CHAPTER 1

INTRODUCTION

COLLEGE BUS FEES MANAGEMENT is a File Structure application which is helpful for admin to manage student and bus details.

This project has been developed to override the problems prevailing in the practicing manual system. This project is supported to eliminate the hardships faced by the existing system. In this Admin can add, delete, search, update and display the student and bus details. Moreover, system is designed for the particular need for the institute to carry out in a smooth and effective manner.

In the following sections, a brief introduction about the scope of project, problems with the existing system and how it can be addressed using proposed system are discussed.

1.1 SCOPE OF THE PROJECT

The proposed college bus fees management system will cover many aspects of time keeping and details process. This includes the capture of information based on the student paying their bus fees and storing their information in files for longer use. In addition to that this project will allow you to display the details of student, bus and their fees.

This project is a desktop system that enables the efficient storage of student and bus records to probably manage the fees billing. The system provides the information about the fees payed by the students.

1.2 Existing System

In the existing system colleges must manually maintain information regarding College Busses and routes. Information related to student passengers and bus passes must be maintained separately.

1.3 Objectives

This Project is mainly based on Interaction with the files .The Main Object of this project is to computerize the manual system and reduce Time consumption. In already existing systems they are using some manual system like registering in ledgers, notebook etc. But this product helps them to store details in computerized form.

In other words, we can say that our project has following objectives.

- Make all the system computerized.
- Reduce time consumption.
- Reduce error scope.
- All System Management are automated.
- Easy operation for operator of the system.
- No paperwork requirement.
- Provide a simpler method to store and access information related to buses and students.

CHAPTER 2
SYSTEM ANALYSIS

In this chapter, complete descriptions regarding the project development are discussed. The requirements of the project identified are showcased. The file design is done Using High-Level Conceptual Data Models.

2.1 Requirement Analysis

Following requirements were identified during the requirement collection and analysis:

1. This is implemented to reduce the paper work.
2. Admin can login to the system.
3. Admin can add, delete, update, search and display the bus and student.
4. Pay bus fees.

2.2 Functional Requirements

Functional requirements of a software project that interpret the function of a part. It defines its functions, input and output. The typical functional requirements include:

- Admin can login into the system.
- Admin can add bus and student details.
- Admin can update the bus and student details.
- Admin can search for the bus and student details.
- Admin can display or view the bus and student details.
- Admin can delete bus and student details.

2.3 Non- Functional Requirements

Non-functional requirements it specifies the canon of the articular process not the judgment of the system and behavior of the process. Non-functional requirements define how the system work.

- This application is developed to make update, insert and delete of any data more efficiently and effectively or in a time saving manner.
- This application work efficiently it works on all logical paths and independently and it should use the user and admin data efficiently.
- This application is available during all the time for registration.
- This application reduces all the complexity during entry of data.
- To run this application efficiently network is the main important factor.
- Using of application is secure, because it displays appropriate information about the data entered.
- The system should be reliable, and it should be related in all the condition and it should be recoverable in all the situation or condition if error occurs.

2.4 Use Case Diagram

The use case diagrams usually refer to behavioral diagrams helps people to understand the interaction between user and system. Use case diagram identify different users of the system. It is used to define some set of actions, which is called as use cases. Actors are the result of some valuable use cases. Use case figures are also called as unified modeling language.

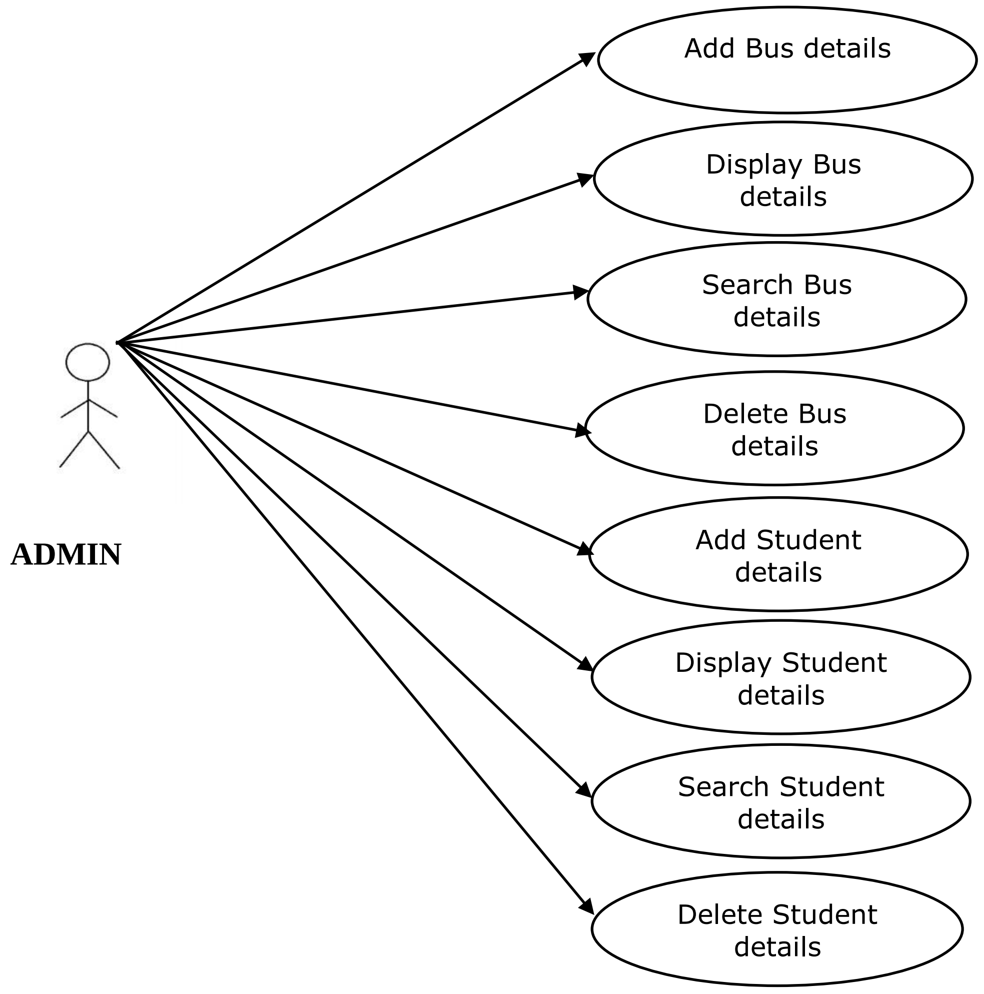


Fig 2.1 Admin Use Case Diagram

In the figure 2.1 Admin can have the functions like add, display, search, delete and modify bus and student details.

2.5 SYSTEM REQUIREMENTS

Technology Implemented	Tkinter
Language Used	Python
Database	Text Files
User Interface Design	Tkinter

Table 2.1 System Requirements

2.6 HARDWARE REQUIREMENTS

Processor	Intel Core Duo 2.0 GHz or more
RAM	1GB or more
Hardware Devices	Keyboard with mouse
Hard Disk	10GB or more
Display	Standard Output Display

Table 2.2 Hardware Requirements

Chapter 3

System Implementation

3.1 Add Bus details into a file

Following code snippet is used to add bus details like bus number, driver name.

```
num_info = bus_number.get()
name_info = driver_name.get()
file_exists = os.path.isfile('test.csv')
with open('test.csv', 'a', newline='') as f:
    header = ['Bus_num', 'Driver_name']
    writer = csv.DictWriter(f, fieldnames=header)
    if not file_exists:
        writer.writeheader()
    writer.writerow({'Bus_num': num_info, 'Driver_name': name_info})
    messagebox.showinfo('Success', "Details added successfully")
```

3.2 Display Bus details from a file

Following code snippet is used to display all the bus details.

```
global display_bus
display_bus = Toplevel(main_screen)
display_bus.title("Add Bus Details")
display_bus.state('zoomed')
display_bus.bind('<Escape>', lambda e: display_bus.destroy())
Label(display_bus, text='Bus Number')
with open("test.csv", newline="") as file:
    reader = csv.reader(file)
    r = 0
    for col in reader:
        c = 0
        for row in col:
            # i've added some styling
            Label(display_bus, width=10, height=2, text=row, relief=RIDGE).grid(row=r, column=c)
            c += 1
        r += 1
```

3.3 Search bus details from a file

Following code snippet is used to search a specific bus from list of bus.

```
bus_text = bus_search.get()
with open('test.csv', 'r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        if bus_text == row[0]:
            messagebox.showinfo('Found', row[:])
            break
```

3.4 Delete bus details from a file

Following code snippet is used to delete a specific bus from list of bus.

```
delete_info = bus_delete.get()
with open('test.csv', 'r', newline='') as infile, open('test1.csv', 'w', newline='') as outfile:
    writer = csv.writer(outfile)
    for row in csv.reader(infile):
        if not any(remove_word in row for remove_word in rem):
            writer.writerow(row)
    infile.close()
    outfile.close()
    os.replace('test1.csv', 'test.csv')
    messagebox.showinfo('Deleted', 'Successfully deleted')
```

3.5 Update bus details on a file

Following code snippet is used to update specific bus details from list of bus.

```
num = bus_update.get()
name = driver_update.get()
with open('test.csv', 'r', newline='') as file, open('temp.csv', 'a', newline='') as tempfile:
    reader = csv.DictReader(file)
    header = ['Bus_num', 'Driver_name']
    writer = csv.DictWriter(tempfile, fieldnames=header)
    writer.writeheader()
    for row in reader:
        if int(row['Bus_num']) == int(num):
            row['Driver_name'] = name
        writer.writerow(row)s
    tempfile.close()
    os.replace('temp.csv', 'test.csv')
    messagebox.showinfo('Update', 'Updated successfully')
```

Chapter 4
Results and Discussion

In this chapter the results of the project are described. The snapshot of the project showing various functionalities like insert, delete, update and retrieval are showcased.

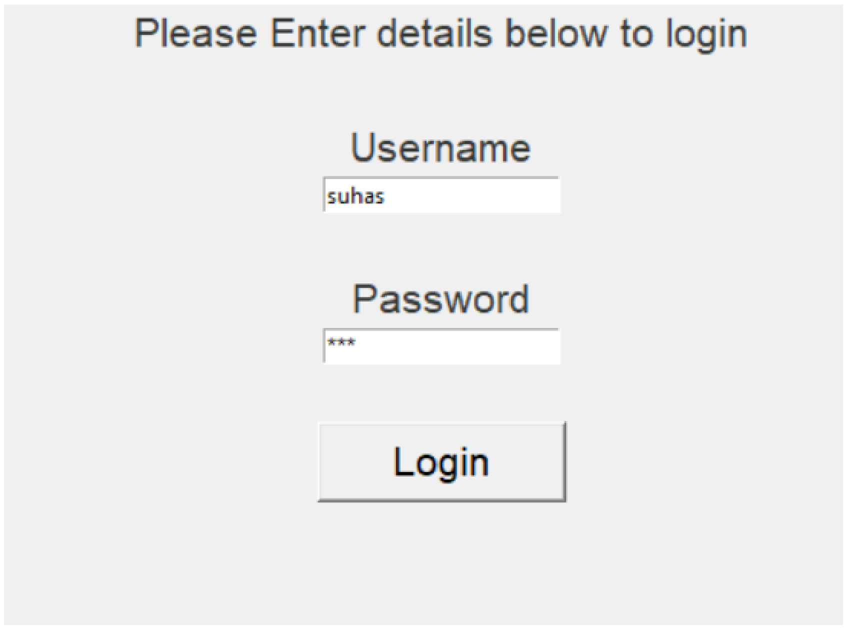


Fig. 4.1 Login page of project

Figure 4.1 shows the login page for the application.

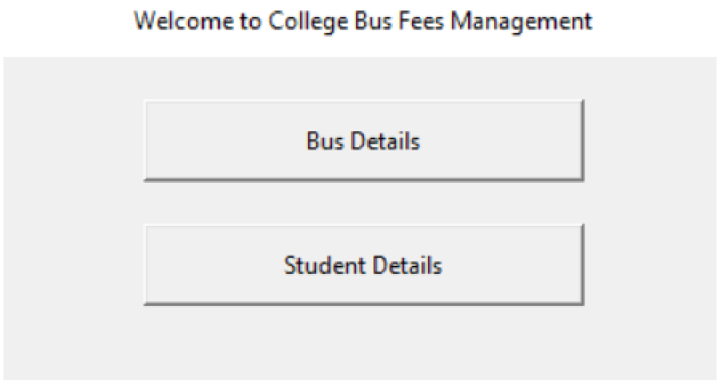


Fig. 4.2 Home Page

Figure 4.2 shows the home page of the application.

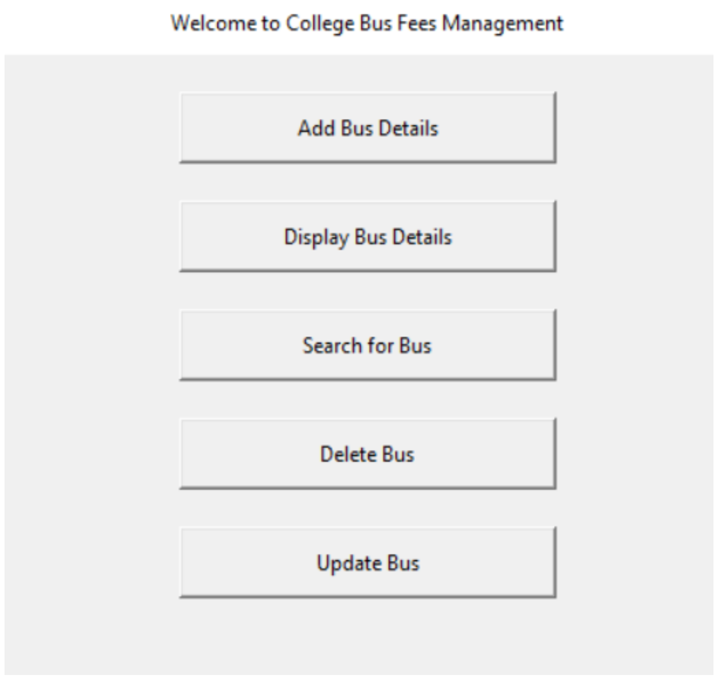


Fig. 4.3 Bus Details Home

Figure 4.3 shows the different functions available for bus.

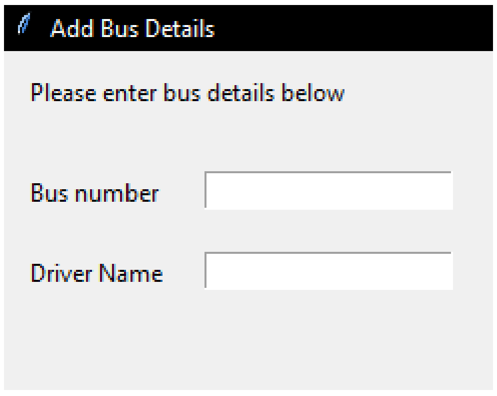
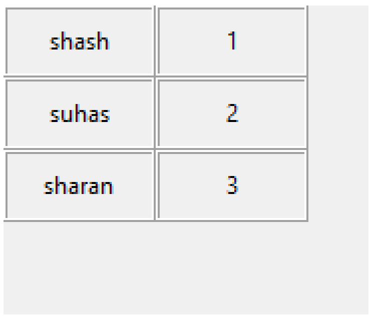


Fig 4.4 Add Bus Details

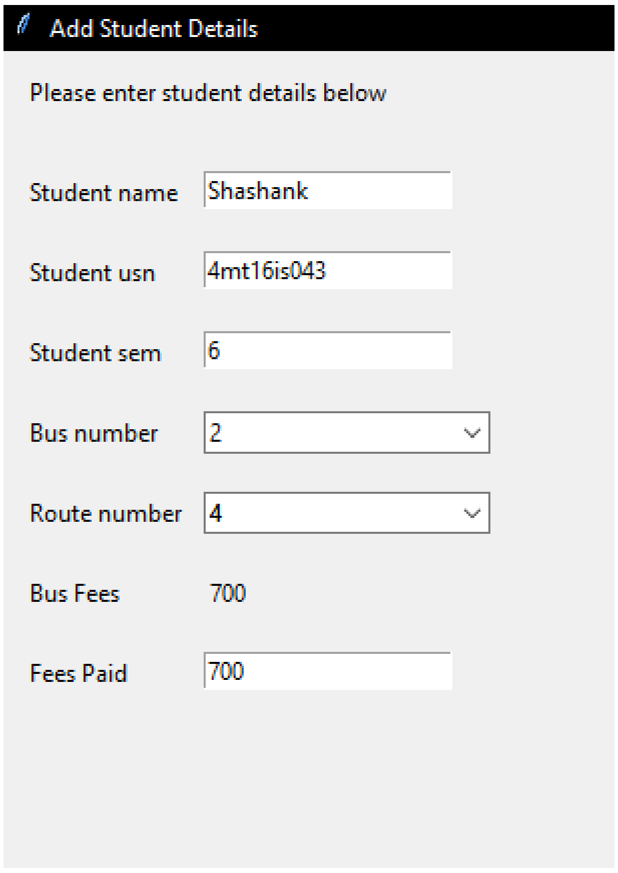
Figure 4.4 shows insertion of bus details.



shash	1
suhas	2
sharan	3

Fig 4.5 Display Bus Details

Figure 4.5 displays the bus details from file.



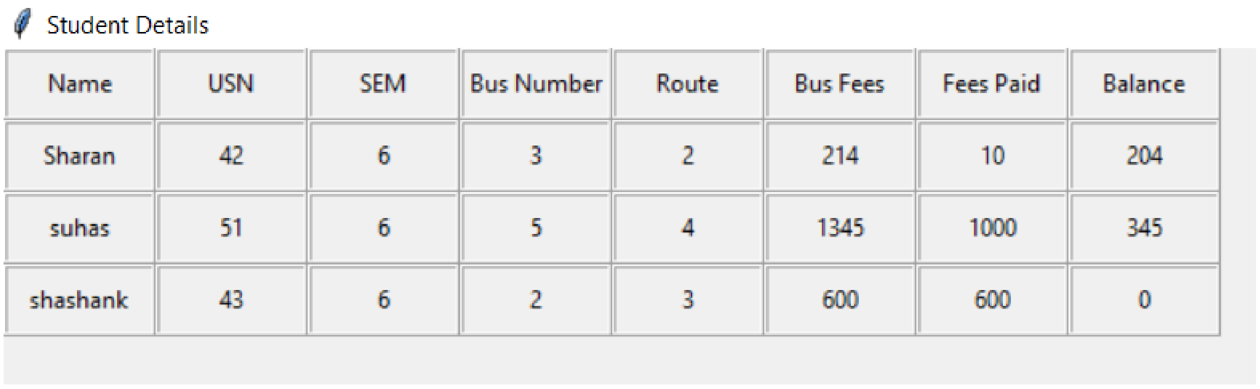
Add Student Details

Please enter student details below

Student name	Shashank
Student usn	4mt16is043
Student sem	6
Bus number	2
Route number	4
Bus Fees	700
Fees Paid	700

Fig. 4.6 Add Student Details

Figure 4.6 shows insertion of student details.



Student Details

Name	USN	SEM	Bus Number	Route	Bus Fees	Fees Paid	Balance
Sharan	42	6	3	2	214	10	204
suhas	51	6	5	4	1345	1000	345
shashank	43	6	2	3	600	600	0

Fig. 4.7 View Student Details

Figure 4.7 displays the student details from file.

CHAPTER 5

CONCLUSION

College Bus Fees Management System helps us in centralizing the data used for managing the tasks performed in a College Bus Fees management. The theoretical process involved in File based design has been practically implemented. The project provides user friendly interface for the users to interact with the database. All File operations including insertion, deletion, updating and Retrievals are supported.

REFERENCES

[1] Michael J. Folk, Bill Zoellick, Greg Riccardi; File Structures-An Object-Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.

[2]https://www.tutorialspoint.com/python/python_files_io.htm

[3]https://www.tutorialspoint.com/python/python_gui_programming.htm

[4]https://www.101computing.net/python-reading-a-text-file/

[5]https://www.devdungeon.com/content/gui-programming-python