HOSTEL MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

GODFREY T R (2303811710421047)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

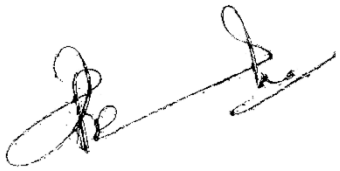
JUNE- 2025

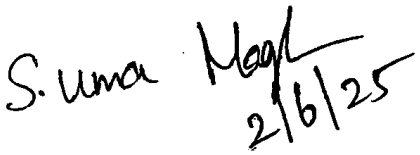
K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

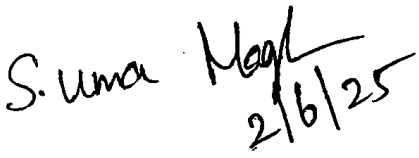
Certified that this project report on **"Hostel Management System"** is the bonafide work of **GODFREY T R (2303811710421047)** who carried out the project work during the academic year 2024 - 2025 under my supervision.





SIGNATURE	**SIGNATURE**
Dr.A.Delphin Carolina Rani, M.E.,Ph.D.,	Ms. S. Uma Mageshwari, M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSISTANT PROFESSOR
Department of CSE	Department of CSE
K.Ramakrishnan College of Technology (Autonomous)	K.Ramakrishnan College of Technology (Autonomous)
Samayapuram–621112.	Samayapuram–621112.

Submitted for the viva-voce examination held on 02/06/2025 .



INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

I declare that the project report on **"HOSTEL MANAGEMENT SYSTEM"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS.**

Signature

GODFREY T R

Place : Samayapuram

Date : 02/06/2025

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. A. DELPHIN CAROLINA RANI, M.E.,Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Ms. S. UMA MAGESHWARI, M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render my sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express my special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To serve the society by offering top-notch technical education on par with global standards

MISSION OF THE INSTITUTION

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of the industry and society.
- Be an institute with world class research facilities
- Be an institute nurturing talent and enhancing the competency of students to transform them as all-round personality respecting moral and ethical values

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

ABSTRACT

The **Hostel Management System** is a desktop application built with Python's Tkinter and MySQL, designed to simplify and digitize hostel administration. It features role-based access for **wardens** and **hostellers**, enabling functions like attendance tracking, outpass requests, and user management.

Wardens can view attendance summaries, approve outpass requests, and filter hosteller data by various criteria. Hostellers can mark attendance, submit and track outpass requests, and review their attendance history.

The system ensures secure login and registration, supports real-time database operations, and promotes efficient communication between hostel staff and residents. With its user-friendly interface and modular design, this project offers a practical solution for managing hostels in academic institutions.

ABSTRACT WTTH POS AND PSOS MAPPING

CO 5: BUILD DATABASE MANAGEMENT SYSTEMSAPPLICATIONS FOR SOLVING REAL - TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The **Hostel Management System** is a desktop application built with Python's Tkinter and MySQL, designed to simplify and digitize hostel administration. It features role-based access for **wardens** and **hostellers**, enabling functions like attendance tracking, outpass requests, and user management. Wardens can view attendance summaries, approve outpass requests, and filter hosteller data by various criteria. Hostellers can mark attendance, submit and track outpass requests, and review their attendance history. The system ensures secure login and registration, supports real-time database operations, and promotes efficient communication between hostel staff and residents. With its user-friendly interface and modular design, this project offers a practical solution for managing hostels in academic institutions.	**PO1 -3** **PO2 -3** **PO3 -3** **PO4 -3** **PO5 -3** **PO6 -3** **PO7 -3** **PO8 -3** **PO9 -3** **PO10 -3** **PO11-3** **PO12 -3**	**PSO1 -3** **PSO2 -3** **PSO3 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO.
	ABSTRACT	ix
	LIST OF FIGURES	xi
	LIST OF ABBREVIATIONS	xii
1	**INTRODUCTION**	1
	1.1 OBJECTIVE	1
	1.2 OVERVIEW	1
	1.3 SQL AND DATABASE CONCEPTS	2
	1.3.1 Basic SQL Concepts	3
	1.3.2 Role of SQL in Hostel Management System	3
	1.3.3 Database Design and Structure	4
	1.3.4 SQL Queries in Application Logic	4
	1.3.5 Authentication and Security	5
	1.3.6 Transactions and ACID Properties	5
2	**PROJECT METHODOLOGY**	6
	2.1 PROPOSED WORK	6
	2.2 BLOCK DIAGRAM	7
3	**MODULE DESCRIPTION**	8
	3.1 USER LOGIN MODULE	8
	3.2 WARDEN DASHBOARD MODULE	8
	3.3 OUTPASS AUTHORIZATION MODULE	8
	3.4 STUDENT DASHBOARD MODULE	9
	3.5 ATTENDANCE/CHECK-IN MODULE	9
4	**CONCLUSION & FUTURE SCOPE**	10
5	**APPENDIX A: SOURCE CODE**	13
	APPENDIX B: SCREENSHOTS	16
	REFERENCES	18

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
2.1	BLOCK DIAGRAM	7

LIST OF ABBREVIATIONS

ABBREVIATION		FULL FORM
GUI	-	GRAPHICAL USER INTERFACE
TKINTER	-	TK INTERFACE (PYTHON'S GUI TOOLKIT)
SQL	-	STRUCTURED QUERY LANGUAGE
MYSQL	-	MY STRUCTURED QUERY LANGUAGE (DATABASE SYSTEM)
API	-	APPLICATION PROGRAMMING INTERFACE
DB	-	DATABASE
CRUD	-	CREATE, READ, UPDATE, DELETE
CLI	-	COMMAND LINE INTERFACE

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The primary objective of the Hostel Management System is to develop a software solution that simplifies and automates the day-to-day activities involved in managing hostel operations. Traditionally, hostel management has relied heavily on manual processes such as maintaining registers for attendance, manually processing outpass requests, and storing hosteller records in physical files. These methods are often time-consuming, prone to human error, and difficult to maintain or retrieve when needed. This project seeks to address these challenges by introducing a digital solution that leverages Python for the application logic, Tkinter for the graphical user interface (GUI), and MySQL for secure and reliable data storage.

The system is designed to cater to two primary user roles: the hosteller and the warden. Hostellers can use the application to mark their attendance, request outpasses for leaving the hostel premises, and view their personal attendance history. On the other hand, wardens can log in to the system to manage student information, view attendance records, filter student data based on department or hostel, and review and approve or reject outpass requests. By implementing role-based access, the system ensures that users interact only with the features relevant to their responsibilities, thereby enhancing usability and security.

Overall, the objective is to provide a centralized, user-friendly, and efficient platform that facilitates smooth communication between students and hostel authorities, reduces administrative workload, and improves the accuracy and accessibility of hostel records. Through this project, institutions can modernize their hostel management practices and create a more transparent and accountable system for all stakeholders involved.

1.2 OVERVIEW

The Hostel Management System is a desktop-based application designed to

streamline and digitize the various processes involved in the administration of a student hostel. This project leverages Python for its core functionality, utilizing the Tkinter library to build an interactive graphical user interface and MySQL to handle the backend database operations. The system was developed to address the inefficiencies of traditional paper-based hostel management, offering a modern and intuitive platform for both students and wardens to manage daily hostel-related tasks.

The application is structured around two primary user roles—hosteller and warden—each with distinct access privileges and functionalities. Hostellers are provided with features such as login authentication, attendance marking, outpass request submission, and the ability to view their attendance history. This allows them to independently manage and track their hostel activities. Meanwhile, wardens have a separate login interface that enables them to oversee and manage student records, review attendance data, process outpass requests, and filter students by department or hostel name.

The project is divided into multiple modules to promote modularity and maintainability. These include user authentication, attendance management, outpass handling, and data filtering—all integrated with a MySQL database to ensure data persistence and consistency. The interface is designed to be user-friendly and visually clear, making it accessible to users who may not be technically inclined.

By digitizing and automating key administrative functions, the Hostel Management System not only saves time and resources but also improves the overall efficiency and transparency of hostel operations. The system reduces the likelihood of errors, ensures better record-keeping, and enhances communication between students and hostel staff, thus contributing to a better-managed hostel environment.

1.3 SQL AND DATABASE CONCEPTS

1.3.1 Basic SQL Concepts

Structured Query Language (SQL) is the foundational language used to interact with relational databases. In the Hostel Management System, SQL

facilitates the creation, maintenance, and querying of a structured data layer. The key components include:

- Data Definition Language (DDL): Used for defining the database schema. Commands like CREATE TABLE, ALTER TABLE, and DROP TABLE help define the structure of tables, data types, and relationships.
- Data Manipulation Language (DML): Used for handling data. Commands like INSERT, UPDATE, DELETE, and SELECT allow data to be added, modified, removed, and retrieved efficiently.
- Constraints: SQL provides mechanisms to enforce data integrity through constraints such as PRIMARY KEY for uniqueness, FOREIGN KEY for relational integrity, and NOT NULL to prevent empty entries in essential fields.
- Joins: SQL supports operations such as INNER JOIN, LEFT JOIN, and RIGHT JOIN to combine records from two or more tables based on related columns.
- Transactions: Enable the grouping of multiple SQL statements into a single unit of work. Commands like COMMIT, ROLLBACK, and SAVEPOINT ensure that data changes occur consistently and reliably.

1.3.2 Role of SQL in Hostel Management System

In this project, SQL serves as the bridge between the application logic and persistent data storage. It enables reliable management of critical hostel operations such as:

- Student registration and authentication
- Attendance tracking and reporting
- Outpass request and approval workflows
- Administrative dashboard access for wardens

By using MySQL, a robust open-source RDBMS, the project benefits from features like ACID compliance, indexing, security, scalability, and support for concurrent access.

1.3.3 Database Design and Structure

The design of the database is central to ensuring performance and maintainability. Key design features include:

- Normalized Tables: Data is organized in normalized tables to eliminate redundancy. For example, students, attendance, and outpass_requests are separate but related through foreign keys.
- Primary Keys: Unique identifiers such as regno in the students table and id in the outpass_requests table ensure each record is distinct and easily accessible.
- Foreign Keys: Used to establish relationships. For instance, the attendance table references the students table via regno, ensuring data consistency between the records.
- Constraints and Defaults: Constraints like NOT NULL and default values for fields ensure data accuracy and avoid inconsistencies due to missing or invalid data.

1.3.4 SQL Queries in Application Logic

SQL queries are embedded in the backend code (Python + mysql.connector) to interact dynamically with the database. Key operations include:

- Inserting Records: When a student registers or submits an outpass, an INSERT INTO query is executed.
- Reading Data: SELECT queries are used to display dashboards, view attendance history, and track outpass statuses.
- Updating Records: A warden may use UPDATE queries to change outpass approval status or correct student details.
- Deleting Data: Administrators may use DELETE queries for cleanup or data correction.

1.3.5 Authentication and Security

Security is a priority in managing user data. Measures include:

- Hashed Passwords (optional extension): Passwords can be stored using secure hashing algorithms (like SHA-256 or bcrypt) instead of plaintext to enhance protection.
- Parameterized Queries: All SQL commands use placeholders (e.g., %s) to prevent SQL injection attacks by ensuring that user input is safely escaped.
- Role-Based Access: The system distinguishes between students and wardens by checking credentials and loading role-specific interfaces.
- Session Management: Sessions can be used to track authenticated users, maintaining their state securely across the application workflow.

1.3.6 Transactions and ACID Properties

Transactions in MySQL ensure that operations are executed with the ACID properties:

- Atomicity: Ensures that all steps in a transaction are completed successfully; otherwise, the entire operation is rolled back.
- Consistency: The database remains in a valid state before and after the transaction.
- Isolation: Transactions are executed independently, preventing interference from other concurrent operations.
- Durability: Once a transaction is committed, the data is saved permanently even in case of a system crash.

CHAPTER 2

PROJECT METHODOLOGY

2.1 PROPOSED WORK

The proposed Hostel Management System is a role-based software solution designed to digitize and streamline the administrative operations of a student hostel. Developed using Python's Tkinter library for the front-end and MySQL for the backend database, the system provides a user-friendly and efficient interface for managing hostellers, tracking attendance, and processing outpass requests. It supports two primary roles: hosteller and warden, each with distinct dashboards and permissions tailored to their responsibilities.

Hostellers can register through a detailed form that captures personal, academic, and contact details. Once registered, they can log in to manage their attendance by marking themselves as present, absent, or on leave using an interactive calendar-based interface. Additionally, they can submit outpass requests specifying the duration and reason, which are then reviewed by the warden. Hostellers also have access to their outpass request history for reference and transparency.

Wardens, on the other hand, have administrative privileges allowing them to view attendance records, approve or reject outpass requests, and access a filtered list of hostellers based on department, year, or hostel name. The system also features birthday alerts, notifying wardens of hostellers who have birthdays on the current day, adding a personalized touch to student management. Attendance summaries are automatically generated monthly, giving wardens insights into overall attendance trends and enabling data-driven decision-making.

Overall, the system replaces traditional paper-based records with a centralized digital platform that enhances efficiency, accuracy, and accessibility. It minimizes manual effort, reduces the chances of errors, and improves communication between students and wardens in Figure 2.1 . The modular structure of the application ensures ease of maintenance and future scalability, allowing additional features like password

encryption, QR code attendance, or mobile integration to be added with minimal disruption.

2.2 BLOCK DIAGRAM

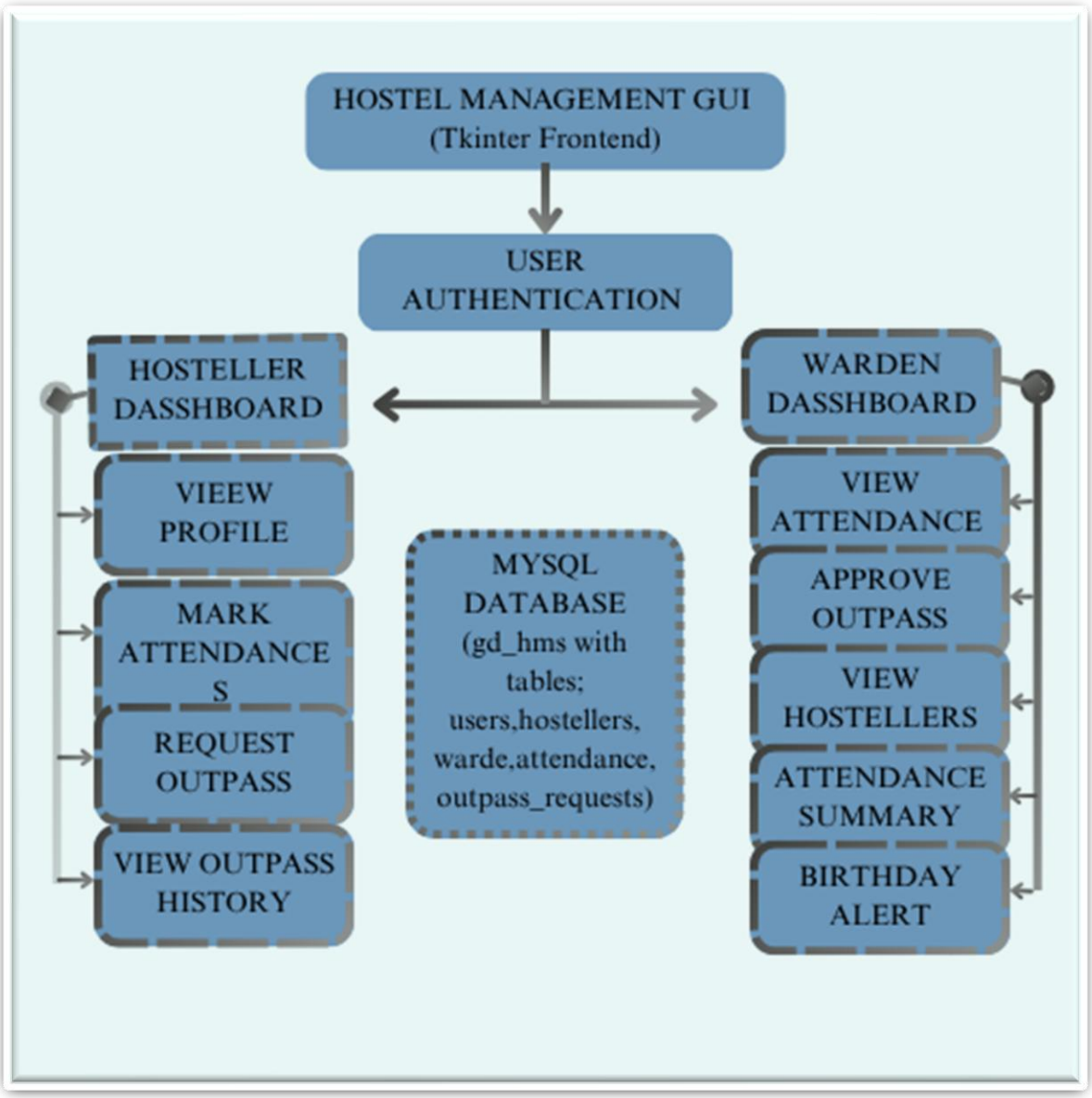


Fig 2.1 BLOCK DIAGRAM

CHAPTER 3

MODULE DESCRIPTION

3.1. USER LOGIN MODULE

The User Login Module facilitates the secure login process for the hostel management system. When users attempt to log in, they are prompted to provide a username and password. Upon successful authentication, the system checks the user's role (either "warden" or "hosteller") to determine which dashboard to open. If the login details are correct, the system will either direct the user to the hosteller dashboard or the warden dashboard. Additionally, users can register themselves if they don't already have an account, and there is an option to switch between light and dark themes for the interface. If invalid credentials are entered, the system will show an error message to the user.

3.2. WARDEN DASHBOARD MODULE

The Warden Dashboard Module acts as the primary control interface for hostel wardens, offering a streamlined and efficient way to oversee hostel operations. Upon logging in, the warden is welcomed with a personalized greeting and a display of the current month's calendar, which includes important events, public holidays, and birthdays. The dashboard is designed for ease of navigation, with each major function represented by a clearly labeled button. These functions include access to student attendance records, the ability to approve or reject outpass requests, and a complete list of current hostellers. The warden can also filter student information based on specific criteria such as room number, academic year, or department. The system ensures that when a particular task is selected—such as managing outpasses—a separate window opens to display relevant data fetched from the database. This modular structure helps wardens manage multiple responsibilities efficiently and with minimal confusion.

3.3. OUTPASS APPROVAL MODULE

The Outpass Approval Module is integrated directly with the warden's dashboard and serves the purpose of handling student requests to leave the hostel premises

temporarily. This module presents a structured table containing all pending outpass requests, with detailed information about each request including the hosteller's name, room number, department, requested leave dates, and reason for the leave. The warden can assess each request and choose to either approve or reject it, often with the option to add a remark, especially in case of rejection. Once a decision is made, the request's status is updated in the database, and the student is notified accordingly through the system. The interface allows for sorting and filtering of requests, making it easier for the warden to manage high volumes of requests effectively. All actions taken within this module are recorded, ensuring transparency and accountability in the approval process.

3.4. STUDENT DASHBOARD MODULE

The Student Dashboard Module provides hostellers with a set of tools to manage their daily tasks. After logging in, the student is presented with a welcome screen, followed by several options including the ability to view personal information, manage attendance, request an outpass, and view their outpass history. The student can mark their attendance for the current day, request time off for specific dates (outpass requests), and review any previous outpass requests they've made, including their approval status. The dashboard also provides the option for the student to log out at any time. This module enhances the student's experience by offering quick access to their data and facilitating key tasks such as attendance marking and outpass management.

3.5. ATTENDANCE/CHECK-IN MODULE

The Student Dashboard Module offers hostellers a user-friendly and comprehensive interface to manage their day-to-day hostel-related tasks. When students log in, they are greeted with a personalized message followed by access to several interactive options. These include the ability to view and update their personal profile, check their attendance, request an outpass, and review their outpass history.

CHAPTER 4

CONCLUSION AND FUTURE ENHANCEMENT

4.1 CONCLUSION

In this project, a hostel management system has been designed and developed with the aim of improving efficiency, reducing manual efforts, and providing a seamless experience for both wardens and students. The system incorporates key modules such as the user login module, warden dashboard, outpass approval system, student dashboard, and attendance management. Through these modules, the system ensures that the daily operations of hostel life, including attendance tracking and outpass management, are streamlined and automated.

The system has successfully addressed the primary challenges of hostel management by enabling real-time updates, easy access to records, and the ability to handle requests with efficiency. By using a simple yet effective user interface, the system facilitates smooth communication between wardens and students, and ensures that all operations are executed in a timely and organized manner. This project has demonstrated the potential of using a digital approach to transform traditional hostel management practices.

Overall, the system is a valuable tool for both students and hostel administrators, and it effectively reduces the administrative workload while improving accuracy and convenience.

4.2 FUTURE ENHANCEMENTS

Although the current version of the hostel management system has provided significant improvements to the management process, there are still several opportunities for enhancement in the future to further improve the system's functionality, usability, and overall effectiveness:

Mobile Application Integration

One of the most impactful future enhancements would be the development of a mobile application for both students and wardens.

This would allow users to access their dashboards and manage their tasks on the go, making the system more accessible and convenient. Push notifications for attendance, outpass approvals, and reminders for important deadlines could further improve user engagement.

Real-Time Notifications

The inclusion of real-time notifications would allow students and wardens to receive immediate updates about attendance status, outpass approvals, and any other relevant information. This feature could be extended to allow for automated email and SMS notifications, ensuring that users stay informed even when they are not actively logged into the system.

Advanced Analytics and Reporting

To provide more value to wardens and hostel administrators, the system could include advanced analytics and reporting features. These would allow the warden to generate detailed reports on student attendance, outpass history, and other metrics that could help improve decision-making and operations. The reports could be customized by date, student, or department to give a deeper insight into hostel activities.

Integration with Payment Systems

Another future enhancement could be the integration of a payment system for hostel fees, fines, or other charges. Students could make payments through the system, and wardens could track the payment status. The integration of online payment gateways would simplify financial transactions and enhance the convenience of managing hostel fees.

Security and Privacy Features

As the system stores sensitive data, including personal and attendance information, future versions of the system could incorporate enhanced security features. This could include multi-factor authentication for login, data encryption, and regular security audits to ensure the safety of user information and prevent unauthorized access.

AI-Based Attendance Monitoring

The future version of the system could incorporate AI-based technologies, such as facial recognition, for automated attendance marking.

This would eliminate the need for students to manually mark attendance, saving time and ensuring accuracy. The AI system could identify students based on their facial features and update attendance records in real time.

User Customization and Personalization

Allowing users to further customize their dashboards and preferences would improve the user experience. Features such as theme personalization, custom notification settings, and personalized reporting could help users feel more in control of their experience within the system.

Better Integration with Hostel Management Tasks

The system could also be extended to include features for managing hostel maintenance requests, room allocation, inventory management, and meal plans. These additions would allow the system to handle a broader range of hostel-related operations, making it an all-in-one solution for hostel management.

In conclusion, while the hostel management system has achieved its intended objectives, continuous improvements and additions will ensure that it remains relevant, efficient, and adaptable to future needs. With the right enhancements, this system could transform hostel management into an even more efficient and user-friendly process, benefiting both students and administrators alike.

APPENDIX A

1.SQL code

```
-- Create the main database for the hostel management system
CREATE DATABASE IF NOT EXISTS gd_hms;
USE gd_hms;

-- Table: attendance
-- Stores attendance records for hostellers
CREATE TABLE attendance (
    id INT(11) NOT NULL AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    date DATE NOT NULL,
    status ENUM('P', 'A', 'L') NOT NULL,  -- P = Present, A = Absent, L = Leave
    PRIMARY KEY (id),
    INDEX (username)
);

-- Table: hostellers
-- Contains basic information about hostellers
CREATE TABLE hostellers (
    hosteller_id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    birth_date DATE DEFAULT NULL,
    email VARCHAR(100) DEFAULT NULL,
    phone_number VARCHAR(15) DEFAULT NULL,
    room_number INT(11) DEFAULT NULL,
    hostel_name VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (hosteller_id)
);
-- Table: outpass_requests
-- Manages requests made by students to leave the hostel
CREATE TABLE outpass_requests (
    request_id INT(11) NOT NULL AUTO_INCREMENT,
    user_id INT(11) DEFAULT NULL,
    request_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    start_date DATE DEFAULT NULL,
    end_date DATE DEFAULT NULL,
    reason TEXT DEFAULT NULL,
    status ENUM('Pending', 'Approved', 'Rejected') DEFAULT 'Pending',
    username VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY (request_id),
    INDEX (user_id)
);
-- Table: users
-- Stores login credentials and profile information of all users
CREATE TABLE users (
    user_id INT(11) NOT NULL AUTO_INCREMENT,
```

```
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('warden', 'hosteller') NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) DEFAULT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone_number VARCHAR(15) DEFAULT NULL,
    year_of_study INT(11) DEFAULT NULL,
    department VARCHAR(100) DEFAULT NULL,
    hostel_name VARCHAR(50) DEFAULT NULL,
    room_number VARCHAR(10) DEFAULT NULL,
    DOB DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    PRIMARY KEY (user_id)
);
-- Table: warden
-- Contains login and contact details of wardens
CREATE TABLE warden (
    warden_id INT(11) NOT NULL AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) DEFAULT NULL,
    phone VARCHAR(20) DEFAULT NULL,
    PRIMARY KEY (warden_id)
);
```

2. Python Code

```
#HOMEPAGE.PY
import tkinter as tk
from tkinter import messagebox

def open_login():
    import login  # Ensure login.py contains open_login_window()
    login.open_login_window()

def show_homepage():
    root = tk.Tk()
    root.title(" 🏠 Hostel Management System")
    root.geometry("800x600")
    root.config(bg="#e6f2ff")

    # --- Navbar ---
    navbar = tk.Frame(root, bg="#ccddff", pady=10)
    navbar.pack(fill='x')

    tk.Label(navbar, text=" 🏠 Hostel Management System", font=("Arial", 16, "bold"),
bg="#ccddff").pack(side="left", padx=20)
```

```
    nav_buttons = [
        ("Home", lambda: messagebox.showinfo("Home", "You are on the Home page.")),
        ("Features", lambda: messagebox.showinfo("Features", "See the features below.")),
        ("Copy Right", lambda: messagebox.showinfo("Copy Right", "GD 2025."))
    ]

    for text, cmd in nav_buttons:
        tk.Button(navbar, text=text, command=cmd, bg="white", font=("Arial",
12)).pack(side="right", padx=10)

    # --- Hero Section ---
    tk.Label(root, text="Welcome to Hostel Management System", font=("Arial", 20,
"bold"), bg="#e6f2ff").pack(pady=30)
    tk.Label(root, text="Efficiently manage hostelers, attendance, and more.",
font=("Arial", 14), bg="#e6f2ff").pack(pady=10)
    tk.Button(root, text="Get Started", command=open_login, font=("Arial", 12),
bg="#3399ff", fg="white", padx=20, pady=5).pack(pady=10)

    # --- Features Section ---
    tk.Label(root, text="Our Features", font=("Arial", 16, "bold"),
bg="#e6f2ff").pack(pady=20)

    features = ["Attendance Tracking", "Out Pass Management", "Warden Notifications",
"Birthday Alerts"]
    for feat in features:
        tk.Label(root, text=feat, font=("Arial", 12), bg="#f0f8ff", width=40, relief="groove",
pady=5).pack(pady=5)

    # --- Footer ---
    tk.Label(root, text="© 2025 | GD | Hostel Management System | All Rights Reserved",
bg="#ccddff", font=("Arial", 10)).pack(side="bottom", fill="x", pady=10)

    root.mainloop()

# Only run if this is the main file
if __name__ == "__main__":
    show_homepage()
#login.py
import tkinter as tk
from tkinter import messagebox
import mysql.connector
from hosteller_dashboard import open_hosteller_dashboard
from warden_dashboard import open_warden_dashboard

# --- Database Connection ---
try:
    db = mysql.connector.connect(
        host="localhost",
```

```
        user="root",
        password="",
        database="gd_hms"
    )
    cursor = db.cursor()
    print("Database connection successful.")
except mysql.connector.Error as err:
    print(f"Database connection failed: {err}")
    exit()

# --- Theme Colors ---
theme_colors = {
    "light": {"bg": "#eafaf1", "fg": "#2c3e50"},
    "dark": {"bg": "#0f2027", "fg": "#7fffd4"}
}
current_theme = "light"

# --- Toggle Theme ---
def toggle_theme(root):
    global current_theme
    current_theme = "dark" if current_theme == "light" else "light"
    root.configure(bg=theme_colors[current_theme]["bg"])
    for widget in root.winfo_children():
        try:
            widget.configure(bg=theme_colors[current_theme]["bg"])
            if "fg" in widget.configure():
                widget.configure(fg=theme_colors[current_theme]["fg"])
        except tk.TclError:
            pass

# --- Register User ---
def register_user(username, password):
    if not username or not password:
        messagebox.showwarning("⚠️ Incomplete", "Please fill out both fields.")
        return
    try:
        cursor.execute(
            "INSERT INTO users (username, password, role) VALUES (%s, %s, 'hosteller')",
            (username, password)
        )
        db.commit()
        messagebox.showinfo("🎉 Success", "User registered successfully!")
    except mysql.connector.IntegrityError:
        messagebox.showerror("❌ Error", "Username already exists.")

# --- Login User ---
def login_user(username, password, win):
```

```
    cursor.execute("SELECT user_id, role FROM users WHERE username=%s AND
password=%s", (username, password))
    result = cursor.fetchone()
    if result:
        user_id, role = result
        print(f"User logged in: {username} | Role: {role}")
        win.destroy()
        if role == "warden":
            open_warden_dashboard(username)
        elif role == "hosteller":
            open_hosteller_dashboard(username)
        else:
            messagebox.showerror("Login Error", f"Unknown role: {role}")
    else:
        messagebox.showerror("❌ Login Failed", "Invalid credentials.")

# --- Login Window ---
def main_login():
    login_win = tk.Tk()
    login_win.title("🔐 Secure Login")
    login_win.geometry("400x370")
    login_win.configure(bg=theme_colors[current_theme]["bg"])

    tk.Label(login_win, text="👩‍💼 User Access", font=("Helvetica", 18, "bold"),
             bg=theme_colors[current_theme]["bg"],
fg=theme_colors[current_theme]["fg"]).pack(pady=20)

    tk.Label(login_win, text="👷 Username:", bg=theme_colors[current_theme]["bg"],
             fg=theme_colors[current_theme]["fg"]).pack(pady=5)
    username_entry = tk.Entry(login_win, width=30)
    username_entry.pack(pady=5)

    tk.Label(login_win, text="🔑 Password:", bg=theme_colors[current_theme]["bg"],
             fg=theme_colors[current_theme]["fg"]).pack(pady=5)
    password_entry = tk.Entry(login_win, show="*", width=30)
    password_entry.pack(pady=5)

    button_frame = tk.Frame(login_win, bg=theme_colors[current_theme]["bg"])
    button_frame.pack(pady=20)

    tk.Button(button_frame, text="✅ Login", width=14, font=("Helvetica", 12, "bold"),
              bg="#66bb6a", fg="white",
              command=lambda: login_user(username_entry.get(), password_entry.get(),
login_win)).grid(row=0, column=0, padx=10)

    tk.Button(button_frame, text="📝 Register", width=14, font=("Helvetica", 12, "bold"),
              bg="#42a5f5", fg="white",
```

```
            command=lambda: register_user(username_entry.get(),
password_entry.get())).grid(row=0, column=1, padx=10)

    tk.Button(login_win, text="🎨 Switch Theme", font=("Helvetica", 12), relief="flat",
            bg=theme_colors[current_theme]["bg"], fg=theme_colors[current_theme]["fg"],
            command=lambda: toggle_theme(login_win)).pack(pady=10)

    login_win.mainloop()

# --- Launch Login ---
main_login()

#Register.py
import tkinter as tk
from tkinter import messagebox
from tkcalendar import DateEntry
import mysql.connector

# --- Database Connection ---
try:
    db = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="gd_hms"
    )
    cursor = db.cursor()
except mysql.connector.Error as err:
    print(f"Database connection failed: {err}")
    exit()

# --- Register Hosteller ---
def register_hosteller(data):
    try:
        query = """
        INSERT INTO users (
            username, password, role, first_name, last_name, email, phone_number,
            year_of_study, department, hostel_name, room_number, DOB
        )
        VALUES (%s, %s, 'hosteller', %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(query, data)
        db.commit()
        messagebox.showinfo("Success", "Hosteller registered successfully!")
    except mysql.connector.IntegrityError:
        messagebox.showerror("Error", "Username or Email already exists.")
    except Exception as e:
        messagebox.showerror("Database Error", str(e))
```

```
# --- Registration Form GUI ---
def open_extended_registration():
    reg_win = tk.Tk()
    reg_win.title("Extended Hosteller Registration")
    reg_win.geometry("400x650")
    reg_win.config(bg="white")

    fields = {}

    def add_field(label, show=None):
        tk.Label(reg_win, text=label, bg="white").pack(pady=2)
        entry = tk.Entry(reg_win, show=show) if not label.startswith("DOB") else
DateEntry(reg_win, date_pattern='yyyy-mm-dd')
        entry.pack(pady=2)
        fields[label] = entry

    labels = [
        "Username", "Password", "First Name", "Last Name", "Email",
        "Phone Number", "Year of Study", "Department", "Hostel Name",
        "Room Number", "DOB"
    ]

    for lbl in labels:
        add_field(lbl, show="*" if lbl == "Password" else None)

    def on_submit():
        values = [fields[label].get() for label in labels]
        if all(values):
            register_hosteller(tuple(values))
        else:
            messagebox.showwarning("Missing Fields", "Please fill in all fields.")

    tk.Button(reg_win, text="Register", bg="skyblue",
command=on_submit).pack(pady=20)
    reg_win.mainloop()

# Run
if __name__ == "__main__":
    open_extended_registration()
# hosteller_dashboard.py
import mysql.connector
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from datetime import datetime
import calendar
from tkcalendar import DateEntry
```

```
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="gd_hms"
    )


# --- Hosteller Functions ---
def view_hosteller_info(username):
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT first_name, last_name, email, phone_number, year_of_study, department,
hostel_name, room_number
        FROM users WHERE username = %s AND role = 'hosteller'
    """, (username,))
    result = cursor.fetchone()

    if not result:
        messagebox.showerror("Error", "Hosteller data not found.")
        return

    info_win = tk.Toplevel()
    info_win.title("Your Information")
    info_win.geometry("400x400")
    info_win.config(bg="#E0FFF0")

    labels = [
        "First Name", "Last Name", "Email", "Phone Number",
        "Year of Study", "Department", "Hostel Name", "Room Number"
    ]

    for label, value in zip(labels, result):
        tk.Label(info_win, text=f"{label}:", font=("Arial", 10, "bold"),
bg="#E0FFF0").pack(pady=2, anchor="w", padx=20)
        tk.Label(info_win, text=value, font=("Arial", 10), bg="#E0FFF0").pack(pady=1,
anchor="w", padx=40)

    tk.Button(info_win, text="Close", command=info_win.destroy).pack(pady=20)

def manage_attendance(username):
    from tkcalendar import Calendar
    import tkinter as tk
```

```
    from tkinter import ttk

    def mark_attendance(status):
        selected_date = cal.get_date()
        conn = get_db_connection()
        cursor = conn.cursor()

        try:
            cursor.execute("""
                INSERT INTO attendance (username, date, status)
                VALUES (%s, %s, %s)
                ON DUPLICATE KEY UPDATE status = VALUES(status)
            """, (username, selected_date, status))
            conn.commit()
            messagebox.showinfo("Success", f"Marked {status} for {selected_date}")
            load_attendance()
        except Exception as e:
            messagebox.showerror("Error", str(e))
        finally:
            conn.close()

    def load_attendance():
        for item in tree.get_children():
            tree.delete(item)

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT date, status FROM attendance
            WHERE username = %s ORDER BY date DESC
        """, (username,))
        for date, status in cursor.fetchall():
            tree.insert('', 'end', values=(date, status))
        conn.close()

    att_win = tk.Toplevel()
    att_win.title("Manage Attendance")
    att_win.geometry("500x500")
    att_win.config(bg="#F0FFF0")

    tk.Label(att_win, text="Select a date and mark your attendance:", font=("Arial", 12,
"bold"), bg="#F0FFF0").pack(pady=10)

    cal = Calendar(att_win, selectmode="day", date_pattern="yyyy-mm-dd")
    cal.pack(pady=10)

    btn_frame = tk.Frame(att_win, bg="#F0FFF0")
    btn_frame.pack(pady=10)
```

```
    for label, status in [("Present", "P"), ("Absent", "A"), ("Leave", "L")]:
        tk.Button(btn_frame, text=label, width=10, command=lambda s=status:
mark_attendance(s)).pack(side="left", padx=5)

    tree = ttk.Treeview(att_win, columns=("Date", "Status"), show="headings")
    tree.heading("Date", text="Date")
    tree.heading("Status", text="Status")
    tree.pack(pady=20, fill="both", expand=True)

    load_attendance()

    tk.Button(att_win, text="Close", command=att_win.destroy).pack(pady=10)

def request_outpass(username):
    def submit_request():
        start_date = start_cal.get_date()
        end_date = end_cal.get_date()
        reason = reason_text.get("1.0", tk.END).strip()

        if start_date > end_date:
            messagebox.showerror("Invalid Dates", "Start date cannot be after end date.")
            return
        if not reason:
            messagebox.showwarning("Missing Reason", "Please enter a reason for your
outpass request.")
            return

        conn = get_db_connection()
        cursor = conn.cursor()
        try:
        # Get user_id from username
            cursor.execute("SELECT user_id FROM users WHERE username = %s",
(username,))
            result = cursor.fetchone()
            if not result:
                messagebox.showerror("User Error", "User not found in the database.")
                return

            user_id = result[0]

        # Insert into outpass_requests with user_id and request_date as NOW()
            cursor.execute("""
                INSERT INTO outpass_requests (user_id, start_date, end_date, reason, status,
request_date)
                VALUES (%s, %s, %s, %s, %s, NOW())
            """, (user_id, start_date, end_date, reason, 'Pending'))

            conn.commit()
```

```
            messagebox.showinfo("Request Submitted", "Your outpass request has been
submitted.")
            outpass_win.destroy()

        except Exception as e:
            messagebox.showerror("Error", str(e))
        finally:
            conn.close()
    # Create popup window
    outpass_win = tk.Toplevel()
    outpass_win.title("Request Outpass")
    outpass_win.geometry("450x400")
    outpass_win.config(bg="#F5F5DC")

    # Start Date (Dropdown calendar)
    tk.Label(outpass_win, text="Select Start Date:", bg="#F5F5DC", font=("Arial",
12)).pack(pady=5)
    start_cal = DateEntry(outpass_win, date_pattern='yyyy-mm-dd', width=15)
    start_cal.pack(pady=5)

    # End Date (Dropdown calendar)
    tk.Label(outpass_win, text="Select End Date:", bg="#F5F5DC", font=("Arial",
12)).pack(pady=5)
    end_cal = DateEntry(outpass_win, date_pattern='yyyy-mm-dd', width=15)
    end_cal.pack(pady=5)

    # Reason Text Field
    tk.Label(outpass_win, text="Reason for Outpass:", bg="#F5F5DC", font=("Arial",
12)).pack(pady=10)
    reason_text = tk.Text(outpass_win, height=5, width=45, wrap="word", font=("Arial",
10))
    reason_text.pack(pady=5)

    # Buttons
    tk.Button(outpass_win, text="Submit Request", command=submit_request,
bg="#4CAF50", fg="white", width=20).pack(pady=15)
    tk.Button(outpass_win, text="Cancel", command=outpass_win.destroy,
width=20).pack()


def view_outpass_history(username):
    import tkinter as tk
    from tkinter import ttk, messagebox

    def load_outpass_history():
        for item in tree.get_children():
            tree.delete(item)
```

```
        try:
            conn = get_db_connection()
            cursor = conn.cursor()

            # Step 1: Get user_id from username
            cursor.execute("SELECT user_id FROM users WHERE username = %s",
(username,))
            result = cursor.fetchone()

            if not result:
                messagebox.showerror("Error", "User not found in the database.")
                return

            user_id = result[0]

            # Step 2: Fetch outpass history using user_id
            cursor.execute("""
                SELECT start_date, end_date, reason, status
                FROM outpass_requests
                WHERE user_id = %s
                ORDER BY request_date DESC
            """, (user_id,))
            rows = cursor.fetchall()

            if not rows:
                messagebox.showinfo("No History", "No outpass requests found.")
                return

            for row in rows:
                tree.insert('', 'end', values=row)

        except Exception as e:
            messagebox.showerror("Error", str(e))
        finally:
            conn.close()

    # Create a new window for outpass history
    history_win = tk.Toplevel()
    history_win.title("Outpass History")
    history_win.geometry("600x400")
    history_win.config(bg="#FFF8DC")

    # Heading
    tk.Label(history_win, text="Your Outpass Request History", font=("Arial", 14, "bold"),
bg="#FFF8DC").pack(pady=10)

    # Treeview to display the outpass history
```

```
    tree = ttk.Treeview(history_win, columns=("Start Date", "End Date", "Reason",
"Status"), show="headings")
    tree.heading("Start Date", text="Start Date")
    tree.heading("End Date", text="End Date")
    tree.heading("Reason", text="Reason")
    tree.heading("Status", text="Status")
    tree.pack(pady=10, fill="both", expand=True)

    # Load data
    load_outpass_history()

    # Close button
    tk.Button(history_win, text="Close", command=history_win.destroy).pack(pady=10)

# --- Hosteller Dashboard ---
def open_hosteller_dashboard(hosteller_username):
    win = tk.Tk()
    win.title("Hosteller Dashboard")
    win.geometry("600x500")
    win.config(bg="#98FB98")

    # Use the passed username instead of 'hosteller_name'
    greeting = f"Welcome, {hosteller_username}!" if hosteller_username else "Welcome,
Hosteller!"
    tk.Label(win, text=greeting, font=("Arial", 16, "bold"), bg="#98FB98").pack(pady=20)
    tk.Label(win, text=calendar.month_name[datetime.now().month], font=("Arial", 14),
bg="#98FB98").pack()

    btns = [
        ("📋 View Personal Info", lambda: view_hosteller_info(hosteller_username)),
        ("🗓️ Manage Attendance", lambda: manage_attendance(hosteller_username)),
        ("📤 Request Out Pass", lambda: request_outpass(hosteller_username)),
        ("🔍 View Out Pass History", lambda: view_outpass_history(hosteller_username)),
        ("🚪 Logout", win.destroy)
    ]
    for text, cmd in btns:
        tk.Button(win, text=text, font=("Arial", 12), bg="lightgreen", width=30,
command=cmd).pack(pady=8)

    win.mainloop()
# --- Only run if executed directly ---
if __name__ == "__main__":
    open_hosteller_dashboard()
# warden_dashboard.py

import tkinter as tk
from tkinter import messagebox
```

```
from datetime import datetime
import mysql.connector
import calendar
from tkinter import ttk

def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="gd_hms"
    )


# --- Warden Dashboard ---
def open_warden_dashboard(username):
    conn = get_db_connection()
    cursor = conn.cursor()
    win = tk.Tk()
    win.title("Warden Dashboard")
    win.geometry("600x600")
    win.config(bg="#FFD700")

    tk.Label(win, text=f"Welcome, Warden {username}!", font=("Arial", 16, "bold"),
bg="#FFD700").pack(pady=20)

    cal_label = tk.Label(win, text=calendar.month_name[datetime.now().month],
font=("Arial", 14), bg="#FFD700")
    cal_label.pack()

    btns = [
        ("View Attendance", view_attendance),
        ("📩 Approve Out Passes", approve_outpass),
        ("📋 View Hostellers", view_hostellers),
        ("🗓️ Attendance Overview", view_attendance_summary),
        ("🎂 Birthday Alerts", show_birthday_alerts),
        ("🔍 Filter Hostellers", filter_hostellers),
        ("🚪 Logout", win.destroy)
    ]
    for text, cmd in btns:
        tk.Button(win, text=text, font=("Arial", 12), bg="skyblue", width=30,
command=cmd).pack(pady=8)

    win.mainloop()

# --- Warden Functional Placeholders ---
def view_attendance():
```

```
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        query = """
            SELECT u.username, u.first_name, u.last_name, a.date, a.status
            FROM attendance a
            JOIN users u ON a.username = u.username
            ORDER BY a.date DESC
        """
        cursor.execute(query)
        records = cursor.fetchall()

        if not records:
            messagebox.showinfo("Attendance", "No attendance records found.")
            return

        att_win = tk.Toplevel()
        att_win.title("Attendance Records")
        att_win.geometry("700x400")
        att_win.config(bg="white")

        tk.Label(att_win, text="Attendance Records", font=("Arial", 14, "bold"),
bg="white").pack(pady=10)

        frame = tk.Frame(att_win, bg="white")
        frame.pack(fill="both", expand=True)

        headings = ["Username", "First Name", "Last Name", "Date", "Status"]
        for idx, heading in enumerate(headings):
            tk.Label(frame, text=heading, font=("Arial", 10, "bold"), bg="lightgray",
width=15, borderwidth=1, relief="solid").grid(row=0, column=idx)

        for row_idx, row in enumerate(records, start=1):
            for col_idx, value in enumerate(row):
                tk.Label(frame, text=value, font=("Arial", 10), bg="white", width=15,
borderwidth=1, relief="solid").grid(row=row_idx, column=col_idx)

        cursor.close()
        conn.close()

    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error fetching attendance
records:\n{err}")

def approve_outpass():
    try:
        conn = get_db_connection()
```

```
    cursor = conn.cursor()

    # Query to get pending outpass requests
    query = """
        SELECT o.request_id, u.first_name, u.last_name, o.request_date, o.start_date,
o.end_date, o.reason
        FROM outpass_requests o
        JOIN users u ON o.user_id = u.user_id
        WHERE o.status = 'Pending'
    """
    cursor.execute(query)
    records = cursor.fetchall()

    if not records:
        messagebox.showinfo("Approve Outpass", "No pending outpasses found.")
        return

    # Create a new window to display pending outpasses
    outpass_win = tk.Toplevel()
    outpass_win.title("Approve Outpass Requests")
    outpass_win.geometry("900x400")
    outpass_win.config(bg="white")

    tk.Label(outpass_win, text="Pending Outpass Requests", font=("Arial", 14, "bold"),
bg="white").pack(pady=10)

    frame = tk.Frame(outpass_win, bg="white")
    frame.pack(fill="both", expand=True)

    # Create headings
    headings = ["Request ID", "First Name", "Last Name", "Request Date", "Start Date",
"End Date", "Reason", "Action"]
    for idx, heading in enumerate(headings):
        tk.Label(frame, text=heading, font=("Arial", 10, "bold"), bg="lightgray",
width=15, borderwidth=1, relief="solid").grid(row=0, column=idx)

    # Display pending outpasses
    for row_idx, row in enumerate(records, start=1):
        for col_idx, value in enumerate(row[:-1]):  # All columns except the last "Reason"
column
            # Handle NULL in request_date by showing "Not Set" or any placeholder
            display_value = value if value is not None else "Not Set"
            tk.Label(frame, text=display_value, font=("Arial", 10), bg="white", width=15,
borderwidth=1, relief="solid").grid(row=row_idx, column=col_idx)

        # Reason column
        tk.Label(frame, text=row[6], font=("Arial", 10), bg="white", width=15,
borderwidth=1, relief="solid").grid(row=row_idx, column=6)
```

```
            # Add approve/reject buttons
            approve_btn = tk.Button(frame, text="Approve", command=lambda id=row[0]:
update_outpass_status(id, 'Approved', outpass_win))
            reject_btn = tk.Button(frame, text="Reject", command=lambda id=row[0]:
update_outpass_status(id, 'Rejected', outpass_win))
            approve_btn.grid(row=row_idx, column=7, padx=5)
            reject_btn.grid(row=row_idx, column=8, padx=5)

        cursor.close()
        conn.close()

    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error fetching outpass records:\n{err}")

def update_outpass_status(outpass_id, status, window):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        # Update the outpass status (approve/reject)
        update_query = """
            UPDATE outpass_requests
            SET status = %s, request_date = IFNULL(request_date, NOW())  -- Set
request_date if it's NULL
            WHERE request_id = %s
        """
        cursor.execute(update_query, (status, outpass_id))
        conn.commit()

        messagebox.showinfo("Outpass Status", f"Outpass has been {status.lower()}.")

        # Close the window after updating the status
        window.destroy()

        cursor.close()
        conn.close()

    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error updating outpass status:\n{err}")

def view_hostellers():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        # Query to fetch all hostellers
        query = """
```

```
        SELECT user_id, first_name, last_name, email, phone_number, year_of_study,
department, hostel_name, room_number
        FROM users
        WHERE role = 'hosteller';
    """
    cursor.execute(query)
    hostellers = cursor.fetchall()

    if not hostellers:
        messagebox.showinfo("View Hostellers", "No hostellers found.")
        return

    # Create a new window to display hostellers
    hosteller_win = tk.Toplevel()
    hosteller_win.title("List of Hostellers")
    hosteller_win.geometry("800x400")
    hosteller_win.config(bg="white")

    tk.Label(hosteller_win, text="List of Hostellers", font=("Arial", 14, "bold"),
bg="white").pack(pady=10)

    frame = tk.Frame(hosteller_win, bg="white")
    frame.pack(fill="both", expand=True)

    # Create headings for the table
    headings = ["User ID", "First Name", "Last Name", "Email", "Phone", "Year of
Study", "Department", "Hostel Name", "Room Number"]
    for idx, heading in enumerate(headings):
        tk.Label(frame, text=heading, font=("Arial", 10, "bold"), bg="lightgray",
width=15, borderwidth=1, relief="solid").grid(row=0, column=idx)

    # Display all hostellers
    for row_idx, row in enumerate(hostellers, start=1):
        for col_idx, value in enumerate(row):
            tk.Label(frame, text=value, font=("Arial", 10), bg="white", width=15,
borderwidth=1, relief="solid").grid(row=row_idx, column=col_idx)

    cursor.close()
    conn.close()

  except mysql.connector.Error as err:
    messagebox.showerror("Database Error", f"Error fetching hostellers:\n{err}")

def view_attendance_summary():
  try:
    conn = get_db_connection()
    cursor = conn.cursor()
```

```
        # Get the current month and year
        current_month = datetime.now().month
        current_year = datetime.now().year

        # Query to get total attendance records for the current month
        query = """
            SELECT u.first_name, u.last_name, a.status
            FROM attendance a
            JOIN users u ON a.username = u.username
            WHERE MONTH(a.date) = %s AND YEAR(a.date) = %s;
        """
        cursor.execute(query, (current_month, current_year))
        attendance_records = cursor.fetchall()

        if not attendance_records:
            messagebox.showinfo("Attendance Summary", "No attendance records found for
the current month.")
            return

        # Calculate the summary details
        total_hostellers = len(set([record[0] + " " + record[1] for record in
attendance_records]))  # unique hostellers
        present_count = sum(1 for record in attendance_records if record[2] == 'P')
        absent_count = sum(1 for record in attendance_records if record[2] == 'A')
        leave_count = sum(1 for record in attendance_records if record[2] == 'L')

        attendance_percentage = (present_count / len(attendance_records)) * 100 if
attendance_records else 0

        # Create a new window to display the attendance summary
        summary_win = tk.Toplevel()
        summary_win.title("Attendance Summary")
        summary_win.geometry("400x300")
        summary_win.config(bg="white")

        tk.Label(summary_win, text="Attendance Summary", font=("Arial", 14, "bold"),
bg="white").pack(pady=10)

        # Display the summary information
        summary_text = f"""
        Total Hostellers: {total_hostellers}
        Present: {present_count}
        Absent: {absent_count}
        On Leave: {leave_count}
        Attendance Percentage: {attendance_percentage:.2f}%
        """
        tk.Label(summary_win, text=summary_text, font=("Arial", 12),
bg="white").pack(pady=20)
```

```
        cursor.close()
        conn.close()

    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error fetching attendance
summary:\n{err}")

def show_birthday_alerts():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        # Get today's month and day (ignore the year)
        today = datetime.now()
        today_month_day = today.strftime('%m-%d')

        # Query to get users with today's birthday (ignoring the year)
        query = """
            SELECT first_name, last_name, email
            FROM users
            WHERE DATE_FORMAT(DOB, '%m-%d') = %s;
        """
        cursor.execute(query, (today_month_day,))
        birthday_users = cursor.fetchall()

        if not birthday_users:
            messagebox.showinfo("Birthday Alerts", "No birthdays today.")
            return

        # Create a new window to display today's birthdays
        birthday_win = tk.Toplevel()
        birthday_win.title("Birthday Alerts")
        birthday_win.geometry("400x300")
        birthday_win.config(bg="white")

        tk.Label(birthday_win, text="Today's Birthdays", font=("Arial", 14, "bold"),
bg="white").pack(pady=10)

        # Display the list of birthday users
        birthday_text = "\n".join([f"{user[0]} {user[1]} (Email: {user[2]})" for user in
birthday_users])
        tk.Label(birthday_win, text=birthday_text, font=("Arial", 12),
bg="white").pack(pady=20)

        cursor.close()
        conn.close()
```

```
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error fetching birthday list:\n{err}")



def filter_hostellers():
    def apply_filter():
        # Get the filter values from the selected dropdown options
        hostel_name = hostel_name_var.get()
        department = department_var.get()
        year_of_study = year_of_study_var.get()

        # Build the query based on the provided filter values
        query = "SELECT first_name, last_name, hostel_name, department, year_of_study
FROM users WHERE role = 'hosteller'"

        # Collect filter conditions
        conditions = []
        params = []

        if hostel_name != "All Hostels":
            conditions.append("hostel_name = %s")
            params.append(hostel_name)
        if department != "All Departments":
            conditions.append("department = %s")
            params.append(department)
        if year_of_study != "All Years":
            conditions.append("year_of_study = %s")
            params.append(year_of_study)

        # Add conditions to the query if any filters are applied
        if conditions:
            query += " AND " + " AND ".join(conditions)

        # Execute the query
        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute(query, tuple(params))
            filtered_hostellers = cursor.fetchall()

            # Display the filtered results
            if filtered_hostellers:
                result_window = tk.Toplevel()
                result_window.title("Filtered Hostellers")
                result_window.geometry("600x400")
                result_window.config(bg="white")
```

```
            # Create a Label to show the count of filtered hostellers
            count_label = tk.Label(result_window, text=f"Total Hostellers Found:
{len(filtered_hostellers)}", font=("Arial", 14, "bold"), bg="white")
            count_label.pack(pady=10)

            # Create a Treeview widget to display the filtered hostellers in columns
            tree = ttk.Treeview(result_window, columns=("First Name", "Last Name",
"Hostel Name", "Department", "Year of Study"), show="headings")

            # Define the columns
            tree.heading("First Name", text="First Name")
            tree.heading("Last Name", text="Last Name")
            tree.heading("Hostel Name", text="Hostel Name")
            tree.heading("Department", text="Department")
            tree.heading("Year of Study", text="Year of Study")

            # Define the column widths
            tree.column("First Name", width=150)
            tree.column("Last Name", width=150)
            tree.column("Hostel Name", width=120)
            tree.column("Department", width=120)
            tree.column("Year of Study", width=100)

            # Insert the filtered data into the Treeview
            for hosteller in filtered_hostellers:
                tree.insert("", tk.END, values=hosteller)

            # Add a scrollbar to the Treeview
            scrollbar = ttk.Scrollbar(result_window, orient="vertical",
command=tree.yview)
            tree.configure(yscrollcommand=scrollbar.set)
            scrollbar.pack(side="right", fill="y")

            tree.pack(pady=20)

        else:
            messagebox.showinfo("No Results", "No hostellers found matching the filter
criteria.")
        cursor.close()
        conn.close()
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Error fetching hosteller data:\n{err}")
# Create a new window for filter options
filter_win = tk.Toplevel()
filter_win.title("Filter Hostellers")
filter_win.geometry("400x300")
filter_win.config(bg="white")
```

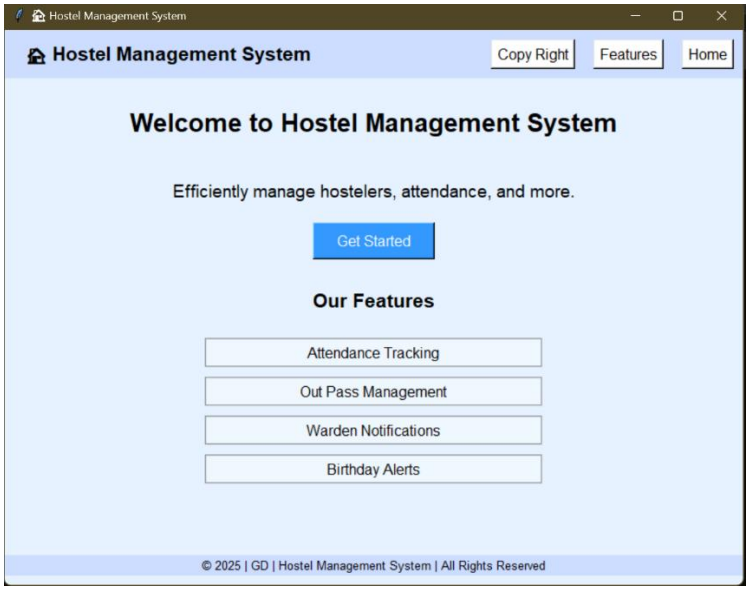
```
    # Hostel Name Dropdown
    tk.Label(filter_win, text="Filter by Hostel Name", bg="white").pack(pady=5)
    hostel_name_var = tk.StringVar()
    hostel_name_dropdown = ttk.Combobox(filter_win, textvariable=hostel_name_var,
font=("Arial", 12))
    hostel_name_dropdown['values'] = ["All Hostels", "Hostel A", "Hostel B", "Hostel C"]
# Example hostel names
    hostel_name_dropdown.current(0)  # Set default value to "All Hostels"
    hostel_name_dropdown.pack(pady=5)
    tk.Label(filter_win, text="Filter by Department", bg="white").pack(pady=5)
    department_var = tk.StringVar()
    department_dropdown = ttk.Combobox(filter_win, textvariable=department_var,
font=("Arial", 12))
    department_dropdown['values'] = ["All Departments", "Computer Science",
"Mechanical", "Electrical"]  # Example departments
    department_dropdown.current(0)  # Set default value to "All Departments"
    department_dropdown.pack(pady=5)
    tk.Label(filter_win, text="Filter by Year of Study", bg="white").pack(pady=5)
    year_of_study_var = tk.StringVar()
    year_of_study_dropdown = ttk.Combobox(filter_win, textvariable=year_of_study_var,
font=("Arial", 12))
    year_of_study_dropdown['values'] = ["All Years", "1", "2", "3", "4"]  # Example year
of study options
    year_of_study_dropdown.current(0)  # Set default value to "All Years"
    year_of_study_dropdown.pack(pady=5)
    tk.Button(filter_win, text="Apply Filter", font=("Arial", 12), bg="skyblue",
command=apply_filter).pack(pady=20)

# --- Only run when directly executed ---
if __name__ == "__main__":
    open_warden_dashboard()
```

APPENDIX B

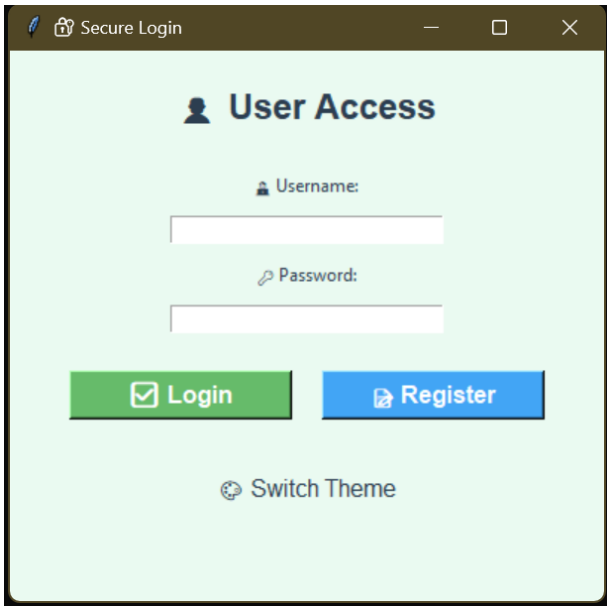
1.SQL TABLES

```
MariaDB [gd_hms]> show tables;
+------------------+
| Tables_in_gd_hms |
+------------------+
| attendance       |
| hostellers       |
| outpass_requests |
| users            |
| warden           |
+------------------+
5 rows in set (0.001 sec)
```

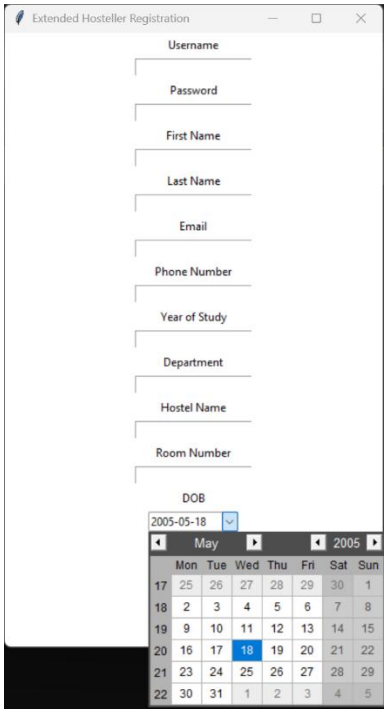
2.HOMEPAGE



3.LOGIN PAGE

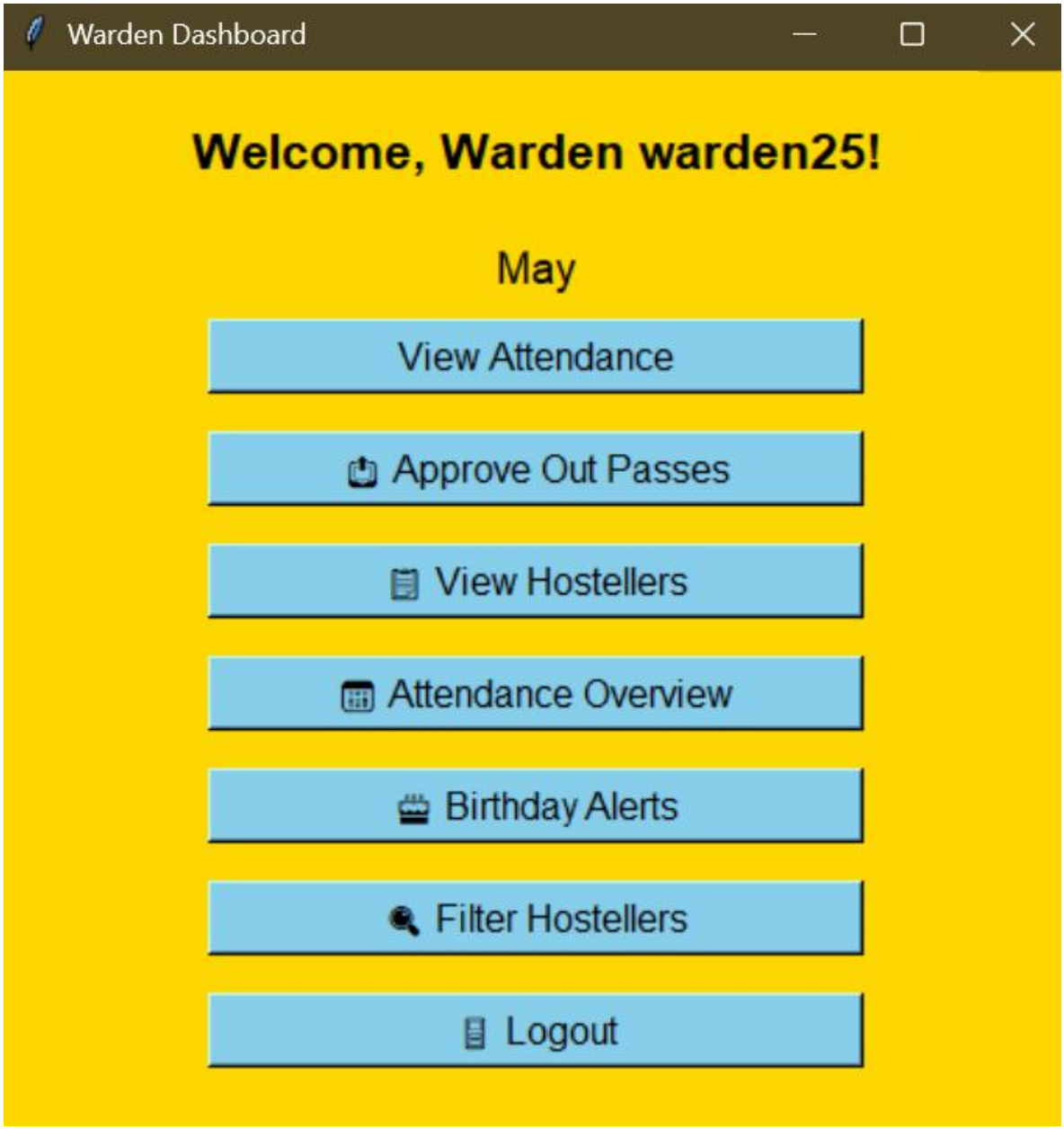


4.REGISTER PAGE



5.HOSTELLER DASHBOARD

6.WARDEN DASHBOARD



REFERENCES

1. System Design and Development

https://www.oreilly.com/library/view/learning-python-5th/9781449355722/

https://www.db-book.com/

2. User Interface and Experience

https://tkdocs.com/tutorial/

https://www.w3schools.com/sql/sql_join.asp

3. Backend Logic and Data Handling

https://docs.python.org/3/

https://www.geeksforgeeks.org/sql-query-questions-and-answers/

4. Testing and Deployment

https://realpython.com/python-testing/

https://www.oracle.com/database/technologies/appdev/testing.html