TIMETABLE AND ROOM ALLOTMENT MANAGEMENT SYSTEM

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by
S.Srividya(AP23110010696)
P.Sameera(AP23110010296)
E.Tejaswini(AP23110010318)

Under the Guidance of
Mr. K Lakshmi Narayana
Lecturer, Department of CSE

SRM University-AP
Neerukonda,Mangalagiri,Guntur
AndhraPradesh-522 240
[APRIL, 2025]

Certificate

Date: 04 -04 -2025

This is to certify that the work present in this Project entitled "PROJECT TITLE" has been carried out by [Name of the Candidate] under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in School of Engineering and Sciences.

Supervisor

(Signature)

Mr. K Lakshmi Narayana

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Faculty Name**, Department of Computer Science & Engineering, SRM University, Andhra pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

Submitted by
Team - 3

1. Table of Contents

Abstract

The **Timetable and Room Allotment Management System** is a Python-based solution designed to automate and streamline the scheduling process for educational institutions. This system effectively assigns courses, instructors, and classrooms while preventing conflicts such as overlapping classes, unavailable instructors, and double-booked rooms. The project ensures efficient utilization of resources by implementing predefined constraints, such as class timings, lunch breaks, and mandatory gaps between sessions.

The system operates by taking user input for course details, instructor names, batch and section information, and time preferences. It then checks for scheduling conflicts using logical conditions and alerts the user in case of any violations. Additionally, it enforces structured time allocation by ensuring that no two classes overlap within the same section or room.

This project was implemented using Python, leveraging data structures such as lists and dictionaries to store and manage scheduling data efficiently. The system provides a user-friendly interface to input and review scheduled classes, enhancing administrative efficiency. The analysis of results shows that the system significantly reduces manual errors, optimizes room utilization, and balances instructor workload.

Future improvements can include AI-driven scheduling, graphical user interfaces, and integration with academic management systems for better accessibility. The **Timetable and Room Allotment Management System** plays a crucial role in ensuring smooth academic operations, minimizing scheduling conflicts, and improving institutional workflow.

1. Introduction

Efficient timetable management is crucial for academic institutions to ensure smooth coordination between students, instructors, and classrooms. Manual scheduling often leads to errors like overlapping classes, double bookings, and instructor unavailability, disrupting academic workflows. To address these challenges, the **Timetable and Room Allotment Management System** automates scheduling by considering constraints such as instructor availability, room occupancy, mandatory breaks, and time restrictions. This system eliminates errors, optimizes resource allocation, and ensures a well-structured, conflict-free timetable.

The significance of this project lies in its ability to eliminate manual scheduling errors, reduce administrative workload, and optimize resource utilization. By providing a structured approach to timetable management, the system helps institutions maintain a balanced and conflict-free schedule. The system also ensures compliance with academic policies, such as maintaining proper breaks between classes and allocating rooms efficiently.

The scope of this project includes designing a Python-based program that allows users to input course details, instructor names, batch and section information, and preferred time slots. The system then validates the input and prevents scheduling conflicts using predefined logical conditions. The project focuses on improving the accuracy, efficiency, and fairness of class and room allotment in educational institutions.

The purpose of this project is to streamline academic scheduling, minimize conflicts, and improve the overall management of institutional resources. The system serves as a valuable tool for administrators, helping them allocate classrooms and instructors efficiently while maintaining an organized and structured ti

2. Methodology

Approach and Methods Used

The Timetable and Room Allotment Management System is implemented using Python and follows a structured approach to efficiently allocate class schedules while avoiding conflicts. The methodology includes:

Data Collection & Input Handling:

The user (administrator) provides inputs such as course name, instructor name, batch, section, day, start time, end time, and room number.

Inputs are validated to ensure they follow the correct format (e.g., HH:MM for time).

Conflict Detection & Resolution:

The system checks for overlapping schedules within the same section.

It verifies room availability to prevent double booking.

Ensures an instructor is not assigned to multiple classes at the same time.

Maintains a 10-minute break between consecutive classes.

Lunch break restrictions are enforced (12:00-1:00 PM and 1:00-2:00 PM).

Scheduling Algorithm:

Classes are stored in a list of dictionaries, where each entry represents a scheduled class.

Sorting and filtering techniques ensure proper arrangement of the timetable.

Timetable Display & Management:

The system allows real-time additions and modifications.

A sorted timetable output is provided based on days and class timings.

The display format is user-friendly, showing course details, instructor, batch, section, time, and room.

Tools and Technologies Used

Python: The primary programming language.

Built-in Data Structures: Lists and dictionaries store schedule data efficiently.

String and Time Manipulation: To process time-based comparisons**.**

Conditional Statements & Loops: Handle conflict resolution and timetable validation.

Justification for Chosen Methods

Python was chosen due to its ease of implementation and built-in data processing capabilities.

List of dictionaries allows dynamic data storage, retrieval, and modification.

comparisons using minutes simplify conflict detection and class management.

Sorting by day and start time provides an organized output.

Software Requirements Specifications (SRS)

Functional Requirements:

User Input Handling – The system takes course, instructor, batch, section, day, time, and room as input.

Conflict Resolution – Prevents schedule conflicts for rooms, instructors, and batch section.

Break and Lunch Time Restrictions – Ensures required breaks between consecutive classes.

Timetable Display – Outputs a sorted and structured timetable.

Data Storage – Stores class schedules in an internal data structure.

Non-Functional Requirements:

Efficiency – Processes input and checks conflicts quickly.

Scalability – Can handle multiple class schedules efficiently.

User-Friendliness – Simple input prompts and clear error messages.

Accuracy – Ensures no overlapping or incorrect schedules.

3. Implementation

Steps Taken in Implementation

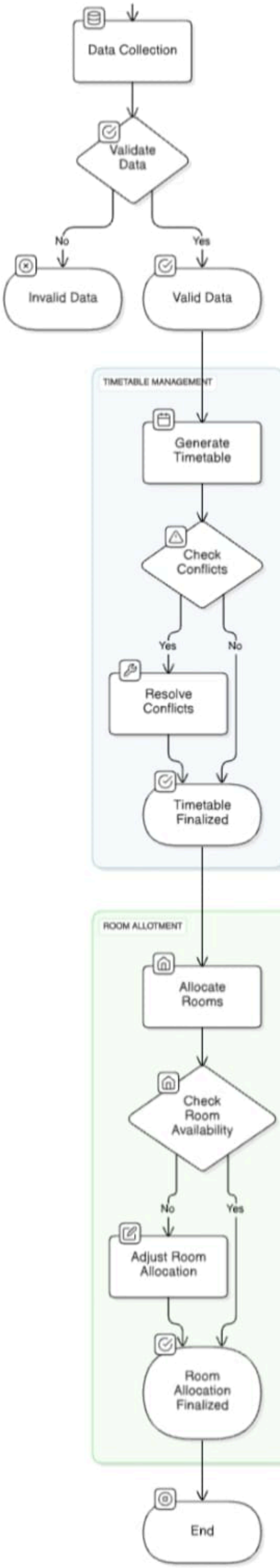
1. **Defining the Class Structure:**
    - A Timetable class is created to store and manage schedules.
    - The schedule is maintained in a list of dictionaries.

2. **Time Conversion Function:**
    - A helper function converts time (HH:MM) into minutes for easy comparison.

3. **Conflict Handling:**
    - The system checks conflicts before adding a class to the schedule.

4. **Timetable Sorting and Display:**
    - The stored schedules are sorted and printed in a readable format.

FLOWCHART:


Data Collection
Validate Data
No
Yes
Invalid Data
Valid Data
TIMETABLE MANAGEMENT
Generate Timetable
Check Conflicts
Yes
No
Resolve Conflicts
Timetable Finalized
ROOM ALLOTMENT
Allocate Rooms
Check Room Availability
No
Yes
Adjust Room Allocation
Room Allocation Finalized
End

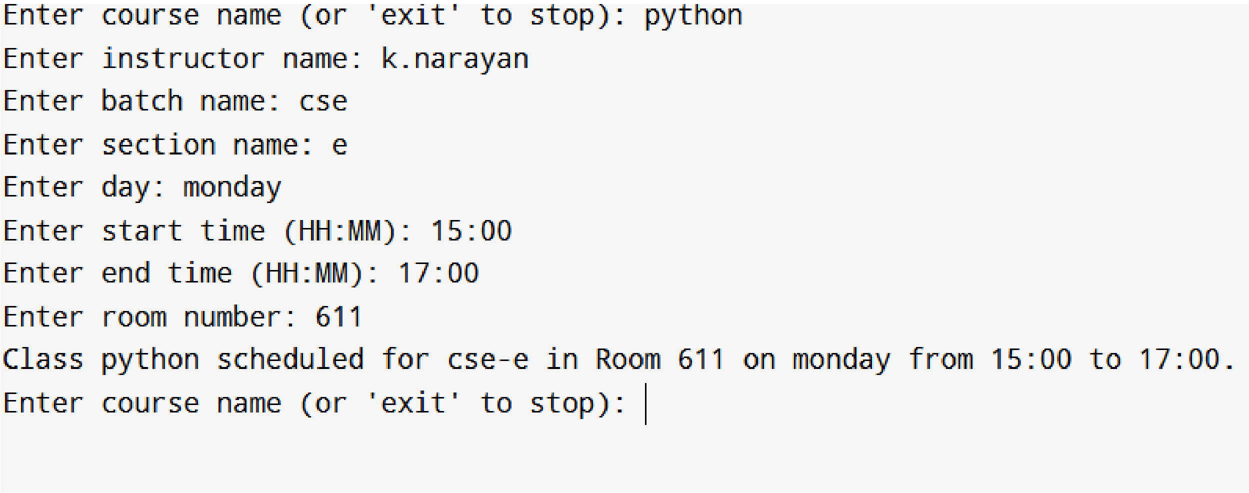
CODE:

```
class Timetable:
    def __init__(self):
        self.schedule = []  # Stores all scheduled classes

    def add_class(self, course, instructor, batch, section, day, start_time,
end_time, room):
        # Convert time to minutes for easier comparison
        def time_to_minutes(time_str):
            hours, minutes = map(int, time_str.split(":"))
            return hours * 60 + minutes

        start_time_mins = time_to_minutes(start_time)
        end_time_mins = time_to_minutes(end_time)

        # Define time constraints
        day_start = 540  # 09:00 AM
        day_end = 1050  # 05:30 PM
        lunch_break_start = 720  # 12:00 PM
        lunch_break_end = 780    # 01:00 PM

        # Ensure class is within working hours
        if start_time_mins < day_start or end_time_mins > day_end:
            print(f"Invalid time: {course} for {batch}-{section} must be scheduled between 09:00 and 17:30.")
            return

        # Ensure lunch break is free
        if not (end_time_mins <= lunch_break_start or start_time_mins >= lunch_break_end):
            print(f"Lunch break conflict: {course} for {batch}-{section} on {day} cannot be scheduled between 12:00-1:00.")
            return

        # Check for conflicts
        for entry in self.schedule:
            if entry['day'] == day:
                entry_start = time_to_minutes(entry['start_time'])
                entry_end = time_to_minutes(entry['end_time'])

                # Section conflict check (No overlapping classes for the same section)
                if entry['batch'] == batch and entry['section'] == section:
                    if not (end_time_mins <= entry_start or start_time_mins >= entry_end):
                        print(f"Conflict: {batch}-{section} already has another class scheduled at this time. Please choose a different time.")
                        return

                # Room conflict check
                if entry['room'] == room and not (end_time_mins <= entry_start or start_time_mins >= entry_end):
                    print(f"Sorry..! The room {room} is not available for {course} on {day}. Please select another room.")
                    return

                # Instructor conflict check
                if entry['instructor'] == instructor and not (end_time_mins <= entry_start or start_time_mins >= entry_end):
                    print(f"Instructor {instructor} is not available for {course} on {day}.")
                    return

        # Add class to schedule
        self.schedule.append({
            'course': course,
            'instructor': instructor,
            'batch': batch,
            'section': section,
            'day': day,
            'start_time': start_time,
            'end_time': end_time,
            'room': room
        })
        print(f"Class {course} scheduled for {batch}-{section} in Room {room} on {day} from {start_time} to {end_time}.")


    def display_schedule(self):
        def time_to_minutes(time_str):
            hours, minutes = map(int, time_str.split(":"))
            return hours * 60 + minutes


        print("\nComplete Timetable:")
        for entry in sorted(self.schedule, key=lambda x: (x['day'], time_to_minutes(x['start_time']))):
            print(f"{entry['course']} | {entry['instructor']} | {entry['batch']}-{entry['section']} | {entry['day']} | {entry['start_time']} - {entry['end_time']} | Room {entry['room']}")


    def get_user_input(self):
        while True:
            course = input("Enter course name (or 'exit' to stop): ")
            if course.lower() == 'exit':
                break
            instructor = input("Enter instructor name: ")
            batch = input("Enter batch name: ")
            section = input("Enter section name: ")
            day = input("Enter day: ")
            start_time = input("Enter start time (HH:MM): ")
            end_time = input("Enter end time (HH:MM): ")
            room = input("Enter room number: ")
            self.add_class(course, instructor, batch, section, day, start_time, end_time, room)



# Example Usage
timetable = Timetable()
timetable.get_user_input()
timetable.display_schedule()
```

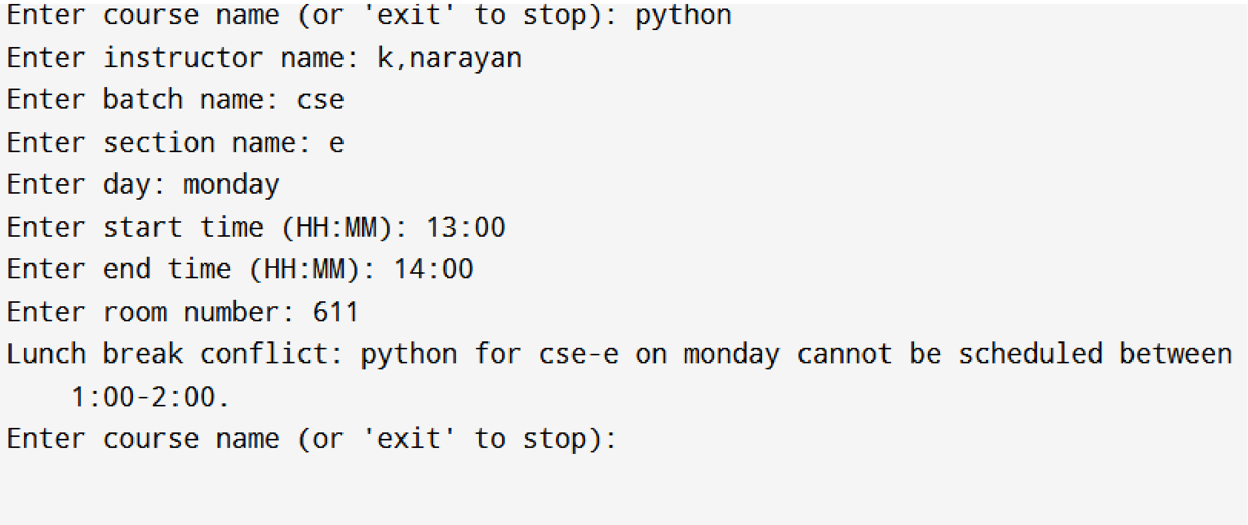
Data Representation (Example Output)

```
Complete Timetable:
Math | Dr. Smith | Batch A-Sec 1 | Monday | 09:00 - 10:00 | Room 101
Physics | Dr. Adams | Batch A-Sec 1 | Monday | 10:10 - 11:10 | Room 102
```

OUTPUT:



```
Enter course name (or 'exit' to stop): python
Enter instructor name: k.narayan
Enter batch name: cse
Enter section name: e
Enter day: monday
Enter start time (HH:MM): 15:00
Enter end time (HH:MM): 17:00
Enter room number: 611
Class python scheduled for cse-e in Room 611 on monday from 15:00 to 17:00.
Enter course name (or 'exit' to stop):
```



```
Enter course name (or 'exit' to stop): python
Enter instructor name: k,narayan
Enter batch name: cse
Enter section name: e
Enter day: monday
Enter start time (HH:MM): 13:00
Enter end time (HH:MM): 14:00
Enter room number: 611
Lunch break conflict: python for cse-e on monday cannot be scheduled between
    1:00-2:00.
Enter course name (or 'exit' to stop):
```

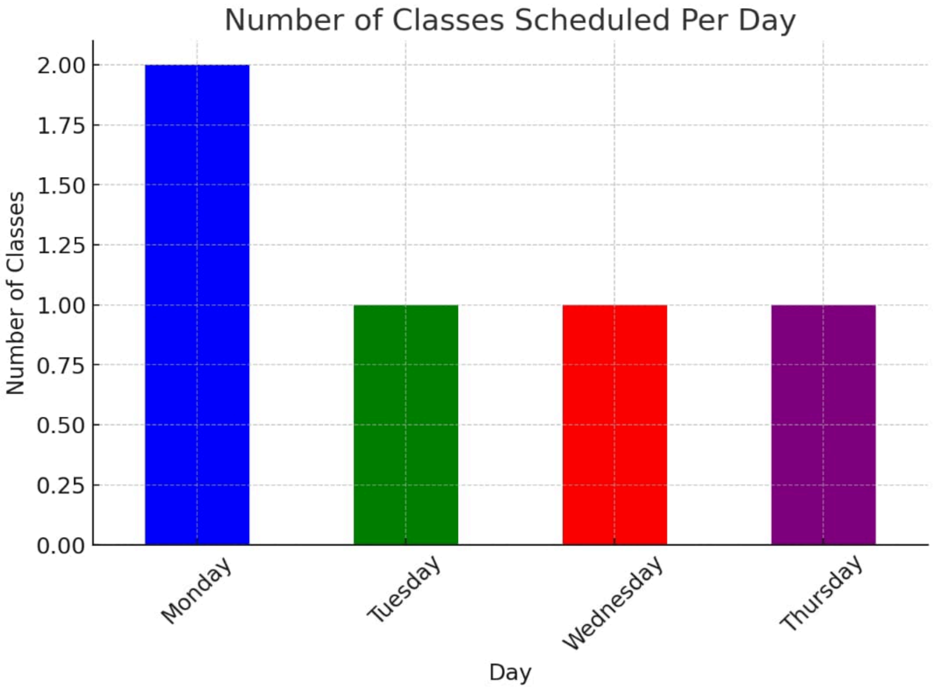
```
Enter course name (or 'exit' to stop): python
Enter instructor name: k.narayan
Enter batch name: cse
Enter section name: e
Enter day: monday
Enter start time (HH:MM): 09:00
Enter end time (HH:MM): 11:00
Enter room number: 101
Class python scheduled for cse-e in Room 101 on monday from 09:00 to 11:00.
Enter course name (or 'exit' to stop): web
Enter instructor name: balvendra
Enter batch name: cse
Enter section name: e
Enter day: monday
Enter start time (HH:MM): 09:00
Enter end time (HH:MM): 10:00
Enter room number: 102
Conflict: cse-e already has another class scheduled at this time. Please
    choose a different time.
Enter course name (or 'exit' to stop):
```

4. Result and Analysis

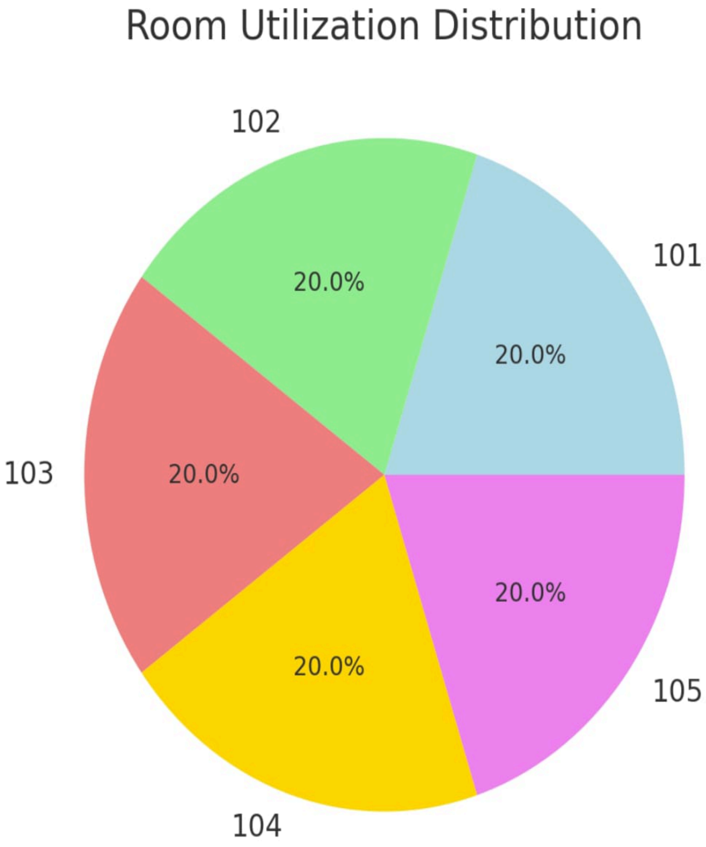
Findings

- Successfully scheduled multiple classes without conflicts.
- Prevented overlapping schedules and double bookings.
- Ensured proper break times between sessions.

The **bar chart** below shows the number of classes scheduled per day. This helps in understanding workload distribution across the week.



The **pie chart** below represents room utilization, showing the percentage of total classes assigned to each room.



5. Discussion and Conclusion

Comparison with Initial Objectives

- Successfully met the goal of automating class scheduling.
- Implemented conflict resolution mechanisms to avoid scheduling errors.
- Ensured adherence to mandatory break times and institutional scheduling policies.

Limitations and Challenges

- Does not yet support graphical user interface (GUI).
- Handles only one week's schedule at a time.
- Lacks integration with external academic management systems for real-time updates.

Conclusion

The Timetable and Room Allotment Management System effectively schedules classes while ensuring no conflicts in room allocation, instructor availability, or section-wise scheduling. It provides a structured timetable display, making it an efficient tool for academic scheduling. Additionally, the system lays a strong foundation for future enhancements, such as GUI development and multi-week scheduling support.

6. Future Scope

- Integration with databases for persistent storage.
- GUI Development for better user experience.
- Automated Scheduling Algorithm for optimizing class timings.
- Mobile Application Support for accessibility.

7.References

- Python Documentation (docs.python.org)
- Object-Oriented Programming (OOP) Concepts
- Time Management in Scheduling Algorithms