GROUP

EduTrack

COSC 612 / AIT 624

SPRINT 4

Group Members:

Naga Dhanushya Ram Munnanuru
Stephen Aboagye-Ntow
Muhammad Adam
Ayandayo Adeleke
Ravinder Maini

Towson University
Fall 2025

TABLE OF CONTENTS

1. Planning and Scheduling

Assignee	Email	Task	Duration	Dependency	Due Date
Naga Dhanushya Ram Munnanuru	nmunnan1@students.towson.edu	Implementation (Frontend and Backend)	8hrs	System Modeling	11/14/25
Muhammad Adam	madam2@students.towson.edu	System Modeling (Architecture Modeling, Behavioral Modeling)	6hrs	Feedback from Sprint A3 and Problem Statement	11/12/25
Ayandayo Adeleke (Coordinator)	aadelek7@students.towson.edu	Planning, Scheduling Report Formatting, Communication and Collaboration	6hrs	All tasks	11/17/25
Ravinder Maini	rmaini1@students.towson.edu	Testing - Creating Testing environment and Implementing Test Cases	8hrs	Sprint A3, Implementation and System Modeling	11/16/25
Stephen Aboagye-Ntow	sabaogy1@students.towson.edu	Testing - Creating Test Cases	6hrs	Implementation and System Modeling	11/16/25

Table 1

2. Problem Statement

a) What is your product, on a high level?

Our product is a Student Grade Prediction & Recommendation System, a machine learning–based web application that predicts whether a student will pass or fail a course. The system provides students with their chances of passing as well as tailored advice (e.g., attend classes more often, study longer, decrease stress levels) to improve their academic standing.

b) Whom is it for?

The system exists for:

- **Students:** Perform predictions and get tailored recommendations to improve their academic standing.
- **Teachers/Advisors:** Monitor student performance, identify at-risk students and if possible, provide personalized guidance to help them succeed.
- **Administrators:** Manage user accounts, control system access, and oversee platform operations.

c) What problem does it solve?

The current gradebook systems, which include Blackboard and Canvas show students their actual scores, but they lack any system to alert them about their performance or offer help. Students discover their risk status only after completing their final exams which creates limited time for improvement. This system solves the problem by:

- The system enables users to predict student achievement results before the official grades are released.
- Generating recommendations based on the student situation and helping them recognize their potential low points to improve

d) What alternatives are available?

- Learning Management Systems (LMS) like Blackboard and Canvas show performance data but lack predictive analytics and personalized recommendations.
- Blackboard Analytics is a commercial, costly tool that targets administrators, not students.
- No widely available low-cost system provides student-level explainable predictions with recommendations.

e) Why is this project compelling and worth developing?

The system offers:

- **Early Intervention:** helping at-risk students succeed before it's too late.
- **Accessible design:** lightweight web interface (Streamlit GUI).
- **Affordable Deployment:** built entirely with free/open-source tools and deployable on free cloud platforms.
- **Dual Academic & Operational Value:** demonstrates ML research and provides universities with a working tool to improve retention and success.

f) Top-level objectives, differentiators, target customers, and scope.

- **Objectives:**
  - Assess the potential success of a student.
  - Show the chances of success together with essential risk elements.
  - The program should create individualized plans which help students develop their study methods and create effective daily routines.
- **Differentiators:**
  - The system merges machine learning prediction models with rule-based recommendation systems.
  - The system bases its operation on explaining its decision-making steps instead of achieving the highest possible accuracy rates.
  - Users can train models with new data through the system while performing batch predictions from imported CSV files.
- **Target Customers:** University students, advisors, and instructors.
- **Scope:**
  - Cloud-hosted web application with a web interface.
  - The system requires student data entry functions, CRUD operations, and an ML-based prediction system.
  - Dashboards for students and advisors.
  - Messaging feature between students and their teachers.

g) What are the competitors, and what is novel in your approach?

- The competition includes both Learning Management System dashboards (Canvas and Blackboard) and commercial analytics platforms.
- **Novelty:**
  - The system integrates explainable AI technology, which reveals the specific reasons behind its failure prediction for students.
  - The system implements low-cost deployment through its use of open-source frameworks, including scikit-learn, Streamlit, and pandas.
  - Provides personalized recommendations that can be customized by advisors.

h) Can the system be built with available resources and technology?

Yes. The system operates by:

- **Front-end:** Streamlit
- **Back-end / ML Engine:** Python 3.10+, scikit-learn, pandas, NumPy, joblib.
- **Database:** SQLite
- **Hosting & Deployment:** Utilize Hugging Face Spaces or Streamlit Community Cloud for early versions, and Docker + Render/Railway for scalable deployment.
- **Baseline Models:** Decision Trees, expandable to Deep Learning architectures.

i) What is interesting about this project from a technical point of view?

- The high-performance ML models achieve the following results: Accuracy ~92.8%, F1 ~94.1%, ROC-AUC ~98.6%.
- Streamlit GUI provides users with an interactive two-column interface that enables real-time prediction functionality.
- The Recommendation Engine follows two rules which state that students who attend less than 75% of classes need to improve their attendance and students who experience stress levels above 7 should receive stress management strategies.
- The Explainable AI system shows which specific features (attendance, assignments, study hours) affect the prediction results.
- Cloud-ready architecture exists as a deployable system that uses Docker and open-source hosting platforms.

3. System Requirements

3.1. Use Case 1: Manage User Accounts

Actors: Admin

Supporting Actors: Database

Description: The Admin can create new user accounts for Teachers and Students, update existing user details, or remove user accounts. This ensures only authorized users have access and keeps the system's user directory up-to-date. The system interacts with the Database to validate inputs, check for duplicates, and store or remove user records.

Alternate Path: If the Admin attempts to add a user that already exists (e.g., duplicate email or ID), the system shows an error and prompts to enter a unique identifier. For updates, if invalid or incomplete data is provided, the system rejects the changes and requests correct input. In the case of deletions, if the user account is linked to other records (e.g. a teacher with classes assigned), the system will warn the Admin and may restrict deletion until those links are handled.

Pre-condition: The Admin is logged into the system with administrator privileges.

Use Case 2: Record Student Performance

Actors: Teacher

Description: The Teacher enters or updates students' performance data into the system. This use case allows teachers to maintain an up-to-date record of each student's academic performance metrics.

Alternate Path: If the Teacher inputs data in an incorrect format or leaves required fields blank, the system will display validation errors and not save the record. The Teacher can then correct the input and resubmit. In cases where a student's record for a particular exam or term already exists, the system will update that record instead of creating a duplicate.

Pre-condition: The Teacher is authenticated and has the appropriate permissions to manage performance data (typically for students in their class or subject).

Use Case 3: Predict Student Performance

Actors: Teacher

Supporting Actors: Database, ML Model

Description: The Teacher triggers the system's predictive model (a trained machine learning model) to forecast a student's future performance or risk level. The system retrieves the student's academic data from the Database, processes the data through the ML Model, and displays the prediction and confidence score. For example, the Teacher can select a student and run the model to predict whether the student is likely to excel, pass, or be at risk of failing based on the data in their Student Record.

Alternate Path: If the student's performance data is incomplete or the model cannot generate a reliable prediction (e.g., due to missing values or the model not being trained on certain new data),

the system will inform the Teacher that the prediction is unavailable or not confident. The Teacher may need to ensure all necessary data (grades, attendance, etc.) are recorded before retrying.

Pre-condition: The system has a trained machine learning prediction model available, and the Teacher is logged in with access to the prediction feature. Relevant student performance data must already be recorded in the system.

Use Case 4: Provide Feedback

Actors: Teacher

Description: The Teacher provides feedback or improvement suggestions to a student based on their performance. For instance, if a student's predicted performance is poor or their recorded grades are low, the Teacher can submit a feedback note or recommendation (such as advising extra tutoring or noting specific areas to improve) which the student can later view.

Alternate Path: If the Teacher submits feedback without selecting a valid target student or leaves the feedback message blank, the system will prompt for the missing information. In case the system supports sending notifications, an alternate flow could include notifying the student once feedback is submitted.

Pre-condition: The Teacher is logged in and has access to students' records for providing feedback. A performance record for the student should exist to contextualize the feedback.

Use case 5: View Personal Performance

Actors: Student

Description: The Student views their own performance data and related information through the system. This includes seeing their grades, attendance, overall progress, as well as any predictions of performance and feedback given by Teachers. The purpose is to keep students informed about their academic standing.

Alternate Path: If the student has no performance data recorded yet (for example, a new student or no grades entered for the term), the system will display a message indicating that no performance records are available. If the Student tries to access another student's data (an unauthorized action), the system will deny access.

Pre-condition: The Student is authenticated in the system. Their performance records (grades, etc.) and any teacher feedback must exist in the system for those to be viewable.

Use Case 6: Generate Performance Report

Actors: Admin

Description: The Admin generates an overall performance report for a class, grade level, or the entire school. This report aggregates student performance data (average grades, pass/fail rates, etc.) and can highlight trends or identify students at risk. It helps administrators and educators assess the effectiveness of instructional programs and allocate resources.

Alternate Path: If there is insufficient data in the system (e.g., no grades have been entered for the term), the system will produce an empty or partial report and inform the Admin that data is missing. In some cases, the Admin can select different parameters (such as date range, specific classes) for the report; an alternate path would handle invalid or out-of-range parameters by prompting the Admin to adjust the report criteria.

Pre-condition: The Admin is logged in. A significant amount of student performance data from teachers should be available in the system for the report generation (otherwise the report will have minimal content).

3.2. Requirements

Requirement 1: Manage User Accounts

Use Case Name: Manage User Accounts

Introduction: This requirement specifies the system's capability to allow an administrator to manage user accounts. The Admin should be able to create new accounts for teachers and students, edit existing account information, or remove accounts as needed. This ensures proper access control and user management for the application.

Inputs: The Admin provides user details such as name, email, role (Teacher or Student), and initial login credentials when creating a new account. For updating an account, the Admin inputs the modified details (e.g., updated email or name). To delete an account, the Admin specifies the target user (e.g., by selecting from a list of users or entering an ID).

Requirement Description: The system **shall** present an account management interface to Admin users. Upon *creating* a new account, the system shall validate that all required fields are provided and that the username/email is unique. If valid, the system stores the new user in the database with the specified role. For *updates*, the system shall allow the Admin to change permissible fields (e.g., update a typo in the name or reset a password) and then save the changes to the database. For *deletions*, the system shall remove the user's record from the database (after confirming the action) and also remove or reassign any records that are dependent on that user (for example, student performance records or feedback entries linked to a deleted user should be handled appropriately). The account management functions are restricted to Admin users only; the system shall enforce authorization checks so that non-admins cannot perform these actions.

Outputs: Confirmation messages are displayed for each successful operation (e.g., "User created successfully", "Account updated", "User deleted"). In case of errors (such as duplicate email, validation failure, or unauthorized attempt), the system shows an error message explaining the issue (e.g., "Email already in use, please choose another"). After changes, the updated list of users is visible to the Admin, reflecting any additions or removals.

Requirement 2: Record Student Performance

Use Case Name: Record Student Performance

Introduction: This requirement covers the input and update of student academic data by a Teacher. The system must allow teachers to record various performance metrics for students, such as exam scores, assignment grades, and attendance, which form the student's performance record.

Inputs: The Teacher selects a target student and enters performance details. Inputs can include course/subject identifiers, exam or assignment names, and the scores or grades achieved. The teacher may also input attendance information (e.g., percentage of classes attended) or other performance indicators.

Requirement Description: The system **shall** provide a form or interface where Teachers can enter student performance data. When a Teacher submits this data, the system will validate it (for example, checking that scores are within a valid range, required fields like student ID and score are not blank, etc.). Valid entries are then saved to the student's performance record in the database. If a record for the specified exam/assignment already exists for that student, the system shall update the existing record instead of creating a duplicate. The design should associate the logged-in Teacher's identity with the data entry for accountability (e.g., log which teacher entered or updated the record). The system shall only allow teachers to record data for students that they are assigned to (ensuring one Teacher cannot accidentally overwrite another's records). All transactions should maintain data integrity, for example, if the database update fails for some reason, the system will report a failure and not partially save incorrect data.

Outputs: After submission, the system provides feedback on the operation. On success, a message like "Performance record saved successfully" is shown, and the new data becomes visible in the student's profile/report. The updated performance can trigger recalculation of any aggregates (e.g., average score) which the system may display. On validation error or failure, an error message is shown (e.g., "Score must be a number between 0 and 100" or "Unauthorized: you cannot record data for this student"). The outputs also include the updated performance listings in the UI for the Teacher to review.

Requirement 3: Predict Student Performance

Use Case Name: Predict Student Performance

Introduction: This requirement defines the system's functionality to use a machine learning model to forecast a student's future performance or risk level. A Teacher can request a prediction for a particular student, and the system will return a prediction based on the data available. The goal is to help identify students who might need intervention or to forecast outcomes like final grades.

Inputs: The Teacher selects which student(s) to run the prediction for. This could be done by choosing a student from a list or viewing a student's profile and clicking a "Predict Performance" button. No direct numerical input is needed from the Teacher, as the prediction uses existing data; however, the Teacher's selection (student identifier or class) is an input that tells the system what data to analyze.

Requirement Description: When the Teacher initiates a prediction, the system **shall** gather the relevant data for the selected student from their Student Record (e.g., past grades, attendance, participation, etc.). This data is fed into the pre-trained machine learning model. The system then

generates a prediction of performance, for instance; it might predict the student's final grade for the term or the probability of the student passing a course. The requirement is that the prediction model is accessible through the system's interface and can be executed quickly. The system shall present the prediction result to the Teacher in a readable format. The system must ensure that only authorized users (Teachers/Admins) can run predictions, as it uses sensitive student data.

Outputs: The primary output is the prediction result for the student's performance. This could be a category (e.g., "Pass" or "Fail"), a score (like a predicted grade or numeric score), or a probability. The system will display this result on the UI, possibly alongside the student's current data. For example, it might show a colored indicator or a textual message such as "Predicted outcome: Pass (85% confidence)". If the prediction cannot be produced, the output would be an error or notification (e.g., "Prediction unavailable: insufficient data"). All prediction outputs should be clearly labeled as predictions (not actual results) so that users understand it's a forecast. The system may also log that a prediction was generated (for auditing or future improvements).

Requirement 4: Provide Feedback

Use Case Name: Provide Feedback

Introduction: This requirement describes how Teachers can input qualitative feedback into the system for a student. The feedback feature complements numeric performance records and predictions by enabling teachers to communicate suggestions, comments, or recommendations to students through the system.

Inputs: The Teacher selects a student to provide feedback for (usually in the context of that student's performance profile) and enters a textual message. Inputs include the student's identifier (or selecting from a list) and the content of the feedback message. Optionally, the Teacher might categorize the feedback (e.g., positive reinforcement, improvement suggestion, and warning) by selecting a type or priority, although this is an extra feature not explicitly required.

Requirement Description: The system **shall** allow a Teacher to submit a feedback comment tied to a specific student. When submitted, the feedback is stored in the system (in a Feedback record linked to the Teacher and student). The system should timestamp the feedback entry. Only teachers (and possibly admins) can create feedback entries; students can only read feedback, not alter it. The requirement includes validation, such as ensuring that the feedback message is not empty and possibly limiting the length to a reasonable amount. The system should also provide an interface for teachers to review and edit their own submitted feedback (in case of mistakes). Privacy controls might be implied: e.g., one Teacher should generally only see feedback they provided or that which is shared among relevant staff. The feedback feature should be integrated such that when a student views their performance, they can also see the feedback messages.

Outputs: After submission, the system confirms that the feedback has been saved (e.g., "Feedback sent to student"). The new feedback entry becomes visible in the student's view (usually along with the author's name and date). In the Teacher's interface, the submitted feedback might appear in the context of the student's record. If an error occurs (like network/database failure or unauthorized access attempt), an error message is shown (e.g., "Failed to submit feedback, please

try again"). If notifications are enabled, the student (and possibly the Admin) will receive a notification alert about the new feedback.

Requirement 5: View Personal Performance

Use Case Name: View Personal Performance

Introduction: This requirement specifies that students can access a personalized view of their academic performance data through the system. The system will serve as a student portal where individuals see their own records, including grades, attendance, any teacher feedback, and predictive information about their performance.

Inputs: The Student initiates this by logging into the system and navigating to a "My Performance" or dashboard section. No manual data input is required from the student's side to view the information. The primary input is the student's identity (taken from their login session), which the system uses to fetch the relevant records.

Requirement Description: Once authenticated, the system **shall** retrieve all performance-related data for the logged-in student. This includes their grades for assignments/exams, attendance percentage, cumulative scores or GPA, and any feedback notes from teachers. The system will compile this into a dashboard or report format for the student. The requirement is that students can only see **their own** data access control is crucial here (a student cannot see other students' records). The system should present the data in an easy-to-understand manner.

Outputs: The output is the student's performance report displayed on the screen. This includes lists of grades, attendance records, and sections for teacher feedback and prediction results.

Requirement 6: Generate Performance Report

Use Case Name: Generate Performance Report

Introduction: This requirement covers the ability of an Admin to produce a comprehensive report of student performance at an aggregate level. The system will compile data across many students to help administrators see overall trends, averages, and identify outliers or at-risk students.

Inputs: The Admin may specify parameters for the report generation. Inputs could include the scope of the report. For a general report, the default might be all students for the current term if no specific filter is given.

Requirement Description: When the Admin requests a report, the system **shall** aggregate relevant data from the database. Depending on the scope, it will gather all students' performance records in that scope. The Admin should be able to save or print this report, so the requirement may include an option to export the report (as PDF or CSV, for instance). Data security is important: only Admin (or authorized management personnel) can generate and view such broad reports, because they contain sensitive information about multiple students.

Outputs: The output is a formatted performance report, displayed on the Admin's screen (and optionally downloadable). Upon successful generation, the report view is the output. If no data is

available for the chosen parameters, the report output will explicitly state “No data available for the selected range.” In case of errors (like a database timeout if the dataset is huge), the system will output an error message to the Admin (e.g., “Report generation failed, please try again later”). A successful report generation might also be logged in the system for auditing (this is not a user-visible output, but part of system outputs in a broader sense).

3.3. Use Case Diagram

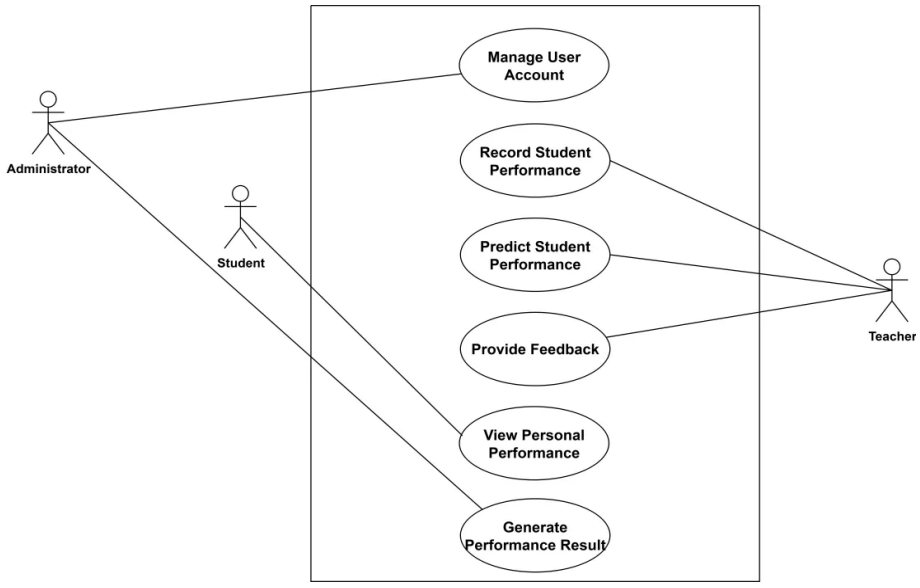


Fig. 1 Use Case Diagram for the system

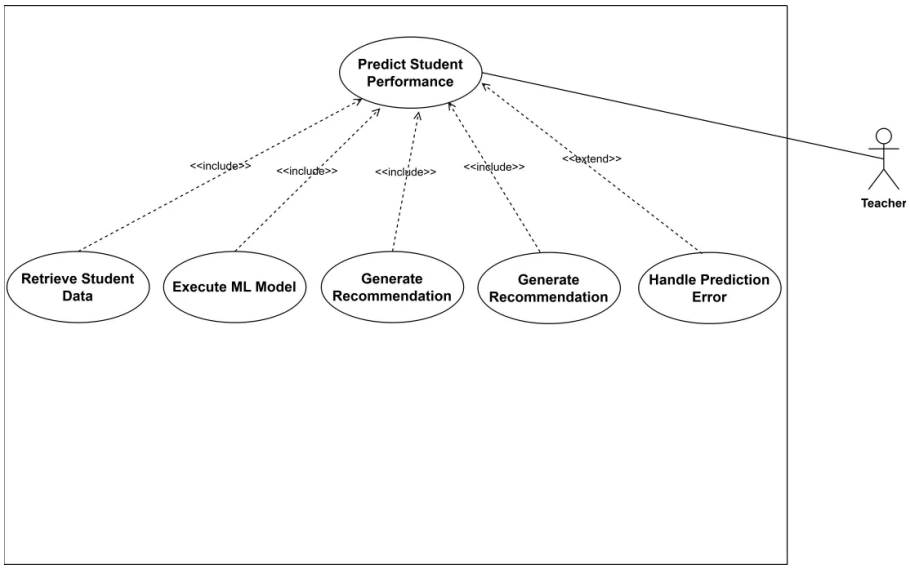


Fig. 2 Use Case Diagram - Predict Student Performance

3.4. Context Diagram

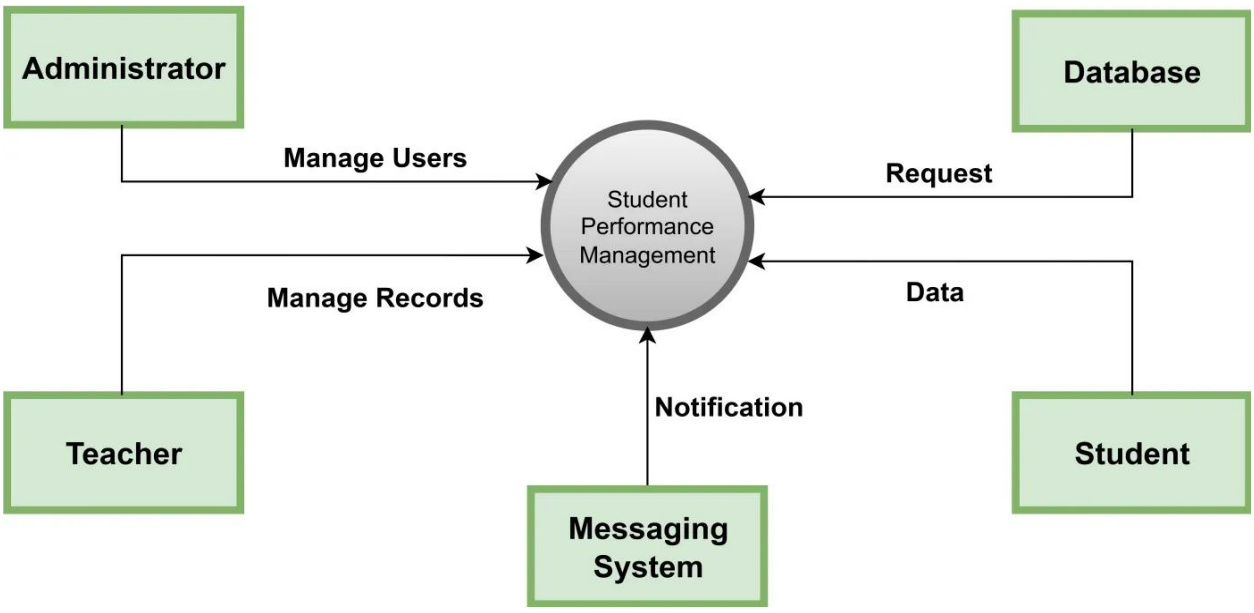


Fig. 3 Context Diagram

4. System Modeling

4.1. Class Diagrams

Class diagrams show the static structure of the system comprised in its classes, attributes, methods and associations amongst objects.

a) Objects

Users (Base for Students, Teachers and Admin)

- Attributes: userID, email, password, name, userRole
- Operations: login(), logout(), updateProfile(), changePassword(), deleteAccout()

Student

- Attributes: studentID, userID, gpa, departmentID, academicStatus
- Operations: submitAcadmeicData(), viewPrediction(), getRecommendation()

Teacher

- Attributes: teacherID, userID, name, email, department
- Operations: viewStudent(), provideFeedback(), customizeRules(), viewPredictionResults(), viewStudentPerformance(), exportStudentReport ()

Administrator

- AdminID, userID, name, email
- Operations: manageUserAccounts(), updateSystemSettings(), createUser()

StudentAcademicData

- Attributes: dataID, studentID, attendance, studyHours, examScores, stressLevel, sleepHours, participation
- Operations: calculateGPA(), compareWithClassAverage(), validateData()

PredictionResults

- Attributes: predictionID, studentID, modelID, predictionStatus, passPercentage, failPercentage
- Operations: PredictPassFail(), getExplanation(), displayProbability(), generateRecommendation(), identifyRiskFactors()

Recommendation

- Attributes: recommendationID, predictionID, studentID, recommendationType, message
- Operations: generateRecommendation(), evaluateThresholds(), snedToStudent(), describepotentialImprovemtn(), custtomizeMessage()

MLModel Class

- Attributes: modelID, modelName, accuracy, modelType
- Operations: trainModel(), predictPassFail(), saveModel(), getFeatureImportance(), retrain()

Message

- Attributes: messageID, senderID, recieverID, subject, content, readStatus, attachmentPath
- Operations: sendMessage(), getContent(), addAttachment(), getAttachment()

Dashboard

- Attributes: DashboardID, userID, dashboardType, customSettings
- Operations: displayMetrics(), generateCharts(), getStudentPresictionSummary(), getPerformanceComaparison()

b) Associations Between Objects

- Users – Students (One to One)
- User - Teacher (One to One)
- User - Administrator - (One to One)
- Student - StudentAcademicData (One to Many)
- Student – PredictionResults (One to Many)
- PredictionResults – Recommendations (One to Many)
- MLModels – PredictionResults (One to One)
- User – Messages (One to Many)
- User – Dashboards (One to One)

c) Multiplicity

- A user can be a student (1:1)
- A user can be a teacher (1:1)
- A user can be an Administrator (1:1)
- A student can submit multiple academic data (1:N)
- A student can have multiple prediction results over time (1:N)
- An ML model can generate multiple prediction results over time (1:N)
- A prediction can generate or suggest multiple recommendations (1:N)
- A User can have receive multiple messages(1:N)
- A User is associated with a dashboard (1:1)

d) Attributes of objects

Each class comprises attributes representing the data it holds. Example:

- Student: name, studentID, userID, gpa, departmentID, academicStatus
- Administrator: adminID, name, email, role
- StudentAcademicData: dataID, studentID, attendance, studyHours, examScores, stressLevel
- recommendation: recommendationID, predictionID, studentID, recommendationType, message

e) Operations/Methods defined on objects

Each class defines operations that can be performed on its instances. Example:

- Student: enterData(), viewPrediction(), viewRecommendation()
- Administrator: manageUsers(), updateSystemSettings()
- Teacher/Professor/Instructor: viewStudent(), provideFeedback(), customizeRules()
- PredictionResults: PredictPassFail(), getExplanation(), displayProbability(), generateRecommendation(), identifyRiskFactors()

f) System Class Diagram

The Class diagram shows the relationship, constraints between entities involved in the EduTrack grade and recommendation system.

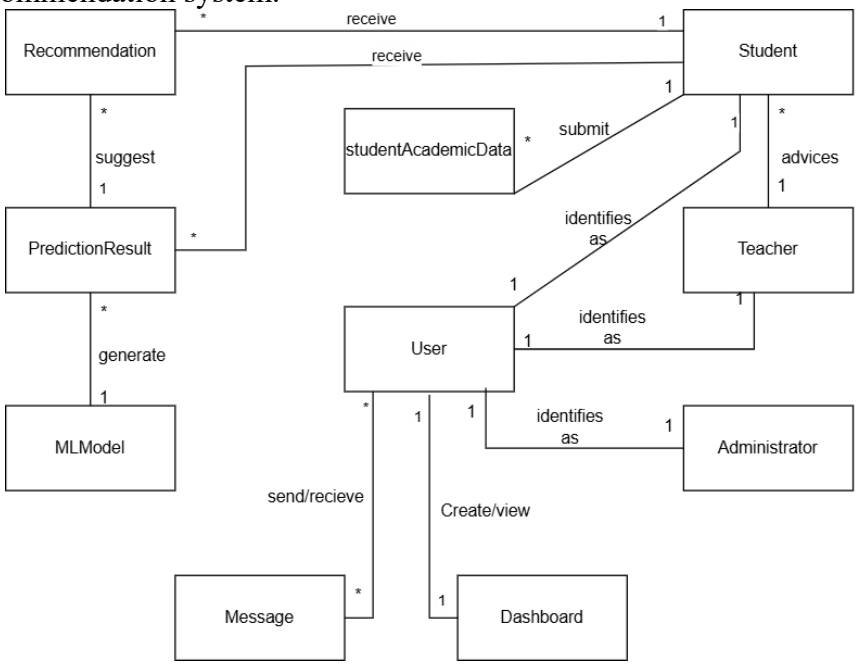


Fig. 4 System Class Diagram

4.2. Architecture Modeling (from A3)

The student performance prediction management system is best described by a **hybrid architecture** combining several patterns. At the highest level, it is a **client-server** system: end users (students, teachers, admins) interact via a web-based front end (a Streamlit GUI), while the core logic and data reside on a server. Internally, the server is structured in **layers:** a presentation layer (GUI), an application/business-logic layer (the ML prediction and recommendation engine), and a data layer (the database). This separation cleanly isolates the UI from the model logic and the persistence. For example, teachers and students use the Streamlit interface to submit or view data, which the system then processes in Machine-Learning model and stores in SQLite (a central repository). Data flows through the system in a **pipe-and-filter** fashion: user input is packaged and fed into the ML pipeline (filter), which outputs a prediction that is then passed to the recommendation filter, and finally displayed. In effect, the ML processing (input → preprocessing → model → recommendation → output) follows a chain of filters. The database acts as a

repository, a shared data store accessed by the various components (user/account management, performance recording, and prediction engine) decoupling business logic from data persistence.

Visually, the **layered architecture** can be depicted as a stack of tiers (UI → Logic → Data). In practice, a student performance prediction management system's layers might be shown as:

- **Presentation Layer** (Streamlit UI)
- **Application/ML Logic Layer** (prediction and recommendation modules).
- **Data Layer** (database and model store). Each layer only uses the services of the layer below.
- The **client-server** view highlights that multiple clients (Student app, Teacher app, Admin app) communicate with a central server containing these layers and the database.
- **pipe-and-filter** architecture is evident in the ML pipeline: data flows sequentially through preprocessing, model inference, and recommendation stages. Together, these patterns ensure modularity (layers), centralized data management (repository), networked operation (client-server), and clear data processing flows (pipe-and-filter).

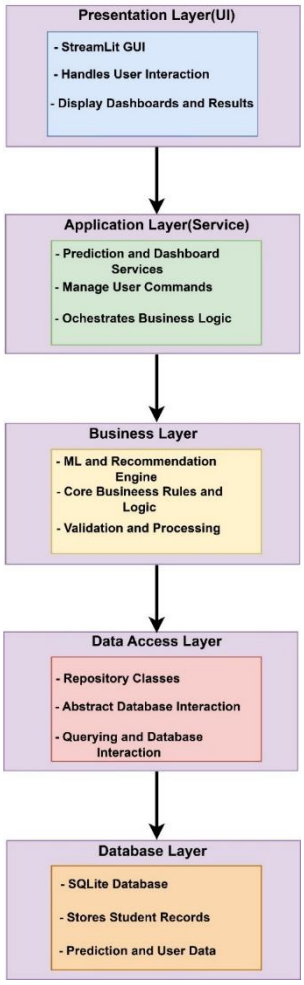


Fig. 5 Architecture Modeling

4.3. Behavioral Modeling - Sequence Diagrams (from A3)

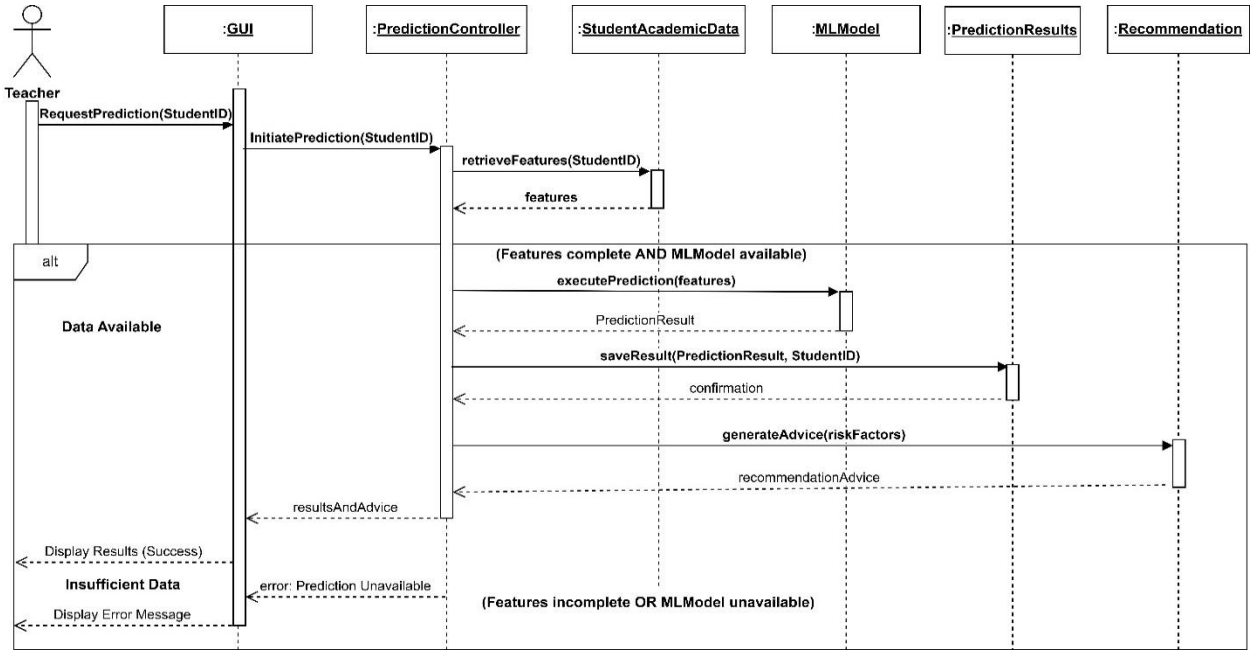


Fig. 6 Use Case 3 – Predict Student Performance

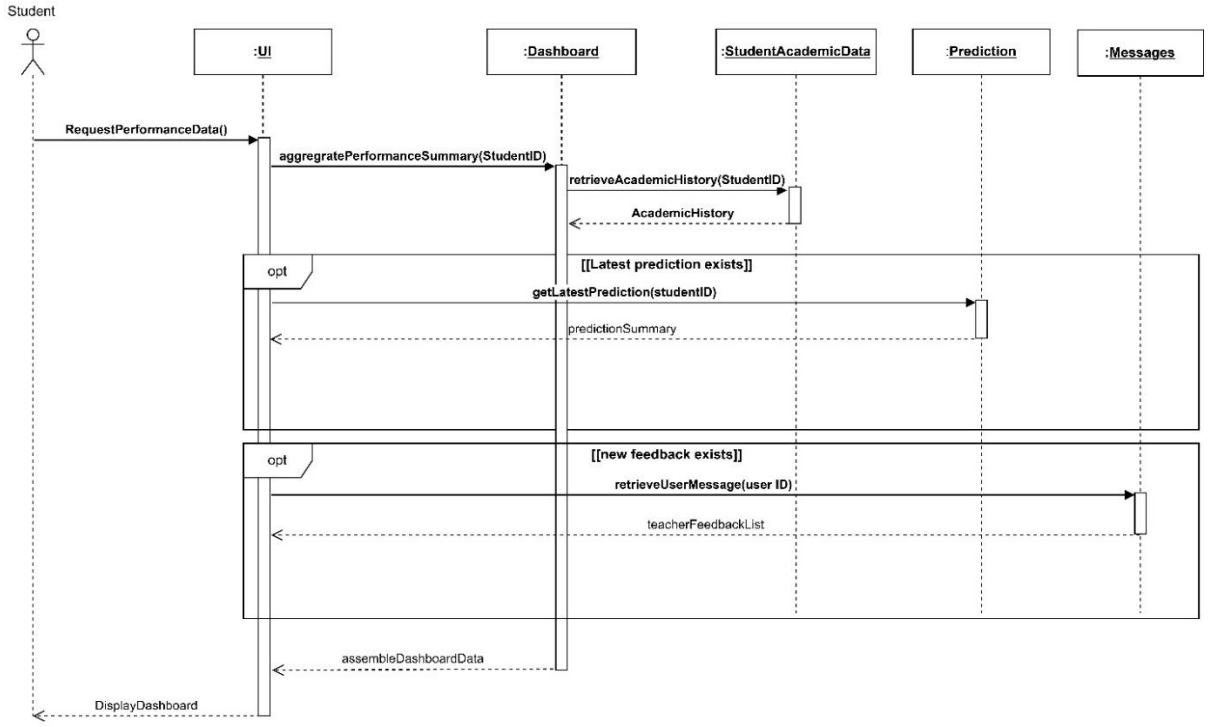


Fig. 7 Use Case 5 – View Personal Performance

4.4. Architecture Modeling (from A4)

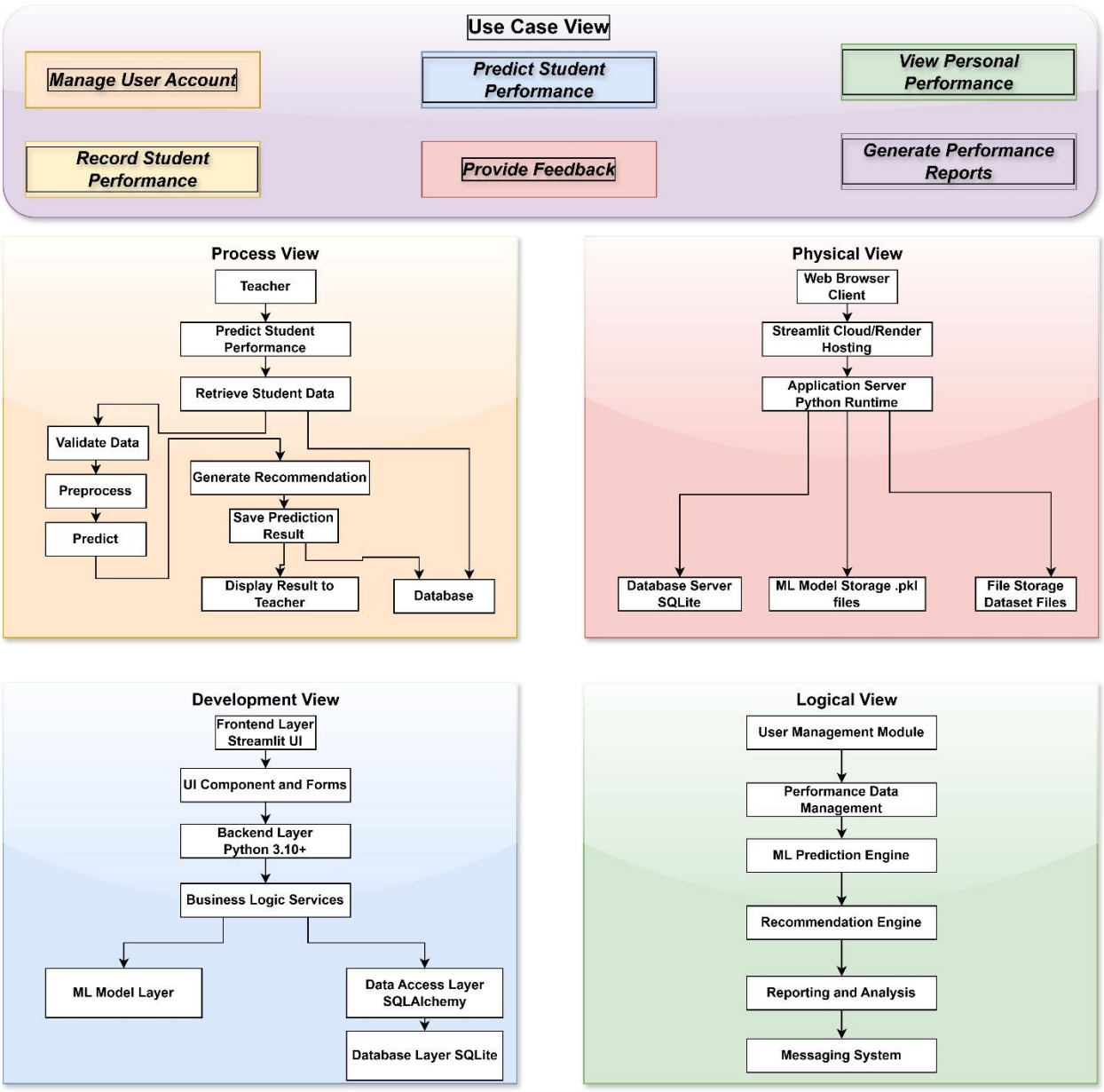


Fig. 8: 4 + 1 Architectural Design

Fig. 8 depicts the 4 + 1 architectural view model that;

- Shows different stakeholder perspectives
- Maintains consistency across views
- Clearly separates concerns (UI, business logic, data, ML)
- Demonstrates a complete web application with ML capabilities
- Uses appropriate technologies for an educational analytics system

Figure 8 effectively communicates the system architecture from multiple angles, making it understandable for developers, stakeholders, and deployment teams. The 4 +1 architecture is modeled as follows;

- **Use Case View:** It shows system functionality from the user's perspective, which includes user management, performance prediction, data recording, and feedback.
- **Process View:** It illustrates the runtime workflow and data flow, and it also connects users, business logic, and the database
- **Physical View:** It shows the deployment infrastructure and hardware, which consists of client browsers, cloud hosting (Streamlit/Render), application server, database (SQLite), ML model
- **Development View:** It displays software layers and development structure. The layers consist of frontend (Streamlit), Backend (Python), Business Logic, ML Models, Data Access (SQLAlchemy), and Database.
- **Logical View:** It represents functional modules and conceptual structure that focuses on what the system does (functionality) rather than how it's built. The core modules include: User management, data management, ML prediction engine, recommendation system, analytics, and messaging.

4.5. Behavioral Modeling (from A4)

Use Case 2: Record Student Performance

Teachers input and update students' academic performance data into the system. It creates the foundational data that enables all predictions and analytics in the system. This includes entering metrics like:

- Exam scores and assignment grades
- Attendance percentages
- Behavioral data (study hours, stress levels, sleep patterns)
- Class participation rates

Key Flow:

- The teacher selects a student and a course
- Enter performance metrics through forms
- System validates data format and ranges
- Data is saved to the database
- Confirmation is displayed to the teacher.

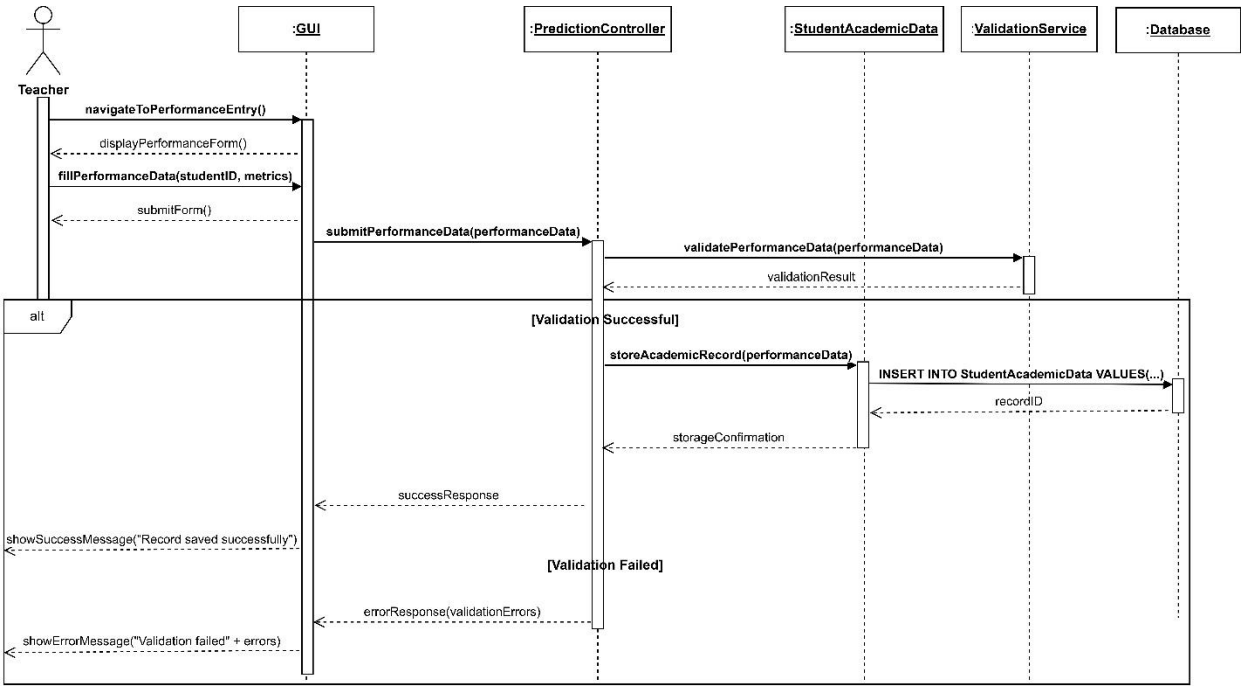


Fig. 9: Use Case 2 – Record Student Performance

Use Case 6: Generate Performance Report

Administrators generate comprehensive reports that aggregate student performance data across classes. It provides institutional-level understandings for decision-making, resource allocation, and identifying areas needing academic intervention. These reports help identify trends, at-risk student populations, and overall academic performance patterns.

Key Flow:

1. Admin selects report parameters (date range, classes, metrics)
2. The system aggregates data from multiple students and courses
3. Generates formatted reports with analytics
4. Reports can be exported as PDF
5. Admin can save or distribute reports

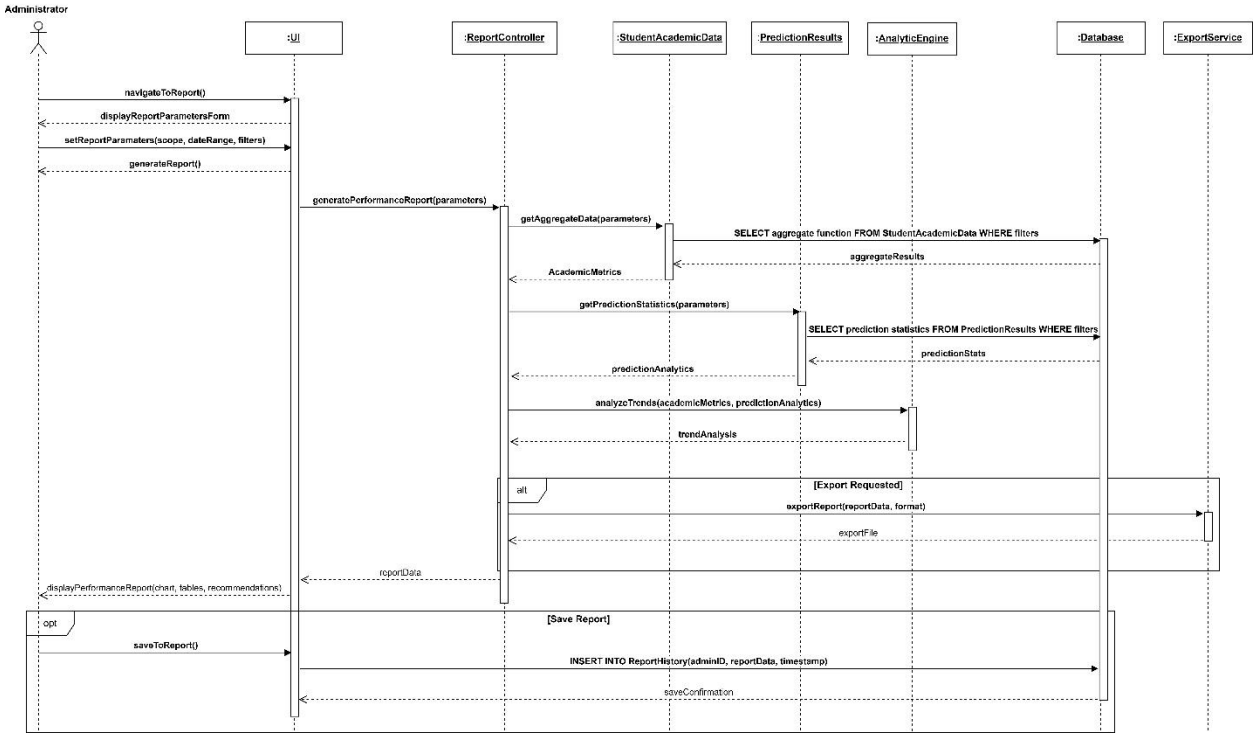


Fig. 10: Use Case 6 - Generate Performance Report

4.6. Database Specification and Analysis

The EduTrack system requires an optimized database design for efficient operations (querying, retrieval and security)

Table: User

Attribute	Type	Key
UserID	INTEGER	Primary Key
name	TEXT	
email	TEXT	UNIQUE
password	TEXT	
userRole	TEXT	

Table: Student

Attribute	Type	Key
UserID	INTEGER	Primary Key
name	TEXT	
email	TEXT	UNIQUE
password	TEXT	

userRole	TEXT	

Table: Teacher

Attribute	Type	Key
UserID	INTEGER	Primary Key
name	TEXT	
email	TEXT	UNIQUE
password	TEXT	
userRole	TEXT	

Table: Administrator

Attribute	Type	Key
adminID	INTEGER	Primary Key
userID	INTEGER	Foreign Key
name	TEXT	
email	TEXT	

Table: StudentAcademicData

Attribute	Type	Key
dataID	INTEGER	Primary Key
studentID	INTEGER	Foreign Key
attendance	REAL	
studyHours	REAL	
examScores	INT	
stressLevel	INTEGER	
sleepHours	REAL	
participation	REAL	

Table: PredictionResult

Attribute	Type	Key
predictionID	INTEGER	Primary Key
StudentID	INTEGER	Foreign Key
ModelID	INTEGER NOT NULL	Foreign Key
predictedStatus	TEXT	
passPercentage	REAL	

Attribute	Type	Key
failPercentage	REAL	

Table: Recommendation

Attribute	Type	Key
recommendationID	INTEGER	Primary Key
predictionID	INTEGER	Foreign Key
studentID	INTEGER	Foreign Key
recommendationType	TEXT	
message	TEXT	

Table: ML Model

Attribute	Type	Key
modelID	INTEGER	Primary Key
modelName	TEXT	
accuracy	REAL	
modelType	TEXT	

Table: Message

Attribute	Type	Key
MessageID	INTEGER	Primary Key
SenderID	INTEGER	
receiverID	INTEGER	
subject	TEXT	
content	TEXT	
readStatus	TEXT	

Table: Dashboard

Attribute	Type	Key
dashboardID	INTEGER	Primary Key
userID	INTEGER	
dashboardType	TEXT	
customSettings	TEXT	

Association between Tables

- User to Student: (1:1)

- User to teacher (1:1)
- User to Administrator (1:1)
- User to Messages (M:N)
- User to Dashboards (1:1)
- Student to academicData (1:N)
- Student to predictionResults (1:N)
- ML model to predictionReslults (1:N)
- predictionResults to recommendations (1:N)

Database Multiplicity

- A User can be a student
- A User can be a teacher
- A User can be an administrator
- A User can receive multiple messages
- A User has access to a dashboard
- A student can have multiple prediction results over time
- An ML model can have multiple predictions over time
- A prediction result can generate multiple recommendations

Security Considerations

- User authentication using hashed passwords
- Data encryption for sensitive fields
- Role-based access control (Least Privilege)
- SQL injection prevention via prepared statements.

5. Implementation

5.1. Implementation from A3

a) Overview from A3

The implementation phase realizes the functional prototype of **EduTrack — Student Grade Prediction and Recommendation System**.
It implements two key use cases identified in design:

1. **UC2 – Record Student Academic Data**
2. **UC3 – Predict Student Performance and Generate Recommendations**

The system integrates a **SQLite database**, **Streamlit web interface**, and a **Random Forest machine learning model** trained on the *Students Performance Dataset*. It provides complete data flow from form submission to prediction storage and recommendation generation.

b) Tools and Technologies

Component	Technology	Purpose
Programming Language	Python 3.10	Unified language for backend, ML, and frontend
Database	SQLite3	Embedded RDBMS for development & testing
ORM / SQL Layer	SQLAlchemy	Safe, parameterized SQL transactions
Frontend/UI	Streamlit	Interactive user interface for data entry & predictions
Machine Learning	scikit-learn, Joblib	Training, saving, and loading the predictive model
Security	SHA-256 with application pepper (demo-grade)	Secure password hashing
Dataset	Students_Performance_Dataset.csv	Basis for model training and testing

Table 2

c) Database Implementation

Schema Overview

The database schema (implemented in db/schema.sql) defines the following tables:

- **users** – stores user identity and roles: user_id, name, email, password_hash, role, created_at. Role restricted to ADMIN, TEACHER, STUDENT.
- **students** – links each student to a user_id and holds GPA, department, and status.
- **teachers** – maps teacher user IDs to departments.
- **student_academic_data** – captures term-wise academic metrics.
    a. Includes numeric fields (attendance, study_hours, exam_score, etc.) and an extra_json column for flexible categorical fields (like gender or income).
    b. Enforces one record per student per term using a unique constraint.
- **ml_models** – registers each trained ML model with metadata (path, version, metrics).
- **prediction_results** – stores predictions, including label (PASS/FAIL) and probabilities.
- **recommendations** – links personalized feedback to prediction results.

Foreign keys enforce referential integrity; all deletions cascade safely. Indices (e.g., idx_pred_student_term) ensure fast lookups by student and term.

d) Backend Logic (src/app_db.py)

- **Schema Initialization:** ensure_schema() loads and executes the SQL file automatically on first run.
- **CRUD Operations:**
    - create_user(), create_student(), create_teacher() handle user creation.
    - upsert_academic_record() implements *insert-or-update* logic ensuring no duplicate term data.
    - save_prediction() stores or updates model predictions atomically.
    - add_recommendation() adds feedback entries.
- **Security:** Passwords are hashed using SHA-256 with an application-specific pepper for secure demo authentication.
- **Model Registry:** register_model() and latest_model() maintain the trained model versions in the ml_models table.

e) Machine Learning Module (src/train_model.py)

The model training script automates:

1. **Data Loading:** Reads Students_Performance_Dataset.csv.

2. **Target Creation:** Derives a binary "Pass" label from existing grade or total columns.
3. **Preprocessing:**
    - Numeric columns → Median imputation + Standard scaling.
    - Categorical columns → One-hot encoding.
4. **Model:** RandomForestClassifier with balanced weights and 400 estimators.
5. **Evaluation:** Computes Accuracy, F1-score, and ROC-AUC.
6. **Persistence:** Saves model as models/model.pkl and exports feature schema to models/feature_schema.json.
7. **Registration:** Logs the trained model into the ml_models table with metrics.

Command to retrain:

python src/train_model

f) Frontend / Application Logic (app.py)

The **Streamlit** app connects UI, database, and ML model:

Section 1 — Record Student Performance

- Teachers log in or use demo accounts.
- Inputs academic and behavioral fields:
    - Core numeric metrics: Attendance, Study Hours, Exam Score, Stress Level, Sleep Hours, Participation.
    - Auto-generated dropdowns (from feature_schema.json) for Gender, Department, Internet Access, etc.
- Submitting the form triggers upsert_academic_record() and saves the data for that term.

Section 2 — Predict Student Performance

- Teacher selects student email and term.
- Application retrieves the most recent record via get_latest_record().
- Features are ordered as per schema and fed into the trained pipeline (model.pkl).
- Prediction output:
    - **Label:** PASS / FAIL
    - **Probability:** Confidence score
    - Stored using save_prediction().
- Automatic recommendation rules suggest actions such as:
    - Low attendance → "Improve attendance to at least 75%."
    - Low study hours → "Increase study time by 2–4 hours per week."

g) Feature Schema (models/feature_schema.json)

The model exports structured metadata for all features used in training:

- Numeric fields (e.g., Midterm_Score, Final_Score, Sleep_Hours_per_Night)
- Categorical dropdowns (e.g., Gender, Department, Family_Income_Level) This file drives dynamic UI generation, ensuring consistency between model and frontend.

h) Execution Steps

1. Create and activate environment
conda create -n gradepred python=3.10 -y
conda activate gradepred

2. Install dependencies
python -m pip install --upgrade pip
python -m pip install streamlit sqlalchemy scikit-learn pandas numpy joblib matplotlib

3. Initialize database and verify schema
sqlite3 db/app.db < db/schema.sql (Note: Run this once during first-time setup to create the database file and apply the schema. Future runs automatically load the schema through ensure_schema().)
python -c "from src.app_db import ensure_schema; ensure_schema(); print('Schema OK')"

4. Train model
python -m src.train_model
Expected Output:

- Saved model to models/model.pkl
- Saved schema to models/feature_schema.json
- Registered model_id=1, name=student_grade_predictor, version=20251103XXXXXX
- Metrics -> acc=0.830, f1=0.812, roc_auc=0.861

5. Run the app
streamlit run src/app.py

No IDE requirement - runs on any terminal or Git Bash.

i) Testing and Verification

- **Schema Verification:** Executed schema.sql successfully created all 7 tables.
- **Data Persistence:** Form submissions create new rows in student_academic_data.

- **Prediction Verification:** Predicted labels and recommendations correctly populate their respective tables.
- **Model Accuracy:** Verified through training log metrics (Accuracy, F1, ROC-AUC).
- **Cross-Integration:** Database, ML, and UI work cohesively with no locking or duplicate-entry issues.

j) Design Highlights

- **Upsert Pattern:** Guarantees one record per student per term.
- **Atomic Transactions:** All inserts/updates use engine.begin() context managers.
- **Model Registry:** Each model version traceable via ml_models.
- **Security:** Bcrypt-hashed passwords for demo users.
- **Extensibility:** extra_json enables dynamic categorical expansion without altering schema.

k) Limitations and Future Work

- Local SQLite DB can be migrated to PostgreSQL/MySQL for multi-user deployment.
- Authentication system can be extended with JWT or OAuth.
- Model explainability (feature importance) visualization planned for Sprint-3.
- Cloud deployment (Streamlit Cloud / Render) to make the system publicly accessible.

5.2. Implementation from A4

a) Overview from A4

Sprint 4 extends the EduTrack - Student Grade Prediction and Recommendation System by implementing two new use cases that enhance administrative insights and improve communication between teachers and students. The two new use cases implemented are:

1. **UC5 – Generate Performance Report (Admin)**
   Allows the Admin to view consolidated academic performance analytics across all students and departments, including KPIs, summary statistics, departmental performance comparisons, visual charts, and exportable reports.

2. **UC6 – Teacher–Student Messaging System**
   Enables secure two-way communication between teachers and their assigned students. Teachers can message students directly, and students can view messages, mark them as read, and send queries to their assigned academic advisor.

These new features integrate seamlessly with the existing schema, prediction module, and role-based dashboards.

b) Tools and Technologies

Component	Technology	Purpose
Programming Language	Python 3.10+	Backend logic, UI, messaging features
Database	SQLite3	Storage for messages, academic data, predictions
ORM / SQL Layer	SQLAlchemy	Safe SQL operations with transactions
Frontend/UI	Streamlit	Interactive dashboards for Teacher, Student, Admin
Machine Learning	scikit-learn, joblib	Prediction model used as context for performance reports
Visualization	Streamlit charts, pandas	Department-level bar graphs

Table 3

c) Database Implementation

Sprint 4 did *not* require new tables; instead, it built on the existing schema.

1) Tables used by UC5 – Performance Report

- **students**
- **student_academic_data**
- **prediction_results**
- **departments**
- **teachers**

SQL queries aggregate these tables to compute:

- Total number of students/teachers
- Average GPA
- Average attendance

- Pass / Fail counts
- Department-level performance metrics

Indices (idx_pred_student_term, idx_students_dept) ensure fast aggregation.

2) Tables used by UC6 – Messaging

The system uses your already-existing messages table:

```
CREATE TABLE IF NOT EXISTS messages (
  message_id    INTEGER PRIMARY KEY AUTOINCREMENT,
  sender_id     INTEGER NOT NULL,
  receiver_id   INTEGER NOT NULL,
  subject       TEXT,
  content       TEXT,
  readStatus    TEXT DEFAULT 'UNREAD',
  created_at    TEXT DEFAULT (datetime('now')),
  FOREIGN KEY (sender_id) REFERENCES users(user_id),
  FOREIGN KEY (receiver_id) REFERENCES users(user_id)
);
```

No schema changes were required.

The table supports:

- Inbox → filtering by receiver_id
- Sent Messages → filtering by sender_id
- Unread tracking → via readStatus = 'UNREAD'

d) Backend Logic (src/app_db.py)

- **Schema Initialization:** ensure_schema() loads and executes the SQL file automatically on first run.

- **CRUD Operations:**
  - create_user(), create_student(), create_teacher() handle user creation.
  - upsert_academic_record() implements *insert-or-update* logic ensuring no duplicate term data.
  - save_prediction() stores or updates model predictions atomically.
  - add_recommendation() adds feedback entries.

- **Security:** Passwords are hashed using SHA-256 with an application-specific pepper for secure demo authentication.

- **Model Registry:** register_model() and latest_model() maintain the trained model versions in the ml_models table.

i) Performance Report Queries

Two new SQL-based helper functions were added:

1) **get_site_performance_summary():** Computes global KPI metrics.
- Student / Teacher counts
- Average GPA
- Average attendance
- Pass / Fail predictions

2) **get_department_performance():** Returns department-wise.
- Total students
- Average GPA
- Average attendance
- Predictions summary

These queries aggregate large sections of the database with joins across students, academic_data, predictions, and departments.

ii) Messaging Backend (Teacher ↔ Student)

New messaging helpers were implemented:

Function	Purpose
send_message()	Inserts a new message into the database
get_inbox()	Fetches all received messages
get_sent_messages()	Fetches all sent messages

mark_message_read()	Updates readStatus to "READ"
get_teacher_students()	Returns all students assigned to a teacher
get_student_advisor()	Returns a student's assigned teacher

Table 4

These operations use engine.begin() for transactional safety.

e) Machine Learning Module (src/train_model.py)

The model training script automates:

1. **Data Loading:** Reads Students_Performance_Dataset.csv.
2. **Target Creation:** Derives a binary "Pass" label from existing grade or total columns.
3. **Preprocessing:**
    - Numeric columns → Median imputation + Standard scaling.
    - Categorical columns → One-hot encoding.
4. **Model:** RandomForestClassifier with balanced weights and 400 estimators.
5. **Evaluation:** Computes Accuracy, F1-score, and ROC-AUC.
6. **Persistence:** Saves model as models/model.pkl and exports feature schema to models/feature_schema.json.
7. **Registration:** Logs the trained model into the ml_models table with metrics.

Command to retrain:

python src/train_model

f) Frontend / Application Logic (app.py)

Sprint 4 added two major UI features.

Section 1 - Admin Performance Report Page

A new admin dashboard page was created.

Features Implemented:

- Student and Teacher counts
- Average GPA
- Average Attendance
- Predicted Pass / Fail counts
- Department-level table
- Bar charts:
    - Avg GPA by department
    - Avg Attendance by department
- CSV export of the department performance table

Navigation: A new item "Performance Report" was added under the Admin Sidebar.

Function: admin_performance_report_page(auth)

This page integrates backend queries + Streamlit visualization to provide an analytical admin view.

Section 2 - Teacher–Student Messaging System

Messaging was added for both Teacher and Student roles.

Teacher Messaging Page:

Function: messaging_page_teacher(auth)

Teachers can:

- Select a student (only those assigned to them)
- Write subject + message
- Send message
- View Inbox
- View Sent messages
- Mark messages as READ

Teacher sidebar now includes:
["My Profile", "My Students", "Predict", "Messages"]

Student Messaging Page:

Function: messaging_page_student(auth)

Students can:

- View assigned advisor
- Send message to advisor
- View inbox (with read/unread states)
- Mark messages as READ
- View sent messages

Student sidebar now includes:
["My Profile", "My Performance", "Predict", "Messages"]

Read Status Behavior

- Unread messages show a **Mark as read** button
- Once clicked → database updates → UI refreshes → (READ) label appears

g) Execution Steps

1. Create and activate environment
conda create -n gradepred python=3.10 -y
conda activate gradepred

2. Install dependencies
python -m pip install --upgrade pip
python -m pip install streamlit sqlalchemy scikit-learn pandas numpy joblib matplotlib

3. Initialize database and verify schema
sqlite3 db/app.db < db/schema.sql (Note: Run this once during first-time setup to create the database file and apply the schema. Future runs automatically load the schema through ensure_schema().)
python -c "from src.app_db import ensure_schema; ensure_schema(); print('Schema OK')"

4. Train model
python -m src.train_model
Expected Output:
- Saved model to models/model.pkl
- Saved schema to models/feature_schema.json
- Registered model_id=1, name=student_grade_predictor, version=20251103XXXXXX
- Metrics -> acc=0.830, f1=0.812, roc_auc=0.861

5. Run the app
streamlit run src/app.py

No IDE requirement - runs on any terminal or Git Bash.

The menu now includes:

- Admin → Performance Report
- Teacher → Messages
- Student → Messages

h) Testing and Verification

UC5 – Performance Report

- KPIs load successfully
- Department tables populate correctly
- Bar charts render
- CSV export downloads correct data
- Works even with partial data (uses LEFT JOINs)

UC6 – Messaging

- Teacher sends message to student
- Student sees message instantly
- Student can mark message as read
- Read status updates in DB
- Student can send message to advisor
- Sent items tracked correctly
- No duplicate messages
- No UI breakages after reruns

i) Design Highlights

- **Role-Aware Messaging:** Students can message only their own advisor; teachers see only their own students.
- **Unread/Read Mechanism:** Clean and simple UX with automatic re-rendering.
- **Analytics at Admin Level:** Performance dashboards allow Admin to see the entire system health.
- **Seamless Integration:** New use cases fit into existing dashboards without modifying older logic.
- **Re-Usable SQL Functions:** Centralized backend helpers ensure maintainability.

j) Limitations and Future Work

- Messages do not support attachments yet.
- No email notifications (local only).
- Performance reports could include trend analysis or per-term filters.
- Future Sprint may add:
    - Admin ↔ Teacher Messaging
    - Student ↔ Student communication
    - Visualization of ML feature importance
    - Export to PDF

6. Testing

6.1. Testing from A3

USE CASE 1: Enter Student Academic Data

- Features: (**INTEGER** studentID, **TEXT** term, **REAL** attendance, **REAL** studyHours, **REAL** examScores, **INTEGER** stressLevel, REAL sleepHours, **REAL** participation, **TEXT** created_by)

- Input Partition into equivalence classes for each feature:

Feature	**Partition**	**Equivalence Class**
studentID	Invalid: Empty string “” Invalid: Nonempty string Invalid: Negative or zero Valid: Positive integer Invalid: Non integer	“” Strings with one or several characters <1 1-999999 Decimal/Floating point values
term	Invalid: Empty string “” Valid: Standard term format Invalid: integer	“” “Fall 2024”, “Spring 2025” 1-999999
attendance	Invalid: Empty strings “” Invalid: Nonempty strings Invalid: Negative value Invalid: Zero attendance Valid: Normal range Valid: Perfect attendance Invalid: Above maximum attendance	““ Strings with one or several characters < 0.0 0.0 0.1 – 99.9 100.0 >100.0
studyHours	Invalid: Empty string “” Invalid: Nonempty string Invalid: Negative Valid: No study Valid: Low study hours Valid: High study hours Invalid: Exceeds daily hours	“ ” Strings with one or several characters <0.0 0.0 5.1-15.0 15.1-24.0 >24.0
examScores	Invalid: Empty string “” Invalid: Nonempty string Invalid: Negative Valid: Failing Score Invalid: Above maximum score Valid: Good score	“ ” Strings with one or several characters <0 0-59 >100 90-100
stressLevel	Empty string “” Nonempty string Below minimum Low stress	“ ” strings with one or several characters < 1 1 – 3

	Moderate stress Hight stress Above maximum	4 – 7 8 – 10 > 10
sleepHours	Invalid: Empty string “” Invalid: Nonempty string Invalid: Negative Valid: No sleep Valid: insufficient sleep Valid: Moderate sleep Valid: Adequate sleep Valid: Excessive sleep Invalid: Exceeds daily hours	““ strings with one or several characters < 0.0 0.0 0.1 – 4.0 4.1 – 6.0 6.1 – 9.0 9.1 – 24.0 > 24.0
participation	Invalid: Empty strings ““ Invalid: Nonempty strings Valid: No participation Valid: Low participation Valid: Moderate participation Valid: high participation Invalid: Above maximum	““ strings with one or several characters 0.0 0.1 – 49.9 50.0 – 79.9 80.0 – 100.0 >100.0
Created_by	Invalid: Empty string “” Invalid: Negative or zero Valid: valid user_	“” < 1

Table 5

INTEGER StudentID

Test Specifications

- Enter Student Data (studentID < 1)
- Enter Student Data (studentID: 1-999999)

Test Case

- enter studentData (studentID: -5) Invalid
- enter studentData(studentID: 0001) Valid

TEXT term

Test Specifications

- Enter Student Data (term =>> 0)
- Enter Student Data (term: standard format)

Test Case

- enter studentData (term: 1) Invalid
- enter studentData(term: “Fall 2025”) Valid

REAL attendance

Test Specifications

- Enter Student Data (attendance < 0)
- Enter Student Data (attendance: 0.1 – 99.9)

Test Case

- enter Student Data (attendance: -3.2) Invalid
- enter Student Data (attendance: 85.5) Valid

REAL studyHours

Test Specifications

- Enter Student Data (studyHours > 24)
- Enter Student Data (studyHours: 5.1 – 15. 0)

Test Case

- enter Student Data (sudyHours: 27) Invalid
- enter Student Data (studyHours: 10) Valid

REAL examScores

Test Specifications

- Enter Student Data (examScore < 0)
- Enter Student Data (examScore: 90.0 – 100.0)

Test Case

- enter Student Data (examScore: -89.0) Invalid
- enter Student Data (examScore: 95.5) Valid

INTEGER stressLevel

Test Specifications

- Enter Student Data (stressLevel < 1)
- Enter Student Data(stressLevel: 4 – 7)

Test Case

- enter Student Data (stressLevel: 0) Invalid
- enter Student Data (stressLevel: 7) Valid

REAL sleepHours

Test Specification

- Enter Student Data (sleepHours < 0)
- Enter Student Data (sleepHours: 6.1 – 9.0)

Test Case

- enter Student Data (sleepHours: -2.4) Invalid
- enter Student Data (sleepHours: 8.5) Valid

REAL participation

Test Specification

- Enter Student Data (participation > 100)
- Enter Student Data (participation 80.0 – 100.0)

Test Case

- enter Student Data (participation: 105) Invalid
- enter Student Data (participation:90) Valid

USE CASE 2: Student login

- Features: (**TEXT** email, **TEXT** password**, TEXT** role)
- Input Partition into equivalence classes for each feature:

Feature	Partition	Equivalence class
email	Invalid: empty string “” Invalid: missing @ symbol Invalid: missing domain valid: proper email format	“” string without @ user@, user@domain user@domain.com
password	Invalid: Empty string “” valid: minimum length valid: correct password	“” 8 characters matchers stored hash
role	Invalid: Empty string “” Invalid: user is ADMIN valid: User is STUDENT	Role = “” role = “ADMIN” role = “STUDENT”

Table 6

TEXT email

Test Specifications

- Student Login (email: empty/””)
- Student Login (email: valid format / user@domain.com)

Test Case

- Student Login (email: “”) Invalid
- Student Login (email: john@gmail.com) Valid

TEXT password

Test Specifications

- Student Login (password: empty/””)
- Student Login (password: minimum length)

Test Case

- Student Login (password: “”) Invalid
- Student Login (password: Tubalcaine) Valid

TEXT role

Test Specifications

- Student Login (role: empty/””)
- Student Login (role: STUDENT)

Test Case

- Student Login (role: “”) Invalid
- Student Login (role: where user role = “STUDENT”) Valid

6.2. Testing from A4

Framework Installation Steps

a) **Framework:** Playwright
b) **Programming Language:** JavaScript
c) **Steps:**
   - Initialize node project in the existing project using npm init -y
   - npm init playwright@latest
d) **How to run the test cases**
   - npm install
   - npx playwright install
   - npx playwright test –headed
   - Passed Test Case
   - npx playwright test –headed

Test Case 1: Student Login (valid)

Test ID	**TC-001-Student-Login-Valid**
Purpose of Test	To verify that a valid student user can successfully log in using correct credentials and the STUDENT role.
Test Environment	Chrome (latest), macOS Ventura, valid student account (Dhanush@university.edu / Admin).
Test Steps	1. Open http://localhost:8501 2. Enter email

	3. Select STUDENT role 4. Enter password 5. Click Sign in 6. Verify dashboard
Test Input	Email: Dhanush@university.edu Password: Admin Role: STUDENT
Expected Result	Student is logged in and sees 'Signed in as Dhanush@university.edu (STUDENT)'.
Likely Problems/Bugs Revealed	missing confirmation message.

Table 7

Test Case 2: Student Login (Invalid)

Test ID	**TC-002-Student-Login-Invalid**
Purpose of Test	To verify login fails with invalid email or password and error message is shown.
Test Environment	Chrome (latest), macOS Ventura, no account for invalidUser@university.edu.
Test Steps	1. Open http://localhost:8501 2. Enter invalid email 3. Select STUDENT 4. Enter wrong password 5. Click Sign in 6. Check error message
Test Input	Email: invalidUser@university.edu Password: WrongPassword123 Role: STUDENT
Expected Result	System displays 'User not found.'
Likely Problems/Bugs Revealed	missing errors, wrong validation.

Table 8

Test Case 3: Admin Login (Valid)

Test ID	**TC-003-Admin-Login-Valid**
Purpose of Test	To verify Admin can log in and UI identifies user as ADMIN.
Test Environment	Chrome (latest), macOS Ventura, admin account ravi@university.edu.
Test Steps	1. Open app 2. Enter admin email 3. Enter password

	4. Click Sign in 5. Verify email and (ADMIN) label
Test Input	Email: ravi@university.edu Password: Admin
Expected Result	Admin sees email displayed and '(ADMIN)' label.
Likely Problems/Bugs Revealed	No

Table 9

Test Case 4: Teacher Predict & Performance Entry

Test ID	**TC-004-Teacher-Predict-Student-Performance**
Purpose of Test	Verify Teacher can log in, select Predict, choose student Dayo (#005), fill performance form, and save.
Test Environment	Chrome latest, macOS Ventura, teacher account Steve@university.edu, student list with Dayo (#005).
Test Steps	1. Log in as Teacher 2. Select Predict 3. Open dropdown 4. Select Dayo (#005) 5. Fill fields 6. Move stress slider 7. Save performance
Test Input	Teacher: Steve@university.edu Password: Admin Student: Dayo (#005) Various performance values
Expected Result	Performance saved and Student Profile & Performance section remains visible.
Likely Problems/Bugs Revealed	No

Table 10

6.3. Test Documentation

Bug	The test uncovered the bug	Description of the bug	Action was taken to fix the bug
B1	Test Case: Sign in button enabled with empty username and password	The login UI allowed the Sign in button to remain enabled even when both the username and password fields were empty or null. Button should remain disabled until both fields contain valid input.	implemented the frontend validation to disable the Sign in button when either username or password is empty/null.
B2	Test Case: Username is case-sensitive during login	A user account created with the username for e.g “Steve” cannot log in when the user enters “steve” in lowercase. The system treats the username field as case-sensitive, causing valid users to be unable to authenticate when using different letter casing.	Updated the authentication logic to make username comparisons case insensitive. Implemented normalization to convert both stored and entered usernames to a consistent format before validation.
B3	Test Case: Dropdown menus display empty string as the first option	Several dropdown fields (Gender, Extracurricular Activities, Internet Access at Home, Parent Education Level, Family Income Level) display an empty string as the first selectable option. This causes users to mistakenly submit invalid selections and leads to inconsistent input validation.	Implemented a required placeholder and block empty dropdown submissions

Table 11

7. Appendix

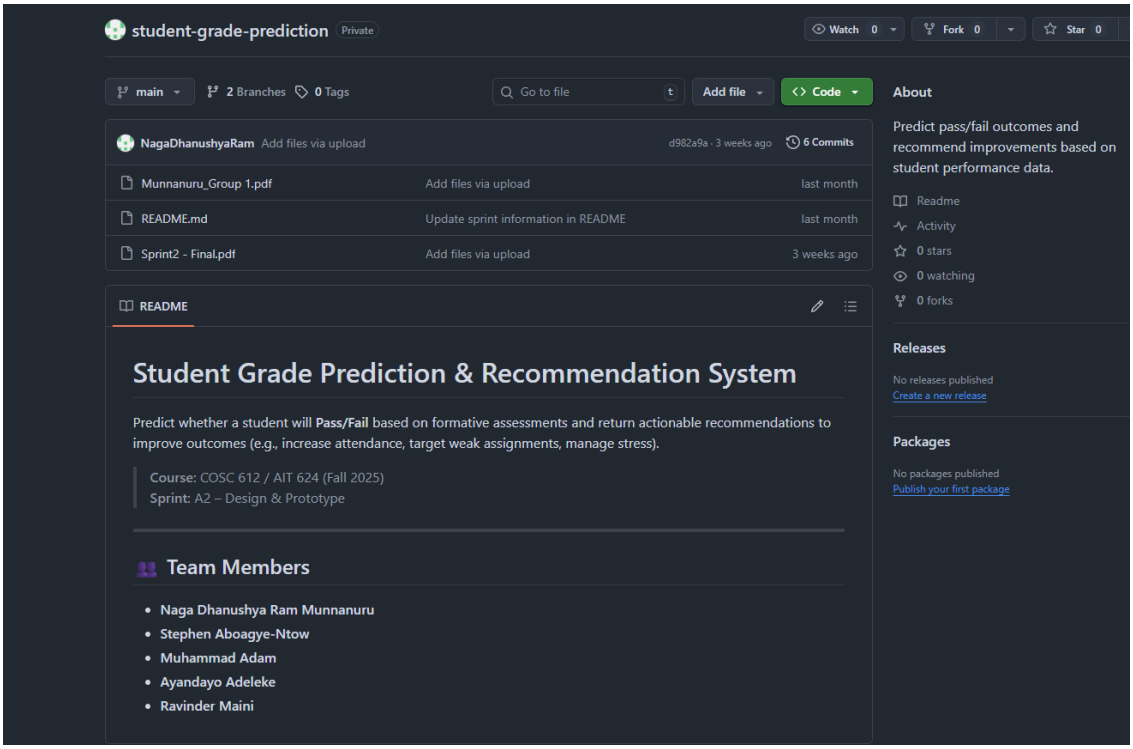


Fig. 11 – GitHub Repository Homepage

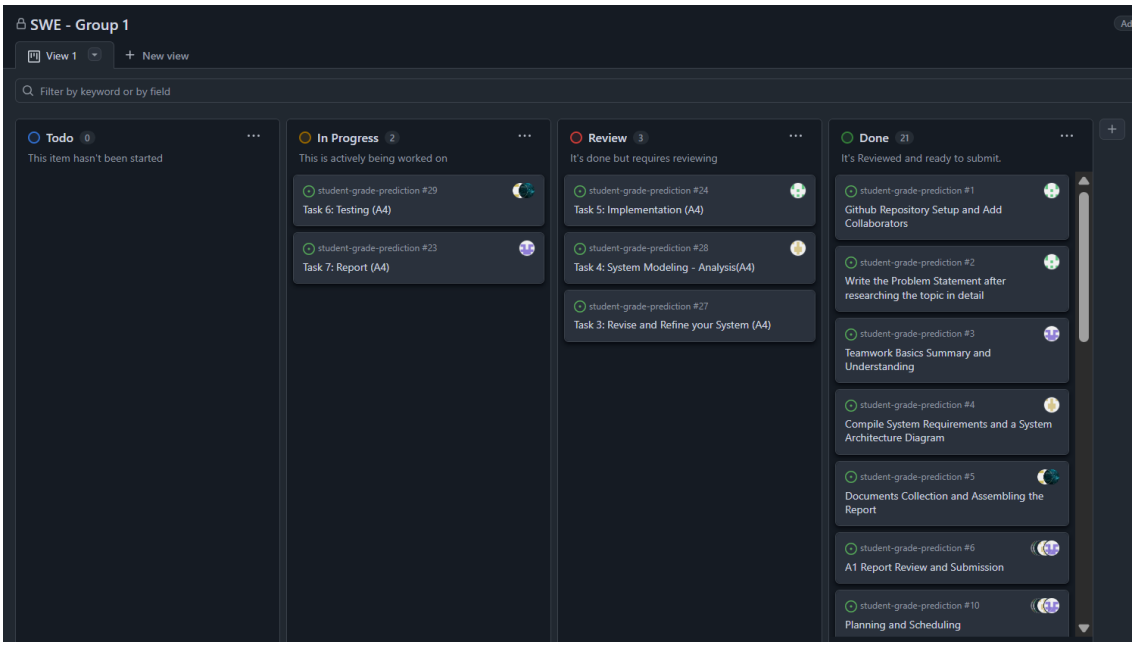


Fig. 12 – Kanban Board on GitHub

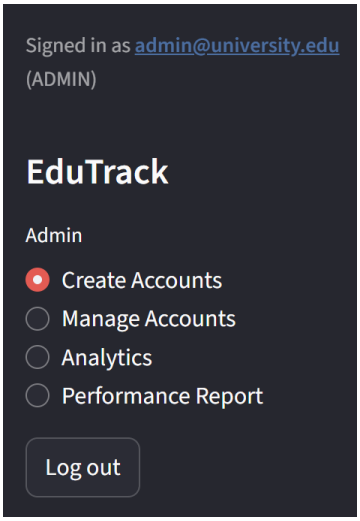
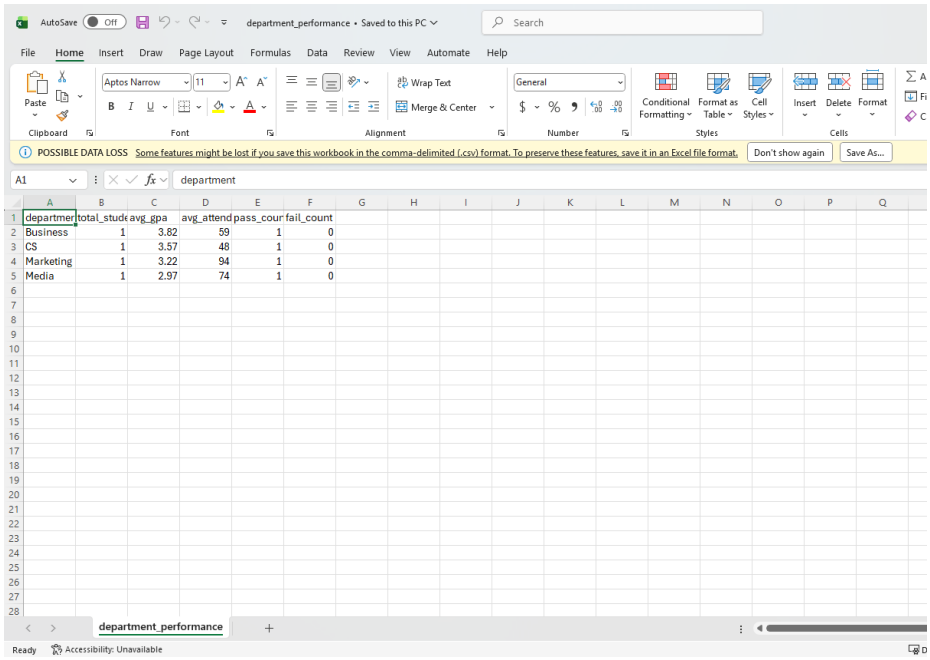


Fig. 13 - Updated Menu in Admin Dashboard which shows "Performance Report Page"

Fig. 14 - Performance Report Page of the website currently



department	total_stude	avg_gpa	avg_attend	pass_cour	fail_count
Business	1	3.82	59	1	0
CS	1	3.57	48	1	0
Marketing	1	3.22	94	1	0
Media	1	2.97	74	1	0

Fig. 15 - The downloaded CSV of the department performance report

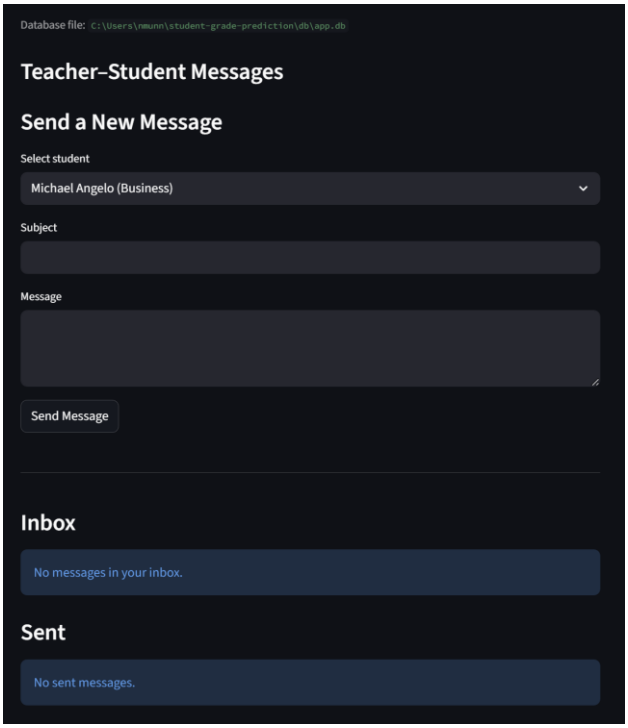


Fig. 16 - Message page of Teacher Dashboard before sending any messages

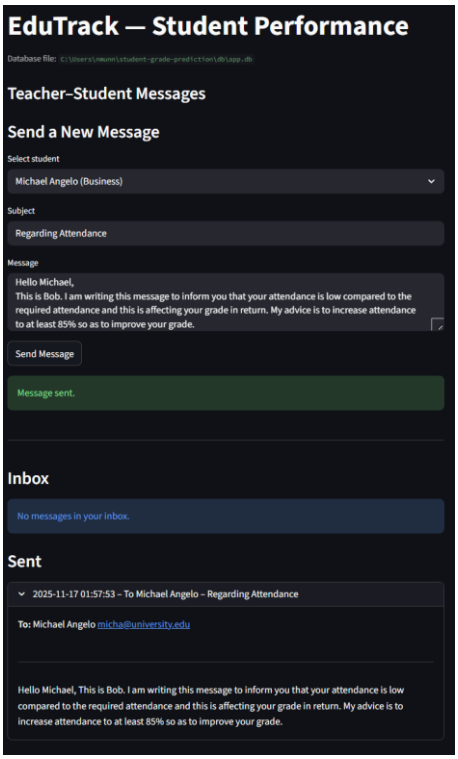


Fig. 17 - Message Page after sending a message from Teacher to Student.

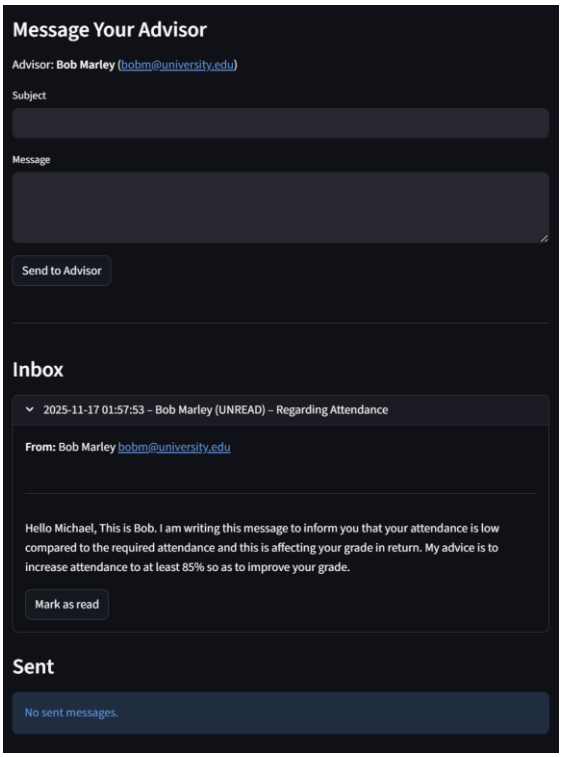


Fig. 18 - Message Page from Student Dashboard before clicking on "Mark as Read"

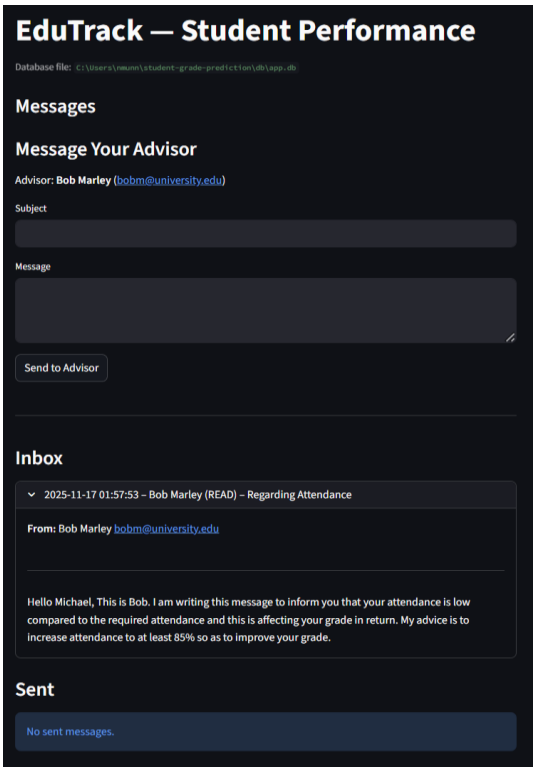


Fig. 19 - Message page from Student Dashboard after clicking on "Mark as Read"

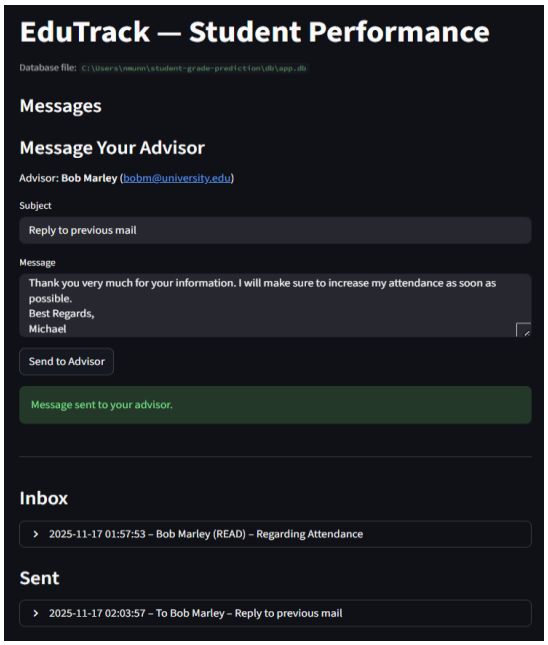


Fig. 20 - Message page from Student Dashboard after sending a reply to advisor

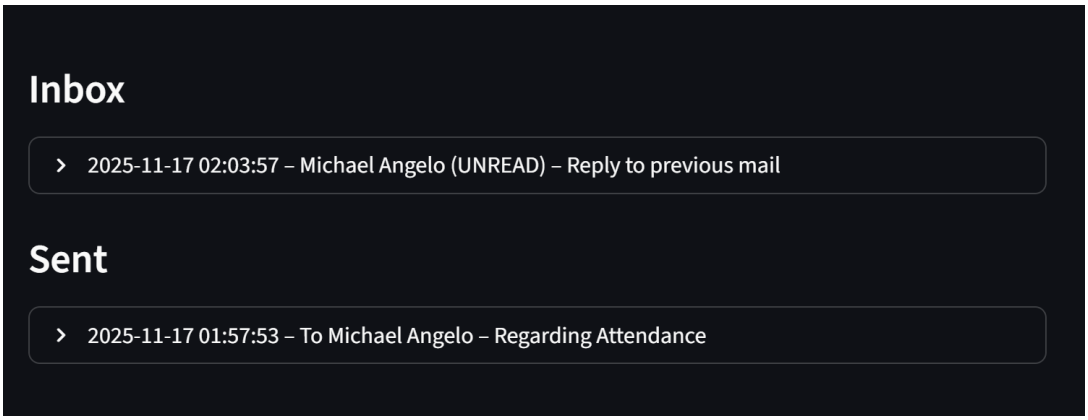


Fig. 21 - Inbox of Teacher after the student sends a message

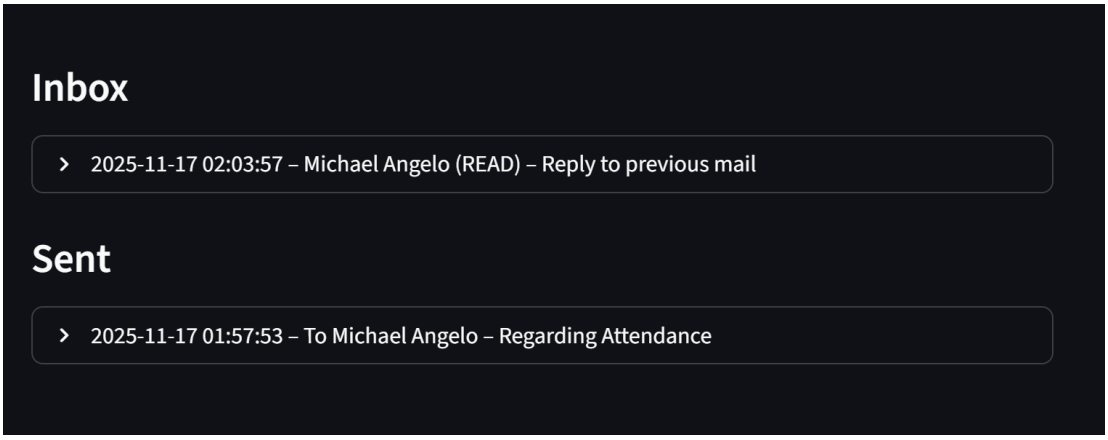
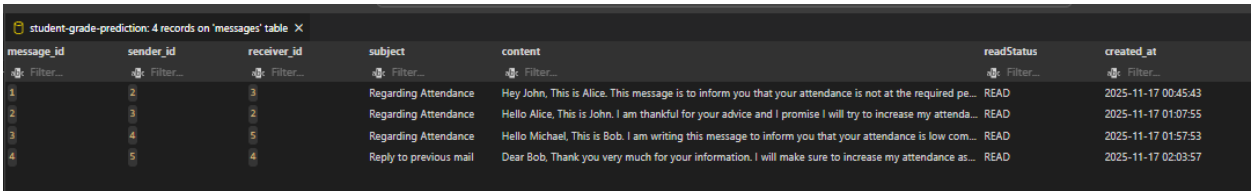


Fig. 22 - Inbox of Teacher after clicking on "Mark as Read" (Changes from (UNREAD) to (READ))


student-grade-prediction: 4 records on 'messages' table

message_id	sender_id	receiver_id	subject	content	readStatus	created_at
1	2	3	Regarding Attendance	Hey John, This is Alice. This message is to inform you that your attendance is not at the required pe...	READ	2025-11-17 00:45:43
2	3	2	Regarding Attendance	Hello Alice, This is John. I am thankful for your advice and I promise I will try to increase my attenda...	READ	2025-11-17 01:07:55
3	4	5	Regarding Attendance	Hello Michael, This is Bob. I am writing this message to inform you that your attendance is low com...	READ	2025-11-17 01:57:53
4	5	4	Reply to previous mail	Dear Bob, Thank you very much for your information. I will make sure to increase my attendance as...	READ	2025-11-17 02:03:57

Fig. 23 - Database Updation in the messages table which shows all the messages sent and received including sender_id and receiver_id

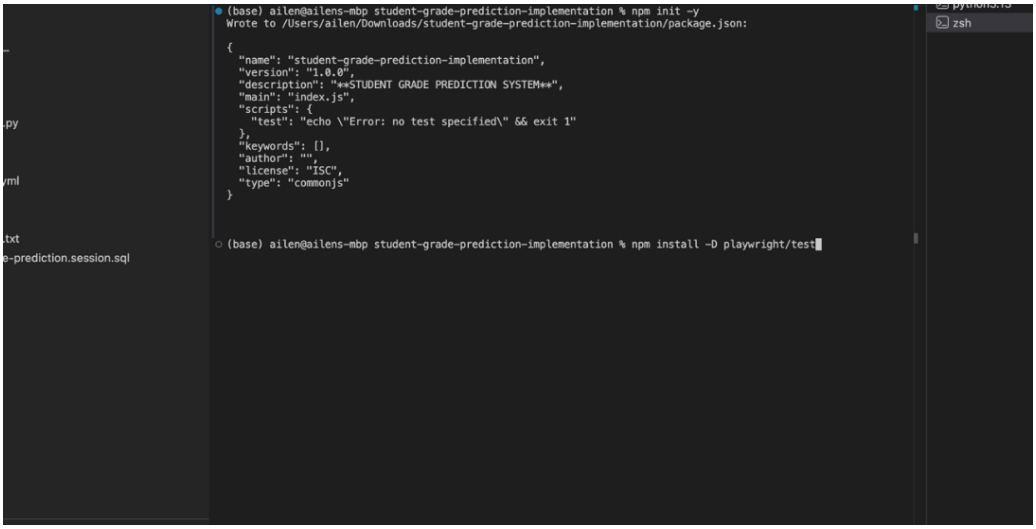


Fig. 24 Initalize node project in the existing project using npm init -y

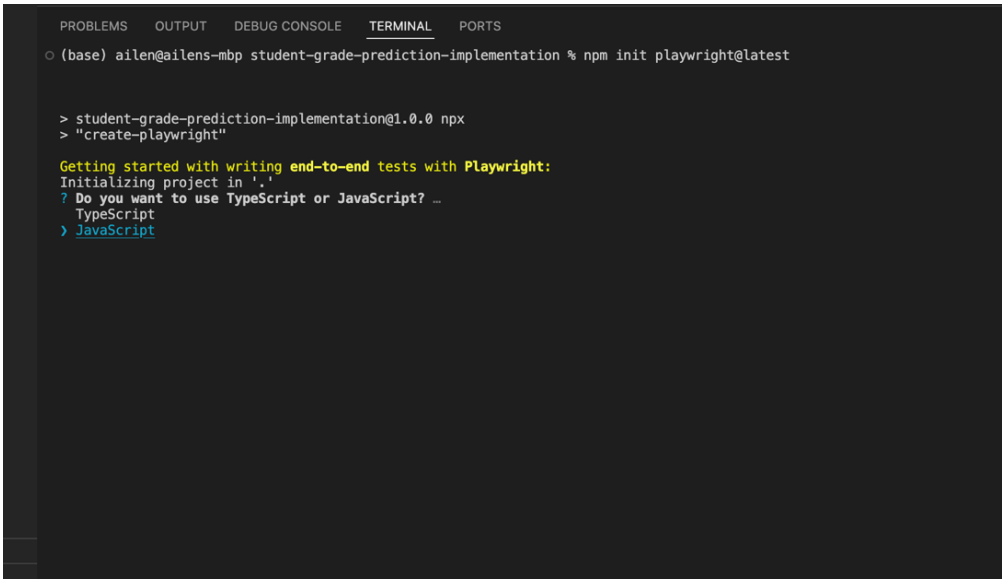


Fig. 25 npm init playwright@latest

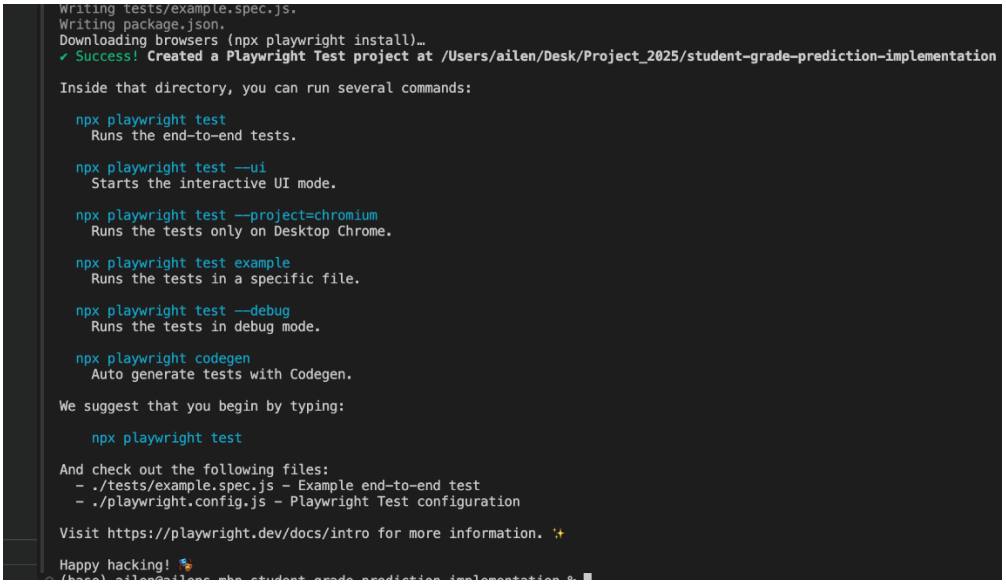


Fig. 26 npm init playwright@latest

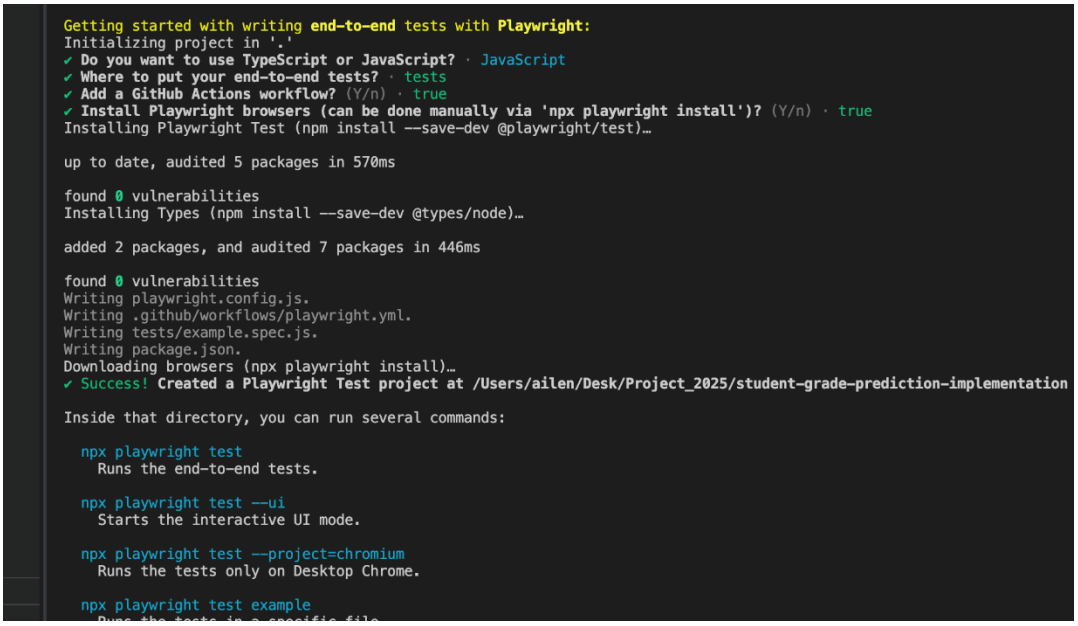


Fig. 27 npm init playwright@latest

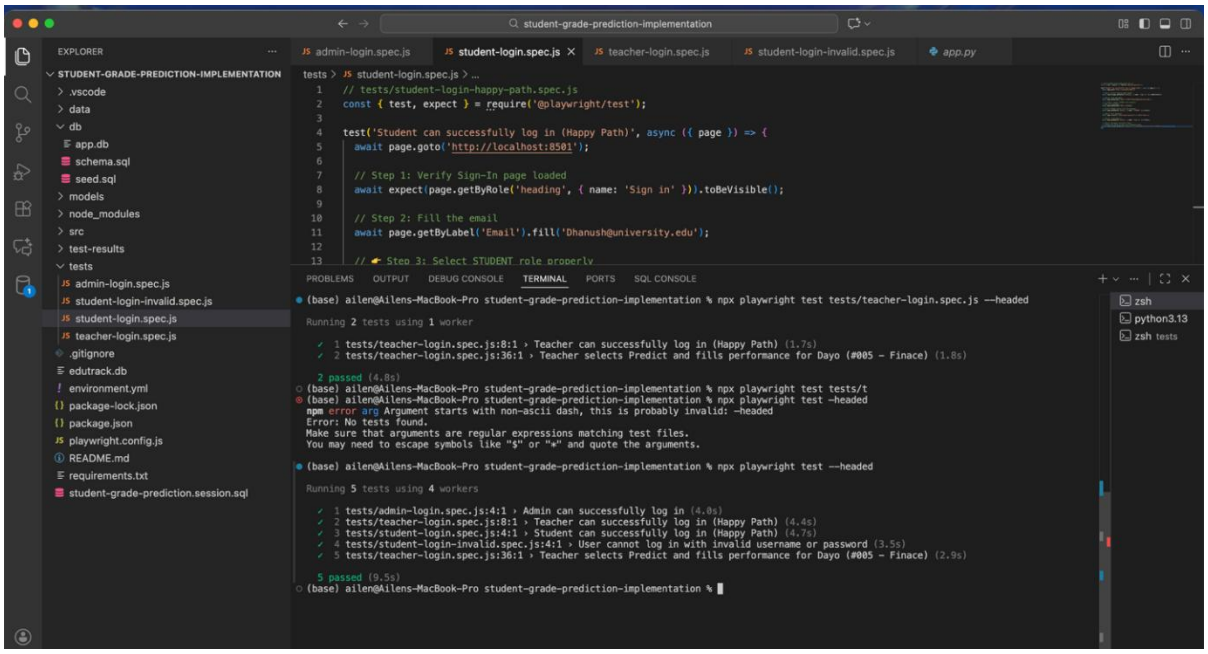


Fig. 28 Playwright Test Execution Results

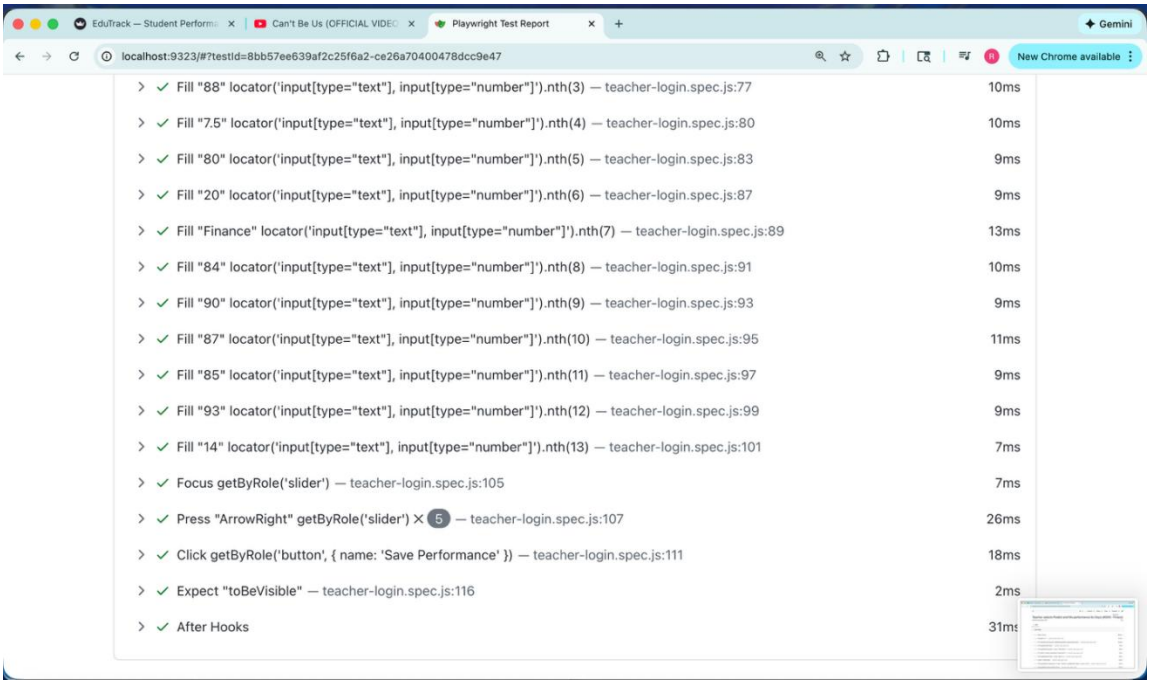


Fig 29. Playwright Test Report

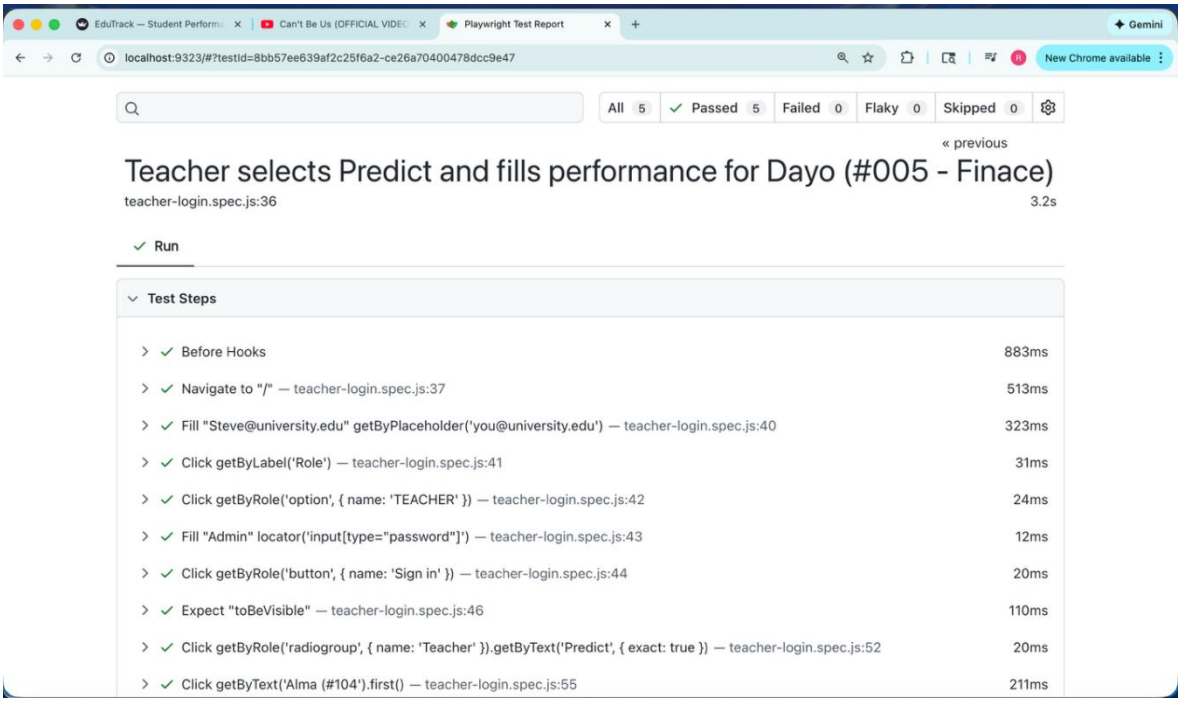


Fig. 30 Playwright test report – Teacher Selects Predict and fills performance

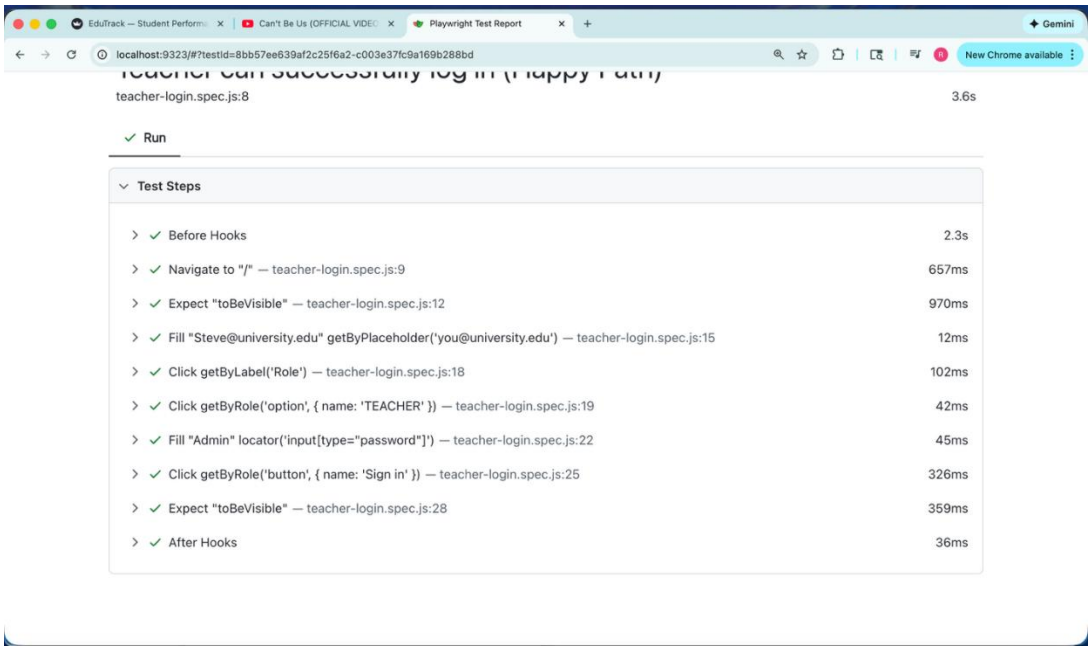


Fig. 31 Playwright Test Report – Teacher can successfully log in

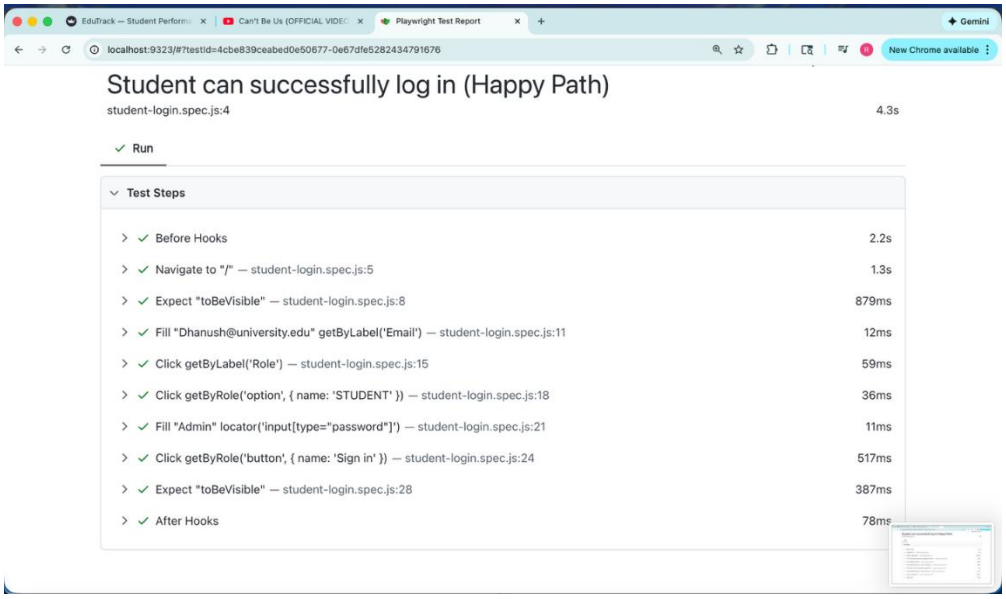


Fig. 32 Playwright Test Report – Student can successfully log in

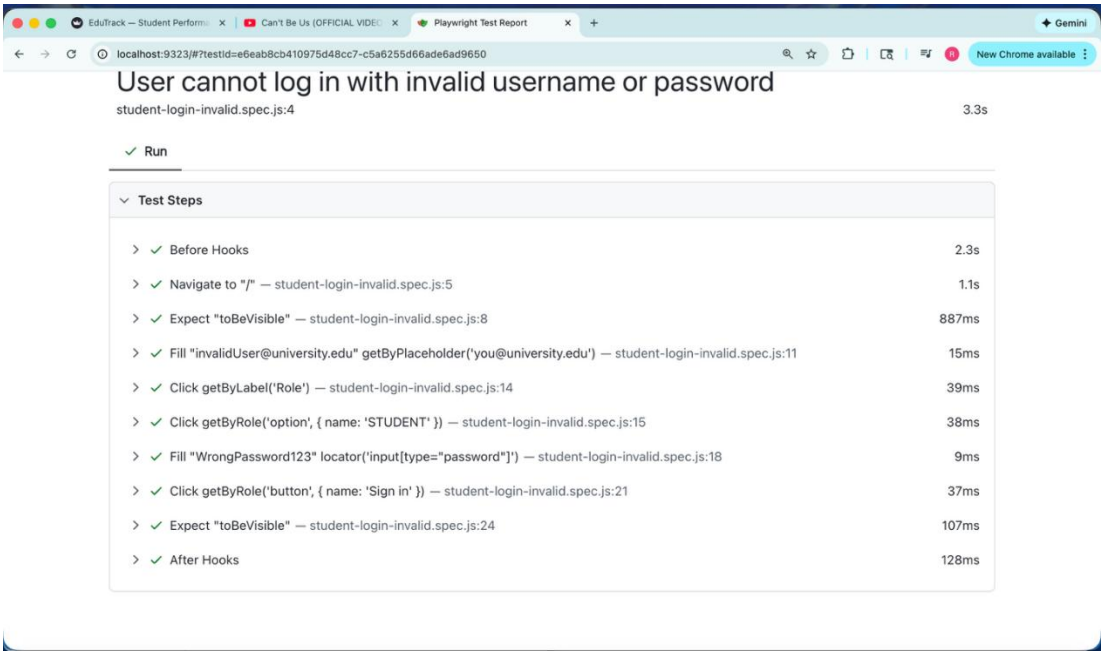


Fig. 33 User cannot log in with invalid username or password

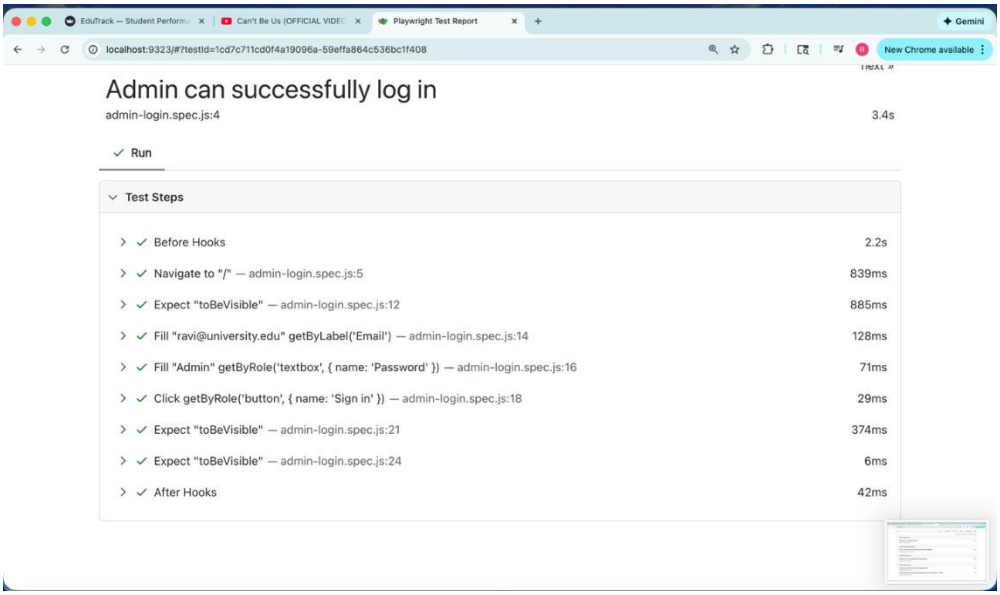


Fig. 34 Playwright Test Report – Admin can successfully log in

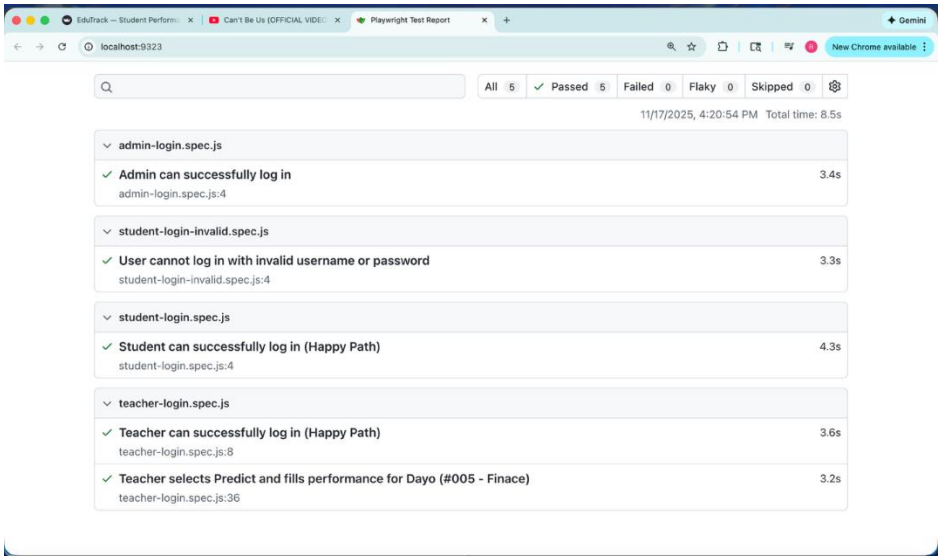


Fig. 35 Playwright Test Report