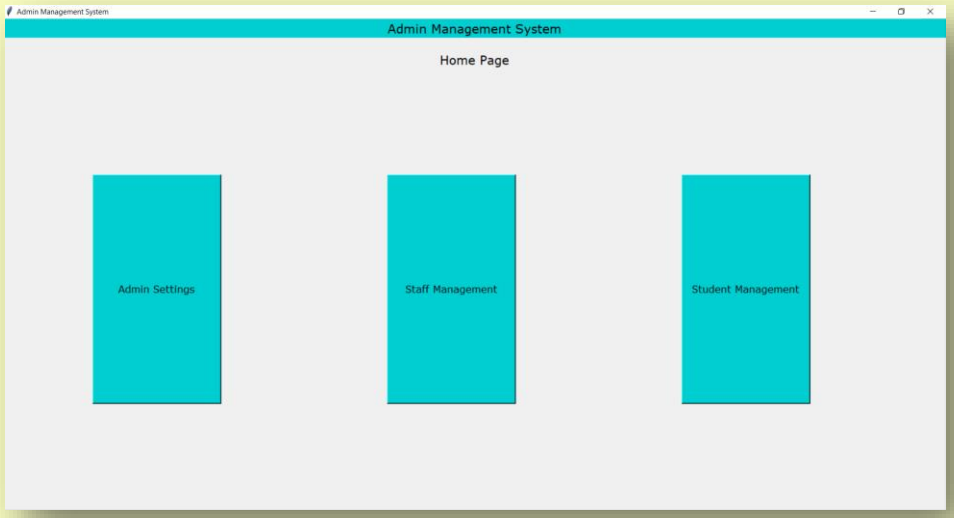


ADMIN MANAGEMENT SYSTEM

IT WORKSHOP – 1 PROJECT

PARTICIPANTS:

1) Aman Verma (BT19CSE125)
2) Shubham Pandit (BT19CSE119)
3) Anuj Kapse (BT19CSE129)
4) Vanshika Shukla (BT19CSE105)
5) Astitva (BT19CSE122)

Computer Science and Engineering

Under the Guidance of: Dr. Mayuri Digalwar.

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY (IIIT), NAGPUR

ACKNOWLEDGEMENT

IT'S A MATTER OF PLEASURE AND PRIVILEGE FOR US TO ACKNOWLEDGE OUR PROFOUND GRATITUDE TOWARDS OUR COURSE COORDINATOR AND PROJECT GUIDE " **DR. MAYURI DIGALWAR** ", FOR GIVING US A GOLDEN OPPORTUNITY TO UNDERTAKE THIS PROJECT - " **ADMIN MANAGEMENT SYSTEM** " AND HER CONTINUOUS GUIDANCE THAT HELPED ME TO COMPLETE THIS PROJECT.

WE WOULD ALSO LIKE TO EXTEND OUR THANKS AND GRATITUDE TOWARDS ALL THE GROUP MEMBERS WHO HAVE CONTINUOUSLY WORKED ON GROUND IN ORDER TO ADD TO THIS PROJECT.

AT LAST, WE WOULD LIKE TO EXTEND OUR THANKS TOWARDS EVERYONE WHO HAVE INVOLVED AND SUPPORTED ME IN COMPLETING THIS PROJECT.

- GROUP -1

INDEX

Sr. No.	Work by	Worked on
1.	Aman Verma	Database Management :: Function write_to_csv(), Function search(), Function deleteRecord, Function modify_details(), Function get_details() PDF (Marksheet) Class, Function create_pdf() ID Card Generator
2.	Vanshika Shukla	Data File Management, Database Creation ,Helper functions Application Testing
3.	Anuj Kapse	G.U.I :: Class ReportCart(), Linker Class :: Class Student(), Application Testing
4.	Astitva	G.U.I. :: Class EnterMarks(),Helper functions Application Testing
5.	Shubham Pandit	G.U.I :: Class MainApp(), Class HomePage(), Class AdminSettings(), Class StudentManagement(), Class ViewAllStudent(), Class StaffManagement(), Class ViewAllStaff(), Linker Class :: Class Admin(), Class Staff()

INTRODUCTION

In this age of Technology, efficiency and speed of operation is everything. The requirement of database management is simple: Database Management systems help increase organizational accessibility to data, which in turn helps the end users share the data quickly and effectively across the organization. With keeping such goal in our minds, we have developed our “Admin Management System”

As the name of our project suggests, “Admin Management System” is a tool that can be used to streamline the working of any educational institute and can be used as a complete all-in-one application for various purposes needed for a typical institution.

Our “Admin Management System” can be used for efficient data storage of details for both Students and Staff of an institution. It has all the functions that are necessary regarding a student such as Storing all the details of a student (Such as Name, department, roll no. etc.), Deleting the details of a student, entering marks of students into the database, viewing entire details of all students at once, viewing marks of individual students as well as saving the marks and details of the student as a PDF file.

Similarly, it can be used to store details of staff as well such as their college ID, the previous colleges that they taught at and their email etc. All the basic functions such as adding, deletion and viewing of all the staff are available as well. We can also get their teacher ID using this application, which acts as an ID card for teachers.

Overall, we consider this to be a multi-faceted application and can be used for a variety of functions. This can be improved as well to embed it with more features and take its utility to the next level.

PROJECT STRUCTURE

The project folder contains **3 Folders** and a main program named "**app.py"**.

"app.py" contains the entire tkinter code for the construction of GUI and all the supporting functions are connected via this program file only to execute various operations like reading, writing and saving etc.

The three other folders in the project folder are: -

1. **Assets**
2. **Databases**
3. **PDF**
4. **StaffID**
5. **Supporitng_Programs**

Databases folder contains all the CSV files that are the databases for the system created. It contains **"admin.csv"** for storing the current settings, **"staff.csv"** to store staff details, **"student.csv"** for storing student details and **"studentMarks.csv"** to store marks of students with their roll number. There is also a python file in the folder named "**createDatabase.py"** which created these databases initially i.e. empty files.

The PDF folder is initially empty and is there to store the pdfs created when the **'Save PDF'** button is clicked in the view grade card frame.

The StaffID folder is initially empty and is there to store the generated staff identity card by pressing the **"Get ID"** button in the Staff Management frame.

The **"Supporting_Programs"** folder contains all the programs required by the system to work logically behind the screen i.e. all the backend functions. It contains "**admin.py"** which handles the setting part of the system, database.py for handling the admin settings database, so admin.py and database.py work together to accept data from "**app.py"** and store it to "**admin.csv"**. Now, "**student.py"** and "**staff.py"** handles the student data and staff data respectively and with the help of another program in the folder i.e. "**databasesSS.py"** student data and staff data are stored in the "**student.csv"** and "**staff.csv"** respectively. It also contains "**pdfCreator.py"** which can create a pdf file and save it by accepting a dictionary and a idCardCreator.py which can create a png file and save it by accepting a dictionary.

"**app.py"** imports these programs and calls the functions when required to make the system work harmonically.

WORKING & I/O

❖G.U.I. :

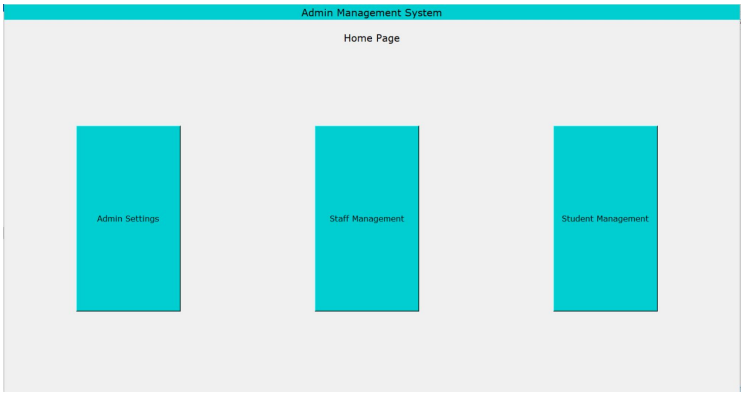
FRAMEWORK USED :: TKINTER

1. Class MainApp(tk.Tk):

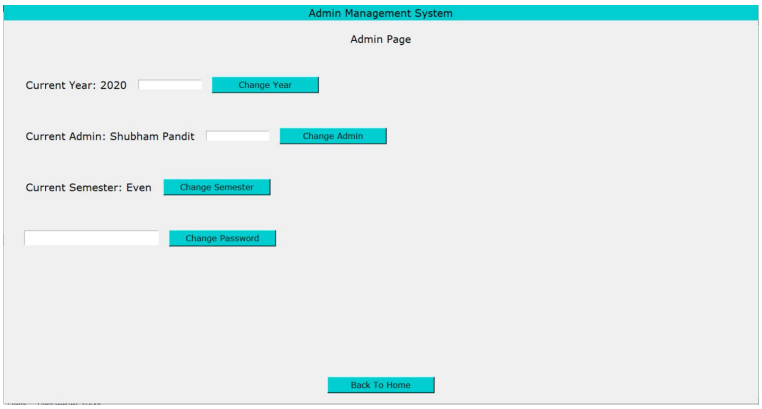
This **class inherits Tk()** class from tkinter and is the main application class which will **run in a loop continuously till the program is running**. It **creates a title** for the window and **a container** in which all the frames will be placed whenever called. There is a **show_frame** function which accepts the container and **raises the frame** accordingly.

2. Class HomePage(tk.Frame):

This **class inherits Frame()** class from tkinter and is the first frame for the system designed. It shows three buttons i.e **Admin Settings, Student Management, Staff Management** and **changes the frame in the MainApp** according to **the button press** by **calling the show_frame** function in the command attribute of the buttons.



3. Class AdminSetting(tk.Frame):



This **class inherits Frame()** class from tkinter and is the **frame for the Admin Settings page** of the system which **contains a header at top** and **then four different frames for four different**

options for **changing year, changing Admin, changing Semester and changing Password** containing **labels, entry fields and buttons** wherever required. Those buttons have **commands which call functions in admin.py** and then **update the labels to show the current values** from the **admin.csv file** using **functions from admin.py.**

4. Class StudentManagement(tk.Frame):

This **class inherits Frame()** class from tkinter. Features that are being handled by this class are: **name, roll number, department, year, hostel, name of courses(course1, course2, course3, course4, course5, course6)** of the student.

An attribute outputList is defined which stores a list returned from the search function later. Every component of the program is designed in the constructor of the class. There are three main tkinter frames defined in this class named **insFrame, inputFrame** and **viewFrame** and are placed on the window using .place() function.

a. In the **insFrame**, **two labels are defined** to show "instructions" on the top.

b. In the **inputFrame**, **pairs of Label and Entry are created** and placed in the frame for the **taking input from the user**. Used ttk.Combobox for the dropdown menu. It contains another frame named viewButtonFrame1 which contains 2 buttons for "Add" and "Modify" whose commands are addStudent() and modifyStudent() respectively.

   i. A function named **getInput()** which **takes the current string present in the .Entry() widgets using .get() function** and **stores them in temp variables** and **creates a list** Slist and **returns it**.

   ii. A function named **deleteEntryFields()** which **deletes the present string in the .Entry** fields using **.delete(0, tk.END)** function.

   iii. A function **addStudent()** which **calls Search function** from databaseSS.py **for studnet.csv file with the current value in the rollEntry**. It then **calls the add_student** function of the **Student() Class by passing the entryList** from **getInput()** function and then **deleteEntryFields() if the checkList[0] is "Not Available"** and the **entryList is not empty**. **If ""** is **in entryList** or **if the checkList[0] is not "Not Available"** it **shows a warning.**

   iv. A function **modifyStudent()** which **calls Search function** from databaseSS.py by **for student.csv with the current value in the rollEntry** using .get() function. It then check the **status of rollEntry.get()** and **if it is empty it shows a warning**, **else if** the checkList's first element is "Not Available" it shows a warning message **else** it calls **modify_student()** method of the **Student() Class** and then it calls **deleteEntryFields().**

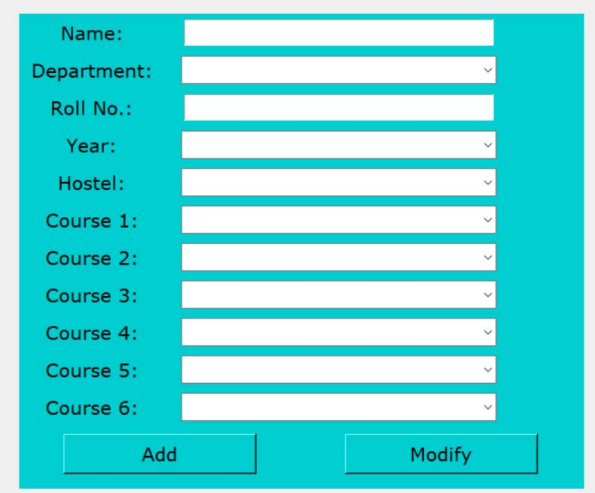
c. In the **viewFrame**, there **are pair of Labels one to contain the features** i.e. "Name: ", "Dept: ", etc. and the **other to show features' value on a button press**. A frame **SearchFrame** is on top of it which contains a **tk.Entry widget for roll number**, a **search button** having command **searchButtonCall()** and an **Add Marks** button having command **show_frame(EnterMarks)**. Another frame placed at the bottom of this frame containing 3 buttons for **"Delete", "View All" and "View Grade Card"** having commands **deleteButtonCall(), show_frame(ViewAllStudent) and show_frame(Report Card)** respectively. In the end, the class contains a **"Back To Home"** button with command **show_frame(HomePage)** to raise that page on the top.

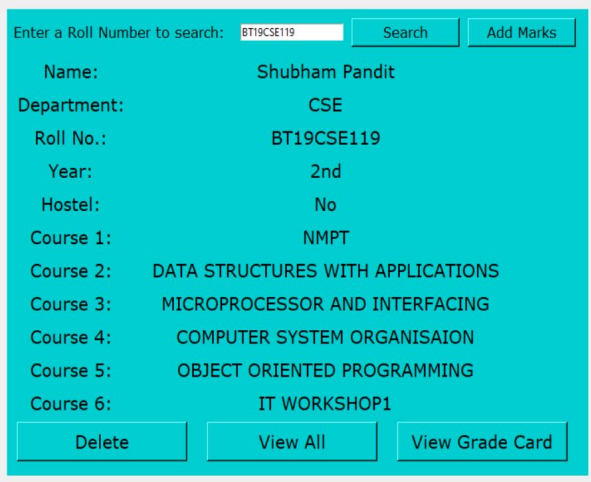
i. A function **searchDetails()** for calling **search function for student.csv with rollno** that is passed to it. It stores **the returned list in the outputList** attribute.

ii. A function **updateLabels()** which **has roll no as parameter** and then **updates the text of the second Label** using **.config**() function to the **outputList values**.

iii. A function **searchButtonCall()** to handle the pressing of the **"Search"** button. It checks the **value in rollNoEntry** and gives a **warning if the value is ""** and **if it is not** it calls **searchDetails() and updateLabels()** respectively.

iv. A function **DeleteButtonCall()** to handle the pressing of **"Delete"** which calls **deleteRecord() twice** for **student.csv and studentMarks.csv** and then **updates the labels** if the entry in **rollNoEntry** is **not empty** and if it is, it **shows a warning message box**.





5. Class EnterMarks (tk.Frame):

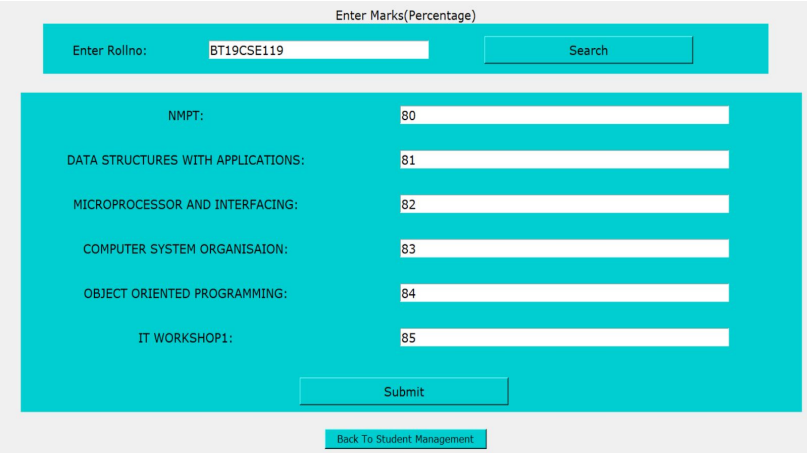
This **class inherits Frame()** class from tkinter. It **creates a sub-heading** below the heading and is **horizontally centred**. It **creates 6 more rows below the sub-heading which takes marks of each subject as input**.

The functions used in the above class are:

i. **pack** : It organizes widgets in blocks before placing them in the parent widget.
ii. **place** : It organizes widgets by placing them in a specific position in the parent widget.
iii. **grid** : It organizes widgets in a table-like structure in the parent widget.
iv. **entry** : It used to accept single-line text strings from a user.
v. **frame** : It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets.
vi. **label** : It implements a display box where you can place text or images.

Function defined :

i. **updateLabels(rollno)**: update courses name to the course in which the student is currently rolled in.
ii. **getInput():** gets the marks as input and stores it in a list and then returns that list.
iii. **deleteEntryFields():** it empties the entry field by deleting the elements in it.
iv. **addMarks():** searches for the student with roll number and then adds the marks of the student in the student database.



6. Class ViewAllStudent(tk.Frame):

This **class inherits Frame()** class from tkinter. It c**reates a label for subheading** packed at the top. Then there is a frame **ViewAllFrame** placed at the center of the window. **Two scrollbars are defined** using **tk.Scrollbar one vertical and one horizontal** placed at the **right side and the bottom** of the viewAllFrame respectively. Function **get_details** is called for **student.csv**. A **listbox is created** using **tk.Listbox** having **yscrollcommand set to scrollbar1 and xscrollcommand set to scrollbar2.** There is a **viewButtonFrame to place refreshButton** in it.

i. A function **refreshPage()** to handle the pressing of the **"Refresh"** button. It **deletes all the entries in the listbox**. Then it **calls the get_details()** function again and **insert all the entries** again to the listbox. In this manner the **listbox updates** every time the "Refresh" button is pressed.

7. Class StaffManagement(tk.Frame):

This **class inherits Frame()** class from tkinter. Features that are being handled by this class are: **name, college ID, department, email, last college, gender and address** of the staff.

An attribute outputList is defined which stores a list returned from the search function later. Every component is designed in the constructor. There are three main tkinter frames defined in this class named **insFrame, inputFrame** and **viewFrame** and are placed on the window using .place() function.

a. In the **insFrame**, **two labels are defined** to show "instructions" on the top of the window. .grid() function is used to arrange the labels on the window.

b. In the **inputFrame**, **pairs of Label and Entry are created** and placed in the frame for the **taking user Inputs**. It contains another frame named viewButtonFrame1 which contains 2 buttons for "Add" and "Modify".

i. A function named **getInput()** which **takes the value from .Entry() using .get()** and **stores them in temp variables** and **creates a list** Slist and **returns it**.

ii. A function named **deleteEntryFields()** which **deletes the present string in the .Entry** fields using **.delete(0, tk.END)** function.

iii. A function **addStaff()** which **calls Search function** from databaseSS.py **for staff.csv with the current value from collegeIdEntry** using .get(). It then **calls the add_staff** method of the **Staff() Class by passing an entrylist** from **getInput()** function and then calls **deleteEntryFields() if the checkList[0] is "Not Available"** and the **entryList is not empty**. **If there is "" in entryList or if the checkList's first element is not "Not Available"** it **shows a warning.**

iv. A function **modifyStaff()** which **calls Search function** from databaseSS.py **for staff.csv with current value from collegeIdEntry** using .get() function. It then check the **status of collegeIdEntry.get()** and **if it is empty else if** the checkList[0] is "Not Available" **it shows a warning else** it calls **modify_staff()** method of the **Staff() Class** and then it calls **deleteEntryFields().**

c. In the **viewFrame**, there **are pair of Labels one to contain the features** i.e. "Name: ", "Department: ", etc. and the **other to show the feature's value on a button press**. A frame **SearchFrame** is on top of it which contains a **.Entry widget for college ID**, a **search button** having command **searchButtonCall()** and a **Get ID** button having command **getIDCall()**. Another frame placed at the bottom of this frame containing 2 buttons for **"Delete" and "View All"** which have commands **deleteButtonCall() and show_frame(ViewAllStaff)** respectively. In the end, the frame contains a **"Back To Home"** button with command **show_frame(HomePage)** to raise that page on the top.

i. A function **searchDetails()** for calling **search function for staff.csv file with collegeId** that is passed to it.. It stores **the returned list in the outputList**.

ii. A function **updateLabels()** which **takes collegeId as parameter** and then **updates the text of the second Label** using **.config**() to the **outputList values**.

iii. A function **searchButtonCall()** to handle the pressing of the **"Search"** button. It checks the **value in collegeIDEntry** using .get() and gives a **warning if the field is empty** and **if it is not** it calls **searchDetails() and updateLabels()**.

iv. A function **DeleteButtonCall()** to handle the pressing of **"Delete"** which calls **deleteRecord()** for **staff.csv** and then **updates the labels**.

v. A function **getIDCall()** to handle the pressing of the **"Get ID"** button. It calls **searchDetails()** and **creates a dictionary detailsToId** to **call idGenerator** function from Supporting_Programs.idCardCreator.

8. Class ViewAllStaff(tk.Frame):

This **class inherits Frame()** class from tkinter. It c**reates subheading** packed at the top. Then there is a frame **ViewAllFrame** placed at the center of the window. **Two scrollbars are defined** using **tk.Scrollbar one vertical and one horizontal** placed at the **right side and the bottom** of the viewAllFrame respectively. Function **get_details** is called for **staff.csv**. A **listbox is created** using **tk.Listbox** having y**scrollcommand set to scrollbar1 and xscrollcommand set to scrollbar2.** There is a **viewButtonFrame for refreshButton** having command **refreshPage().**

i. A function **refreshPage()** to handle the pressing of the **"Refresh"** button. It **deletes all the entries in the listbox** and **calls the get_details()** function and **inserts all the entries** to the listbox. The **listbox updates** every time the "Refresh" button is pressed.

9. Class ReportCard(tk.Frame):

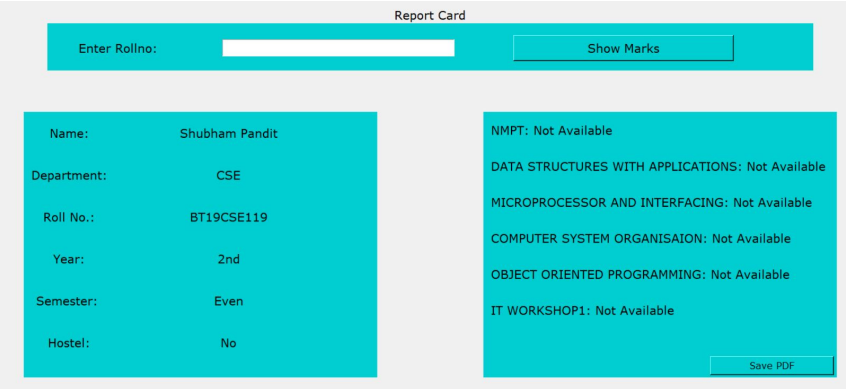
The frame that we have used to **design the report card** has been made purely using tkinter. We have used the **Grid Function** to place the elements. The **constant values of the report card ("Name", "Class") etc have been hardcoded. Each of the grid cells has a Label that is displaying the data within**. The label code looks like this:

dept = tk.Label(inputFrame, text = "Department: ", font = Large_Font, bg = "dark turquoise")

We created 2 different frames in our window. One to display the details of the student and the other to display the marks of the student.

i. A function **savePDF()** to handle the pressing of the button **"Save PDF".** It calls the search function for student.csv and studentMarks.csv and creates a dictionary of all the details. It then calls the create_pdf function of Supporting_Programs.pdfCreator.

ii. A **function updateLabels()** to handle the pressing of the button **"Show"**. It takes a parameter rollno and calls the search function for student.csv and studentMarks.csv and updates the text attribute of the labels using .config() function.



10. Creating the App.

app = MainApp() : This line creates an object of the MainApp() class.

app.mainloop() : This executes the app object in a loop until the window is not closed.

❖DATA FILE MANAGEMENT :

1. Function create_database(details, filename):

@Parameters : dictionary of details , file_name
@Output : writes details to file provided

This function accepts the file name and dictionary. It converts the provided dictionary to dataframe using pandas and writes it to the provided file with header. This function creates the csv file

2. Function write_database(details, filename):

@Parameters : dictionary of details , file_name
@Output : writes details to file provided

The above function takes details and filename as arguments and writes the details provided to it to the CSV file. For the details, a dictionary is used and string is used for containing the location and details of the file. With the help of dataframe we have created a table consisting of two columns for storing the data from the CSV file.

Here the inputs are the dictionary(details) which contains the details of class admin and name of the file is a string from which data is extracted.

3. Function read_database(filename):

@Parameters : file_name
@Return : dictionary of details

This function accepts a filename containing the database and then it reads the database through pandas and gets the header of the file that marks the label of database and the details corresponding to the first student.

❖DATABASE MANAGEMENT :

FRAMEWORK USED :: PANDAS

1. Function write_to_csv(dictionary,file) :

@Parameters : file_name , dictionary containing details.
@Output : writes details to file provided
@Usage : 'Add' button

This function converts the dictionary provided to the dataframe using **pandas.DataFrame** function and writes that dataframe to the file provided in append mode without headers.

2. Function search(file,attribute,value) :

@Parameters : file_name , attribute, value to be searched
@Return : returns list of all the values in dataframe corresponding to value to be searched if found else returns a list with values as 'NOT AVAILABLE'.
@Usage : 'Search' button

This function accepts the value to be searched (value) in along with the type of value (attribute) and the file name containing the database. It first converts the csv file to a dataframe with "attribute" as index value and then searches for the value in the index. If value is found then it converts all the other values in row to list , else if value is not found then it appends **'NOT AVAILABLE'** to list. This list is returned to the calling function.

3. Function deleteRecord(file,value):

@Parameters : file_name , value to be deleted
@Return : returns status of deletion operation
@Usage : 'Delete' button

This function accepts the roll number corresponding to which the record is to be deleted and the file name containing the database. It first calls the search function to check the presence of a record. If a record is not found then it passes the message that record is not present. Else it opens a new file with name "temp.csv" copies all the details of file_name except the record corresponding to the value/roll number whose record was to be deleted. Then the previous file is deleted and the new file "temp.csv" is renamed as the previous deleted file. Then it returns the status of this operation as : **" RECORD DELETED SUCCESSFULLY "**.

4. Function modify_details(file,value,dic):

@Parameters : file_name , value to be modified, dictionary of modified details
@Output : modifies the details
@Usage : 'Modify' button

This function accepts the roll number corresponding to which the record is to be modified, a dictionary of modified values and the file name containing the database. It first calls the search function to check the presence of a record. If a record is not found then it passes the message that record is not present. Else it opens a new file with name "temp.csv" copies all the details of file_name except the record corresponding to the value/roll number whose record was to be modified and in place of it details from the new dictionary are written. Then the previous file is deleted and the new file "temp.csv" is renamed as the previous deleted file.

5. Function get_details(file):

@Parameters : file_name having database
@Return : returns the number of records and dictionary of details hashed with index numbers
@Usage : 'View All' button

This function accepts the file name and reads the file line by line removing the newline character and splitting string to list. Later this list of lists is converted to a dictionary by hashing a list with an iterable index number.

❖P.D.F. (MARKSHEET) GENERATOR :

FRAMEWORK USED :: FPDF, MATPLOTLIB

Function create_pdf(dictionary):

@Parameters : dictionary of details
@Usage : 'Save PDF' button

This function is called when the "Save PDF" button is pressed in the ReportCard Frame. It basically accepts the dictionary of details corresponding to one student and then it instaniates PDF class that creates, writes and saves Mark Sheet as pdf with filename as Roll Number of student.

CLASS PDF(FPDF) :

This class contains the member functions that are used to write to a pdf file by creating cells. pdf.alias_nb_pages() is called to make count of the number of pages in pdf that is further utilised by footer function. At the end it saves the file with the name of the student provided in the dictionary.

Method header(self):

This is the default function defined in FPDF to write a header to every page that is added to all the pages of the pdf. This function is automatically called on adding a new page to the pdf. This basically set_fonts and writes **"INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, NAGPUR"** on every page.

Method sub_head(self):

This function is user defined and is used to write subheading to the file. This writes "PROVISIONAL GRADE SHEET" to the file.

Method side_head(self,title):

@Parameters : title to be written

This function is a generalised function which accepts a string and writes it to file as a center heading.

Method personal_details(self,dictionary):

@Parameters : dictionary containing details

This function is used to write the multiple cells having personal details of students to the file.

Method blank(self):

This function is used to write the blank line to the file.

Method academic_details(self,dictionary):

@Parameters : dictionary containing student details

This function is used to write the multiple cells having academic details of students to the file.

Method remark(self):

This function is used to write the remark container to the pdf file.

Method graph(self,dictionary):

@Parameters : dictionary containing student details

This function creates the lists of courses and student marks from a dictionary and calculates mean,grand total and average along with the bar chart of marks and subjects, to show the relative student performance, using matplotlib and saves the image in the default directory. Then this image is written to the pdf and that image is removed using the “os” module.

Method stamp(self):

This function is used to create STAMP lines and write creation time to file using datetime module.

Method footer(self):

This function is used to create footers on every page. This function is itself called on adding newpage. In general this is used to write page numbers to the file.

❖I.D. CARD GENERATOR

FRAMEWORK USED :: PIL (PYTHON IMAGE LIBRARY)

(Python Imaging Library is an open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.)

Function idGenerator(dictionary):

@Parameters : dictionary of details of staff
@Output : id card is generated and saved in "StaffID" Folder
@Usage : 'Get ID' button

This function accepts a dictionary of details of staff members when the "Get ID" button is clicked. It basically creates an ID card for the staff member for which it is called and saves it to the default location ie., in "StaffID" Folder. It typically reads the details from the dictionary and writes it to the blank white picture through the **"draw.text()"** function which takes a tuple as input that determines the position where text is to be drawn or written. At last it writes a default image provided in the asset folder to the image through **".paste()"** and saves it with the name of the staff member.

❖Linker Classes :

1. Class Admin():

A class to represent admin. This class contains attributes and methods to store, read and write the properties related to settings in the project. The attributes are password, year, name and semester. The methods are change_year, change_admin, change_semester, compress_to_dictionary.

i. The constructor of the Admin class calls the read_database function by passing 'Databases\\admin.csv' and stores the returned dictionary to details. It then initialises the attributes of class Admin to the values from the dictionary details.

ii. The change_year, change_admin method accepts newYear, newName and assigns it to the year, name respectively. Then it calls the method compress_to_dictionary and stores the returned dictionary into details. Finally, it calls write_database.

iii. The change_semester method checks the value in semester which can either be 'Even' or 'Odd' and changes the value of the semester to the other value

iv. The compress_to_dictionary function creates a dictionary with all the attributes and returns it.

2. Class Student():

The student class is one of the supporting functions. It starts with declaring all the parameters used in the Student management menu like Name, Roll, Year etc.

i. Function add_student (), it is used to add a new student in the database along with all the details. All the objects in the student class are assigned the values.

self.name = inputList [0]

It then accepts all the entered values and calls compress_to_dictionary.

details = self.compress_to_dictionary()

ii. Function modify_student() is used to modify pre-existing Student details. Here, first the details of the student whose roll number is entered is searched in the student databases by calling search function.

After the details have been searched listCheck is called.

self.name = self.listCheck(prevInput[0], inputList[0])

The modify_student() function is basically used in conjunction with modify_student(). The prevInput and inputList are used to store the existing variables of the student.

iii. The listCheck() function is used to compare the parameter passed to it.

3. Class Staff()

A class to represent staff. This class contains attributes and methods to store, read, write and modify the staff.csv file by the help of databaseSS.py. The attributes are name, dept, collegeId, email, lastCollege, gender and address. The methods are add_staff, listCheck, modify_staff and compress_to_dictionary.

i. Method add_staff():

@parameters: inputList, a list containing the values for all the attributes.

This function assigns values from the inputList to the attributes of the class. It then calls compress_to_dictionary and stores the returned dictionary in detail. Finally, it calls write_to_file from databasesSS.py to write details to the file staff.csv.

ii. Method list_check():

@parameters : a, b, to store two strings
@return : a, if b is equal to “” else b

Use if else structure and return a if b is equal to “” else b.

iii. Method modify_staff():

@parameters : inputList, a list containing the values for all the attributes.

This function calls the search function of databasesSS.py and stores the value for the attributes and then updates them by passing corresponding elements of

prevList and inputList to self.listCheck. Then compress_to_dictionary is called and the returned value is stored in detail. Finally, it calls write_to_file from databasesSS.py to write details to the file staff.csv.

iv. **Method compress_to_dictionary():**

@return : details, a dictionary containing all the attributes with keys.

This function creates a dictionary of all the attributes of the class and returns it.

Problems & Challenges Faced

1. **Challenge:** Creating a function which will update labels and call other various functions was challenging as adding all those to the same button was not possible.

   Solution: We created various helper functions inside the __init__ method of the class and then called all the functions in a single function and then added that function as the command of the button. This made the process very easy and reduced the complexity of the code.

2. **Challenge:** In ViewAll class in app.py, updating of listbox was not happening if we added, modified or deleted some details.

   Solution: We added a refresh button which on pressing will call a function which will delete the entries in the listbox and then re-read the files for inserting values in the listbox and then will recreate the listbox with those details in it.

3. **Challenge:** Integrating Matplotlib with tkinter was a big challenge as tkinter works on frames and when matplotlib is used it also raises its frame to show the graph. So these multiple main event loops are running simultaneously and they interfere with each other. This forces tkinter frames to stick.

   Solution: This problem can be resolved by forcing matplotlib to work as a non-interactive backend, by including **matplotlib.use('Agg')** in the main code**.** This makes sure that matplotlib works as a backend.

REFERENCE

1. https://docs.python.org/3/library/tkinter.html
2. https://matplotlib.org/
3. https://pandas.pydata.org/docs/user_guide/index.html#user-guide
4. https://docs.python.org/3/library/tkinter.messagebox.html