

B.M.S COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU



UNIX SHELL PROGRAMMING
Report on
STUDENT INFORMATION MANAGEMENT SYSTEM

Submitted in partial fulfillment of the requirements for AAT

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

RAJA VISHWANATH DASARI
ROHAN B SHEKHAR
RITHVIK N

1BM23CS259
1BM23CS272
1BM23CS270

Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2024-2025

B.M.S COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DECLARATION

We, RAJA VISHWANATH DASARI (1BM23CS259), ROHAN B SHEKHAR (1BM23CS272), RITHVIK N (1BM23CS270) students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this AAT Project entitled "**STUDENT INFORMATION MANAGEMENT SYSTEM**" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester September 2024 - January 2025. We also declare that to the best of our knowledge and belief, the Project report is not from part of any other report by any other students.

Signature of the Candidates

RAJA VISHWANATH DASARI (1BM23CS259)

ROHAN B SHEKHAR (1BM23CS272)

RITHVIK N (1BM23CS270)

BMS COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the AAT Project titled “**STUDENT INFORMATION MANAGEMENT SYSTEM**” has been carried out by RAJA VISHWANATH DASARI (1BM23CS259), ROHAN B SHEKHAR (1BM23CS272), RITHVIK N (1BM23CS270) during the academic year 2024-2025.

Signature of the Guide

Signature of Head of the Department

Signature of Examiners with date

1. Internal Examiner

2. External Examiner

ABSTRACT

The Student Management System is a Bash-based application enhanced with Zenity to provide an intuitive graphical interface for managing student records efficiently. The project is structured to organize data hierarchically by branch (CSE, ECE, EEE, MECH), semester (1st to 8th), and section (A and B). Each section has its dedicated file to store student information, including unique IDs, names, contact numbers, and section identifiers. This design ensures a clean and accessible database while supporting scalability for larger datasets.

The system offers several user-friendly features that make student management simple and effective. Users can add new students, view existing records, mark attendance, and input marks for specific subjects. Attendance tracking includes timestamps and status (present/absent) for each student, while marks entry is validated to ensure accurate data. These records are easily retrievable for review or updates. One of the system's standout features is its ability to generate detailed reports that summarize attendance percentages and average marks for all students in a section. For quick academic assessments, the system also identifies and highlights the top five students based on their average performance.

Zenity's graphical interface enhances the user experience, offering clear menus and forms for selecting branches, semesters, and sections. This simplifies navigation and ensures the process of managing data remains seamless and error-free. The project also automates report generation, reducing manual effort and eliminating the risk of calculation errors. Attendance records, marks data, and reports are securely stored for future reference, ensuring reliable documentation.

This Student Management System is a practical and scalable tool tailored for academic institutions. It reduces the complexity of handling student data manually and provides a streamlined approach to managing academic records. By integrating automation and user-friendly features, the system offers an effective solution for tracking attendance, recording performance, and generating insightful reports, making it an essential asset for any educational environment.

TABLE OF CONTENTS

1. Introduction	
1.1. Introduction to UNIX	1
1.2. Motivation for the Work	3
2. Methodology and Framework	
2.1. System Requirement	4
2.2. Algorithms, Techniques	5
3. Implementation	
3.1. Source Code	9
3.2. List of Main UNIX Commands	24
3.3. Uses and its Syntax	25
4. Result and Analysis	27
5. Conclusion and Future Enhancement	34
References	35

LIST OF FIGURES

Figure 4.1	
• Main Menu	27
Figure 4.2	
• Selection Of Branch	27
Figure 4.3	
• Selection of Semester	27
Figure 4.4	
• Selection Of Section	27
Figure 4.5	
• Label showing selected details	28
Figure 4.6	
• Form of adding students	28
Figure 4.7	
• Message for successful addition of student	28
Figure 4.8	
• Error if null values in the form	28
Figure 4.9	
• Error message when contact field has entry other than 10 digits	29
Figure 4.10	
• Student Records	29
Figure 4.11	
• Radiolist to mark attendance	30
Figure 4.12	
• Success message after taking attendance	30
Figure 4.13	
• Attendance Records	30

Figure 4.14

- Form for adding marks 31

Figure 4.15

- Success message after adding marks 31

Figure 4.16

- Error when marks are not in between 0 to 100 31

Figure 4.17

- Marks Record 32

Figure 4.18

- Top 5 Students 32

Figure 4.19

- Message on generation of report 32

Figure 4.20

- View of generated report 33

Chapter 1: Introduction

1.1 Introduction to UNIX

UNIX is one of the most groundbreaking operating systems in the history of computing. It was developed in the late 1960s at Bell Laboratories by a team led by Ken Thompson and Dennis Ritchie. At the time, computers were large and complex, and most operating systems were tightly bound to specific hardware. UNIX was designed to break away from this norm by being simple, portable, and flexible. Over time, it has grown into a powerful system that has influenced the development of countless modern operating systems, including Linux and macOS. Its longevity and widespread adoption are a testament to its innovative design and practical utility.

One of UNIX's most notable features is its ability to manage multiple tasks and users at the same time. This multitasking and multiuser functionality was a major advancement in computing. It allows multiple users to log into the same machine, each performing their own operations without conflict. This feature has made UNIX a staple in environments where reliability and efficiency are critical, such as servers, academic institutions, and research facilities. It also set the stage for modern cloud computing, where many users share resources on powerful systems.

Another key feature of UNIX is its hierarchical file system. Unlike earlier systems, which often stored files in disorganized flat structures, UNIX organizes files into a logical tree-like directory structure. This approach makes it easy to navigate and manage large amounts of data. One of UNIX's defining principles is treating everything as a file, whether it is text, devices, or running processes. This abstraction simplifies interactions with the system, as users can access and manipulate resources in a consistent manner. For example, reading from a file and reading input from a keyboard can use the same commands, which enhances the system's usability and flexibility.

UNIX primarily operates through a command-line interface, where users issue commands by typing them directly into the system. While this method may seem intimidating at first, it provides immense control over the system. Users can execute complex operations, automate tasks, and manage processes with precision. The UNIX philosophy emphasizes creating small, modular programs that perform a single task well. These programs can be combined in countless ways using

pipes and redirection, enabling users to solve complex problems efficiently. This design principle has not only shaped UNIX but has also influenced how modern software is developed.

Portability is another major factor in UNIX's success. Unlike many of its contemporaries, UNIX was written in the C programming language, which made it easier to adapt to new hardware platforms. This feature helped UNIX gain widespread use across a variety of systems, from personal computers to powerful mainframes. Its adaptability allowed it to flourish in academic and research settings, where it became a teaching tool and a foundation for innovation. Many computer science students and researchers have worked on UNIX or UNIX-like systems, further cementing its role in the development of modern computing.

Networking capabilities are one of UNIX's strongest features. UNIX includes built-in tools for connecting multiple systems, sharing files, and running applications across a network. Many of the protocols that drive the modern internet, including TCP/IP, were developed and tested on UNIX systems. These tools made UNIX an essential part of early internet development and continue to make it a reliable choice for servers and networked environments. Even today, UNIX and its derivatives are widely used to host websites, manage cloud infrastructure, and operate large-scale networks.

Security has always been a priority in UNIX design. It was among the first operating systems to implement user authentication, file permissions, and access control mechanisms to protect data and processes. By assigning different levels of permissions to users and files, UNIX ensures that sensitive information remains secure and that unauthorized users cannot disrupt the system. This focus on security, along with its stability, has made UNIX a trusted platform for critical applications such as financial systems, scientific research, and enterprise-level operations.

Over the decades, UNIX has had a profound influence on the computing world. Its design principles have inspired numerous operating systems, including Linux, which powers millions of devices worldwide, and macOS, which serves as Apple's flagship system. These systems have inherited UNIX's strengths, such as its modularity, reliability, and emphasis on simplicity, while also introducing new features to meet the demands of modern users. The enduring popularity of UNIX and its derivatives highlights its importance in shaping the technological landscape.

In conclusion, UNIX is much more than an operating system. It is a set of ideas and principles that prioritize simplicity, efficiency, and adaptability. These qualities have enabled it to remain relevant for over five decades, even as technology has advanced at a rapid pace. UNIX has not only laid the foundation for modern computing but also continues to play a vital role in the development of new technologies. Its legacy is a reminder that good design can stand the test of time.

1.2 Motivation

When deciding on a project to develop in UNIX, we chose a Student Management System because it brings together practicality, relevance, and the opportunity to explore key UNIX concepts. Managing student data, attendance, and performance is an essential part of any academic institution, which makes this project both useful and impactful for real-world applications. It allows us to delve into UNIX's powerful features, including its file handling capabilities, process management, and scripting tools. The project provides an excellent platform to design a structured file system, organizing data by branches, semesters, and sections—showcasing UNIX's core principle of logical and hierarchical file systems. Moreover, it emphasizes the use of shell scripting to automate processes such as inputting student data, generating reports, and tracking attendance, underlining the efficiency and flexibility of the command-line interface. UNIX's robust text processing utilities, such as `grep`, `awk`, and `sed`, will be leveraged to manipulate and extract data, enhancing the system's functionality and performance. The integration of Zenity for graphical interfaces adds a layer of accessibility and user-friendliness, blending the power of the command line with modern graphical elements. By taking on this project, we will demonstrate our understanding of UNIX's multitasking capabilities, modular design, file permissions, and security features, all while solving a practical problem in a manner that aligns with UNIX's philosophy of simplicity, efficiency, and scalability.

Chapter 2: Methodology and Framework

2.1. System Requirements

Hardware Requirements:

- **Processor:** A minimum of 1 GHz CPU is recommended.
- **RAM:** 2 GB of RAM is sufficient for handling moderate datasets. However, higher RAM (4 GB or more) is recommended for larger data processing.
- **Storage:** At least 5 GB of free disk space for storing student data files (CSV files for students, attendance, and marks) and reports.
- **Display:** A monitor with a resolution of 1024x768 or higher for viewing the graphical user interface and terminal outputs.
- **Input Devices:** Keyboard and mouse for interacting with the Zenity graphical interface and entering commands via the terminal.

Software Requirements:

- **Operating System:** A UNIX-based OS (such as **Linux** or **macOS**) is required for shell scripting and using the tools mentioned in the code.
- **Shell: Bash** (or another compatible shell) is required to execute the script. The script uses standard shell commands, and Bash is compatible with all the used functionalities.
- **Zenity:** This tool is required to display graphical user interfaces (GUI) for actions like forms, lists, and text windows. It simplifies user interaction with the system.
- **Text Editors:** Any text editor for creating and modifying shell scripts, such as **nano**, **vim**, or **gedit**.
- **Command-Line Utilities:**
 - **grep, sed:** These tools are used for text processing and data extraction/manipulation from CSV files.
 - **bc:** Used for performing mathematical calculations, such as calculating attendance percentages and average marks.
 - **date:** To handle the current date when marking attendance.
 - **wc:** Used to count the number of lines in files (for generating student IDs).

2.2. Algorithms, Techniques

2.2.1 Algorithm

STEP 1: Display Main Menu for Branch, Semester, and Section Selection

The system presents a menu where the user selects the branch, semester, and section. This selection sets the context for subsequent operations.

STEP 2: Provide Functional Options

After selecting the section, the user is presented with a menu containing options such as "Add Student," "View Students," "Mark Attendance," "View Attendance," "Add Marks," "View Marks," "Top Students," and "Generate Reports."

STEP 3: Perform Selected Operation

Based on the user's choice, the corresponding functionality is executed. This could involve interacting with student data, recording attendance, or generating reports, with appropriate validations and feedback provided.

STEP 4: Return to the Main Menu

After completing an operation, the system returns to the main menu, allowing the user to perform additional actions or select a different section.

STEP 5: Exit the Program

The user can exit the system either by selecting the "Exit" option from the menu or by clicking the "Cancel" button during any operation, terminating the program gracefully.

2.2.2 Techniques

1. Initialization

The system begins by creating a directory structure that represents the hierarchy of branches, semesters, and sections. This is achieved under a root directory named students. For each combination of branch, semester, and section, a specific directory is created. Within each section directory, a CSV file is initialized to store student records, with headers including Student ID, Name, Section, and Contact. This ensures that the structure is ready for data input and retrieval.

2. Selection

The user interacts with the program via graphical dialogs powered by Zenity to select a branch, semester, and section. These selections are stored in global variables and determine the current working context, ensuring that subsequent operations, such as adding students or viewing attendance, are applied to the correct section.

3. Add Student

To add a student, the system collects details such as the student's name, section, and contact number using a form interface. Input validation ensures that all fields are filled, the section is valid, and the contact number adheres to a 10-digit format. Once validated, a unique student ID is generated, and the record is appended to the relevant section's CSV file, maintaining accurate and organized records.

4. View Students

The system provides the ability to view student records from the selected section. It reads the data from the corresponding CSV file and displays it in a Zenity text information dialog. If no records exist, the user is informed of the absence of data.

5. Mark Attendance

Attendance is marked for students by iterating through the section's student list. The user is prompted to mark each student as "Present" or "Absent" via a graphical interface. The attendance records, including the student ID, date, time, and status, are appended to a separate attendance CSV file for the section, ensuring historical tracking of attendance data.

6. View Attendance

Attendance data is displayed for the selected section by reading from the relevant attendance CSV file. The system checks for the existence of records and notifies the user if no data is available. If records are present, they are shown in an easy-to-read format using Zenity.

7. Add Marks

Marks for students are recorded by collecting the student ID, subject name, and marks through a form. The system validates the input to ensure the marks are numeric and within the range of 0–100. Additionally, it checks if the provided student ID exists in the section. Once validated, the data is appended to a marks CSV file for the section.

8. View Marks

Marks records for the selected section are retrieved from the marks CSV file and displayed to the user. If no data is found, the user is informed. Otherwise, the system presents the marks in a structured format, allowing easy access to performance details.

9. Top Students

The system identifies the top-performing students in the selected section by calculating the average marks for each student based on their scores. The data is then sorted in descending order, and the top five students are displayed along with their average marks. This provides a quick insight into the highest achievers in the section.

10. Generate Reports

Reports are generated by aggregating data on attendance and marks for each student in the section. Attendance percentage is calculated based on the number of sessions attended, while average marks are computed from the recorded scores. The report is saved as a CSV file and includes details such as student ID, name, attendance percentage, and average marks. This comprehensive summary provides a holistic view of student performance.

11. View Reports

Generated reports can be viewed directly through the program. If a report exists for the selected section, it is displayed in Zenity. If no report is found, the system notifies the user. This functionality ensures easy access to consolidated performance data.

12. Main Menu

The main menu serves as the central interface for the user, presenting options to perform all the system's functions, including selecting a section, adding students, marking attendance, and generating reports. The menu continuously loops until the user chooses to exit, providing seamless navigation and control over the system's features.

Chapter 3. Implementation

3.1 Source Code

```
#!/bin/bash

# Root directory for storing student data
students_dir="students"

# Function to initialize the directory structure
initialize_directories() {
    mkdir -p "$students_dir"

    branches=("CSE" "ECE" "EEE" "MECH")

    semesters=("1st_Sem" "2nd_Sem" "3rd_Sem" "4th_Sem" "5th_Sem" "6th_Sem" "7th_Sem"
"8th_Sem")

    sections=("A.csv" "B.csv")

    for branch in "${branches[@]}; do
        for semester in "${semesters[@]}; do
            mkdir -p "$students_dir/$branch/$semester"

            for section in "${sections[@]}; do
                file_path="$students_dir/$branch/$semester/$section"

                if [[ ! -f "$file_path" ]]; then
                    echo "Student ID,Name,section,Contact" > "$file_path"
                fi
            done
        done
    done

    echo "Directory structure initialized."
}

initialize_directories
```



```

# Global variables for branch, semester, and section

branch=""

semester=""

section=""

section_file=""


# Function to select branch, semester, and section once
select_branch_semester_section() {

    branch=$(zenity --list --title="Select Branch" --column="Branches" "CSE" "ECE" "EEE"
"MECH")

    if [[ -z "$branch" ]]; then return 1; fi


    semester=$(zenity --list --title="Select Semester" --column="Semesters" "1st_Sem"
"2nd_Sem" "3rd_Sem" "4th_Sem" "5th_Sem" "6th_Sem" "7th_Sem" "8th_Sem")

    if [[ -z "$semester" ]]; then return 1; fi


    section=$(zenity --list --title="Select Section" --column="Sections" "A" "B")

    if [[ -z "$section" ]]; then return 1; fi


    section_file="$students_dir/$branch/$semester/${section}.csv"

    echo "Selected: $branch $semester $section"

}


add_student() {

    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then

        zenity --error --text="Please select a branch, semester, and section first."

        return

    fi
}

```

```

while true; do

    student_details=$(zenity --forms --title="Add Student" \
        --add-entry="Name" \
        --add-entry="Section (A/B)" \
        --add-entry="Contact (10-digit number)" \
        --separator=",")

    if [[ $? -ne 0 ]]; then
        zenity --error --text="No student added."
        return
    fi

    IFS=',' read -r name section contact <<< "$student_details"

    # Validation for empty fields
    if [[ -z "$name" || -z "$section" || -z "$contact" ]]; then
        zenity --error --text="All fields are required. Please try again."
        continue
    fi

    # Validation for section
    if [[ "$section" != "A" && "$section" != "B" ]]; then
        zenity --error --text="Section must be 'A' or 'B'. Please try again."
        continue
    fi

    # Validation for contact number
    if [[ ! "$contact" =~ ^[0-9]{10}$ ]]; then

```

```

        zenity --error --text="Contact must be a 10-digit number. Please try again."

        continue
    fi

    # Generate a new unique Student ID
    student_id=$(( $(wc -l < "$section_file") - 1 ))

    echo "$student_id,$name,$section,$contact" >> "$section_file"

    zenity --info --text="Student '$name' added successfully with ID $student_id."

    break
done
}

# Function to view students
view_students() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."

        return
    fi

    student_records=$(tail -n +2 "$section_file")

    if [[ -z "$student_records" ]]; then
        zenity --info --text="No student records found."
    else
        zenity --text-info --title="Student Records" --filename="$section_file"
    fi
}

# Function to mark attendance

```

```

mark_attendance() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

    attendance_file="${section_file%.csv}_attendance.csv"
    if [[ ! -f "$attendance_file" ]]; then
        echo "Student ID,Date,Time,Status" > "$attendance_file"
    fi

    current_date=$(date +%Y-%m-%d)
    current_time=$(date +%H:%M:%S)
    student_records=$(tail -n +2 "$section_file")

    if [[ -z "$student_records" ]]; then
        zenity --info --text="No students found to mark attendance."
        return
    fi

    while IFS=' ' read -r student_id name section contact; do
        status=$(zenity --list --title="Mark Attendance" --radiolist \
            --column="Select" --column="Status" TRUE "Present" FALSE "Absent" \
            --text="Mark attendance for $name (ID: $student_id)")

        if [[ $? -ne 0 ]]; then
            zenity --error --text="Attendance marking canceled."
            return
        fi
    done
}

```

```

fi

    echo "$student_id,$current_date,$current_time,$status" >> "$attendance_file"
done <<< "$student_records"

zenity --info --text="Attendance marked successfully."
}

# Function to view attendance
view_attendance() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

    attendance_file="${section_file%.csv}_attendance.csv"
    if [[ ! -f "$attendance_file" ]]; then
        zenity --info --text="No attendance records found."
        return
    fi

    attendance_records=$(tail -n +2 "$attendance_file")
    if [[ -z "$attendance_records" ]]; then
        zenity --info --text="No attendance records found."
    else
        zenity --text-info --title="Attendance Records" --filename="$attendance_file"
    fi
}

```

```

add_marks() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

    marks_file="${section_file%.csv}_marks.csv"
    if [[ ! -f "$marks_file" ]]; then
        echo "Student ID,Subject,Marks" > "$marks_file"
    fi

    while true; do
        marks_details=$(zenity --forms --title="Add Marks" \
            --add-entry="Student ID" \
            --add-entry="Subject" \
            --add-entry="Marks (0-100)" \
            --separator=",")
        if [[ $? -ne 0 ]]; then
            zenity --error --text="No marks added."
            return
        fi

        IFS=' ' read -r student_id subject marks <<< "$marks_details"

        # Validation for empty fields
        if [[ -z "$student_id" || -z "$subject" || -z "$marks" ]]; then
            zenity --error --text="All fields are required. Please try again."

```

```

        continue
    fi

    # Validation for numeric marks between 0 and 100
    if [[ ! "$marks" =~ ^[0-9]+$ || "$marks" -lt 0 || "$marks" -gt 100 ]]; then
        zenity --error --text="Marks must be a number between 0 and 100. Please try again."
        continue
    fi

    # Validation for valid Student ID
    if ! grep -q "^$student_id," "$section_file"; then
        zenity --error --text="Student ID $student_id does not exist. Please try again."
        continue
    fi

    echo "$student_id,$subject,$marks" >> "$marks_file"
    zenity --info --text="Marks added successfully for Student ID $student_id in $subject."
    break
done
}

# Function to view marks
view_marks() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

```

```

marks_file="${section_file%.csv}_marks.csv"

if [[ ! -f "$marks_file" ]]; then

    zenity --info --text="No marks records found."

    return

fi

marks_records=$(tail -n +2 "$marks_file")

if [[ -z "$marks_records" ]]; then

    zenity --info --text="No marks records found."

else

    zenity --text-info --title="Marks Records" --filename="$marks_file"

fi

}

# Function to generate reports
generate_reports() {

    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then

        zenity --error --text="Please select a branch, semester, and section first."

        return

    fi

    report_file="students/$branch/$semester/report.csv"

    echo "Student ID,Name,Attendance Percentage,Average Marks" > "$report_file"

    while IFS=, read -r student_id name section contact; do

        local attendance_file="${section_file%.csv}_attendance.csv"

        local marks_file="${section_file%.csv}_marks.csv"

```



```

if [[ -f $attendance_file ]]; then
    local total_records=$(grep -c "^$student_id," "$attendance_file")
    local present_records=$(grep -c "^$student_id,*,Present" "$attendance_file")
    local attendance_percentage=$(echo "scale=2; ($present_records / $total_records) * 100"
| bc)
else
    attendance_percentage="N/A"
fi

if [[ -f $marks_file ]]; then
    local total_marks=0
    local marks_count=0
    while IFS=, read -r id sub marks; do
        if [[ "$id" == "$student_id" ]]; then
            total_marks=$((total_marks + marks))
            marks_count=$((marks_count + 1))
        fi
    done < "$marks_file"

    if ((marks_count > 0)); then
        average_marks=$(echo "scale=2; $total_marks / $marks_count" | bc)
    else
        average_marks="N/A"
    fi
else
    average_marks="N/A"
fi

```

```

        echo "$student_id,$name,$attendance_percentage,$average_marks" >> "$report_file"
    done <<(tail -n +2 "$section_file")

    zenity --info --text="Report generated at $report_file."
}

get_top_students() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

    # Ensure the marks file exists
    marks_file="${section_file%.csv}_marks.csv"
    if [[ ! -f "$marks_file" ]]; then
        zenity --error --text="No marks records found for the selected section."
        return
    fi

    # Extract the marks records (excluding the header)
    marks_records=$(tail -n +2 "$marks_file")
    if [[ -z "$marks_records" ]]; then
        zenity --info --text="No marks records found in the section."
        return
    fi

    # Calculate the average marks for each student
    declare -A student_totals

```

```

declare -A student_counts

while IFS=' ' read -r student_id subject marks; do
    student_totals["$student_id"]=$(( ${student_totals["$student_id"]} + marks ))
    student_counts["$student_id"]=$(( ${student_counts["$student_id"]} + 1 ))
done <<< "$marks_records"

# Compute average marks
declare -A student_averages

for student_id in "${!student_totals[@]}"; do
    student_averages["$student_id"]=$(echo "scale=2; ${student_totals["$student_id"]} /
${student_counts["$student_id"]}" | bc)
done

# Combine average marks with student details
student_details=$(tail -n +2 "$section_file")

top_students=$(while IFS=' ' read -r student_id name section contact; do
    avg_marks=${student_averages["$student_id"]:-0}
    echo "$student_id,$name,$avg_marks"
done <<< "$student_details" | sort -t',' -k3 -nr | head -n 5)

# Prepare the output for display
output="Rank\tID\tName\tAverage Marks\n"

rank=1

while IFS=' ' read -r student_id name avg_marks; do
    output+="$rank\t$student_id\t$name\t$avg_marks\n"
    ((rank++))
done <<< "$top_students"

```

```

# Display the top students using Zenity

echo -e "$output" | zenity --text-info \
    --title="Top 5 Students by Average Marks" \
    --width=600 --height=400 \
    --filename=/dev/stdin
}

# Function to get the current selection label
get_selection_label() {
    echo "${branch:-NIL}, ${semester:-NIL}, ${section:-NIL}"
}

# Function to view generated reports
view_report() {
    if [[ -z "$branch" || -z "$semester" || -z "$section" ]]; then
        zenity --error --text="Please select a branch, semester, and section first."
        return
    fi

    report_file="students/$branch/$semester/report.csv"

    if [[ ! -f "$report_file" ]]; then
        zenity --info --text="No report found for the selected section."
        return
    fi

    report_records=$(tail -n +2 "$report_file")

```

```

if [[ -z "$report_records" ]]; then
    zenity --info --text="The report is empty."
else
    zenity --text-info --title="Generated Report" --filename="$report_file" --width=800 --
height=600
fi
}

```

Update the main menu to include View Report

```

main_menu() {
    while true; do
        # Get the current selection label
        current_selection=$(get_selection_label)

        # Show main menu options
        action=$(zenity --list --title="Main Menu" \
            --text="Current Selection: Branch: $current_selection" \
            --column="Options" \
            "Select Branch, Semester, and Section" \
            "Add Student" \
            "View Students" \
            "Mark Attendance" \
            "View Attendance" \
            "Add Marks" \
            "View Marks" \
            "Top 5 Students" \
            "Generate Reports" \
            "View Report" \

```

```

"Exit")

# Check Zenity exit status
if [[ $? -ne 0 ]]; then
    # If Cancel or Close button is clicked, exit the app
    exit 0
fi

case $action in
    "Select Branch, Semester, and Section") select_branch_semester_section ;;
    "Add Student") add_student ;;
    "View Students") view_students ;;
    "Mark Attendance") mark_attendance ;;
    "View Attendance") view_attendance ;;
    "Add Marks") add_marks ;;
    "View Marks") view_marks ;;
    "Top 5 Students") get_top_students ;;
    "Generate Reports") generate_reports ;;
    "View Report") view_report ;;
    "Exit") exit 0 ;;
    *) zenity --error --text="Invalid option. Please try again." ;;
esac

done
}

# Start the main menu
main_menu

```

3.2. List of Main UNIX Commands

Directory and File Management Commands

- **mkdir -p**: Used to create nested directories for organizing student data by branch, semester, and section. The -p option ensures that parent directories are created as needed.
- **echo**: Writes text to files, such as initializing CSV headers or appending student and attendance data.
- **wc -l**: Counts the number of lines in a file, used to generate unique student IDs.
- **tail -n +2**: Extracts all lines except the header line from CSV files, used to process data records without headers.
- **grep**: Searches for specific patterns in files, such as validating student IDs or counting attendance records.

String and Variable Processing Commands

- **IFS=',' read -r**: Reads and parses CSV-formatted strings into variables for further processing.
- **bc**: Used for performing floating-point arithmetic, such as calculating attendance percentages or average marks.

Conditional and Loop Constructs in Shell

- **if [[...]]**: Used for conditional checks, such as verifying inputs, validating file existence, or ensuring valid data.
- **while ... do ... done**: Loops for iterating through records or repeatedly prompting the user for input.
- **case ... esac**: Implements menu-based functionality by branching logic based on the user's selection.

Date and Time Commands

- **date**: Retrieves the current date and time, used for marking attendance timestamps.

Zenity for GUI Dialogs

- **zenity --list**: Creates dropdown menus for selecting options like branch, semester, and section.
- **zenity --forms**: Prompts the user to input data through a form interface.
- **zenity --info**: Displays informational messages to the user.
- **zenity --error**: Displays error messages when validations fail.
- **zenity --text-info**: Displays text information, such as student records or reports, in a scrollable dialog.

File Redirection and Manipulation

- **>>**: Appends output to a file, used to add new records to CSV files.
- **< and <<**: Reads input from files or here-documents, used for processing file contents.

3.3 Uses of the commands in the main script

Directory and File Management Commands

- **mkdir -p**: Used to create directories for each branch, semester, and section dynamically before storing student data.
- **echo**: Used to write headers or append data like student details or attendance records into CSV files.
- **wc -l**: Counts the lines in a file to generate a unique student ID by adding 1 to the current line count.
- **tail -n +2**: Skips the header in CSV files when processing records for calculations or display.
- **grep**: Validates inputs like student IDs or searches for attendance records in files.

String and Variable Processing Commands

- **IFS=','** **read -r**: Parses CSV data to assign values like student name or ID to variables for further operations.
- **bc**: Calculates percentages for attendance or average marks with precision.

Conditional and Loop Constructs in Shell

- **if [[...]]**: Ensures inputs are valid and checks the existence of directories or files.
- **while ... do ... done**: Loops through records for displaying data or repeatedly prompts users for input.
- **case ... esac**: Handles menu options like adding students, marking attendance, or exiting the application.

Date and Time Commands

- **date**: Marks attendance entries with the current timestamp for tracking purposes.

Zenity for GUI Dialogs

- **zenity --list**: Displays options for branch, semester, and section in dropdown menus.
- **zenity --forms**: Collects user inputs for adding student details or other operations through a form interface.
- **zenity --info**: Confirms successful operations like adding a student or saving attendance.
- **zenity --error**: Notifies users about invalid inputs or missing files.
- **zenity --text-info**: Shows detailed student records or attendance reports in a scrollable window.

File Redirection and Manipulation

- **>>**: Appends new student records or attendance data to existing files without overwriting.
- **<** **and** **<<**: Reads file content for processing or utilizes here-documents for displaying multi-line messages.

Chapter 4: Result and Analysis

4.1 Main Menu And Selection of Branch, Semester and Section

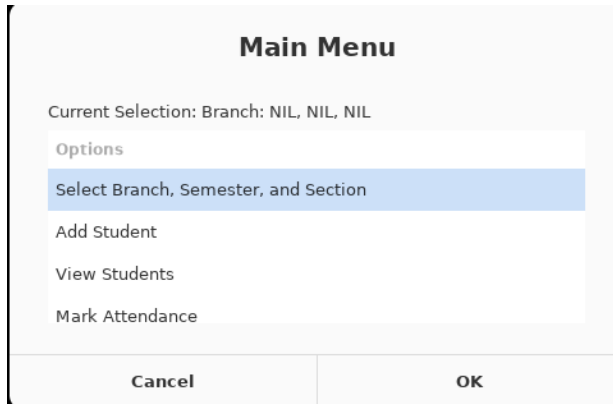


Fig 4.1 Main Menu

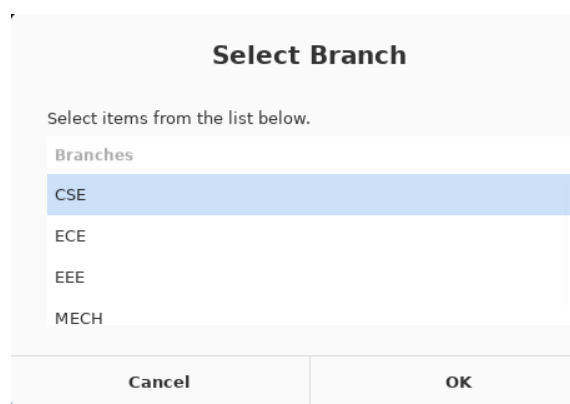


Fig 4.2 Selection Of Branch

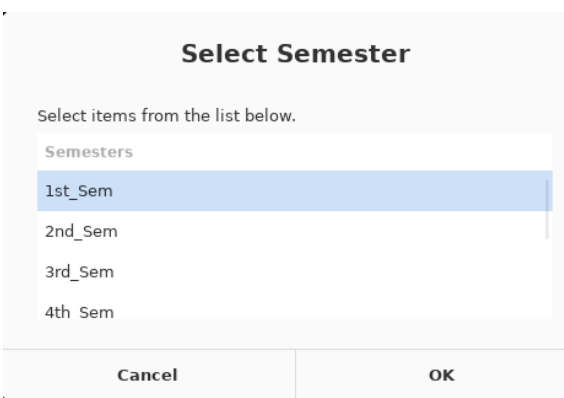


Fig 4.3 Selection Of Semester

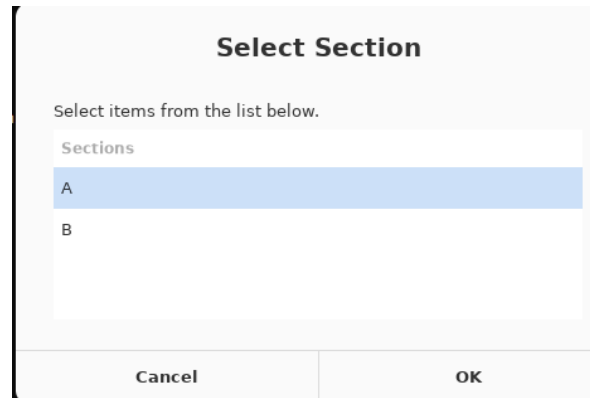


Fig 4.4 Selection Of Section

Figure 4.1 represents the main menu of the application, where users can select options such as adding a student, marking attendance, viewing records, or exiting the system.

Figure 4.2 illustrates the branch selection menu, allowing the user to choose a specific branch of study for organizing data.

Figure 4.3 showcases the year or semester selection interface, where users can pick the academic year or semester to refine data categorization.

Figure 4.4 displays the section selection screen, enabling users to specify the section for managing students and attendance.

Main Menu

Current Selection: Branch: CSE, 1st_Sem, A

Options

Select Branch, Semester, and Section

Add Student

View Students

Mark Attendance

Cancel OK

Fig 4.5 Label showing selected details

Add Student

Forms dialog

Name Raj

Section (A/B) A

Contact (10-digit number) 1234567890

Cancel OK

Fig 4.6 Form of adding student

Figure 4.5 displays the label that shows the currently selected branch, year (semester), and section. This label dynamically updates to reflect the user's selections made in Figures 4.2, 4.3, and 4.4. It provides a clear and concise summary of the selected options, ensuring users are always aware of their current context. The label is prominently shown in the main menu to avoid confusion and improve usability.

Figure 4.6 represents the form used for adding a new student to the system. It prompts the user to input details such as the student's name, section, and contact number, ensuring all necessary information is collected accurately.

Information

Student 'Raj' added successfully with ID 10.

OK

Fig 4.7 Message for successful addition of student

Error

All fields are required. Please try again.

OK

Fig 4.8 Error if null values in the form

Figure 4.7 displays a confirmation message indicating the successful addition of a student to the system. It provides feedback with details such as the student's name and assigned ID.

Figure 4.8 shows an error message triggered when null or empty values are entered in the form. It prompts the user to provide valid inputs for all required fields before proceeding.

Figure 4.9 presents an error message displayed when the contact field contains an entry that is not exactly 10 digits. It ensures that the contact number adheres to the required format for validation.

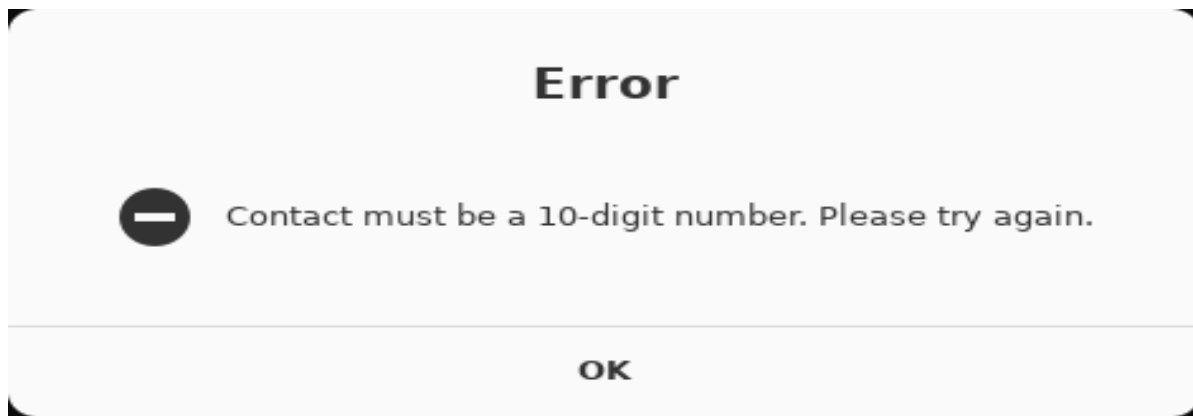


Fig 4.9 Error message when contact field has entry other than 10 digits

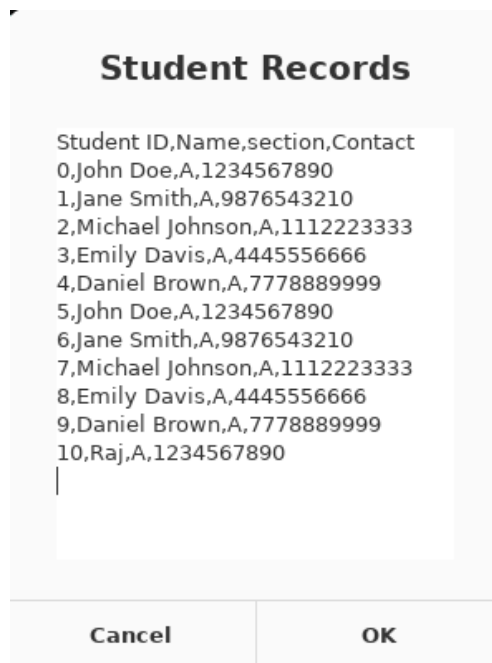


Fig 4.10 Student Records

Figure 4.10 displays the student records when the "View Students" option is clicked. It provides a detailed list of students, including their respective details, in a scrollable format for easy reference and review.

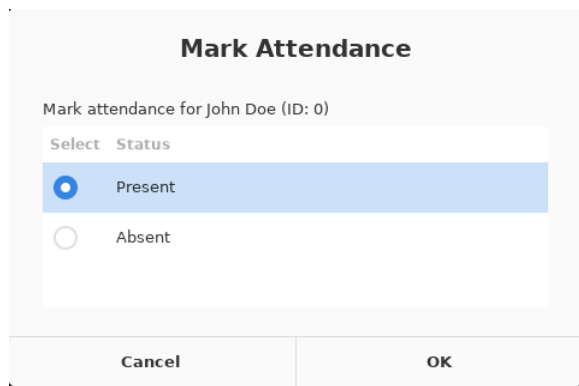


Fig 4.11 Radiolist to mark attendance

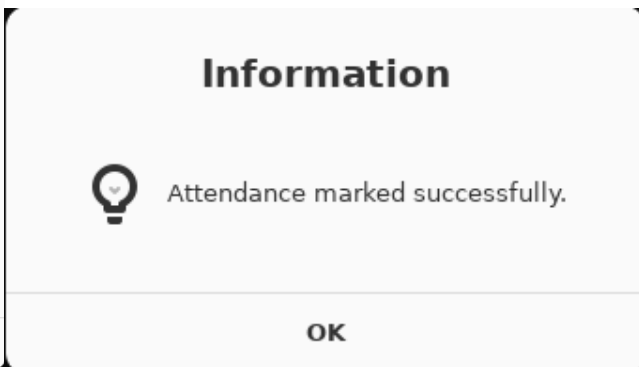
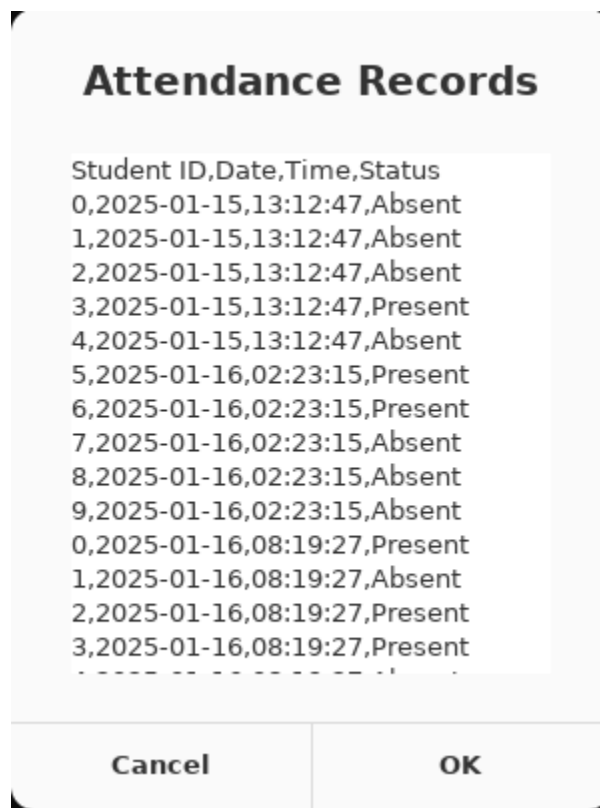
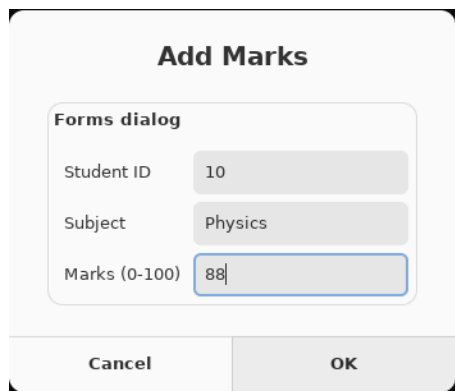


Fig 4.12 Success message after taking attendance



A screenshot of a web form titled "Add Marks". It contains three input fields: "Student ID" with the value "10", "Subject" with the value "Physics", and "Marks (0-100)" with the value "88". The "Marks" field is highlighted with a blue border. At the bottom, there are two buttons: "Cancel" and "OK".

Add Marks

Forms dialog

Student ID 10

Subject Physics

Marks (0-100) 88

Cancel OK

Fig 4.14 Form for adding marks

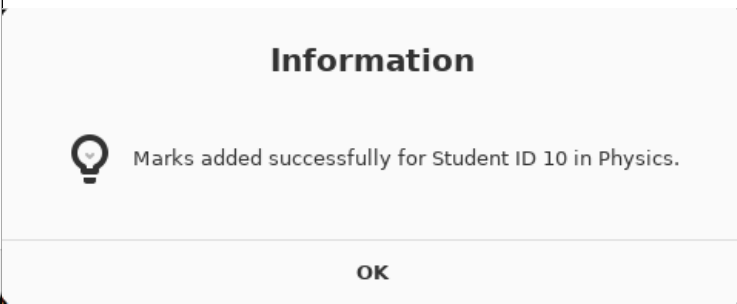


Fig 4.15 Success message after adding marks

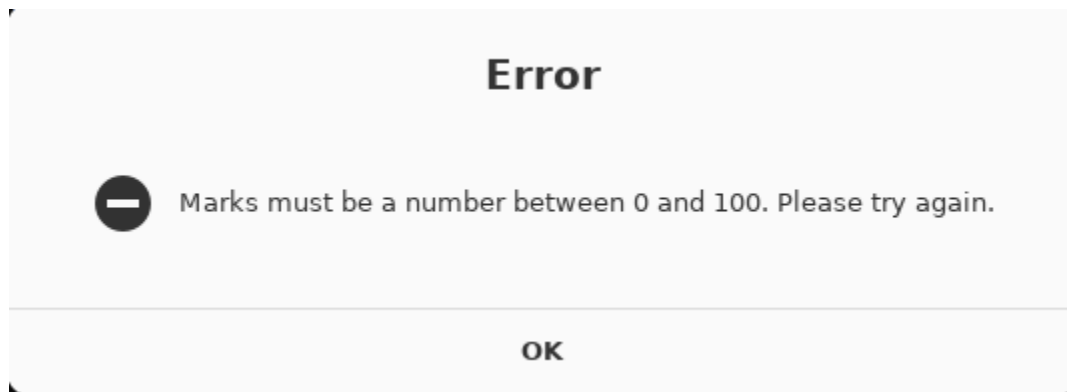


Fig 4.16 Error when marks are not in between 0 to 100

Figure 4.14 shows a form interface for adding marks, where the user can input subject-wise scores for a student. The structured format facilitates efficient data entry.

Figure 4.15 presents a success message confirming that the marks have been successfully added to the student's record. This acknowledgment assures the user that the operation was completed

Figure 4.16 displays an error message that appears when the entered marks are not within the valid range of 0 to 100. This ensures data accuracy and prevents invalid entries.

Marks Records		
Student ID	Subject	Marks
0	Maths	3
0	Physics	85
0	Chemistry	82
0	Computer Science	76
0	English	15
1	Maths	37
1	Physics	39
1	Chemistry	83
1	Computer Science	71
1	English	4
2	Maths	20
2	Physics	60
2	Chemistry	91
2	Computer Science	97
...
Cancel		OK

Fig 4.17 Marks Record

Top 5 Students by Average Marks			
Rank	ID	Name	Average Marks
1	10	Raj	88.00
2	7	Michael Johnson	71.40
3	2	Michael Johnson	69.60
4	8	Emily Davis	61.20
5	3	Emily Davis	61.20
Cancel		OK	

Fig 4.18 Top 5 Students

Figure 4.17 displays the marks record, presenting detailed subject-wise scores for all students in a structured tabular format. This view allows for easy evaluation of academic performance.

Figure 4.18 highlights the top 5 students based on their overall scores. The ranked list provides a quick overview of the highest achievers, promoting recognition and motivation among students.

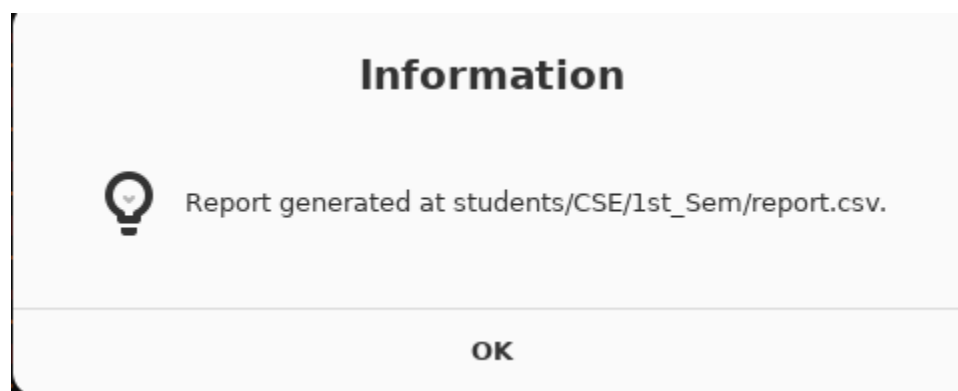


Fig 4.19 Message on generation of report

Figure 4.19 shows a message indicating the successful generation of the report. It informs the user about the location where the report has been saved, confirming that the process was completed without any issues.

Generated Report			
Student ID	Name	Attendance Percentage	Average Marks
0	John Doe	50.00	52.20
1	Jane Smith	0	46.80
2	Michael Johnson	50.00	69.60
3	Emily Davis	100.00	61.20
4	Daniel Brown	0	54.60
5	John Doe	100.00	44.40
6	Jane Smith	50.00	33.40
7	Michael Johnson	50.00	71.40
8	Emily Davis	50.00	61.20
9	Daniel Brown	50.00	46.20
10	Raj	100.00	88.00

Fig 4.20 View of generated report

Figure 4.20 displays the view of the generated report. It shows the detailed student information, including attendance percentages and average marks, in a formatted table, allowing users to review the data.

Chapter 5: Conclusion and Future Enhancement

5.1 Conclusion

This project has successfully implemented a Student Management System using Bash scripting, designed to efficiently handle and manage student data, including their personal details, attendance, marks, and reports. By using Zenity for the graphical user interface (GUI), it enables a user-friendly experience to select branches, semesters, and sections, as well as to add, view, and manage student records. The system provides essential features for marking attendance, adding marks, generating reports, and maintaining student records across different sections, making it a practical tool for educational institutions to manage student data and track academic progress.

5.2 Future Enhancement

The system can be further enhanced by integrating advanced features such as automated report generation based on specific criteria, like performance trends or comparative analysis. Adding a database backend, such as SQLite or MySQL, could improve the scalability and performance of data management, allowing for more extensive data handling and querying. Furthermore, the user interface can be enhanced with additional functionalities like student search, bulk data import/export, and email notifications for attendance or marks updates, making the system more robust and adaptable to modern educational environments.

References

- [1] Bash Scripting Tutorial - <https://www.gnu.org/software/bash/manual/>
- [2] Zenity Documentation - <https://help.gnome.org/users/zenity/stable/>
- [3] UNIX/Linux Command Manual - <https://man7.org/linux/man-pages/>
- [4] Introduction to Shell Scripting - <https://opensource.com/article/19/11/intro-shell-scripting>
- [5] Bash Reference Manual - <https://www.gnu.org/software/bash/manual/bash.html>
- [6] GNU Coreutils Documentation - <https://www.gnu.org/software/coreutils/manual/>
- [7] Zenity User Guide - <https://www.linuxlinks.com/zenity/>
- [8] Shell Scripting Wiki - https://en.wikipedia.org/wiki/Shell_script