NEXT-GEN STUDENT MANAGEMENT SYSTEM

A Project Report Submitted in the fulfilment of the requirement for the award of degree

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND TECHNOLOGY

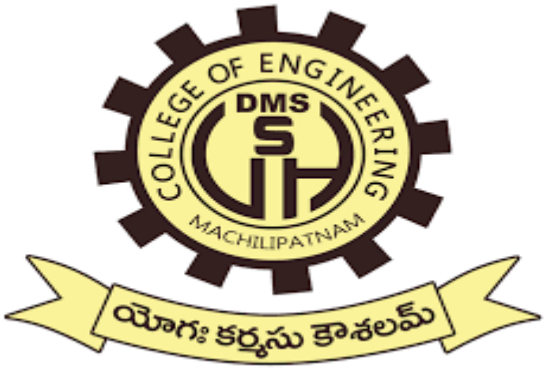
SUBMITTED BY

N. CHAITANYA RAJESWARI	**(213C1A0525)**
S. HAREESH	**(213C1A0531)**
D. MANOJ RATNA	**(213C1A0511)**
B. DURGA RAO	**(213C1A0505)**

Under the Esteemed Guidance of

Smt. N. SUSHMA

(Associate Professor, Department of CSE)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

D.M.S.S.V.H COLLEGE OF ENGINEERING

(Affiliated to Jawaharlal Nehru Technological University, Kakinada Approved by AICTE and GOVT of AP, An ISO 9001:2015 Certified Institution)

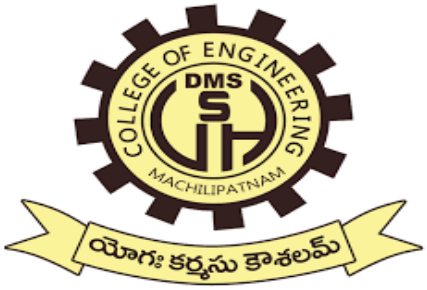
MACHILIPATNAM-521002, KRISHNA, AP

APRIL-2025

D.M.S.S.V.H COLLEGE OF ENGINEERING

(Affiliated to Jawaharlal Nehru Technological University, Kakinada Approved by AICTE and GOVT of AP, AN ISO 9001:2015 Certified Institution)

MACHILIPATNAM-521002, KRISHNA, AP



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project work entitled

NEXT-GEN STUDENT MANAGEMENT SYSTEM

Is the Bonafide work done by

N. CHAITANYA RAJESWARI	(213C1A0525)
S. HAREESH	(213C1A0531)
D. MANOJ RATNA	(213C1A0511)
B. DURGA RAO	(213C1A0505)

Submitted in the partial fulfilment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND TECHNOLOGY

2024-2025

Smt. N. SUSHMA	Smt. N. SUSHMA
Head of the Department, CSE	Project Guide

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any work would be incomplete without mentioning the people who made it possible and whose encouragement and guidance has been a source of inspiration throughout the course of the project **”NEXT-GEN STUDENT MANAGEMENT SYSTEM”** . We are thankful to sanctum **DMSSVH COLLEGE OF ENGINEERING** for giving us the opportunity to fulfil our aspirations.

We take the opportunity to express our heartfelt gratitude to **Prof. Dr. T. RAVI KUMAR, Principal**, for his kind support in doing this project.

We are privileged to express our sincere gratitude to **Head of the Department Prof. N. SUSHMA**, for giving her continuous support and guidance in our endeavours.

We express our sincere thanks to our guide **Prof. N. SUSHMA**, for being a source of inspiration and for constantly encouraging us throughout the course to pursue new goal and ideas to implement this project.

We would like to express our deepest gratitude to our internship trainer **Sri. D. Chinna Venkataswamy garu, Director, App Genesis Soft Solutions Pvt. Ltd recognized by AICTE & APSCHE.** We are especially grateful for your invaluable guidance and mentoring throughout our internship, which has significantly contributed to our professional growth. Your expertise in **"Full Stack Development with Django"** helped us to develop strong understanding of industry specific standards and boosted our confidence. Thank you for making this internship so valuable.

Finally, we express our thanks to all the teaching and supporting staff and associates of the department, Parents and also our friends for their wishes and helping hands in successful completion if this project.

Project Team

(213C1A0525)
N. Chaitanya Rajeswari

(213C1A0531)
S. Hareesh

(213C1A0511)
D. Manoj Ratna

(213C1A0505)
B. Durga Rao

LIST OF CONTENTS

ABSTRACT

The Next-Gen Student Management System (Next-Gen SMS) is an essential tool for modern educational institutions seeking to optimize their learning operations and provide students with a seamless, technology-driven academic journey. It serves as a centralized hub where students can create, update, and manage their profiles efficiently. The system provides a user-friendly interface for students to register and access personal information, enrolled courses, and learning progress etc. and institutional growth. With features that simplify tasks and improve user engagement, SMS plays a vital role in shaping a more productive and technology-driven education system.

1. INTRODUCTION

1.1 Scope

The current scope of the Student Management System includes essential features such as managing students, attendance, courses, exams, and marks. However, there are some limitations and missing functionalities. Notably, the system does not support updating student fees or handling transactions, which are crucial for complete administrative management. Additionally, the profile picture uploaded by users is not visible, indicating a possible issue with image rendering or storage. These gaps highlight the need for further enhancement to ensure a more comprehensive and user-friendly experience in managing all aspects of student records and interactions.

1.2 Purpose of the project

The primary purpose of the NEXT-GEN Student Management System (NGSMS) is to develop an efficient, secure, and user-friendly platform that automates and streamlines the management of student-related academic activities within an educational institution. This system is intended to replace traditional manual processes, which are often time-consuming, error-prone, and difficult to maintain as the number of students grows.

1.3 Keywords

- **NGSMS** – Next-Gen Student Management System
  An upgraded, web-based version of a traditional SMS using modern tech like Django and Python.
- **UML** – Unified Modeling Language
  A standardized modeling language used in software engineering to visualize system design.
- **DFD** – Data Flow Diagram
  A diagram representing how data flows within a system, showing processes, inputs, and outputs.
- **IDE** – Integrated Development Environment
  A software suite (like VS Code or PyCharm) that combines tools for writing, testing, and debugging code.

- **ORM** – Object Relational Mapper
  A tool in Django that maps Python objects to database tables, making database operations easier.
- **SQL** – Structured Query Language
  A language used for managing and querying data in relational databases.
- **API** – Application Programming Interface
  A set of rules and tools that allows different software components to communicate.
- **CRUD** – Create, Read, Update, delete
  The four basic functions of persistent storage used in databases and web applications.
- **GUI** – Graphical User Interface
  The visual part of a software application that users interact with.
- **HTML/CSS/JS** – Hypertext Markup Language, Cascading Style Sheets, JavaScript
  Core technologies for front-end web development.
- **HTTP** – Hypertext Transfer Protocol
  The protocol used for transmitting data on the web.
- **CRUD Operations** – Basic functions for interacting with a database (Create, Read, Update, Delete).
- **JWT** (mentioned in advanced systems) – JSON Web Token
  A compact, URL-safe means of representing claims to be transferred between two parties, used in authentication.
- **CRUD** – Common in web applications for handling basic database operations.

2. OBJECTIVE OF THE PROJECT

The objective of this Student Management System is to design and develop a centralized platform that streamlines and automates various academic and administrative activities related to student data management. The system aims to manage student records, track attendance, handle marks efficiently, enrol students in courses, assign tasks, and facilitate event participation. By providing a user-friendly interface for administrators, teachers, and students, the project seeks to enhance productivity, reduce manual effort, and ensure accuracy and transparency in academic workflows.

2.1 Traditional System

In the existing system, most student-related information is managed manually or using outdated software that lacks integration and automation. This leads to various challenges, such as:

- Difficulty in accessing and updating student data.
- Manual attendance and mark recording are prone to human errors.
- Lack of a centralized database for certifications, events, and tasks.
- Limited student engagement due to the absence of interactive dashboards or tracking features.
- Time-consuming administrative processes and delayed decision-making due to inefficient data handling.

Overall, the traditional system lacks scalability, data integrity, and modern user experience.

2.2 Proposed System

The proposed Student Management System is a web-based application developed using Django framework. It introduces a structured, secure, and scalable solution for managing student data. Key features include:

- Role-based login for administrators, teachers, and students.
- Profile management with support for profile pictures, bio, and skills.
- Automated attendance tracking and real-time status updates.
- Semester-wise marks management with performance analysis.
- Course enrolment based on student skills and interests.

- Daily task tracking and event participation management.
- Certification uploads and centralized document storage.
- Modern, responsive UI with administrative control over users and data.
- This system aims to minimize manual workload, improve data accuracy, and offer a smooth digital experience for educational institutions.

3. LITERATURE SURVEY

Student Management System Paper on IJERT

Authors:

Mr. Sangamesh K, Mr.Akash Samanekar, Mr.Ningappa T Pujar

In India there are many academic institutions. But very few institutions are modernized and use software to manage their day to day work. In a city like Bengaluru there are around 1000 schools, more than 300 pre-university colleges and degree colleges. Most of these academic institutions still relay on traditional way of management which mainly involves paper-work, much of human effort.

The students, who are admitted to those institutions which are dependent on traditional way of managing things, have to struggle a lot just to get a certificate or any other documents. Also the administrations face difficulty in maintaining all the records, tracking the records and fetching the record of their interest in time. The administrations of those institutions also have to employ a number of employees just to maintain the records required to manage and support their daily work.

Some of the universities like PESIT and Christ University in Bengaluru have their own web application to address the previously mentioned issues.

The web application that is being used by these and many other institutions have the following features and functionalities such as, Login/Sign Up, Dashboard, Viewing of results, attendance, courses, time table, assignments and students progress, upload/download documents and notifications.

Student Management System vol7 Issue -3 2021

Authors:

Deepak Saini, Payal, Mansi Ghadigaonkar, Prof. Sujata Kadu - Dept. of Information Technology Terna Engineering College Nerul, Navi Mumbai

Automation can be defined as the process of reducing or minimalizing the manual hard work with help of computers, computer operated software and devices. There are certain works that are beyond human capacities which can be carried out through automation techniques. Library Automation System of the University of Toronto in 1963-1972 was one of the first achievements to manage the data with the help of automated system. The real idea of implementing Automation is to enhance efficiency, reduce delays, increase production flexibility, reduce prices, human error elimination, and alleviate labour shortage, high degree of accuracy. Automation in Educational Assessment created in Nigeria shows how an online automation system can be implemented to eradicate human errors and bring fairness during the exams. Defining the Paperless Workplace with the Paper Metaphor, has explained the difficulties faced by the organization while switching from conventionally used paper-based system to an online automated system as they were not able to draw the gap between both the systems but automated Project Grading & Instant Feedback System provides an example of an automated system which enhances the efficiency of manual project grading system with feedbacks can being easily managed.

4. SYSTEM ANALYSIS

System analysis is a crucial phase in the software development lifecycle. It involves understanding the requirements, identifying problems in the current system, and analysing the potential solutions. This phase ensures that the system is well-planned, feasible, and aligned with the actual needs of stakeholders.

4.1 Preliminary Investigation

The preliminary investigation focuses on identifying the key challenges in managing student data manually or using outdated systems. It was observed that most institutions rely on spreadsheets, physical records, or fragmented software tools, resulting in:

- Inefficient data handling.
- Lack of real-time updates.
- Difficulty in accessing historical records.
- Limited user interaction and engagement.

The need for a centralized, accessible, and secure system became evident through initial research and discussions with students, faculty, and administrators.

4.2 Request Clarification

To ensure a clear understanding of the project requirements, detailed discussions were held with stakeholders. The main expectations gathered include:

- A user-friendly platform for students and staff.
- Easy attendance and marks tracking.
- Profile and course management features.
- Administrative controls and reporting options.
- Secure login and data privacy.

Clarifications also included the type of data to be stored, access levels for different users, and the need for future scalability of the system.

4.3 Feasibility Analysis

Feasibility analysis was conducted across four dimensions:

- **Technical Feasibility**: The system is technically feasible with available resources like Django (backend), React (optional frontend), MySQL/PostgreSQL database, and standard hosting services.
- **Operational Feasibility**: The system is easy to use for both students and staff. Basic training and documentation ensure smooth adoption and minimal disruption to current processes.
- **Economic Feasibility**: The development cost is low due to the use of open-source tools. The benefits in terms of time saved, reduced errors, and improved academic tracking far outweigh the initial investment.
- **Schedule Feasibility**: The project timeline is realistic, with core features achievable within the planned development cycle. Additional modules can be added in future iterations.

5. SYSTEM REQUIREMENT SPECIFICATION

5.1 Functional Requirements

For a Next-Gen Student Management System project, functional requirements include functionalities like student registration, course management, attendance tracking, grade management, and report generation, ensuring the system effectively manages student data and academic processes.

Here's a more detailed breakdown of functional requirements for a Student Management System:

1. Core Student Management:

Student Registration/Enrollment: Allowing administrators to register new students with necessary details (name, ID, contact information, etc.). Enabling students to register for courses and programs.

Student Information Management: Storing and updating student personal information. Managing student address, contact details, and other relevant data.

Course Management: Assigning students to specific courses.

Attendance Tracking: Recording and managing student attendance for each course. Generating attendance reports.

Grade Management: Entering and updating student grades for each course. Generating and displaying grade reports.

Reporting: Generating various reports, including student lists, attendance records, grade reports, and other relevant data.

2. Exam Management:

Exam Scheduling: Allowing administrators to schedule exams for different courses and programs. Assigning exams to specific students.

Mark Evaluation: Entering and managing exam marks for each student. Calculating and displaying final grades.

3. Fees Management:

Fee Structure: Managing fee structures for different programs and courses. Allowing students to view their fee dues.

Payment Processing: Enabling students to make fee payments online or offline. Tracking fee payments and generating receipts.

4. User Roles and Permissions:

Administrator: Managing users, roles, and permissions. Accessing all system functionalities.

Faculty/Teachers: Accessing and managing their assigned courses. Entering grades and attendance.

Students: Accessing their personal information and grades. Registering for courses.

5. Additional Features:

Communication: Allowing communication between students, teachers, and administrators (e.g., through announcements, emails, or a messaging system).

Library Management: Managing library resources and student access.

Event Management: Managing school events and activities.

User Authentication: Providing secure login and authentication for users.

5.2 Non-Functional Requirements

1. Performance

It can handle at least 1000 concurrent users without performance degradation.

2. Security

Passwords must be encrypted using secure hashing

3. Scalability

Increasing number of students, teachers, and courses.

4. Data Integrity

Prevent duplicate entries and invalid relationships.

5. Backup and Recovery

Enable automated backups.

5.3 System Requirements

The Student Management System is designed to run efficiently on modern systems with moderate specifications. The requirements ensure optimal performance, smooth interaction, and secure data handling. The system can be deployed on a local server for development and testing or a cloud-based server for broader accessibility.

5.3.1 Hardware Requirements

Component	Minimum Requirement
Processor	Intel Core i3 or equivalent
RAM	4 GB (8 GB recommended)
Hard Disk	100 GB of free space
Display	1024x768 resolution or higher
Internet Connection	Required for deployment and updates

5.3.2 Software Requirements

Software	Version / Details
Operating System	Windows 10+, Linux (Ubuntu), or macOS
Backend Framework	Python 3.8+ with Django Framework
Frontend (Optional)	HTML, CSS, JavaScript, React (for UI)
Database	MySQL
Web Server (for deployment)	Apache / Nginx
IDE / Editor	VS Code / PyCharm
Browser	Chrome, Firefox, or Edge (latest versions)
Other Tools	Git, Postman, XAMPP (if using locally)

6. SYSTEM DESIGN

6.1 Data Flow Diagram

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.

2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

6.2 UML Diagrams

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. The Unified Modeling Language is a standard language for specifying, Visualization, constructing and documenting the artifacts of software system, as well as for business modeling and other non- software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development

process. The UML uses mostly graphical notations to express the design of software projects.

GOALS

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to use, expressive visual modeling Languages so that they can develop and exchange meaningful models.

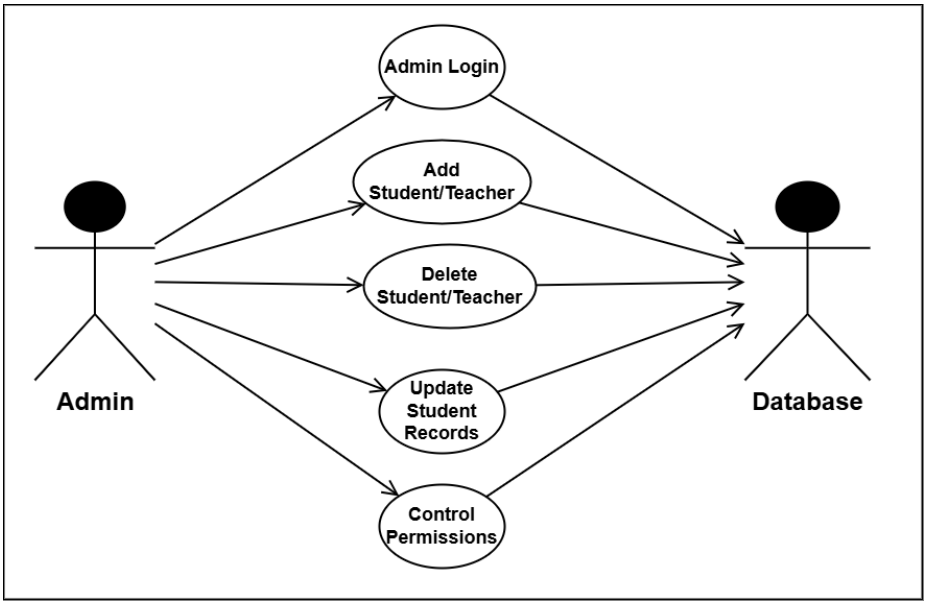
2. Provide extendibility and specialization mechanisms to extend the core concepts.

3. Be independent of particular pro

6.2.1 Use Case Diagram

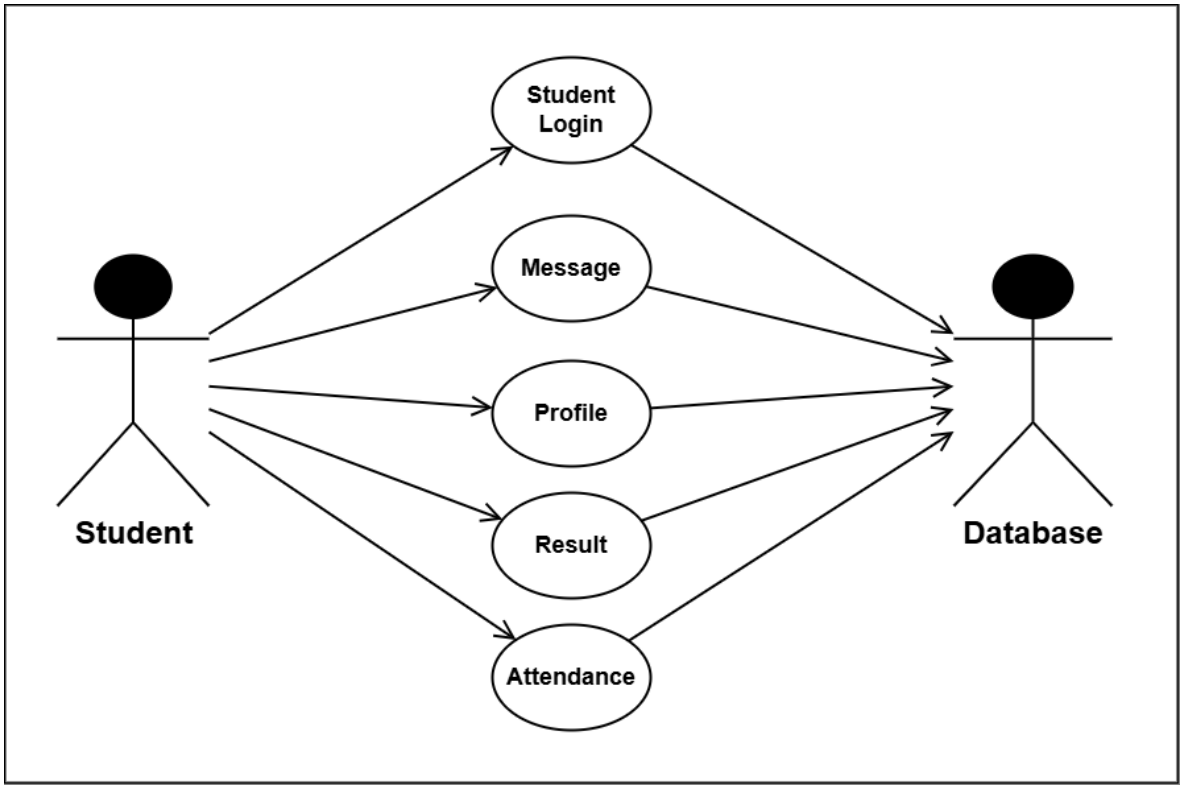
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

For Admin:



This use case diagram illustrates the functionalities available to the Admin in the Student Management System. The admin can perform actions such as logging in, adding or deleting students and teachers, updating student records, and controlling user permissions. Each of these use cases is directly connected to the system's database, indicating that the Admin's activities involve data manipulation and management to maintain the integrity and structure of the system.

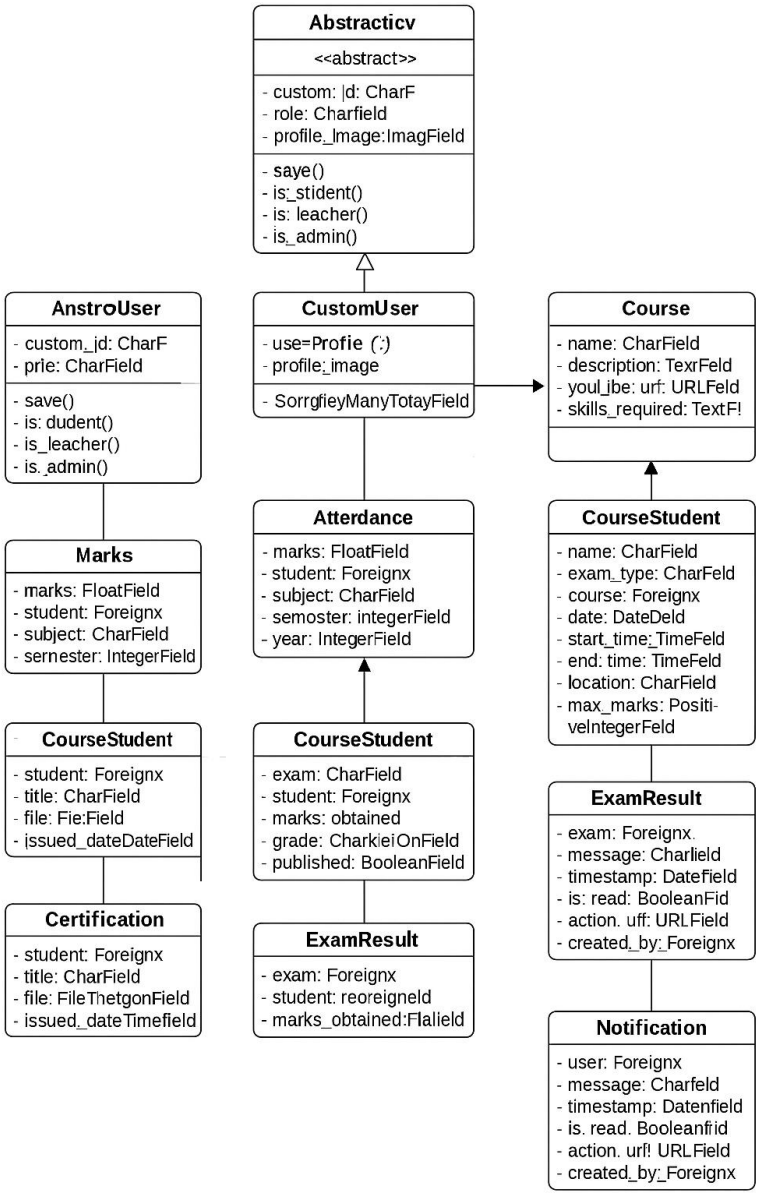
For Student:



The second use case diagram focuses on the student role. Students can log into the system, send or receive messages, view and update their profile, check academic results, and track their attendance. Similar to the admin use cases, all student interactions are linked to the database, emphasizing that these activities involve data retrieval and updates. This diagram highlights the essential features that support a student's academic journey within the system.

6.2.1 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



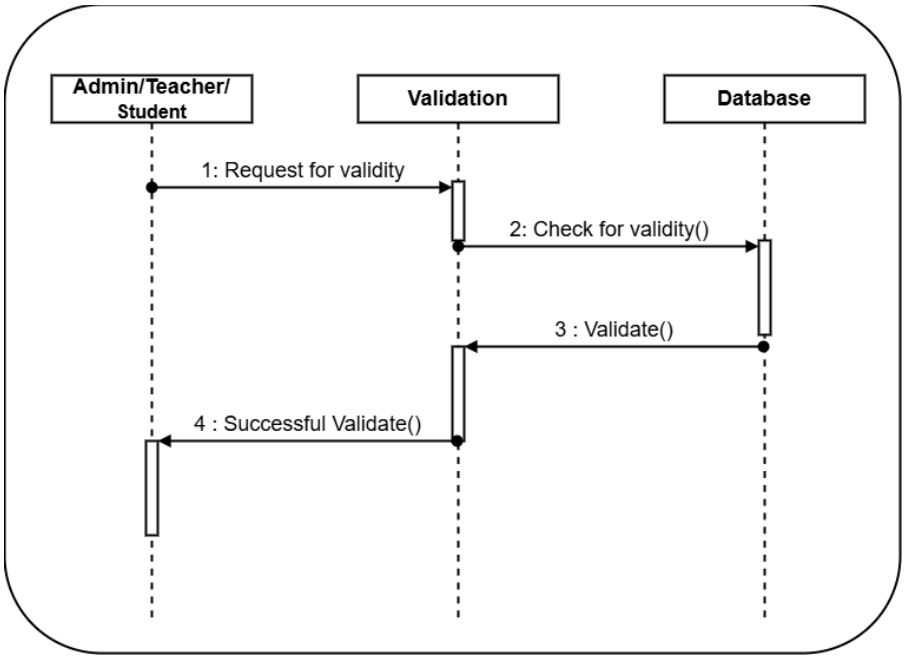
This class diagram represents a student-course management system with user roles, course management, and academic tracking. At its core, the

Abstractcv abstract class defines common attributes (like ID, role, profile image) and role-checking methods. CustomUser inherits from it and associates with user profiles. AnstroUser is another user class with similar role-checking methods. The system tracks Course details, Attendance, Marks, ExamResult, and Notification. CourseStudent handles course participation, grading, and certification, while Certification records issued course certificates. However, the diagram contains several typos (e.g., “CharF”, “Foreignx”, “SorrgifyManyTotayField”) and repeated or inconsistent class names like CourseStudent, which may indicate design or documentation issues.

6.2.2 Sequence Diagram

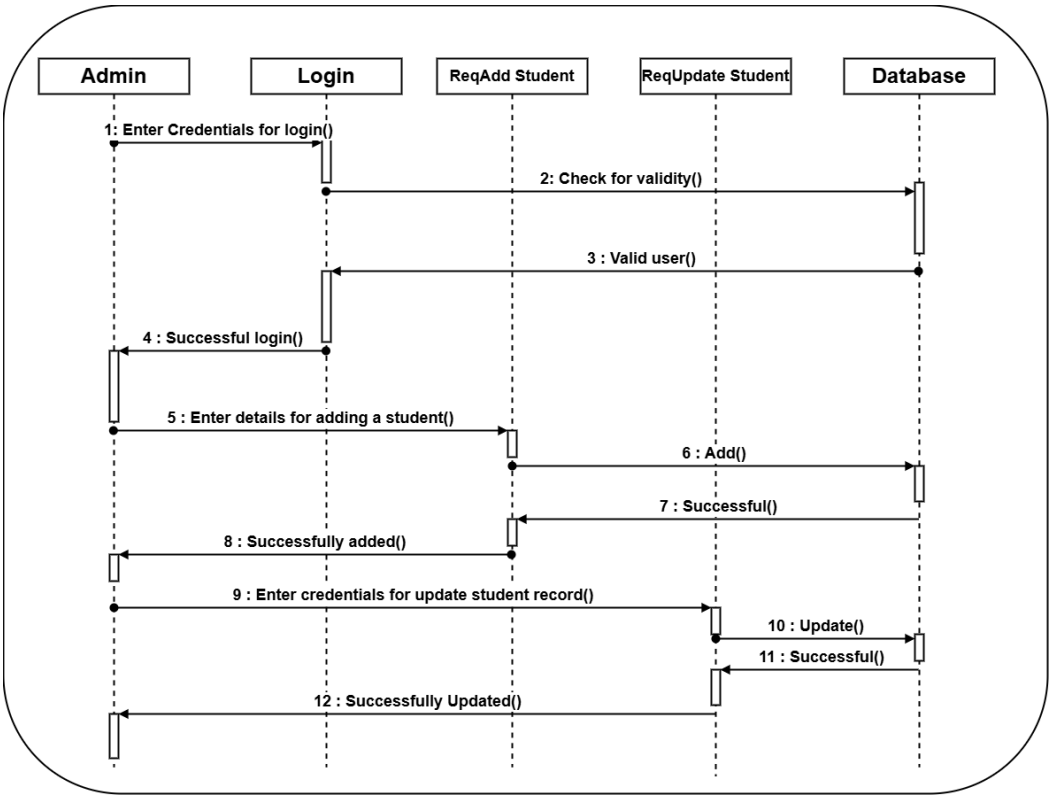
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

For validity:



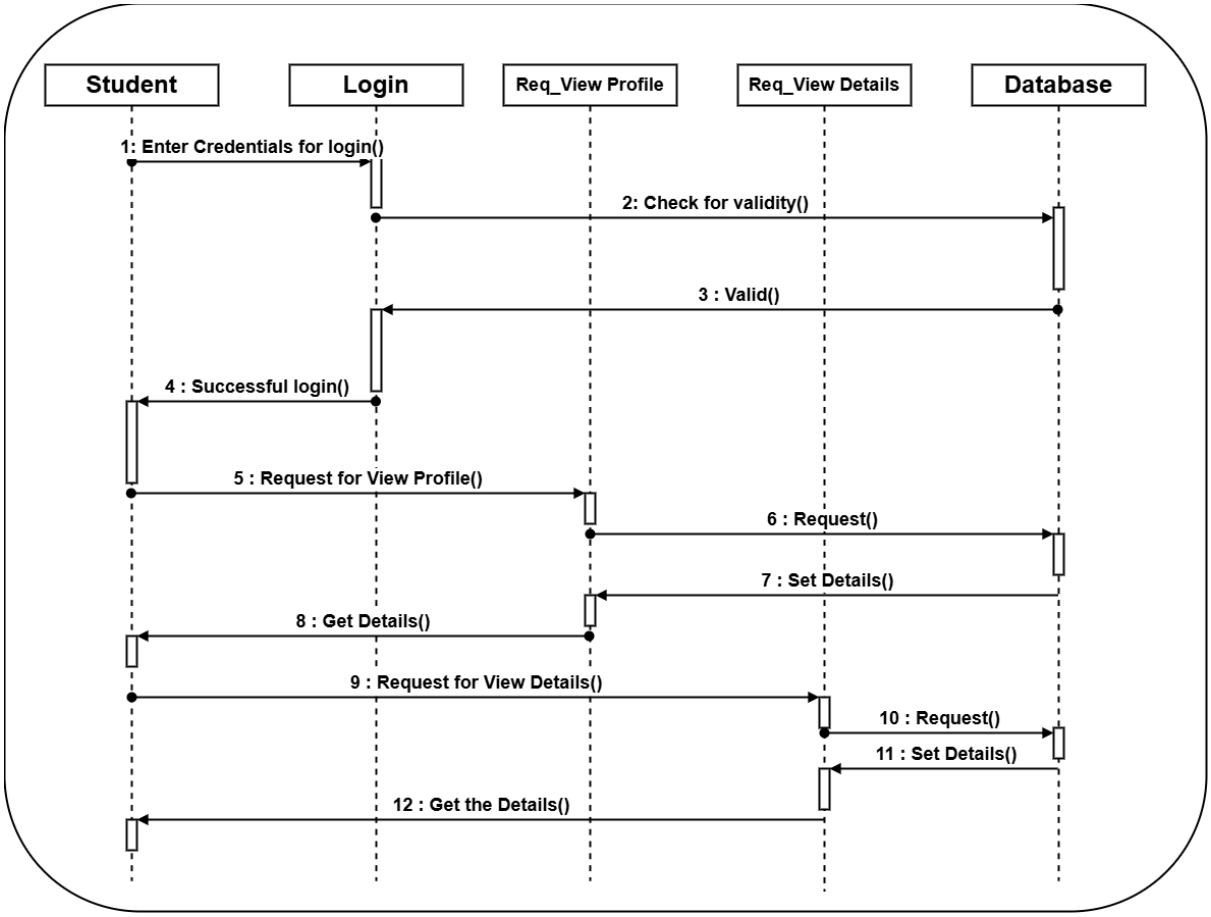
This sequence diagram illustrates the validation process initiated by a user (Admin, Teacher, or Student). It starts when the user sends a request for validity to the Validation system. The Validation component then interacts with the Database to check for validity, which involves data lookup or verification. Once the database processes this and performs the validate() operation, the result is returned to the Validation component. Finally, a successful validation response is sent back to the user, confirming that the request has passed the necessary checks. The flow ensures data integrity and access control across the system.

For Admin:



This sequence diagram represents the admin workflow for managing student records. The process begins with the admin entering login credentials, which are validated through the Login and Database components. Upon a successful login, the admin proceeds to add a student by submitting details, which are processed via the ReqAddStudent module and stored in the Database. A confirmation of successful addition is returned. Later, the admin can update student records by providing the necessary credentials to the ReqUpdateStudent module, which sends the update request to the database. A successful update response completes the flow. This diagram outlines a secure and structured interaction between the admin, validation modules, and database.

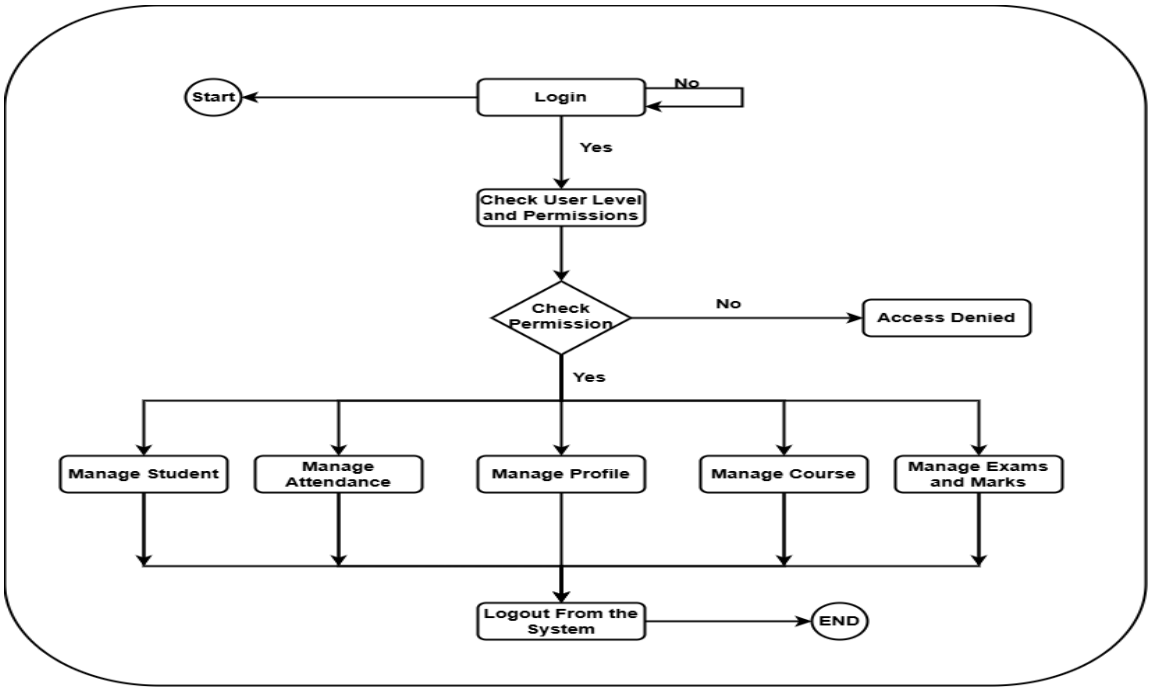
For Student:



This sequence diagram illustrates a student's interaction with the system to log in and view their profile and academic details. The student begins by entering login credentials, which are validated via the Login module and the Database. Upon successful login, the student sends a profile view request, which triggers a data fetch operation from the database, and the profile details are returned. Next, the student sends a request to view more detailed academic information, which again involves retrieving the relevant data from the database. Finally, the student receives the requested details, completing the interaction. This flow ensures secure access and efficient retrieval of student-related information.

6.2.3 Activity Diagram

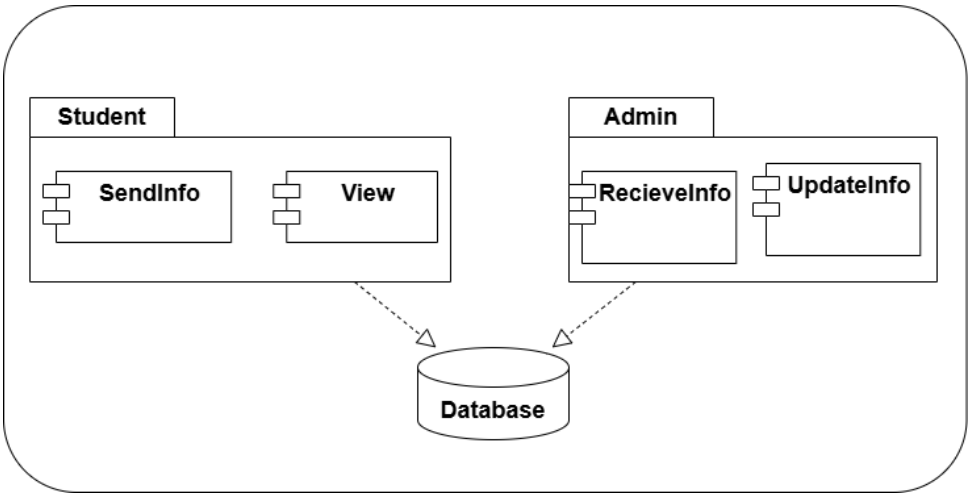
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



This activity diagram outlines the workflow of a user accessing and managing various functionalities in a system. The process starts with a Login, followed by checking the user's level and permissions. If the user lacks proper permissions, access is denied. If granted, the user can perform various management tasks such as Managing Students, Attendance, Profile, Courses, and Exams and Marks. Once tasks are completed, the user logs out, ending the session. This flow ensures that only authorized users can access and manage specific parts of the system, maintaining secure and organized operations

6.2.4Component Diagram

A component diagram is used to breakdown a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides with in the node.



This component diagram depicts the interaction between Student, Admin, and the Database through specific modules. The Student component includes two functions: SendInfo, which allows students to submit data, and View, which lets them retrieve or view information from the database. The Admin component features ReceiveInfo to access student-submitted data and UpdateInfo to modify or manage records. Both components interact with the shared Database, ensuring centralized storage and consistent data access. This setup highlights a structured and modular approach to handling information flow within an academic management system.

7. TECHNOLOGIES

This section discusses the technologies used in the development of the Student Management System, including both frontend and backend components, the database used, and the development and testing tools that supported the implementation process.

7.1 Frontend

The frontend is the user-facing part of the application, which interacts with users and collects data input.

7.1.1 Django template Language

The Django Template Language (DTL) is used to create dynamic HTML pages. It allows embedding Python-like expressions within HTML using special tags ({{ }} for variables, {% %} for logic). It ensures clear separation of logic and presentation, improving maintainability and reusability.

7.2 Backend

The backend processes user requests, performs operations, and interacts with the database.

7.2.1 About Django

Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. It comes with built-in features such as authentication, admin panel, and ORM (Object Relational Mapper) for database management, which makes development easier and more secure.

7.2.2 About Python

Python is a powerful, easy-to-read, high-level programming language known for its simplicity and versatility. It supports object-oriented and functional programming and has an extensive ecosystem of libraries and frameworks, making it ideal for web development.

7.3 Database

The database is the backbone of the application, storing and managing all the system data efficiently.

7.3.1 About SQL

SQL (Structured Query Language) is used to interact with relational databases. In this project, SQL is used with **SQLite/MySQL/PostgreSQL** (you can mention the one you're using) to perform operations like data insertion, updates, and retrieval. Django's ORM translates Python objects to SQL queries, abstracting complex SQL operations.

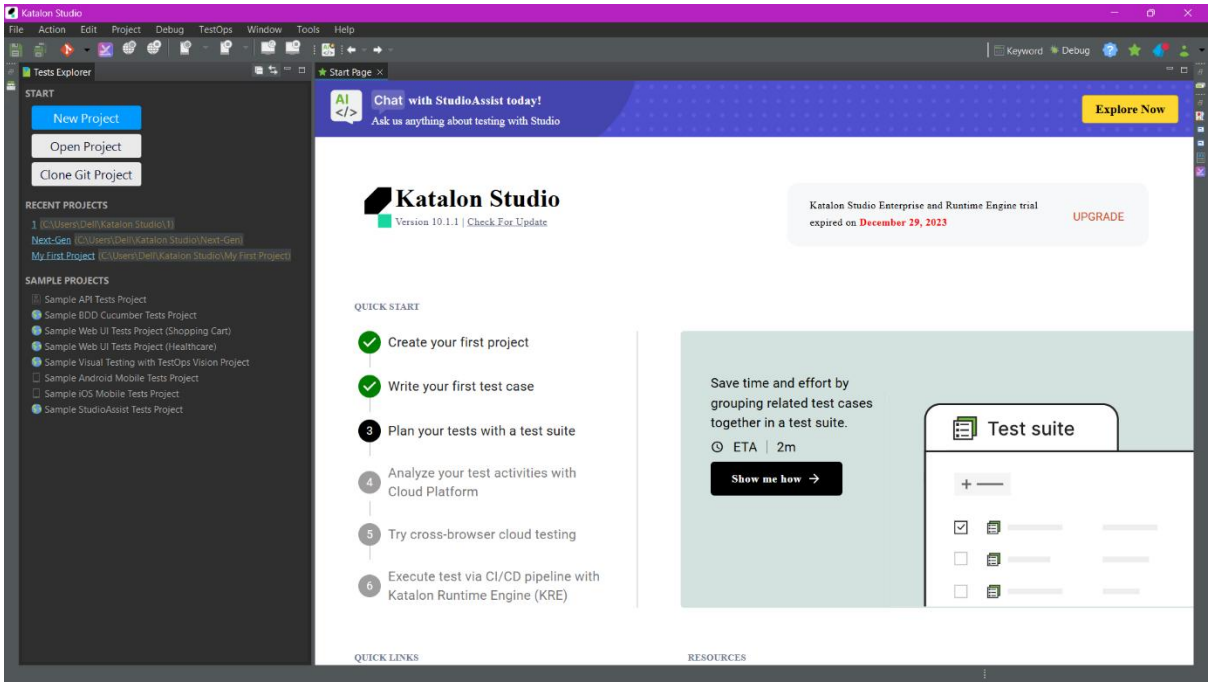
7.4 Development Tools

- **Visual Studio Code**: A powerful and lightweight code editor with support for Python, Django, and frontend development.
- **Git:** A Version control system to track changes and collaborate efficiently.
- **GitHub/GitLab**: For code repository and project collaboration.

7.5 Testing tools

Katalon :

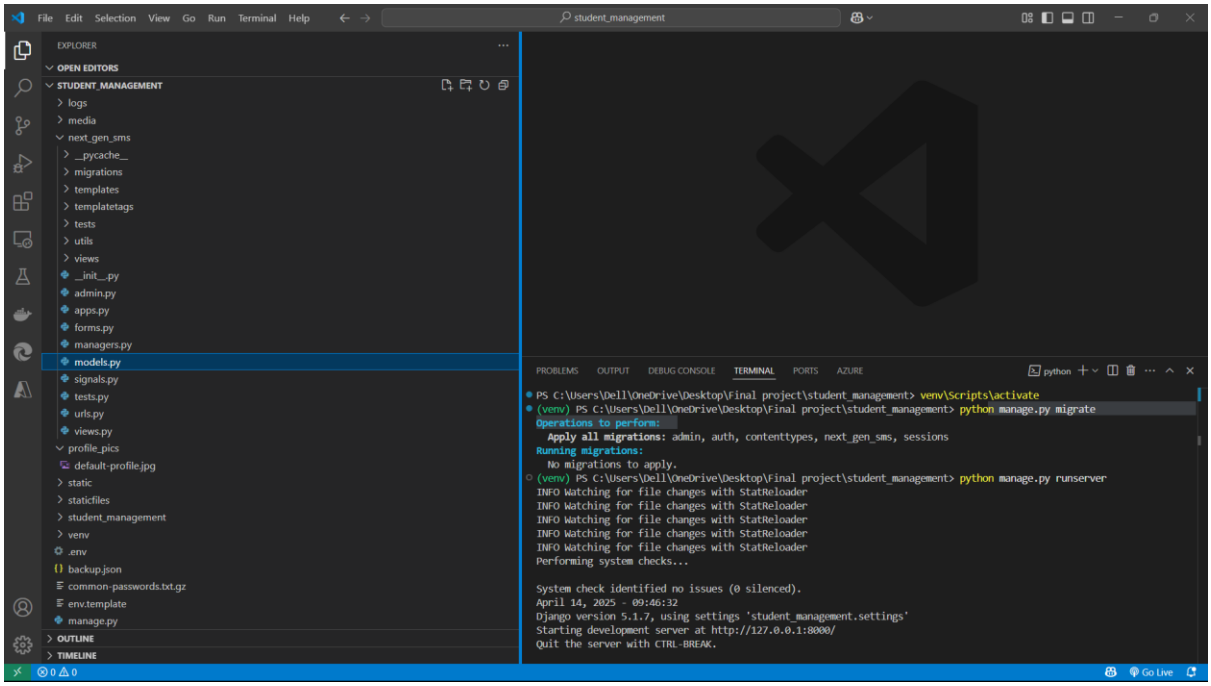
Katalon Studio is an automation testing IDE built on the Selenium framework, designed to make test automation easier for users of all skill levels. It allows you to create and execute tests across multiple application types, including web, mobile, API, and desktop applications, from a single project. Katalon Studio offers both no-code and full-code testing options, along with AI-powered features to enhance the testing process.



Katalon Interface

8. IMPLEMENTATION

Project structure Screenshot:



admin.py

(Registers models in admin)

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import (
    CustomUser,
    UserProfile,
    Attendance,
    Marks,
    Course,
    CourseStudent,
    DailyTask,
    Event,
    Certification,
```

```
    CoursePlaylist,
    Notification,
    Exam,
    ExamResult
)
@admin.register(CustomUser)
class CustomUserAdmin(UserAdmin):
    model = CustomUser
    list_display = ['username', 'email', 'role', 'custom_id', 'is_active', 'is_staff']
    search_fields = ['username', 'email', 'custom_id']
    ordering = ['username']
    fieldsets = UserAdmin.fieldsets + (
        (None, {'fields': ('role', 'profile_image', 'custom_id')}),  # Removed 'email'
    )
    add_fieldsets = UserAdmin.add_fieldsets + (
        (None, {'fields': ('email', 'role', 'profile_image', 'custom_id')}),  # Keep 'email' here, it's
fine for user creation
    )
@admin.register(Notification)
class NotificationAdmin(admin.ModelAdmin):
    list_display = ('user', 'message', 'is_read', 'timestamp', 'created_by')
    list_filter = ('is_read', 'timestamp')
    search_fields = ('user__username', 'message', 'created_by__username')
    readonly_fields = ('timestamp',)
# Register remaining models with basic admin interface
admin.site.register(Attendance)
admin.site.register(Marks)
admin.site.register(Course)
admin.site.register(CourseStudent)
admin.site.register(DailyTask)
admin.site.register(Event)
admin.site.register(Certification)
admin.site.register(CoursePlaylist)
```

models.py

(Contains your database design)

```
# Custom User Model
class CustomUser(AbstractUser, PermissionsMixin):
    ROLE_CHOICES = (
        ('admin', 'Admin'),
        ('student', 'Student'),
        ('teacher', 'Teacher'),
    )

    custom_id = models.CharField(max_length=15, unique=True, blank=True, null=True)
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default='student')
    profile_image = models.ImageField(upload_to='profile_pics/', default='default.jpg',
blank=True, null=True)
    USERNAME_FIELD = 'username'
    email = models.EmailField(unique=True, null=False, blank=False)
    REQUIRED_FIELDS = ['email']

    objects = CustomUserManager()

    def __str__(self):
        return f"{self.username} ({self.custom_id or self.id})"

    def save(self, *args, **kwargs):
        if not self.custom_id:
            self.custom_id = str(uuid4())[:10]
        super().save(*args, **kwargs)

    def is_student(self):
        return self.role == 'student'

    def is_teacher(self):
```

```
        return self.role == 'teacher'
    def is_admin(self):
        return self.role == 'admin'
# Attendance
class Attendance(models.Model):
    student = models.ForeignKey(CustomUser, on_delete=models.CASCADE,
limit_choices_to={'role': 'student'})
    teacher = models.ForeignKey(CustomUser, on_delete=models.CASCADE,
limit_choices_to={'role': 'teacher'}, related_name='attendance_records', null=True,
blank=True)
    date = models.DateField(auto_now_add=True)
    status = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.student.username} - {self.date} - {'Present' if self.status else 'Absent'}"

    class Meta:
        ordering = ['-date']
        verbose_name_plural = 'Attendance Records'
```

forms.py

(Handles user input forms)

```
class AttendanceForm(forms.ModelForm):
    """ Form for managing student attendance. Restricted to instructors. """
    def __init__(self, *args, **kwargs):
        self.user = kwargs.pop('user', None)
        super().__init__(*args, **kwargs)
        if self.user:
            restrict_form_access(self.user, ['teacher'])
            # Only show students in the student dropdown
            self.fields['student'].queryset = CustomUser.objects.filter(role='student')
```

```
    def save(self, commit=True):
        instance = super().save(commit=False)
        instance.teacher = self.user
        if commit:
            instance.save()
        return instance

    class Meta:
        model = Attendance
        fields = ['student', 'status']
        widgets = {
            'status': forms.CheckboxInput(attrs={'class': 'form-check-input'})
        }

# ---- Certification Form ----

class CertificationForm(forms.ModelForm):
    """ Form for uploading certifications. Restricted to students. """
    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
        super().__init__(*args, **kwargs)
        if user:
            restrict_form_access(user, ['student'])

    class Meta:
        model = Certification
        fields = ['title', 'file', 'issued_date']
        widgets = {
            'issued_date': forms.DateInput(attrs={'type': 'date'}),
        }
class MarksForm(forms.ModelForm):
    """ Form for managing student marks. Restricted to instructors. """
    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
```

```
        super().__init__(*args, **kwargs)
        if user:
            restrict_form_access(user, ['instructor'])

    class Meta:
        model = Marks
        fields = ['student', 'subject', 'semester', 'year', 'marks']

class EventForm(forms.ModelForm):
    """ Form for creating or updating an event. Restricted to admins and instructors. """
    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
        super().__init__(*args, **kwargs)
        if user:
            restrict_form_access(user, ['admin', 'instructor'])

    class Meta:
        model = Event
        fields = ['title', 'description', 'date', 'time', 'location']
        widgets = {
            'date': forms.DateInput(attrs={'type': 'date'}),
            'time': forms.TimeInput(attrs={'type': 'time'}),
        }
class CourseEnrollForm(forms.ModelForm):
    """ Form for enrolling in a course. """
    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
        super().__init__(*args, **kwargs)
        if user:
            restrict_form_access(user, ['student'])

    class Meta:
        model = CourseStudent
        fields = ['course']
```

signals.py

(Handles model event triggers.)

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.conf import settings
from .models import UserProfile

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_or_update_user_profile(sender, instance, created, **kwargs):
    """Ensure every user has a profile."""
    if created or not hasattr(instance, 'profile'):
        UserProfile.objects.get_or_create(user=instance)
    else:
        instance.profile.save()
```

urls.py

(Maps URLs to views)

```
from django.urls import path, include
from .views import notifications_view, mark_notification_read
from django.conf import settings
from django.conf.urls.static import static
from . import views
from .views import (
    custom_logout,
    delete_task,
    events_list,
    add_event,
    edit_event,
    upload_certification,
)
from next_gen_sms.views import (
```

```
    marks_update,
    marks_delete,
    marks_list,
    marks_detail,
)
from next_gen_sms.views import ExamManagementView

app_name = "next_gen_sms"

urlpatterns = [
    # 🔐 Authentication (Login, Logout, Password Reset)
    path("accounts/", include("django.contrib.auth.urls")),

    # 🏠 Dashboard & Profile
    path("", views.dashboard, name="dashboard"),
    path("profile/", views.profile, name="profile"),
    path("profile/update/", views.update_profile, name="update_profile"),
    path("profile/update-section/<str:section>/", views.update_profile_section,
name="update_profile_section"),

    # 📘 Courses
    path('courses/', views.manage_courses, name='manage_courses'),
    path('courses/list/', views.student_courses, name='student_courses'),
    path('courses/add/', views.add_course, name='add_course'),
    path('courses/enroll/<int:course_id>/', views.enroll_course, name='enroll_course'),
    path('courses/<int:course_id>/', views.course_detail, name='course_detail'),
    path('courses/edit/<int:course_id>/', views.edit_course, name='edit_course'),
    path('courses/delete/<int:course_id>/', views.delete_course, name='delete_course'),

    # 📋 Tasks
    path("tasks/", views.manage_tasks, name="manage_tasks"),
    path("tasks/toggle/<int:task_id>/", views.toggle_task, name="toggle_task"),
    path("tasks/delete/<int:task_id>/", delete_task, name="delete_task"),
```

```
    # 📝 Marks & Attendance
    path("marks/", views.marks_overview, name="marks_overview"),
    path("marks/create/", views.marks_create, name="marks_create"),
    path("marks/<int:pk>/", marks_detail, name="marks_detail"),
    path("marks/<int:marks_id>/update/", marks_update, name="marks_update"),
    path("marks/<int:marks_id>/delete/", marks_delete, name="marks_delete"),
    path("marks/list/", marks_list, name="marks_list"),
    path("attendance/", views.attendance_overview, name="attendance_overview"),
    path("attendance/create/", views.attendance_create, name="attendance_create"),
    path("attendance/<int:pk>/", views.attendance_detail, name="attendance_detail"),
    path("attendance/<int:pk>/edit/", views.attendance_edit, name="attendance_edit"),

    # 🏆 Certifications
    path("certifications/", views.certifications_view, name="certifications"),
    path("upload_certification/", upload_certification, name="upload_certification"),
    path("certifications/delete/<int:cert_id>/", views.delete_certification,
name="delete_certification"),

    # 👮 Role-Based Dashboards
    path("admin-dashboard/", views.admin_dashboard, name="admin_dashboard"),
    path("student/dashboard/", views.student_dashboard, name="student_dashboard"),
    path("teacher/dashboard/", views.teacher_dashboard, name="teacher_dashboard"),
    path("student/dashboard/api/", views.student_dashboard_api,
name="student_dashboard_api"),
    path("admin/dashboard/api/", views.admin_dashboard_api,
name="admin_dashboard_api"),

    # 🗓️ Events
    path("events/", views.events_list, name="events_list"),
    path("events/add/", views.add_event, name="add_event"),
    path("events/edit/<int:event_id>/", views.edit_event, name="edit_event"),
    path("events/delete/<int:event_id>/", views.delete_event, name="delete_event"),

    # ⚙️ Settings & Logout
```

```
    path("settings/", views.settings_view, name="settings"),
    path("performance/", views.performance_dashboard, name="performance_dashboard"),
    path("delete_account/", views.delete_account, name="delete_account"),
    path("logout/", custom_logout, name="logout"),
    path('manage-users/', views.manage_users, name='manage_users'),

    # 🔔 Notifications
    path('notifications/', views.notifications_view, name='notifications'),
    path('notifications/<int:notification_id>/mark-read/', views.mark_notification_read,
name='mark_notification_read'),

    # 🏫 Exam Management
    path('exams/', ExamManagementView.as_view(), name='exam_management'),
    path('exams/<int:exam_id>/edit/', ExamManagementView.as_view(), name='exam_edit'),
    path('exams/<int:exam_id>/delete/', ExamManagementView.as_view(),
name='exam_delete'),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Views.py

(Handles the logic)

```
@login_required
@user_passes_test(is_teacher)
def attendance_create(request):
    """View for creating new attendance records"""
    print(f"User authenticated: {request.user.is_authenticated}")  # Debug
    print(f"User role: {request.user.role}")  # Debug

    if request.method == 'POST':
        form = AttendanceForm(request.POST)
        if form.is_valid():
            try:
```

```
                attendance = form.save(commit=False)
                attendance.teacher = request.user
                attendance.save()
                messages.success(request, 'Attendance record created successfully!')
                return redirect('next_gen_sms:attendance_overview')
            except Exception as e:
                print(f"Error saving attendance: {str(e)}")  # Debug
                messages.error(request, f'Error creating attendance: {str(e)}')
                return redirect('next_gen_sms:attendance_create')
    else:
        form = AttendanceForm()

    return render(request, 'next_gen_sms/attendance_form.html', {
        'form': form,
        'title': 'Create Attendance Record'
    })

@login_required
def attendance_detail(request, pk):
    """View for showing attendance record details"""
    record = get_object_or_404(Attendance, pk=pk)
    return render(request, 'next_gen_sms/attendance_detail.html', {
        'record': record
    })

@login_required
@user_passes_test(is_teacher)
def attendance_edit(request, pk):
    """View for editing attendance records"""
    record = get_object_or_404(Attendance, pk=pk)
    if request.method == 'POST':
        form = AttendanceForm(request.POST, instance=record)
        if form.is_valid():
            form.save()
```

```
            messages.success(request, 'Attendance record updated successfully!')
            return redirect('next_gen_sms:attendance_detail', pk=record.pk)
    else:
        form = AttendanceForm(instance=record)

    return render(request, 'next_gen_sms/attendance_form.html', {
        'form': form,
        'title': 'Edit Attendance Record',
        'record': record
    })

# ======================
# Marks Views
# ======================
@login_required
@user_passes_test(lambda u: is_teacher(u) or is_admin(u))
def marks_create(request):
    """
    Teacher/Admin view for creating new marks entries.

    Args:
        request: HttpRequest object

    Returns:
        Rendered marks creation template with form on GET
        Redirect to marks overview on successful POST

    Raises:
        PermissionDenied if user is not a teacher or admin
    """
    # Get students based on user role
    if request.user.is_admin():
        students = CustomUser.objects.filter(role='student').select_related('profile')
    else:  # Teacher
```

```
        student_ids = Marks.objects.filter(teacher=request.user).values_list('student',
flat=True).distinct()
        students = CustomUser.objects.filter(id__in=student_ids,
role='student').select_related('profile')


    if request.method == 'POST':
        form = MarksForm(request.POST)
        if form.is_valid():
            marks = form.save(commit=False)
            marks.teacher = request.user
            marks.save()
            return redirect('next_gen_sms:marks_overview')
    else:
        form = MarksForm()
        form.fields['student'].queryset = students


    return render(request, 'marks/marks_create.html', {
        'form': form,
        'students': students
    })
```

9. TESTING AND TESTCASES

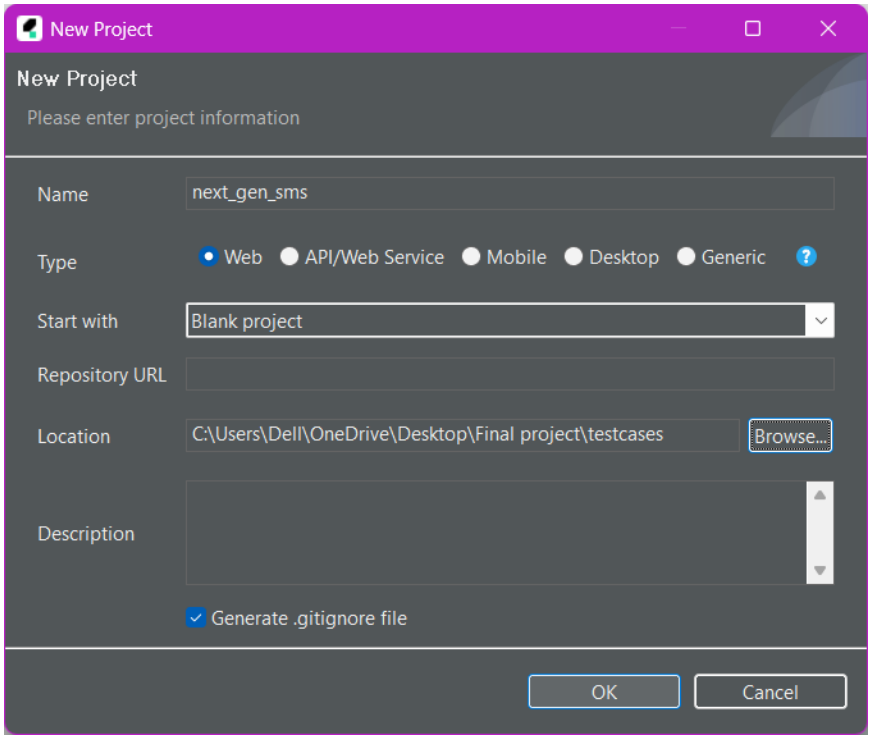
9.1 SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

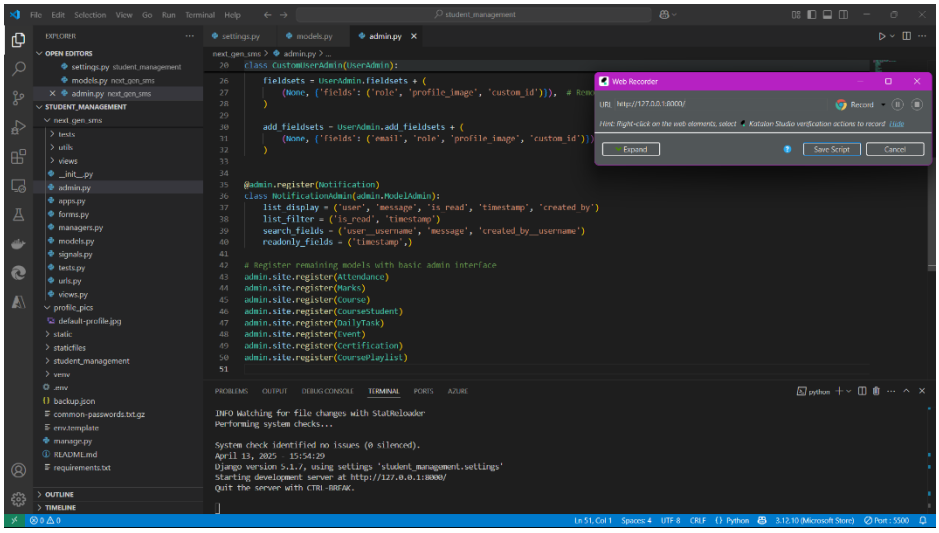
9.2 Katalon Testing

We used Katalon for testing the functionality and performance of the system. Below are the test cases that were executed to ensure the reliability and accuracy of the application. These test cases cover a range of scenarios, from user authentication to data handling, to validate each feature thoroughly. The results from Katalon provided insights into the system's behaviour under different conditions, helping us to identify and resolve potential issues before deployment.

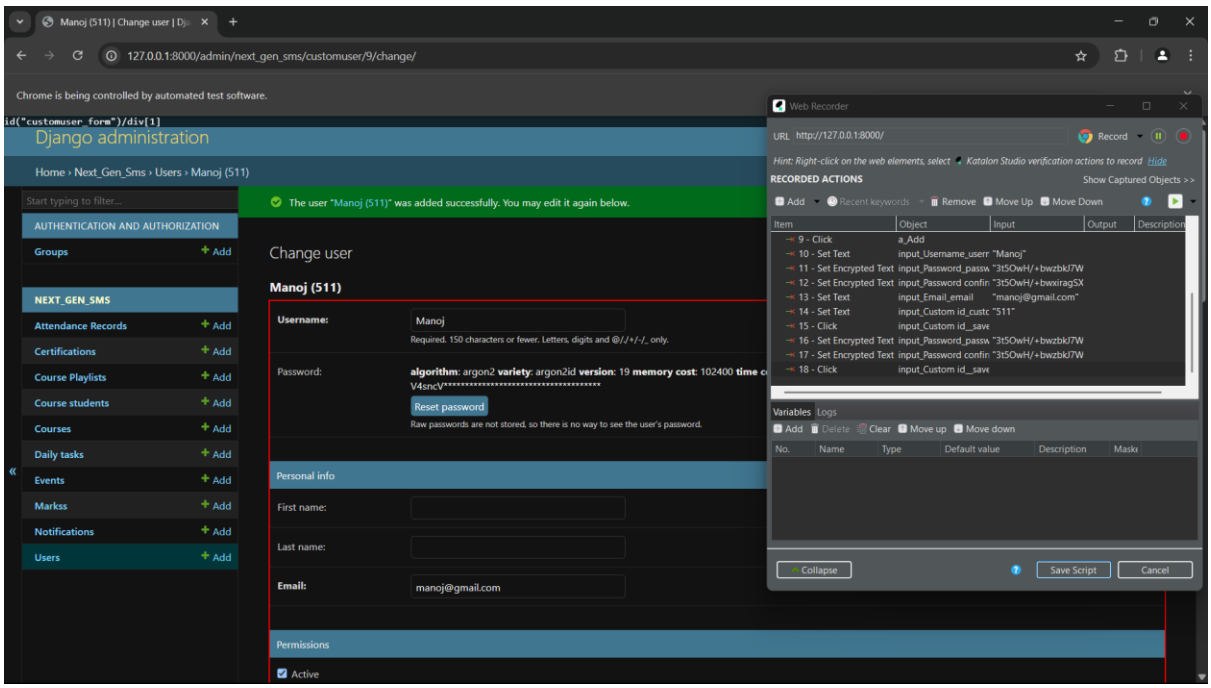
9.3 Screen Shots



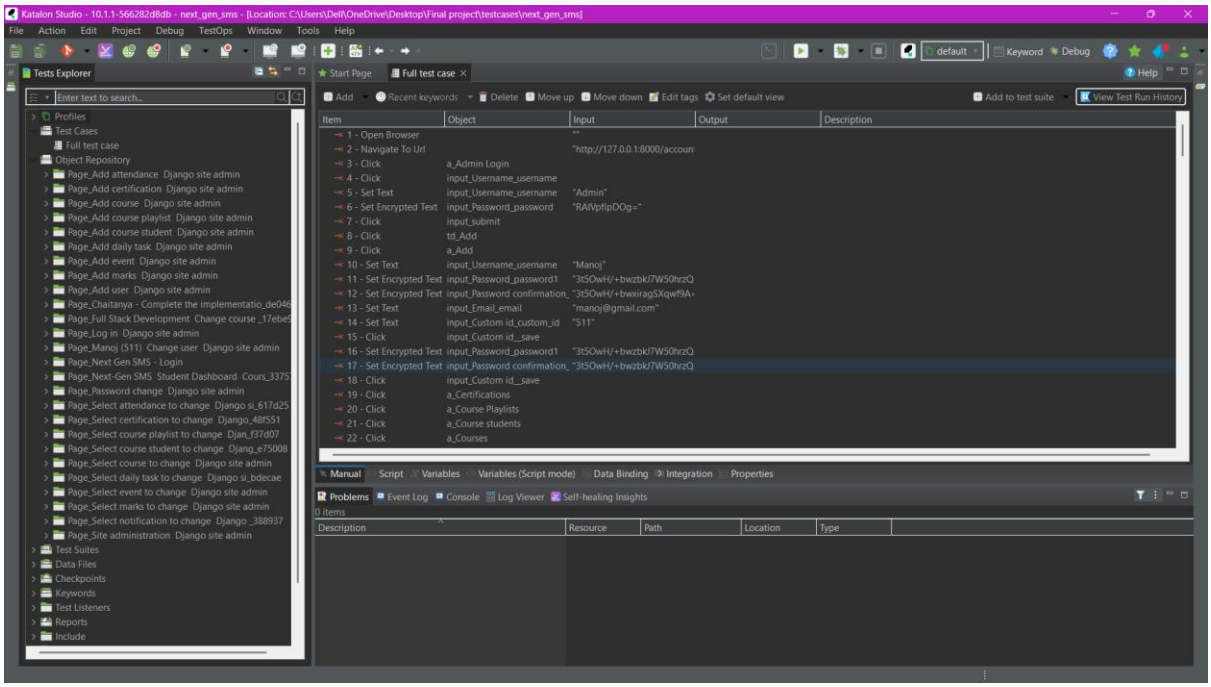
Creating a new project in katalon



Entering the web URL which should be tested and recording it.



Recording of test cases



The final recorded test cases

9.4 Test Cases :

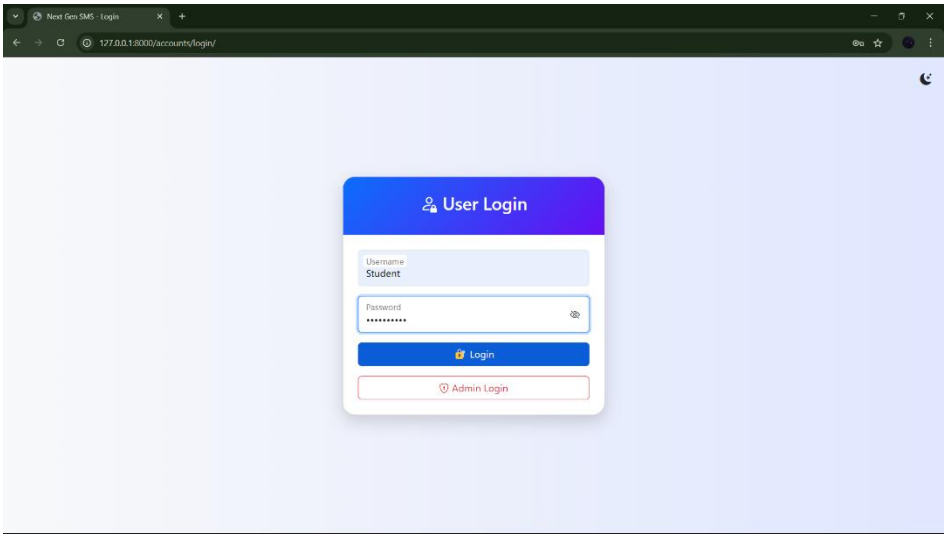
No	Test Case	Input	Expected Result	Result
1. Login Module Test Case				
1	Blank Username and password	Username = Null Password = Null	Error Message	Pass
2.	Wrong User Credentials	Username = Admin Password = 123	Invalid Login Credentials	Pass
3.	Valid Username and Password	Username = Admin Password = admin	Successful Login	Pass
2. Add New Student Record				
1.	Add new Student	Enter student details like name, email etc.	The new student record should be saved and displayed	Pass
3. Update Student Record				
1.	Update Student Record	Click Update	The Update button should work and show existing student details	Pass
2.	Update Student Record	Enter details	The student details should be saved and displayed	Pass
4. Delete Student Record				
1.	Delete Student Record	Delete Button	The delete button should work	Pass
5. Upload File Test Case				
1.	Upload profile picture	A picture should be uploaded	The upload button should work and upload successfully	Pass
2.	Display profile picture	Upload profile picture	The profile picture should be uploaded	Fail
3.	Upload Certification	A PDF/PNG format should be uploaded	The upload button should work and	Pass

			certification uploaded successfully	
6. View Student Details				
1.	View Student details	Student Clicks on My Profile	The details of the student should be displayed and editable	Pass
2.	Admin View	Admin searches for student details	The admin should see the student details and it should be editable	Pass
7. Logout				
1.	Logout	The user clicks on logout button	The use should be redirected to login page	Pass

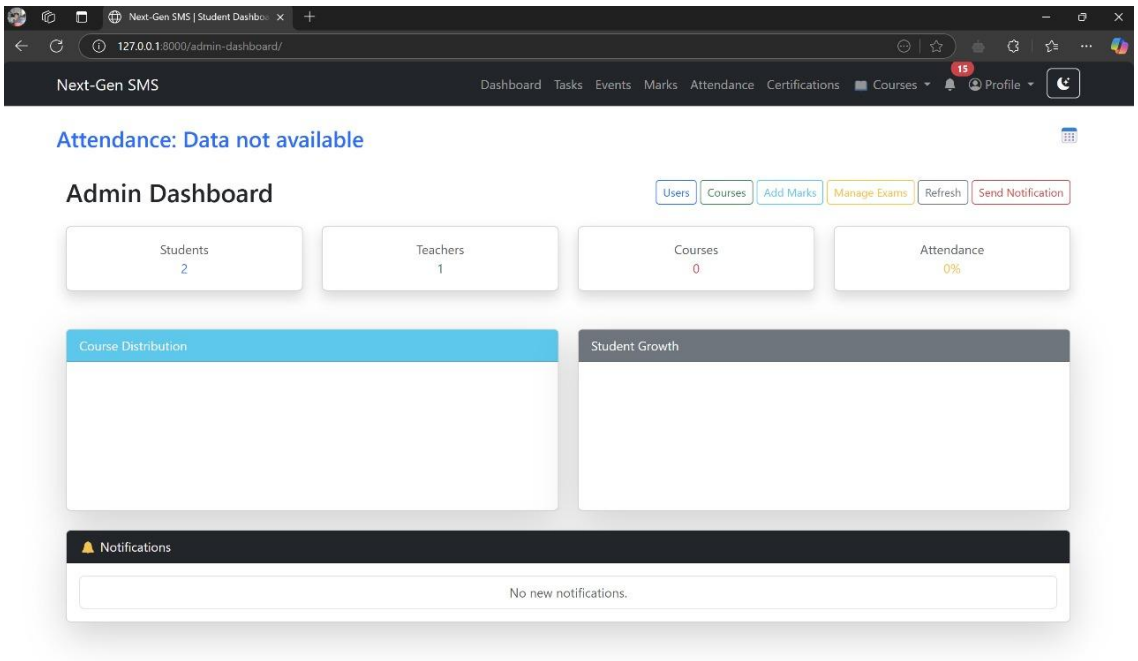
10. OUTPUT SCREENS

Login page: The below two screenshots shows us the login page of student and admin.

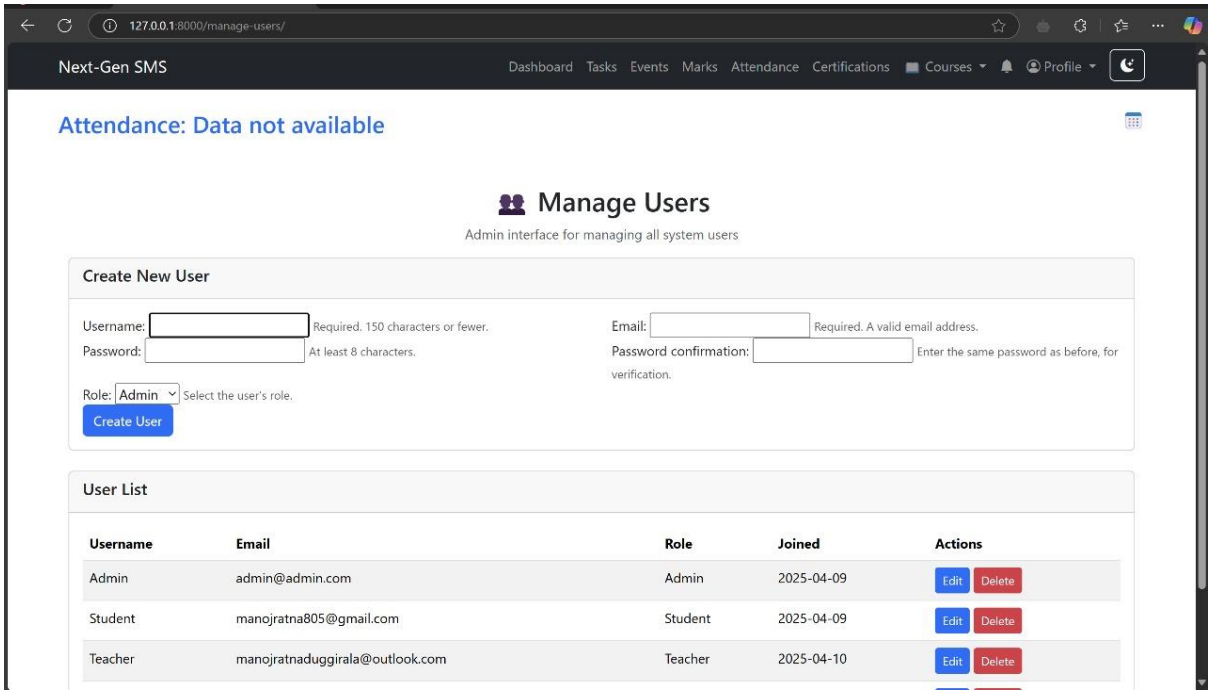
Student login:



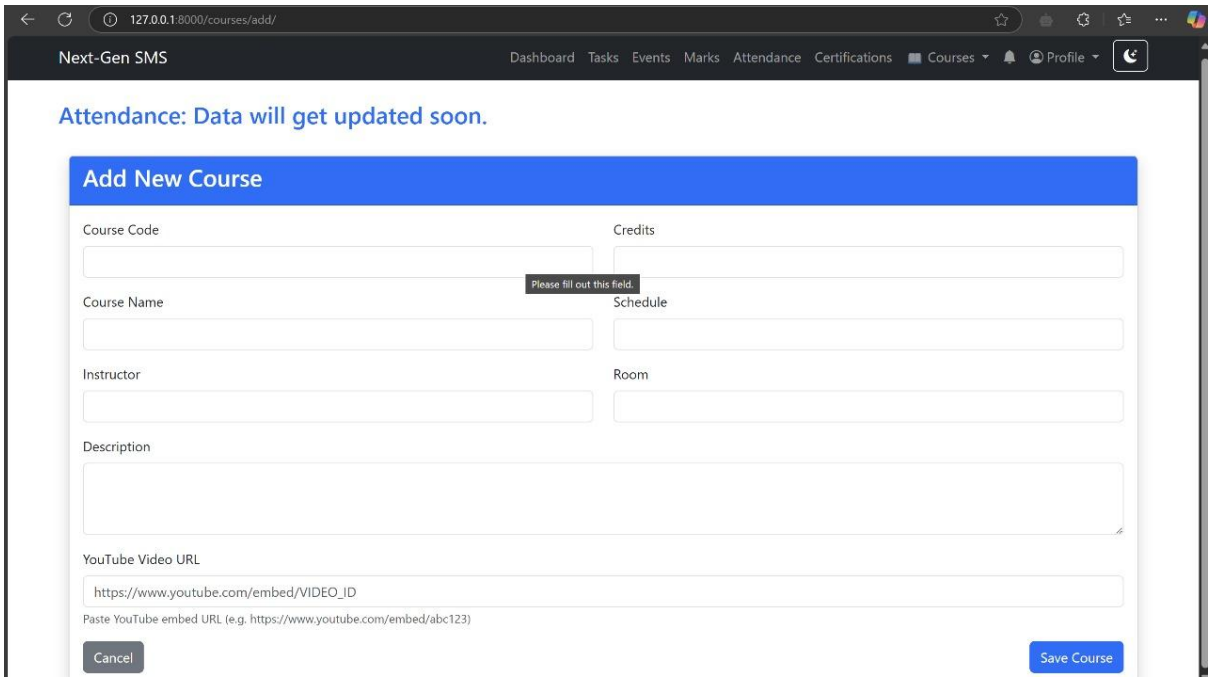
Admin Dashboard: Here in the below screenshot, we can see the admin dashboard after login this is where the admin adds the student records like attendance, marks student details etc.



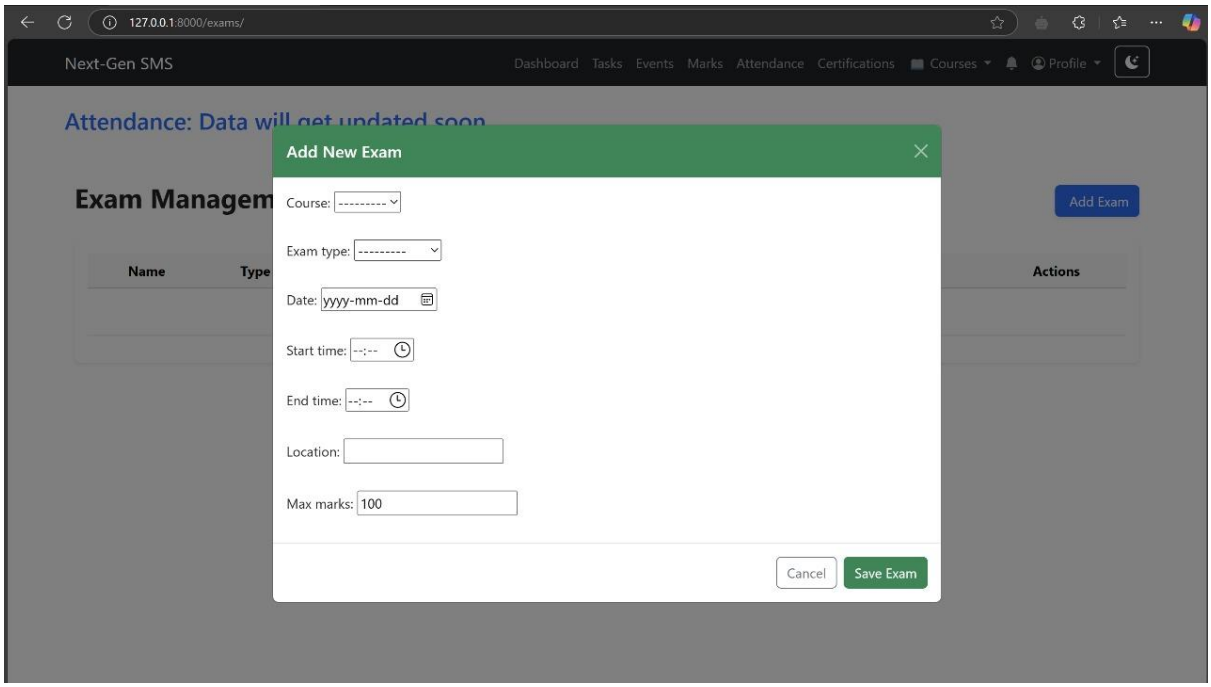
Adding attendance: The below screenshot is where admin adds the attendance of the students



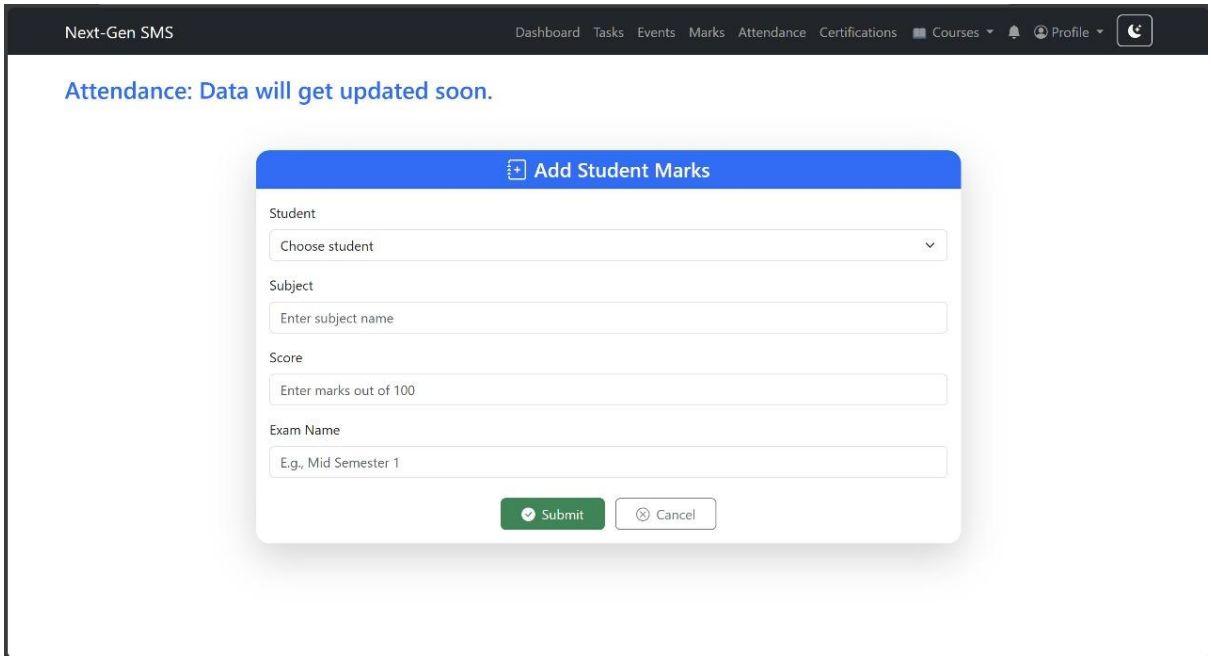
Add Course: The below screen short is where admin adds courses.



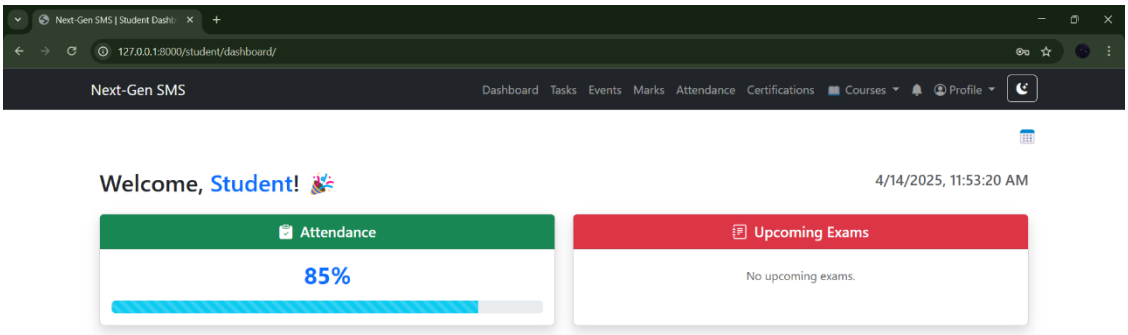
Add Exam: The below screenshot is where admin schedules an exam.



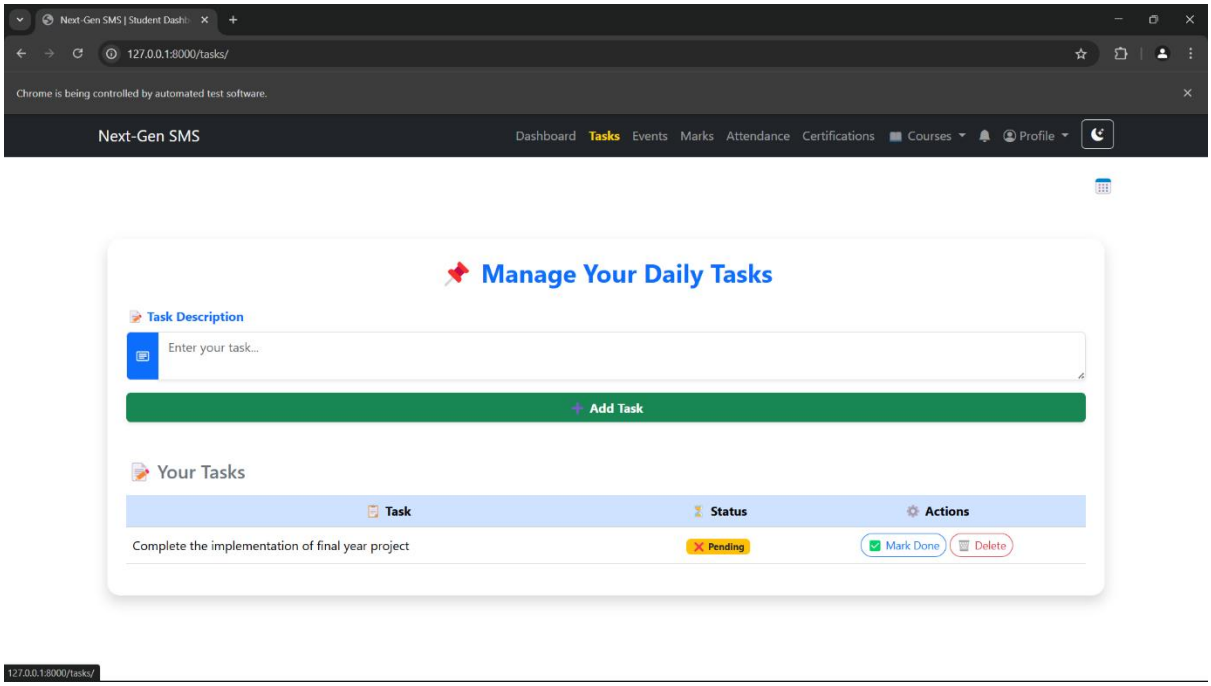
Add Marks: The below screenshot shows us that admin can add marks to the exam conducted.



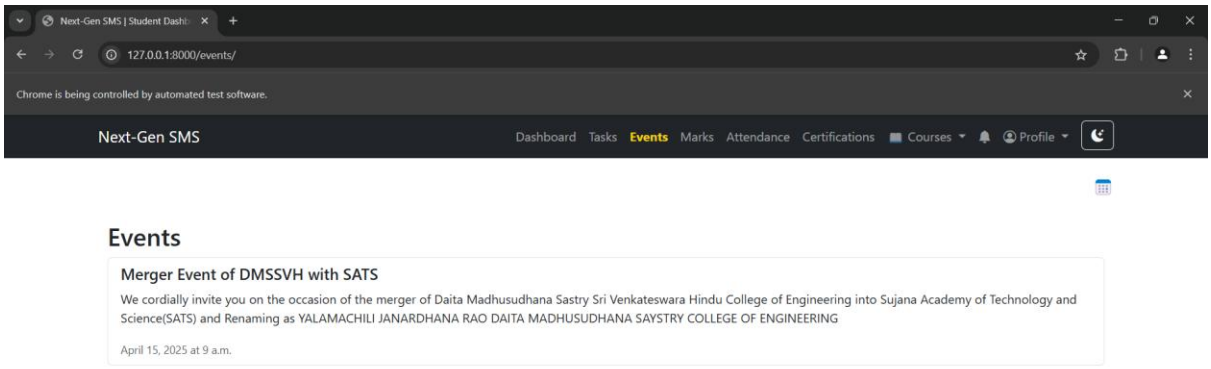
Student Dashboard: The below screenshot is the dashboard of the student profile which shows us the attendance percentage of the student and upcoming exam.



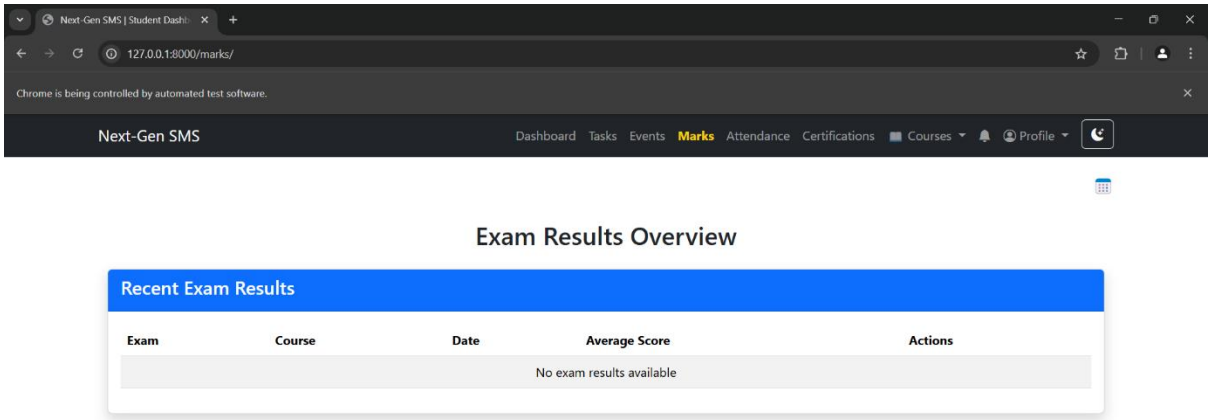
Tasks: In the tasks section students can manage their daily tasks.



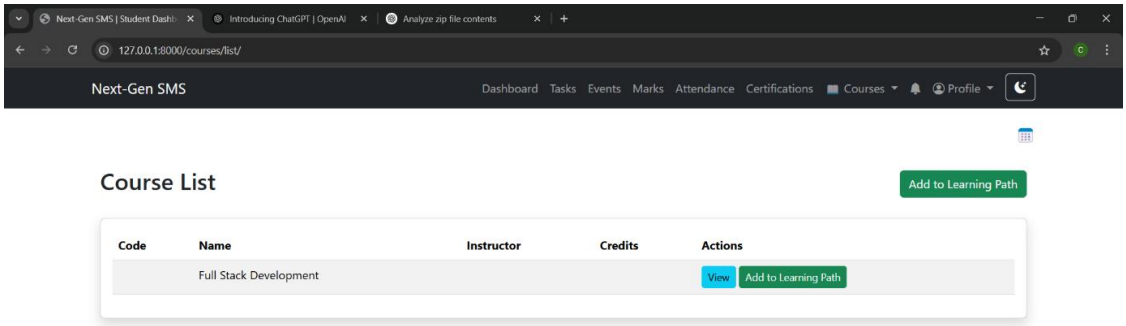
Events : In the events section the students can view the events posted by the admin.



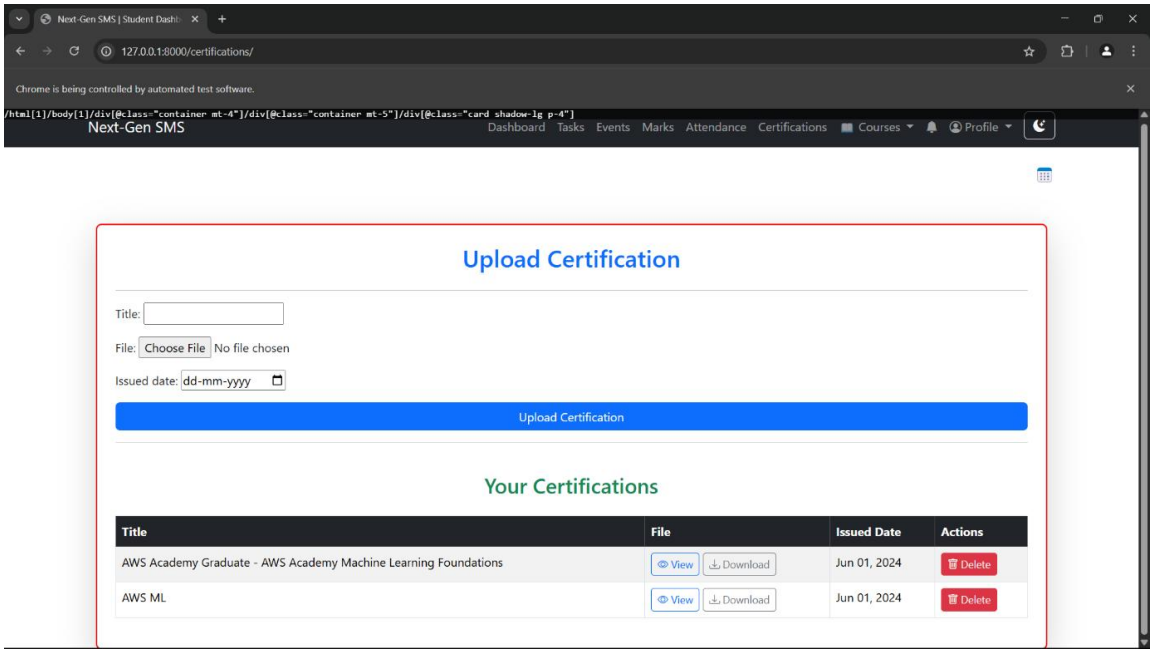
Exam Results: In this section studetns can view therir results posted by the admin.



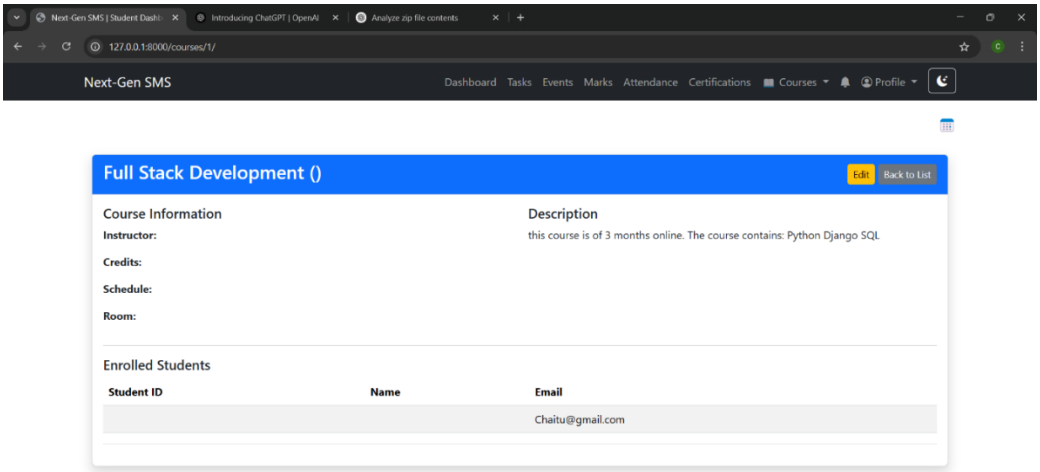
Course list : The student scan view the available courses and add it to their learning path



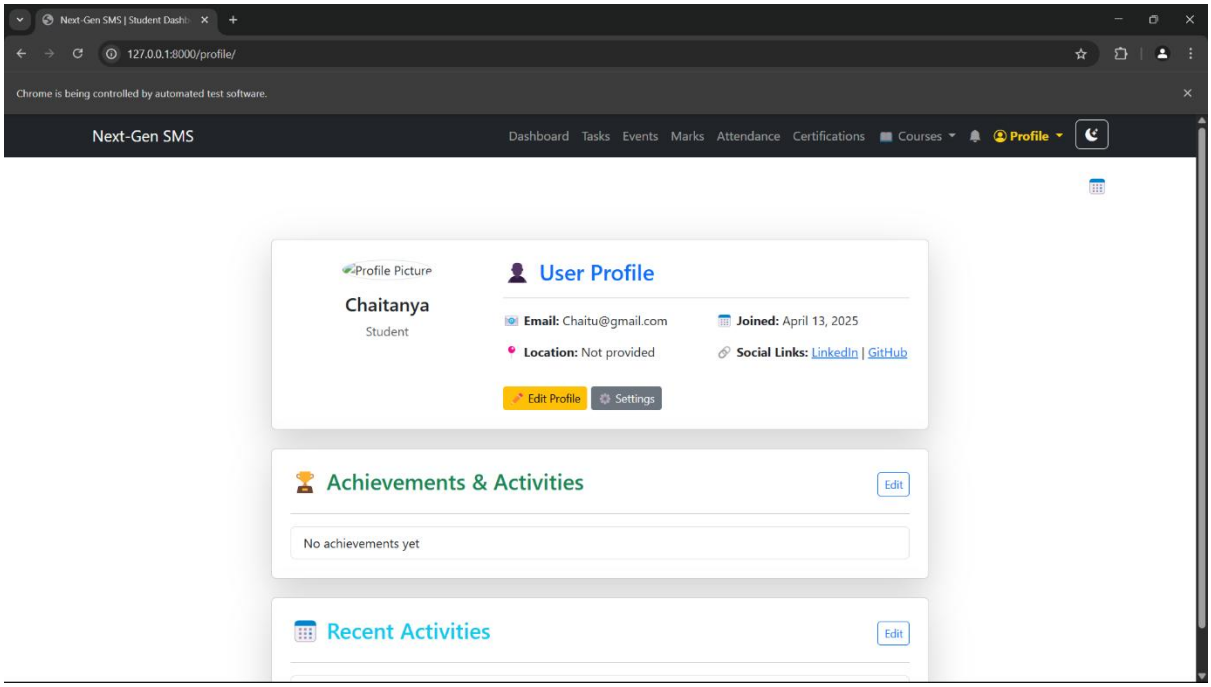
Certifications : In this section student s can view and upload their certifications which are not optioned in the portal



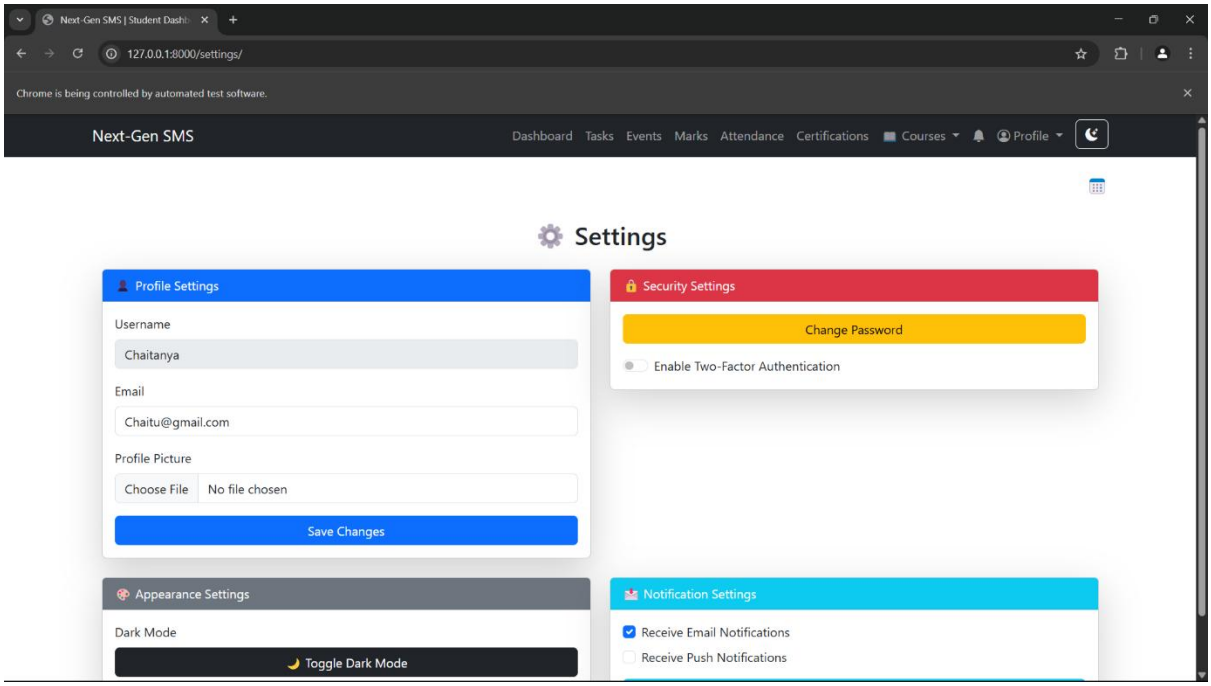
The below picture is the view after clicking the view button in the course



User Profile : in this section the student can view, update their details.



Settings: In this section the student can change password their login credentials etc.


Next-Gen SMS | Student Dashb
127.0.0.1:8000/settings/
Chrome is being controlled by automated test software.
Next-Gen SMS
Dashboard
Tasks
Events
Marks
Attendance
Certifications
Courses
Profile
Settings
Profile Settings
Username
Chaitanya
Email
Chaitu@gmail.com
Profile Picture
Choose File
No file chosen
Save Changes
Security Settings
Change Password
Enable Two-Factor Authentication
Appearance Settings
Dark Mode
Toggle Dark Mode
Notification Settings
Receive Email Notifications
Receive Push Notifications

11. CONCLUSION AND FUTURE WORK

Conclusion:

Finally, with due diligence, the student management web-based Application system is carried out. It is system that assists the user to work with the day-to-day activities involved in the academic institution. It lessens the amount of manual hard work and provides greater efficiency diminishing the amount of time taken for detailing different modules. The interface provides user-friendly experience to everyone. Only verified users can access the information concerning students and faculties. At last, we may state that the performance of this new system is accurate, precise and it successfully performs the assigned tasks.

Future work:

The results obtained from the experiments and testing ensures that the proposed method is efficient and user-friendly. As compared to existing methods of managing the academic institutions, this project which yields centralized software makes the work administration and management easier and provides detailed information about the topic of users interest in just one mouse click. The educational institution can be provided with an easy-to-use user interface centralized software in which all services associated with the institution can interact with each other and share the data. As this is a ReST API hosted in the AWS Cloud server, the user will be able to access the resources from remote places. As the application is developed using micro-service architecture and agile methodology, in the future services can be added without having to make changes to the existing code.

12. REFERENCES

- **Django Project Documentation** – https://docs.djangoproject.com
- **Python Official Documentation** – https://www.python.org/doc/
- **W3Schools – HTML, CSS, SQL, Python tutorials** – https://www.w3schools.com
- **GeeksforGeeks – Programming and CS tutorials** – https://www.geeksforgeeks.org
- **Stack Overflow** – Community support and code-related troubleshooting – https://stackoverflow.com
- **GitHub – For version control and accessing open-source projects** – https://github.com
- **Visual Studio Code – Official site and documentation** – https://code.visualstudio.com
- **DBMS & SQL Notes – Lecture materials and tutorials provided by faculty.**
- **Android Developers – For reference in case of mobile extension** – https://developer.android.com
- **Bootstrap & CSS Libraries** – https://getbootstrap.com