National Teachers College
School of Arts, Science &Technology
629 J. Nepomuceno, Quiapo,Manila

“SDG 4:Student Performance Tracker”

GROUP MEMBERS:
ABDUL, MUHAMMAD SIDDIQ M.
DOMINGO, JAYSON B.

Submitted to:
Justine Louise R. Neypes

Table of Contents

I. Introduction

Goal:United Nations sustainable development Goal (SDG) 4 - Quality Education.

1.1 Project Overview

The Student Performance Tracker is a Python-based web application built using Streamlit. It allows teachers or users to manage student records, add and update scores, display rankings, and track recent attempts using data structures such as Linked Lists and Stacks. The system stores data persistently using JSON files and provides visual analytics through charts. Its goal is to provide a simple, efficient, and structured way to manage academic performance.

1.2 SDG Target Alignment

This project aligns with SDG 4: Quality Education, specifically focusing on improving learning outcomes through digital tools. By providing automated tracking, performance visualization, and structured data management, the system supports inclusive and quality education and promotes lifelong learning opportunities.

1.3 Problem Statement

Managing student performance manually can lead to errors, data loss, inconsistencies, and difficulty in tracking progress. Teachers need a simple system that organizes students, stores scores securely, and visualizes performance. The lack of accessible, lightweight tools motivates the creation of a structured, automated solution.

1.4 Project Objectives

- To develop a functional system for storing and managing student records.
- To apply key Data Structures and Algorithms (Linked List, Stack, Search, Sorting).
- To support persistent data storage using JSON files.
- To provide ranking and visualization features for performance evaluation.
- To demonstrate a secure login mechanism within the application.

1.5 Scope and Limitations

Scope:

- Add, update, remove students
- Add, update, remove scores
- Rank students
- Display lists and attempts history
- Undo last score entry
- Login and recovery word feature

Limitations:

- Data stored only in local JSON files (no SQL database).
- Single password for login (no multi-user system).
- No advanced analytics beyond charts and ranking.

II. Requirements & Analysis

2.1 Functional Requirements (FR)

The following functional requirements define the expected behaviors and capabilities of the Student Performance Tracker system:

ID	Requirement
FR1	The system must store students using a Linked List structure.
FR2	The system must allow adding, updating, and removing students and scores.
FR3	The system must track recent attempts using a Stack.
FR4	The system must display rankings using sorting.
FR5	The system must save and load all records via JSON files.
FR6	The system must support user login and password recovery.

2.2 Non-Functional Requirements (NFR)

The following non-functional requirements describe the quality attributes and constraints that the system must satisfy:

ID	Requirement	Description
NFR1	Performance	Should handle 50+ students without lag.
NFR2	Usability	UI must be simple and intuitive.
NFR3	Reliability	Data must not be lost on refresh (persistent JSON storage).
NFR4	Security	Password gate protects the system.

2.3 Data Requirements

- JSON files store students' IDs, names, timestamps, and list of scores.
- Another JSON stores recent attempts (Stack).
- Each score contains: {type, score, timestamp}.

2.4 User Requirements

- Users must be able to add and manage students easily.
- Users must log in before accessing the system.
- Users must view rankings and performance charts.

2.5 Constraints and Assumptions

Constraints:

- The program requires Python and Streamlit.
- Runs only with active internet browser sessions.

Assumptions:

- Users enter valid score types.
- JSON files remain accessible in the project folder.

2.6 Complexity Analysis (Big O Notation)

Operation	Data Structure	Complexity
Add student	Linked List	O(n)
Find student	Linked List Search	O(n)
Add attempt	Stack push	O(1)
Undo attempt	Stack pop	O(1)
Sorting for ranking	Timsort sorting	O(n log n)
Adding Score	Linked List	O(1)

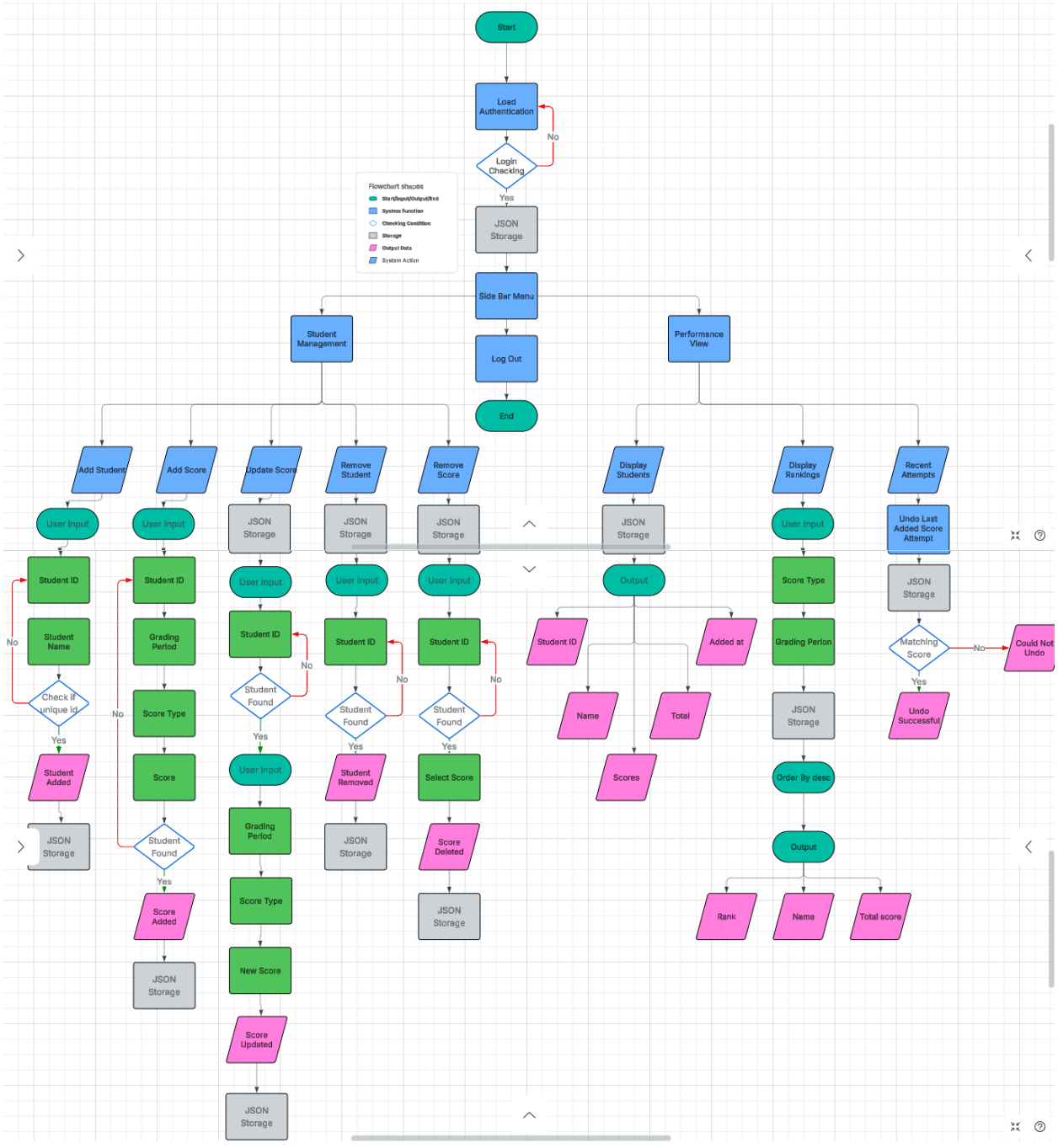
Based on our chosen data structures. Adding a student requires traversing the linked list**(StudentList)** to ensure the uniqueness of every node in the list; this results in **O(n)** time complexity. Searching for a student also performs a linear traversal, resulting in **O(n)** time complexity.

Adding(push()) and popping(pop()) elements to the Stack(Stack) Container result in O(1) time complexity since the stack only does removing and adding elements from the top of the container. The code implemented in sorting for ranking is (Timsort) which combines merge sort and insertion sort. This results in O(n log n) time Complexity. Adding a score to student is O(1) time complexity once the student is found in the list, it will append the new score to their score list,

III. System Design Specification

3.1 System Flow / Flowchart

The flowchart represents the workflow of the Student Performance Tracker application, showing the operation from login to performing various student management and performance tasks.



Start & Authentication

- The system begins at the Start node and first loads authentication data from a storage file.
- A login check is performed. If the credentials are correct, the system proceeds; otherwise, it will not proceed, requiring correct authentication before accessing the main interface.

Main Interface / JSON Storage Access

- Upon successful login, the system loads existing data from **JSON storage**, which contains student records and score history.
- The user is presented with the **Sidebar Menu**, which offers options for student management, performance views, and logging out.

Student Management Module

This module handles CRUD operations for student records and scores:

- **Add Student:** The system requests student ID and name, checks if the ID is unique, adds the student, and updates JSON storage.
- **Add Score:** The system collects the student ID, grading period, score type, and score value, verifies the student exists, and adds the score to the student record.
- **Update Score:** The system verifies the student exists, collects the new score, updates the relevant score entry, and saves changes to JSON storage.
- **Remove Student / Remove Score:** The system identifies the student and either deletes the student record or removes a selected score, updating JSON storage accordingly.

Performance View Module

This module focuses on analyzing and displaying student performance:

- **Display Students:** Shows all student records with their scores and total performance.
- **Display Rankings:** Computes total scores based on selected grading period and score type, sorts students in descending order, and outputs their rank, name, and total score.
- **Recent Score Attempts & Undo:** Displays the last added scores stored in a stack, allowing the user to undo the last score entry if needed. The system checks for a matching record and removes it from both the student record and the stack.

You may visit our Flowchart by clicking the link below:
https://lucid.app/lucidchart/b8847d26-7c51-4059-a77d-941db593a5f7/edit?viewport_loc=-674%2C677%2C3632%2C1647%2C0_0&invitationId=inv_120d293e-c0b2-4555-bdb7-398e40a2eea6

3.3 Core Data Structures Used

3.3.1 Prelim DSA Concept (Linked List)

The linked list was chosen to represent the collection of students because it provides a simple, flexible, and memory-efficient way to model an unknown or changing number of records without preallocating space.

In the context of the Student Performance Tracker, each student is naturally represented as a node containing fields for ID, name, timestamps, and a list of scores; the pointer-based structure of a linked list mirrors this node-centric organization and makes adding or removing students straightforward. Traversal through the list supports operations such as searching for a student, aggregating scores, and building views for reporting.

While random-access operations are O(n), typical class-management tasks (append, remove by ID, sequential display, and incremental updates) are frequent and are handled comfortably by a linked list, keeping the implementation conceptually simple and easy to reason about for maintenance and grading.

Implementation:

The linked list is implemented using two custom classes Student and StudentList.

- The Student class represents each node, containing the student ID, name, list of scores, and a pointer next referencing the next student node.
- The StudentList class manages the linked list operations, including insertion and traversal. The insertStudent() function creates a new Student object, assigns the data provided by the user, and attaches it to the end of the list by updating the next pointer of the last node.

  Insertion works by creating a new Student node and adding it either as the head (if the list is empty) or at the end of the list by moving from one node to the next until the final node is reached. Traversal is performed in functions that display or process student information, where the system moves through each node starting from the head.

3.3.2 Midterm DSA Concept (Stack)

A stack was implemented to track recent attempts and provide an undo feature because stacks naturally implement Last-In-First-Out semantics, which exactly match the expected behavior for undoing the most recent user action.

Each push stores a compact representation of an operation (student ID, score details, timestamp) so that a single pop can reverse that change deterministically. The stack enforces a bounded history (a fixed capacity) so memory usage remains predictable, and its O(1) push/pop performance ensures undo and history-display operations are instantaneous from a user perspective.

This makes the system robust against accidental inputs and supports a user-friendly workflow, while keeping the implementation small and verifiable — important for both classroom demonstration and automated testing.

Implementation:

The stack is implemented through the **Stack class**, which is used to store the most recent score entries added by the user. Its main purpose is to enable the **"undo last score"** functionality in the Student Performance Tracker. The stack ensures that the system can quickly remove the most recent score added if the user wants to reverse an action, following the **Last-In, First-Out (LIFO)** principle.

The **Stack class** contains three main components in your code:

- **Data** – a list that stores the score attempt records as dictionaries. Each record includes the student ID, name, grading period, score type, score value, and timestamp.
- **Push()**– adds a new score record to the stack. If the stack has reached its limit (5 entries), it removes the oldest entry before adding the new one.
- **Pop()** – removes and returns the most recent score record, which allows the system to undo the last score entry.
- **Display()** – shows all recent score attempts in the interface, letting the user see the latest actions.

This stack implementation directly supports tracking user actions in real-time and allows undoing mistakes efficiently, making it a perfect example of a Midterm DSA concept applied in our project.

3.3.3 Final DSA Concept (Sorting)

Sorting was selected as the Finals concept to compute student rankings because generating ordered lists by aggregated metrics (total or filtered scores) is a fundamental analytics operation that directly supports the project's main functionality.

The implementation leverages the language's efficient comparison-based sorting routine to order students by computed totals and produce stable, reproducible rank lists. Sorting converts the node-based student collection into an array or list for efficient comparison and ordering; although the conversion costs O(n), the overall cost of sorting dominates as O(n log n).

The choice of sorting provides a clear, testable output (rank tables and bar charts), demonstrates algorithmic reasoning (complexity and stability), and gives instructors an easy way to validate correctness and performance.

Implementation:

The sorting is used to generate **student rankings** based on their total scores for a specific grading period and score type. Its main purpose is to organize student data in descending order of performance, making it easy to display rankings and charts in the interface.

- The linked list of students **(StudentList)** is first **converted into a temporary list** to allow sorting.
- Each student's **total score** is computed by summing the relevant scores according to the selected period and type.
- The list of students is then **sorted in descending order of total scores** using a custom comparator. This produces a clear ranking from highest to lowest.
- The sorted results are displayed as a table and chart without altering the underlying linked list structure, preserving the integrity of the main data.

This implementation of non linear data structure is a perfect example of a Final DSA concept applied in our project.

IV. System Implementation

4.1 Development Environment and Tools

The system is implemented using **Python 3** as the programming language, with **Streamlit** for the interactive web-based interface. **Pandas** is used for data manipulation and tabular display, while **Altair** provides charting and visualization capabilities. **JSON** files are used for persistent storage of student records and score history, ensuring that data is retained between sessions.

4.2 System Modules and Features

- **Authentication Module:** Manages user login and password recovery to secure access.
- **Student Management Module:** Handles adding, updating, and removing student records.
- **Score Management Module:** Allows users to add, update, and remove scores for each student.
- **Ranking Module:** Computes and displays student rankings based on total scores, with sorting for performance comparisons.
- **Undo System:** Tracks recent score additions using a stack, allowing users to undo the last action.
- **Data Visualization Module:** Generates tables and bar charts to display student scores and rankings clearly.

4.3 File Structure (GitHub Repository)

The github repository is provided below you can check out the full implementation of our project https://github.com/karasmaiAS/Pro-grammerzzz-sdg4-dsa.git

V. Testing & Evaluation

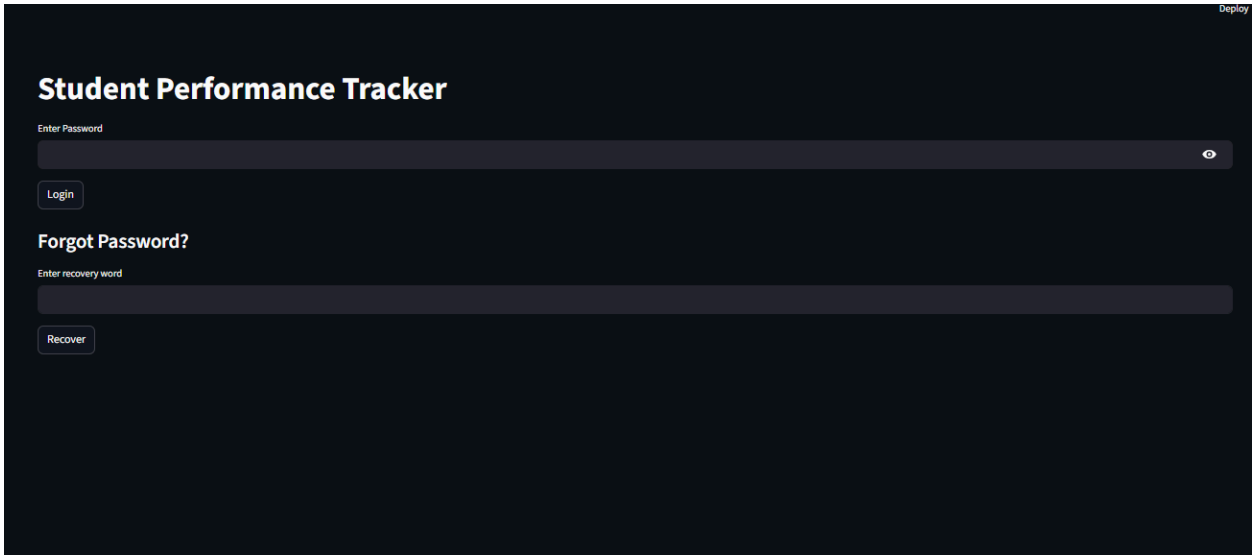
5.1 Test Plan

Test essential functions:

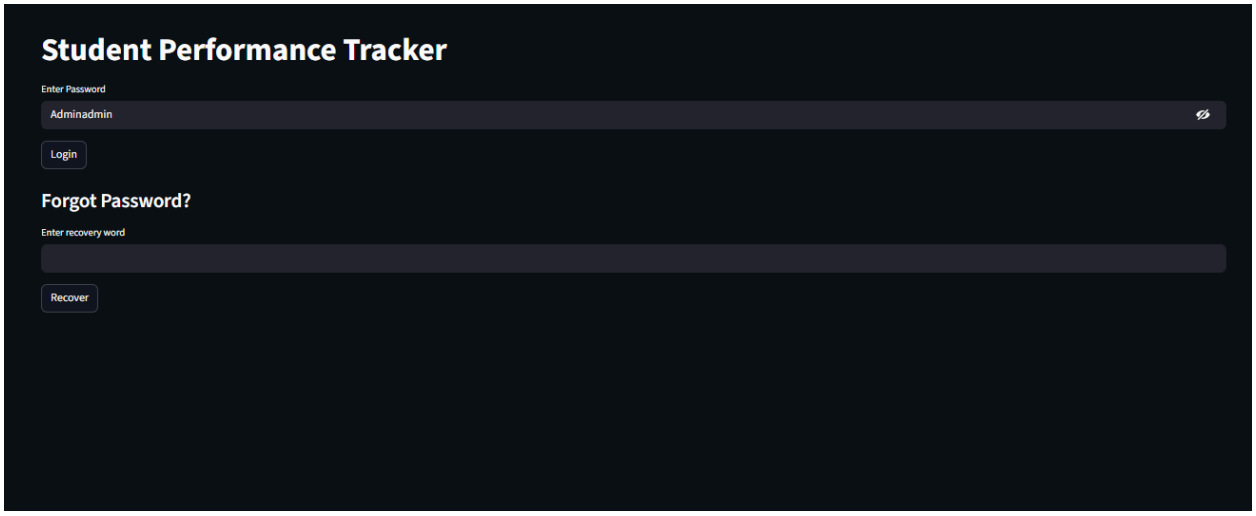
- **Authentication** - The user must try to enter an incorrect and correct password, and recover password by recoveryword.
- **Add Student** - The user must successfully add a new student and try adding a student with the same student id.
- **Add score** - The user must successfully add a new score of student, and try adding score with the same student id, score type, grading period.
- **Update score** - The user must successfully update the student's current score and check if the student's score is successfully updated.
- **Remove score** - The user must successfully remove an existing student's score on every score type.
- **Remove Student -** The user must successfully remove existing students in the list.
- **Undo Recent Score Attempt -** The user must successfully perform an undo recent score.
- **Display Ranking -** The user must Check every Grading period ranking if it's workingly fine.

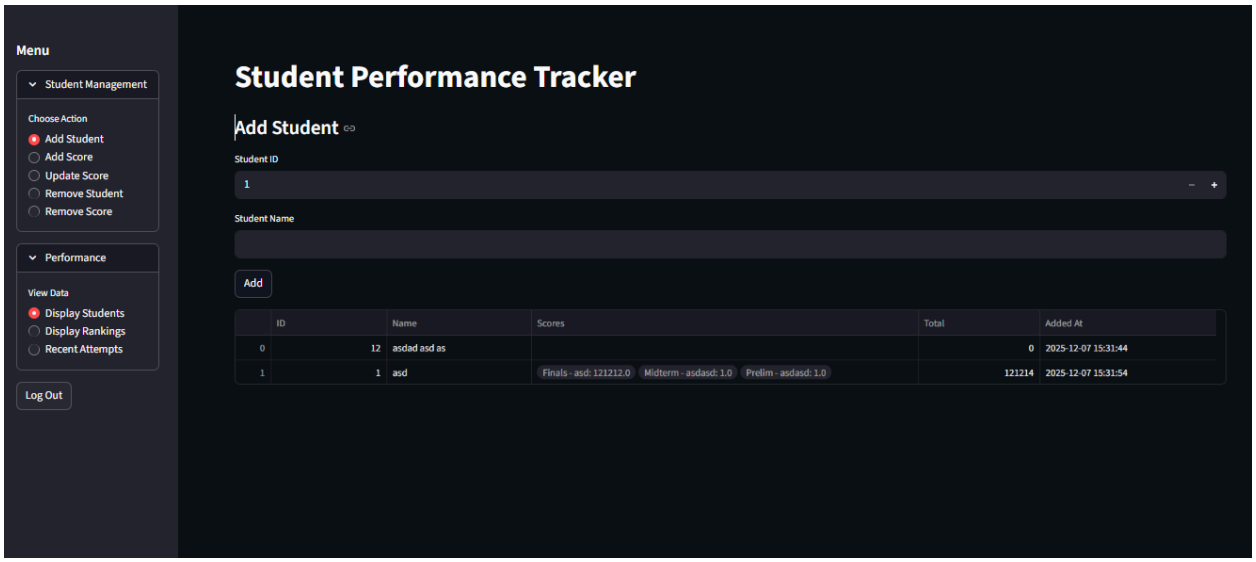
5.2 Test Cases and Results

Authentication - (Passed)

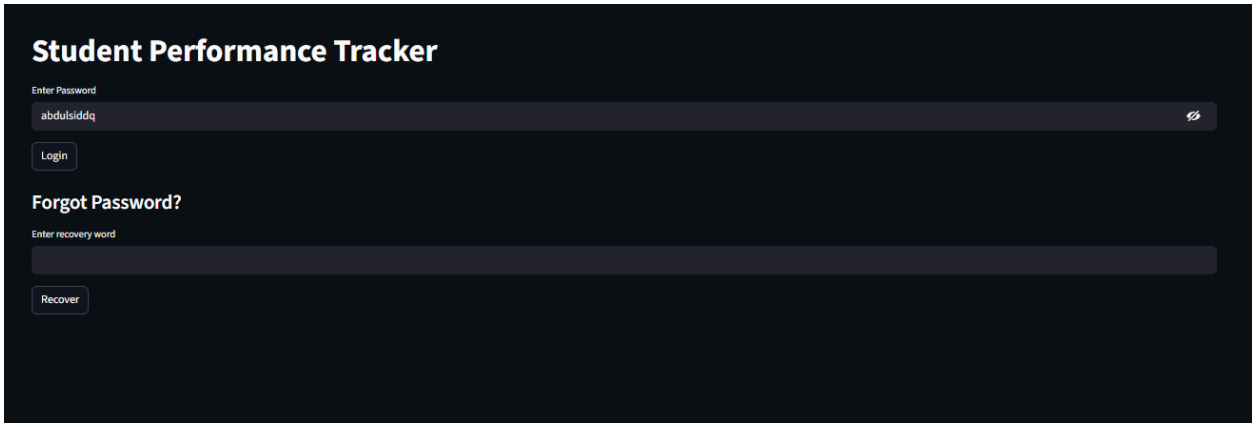


Entering Correct Password

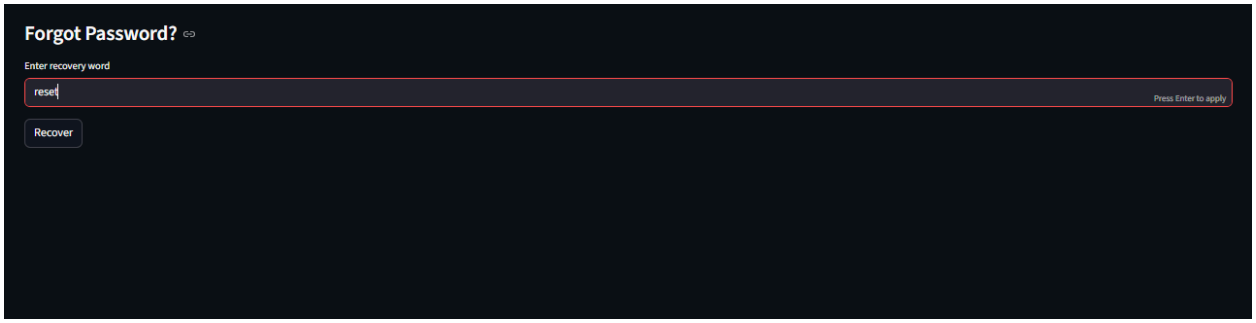




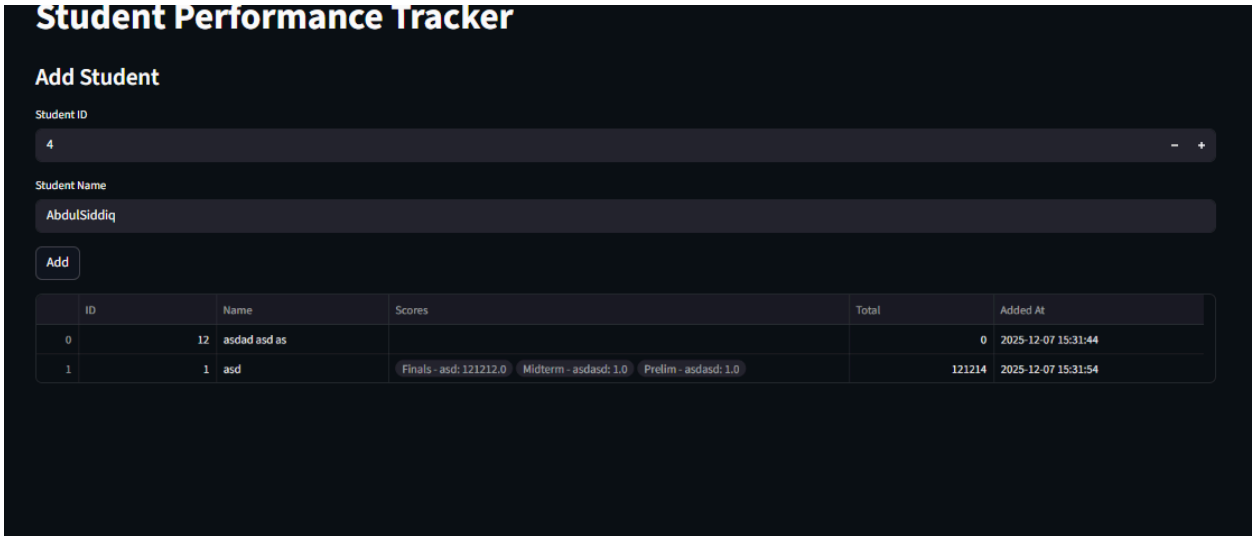
Entering incorrect password

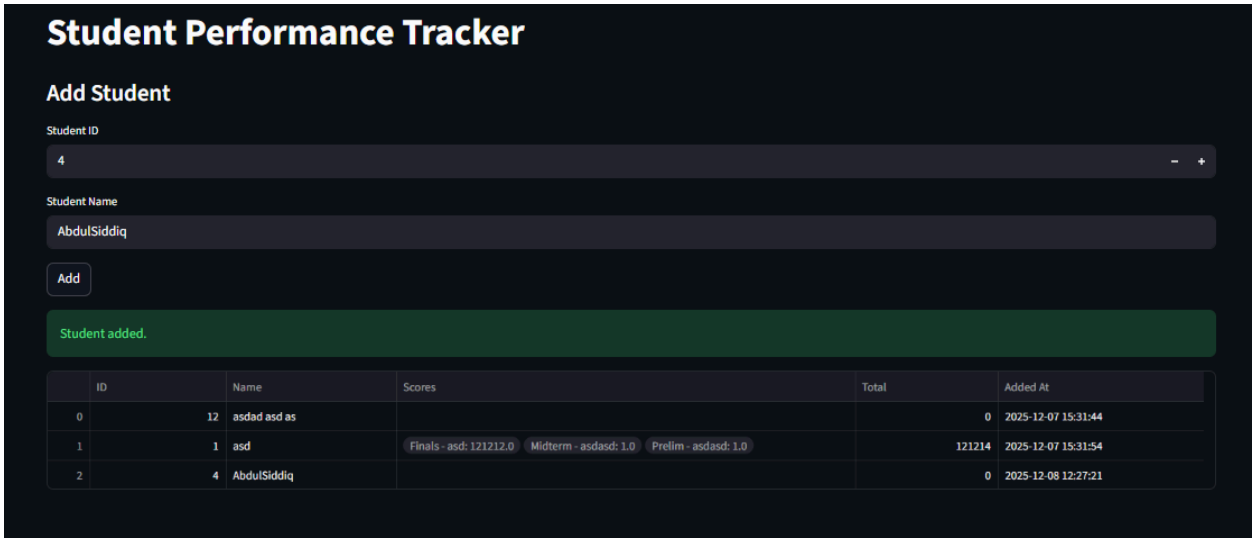


Entering RecoveryPassword

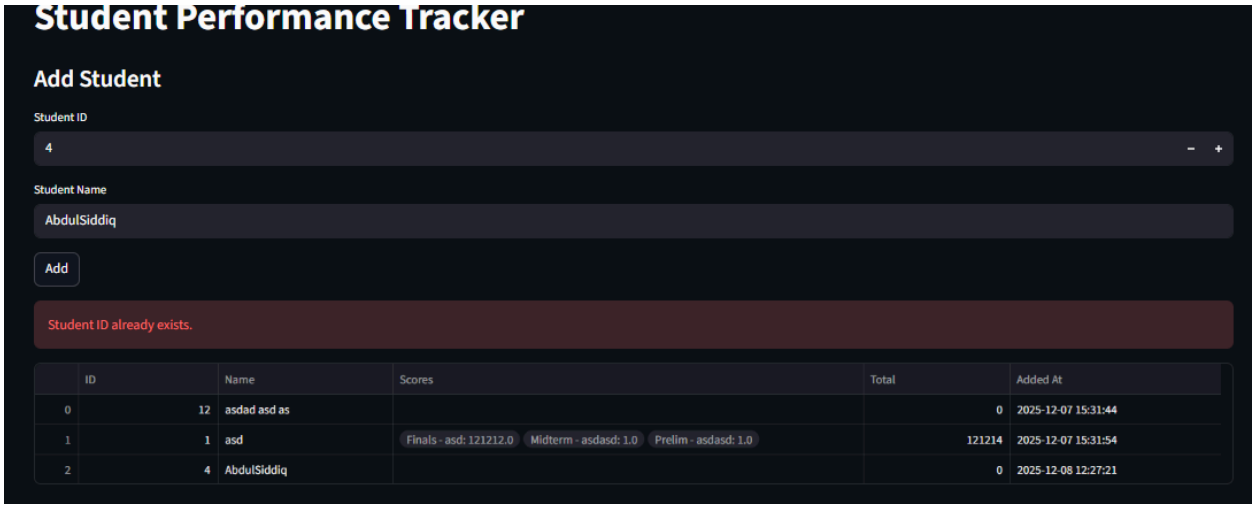


Add Student - (Passed)



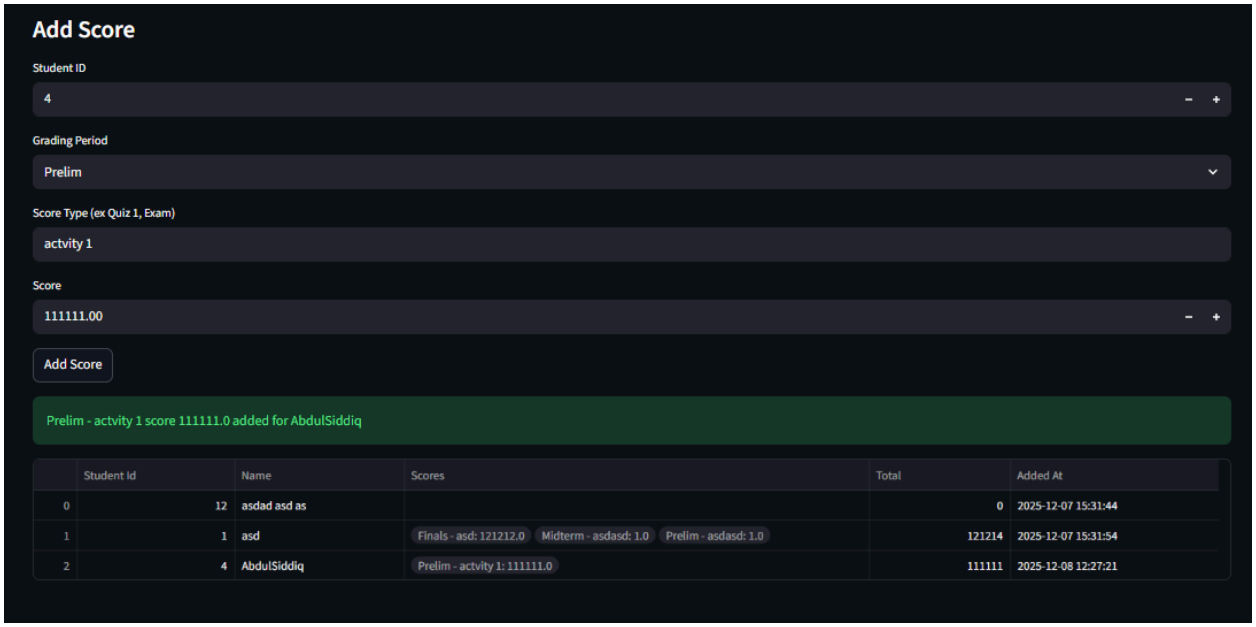


Adding New Student with Existing student id

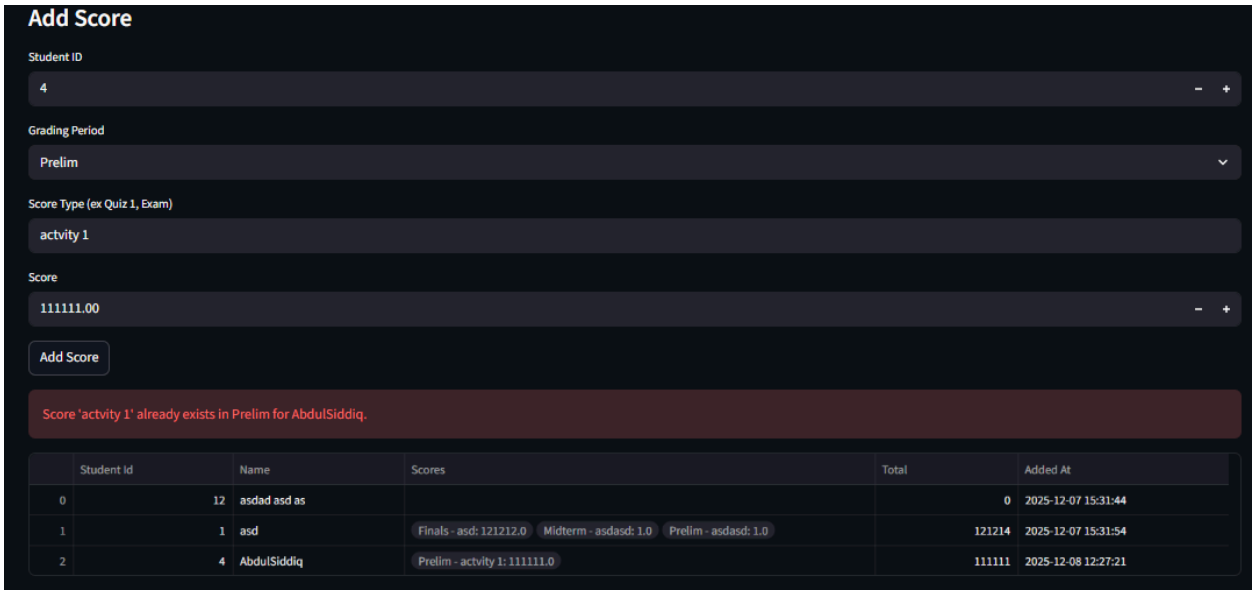


Add Score - (Passed)

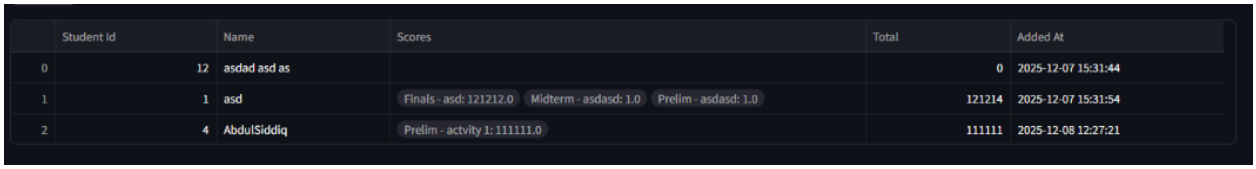
Adding New Score



Adding new score with the same grading period and score type

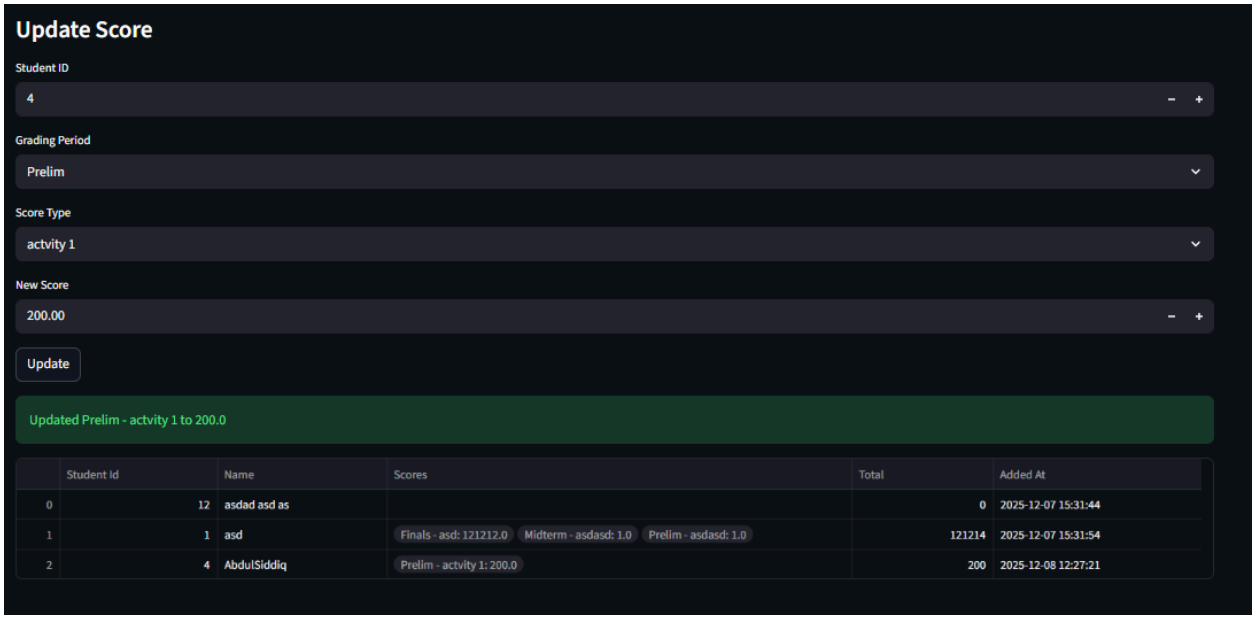


Update Score - (Passed)



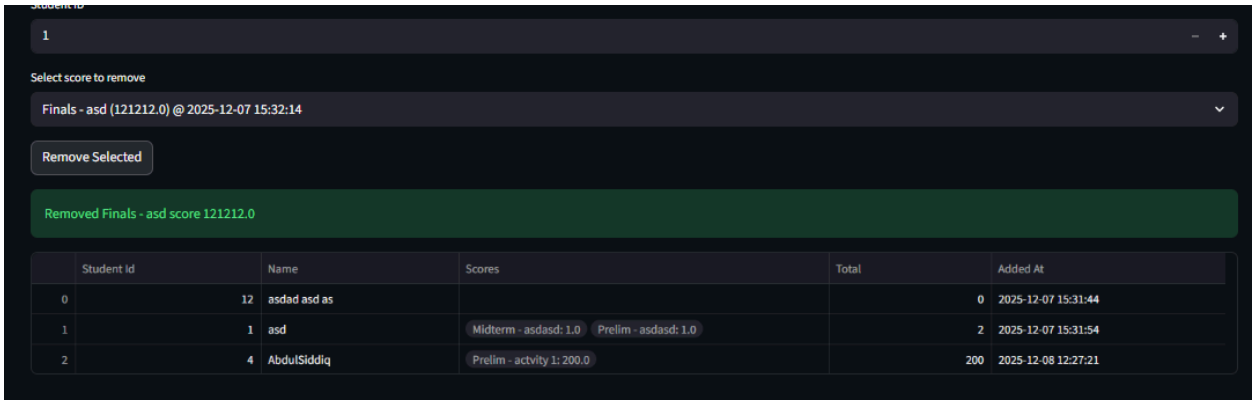
	Student Id	Name	Scores	Total	Added At
0	12	asdad asd as		0	2025-12-07 15:31:44
1	1	asd	Finals - asd: 121212.0 Midterm - asdasd: 1.0 Prelim - asdasd: 1.0	121214	2025-12-07 15:31:54
2	4	AbdulSiddiq	Prelim - actvity 1: 111111.0	111111	2025-12-08 12:27:21

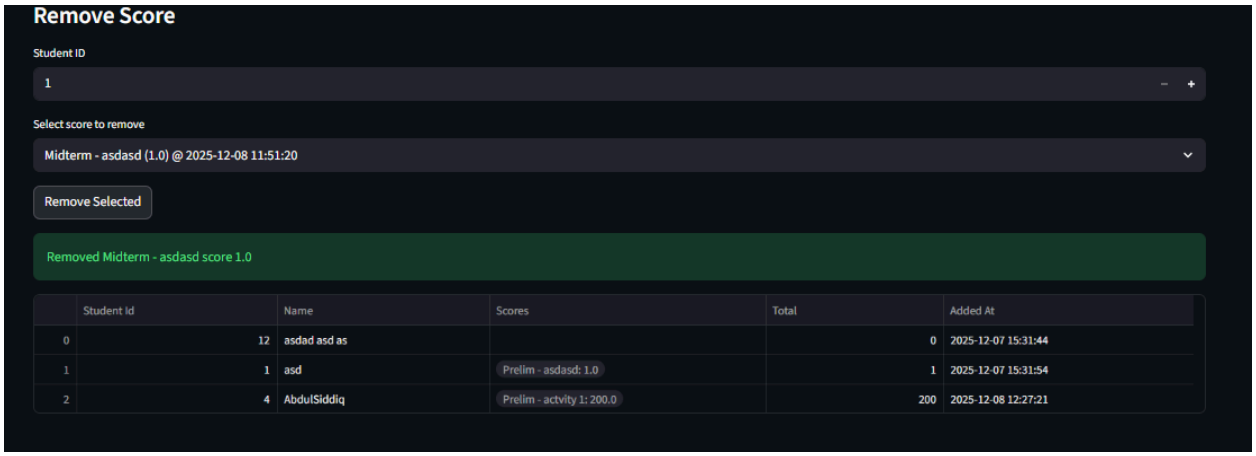
Updating Student's Score

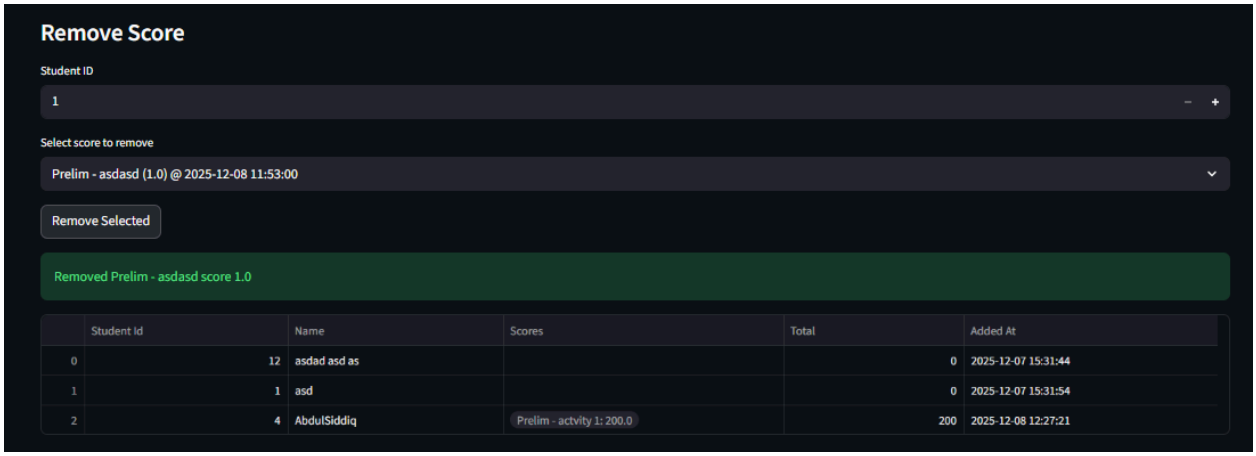


Remove Score - (Passed)

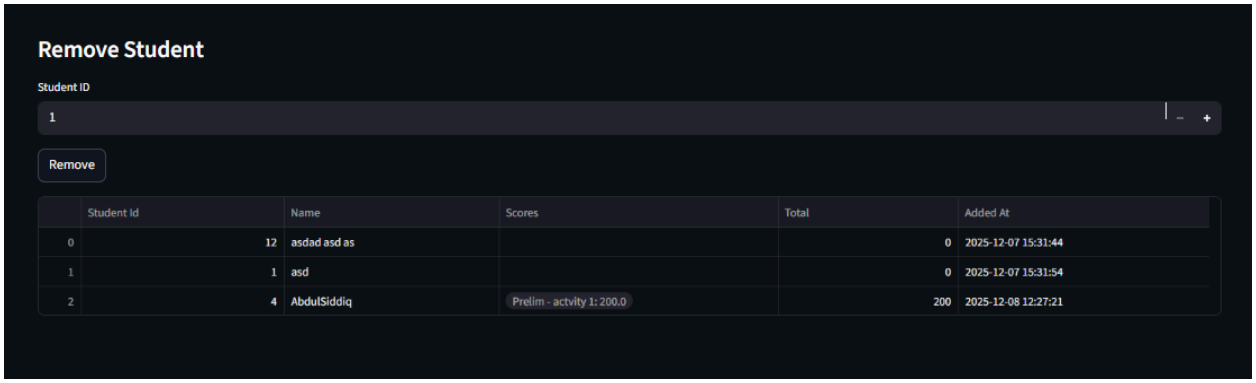
Removing every existing score of specified student



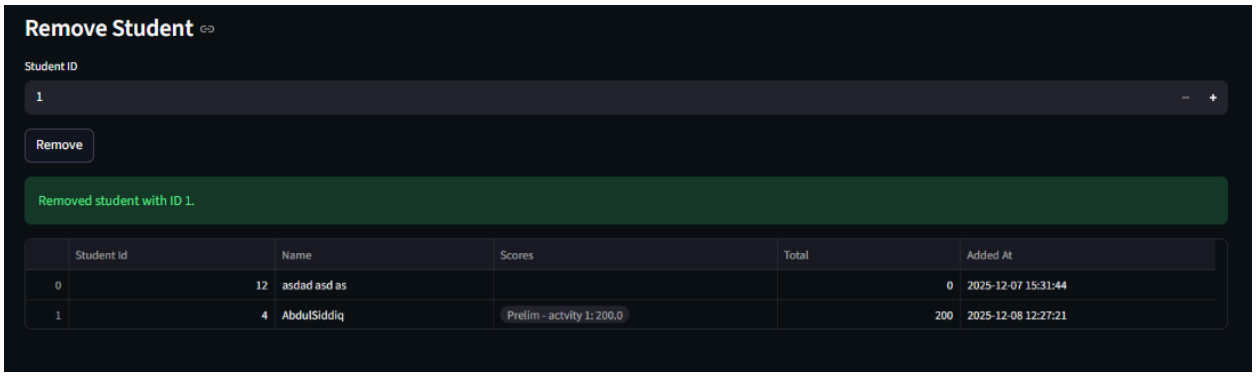




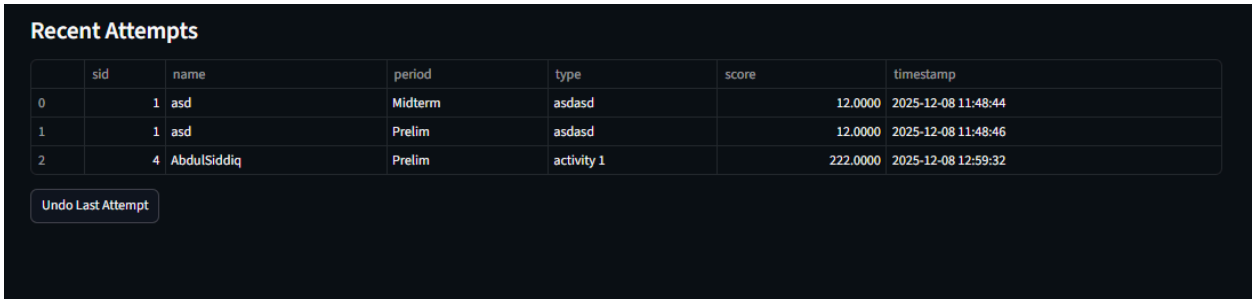
Remove Student - (Passed)



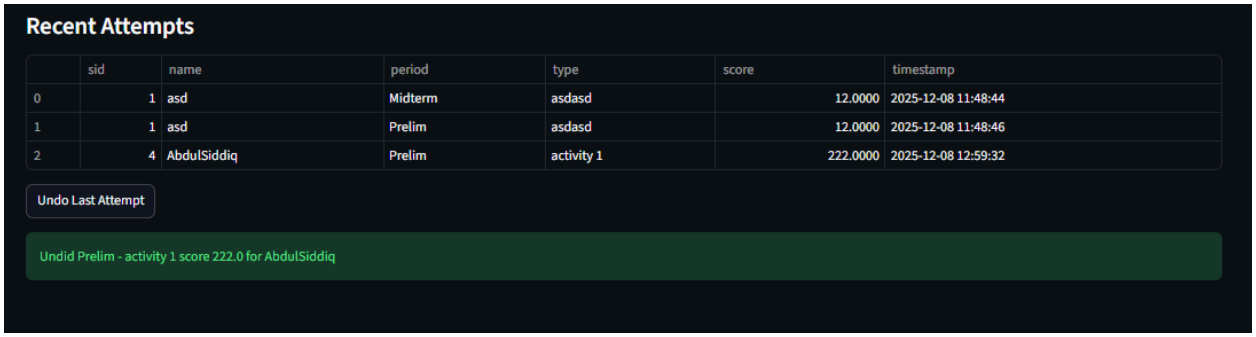
Removing Existing Student

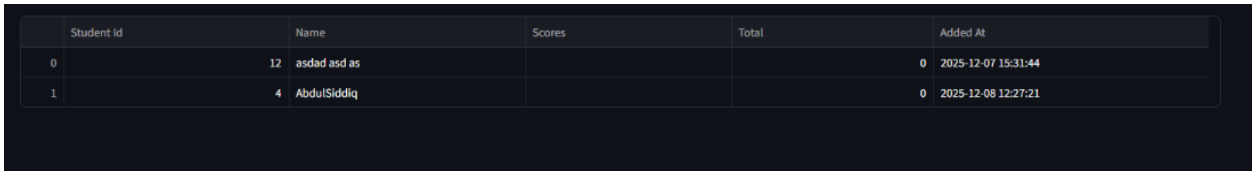


Undo Recent Score Attempt - (Passed)



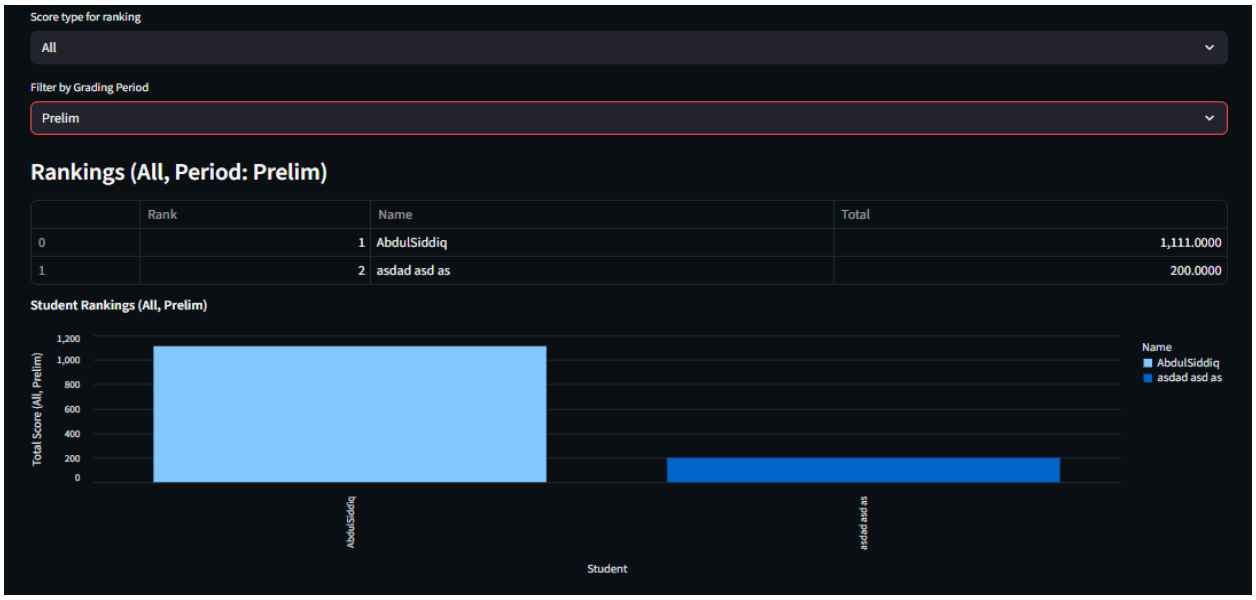
Undoing last attempt

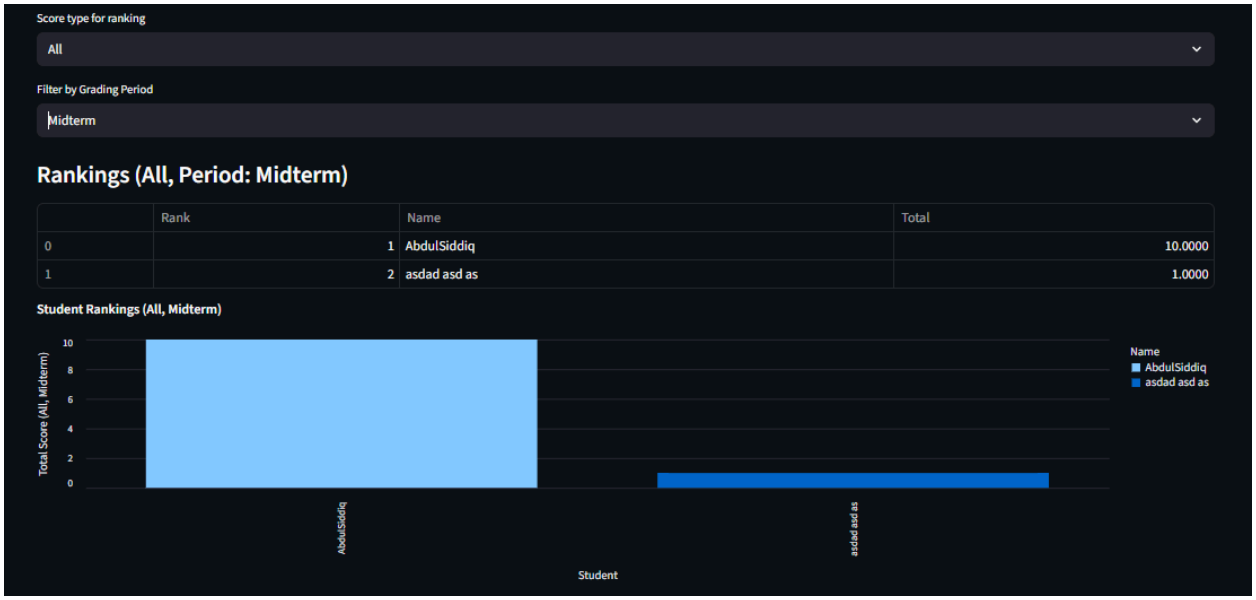




Display Ranking - (Passed)

added new score for existing student




Score type for ranking
All
Filter by Grading Period
Midterm
Rankings (All, Period: Midterm)
Rank
Name
Total
0
1
AbdulSiddiq
10.0000
1
2
asdad asd as
1.0000
Student Rankings (All, Midterm)
Total Score (All, Midterm)
10
8
6
4
2
0
AbdulSiddiq
asdad asd as
Student
Name
AbdulSiddiq
asdad asd as


Score type for ranking
All
Filter by Grading Period
Finals
Rankings (All, Period: Finals)
Rank
Name
Total
0
1
AbdulSiddiq
50.0000
1
2
asdad asd as
3.0000
Student Rankings (All, Finals)
Total Score (All, Finals)
50
40
30
20
10
0
AbdulSiddiq
asdad asd as
Student
Name
AbdulSiddiq
asdad asd as

VI. Results & Discussion

6.1 Analysis of System Behavior

The system demonstrates efficient performance in managing student scores, generating rankings, and maintaining data integrity. The score handling and ranking module accurately computes totals and orders students by performance, allowing users to clearly assess academic standing.

The **Linked List** structure contributes significantly to system flexibility by enabling dynamic insertion, searching, and removal of student records without needing to reorganize the entire dataset. Meanwhile, the **stack-based undo feature** provides a dependable and intuitive mechanism for reversing score-related actions, improving user confidence and reducing accidental data errors. Overall, the system functions smoothly and consistently across all modules, aligning well with the intended design goals.

6.2 Limitations and Constraints

Despite its functionality, the system has several limitations. It currently supports only a **single-user login**, which restricts its use in multi-admin or multi-teacher environments. Data is stored using **JSON**, which is efficient for small projects but vulnerable to corruption if users manually edit the file outside the system. Additionally, the reliance on **Streamlit** introduces scalability constraints, as very large datasets may lead to slow performance or interface lag due to Streamlit's real-time refresh behavior.

VII. **Conclusion & Distribution**

The Student Performance Tracker is a tool that efficiently and effectively meets its objective of providing an organized system for the observation and evaluation of student performance across various grading periods. The system shows a real-life and efficient usage of fundamental DSA concepts by the combination of three main data structures - Linked List, Stack, and Sorting. The Linked List is used for the management of student profiles and their performance data in a way that is flexible, the Stack is used for the undo mechanism of recent scoring operations in a safe way, and the sorting algorithm is used for the most up-to-date and clear performance rankings.

Building on Python and Streamlit, the tool delivers an intuitive and user-friendly interface, thus allowing free movement among the modules of Password authentication, score tracking, performance viewing, and ranking visualization. JSON-based data persistence makes performance data available across different sessions, thus establishing the system as a go-to tool for academic monitoring on an ongoing basis.

The project is limited to some extent by factors such as single-user access and limited scalability because of JSON and Streamlit, but it performs well in scenarios of small to medium-scale use where performance tracking is the main focus.

In summary, the Student Performance Tracker is a better choice for the three chosen DSA(linked list, stack, sorting) creating a simple tracker for SDG - 4 (quality Education).

Distribution

-

Name	**Individual Contribute**
Abdul, Muhammad Siddiq M.	(Github Repository Submission), (75% of Documentation), (Midterm, Final DSA Code Implementation),(Academic Format Document Implementation), (Power Point Presentation)
DOMINGO, JAYSON B.	(25% of Documentation), (Prelim DSA Code Implementation)

VIII. **References**

https://github.com/saadahmadrana/Student-Performance-Tracker

https://github.com/AmirhosseinHonardoust/AI-Personal-Study-Tracker

https://www.w3schools.com/python/pandas/pandas_intro.asp#:~:text=What%20is%20Pandas%3F,by%20Wes%20McKinney%20in%202008

https://www.geeksforgeeks.org/data-science/introduction-to-altair-in-python/

https://www.geeksforgeeks.org/python/a-beginners-guide-to-streamlit/