Fundamentals of Computing System Designs - CS540

Quiz Web Application -QuickQuiz

By Kashyap, Manvi, Nimra, Nomaan

Table of contents

01 Problem Statement

02 Methodology

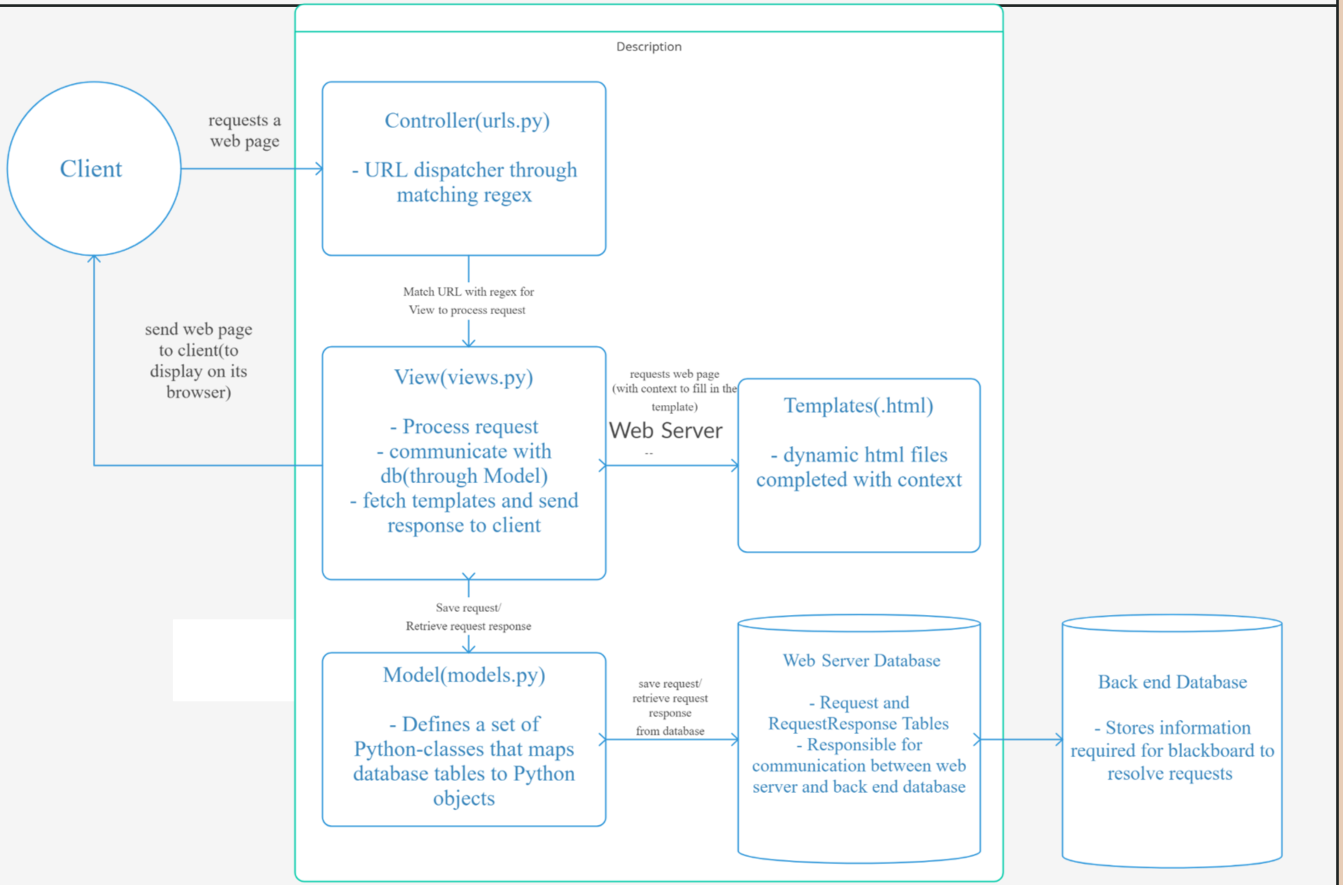
03 Tech Stack

04 Project Demonstration

Problem Statement

Design and develop a real-time, multi-user quiz application using socket programming. The application should allow multiple clients (students & teachers) to connect to a server, participate in quizzes, and receive real-time feedback on their answers. The server should manage quiz questions, broadcast questions to all connected students, and evaluate responses. Additionally, it should track scores for each student, and notify students of their performance after each quiz. The communication between the server and clients should be reliable, ensuring low-latency interactions. The application must handle concurrent connections, manage client disconnections gracefully, and provide a seamless user experience.

DJango App Architecture:



Description

Client

requests a web page

Controller(urls.py)

- URL dispatcher through matching regex

Match URL with regex for View to process request

View(views.py)

- Process request
- communicate with db(through Model)
- fetch templates and send response to client

send web page to client(to display on its browser)

requests web page (with context to fill in the template)

Web Server

Templates(.html)

- dynamic html files completed with context

Save request/ Retrieve request response

Model(models.py)

- Defines a set of Python-classes that maps database tables to Python objects

save request/ retrieve request response from database

Web Server Database

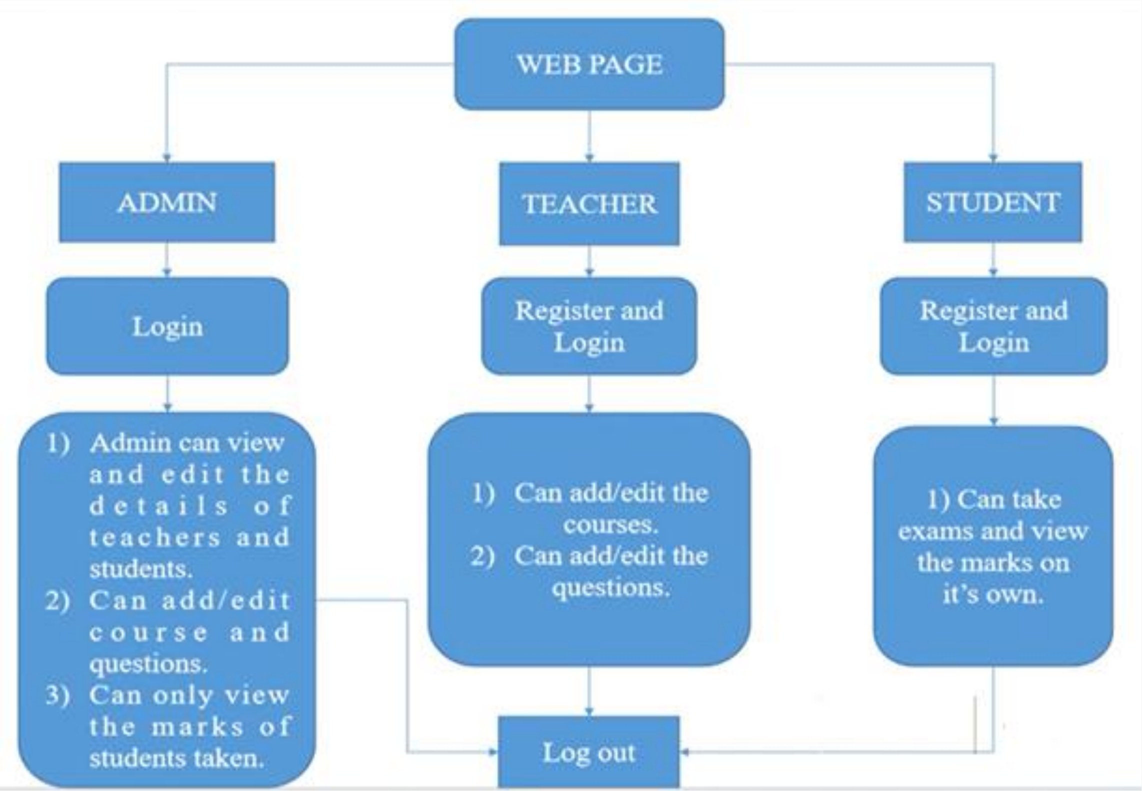
- Request and RequestResponse Tables
- Responsible for communication between web server and back end database

Back end Database

- Stores information required for blackboard to resolve requests

Methodology:

Application workflow



WEB PAGE

ADMIN

Login

1) Admin can view and edit the details of teachers and students.
2) Can add/edit course and questions.
3) Can only view the marks of students taken.

TEACHER

Register and Login

1) Can add/edit the courses.
2) Can add/edit the questions.

STUDENT

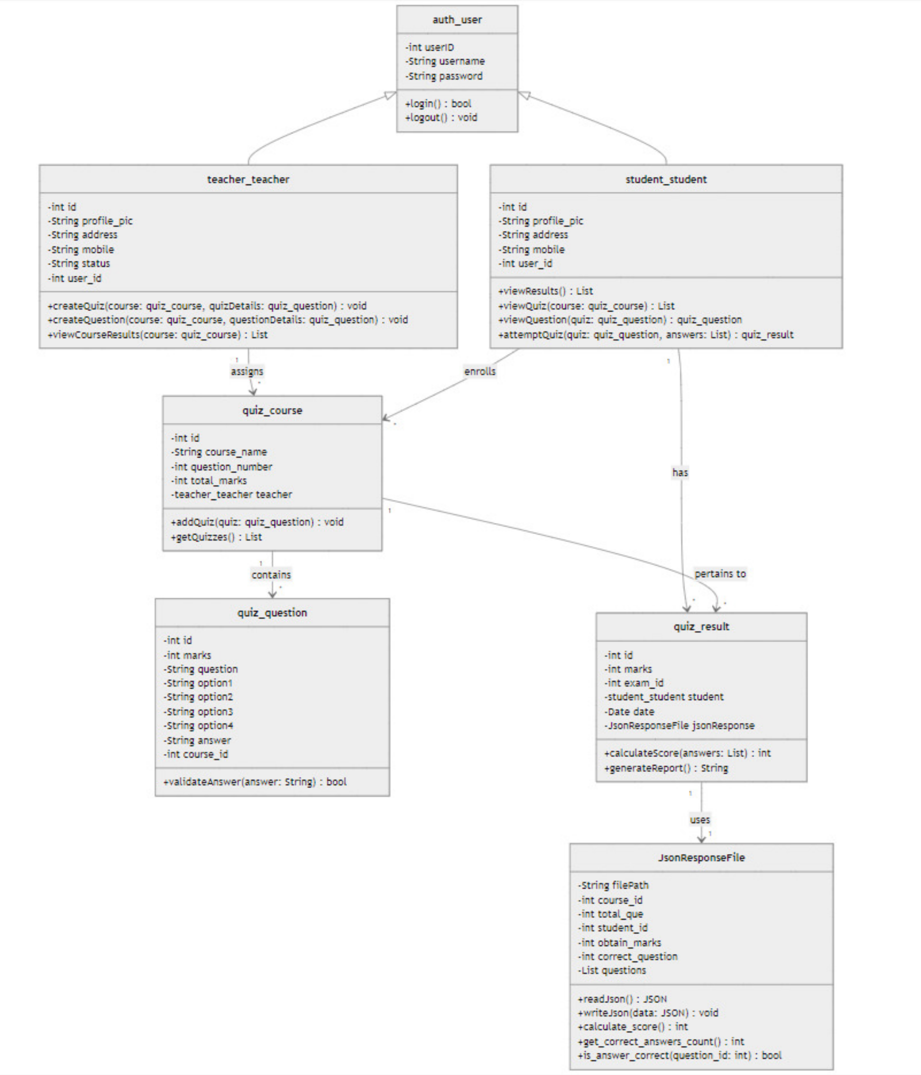
Register and Login

1) Can take exams and view the marks on it's own.

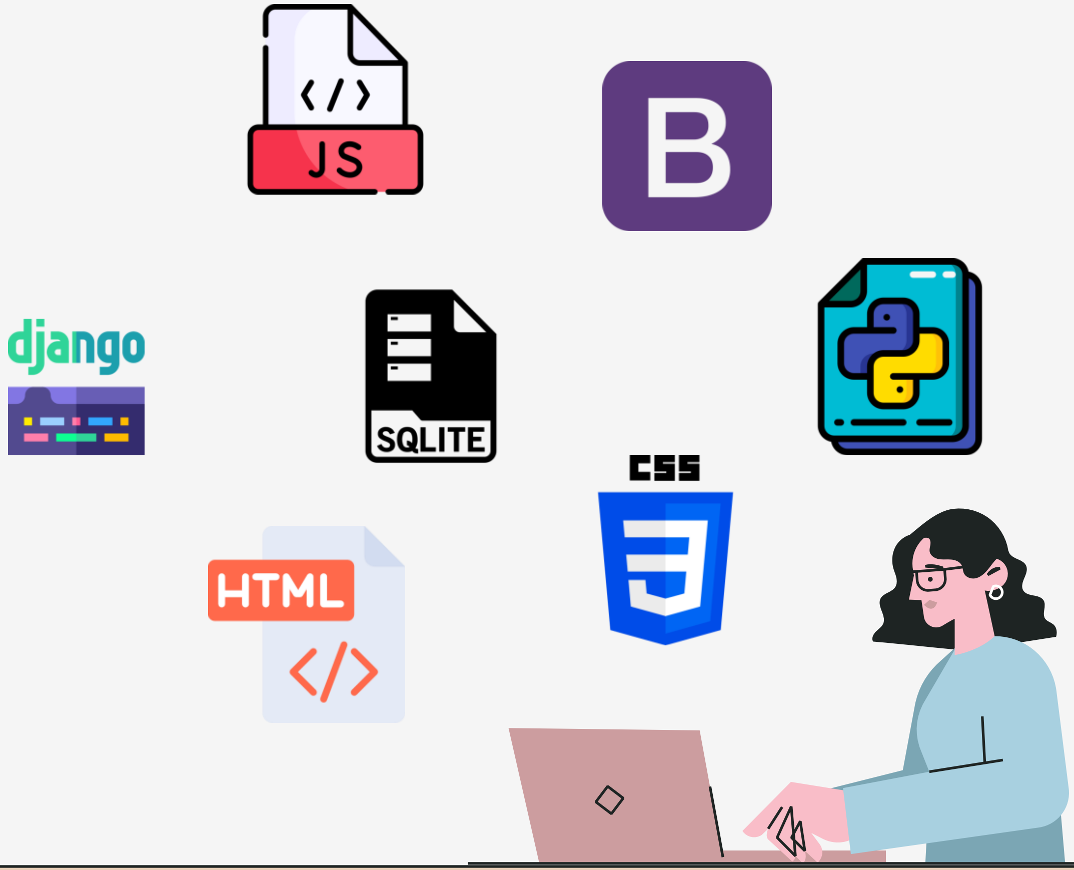
Log out

Methodology:

UML Diagram:



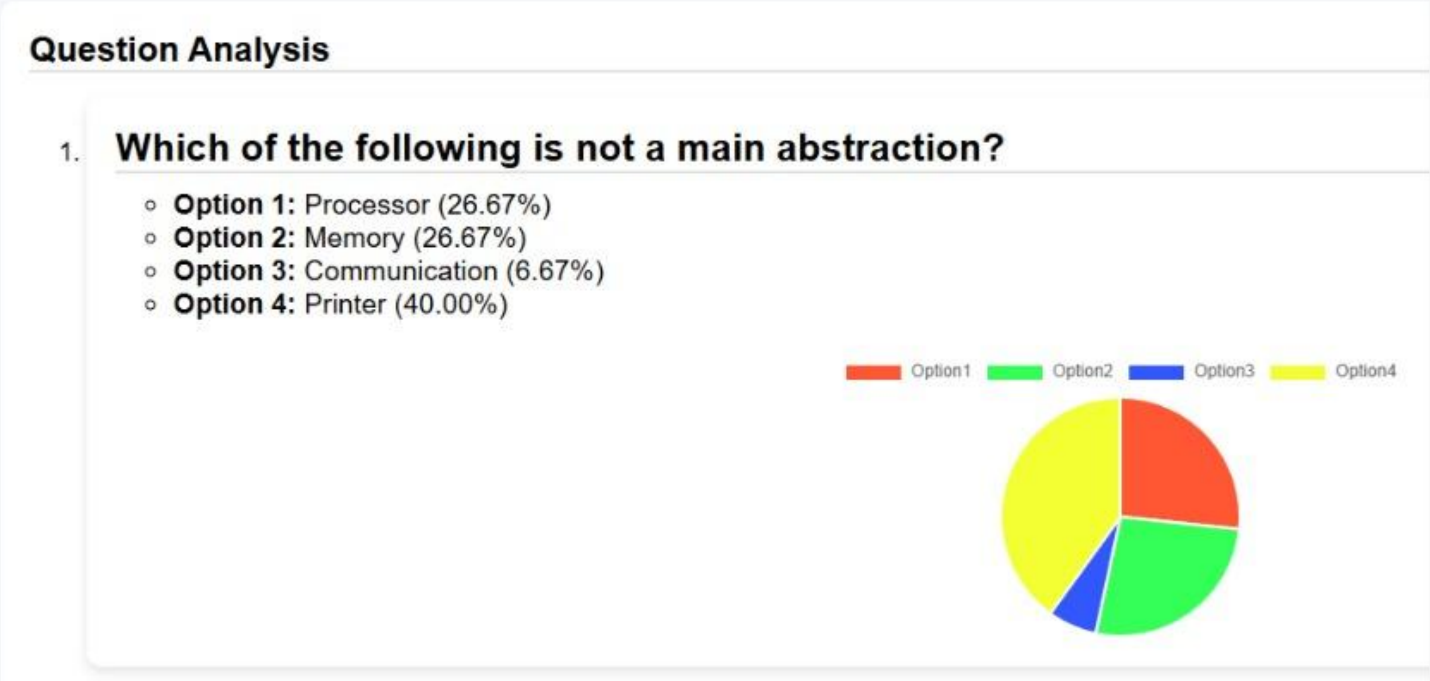
Tech Stack



Features:

- Supports MCQ question types
- Time-limited quizzes with auto-submit feature
- Role-based functionality for each user
- Class-wide performance analytics
- Detailed Individual Report Generation
- Leaderboard Rankings
- Admin approval for each new user

Demonstration

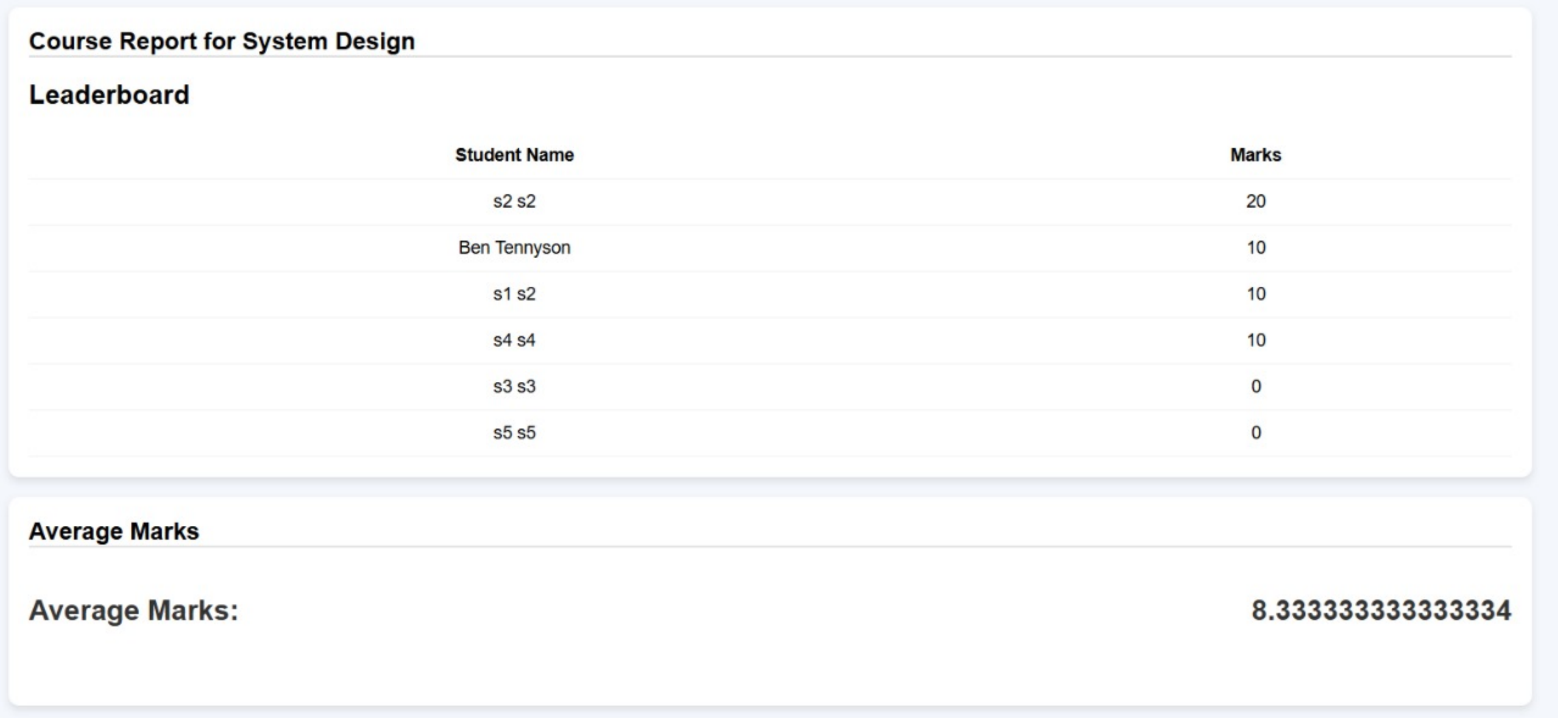


Question Analysis

1. **Which of the following is not a main abstraction?**

   - **Option 1:** Processor (26.67%)
   - **Option 2:** Memory (26.67%)
   - **Option 3:** Communication (6.67%)
   - **Option 4:** Printer (40.00%)

Demonstration



Course Report for System Design

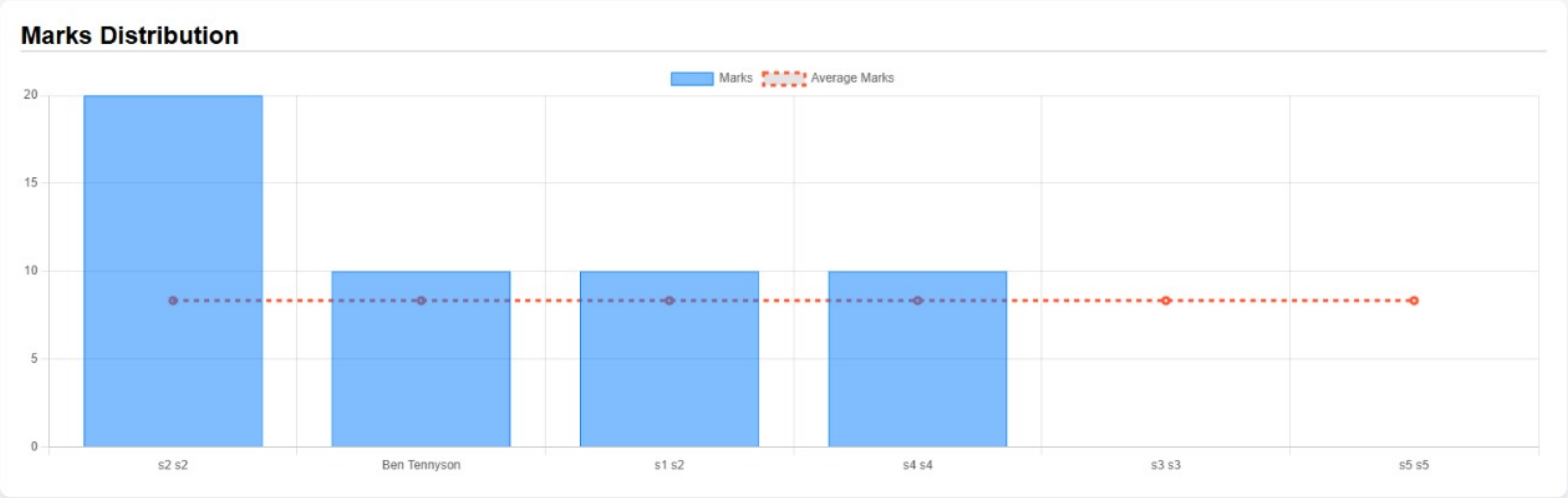
Leaderboard

Student Name	Marks
s2 s2	20
Ben Tennyson	10
s1 s2	10
s4 s4	10
s3 s3	0
s5 s5	0

Average Marks

Average Marks: 8.333333333333334

Demonstration



Marks Distribution

Contributions:

Kashyap & Nomaan worked alongside to develop & implement the following:

Framework: Implemented using Django

Database Management: Designed and managed a relational database (SQLite3)

Quiz Logic Implementation: Engineered back-end logic

Data Aggregation & Analytics: Implemented aggregation functions to generate class-wide performance graphs, MCQ answer distributions, and score analytics for instructors to view collective insights.

Security & Authentication: Used Django’s built-in authentication mechanisms for secure user login and session management, ensuring data privacy and secure access to quiz content.

Contributions:

Nimra & Manvi worked alongside to design & implement the following

Dashboard UI Design: Developed dashboards for admin, students and instructors using HTML, CSS, and JavaScript.
Visual Data Representation with Jinja: Utilized Jinja templating in Django to dynamically pass performance data and render interactive graphs on the front end.
Leaderboard Display: Created a real-time leaderboard interface, visually ranking students based on quiz scores and highlighting top performers.
Frontend-Backend Integration: Worked with the back-end team to integrate front-end, enabling smooth data flow for real-time updates on quiz results, performance graphs, and leaderboard positions.
Interactive Elements: Implemented JavaScript interactivity for elements like quiz timers, answer selection, and feedback displays to enhance the quiz-taking experience.

Thank You!