BONAFIDE CERTIFICATE

Certified that this project report **"STUDENT MANAGEMENT SYSTEM"** is the bonafide work of **"ASWIN K (231501027), ASWIN J (231501026) HARSITH C M (231501061),ASAWANTH A (231501025). "** who carried out the project work under my supervision.

Submitted for the Practical Examination held on ______________________

SIGNATURE

Mr. U. Kumaran,
Assistant Professor (SS)
AIML,
Rajalakshmi Engineering College (Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

STUDENT MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

K ASWIN	**231501027**
ASWANTH A	**231501025**
HARSITH C M	**231501061**
ASWIN J	**231501026**

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

RAJALAKSHMI ENGINEERING COLLEGE

(AUTONOMOUS) THANDALAM

CHENNAI-602105

2024 - 2025

1.INTRODUCTION

1.1 INTRODUCTION TO PROJECT

The College Resource Management System (CRMS) is a comprehensive digital solution designed to streamline the allocation, management, and tracking of resources within a college or university setting. Colleges often require efficient use of shared resources—such as classrooms, projectors, laptops, and lab equipment—to support various academic and administrative activities. The CRMS centralizes the management of these resources by allowing staff members to request resources based on need, while administrators can easily approve, allocate, and monitor their usage.

This system simplifies the resource request and approval process, reduces resource conflicts, and provides an organized, transparent system for all stakeholders. With CRMS, staff members can submit resource requests, view the availability of resources, and specify their needs for specific periods, thus facilitating effective planning and time management. On the administrative side, CRMS offers an efficient way to approve requests, allocate resources, and monitor returns, which is particularly useful for maintaining resource availability and planning maintenance schedules.

Built with a user-friendly interface, the CRMS provides a seamless interaction experience, supported by backend functionalities that ensure data consistency and accuracy. Additionally, CRMS includes automated features such as status updates, approval notifications, and overdue alerts, making it an invaluable tool for colleges aiming to enhance productivity and resource utilization within their institutions.

1.2 OBJECTIVES:

- **Efficient Resource Allocation:**
  Streamline the process of allocating resources such as classrooms, lab equipment, projectors, and laptops, ensuring they are assigned based on availability and priority.
- **Centralized Management:**
  Provide a centralized platform for tracking and managing resource requests, approvals, and usage, making it easier for administrators to monitor and control resources effectively.
- **Improved Transparency:**
  Enhance transparency by enabling staff members to view the status of their requests and the availability of resources, thereby reducing conflicts and confusion.
- **Optimized Resource Utilization:**
  Maximize the usage of resources by minimizing idle time, scheduling maintenance effectively, and ensuring that resources are returned promptly.
- **Automated Approval and Notification Process:**
  Automate the request approval, status updates, and notification process, reducing manual effort for administrators and ensuring timely updates for staff.
- **Conflict Prevention and Resolution:**
  Minimize resource conflicts through a system that flags overlapping requests or unavailable resources, helping to avoid scheduling issues and potential disputes.
- **Data-Driven Decision Making:**
  Provide data analytics and reporting capabilities to help administrators understand usage patterns, identify high-demand resources, and make informed decisions for future resource planning and procurement.

- **Cost Efficiency:**
  Support cost-saving initiatives by enabling more effective use of available resources, reducing the need for redundant purchases, and lowering maintenance costs through regular tracking.
- **Enhanced User Experience:**
  Offer an intuitive, user-friendly interface that enables quick and easy access for staff to request resources and for administrators to manage allocations, with minimal training required.
- **Compliance and Accountability:**
  Establish clear accountability by tracking resource allocations, requests, and returns, and generating logs for administrative review or auditing purposes.

2. STUDY OF TECHNOLOGIES

1. SOFTWARE DESCRIPTION AND FEATURES:

The **College Resource Management System (CRMS)**, when implemented using **Python** and **MySQL**, operates as a software application designed to facilitate resource allocation and management within a college environment. This system leverages Python as the primary programming language for backend development, connecting to a MySQL database for data storage and retrieval, ensuring robust, efficient data management.

Features :

Resource Request by Staff:

- Staff log in to their accounts and request resources, specifying details like the type of resource, priority, and required date.
- The request is recorded in the MySQL database and is visible to administrators on their dashboards.

Resource Approval and Allocation by Admin:

- Administrators receive requests, check the availability of resources, and approve or reject them. Approval triggers the allocation process, changing the resource's status and updating the database.
- Upon allocation, staff members are notified of their request's approval, including details like the allocation and return dates.

Resource Monitoring and Return:

- The system tracks each resource's status and sends reminders as the return date approaches. When a resource is returned, the system updates its status in the database, marking it as available for future requests.

Reporting and Analytics:

- Administrators can generate reports that analyze resource usage, identify high-demand items, and view request trends, helping to optimize resource planning and allocation

2. LANGUAGES USED:

1. DATABASE MANAGEMENT - MySQL:

- MySQL acts as the database for storing all CRMS data, including staff details, resource inventory, resource requests, allocation logs, and historical data for audit and reporting purposes.
- The relational schema in MySQL is designed to handle the relationships between entities, like linking resources to requests and allocations to staff.
- SQL stored procedures, triggers, and PL/SQL functions can manage critical operations directly in the database, ensuring data integrity and enforcing rules, like updating resource status upon allocation or return.

2. BACKEND - PYTHON:

- Python serves as the backend logic layer, managing requests from the frontend and processing interactions with the MySQL database.
- Using an ORM (Object-Relational Mapping) like SQLAlchemy (for Flask) makes it easier to perform database operations by translating Python objects into database entries.
- The backend includes modules for different functionalities, such as Request Management, Resource Allocation, User Authentication, and Notification Handling.
- Python libraries like **Flask-Login** manage user authentication and access control, differentiating between staff members, administrators, and other roles.

- Password hashing and encryption are applied to secure user credentials, and role-based access control ensures only authorized users perform actions like resource approvals or view sensitive data

2.2.3 FRONTEND:

- The user interface for CRMS has been developed using Python framework Flask which provide web-based, intuitive, and interactive user interfaces.
- The frontend will present dashboards for administrators and staff, where each user type has specific functionalities, such as requesting resources, viewing status, and approving or allocating resources.
- HTML and CSS, have been used to enhance user experience by providing a visually appealing and responsive design, ensuring accessibility across various devices.

3. REQUIREMENTS AND ANALYSIS

1. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements

1. Server Requirements (if deployed on-premises):
   - Processor: Intel Xeon or AMD equivalent (Quad-Core or higher)
   - RAM: Minimum 8 GB (16 GB recommended for higher user volumes)
   - Storage: SSD storage with at least 100 GB (expandable based on data volume)
   - Network: High-speed network interface (1 Gbps or higher recommended)
   - Backup: External storage or cloud backup for database and system logs
2. Development Machines (for local development and testing):
   - Processor: Intel i5/i7 or AMD Ryzen equivalent
   - RAM: Minimum 4 GB (8 GB recommended)
   - Storage: 10 GB available storage (preferably SSD)
   - Operating System: Windows 10/11, macOS, or Linux (Ubuntu, CentOS)
3. Client Machines (User Access):
   - Processor: Dual-Core or higher
   - RAM: Minimum 4 GB
   - Operating System: Any OS with a modern browser (Windows, macOS, Linux)
   - Web Browser: Chrome, Firefox, Safari, or Edge (latest versions recommended)

Software Requirements:

1. Backend Development:
    - Programming Language: Python (version 3.8 or higher)
    - Framework: Flask (for lightweight development) or Libraries:
        - SQLAlchemy (ORM for Flask)
        - Flask-Mail
        - Pandas and Matplotlib (for data analysis and reporting)
    - Environment: Virtualenv or Conda for dependency management
2. Database:
    - Database Server: MySQL (version 8.0 or higher recommended) or MariaDB as an alternative
    - Database Management Tool: MySQL Workbench (for database management and testing)
3. Frontend Development:
    - HTML, CSS, JavaScript: For basic user interface design
    - Frontend Libraries: Bootstrap (for responsive design), jQuery (for enhanced interactivity)
    - Template Engine: Jinja2 (for Flask)
4. Application Server (for deployment):
    - Web Server: Apache or Nginx (with uWSGI or Gunicorn for serving Python apps)
    - Operating System: Ubuntu 20.04 LTS or CentOS 8 (recommended for deployment)
5. Development and Testing Tools:
    - IDE/Text Editor: VS Code, PyCharm, or Sublime Text (for code development)
    - Version Control: Git (with GitHub or GitLab for collaboration and code backup)

    - Testing Tools: PyTest (for unit testing), Postman (for API testing)
6. Security:
    - SSL Certificate: For secure HTTPS access to the web application
    - Firewall: Configured on the server to protect against unauthorized access
    - Authentication: Implement OAuth or JWT (JSON Web Tokens) if additional security layers are needed
7. Other Recommended Software:
    - Backup Solutions: Automated daily backup tools for database and server data (e.g., rsync for Linux, or cloud storage solutions)
    - Monitoring: Tools like New Relic, Grafana, or simple logging tools for monitoring application performance and resource usage
    - Notification System: SMTP server configuration (e.g., Gmail SMTP or a dedicated SMTP service) to send email notifications to users

3.2 ENTITIES,RELATION SCHEMA AND RELATIONSHIP:

Entities:

1. **Staff:**

   Each staff member can request multiple resources.
2. **Resource:**

   Different types of resources like rooms, projectors, laptops, etc**.**
3. **Request:**

   Staff members submit requests for resources**.**
4. **Allocation:**

   Each request can result in one or more resource allocations.

Relational Schema:

1. **Staff:**

   Represents the staff members who can request resource**s.**

Schema:

```
Staff(
staff_id      INT PRIMARY KEY,
name          VARCHAR(100),
department    VARCHAR(100),
email         VARCHAR(100) UNIQUE,
phone_number  VARCHAR(15)
)
```

2. **Resource:**

Stores the information about the resources available for allocation.

Schema:

```
Resource(
resource_id   INT PRIMARY KEY,
resource_type VARCHAR(50),  -- e.g., room, projector, laptop
resource_name VARCHAR(100), -- unique name/identifier for the resource
status        VARCHAR(20),  -- e.g., available, allocated,under_maintenance
description   TEXT          -- additional details about the resource
)
```

3. **Request:**

Represents the request made by a staff member for resources.

Schema:

```
Request(
request_id    INT PRIMARY KEY,
staff_id      INT,          -- FK referencing Staff
request_date  DATE,
request_status VARCHAR(20),        -- e.g., pending, approved, rejected
priority       VARCHAR(20),        -- e.g., high, medium, low
required_date  DATE,
return_date    DATE,
FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
)
```

4. **Request_resource:**

Represents the resources requested in each request (one-to-many between Request and Resource).

Schema:

```
Requested_Resources(
request_id    INT,              -- FK referencing Request
resource_id   INT,              -- FK referencing Resource
quantity      INT,
FOREIGN KEY (request_id) REFERENCES Request(request_id),
FOREIGN KEY (resource_id) REFERENCES Resource(resource_id),
PRIMARY KEY (request_id, resource_id)
)
```

5. **Allocation:**

Represents the allocation of resources based on the request made by staff.

Schema:

```
Allocation(
allocation_id  INT PRIMARY KEY,
request_id     INT,             -- FK referencing Request
resource_id    INT,             -- FK referencing Resource
allocation_date DATE,
return_date     DATE,
status          VARCHAR(20),      -- e.g., allocated, returned, late
FOREIGN KEY (request_id) REFERENCES Request(request_id),
FOREIGN KEY (resource_id) REFERENCES Resource(resource_id)
)
```

Relationships:

- Staff to Request is a one-to-many relationship: each staff member can make multiple requests.
- Request to Requested_Resources is a one-to-many relationship: each request can involve multiple resources.
- Requested_Resources to Resource is a many-to-one relationship: each resource can be requested in multiple requests.
- Request to Allocation is a one-to-one or one-to-many relationship: each request can result in one or more resource allocations.
- Resource to Allocation is a one-to-one or one-to-many relationship: each resource can be allocated multiple times over time.

Optional Add-ons:

1. **Audit:**

   To track changes in resource allocations (for history or reporting).

   Schema:

```
Audit(
audit_id     INT PRIMARY KEY,
allocation_id  INT,              -- FK referencing Allocation
action         VARCHAR(20),        -- e.g., allocated, returned, updated
action_date    DATE,
FOREIGN KEY (allocation_id) REFERENCES Allocation(allocation_id)
)
```

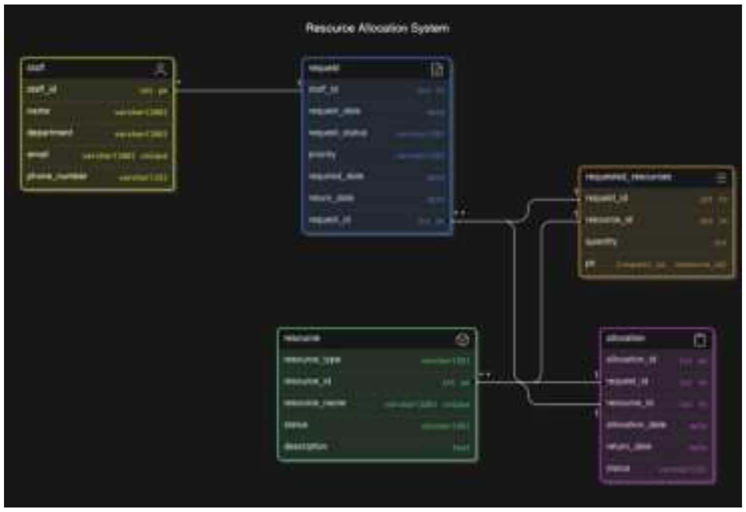
2. Maintenance:

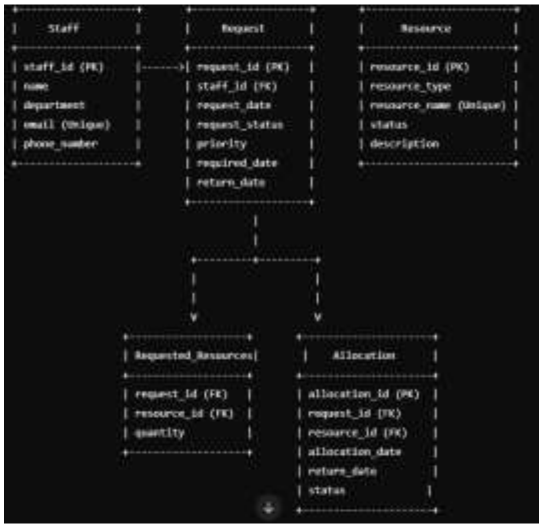
To track resources under maintenance.

Schema:

```
Maintenance(
maintenance_id   INT PRIMARY KEY,
resource_id      INT,              -- FK referencing Resource
start_date       DATE,
end_date         DATE,
maintenance_desc TEXT,
FOREIGN KEY (resource_id) REFERENCES Resource(resource_id)
)
```

3.3 ER DIAGRAM





4. PROGRAM CODE

PYTHON BACKEND CODE:

```
from flask import Flask, render_template, request, redirect,
url_for, flash, session
import mysql.connector
from   werkzeug.security   import   generate_password_hash,
check_password_hash
import hashlib
from config import Config
from datetime import datetime


app = Flask(__name__)
app.config.from_object(Config)
app.secret_key = Config.SECRET_KEY  # Required for session
management


# Database connection function
def get_db_connection():
   return mysql.connector.connect(
       host=Config.MYSQL_HOST,
       user=Config.MYSQL_USER,
       password=Config.MYSQL_PASSWORD,
       database=Config.MYSQL_DB
   )
def hash_password(password):
   return hashlib.sha256(password.encode()).hexdigest()
```

```
@app.route('/')
def index():
   return redirect(url_for('login'))


# Login page with separate admin and staff login
@app.route('/login', methods=['GET', 'POST'])
def login():
   if request.method == 'POST':
       role = request.form['role']   # 'admin' or 'staff'
       username_or_id = request.form['username_or_id']
       password = request.form['password'].strip()



       conn = get_db_connection()
       cur = conn.cursor(dictionary=True)

       try:
           if role == 'admin':
               # Fetch admin by username with parameterized
query
               cur.execute("SELECT admin_id, password FROM
Admin_Login WHERE admin_id = %s", (username_or_id,))
              admin = cur.fetchone()
                  print("Admin query result:", admin)   #
Debugging print
```

```
                # Check if admin exists and password matches
                if admin:
                              print("Stored hash for admin:",
admin['password'])   # Debugging print
                    if admin['password'] == password:
                                          session['user_id']  =
admin['admin_id']
                        session['user_role'] = 'admin'
                                        flash("Admin  logged  in
successfully.")
                                                            return
redirect(url_for('admin_dashboard'))
                    else:
                                flash("Incorrect password for
admin.")
                else:
                    flash("Admin username not found.")


            elif role == 'staff':
                 # Fetch staff by staff_id with parameterized
query
                 cur.execute("SELECT staff_id, password FROM
Staff_Login WHERE staff_id = %s", (username_or_id,))
                staff = cur.fetchone()
                     print("Staff query result:", staff)   #
Debugging print
```

```
                # Check if staff exists and password matches
                if staff:
                              print("Stored hash for staff:",
staff['password'])   # Debugging print
                    if staff['password'] == password:
                                          session['user_id']  =
staff['staff_id']
                        session['user_role'] = 'staff'
                                         flash("Staff  logged  in
successfully.")
                                                           return
redirect(url_for('staff_dashboard'))
                    else:
                                flash("Incorrect password for
staff.")
                else:
                    flash("Staff ID not found.")



            else:
                flash("Invalid role selected.")


        finally:
            cur.close()
            conn.close()


    return render_template('login.html')
```

```
# Admin dashboard
@app.route('/add_staff', methods=['POST'])
def add_staff():
   if 'user_role' not in session or session['user_role'] !=
'admin':
       flash("Access denied.")
       return redirect(url_for('login'))

   # Get form data
   staff_id = request.form['staff_id']
   name = request.form['name']
   department = request.form['department']
   email = request.form['email']
   phone_number = request.form['phone_number']
   password = request.form['password']

   # Hash the password


   conn = get_db_connection()
   cur = conn.cursor()

   try:
       # Insert into Staff table
       cur.execute(
            "INSERT INTO Staff (staff_id, name, department,
email, phone_number) VALUES (%s, %s, %s, %s, %s)",
           (staff_id, name, department, email, phone_number)
```

```
        )

        # Insert into Staff_Login table
        cur.execute(
                "INSERT INTO Staff_Login (staff_id, password)
VALUES (%s, %s)",
            (staff_id, password)
        )

        conn.commit()
        flash("Staff member added successfully!")
    except Exception as e:
        conn.rollback()
         flash("Error adding staff member. Please check data
and try again.")
        print(f"Error adding staff: {e}")
    finally:
        cur.close()
        conn.close()

    return redirect(url_for('admin_dashboard'))


# Display requests and allocation details for admin
@app.route('/admin_dashboard')
def admin_dashboard():
    if 'user_role' not in session or session['user_role'] !=
'admin':
        flash("Access denied.")
        return redirect(url_for('login'))


    conn = get_db_connection()
    cur = conn.cursor(dictionary=True)
    try:
        # Fetch all pending requests with staff details
        cur.execute("""
                 SELECT Request.request_id, Request.staff_id,
Request.request_date, Request.request_status,
                      Request.priority, Request.required_date,
Request.return_date, Staff.name AS staff_name
            FROM Request
            JOIN Staff ON Request.staff_id = Staff.staff_id
            WHERE Request.request_status = 'pending'
        """)
        requests = cur.fetchall()


        # Fetch allocation details
        cur.execute("""
                              SELECT  Allocation.allocation_id,
Allocation.request_id, Allocation.resource_id,
                                    Allocation.allocation_date,
Allocation.return_date, Allocation.status,
                    Resource.resource_name
            FROM Allocation
                  JOIN  Resource  ON  Allocation.resource_id  =
Resource.resource_id
```

```
        """)
        allocations = cur.fetchall()
    finally:
        cur.close()
        conn.close()


            return    render_template('admin_dashboard.html',
requests=requests, allocations=allocations)


# Handle resource allocation
@app.route('/allocate_resource', methods=['POST'])
def allocate_resource():
    if 'user_role' not in session or session['user_role'] !=
'admin':
        flash("Access denied.")
        return redirect(url_for('login'))


    allocation_data = request.form
    request_id = allocation_data['request_id']
    resource_id = allocation_data['resource_id']
    allocation_date = allocation_data['allocation_date']
    return_date = allocation_data['return_date']


    conn = get_db_connection()
    cur = conn.cursor()
    try:
        # Insert allocation details
        cur.execute(
            "INSERT INTO Allocation (request_id, resource_id,
allocation_date,  return_date, status) VALUES (%s, %s, %s,
%s, %s)",
                  (request_id, resource_id, allocation_date,
return_date, 'allocated')
        )
        # Update resource status to 'allocated'
          cur.execute("UPDATE Resource SET status  = %s WHERE
resource_id = %s", ('allocated', resource_id))
        conn.commit()
        flash("Resource allocated successfully!")
    except Exception as e:
        flash("Error allocating resource.")
        print(f"Allocation error: {e}")
    finally:
        cur.close()
        conn.close()


    return redirect(url_for('admin_dashboard'))


@app.route('/staff_dashboard', methods=['GET', 'POST'])
def staff_dashboard():
     if 'user_role' in session and session['user_role'] ==
'staff':
        staff_id = session['user_id']

        conn = get_db_connection()
        cur = conn.cursor(dictionary=True)

        # Fetch available resources for dropdown selection in
the request form
         cur.execute("SELECT resource_id, resource_name FROM
Resource WHERE status = 'available'")
        resources = cur.fetchall()


        # Fetch staff's existing requests
         cur.execute("SELECT * FROM Request WHERE staff_id =
%s", (staff_id,))
        requests = cur.fetchall()


           # Fetch allocation details linked to the staff's
requests
        cur.execute("""
                     SELECT a.allocation_id, a.request_id,
a.resource_id, r.resource_name, a.allocation_date,
                  a.return_date, a.status
            FROM Allocation a
            JOIN Resource r ON a.resource_id = r.resource_id
            JOIN Request req ON a.request_id = req.request_id
            WHERE req.staff_id = %s
        """, (staff_id,))
        allocations = cur.fetchall()


        cur.close()
        conn.close()

              return  render_template('staff_dashboard.html',
requests=requests,                            resources=resources,
allocations=allocations)

    flash("Access denied.")
    return redirect(url_for('login'))
# Route to handle request submission
@app.route('/submit_request', methods=['POST'])
def submit_request():
     if  'user_role'  in  session  and  session['user_role']  ==
'staff':
        staff_id = session['user_id']
        priority = request.form['priority']
        required_date = request.form['required_date']
        return_date = request.form['return_date']
        resource_id = request.form['resource_id']
        quantity = request.form['quantity']


        conn = get_db_connection()
        cur = conn.cursor()

        # Insert new request into the Request table
        cur.execute("""
                INSERT INTO Request (staff_id, request_date,
request_status, priority, required_date, return_date)
            VALUES (%s, %s, %s, %s, %s, %s)
           """, (staff_id, datetime.now().date(), 'pending',
priority, required_date, return_date))

           # Get the new request_id for linking requested
resources
        request_id = cur.lastrowid


          # Insert resource request into Requested_Resources
table
        cur.execute("""
                INSERT INTO Requested_Resources (request_id,
resource_id, quantity)
            VALUES (%s, %s, %s)
        """, (request_id, resource_id, quantity))

        conn.commit()
        cur.close()
        conn.close()

        flash("Request submitted successfully!")
        return redirect(url_for('staff_dashboard'))


    flash("Access denied.")
    return redirect(url_for('login'))


@app.route('/logout')
def logout():
    session.clear()
    flash("Logged out successfully.")
    return redirect(url_for('login'))
if __name__ == '__main__':
    app.run(debug=True,port=3000)
```

SQL INTERCONNECTIVITY CODE:

```
# config.py
import os
class Config:
    MYSQL_HOST = 'localhost'
    MYSQL_USER = 'root'
    MYSQL_PASSWORD = 'root@123'
    MYSQL_DB = 'ResourceAllocation'
    SECRET_KEY = os.urandom(24)  # Secret key for session
management
```

HTML AND CSS FRONTEND CODE:

Login page:

```
<!-- templates/login.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
```

```
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css
/bootstrap.min.css">
</head>
<body>
   <div class="container mt-5">
       <div class="row justify-content-center">
           <div class="col-md-4">
               <h2 class="text-center">Login</h2>
               {% with messages = get_flashed_messages() %}
                 {% if messages %}
                   <div class="alert alert-info">{{
messages[0] }}</div>
                 {% endif %}
               {% endwith %}
               <form method="POST" action="{{
url_for('login') }}">
                   <div class="form-group">
                       <label for="role">Role</label>
                       <select class="form-control"
id="role" name="role" required>
                           <option
value="admin">Admin</option>
                           <option
value="staff">Staff</option>
                       </select>
                   </div>
                   <div class="form-group">
```

```
                        <label for="username_or_id">Username
or ID</label>
                        <input type="text"
class="form-control" id="username_or_id"
name="username_or_id" required>
                    </div>
                    <div class="form-group">
                        <label
for="password">Password</label>
                        <input type="password"
class="form-control" id="password" name="password" required>
                    </div>
                    <button type="submit" class="btn
btn-primary btn-block">Login</button>
                </form>
            </div>
        </div>
    </div>
</body>
</html>
```

Admin dashboard page:

```
<!-- templates/admin_dashboard.html -->
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Admin Dashboard</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css
/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h2>Admin Dashboard</h2>
        {% with messages = get_flashed_messages() %}
          {% if messages %}
            <div class="alert alert-info">{{ messages[0]
}}</div>
          {% endif %}
        {% endwith %}

        <a href="{{ url_for('logout') }}" class="btn
btn-danger float-right">Logout</a>

        <!-- Button to open Add Staff modal -->
        <button type="button" class="btn btn-primary mt-4"
data-toggle="modal" data-target="#addStaffModal">
            Add New Staff
        </button>


        <!-- Add Staff Modal -->
        <div class="modal fade" id="addStaffModal"
tabindex="-1" aria-labelledby="addStaffModalLabel"
aria-hidden="true">
            <div class="modal-dialog">
```

```
                <div class="modal-content">
                    <div class="modal-header">
                        <h5 class="modal-title"
id="addStaffModalLabel">Add New Staff</h5>
                        <button type="button" class="close"
data-dismiss="modal" aria-label="Close">
                            <span
aria-hidden="true">&times;</span>
                        </button>
                    </div>
                    <form action="{{ url_for('add_staff') }}"
method="post">
                        <div class="modal-body">
                            <div class="form-group">
                                <label for="staff_id">Staff
ID</label>
                                <input type="text"
class="form-control" id="staff_id" name="staff_id" required>
                            </div>
                            <div class="form-group">
                                <label
for="name">Name</label>
                                <input type="text"
class="form-control" id="name" name="name" required>
                            </div>
                            <div class="form-group">
                                <label
for="department">Department</label>
```

```
                                <input type="text"
class="form-control" id="department" name="department"
required>
                            </div>
                            <div class="form-group">
                                <label
for="email">Email</label>
                                <input type="email"
class="form-control" id="email" name="email" required>
                            </div>
                            <div class="form-group">
                                <label
for="phone_number">Phone Number</label>
                                <input type="text"
class="form-control" id="phone_number" name="phone_number"
required>
                            </div>
                            <div class="form-group">
                                <label
for="password">Password</label>
                                <input type="password"
class="form-control" id="password" name="password" required>
                            </div>
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn
btn-secondary" data-dismiss="modal">Close</button>
                            <button type="submit" class="btn
btn-primary">Add Staff</button>
```

```
                </div>
            </form>
        </div>
    </div>
</div>


<!-- Pending Requests Table -->
<h3 class="mt-4">Pending Requests</h3>
<table class="table table-bordered">
    <!-- Table headers and data as previously defined
-->
</table>


<!-- Allocation Details Table -->
<h3>Allocation Details</h3>
<table class="table table-bordered">
    <!-- Table headers and data as previously defined
-->
</table>
</div>


<!-- Bootstrap and jQuery JS -->
<script
src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></scr
ipt>
```

```
   <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.2/dist/js/bo
otstrap.bundle.min.js"></script>
</body>
</html>
```

Add staff:

```
<!-- templates/add_staff.html -->
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Add Staff</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css
/bootstrap.min.css">
</head>
<body>
   <div class="container mt-5">
       <h2>Add Staff</h2>
       {% with messages = get_flashed_messages() %}
         {% if messages %}
           <div class="alert alert-info">{{ messages[0]
}}</div>
         {% endif %}
       {% endwith %}
       <form method="POST" action="{{ url_for('add_staff')
}}">
           <div class="form-group">
```

```
                <label for="staff_id">Staff ID</label>
                <input type="text" class="form-control"
id="staff_id" name="staff_id" required>
            </div>
            <div class="form-group">
                <label for="name">Name</label>
                <input type="text" class="form-control"
id="name" name="name" required>
            </div>
            <div class="form-group">
                <label for="department">Department</label>
                <input type="text" class="form-control"
id="department" name="department" required>
            </div>
            <div class="form-group">
                <label for="email">Email</label>
                <input type="email" class="form-control"
id="email" name="email" required>
            </div>
            <div class="form-group">
                <label for="phone_number">Phone
Number</label>
                <input type="text" class="form-control"
id="phone_number" name="phone_number" required>
            </div>
            <button type="submit" class="btn btn-primary">Add
Staff</button>
        </form>
    </div>
```

```
</body>
</html>
```

Staff dashboard page:

```
<!-- templates/staff_dashboard.html -->
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Staff Dashboard</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css
/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h2>Staff Dashboard</h2>

        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <div class="alert alert-info">{{ messages[0]
}}</div>
            {% endif %}
        {% endwith %}


        <a href="{{ url_for('logout') }}" class="btn
btn-danger float-right">Logout</a>
```

```
<h3 class="mt-4">Your Requests</h3>
<table class="table table-bordered">
    <thead>
        <tr>
            <th>Request ID</th>
            <th>Request Date</th>
            <th>Priority</th>
            <th>Required Date</th>
            <th>Return Date</th>
            <th>Status</th>
        </tr>
    </thead>
    <tbody>
        {% for request in requests %}
            <tr>
                <td>{{ request.request_id }}</td>
                <td>{{ request.request_date }}</td>
                <td>{{ request.priority }}</td>
                <td>{{ request.required_date }}</td>
                <td>{{ request.return_date }}</td>
                <td>{{ request.request_status }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>


<h3>Resource Allocation Details</h3>
```

```
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Allocation ID</th>
                    <th>Resource ID</th>
                    <th>Resource Name</th>
                    <th>Allocation Date</th>
                    <th>Return Date</th>
                    <th>Status</th>
                </tr>
            </thead>
            <tbody>
                {% for allocation in allocations %}
                    <tr>
                        <td>{{ allocation.allocation_id
}}</td>
                        <td>{{ allocation.resource_id }}</td>
                        <td>{{ allocation.resource_name
}}</td>
                        <td>{{ allocation.allocation_date
}}</td>
                        <td>{{ allocation.return_date }}</td>
                        <td>{{ allocation.status }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
        <h3 class="mt-4">Submit New Resource Request</h3>
        <form action="{{ url_for('submit_request') }}"
method="POST">
            <div class="form-group">
                <label for="priority">Priority</label>
                <select class="form-control" id="priority"
name="priority" required>
                    <option value="high">High</option>
                    <option value="medium">Medium</option>
                    <option value="low">Low</option>
                </select>
            </div>


            <div class="form-group">
                <label for="required_date">Required
Date</label>
                <input type="date" class="form-control"
id="required_date" name="required_date" required>
            </div>


            <div class="form-group">
                <label for="return_date">Return Date</label>
                <input type="date" class="form-control"
id="return_date" name="return_date" required>
            </div>
            <div class="form-group">
                <label for="resource_id">Resource</label>
                <select class="form-control" id="resource_id"
name="resource_id" required>
                </select>
            </div>


            <div class="form-group">
                <label for="quantity">Quantity</label>
                <input type="number" class="form-control"
id="quantity" name="quantity" min="1" required>
            </div>


            <button type="submit" class="btn
btn-primary">Submit Request</button>
        </form>
   </div>
</body>
</html>
```

Submit request page:

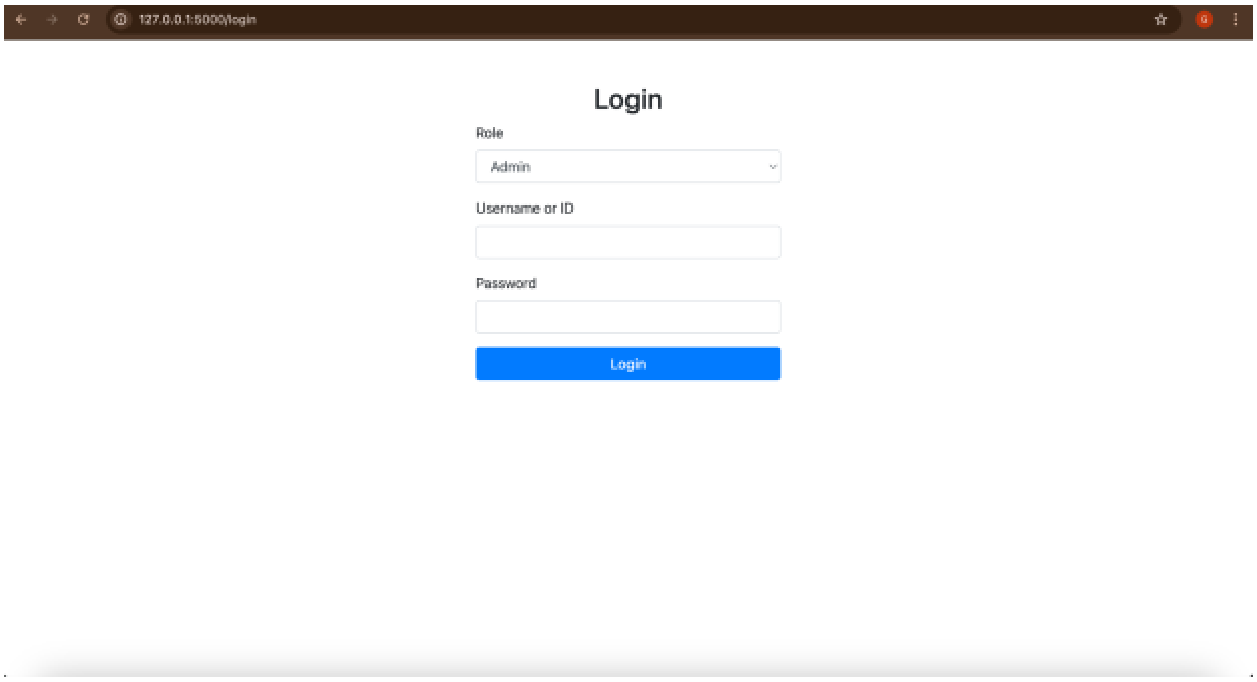
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Submit Resource Request</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css
/bootstrap.min.css">
</head>
<body>
   <div class="container mt-5">
       <h2>Submit Resource Request</h2>


       {% with messages = get_flashed_messages() %}
         {% if messages %}
           <div class="alert alert-info">{{ messages[0]
}}</div>
         {% endif %}
       {% endwith %}


       <form action="{{ url_for('submit_request') }}"
method="POST">
           <div class="form-group">
               <label for="priority">Priority</label>
               <select class="form-control" id="priority"
name="priority" required>
                   <option value="high">High</option>
                   <option value="medium">Medium</option>
                   <option value="low">Low</option>
               </select>
           </div>
```

```
            <div class="form-group">
                <label for="required_date">Required
Date</label>
                <input type="date" class="form-control"
id="required_date" name="required_date" required>
            </div>


            <div class="form-group">
                <label for="return_date">Return Date</label>
                <input type="date" class="form-control"
id="return_date" name="return_date" required>
            </div>


            <div class="form-group">
                <label for="resource_id">Resource</label>
                <select class="form-control" id="resource_id"
name="resource_id" required>
                </select>
            </div>


            <div class="form-group">
                <label for="quantity">Quantity</label>
                <input type="number" class="form-control"
id="quantity" name="quantity" min="1" required>
```

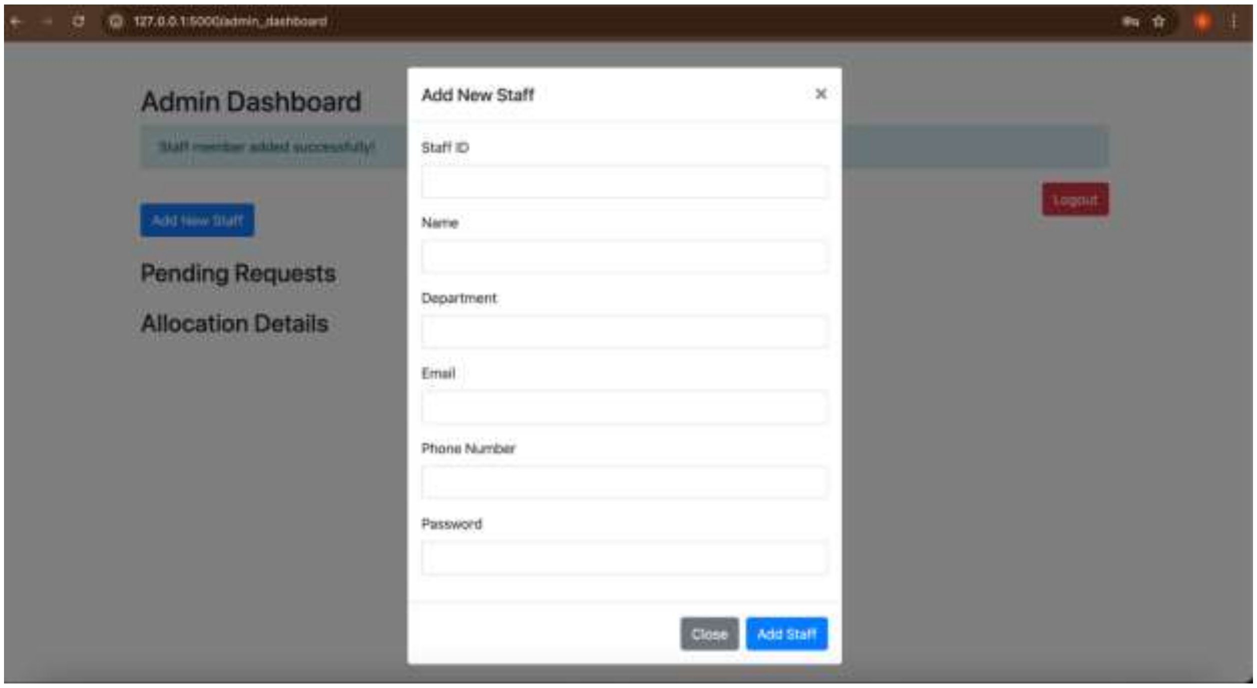
```
           </div>


           <button type="submit" class="btn
btn-primary">Submit Request</button>
           <a href="{{ url_for('staff_dashboard') }}"
class="btn btn-secondary ml-2">Back to Dashboard</a>
       </form>
   </div>
</body>
</html>
```

5. RESULTS AND DISCUSSIONS

RESULTS:

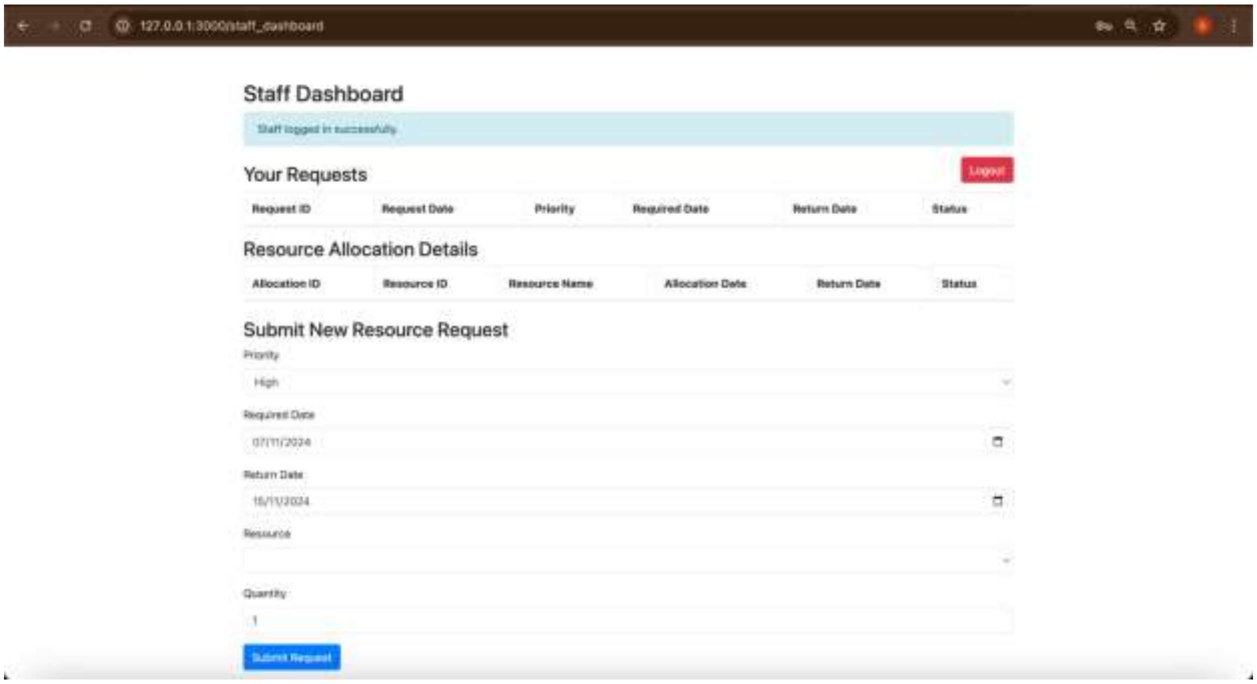


Login Page of the Software



Staff member added successfully!

Administrator Dashboard, with Add Staff Feature



127.0.0.1:3000/staff_dashboard

Staff Dashboard

Staff logged in successfully.

Your Requests

Logout

Request ID	Request Date	Priority	Required Date	Return Date	Status

Resource Allocation Details

Allocation ID	Resource ID	Resource Name	Allocation Date	Return Date	Status

Submit New Resource Request

Priority: High

Required Date: 07/11/2024

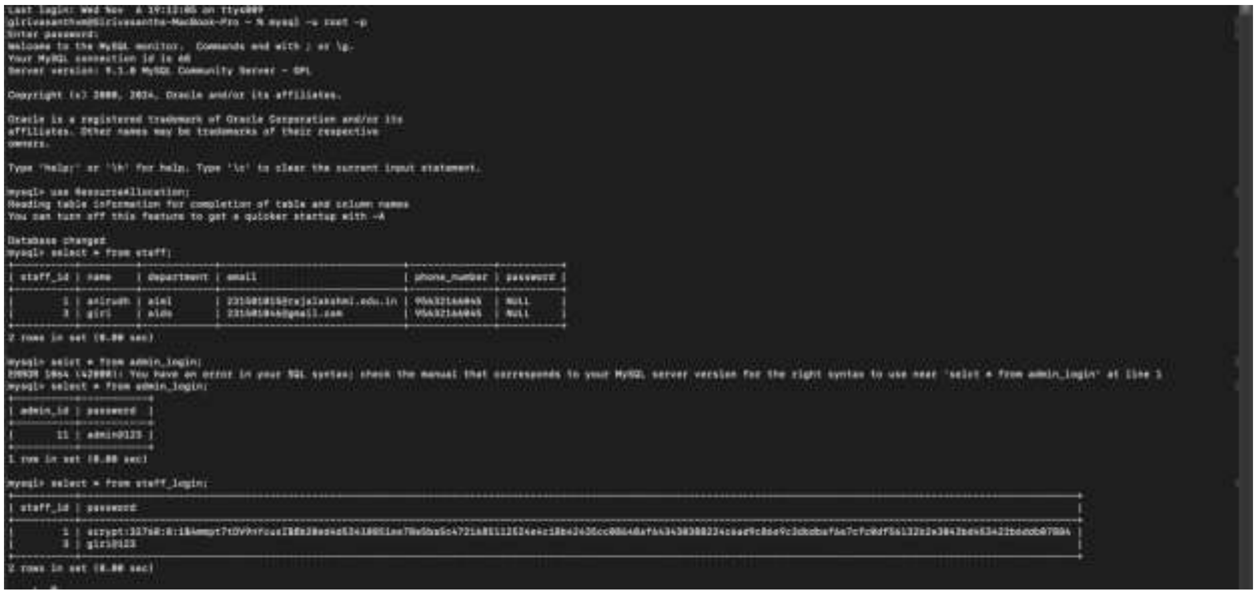
Return Date: 15/11/2024

Resource:

Quantity: 1

Submit Request

Staff Dashboard Page, with Item Request Feature



```
Last login: Wed Nov  6 19:12:05 on ttys009
girivasanthan@Girivasanths-MacBook-Pro ~ % mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 46
Server version: 9.1.0 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use ResourceAllocation;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from staff;
+----------+---------+------------+-----------------------------+--------------+----------+
| staff_id | name    | department | email                       | phone_number | password |
+----------+---------+------------+-----------------------------+--------------+----------+
|        1 | anirudh | aiml       | 231501015@rajalakshmi.edu.in | 9563214464 5 | NULL     |
|        3 | giri    | aids       | 231501046@gmail.com          | 95632144645  | NULL     |
+----------+---------+------------+-----------------------------+--------------+----------+
2 rows in set (0.00 sec)

mysql> selct * from admin_login;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'selct * from admin_login' at line 1
mysql> select * from admin_login;
+----------+----------+
| admin_id | password |
+----------+----------+
|       11 | admin@123 |
+----------+----------+
1 row in set (0.00 sec)

mysql> select * from staff_login;
+----------+----------------------------------------------------------------------------------+
| staff_id | password                                                                         |
+----------+----------------------------------------------------------------------------------+
|        1 | scrypt:32768:8:1$[illegible]                                                     |
|        3 | giri@123                                                                         |
+----------+----------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

SQL Database with Staff and Administrator Credentials

DISCUSSIONS:

Due to the very huge scope of this project and shortage of time and resources, we were only able to make a local, working prototype of the intended software, without its network-based features. We plan to work on this project further in the coming years, for we believe that this will streamline the administrative working of institutions.

6.CONCLUSION

To conclude, this project gave us an opportunity to implement the knowledge about Database Management Systems that we learned in this semester. This project also gave us a chance to learn about basic UI/UX Development.This project has been a brainchild of our collective effort and knowledge, and has helped us learn cooperation, team dynamics, and time management. We plan to complete this project in further years and see it implemented as a product that can help institutions reduce the administrative paper trail and hasten the working of educational institutions and see to it that its scope gets expanded and implement it in all sectors of society.

7.REFERENCES

Textbooks:

Database System Concepts(6th Edition) by Abraham Silberschatz, Henry F. Korth,S. Sudarshan

Python: The Complete Reference by Martin C.Brown

Websites:

www.geeksforgeeks.com

www.w3schools.com

Videos:

HTML and CSS Tips and Tricks:

https://www.youtube.com/watch?v=AActXSWxsRo&list=PL4-IK0AVhVjOEub8jm2W_XKhh9V0G2ot4&pp=iAQB

HTML And CSS for Beginners:

https://www.youtube.com/watch?v=LGQuIIv2RVA&list=PL4-IK0AVhVjM0xE0K2uZRvsM7LkIhsPT-&pp=iAQB

DBMS(Database Management Systems):

https://www.youtube.com/watch?v=T7AxM7Vqvaw&list=PLdo5W4Nhv31b33kF46f9aFjoJPOkdlsRc

Python Flask Basics:

https://www.youtube.com/watch?v=Kja_28SNIow&list=PLS1QulWo1RIZ6OujqIAXmLR3xsDn_ENHI

GitHub Page:

https://github.com/GirivasanthVimalan/DBMS-Mini-Project