STUDENT LEAVE MANAGEMENT SYSTEM

PROJECT REPORT

Submitted by

CH.EN.U4AIE22041	P Mokshajna Venkata Krushna
CH.EN.U4AIE22061	T Bhanu Koushik
CH.EN.U4AIE22014	G Muralidhar Reddy
CH.EN.U4AIE22068	V S V R Laxman Sai
CH.EN.U4AIE22040	P Manoj Kumar

is partial fulfilment of the requirements for the Course – 22AIE301 (Software Engineering)

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

CHENNAI-601103

MARCH 2025

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING, CHENNAI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that the report entitled "Leave Management System" submitted by

CH.EN.U4AIE22041	P Mokshajna Venkata Krushna
CH.EN.U4AIE22061	T Bhanu Koushik
CH.EN.U4AIE22014	G Muralidhar Reddy
CH.EN.U4AIE22068	V S V R Laxman Sai
CH.EN.U4AIE22040	P Manoj Kumar

is partial fulfilment of the requirements for the award of grade in course – **22AIE311 (Software Engineering)** is a Bonafide record of the work carried out at Amrita School of Engineering, Coimbatore.

Evaluated on:

Course Faculty Faculty Examiner

TABLE OF CONTENTS

1.PREVIEW OF THE PROJECT

ABSTRACT:

The Student Leave Management System (SLMS) is a software application created to assist pupils and teachers leave management in educational institutions. The system is implemented using Django with an backend powered by SQLite, HTML, CSS, and JS at the frontend. SLMS eliminated the old paper procedures with a more secure and convenient system. This system is give role-based access for students to apply for casual leave, on-duty leave, medical leave, and emergency leave while enforcing policies like monthly leave quotas with parental approvals and faculty-admin workflows. Additional function includes leave email notifications, leave request verification documents for medical leave, explainable, auditable, and OTP authentication leaves with the realtime tracking which is free of mistakes and compliant. The automation of approvals and centralized system of records brings down the tedious work SLMS performs. Having a centralized SLMS allows all institutions to have the same system and flow of information. It also allows for scaling for mobile, AI driven analytics, and support for multiple institutions.

1.1 Statement of Need:

Educational institutions often face challenges in traditional leave management which often proves to be inefficient, unclear, and has high chances of making errors. The Student Leave Management System (SLMS) replaces the traditional process by reducing paperwork, delay, and mismanagement. It enables tracking in real time with secure login, automated notifications, and policy compliance. By using OTP verification, document validation, and parental approval, SLMS enhances productivity, security, and ease of access to students, faculty, and administrations alike.

1.2 Statement of Motivation:

The absence management system relies heavily on manually managing leaves which can be carried out only through basic communication methods, as most of them make errors, delays, and cause obscurity, which leads to frustration for learners and staff. Moreover, digital leaves tracking systems heavily depend on electronic document submission and rely on fragmented communication methods which are inefficient intrinsically. Our Student Leave Management System (SLMS) is designed to eliminate these obstacles and automate surpassing the entire flow with proper tracking, secured notifications, and easy verification. The SLMS offers an automated leave authorization system that works together with OTP verification and statuses in

real-time which increases productivity, precision, availability, and makes absent management stress-free for both learners and administrators.

1.3 Instruments and Technology:

- **Frontend:** HTML, CSS, JavaScript – to create an engaging and easy to use user interface.
- **Backend Framework:** Django – for the seamless integration of server logic and database communication.
- **Database:** SQLite – To provide security for leave record management and storage.
- **Code Editor:** Visual Studio Code – to facilitate faster and clear coding.
- **Project Tracking:** JIRA - For task organization and project progress tracking.
- **Code Repository:** GitHub – for collaborative development and versioning.
- **Diagram Modelling:** StarUML – for system architecture and workflow modeling, design DFD, UML, and class diagrams.

Our Novelty:

- Students receive email notifications for leave application status (approved/rejected) and confirmation upon applying.
- Email OTP verification ensures portal security and integrity.
- OD types: Short-term OD (hackathons) & Long-term OD (internships).
- Students get 4 casual leaves/month; excess requests convert to an emergency pass, requiring document upload.
- A Technical Manager oversees portal issues and student-admin workflow.

2. SOFTWARE REQUIRMENTS SPECIFICATION

2.1 Functional And Non-Functional Requirements

Functional Requirements	Non-Functional Requirements
Students can apply for leave, and Admins (faculty) can approve/reject leave requests.	Ensures quick retrieval and storage of leave records using SQLite.
Students can apply for different leave types (Casual, On-Duty, Medical, Emergency).	Handles multiple users efficiently with optimized response times.
OTP-based login for students; 6-digit OTP sent after three failed login attempts.	User-friendly interface, accessible from any device with an internet connection.
Faculty can approve/reject leave requests, and students receive email notifications.	Ensures minimal downtime and automated email notifications for leave status.
Casual/On-Duty leave is limited to 4 per month; exceeding requires Emergency Leave with parental consent.	Validates incorrect leave applications and logs unauthorized access attempts.
Students can view their leave history, unique ID, and profile image.	Encrypted password storage with AES, RSA key encryption for AES keys.
Admins can add, delete, and search students.	The system ensures seamless navigation, with a maximum response time of 10 seconds for transitioning between pages, providing a smooth and efficient user experience.

2.2 Weekly Plan For Sprint 1:

Week	Plan Of Action
1	Setup development environment and necessary tools (Django, SQLite, VS Code, GitHub, JIRA)
2	Implement User Registration feature
3	Develop User Login with validation and session management
4	Implement Integrity and Security with email-based OTP verification

5	Design and implement Database Schema for users, leave requests, and login attempts

2.3 Weekly Plan For Sprint 2:

Week	Plan Of Action
1	Implement Check Leave Application Status feature with a dashboard for tracking leave status
2	Develop Upload Supporting Documents functionality with validation and final testing
3	Implement Apply for Leave with form validation and integration

2.4 Weekly Plan For Sprint 3:

Week	Plan Of Action
1	Implement Verify Emergency Leave Applications with email verification
2	Develop Review Leave Applications with a dashboard and filtering options
3	Implement View Student's Leave History with a searchable leave history table

2.5 Weekly Plan For Sprint 4:

Week	Plan Of Action
1	Implement Receive Email Notifications for Leave Status with email alerts
2	Develop Upload Supporting Documents with parent email verification
3	Implement Apply for Leave with low-balance email reminders

2.6 Boundaries and Constraints:

- Students need to authenticate via OTP ahead of system logins.
- A leave application cannot be made if the student has surpassed the leave allowance.
- Casual and on-duty leaves are capped at 4 a month—overspending this cap requires emergency leave which shall be cleared by the parents via email.
- Document uploads are mandatory for admin verification for Medical leave requests.
- All admin decisions made towards applications received shall be decided prior to finalization.
- Students with pending approvals on previously submitted requests are not able to apply for leave.
- The allowance for failing to log-in to the system is restricted to 3 attempts without success—surpassing this results in the need to issue OTP for verification purposes.
- There is no provision for the creation of a new account; hence, access to the system is only limited to logged in students with valid passwords.
- After submission, leave requests cannot be edited; if students have to make changes, they need to cancel the request and submit it again.

3. PRODUCT BACKLOG

3.1 Initial Product Backlog

ID	User Story / Feature	Priority (High/Med/Low)	Estimation (Story Points)	Acceptance Criteria	Status
LMS-01	As a student, I want to securely log in and access my dashboard	High	3	Login functionality with email & password, secured via OAuth/MFA	To Do
LMS-02	As a student, I want to submit leave applications with documents.	High	5	Form with leave type, date range, reason, and file upload working	To Do
LMS-03	As a class advisor, I want to view, approve, or reject leave requests	High	4	Leave requests displayed with approve/reject buttons and comment option	To Do
LMS-04	As a student, I want to receive email and SMS notifications	High	3	Automated alerts after submission, approval, or rejection	To Do
LMS-05	As a teacher/admin, I want to generate leave history reports	Medium	4	Reports available in PDF/CSV export with filtering	To Do

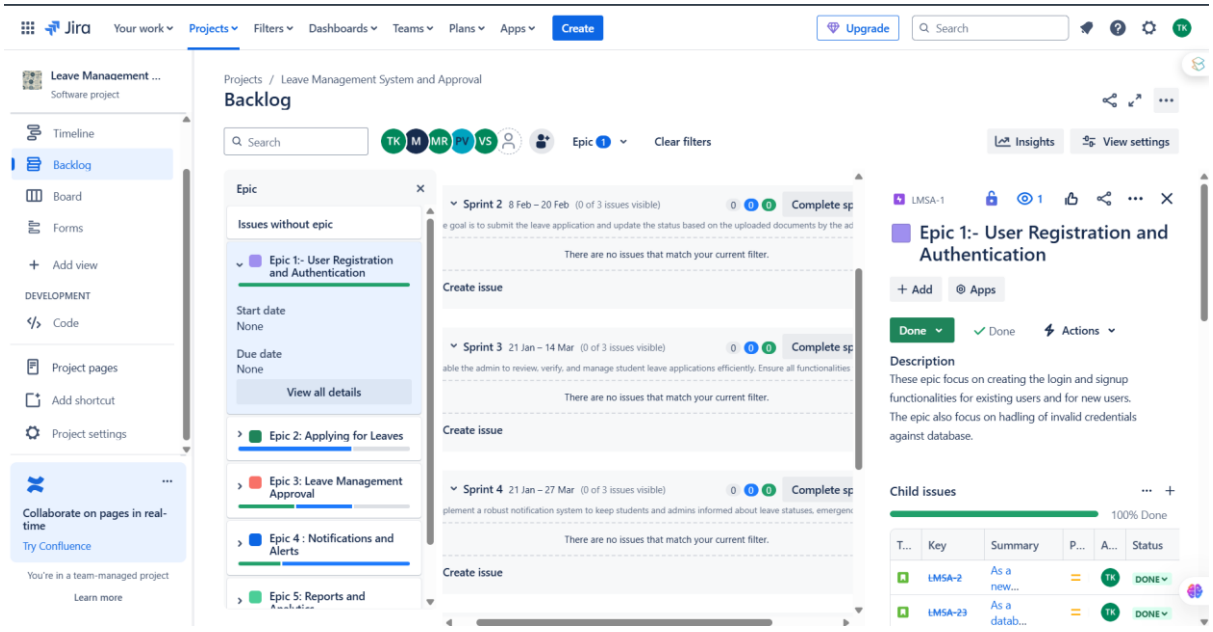
LMS-06	As an admin, I want to manage student accounts	Medium	5	Create, update, delete student records with role-based restrictions	To Do
LMS-07	System should automatically backup and secure data	Medium	4	Daily incremental and weekly full backup with alerts	In Progress
LMS-08	Role-based access management for students, admins, and technical staff	High	3	Each user role sees specific modules and actions	In Progress

3.2 Product Backlog Release

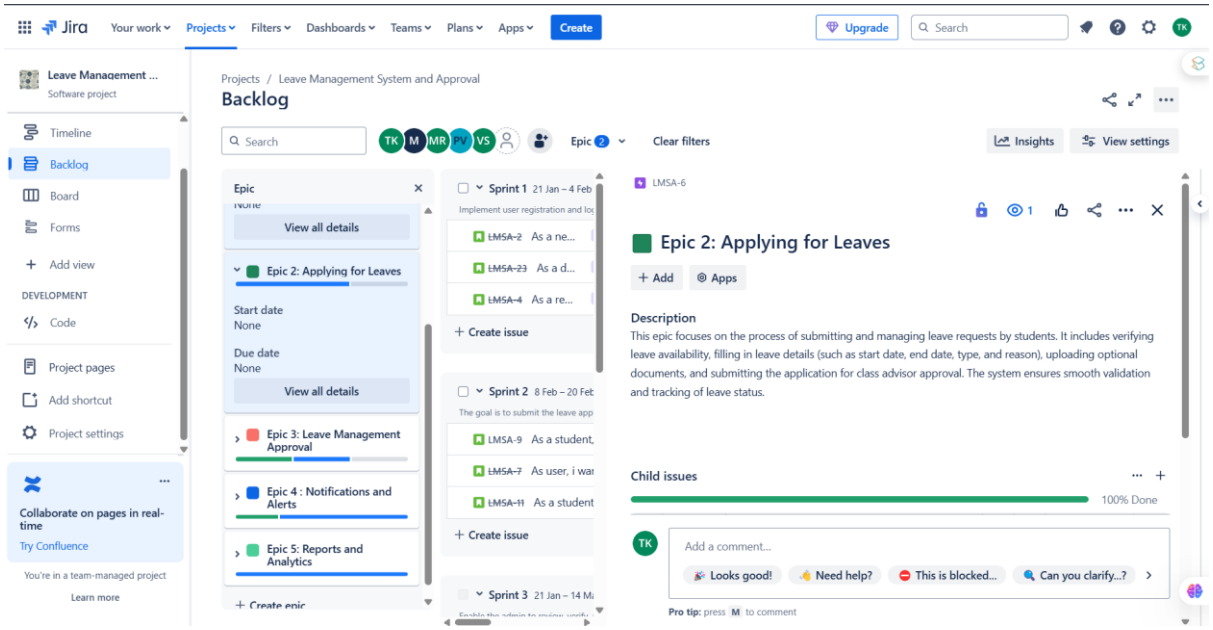
Sprint	Duration	Backlog Items Included	Status	Comments
Sprint 1	03-Feb to 25-Feb 2025	LMS-01, LMS-02 (Login and Leave Submission Modules)	Released	Initial version deployed to staging
Sprint 2	26-Feb to 10-Mar 2025	LMS-03, LMS-04, LMS-07 (Approval flow, Notifications, Backups)	In Progress	Integrating security alerts and backup monitoring
Sprint 3	11-Mar to 25-Mar 2025 (Ongoing)	LMS-05, LMS-06, LMS-08 (Reports, Admin panel, Role-based control)	Planned	Performance optimization & scalability planned

3.3 Epic Wise Report

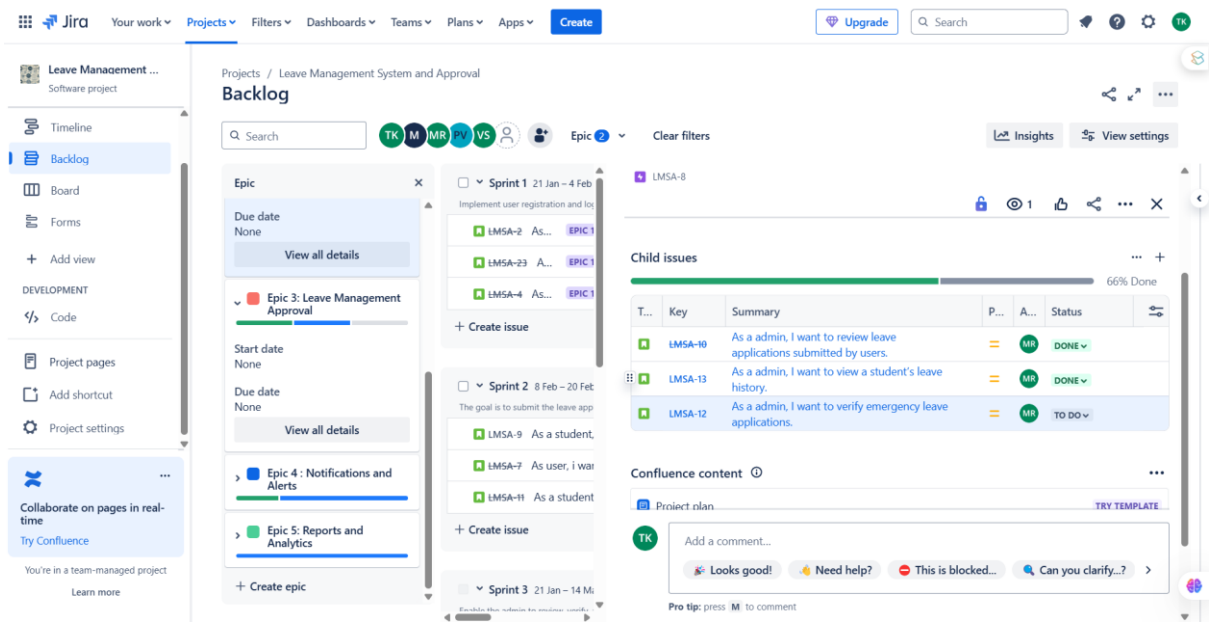
3.2.1 Epic 1: User Registration and Authentication



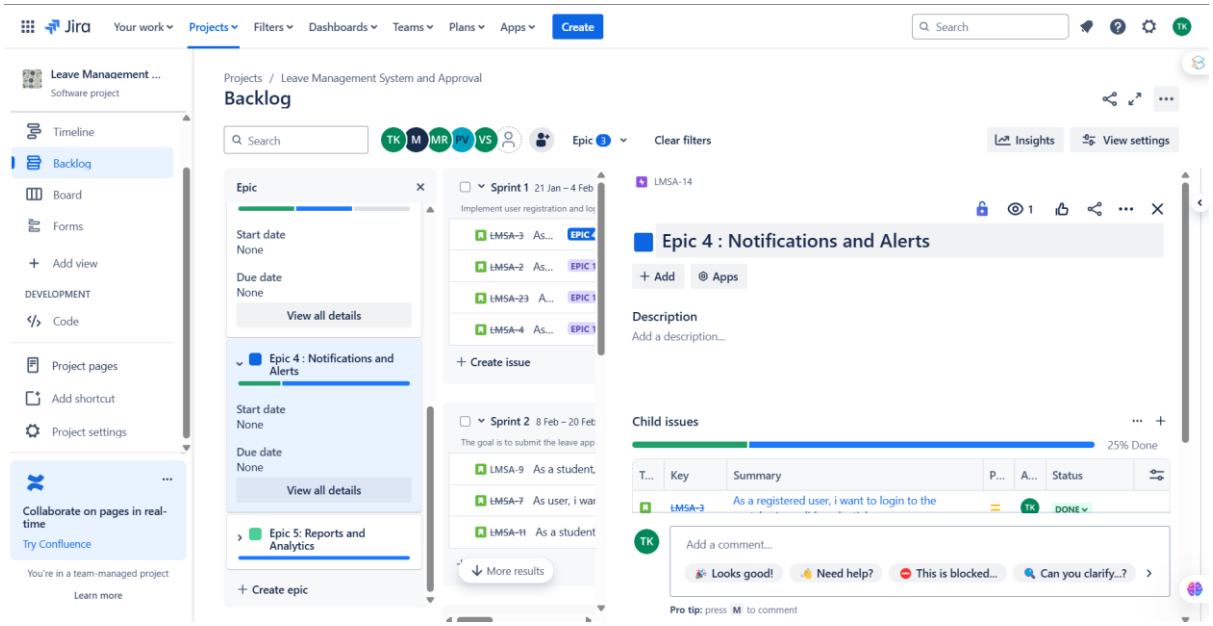
3.2.2 Epic 2: Applying for Leaves



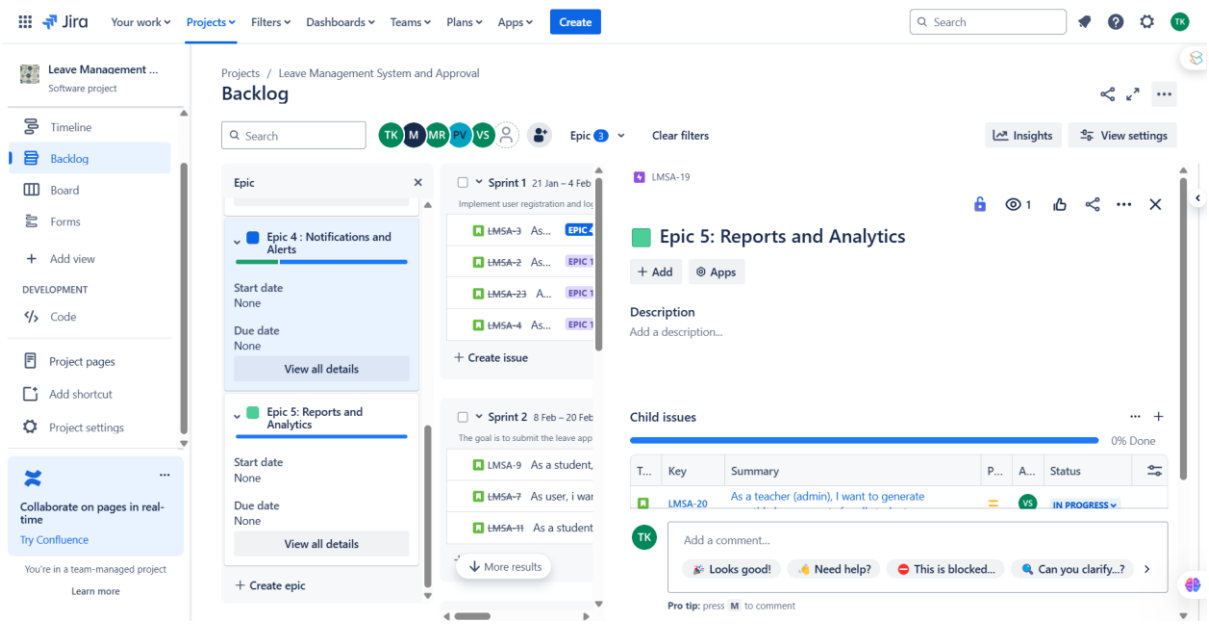
3.3.3 Epic 3: Leave Management Approval



3.3.4 Epic 4: Notifications And Alerts

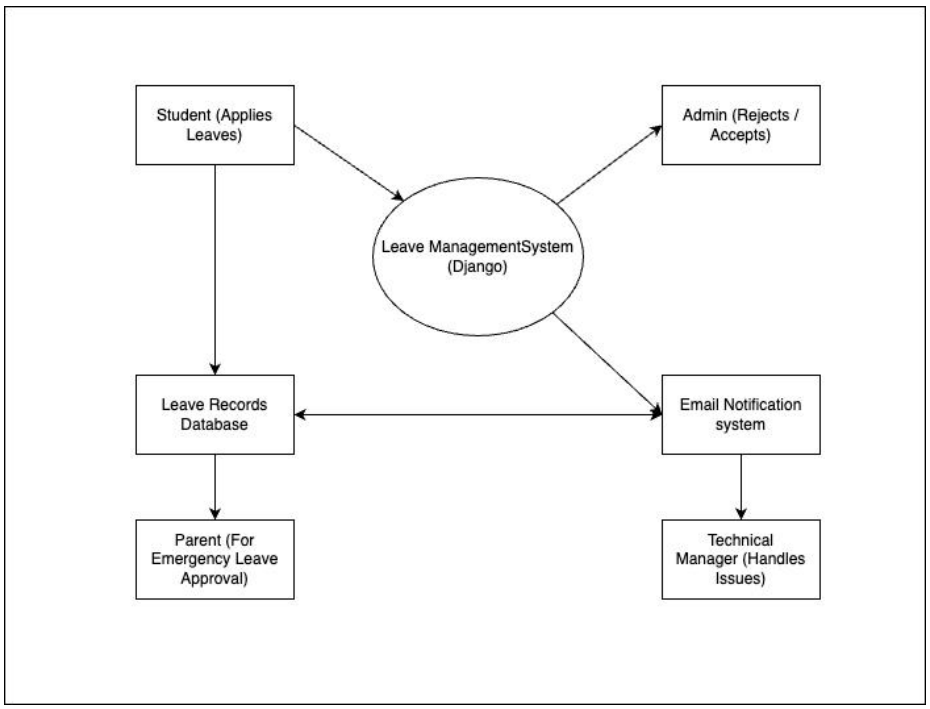


3.3.5 Epic 5: Reports and Analytics

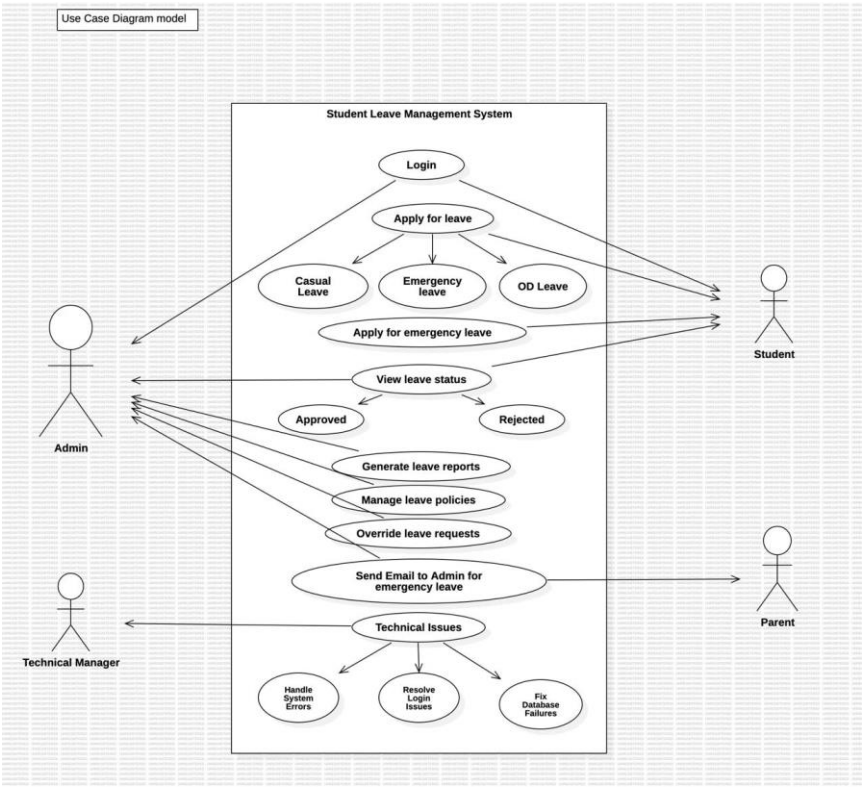


4. DIAGRAMS

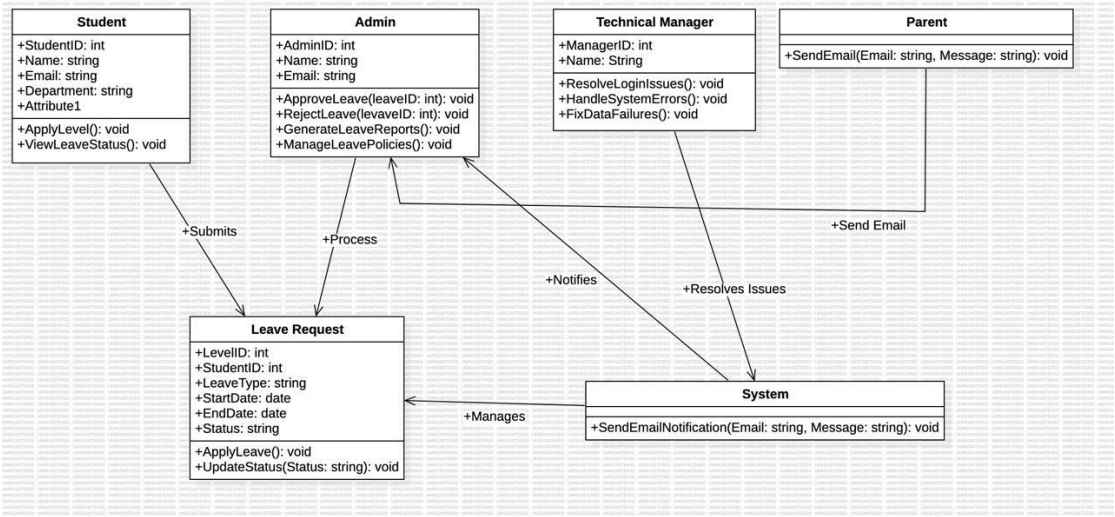
4.1 Data Flow Diagram:



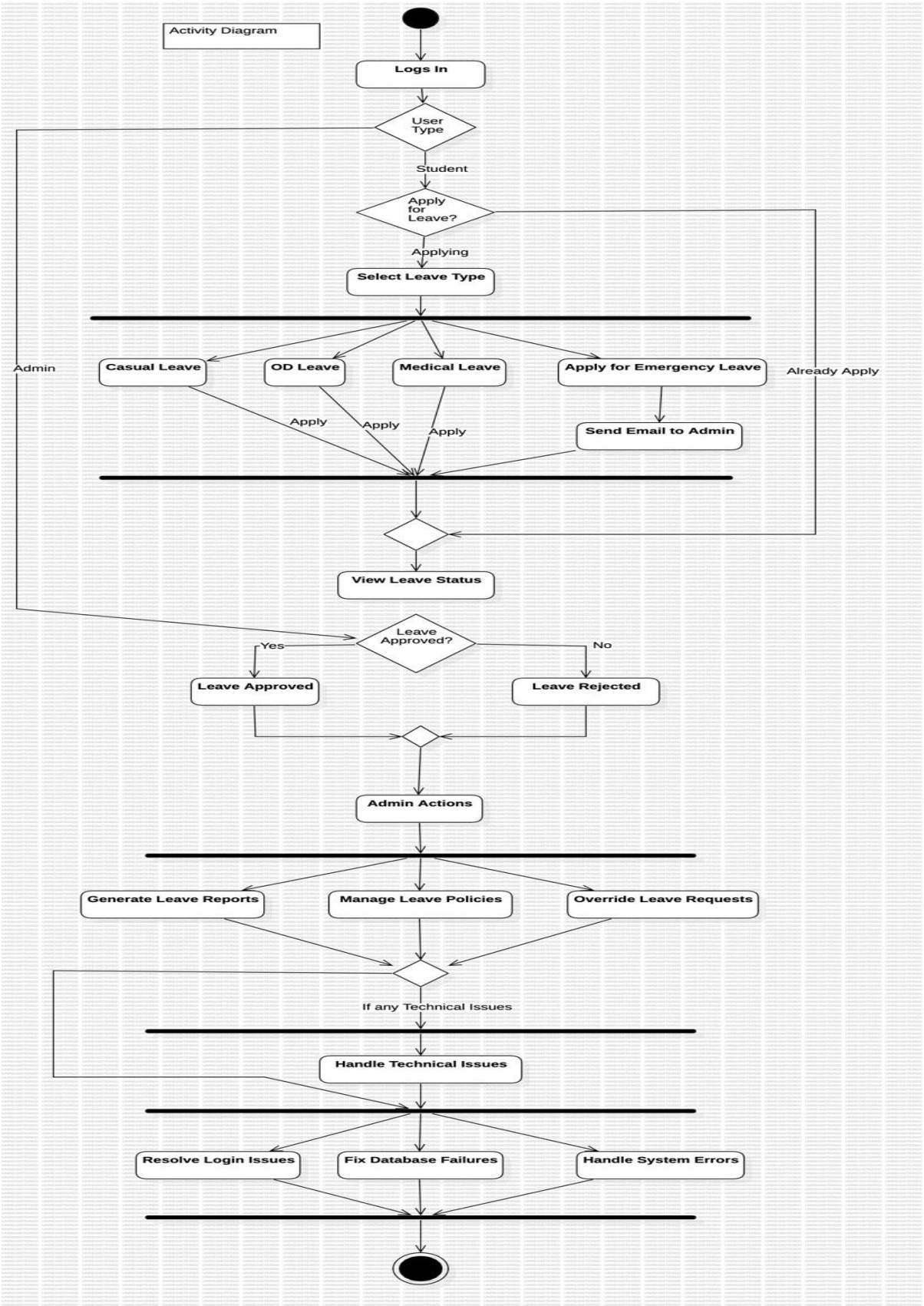
4.2 Use Case Diagram:



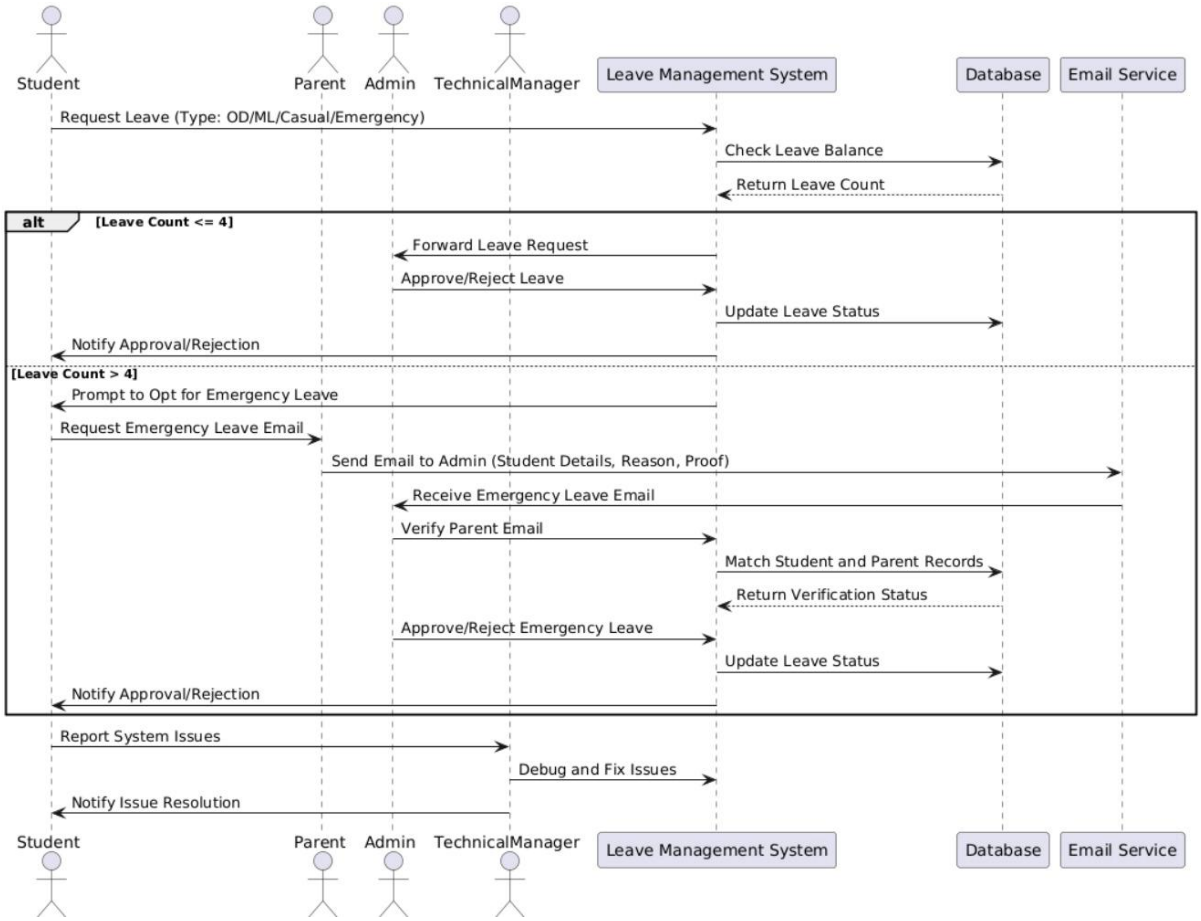
4.3 Class Diagram:



4.4 Activity Diagram:



4.5 Sequence Diagram:



5. SCRUM ACTIVITIES

5.1 Sprint 1 stand-up notes:

As of now, we have a planned meeting on developing the Leave Management System for 25/01/2021. In this first meeting, we delineated the main functionalities to be accomplished for Sprint-2 and paired up with the frontend and backend teams. In parallel, the backend team was busy creating the required collections in the cloud MongoDB and Firebase so that smooth data handling and authentication with OTP based login verification could be done. Each team was able to adjust its ways of working to complete the tasks for the deadline set. After a certain level of development work had been done on the software, we proceeded with the testing phase, which involved using the Jest and Enzyme testing frameworks to check that the system functions as expected. In the end, we met all deadlines for the solution of the Leave Management System and constructed an efficient tool that allows easy management of student leaves.

5.2 Scrum Standup Meeting Schedule:

DATE	TIME	MODE	DESCRIPTION
25/01/2025	5pm-7pm	Teams	Discussed about the flow and division of components
02/02/2025	4pm-6pm	Teams	Finalised on the priority of each story and proceeded to development phase
06/02/2025	4pm-6pm	Teams	Discussions about design, implementation and database design of login page
17/02/2025	2pm-5pm	Teams	Final discussion about login page and initiating the creation of home page
23/02/2025	10pm-11pm	Teams	Discussion about home page focused around the different API retrieval calls
07/03/2021	6pm-7:30pm	Teams	Final discussion about homepage and initiating the implementation of management end of the webapp

24/03/2021	3pm-5pm	Teams	Discussed about the implementation of lecture registration feature for users and lecture manipulation feature for manager
06/04/2021	7pm-9pm	Teams	Discussion about final presentation and sprint 1

5.3 Sprint 2 stand-up notes

5.3.1 Week 1

Date: 28/01/2025 - 6.30 PM

Summary: Commencing with Sprint 2

Description: We outlined the features of the Leave Management System that are outstanding, and we allocated responsibilities to each team member. We also gave an overview of how the development and testing phases of Sprint 2 would unfold. Moreover, a decision was made to commence the Admin Dashboard design for easier leave approval, tracking, and analytics.

Name	Question	Answer	Sign
Krushna	What did you do yesterday?	Completed documentation work for Sprint 1 and explored UI components for the Admin Dashboard.	Krushna
	What will you do today?	Start designing the Admin Dashboard UI and structure the leave request form.	Krushna
	Are there any impediments in your way?	No	Krushna

Bhanu	What did you do yesterday?	Completed documentation work for Sprint 1 and made UI modifications for the student leave application page.	Bhanu
	What will you do today?	Start designing the Admin Dashboard UI for leave tracking.	Bhanu
	Are there any impediments in your way?	No	Bhanu
Murali	What did you do yesterday?	Completed documentation work for Sprint 1.	Murali
	What will you do today?	Work on designing the Technical Management Page UI.	Murali
	Are there any impediments in your way?	No	Murali
Manoj	What did you do yesterday?	Completed documentation work for Sprint 1.	Manoj
	What will you do today?	Work on designing the Holiday Calendar Page UI.	Manoj
	Are there any impediments in your way?	Yes	Manoj
Laxman	What did you do yesterday?	Fixed bugs in API calls and helped with documentation.	Laxman

	What will you do today?	Work on forgot password UI and backend OTP verification flow.	Laxman
	Are there any impediments in your way?	No	Laxman
Krushna	What did you do yesterday?	Assisted in documentation.	Krushna
	What will you do today?	Structure database collections for leave requests and approvals.	Krushna
	Are there any impediments in your way?	No	Krushna

Time: 18/02/2025 - 8:00pm

In short: Discussed and designed the Technical page and assigned individual tasks for the week.

In detail: Listed out the key details that should be present on the Technical page, and referred to multiple sources, and designed the technical page in Figma. Then the page was split into different components and each component was assigned to each individual.

Name	**Question**	**Answer**	**Sign**
Murali	What did you do yesterday?	Designed the UI for the Leave Application Page	Murali
	What will you do today?	Edit the UI with feedback from the team and integrate OTP-based login verification	Murali

	Are there any impediments in your way?	No, discussed with the team and refined the UI	Murali
Krushna	What did you do yesterday?	Worked on the Leave Approval UI for Admin Dashboard	Krushna
	What will you do today?	Implement the UI for Admin Dashboard in React, including leave status updates	Krushna
	Are there any impediments in your way?	The animation for leave status updates needs optimization	Krushna
Bhanu	What did you do yesterday?	Designed the UI for the Leave Request Form	Bhanu
	What will you do today?	Start implementing the Leave Request UI with document upload feature	Bhanu

	Are there any impediments in your way?	No	Bhanu
Laxman	What did you do yesterday?	Designed the Holiday Calendar UI for the student dashboard	Laxman
	What will you do today?	Start implementing the Holiday Calendar and leave tracking features	Laxman

	Are there any impediments in your way?	No	Laxman
Manoj	What did you do yesterday?	Designed the UI for OTP verification page	Manoj
	What will you do today?	Implement OTP authentication and integrate it with email verification	Manoj
	Are there any impediments in your way?	Email notification setup requires additional configuration	Manoj
Laxman	What did you do yesterday?	Planned the database structure for leave applications in MongoDB	Laxman
	What will you do today?	Create collections for leave requests and approvals in MongoDB	Laxman
	Are there any impediments in your way?	No	Laxman

5.3.2 Week 2

Date:27/02/2025 - 6pm

In short: I followed up on my team's progress toward the UI implementation of their respective components and the creation of the database.

Rephrased: A developer started this meeting by reporting on if he managed to implement the UI for his part of the system and provided the necessary completions for the database. The room was divided in two for better explaining the UIs - the leave tracking and the leave approval. In order to enable better management of the students the database was enhanced, which allows for more efficient tracking and approving of leaves. A decision was made to give extra time before the deadline on the UI development to make sure it was correct.

Name	**Question**	**Answer**	**Sign**
Bhanu	What did you do yesterday?	Finished the Leave Request Form UI and student profile integration	Bhanu
	What will you do today?	Work on the UI design for the Leave History page in Figma	Bhanu
	Are there any impediments in your way?	Designing a non-repetitive pattern for displaying leave history records	Bhanu
Laxman	What did you do yesterday?	Refactored all routes for the Leave Management System and created a new route for the Admin Dashboard	Laxman
	What will you do today?	Complete my assigned UI part and integrate all UI components together in the system	Laxman
	Are there any impediments in your way?	No	Laxman

Manoj	What did you do yesterday?	Learned basic UI implementation in React and started working on OTP verification UI	Manoj

	What will you do today?	Implement the OTP-based login UI and integrate it with email verification	Manoj
	Are there any impediments in your way?	No	Manoj
Murali	What did you do yesterday?	Learned how to perform API calls for backend integration.	Murali
	What will you do today?	Research more on API calls and implement backend integration for leave requests.	Murali
	Are there any impediments in your way?	Yes, handling authentication for OTP verification requires additional study	Murali
Krushna	What did you do yesterday?	Restructured MongoDB collections to facilitate leave applications and approvals	Krushna
	What will you do today?	Create a new collection for storing student leave records and approvals.	Krushna

	Are there any impediments in your way?	Managing leave status updates efficiently in MongoDB requires further study	Krushna
Bhanu	What did you do yesterday?	Learned how to create models for a collection in MongoDB	Bhanu
	What will you do today?	Start backend modelling for storing leave application data.	Bhanu
	Are there any impediments in your way?	No	Bhanu

Date: 05/03/2025 - 6 PM

Summary: Finalized individual UI components, discussed Admin Dashboard UI.

Detailed: All team members completed their individual UI components and integrated them into the Leave Management System. The new Admin Dashboard was discussed and its UI was worked on in Adobe XD, focusing on a logical layout for leaves and approvals to aid in remaining organized. The structure was then broken down into different parts and given to other team members.

Upon completion, the structure of the database was designed and set up to store leave applications, approvals, and the student record. Also, the supporting API calls for application of leaves, OTP verification, and email derangements were talked over and set to be constructed.

Name	Question	Answer	Sign
Manoj	What did you do yesterday?	Designed the UI for the Leave History page using Figma	Manoj
	What will you do today?	Code the UI component for the Leave History page and integrate it with the dashboard	Manoj
	Are there any impediments in your way?	Displaying leave records along with status updates requires optimization	Manoj
Krushna	What did you do yesterday?	Completed my assigned UI for the Admin Dashboard and started integration	Krushna
	What will you do today?	Integrate all UI components into the Admin Dashboard and start implementing the Leave Analytics page	Krushna
	Are there any impediments in your way?	Implementing graphical representations for leave statistics needs further exploration.	Krushna
Laxman	What did you do yesterday?	Implemented the given React component and explored responsive design changes	Laxman
	What will you do today?	Explore and implement the assigned component for the Holiday Calendar UI	Laxman

	Are there any impediments in your way?	No	Laxman
Bhanu	What did you do yesterday?	Designed the UI for the Leave Request Form	Bhanu
	What will you do today?	Implement the UI for the Leave Request Form and integrate document upload functionality	Bhanu
	Are there any impediments in your way?	No	Bhanu
Murali	What did you do yesterday?	Studied the method of storing leave request data in MongoDB	Murali
	What will you do today?	Implement the UI for the Forgot Password page with OTP verification	Murali
	Are there any impediments in your way?	No doubts as of now	Murali
	What did you do yesterday?	Explored backend API calls with Express.js for leave request handling	Murali
	What will you do today?	Implement the assigned API calls for leave requests and approvals	Murali

	Are there any impediments in your way?	No	Murali

5.3.3 Week 3

Date:17/03/2025-8:00pm
Action items: Created and discussed a design for the Technical Management Page and its tasks for the week.

In-depth: The students discussed the relevant information that should appear in a Technical Management Page. This page should enable system monitoring and ensure that smooth functioning is guaranteed to the maximum possible extent. While designing the page in Figma, various references were consulted. Once the design was approved, the page parts were integrated into modules for purposes of component based development. Each module is given to a different developer for implementation.

Name	**Question**	**Answer**	**Sign**
Krushna	What did you do yesterday?	Finished a part of the Leave Application UI Component	Krushna
	What will you do today?	Work on the UI for the Leave Approval page	Krushna
	Are there any impediments in your way?	No	Krushna
Murali	What did you do yesterday?	Finished a part of the Leave Request UI Component and API calls for it	Murali
	What will you do today?	Work on the UI for the Leave Management Dashboard	Murali

	Are there any impediments in your way?	No	Murali
Laxman	What did you do yesterday?	Created the required API calls for the Leave Request system	Laxman
	What will you do today?	Implement the assigned UI component for the Admin Dashboard	Laxman
	Are there any impediments in your way?	No	Laxman
Bhanu	What did you do yesterday?	Finished a part of the Leave Request UI Component	Bhanu
	What will you do today?	Work on the UI for the Leave History page	Bhanu
	Are there any impediments in your way?	No	Bhanu
Manoj	What did you do yesterday?	Created the required API calls for the Leave Application page and started unit testing	Manoj
	What will you do today?	Implement the assigned UI component for OTP-based authentication.	Manoj
	Are there any impediments in your way?	No	Manoj

5.3.4 Week 4

Date: 22/03/2025 - 3 PM

Summary: Checked remaining tasks, started using the DevOps tools.

Detailed: We monitored Jira for pending work for Leave Management Systems and divided tasks among the programmers. We also started to look into Jenkins and Docker for possible improvement on deployment and automation. The team set what they considered to be minimum adequate documentation for the integration and deployment of the system in the established CI/CD pipelines.

Name	Question	Answer	Sign
Laxman	What did you do yesterday?	Finished the webpage development part for the Leave Management System	Laxman
	What will you do today?	Work on testing and bug fixes	Laxman
	Are there any impediments in your way?	No	Laxman
Manoj	What did you do yesterday?	Finished the webpage development part for leave request handling	Manoj
	What will you do today?	Work on Docker containerization for deployment	Manoj
	Are there any impediments in your way?	No	Manoj
Bhanu	What did you do yesterday?	Completed testing for last week's UI and backend integration	Bhanu

	What will you do today?	Continue testing and validation of leave approval flow	Bhanu
	Are there any impediments in your way?	No	Bhanu
Murali	What did you do yesterday?	Finished the development of the admin panel UI	Murali
	What will you do today?	Work on testing and integrate all individual components	Murali
	Are there any impediments in your way?	No	Murali
Krushna	What did you do yesterday?	Finished the webpage development for leave reports and analytics	Krushna
	What will you do today?	Work on Jenkins pipeline setup for CI/CD.	Krushna
	Are there any impediments in your way?	No	Krushna
	What did you do yesterday?	Completed the final UI integration for leave tracking	Krushna
	What will you do today?	Work on Azure deployment and Jenkins automation	Krushna
	Are there any impediments in your way?	No	Krushna

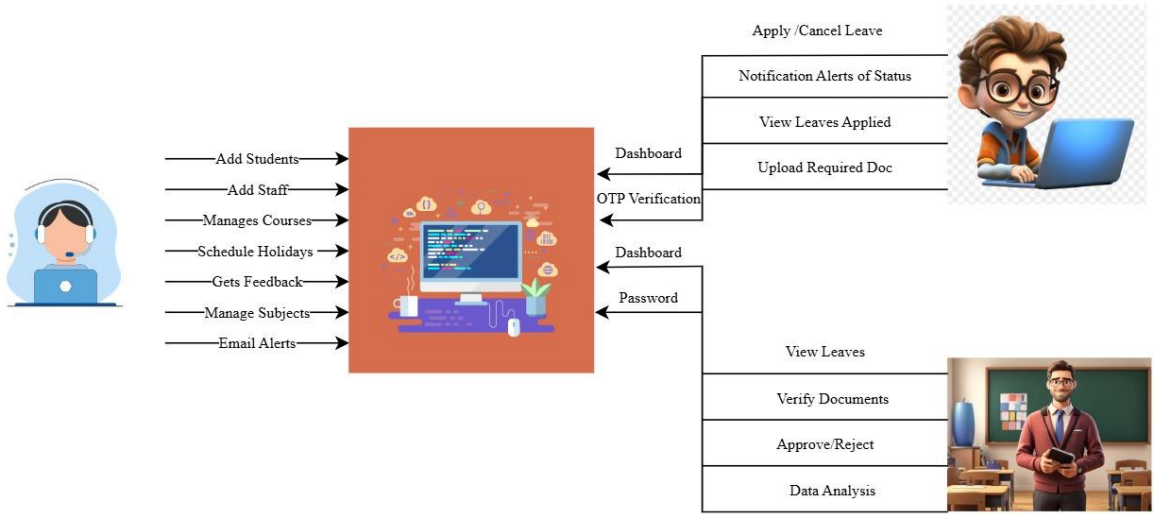
6. IMPLEMENTATION

The system can keep a record of leaves for educational organizations while simplifying the processes for application, monitoring, and approval. The system was built using HTML, CSS and JavaScript for frontend design. Backend logic and access control were done through Django (Python) and the information is kept in a MySQL database. Users authenticate using OTP , while data at rest security is protected by AES, data in transit security is guarded by RSA, and strong authentication is done using OTP.

Each student can access the system and log in using an OTP, check their leave balance, and apply for Emergency, On-Duty, Medical, or Casual leaves. To apply for Emergency Leave, a student must submit certain documents and get parental email verification. An admin gets to approve or deny documents submitted to him/her, validate them and then via the system inform what changes need to be made. Students then receive automated emails determined by the system through an SMTP service to inform them of their leave status.

A technical manager monitors the deadlines for the system ToR and also monitors who logs in, sets passwords, verifies database security, and troubleshoots any problems that may come up. Development work is supervised by means of UML diagrams in the form of sequence, use case, class and DFD. For more effective planning, new helpful functionalities such as holiday calendars, automatic changes of leave restrictions, or changeable dashboards have been implemented.

Architecture Diagram:



7. SAMPLE CODE

Admin.py

```python
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import CustomUser, AdminHOD, Staffs, Courses, Subjects, Students, Attendance, AttendanceReport, LeaveReportStudent, LeaveReportStaff, FeedBackStudent, FeedBackStaffs, Notificatio

# Register your models here.
class UserModel(UserAdmin):
    pass


admin.site.register(CustomUser, UserModel)

admin.site.register(AdminHOD)
admin.site.register(Staffs)
admin.site.register(Courses)
admin.site.register(Subjects)
admin.site.register(Students)
admin.site.register(Attendance)
admin.site.register(AttendanceReport)
admin.site.register(LeaveReportStudent)
admin.site.register(LeaveReportStaff)
admin.site.register(FeedBackStudent)
admin.site.register(FeedBackStaffs)
admin.site.register(NotificationStudent)
admin.site.register(NotificationStaffs)
```

Forms.py

```python
from django import forms
from django.forms import Form
from student_management_app.models import Courses, SessionYearModel


class DateInput(forms.DateInput):
    input_type = "date"


class AddStudentForm(forms.Form):
    email = forms.EmailField(label="Email", max_length=50, widget=forms.EmailInput(attrs={"class":"form-control"}))
    password = forms.CharField(label="Password", max_length=50, widget=forms.PasswordInput(attrs={"class":"form-control"}))
    first_name = forms.CharField(label="First Name", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    last_name = forms.CharField(label="Last Name", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    username = forms.CharField(label="Username", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    address = forms.CharField(label="Address", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    #For Displaying Courses
    try:
        courses = Courses.objects.all()
        course_list = []
        for course in courses:
            single_course = (course.id, course.course_name)
            course_list.append(single_course)
    except:
        course_list = []

    #For Displaying Session Years
    try:
        session_years = SessionYearModel.objects.all()
        session_year_list = []
        for session_year in session_years:
            single_session_year = (session_year.id, str(session_year.session_start_year)+" to "+str(session_year.session_end_year))
            session_year_list.append(single_session_year)

    except:
        session_year_list = []


    gender_list = (
        ('Male','Male'),
        ('Female','Female')
    )
```

```
    course_id = forms.ChoiceField(label="Course", choices=course_list, widget=forms.Select(attrs={"class":"form-control"}))
    gender = forms.ChoiceField(label="Gender", choices=gender_list, widget=forms.Select(attrs={"class":"form-control"}))
    # session_start_year = forms.DateField(label="Session Start", widget=DateInput(attrs={"class":"form-control"}))
    # session_end_year = forms.DateField(label="Session End", widget=DateInput(attrs={"class":"form-control"}))
    profile_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))



class EditStudentForm(forms.Form):
    email = forms.EmailField(label="Email", max_length=50, widget=forms.EmailInput(attrs={"class":"form-control"}))
    first_name = forms.CharField(label="First Name", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    last_name = forms.CharField(label="Last Name", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    username = forms.CharField(label="Username", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))
    address = forms.CharField(label="Address", max_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

    #For Displaying Courses
    try:
        courses = Courses.objects.all()
        course_list = []
        for course in courses:
            single_course = (course.id, course.course_name)
            course_list.append(single_course)
    except:
        course_list = []

    #For Displaying Session Years
    try:
        session_years = SessionYearModel.objects.all()
        session_year_list = []
        for session_year in session_years:
            single_session_year = (session_year.id, str(session_year.session_start_year)+" to "+str(session_year.session_end_year))
            session_year_list.append(single_session_year)

    except:
        session_year_list = []
```

```
gender_list = (
    ('Male','Male'),
    ('Female','Female')
)

course_id = forms.ChoiceField(label="Course", choices=course_list, widget=forms.Select(attrs={"class":"form-control"}))
gender = forms.ChoiceField(label="Gender", choices=gender_list, widget=forms.Select(attrs={"class":"form-control"}))
session_year_id = forms.ChoiceField(label="Session Year", choices=session_year_list, widget=forms.Select(attrs={"class":"form-control"}))
# session_start_year = forms.DateField(label="Session Start", widget=DateInput(attrs={"class":"form-control"}))
# session_end_year = forms.DateField(label="Session End", widget=DateInput(attrs={"class":"form-control"}))
profile_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))
```

Login_check_middle_ware.py

```
from django.utils.deprecation import MiddlewareMixin
from django.shortcuts import render, redirect
from django.urls import reverse


class LoginCheckMiddleWare(MiddlewareMixin):

    def process_view(self, request, view_func, view_args, view_kwargs):
        modulename = view_func.__module__
        # print(modulename)
        user = request.user

        #Check whether the user is logged in or not
        if user.is_authenticated:
            if user.user_type == "1":
                if modulename == "student_management_app.HodViews":
                    pass
                elif modulename == "student_management_app.views" or modulename == "django.views.static":
                    pass
                else:
                    return redirect("admin_home")

            elif user.user_type == "2":
                if modulename == "student_management_app.StaffViews":
                    pass
                elif modulename == "student_management_app.views" or modulename == "django.views.static":
                    pass
                else:
                    return redirect("staff_home")

            elif user.user_type == "3":
                if modulename == "student_management_app.StudentViews":
                    pass
                elif modulename == "student_management_app.views" or modulename == "django.views.static":
                    pass
                else:
                    return redirect("student_home")

            else:
                return redirect("login")

        else:
            # Allow access to login, OTP verification, and other authentication pages without login
            if request.path == reverse("login") or \
               request.path == reverse("doLogin") or \
               request.path == reverse("verify_otp") or \
               request.path == reverse("resend_otp"):
                pass
```

```
        else:
            # Allow access to login, OTP verification, and other authentication pages without login
            if request.path == reverse("login") or \
               request.path == reverse("doLogin") or \
               request.path == reverse("verify_otp") or \
               request.path == reverse("resend_otp"):
                pass
            else:
                return redirect("login")
```

Model.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver



class SessionYearModel(models.Model):
    id = models.AutoField(primary_key=True)
    session_start_year = models.DateField()
    session_end_year = models.DateField()
    objects = models.Manager()



# Overriding the Default Django Auth User and adding One More Field (user_type)
class CustomUser(AbstractUser):
    user_type_data = ((1, "HOD"), (2, "Staff"), (3, "Student"))
    user_type = models.CharField(default=1, choices=user_type_data, max_length=10)



class AdminHOD(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()


class Courses(models.Model):
    id = models.AutoField(primary_key=True)
    course_name = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

    # def __str__(self):
    #     return self.course_name
```

```python
class Staffs(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    address = models.TextField()
    department = models.ForeignKey('Courses', on_delete=models.DO_NOTHING, null=True, blank=True, related_name="staff_department")
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()



class Subjects(models.Model):
    id =models.AutoField(primary_key=True)
    subject_name = models.CharField(max_length=255)
    course_id = models.ForeignKey(Courses, on_delete=models.CASCADE, default=1) #need to give defauult course
    staff_id = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()



class Students(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    gender = models.CharField(max_length=50)
    profile_pic = models.FileField()
    address = models.TextField()
    course_id = models.ForeignKey(Courses, on_delete=models.DO_NOTHING, default=1)
    session_year_id = models.ForeignKey(SessionYearModel, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()


class Attendance(models.Model):
    # Subject Attendance
    id = models.AutoField(primary_key=True)
    subject_id = models.ForeignKey(Subjects, on_delete=models.DO_NOTHING)
    attendance_date = models.DateField()
    session_year_id = models.ForeignKey(SessionYearModel, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
```

```python
class AttendanceReport(models.Model):
    # Individual Student Attendance
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.DO_NOTHING)
    attendance_id = models.ForeignKey(Attendance, on_delete=models.CASCADE)
    status = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()


class LeaveReportStudent(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)

    # Leave types
    LEAVE_TYPE_CHOICES = (
        ('CL', 'Casual Leave'),
        ('OD', 'On-Duty Leave'),
        ('ML', 'Medical Leave'),
        ('EL', 'Emergency Leave')
    )
    leave_type = models.CharField(max_length=2, choices=LEAVE_TYPE_CHOICES, default='CL')

    # For On-Duty Leave categories
    OD_CATEGORY_CHOICES = (
        ('S', 'Short-Term (1 Day to 1 Week)'),
        ('L', 'Long-Term (More than 1 Week)')
    )
    od_category = models.CharField(max_length=1, choices=OD_CATEGORY_CHOICES, null=True, blank=True)

    # Date fields for leave period
    leave_date = models.CharField(max_length=255)  # Keeping existing field for backward compatibility
    leave_start_date = models.DateField(null=True, blank=True)
    leave_end_date = models.DateField(null=True, blank=True)

    # For Medical Leave documents
    medical_certificate = models.FileField(upload_to='medical_certificates/', null=True, blank=True)

    leave_message = models.TextField()

    # Leave status: 0-Pending, 1-Approved, 2-Rejected
    leave_status = models.IntegerField(default=0)
```

```
    # Timestamps
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

    def leave_days(self):
        """Calculate number of days in the leave period"""
        if self.leave_start_date and self.leave_end_date:
            return (self.leave_end_date - self.leave_start_date).days + 1
        return 1  # Default to 1 day if not specified


class LeaveReportStaff(models.Model):
    id = models.AutoField(primary_key=True)
    staff_id = models.ForeignKey(Staffs, on_delete=models.CASCADE)
    leave_date = models.CharField(max_length=255)
    leave_message = models.TextField()
    leave_status = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()


class FeedBackStudent(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)
    feedback = models.TextField()
    feedback_reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()


class FeedBackStaffs(models.Model):
    id = models.AutoField(primary_key=True)
    staff_id = models.ForeignKey(Staffs, on_delete=models.CASCADE)
    feedback = models.TextField()
    feedback_reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
```

HodViews.py

```
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect, JsonResponse
from django.contrib import messages
from django.core.files.storage import FileSystemStorage #To upload Profile Picture
from django.urls import reverse
from django.views.decorators.csrf import csrf_exempt
from django.core import serializers
import json
from .models import Students
from student_management_app.models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel, FeedBackStudent, FeedBackStaffs, LeaveReportStudent, LeaveReportStaff, Attendanc
from .forms import AddStudentForm, EditStudentForm


def admin_home(request):
    all_student_count = Students.objects.all().count()
    subject_count = Subjects.objects.all().count()
    course_count = Courses.objects.all().count()
    staff_count = Staffs.objects.all().count()

    # Total Subjects and students in Each Course
    course_all = Courses.objects.all()
    course_name_list = []
    subject_count_list = []
    student_count_list_in_course = []

    for course in course_all:
        subjects = Subjects.objects.filter(course_id=course.id).count()
        students = Students.objects.filter(course_id=course.id).count()
        course_name_list.append(course.course_name)
        subject_count_list.append(subjects)
        student_count_list_in_course.append(students)

    subject_all = Subjects.objects.all()
    subject_list = []
    student_count_list_in_subject = []
    for subject in subject_all:
        course = Courses.objects.get(id=subject.course_id.id)
        student_count = Students.objects.filter(course_id=course.id).count()
        subject_list.append(subject.subject_name)
        student_count_list_in_subject.append(student_count)

    # For Students
    student_attendance_present_list=[]
    student_attendance_leave_list=[]
    student_name_list=[]
```

```
    students = Students.objects.all()
    for student in students:
        attendance = AttendanceReport.objects.filter(student_id=student.id, status=True).count()
        absent = AttendanceReport.objects.filter(student_id=student.id, status=False).count()
        leaves = LeaveReportStudent.objects.filter(student_id=student.id, leave_status=1).count()
        student_attendance_present_list.append(attendance)
        student_attendance_leave_list.append(leaves+absent)
        student_name_list.append(student.admin.first_name+" "+ student.admin.last_name)


    context={
        "all_student_count": all_student_count,
        "subject_count": subject_count,
        "course_count": course_count,
        "staff_count": staff_count,
        "course_name_list": course_name_list,
        "subject_count_list": subject_count_list,
        "student_count_list_in_course": student_count_list_in_course,
        "subject_list": subject_list,
        "student_count_list_in_subject": student_count_list_in_subject,
        "student_attendance_present_list": student_attendance_present_list,
        "student_attendance_leave_list": student_attendance_leave_list,
        "student_name_list": student_name_list,
    }
    return render(request, "hod_template/home_content.html", context)


def add_staff(request):
    courses = Courses.objects.all()
    context = {
        "courses": courses
    }
    return render(request, "hod_template/add_staff_template.html", context)


def add_staff_save(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method ")
        return redirect('add_staff')
```

```
    else:
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        address = request.POST.get('address')
        department_id = request.POST.get('department')

        try:
            user = CustomUser.objects.create_user(username=username, password=password, email=email, first_name=first_name, last_name=last_name, user_type=2)
            user.staffs.address = address

            # Assign department if selected
            if department_id:
                department = Courses.objects.get(id=department_id)
                user.staffs.department = department

            user.save()
            messages.success(request, "Staff Added Successfully!")
            return redirect('add_staff')
        except Exception as e:
            messages.error(request, f"Failed to Add Staff: {str(e)}")
            return redirect('add_staff')


def manage_staff(request):
    staffs = Staffs.objects.all()
    context = {
        "staffs": staffs
    }
    return render(request, "hod_template/manage_staff_template.html", context)


def edit_staff(request, staff_id):
    staff = Staffs.objects.get(admin=staff_id)

    context = {
        "staff": staff,
        "id": staff_id
    }
    return render(request, "hod_template/edit_staff_template.html", context)
```

```python
def edit_staff_save(request):
    if request.method != "POST":
        return HttpResponse("<h2>Method Not Allowed</h2>")
    else:
        staff_id = request.POST.get('staff_id')
        username = request.POST.get('username')
        email = request.POST.get('email')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        address = request.POST.get('address')

        try:
            # INSERTING into Customuser Model
            user = CustomUser.objects.get(id=staff_id)
            user.first_name = first_name
            user.last_name = last_name
            user.email = email
            user.username = username
            user.save()

            # INSERTING into Staff Model
            staff_model = Staffs.objects.get(admin=staff_id)
            staff_model.address = address
            staff_model.save()

            messages.success(request, "Staff Updated Successfully.")
            return redirect('/edit_staff/'+staff_id)

        except:
            messages.error(request, "Failed to Update Staff.")
            return redirect('/edit_staff/'+staff_id)


def delete_staff(request, staff_id):
    staff = Staffs.objects.get(admin=staff_id)
    try:
        staff.delete()
        messages.success(request, "Staff Deleted Successfully.")
        return redirect('manage_staff')
    except:
        messages.error(request, "Failed to Delete Staff.")
        return redirect('manage_staff')
```

```python
    def __str__(self):
        return f"{self.name} - {self.date}"

    class Meta:
        ordering = ['date']
        verbose_name_plural = 'Holidays'


class OTPVerification(models.Model):
    id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    otp = models.CharField(max_length=6)
    is_verified = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def is_expired(self):
        """Check if OTP is expired based on settings.OTP_EXPIRY_TIME (in minutes)"""
        from django.utils import timezone
        from django.conf import settings

        expiry_time = self.created_at + timezone.timedelta(minutes=settings.OTP_EXPIRY_TIME)
        return timezone.now() > expiry_time
    objects = models.Manager()


#Creating Django Signals

# It's like trigger in database. It will run only when Data is Added in CustomUser model

@receiver(post_save, sender=CustomUser)
# Now Creating a Function which will automatically insert data in HOD, Staff or Student
def create_user_profile(sender, instance, created, **kwargs):
    # if Created is true (Means Data Inserted)
    if created:
        # Check the user_type and insert the data in respective tables
        if instance.user_type == 1:
            AdminHOD.objects.create(admin=instance)
        if instance.user_type == 2:
            Staffs.objects.create(admin=instance)
```

```
        try:
            course = Courses.objects.get(id=1)
        except Courses.DoesNotExist:
            # Create a default course if none exists
            course = Courses.objects.create(course_name="Default Course")

        try:
            session_year = SessionYearModel.objects.first()
            if not session_year:
                # Create a default session if none exists
                from datetime import date
                session_year = SessionYearModel.objects.create(
                    session_start_year=date.today(),
                    session_end_year=date(date.today().year + 1, date.today().month, date.today().day)
                )
        except Exception as e:
            print(f"Error creating default session year: {e}")

        Students.objects.create(
            admin=instance,
            course_id=course,
            session_year_id=session_year,
            address="",
            profile_pic="",
            gender=""
        )


@receiver(post_save, sender=CustomUser)
def save_user_profile(sender, instance, **kwargs):
    if instance.user_type == 1:
        instance.adminhod.save()
    if instance.user_type == 2:
        instance.staffs.save()
    if instance.user_type == 3:
        instance.students.save()
```

```
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect, JsonResponse
from django.contrib import messages
from django.core.files.storage import FileSystemStorage #To upload Profile Picture
from django.urls import reverse
from django.views.decorators.csrf import csrf_exempt
from django.core import serializers
import json
from .models import Students
from student_management_app.models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel, FeedBackStudent, FeedBackStaffs, LeaveReportStudent, LeaveReportStaff, Attendanc
from .forms import AddStudentForm, EditStudentForm


def admin_home(request):
    all_student_count = Students.objects.all().count()
    subject_count = Subjects.objects.all().count()
    course_count = Courses.objects.all().count()
    staff_count = Staffs.objects.all().count()

    # Total Subjects and students in Each Course
    course_all = Courses.objects.all()
    course_name_list = []
    subject_count_list = []
    student_count_list_in_course = []

    for course in course_all:
        subjects = Subjects.objects.filter(course_id=course.id).count()
        students = Students.objects.filter(course_id=course.id).count()
        course_name_list.append(course.course_name)
        subject_count_list.append(subjects)
        student_count_list_in_course.append(students)

    subject_all = Subjects.objects.all()
    subject_list = []
    student_count_list_in_subject = []
    for subject in subject_all:
        course = Courses.objects.get(id=subject.course_id.id)
        student_count = Students.objects.filter(course_id=course.id).count()
        subject_list.append(subject.subject_name)
        student_count_list_in_subject.append(student_count)

    # For Students
    student_attendance_present_list=[]
    student_attendance_leave_list=[]
    student_name_list=[]
```

StaffView.py

```python
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect, JsonResponse
from django.contrib import messages
from django.core.files.storage import FileSystemStorage #To upload Profile Picture
from django.urls import reverse
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings
from django.core import serializers
import json
from student_management_app.models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel, FeedBackStudent, FeedBackStaffs, LeaveReportStudent, LeaveReportStaff, Attendance,
from .forms import AddStudentForm, EditStudentForm
from student_management_app.utils import send_leave_status_update_email

from student_management_app.models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel, Attendance, AttendanceReport, LeaveReportStaff, FeedBackStaffs, StudentResult


def staff_home(request):
    # Fetching All Students under Staff

    subjects = Subjects.objects.filter(staff_id=request.user.id)
    course_id_list = []
    for subject in subjects:
        course = Courses.objects.get(id=subject.course_id.id)
        course_id_list.append(course.id)

    final_course = []
    # Removing Duplicate Course Id
    for course_id in course_id_list:
        if course_id not in final_course:
            final_course.append(course_id)

    students_count = Students.objects.filter(course_id__in=final_course).count()
    subject_count = subjects.count()

    # Fetch All Attendance Count
    attendance_count = Attendance.objects.filter(subject_id__in=subjects).count()
    # Fetch All Approve Leave
    staff = Staffs.objects.get(admin=request.user.id)
    leave_count = LeaveReportStaff.objects.filter(staff_id=staff.id, leave_status=1).count()

    #Fetch Attendance Data by Subjects
    subject_list = []
    attendance_list = []
    for subject in subjects:
        attendance_count1 = Attendance.objects.filter(subject_id=subject.id).count()
        subject_list.append(subject.subject_name)
        attendance_list.append(attendance_count1)
```

```python
    students_attendance = Students.objects.filter(course_id__in=final_course)
    student_list = []
    student_list_attendance_present = []
    student_list_attendance_absent = []
    for student in students_attendance:
        attendance_present_count = AttendanceReport.objects.filter(status=True, student_id=student.id).count()
        attendance_absent_count = AttendanceReport.objects.filter(status=False, student_id=student.id).count()
        student_list.append(student.admin.first_name+" "+ student.admin.last_name)
        student_list_attendance_present.append(attendance_present_count)
        student_list_attendance_absent.append(attendance_absent_count)

    context={
        "students_count": students_count,
        "attendance_count": attendance_count,
        "leave_count": leave_count,
        "subject_count": subject_count,
        "subject_list": subject_list,
        "attendance_list": attendance_list,
        "student_list": student_list,
        "attendance_present_list": student_list_attendance_present,
        "attendance_absent_list": student_list_attendance_absent
    }
    return render(request, "staff_template/staff_home_template.html", context)



def staff_take_attendance(request):
    subjects = Subjects.objects.filter(staff_id=request.user.id)
    session_years = SessionYearModel.objects.all()
    context = {
        "subjects": subjects,
        "session_years": session_years
    }
    return render(request, "staff_template/take_attendance_template.html", context)



def staff_apply_leave(request):
    staff_obj = Staffs.objects.get(admin=request.user.id)
    leave_data = LeaveReportStaff.objects.filter(staff_id=staff_obj)
    context = {
        "leave_data": leave_data
    }
    return render(request, "staff_template/staff_apply_leave_template.html", context)
```

```python
def staff_apply_leave_save(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method")
        return redirect('staff_apply_leave')
    else:
        leave_date = request.POST.get('leave_date')
        leave_message = request.POST.get('leave_message')

        staff_obj = Staffs.objects.get(admin=request.user.id)
        try:
            leave_report = LeaveReportStaff(staff_id=staff_obj, leave_date=leave_date, leave_message=leave_message, leave_status=0)
            leave_report.save()
            messages.success(request, "Applied for Leave.")
            return redirect('staff_apply_leave')
        except:
            messages.error(request, "Failed to Apply Leave")
            return redirect('staff_apply_leave')


def staff_feedback(request):
    staff_obj = Staffs.objects.get(admin=request.user.id)
    feedback_data = FeedBackStaffs.objects.filter(staff_id=staff_obj)
    context = {
        "feedback_data":feedback_data
    }
    return render(request, "staff_template/staff_feedback_template.html", context)


def staff_feedback_save(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method.")
        return redirect('staff_feedback')
    else:
        feedback = request.POST.get('feedback_message')
        staff_obj = Staffs.objects.get(admin=request.user.id)

        try:
            add_feedback = FeedBackStaffs(staff_id=staff_obj, feedback=feedback, feedback_reply="")
            add_feedback.save()
            messages.success(request, "Feedback Sent.")
            return redirect('staff_feedback')
        except:
            messages.error(request, "Failed to Send Feedback.")
            return redirect('staff_feedback')
```

StudentsView.py

```python
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect
from django.contrib import messages
from django.core.files.storage import FileSystemStorage #To upload Profile Picture
from django.urls import reverse
import datetime # To Parse input DateTime into Python Date Time Object
import calendar
from django.db.models import Q
import json
import datetime
import calendar
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect, JsonResponse
from django.contrib import messages
from django.core.files.storage import FileSystemStorage
from django.urls import reverse
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings

from student_management_app.models import CustomUser, Staffs, Courses, Subjects, Students, Attendance, AttendanceReport, LeaveReportStudent, FeedBackStudent, StudentResult, Holiday
from student_management_app.utils import send_leave_application_email, send_leave_limit_exceeded_email


def student_home(request):
    try:
        student_obj = Students.objects.get(admin=request.user.id)

        # Initialize variables with default values
        total_attendance = 0
        attendance_present = 0
        attendance_absent = 0
        total_subjects = 0
        subject_name = []
        data_present = []
        data_absent = []

        # Get attendance stats
        total_attendance = AttendanceReport.objects.filter(student_id=student_obj).count()
        attendance_present = AttendanceReport.objects.filter(student_id=student_obj, status=True).count()
        attendance_absent = AttendanceReport.objects.filter(student_id=student_obj, status=False).count()

        # Get course info if available
        if student_obj.course_id:
            try:
                course_obj = Courses.objects.get(id=student_obj.course_id.id)
                total_subjects = Subjects.objects.filter(course_id=course_obj).count()
```

```
            # Get subject attendance data
            subject_data = Subjects.objects.filter(course_id=student_obj.course_id)
            for subject in subject_data:
                attendance = Attendance.objects.filter(subject_id=subject.id)
                attendance_present_count = AttendanceReport.objects.filter(
                    attendance_id__in=attendance, status=True, student_id=student_obj.id).count()
                attendance_absent_count = AttendanceReport.objects.filter(
                    attendance_id__in=attendance, status=False, student_id=student_obj.id).count()
                subject_name.append(subject.subject_name)
                data_present.append(attendance_present_count)
                data_absent.append(attendance_absent_count)
        except Courses.DoesNotExist:
            # Handle case where course doesn't exist
            messages.error(request, "Your course information is incomplete. Please contact admin.")

    context={
        "total_attendance": total_attendance,
        "attendance_present": attendance_present,
        "attendance_absent": attendance_absent,
        "total_subjects": total_subjects,
        "subject_name": subject_name,
        "data_present": data_present,
        "data_absent": data_absent,
        "student": student_obj
    }
    return render(request, "student_template/student_home_template.html", context)

except Students.DoesNotExist:
    messages.error(request, "Your student profile is not set up correctly. Please contact admin.")
    return redirect("logout_user")
except Exception as e:
    messages.error(request, f"An error occurred: {str(e)}")
    return redirect("logout_user")


def student_view_attendance(request):
    student = Students.objects.get(admin=request.user.id) # Getting Logged in Student Data
    course = student.course_id # Getting Course Enrolled of LoggedIn Student
    # course = Courses.objects.get(id=student.course_id.id) # Getting Course Enrolled of LoggedIn Student
    subjects = Subjects.objects.filter(course_id=course) # Getting the Subjects of Course Enrolled
    context = {
        "subjects": subjects
    }
    return render(request, "student_template/student_view_attendance.html", context)
```

```
def student_view_attendance_post(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method")
        return redirect('student_view_attendance')
    else:
        # Getting all the Input Data
        subject_id = request.POST.get('subject')
        start_date = request.POST.get('start_date')
        end_date = request.POST.get('end_date')

        # Parsing the date data into Python object
        start_date_parse = datetime.datetime.strptime(start_date, '%Y-%m-%d').date()
        end_date_parse = datetime.datetime.strptime(end_date, '%Y-%m-%d').date()

        # Getting all the Subject Data based on Selected Subject
        subject_obj = Subjects.objects.get(id=subject_id)
        # Getting Logged In User Data
        user_obj = CustomUser.objects.get(id=request.user.id)
        # Getting Student Data Based on Logged in Data
        stud_obj = Students.objects.get(admin=user_obj)

        # Now Accessing Attendance Data based on the Range of Date Selected and Subject Selected
        attendance = Attendance.objects.filter(attendance_date__range=(start_date_parse, end_date_parse), subject_id=subject_obj)
        # Getting Attendance Report based on the attendance details obtained above
        attendance_reports = AttendanceReport.objects.filter(attendance_id__in=attendance, student_id=stud_obj)

        # for attendance_report in attendance_reports:
        #     print("Date: "+ str(attendance_report.attendance_id.attendance_date), "Status: "+ str(attendance_report.status))

        # messages.success(request, "Attendacne View Success")

        context = {
            "subject_obj": subject_obj,
            "attendance_reports": attendance_reports
        }

        return render(request, 'student_template/student_attendance_data.html', context)


def count_monthly_leaves(student_obj, month=None, year=None):
    """
    Count the number of CL and OD leaves taken by a student in a specific month
    """
    if month is None or year is None:
        # Use current month and year if not specified
        today = datetime.datetime.now()
        month, year = today.month, today.year
```

```
    # Filter leaves by month and year and leave type (CL or OD)
    leaves = LeaveReportStudent.objects.filter(
        student_id=student_obj,
        leave_type__in=['CL', 'OD'],
        leave_status=1,  # Only count approved leaves
        created_at__month=month,
        created_at__year=year
    )

    return leaves.count()

def student_apply_leave(request):
    try:
        student_obj = Students.objects.get(admin=request.user.id)
        leave_data = LeaveReportStudent.objects.filter(student_id=student_obj).order_by('-created_at')

        # Get current month and year for leave count display
        today = datetime.datetime.now()
        current_month_name = calendar.month_name[today.month]
        current_year = today.year

        # Count leaves taken in current month
        monthly_cl_od_count = count_monthly_leaves(student_obj)
        monthly_leave_limit = 4  # As per requirements
        can_apply_regular_leave = monthly_cl_od_count < monthly_leave_limit

        context = {
            "leave_data": leave_data,
            "leave_types": LeaveReportStudent.LEAVE_TYPE_CHOICES,
            "od_categories": LeaveReportStudent.OD_CATEGORY_CHOICES,
            "monthly_cl_od_count": monthly_cl_od_count,
            "monthly_leave_limit": monthly_leave_limit,
            "can_apply_regular_leave": can_apply_regular_leave,
            "current_month": current_month_name,
            "current_year": current_year
        }
        return render(request, 'student_template/student_apply_leave.html', context)
    except Exception as e:
        messages.error(request, f"An error occurred: {str(e)}")
        return redirect('student_home')
```

EmailBackEnd.py

```
from django.contrib.auth import get_user_model
from django.contrib.auth.backends import ModelBackend


class EmailBackEnd(ModelBackend):
    def authenticate(self, request, username=None, password=None, **kwargs):
        UserModel = get_user_model()
        try:
            user = UserModel.objects.get(email=username)
            if user.check_password(password):
                # Don't require OTP for admin or staff
                if user.user_type in ['1', '2']:  # Admin or Staff
                    return user

                # For students, we'll handle OTP verification in the view
                return user
        except UserModel.DoesNotExist:
            return None
        return None
```

Utils.py

```python
import random
import string
from django.core.mail import send_mail
from django.conf import settings
from django.template.loader import render_to_string
from django.utils.html import strip_tags

def generate_otp(length=6):
    """Generate a random OTP of specified length"""
    return ''.join(random.choices(string.digits, k=length))

def send_otp_email(user, otp):
    """
    Send OTP to user's email
    """
    subject = 'OTP Verification for Leave Management System'
    html_message = render_to_string('email_templates/otp_email.html', {
        'user': user,
        'otp': otp,
        'expiry_time': settings.OTP_EXPIRY_TIME
    })
    plain_message = strip_tags(html_message)

    try:
        send_mail(
            subject,
            plain_message,
            settings.EMAIL_HOST_USER,
            [user.email],
            html_message=html_message,
            fail_silently=False
        )
        return True, None
    except Exception as e:
        error_message = str(e)
        print(f"Error sending email: {error_message}")

        # Check if it's a credentials issue
        if "SMTPAuthenticationError" in error_message or "535" in error_message:
            return False, "Email authentication failed. Please check your email provider settings."
        # Check if it's a connection issue
        elif "Connection refused" in error_message or "10061" in error_message:
            return False, "Could not connect to email server. Please check your internet connection and email settings."
        else:
            return False, f"Failed to send email: {error_message}"
```

```python
def send_leave_application_email(student, leave_report):
    """
    Send email notification about leave application submission
    """
    subject = 'Leave Application Submitted Successfully'

    # Determine leave type display name
    from student_management_app.models import LeaveReportStudent
    leave_types = dict(LeaveReportStudent.LEAVE_TYPE_CHOICES)
    leave_type_display = leave_types.get(leave_report.leave_type, leave_report.leave_type)

    html_message = render_to_string('email_templates/leave_application.html', {
        'student': student,
        'leave_report': leave_report,
        'leave_type': leave_type_display,
        'start_date': leave_report.leave_start_date,
        'end_date': leave_report.leave_end_date
    })
    plain_message = strip_tags(html_message)

    try:
        send_mail(
            subject,
            plain_message,
            settings.EMAIL_HOST_USER,
            [student.admin.email],
            html_message=html_message,
            fail_silently=False
        )
    except Exception as e:
        error_message = str(e)
        print(f"Error sending email: {error_message}")

        # Check if it's a credentials issue
        if "SMTPAuthenticationError" in error_message or "535" in error_message:
            print("Email authentication failed. Please check your email provider settings.")
        # Check if it's a connection issue
        elif "Connection refused" in error_message or "10061" in error_message:
            print("Could not connect to email server. Please check your internet connection and email settings.")
        else:
            print(f"Failed to send email: {error_message}")
```

```python
def send_leave_status_update_email(student, leave_report, status):
    """
    Send email notification about leave application status update
    """
    status_text = "Approved" if status == 1 else "Rejected"
    subject = f'Leave Application {status_text}'

    # Determine leave type display name
    if hasattr(leave_report, 'leave_type'):
        from student_management_app.models import LeaveReportStudent
        leave_types = dict(LeaveReportStudent.LEAVE_TYPE_CHOICES)
        leave_type_display = leave_types.get(leave_report.leave_type, leave_report.leave_type)
    else:
        leave_type_display = "Leave"

    # Get dates from leave report
    start_date = getattr(leave_report, 'leave_start_date',
                         getattr(leave_report, 'leave_date', 'Not specified'))
    end_date = getattr(leave_report, 'leave_end_date', start_date)

    html_message = render_to_string('email_templates/leave_status_update.html', {
        'student': student,
        'leave_report': leave_report,
        'leave_type': leave_type_display,
        'start_date': start_date,
        'end_date': end_date,
        'status': status_text
    })
    plain_message = strip_tags(html_message)

    # Make sure we have a valid email
    if hasattr(student, 'admin') and hasattr(student.admin, 'email'):
        recipient_email = student.admin.email
    else:
        # If student object doesn't have admin attribute, it might be a CustomUser
        recipient_email = getattr(student, 'email', None)

    if not recipient_email:
        raise ValueError("No valid email address found for this student")
```

Views.py

```python
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponseRedirect, HttpResponse
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.views.decorators.csrf import ensure_csrf_cookie, csrf_exempt
from django.utils.decorators import method_decorator
from django.contrib.auth.backends import ModelBackend
from django.conf import settings
from django.utils import timezone
from django.urls import reverse
import datetime
import logging

from student_management_app.EmailBackEnd import EmailBackEnd
from student_management_app.models import CustomUser, OTPVerification, Students
from student_management_app.utils import generate_otp, send_otp_email


def clear_sessions():
    # Clear all sessions to fix corrupted session data
    Session.objects.all().delete()


def home(request):
    # First clear any corrupted sessions
    try:
        # Check if the session is corrupted
        request.session.test_cookie_worked()
    except:
        # If we get an error, clear the session
        if hasattr(request, 'session'):
            request.session.flush()
        # Also clear the cookies
        response = render(request, 'index.html')
        response.delete_cookie('sessionid')
        return response

    return render(request, 'index.html')
```

```python
@ensure_csrf_cookie
def loginPage(request):
    if request.user.is_authenticated:
        if request.user.user_type == '1':
            return redirect('admin_home')
        elif request.user.user_type == '2':
            return redirect('staff_home')
        else:
            return redirect('student_home')
    return render(request, 'login.html')


@csrf_exempt
def doLogin(request):
    if request.method != "POST":
        return HttpResponse("<h2>Method Not Allowed</h2>")
    else:
        # Get the authentication backend
        email = request.POST.get('email')
        password = request.POST.get('password')
        user_type = request.POST.get('user_type', '')

        # Use Django's authenticate function instead of calling the backend directly
        user = authenticate(request, username=email, password=password)

        # Check if user exists and matches the selected user type
        if user is not None:
            # If a specific user type was selected, verify that the user matches that type
            if user_type and user.user_type != user_type:
                if user_type == '1':
                    messages.error(request, "This account is not a Technical account!")
                elif user_type == '2':
                    messages.error(request, "This account is not a Teacher account!")
                elif user_type == '3':
                    messages.error(request, "This account is not a Student account!")
                else:
                    messages.error(request, "Invalid user type selected!")
                return redirect('login')

            # Check if user is a student who needs OTP verification
            if user.user_type == '3':  # Student
                # Generate and save OTP
                otp = generate_otp()

                # Delete any existing OTP records for this user
                OTPVerification.objects.filter(user=user).delete()
```

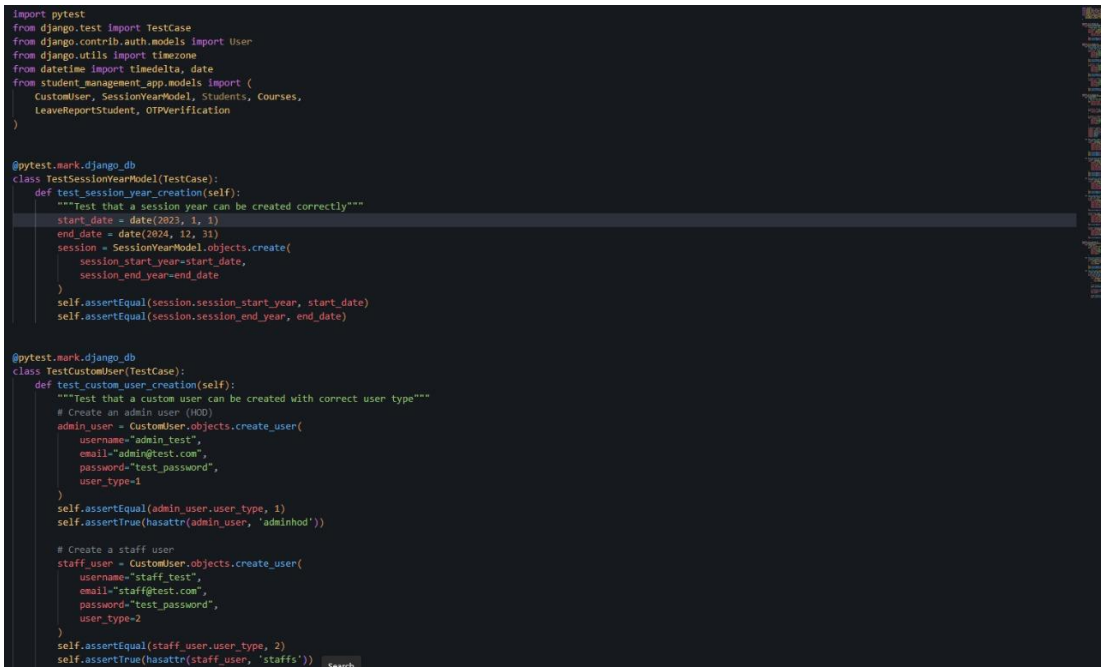
```python
                otp_record = OTPVerification(
                    user=user,
                    otp=otp,
                    is_verified=False
                )
                otp_record.save()

                # Send OTP by email
                try:
                    success, error_message = send_otp_email(user, otp)

                    # Pass user ID to OTP verification page
                    context = {
                        'user_id': user.id,
                        'expiry_time': settings.OTP_EXPIRY_TIME
                    }

                    if not success:
                        messages.error(request, error_message)
                        # Still allow testing with the generated OTP
                        context['show_test_otp'] = True
                        context['test_otp'] = otp
                        context['email_failed'] = True

                    return render(request, 'otp_verification.html', context)
                except Exception as e:
                    messages.error(request, f"Error in OTP process: {str(e)}")
                    return redirect('login')

            # For non-student users, proceed with login
            login(request, user)
            user_type = user.user_type

            if user_type == '1':
                return redirect('admin_home')

            elif user_type == '2':
                return redirect('staff_home')

            else:
                messages.error(request, "Invalid Login!")
                return redirect('login')
```

8. SOFTWARE ENGINEERING TOOLS USED

Block Box Testing:

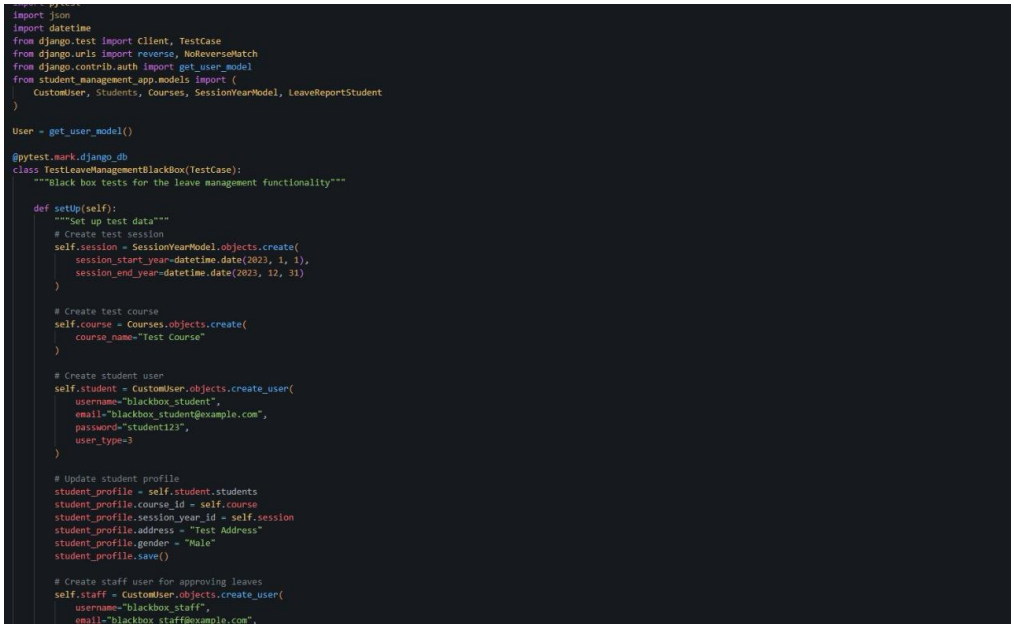
Black-box testing checks the functionality of the application without knowing its internal code structure. It is often used for UI testing, functional testing, and end-to-end testing.

Test_authentication.py



```python
import pytest
from django.test import TestCase
from django.contrib.auth.models import User
from django.utils import timezone
from datetime import timedelta, date
from student_management_app.models import (
    CustomUser, SessionYearModel, Students, Courses,
    LeaveReportStudent, OTPVerification
)


@pytest.mark.django_db
class TestSessionYearModel(TestCase):
    def test_session_year_creation(self):
        """Test that a session year can be created correctly"""
        start_date = date(2023, 1, 1)
        end_date = date(2024, 12, 31)
        session = SessionYearModel.objects.create(
            session_start_year=start_date,
            session_end_year=end_date
        )
        self.assertEqual(session.session_start_year, start_date)
        self.assertEqual(session.session_end_year, end_date)


@pytest.mark.django_db
class TestCustomUser(TestCase):
    def test_custom_user_creation(self):
        """Test that a custom user can be created with correct user type"""
        # Create an admin user (HOD)
        admin_user = CustomUser.objects.create_user(
            username="admin_test",
            email="admin@test.com",
            password="test_password",
            user_type=1
        )
        self.assertEqual(admin_user.user_type, 1)
        self.assertTrue(hasattr(admin_user, 'adminhod'))

        # Create a staff user
        staff_user = CustomUser.objects.create_user(
            username="staff_test",
            email="staff@test.com",
            password="test_password",
            user_type=2
        )
        self.assertEqual(staff_user.user_type, 2)
        self.assertTrue(hasattr(staff_user, 'staffs'))
```

Name test_leave_management.py



```python
import json
import datetime
from django.test import Client, TestCase
from django.urls import reverse, NoReverseMatch
from django.contrib.auth import get_user_model
from student_management_app.models import (
    CustomUser, Students, Courses, SessionYearModel, LeaveReportStudent
)

User = get_user_model()

@pytest.mark.django_db
class TestLeaveManagementBlackBox(TestCase):
    """Black box tests for the leave management functionality"""

    def setUp(self):
        """Set up test data"""
        # Create test session
        self.session = SessionYearModel.objects.create(
            session_start_year=datetime.date(2023, 1, 1),
            session_end_year=datetime.date(2023, 12, 31)
        )

        # Create test course
        self.course = Courses.objects.create(
            course_name="Test Course"
        )

        # Create student user
        self.student = CustomUser.objects.create_user(
            username="blackbox_student",
            email="blackbox_student@example.com",
            password="student123",
            user_type=3
        )

        # Update student profile
        student_profile = self.student.students
        student_profile.course_id = self.course
        student_profile.session_year_id = self.session
        student_profile.address = "Test Address"
        student_profile.gender = "Male"
        student_profile.save()

        # Create staff user for approving leaves
        self.staff = CustomUser.objects.create_user(
            username="blackbox_staff",
            email="blackbox_staff@example.com",
```

TEST CASES STATISTICS:

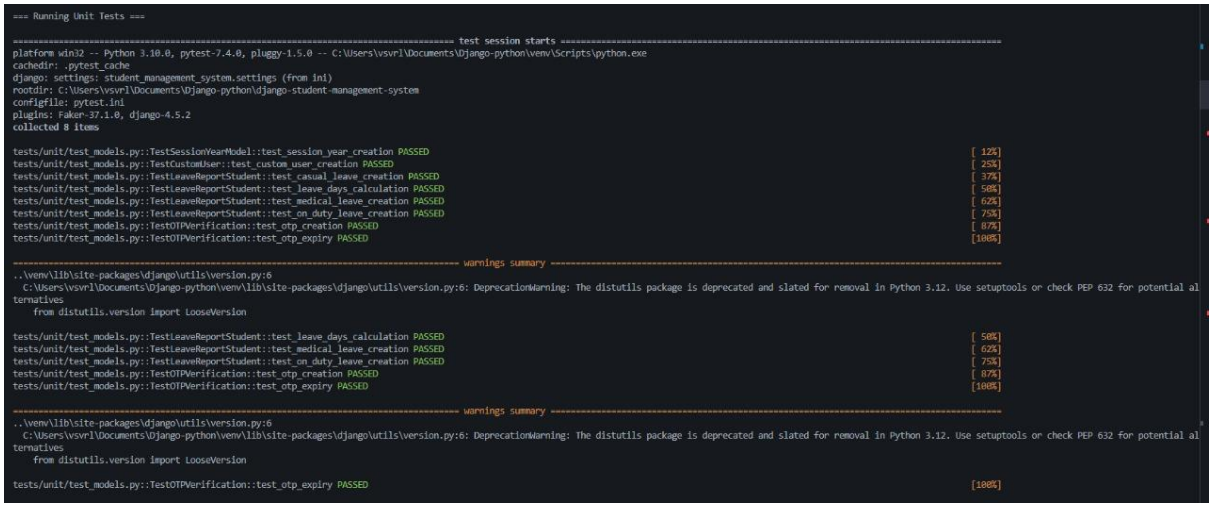
```
=== Running Black Box Tests ===

============================= test session starts ==============================
platform win32 -- Python 3.10.0, pytest-7.4.0, pluggy-1.5.0 -- C:\Users\vsvrl\Documents\Django-python\venv\Scripts\python.exe
cachedir: .pytest_cache
django: settings: student_management_system.settings (from ini)
rootdir: C:\Users\vsvrl\Documents\Django-python\django-student-management-system
configfile: pytest.ini
plugins: Faker-37.1.0, django-4.5.2
collected 11 items

tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_expired_otp_rejection FAILED                                  [  9%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_invalid_otp_rejection PASSED                                  [ 18%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_login_page_access PASSED                                      [ 27%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_student_login_with_otp PASSED                                 [ 36%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_teacher_user_login PASSED                                     [ 45%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_technical_user_login PASSED                                   [ 54%]
tests/blackbox/test_authentication.py::TestAuthenticationBlackBox::test_wrong_user_type_selection FAILED                              [ 63%]
tests/blackbox/test_leave_management.py::TestLeaveManagementBlackBox::test_casual_leave_application_workflow SKIPPED (Could not find the staff_approve_leave URL - update test with corr...) [ 72%]
tests/blackbox/test_leave_management.py::TestLeaveManagementBlackBox::test_leave_date_validation PASSED                               [ 81%]
tests/blackbox/test_leave_management.py::TestLeaveManagementBlackBox::test_medical_leave_without_document PASSED                      [ 90%]
tests/blackbox/test_leave_management.py::TestLeaveManagementBlackBox::test_monthly_leave_limit_enforcement PASSED                     [100%]
```

Unit Testing:

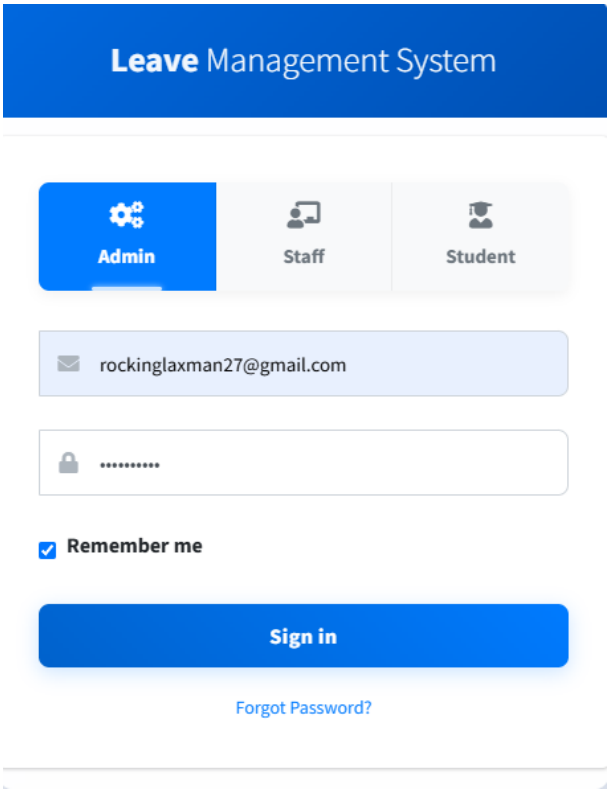
Unit testing is a low-level test that verifies individual components (functions, methods, or classes) of the application. It ensures that each piece of code works correctly in isolation.

```python
import pytest
from django.test import TestCase
from django.contrib.auth.models import User
from django.utils import timezone
from datetime import timedelta, date
from student_management_app.models import (
    CustomUser, SessionYearModel, Students, Courses,
    LeaveReportStudent, OTPVerification
)


@pytest.mark.django_db
class TestSessionYearModel(TestCase):
    def test_session_year_creation(self):
        """Test that a session year can be created correctly"""
        start_date = date(2023, 1, 1)
        end_date = date(2024, 12, 31)
        session = SessionYearModel.objects.create(
            session_start_year=start_date,
            session_end_year=end_date
        )
        self.assertEqual(session.session_start_year, start_date)
        self.assertEqual(session.session_end_year, end_date)


@pytest.mark.django_db
class TestCustomUser(TestCase):
    def test_custom_user_creation(self):
        """Test that a custom user can be created with correct user type"""
        # Create an admin user (HOD)
        admin_user = CustomUser.objects.create_user(
            username="admin_test",
            email="admin@test.com",
            password="test_password",
            user_type=1
        )
        self.assertEqual(admin_user.user_type, 1)
        self.assertTrue(hasattr(admin_user, 'adminhod'))

        # Create a staff user
        staff_user = CustomUser.objects.create_user(
            username="staff_test",
            email="staff@test.com",
            password="test_password",
            user_type=2
        )
        self.assertEqual(staff_user.user_type, 2)
        self.assertTrue(hasattr(staff_user, 'staffs'))
```

TEST CASES STATISTICS

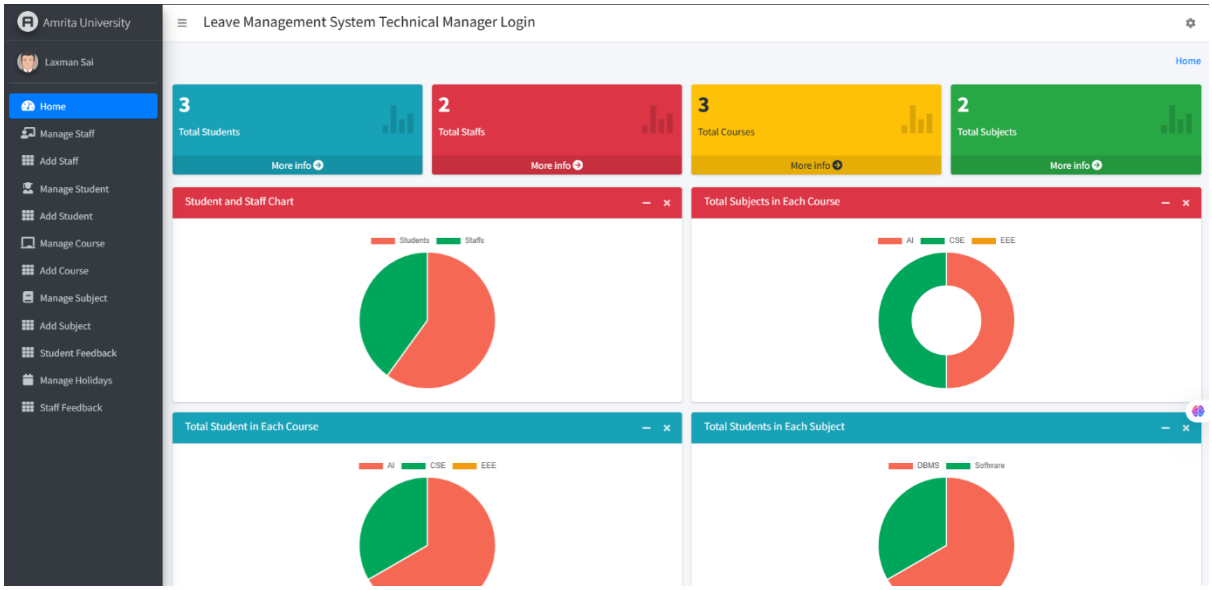


9. SCREENSHOTS OF THE PROJECT

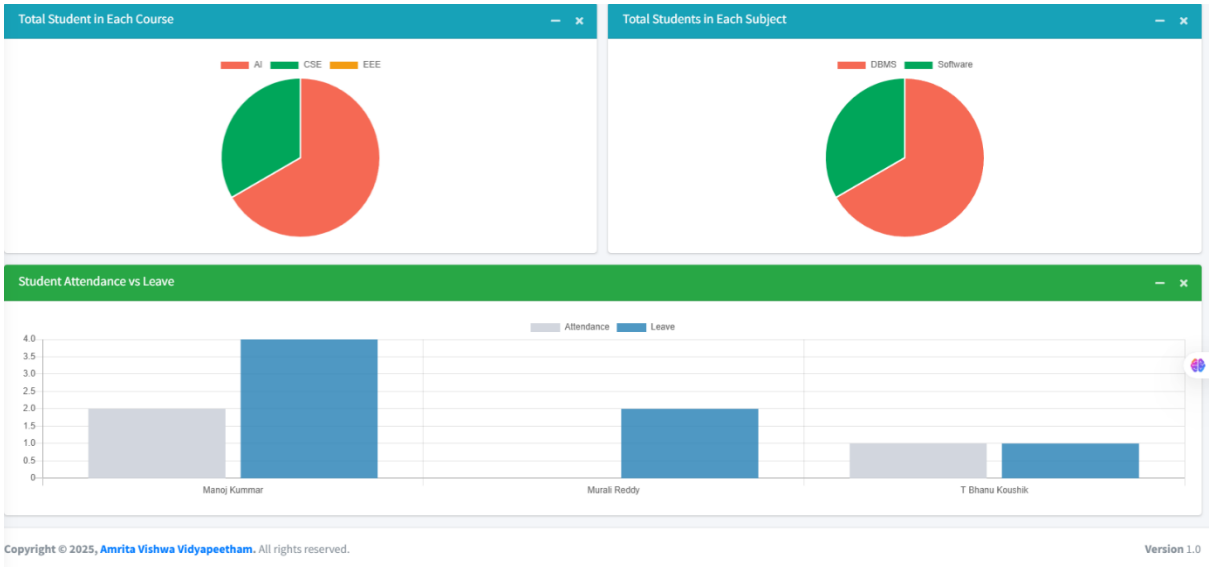


Login Page

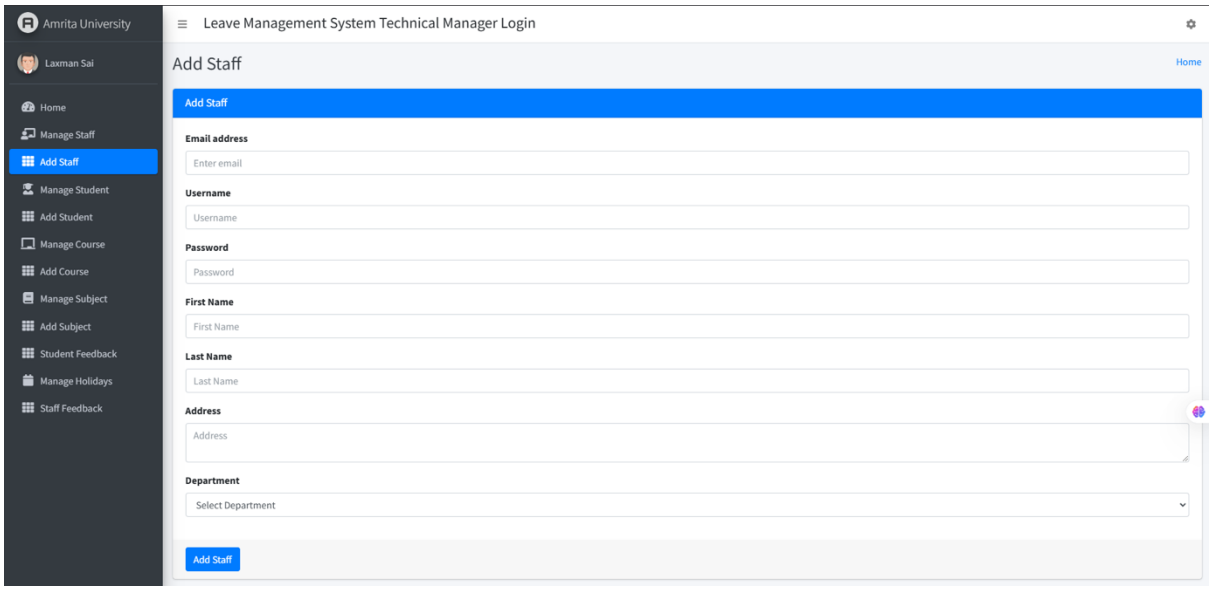
TECHNICAL MANAGEMENT DASHBOARD



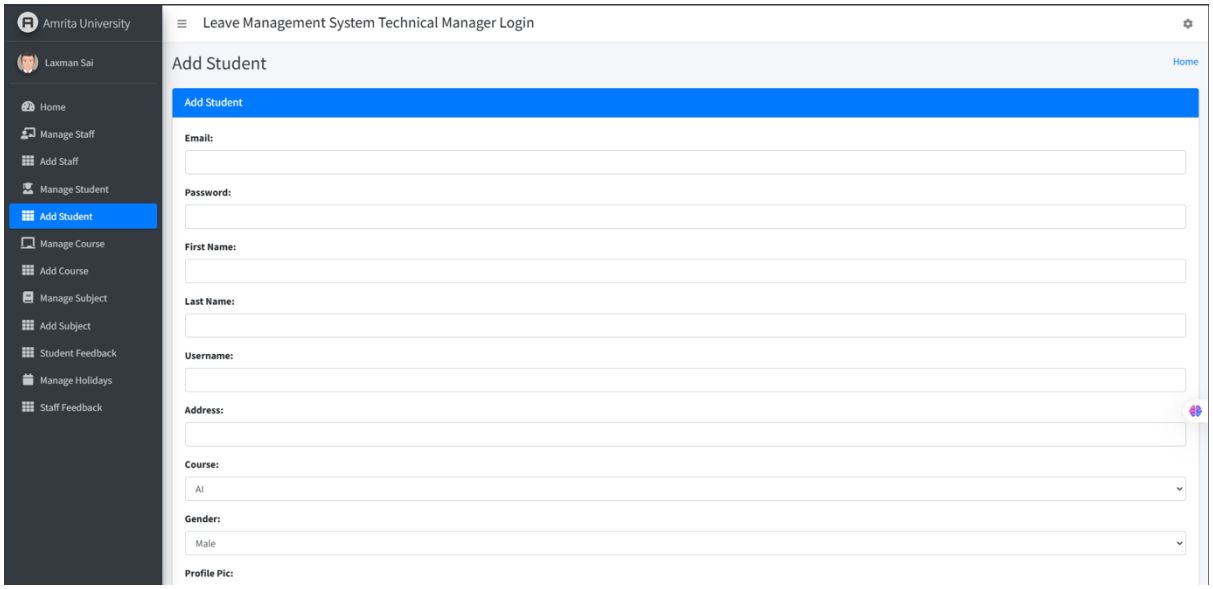
Admin Dashboard



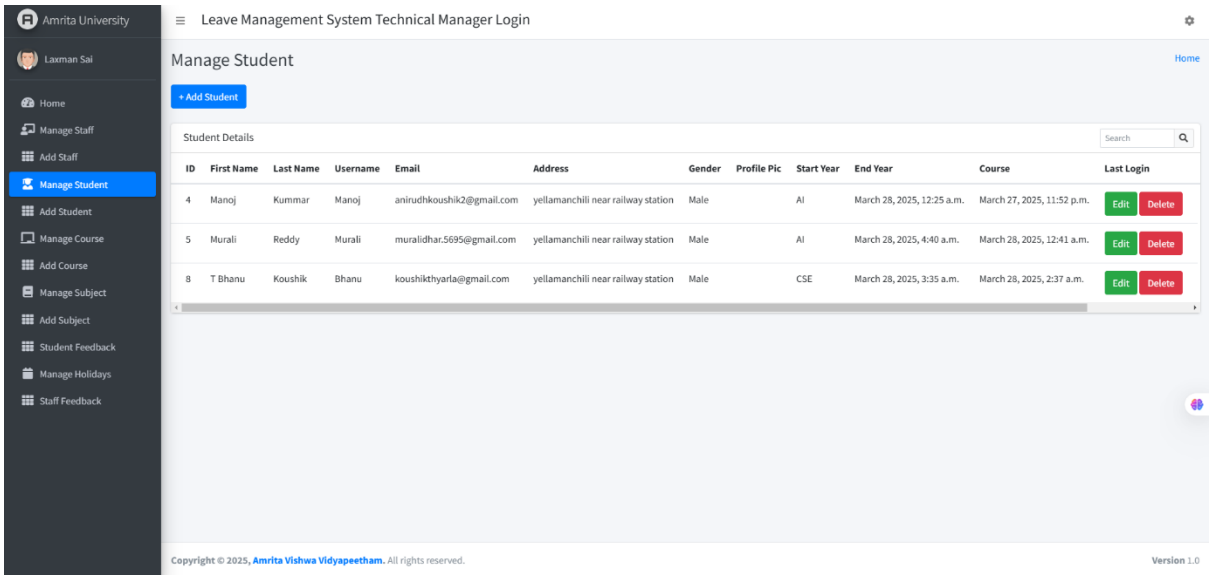
Admin Dashboard



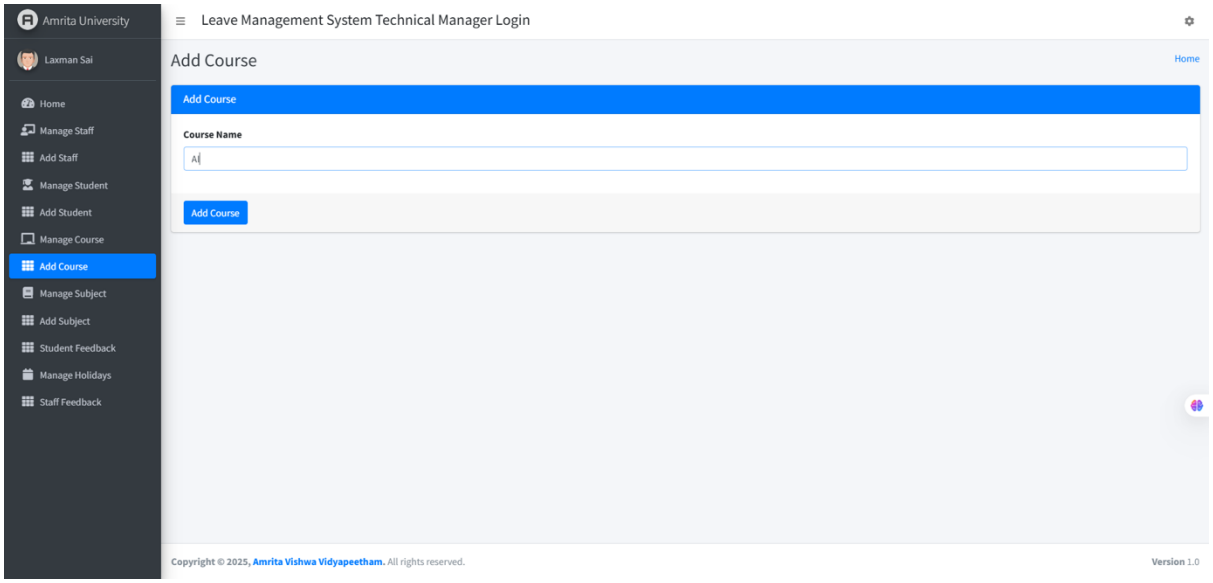
Admin can Add Staff



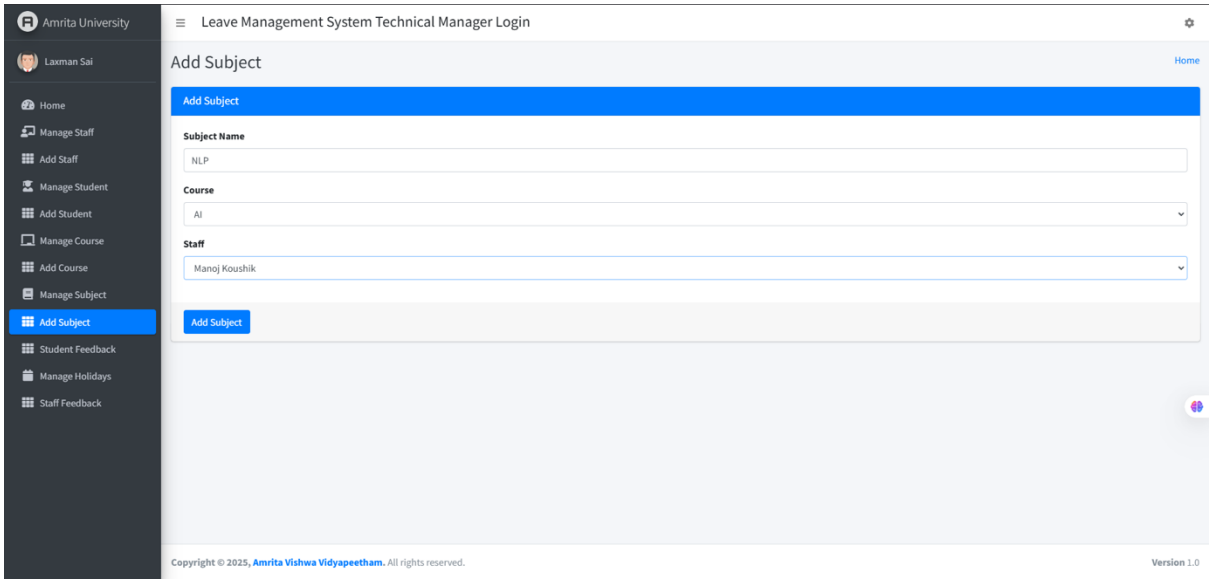
Admin can Add Students



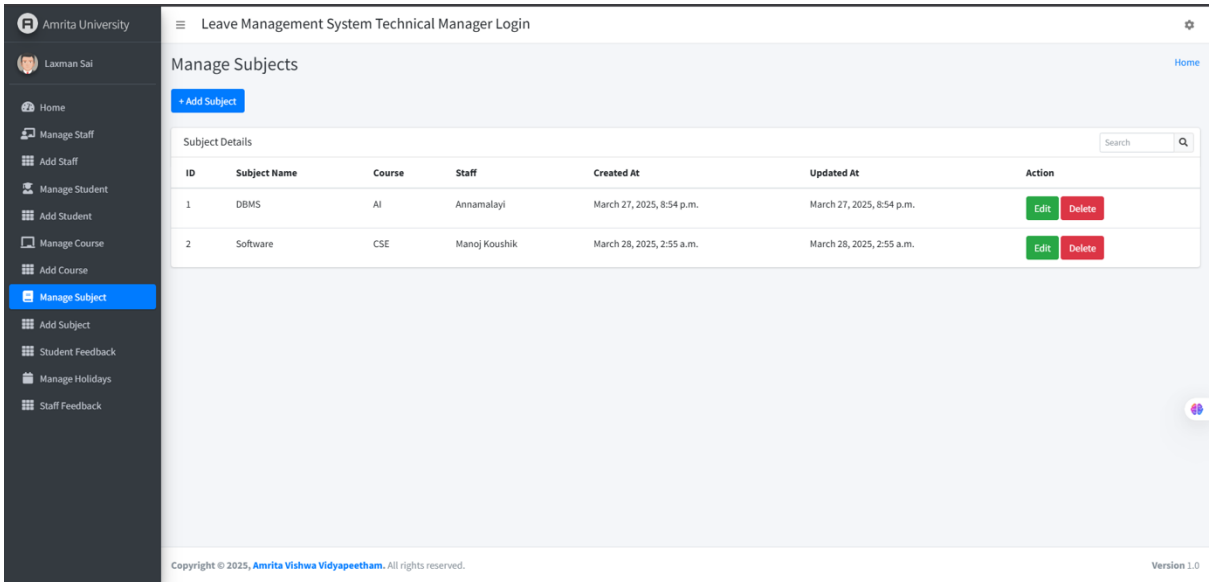
Admin can Manage Students



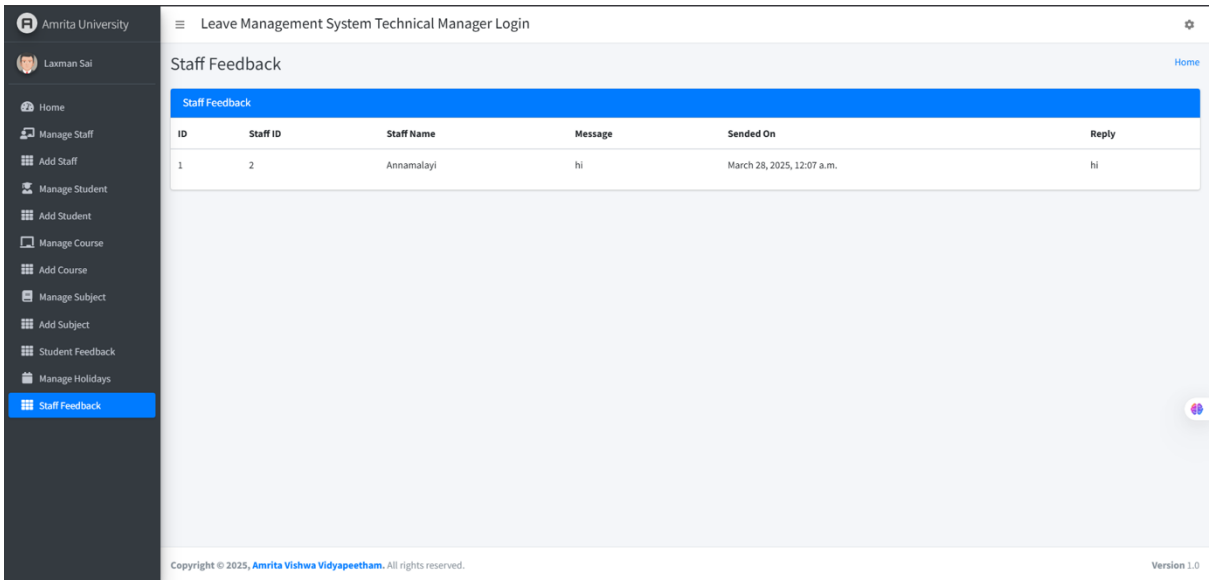
Admin can Add Course



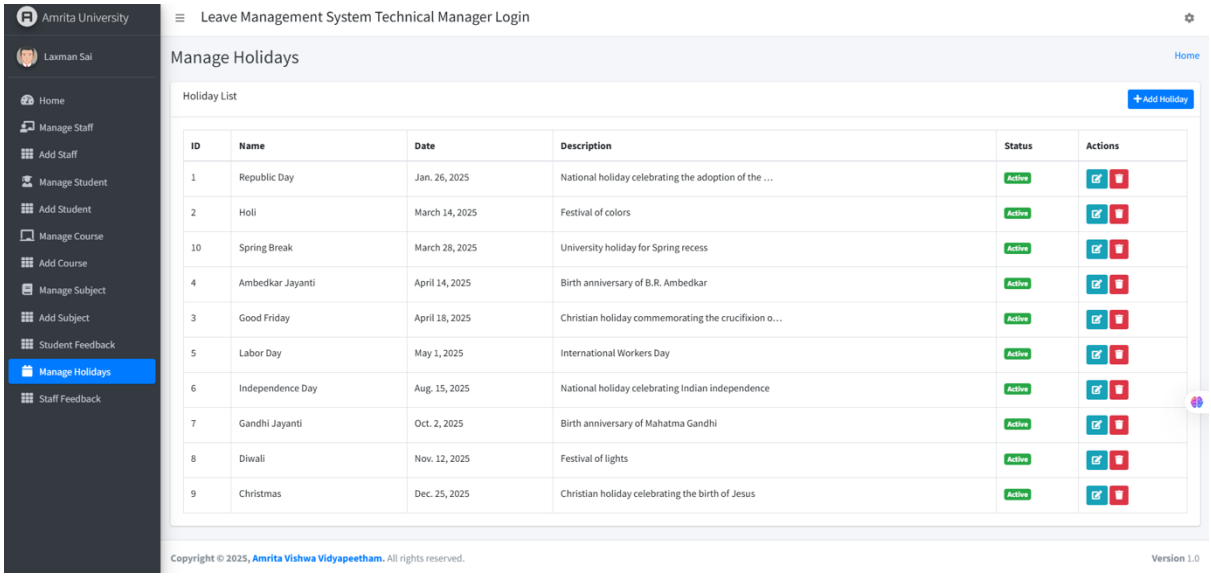
Admin can Add Subjects



Admin can Manage Subjects

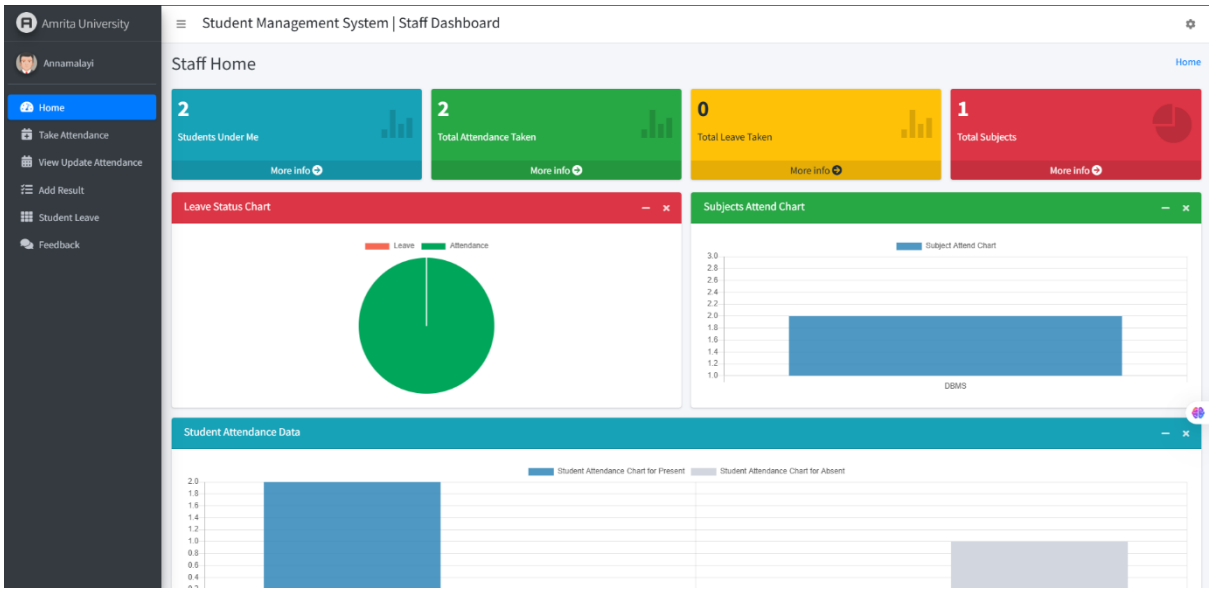


Admin can receive staff feedback

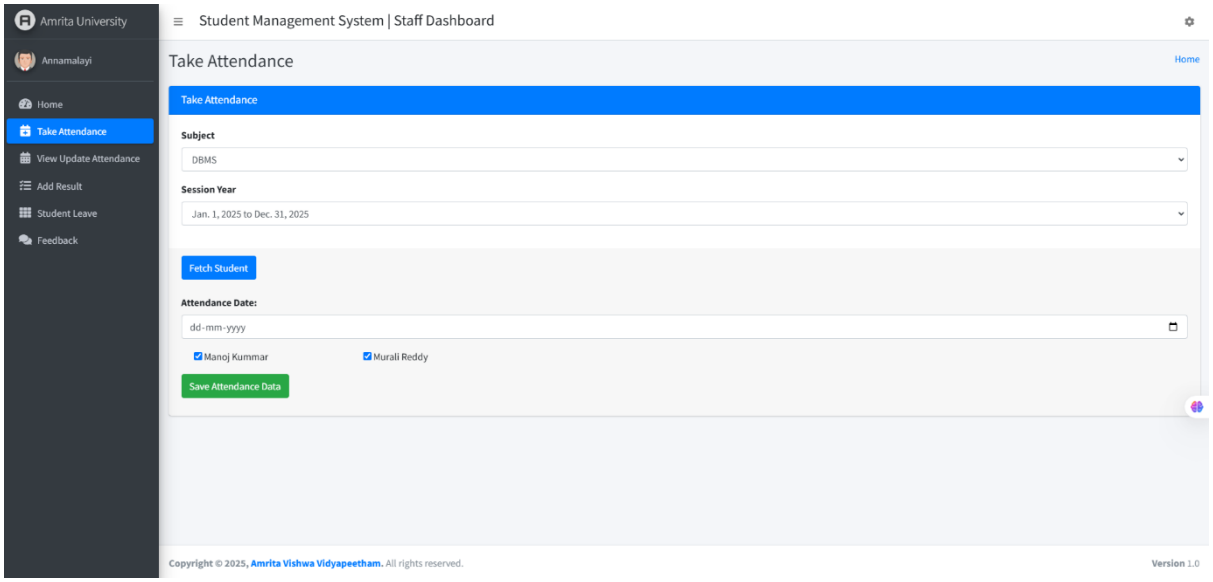


Admin can Manage Holidays

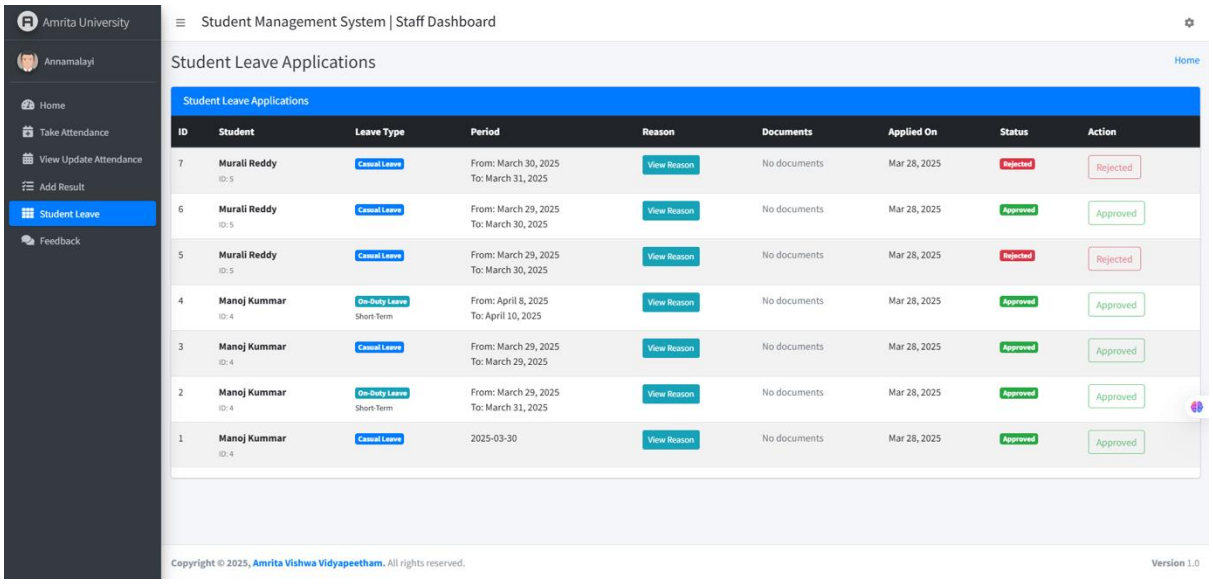
STAFF DASHBOARD



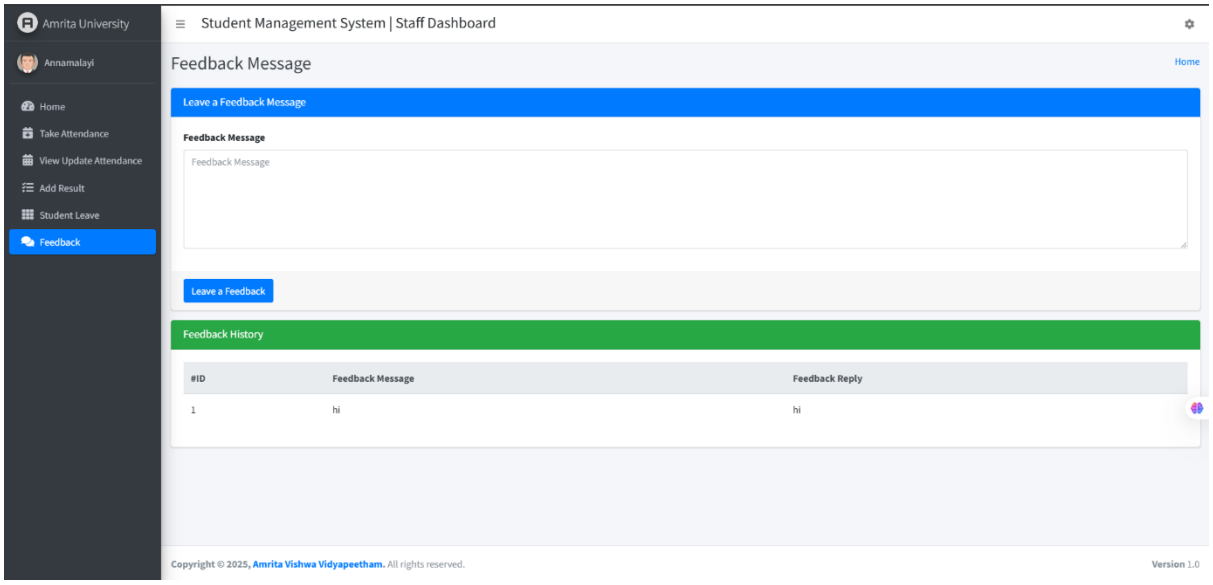
Staff can see overall things on Home Page



Staff can take Attendance

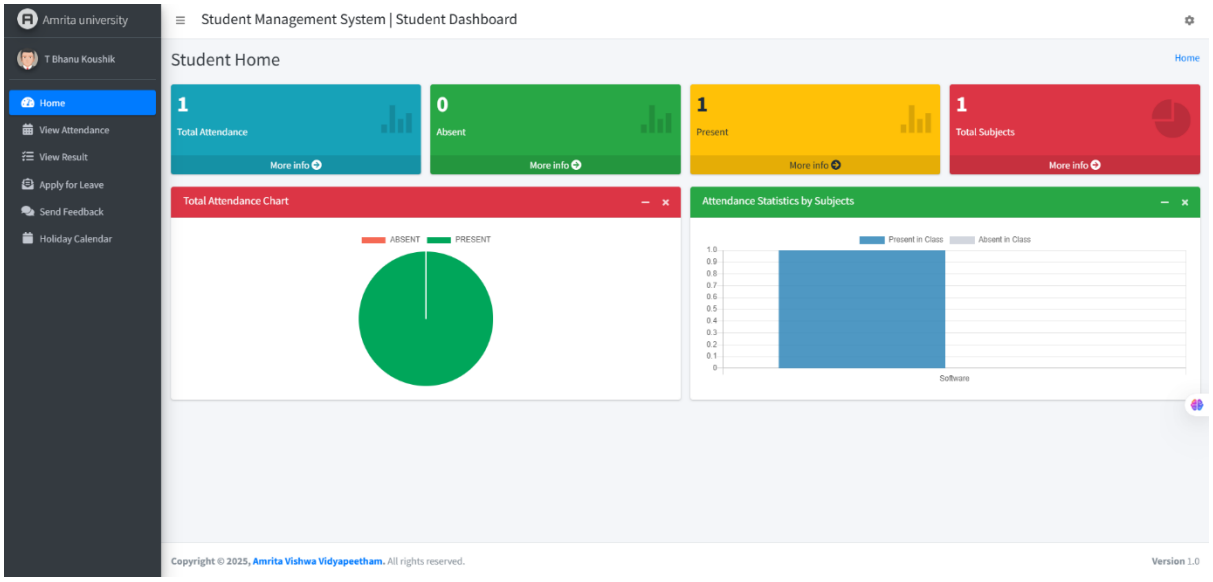


Students Leave Page (Approved / Rejected)

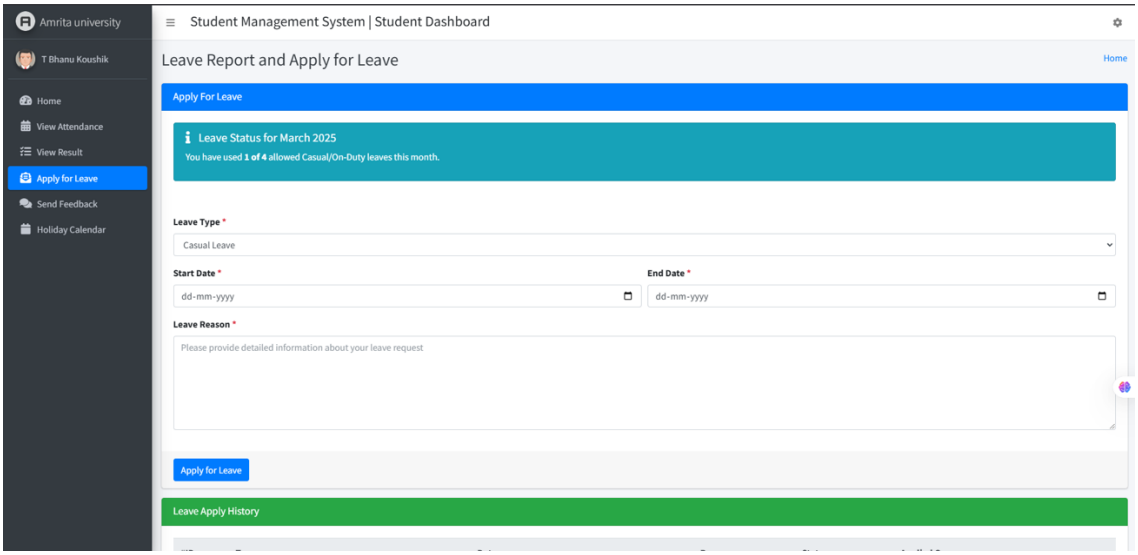


Staff Can Receives the feedback from Students

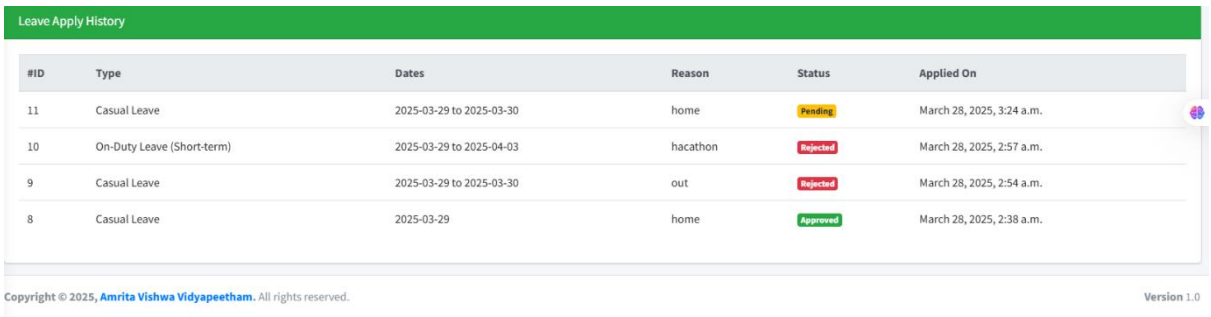
STUDENT DASHBOARD



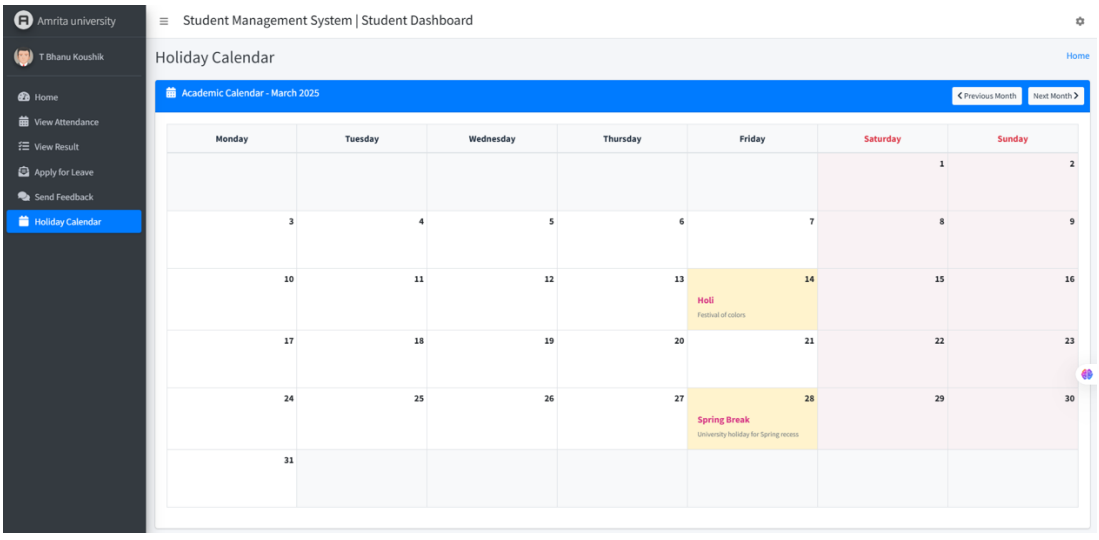
Student Home page



Apply for Leave.



Students can watch their Leave Status



Students can check the Holidays form Callender.

CONCLUSION

Both Students and the administrator (Staff) and Technical Manager can conveniently submit, review, and handle leave requests respectively with the help of the Leave Management System portal. Reduced manual work and paperwork, coupled with greater clarity and the use of automation, enables more streamlined absence management. This project clearly illustrates the integration of modern frontend and backend technologies for an easy to use and scalable solution. Productivity is improved with the use of role-based access control, real-time leave monitoring, and automated notifications which further aid in complying with company policies. Expansion of mobile application functionalities, more advanced analytics, and the use of artificial intelligence for analysing leave patterns could further enhance workforce management in the future. Overall, this leave management system meets the needs and expectations set forth and is a useful strategy for modern organizations.

FUTURE WORK

In the near future, a mobile app could be developed to the Leave Management System to ease the leave request application and management process for the students and faculty. Mobile push notification features can also be added for users to receive notifications for approved or rejected leaves and other necessary reminders. Furthermore, integrating AI-based analytics could further automate the process by performing leave approvals based on historical data, students' attendance records, and available faculty, thus minimizing administrators manual efforts.

Biometric identification or multi-factor authentication (MFA) could be added to the system to improve security and limit the access to authorized personnel only. In addition, this integration with academic portals or HR management systems would serve the purpose of monitoring the student's attendance, performance, and leave records on a single platform which would be much more convenient. By implementing smart scheduling, students could be allowed to plan their leaves proactively by selecting the most appropriate timings based on academic deadlines and workloads.

In order to guarantee record of all leave requests on a leave management system, blockchain technology could be used to reinforce traceability and credibility. In addition, a parent dashboard could be developed to enable parents to track their children's leaves and monitor the child's emergency leave application in real-time. Such improvements would strengthen the user experience while providing a more organized, safe, and efficient automation of the entire leave management process.