KISHKINDA UNIVERSITY

Mini Project Report

On

“Analysis of Students Marks”

Department of Computer Science and Engineering

Submitted By:

JASON KENNETH N	KUB23CSE045
BHARATH KUMAR JM	KUB23CSE025
C ANIRUDH	KUB23CSE028
NITHYA PRAJWAL P	KUB23CSE096

Faculty Coordinator: Mujeeb Ansari

Chairperson: Dr. Rajashree V Biradar

Table of Contents

Introduction

- The "Analysis of Students Marks" project is designed to manage and analyze student academic performance efficiently. Educational institutions generate vast amounts of student data, including marks, attendance, and personal details. This project aims to provide a structured approach to store, retrieve, and analyze this data to enhance academic planning and reporting.

- Key benefits of the project include:
- Efficient data management and retrieval.
- Improved communication between students, faculty, and administrators.
- Data-driven decisions to improve student retention and success rates.

ProjectOverview:

- The project focuses on creating a system to analyze student marks and generate performance reports. It includes the following components:
- **Student Database:** Stores student details, marks, and academic records.
- **Analysis Module:** Analyzes marks to generate performance reports.
- **Reporting Module:** Provides visual and textual reports for faculty and administrators.

ProblemStatement:

Educational institutions often struggle with:

- Manual and inefficient mark analysis processes.
- Lack of centralized data storage leading to inconsistencies.
- Difficulty in generating actionable insights from student performance data.

This project addresses these challenges by automating mark analysis and providing a centralized platform for data management.

Objective

- Develop a system to store and manage student marks efficiently.
- Analyze marks to identify trends and performance patterns.
- Generate comprehensive reports for faculty and administrators.
- Ensure data security and compliance with educational regulations.

SoftwareRequirements:

- Python3.12(64-bit)
- VisualStudioCode

HardwareRequirements:

Processor	: Inteli5
Ram	:16 GB
HardDisk	:500GB

5. Methodology

The project follows a structured development approach:

5.1 Requirements Gathering:

- Identified functional and non-functional requirements through stakeholder interviews and literature review.
- Focused on data storage, analysis, and reporting needs.

5.2 System Design

- **Database Design:** Utilized relational database principles with tables for Students, Marks, and Subjects.
    - **Students Table:** Stores student details (StudentNo, Name, etc.).
    - **Marks Table:** Stores marks for each subject (StudentNo, SubjectCode, Mark, Year).
    - **Subjects Table:** Stores subject details (SubjectCode, SubjectName).
- **Normalization:** Applied to reduce redundancy and improve data integrity.

5.3 Implementation

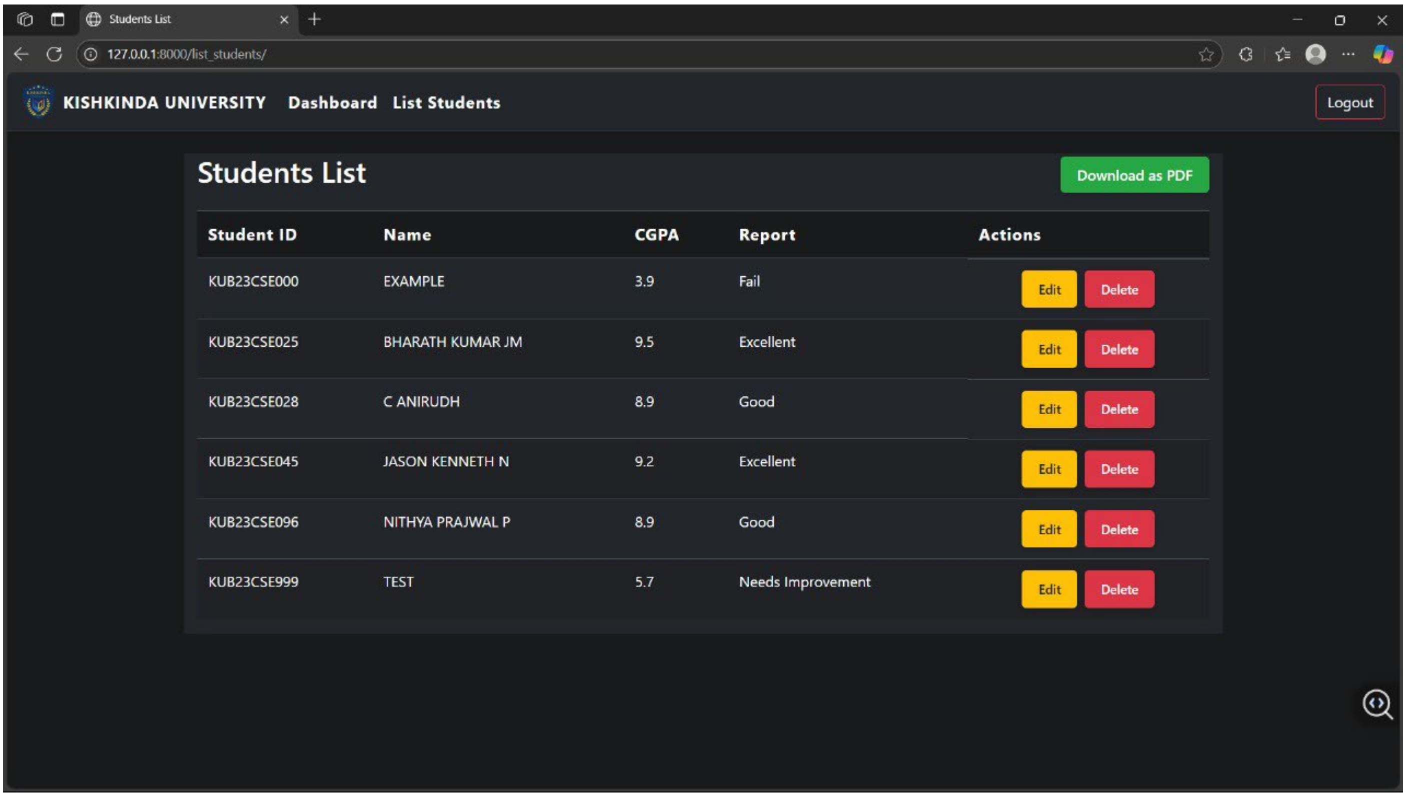
- **Backend:** Python-based application for data processing and analysis.
- **Frontend:** Simple interface for data entry and report generation.
- **CRUD Operations:** Implemented for managing student and marks data.

5.4 Testing

- Conducted unit testing to validate functionalities.
- Tested data accuracy and report generation.

6. System Design

6.1 Database Schema



Students List

Student ID	Name	CGPA	Report	Actions
KUB23CSE000	EXAMPLE	3.9	Fail	Edit Delete
KUB23CSE025	BHARATH KUMAR JM	9.5	Excellent	Edit Delete
KUB23CSE028	C ANIRUDH	8.9	Good	Edit Delete
KUB23CSE045	JASON KENNETH N	9.2	Excellent	Edit Delete
KUB23CSE096	NITHYA PRAJWAL P	8.9	Good	Edit Delete
KUB23CSE999	TEST	5.7	Needs Improvement	Edit Delete

6.2 Modules

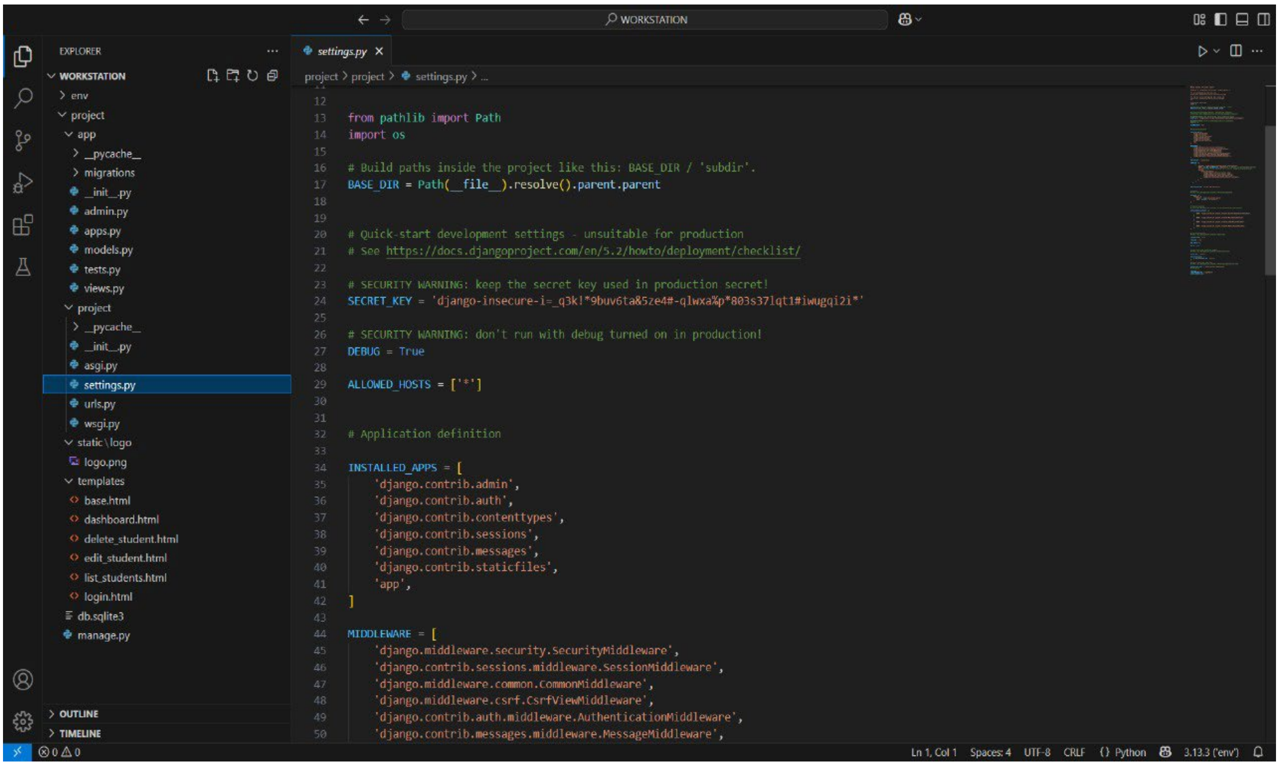
- **Student Module:** Manages student details.
- **Marks Module:** Handles mark entry and updates.
- **Report Module:** Generates performance reports

7. Implementation

7.1 Key Features

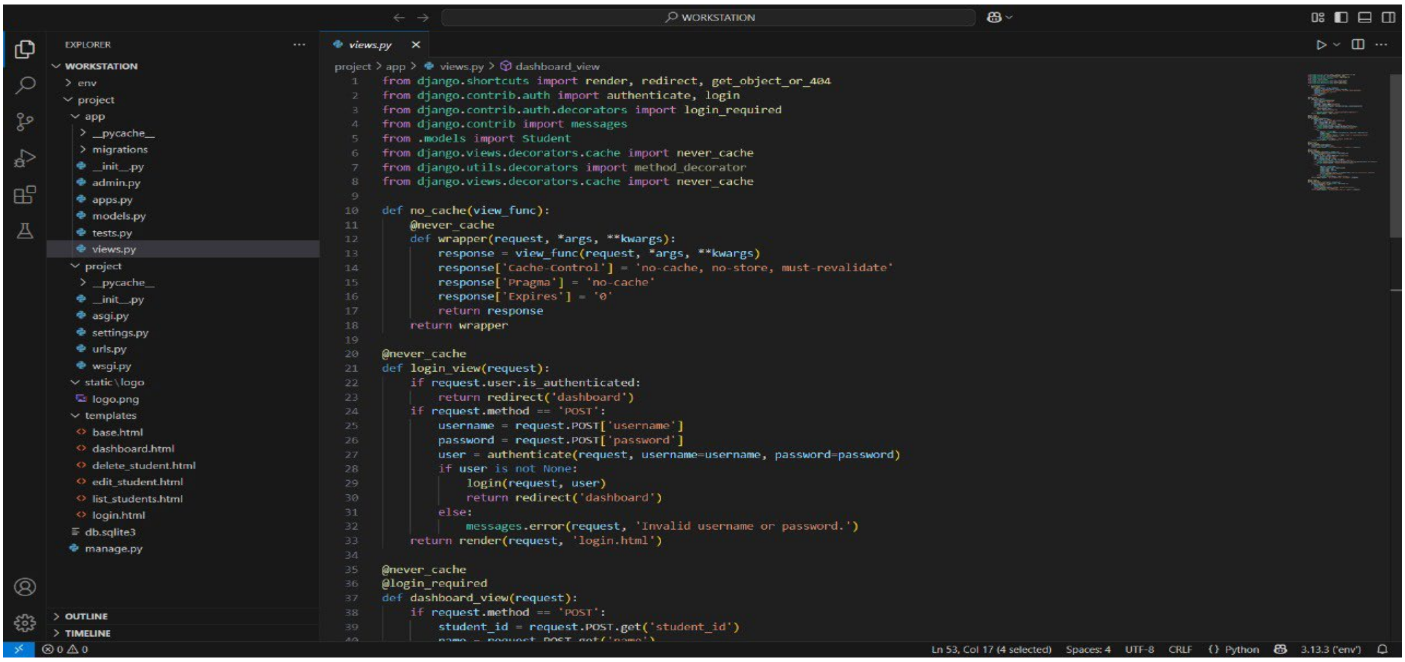
- **Student Data Entry:** Faculty can enter and update student marks.
- **Performance Analysis:** Calculates CGPA and generates performance reports.
- **Report Generation:** Exports reports in PDF format.

7.2 Sample Code Snippet



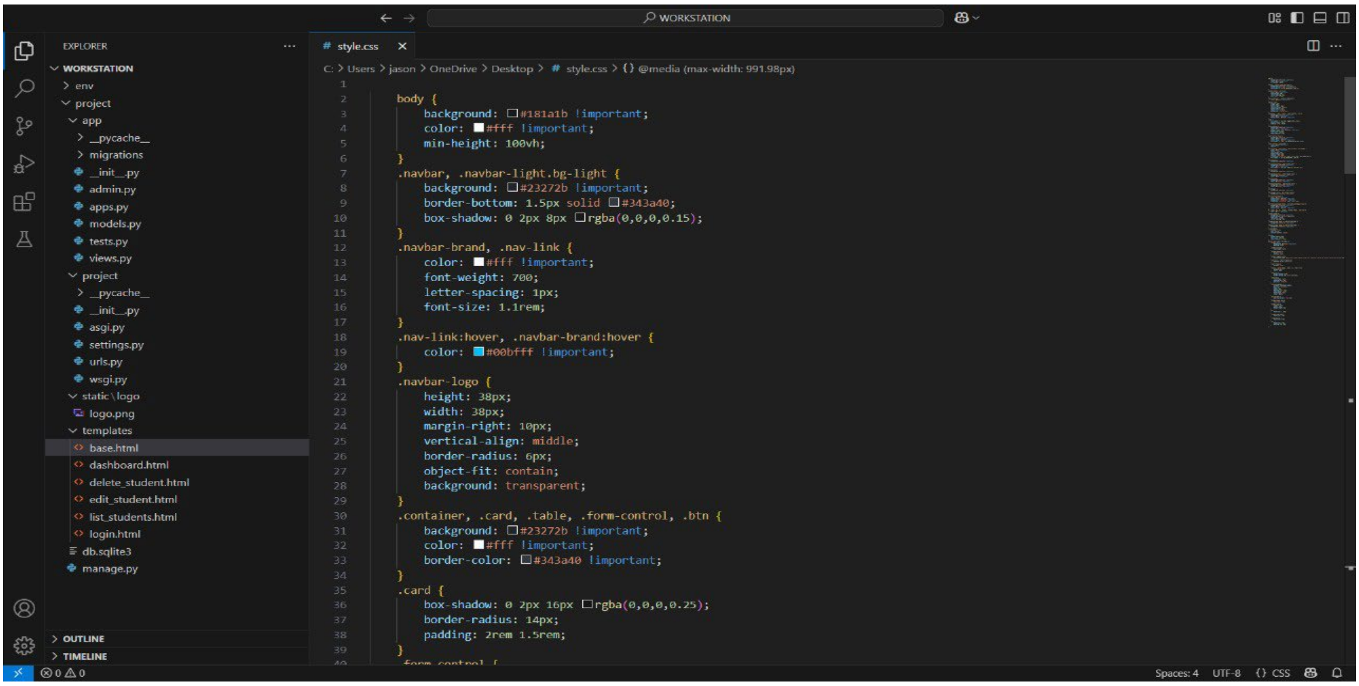
```python

from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-i=_q3k!*9buv6ta&5ze4#-qlwxa%p*803s37lqt1#iwugqi2i*'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['*']


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

Django Settings



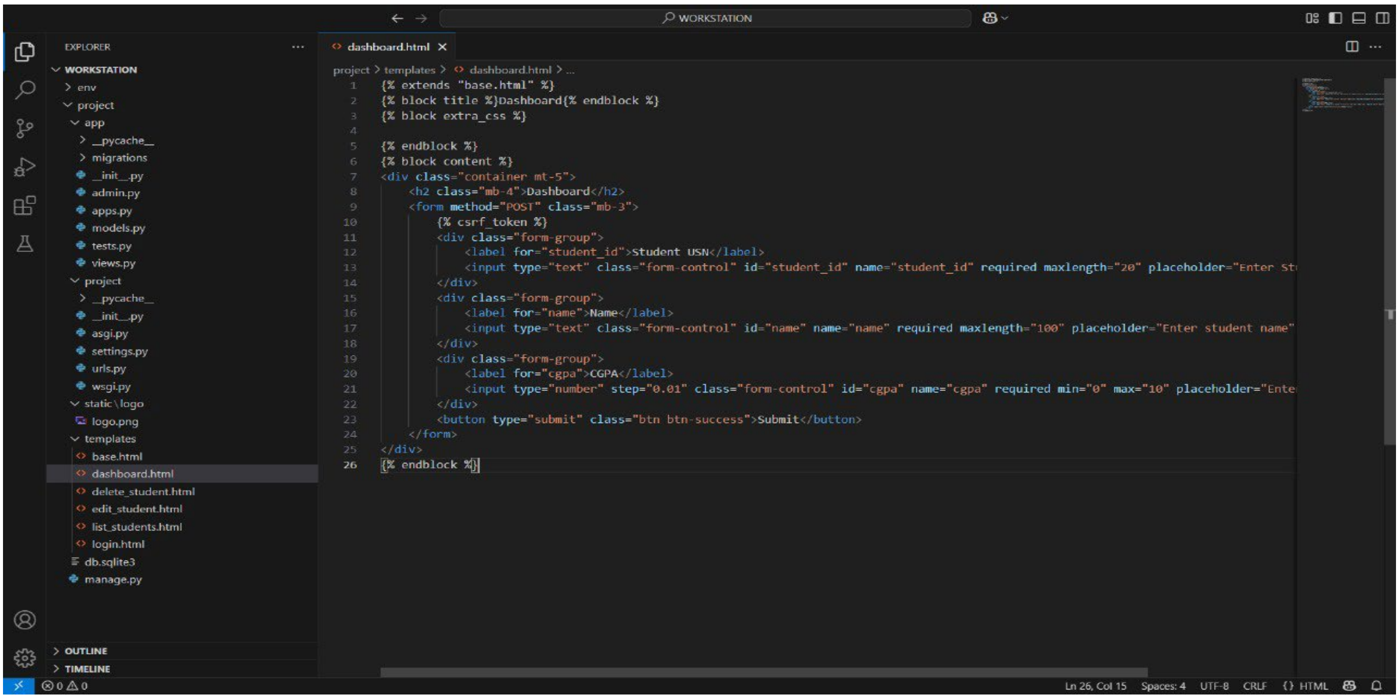
```python
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .models import Student
from django.views.decorators.cache import never_cache
from django.utils.decorators import method_decorator
from django.views.decorators.cache import never_cache

def no_cache(view_func):
    @never_cache
    def wrapper(request, *args, **kwargs):
        response = view_func(request, *args, **kwargs)
        response['Cache-Control'] = 'no-cache, no-store, must-revalidate'
        response['Pragma'] = 'no-cache'
        response['Expires'] = '0'
        return response
    return wrapper

@never_cache
def login_view(request):
    if request.user.is_authenticated:
        return redirect('dashboard')
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('dashboard')
        else:
            messages.error(request, 'Invalid username or password.')
    return render(request, 'login.html')

@never_cache
@login_required
def dashboard_view(request):
    if request.method == 'POST':
        student_id = request.POST.get('student_id')
```

Logic Code



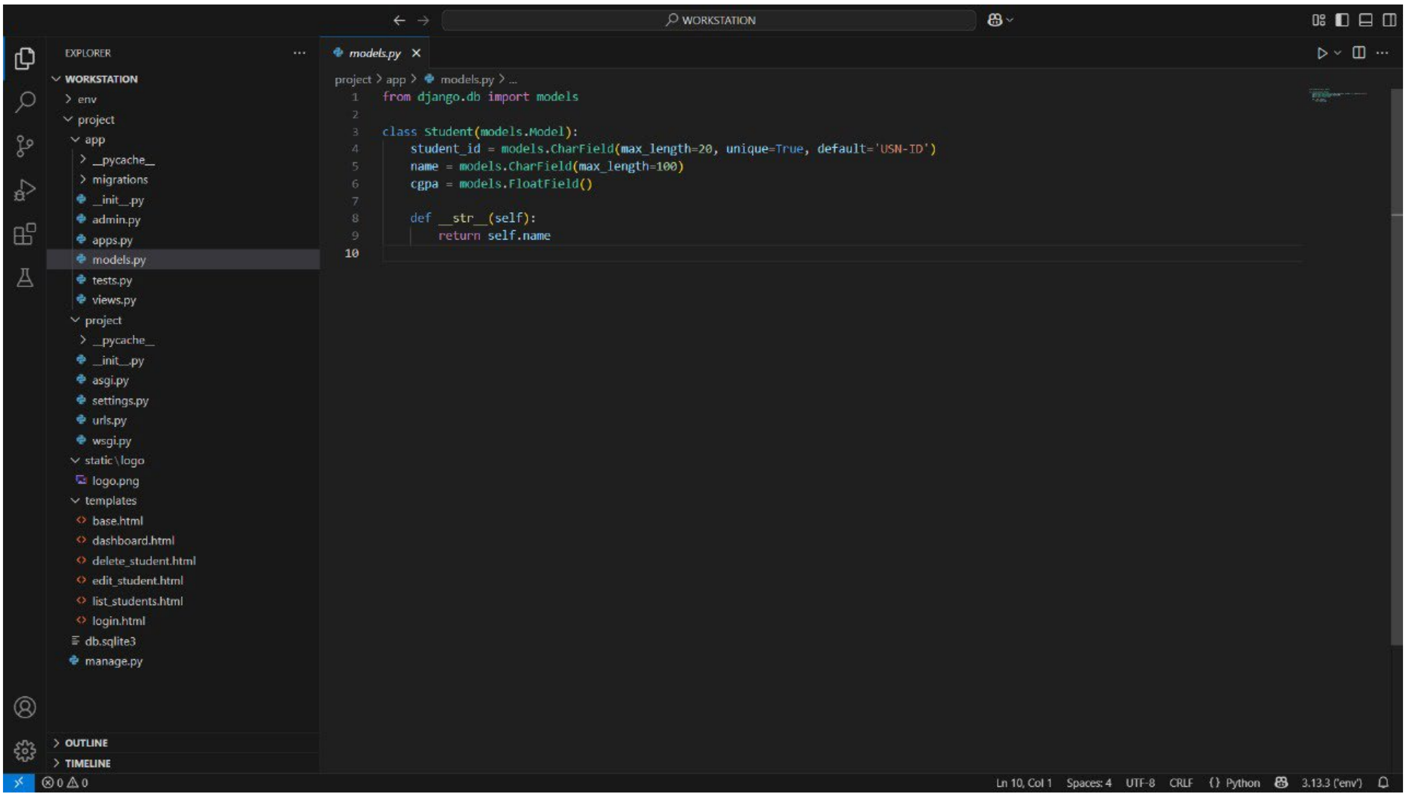
```css

    body {
        background: #181a1b !important;
        color: #fff !important;
        min-height: 100vh;
    }
    .navbar, .navbar-light.bg-light {
        background: #23272b !important;
        border-bottom: 1.5px solid #343a40;
        box-shadow: 0 2px 8px rgba(0,0,0,0.15);
    }
    .navbar-brand, .nav-link {
        color: #fff !important;
        font-weight: 700;
        letter-spacing: 1px;
        font-size: 1.1rem;
    }
    .nav-link:hover, .navbar-brand:hover {
        color: #00bfff !important;
    }
    .navbar-logo {
        height: 38px;
        width: 38px;
        margin-right: 10px;
        vertical-align: middle;
        border-radius: 6px;
        object-fit: contain;
        background: transparent;
    }
    .container, .card, .table, .form-control, .btn {
        background: #23272b !important;
        color: #fff !important;
        border-color: #343a40 !important;
    }
    .card {
        box-shadow: 0 2px 16px rgba(0,0,0,0.25);
        border-radius: 14px;
        padding: 2rem 1.5rem;
    }
```

CSS Stylin



```
{% extends "base.html" %}
{% block title %}Dashboard{% endblock %}
{% block extra_css %}

{% endblock %}
{% block content %}
<div class="container mt-5">
    <h2 class="mb-4">Dashboard</h2>
    <form method="POST" class="mb-3">
        {% csrf_token %}
        <div class="form-group">
            <label for="student_id">Student USN</label>
            <input type="text" class="form-control" id="student_id" name="student_id" required maxlength="20" placeholder="Enter St
        </div>
        <div class="form-group">
            <label for="name">Name</label>
            <input type="text" class="form-control" id="name" name="name" required maxlength="100" placeholder="Enter student name"
        </div>
        <div class="form-group">
            <label for="cgpa">CGPA</label>
            <input type="number" step="0.01" class="form-control" id="cgpa" name="cgpa" required min="0" max="10" placeholder="Ente
        </div>
        <button type="submit" class="btn btn-success">Submit</button>
    </form>
</div>
{% endblock %}
```

HTML



```python
from django.db import models

class Student(models.Model):
    student_id = models.CharField(max_length=20, unique=True, default='USN-ID')
    name = models.CharField(max_length=100)
    cgpa = models.FloatField()

    def __str__(self):
        return self.name

```

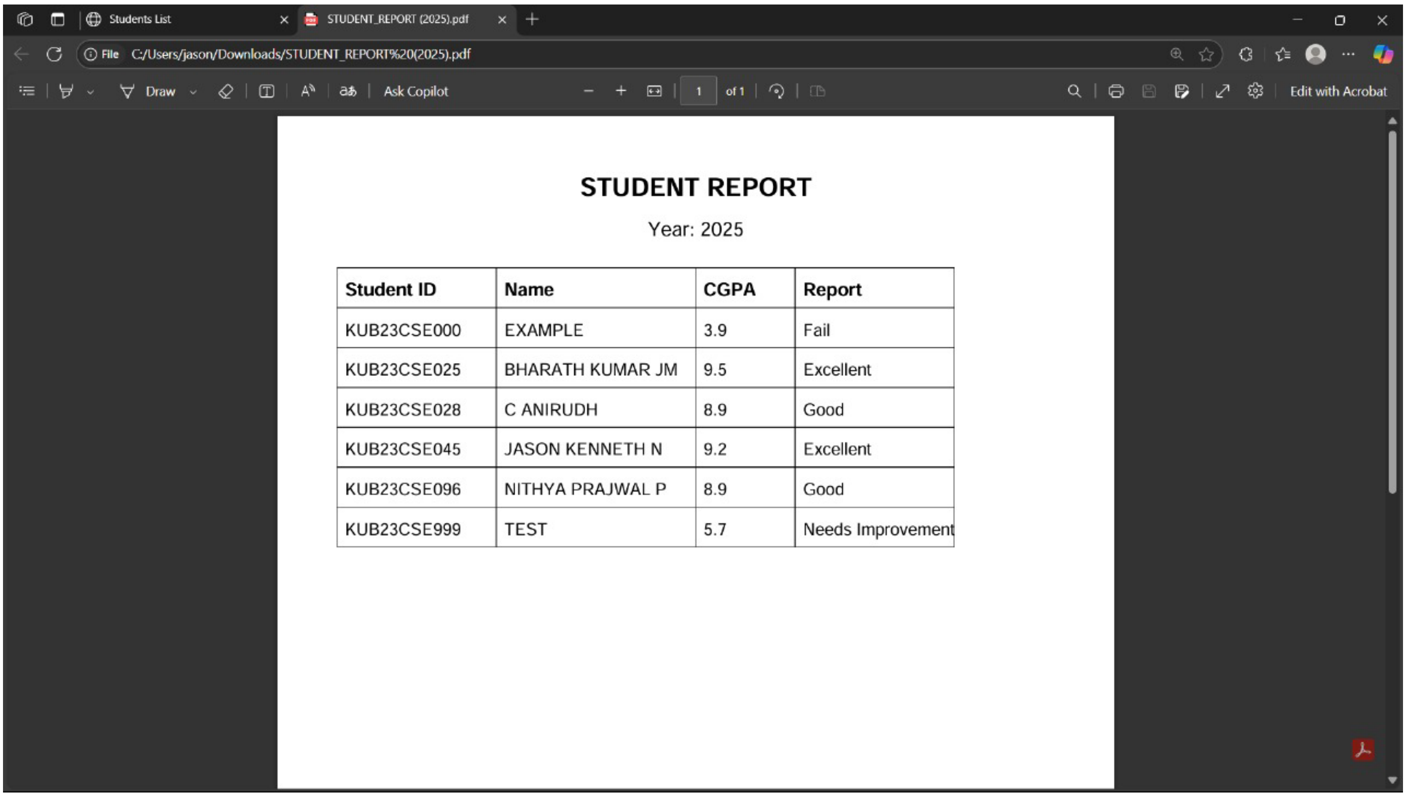
Databasse

8. Results

The system successfully:

- Stores and retrieves student marks efficiently.
- Generates accurate performance reports.
- Provides actionable insights for faculty.

Sample Report Output


Students List
STUDENT_REPORT (2025).pdf
File C:/Users/jason/Downloads/STUDENT_REPORT%20(2025).pdf
Draw
Ask Copilot
1
of 1
Edit with Acrobat

STUDENT REPORT

Year: 2025

Student ID	Name	CGPA	Report
KUB23CSE000	EXAMPLE	3.9	Fail
KUB23CSE025	BHARATH KUMAR JM	9.5	Excellent
KUB23CSE028	C ANIRUDH	8.9	Good
KUB23CSE045	JASON KENNETH N	9.2	Excellent
KUB23CSE096	NITHYA PRAJWAL P	8.9	Good
KUB23CSE999	TEST	5.7	Needs Improvement

9. Conclusion

The "Analysis of Students Marks" project provides a robust solution for managing and analyzing student performance data. By automating mark analysis and report generation, it enhances academic planning and decision-making.

10. Future Enhancements

1. **Integration with Machine Learning:** Predict student performance trends.
2. **User Authentication:** Secure login for faculty and administrators.
3. **Data Visualization:** Interactive dashboards for better insights.
4. **Mobile Application:** Enhance accessibility for faculty and students.

11. References

- Smith, J. (2022). *Database Management in Education: Best Practices*. Academic Publishing.
- Python Documentation. Retrieved from https://docs.python.org/3/library/
- Git Repository: https://github.com/jasonkennethn/WEB_DEVELOPMENT_2025_3RD_SEM