktop(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        username = getpass.getuser()

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Users\\' + username  + '\\Desktop'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Users\\' + username  + '\\Desktop'

    #Go to Downloads

**def** downloads(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        username = getpass.getuser()

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Users\\' + username  + '\\Downloads'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Users\\' + username  + '\\Downloads'

    #Go to Documents

**def** documents(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        username = getpass.getuser()

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Users\\' + username  + '\\Documents'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Users\\' + username  + '\\Documents'

    #Go to Images

**def** images(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        username = getpass.getuser()

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Users\\' + username  + '\Pictures'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Users\\' + username  + '\\Pictures'

    #Go to Videos

**def** videos(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        username = getpass.getuser()

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Users\\' + username  + '\\Videos'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Users\\' + username  + '\\Videos'

    #Go to Recycle Bin

**def** recycle\_bin(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        MainScreenvar.ids.filechooser\_icon.path = 'C:\\Recycle bin'

        MainScreenvar.ids.folder\_selector.path = 'C:\\Recycle bin'

###Define The Classes for the various Screens and also screen Manager

**class** LoginScreen(Screen):

    loginvar= Login()

**pass**

**class** MainScreen(Screen):

**def** \_\_init\_\_(self, \*\*kwargs):

        super(MainScreen, self).\_\_init\_\_(\*\*kwargs)

        Window.bind(on\_key\_down=self.\_on\_keyboard\_down)

        drivebay.load\_drivebay(self)

    #This functions handles executing the necesaary function on click of t

# he quick actions button

**def** callback(self, instance):

        MainScreenvar =sm.get\_screen('main\_screen')

**if** instance.icon == 'delete':

            Select\_Functions.delete\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

**elif** instance.icon == 'content-copy':

            Select\_Functions.copy\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

**elif** instance.icon == 'content-cut':

            Select\_Functions.cut\_select(self, MainScreenvar.ids.filechooser\_icon.selection)

**elif** instance.icon == 'rename-box':

            MainApp.show\_rename\_element(self)

**elif** instance.icon == 'folder-plus':

            MainApp.show\_new\_folder(self)

    #This function deals with executing functions when you press shortcut  #keys on the keyboard

**def** \_on\_keyboard\_down(self, instance, keyboard, keycode, text, modifiers,):

        MainScreenvar =sm.get\_screen('main\_screen')

        #Copy

**if** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='c':

            Select\_Functions.copy\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'c':

            Select\_Functions.copy\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

        #Cut

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='x':

            Select\_Functions.cut\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'x':

            Select\_Functions.cut\_select(self,MainScreenvar.ids.filechooser\_icon.selection)

        #execute

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='e':

            Execute\_Functions.execute(self,MainScreenvar.ids.filechooser\_icon.path)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'e':

            Execute\_Functions.execute(self, MainScreenvar.ids.filechooser\_icon.path)

        #Undo

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='z':

            undo\_redo\_functions.undo(self)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'z':

            undo\_redo\_functions.undo(self)

        #Redo

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='y':

            undo\_redo\_functions.redo(self)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'y':

            undo\_redo\_functions.redo(self)

        #Delete

**elif** keycode == 76:

            Select\_Functions.delete\_select( self, MainScreenvar.ids.filechooser\_icon.selection)

        #New Folder

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='n':

            MainApp.show\_new\_folder(self)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'n':

            MainApp.show\_new\_folder(self)

        #Rename

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='r':

            MainApp.show\_rename\_element(self)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 'r':

            MainApp.show\_rename\_element(self)

        #Search

**elif** len(modifiers) > 0 **and** modifiers[0] == 'ctrl' **and** text =='s':

            search.filters(self, self.ids.search.text)

**elif** len(modifiers) > 1 **and** modifiers[1] == 'ctrl' **and** text == 's':

            search.filters(self, self.ids.search.text)

#This function sets the drive path when user clicks on drive bay element

**def** callback\_function(self,instance):

        MainScreenvar =sm.get\_screen('main\_screen')

        MainScreenvar.ids.filechooser\_icon.path = instance.text

        MainScreenvar.ids.folder\_selector.path = instance.text

**def** customcallback\_function(self,instance):

        MainScreenvar =sm.get\_screen('main\_screen')

        MainScreenvar.ids.filechooser\_icon.path = instance.secondary\_text

        MainScreenvar.ids.folder\_selector.path = instance.secondary\_text

  #These are just some variables that are used within kivy for

#accesing widgets from kivy easily

    selectionvar = Select\_Functions

    executevar = Execute\_Functions

    undo\_redo\_var = undo\_redo\_functions

    file\_cart\_var = FileCart

    updatevar = new\_path\_filechooser

    shortcuts\_var = Shortcuts

### Assign the Screen Manager to a global variable so that it can be calle###d anywhere within the program

sm = ScreenManager()

###Building the app and loading KV file in program

**class** MainApp(MDApp):

    #Assign the data for the quick actions button

    data = {

        'folder-plus': 'New Folder',

        'rename-box': 'Rename',

        'content-cut': 'Cut',

        'content-copy': 'Copy',

        'delete': 'Delete'

    }

    #The main app building process

**def** build(self):

        Builder.load\_file("file\_explorer.kv")

        #Add the login screen and main screen to the screen manager

        sm.add\_widget(LoginScreen(name='login\_screen'))

        sm.add\_widget(MainScreen(name='main\_screen'))

**return** sm # return our screen Manager

    #This function is called when you want to create a new folder

**def** show\_new\_folder(self):

        self.folder\_name = MDTextField( hint\_text = "Enter A valid Folder name",

                                       required = True)

        self.dialog = MDDialog(

            title="New Folder",

            type="custom",

            content\_cls=self.folder\_name,

            radius = (30,30,30,30),

            buttons=[

                MDFlatButton(

                    text="Create",on\_press=**lambda** a:MainApp.create\_folder(self),

                ),

            ],

        )

        self.dialog.open()

**return** self.folder\_name

    #Actually creates the new folder

**def** create\_folder(self):

        self.dialog.dismiss()

        MainScreenvar =sm.get\_screen('main\_screen')

**try**:

            os.mkdir(os.path.join(MainScreenvar.ids.filechooser\_icon.path, self.folder\_name.text))

            MainScreenvar.ids.filechooser\_icon.\_update\_files()

**except**:

            Errors.folder\_cannot\_be\_created(self)

**def** show\_rename\_element(self):

        MainScreenvar = sm.get\_screen('main\_screen')

        self.element\_name = MDTextField( hint\_text = "Enter A valid new name",

                                       required = True)

        self.dialog = MDDialog(

            title="Rename",

            type="custom",

            content\_cls=self.element\_name,

            radius = (30,30,30,30),

            buttons=[

                MDFlatButton(

                    text="Rename",on\_press=**lambda** a:MainApp.rename\_element(self),

                ),

            ],

        )

**if** len(MainScreenvar.ids.filechooser\_icon.selection) == 1:

            self.dialog.open()

**return** self.element\_name

**else**:

            Errors.select\_to\_rename(self)

    #This function is called whenever an element needs to be renamed

**def** rename\_element(self):

**try**:

            MainScreenvar = sm.get\_screen('main\_screen')

            self.dialog.dismiss()

            self.from\_location = MainScreenvar.ids.filechooser\_icon.selection[0]

            self.to\_location = os.path.join(os.path.split(self.from\_location)[0], self.element\_name.text)

            self.to\_location\_with\_ext = self.to\_location + os.path.splitext(self.from\_location)[1]

            os.rename(self.from\_location, self.to\_location\_with\_ext)

            MainScreenvar.ids.filechooser\_icon.\_update\_files()

**except**:

            Errors().element\_cannot\_be\_renamed(self)

###The actual running code of the program

#We create a folder for the program recycle bin if it is not already created

**if** os.path.isdir('C:\Recycle Bin') == True:

**pass**

**else**:

    os.mkdir('C:\Recycle Bin')

### The very small run code that makes all of this possible

Window.size = (1366, 768)

MainApp().run()

###Clears all the tables after the gui is closed

mycursor.execute("DELETE FROM SELECTION")

mydb.commit()

mycursor.execute("DELETE FROM user\_action")

mydb.commit()

mycursor.execute("DELETE FROM undo\_history")

mydb.commit()

#KEE

Kivy Code:

<**Loginscreen**>:

**canvas**:

Color:

rgba:0.05,0.05,0.05,1

Rectangle:

pos:self.pos

size:self.size

Color:

rgba:0.1,0.1,0.1,1

RoundedRectangle:

size:(root.width-10, root.height-10)

pos:(5,5)

radius: (30,30,30,30)

Color:

rgba: 0.15, 0.15, 0.15, 1

RoundedRectangle:

pos:(450,150)

size:(500,400)

radius: (30,30,30,30)

**MDLabel**:

pos:670,100

text:"Login"

text\_size: 50,50

size:self.size

**MDTextFieldRound**:

id:userid

icon\_left: 'account'

normal\_color: .1,.1,.1,1

color\_active: app.theme\_cls.primary\_color

pos\_hint:{"top": .52, "right": 0.635}

size\_hint\_x:.25

**MDTextFieldRound**:

id: passwd

icon\_left: 'key-variant'

normal\_color: .1,.1,.1,1

color\_active: app.theme\_cls.primary\_color

pos\_hint:{"top": .40, "right": 0.635}

size\_hint\_x:.25

**MDRectangleFlatButton**:

text: 'Explore'

font\_size: 10

on\_press:root.loginvar.loginop(userid.text, passwd.text )

pos\_hint:{"top": .29, "right": 0.54}

<**FileCart@ScrollView**>:

pos:root.pos

size:root.size

MDList:

id:container

<**MyBackdropFrontLayer@FileChooserIconView**>:

backdrop:None

pos\_hint: {"top": 1}

size:self.size

<**MyBackdropBackLayer@FileCart**>:

pos\_hint: {"center\_x": .5, "center\_y": .38}

size: root.size

<**Shortcuts@GridLayout**>

cols:2

padding: 20,20,20,20

spacing: 40,40

**MDRoundFlatIconButton**:

text:"Desktop"

icon: "monitor"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.desktop(self)

**MDRoundFlatIconButton**:

text:"Download"

icon: "folder-download"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.downloads(self)

**MDRoundFlatIconButton**:

text:"Document"

icon: "file-document"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.documents(self)

**MDRoundFlatIconButton**:

text:"images"

icon: "folder-download"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.images(self)

**MDRoundFlatIconButton**:

text:"Videos"

icon: "movie-open"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.videos(self)

**MDRoundFlatIconButton**:

text:"Recyle Bin"

icon: "delete"

on\_press: root.parent.parent.parent.parent.parent.parent.shortcuts\_var.recycle\_bin(self)

<**Drive\_bay@ScrollView**>:

MDList:

id:drivebay

<**Drives@Drive\_bay**>:

pos\_hint: {"center\_x": .5, "center\_y": .237}

size\_x: root.size\_x

size\_y:root.size\_y

<**MainScreen**>:

**FloatLayout**:

canvas:

Color:

rgba: 0.05,0.05,0.05, 1

Rectangle:

pos: root.pos

size: root.size

Color:

rgba: 0.1, 0.1, 0.1, 1

RoundedRectangle:

size:(root.width-10, root.height-10)

pos:(5,5)

radius: (30,30,30,30)

Color:

rgba: 0.15, 0.15, 0.15, 1

RoundedRectangle:

pos:(10,380)

size:(420,root.height-390)

radius: (30,30,30,30)

**BoxLayout**:

size\_hint\_y : 0.2

size\_hint\_x : 0.57

pos\_hint: {'right': 0.97, "top": 0.215}

**MDTextFieldRound**:

id:search

icon\_left: 'folder-search'

normal\_color: 0.15,0.15,0.15,1

color\_active: app.theme\_cls.primary\_color

elevation: 20

**BoxLayout**:

size\_hint\_y : 0.92

size\_hint\_x : 0.6

pos\_hint: {'top': 0.985, "right": 0.99}

**MDBackdrop**:

id:backdrop

title: "File Cart"

header\_text: "Browse files"

**MDBackdropFrontLayer**:

**MyBackdropFrontLayer**:

id:filechooser\_icon

backdrop:backdrop

**MDBackdropBackLayer**:

MyBackdropBackLayer:

id: back\_layer

**BoxLayout**:

size\_hint\_y : 0.44

size\_hint\_x : 0.30

pos\_hint: {'top': 0.98, "right": 0.311}

**FileChooserListView**:

id:folder\_selector

on\_selection: root.updatevar.update\_filechooser(self,self.selection)

**BoxLayout**:

size\_hint\_y : 0.48

size\_hint\_x : 0.30

pos\_hint: {'top': 0.50, "right": 0.31}

**MDBackdrop**:

id:secondary\_backdrop

title: "Drives"

header\_text: "Shortcuts"

**MDBackdropFrontLayer**:

Shortcuts:

id:Shortcuts

backdrop:backdrop

**MDBackdropBackLayer**:

Drives:

id:drivers

**BoxLayout**:

**MDFloatingActionButtonSpeedDial**:

data: app.data

rotation\_root\_button: True

label\_text\_color: app.theme\_cls.primary\_color

color\_icon\_stack\_button: app.theme\_cls.primary\_color

color\_icon\_root\_button: app.theme\_cls.primary\_color

icon: 'pencil'

hint\_animation:True

bg\_color\_root\_button: 0.1,0.1,0.1,1

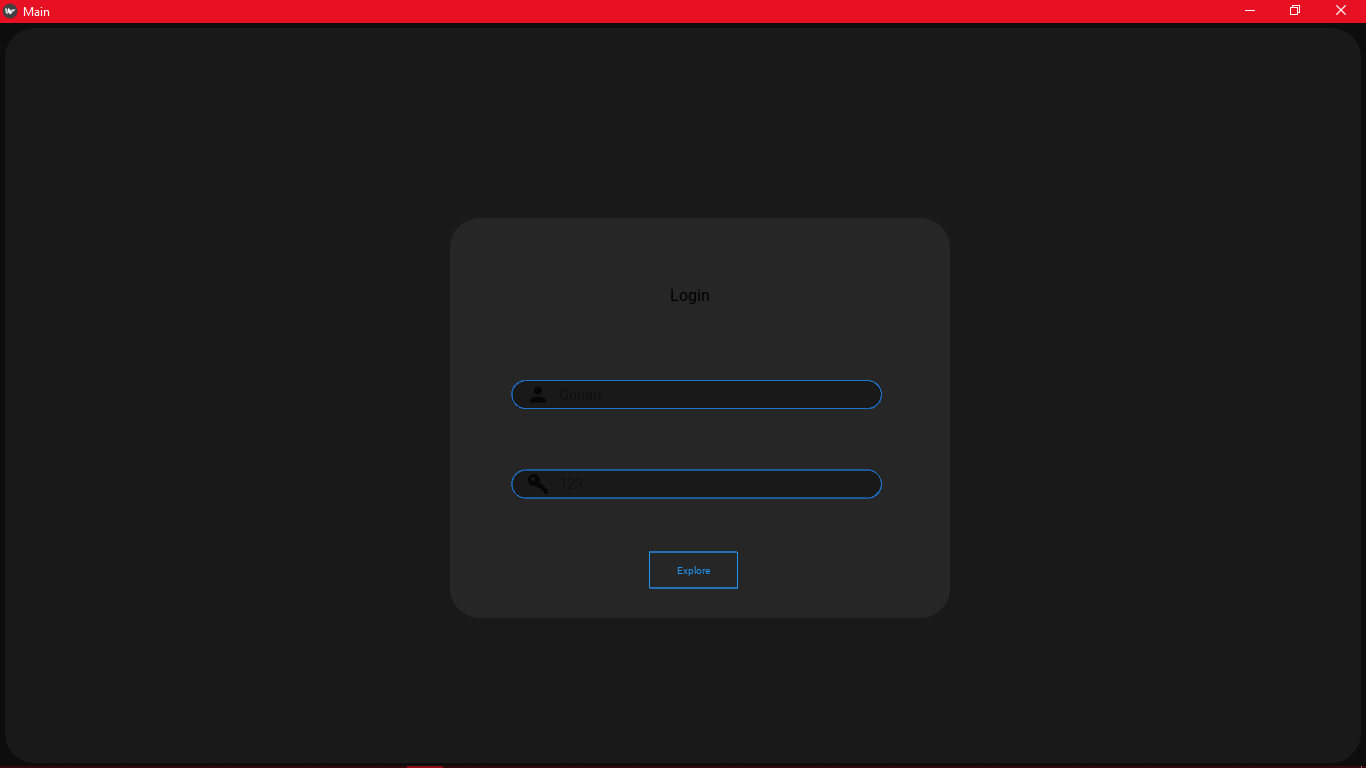
bg\_color\_stack\_button: 0.15,0.15,0.15,1

bg\_hint\_color: 0.1,0.1,0.1,1

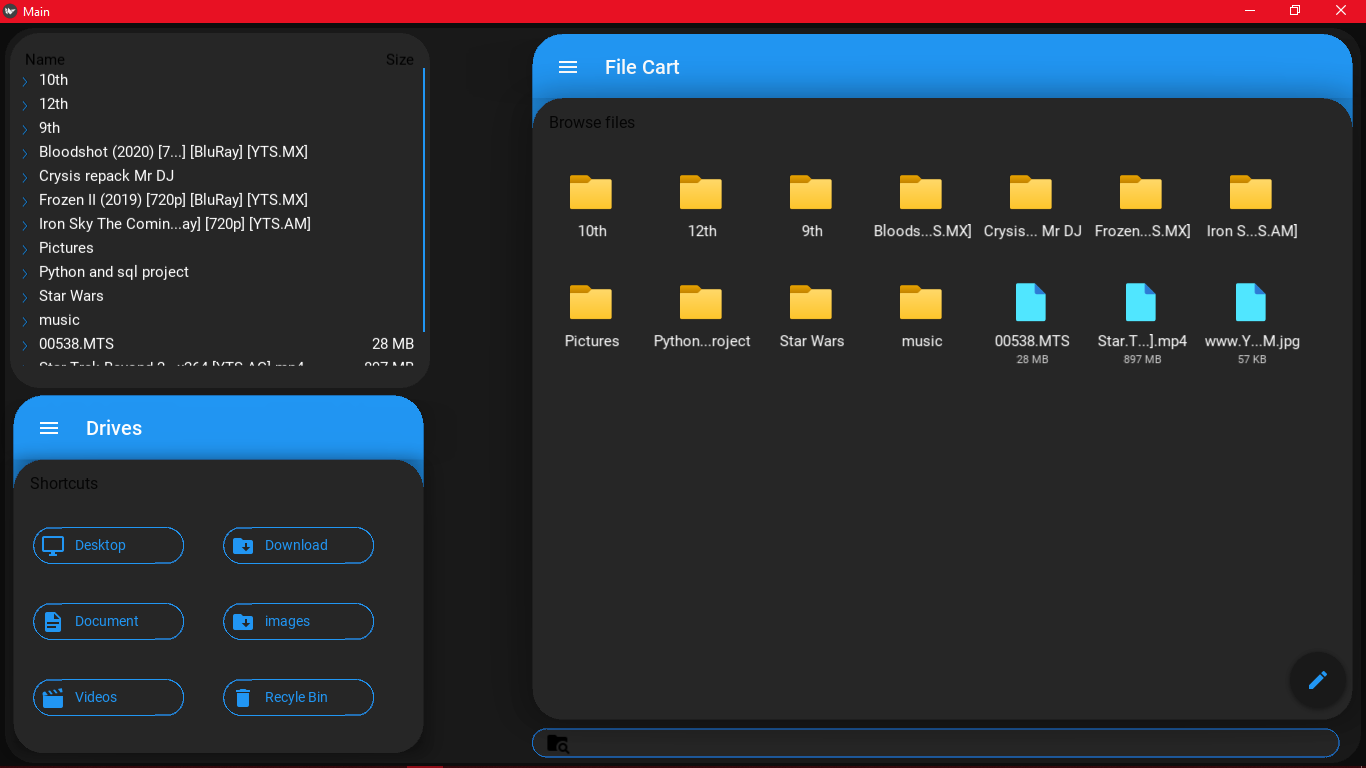
callback:root.callback

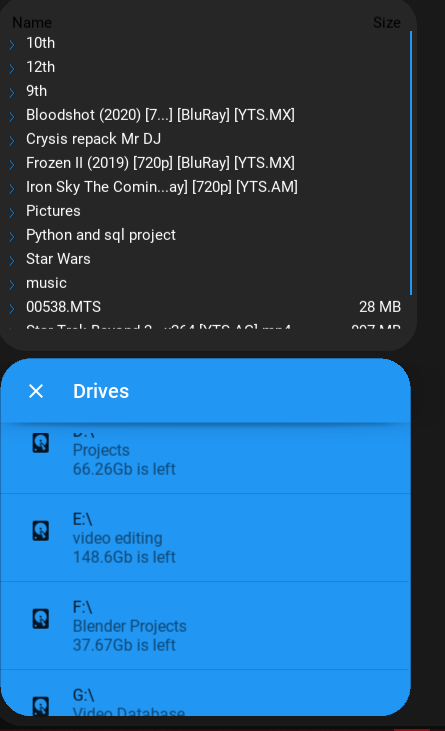
**Output:**

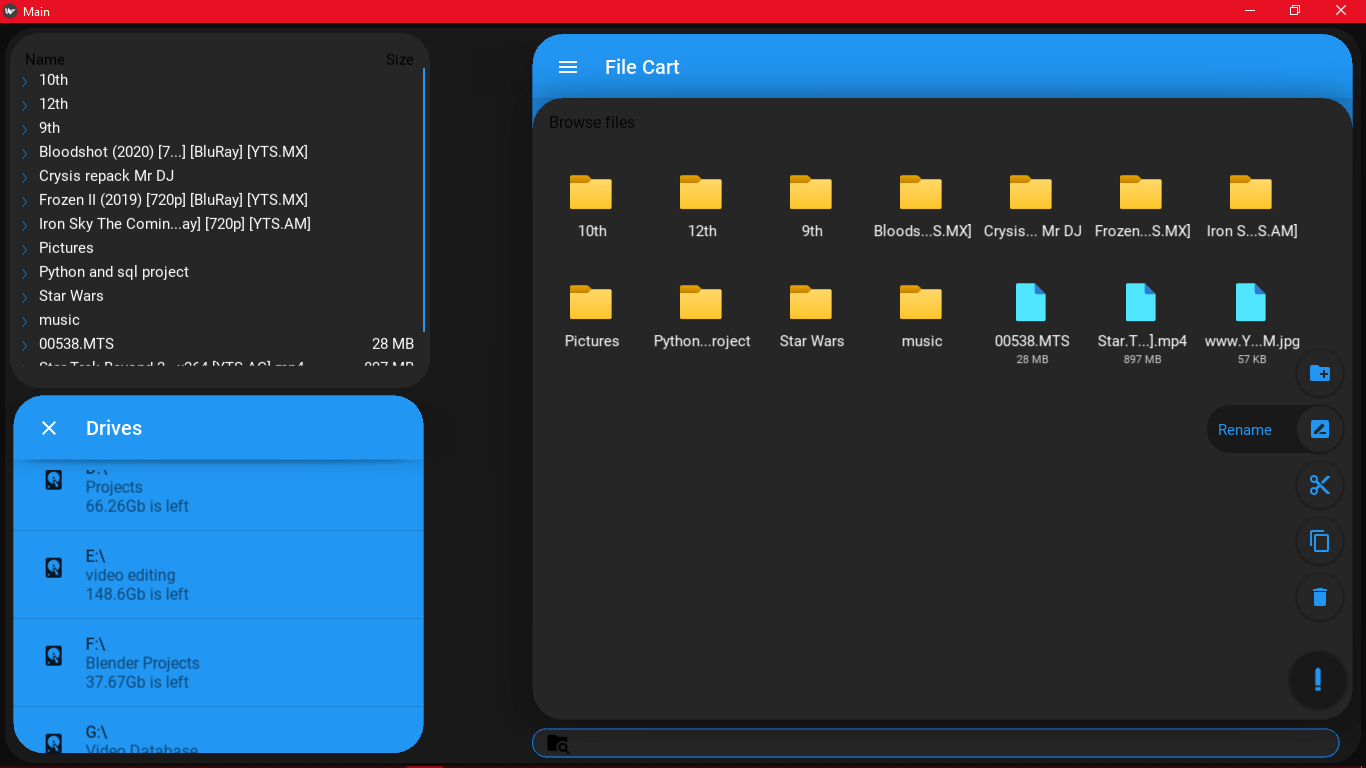
This is the login Screen. The currently set password is 123.

****

This is the main view of the file explorer. On the right side is your normal icon view and on the left or the shortcuts and list view



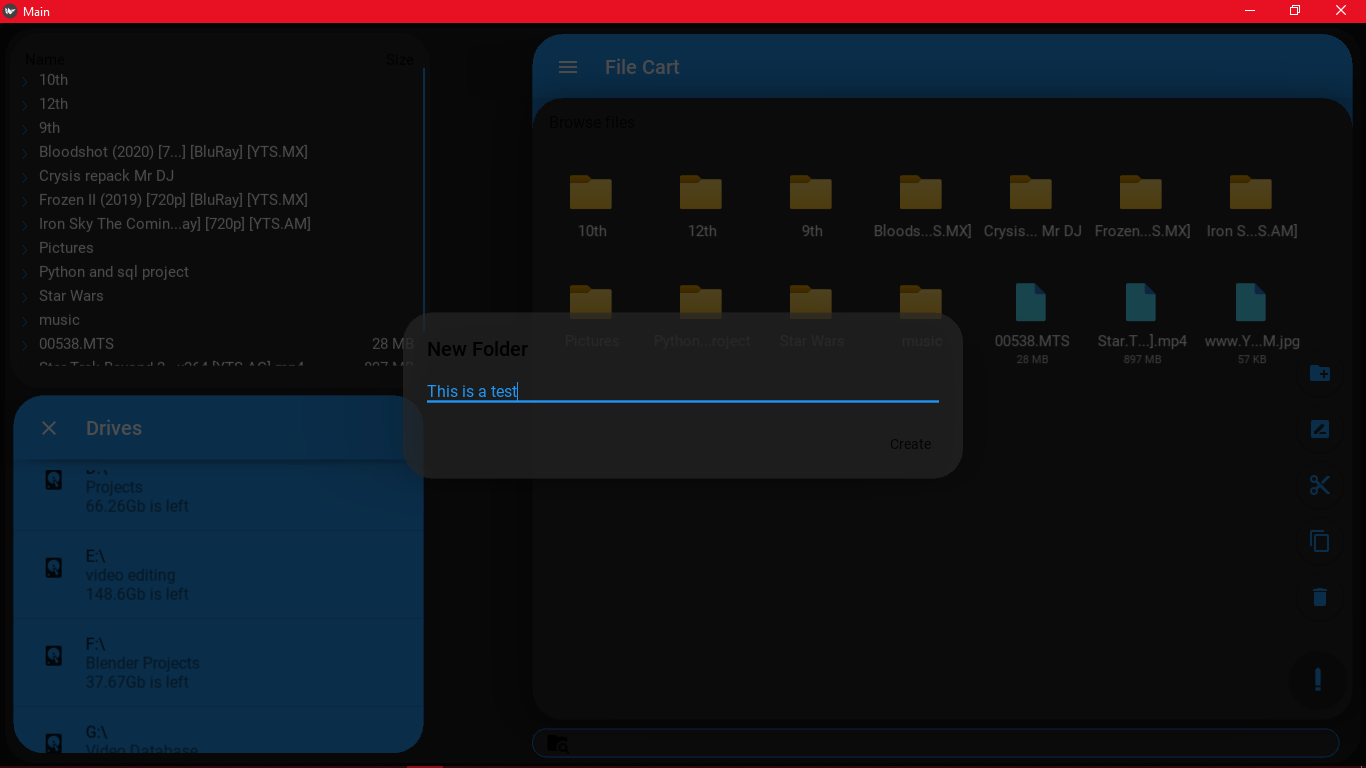
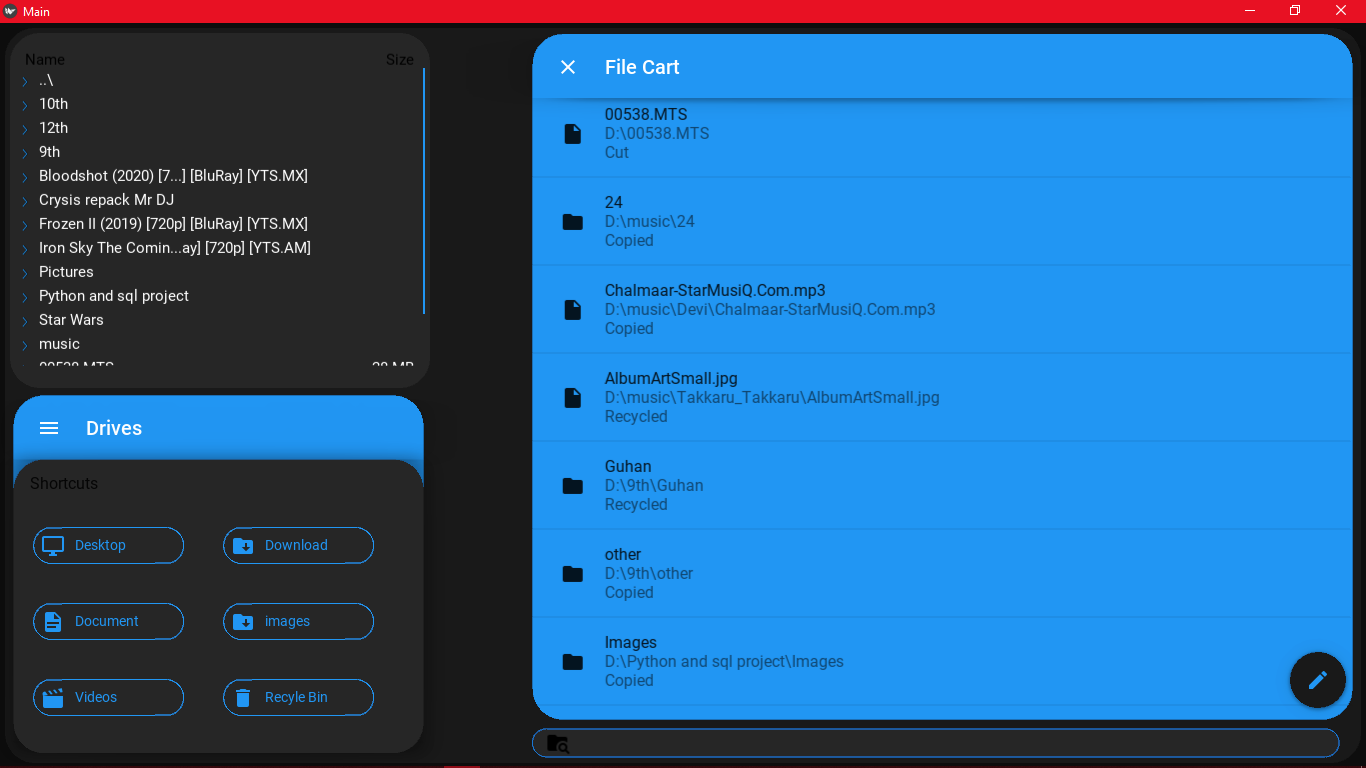




On the far right, the quick access menue is shown expanded with all of its functions available.Tooltips about the function are available on mousover the respective icon. The action can be executed by simply clicking on the icon

This is the left most view of the file explorer. When the top of the shortcuts menue is clicked it expands to show all the drives along with their name and how much gb is left on them

An example of one of the many popups in the program. This one is shown when a new folder needs to be created



This is the filecart. It can be accesed by clicking at the top of drop down menue to expand the interface. Here may files from different locations are shown along with they operations executed on them. When ctrl+e is pressed all respective operations are executed.Clicking on any elements takes you directly to that file.It is also a scrollable view.

**Bibliography:**

<https://kivymd.readthedocs.io/en/latest/?badge=latest>

<https://kivy.org/doc/stable/>

<https://www.geeksforgeeks.org/kivy-tutorial/>

<https://pythonprogramming.net/kivy-application-development-tutorial/>

<https://www.geeksforgeeks.org/os-module-python-examples/>

<https://www.geeksforgeeks.org/python-shutil-copy-method/>

<https://www.geeksforgeeks.org/python-shutil-copy2-method/>

<https://www.geeksforgeeks.org/python-os-rename-method/>

<https://www.geeksforgeeks.org/python-os-makedirs-method/>

<https://stackoverflow.com/questions/62262896/pos-hint-property-not-working-properly-in-kivymd-mdbackdrop-python>

<https://stackoverflow.com/questions/62069611/how-to-make-kivy-file-chooser-auto-refresh-in-python>

<https://stackoverflow.com/questions/62001708/how-can-you-change-the-kivymd-file-explorer-in-python-so-that-a-double-click-open>

<https://www.youtube.com/watch?v=x7SwgcpACng&list=PLB5jA40tNf3tRMbTpBA0N7lfDZNLZAa9G>

<https://stackoverflow.com/questions/4251124/inserting-json-into-mysql-using-python>

<https://scotch.io/tutorials/working-with-json-in-mysql>

**Conclusion**