Tuition Centre Manager

Project Documentation

by Danny Ho

EXECUTIVE SUMMARY

In today's competitive academic environment, the increasing emphasis on academic success and the complexity of school curricula have resulted in a surge of student enrollment at tuition centres. Astonishingly, many of these institutions still rely on outdated manual methods for managing administrative tasks, leading to inefficiencies, errors, and data security risks. Tuition Centre Manager addresses these challenges by providing a comprehensive, computerized solution designed to optimize operations and enhance educational outcomes. Tuition Centre Manager offers four key modules: a Student Database Module, Class Management Module, Student Performance Tracker Module, and Finance Management Module. Each module incorporates CRUD functionalities, with admins authorized to update data in all the modules except the Student Performance Tracker Module. The teaching staff are, instead, responsible for performing add/update/delete operations in this particular module. This project takes pride in its user-friendly design, which allows for simple navigation and quick access to critical data. By transitioning from a paper-based system to a centralized database, the Tuition Centre Manager conserves time, reduces human error, and minimizes data redundancy.

CHAPTER 1: INTRODUCTION

1.1 Introduction

Amidst today’s competitive academic environment, the growing emphasis on academic success, coupled with the increasing complexity of school curricula, has led to a surge in student enrollment at tuition centres. As a result, managing student databases, along with other administrative tasks, has become increasingly time-consuming and inefficient with the old-school pen-and-paper method. Fortunately, advancements in information technology have made it possible to develop systems that can significantly reduce these administrative burdens, streamlining operations in tuition centres.

The 'Tuition Centre Manager' is designed to help both administrative and teaching staff efficiently manage all aspects of a tuition centre. With dedicated, well-organized modules for student databases, class scheduling, performance tracking, and financial management, the system streamlines the process of updating and retrieving essential data for in-class operations, while also generating detailed academic and financial reports to ensure both high-quality education and the institute’s sustainability. Its centralized online database eliminates redundancies often found in physical records, while robust security features — such as a login system — ensure data authenticity, confidentiality, and integrity. This comprehensive solution optimizes administrative workflows, allowing staff to focus more on delivering quality education.

Tuition centres are a prime example of the rapidly growing education industry. Surprisingly, many such institutions still rely on traditional methods for database management. However, Tuition Centre Manager shall ultimately prove to offer higher reliability and efficiency as a computerised replacement.

1.2 Problem Statement

Nowadays, the growing enrollment in tuition centres has made managing student data, scheduling, student performance and finance tracking increasingly complex. Many centres still rely on manual, paper-based methods, which are prone to errors, redundancies and inefficiencies, besides posing risks to data security and integrity, with physical records being susceptible to loss, damage, or unauthorized access and modification.

The employment of 'Tuition Centre Manager' helps mitigate above-stated human errors and enables systematic tuition centre management by implementing semi-automated modules for student database, class management, as well as the tracking, analysis and reporting of student performance as well as the institution's finance. The centralized system is expected to ensure data accuracy and security, besides allowing staff to focus on delivering quality education.

1.3 Objective(s) of The Project

This project embarks on the following objectives:

a) to implement 4 modular functionalities with CRUD database management

- student database module
- class management module
- student performance tracker module
- finance management module

b) to perform complex calculation on student performance trend as well as institution income and expenses

c) to develop a report generator for student performance analysis and finance analysis

1.4 Scope

1.4.1 Modules to be developed

Student Database Module

Class Management Module

Performance Tracker Module

Finanace Management Module

1.4.2 Target Users

Admins are authorized to:

Student Database Module

add/update/delete/view/search student details (e.g. name, subject taken, parent name)

Class Management Module

view/add/update/delete class details (e.g. scheduling, teacher assignation)

Performance Tracker Module

view student performance

perform searching operation to generate academic report of particular students

Finance Management Module

add/update/delete/view/search financial income and expenses

Staffs are authorized to:

Class Management Module

view class details

Performance Tracker Module

add/update/delete student performance-related parameters (test scores etc.)

perform searching operation

CHAPTER 2: Analysis of Problem

2.1 Structured Chart

Figure 2.1 illustrates a structured chart depicting the operation flow for different user types (admin and user), highlighting their accessible modules and corresponding operations.

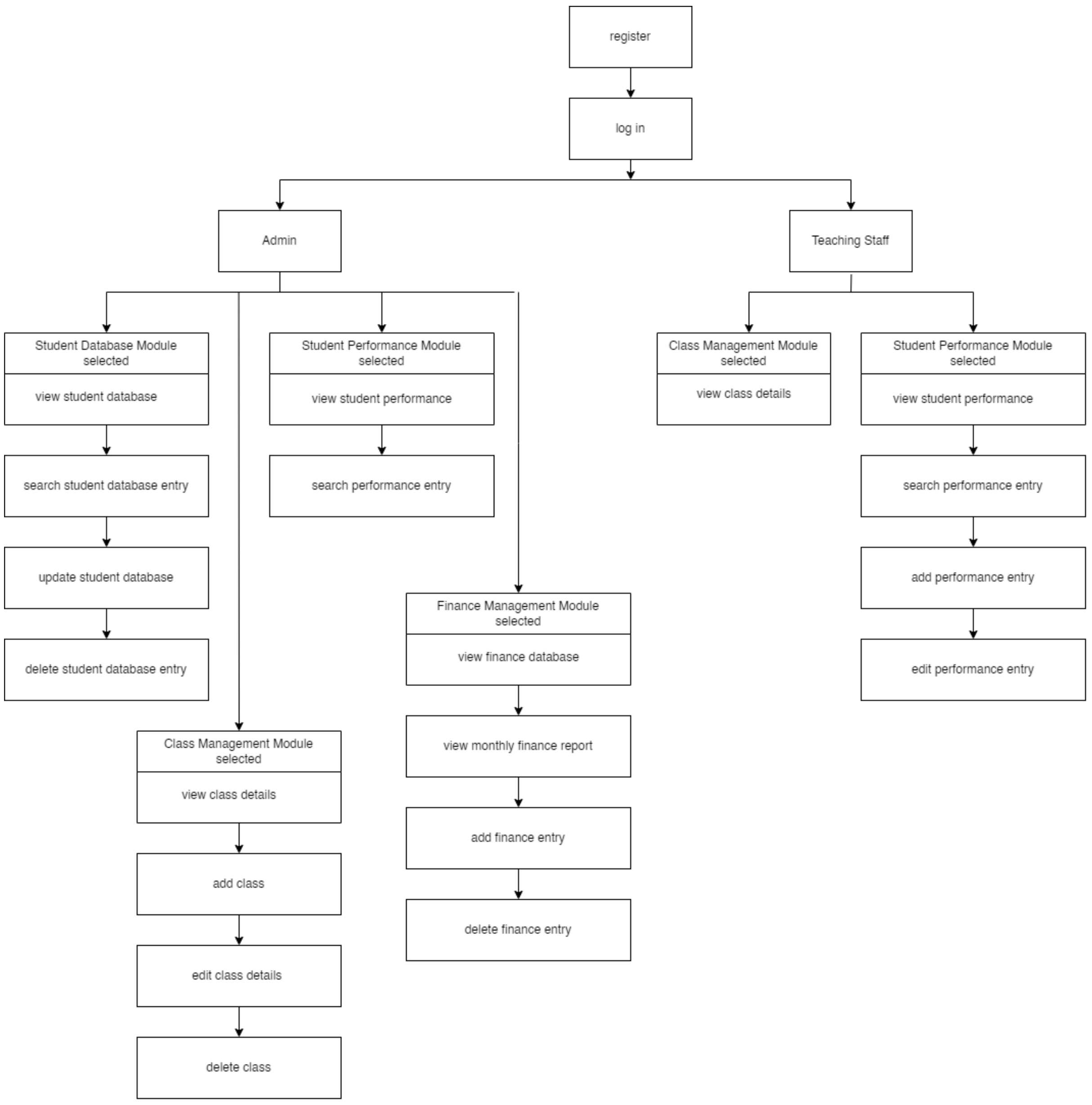
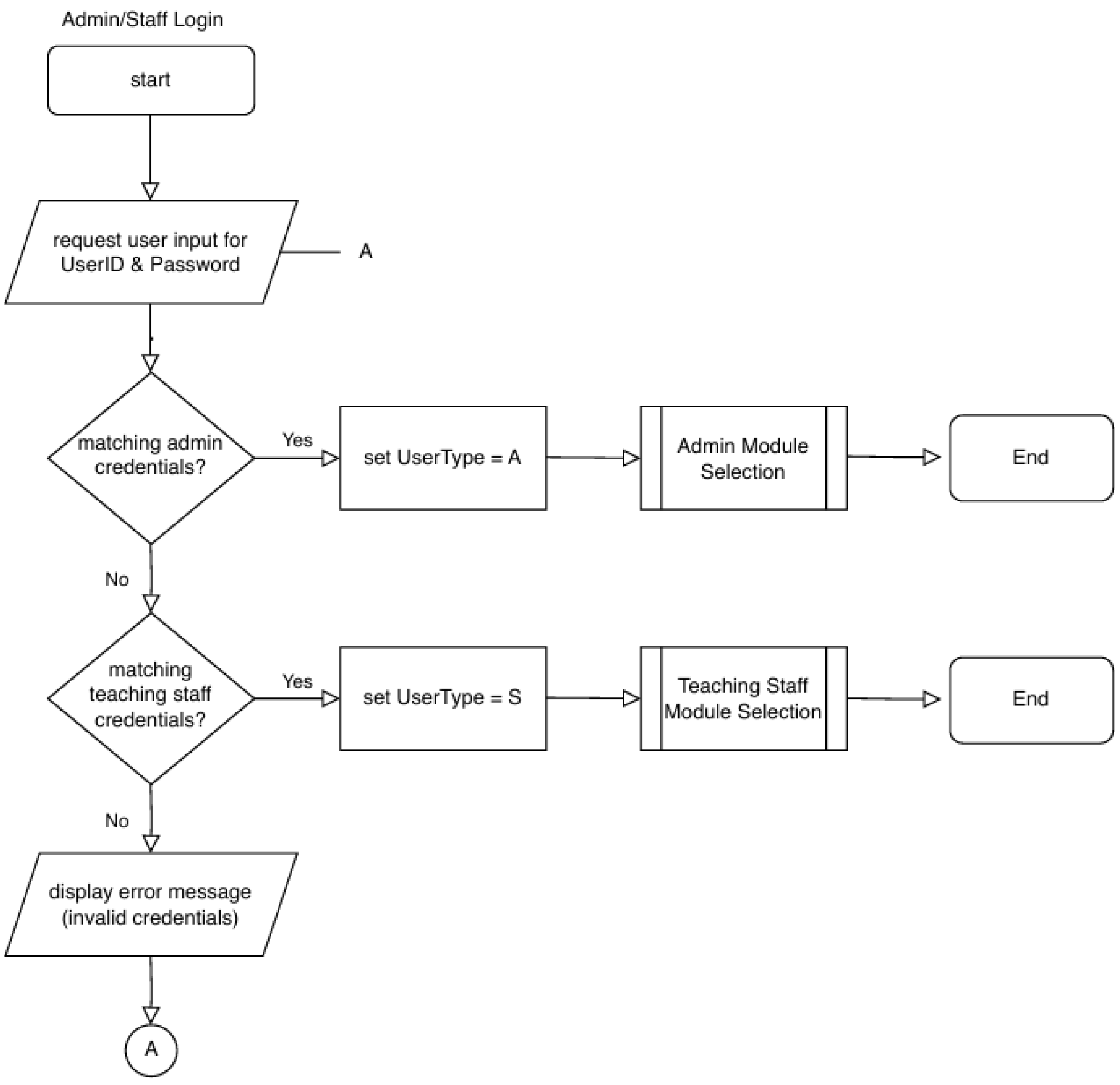


Figure 2.1 Structured Chart

CHAPTER 3: DESIGN

3.1 Flowchart

Figure 3.1 shows the process for user login. After inserting correct credentials, the users are brought to main menus tailored to their user type.



Admin/Staff Login
start
request user input for UserID & Password
A
matching admin credentials?
Yes
set UserType = A
Admin Module Selection
End
No
matching teaching staff credentials?
Yes
set UserType = S
Teaching Staff Module Selection
End
No
display error message (invalid credentials)
A

Figure 3.1 Login

Figure 3.2 shows the process for admin operation selection in main menu. Admins may choose to enter one of the four modules or log out from this page.

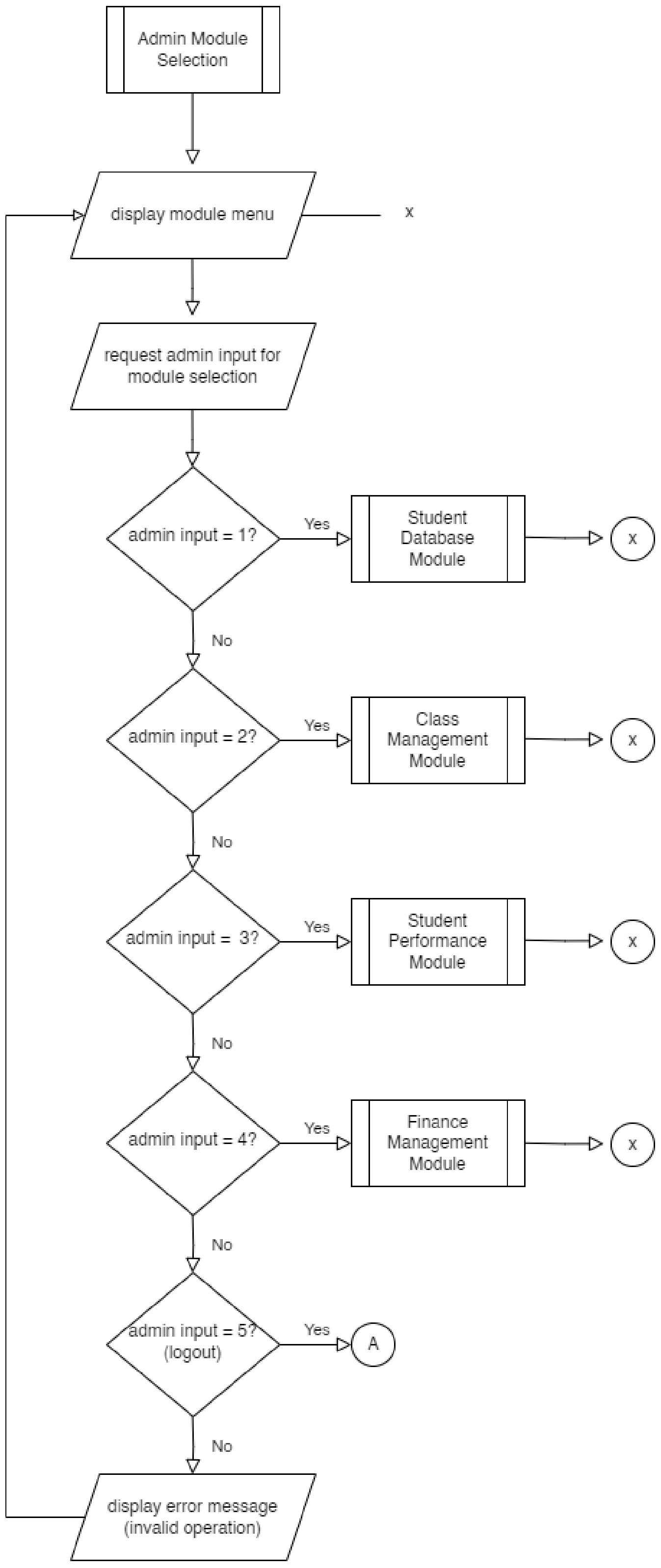
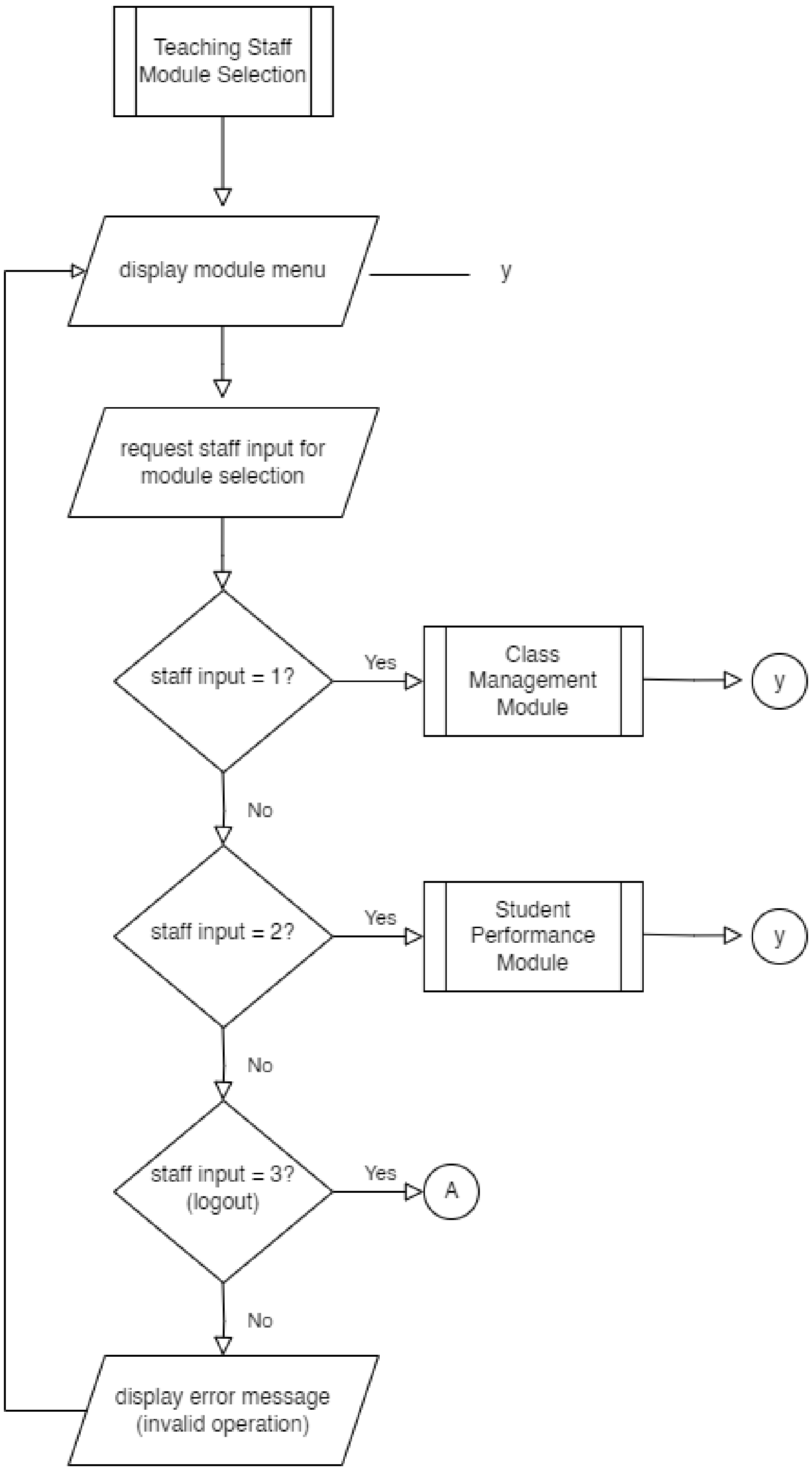


Figure 3.2 Admin Main Menu (Module Selection)

Figure 3.3 shows the process for staff operation selection in main menu. Staff may choose to enter one of the two modules or log out from this page.



Teaching Staff Module Selection
display module menu
y
request staff input for module selection
staff input = 1?
Yes
Class Management Module
y
No
staff input = 2?
Yes
Student Performance Module
y
No
staff input = 3? (logout)
Yes
A
No
display error message (invalid operation)

Figure 3.3 Staff Main Menu (Module Selection)

Figures 3.4 – 3.7 show the admin operation flow in student database module. Admins may choose to view the student database, add a new student entry, or update existing student entries. In Student Database View (Figure 3.5), Admins may also choose to filter details of a particular student by inserting his or her StudentID.

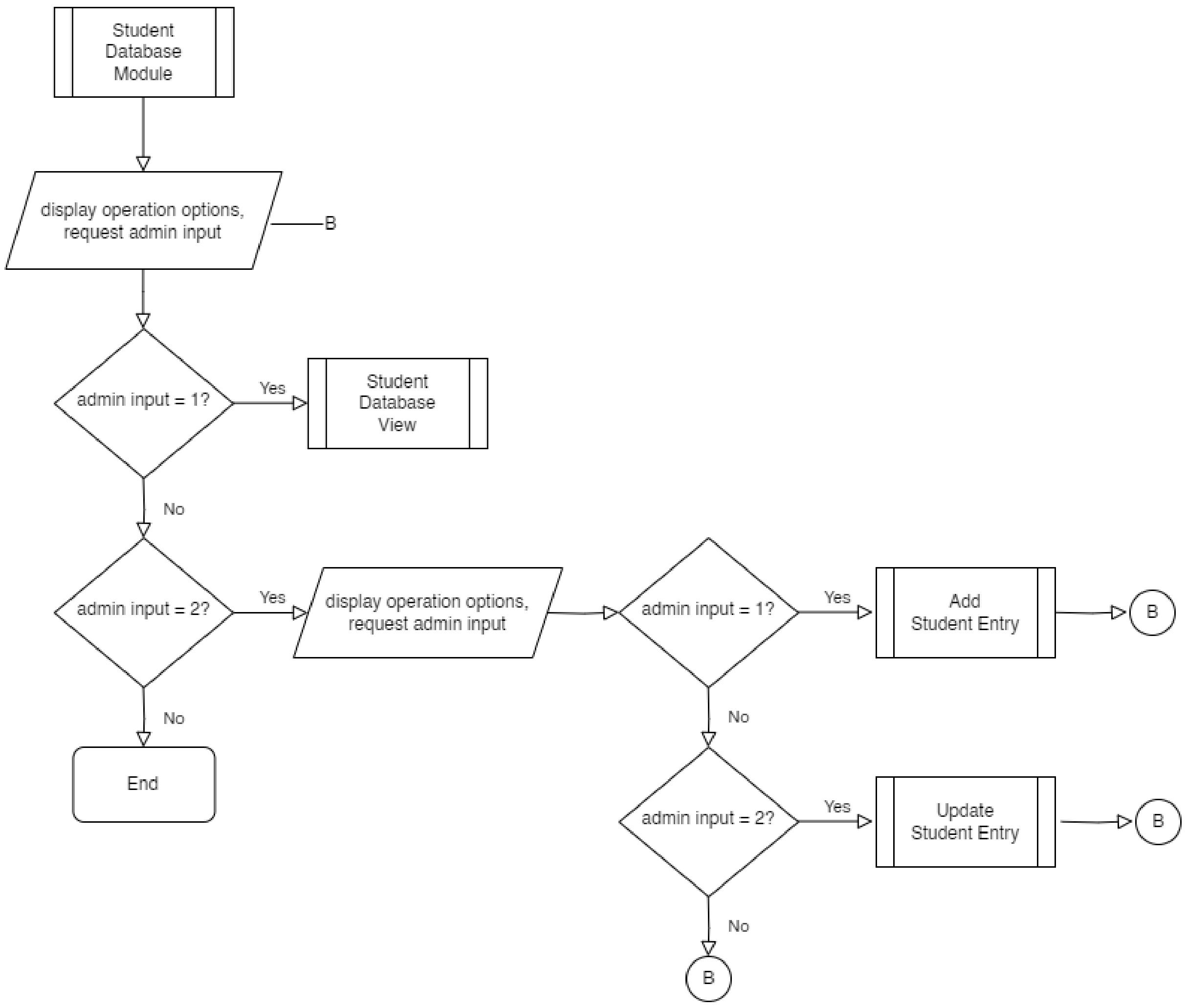


Figure 3.4 Student Database Module

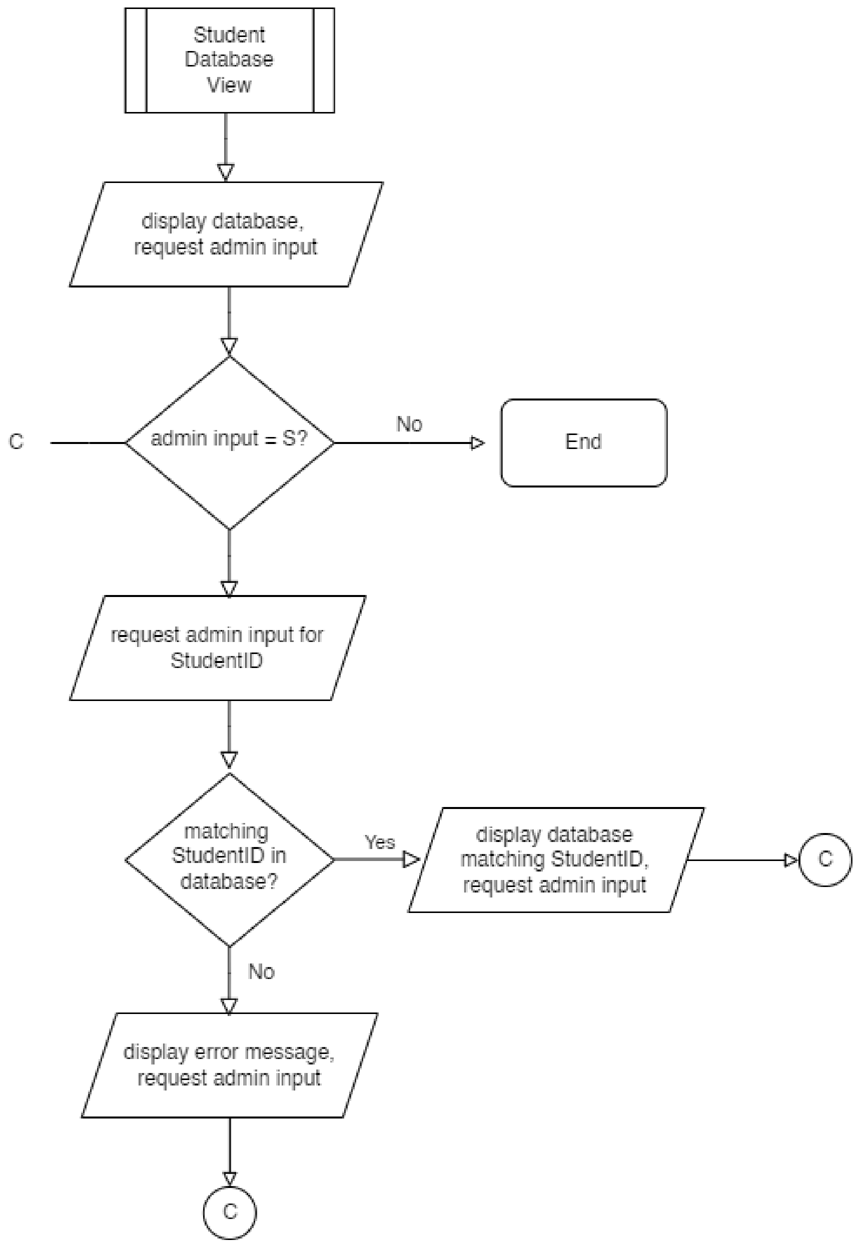

Student Database View
display database, request admin input
C
admin input = S?
No
End
request admin input for StudentID
matching StudentID in database?
Yes
display database matching StudentID, request admin input
C
No
display error message, request admin input
C

Figure 3.5 Student Database View

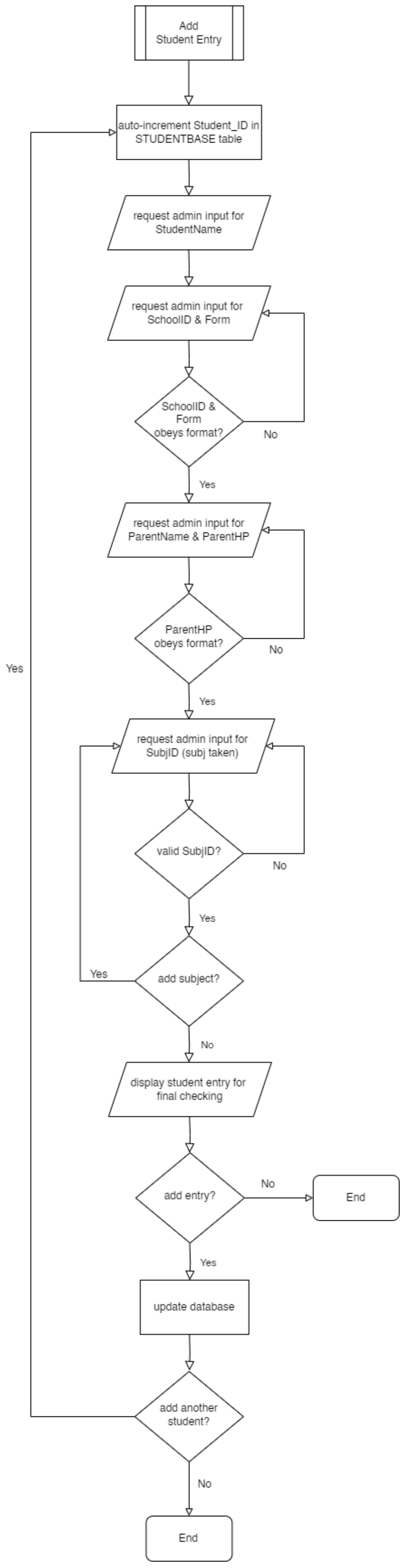

Add Student Entry
auto-increment Student_ID in STUDENTBASE table
request admin input for StudentName
request admin input for SchoolID & Form
SchoolID & Form obeys format?
No
Yes
request admin input for ParentName & ParentHP
ParentHP obeys format?
No
Yes
request admin input for SubjID (subj taken)
valid SubjID?
No
Yes
add subject?
Yes
No
display student entry for final checking
add entry?
No
End
Yes
update database
add another student?
Yes
No
End

Figure 3.6 Add Student Entry

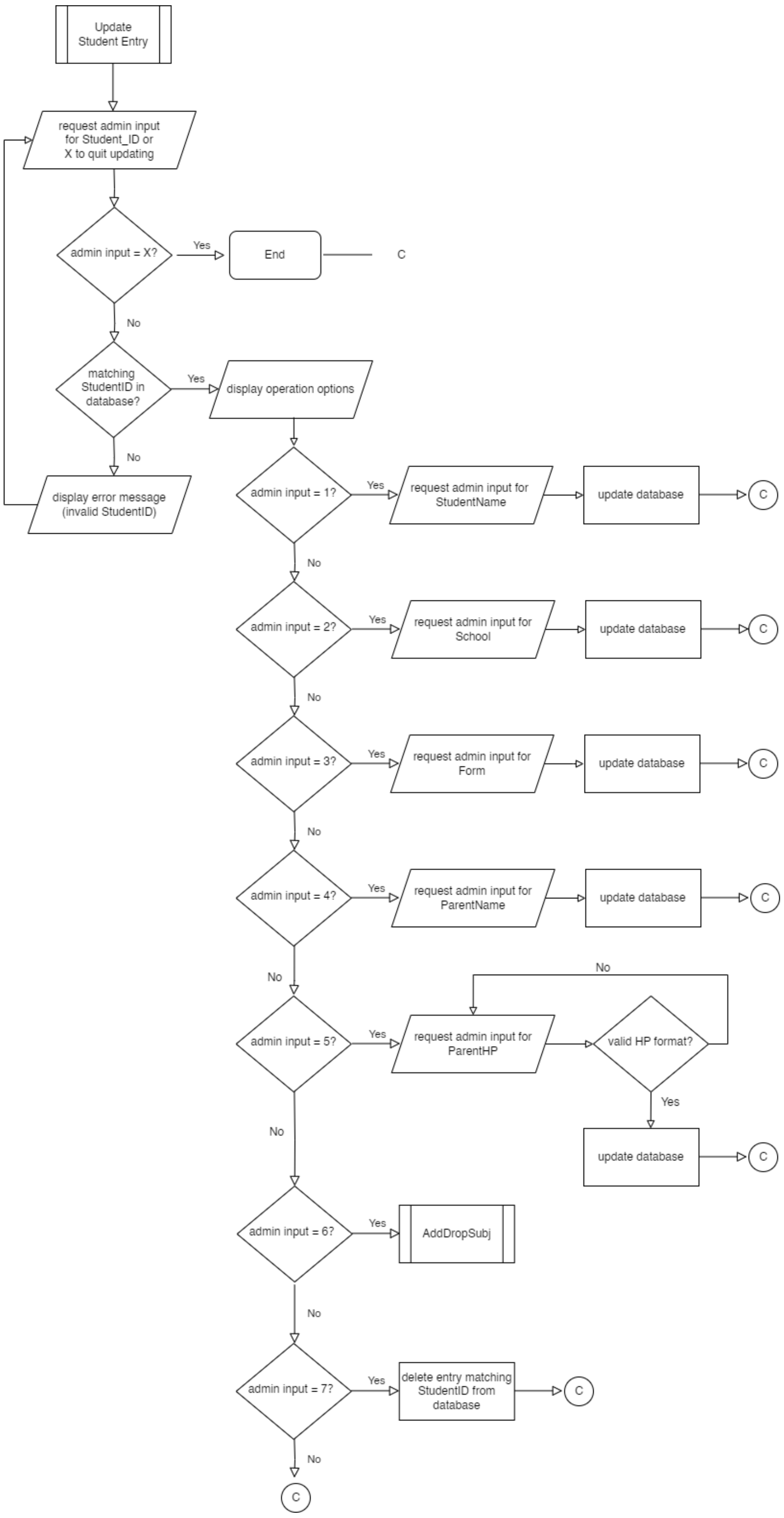

Update Student Entry
request admin input for Student_ID or X to quit updating
admin input = X?
Yes
End
C
No
matching StudentID in database?
Yes
display operation options
No
display error message (invalid StudentID)
admin input = 1?
Yes
request admin input for StudentName
update database
C
No
admin input = 2?
Yes
request admin input for School
update database
C
No
admin input = 3?
Yes
request admin input for Form
update database
C
No
admin input = 4?
Yes
request admin input for ParentName
update database
C
No
admin input = 5?
Yes
request admin input for ParentHP
valid HP format?
No
Yes
update database
C
No
admin input = 6?
Yes
AddDropSubj
No
admin input = 7?
Yes
delete entry matching StudentID from database
C
No
C

Figure 3.7 Update Student Entry

Figures 3.8 – 3.15 show the user operation flow in student database module. Admin may choose to view subject schedule (Figure 3.9), establish new classes (Figure 3.10) or updating existing classes (Figure 3.11 - Figure 3.14), while staff may view subject schedule and namelists (Figure 3.15) of their assigned classes.

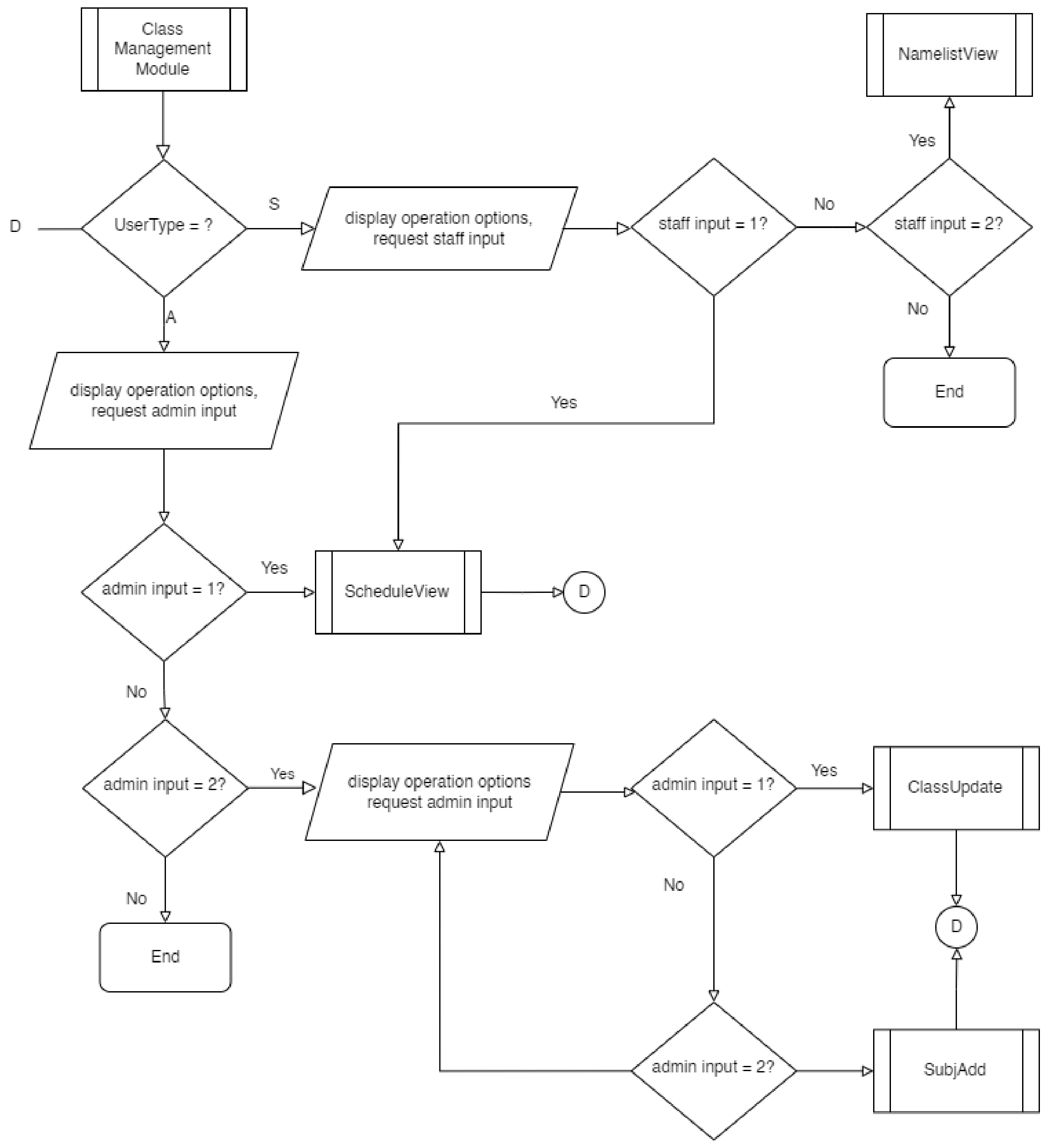


Figure 3.8 Class Management Module

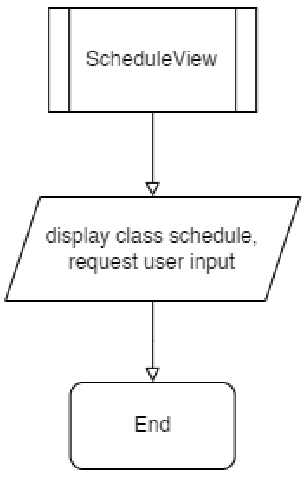


Figure 3.9 Schedule View

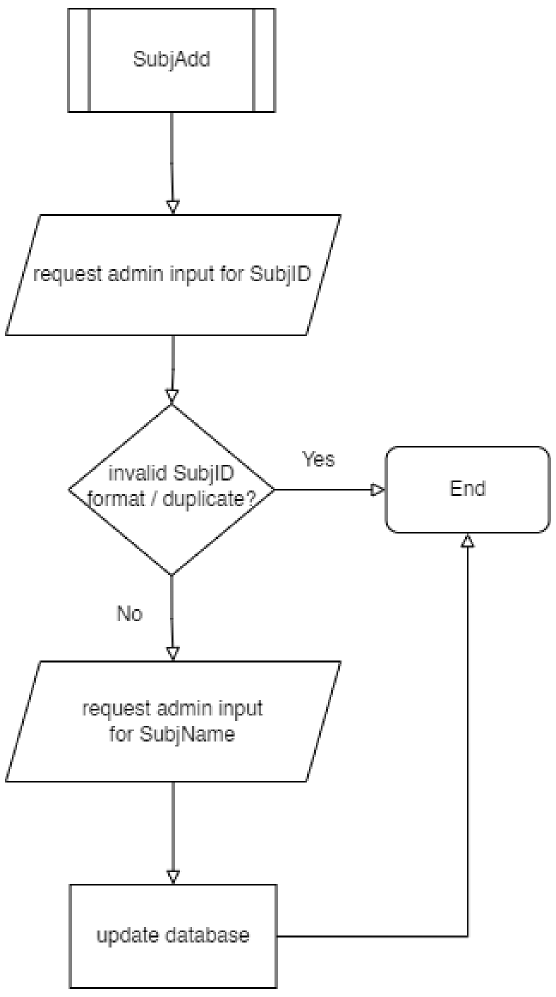

SubjAdd
request admin input for SubjID
invalid SubjID format / duplicate?
Yes
End
No
request admin input for SubjName
update database

Figure 3.10 Adding Subject

Figures 3.11 – 3.14 show the operation flow for updating subject details. Admins can reschedule a subject, update teacher assignation, and disestablish a subject.

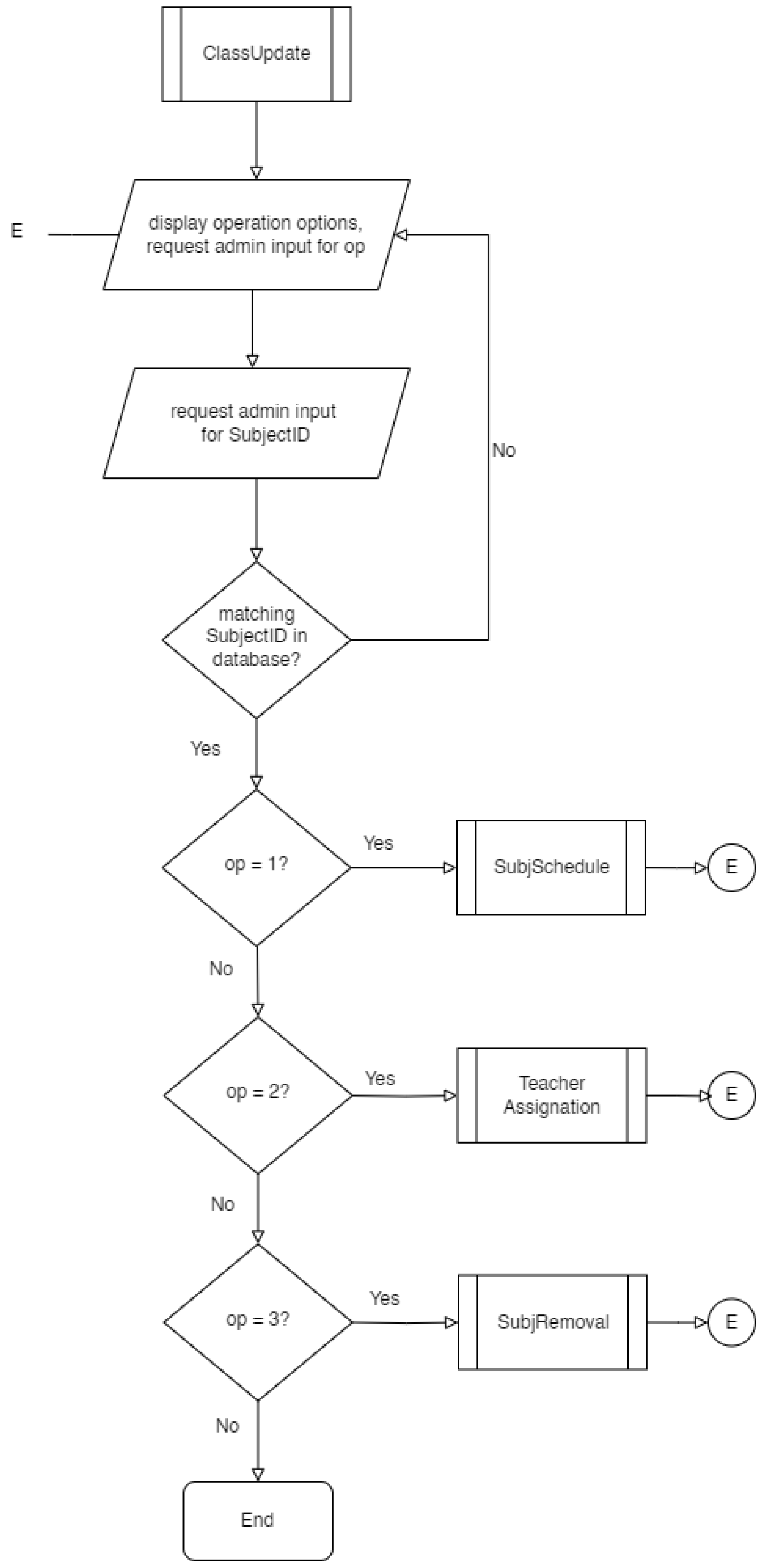

ClassUpdate
E
display operation options, request admin input for op
request admin input for SubjectID
No
matching SubjectID in database?
Yes
op = 1?
Yes
SubjSchedule
E
No
op = 2?
Yes
Teacher Assignation
E
No
op = 3?
Yes
SubjRemoval
E
No
End

Figure 3.11 Updating Subject

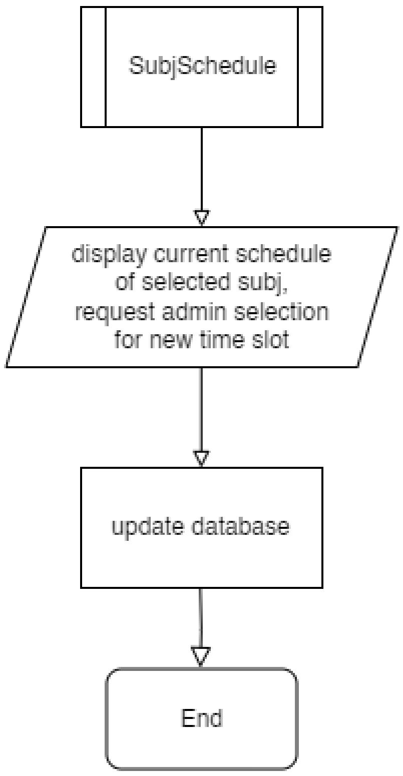

SubjSchedule
display current schedule of selected subj, request admin selection for new time slot
update database
End

Figure 3.12 Subject Scheduling

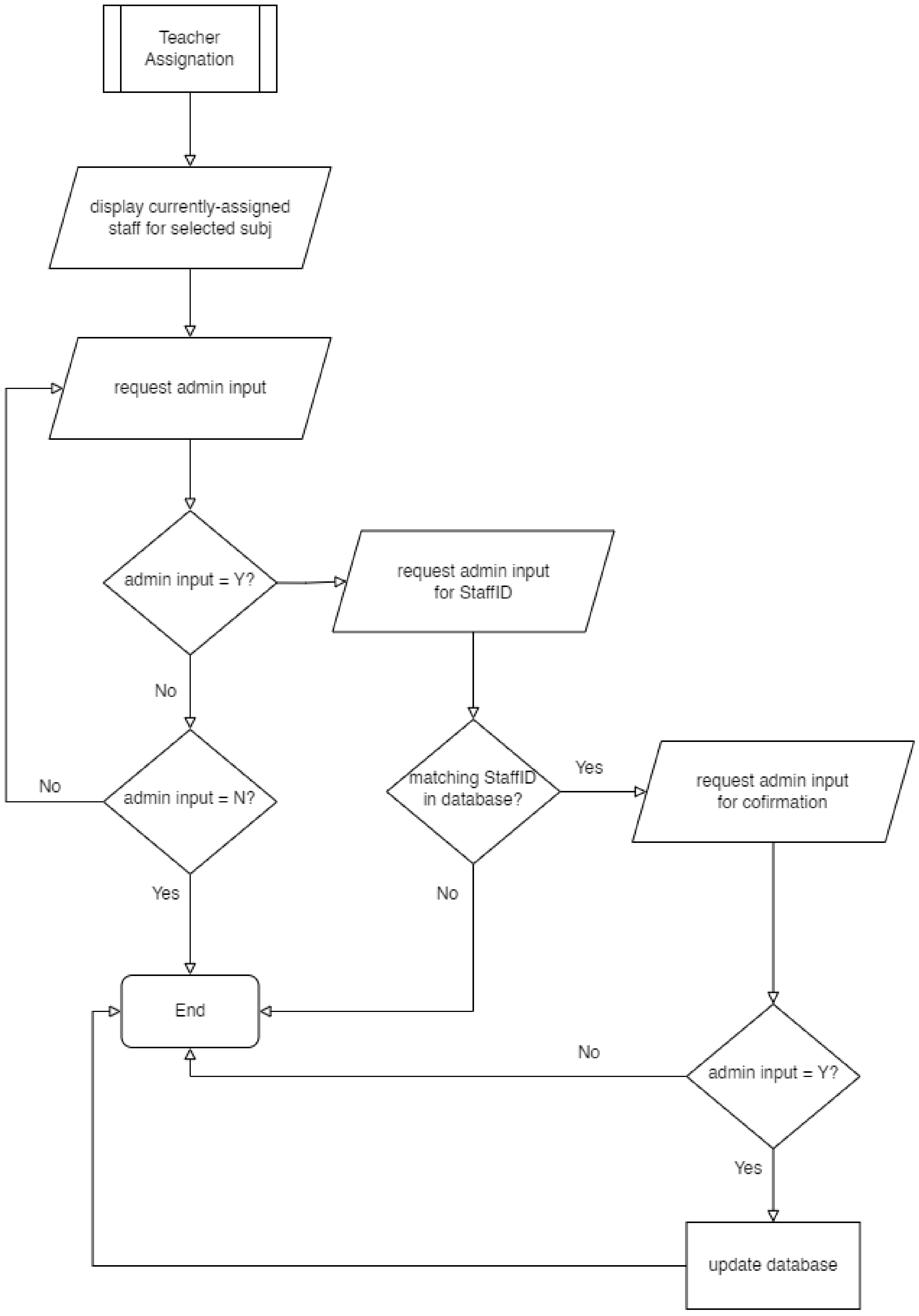

Teacher Assignation
display currently-assigned staff for selected subj
request admin input
admin input = Y?
request admin input for StaffID
No
admin input = N?
No
matching StaffID in database?
Yes
request admin input for cofirmation
Yes
No
End
No
admin input = Y?
Yes
update database

Figure 3.13 Teacher Assignation

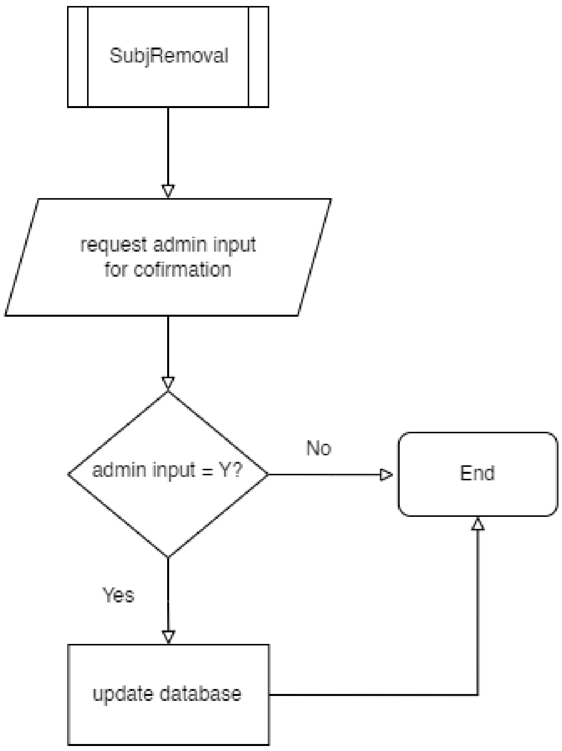


Figure 3.14 Subject Removal

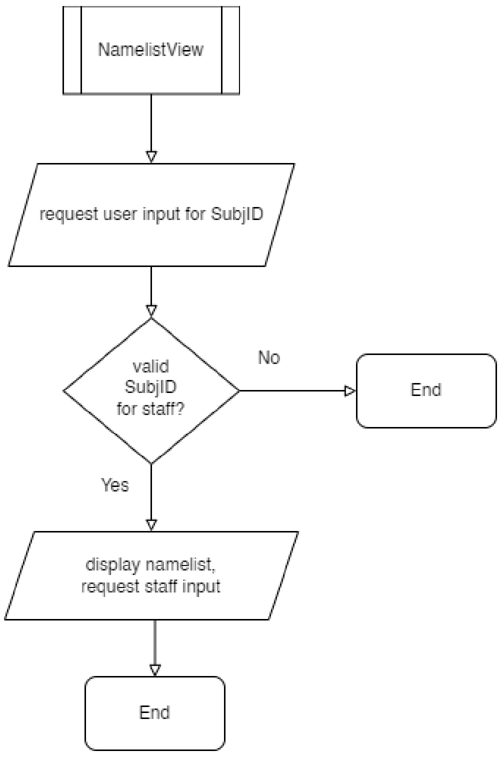

NamelistView
request user input for SubjID
valid SubjID for staff?
No
End
Yes
display namelist, request staff input
End

Figure 3.15 Namelist View

Figures 3.16 – 3.20 demonstrate the operation flow in student performance module. While both admins and staff may view the student performance tracksheet (Figure 3.17), only staff are allowed to update the students' performance record (Figure 3.18 – 3.20).

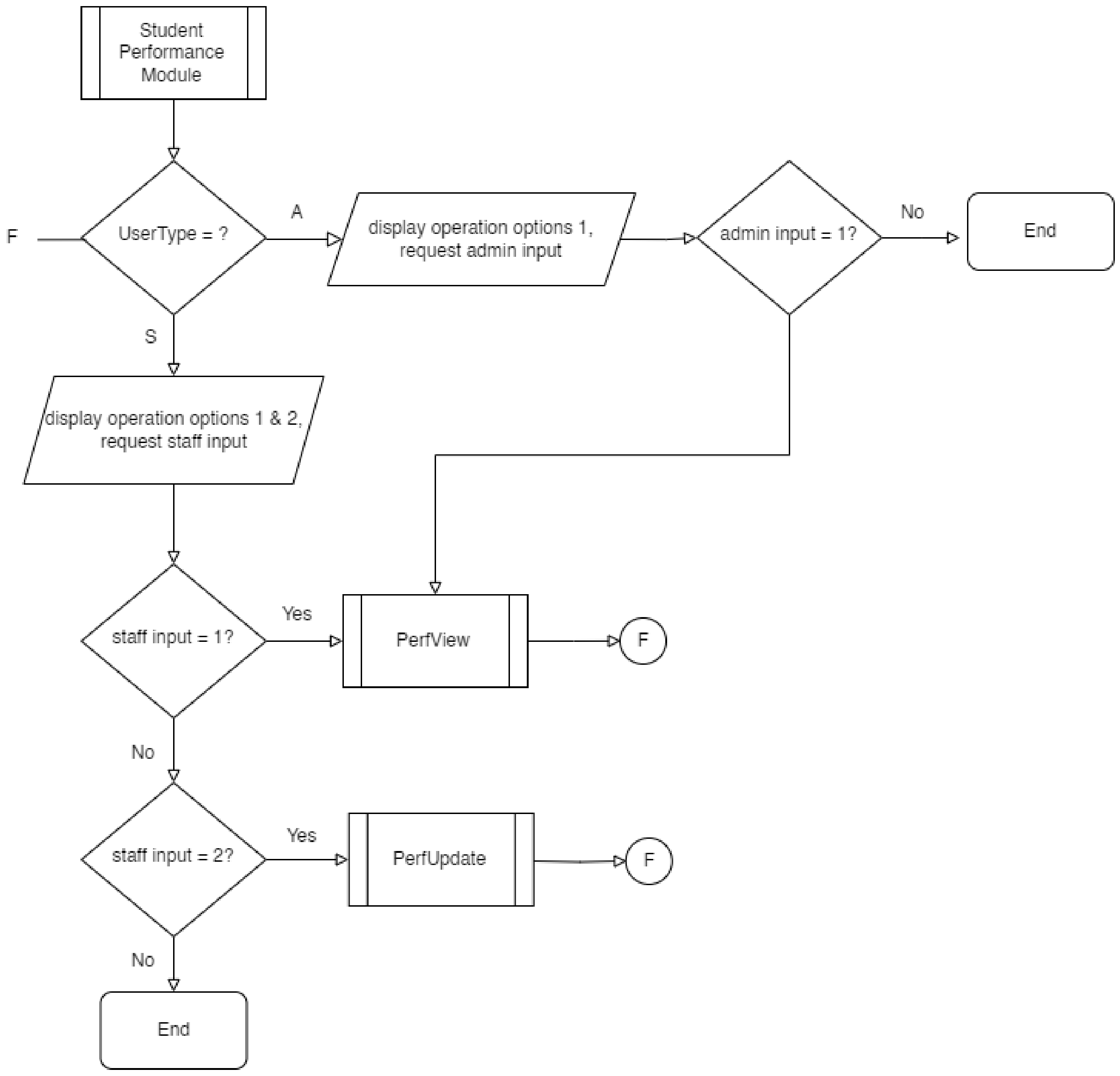

Student Performance Module
F
UserType = ?
A
display operation options 1, request admin input
admin input = 1?
No
End
S
display operation options 1 & 2, request staff input
staff input = 1?
Yes
PerfView
F
No
staff input = 2?
Yes
PerfUpdate
F
No
End

Figure 3.16 Student Performance Module

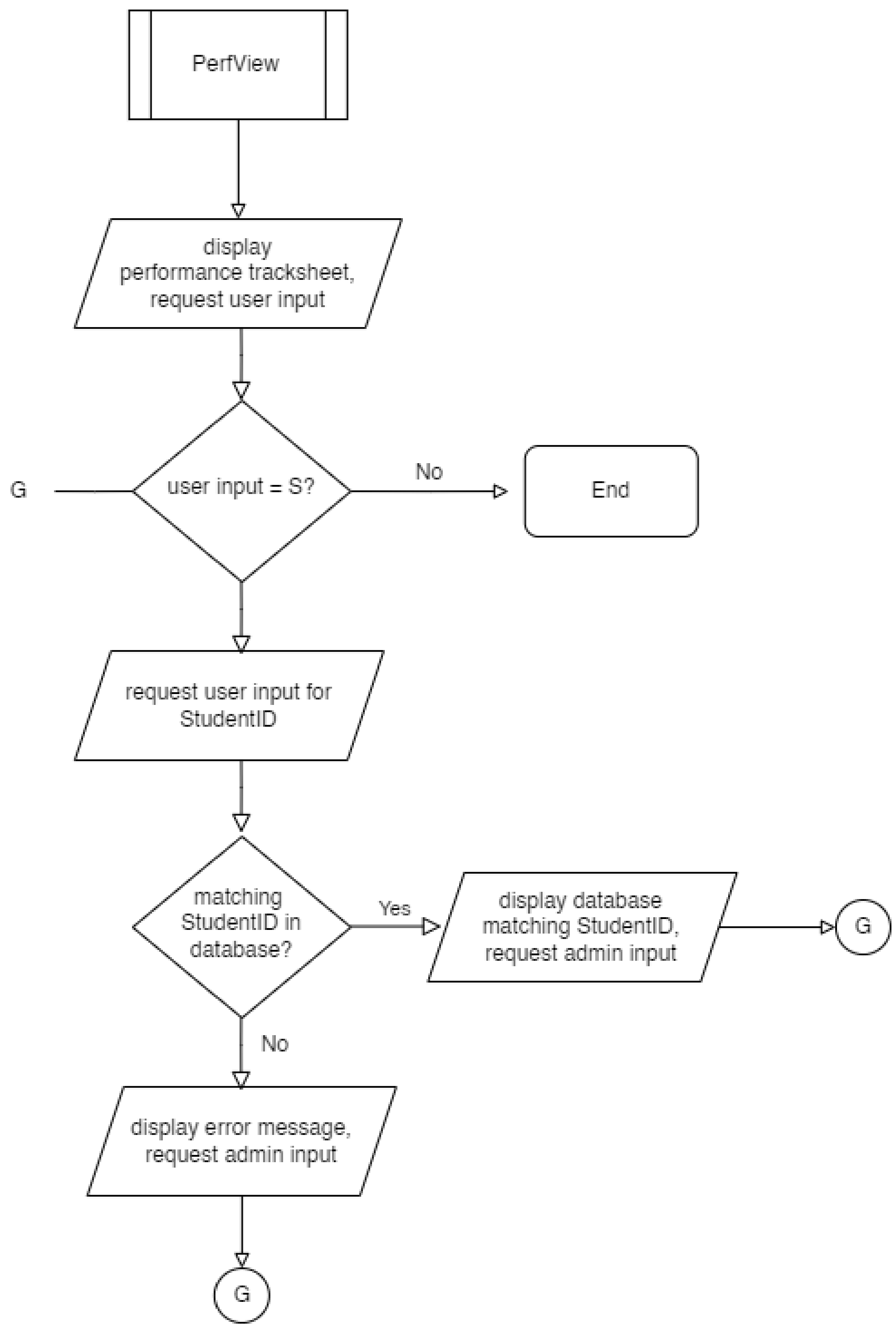

PerfView
display performance tracksheet, request user input
G
user input = S?
No
End
request user input for StudentID
matching StudentID in database?
Yes
display database matching StudentID, request admin input
G
No
display error message, request admin input
G

Figure 3.17 Performance View

Figures 3.18 – 3.20 show the operation flow for student performance update. Once a staff has recorded 3 or more scores for a student, a comment on the student's performance trend will be auto-generated.

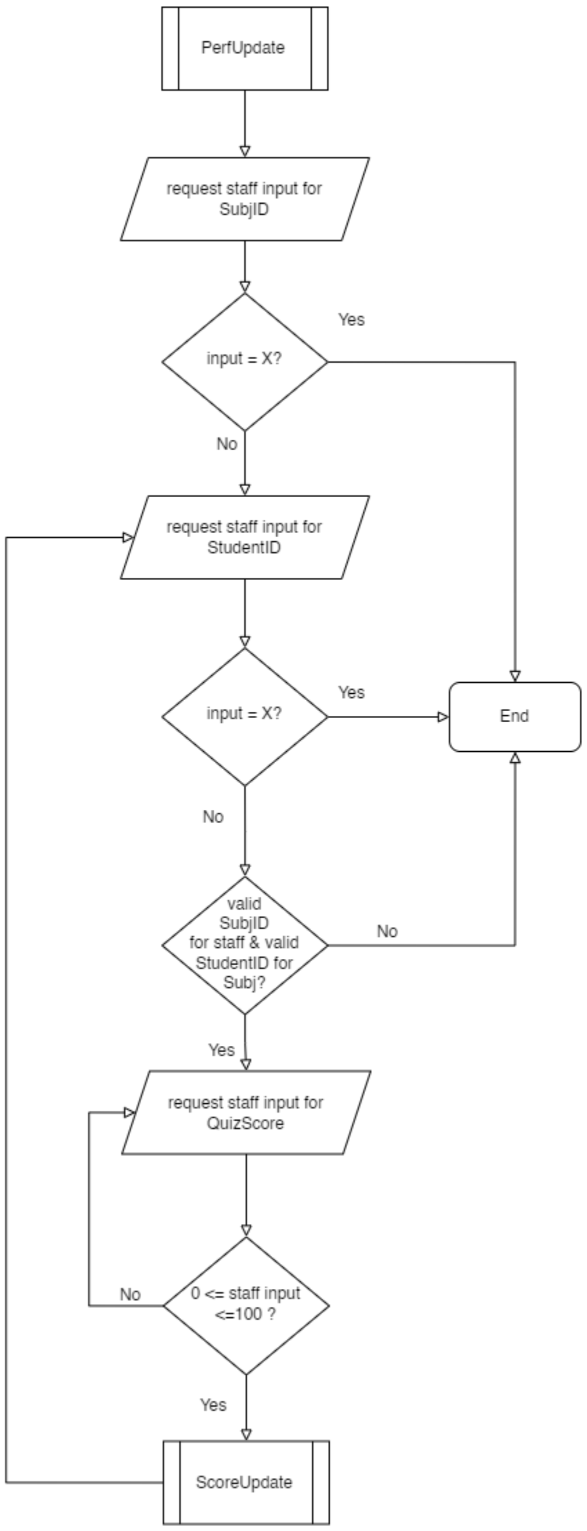

PerfUpdate
request staff input for SubjID
input = X?
Yes
No
request staff input for StudentID
input = X?
Yes
End
No
valid SubjID for staff & valid StudentID for Subj?
No
Yes
request staff input for QuizScore
No
0 <= staff input <=100 ?
Yes
ScoreUpdate

Figure 3.18 Performance Update

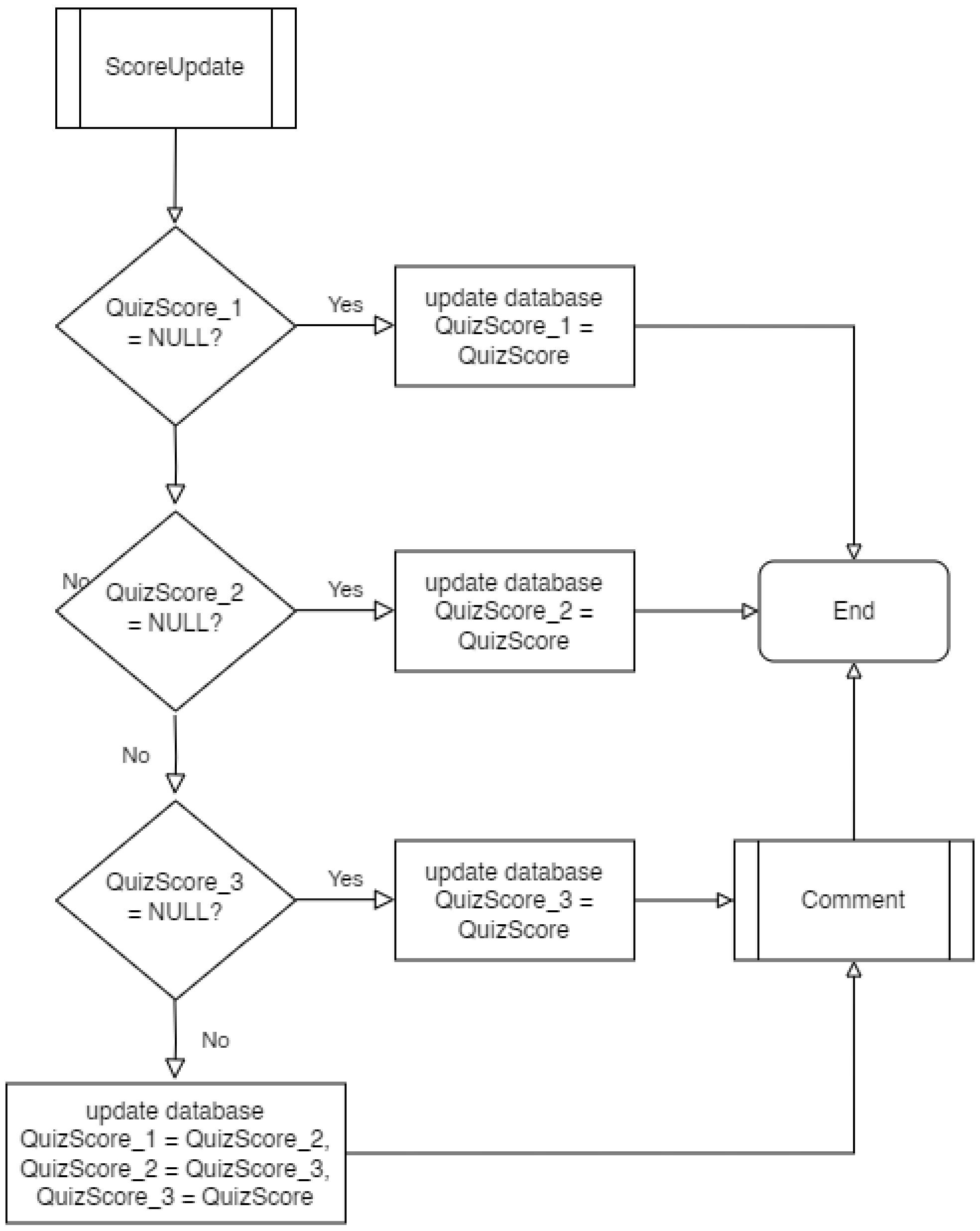

ScoreUpdate
QuizScore_1 = NULL?
Yes
update database QuizScore_1 = QuizScore
No
QuizScore_2 = NULL?
Yes
update database QuizScore_2 = QuizScore
End
No
QuizScore_3 = NULL?
Yes
update database QuizScore_3 = QuizScore
Comment
No
update database QuizScore_1 = QuizScore_2, QuizScore_2 = QuizScore_3, QuizScore_3 = QuizScore

Figure 3.19 Score Update

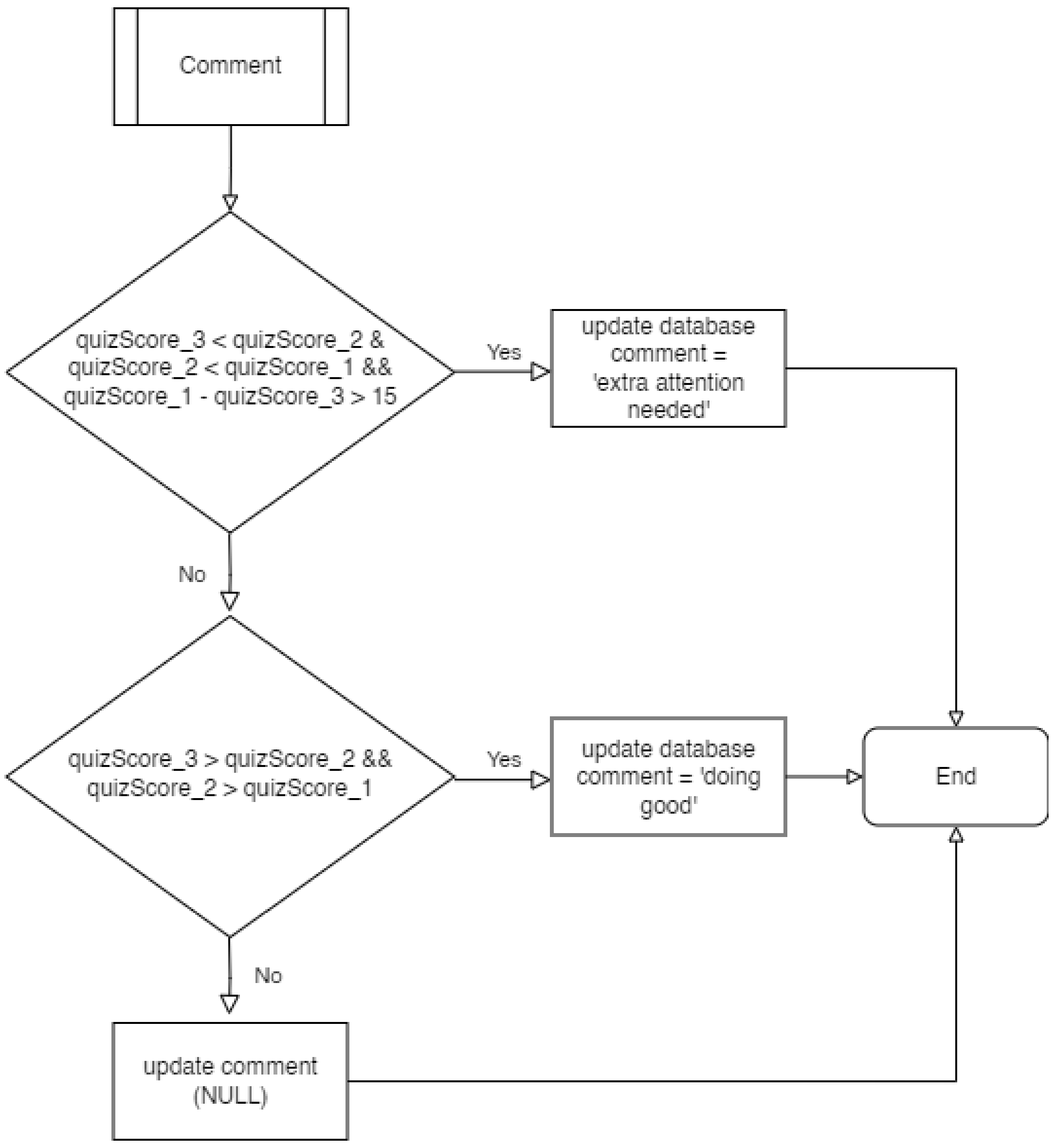

Comment
quizScore_3 < quizScore_2 & quizScore_2 < quizScore_1 && quizScore_1 - quizScore_3 > 15
Yes
update database comment = 'extra attention needed'
No
quizScore_3 > quizScore_2 && quizScore_2 > quizScore_1
Yes
update database comment = 'doing good'
End
No
update comment (NULL)

Figure 3.20 Comment Generation

Figures 3.21 – 3.23 demonstrate the operation flow in finance management module. Admin may record and delete finance entries (Figure 3.22), besides viewing the financial report (Figure 3.23).

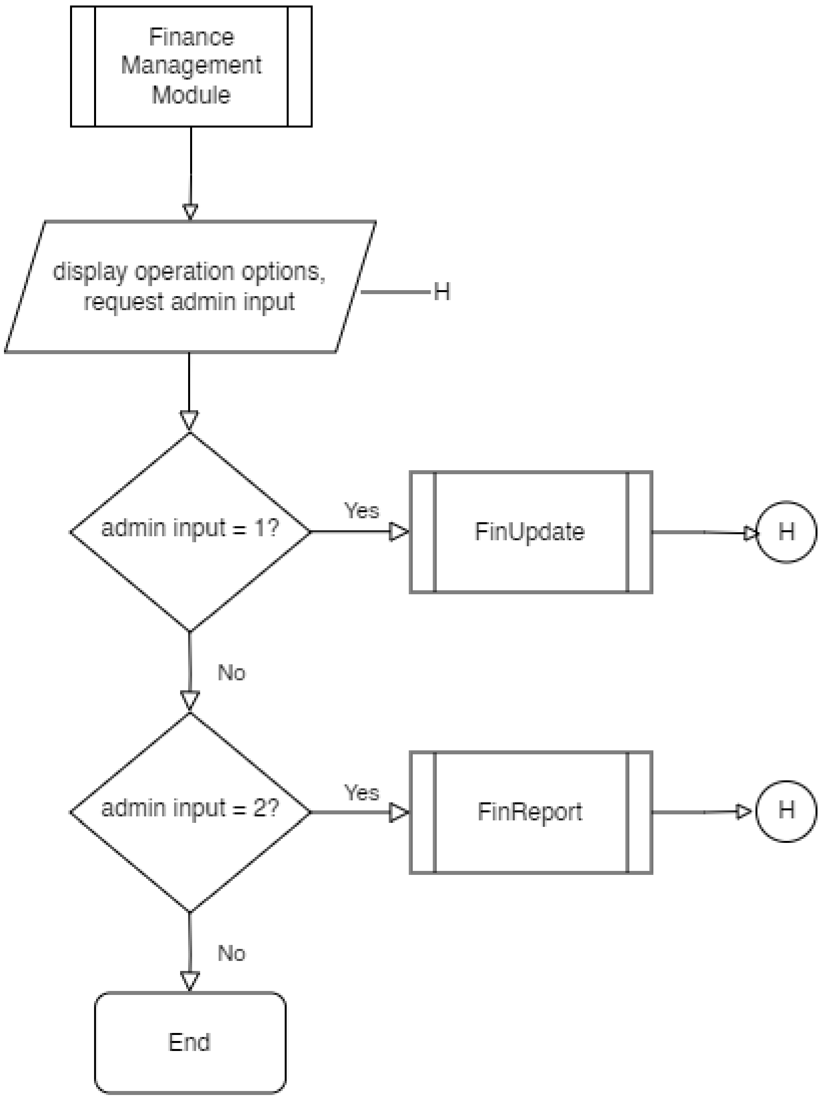

Finance Management Module
display operation options, request admin input
H
admin input = 1?
Yes
FinUpdate
H
No
admin input = 2?
Yes
FinReport
H
No
End

Figure 3.21 Finance Management Module

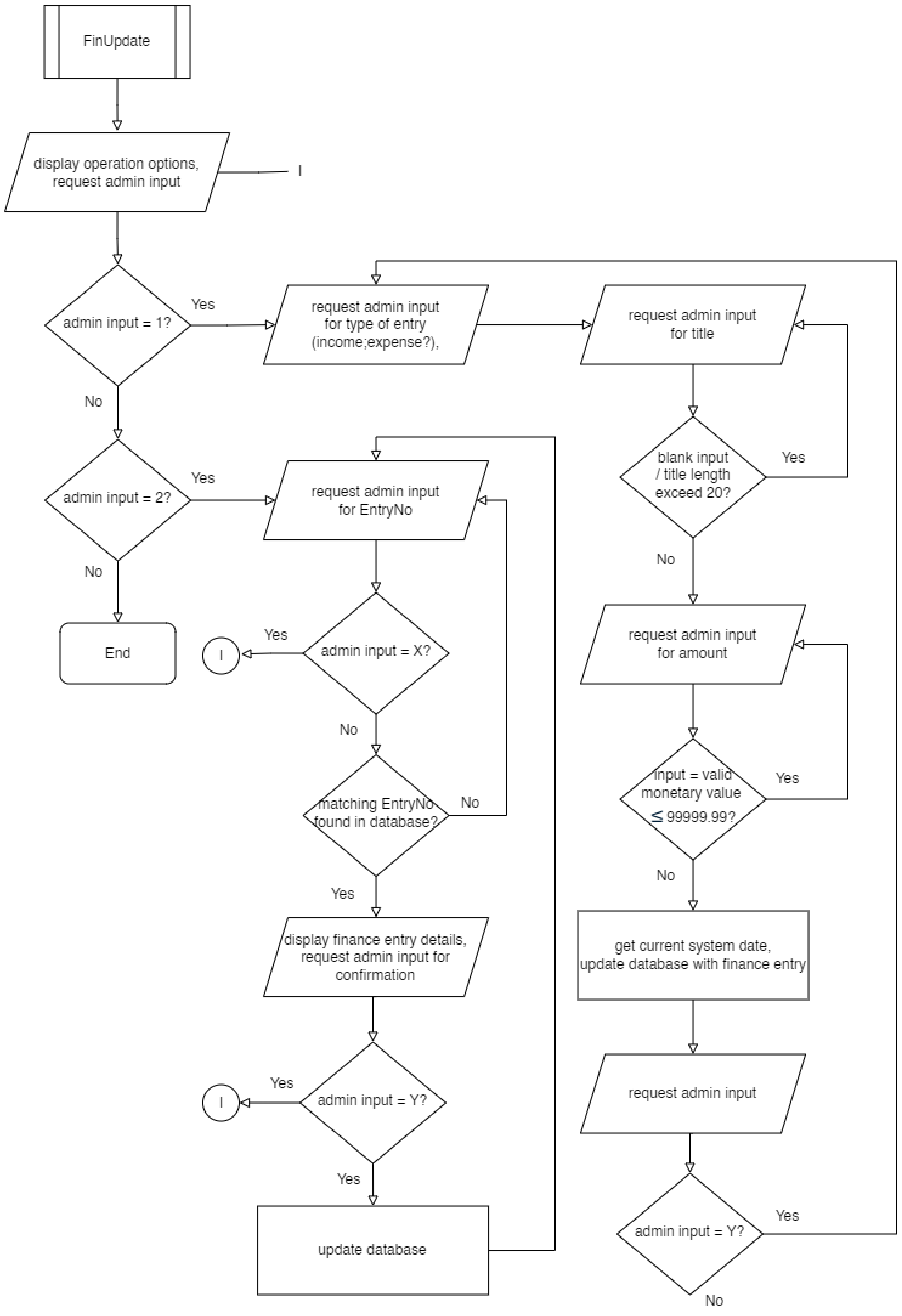

FinUpdate
display operation options, request admin input
I
admin input = 1?
Yes
No
request admin input for type of entry (income;expense?),
request admin input for title
blank input / title length exceed 20?
Yes
No
request admin input for amount
input = valid monetary value ≤ 99999.99?
Yes
No
get current system date, update database with finance entry
request admin input
admin input = Y?
Yes
No
admin input = 2?
Yes
No
End
request admin input for EntryNo
admin input = X?
Yes
I
No
matching EntryNo found in database?
No
Yes
display finance entry details, request admin input for confirmation
admin input = Y?
Yes
I
Yes
update database

Figure 3.22 Finance Update

Figure 3.23 shows the operation flow for viewing financial report. Besides the overall report, admin may also filter financial report by month and generate corresponding analysis on monthly balance.

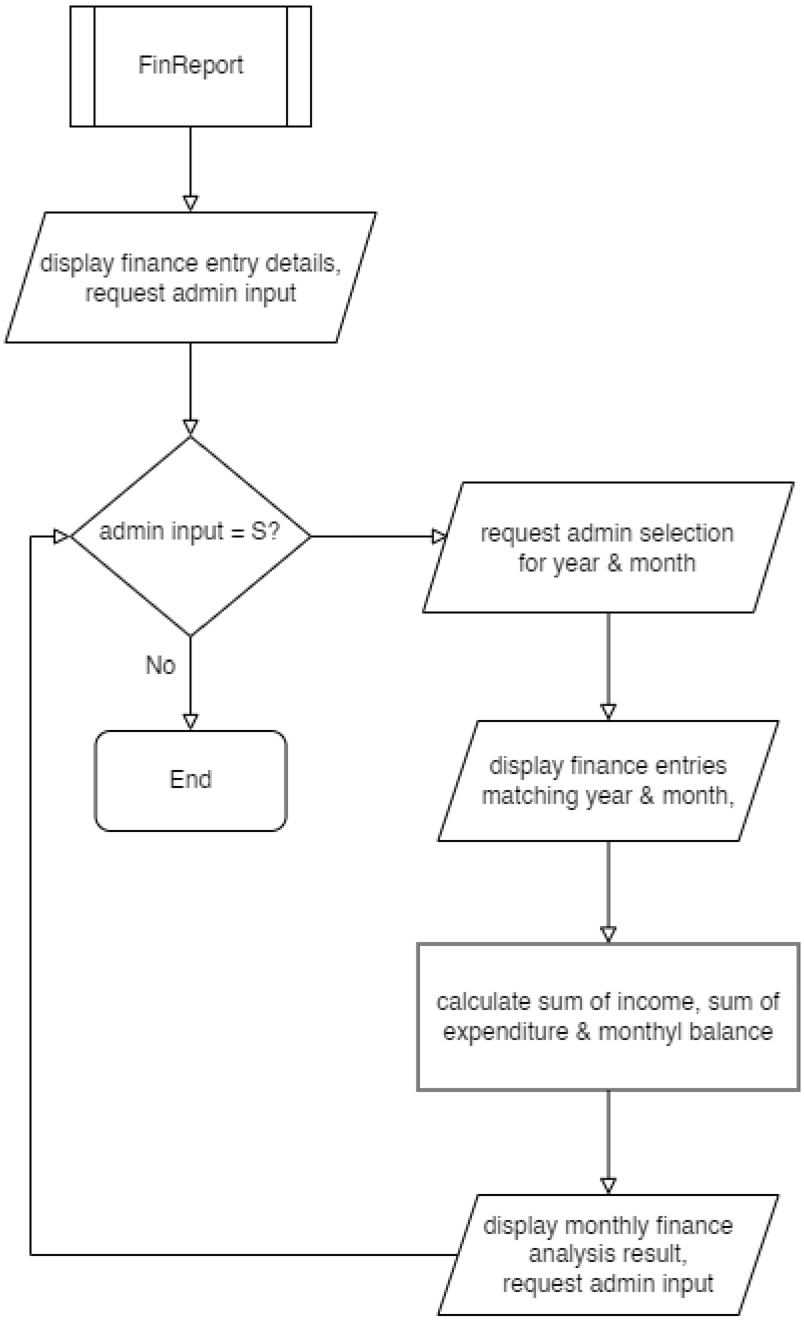

FinReport
display finance entry details, request admin input
admin input = S?
No
End
request admin selection for year & month
display finance entries matching year & month,
calculate sum of income, sum of expenditure & monthyl balance
display monthly finance analysis result, request admin input

Figure 3.23 Financial Report Generation

3.2 ERD

Figure 3.24 shows the entity-relationship diagram (ERD) for Tuition Centre Manager's MYSQL database.

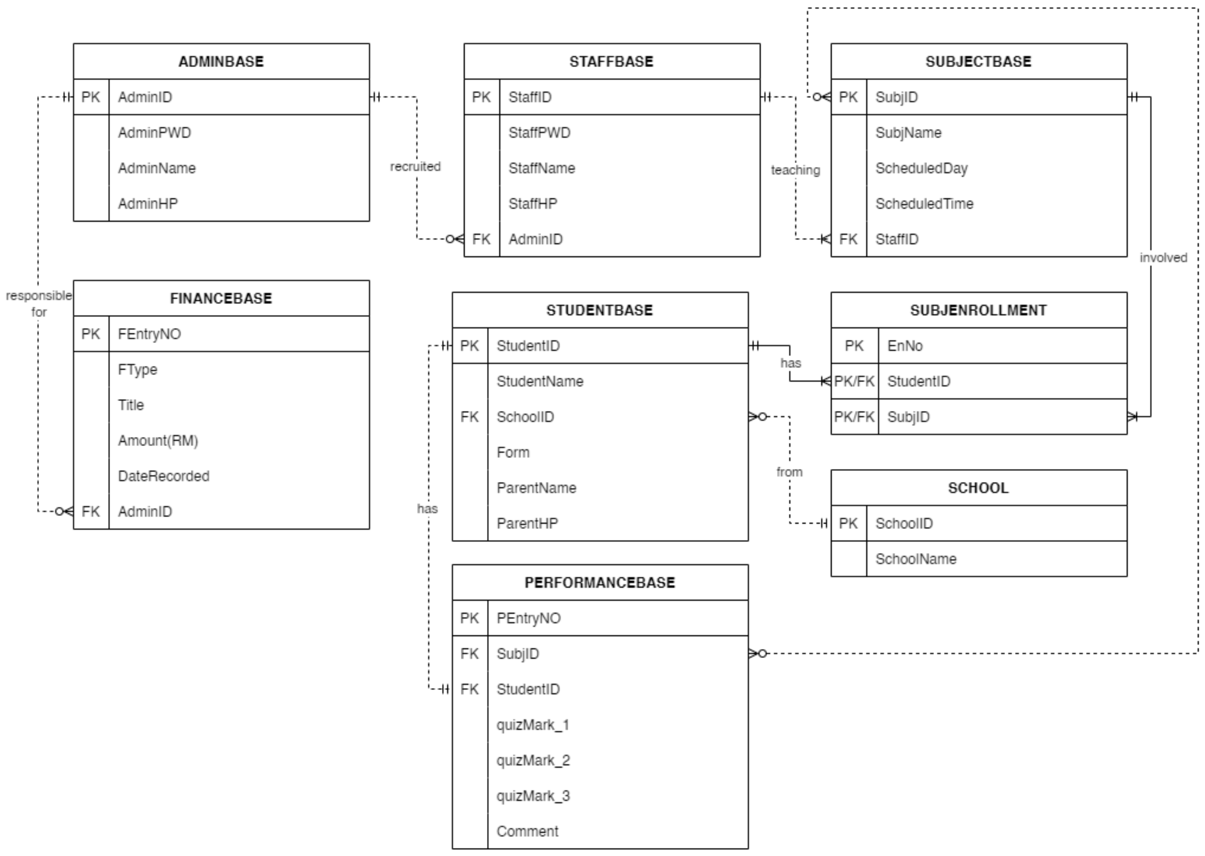
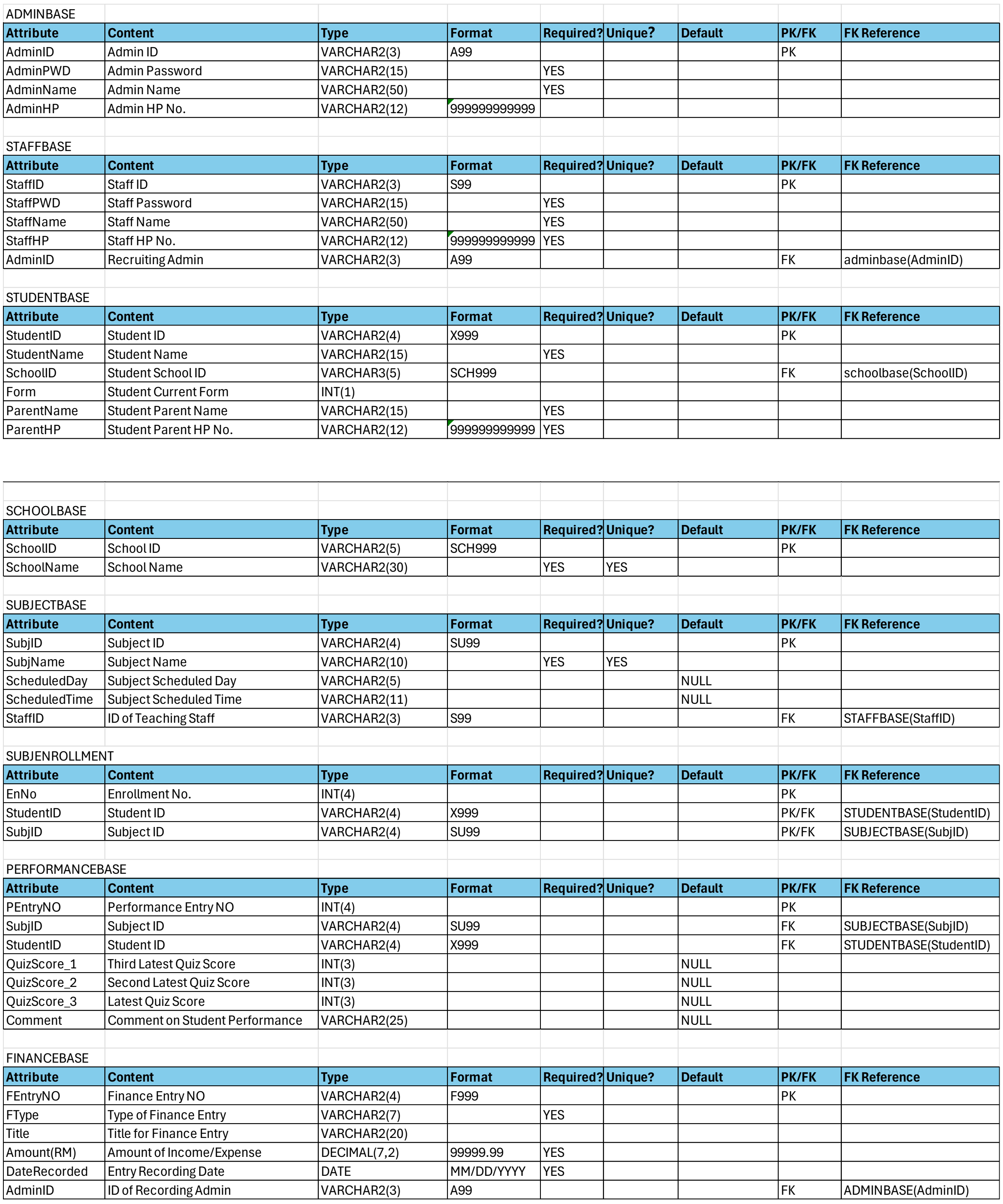


Figure 3.24 ERD

3.3 Data Dictionary

Figure 3.25 presents the data dictionary.



ADMINBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
AdminID	Admin ID	VARCHAR2(3)	A99				PK	
AdminPWD	Admin Password	VARCHAR2(15)		YES				
AdminName	Admin Name	VARCHAR2(50)		YES				
AdminHP	Admin HP No.	VARCHAR2(12)	999999999999					

STAFFBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
StaffID	Staff ID	VARCHAR2(3)	S99				PK	
StaffPWD	Staff Password	VARCHAR2(15)		YES				
StaffName	Staff Name	VARCHAR2(50)		YES				
StaffHP	Staff HP No.	VARCHAR2(12)	999999999999	YES				
AdminID	Recruiting Admin	VARCHAR2(3)	A99				FK	adminbase(AdminID)

STUDENTBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
StudentID	Student ID	VARCHAR2(4)	X999				PK	
StudentName	Student Name	VARCHAR2(15)		YES				
SchoolID	Student School ID	VARCHAR3(5)	SCH999				FK	schoolbase(SchoolID)
Form	Student Current Form	INT(1)						
ParentName	Student Parent Name	VARCHAR2(15)		YES				
ParentHP	Student Parent HP No.	VARCHAR2(12)	999999999999	YES				

SCHOOLBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
SchoolID	School ID	VARCHAR2(5)	SCH999				PK	
SchoolName	School Name	VARCHAR2(30)		YES	YES			

SUBJECTBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
SubjID	Subject ID	VARCHAR2(4)	SU99				PK	
SubjName	Subject Name	VARCHAR2(10)		YES	YES			
ScheduledDay	Subject Scheduled Day	VARCHAR2(5)				NULL		
ScheduledTime	Subject Scheduled Time	VARCHAR2(11)				NULL		
StaffID	ID of Teaching Staff	VARCHAR2(3)	S99				FK	STAFFBASE(StaffID)

SUBJENROLLMENT

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
EnNo	Enrollment No.	INT(4)					PK	
StudentID	Student ID	VARCHAR2(4)	X999				PK/FK	STUDENTBASE(StudentID)
SubjID	Subject ID	VARCHAR2(4)	SU99				PK/FK	SUBJECTBASE(SubjID)

PERFORMANCEBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
PEntryNO	Performance Entry NO	INT(4)					PK	
SubjID	Subject ID	VARCHAR2(4)	SU99				FK	SUBJECTBASE(SubjID)
StudentID	Student ID	VARCHAR2(4)	X999				FK	STUDENTBASE(StudentID)
QuizScore_1	Third Latest Quiz Score	INT(3)				NULL		
QuizScore_2	Second Latest Quiz Score	INT(3)				NULL		
QuizScore_3	Latest Quiz Score	INT(3)				NULL		
Comment	Comment on Student Performance	VARCHAR2(25)				NULL		

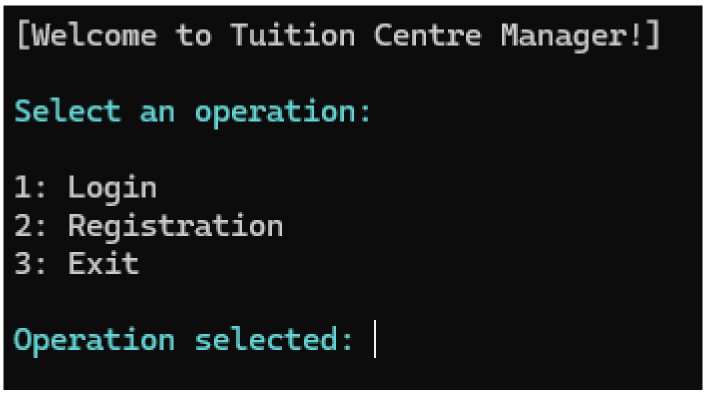
FINANCEBASE

Attribute	Content	Type	Format	Required?	Unique?	Default	PK/FK	FK Reference
FEntryNO	Finance Entry NO	VARCHAR2(4)	F999				PK	
FType	Type of Finance Entry	VARCHAR2(7)		YES				
Title	Title for Finance Entry	VARCHAR2(20)						
Amount(RM)	Amount of Income/Expense	DECIMAL(7,2)	99999.99	YES				
DateRecorded	Entry Recording Date	DATE	MM/DD/YYYY	YES				
AdminID	ID of Recording Admin	VARCHAR2(3)	A99				FK	ADMINBASE(AdminID)

Figure 3.25 Data Dictionary

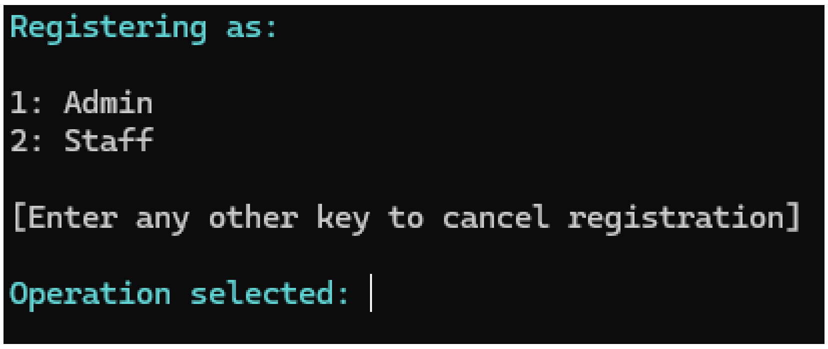
3.4 Interface Design

Figure 3.26 – 3.28 show the very first operations encountered by user. Admins and staff both need to register an account before loggin into the system.



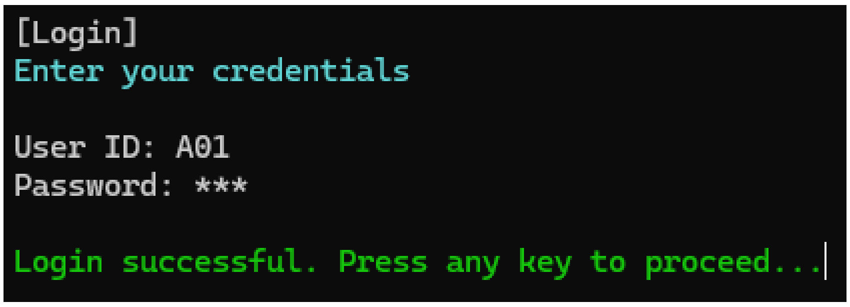
```
[Welcome to Tuition Centre Manager!]

Select an operation:

1: Login
2: Registration
3: Exit

Operation selected:
```

Figure 3.26: Welcome Page



```
Registering as:

1: Admin
2: Staff

[Enter any other key to cancel registration]

Operation selected:
```

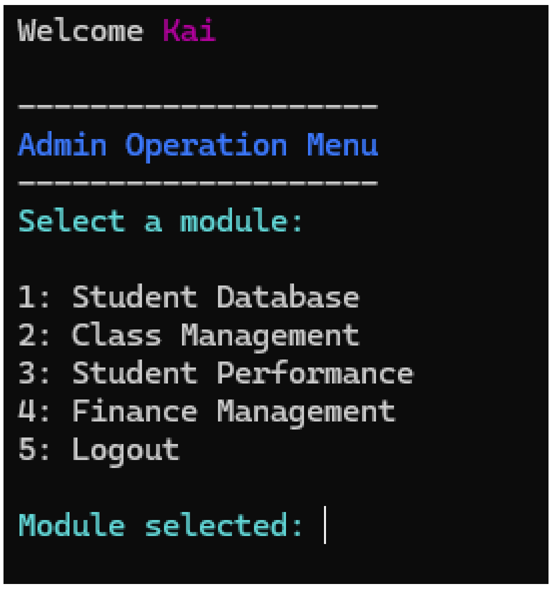
Figure 3.27: Registration Page



```
[Login]
Enter your credentials

User ID: A01
Password: ***

Login successful. Press any key to proceed...
```

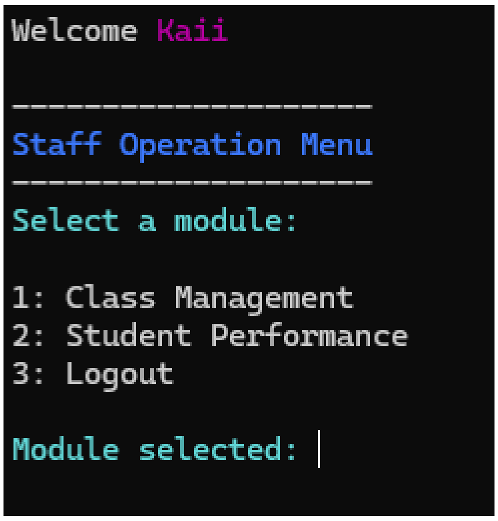
Figure 3.28: Login Page

Figures 3.29 and 3.30 show the main operation menu of an admin and staff respectively. Only the modules authorized for the user type will be visible and selectable.



```
Welcome Kai

--------------------
Admin Operation Menu
--------------------
Select a module:

1: Student Database
2: Class Management
3: Student Performance
4: Finance Management
5: Logout

Module selected:
```

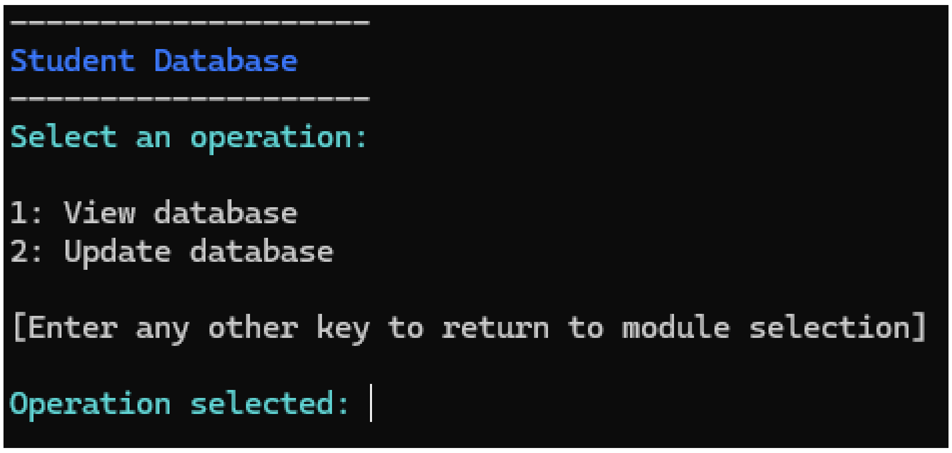
Figure 3.29: Admin Main Operation Menu



```
Welcome Kaii

--------------------
Staff Operation Menu
--------------------
Select a module:

1: Class Management
2: Student Performance
3: Logout

Module selected:
```

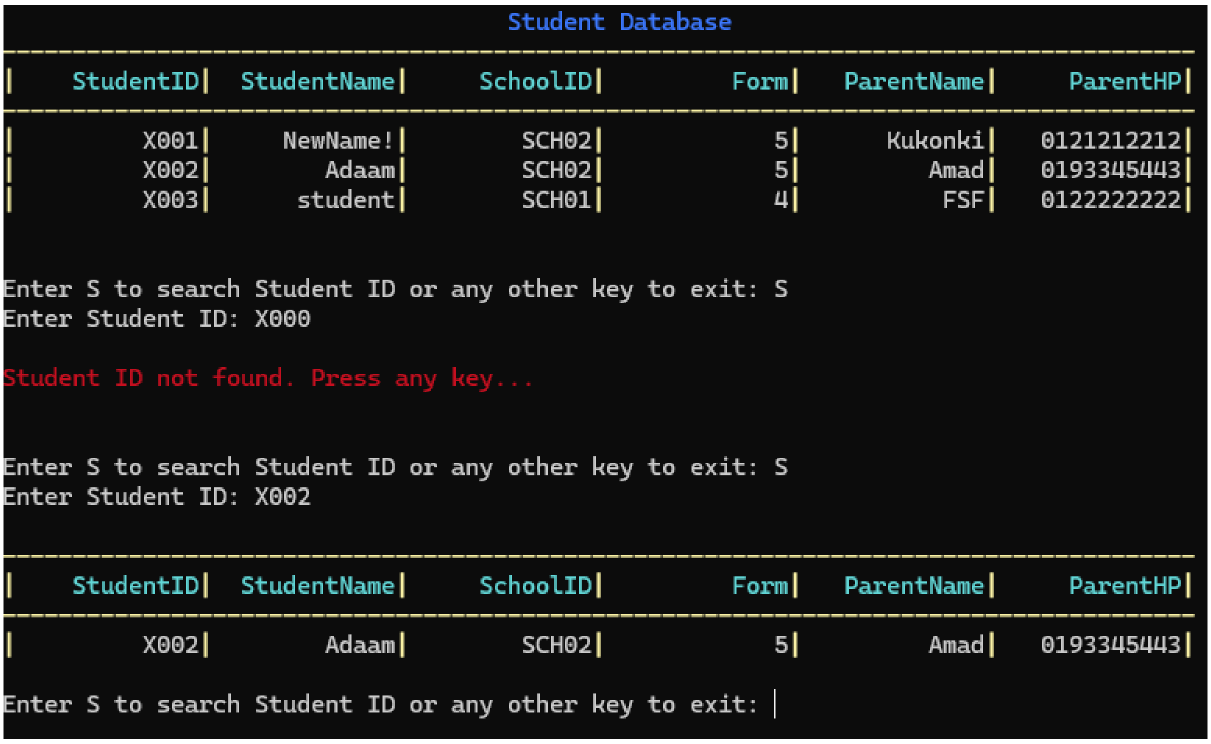
Figure 3.30: Staff Main Operation Menu

Figures 3.31 – 3.46 illustrate the student database module operations. Figure 3.32 focuses on student database viewing and filtering, while figures 3.33 – 3.46 shows operations on updating student database.



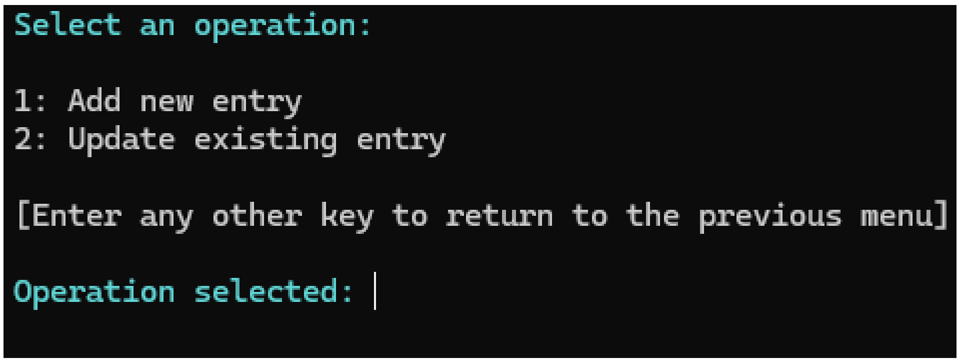
```
--------------------
Student Database
--------------------
Select an operation:

1: View database
2: Update database

[Enter any other key to return to module selection]

Operation selected:
```

Figure 3.31: Student Database Module



```
                              Student Database
-----------------------------------------------------------------------------------
|     StudentID|  StudentName|       SchoolID|           Form|   ParentName|      ParentHP|
-----------------------------------------------------------------------------------
|          X001|     NewName!|          SCH02|              5|      Kukonki|    0121212212|
|          X002|        Adaam|          SCH02|              5|         Amad|    0193345443|
|          X003|      student|          SCH01|              4|          FSF|    0122222222|


Enter S to search Student ID or any other key to exit: S
Enter Student ID: X000

Student ID not found. Press any key...


Enter S to search Student ID or any other key to exit: S
Enter Student ID: X002

-----------------------------------------------------------------------------------
|     StudentID|  StudentName|       SchoolID|           Form|   ParentName|      ParentHP|
-----------------------------------------------------------------------------------
|          X002|        Adaam|          SCH02|              5|         Amad|    0193345443|

Enter S to search Student ID or any other key to exit:
```

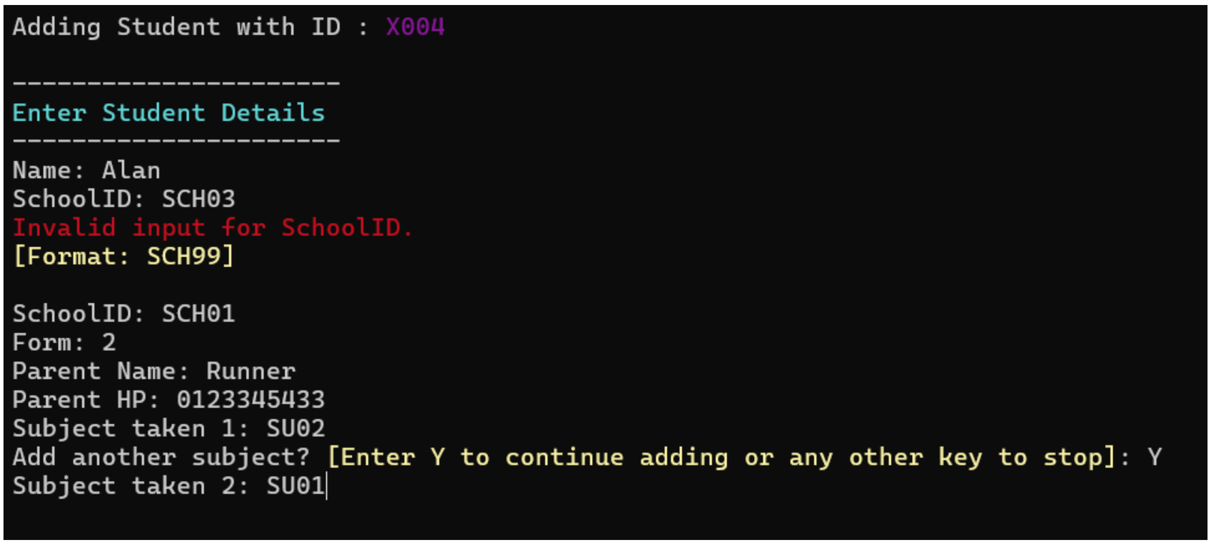
Figure 3.32: Student Database Viewing & Searching



```
Select an operation:

1: Add new entry
2: Update existing entry

[Enter any other key to return to the previous menu]

Operation selected:
```

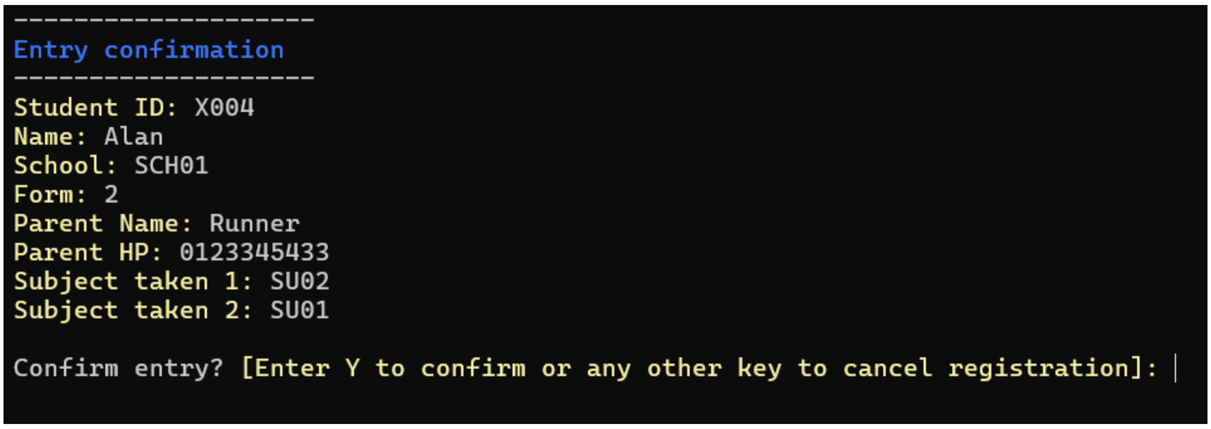
Figure 3.33: Options for Updating Student Database

Figures 3.34 – 3.35 focus on the process of adding new student entry. Admins need to fill in a student's details before being brought to a confirmation page to confirm the insertion of student entry to database.



```
Adding Student with ID : X004

----------------------
Enter Student Details
----------------------
Name: Alan
SchoolID: SCH03
Invalid input for SchoolID.
[Format: SCH99]

SchoolID: SCH01
Form: 2
Parent Name: Runner
Parent HP: 0123345433
Subject taken 1: SU02
Add another subject? [Enter Y to continue adding or any other key to stop]: Y
Subject taken 2: SU01
```

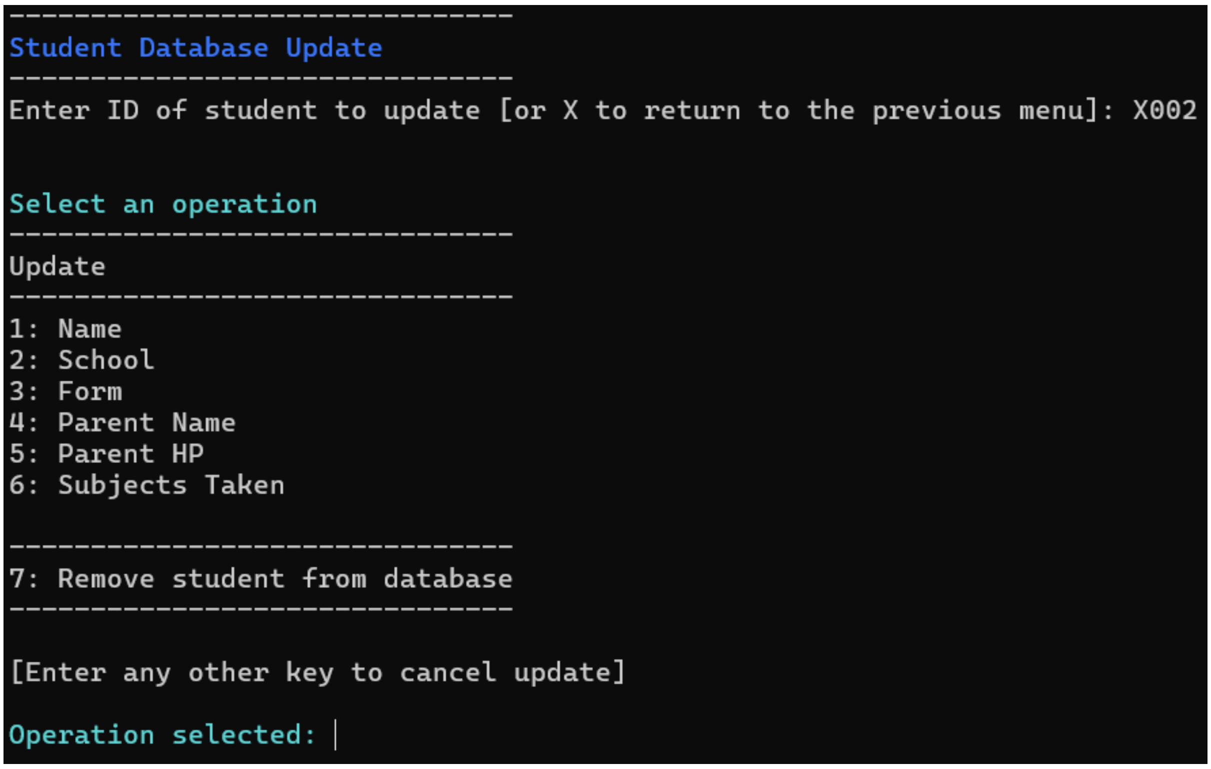
Figure 3.34: Adding Student Entry



```
--------------------
Entry confirmation
--------------------
Student ID: X004
Name: Alan
School: SCH01
Form: 2
Parent Name: Runner
Parent HP: 0123345433
Subject taken 1: SU02
Subject taken 2: SU01

Confirm entry? [Enter Y to confirm or any other key to cancel registration]:
```

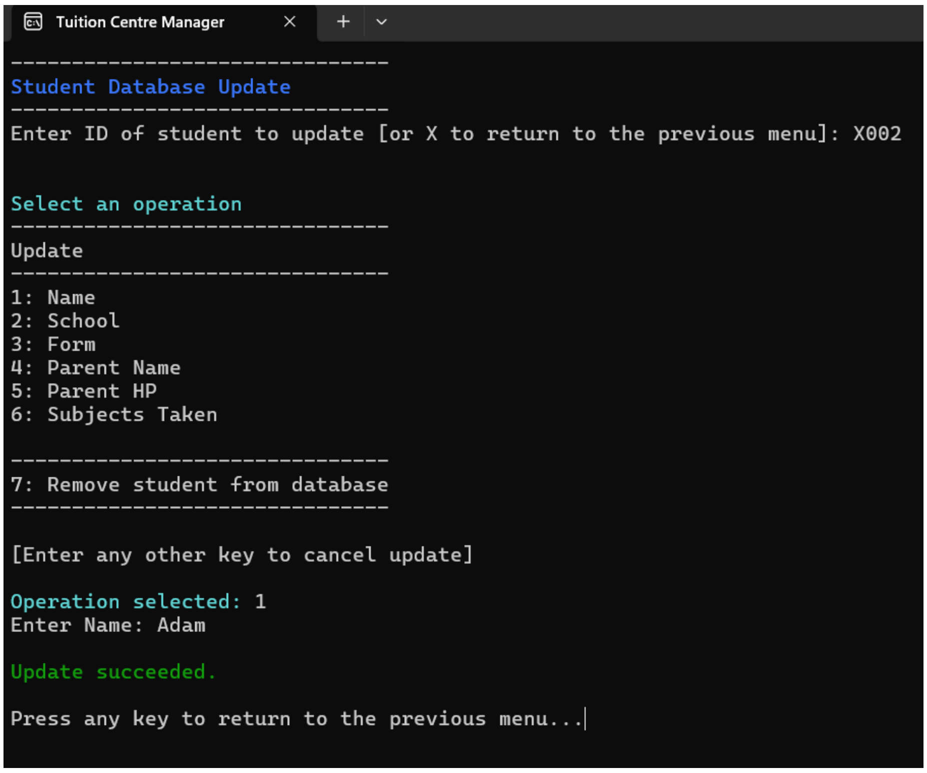
Figure 3.35: Adding Student Entry [Confirmation]

Figures 3.36 – 3.45 focus on updating the details of a student that already exists in database.



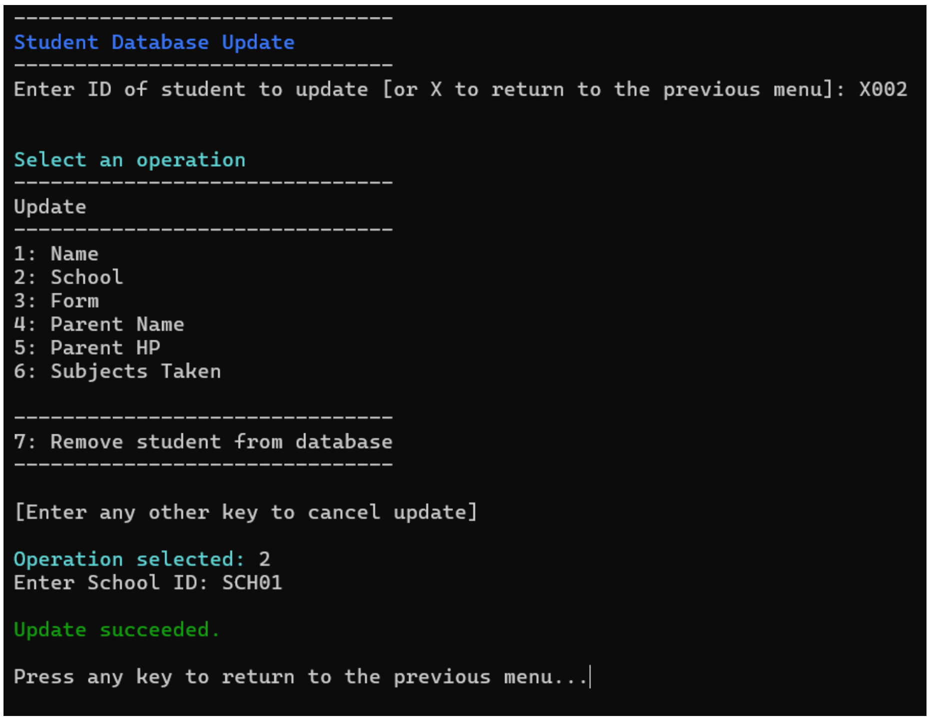
```
-------------------------------
Student Database Update
-------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
-------------------------------
Update
-------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

-------------------------------
7: Remove student from database
-------------------------------

[Enter any other key to cancel update]

Operation selected:
```

Figure 3.36: Student Database Entry Update Operation Selection



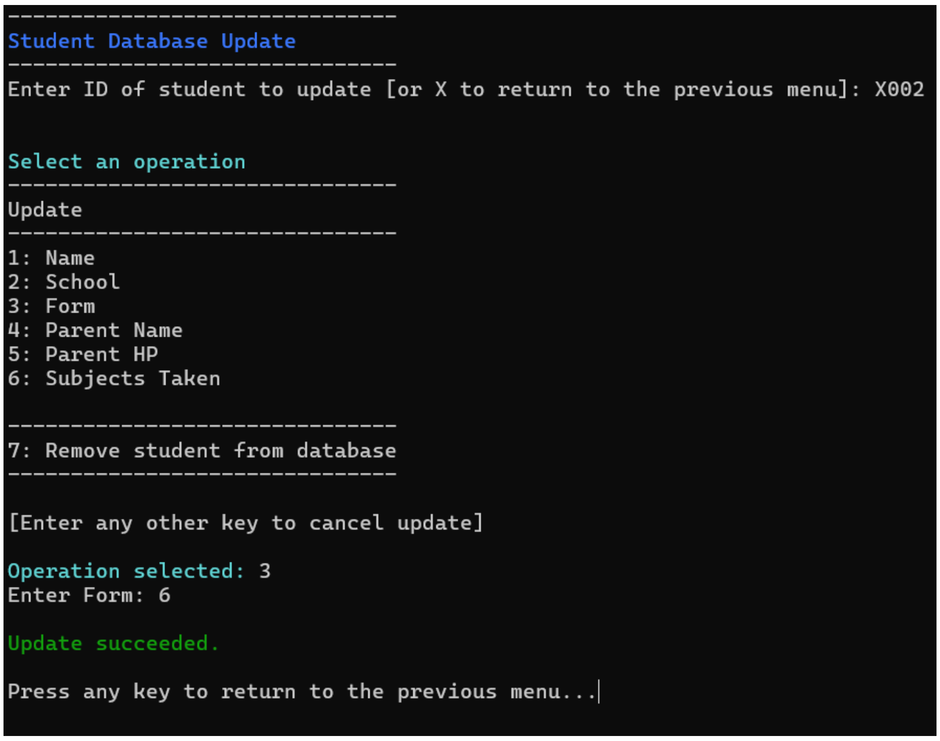
```
-------------------------------
Student Database Update
-------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
-------------------------------
Update
-------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

-------------------------------
7: Remove student from database
-------------------------------

[Enter any other key to cancel update]

Operation selected: 1
Enter Name: Adam

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.37: Updating Student Name



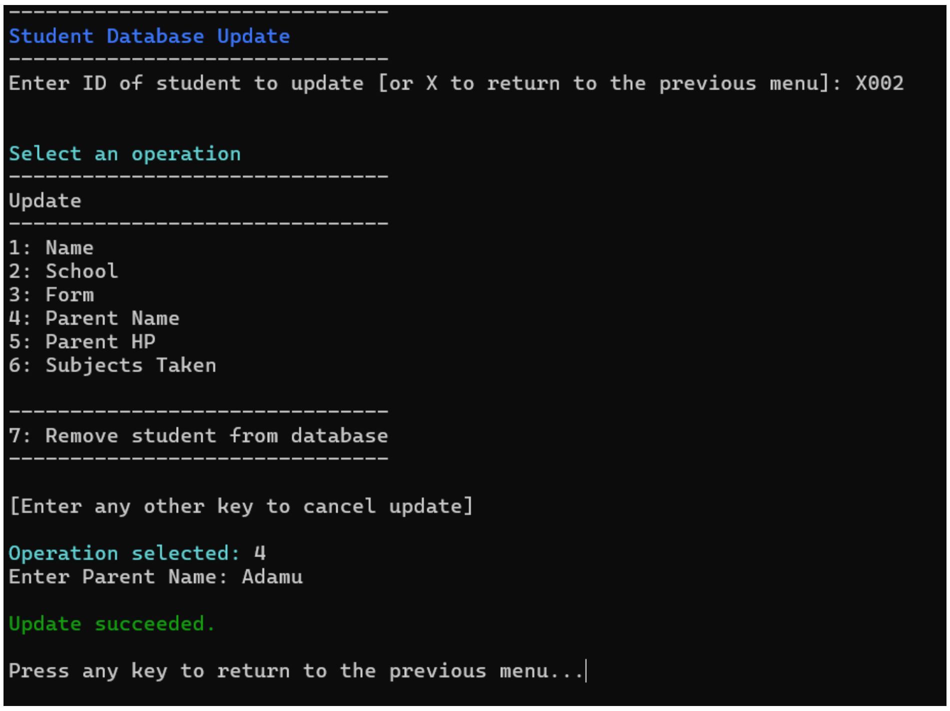
```
-------------------------------
Student Database Update
-------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
-------------------------------
Update
-------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

-------------------------------
7: Remove student from database
-------------------------------

[Enter any other key to cancel update]

Operation selected: 2
Enter School ID: SCH01

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.38: Updating Student School



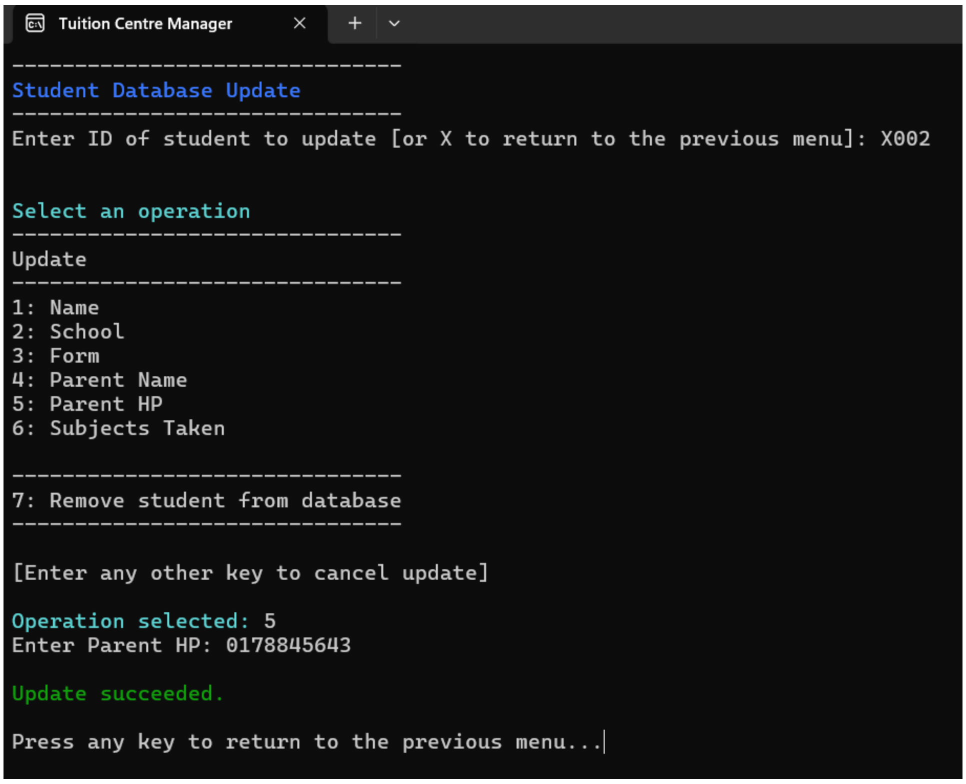
```
-------------------------------
Student Database Update
-------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
-------------------------------
Update
-------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

-------------------------------
7: Remove student from database
-------------------------------

[Enter any other key to cancel update]

Operation selected: 3
Enter Form: 6

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.39: Updating Student Grade



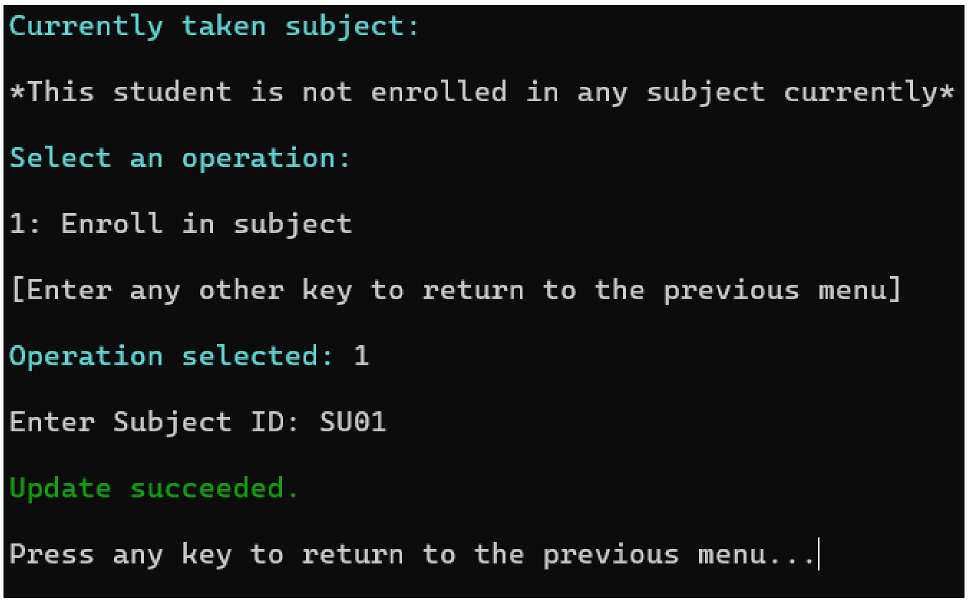
```
--------------------------------
Student Database Update
--------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
--------------------------------
Update
--------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

--------------------------------
7: Remove student from database
--------------------------------

[Enter any other key to cancel update]

Operation selected: 4
Enter Parent Name: Adamu

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.40: Updating Student Parent Name



```
--------------------------------
Student Database Update
--------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
--------------------------------
Update
--------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

--------------------------------
7: Remove student from database
--------------------------------

[Enter any other key to cancel update]

Operation selected: 5
Enter Parent HP: 0178845643

Update succeeded.

Press any key to return to the previous menu...
```

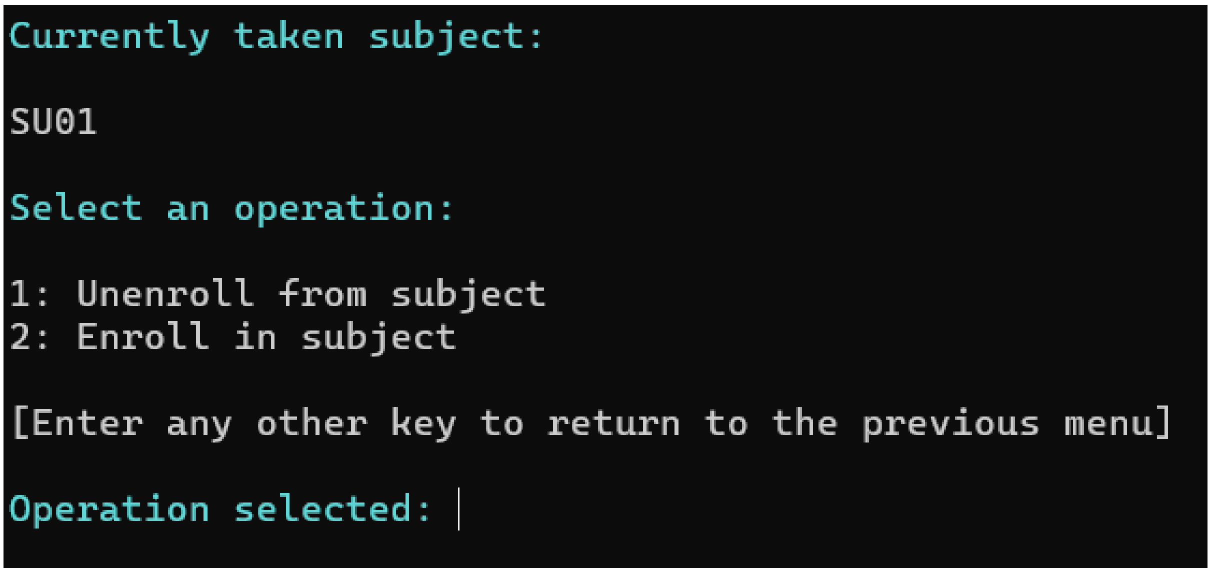
Figure 3.41: Updating Student Parent's HP



```
Currently taken subject:

*This student is not enrolled in any subject currently*

Select an operation:

1: Enroll in subject

[Enter any other key to return to the previous menu]

Operation selected: 1

Enter Subject ID: SU01

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.42: Updating Student’s Subject Enrollment

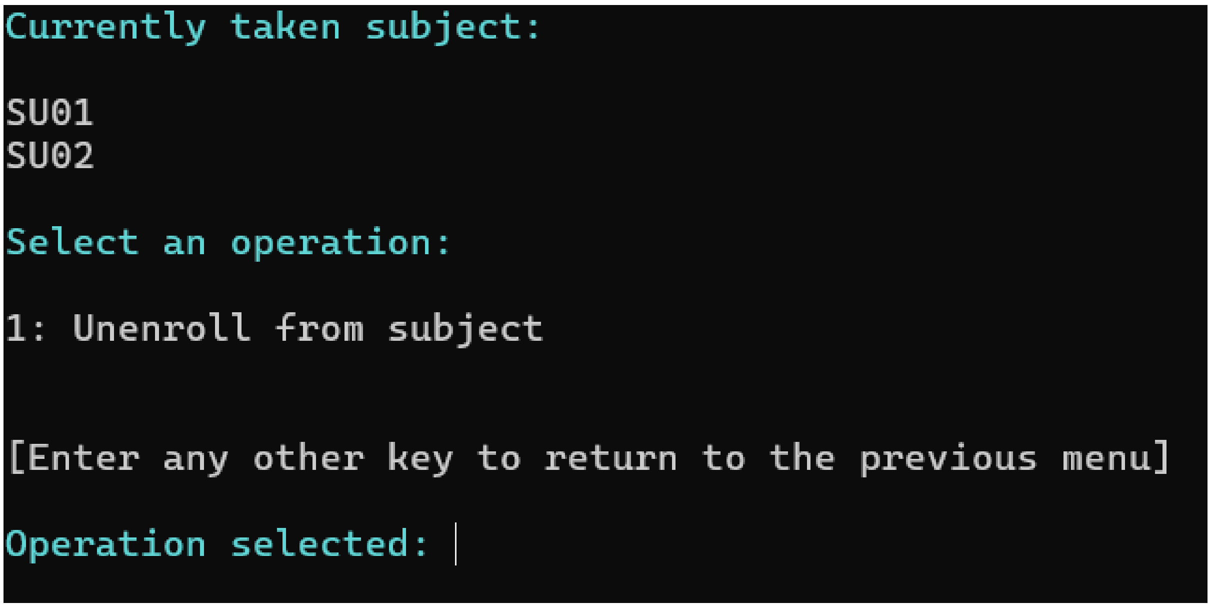
[Student Is Not Currently Enrolled in Any Subject]



```
Currently taken subject:

SU01

Select an operation:

1: Unenroll from subject
2: Enroll in subject

[Enter any other key to return to the previous menu]

Operation selected: 
```

Figure 3.43: Updating Student’s Subject Enrollment

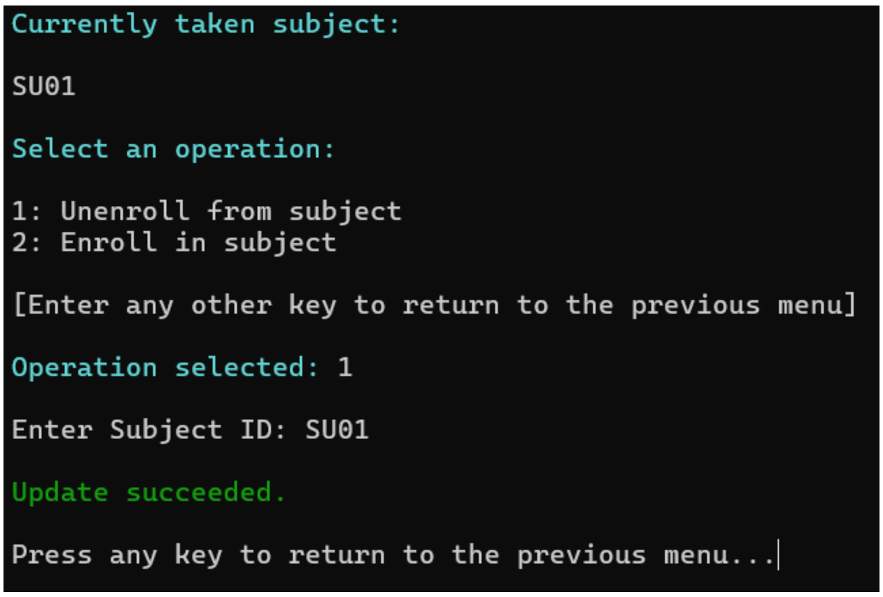
[Student Currently Enrolled in Subject(s)]



```
Currently taken subject:

SU01
SU02

Select an operation:

1: Unenroll from subject


[Enter any other key to return to the previous menu]

Operation selected:
```

Figure 3.44: Updating Student's Subject Enrollment

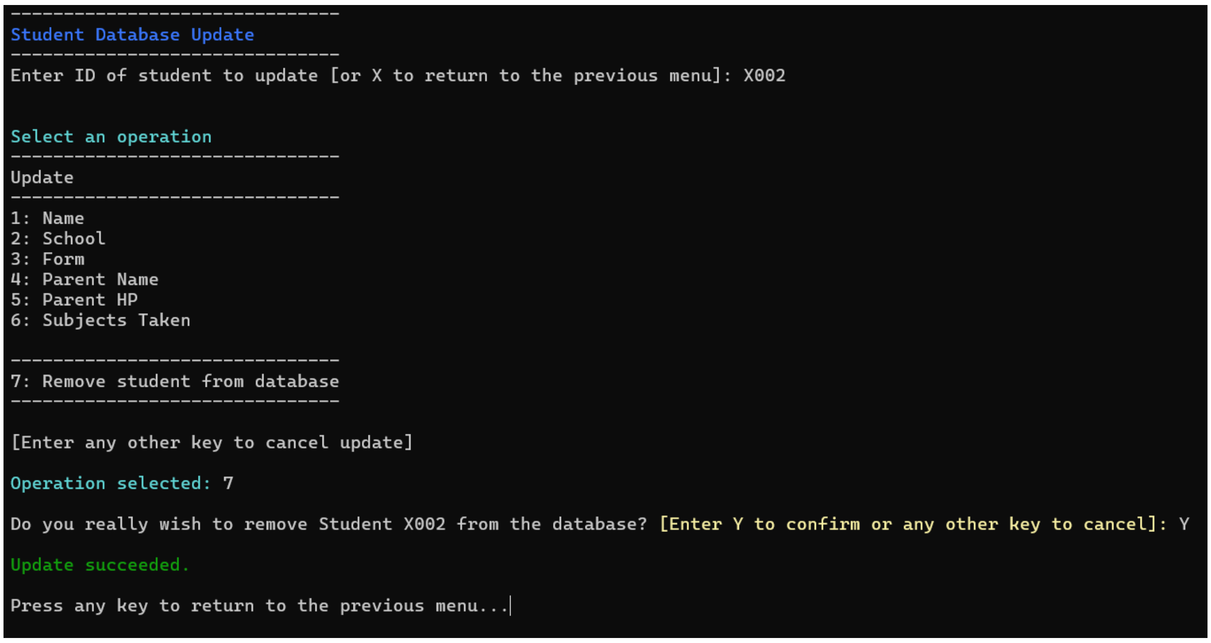
[Student Currently Enrolled in All Available Subjects]



```
Currently taken subject:

SU01

Select an operation:

1: Unenroll from subject
2: Enroll in subject

[Enter any other key to return to the previous menu]

Operation selected: 1

Enter Subject ID: SU01

Update succeeded.

Press any key to return to the previous menu...
```

Figure 3.45: Unenroll Student From Subject

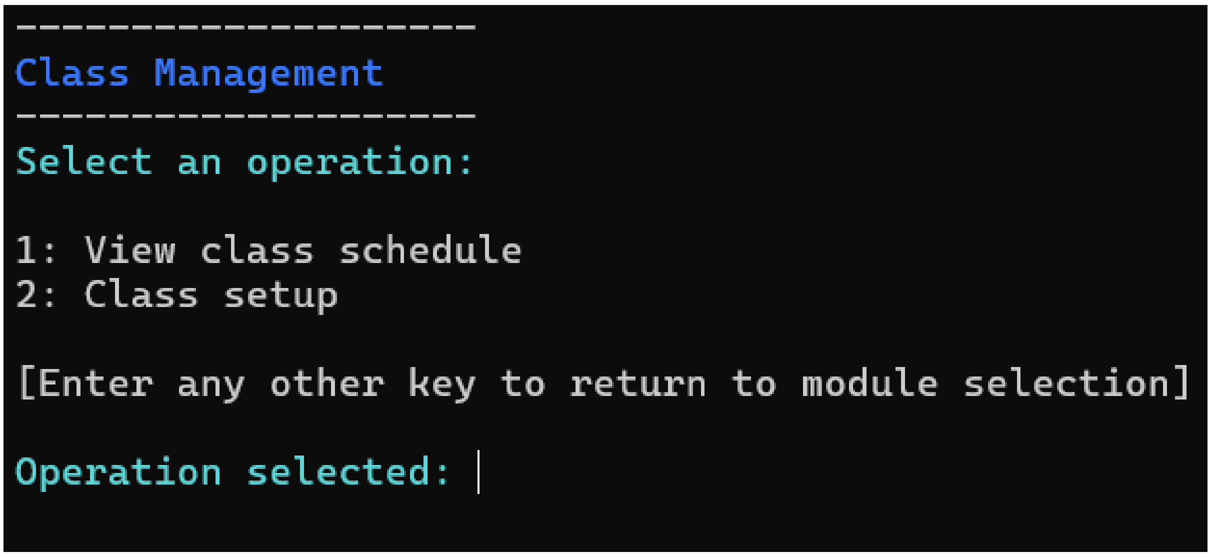
Figure 3.46 focuses on the removal of student from database, which requires admin confirmation to reduce the risk of wrongly executing this irreversible action.



```
-------------------------------
Student Database Update
-------------------------------
Enter ID of student to update [or X to return to the previous menu]: X002


Select an operation
-------------------------------
Update
-------------------------------
1: Name
2: School
3: Form
4: Parent Name
5: Parent HP
6: Subjects Taken

-------------------------------
7: Remove student from database
-------------------------------

[Enter any other key to cancel update]

Operation selected: 7

Do you really wish to remove Student X002 from the database? [Enter Y to confirm or any other key to cancel]: Y

Update succeeded.

Press any key to return to the previous menu...
```

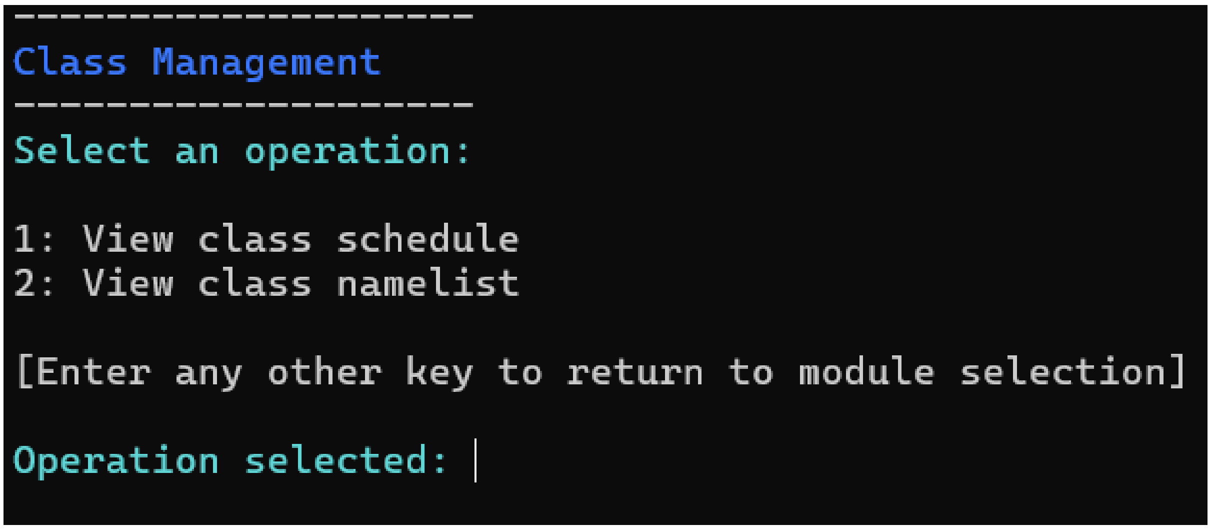
Figure 3.46: Removing Student From Database

Figures 3.47 – 3.56 illustrate the class management module operations. While both admins and staff can view the class schedule (Figure 3.49), only admins are allowed to process class set-ups (Figures 3.50 – 3.55), and only staff will be able to check the namelist of their assigned classes (Figure 3.56).



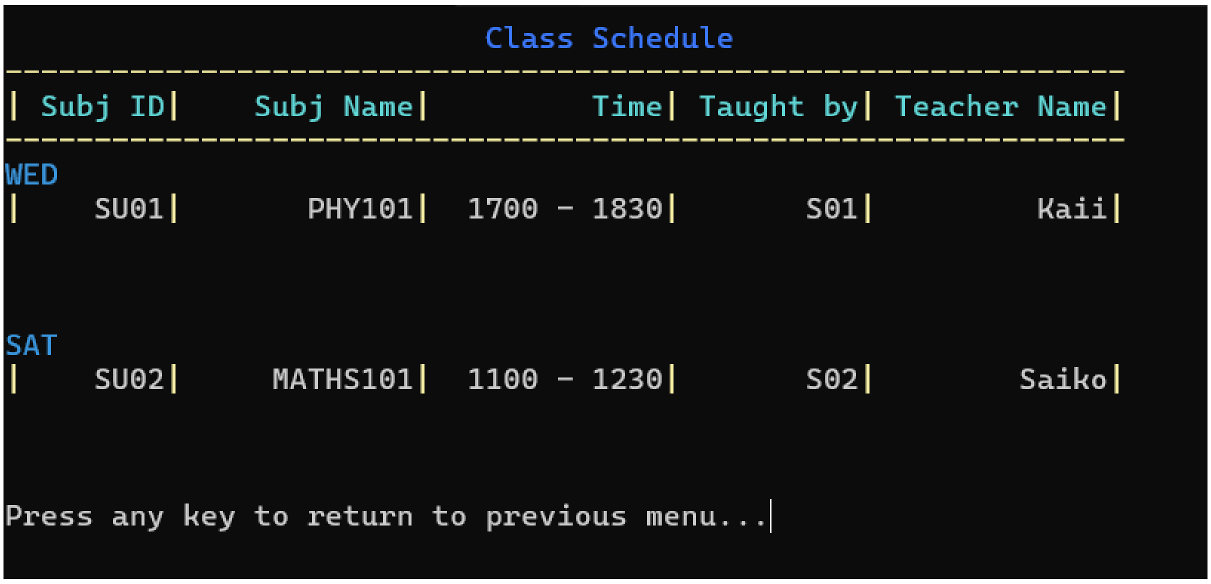
```
---------------------
Class Management
---------------------
Select an operation:

1: View class schedule
2: Class setup

[Enter any other key to return to module selection]

Operation selected:
```

Figure 3.47: Class Management Module [Admin Mode]



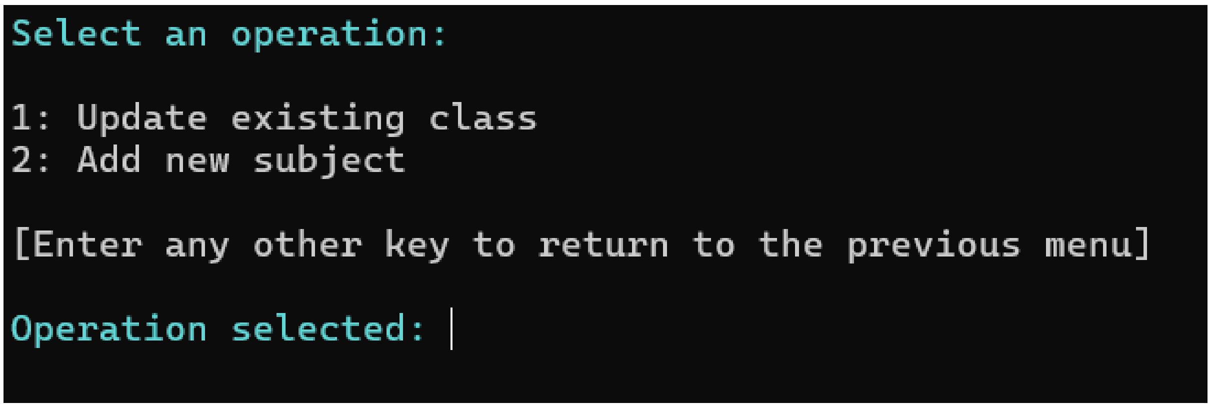
```
--------------------
Class Management
--------------------
Select an operation:

1: View class schedule
2: View class namelist

[Enter any other key to return to module selection]

Operation selected: 
```

Figure 3.48: Class Management Module [Staff Mode]



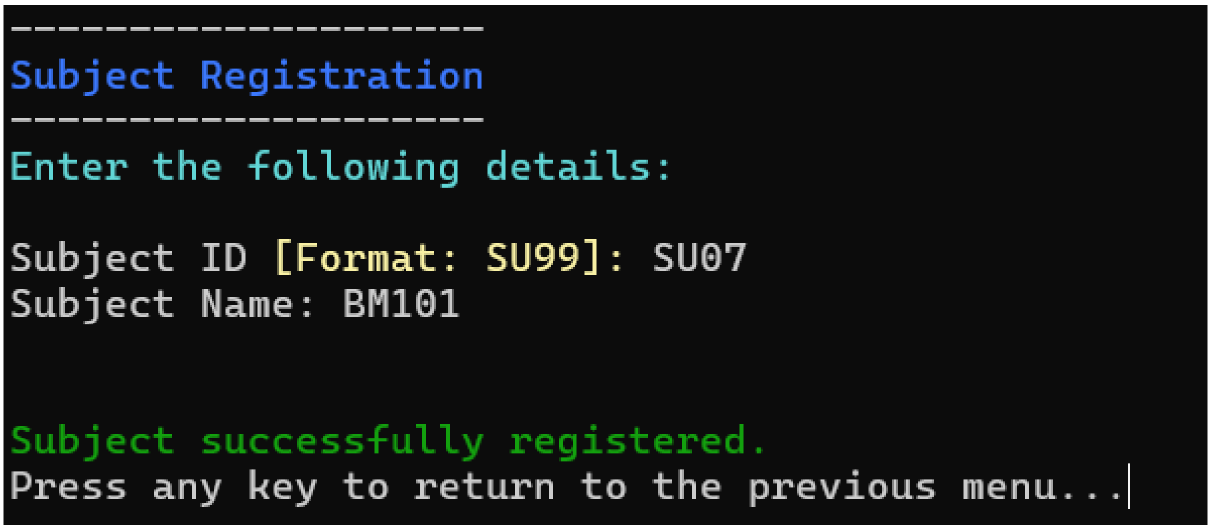
```
                    Class Schedule
---------------------------------------------------------------
| Subj ID|     Subj Name|          Time| Taught by| Teacher Name|
---------------------------------------------------------------
WED
|     SU01|        PHY101|   1700 - 1830|       S01|         Kaii|


SAT
|     SU02|      MATHS101|   1100 - 1230|       S02|        Saiko|


Press any key to return to previous menu...
```

Figure 3.49: Class Schedule Viewing



```
Select an operation:

1: Update existing class
2: Add new subject

[Enter any other key to return to the previous menu]

Operation selected: 
```

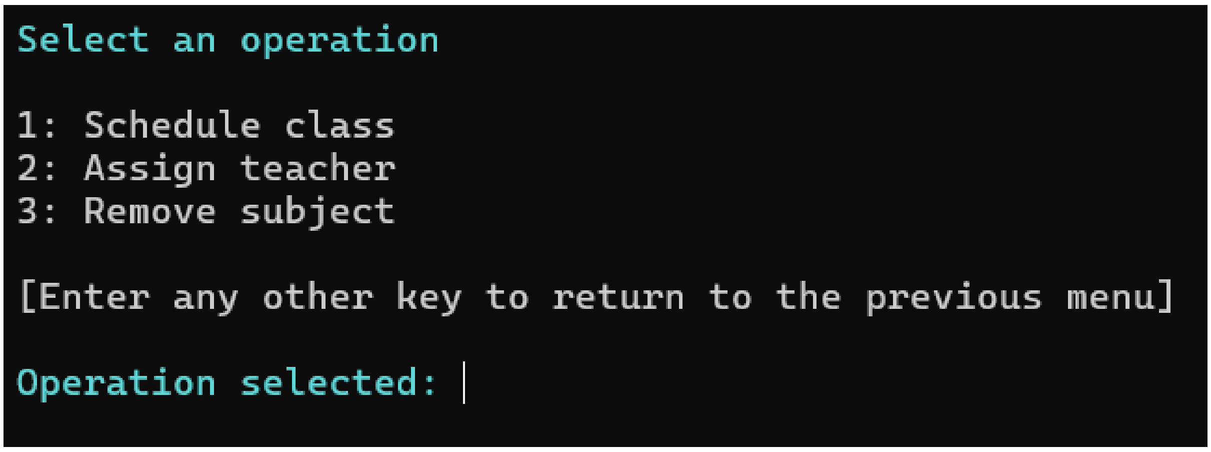
Figure 3.50: Class Setup Operations



```
--------------------
Subject Registration
--------------------
Enter the following details:

Subject ID [Format: SU99]: SU07
Subject Name: BM101


Subject successfully registered.
Press any key to return to the previous menu...
```

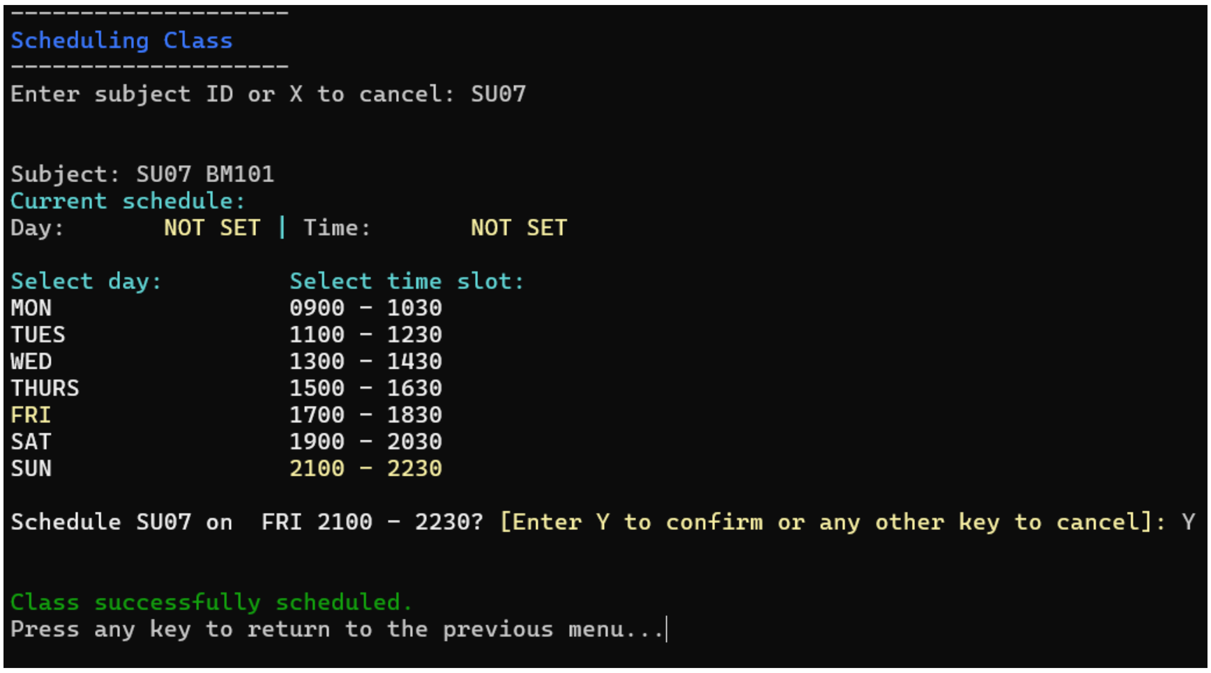
Figure 3.51: Subject Registration

Figures 3.52 – 3.54 focus on how, after registering a subject, an admin can schedule it and assign a teaching staff before the subject appears on the timetable. A subject can also be rescheduled, or have a teaching staff reassigned to it.



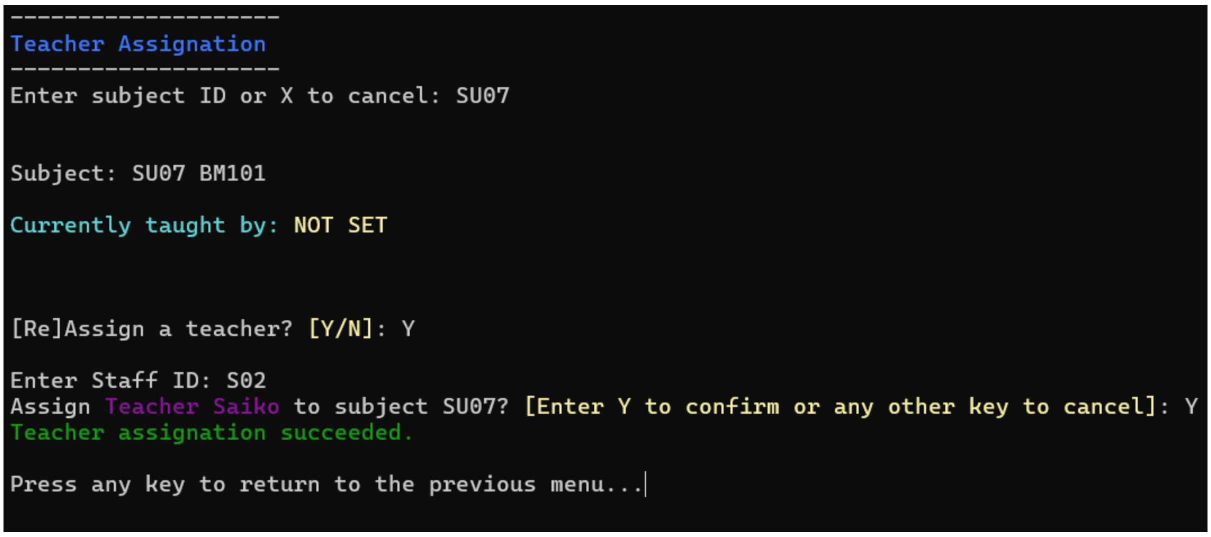
```
Select an operation

1: Schedule class
2: Assign teacher
3: Remove subject

[Enter any other key to return to the previous menu]

Operation selected:
```

Figure 3.52: Operation Selection for Updating Existing Classes



```
--------------------
Scheduling Class
--------------------
Enter subject ID or X to cancel: SU07


Subject: SU07 BM101
Current schedule:
Day:        NOT SET | Time:       NOT SET

Select day:         Select time slot:
MON                 0900 - 1030
TUES                1100 - 1230
WED                 1300 - 1430
THURS               1500 - 1630
FRI                 1700 - 1830
SAT                 1900 - 2030
SUN                 2100 - 2230

Schedule SU07 on  FRI 2100 - 2230? [Enter Y to confirm or any other key to cancel]: Y


Class successfully scheduled.
Press any key to return to the previous menu...
```

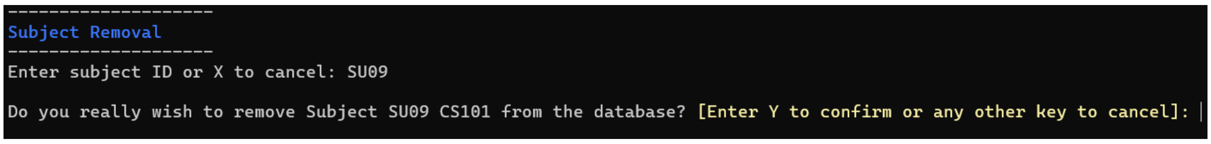
Figure 3.53: Class Scheduling



```
--------------------
Teacher Assignation
--------------------
Enter subject ID or X to cancel: SU07


Subject: SU07 BM101

Currently taught by: NOT SET



[Re]Assign a teacher? [Y/N]: Y

Enter Staff ID: S02
Assign Teacher Saiko to subject SU07? [Enter Y to confirm or any other key to cancel]: Y
Teacher assignation succeeded.

Press any key to return to the previous menu...
```

Figure 3.54: Teacher Assignation

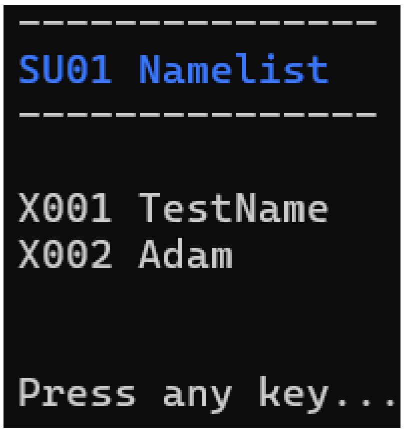
Figure 3.55 shows how an admin may remove an existing subject from database. A confirmation message will be prompted to prevent accidental removal.



```
--------------------
Subject Removal
--------------------
Enter subject ID or X to cancel: SU09

Do you really wish to remove Subject SU09 CS101 from the database? [Enter Y to confirm or any other key to cancel]:
```

Figure 3.55: Subject Removal from Database

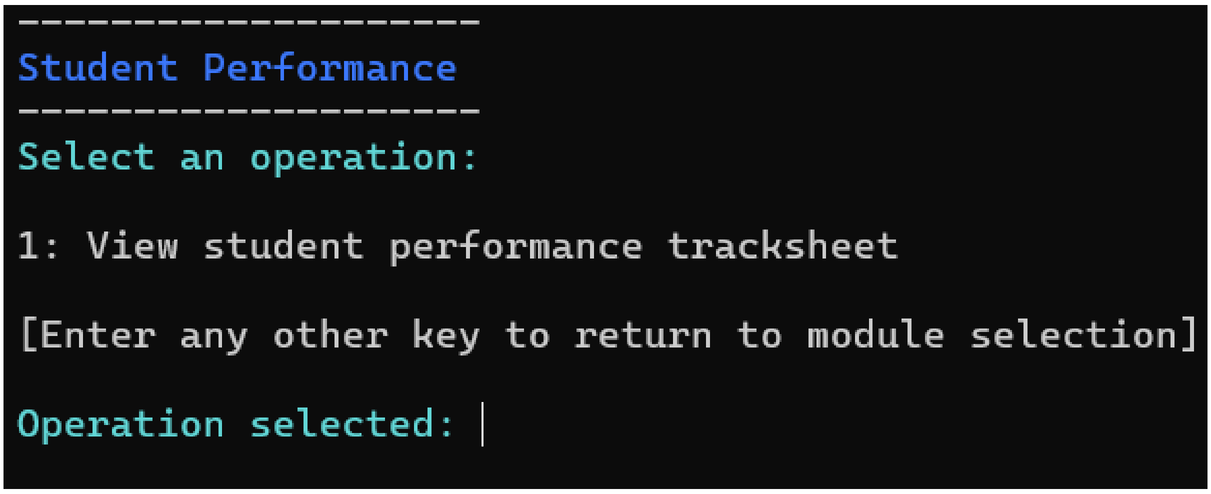
Figure 3.56 show the namelist that staff may view for their assigned subjects.



```
---------------
SU01 Namelist
---------------

X001 TestName
X002 Adam


Press any key...
```

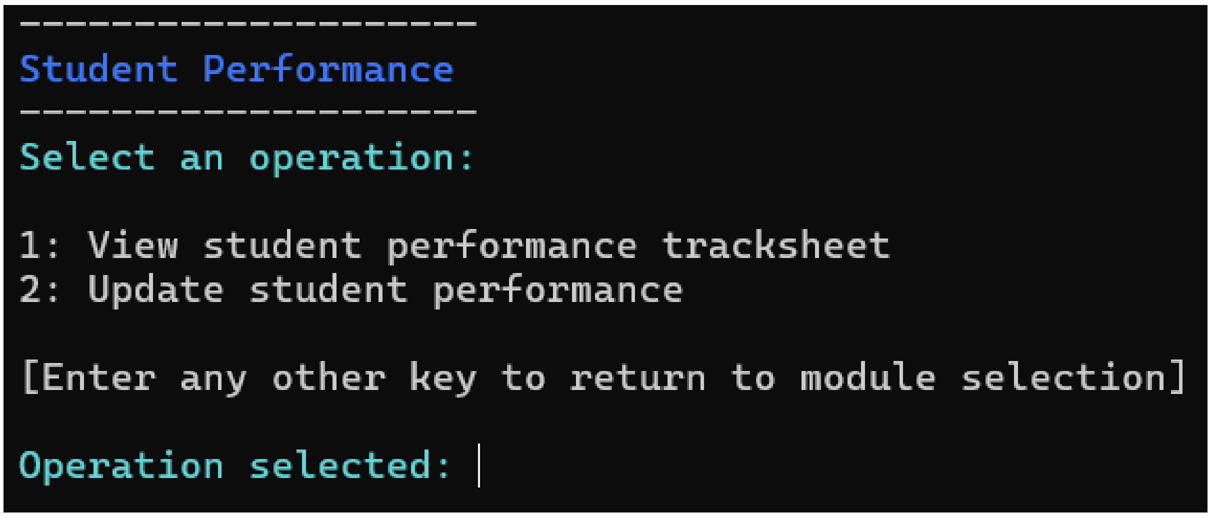
Figure 3.56: Subject Namelist Viewing

Figures 3.57 – 3.63 illustrate the student performance module operations. Only staff are allowed to update the students' performance parameters (quiz scores) (Figure 3.59 – 3.60), but both admin and staff are allowed to view the student performance tracksheet (Figure 3.61 – 3.63).



```
--------------------
Student Performance
--------------------
Select an operation:

1: View student performance tracksheet

[Enter any other key to return to module selection]

Operation selected: 
```

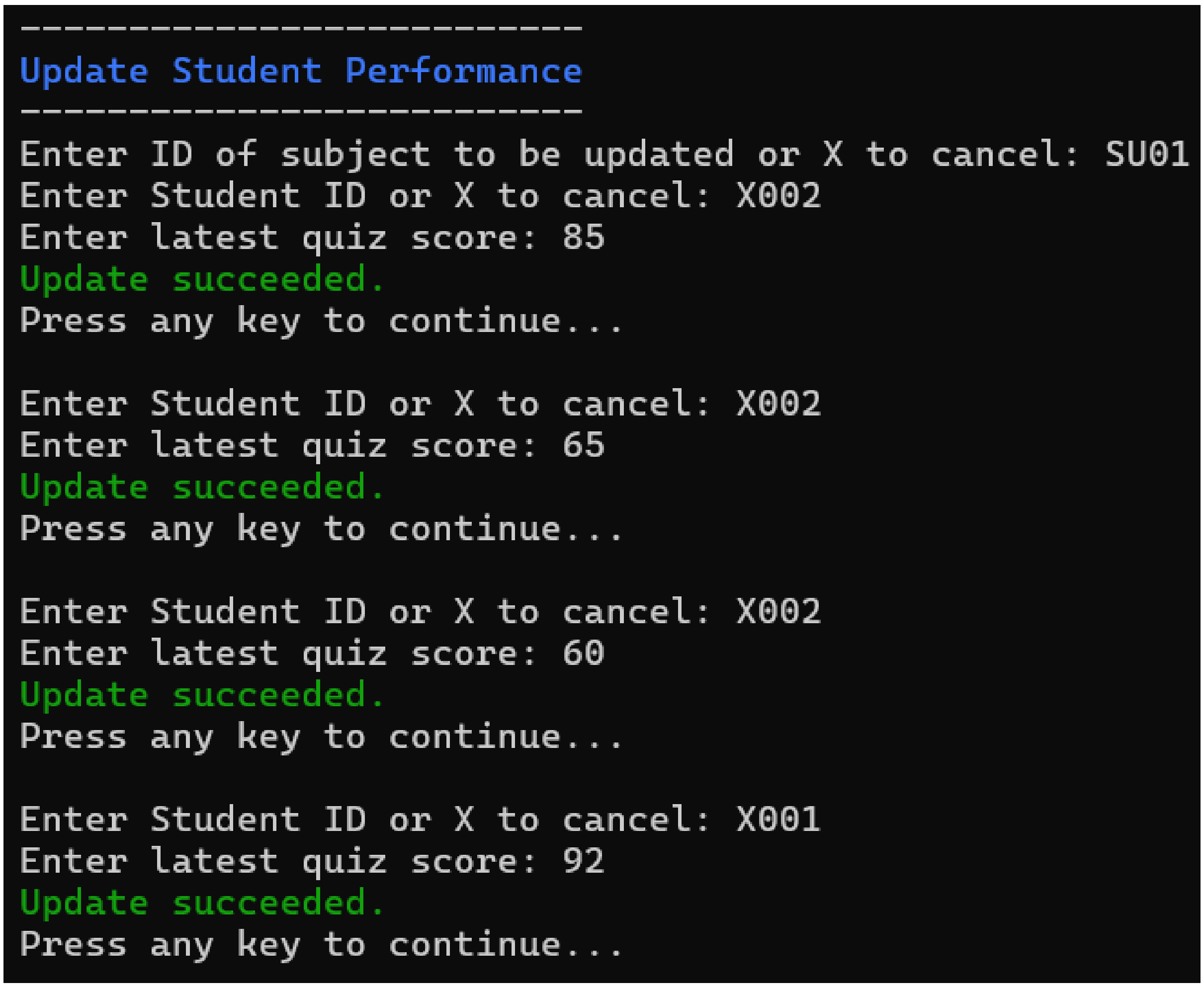
Figure 3.57: Student Performance Module [Admin Mode]



```
--------------------
Student Performance
--------------------
Select an operation:

1: View student performance tracksheet
2: Update student performance

[Enter any other key to return to module selection]

Operation selected: 
```

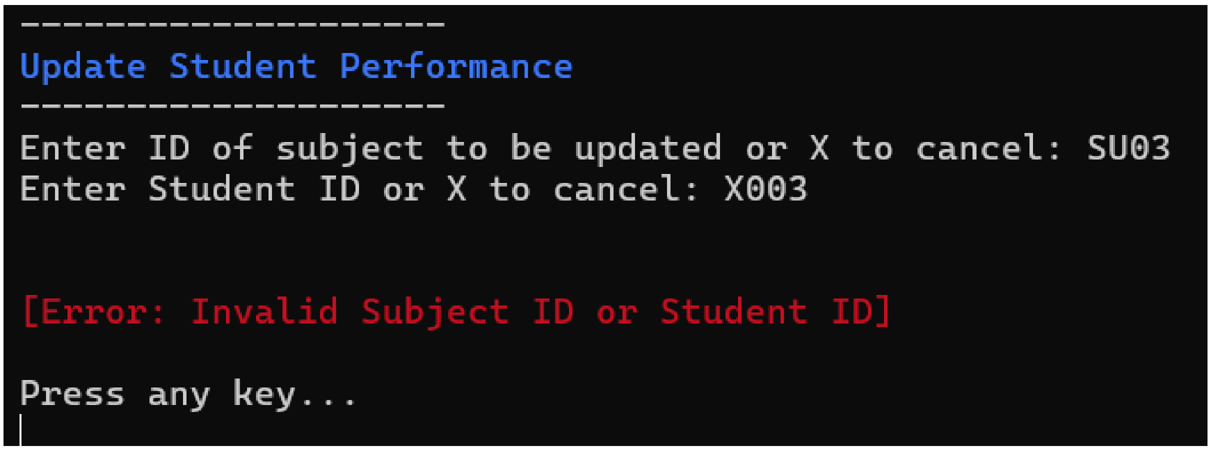
Figure 3.58: Student Performance Module [Staff Mode]

Figures 3.59 – 3.60 focus on the staff's process of updating a student's performance. The updated quiz scores will then be reflected in the student performance tracksheet.



```
-------------------------
Update Student Performance
-------------------------
Enter ID of subject to be updated or X to cancel: SU01
Enter Student ID or X to cancel: X002
Enter latest quiz score: 85
Update succeeded.
Press any key to continue...

Enter Student ID or X to cancel: X002
Enter latest quiz score: 65
Update succeeded.
Press any key to continue...

Enter Student ID or X to cancel: X002
Enter latest quiz score: 60
Update succeeded.
Press any key to continue...

Enter Student ID or X to cancel: X001
Enter latest quiz score: 92
Update succeeded.
Press any key to continue...
```

Figure 3.59: Updating Student Performance

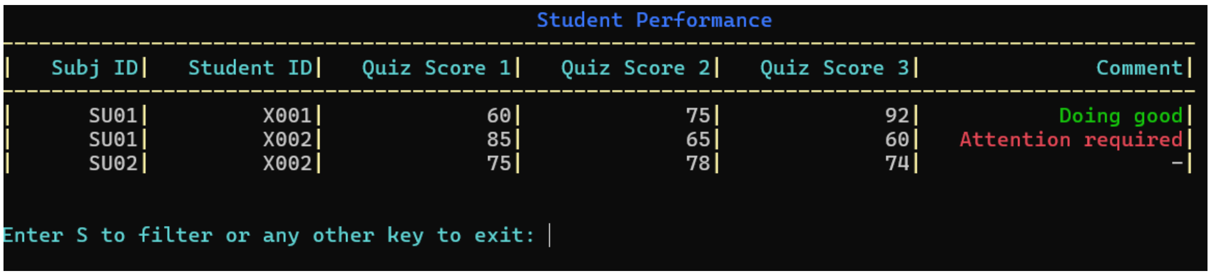


```
----------------------
Update Student Performance
----------------------
Enter ID of subject to be updated or X to cancel: SU03
Enter Student ID or X to cancel: X003


[Error: Invalid Subject ID or Student ID]

Press any key...
```

Figure 3.60: Updating Student Performance

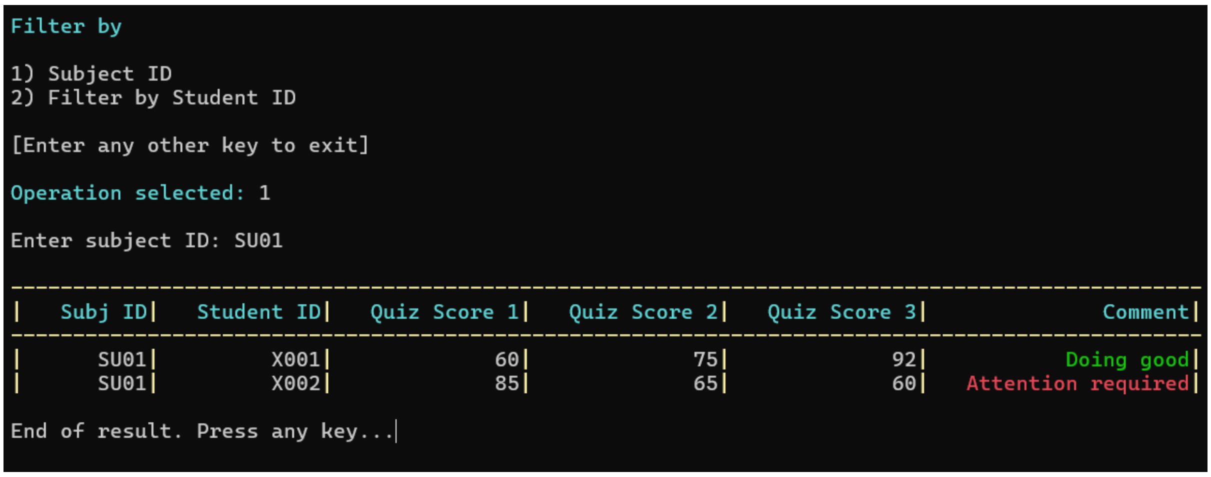
(Attempting to Update Scores for Subject Not Under Staff / Student Not Taking Subject)

Figures 3.61 – 3.63 focus on student performance tracksheet viewing. Users may view the tracksheet as it is, or filter students' performance by SubjectID or StudentID according to their needs.



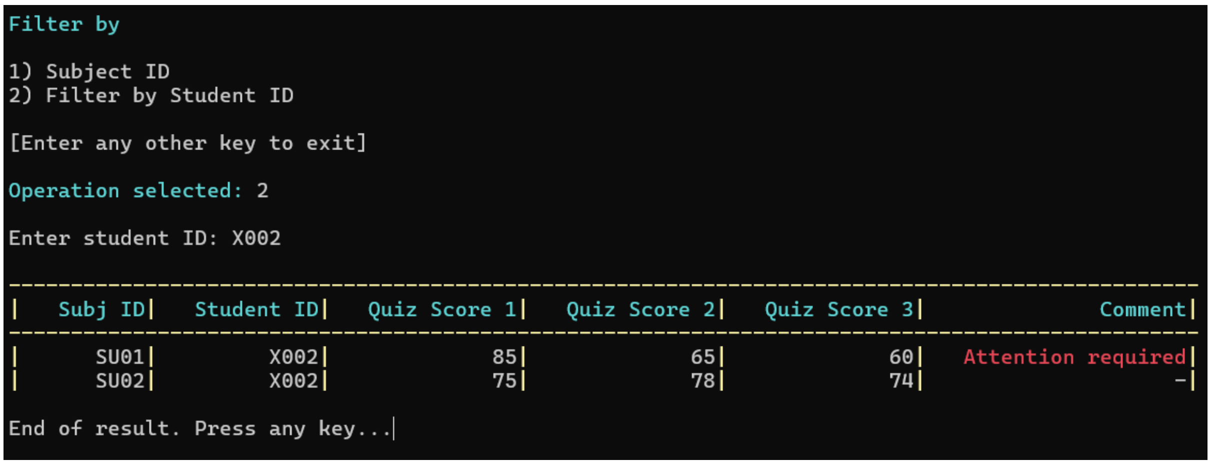
```
                                          Student Performance
-----------------------------------------------------------------------------------------------
|   Subj ID|    Student ID|     Quiz Score 1|     Quiz Score 2|     Quiz Score 3|          Comment|
-----------------------------------------------------------------------------------------------
|      SU01|          X001|               60|               75|               92|       Doing good|
|      SU01|          X002|               85|               65|               60|Attention required|
|      SU02|          X002|               75|               78|               74|                -|


Enter S to filter or any other key to exit:
```

Figure 3.61: Student Performance Viewing



```
Filter by

1) Subject ID
2) Filter by Student ID

[Enter any other key to exit]

Operation selected: 1

Enter subject ID: SU01

-----------------------------------------------------------------------------------------------
|   Subj ID|    Student ID|     Quiz Score 1|     Quiz Score 2|     Quiz Score 3|          Comment|
-----------------------------------------------------------------------------------------------
|      SU01|          X001|               60|               75|               92|       Doing good|
|      SU01|          X002|               85|               65|               60|Attention required|

End of result. Press any key...
```

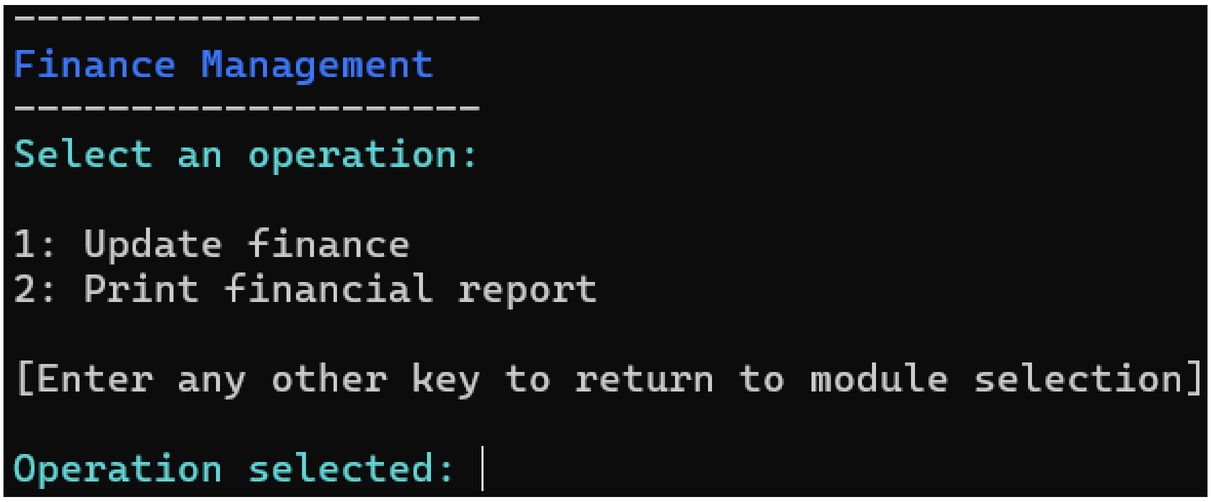
Figure 3.62: Filtering Student Performance by Subject ID



```
Filter by

1) Subject ID
2) Filter by Student ID

[Enter any other key to exit]

Operation selected: 2

Enter student ID: X002

-----------------------------------------------------------------------------------------------
|    Subj ID|    Student ID|    Quiz Score 1|    Quiz Score 2|    Quiz Score 3|          Comment|
-----------------------------------------------------------------------------------------------
|       SU01|          X002|              85|              65|              60|  Attention required|
|       SU02|          X002|              75|              78|              74|                -|

End of result. Press any key...
```

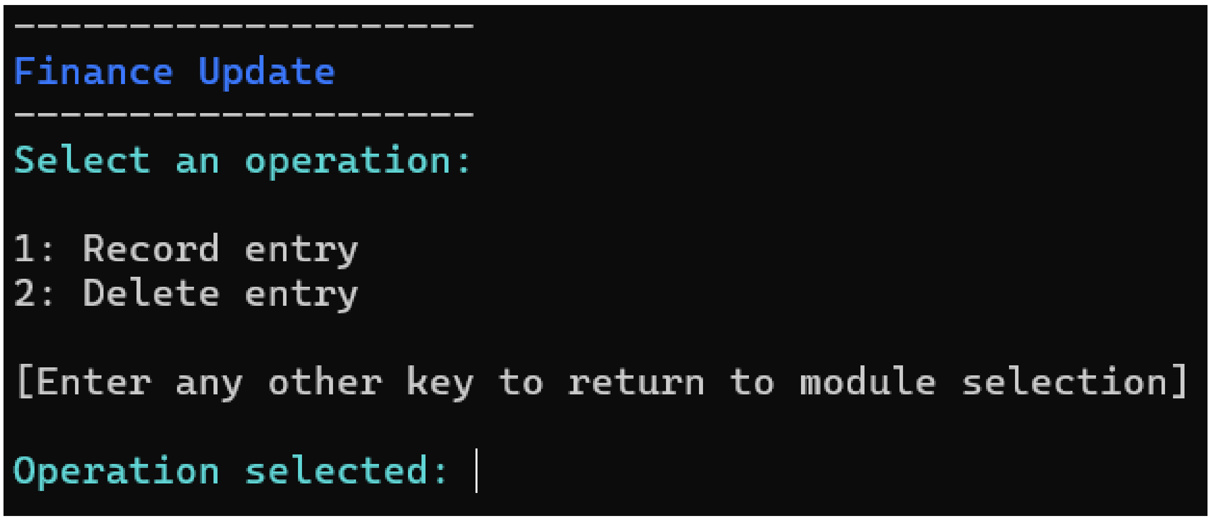
Figure 3.63: Filtering Student Performance by Student ID

Figures 3.64 - 68 illustrate the finance tracking module operations, which are only accessible by admins. Admins may update finance entries (Figures 3.65 – 3.67) or view financial report (Figures 3.68 – 3.69).



```
--------------------
Finance Management
--------------------
Select an operation:

1: Update finance
2: Print financial report

[Enter any other key to return to module selection]

Operation selected: 
```

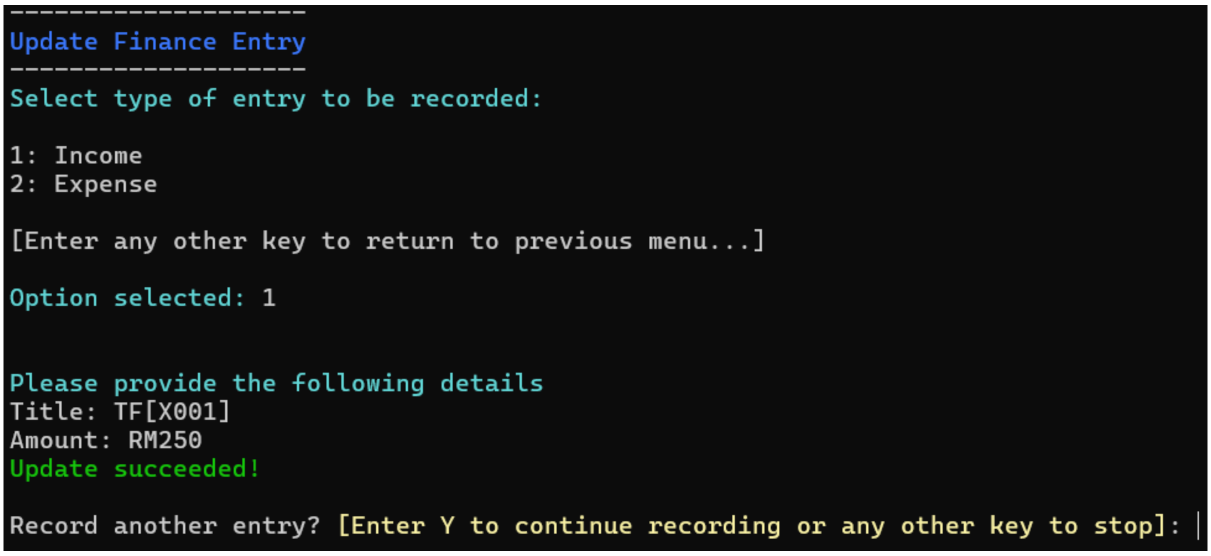
Figure 3.64: Finance Management Module



```
---------------------
Finance Update
---------------------
Select an operation:

1: Record entry
2: Delete entry

[Enter any other key to return to module selection]

Operation selected: 
```

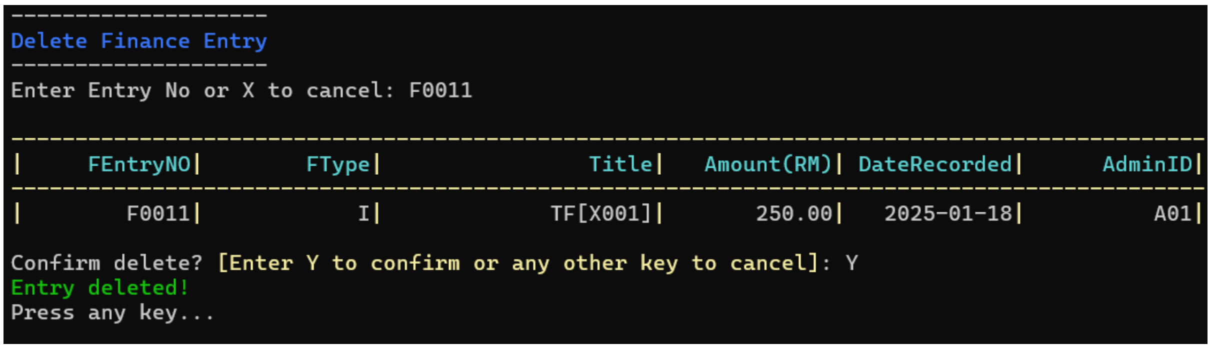
Figure 3.65: Finance Update Options

Figures 3.66 – 3.67 focus on the process of updating finance entries. Admins may record an income or expense entry, as well as delete them (a confirmation message will be prompted).



```
--------------------
Update Finance Entry
--------------------
Select type of entry to be recorded:

1: Income
2: Expense

[Enter any other key to return to previous menu...]

Option selected: 1


Please provide the following details
Title: TF[X001]
Amount: RM250
Update succeeded!

Record another entry? [Enter Y to continue recording or any other key to stop]: 
```

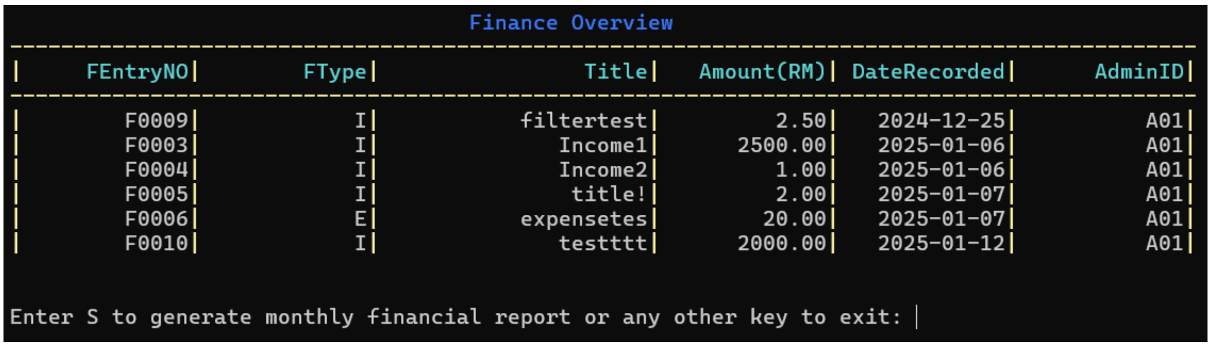
Figure 3.66: Recording Finance Entry



```
--------------------
Delete Finance Entry
--------------------
Enter Entry No or X to cancel: F0011

---------------------------------------------------------------------------------------------
|     FEntryNO|         FType|                 Title|  Amount(RM)| DateRecorded|       AdminID|
---------------------------------------------------------------------------------------------
|        F0011|             I|              TF[X001]|      250.00|   2025-01-18|           A01|

Confirm delete? [Enter Y to confirm or any other key to cancel]: Y
Entry deleted!
Press any key...
```

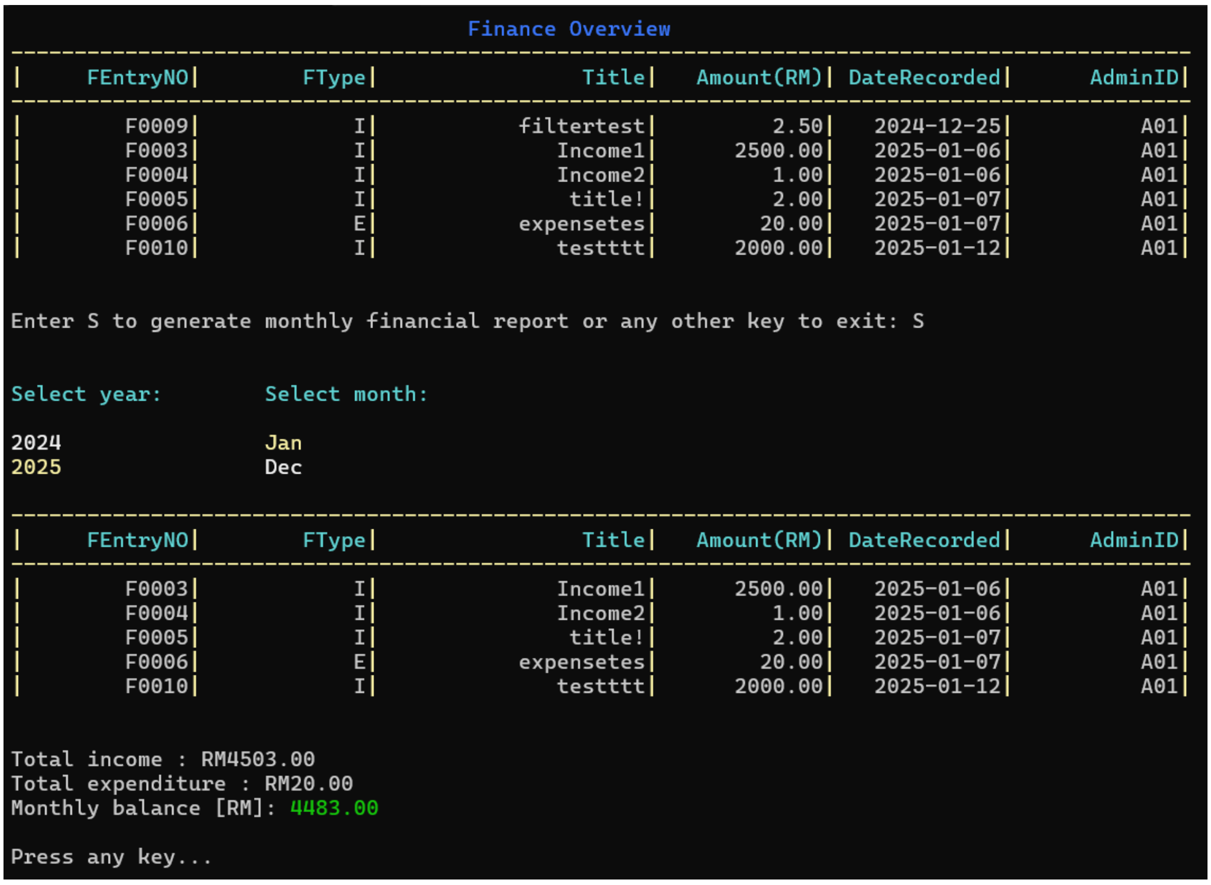
Figure 3.67: Deleting Finance Entry

Figures 3.68 – 3.69 focus on financial report viewing. Admins have access to the finance overview which displays the details of every single finane entry, and are also allowed to filter the report by month. Doing the latter will also prompt the system to provide a brief analysis for the monthly balance of the selected month.



```
                                    Finance Overview
----------------------------------------------------------------------------------------------
|     FEntryNO|         FType|              Title|   Amount(RM)| DateRecorded|       AdminID|
----------------------------------------------------------------------------------------------
|        F0009|             I|         filtertest|         2.50|   2024-12-25|           A01|
|        F0003|             I|            Income1|      2500.00|   2025-01-06|           A01|
|        F0004|             I|            Income2|         1.00|   2025-01-06|           A01|
|        F0005|             I|             title!|         2.00|   2025-01-07|           A01|
|        F0006|             E|         expensetes|        20.00|   2025-01-07|           A01|
|        F0010|             I|            testttt|      2000.00|   2025-01-12|           A01|


Enter S to generate monthly financial report or any other key to exit: |
```

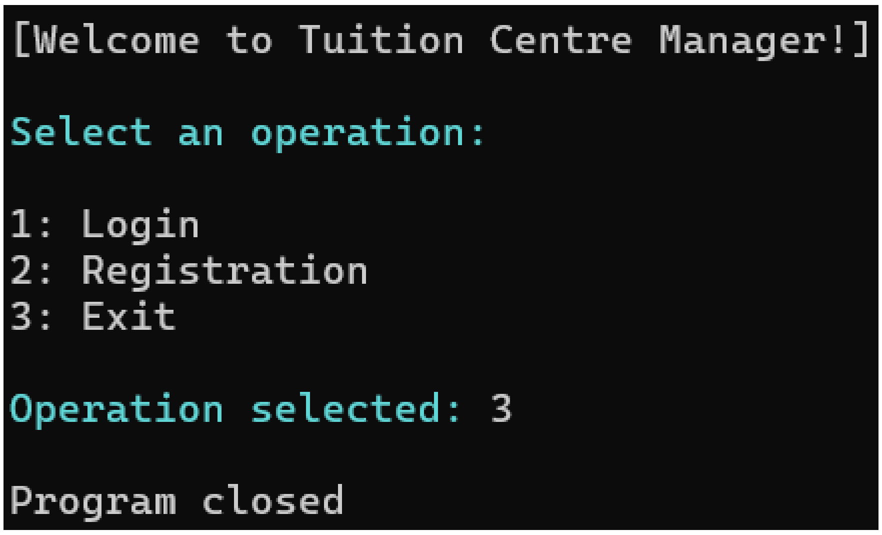
Figute 3.68: Finance Overview



```
                                    Finance Overview
----------------------------------------------------------------------------------------------
|     FEntryNO|         FType|              Title|   Amount(RM)| DateRecorded|       AdminID|
----------------------------------------------------------------------------------------------
|        F0009|             I|         filtertest|         2.50|   2024-12-25|           A01|
|        F0003|             I|            Income1|      2500.00|   2025-01-06|           A01|
|        F0004|             I|            Income2|         1.00|   2025-01-06|           A01|
|        F0005|             I|             title!|         2.00|   2025-01-07|           A01|
|        F0006|             E|         expensetes|        20.00|   2025-01-07|           A01|
|        F0010|             I|            testttt|      2000.00|   2025-01-12|           A01|


Enter S to generate monthly financial report or any other key to exit: S


Select year:         Select month:

2024                 Jan
2025                 Dec

----------------------------------------------------------------------------------------------
|     FEntryNO|         FType|              Title|   Amount(RM)| DateRecorded|       AdminID|
----------------------------------------------------------------------------------------------
|        F0003|             I|            Income1|      2500.00|   2025-01-06|           A01|
|        F0004|             I|            Income2|         1.00|   2025-01-06|           A01|
|        F0005|             I|             title!|         2.00|   2025-01-07|           A01|
|        F0006|             E|         expensetes|        20.00|   2025-01-07|           A01|
|        F0010|             I|            testttt|      2000.00|   2025-01-12|           A01|


Total income : RM4503.00
Total expenditure : RM20.00
Monthly balance [RM]: 4483.00

Press any key...
```

Figure 3.69: Monthly Finance Report Generation

Figure 3.70 shows how users may exit the program once they are done with their operations.



```
[Welcome to Tuition Centre Manager!]

Select an operation:

1: Login
2: Registration
3: Exit

Operation selected: 3

Program closed
```

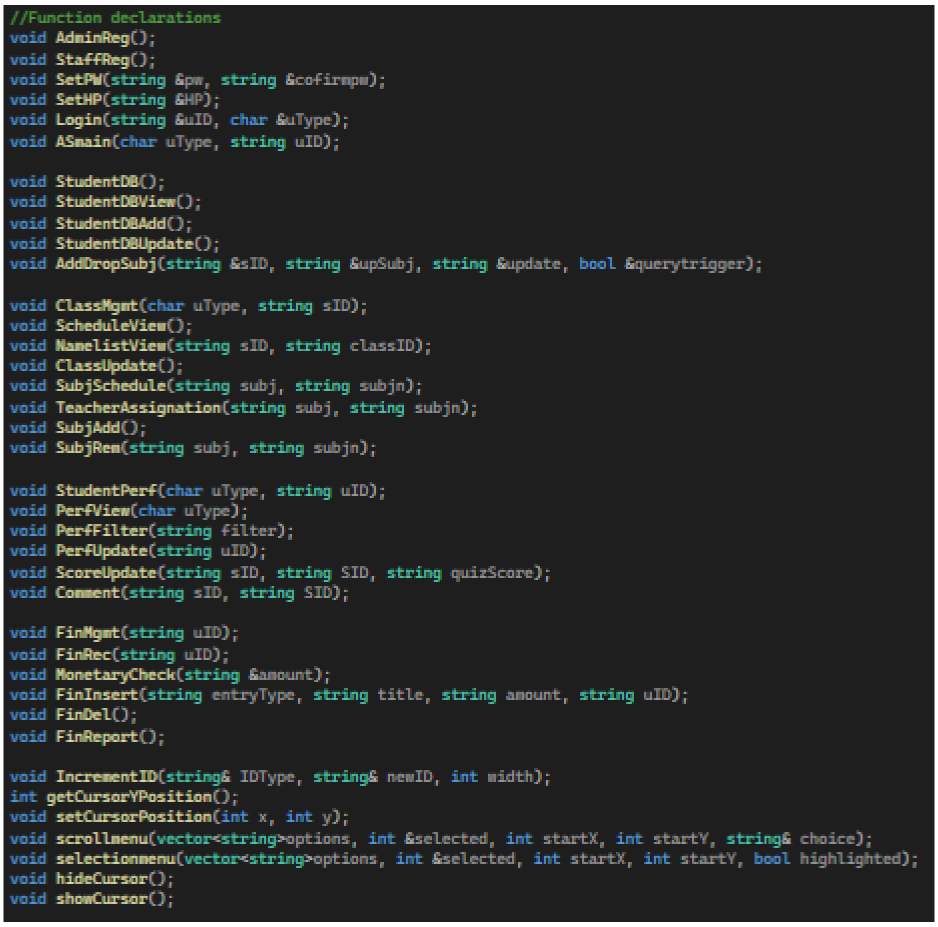
Figure 3.70: Closing Program

CHAPTER 4: IMPLEMENTATION

4.1 Function

Tuition Centre Manager employs a vast array of functions to implement the modules (Student Database, Class Management, Student Performance, Finance Management) in a systematic structure, as well as miscellaneous features to enhance the system user's experience.

Figure 4.1 shows the declaration for every function employed in the system.



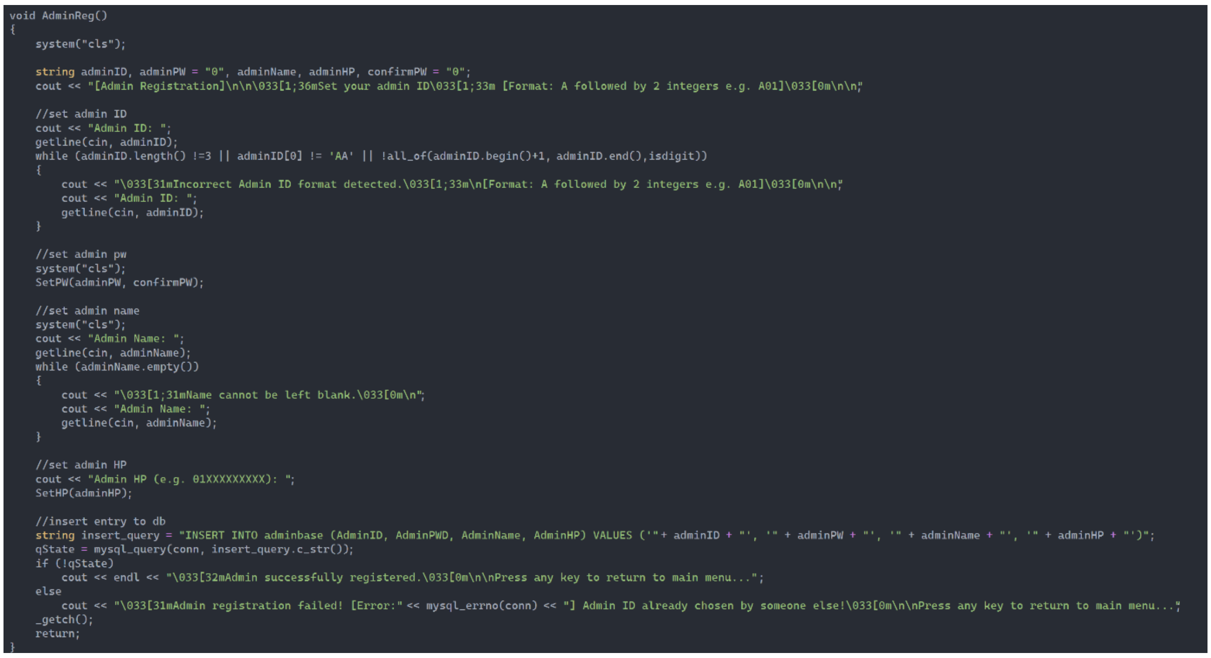
```
//Function declarations
void AdminReg();
void StaffReg();
void SetPW(string &pw, string &cofirmpw);
void SetHP(string &HP);
void Login(string &uID, char &uType);
void ASmain(char uType, string uID);

void StudentDB();
void StudentDBView();
void StudentDBAdd();
void StudentDBUpdate();
void AddDropSubj(string &sID, string &upSubj, string &update, bool &querytrigger);

void ClassMgmt(char uType, string sID);
void ScheduleView();
void NamelistView(string sID, string classID);
void ClassUpdate();
void SubjSchedule(string subj, string subjn);
void TeacherAssignation(string subj, string subjn);
void SubjAdd();
void SubjRem(string subj, string subjn);

void StudentPerf(char uType, string uID);
void PerfView(char uType);
void PerfFilter(string filter);
void PerfUpdate(string uID);
void ScoreUpdate(string sID, string SID, string quizScore);
void Comment(string sID, string SID);

void FinMgmt(string uID);
void FinRec(string uID);
void MonetaryCheck(string &amount);
void FinInsert(string entryType, string title, string amount, string uID);
void FinDel();
void FinReport();

void IncrementID(string& IDType, string& newID, int width);
int getCursorYPosition();
void setCursorPosition(int x, int y);
void scrollmenu(vector<string>options, int &selected, int startX, int startY, string& choice);
void selectionmenu(vector<string>options, int &selected, int startX, int startY, bool highlighted);
void hideCursor();
void showCursor();
```

Figure 4.1 Function Declarations

Inserting Record & Validation

These functions take user input and perform proper varification on data validity before adding them to the database.

Figure 4.2 shows the Admin Registration function, which acquires acquires user input to initialize AdminID, AdminPWD (password), AdminName, as well as AdminHP.



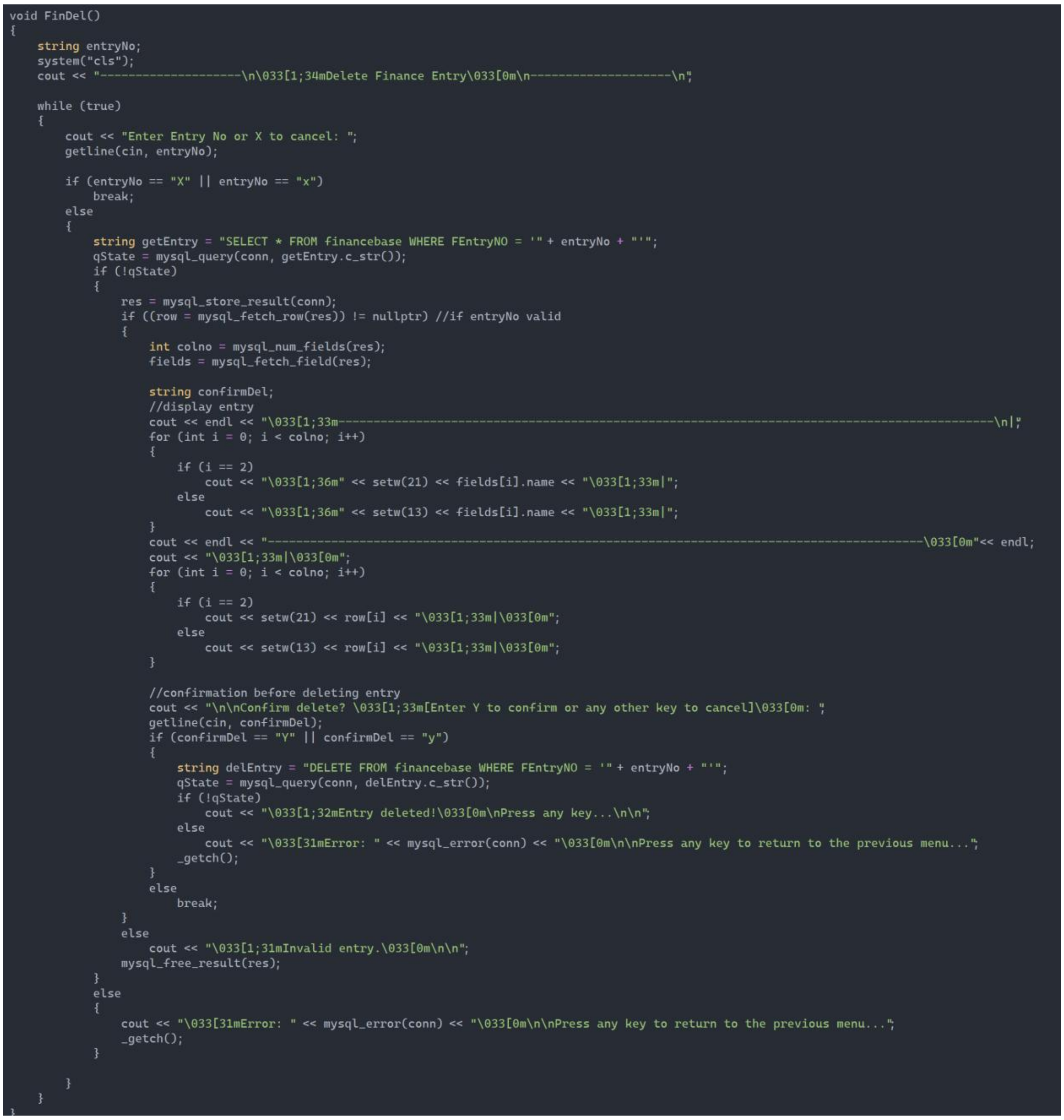
```cpp
void AdminReg()
{
    system("cls");

    string adminID, adminPW = "0", adminName, adminHP, confirmPW = "0";
    cout << "[Admin Registration]\n\n\033[1;36mSet your admin ID\033[1;33m [Format: A followed by 2 integers e.g. A01]\033[0m\n\n";

    //set admin ID
    cout << "Admin ID: ";
    getline(cin, adminID);
    while (adminID.length() !=3 || adminID[0] != 'AA' || !all_of(adminID.begin()+1, adminID.end(),isdigit))
    {
        cout << "\033[31mIncorrect Admin ID format detected.\033[1;33m\n[Format: A followed by 2 integers e.g. A01]\033[0m\n\n";
        cout << "Admin ID: ";
        getline(cin, adminID);
    }

    //set admin pw
    system("cls");
    SetPW(adminPW, confirmPW);

    //set admin name
    system("cls");
    cout << "Admin Name: ";
    getline(cin, adminName);
    while (adminName.empty())
    {
        cout << "\033[1;31mName cannot be left blank.\033[0m\n";
        cout << "Admin Name: ";
        getline(cin, adminName);
    }

    //set admin HP
    cout << "Admin HP (e.g. 01XXXXXXXXX): ";
    SetHP(adminHP);

    //insert entry to db
    string insert_query = "INSERT INTO adminbase (AdminID, AdminPWD, AdminName, AdminHP) VALUES ('"+ adminID + "', '" + adminPW + "', '" + adminName + "', '" + adminHP + "')";
    qState = mysql_query(conn, insert_query.c_str());
    if (!qState)
        cout << endl << "\033[32mAdmin successfully registered.\033[0m\n\nPress any key to return to main menu...";
    else
        cout << "\033[31mAdmin registration failed! [Error:" << mysql_errno(conn) << "] Admin ID already chosen by someone else!\033[0m\n\nPress any key to return to main menu...";
    _getch();
    return;
}
```

Figure 4.2 Admin Registration Function

Deleting Records

These functions remove user-desired entries from the database, provided that such entries had been properly initialized within the database prior to deletion.

Figure 4.3 shows the FinDel function which checks if the finance entry to be removed had previously been established in the database, then displays the finance entry for user checking and asks for reconfirmation, before removing the entry from database.



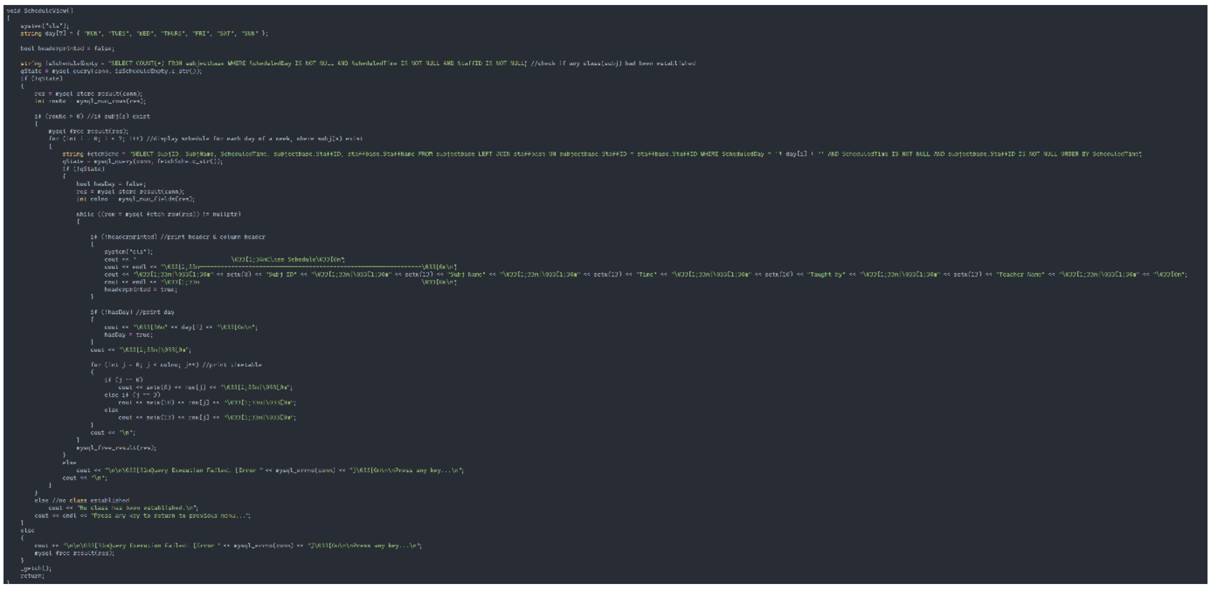
```cpp
void FinDel()
{
    string entryNo;
    system("cls");
    cout << "-------------------\n\033[1;34mDelete Finance Entry\033[0m\n-------------------\n";

    while (true)
    {
        cout << "Enter Entry No or X to cancel: ";
        getline(cin, entryNo);

        if (entryNo == "X" || entryNo == "x")
            break;
        else
        {
            string getEntry = "SELECT * FROM financebase WHERE FEntryNO = '" + entryNo + "'";
            qState = mysql_query(conn, getEntry.c_str());
            if (!qState)
            {
                res = mysql_store_result(conn);
                if ((row = mysql_fetch_row(res)) != nullptr) //if entryNo valid
                {
                    int colno = mysql_num_fields(res);
                    fields = mysql_fetch_field(res);

                    string confirmDel;
                    //display entry
                    cout << endl << "\033[1;33m---------------------------------------------------------------------------------------------\n|";
                    for (int i = 0; i < colno; i++)
                    {
                        if (i == 2)
                            cout << "\033[1;36m" << setw(21) << fields[i].name << "\033[1;33m|";
                        else
                            cout << "\033[1;36m" << setw(13) << fields[i].name << "\033[1;33m|";
                    }
                    cout << endl << "---------------------------------------------------------------------------------------------\033[0m"<< endl;
                    cout << "\033[1;33m|\033[0m";
                    for (int i = 0; i < colno; i++)
                    {
                        if (i == 2)
                            cout << setw(21) << row[i] << "\033[1;33m|\033[0m";
                        else
                            cout << setw(13) << row[i] << "\033[1;33m|\033[0m";
                    }

                    //confirmation before deleting entry
                    cout << "\n\nConfirm delete? \033[1;33m[Enter Y to confirm or any other key to cancel]\033[0m: ";
                    getline(cin, confirmDel);
                    if (confirmDel == "Y" || confirmDel == "y")
                    {
                        string delEntry = "DELETE FROM financebase WHERE FEntryNO = '" + entryNo + "'";
                        qState = mysql_query(conn, delEntry.c_str());
                        if (!qState)
                            cout << "\033[1;32mEntry deleted!\033[0m\nPress any key...\n\n";
                        else
                            cout << "\033[31mError: " << mysql_error(conn) << "\033[0m\n\nPress any key to return to the previous menu...";
                        _getch();
                    }
                    else
                        break;
                }
                else
                    cout << "\033[1;31mInvalid entry.\033[0m\n\n";
                mysql_free_result(res);
            }
            else
            {
                cout << "\033[31mError: " << mysql_error(conn) << "\033[0m\n\nPress any key to return to the previous menu...";
                _getch();
            }

        }
    }
}
```

Figure 4.3 FinDel Function

Viewing Records

These functions check if any entry exists for the database to be viewed and if so, display desired details in a tabulated manner to facilitate efficient checking.

Figure 4.4 shows the ScheduleView Function `which` checks if any class had been established in the database and provide outputs accordingly. (A message indicating no class exists within database, or a table detailing each established class's schedule and assigned teaching staff)



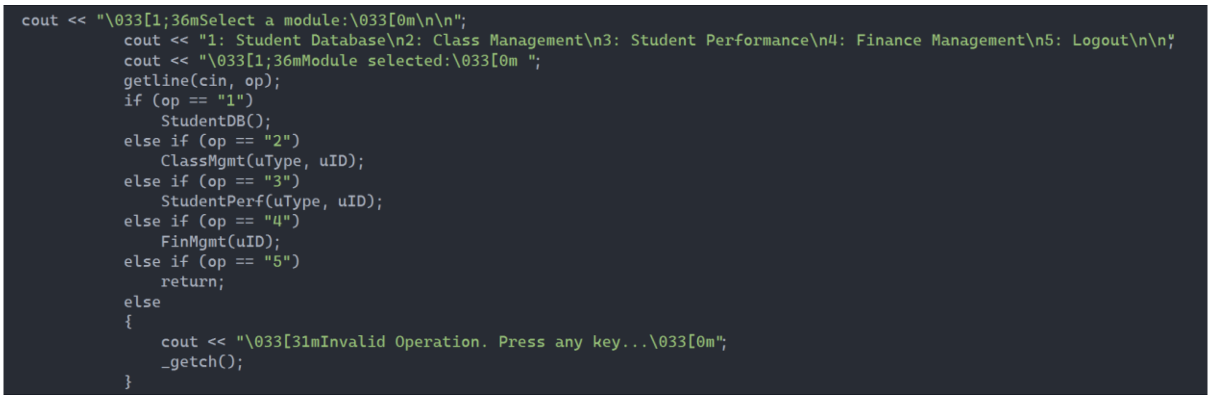
```
void ScheduleView()
{
    system("cls");
    string day[7] = { "MON", "TUES", "WED", "THURS", "FRI", "SAT", "SUN" };

    bool headerprinted = false;

    string isScheduleEmpty = "SELECT COUNT(*) FROM subjectbase WHERE ScheduledDay IS NOT NULL AND ScheduledTime IS NOT NULL AND StaffID IS NOT NULL"; //check if any class(subj) had been established
    qState = mysql_query(conn, isScheduleEmpty.c_str());
    if (!qState)
    {
        res = mysql_store_result(conn);
        int rowNo = mysql_num_rows(res);

        if (rowNo > 0) //if subj(s) exist
        {
            mysql_free_result(res);
            for (int i = 0; i < 7; i++) //display schedule for each day of a week, where subj(s) exist
            {
                string fetchSche = "SELECT SubjID, SubjName, ScheduledTime, subjectbase.StaffID, staffbase.StaffName FROM subjectbase LEFT JOIN staffbase ON subjectbase.StaffID = staffbase.StaffID WHERE ScheduledDay = '" + day[i] + "' AND ScheduledTime IS NOT NULL AND subjectbase.StaffID IS NOT NULL ORDER BY ScheduledTime";
                qState = mysql_query(conn, fetchSche.c_str());
                if (!qState)
                {
                    bool hasDay = false;
                    res = mysql_store_result(conn);
                    int colno = mysql_num_fields(res);

                    while ((row = mysql_fetch_row(res)) != nullptr)
                    {

                        if (!headerprinted) //print header & column header
                        {
                            system("cls");
                            cout << "                              \033[1;34mClass Schedule\033[0m";
                            cout << endl << "\033[1;33m----------------------------------------------------------------\033[0m\n";
                            cout << "\033[1;33m|\033[1;36m" << setw(8) << "Subj ID" << "\033[1;33m|\033[1;36m" << setw(13) << "Subj Name" << "\033[1;33m|\033[1;36m" << setw(13) << "Time" << "\033[1;33m|\033[1;36m" << setw(10) << "Taught by" << "\033[1;33m|\033[1;36m" << setw(13) << "Teacher Name" << "\033[1;33m|\033[1;36m" << "\033[0m";
                            cout << endl << "\033[1;33m                                                                \033[0m\n";
                            headerprinted = true;
                        }

                        if (!hasDay) //print day
                        {
                            cout << "\033[36m" << day[i] << "\033[0m\n";
                            hasDay = true;
                        }
                        cout << "\033[1;33m|\033[0m";

                        for (int j = 0; j < colno; j++) //print timetable
                        {
                            if (j == 0)
                                cout << setw(8) << row[j] << "\033[1;33m|\033[0m";
                            else if (j == 3)
                                cout << setw(10) << row[j] << "\033[1;33m|\033[0m";
                            else
                                cout << setw(13) << row[j] << "\033[1;33m|\033[0m";
                        }
                        cout << "\n";
                    }
                    mysql_free_result(res);
                }
                else
                    cout << "\n\n\033[31mQuery Execution Failed. [Error " << mysql_errno(conn) << "]\033[0m\n\nPress any key...\n";
                cout << "\n";
            }
        }
        else //no class established
            cout << "No class has been established.\n";
        cout << endl << "Press any key to return to previous menu...";
    }
    else
    {
        cout << "\n\n\033[31mQuery Execution Failed. [Error " << mysql_errno(conn) << "]\033[0m\n\nPress any key...\n";
        mysql_free_result(res);
    }
    _getch();
    return;
}
```

Figure 4.4 ScheduleView Function

4.2 Selection

If-else statements are extensively used for either user operation selection or conditional execution of code blocks.

Figure 4.5 shows a snippet of code that demonstrates the use of if-else blocks for leading users to different functions according to their input.



```
cout << "\033[1;36mSelect a module:\033[0m\n\n";
            cout << "1: Student Database\n2: Class Management\n3: Student Performance\n4: Finance Management\n5: Logout\n\n";
            cout << "\033[1;36mModule selected:\033[0m ";
            getline(cin, op);
            if (op == "1")
                StudentDB();
            else if (op == "2")
                ClassMgmt(uType, uID);
            else if (op == "3")
                StudentPerf(uType, uID);
            else if (op == "4")
                FinMgmt(uID);
            else if (op == "5")
                return;
            else
            {
                cout << "\033[31mInvalid Operation. Press any key...\033[0m";
                _getch();
            }
```

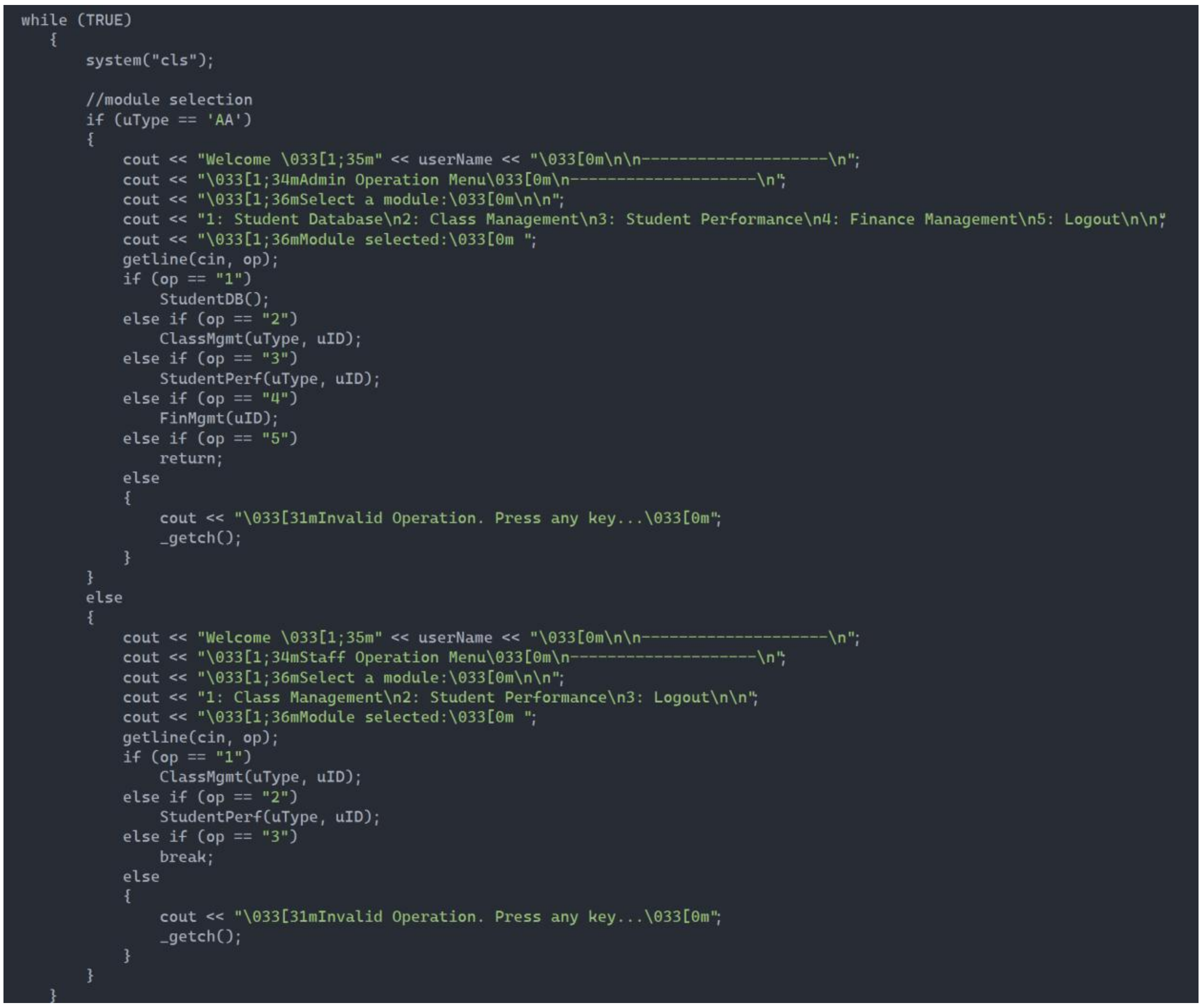
Figure 4.5 If-else statement

4.3 Control

While loops are mainly used in 2 areas in this system:

a) major operation menus (allow users to seamlessly navigate through different operations)

Figure 4.6 shows a snippet of code that allows users to navigate through different operations (e.g. the Student Module) before returning to the main menu, be it to execute another module or exit the program

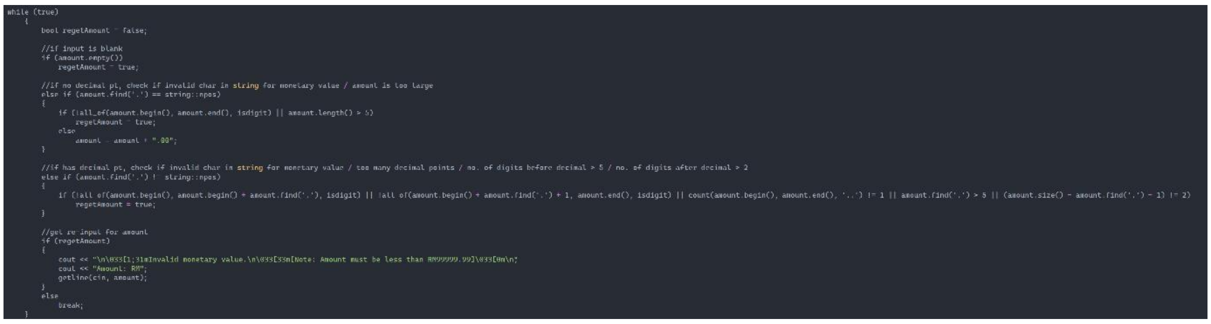


```
while (TRUE)
    {
        system("cls");

        //module selection
        if (uType == 'AA')
        {
            cout << "Welcome \033[1;35m" << userName << "\033[0m\n\n--------------------\n";
            cout << "\033[1;34mAdmin Operation Menu\033[0m\n--------------------\n";
            cout << "\033[1;36mSelect a module:\033[0m\n\n";
            cout << "1: Student Database\n2: Class Management\n3: Student Performance\n4: Finance Management\n5: Logout\n\n";
            cout << "\033[1;36mModule selected:\033[0m ";
            getline(cin, op);
            if (op == "1")
                StudentDB();
            else if (op == "2")
                ClassMgmt(uType, uID);
            else if (op == "3")
                StudentPerf(uType, uID);
            else if (op == "4")
                FinMgmt(uID);
            else if (op == "5")
                return;
            else
            {
                cout << "\033[31mInvalid Operation. Press any key...\033[0m";
                _getch();
            }
        }
        else
        {
            cout << "Welcome \033[1;35m" << userName << "\033[0m\n\n--------------------\n";
            cout << "\033[1;34mStaff Operation Menu\033[0m\n--------------------\n";
            cout << "\033[1;36mSelect a module:\033[0m\n\n";
            cout << "1: Class Management\n2: Student Performance\n3: Logout\n\n";
            cout << "\033[1;36mModule selected:\033[0m ";
            getline(cin, op);
            if (op == "1")
                ClassMgmt(uType, uID);
            else if (op == "2")
                StudentPerf(uType, uID);
            else if (op == "3")
                break;
            else
            {
                cout << "\033[31mInvalid Operation. Press any key...\033[0m";
                _getch();
            }
        }
    }
```

Figure 4.6 While Loop Example 1 [Menu Navigation]

b) error handling (prevent users from proceeding with invalid data insertion)

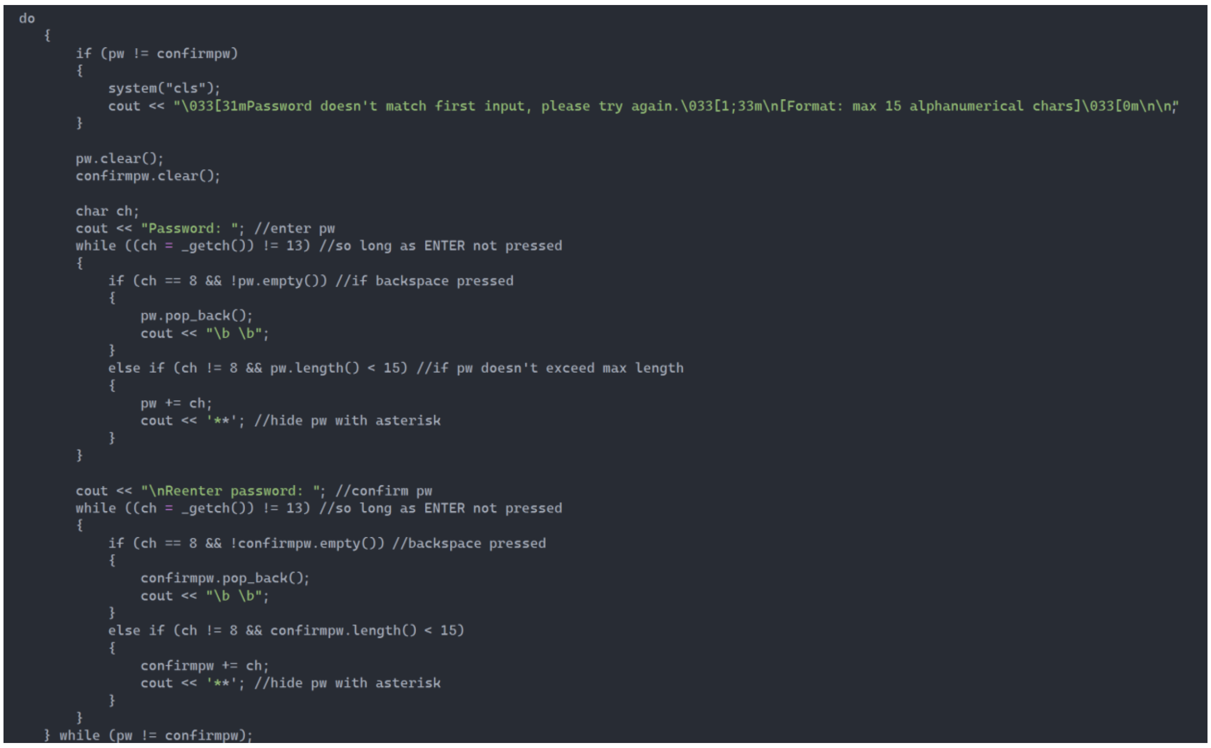
Figure 4.7 shows a snippet of code that checks if user-input value matches correct monetary format, and demands re-input if a fault is detected.



```
while (true)
    {
        bool regetAmount = false;

        //if input is blank
        if (amount.empty())
            regetAmount = true;

        //if no decimal pt, check if invalid char in string for monetary value / amount is too large
        else if (amount.find('.') == string::npos)
        {
            if (!all_of(amount.begin(), amount.end(), isdigit) || amount.length() > 5)
                regetAmount = true;
            else
                amount = amount + ".00";
        }

        //if has decimal pt, check if invalid char in string for monetary value / too many decimal points / no. of digits before decimal > 5 / no. of digits after decimal > 2
        else if (amount.find('.') != string::npos)
        {
            if (!all_of(amount.begin(), amount.begin() + amount.find('.'), isdigit) || !all_of(amount.begin() + amount.find('.') + 1, amount.end(), isdigit) || count(amount.begin(), amount.end(), '.') != 1 || amount.find('.') > 5 || (amount.size() - amount.find('.') - 1) != 2)
                regetAmount = true;
        }

        //get re-input for amount
        if (regetAmount)
        {
            cout << "\n\033[1;31mInvalid monetary value.\n\033[33m[Note: Amount must be less than RM99999.99]\033[0m\n";
            cout << "Amount: RM";
            getline(cin, amount);
        }
        else
            break;
    }
```

Figure 4.7 While Loop Example 2 [Monetary Value Error Handling]

Do-while loop is also utilised where certain conditions (e.g. reentered password matches initial input) has to be met for the program to proceed, as exemplified in the SetPW function shown in figure 4.8.



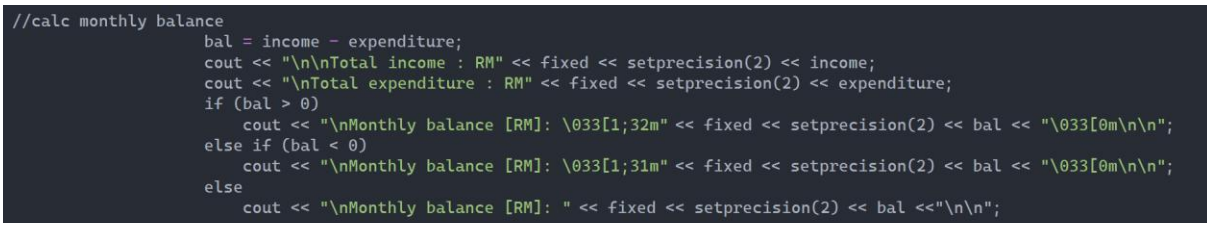
```
do
    {
        if (pw != confirmpw)
        {
            system("cls");
            cout << "\033[31mPassword doesn't match first input, please try again.\033[1;33m\n[Format: max 15 alphanumerical chars]\033[0m\n\n";
        }

        pw.clear();
        confirmpw.clear();

        char ch;
        cout << "Password: "; //enter pw
        while ((ch = _getch()) != 13) //so long as ENTER not pressed
        {
            if (ch == 8 && !pw.empty()) //if backspace pressed
            {
                pw.pop_back();
                cout << "\b \b";
            }
            else if (ch != 8 && pw.length() < 15) //if pw doesn't exceed max length
            {
                pw += ch;
                cout << '*'; //hide pw with asterisk
            }
        }

        cout << "\nReenter password: "; //confirm pw
        while ((ch = _getch()) != 13) //so long as ENTER not pressed
        {
            if (ch == 8 && !confirmpw.empty()) //backspace pressed
            {
                confirmpw.pop_back();
                cout << "\b \b";
            }
            else if (ch != 8 && confirmpw.length() < 15)
            {
                confirmpw += ch;
                cout << '*'; //hide pw with asterisk
            }
        }
    } while (pw != confirmpw);
```

Figure 4.8 Do-while Loop [Password Confirmation]

4.4 Calculation

Calulations are performed in several areas of the program to generate an output(report) from analyzing user-input values.

Figure 4.9 shows a snippet of code that performs calculation for monthly financial balance by subtracting expenditure from income over a set period, both of which data extracted from the database.



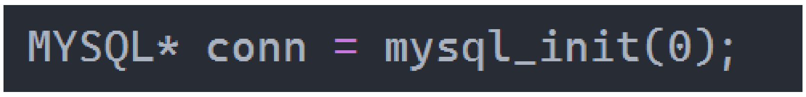
```
//calc monthly balance
                    bal = income - expenditure;
                    cout << "\n\nTotal income : RM" << fixed << setprecision(2) << income;
                    cout << "\nTotal expenditure : RM" << fixed << setprecision(2) << expenditure;
                    if (bal > 0)
                        cout << "\nMonthly balance [RM]: \033[1;32m" << fixed << setprecision(2) << bal << "\033[0m\n\n";
                    else if (bal < 0)
                        cout << "\nMonthly balance [RM]: \033[1;31m" << fixed << setprecision(2) << bal << "\033[0m\n\n";
                    else
                        cout << "\nMonthly balance [RM]: " << fixed << setprecision(2) << bal <<"\n\n";
```

Figure 4.8 Calculation of Monthly Balance

4.5 Pointer

Pointers were integral in database operations, including connecting to the database and managing query results.

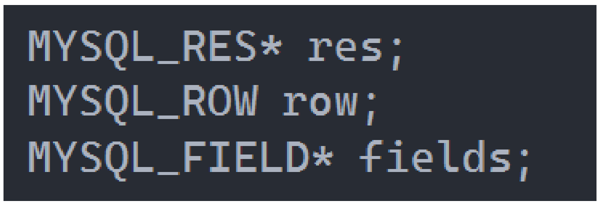
Figure 4.9 shows a pointer named conn for database connection.



```
MYSQL* conn = mysql_init(0);
```

Figure 4.9 Pointer for Database Connection

Figure 4.10 shows pointers res, row, fields which are used for managing query results.



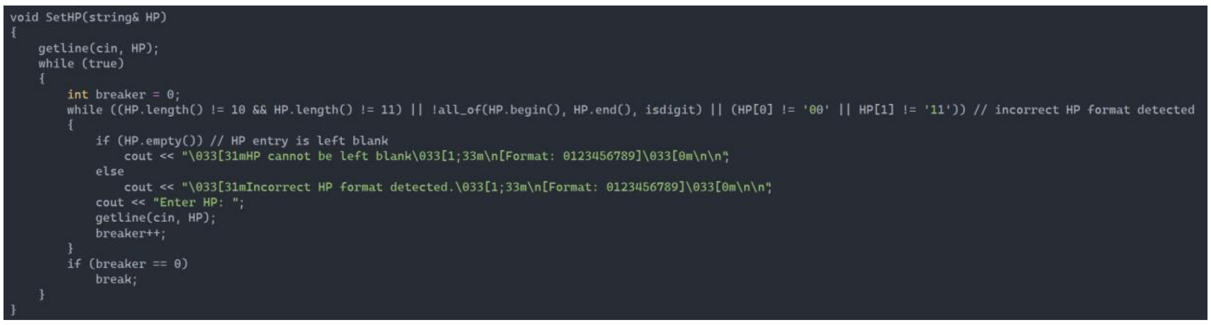
```
MYSQL_RES* res;
MYSQL_ROW row;
MYSQL_FIELD* fields;
```

Figure 4.10 Pointers for Managing Query Results

4.6 Error Handling

Error handling is implemented throughout the program to ensure accurate data processing and user operations executions.

Figure 4.11 shows the SetHP function which validates if the user input follows the correct Malaysian HP format before insertion to database.



```
void SetHP(string& HP)
{
    getline(cin, HP);
    while (true)
    {
        int breaker = 0;
        while ((HP.length() != 10 && HP.length() != 11) || !all_of(HP.begin(), HP.end(), isdigit) || (HP[0] != '00' || HP[1] != '11')) // incorrect HP format detected
        {
            if (HP.empty()) // HP entry is left blank
                cout << "\033[31mHP cannot be left blank\033[1;33m\n[Format: 0123456789]\033[0m\n\n";
            else
                cout << "\033[31mIncorrect HP format detected.\033[1;33m\n[Format: 0123456789]\033[0m\n\n";
            cout << "Enter HP: ";
            getline(cin, HP);
            breaker++;
        }
        if (breaker == 0)
            break;
    }
}
```

Figure 4.11 SetHP function [Error Handling]

Chapter 5: Conclusion

5.1 Constraints

Tuition Centre Manager is far from perfect with several constraints warranting attention. Firstly, some parameters within the system are hardcoded, such as the comment generation mechanism in the student performance module. This method, which bases comments on the difference in scores, may not be suitable for tests with varying grading scales, potentially leading to inaccurate or irrelevant feedback. Secondly, due to limitations in ability and time constraints during development, the user experience in certain areas remains underdeveloped. For instance, users may still be required to fill in all details before they can choose to cancel an update operation, making the process unnecessarily tedious and unintuitive. Furthermore, the system currently relies on hardcoded SQL queries, which introduce significant security risks by making the system vulnerable to SQL injection attacks if improperly handled. These constraints highlight areas for improvement in future iterations to enhance functionality, security, and user experience.

5.2 Future Improvements

To address the aforementioned limitations, one priority is to replace hardcoded parameters, such as the comment generation logic, with dynamic configurations that can adapt to tests with varying grading scales, such as allowing user-defined grading criterias and modifying the program as such dynamically. Another area for enhancement is the user experience, particularly by streamlining processes like canceling updates. Providing a more intuitive and user-friendly workflow, where users can exit operations without unnecessary steps, would significantly improve efficiency. To mitigate security risks, the system will have to adopt parameterized queries or prepared statements to prevent SQL injection attacks, thereby safeguarding data integrity and user information. These improvements would hopefully refine the system’s performance and usability, making it more robust and effective for its intended purposes.

References

kinsta. (2021, March 9). *How to fix the XAMPP error "MySQL shutdown unexpectedly" (3 methods).* Retrieved from kinsta: https://kinsta.com/knowledgebase/xampp-mysql-shutdown-unexpectedly/

Oracle. (2024, December 24). *MYSQL Connector/C++ 9.1 Developer Guide.* Retrieved from MYSQL Documentation: https://downloads.mysql.com/docs/connector-cpp-9.1-en.a4.pdf