SAMSUNG INNOVATION CAMPUS

GSSSIETW , MYSURU

Student Academic Performance Evaluation with Pandas Joins

Problem Statement :- Develop a Pandas-based web dashboard to analyze student data (from a CSV dataset) and provide performance insights.

Name - Preeti Umesh Halamani

Semester : 5th

Department : AI&DS

USN – 4GW23AD041

Email - preetiuhalamani123@gmail.com

1. Problem Statement

Students and mentors often face difficulty tracking academic progress in real time. Existing systems lack simplicity, role-based dashboards, and visualization of performance. Hence, a centralized portal is needed to:

- Manage student academic records.
- Provide role-based access for students and mentors.
- Visualize performance metrics for analysis and decision-making.

2. Project Overview

The **Student Performance Portal** is a web-based platform built using **Flask** (Python) with **Bootstrap-based frontend**.

- **Students** can view their academic performance, GPA, and ranking compared to peers.
- **Mentors** can view department-wise performance, filter students by results, and export reports.
- Data is fetched from CSV files, processed using **Pandas**, and displayed on dashboards with real-time updates.

3. Dataset Description

students.csv	**:**	Contains StudentID, Name, Department.
courses.csv	**:**	Contains CourseID, CourseName, Department for each course.
enrollments.csv	**:**	Maps StudentID to CourseID with semester information.
grades.csv	**:**	Stores grades for each student in each course. Missing grades handled as 0.

Code :-

```
import pandas as pd

# Load datasets

students = pd.read_csv('students.csv')

courses = pd.read_csv('courses.csv')

enrollments = pd.read_csv('enrollments.csv')

grades = pd.read_csv('grades.csv')

# Inspect

print("Students:\n", students.head())

print("Courses:\n", courses.head())

print("Enrollments:\n", enrollments.head())

print("Grades:\n", grades.head())

# Fill missing grades with 0

grades['Grade'] = grades['Grade'].fillna(0)

# Drop duplicates

students.drop_duplicates(inplace=True)

print("Grades after filling missing values:\n", grades)

print("Students after dropping duplicates:\n", students)

# Merge enrollments with students

student_courses = pd.merge(enrollments, students, on='StudentID', how='inner')

# Merge with grades

student_performance = pd.merge(student_courses, grades, on=['StudentID', 'CourseID'], how='left')

# Merge with courses for subject info

full_data = pd.merge(student_performance, courses, on='CourseID', how='left')

# View final merged data

print("Full Merged Data:\n", full_data.head())

# GPA per student

gpa = full_data.groupby('Name')['Grade'].mean().reset_index().rename(columns={'Grade': 'GPA'})

# Subject-wise stats

subject_stats = full_data.groupby('CourseName')['Grade'].agg(['mean', 'max', 'min']).reset_index()

# Result column

full_data['Result'] = full_data['Grade'].apply(lambda x: 'Pass' if x >= 40 else 'Fail')

# Grade category

def grade_category(avg):

    if avg >= 90: return 'A'

    elif avg >= 75: return 'B'

    elif avg >= 60: return 'C'
```

```
elif avg >= 40: return 'D'
else: return 'F'
gpa['Grade'] = gpa['GPA'].apply(grade_category)
# View GPA and grade
print("Student GPA and Grade:\n", gpa)
# Subject-wise stats
subject_stats = full_data.groupby('CourseName')['Grade'].agg(['mean', 'max', 'min']).reset_index()
print("Subject Stats:\n", subject_stats)
# Top 3 performers
top_3 = gpa.sort_values(by='GPA', ascending=False).head(3)
# Pass/Fail count
pass_fail = full_data['Result'].value_counts()
# Subject with highest average
top_subject = subject_stats.sort_values(by='mean', ascending=False).head(1)
print("Top Subject by Average Grade:\n", top_subject)
gpa.to_csv('student_gpa.csv', index=False)
subject_stats.to_csv('subject_stats.csv', index=False)
```

Data Loading & Cleaning: Four CSVs—students, courses, enrollments, and grades—are loaded; missing grades are filled with 0 and duplicate student records are removed.

1. **Data Merging**: Enrollments are merged with student info, then with grades, and finally with course details to create a unified dataset full_data.
2. **Result Classification**: A new column Result is added to label each grade as 'Pass' or 'Fail' based on a 40-mark threshold.
3. **GPA Calculation**: Average grade per student is computed as GPA, and each GPA is mapped to a letter grade (A–F) using a custom function.
4. **Subject Statistics**: For each course, mean, max, and min grades are calculated to analyze overall performance.
5. **Top Performers**: The top 3 students are identified by sorting GPA in descending order.
6. **Pass/Fail Summary**: A count of how many students passed or failed is generated for quick insight.
7. **Export & Highlight**: GPA and subject stats are saved to CSVs, and the subject with the highest average grade is highlighted

Route code :

```
from logger import logger
from flask import render_template, request, redirect, url_for
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required, current_user
from app import app, login_manager
from analyzer import load_data, get_student_view, get_mentor_view
# Dummy user store
users = {
    'alice': {'password': '123', 'role': 'student'},
    'preeti': {'password': '456', 'role': 'mentor'}
}
class User(UserMixin):
    def __init__(self, username, role):
        self.id = username
        self.role = role
@login_manager.user_loader
def load_user(user_id):
    role = users[user_id]['role']
    return User(user_id, role)
@app.route('/')
def home():
    return redirect(url_for('login'))
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username, users[username]['role'])
            login_user(user)
            return redirect(url_for('dashboard'))
    return render_template('login.html')
@app.route('/dashboard')
@login_required
def dashboard():
    data = load_data()
    if current_user.role == 'student':
```

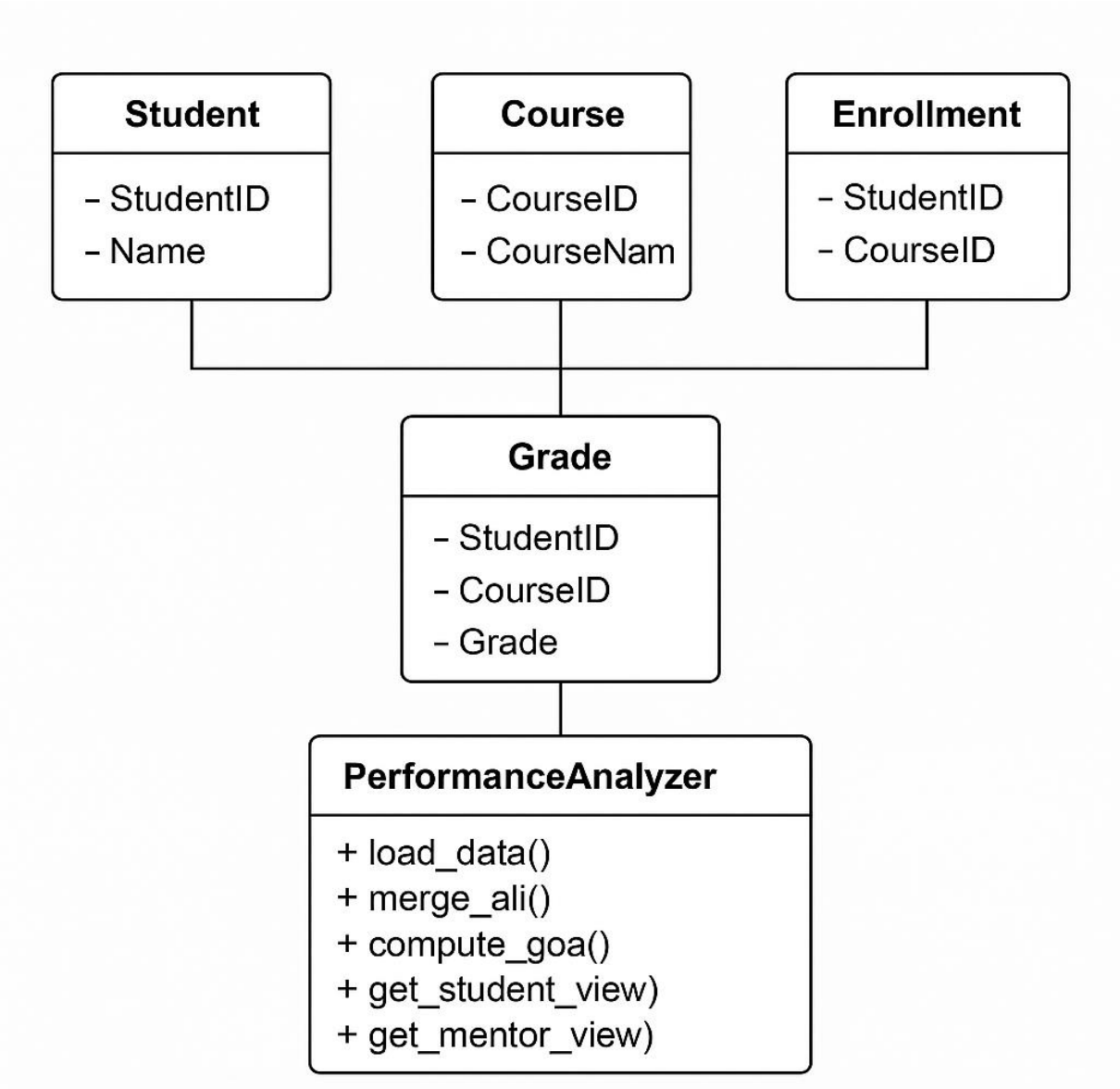
```
        student_view = get_student_view(current_user.id, data)
        return render_template('student_dashboard.html', student=student_view)
    else:
        sort_by = request.args.get('sort', 'Grade')
        filter_result = request.args.get('result')
        mentor_view = get_mentor_view(data, sort_by, filter_result)
        return render_template('mentor_dashboard.html', students=mentor_view)
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
@app.route('/export')
@login_required
def export():
    if current_user.role != 'mentor':
        return "Unauthorized", 403
    data = load_data()
    summary = get_mentor_view(data)
    summary.to_excel('data/exported_summary.xlsx', index=False)
    logger.info(f"Mentor {current_user.id} exported student summary.")
    return "Exported successfully!"
from logger import logger
logger.info("User alice logged in successfully.")
logger.warning("Missing grades found and filled with 0.")
logger.error("Failed to load students.csv")
```

User Setup: Defines dummy users with roles (student/mentor) and a User class for login management.

1. Login Flow: Authenticates users via form, logs them in, and redirects to the dashboard.
2. Dashboard View: Shows student-specific data or mentor dashboard with sorting/filtering using get_student_view or get_mentor_view.
3. Logout Route: Logs out the current user and redirects to login.
4. Export Route: Allows mentors to export student summary to Excel and logs the action.

5. Role-Based Access: Ensures only mentors can export data; students get a 403 error.
6. Logging: Tracks login success, missing grade warnings, and CSV load failures using a custom logger.
7. Routing: Uses Flask decorators to define page behavior and protect routes with login_required.

UML Diagram :



Other Codes Discription

◇ app.py

- Entry point for the Flask app
- Initializes Flask, login manager, and routes

◇ routes.py

- Contains all Flask routes
- Handles login, dashboard rendering, logout, and export

◇ analyzer.py

- Core Pandas logic: data loading, merging, GPA calculation, progress tracking

◇ logger.py

- Centralized logging setup
- Tracks login attempts, data processing, and export actions

◇ templates

- HTML files rendered by Flask
- Includes separate views for students and mentors
- navbar.html is included in all pages for navigation

◇ style

- Stores CSS and image assets
- Used for styling and frontend interactivity

◇ data

- CSV files storing student, course, enrollment, and grade data
- Easily editable and transparent

◇ logs

- Stores app.log file generated by logger
- Useful for debugging and tracking user activity

◇ README.md

- Project overview, setup instructions, and usage guide

4. Features Implemented

- Login System: Role-based authentication (Student/Mentor).
- Student Dashboard: GPA, Results, and Rankings displayed.
- Mentor Dashboard: Sort/Filter by GPA, Department, Result.
- Profile Page: Detailed student profile view.
- Data Analysis: Pandas-based aggregation for GPA & Results.
- Logging: Tracks user activities and errors in app.log.
- Export Option: Mentor can export summary as Excel file.
- Bootstrap UI: Responsive frontend design with progress bars.

5. Technical Architecture

Frontend:

- HTML templates (login.html, student_dashboard.html, mentor_dashboard.html, profile.html, navbar.html)
- Bootstrap for styling & responsiveness

Backend:

- **Flask** for routing and role-based access control.
- **Flask-Login** for session management.
- **Pandas** for CSV data processing.
- **Logging module** for activity tracking.

Database:

- CSV files as data source (students.csv, courses.csv, grades.csv, enrollments.csv).

6. Code Walkthrough

a) Data Loading & Processing – *analyzer.py*

- Loads CSV files using Pandas.
- Merges data to compute GPA & Result.
- Handles missing grades as 0.

b) Routing & Login – *routes.py*

- /login → Login page
- /dashboard → Role-based dashboards (Student/Mentor)
- /logout → Logs user out
- /export → Exports data for mentors

c) Templates – *HTML Files*

- Student & mentor dashboards with dynamic tables & progress bars.
- Navigation bar for routing between pages.

d) Logging – *logger.py*

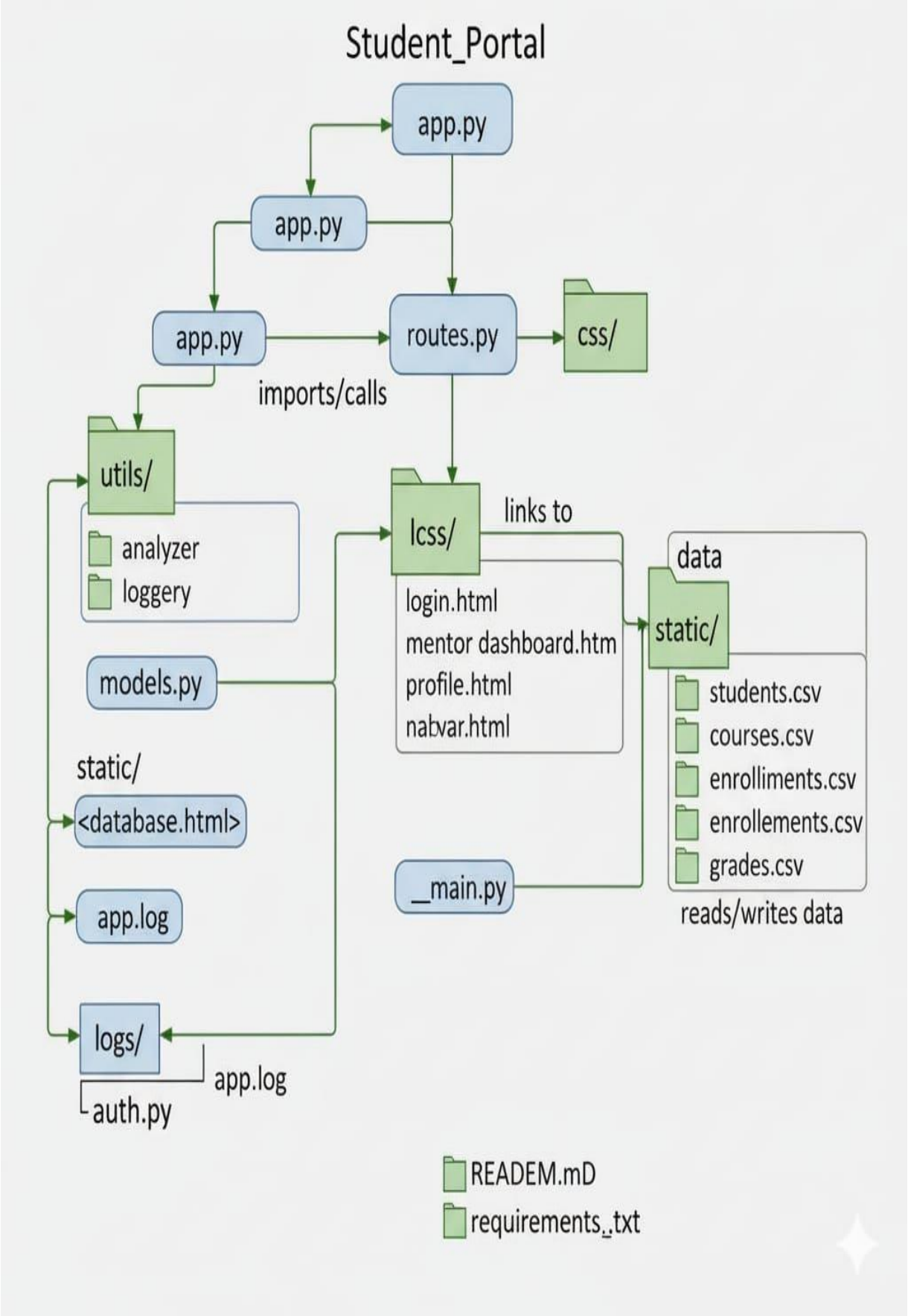
- Tracks info, warning, and error messages in logs/app.log.

7. Setup Instructions

1.**Install dependencies** pip install flask

flask-login pandas openpyxl

2.Project Structure :



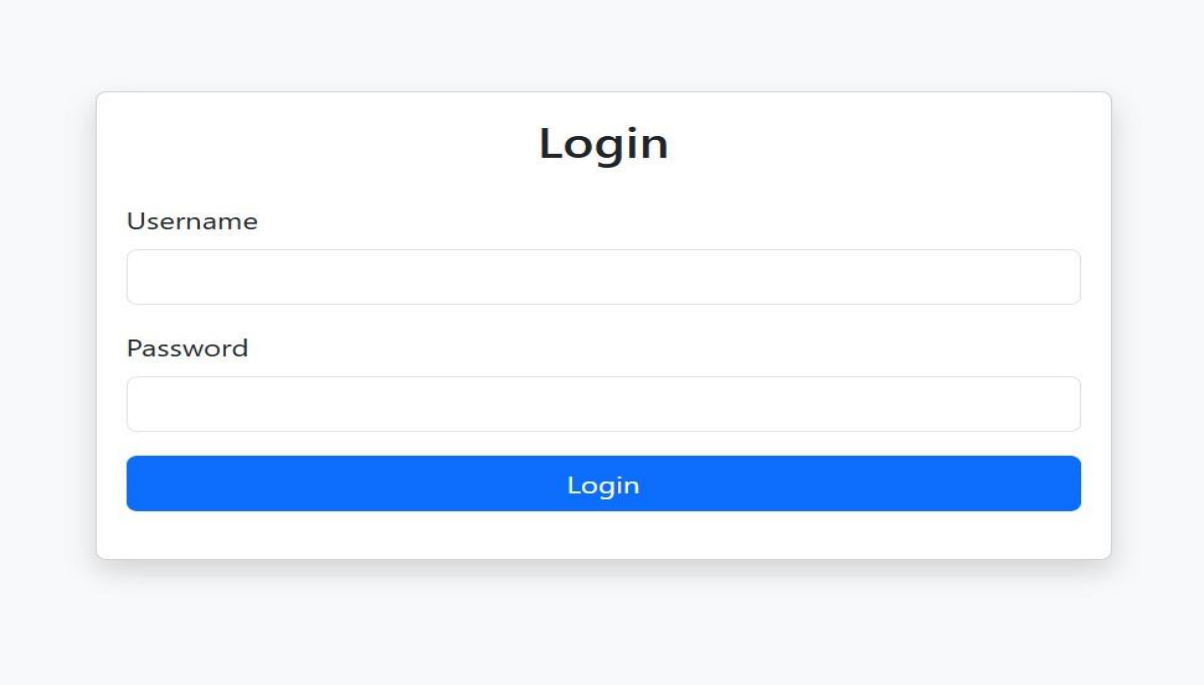
3.Run the app : python

app.py

Access at http://127.0.0.1:5000/

8. Screenshots :

Login Page



Student login :-

: alice

Password : 123

Mentor login :- Username

Username : preeti

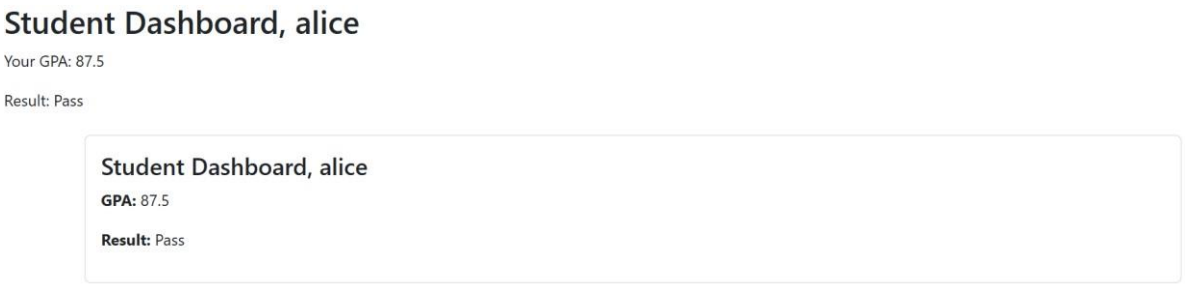
Password : 456

When a student logs in:

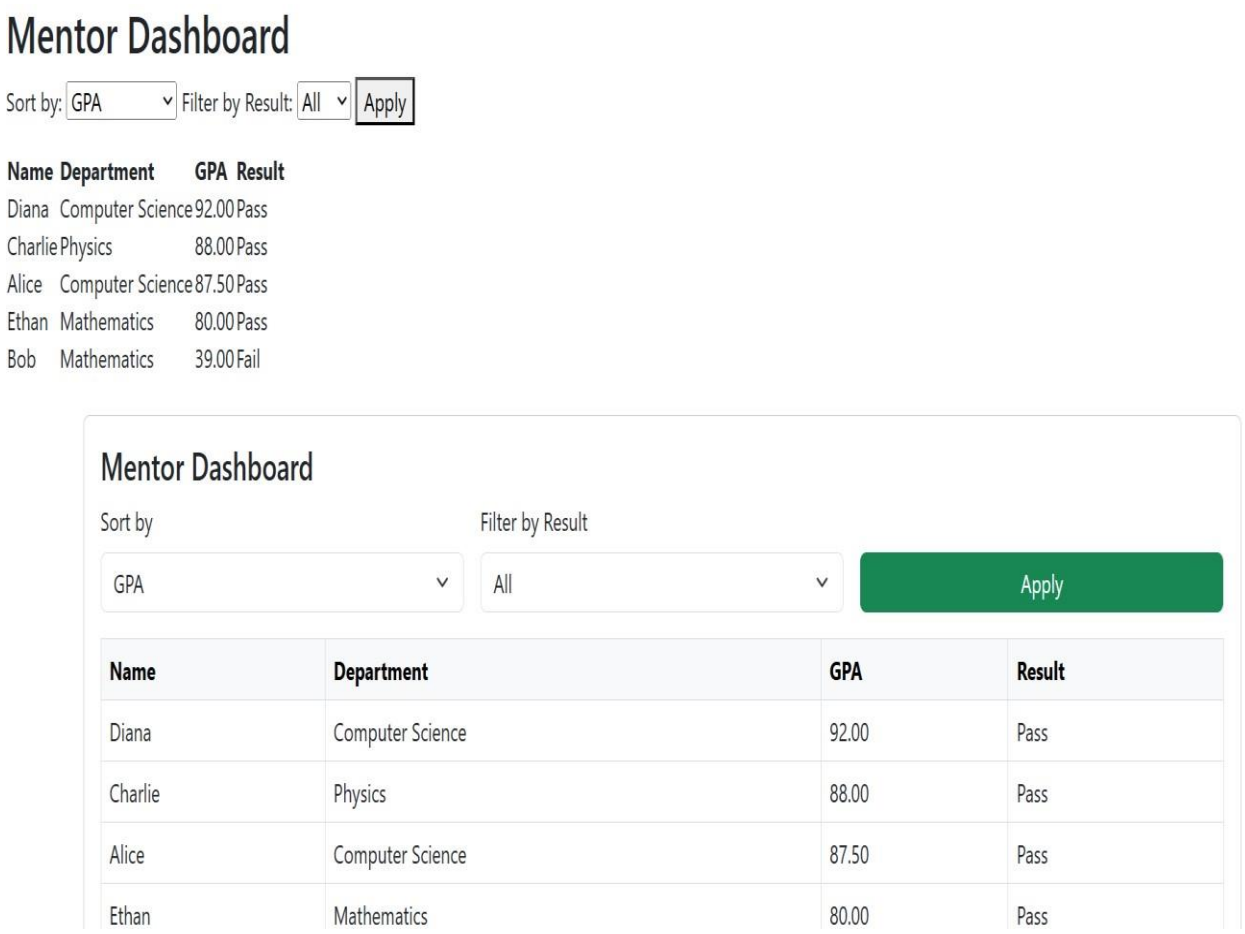
- We use the get_student_view() function.
- This function takes the student's ID and marks data to prepare:
    1. GPA → Student's overall grade performance.
    2. Result → Pass/Fail or similar status.

After login as the student we will get to the student dashboard or if we login in as a mentor username and password we will get to the mentor dashboard , dashboards looks like as follows :

Student Dashboard :



Mentor Dashboard :



When a mentor logs in:

- We use the get_mentor_view() function.
- Mentors usually need overview information for all students:

    1. Student Names
    2. Ranked List by GPA or Marks
    3. We can use a filters as

        - GPA , Name , Department wise
        - Students who have Passed , Failed and All

    4.By clicking apply our filter will apply to the dashboard and can view the results.