

ON-DUTY MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

PADMAPRIYA S

220701193

NANDHANA C H

220701181

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 – 24

BONAFIDE CERTIFICATE

Certified that this project report “**ON-DUTY MANAGEMENT SYSTEM**” is the bonafide work of “**PADMAPRIYA S (220701193), NANDHANA C H (220701181)**” who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA

**Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105**

SIGNATURE

Mrs.D.KALPANA

**Assistant Professor,
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105**

ABSTRACT

The ON-DUTY Management System for colleges is designed to automate and streamline the process of handling on-duty (OD) requests submitted by students. This system encompasses various roles including students, staff, wardens, and Heads of Departments (HODs) to ensure an efficient and accurate approval process. Students with attendance greater than 75% are eligible for OD requests, which are first reviewed by respective staff members or wardens and then require final approval from HODs. The project aims to enhance administrative efficiency, maintain accurate records, and ensure only eligible students receive approvals.

The system is developed using a combination of MySQL for database management, Python for server-side scripting, and Python (tkinter) for the user interface. The architecture includes a central database to store attendance records, OD requests, and user information, with a web server handling requests and serving the application. Functional requirements include user authentication, role-based access control, attendance verification, OD request submission and approval workflows, and notification systems for request status updates. Non-functional requirements focus on security, performance, usability, and scalability to handle a growing number of users and requests. The project covers detailed modules for students, staff, wardens, HODs, and administrators, ensuring a comprehensive solution for OD management. The design involves normalization to ensure data integrity and eliminate redundancy. The system's effectiveness will be evaluated based on performance metrics, user feedback, and the ability to streamline the OD approval process.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 EXISTING AND PROPOSED SYSTEM

1.3 OBJECTIVES

1.4 MODULES

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 MySQL

2.2.2 PYTHON(tkinter)

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6. CONCLUSION

7. REFERENCES

1. INTRODUCTION

1.1 INTRODUCTION

The Duty Management System is a comprehensive application designed to streamline the process of handling on-duty (OD) requests in a college environment. This system is essential for managing student attendance and ensuring that only those students who meet the required attendance criteria are granted permission to miss classes for valid reasons. The system integrates the roles of students, staff, wardens, and Heads of Departments (HODs) to create an efficient and transparent process for submitting, reviewing, and approving OD requests.

The current manual process of handling OD requests can be cumbersome, prone to errors, and time-consuming for both students and administrative staff. By digitizing this process, the Duty Management System aims to enhance administrative efficiency, reduce paperwork, and ensure accurate record-keeping. Additionally, it helps in maintaining fairness and transparency, as decisions are based on predefined rules and attendance data.

1.2 EXISTING SYSTEM

In many educational institutions, the process of managing on-duty requests is often manual and paper-based. Students fill out physical forms to request on-duty leaves, which are then submitted to faculty members or administrative staff. These requests are reviewed manually, with staff checking attendance records and other criteria before approving or rejecting the requests. This manual system has several drawbacks:

Inefficiency: The process is time-consuming, requiring physical handling of documents and manual checking of records.

Prone to Errors: Manual data entry and processing can lead to errors, such as misplacing requests or incorrectly calculating attendance.

Lack of Transparency: Students may not have visibility into the status of their requests, leading to uncertainty and repeated follow-ups.

Limited Accessibility: Requests and records are often only accessible during office hours, causing delays.

Administrative Burden: Staff must manage and process numerous requests, which can be overwhelming during peak times.

PROPOSED SYSTEM

The proposed On-Duty Management System (ODMS) aims to automate and streamline the process of handling on-duty requests. The system will provide a digital platform where students can submit their requests, and staff can review and process them efficiently. Key features of the proposed system include:

Automated Request Submission: Students can submit on-duty requests through a user-friendly GUI built using Python's Tkinter. They can provide reasons and upload supporting documents digitally.

Efficient Review Process: Faculty, HODs, and wardens can review requests online and checking students' attendance records.

Real-Time Notifications: Students receive real-time updates on the status of their requests, improving transparency.

Centralized Database: All data related to users, attendance, and requests is stored in a centralized MySQL database, ensuring data integrity and easy access.

Role-Based Access Control: Different user roles (students, faculty, HODs, wardens) have specific permissions, ensuring that only authorized personnel can approve or reject requests.

FUTURE ENHANCEMENTS

To further improve the On-Duty Management System, several future enhancements can be considered:

Mobile Application: Develop a mobile app version of the ODMS to allow students and staff to access the system on-the-go. This would increase accessibility and convenience.

Integration with College Management Systems: Integrate ODMS with existing college management systems for seamless data exchange, such as syncing student attendance records and academic calendars.

Advanced Analytics and Reporting: Implement advanced analytics to generate insightful reports on attendance trends, request patterns, and approval rates. This can help in decision-making and identifying areas for improvement.

Automated Notifications: Enhance the notification system to include email and SMS alerts for

important updates, ensuring that students and staff are promptly informed.

Enhanced Security: Implement advanced security measures, such as multi-factor authentication and data encryption, to protect sensitive information.

Customizable Workflows: Allow institutions to customize the approval workflow to match their specific administrative processes and policies.

Feedback System: Introduce a feedback mechanism where students and staff can provide feedback on the ODMS, helping to continuously improve the system based on user input.

Support for Multiple Campuses: Extend the system to support multiple campuses within an institution, allowing centralized management of on-duty requests across different locations.

AI-Based Recommendations: Utilize AI to provide recommendations on approval decisions based on historical data and predefined rules, assisting staff in making informed decisions.

Enhanced Document Management: Improve the document management system to support various file formats and provide easy retrieval and management of uploaded documents.

1.3 OBJECTIVES:

- Streamline the submission, review, and approval of OD requests through a digital platform.
- Only eligible students (attendance > 75%) can have their OD requests approved.
- Include the roles of staff and wardens for initial approval and HODs for final approval, ensuring a comprehensive review process.
- Include wardens to manage hostel students' OD requests.
- Keep precise records of attendance and OD requests, facilitating easy access and reporting for administrative purposes.
- Provide reporting capabilities for staff and administration to monitor and manage OD requests and student attendance.

1.4 MODULES:

The On- Duty Management System is designed with distinct modules, each tailored to handle specific roles and responsibilities within the college. This modular approach ensures that each user has access to the functionalities relevant to their role, contributing to a streamlined and efficient process for handling on-duty (OD) requests.

Student Module:

The Student Module is the interface through which students interact with the On- Duty Management System. It provides students with the ability to submit OD requests and track the status of these requests.

Functionality:

- * Students can fill out and submit OD request forms online, specifying the reason for the request and the dates they seek to be excused.
- * Students can view the status of their submitted OD requests (e.g., pending, approved, rejected).
- * Students can access their attendance records to ensure they meet the minimum attendance requirement of 75%.

Key Features:

- * Ensures that only authorized students can access the system and submit OD requests.
- * An intuitive form for OD request submission, capturing necessary details like dates and reasons.
- * Real-time updates on the status of OD requests, including notifications for approvals and rejections.
- * A dashboard displaying the student's current attendance percentage.

Faculty Module:

The Staff Module allows academic staff to review and process OD requests submitted by students. Staff members play a crucial role in verifying attendance and providing initial approval or rejection of OD requests.

Functionality:

- * Staff can access a list of pending OD requests submitted by students in their classes.
- * Staff can check the attendance records of students to determine if they meet the 75% attendance requirement.
- * Based on attendance and other criteria, staff can approve or reject OD requests.

Key Features:

- * Ensures that only authorized staff members can access the system and process OD requests.
- * Tools to easily access and verify student attendance records.
- * Options to approve or reject OD requests, with comments or reasons for the decision.
- * Automated notifications to students regarding the status of their requests.

Warden Module:

The Warden Module is designed specifically for wardens to manage OD requests from students residing in hostels. Wardens ensure that hostel students' attendance and conduct are considered before approval.

Functionality:

- * Wardens can view attendance records specific to hostel residents.
- * Wardens have the authority to approve or reject OD requests for hostel students.

Key Features:

- * Secure login for wardens to access the system.
- * Access to attendance and conduct records of hostel students.
- * Tools for wardens to approve or reject OD requests based on hostel policies and attendance.
- * Mechanisms to communicate decisions and provide feedback to students.

HOD Module:

The HOD Module provides Heads of Departments with the ability to oversee and provide final approval for OD requests within their departments. HODs ensure that the requests align with departmental policies and overall student performance.

Functionality:

- * HODs can view all OD requests within their department, including those pending, approved, and rejected by staff or wardens.
- * HODs have the final say in the approval or rejection of OD requests.
- * Generate reports on OD requests and student attendance for administrative purposes.

Key Features:

- * Secure access for HODs to the system.
- * Comprehensive tools to review, approve, or reject OD requests.
- * The ability to provide final approval or reversal of decisions made by staff or wardens.
- * Advanced reporting capabilities to analyze trends in OD requests and attendance.

Each module is designed to ensure that the On-Duty Management System operates efficiently, with clear responsibilities and streamlined processes for all users involved. This modular approach not only enhances functionality but also ensures that the system can be easily maintained and scaled as needed.

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

The On-Duty Management System (ODMS) is designed to efficiently manage and automate the process of handling on-duty requests from students in a college setting. The system is built to streamline the workflow, from submitting requests to final approval, ensuring a seamless experience for both students and staff. The primary components of ODMS include a graphical user interface (GUI) built using Python's Tkinter and a robust backend database managed with MySQL.

The GUI, developed with Tkinter, provides an intuitive and user-friendly interface. The Tkinter-based GUI ensures that users, irrespective of their technical expertise, can navigate and utilize the system efficiently.

The backend of the system is powered by MySQL, a widely-used open-source relational database management system. MySQL is chosen for its reliability, performance, and ability to handle complex queries and large volumes of data efficiently.

2.2 LANGUAGES

2.2.1 MySQL

MySQL is a powerful, open-source relational database management system known for its performance, reliability, and ease of use. In the context of ODMS, MySQL serves as the backbone for data storage and management. The database schema includes tables for users, students, and on-duty requests, with relationships ensuring data integrity and consistency. MySQL's robust querying capabilities allow for efficient retrieval and manipulation of data, facilitating the approval workflow and attendance checks critical to the system's functionality.

2.2.2 PYTHON (Tkinter)

Python is a versatile and high-level programming language renowned for its readability and simplicity. Tkinter, the standard GUI library for Python, provides a fast and easy way to create graphical user interfaces. Tkinter widgets, such as buttons, labels, entry fields, and message boxes, allow developers to design a responsive and user-friendly interface. In ODMS, Tkinter is used to create the main interface for students to submit on-duty requests and for staff to review and manage these requests. The combination of Python and Tkinter ensures that the application is not only functional but also easy to use and maintain, providing a seamless experience for all users involved.

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

User Roles: The system will have different user roles including students, faculty, HOD, and warden.

Student Module: Students can request on-duty (OD) by providing a reason and supporting documents.

Staff Module: Faculty members, HODs, and wardens can approve or reject OD requests.

Attendance Check: The system will check if a student's attendance is greater than 75% before allowing them to submit an OD request.

User Authentication: Secure login for all users.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

Processor: Intel Core i or equivalent

RAM: 4 GB or higher

Storage: 500 GB HDD or SSD

Software Requirements:

Operating System: Windows

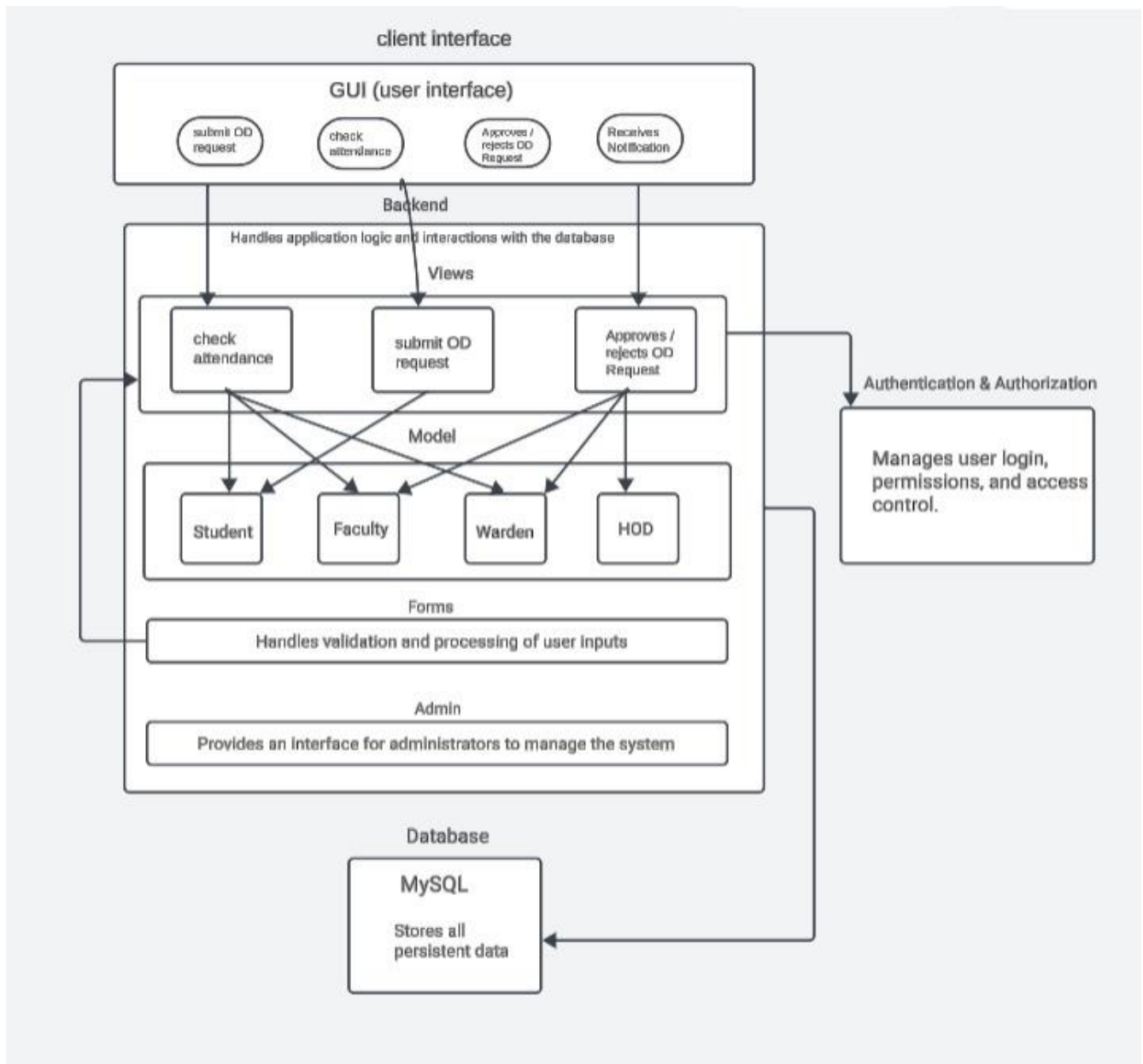
Python 3.12.3

MySQL Server

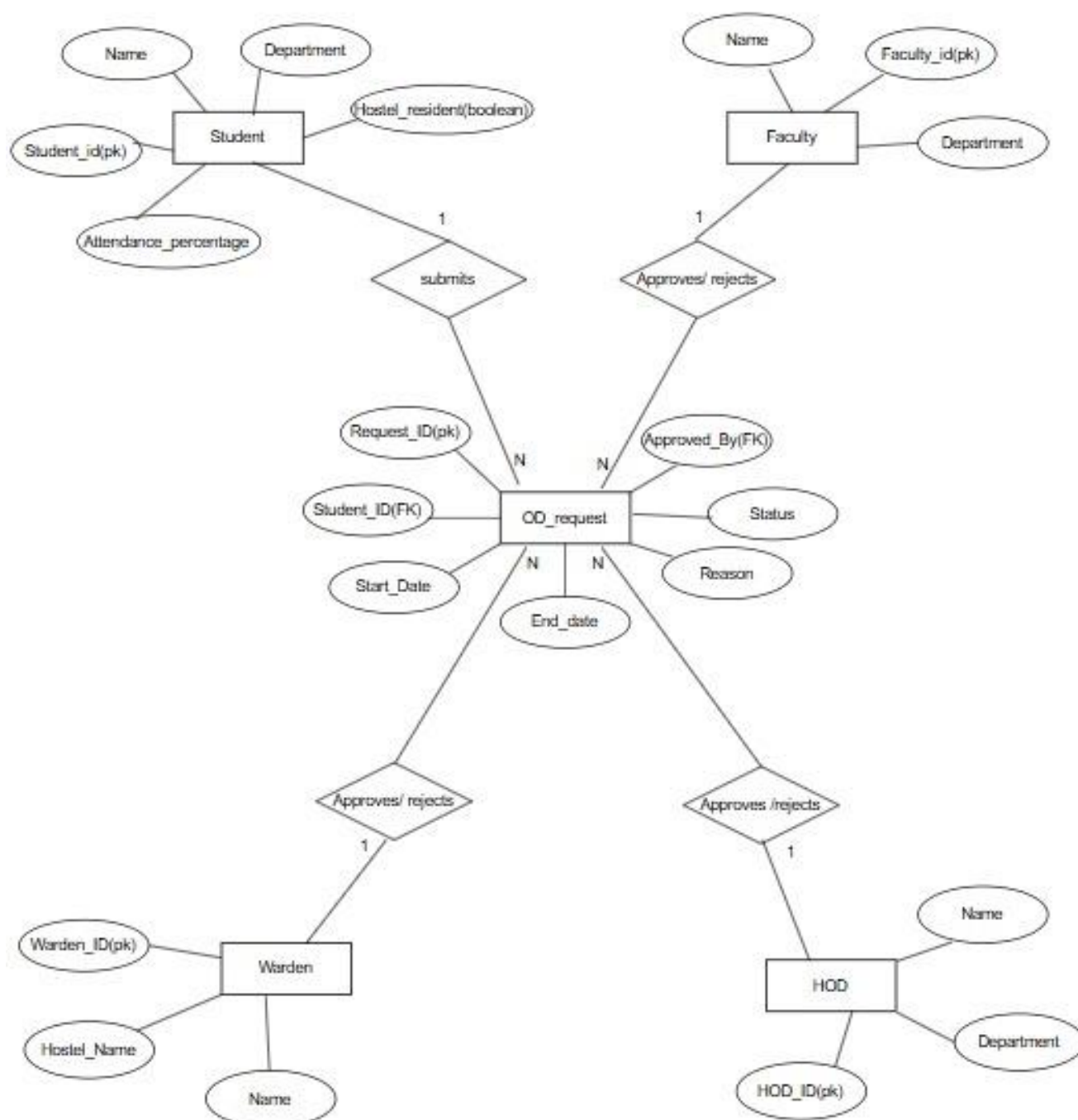
Required Python Libraries:

- mysql-connector-python,
- tkinter

3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



3.5 NORMALIZATION:

Normalization is a database design technique that organizes tables to reduce redundancy and dependency. Normalization involves dividing a database into two or more tables and defining relationships between the tables. The objective is to isolate data so that additions, deletions, and modifications can be made in a single table and then propagated through the rest of the database via relationships.

Steps of Normalization:

- 1. First Normal Form (1NF) :** Eliminate repeating groups: Ensure that each column contains atomic values, and each table has a primary key.
- 2. Second Normal Form (2NF):** Eliminate partial dependencies: Ensure that non-key attributes are fully functional dependent on the entire primary key.
- 3. Third Normal Form (3NF):** Eliminate transitive dependencies: Ensure that non-key attributes are dependent only on the primary key.
- 4. Boyce-Codd Normal Form (BCNF):** Stricter version of 3NF: Every determinant must be a candidate key.
- 5. Fourth Normal Form (4NF):** Eliminate multi-valued dependencies: Ensure no table contains two or more independent multi-valued facts about an entity.
- 6. Fifth Normal Form (5NF):** Isolate semantically related multiple relationships: Ensure that join dependencies are implied by candidate keys.

Unnormalized Form (UNF)

Initially, we have all data in a single table.

```
CREATE TABLE unnormalized_table (  
    request_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT,
```

```

user_id INT,

username VARCHAR(50),

password VARCHAR(50),

role ENUM('student', 'faculty', 'hod', 'warden'),

department VARCHAR(50),

name VARCHAR(100),

register_no VARCHAR(50),

day_scholar BOOLEAN,

attendance FLOAT,

reason TEXT,

proof_path VARCHAR(255),

status ENUM('pending', 'approved_by_faculty', 'approved_by_hod', 'rejected') DEFAULT 'pending');

```

In the unnormalized form, we would have all data in a single table. This table might contain repeating groups and nested records

RequestID	StudentID	UserID	Username	Password	Role	Department	Name
1	101	201	alice	pass123	student	CS	Alice
2	102	202	bob	pass456	student	CS	Bob
RegisterNo	DayScholar	Attendance	Reason	ProofPath	Status		
CS001	0	85%	Workshop	path/to/proof1	pending		
CS002	1	80%	Conference	path/to/proof2	pending		

First Normal Form (1NF)

In the first normal form, we eliminate repeating groups and ensure each column contains atomic values

Users Table

UserID	Username	Password	Role	Department
201	alice	pass123	student	CS
202	bob	pass456	student	CS

Students Table

StudentID	UserID	Name	Department	RegisterNo	DayScholar	Attendance
101	201	Alice	CS	CS001	0	85
102	202	Bob	CS	CS002	1	80

ODRequests Table

RequestID	StudentID	Reason	ProofPath	Status
1	101	Workshop	path/to/proof1	pending
2	102	Conference	path/to/proof2	pending

Second Normal Form (2NF)

In the second normal form, we eliminate partial dependencies. Each non-key attribute must be fully dependent on the entire primary key. The current tables already meet the requirements of 2NF because they have no partial dependencies.

Third Normal Form (3NF)

In the third normal form, we eliminate transitive dependencies. Each non-key attribute must depend only on the primary key.

Users Table

UserID	Username	Password	Role	Department
201	alice	pass123	student	CS
202	bob	pass456	student	CS

Already in 3NF as all attributes depend on the primary key user_id.

Students Table

Already in 3NF as all attributes depend on the primary key student_id.

ODRequests Table

Already in 3NF as all attributes depend on the primary key request_id.

Boyce-Codd Normal Form (BCNF)

BCNF is a stricter version of 3NF. Each determinant must be a candidate key. The tables already meet BCNF requirements because every determinant is a candidate key.

Fourth Normal Form (4NF)

In the fourth normal form, we eliminate multi-valued dependencies. The current schema does not have any multi-valued dependencies.

Fifth Normal Form (5NF)

In the fifth normal form, we ensure that join dependencies are implied by candidate keys. The current schema already satisfies this requirement.

Final Normalized Tables (5NF)

Users Table

UserID	Username	Password	Role	Department
201	alice	pass123	student	CS
202	bob	pass456	student	CS

```
CREATE TABLE users (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) UNIQUE,  
    password VARCHAR(50),  
    role ENUM('student', 'faculty', 'hod', 'warden'),  
    department VARCHAR(50)  
);
```

Students Table

StudentID	UserID	Name	Department	RegisterNo	DayScholar	Attendance
101	201	Alice	CS	CS001	0	85
102	202	Bob	CS	CS002	1	80

```
CREATE TABLE students (  
    student_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT,
```

```

name VARCHAR(100),

department VARCHAR(50),

register_no VARCHAR(50),

day_scholar BOOLEAN,

attendance FLOAT,

FOREIGN KEY (user_id) REFERENCES users(user_id)

);

```

ODRequests Table

RequestID	StudentID	Reason	ProofPath	Status
1	101	Workshop	path/to/proof1	pending
2	102	Conference	path/to/proof2	pending

```

CREATE TABLE od_requests (

    request_id INT AUTO_INCREMENT PRIMARY KEY,

    student_id INT,

    reason TEXT,

    proof_path VARCHAR(255),

    status ENUM('pending', 'approved_by_faculty', 'approved_by_hod', 'rejected') DEFAULT
'pending',

    FOREIGN KEY (student_id) REFERENCES students(student_id)

);

```

By following the normalization steps from UNF to 5NF, we've ensured that the database schema for the On-Duty Management System is efficient, free of redundancy, and maintains data integrity. This structured approach helps in maintaining consistency and making the database robust against anomalies.

4. PROGRAM CODE

PYTHON CODE :

```
import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import mysql.connector

import os

# Connect to MySQL Database

db = mysql.connector.connect(

    host="localhost",

    user="root",

    password="nandhu@171220",

    database="od_management")

cursor = db.cursor()

# Tkinter Window Setup

root = tk.Tk()

root.geometry("800x600")

root.title("OD Management System")

# Login Frame

login_frame = tk.Frame(root)

login_frame.pack()

tk.Label(login_frame, text="Username").grid(row=0, column=0)

tk.Label(login_frame, text="Password").grid(row=1, column=0)

tk.Label(login_frame, text="Department").grid(row=2, column=0)

username_entry = tk.Entry(login_frame)

password_entry = tk.Entry(login_frame, show="*")

department_entry = tk.Entry(login_frame)
```

```

username_entry.grid(row=0, column=1)

password_entry.grid(row=1, column=1)

department_entry.grid(row=2, column=1)

def login():

    username = username_entry.get()

    password = password_entry.get()

    department = department_entry.get()

    cursor.execute("SELECT user_id, role FROM users WHERE username=%s AND password=%s
AND department=%s", (username, password, department))

    result = cursor.fetchone()

    if result:

        user_id, role = result

        login_frame.pack_forget()

        if role == 'student':

            student_dashboard(user_id)

        elif role == 'faculty':

            faculty_dashboard(department)

        elif role == 'hod':

            hod_dashboard(department)

        elif role == 'warden':

            warden_dashboard()

    else:

        messagebox.showerror("Login Failed", "Invalid credentials")

tk.Button(login_frame, text="Login", command=login, bg='yellow').grid(row=3, column=0,
columnspan=2)

# Student Dashboard

def student_dashboard(user_id):

    student_frame = tk.Frame(root)

```

```
student_frame.pack()

tk.Label(student_frame, text="Name").grid(row=0, column=0)
tk.Label(student_frame, text="Department").grid(row=1, column=0)
tk.Label(student_frame, text="Register No").grid(row=2, column=0)
tk.Label(student_frame, text="Day Scholar").grid(row=3, column=0)
tk.Label(student_frame, text="Attendance").grid(row=4, column=0)
tk.Label(student_frame, text="Reason for OD").grid(row=5, column=0)
tk.Label(student_frame, text="Upload Proof").grid(row=6, column=0)
tk.Label(student_frame, text="OD Date").grid(row=7, column=0)

name_entry = tk.Entry(student_frame)
department_entry = tk.Entry(student_frame)
register_no_entry = tk.Entry(student_frame)
day_scholar_var = tk.IntVar()
day_scholar_check = tk.Checkbutton(student_frame, variable=day_scholar_var)
attendance_entry = tk.Entry(student_frame)
reason_text = tk.Text(student_frame, height=1, width=15)
proof_path_entry = tk.Entry(student_frame, state='readonly')
od_date_entry = tk.Entry(student_frame)

name_entry.grid(row=0, column=1)
department_entry.grid(row=1, column=1)
register_no_entry.grid(row=2, column=1)
day_scholar_check.grid(row=3, column=1)
attendance_entry.grid(row=4, column=1)
reason_text.grid(row=5, column=1)
proof_path_entry.grid(row=6, column=1)
od_date_entry.grid(row=7, column=1)
```

```

def upload_proof():

    file_path = filedialog.askopenfilename(filetypes=[("PDF files", "*.pdf")])

    if file_path:

        proof_path_entry.config(state='normal')

        proof_path_entry.delete(0, tk.END)

        proof_path_entry.insert(0, file_path)

        proof_path_entry.config(state='readonly')

tk.Button(student_frame, text="Browse", command=upload_proof).grid(row=6, column=2)

def submit_request():

    name = name_entry.get()

    department = department_entry.get()

    register_no = register_no_entry.get()

    day_scholar = day_scholar_var.get()

    attendance = float(attendance_entry.get())

    reason = reason_text.get("1.0", tk.END).strip()

    proof_path = proof_path_entry.get()

    od_date = od_date_entry.get()

    if not all([name, department, register_no, reason, proof_path, od_date]):

        messagebox.showerror("Error", "All fields are required")

        return

    cursor.execute("INSERT INTO students (user_id, name, department, register_no, day_scholar,
attendance) VALUES (%s, %s, %s, %s, %s, %s)", (user_id, name, department, register_no,
day_scholar, attendance))

    student_id = cursor.lastrowid

    cursor.execute("INSERT INTO od_requests (student_id, reason, proof_path, od_date) VALUES
(%s, %s, %s, %s)", (student_id, reason, proof_path, od_date))

    db.commit()

    messagebox.showinfo("Success", "Request Submitted")

```

```
tk.Button(student_frame, text="Submit", command=submit_request).grid(row=8, column=0,
columnspan=2)
```

```
# Status Check Section
```

```
def check_status():
```

```
    cursor.execute("""
```

```
        SELECT od_requests.reason, od_requests.status, od_requests.od_date
```

```
        FROM od_requests
```

```
        JOIN students ON od_requests.student_id = students.student_id
```

```
        WHERE students.user_id = %s """, (user_id,))
```

```
    requests = cursor.fetchall()
```

```
    status_window = tk.Toplevel(student_frame)
```

```
    status_window.title("Request Status")
```

```
    for idx, request in enumerate(requests):
```

```
        reason, status, od_date = request
```

```
        tk.Label(status_window, text=f"Request: {reason[:50]}...").grid(row=idx, column=0)
```

```
        tk.Label(status_window, text=f"Status: {status}").grid(row=idx, column=1)
```

```
        tk.Label(status_window, text=f"OD Date: {od_date}").grid(row=idx, column=2)
```

```
    tk.Button(student_frame, text="Check Request Status", command=check_status).grid(row=9,
column=0, columnspan=2)
```

```
# Faculty Dashboard
```

```
def faculty_dashboard(department):
```

```
    faculty_frame = tk.Frame(root)
```

```
    faculty_frame.pack()
```

```
    tk.Label(faculty_frame, text="Pending Requests").pack()
```

```
    pending_tree = ttk.Treeview(faculty_frame, columns=('Name', 'Department', 'Register No', 'Day
Scholar', 'Attendance', 'Reason', 'OD Date', 'Status'))
```

```
    pending_tree.heading('#0', text='Request ID')
```

```
    pending_tree.heading('Name', text='Name')
```

```
    pending_tree.heading('Department', text='Department')
```

```

pending_tree.heading('Register No', text='Register No')

pending_tree.heading('Day Scholar', text='Day Scholar')

pending_tree.heading('Attendance', text='Attendance')

pending_tree.heading('Reason', text='Reason')

pending_tree.heading('OD Date', text='OD Date')

pending_tree.heading('Status', text='Status')

pending_tree.pack(expand=True, fill='both')

tk.Label(faculty_frame, text="Approved Requests").pack()

approved_tree = ttk.Treeview(faculty_frame, columns=('Name', 'Department', 'Register No', 'Day
Scholar', 'Attendance', 'Reason', 'OD Date', 'Status'))

approved_tree.heading('#0', text='Request ID')

approved_tree.heading('Name', text='Name')

approved_tree.heading('Department', text='Department')

approved_tree.heading('Register No', text='Register No')

approved_tree.heading('Day Scholar', text='Day Scholar')

approved_tree.heading('Attendance', text='Attendance')

approved_tree.heading('Reason', text='Reason')

approved_tree.heading('OD Date', text='OD Date')

approved_tree.heading('Status', text='Status')

approved_tree.pack(expand=True, fill='both')

cursor.execute("""

    SELECT od_requests.request_id, students.name, students.department, students.register_no,
students.day_scholar, students.attendance, od_requests.reason, od_requests.od_date, od_requests.status

    FROM od_requests

    JOIN students ON od_requests.student_id = students.student_id

    WHERE students.department = %s AND (od_requests.status = 'pending' OR od_requests.status =
'approved_by_faculty')

    """, (department,))

requests = cursor.fetchall()

```



```

for request in requests:

    if request[-1] == 'pending':

        pending_tree.insert("", 'end', text=request[0], values=request[1:])

    else:

        approved_tree.insert("", 'end', text=request[0], values=request[1:])

def approve_request():

    selected_item = pending_tree.selection()

    if selected_item:

        request_id = pending_tree.item(selected_item, 'text')

        cursor.execute("UPDATE od_requests SET status='approved_by_faculty' WHERE
request_id=%s", (request_id,))

        db.commit()

        messagebox.showinfo("Success", "Request Approved")

        values = pending_tree.item(selected_item, 'values')

        approved_tree.insert("", 'end', text=request_id, values=values)

        pending_tree.delete(selected_item)

    else:

        messagebox.showwarning("No Selection", "Please select a request to approve.")

def reject_request():

    selected_item = pending_tree.selection()

    if selected_item:

        request_id = pending_tree.item(selected_item, 'text')

        cursor.execute("UPDATE od_requests SET status='rejected' WHERE request_id=%s",
(request_id,))

        db.commit()

        messagebox.showinfo("Success", "Request Rejected")

        pending_tree.delete(selected_item)

    else:

```

```

        messagebox.showwarning("No Selection", "Please select a request to reject.")

    def view_proof(tree):
        selected_item = tree.selection()

        if selected_item:
            request_id = tree.item(selected_item, 'text')

            cursor.execute("SELECT proof_path FROM od_requests WHERE request_id=%s",
                (request_id,))

            proof_path = cursor.fetchone()[0]

            if proof_path:
                os.system(f'xdg-open "{proof_path}"')
            else:
                messagebox.showwarning("No Proof", "No proof available for this request.")
        else:
            messagebox.showwarning("No Selection", "Please select a request to view proof.")

    tk.Button(faculty_frame, text="Approve", command=approve_request).pack()

    tk.Button(faculty_frame, text="Reject", command=reject_request).pack()

# HOD Dashboard
def hod_dashboard(department):
    hod_frame = tk.Frame(root)

    hod_frame.pack()

    tk.Label(hod_frame, text="Requests to Approve").pack()

    pending_tree = ttk.Treeview(hod_frame, columns=('Name', 'Department', 'Register No', 'Day
Scholar', 'Attendance', 'Reason', 'OD Date', 'Status'))

    pending_tree.heading('#0', text='Request ID')

    pending_tree.heading('Name', text='Name')

    pending_tree.heading('Department', text='Department')

    pending_tree.heading('Register No', text='Register No')

    pending_tree.heading('Day Scholar', text='Day Scholar')

```

```

pending_tree.heading('Attendance', text='Attendance')

pending_tree.heading('Reason', text='Reason')

pending_tree.heading('OD Date', text='OD Date')

pending_tree.heading('Status', text='Status')

pending_tree.pack(expand=True, fill='both')

tk.Label(hod_frame, text="Approved Requests").pack()

approved_tree = ttk.Treeview(hod_frame, columns=('Name', 'Department', 'Register No', 'Day
Scholar', 'Attendance', 'Reason', 'OD Date', 'Status'))

approved_tree.heading('#0', text='Request ID')

approved_tree.heading('Name', text='Name')

approved_tree.heading('Department', text='Department')

approved_tree.heading('Register No', text='Register No')

approved_tree.heading('Day Scholar', text='Day Scholar')

approved_tree.heading('Attendance', text='Attendance')

approved_tree.heading('Reason', text='Reason')

approved_tree.heading('OD Date', text='OD Date')

approved_tree.heading('Status', text='Status')

approved_tree.pack(expand=True, fill='both')

cursor.execute("""

SELECT od_requests.request_id, students.name, students.department, students.register_no,
students.day_scholar, students.attendance, od_requests.reason, od_requests.od_date, od_requests.status

FROM od_requests

JOIN students ON od_requests.student_id = students.student_id

WHERE students.department = %s AND (od_requests.status = 'approved_by_faculty' OR
od_requests.status = 'approved_by_hod'  """, (department,))

requests = cursor.fetchall()

for request in requests:

    if request[-1] == 'approved_by_faculty':

```

```

        pending_tree.insert("", 'end', text=request[0], values=request[1:])

    else:

        approved_tree.insert("", 'end', text=request[0], values=request[1:])

def approve_request():

    selected_item = pending_tree.selection()

    if selected_item:

        request_id = pending_tree.item(selected_item, 'text')

        cursor.execute("UPDATE od_requests SET status='approved_by_hod' WHERE
request_id=%s", (request_id,))

        db.commit()

        messagebox.showinfo("Success", "Request Approved")

        values = pending_tree.item(selected_item, 'values')

        approved_tree.insert("", 'end', text=request_id, values=values)

        pending_tree.delete(selected_item)

    else:

        messagebox.showwarning("No Selection", "Please select a request to approve.")

def reject_request():

    selected_item = pending_tree.selection()

    if selected_item:

        request_id = pending_tree.item(selected_item, 'text')

        cursor.execute("UPDATE od_requests SET status='rejected' WHERE request_id=%s",
(request_id,))

        db.commit()

        messagebox.showinfo("Success", "Request Rejected")

        pending_tree.delete(selected_item)

    else:

        messagebox.showwarning("No Selection", "Please select a request to reject.")

tk.Button(hod_frame, text="Approve", command=approve_request).pack()

```

```

tk.Button(hod_frame, text="Reject", command=reject_request).pack()

# Warden Dashboard

def warden_dashboard():

    warden_frame = tk.Frame(root)

    warden_frame.pack()

    tk.Label(warden_frame, text="Approved Requests").pack()

    tree = ttk.Treeview(warden_frame, columns=('Name', 'Department', 'Register No', 'Day Scholar',
'Attendance', 'Reason', 'OD Date', 'Status'))

    tree.heading('#0', text='Request ID')

    tree.heading('Name', text='Name')

    tree.heading('Department', text='Department')

    tree.heading('Register No', text='Register No')

    tree.heading('Day Scholar', text='Day Scholar')

    tree.heading('Attendance', text='Attendance')

    tree.heading('Reason', text='Reason')

    tree.heading('OD Date', text='OD Date')

    tree.heading('Status', text='Status')

    tree.pack(expand=True, fill='both')

    cursor.execute("""

        SELECT od_requests.request_id, students.name, students.department, students.register_no,
students.day_scholar, students.attendance, od_requests.reason, od_requests.od_date, od_requests.status

        FROM od_requests

        JOIN students ON od_requests.student_id = students.student_id

        WHERE od_requests.status = 'approved_by_hod' AND students.day_scholar = 0""")

    requests = cursor.fetchall()

    for request in requests:

        tree.insert("", 'end', text=request[0], values=request[1:])

root.mainloop()

```

MYSQL CODE :

```
CREATE DATABASE od_management;
```

```
USE od_management;
```

```
CREATE TABLE users (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) UNIQUE,  
    password VARCHAR(50),  
    role ENUM('student', 'faculty', 'hod', 'warden'),  
    department VARCHAR(50)  
);
```

```
CREATE TABLE students (  
    student_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT,  
    name VARCHAR(100),  
    department VARCHAR(50),  
    register_no VARCHAR(50),  
    day_scholar BOOLEAN,  
    attendance FLOAT,  
    FOREIGN KEY (user_id) REFERENCES users(user_id)  
);
```

```
CREATE TABLE od_requests (  
    request_id INT AUTO_INCREMENT PRIMARY KEY,  
    student_id INT,  
    reason TEXT,
```

proof_path VARCHAR(255),

od_date DATE ,

status ENUM('pending', 'approved_by_faculty', 'approved_by_hod', 'rejected') DEFAULT
'pending',

FOREIGN KEY (student_id) REFERENCES students(student_id)

);

INSERT into users (username, password, role, department) values('student1', 'password1', 'student',
'CSE');

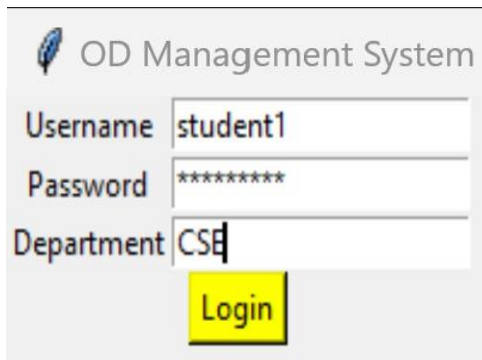
INSERT into users (username, password, role, department) values('faculty1', 'password1', 'faculty',
'CSE');

INSERT into users (username, password, role, department) values('hod1', 'password1', 'hod', 'CSE');

INSERT into users (username, password, role, department) values('warden1', 'password1', 'warden',
'Hostel');

5. RESULTS

STUDENT MODULE:



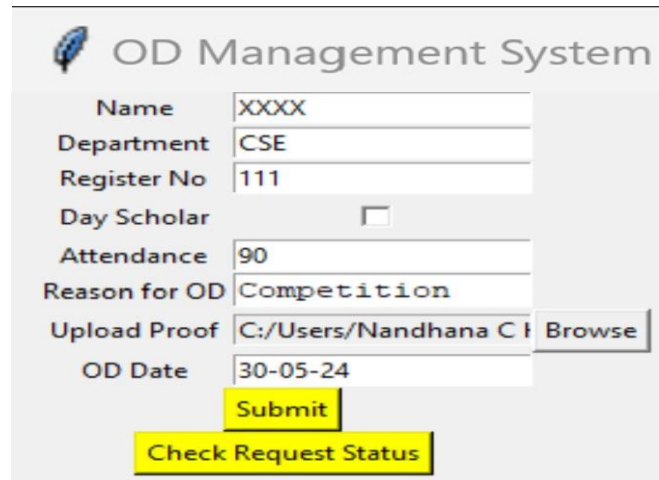
OD Management System

Username: student1

Password: *****

Department: CSE

Login



OD Management System

Name: XXXX

Department: CSE

Register No: 111

Day Scholar: ☐

Attendance: 90

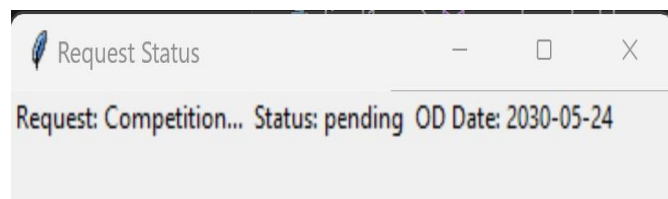
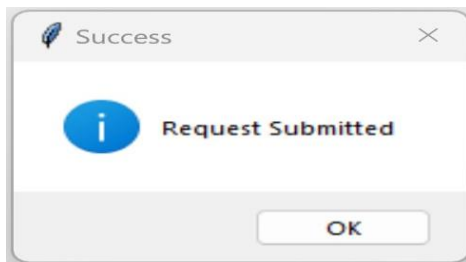
Reason for OD: Competition

Upload Proof: C:/Users/Nandhana C | Browse

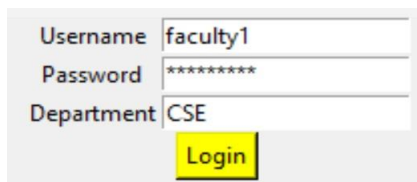
OD Date: 30-05-24

Submit

Check Request Status



FACULTY MODULE:

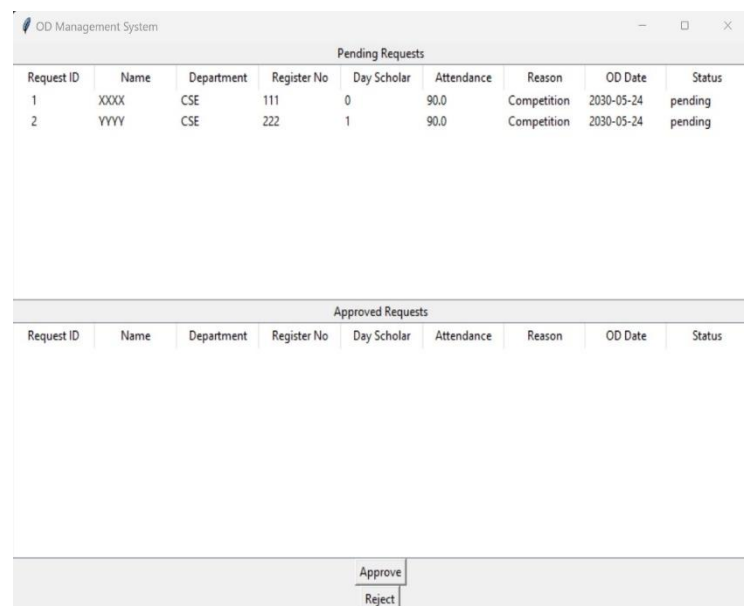


Username: faculty1

Password: *****

Department: CSE

Login



OD Management System

Pending Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
1	XXXX	CSE	111	0	90.0	Competition	2030-05-24	pending
2	YYYY	CSE	222	1	90.0	Competition	2030-05-24	pending

Approved Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
1	XXXX	CSE	111	0	90.0	Competition	2030-05-24	approved_by_fa

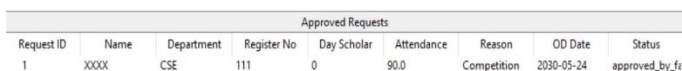
Approve

Reject



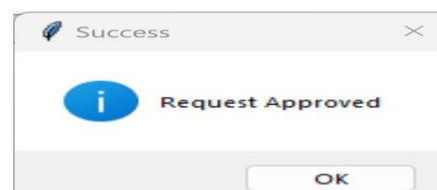
OD Management System

Pending Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
2	YYYY	CSE	222	1	90.0	Competition	2030-05-24	pending



OD Management System

Approved Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
1	XXXX	CSE	111	0	90.0	Competition	2030-05-24	approved_by_fa



HOD MODULE:

Username

Password


Department


Requests to Approve								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
1	XXXX	CSE	111	0	90.0	Competition	2030-05-24	approved_by_fe
2	YYYY	CSE	222	1	90.0	Competition	2030-05-24	approved_by_fe

Approved Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status

Requests to Approve								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status

Approved Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
2	YYYY	CSE	222	1	90.0	Competition	2030-05-24	approved_by_fe

 Success

 Request Rejected

OK

WARDEN MODULE:

Username

Password

Department

Approved Requests								
Request ID	Name	Department	Register No	Day Scholar	Attendance	Reason	OD Date	Status
3	ZZZZ	CSE	333	0	90.0	Competition	2030-05-24	approved_by_hc

STUDENT MODULE (REQUEST APPROVED)

 Request Status

Request: Competition... Status: approved_by_hod OD Date: 2030-05-24

6. CONCLUSION

The On-Duty Management System (ODMS) project successfully addresses the need for a streamlined and efficient process for managing on-duty requests in a college environment. By leveraging Python with Tkinter for the frontend and MySQL for the backend database, the system provides a user-friendly interface and robust data management capabilities.

The implementation of ODMS offers several key benefits:

1. **Efficiency:** Automates the process of submitting, reviewing, and approving on-duty requests, reducing the administrative burden on staff and eliminating the need for manual paperwork.
2. **Accuracy:** Ensures that student attendance records are accurately checked before approving on-duty requests, helping to maintain academic standards.
3. **Transparency and Accountability:** Tracks and logs every request and decision, ensuring that students and staff can monitor the status and history of each request. This transparency fosters trust and accountability within the institution.
4. **User-Friendly Interface:** The intuitive Tkinter-based GUI ensures that users of all technical backgrounds can navigate the system with ease, improving user satisfaction and engagement.
5. **Scalability:** The use of MySQL allows the system to handle large volumes of data and complex queries, making it scalable to accommodate the growing needs of the institution.

Overall, the ODMS project demonstrates the effective use of modern software development technologies to solve real-world problems in educational administration. The system not only simplifies the on-duty request process but also enhances the overall operational efficiency of the college. By integrating automation, data accuracy, and user-friendly design, the ODMS sets a precedent for future technological improvements in academic management systems.

This project underscores the importance of leveraging technology to enhance administrative processes, ultimately contributing to a more organized, efficient, and student-friendly educational environment.

7. REFERENCES

1. John Zelle, "Python Programming: An Introduction to Computer Science," Second Edition, Franklin, Beedle & Associates Inc., 2010.
2. Michael Urban, Joel Murach, "Murach's MySQL," Mike Murach & Associates, 2012.
3. Grzegorz Kraszewski, "Tkinter GUI Application Development Blueprints," Packt Publishing, 2015.