IIT Madras

SOFTWARE ENGINEERING PROJECT

May2025-Team22

MILESTONE-6

Prepared by :

Deepak Kumar Singh (21f1006490)
Kumar Rishabh (23f1000391)
Aarush Saxena (23f2005702)
Rahul Sharma (23f1001879)
Arya Chavan (21f1002763)
Nikhil Sai Kasireddy (21f3002651)
Gopu Vishal Reddy (21f2001548)

IITM BS DEGREE PROGRAM

CONTENTS

ABOUT THE PROJECT

This project aims to cater requirements for a children aged 8-14 through Life Skills App. By engaging with students, parents, and educators, the project aims to create an engaging, age-appropriate solution that builds essential life skills, such as time management, emotional intelligence, and healthy habits, while fostering independence and real-world application.

TEAM MEMBERS

Deepak Kumar Singh

Kumar Rishabh

Rahul Sharma

Aarush Saxena

Arya Chavan

Gopu Vishal Reddy

Nikhil Sai Kasireddy

MILESTONE-1

IDENTIFY USER REQUIREMENT

IITM BS DEGREE PROGRAM

1. IDENTIFYING USERS

1.1 User Requirements

The goal is to define user requirements for a Life Skills App for children aged 8–14. This involves understanding the needs of primary, secondary, and tertiary users, and developing user stories that inform the app's features. By engaging with students, parents, and educators, the project aims to create an engaging, age-appropriate solution that builds essential life skills, such as time management, emotional intelligence, and healthy habits, while fostering independence and real-world application.

1.2 User Categorization

PRIMARY USERS

School-Aged Children (8-14 years)

- Core users who engage with skill-building activities
- Participate in progress tracking features

SECONDARY USERS

Parents/Guardians & Teachers

- Monitor child progress and performance
- Manage screen time and usage
- Receive reports on student achievements
- Support student learning and development

TERTIARY USERS

Administrators

- Oversee user management and access
- Manage content, settings, and compliance
- Ensure smooth operation and regulatory adherence

USER STORIES

TIME MANAGEMENT

FOR STUDENTS:

01 As a student, I want to set reminders for my homework and chores, so that I can manage my time better.

02 As a student, I want to track my daily and weekly goals, so that I can see my progress and stay motivated.

03 As a student, I want to use a timer for study sessions, so that I can focus and take regular breaks.

FOR PARENTS:

01 As a parent, I want to view my child's task completion and time management progress, so that I can encourage good habits at home.

02 As a parent, I want to receive notifications if my child misses important deadlines, so that I can offer timely support.

FOR TEACHERS:

01 As a teacher, I want to assign tasks or projects to my students through the app, so that I can help them learn to plan ahead.

02 As a teacher, I want to monitor student's time management skills, so that I can provide guidance and feedback.

COMMUNICATION SKILLS

FOR STUDENTS:

01 As a student, I want to practice conversation scenarios with a virtual coach, so that I can improve my speaking and listening skills.

02 As a student, I want to get feedback on my communication exercises, so that I can learn from my mistakes.

03 As a student, I want to participate in group chat activities, so that I can build confidence in social settings.

FOR PARENTS:

01 As a parent, I want to see my child's progress in communication modules, so that I can support their development at home.

02 As a parent, I want to receive suggestions for conversation starters, so that I can help my child practice communication skills.

FOR TEACHERS:

01 As a teacher, I want to assign group communication exercises, so students can practice teamwork and empathy.

01 As a teacher, I want to review student's performance in communication activities, so I can provide targeted feedback.

INTERVIEW

Student 1: *Interview Link*

Student 2: *Interview Link*

Parent/Guardian: *Interview Link*

Teacher: *Interview Link*

IIT Madras

MILESTONE-2

USER INTERFACES

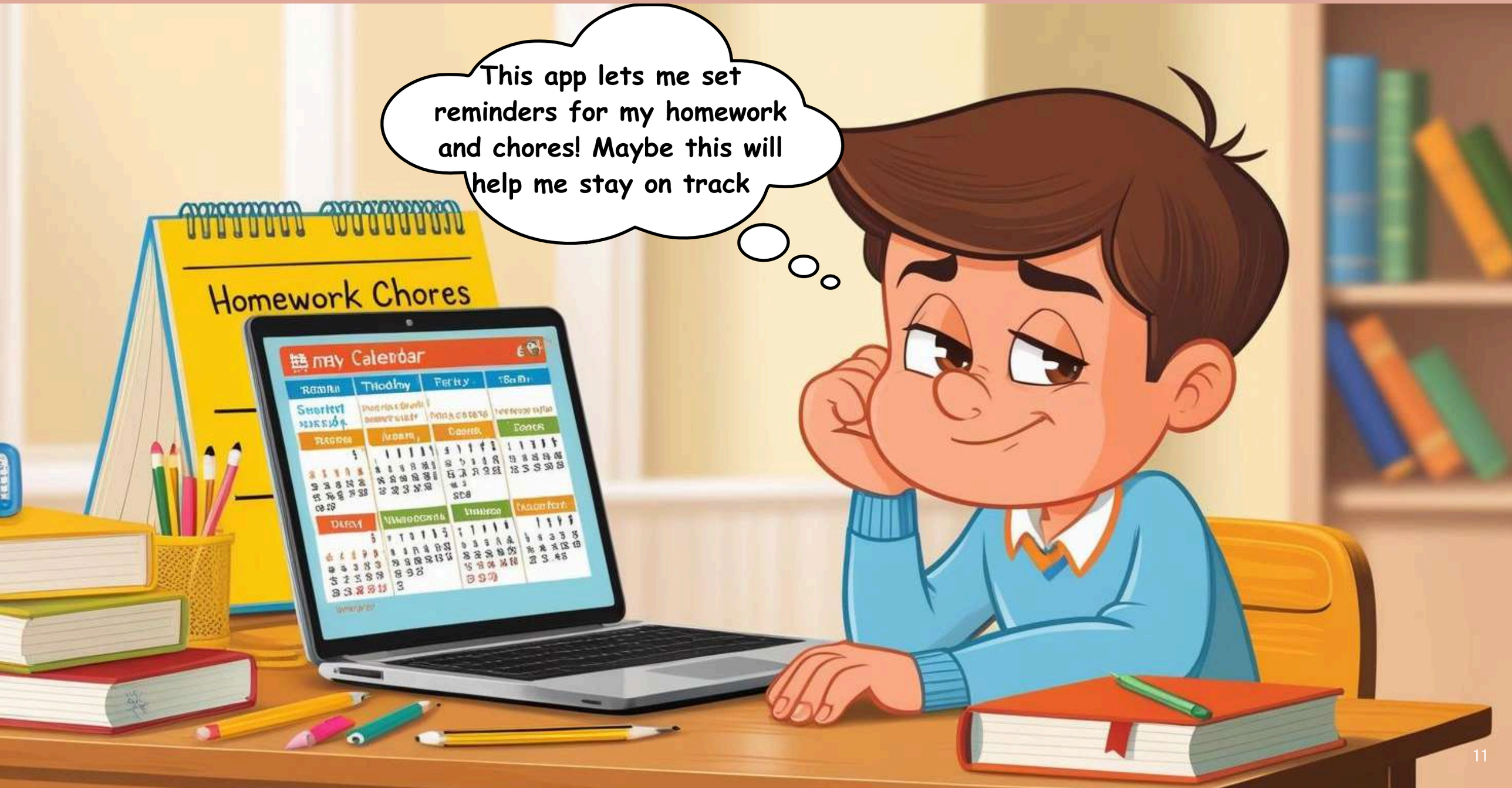
STORYBOARD

This storyboard visually demonstrates how our app supports students, parents, and teachers in building essential time management and communication skills. Each panel captures a real-world scenario inspired by each user's stories, highlighting how intuitive digital tools, such as reminders, progress trackers, virtual coaches, and feedback modules, empower users to develop good habits, foster collaboration, and achieve their goals.

The storyboard and the pages preceding the wireframes represent the app's original design direction prior to conducting user interviews, offering a baseline that informed the updated, user-centred wireframes.

Student willing to sets reminders to manage homework and chores

A student happily checking off goals on a digital checklist on a laptop.



My Calendar

Stay organized and never miss a task!

‹ **June 2025** ›

Add new reminder... | dd-mm-yyyy | Set Reminder

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12 Prepare for quiz	13	14 Math Homework Due
15	16	17	18 Science Project	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Student uses a timer to focus during study sessions.

Good morning, Learner

Here's a quick glance at your progress

Knowledge base

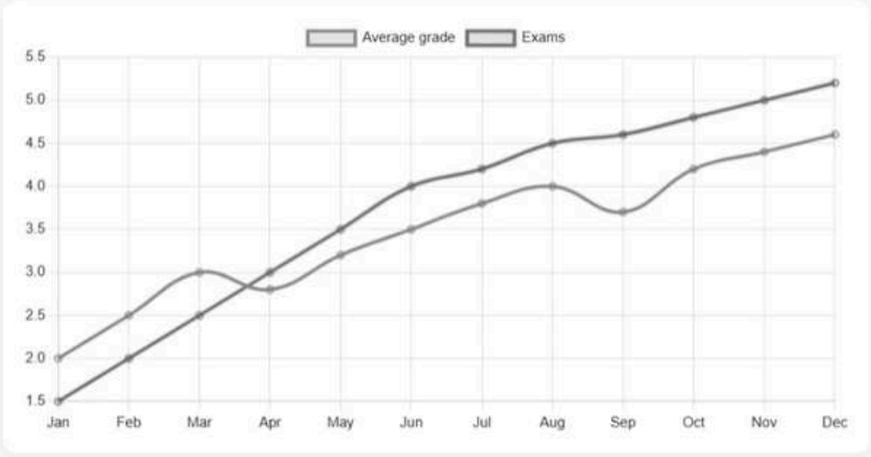
Vocabulary | Phrasal verbs | Grammar

Cancel Focus Mode

Stats

Classes attended	Assignments submitted	Tests taken
0	0	0

Statistics



Focus Mode

25 Enter Focus Mode

Calendar

Today: 11 Jun 2025 | 02:35 am

Upcoming Classes

- 10:00 AM - English
- 12:00 PM - Math
- 3:00 PM - Science

See more

Focus Mode — Time Left: **24:10**

Student practices conversation scenarios with AI assistant.

Good morning, Learner 👋

LifeSkills - AI ChatBot

Student:

Hi, can you help me prepare for my upcoming math test?

Bot:

Of course! Let's review some key math concepts and practice a few problems together.

Student:

That sounds great. Can we also go over some tips for staying focused while studying?

Bot:

Absolutely! Taking short breaks, setting clear goals, and practicing regularly can really help you stay focused.

Student:

Thanks! Also, I'm struggling a bit with algebra. Any advice?

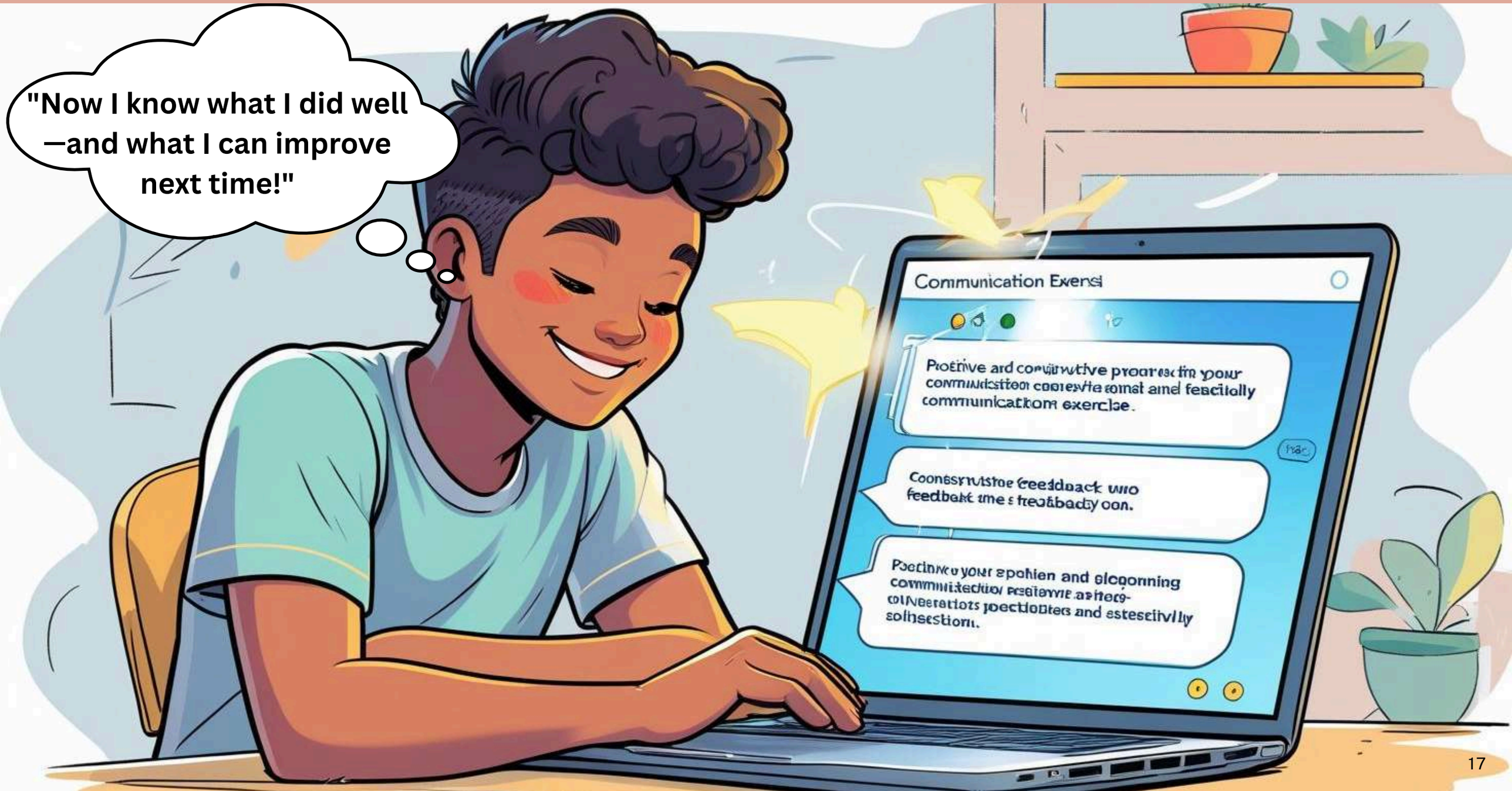
Bot:

For algebra, try to break down each problem into smaller steps and practice solving equations daily. Would you like a sample problem to try?

Ask anything...

Send

Student receives notifications and feedback on various exercises.



Messages & Feedback

Stay updated with important notifications and feedback.

Assignment Feedback: Essay on Climate Change

Feedback

Great structure and strong arguments. You've shown excellent improvement in forming cohesive paragraphs.

From: Mrs. Sharma • Mon, 9 Jun

Upcoming Workshop: Debate & Public Speaking

Notification

You are enrolled for the debate skills workshop scheduled this weekend. Be prepared!

From: Admin Team • Sat, 7 Jun

New Assignment Posted: Math Practice Set 5

Announcement

Please complete the algebra and geometry sections before Friday. Let me know if you face any doubts.

From: Mr. Verma • Fri, 6 Jun

Participation Feedback: Science Fair

Feedback

Your model was well-constructed and creative. Excellent effort in explaining the concept to the visitors.

From: Dr. Rao • Tue, 3 Jun

Teacher assigns tasks or projects to students via the app.

Assignments

Manage, track and assign tasks to your students

Create New Assignment

Title

Description

dd-mm-yyyy

Assign

All Assignments

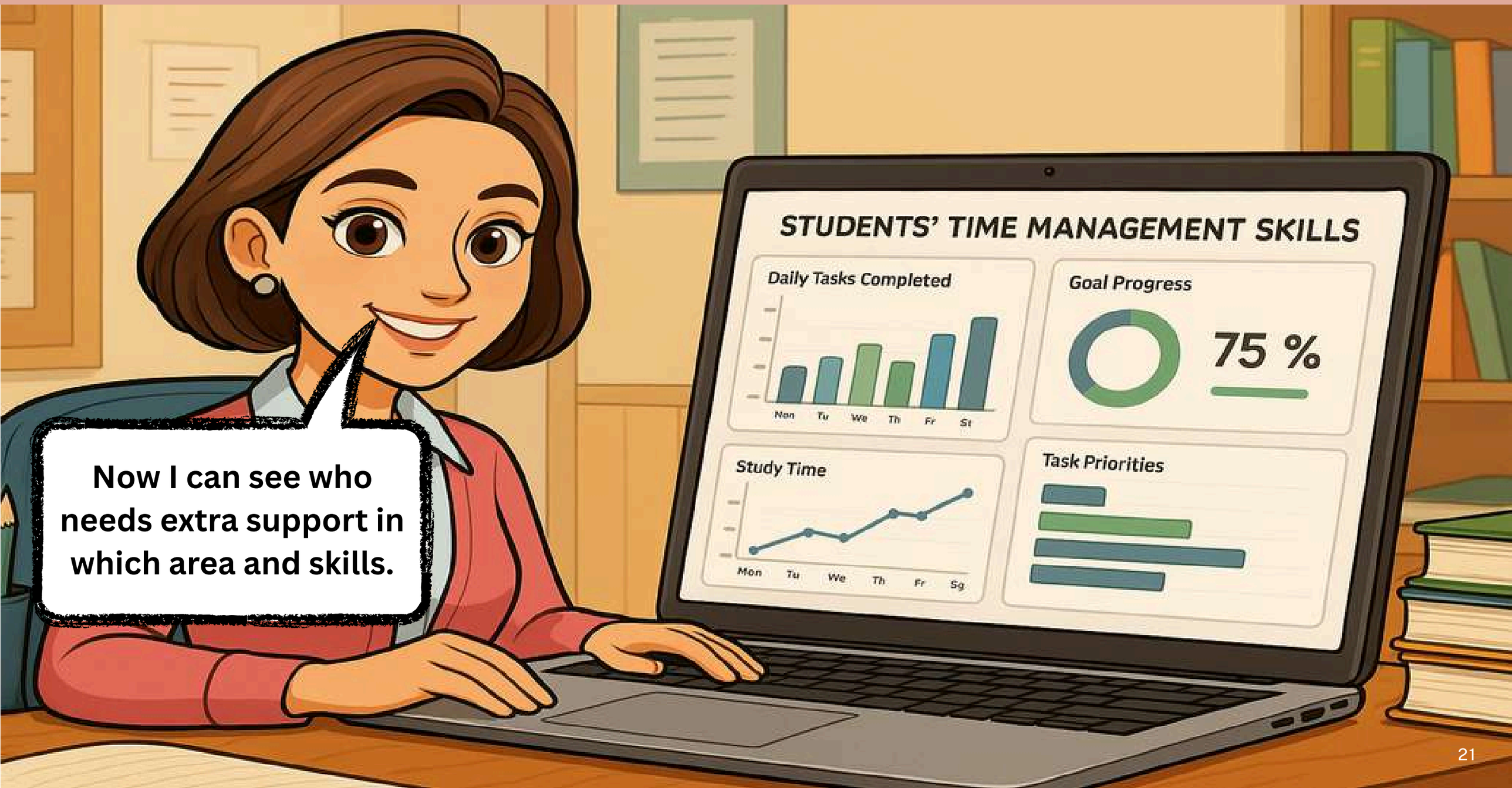
Data Science Project

Learning AI basics with IIT BS LLM Course :)

Due: 12/6/2025

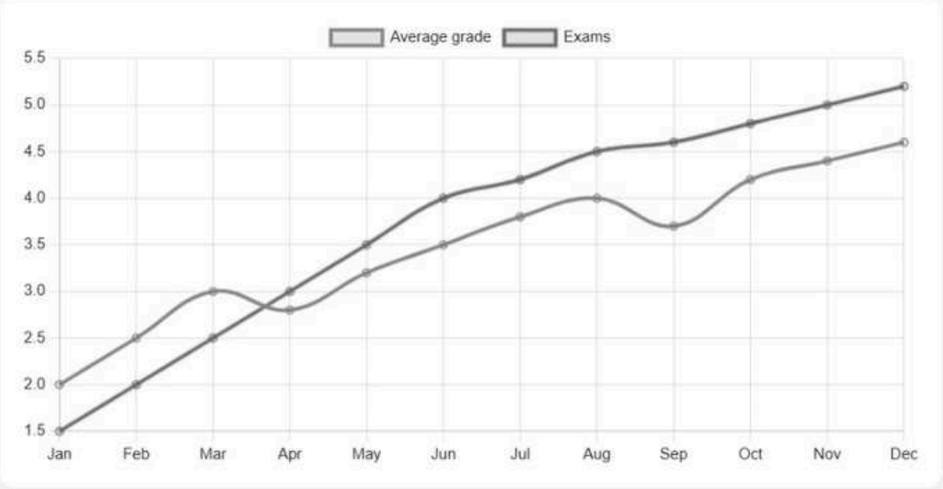
Student	Status
Rahul Sharma	Completed
Arya Chavan	In Progress
Aarush Saxena	Not Started
Gopu Vishal Reddy	Completed

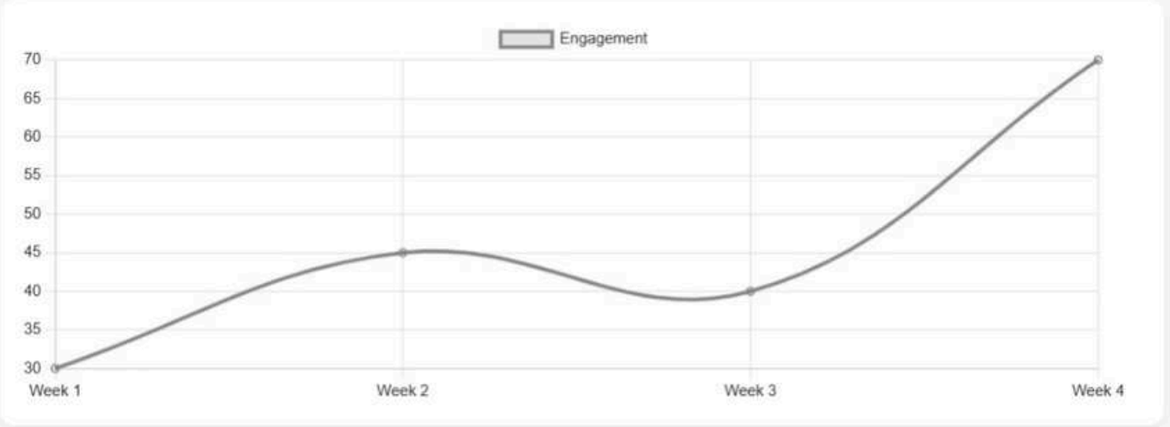
Teacher reviews students’ progress in each skills.



Welcome back, Instructor 👋🏾

Here's a quick glance at your learners





Time Management
43% Progress

Communication
67% Progress

Health & Wellness
56% Progress

Decision Making
32% Progress

Calendar

Today: 11 Jun 2025 | 03:25 am

Upcoming Classes

- 10:00 AM - English
- 12:00 PM - Math
- 3:00 PM - Science

See more

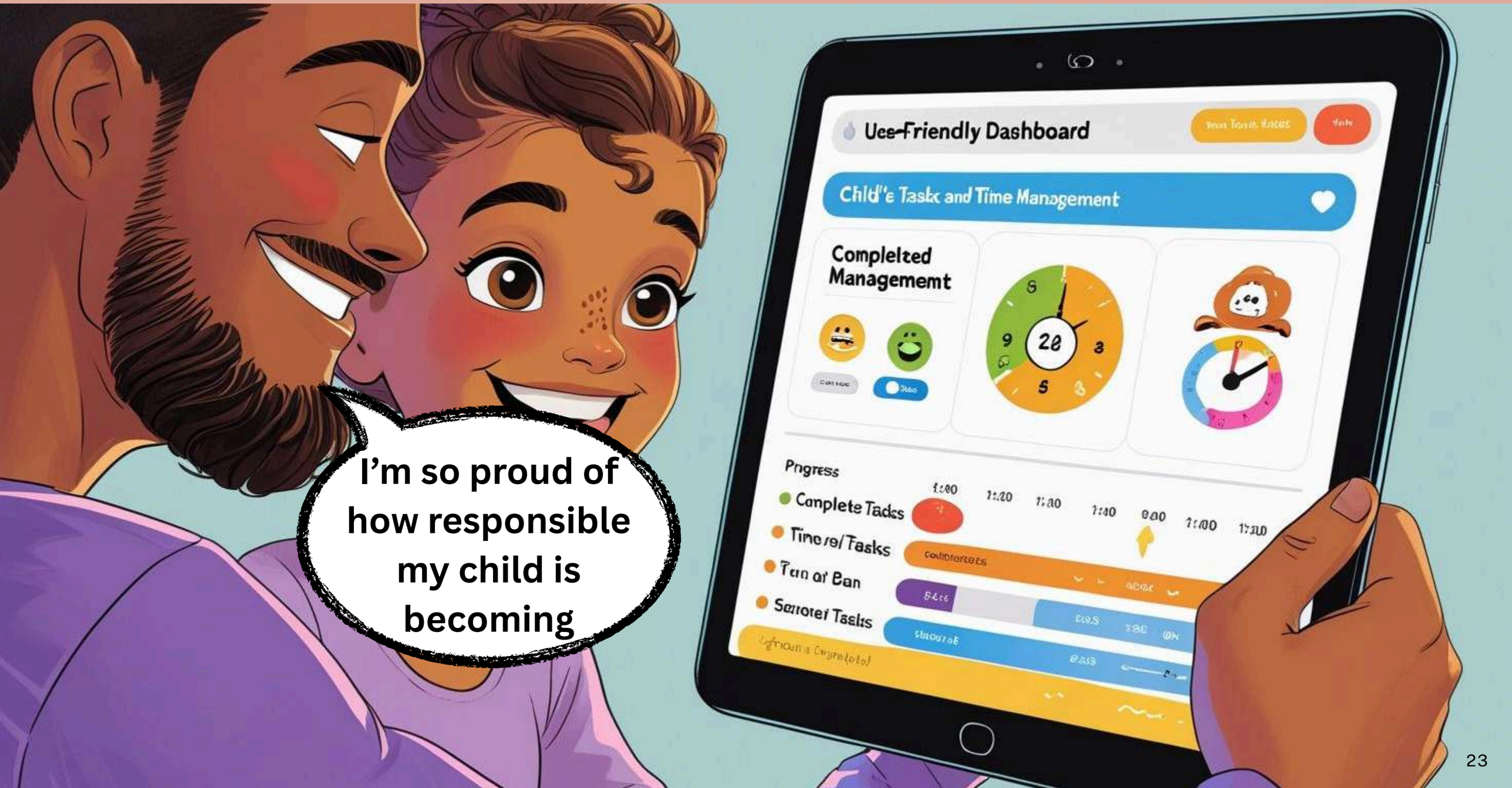
Staff Room

A. Adedayo
Hey, please check analytics
10:25 am

B. Omile
Meeting tomorrow @ 3pm
12:35 pm

E. John
Update session notes
04:30 pm

Parent views their child’s task completion and progress.



Good afternoon, Parent

Here's an overview of your child's learning journey

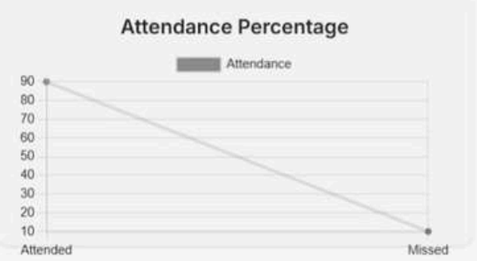
Child Profile

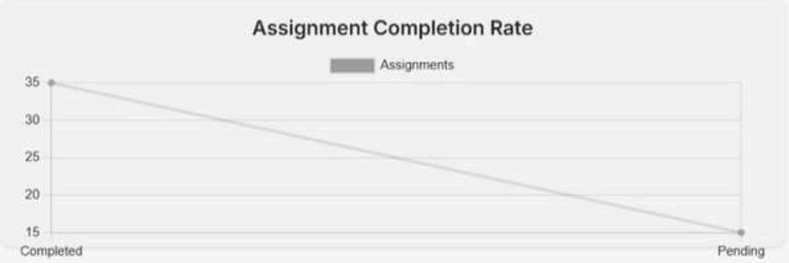
Lil Aarush
Grade: 5th Grade
School: Sunshine and Rainbow

Child Stats

Classes Attended	Assignments Submitted	Tests Taken	Attendance %
42	35	12	90

Progress Charts of your kid





Achievements your lil kid earned

- Top Scorer
- Homework Hero
- Focus Master
- Attendance Star

Calendar

Today: 11 Jun 2025 | 02:49 am

Upcoming Classes

- 10:00 AM - English
- 12:00 PM - Math
- 3:00 PM - Science

See more

Notifications

- New assignment uploaded for Math.
- Parent-teacher meeting scheduled for next Friday.
- Your child earned a new badge: Homework Hero.

Recent Messages

Teacher: Alex is improving well on spelling.

Coach: Keep encouraging Alex in sports activities!

Parent receives notifications about missed deadlines.

Messages & Reports

Stay updated on your child's academic journey

All | Notifications | Reports

Monthly Report Available

Your child's May performance report is ready.

1 Jun 2025

Parent Meeting Reminder

Virtual PTM scheduled for June 15th at 6 PM.

10 Jun 2025

New Assignment Notification

English project submitted successfully.

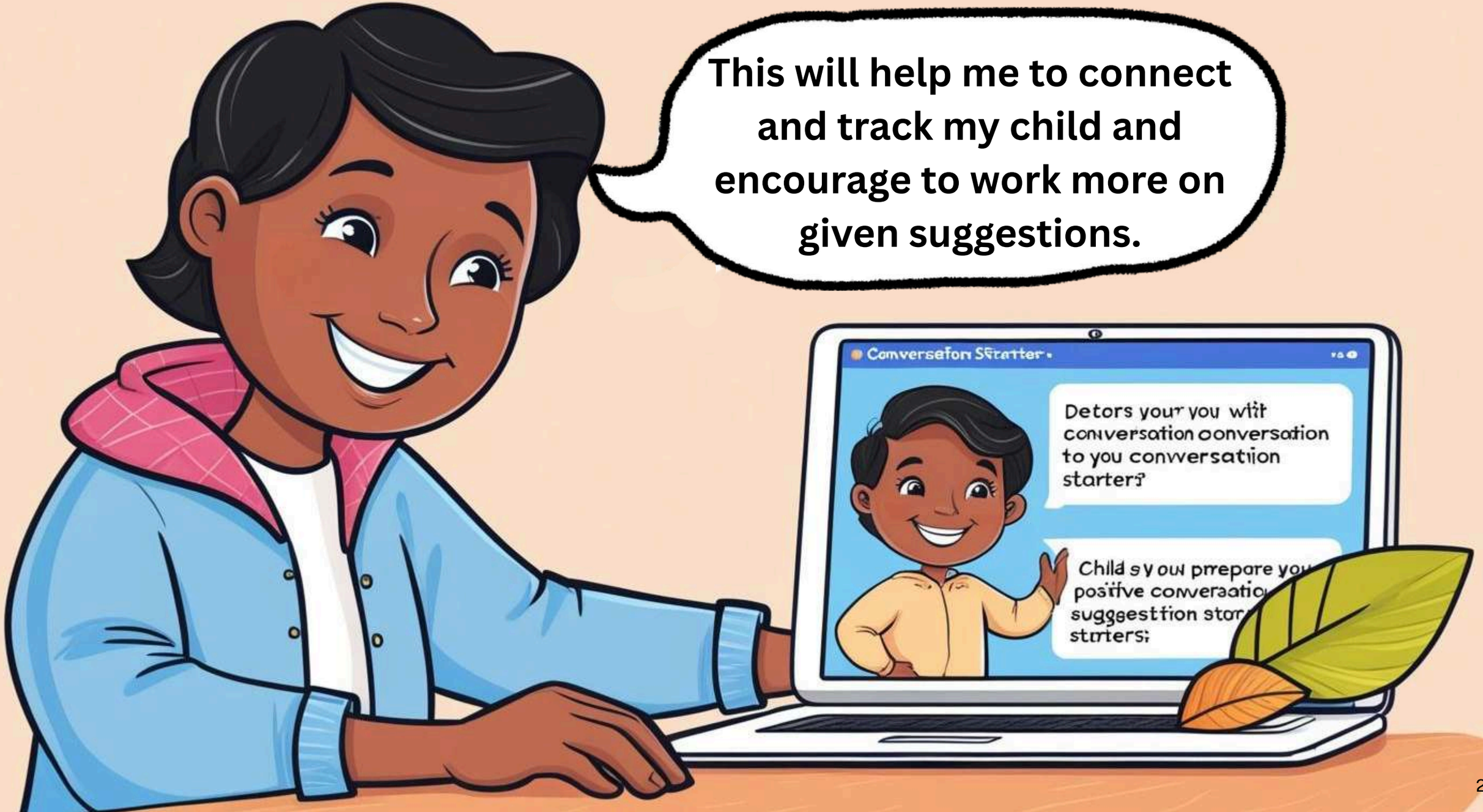
9 Jun 2025

April Report Available

April monthly report now accessible.

2 May 2025

Parent can do conversation with teachers to support their child.



Parent Comms Center

Direct and secure messaging

Chats

Mrs. Davis
Mathematics

Mr. Smith
History

Mrs. Davis
Mathematics

Hi, I'd like to discuss Nikhil's progress.

Type a message...

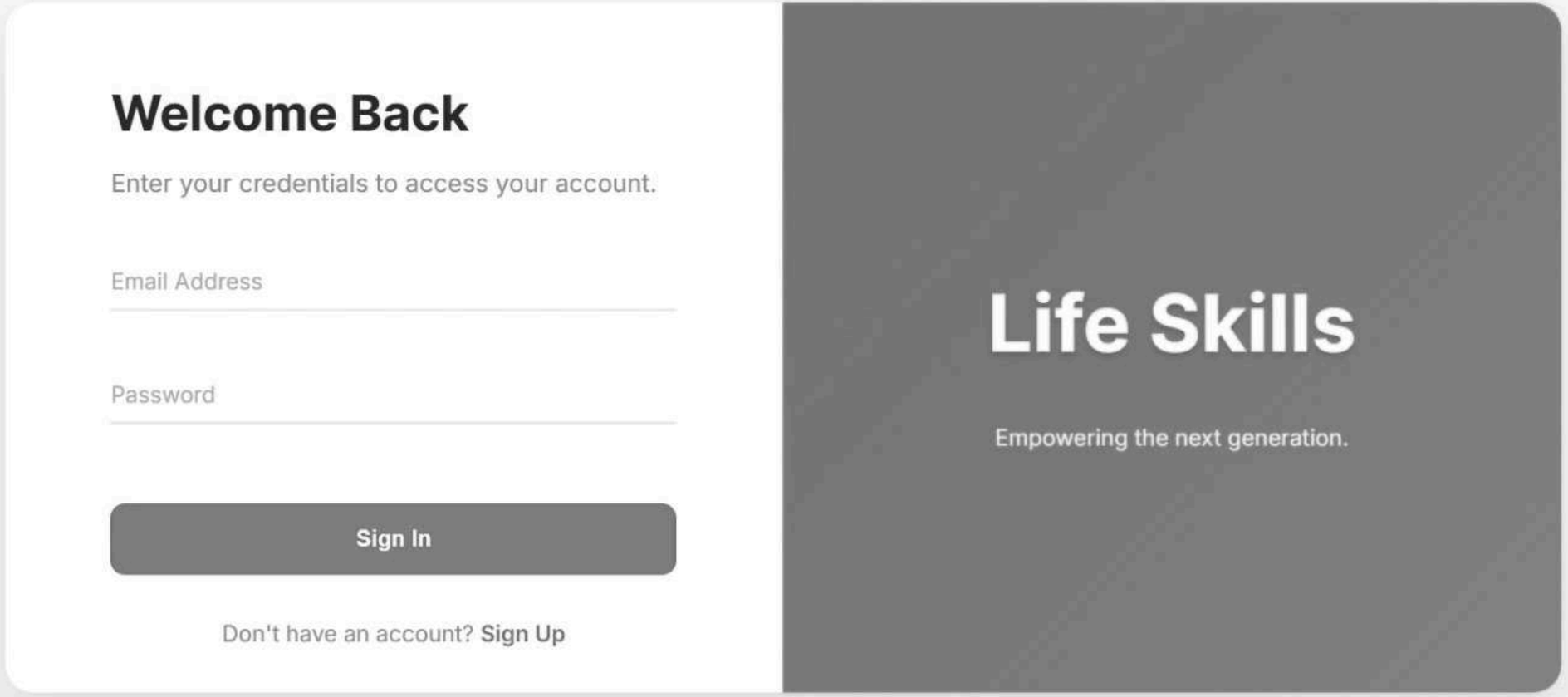
Send

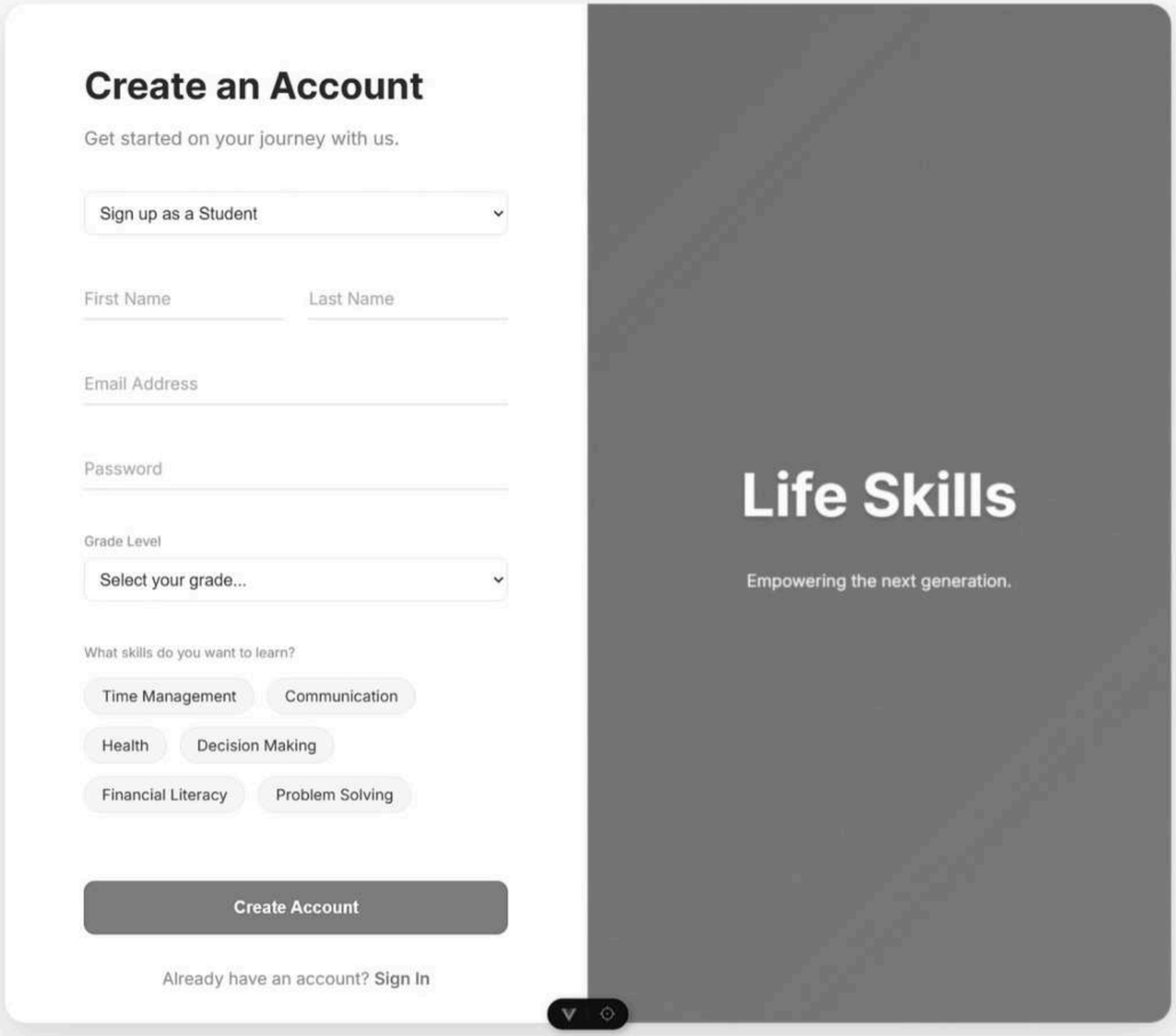
WIREFRAMES

These are the final wireframes providing a visual representation of the app's core features and user flows, demonstrating how students, parents, and teachers interact with the system. Each screen layout highlights essential functionalities such as setting reminders, tracking progress, practicing communication skills, and accessing feedback. By outlining the structure and navigation of the interface, these wireframes serve as a blueprint for the app's design and development, ensuring a user-friendly experience that meets the needs identified in our user stories.

User Interview:- Interview with Teacher

LOGIN


Welcome Back
Enter your credentials to access your account.
Email Address
Password
Sign In
Don't have an account? Sign Up
Life Skills
Empowering the next generation.


Create an Account
Get started on your journey with us.
Sign up as a Student
First Name
Last Name
Email Address
Password
Grade Level
Select your grade...
What skills do you want to learn?
Time Management
Communication
Health
Decision Making
Financial Literacy
Problem Solving
Create Account
Already have an account? Sign In
Life Skills
Empowering the next generation.

Updated Student Dashboard Wireframe Flow

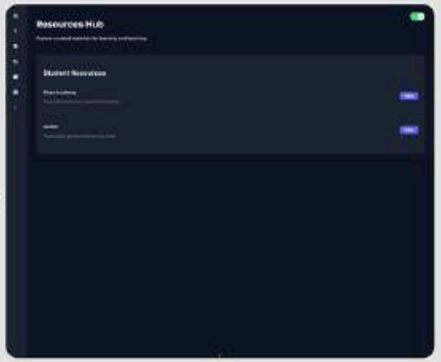
Login Page

Registration Page

Student Dashboard

Skills Page

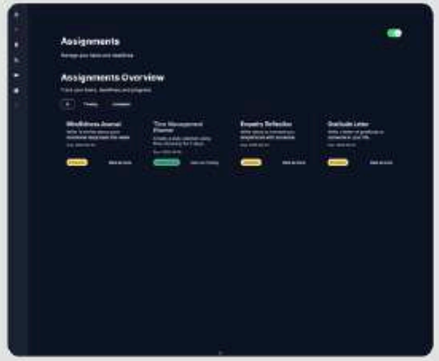
Resources Hub



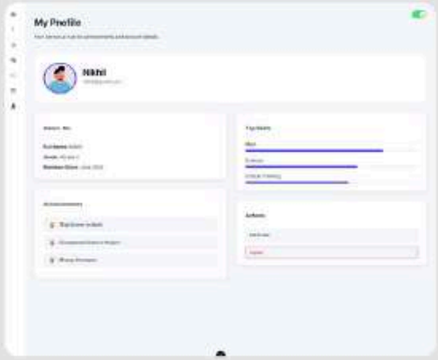
Student's Calendar

Message Center

Assignments Page



Student's Profile

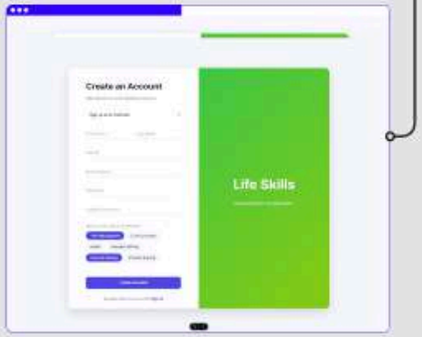


* Updated wireframe after the user feedback interview

Updated Teacher Dashboard Wireframe Flow

Login Page

Registration Page

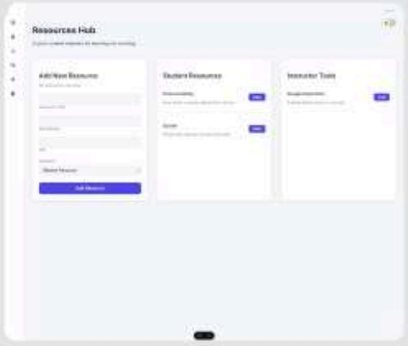


Teacher Dashboard

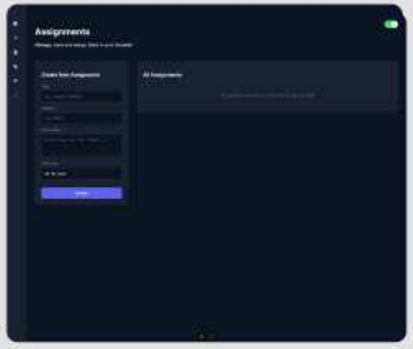
My Students

Teacher's Comms Center

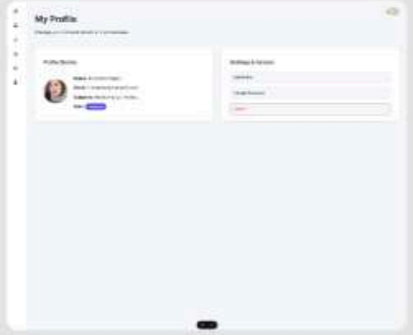
Resources Hub



Assignments Page



Teacher's Profile



* Updated wireframe after the user feedback interview

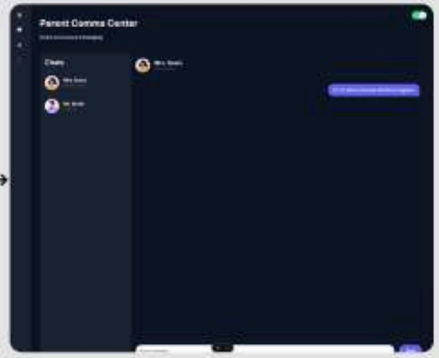
Parent Dashboard Wireframe Flow

Login Page

Registration Page

Parent Dashboard

Communication Center

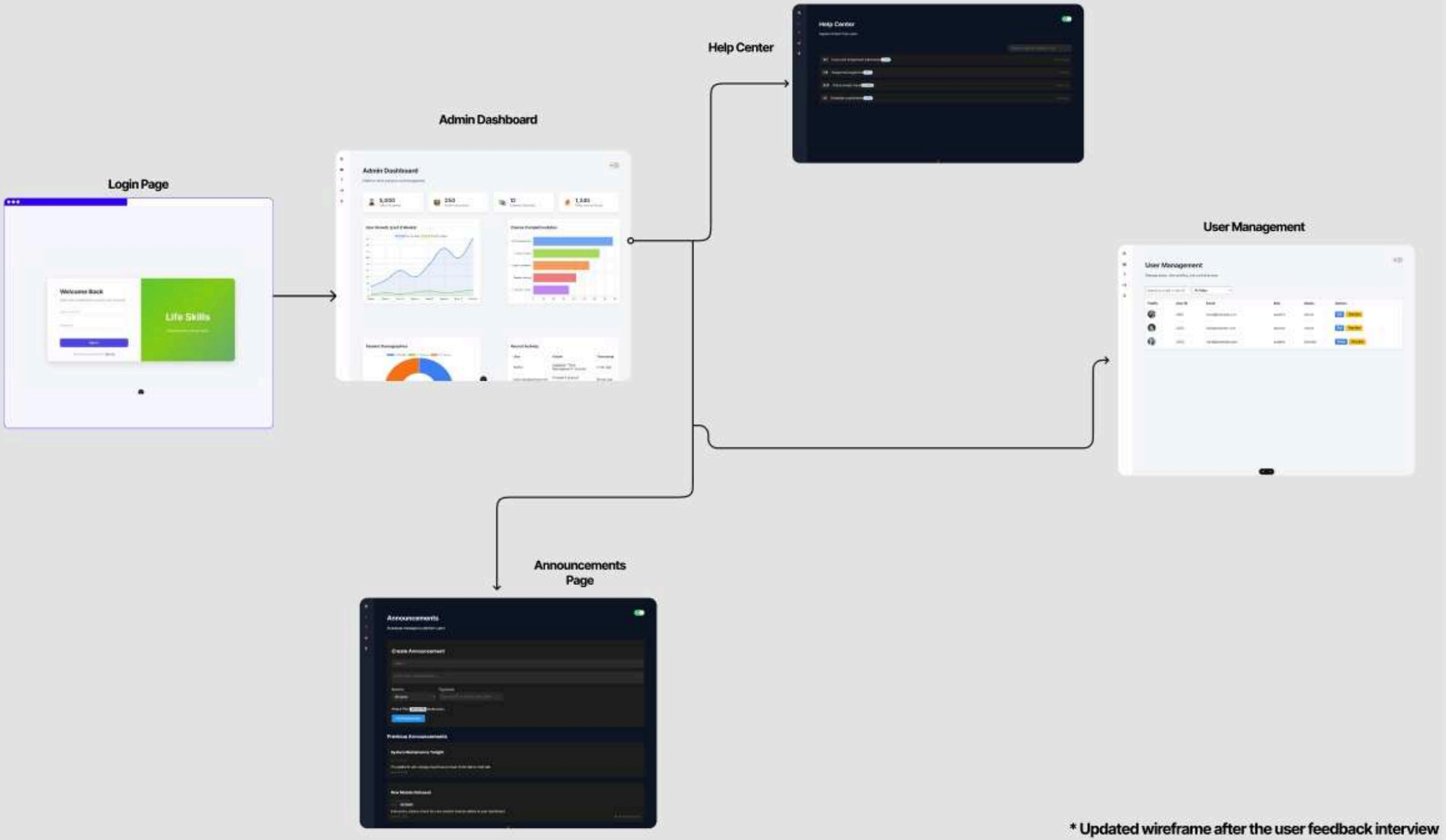


Message Center

Parent's Profile

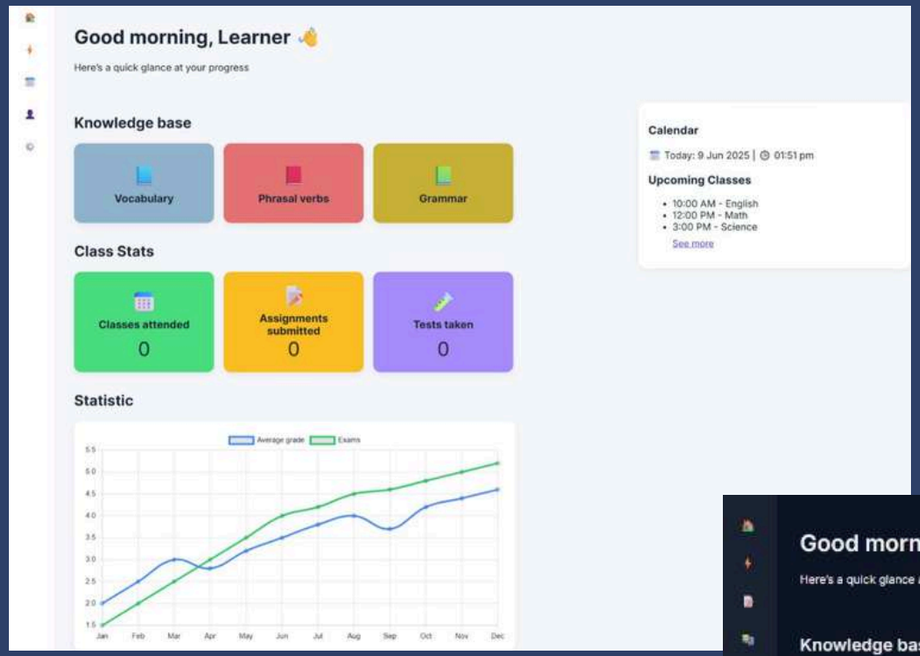
* Updated wireframe after the user feedback intervie

Admin Dashboard Wireframe Flow



* Updated wireframe after the user feedback interview

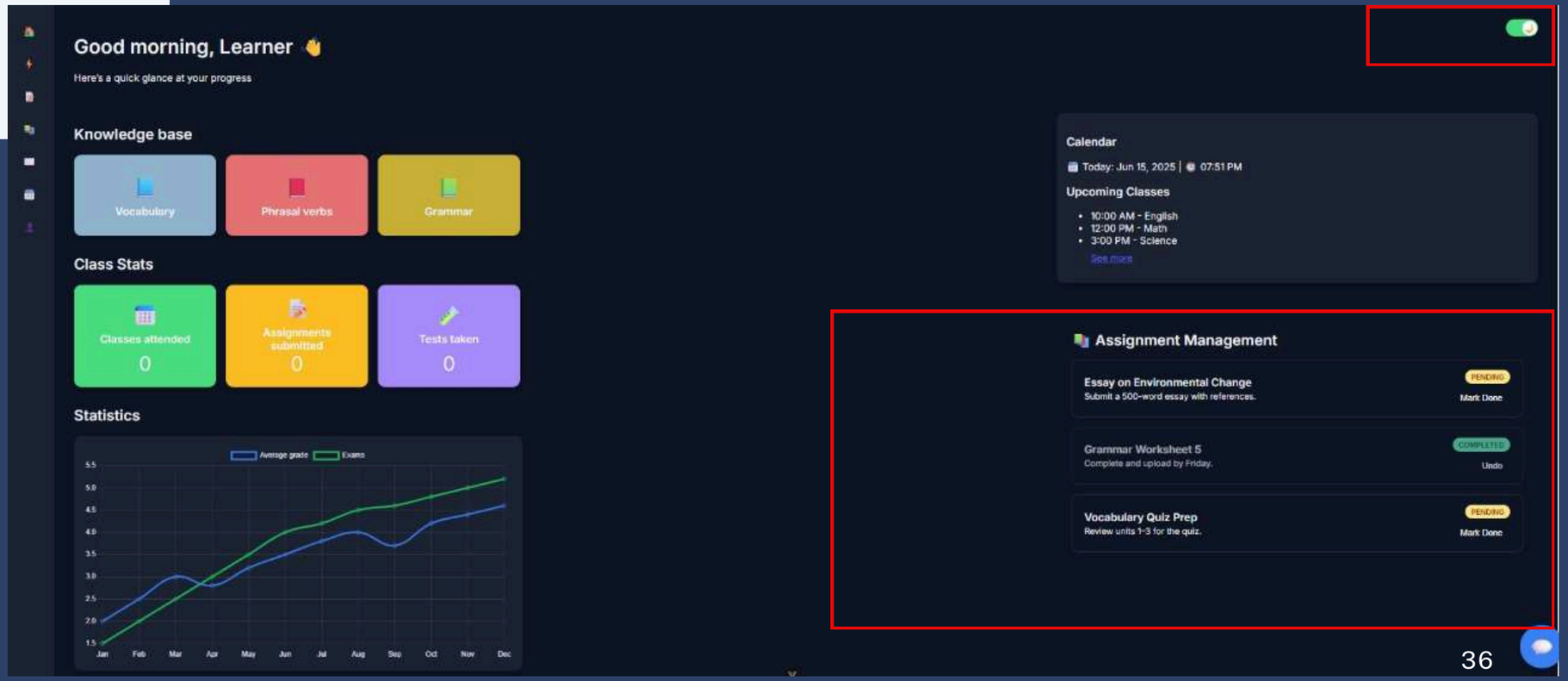
Changes made after getting User’s feedback.



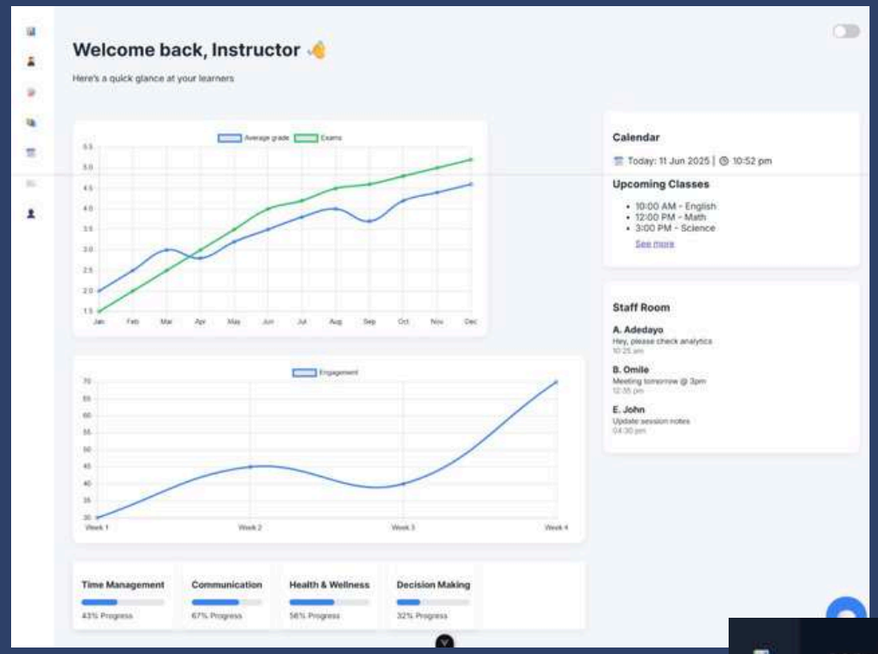
BEFORE

User Feedback:-

1. User requested to have “Task Management section in dashboard too.
2. User requested to have “Dark Mode” as a feature to switch between as per comfort



AFTER

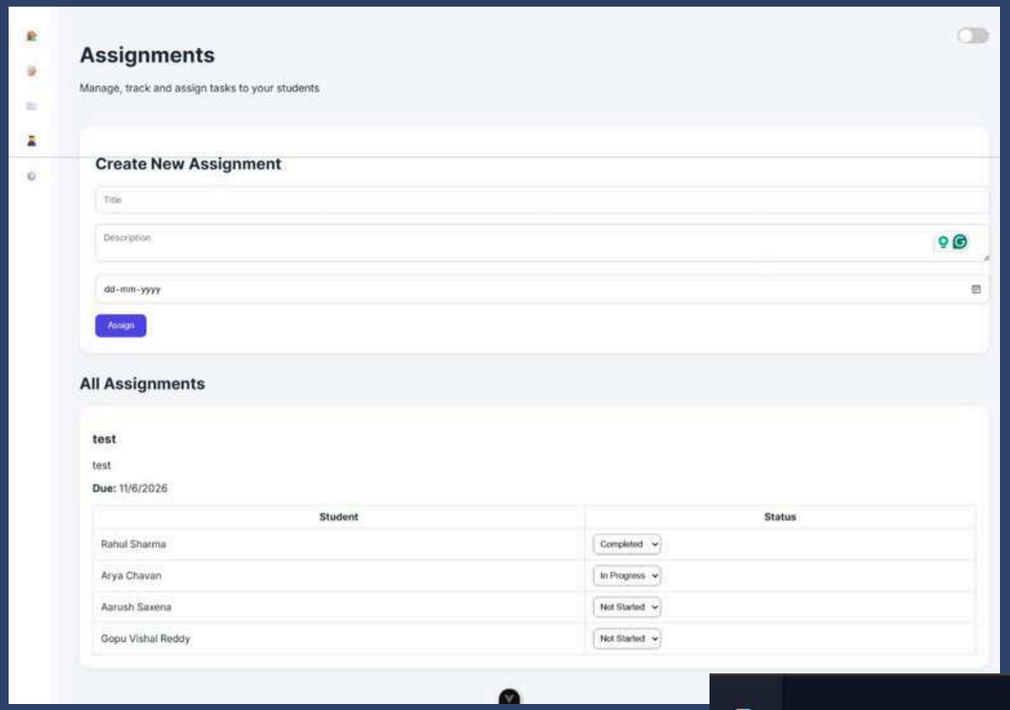


BEFORE

User Feedback:-

1. User requested to have more statistics in dashboard with better UI.
2. User requested “Dark mode” for their dashboard too so we made it common functionality for our app.

AFTER



BEFORE

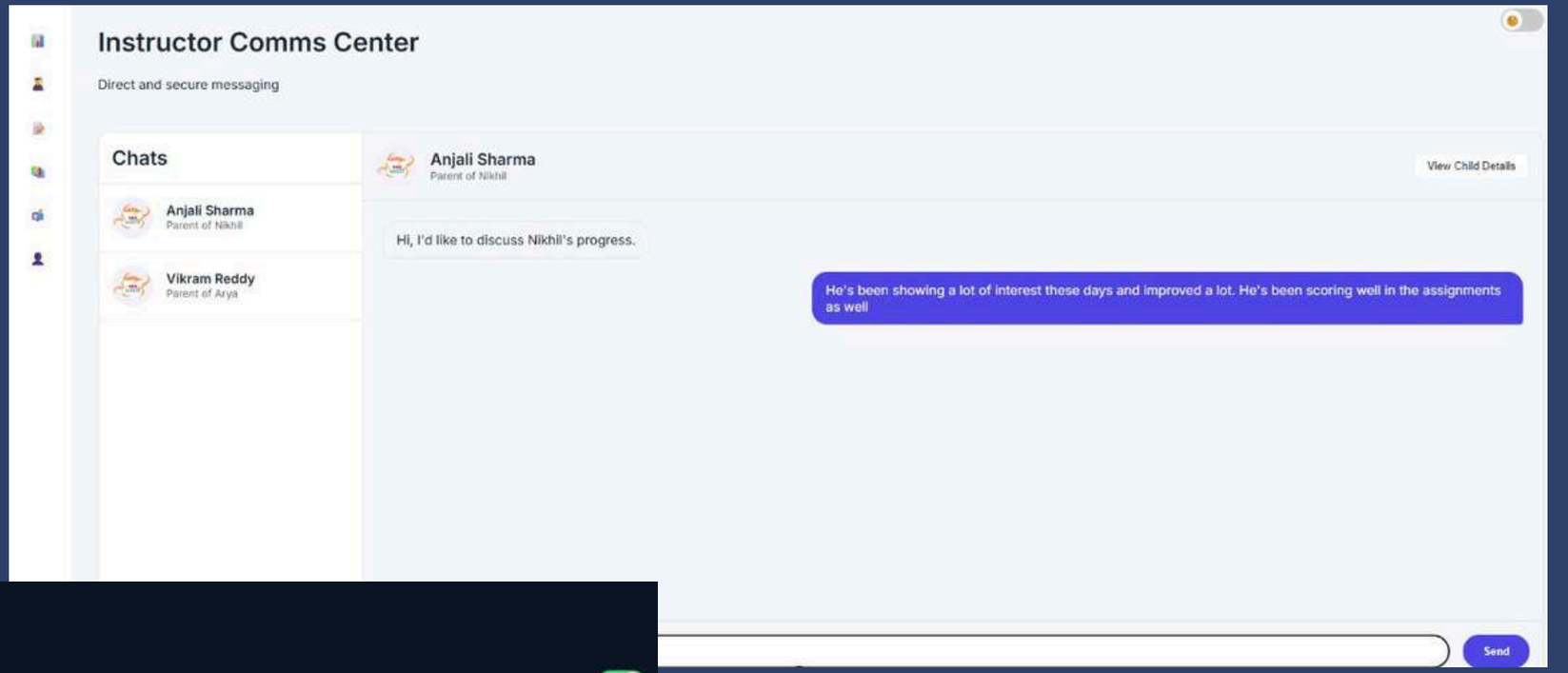
User Feedback:-

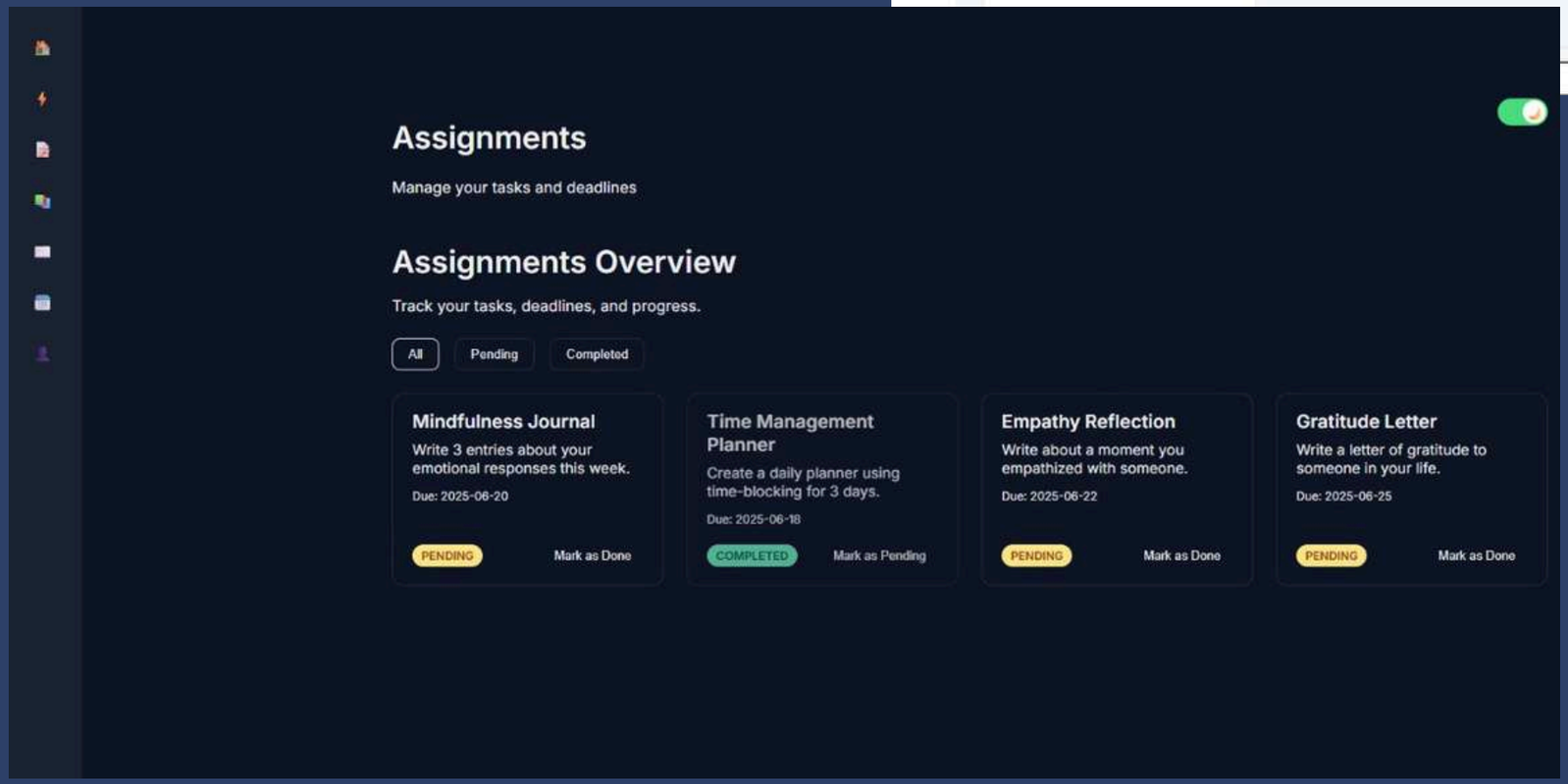
1. User requested to have more functionality in the “Assignments” page.

AFTER

User Feedback:-

1. User requested to have a specific “Communication Centre” between parents and themselves (instructors).





2. User suggested to add assignment manage page for Students.

MILESTONE-3

SCHEDULING AND DESIGN

PROJECT SCHEDULE

1.1 Tasks Distributions

- Project milestones are broken down into smaller, manageable tasks. These tasks are assigned to team members according to their skills, availability, and current workload to ensure an efficient and well-balanced workflow.

MILESTONE	TASKS	START DATE	END DATE	Assignee
MILESTONE 1	USER IDENTIFICATION	16-05-2025	20-05-2025	Kumar Rishabh Rahul Sharma Aarush Saxena Deepak Kumar Singh
	USER INTERVIEWS AND STORIES	21-05-2025	21-05-2025	Kumar Rishabh Rahul Sharma
	DARFT & SUBMIT MILESTONE REPORT	22-05-2025	25-05-2025	Kumar Rishabh Rahul Saxena
MILESTONE 2	STORYBOARD DESIGN	27-05-2025	31-05-2025	Kumar Rishabh
	UI Screens Overview	28-05-2025	04-06-2025	Rahul Sharma Nikhil Sai Kasireddy
	WIREFRAMES DESIGNING	31-05-2025	05-06-2025	Rahul Sharma Nikhil Sai Kasireddy
	DRAFT & SUBMIT MILESTONE REPORT	28-05-2025	06-06-2025	Kumar Rishabh

PROJECT SCHEDULE

MILESTONE	TASKS	START DATE	END DATE	Assignee
MILESTONE 3	PROJECT SCHEDULING USING JIRA	28-05-2025	07-06-2025	Deepak Kumar Singh
	FRONTEND COMPONENTS DESIGN	04-06-2025	18-06-2025	Rahul Sharma Nikhil Sai Kasireddy
	SOFTWARE CLASS DESIGN	20-06-2025	27-06-2025	Arya Chavan Aarush Saxena Gopu Vishal Reddy
	FRONTEND DEVELOPMENT	31-05-2025	20-06-2025	Rahul Sharma Nikhil Sai Kasireddy
	PREPARE AND SUBMIT MILESTONE REPORT	20-06-2025	06-07-2025	Kumar Rishabh
MILESTONE 4	DESIGNING APIs	16-06-2025	15-07-2025	Arya Chavan Aarush Saxena Gopu Vishal Reddy
	CREATING YAML FILE	17-07-2025	18-07-2025	Aarush Saxena
	SUBMIT YAML	20-07-2025	20-07-2025	Aarush Saxena

PROJECT SCHEDULE

MILESTONE	TASKS	START DATE	END DATE	Assignee
MILESTONE 5	WRITE TEST CASES FOR ALL ENDPOINTS	21-07-2025	25-07-2025	Aarush Saxena Arya Chavan Gopu Vishal Reddy
	TEST ALL API ENDPOINTS	26-07-2025	31-07-2025	Aarush Saxena Arya Chavan Gopu Vishal Reddy Rahul Sharma Deepak Kumar Singh
	PREPARE AND SUBMIT MILESTONE REPORT	01-08-2025	05-08-2025	Kumar Rishabh
MILESTONE 6	FINALIZE IMPLEMENTATION AND PROTOTYPE	08-08-2025	12-08-2025	Deepak Kumar Singh Aarush Saxena Arya Chavan Gopu Vishal Reddy
	DRAFT AND SUBMIT MILESTONE REPORT	13-08-2025	20-08-2025	Kumar Rishabh
	PREPARE AND SUBMIT FINAL PRESENTATION	21-08-2025	25-08-2025	Whole Team

PROJECT SCHEDULE

1.2 Sprint Details

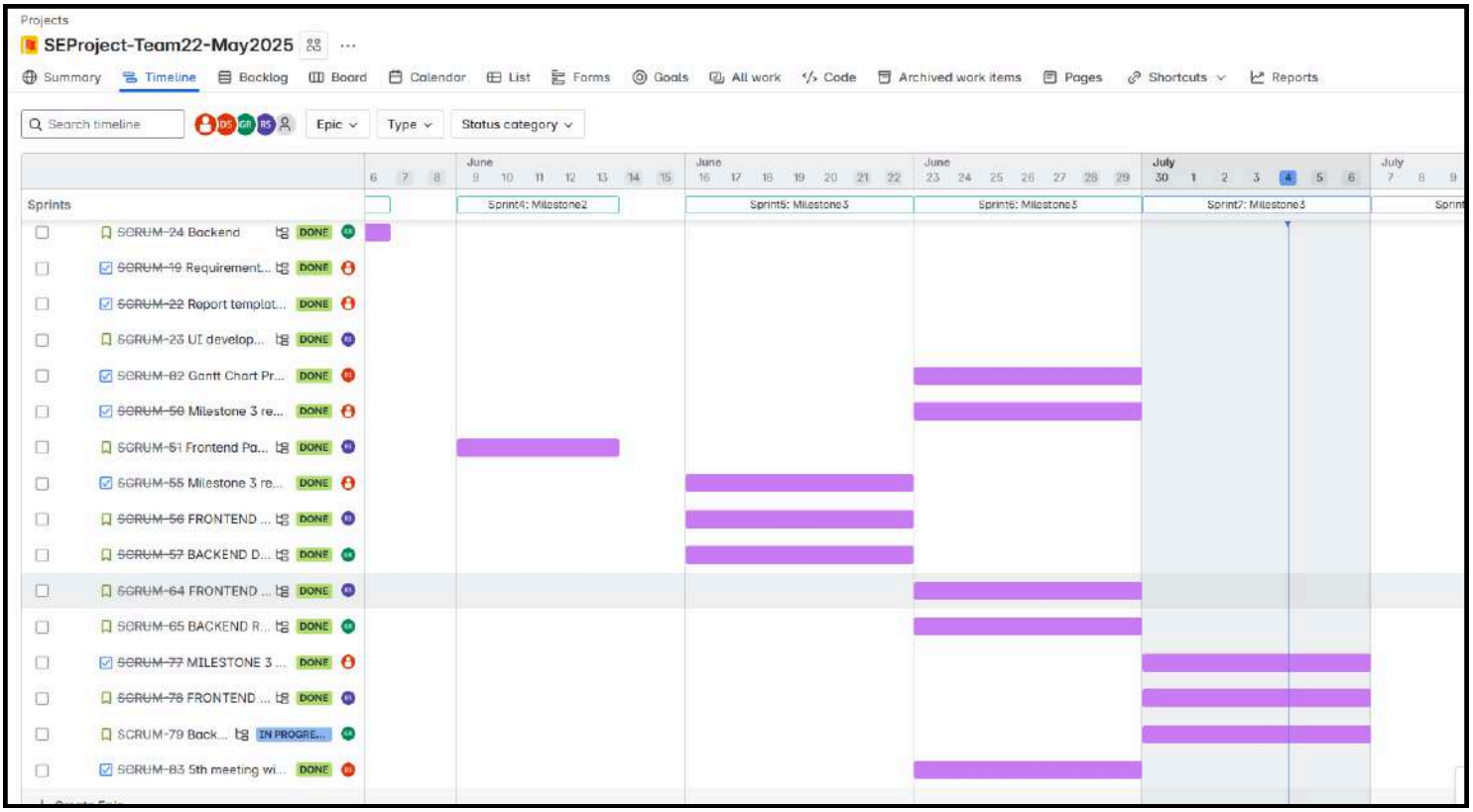
- A Sprint Backlog was constructed using JIRA to outline the tasks planned for each sprint in the project. It provides details about the status of each task, the assigned team members, and the overall progress within each sprint. This structured approach ensures that tasks are distributed effectively, progress is tracked, and project milestones are achieved on time.

Type	Key	Summary	Assignee	Reporter	Status	Resolution	Created	Due date
Sprint5: Milestone3	3 work items	+						
	~~SCRUM-57~~	BACKEND DEVELOPMENT	GOPU VISHAL ...	23f1000391	DONE	Done	Jun 17, 2025, 7:36 AM	Jun 20, 202
	~~SCRUM-56~~	FRONTEND DEVELOPMENT	Rahul Sharma	23f1000391	DONE	Done	Jun 17, 2025, 7:36 AM	Jun 20, 202
	~~SCRUM-55~~	Milestone 3 report requirements gathering.	23f1000391	23f1000391	DONE	Done	Jun 15, 2025, 2:30 PM	Jun 20, 202
Sprint6: Milestone3	3 work items	+						
	SCRUM-65	BACKEND DEVELOPMENT	GOPU VISHAL ...	23f1000391	IN PROGRESS	Unresolved	Jun 21, 2025, 7:09 PM	Jun 27, 202
	SCRUM-64	FRONTEND DEVELOPMENT	Rahul Sharma	23f1000391	IN PROGRESS	Unresolved	Jun 21, 2025, 7:09 PM	Jun 27, 202
	SCRUM-50	Milestone 3 report preperation.	23f1000391	23f1000391	IN PROGRESS	Unresolved	Jun 10, 2025, 11:00 PM	Jun 27, 202
Sprint7: Milestone3	3 work items	+						
	SCRUM-79	BACKEND DEVELOPMENT	GOPU VISHAL ...	23f1000391	TO DO	Unresolved	Jun 21, 2025, 7:33 PM	Jul 04, 202
	SCRUM-78	FRONTEND DEVELOPMENT	Rahul Sharma	23f1000391	TO DO	Unresolved	Jun 21, 2025, 7:33 PM	Jul 04, 202
	SCRUM-77	MILESTONE 3 REPORT FINALISATION	23f1000391	23f1000391	TO DO	Unresolved	Jun 21, 2025, 7:32 PM	Jul 04, 202
Backlog	1 work item	+						
Completed sprint	7 work items							
	~~SCRUM-61~~	Frontend Pages and Wireframes	Rahul Sharma	23f1000391	DONE	Done	Jun 13, 2025, 10:55 AM	Jun 15, 202
	~~SCRUM-43~~	Prepare milestone 2 report	23f1000391	Deepak Kumar ...	DONE	Done	Jun 02, 2025, 8:19 PM	Jun 11, 202
	~~SCRUM-42~~	Front-end Wireframe and Dashboard	Rahul Sharma	Deepak Kumar ...	DONE	Done	Jun 02, 2025, 8:18 PM	Jun 08, 202
	~~SCRUM-24~~	Backend	GOPU VISHAL ...	23f1000391	DONE	Done	May 25, 2025, 11:59 PM	Jun 15, 202
	~~SCRUM-23~~	UI development and Stuffs for Milestone2	Rahul Sharma	23f1000391	DONE	Done	May 25, 2025, 11:58 PM	Jun 05, 202
	~~SCRUM-22~~	Report template design	23f1000391	23f1000391	DONE	Done	May 25, 2025, 11:55 PM	May 30, 20
	~~SCRUM-19~~	Requirement gathering	23f1000391	Deepak Kumar ...	DONE	Done	May 25, 2025, 4:47 PM	None

PROJECT SCHEDULE

1.3 GANTT CHART

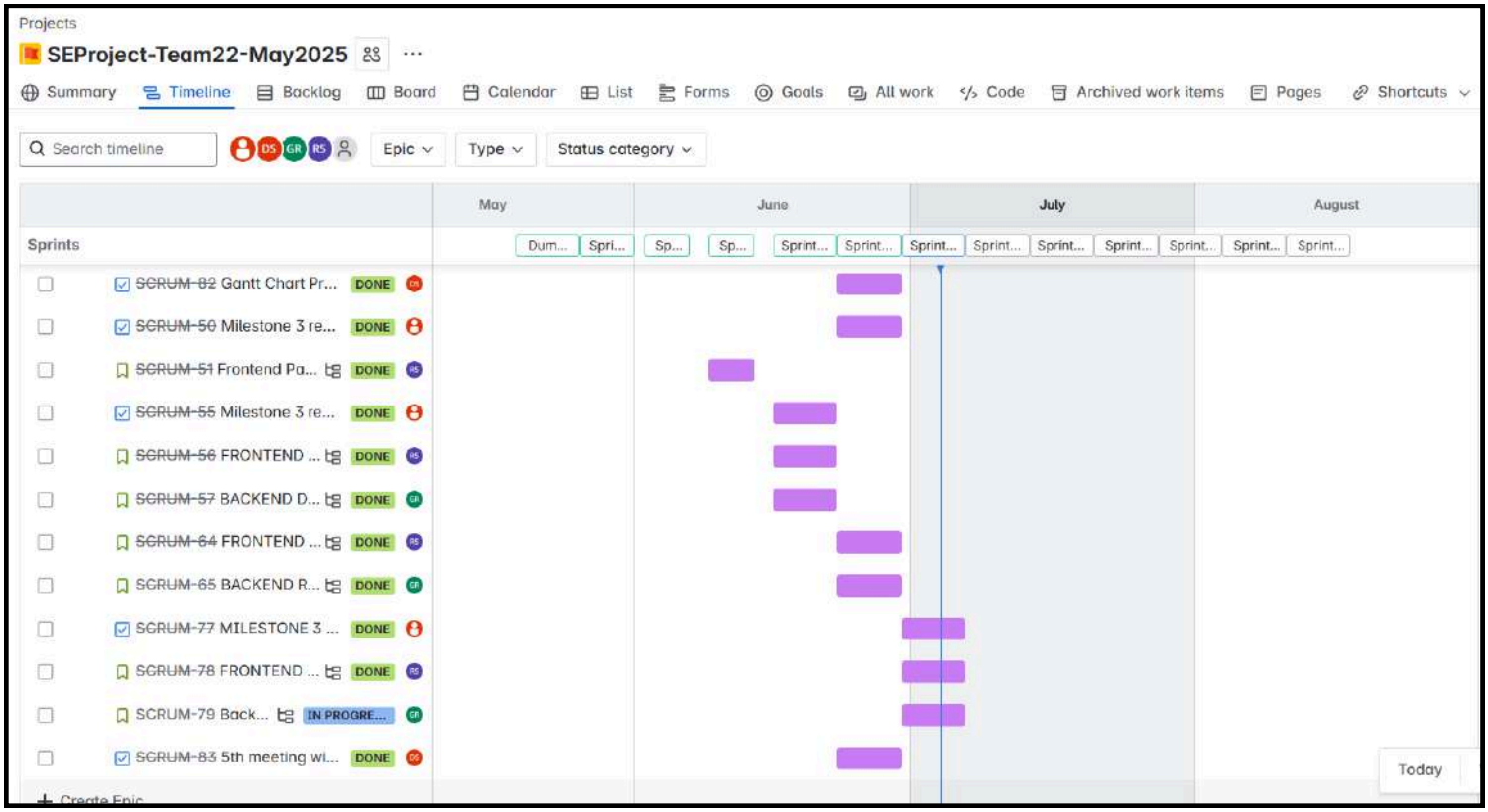
- A Gantt Chart (weekly & monthly) was plotted using JIRA to illustrate the project timeline and progress, structured across multiple sprints. This visual tool effectively organized tasks, defined their durations and dependencies, and tracked their completion status, ensuring alignment among team members regarding milestones and deadlines.



(Weekly Timeline)

PROJECT SCHEDULE

1.3 GANTT CHART



(Monthly Timeline)

SCRUM BOARD MEETINGS

2.1 MEETING 1 - MEETING 3

- **Review of Project Scope and Objective**
  - The project aims to design and develop a web or mobile app that helps children aged 8 to 14 build essential life skills like time management, emotional intelligence, financial literacy, communication, and healthy habits. The solution will be engaging, age-appropriate, and based on real concerns gathered from the target users.
- **Communication and Collaboration Setup**
  - The team agreed on the following tools for communication and collaboration:
  - WhatsApp: For daily updates
  - GitHub: For code collaboration, version control, and task management
  - Google Meet: Two meetings scheduled on every Monday & Saturday at 09:30 PM
- **User Stories and SMART Guidelines**
  - The entire team conducted user identification activities, followed by interviews with selected users. Based on the insights gathered, user stories were drafted and reviewed to ensure they adhere to SMART criteria (Specific, Measurable, Achievable, Relevant, and Time-bound) and align with user expectations.

SCRUM BOARD MEETINGS

2.1 MEETING 1 - MEETING 3

- **Review of Project Scope and Objective**
  - The project aims to design and develop a web or mobile app that helps children aged 8 to 14 build essential life skills like time management, emotional intelligence, financial literacy, communication, and healthy habits. The solution will be engaging, age-appropriate, and based on real concerns gathered from the target users.
- **Role Allocation Matrix Finalization**
  - The finalized role allocation matrix is as follows:

Role	Members
Leader	Deepak Kumar Singh
Scrum Master	Kumar Rishabh
Frontend Development	Rahul Sharma Nikhil Sai Kasireddy
Backend Development	Arya Chavan Aarush Saxena Gopu Vishal Reddy

SCRUM BOARD MEETINGS

2.2 MEETING 4 - MEETING 7

- **UI Wireframes Presentation**
  - Rahul Sharma and Nikhil Sai presented various UI wireframes for the Life Skills App, showcasing the proposed layout, navigation structure, and visual elements. The team reviewed the wireframes and provided constructive feedback to enhance usability and engagement for the target age group (8 to 14 years).
- **Backend and Database Schema Presentation**
  - Aarush, Arya and Gopu Vishal Reddy presented the initial database schema, outlining how user profiles, skill modules, progress tracking, and other essential data will be structured. The schema was reviewed and refined based on team discussions to ensure alignment with project requirements. And also the backend team presented the workflow of the whole backend.
- **Storyboard and Wireframes**
  - The team reviewed and finalized the application storyboard made by Kumar Rishabh, illustrating the key screens and user journey through the app.
  - Low-fidelity wireframes were created for the identified user stories, adhering to usability design principles to ensure an intuitive experience for children.

SCRUM BOARD MEETINGS

2.2 MEETING 4 - MEETING 7

- **GitHub Workflow Implementation**
  - The team discussed and implemented GitHub best practices to streamline collaboration, including:
  - Creating separate branches for individual tasks
  - Making regular commits with meaningful messages
  - Submitting pull requests for task completion and peer reviews

- **Frontend Flow Discussion**
  - The frontend flow for the application was finalized with the following corrections:
  - After login, users will be directed to the dashboard, providing access to life skill modules instead of a generic landing page
  - The chat interface will include an option to reset sessions, along with a simplified, child-friendly design

- **Action Items:**
  - Finalize UI wireframes by 04 June , 2025 (Rahul Sharma, Nikhil Sai Kasireddy).
  - Implement chatbot functionalities by 08 June , 2025 (Aarush Saxena).

SCRUM BOARD MEETINGS

2.3 MEETING 8 - MEETING 10

- **Feature Development Priority List**
  - The team discussed and finalized the priority order for feature development based on the previously drafted user stories. Features were organized to ensure that core functionalities essential to user engagement and learning are developed first.
- **Scheduling Activities**
  - The team created a detailed schedule for sprints and iterations using tools such as the Kanban board and Gantt chart to visualize timelines and task distribution effectively.
- **Progress Review for Milestone 3 Completion**
  - The current project progress was reviewed, and next steps were outlined to ensure timely completion of Milestone 3, which includes finalized design components and key feature implementations.
- **Action Items:**
  - Refine ER diagram by June 30, 2025 (Arya, Aarush, Vishal).
  - Work on Milestone 3 report by June 29, 2025 (Kumar Rishabh, Rahul Sharma).

SOFTWARE ARCHITECTURE COMPONENTS

3.1 FRONTEND

3.1.1 USER INTERFACE

- **Sidebar (App Sidebar):** A role-based side menu providing quick access to essential pages:
  - Admin: Dashboard, Manage Users, Reports, Announcements, Logout
  - Student: Dashboard, Assignments, Resources, Notifications, Profile, Logout
  - Parent: Dashboard, Communication Center, Notifications, Profile, Logout
  - Teacher: Dashboard, Assignments, Notifications, Communication Center, Manage Students, Resources, Profile, Logout

3.1.2 PAGES

- **Admin Pages:**
  - Dashboard Page: Provides a system-wide overview of application status and activities.
  - Manage Users Page: Enables the admin to add, update, or remove users.
  - Messages & Notifications Page: This page Displays the system-wide notifications and messages.
  - Announcements Page: Allows the admin to create and manage announcements visible to all users.

SOFTWARE ARCHITECTURE COMPONENTS

3.1.2 PAGES

- **Student Pages:**
  - Dashboard Page: Provides an overview of the student's progress and activities.
    Includes interactive charts to track performance.
    Displays alerts and important announcements.
    Contains a Focus Mode section to stay focused.
    Integrates a chatbot for instant guidance
  - Assignments and Mock Quizzes Pages: Allows students to view and submit assignment responses shared by teachers. Automatically calculates scores.
  - Resources Page: Displays resources shared by teachers related to specific life skills. Students can view or download study materials, guides, or reference content.
  - Messages & Notifications Page: Displays system notifications and personal messages.
  - Profile Page: Allows students to view and update their personal information.

SOFTWARE ARCHITECTURE COMPONENTS

3.1.2 PAGES

- **Teacher Pages:**
  - Dashboard Page: Provides an overview of ongoing activities, assignments, and student progress.
  - Assignments Page: Allows teachers to create, assign, and manage assignments.
  - Resources Page: Enables teachers to upload and share resources with students. Teachers can add documents, links or files related to skills
  - Manage Students Page: Allows teachers to view and manage student profiles and progress.
  - Communication Center: A chat-like interface resembling little like WhatsApp which Facilitates communication between teachers and parents for progress updates and feedback.
  - Messages & Notifications Page: Displays system notifications and personal messages.
  - Profile Page: Allows teachers to view and update their personal information.

SOFTWARE ARCHITECTURE COMPONENTS

3.1.2 PAGES

- **Parents Pages:**
  - Dashboard Page: Provides an overview of the child's progress and important updates.
  - Communication Center: Enables seamless communication with teachers regarding the child's performance and concerns.
  - Messages & Notifications Page: Displays system notifications and personal messages.
  - Profile Page: Allows parents to view and update their personal information.

3.2 BACKEND

3.2.1 USER Authentication APIs:

- /api/v1/login
  - For Authentication and access token generation.
- /api/v1/signup
  - For registering the new user according to their required role.

3.2.2 Student Role APIs:

- /api/v1/student/{user_id}/dashboard
  - For viewing basic stats and latest updates
- /api/v1/student/{user_id}/assignments
  - For managing assignments
- /api/v1/student/{user_id}/badges
  - For viewing achievements
- /api/v1/student/{user_id}/resoruces
  - For accessing resources by instructors
- /api/v1/student/{user_id}/calendar
  - CRUD operation for task management and reminders setup
- /api/v1/student/{user_id}/profile
  - For viewing profile page and CRUD operations

3.2.3 Instructor Role APIs:

- /api/v1/teacher/{user_id}/dashboard
  - For viewing teaching stats, class performance, and latest updates
- /api/v1/teacher/{user_id}/students
  - For managing student lists and viewing individual progress
- /api/v1/teacher/{user_id}/assignments
  - CRUD operations for the assignments page
- /api/v1/teacher/{user_id}/resources
  - CRUD operations for managing resource materials
- /api/v1/teacher/{user_id}/comms
  - For communicating with students, parents and admin
- /api/v1/teacher/{user_id}/profile
  - For viewing and editing teacher profile info

3.2.4 Parent Role APIs:

- /api/v1/parent/{user_id}/dashboard
  - For viewing child's performance summary and activity updates
- /api/v1/parent/{user_id}/messages
  - For viewing and sending messages to teachers

- /api/v1/parent/{user_id}/comms
  - For communicating with the instructor.
- /api/v1/parent/{user_id}/profile
  - For viewing and editing parent and their kid account info

3.2.5 Admin Role APIs:

- /api/v1/admin/{user_id}/dashboard
  - For monitoring platform metrics and user activity
- /api/v1/admin/{user_id}/usermanagement
  - For managing user roles, accounts, and CRUD operations
- /api/v1/admin/{user_id}/helpcenter
  - For configuring and managing support content
- /api/v1/admin/{user_id}/announcements
  - For creating and managing global announcements

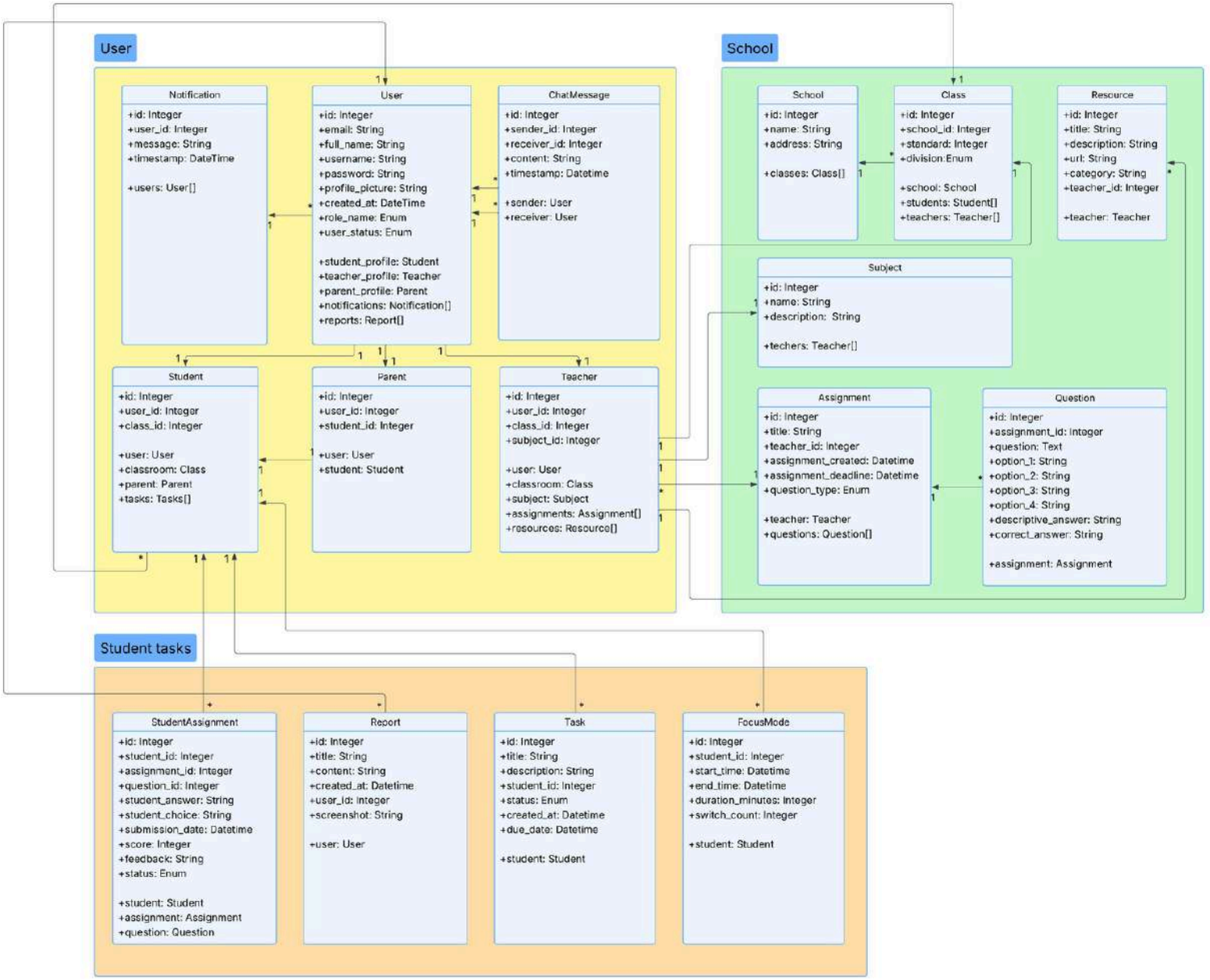
3.3 ChatBot Integration

COMMUNICATION BOT (AI-POWERED COACH)

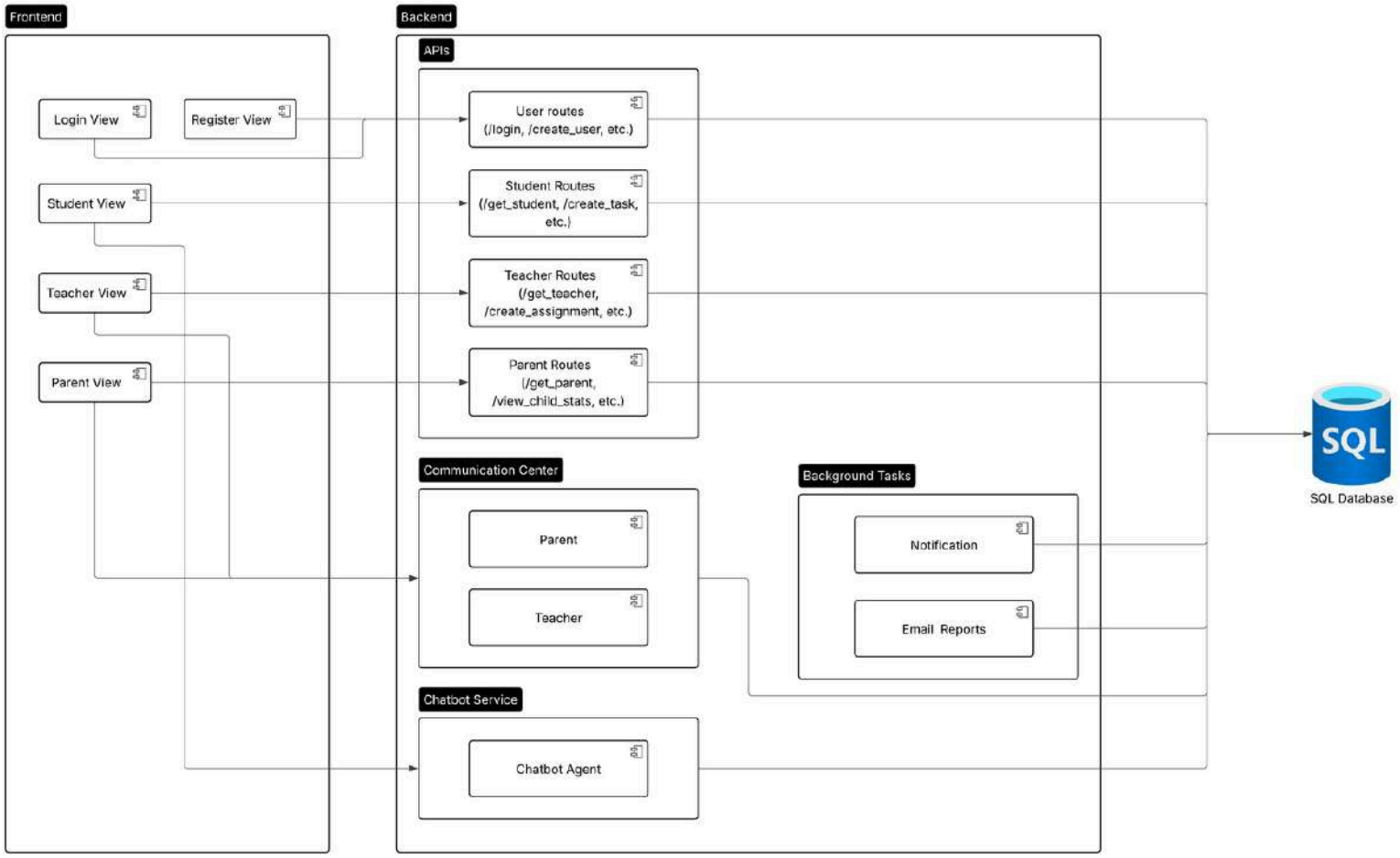
Enhances student speaking skills through interactive conversations.

- **Core Capabilities:**
  - Understands both voice and text input.
  - Gives smart, engaging replies to improve real-time communication.
  - Builds confidence and fluency in shy or hesitant speakers.

CLASS STRUCTURE DIAGRAM

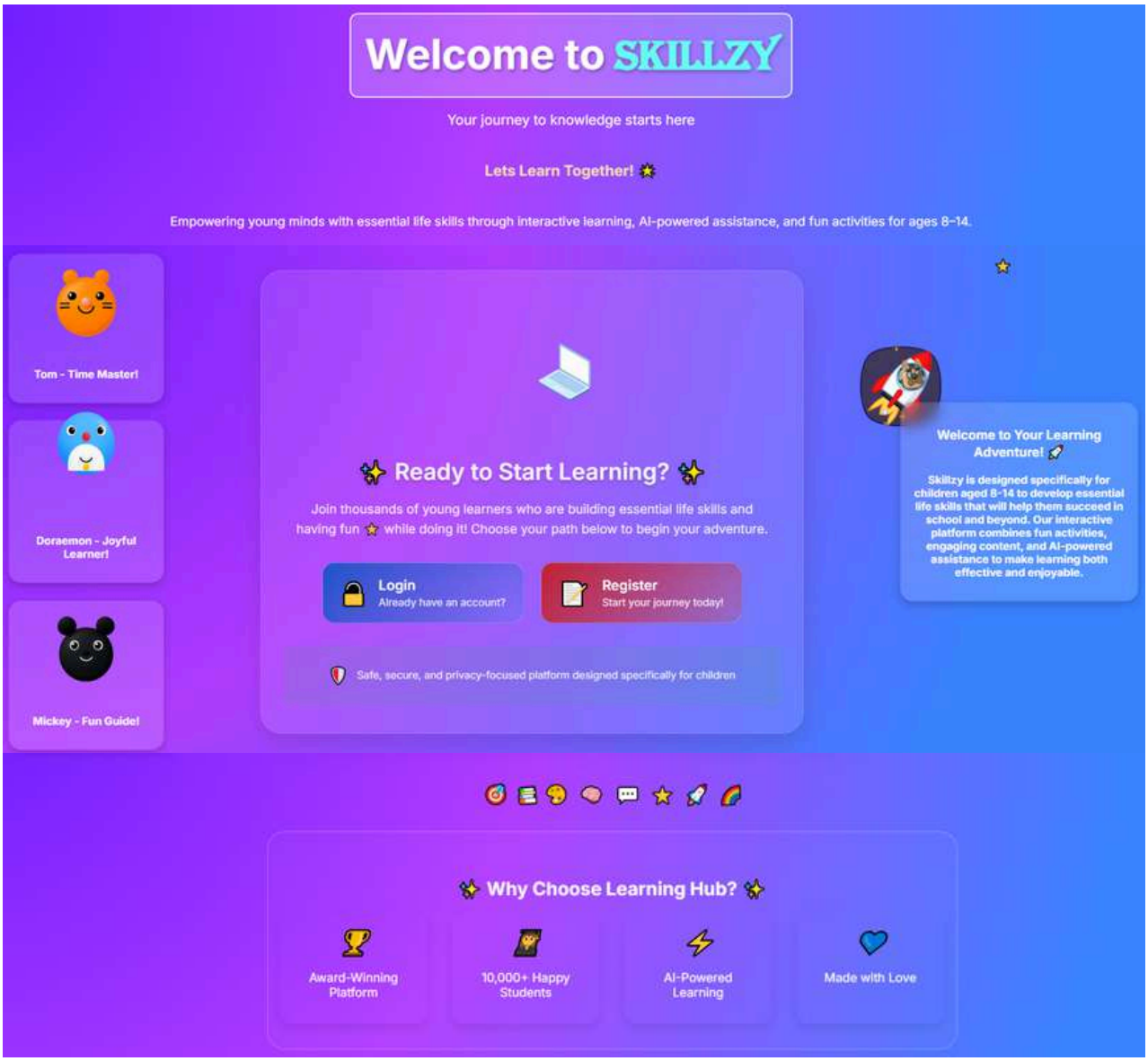


COMPONENT DIAGRAM



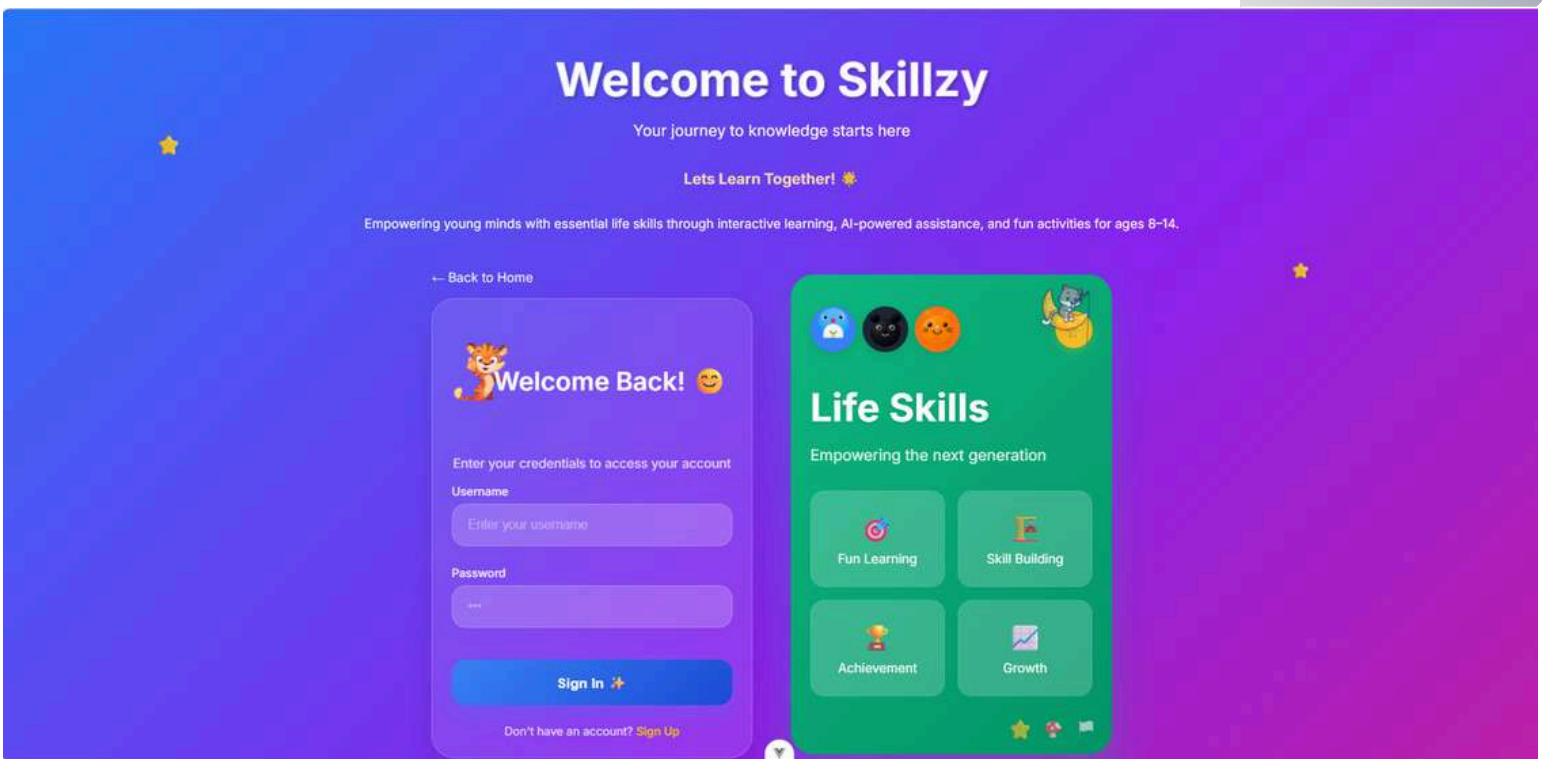
USER INTERFACE SCREENS

USER INTERFACE: WELCOME SCREEN

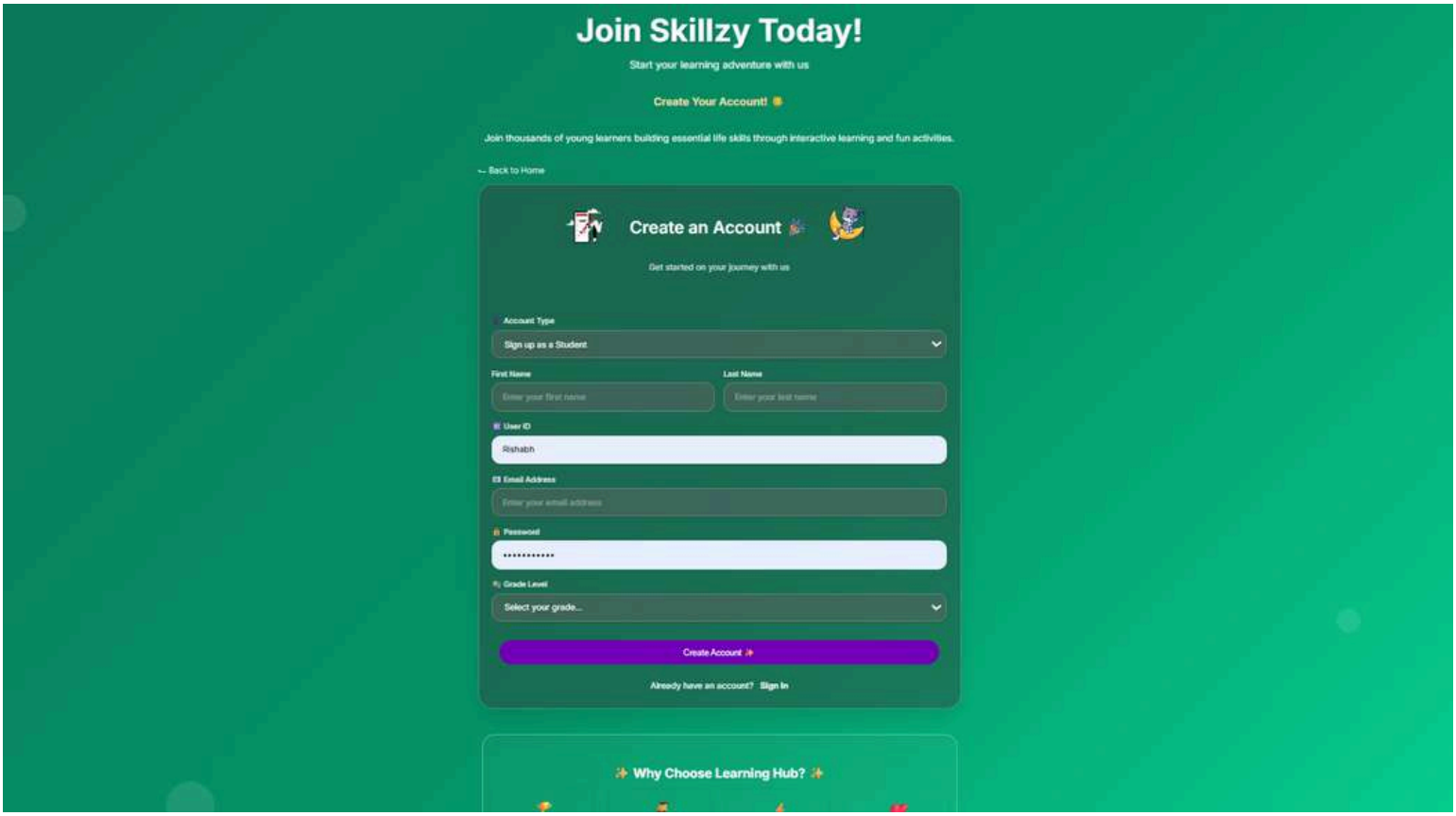


USER INTERFACE SCREENS

Login Page

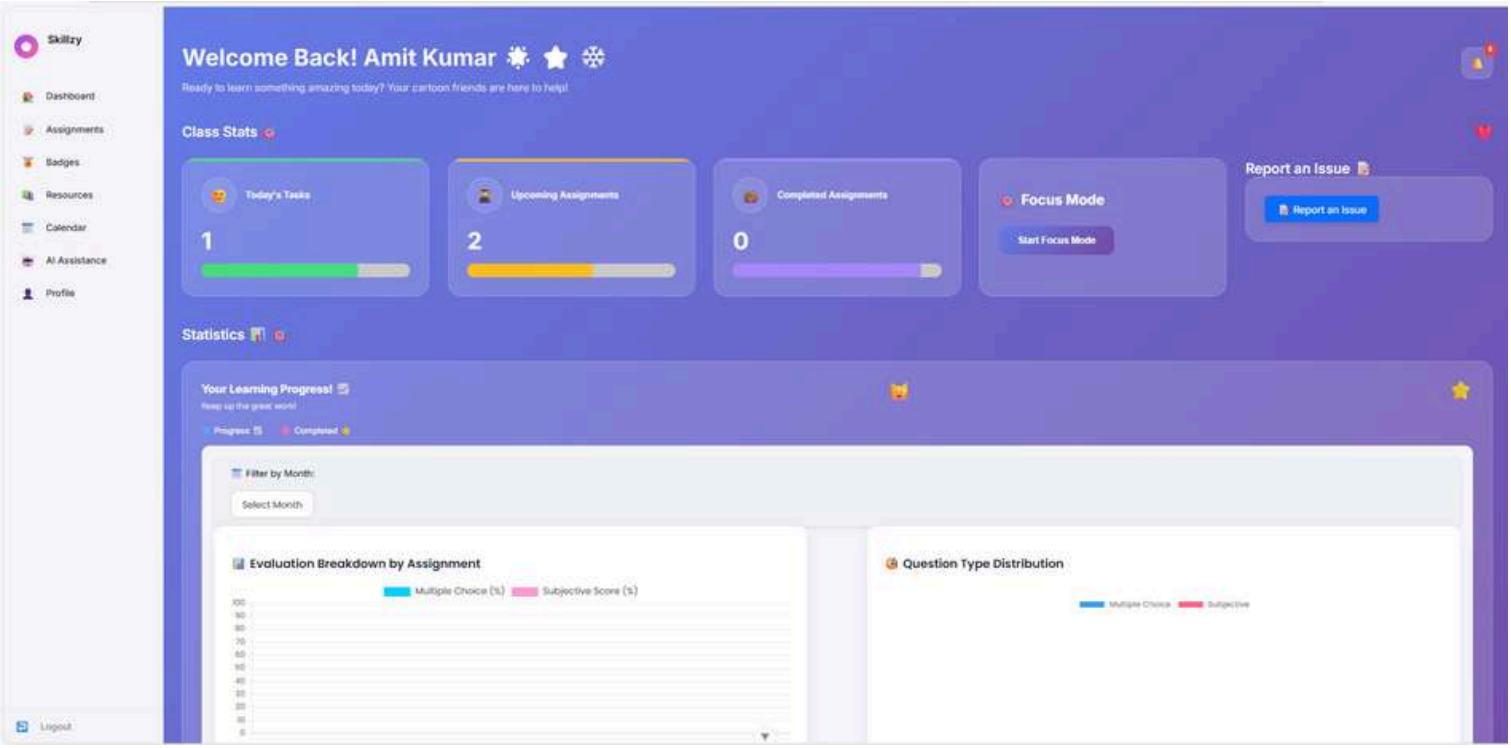


Registration Page

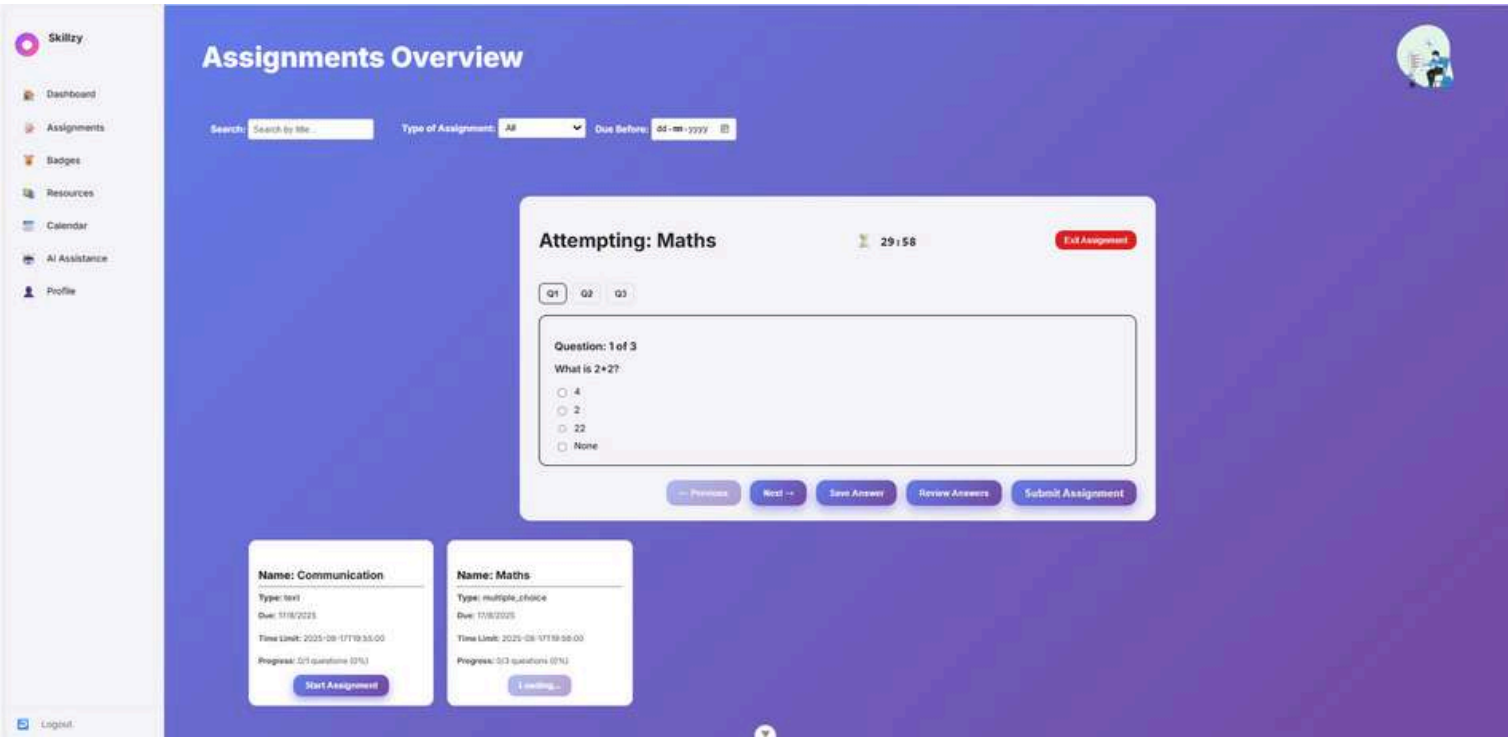


USER INTERFACE SCREENS

Student Dashboard

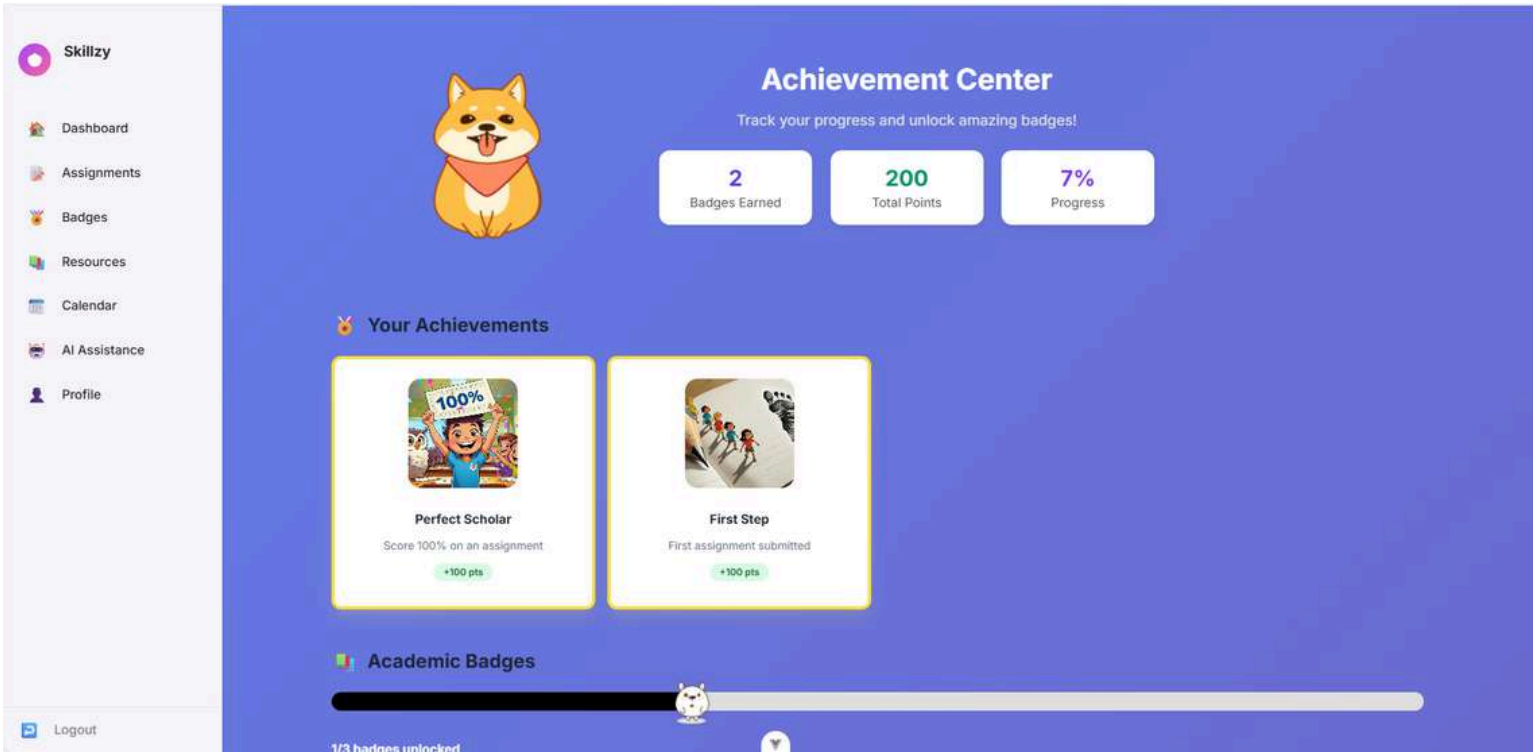


Student Assignments

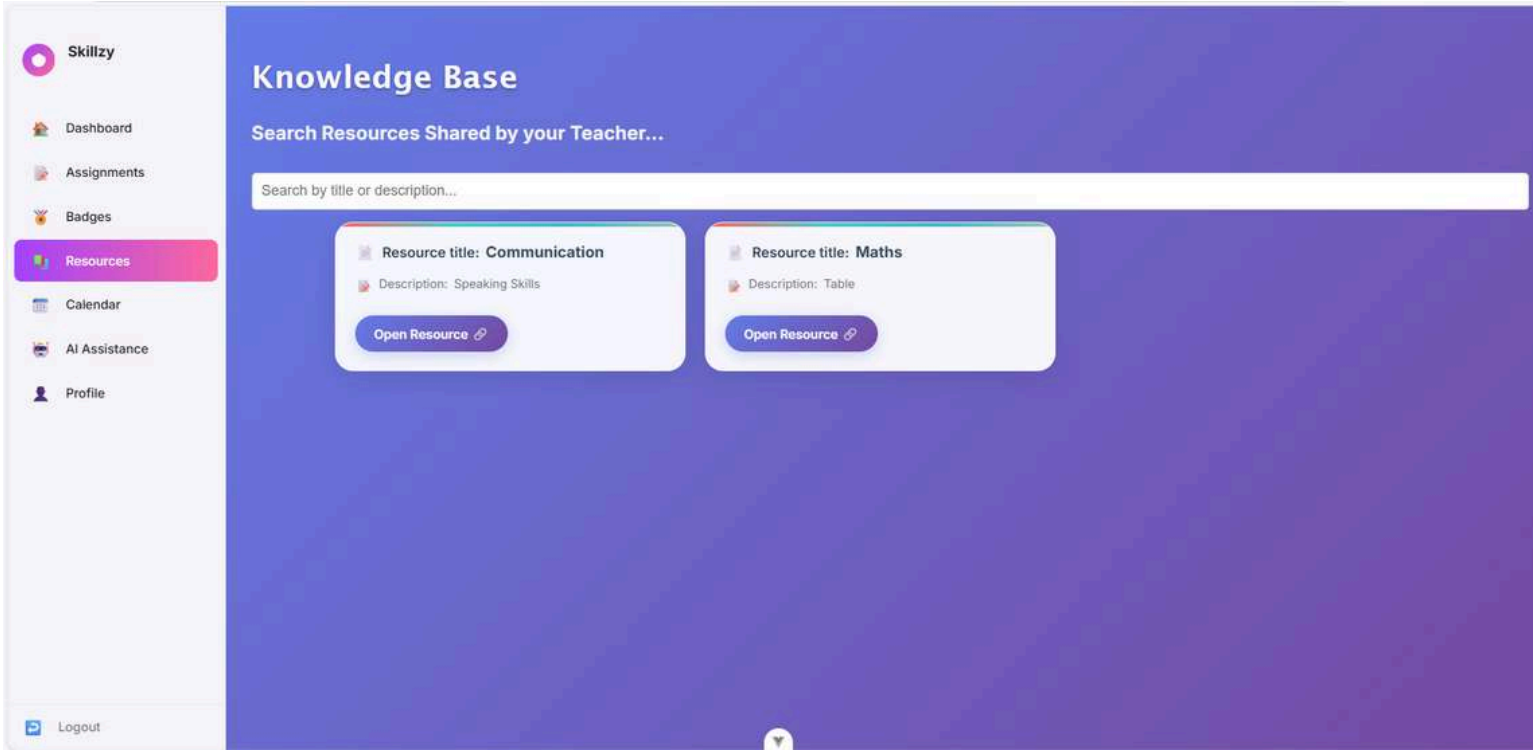


USER INTERFACE SCREENS

Student Badges

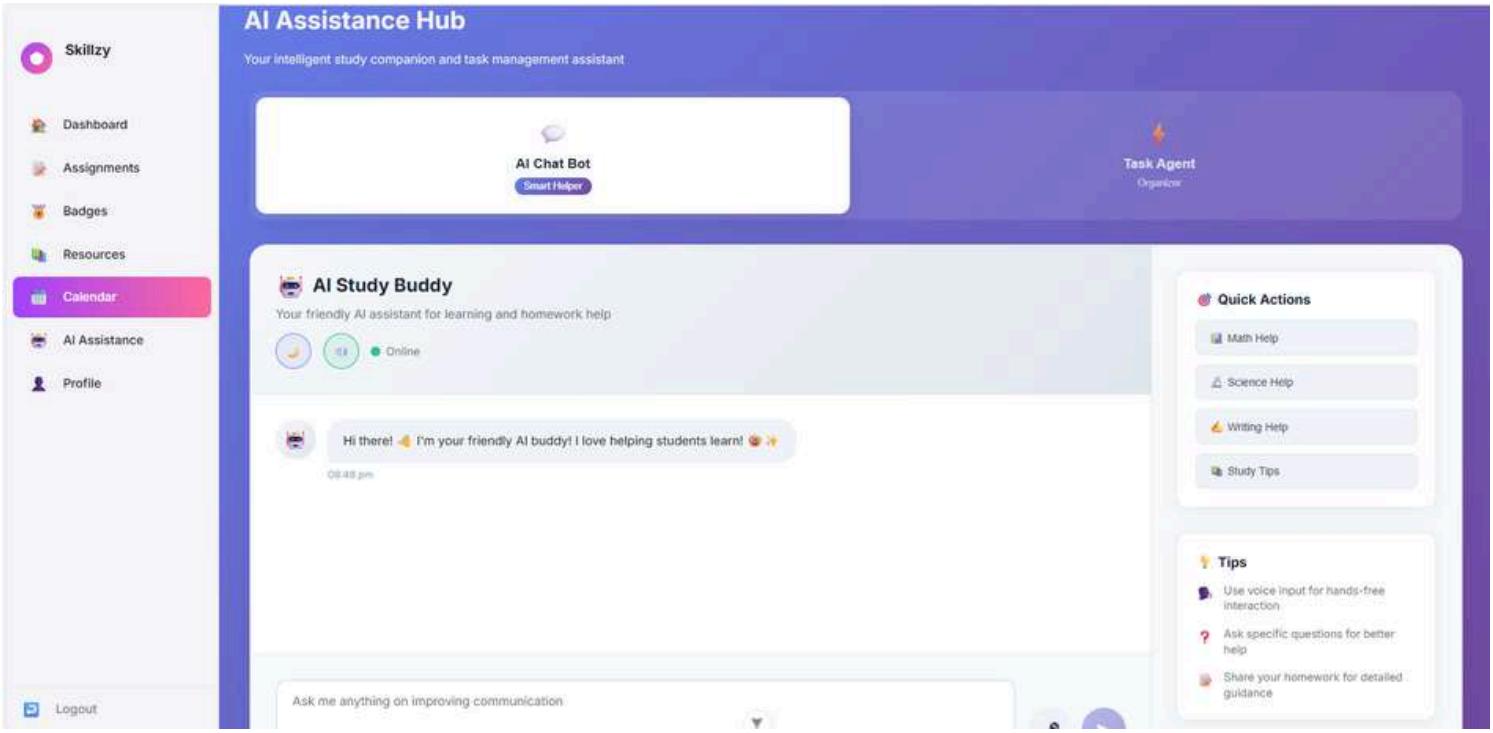


Student Resources

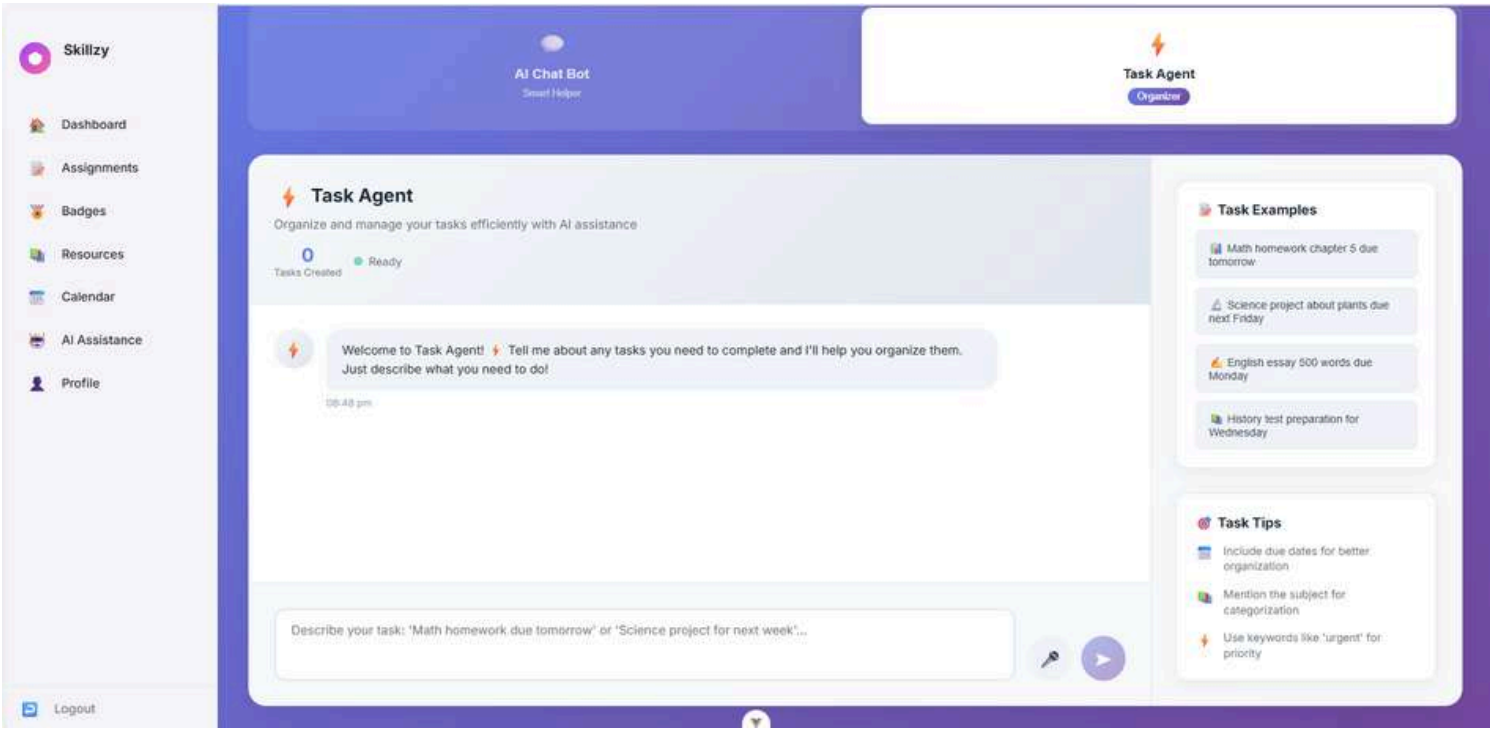


USER INTERFACE SCREENS

Student AI CHATBOT

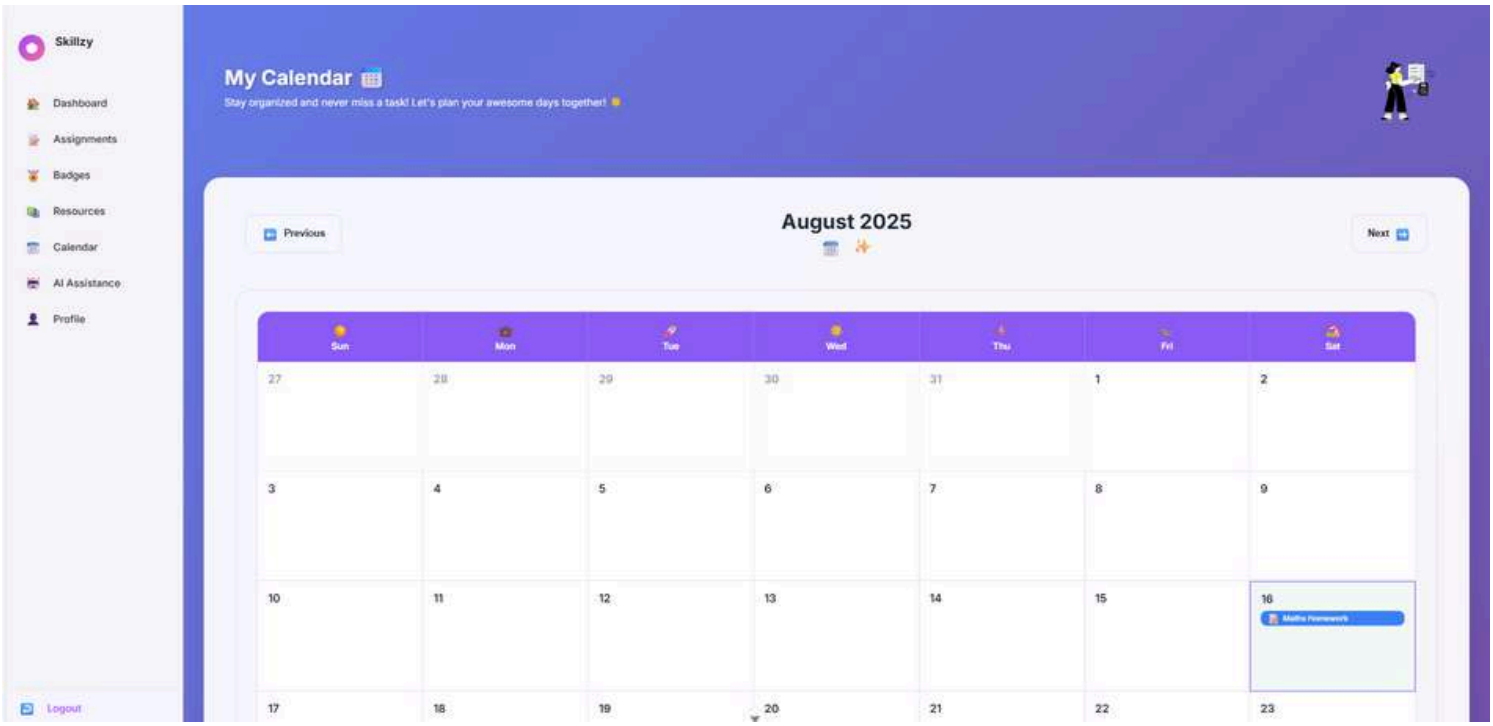


Student AI Task Agent

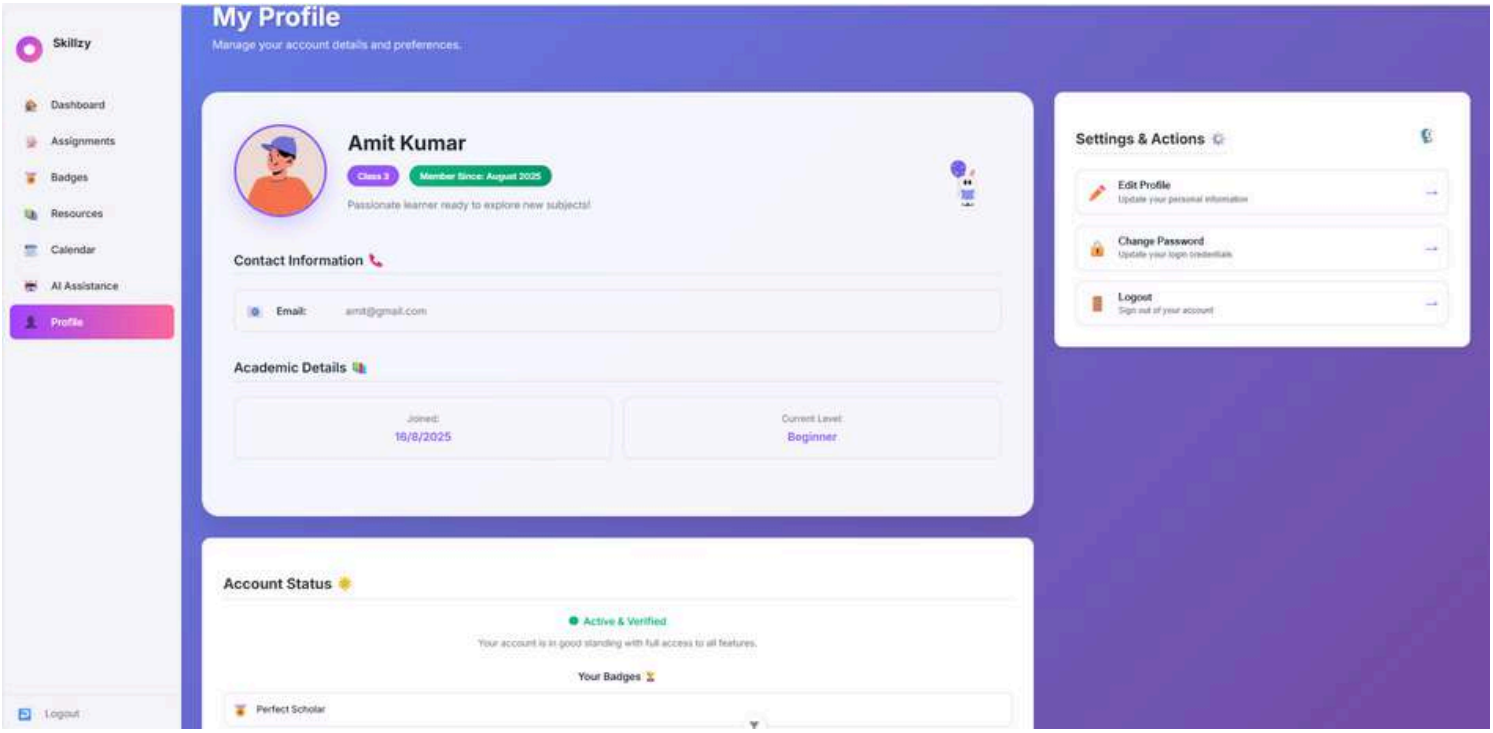


USER INTERFACE SCREENS

Student Calendar

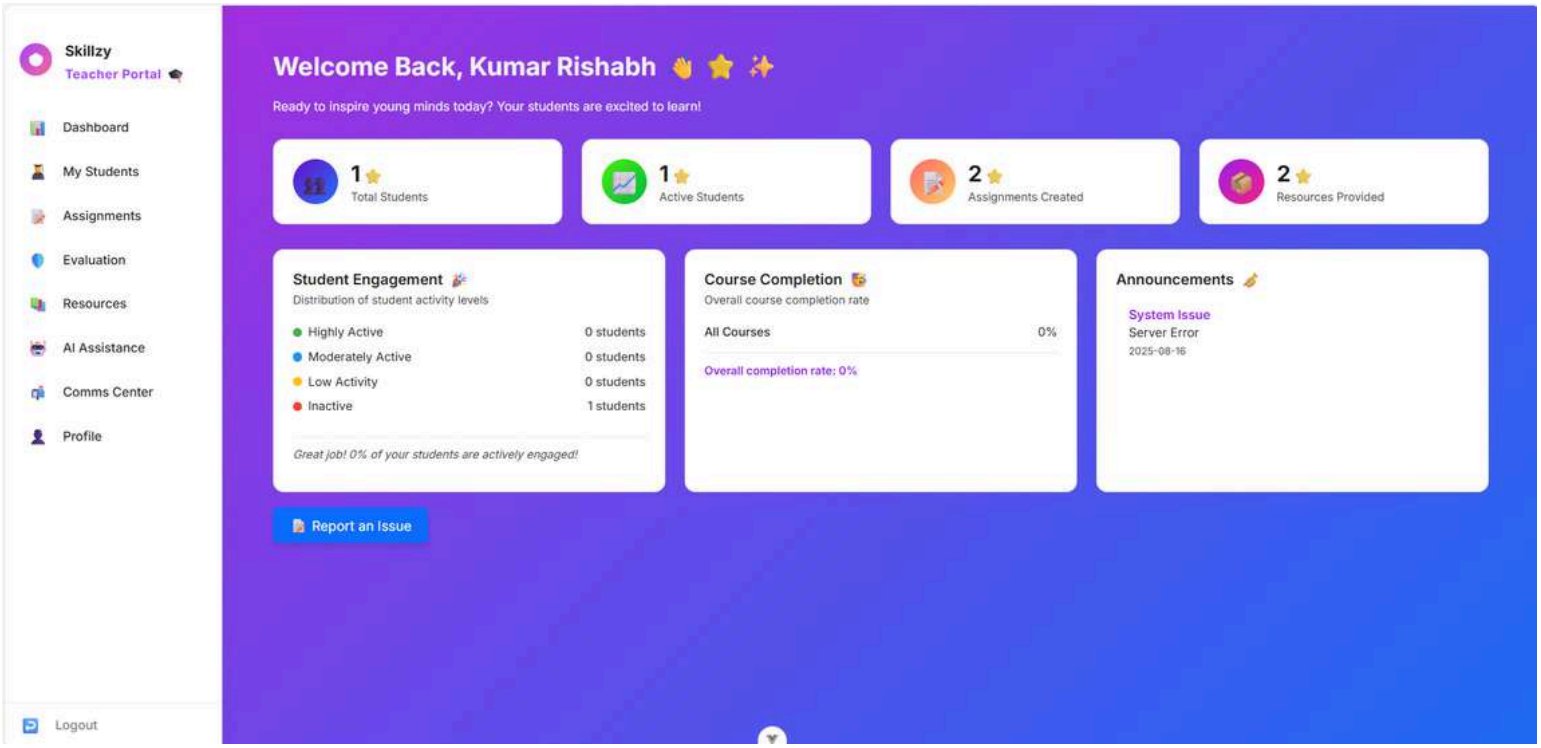


Student Profile

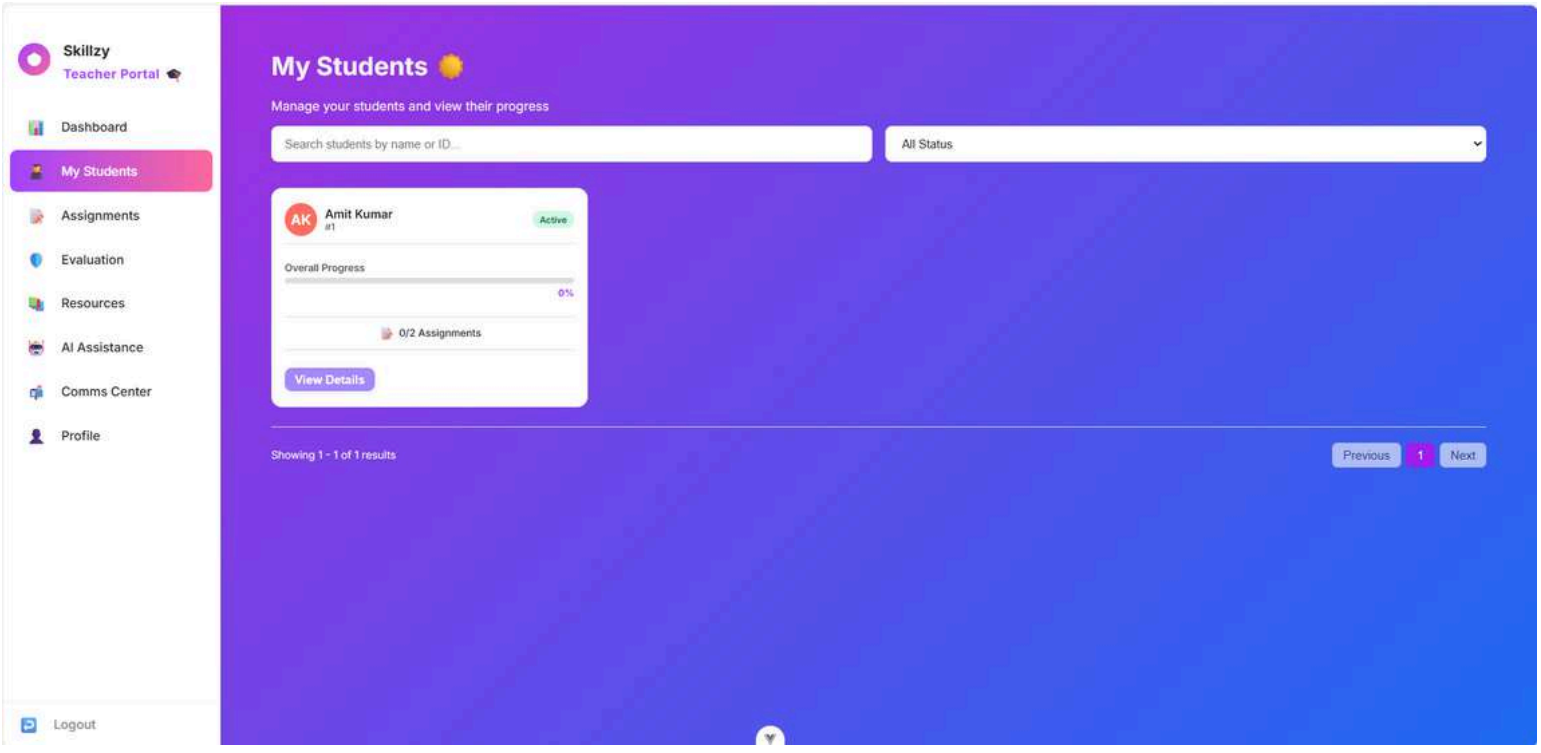


USER INTERFACE SCREENS

Instructor Dashboard

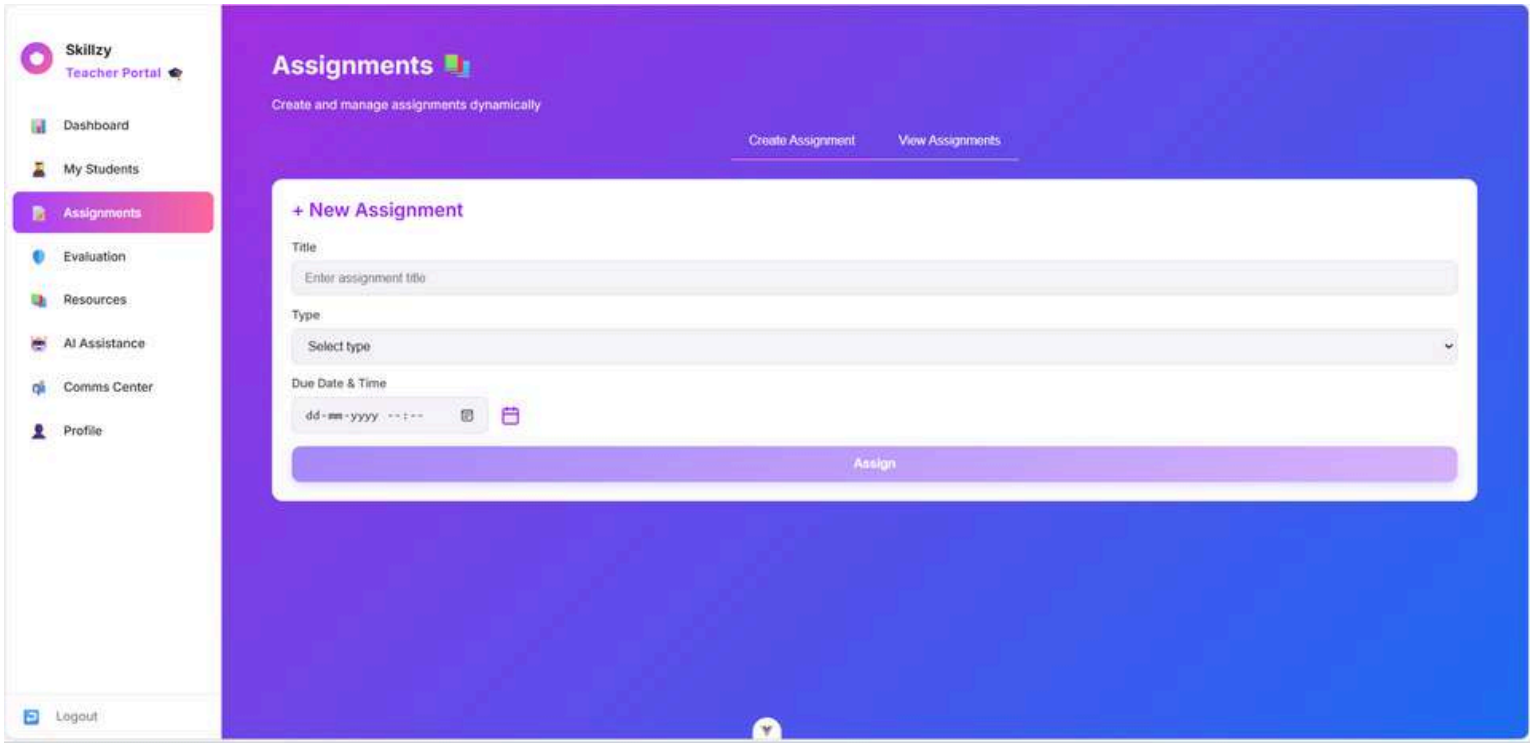


Instructor Students

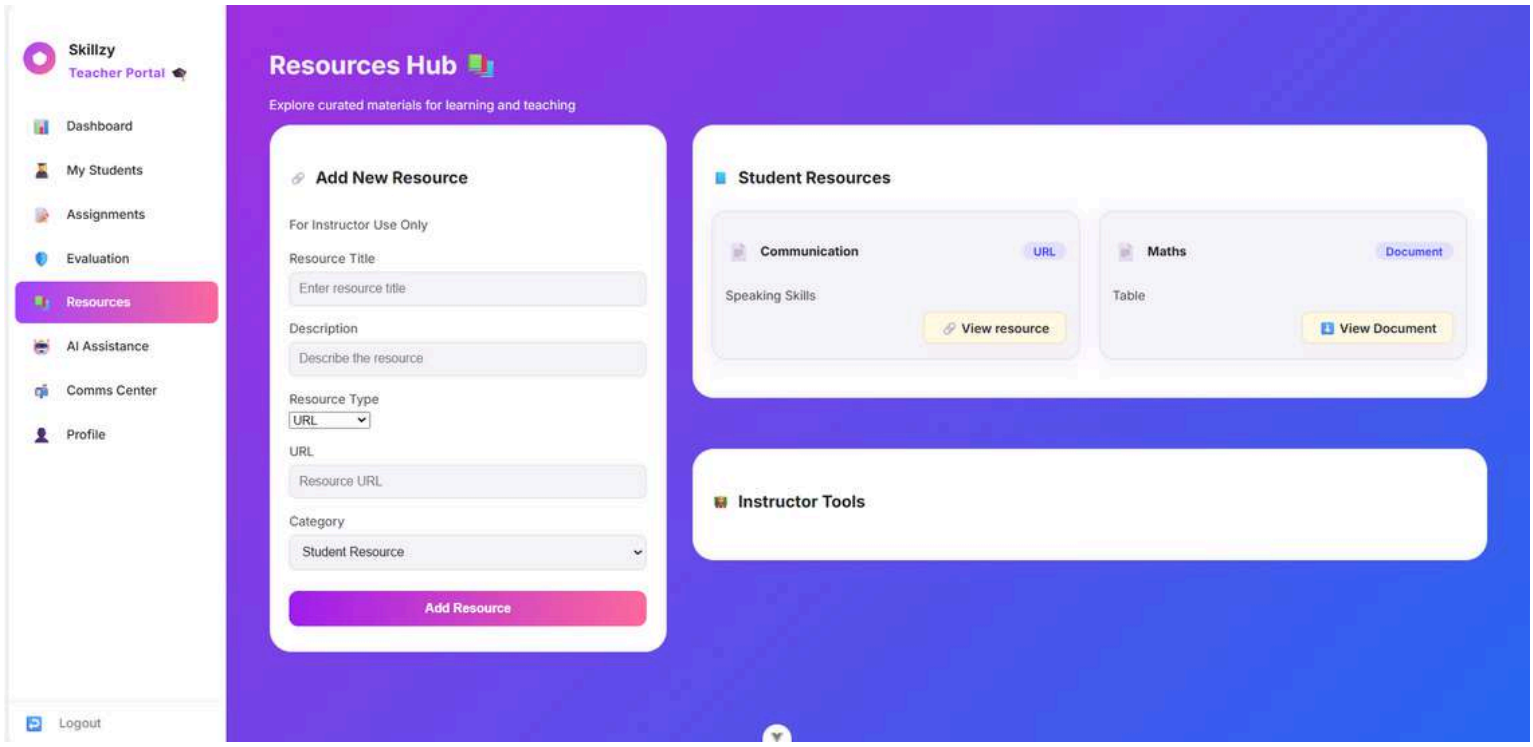


USER INTERFACE SCREENS

Instructor Assignments

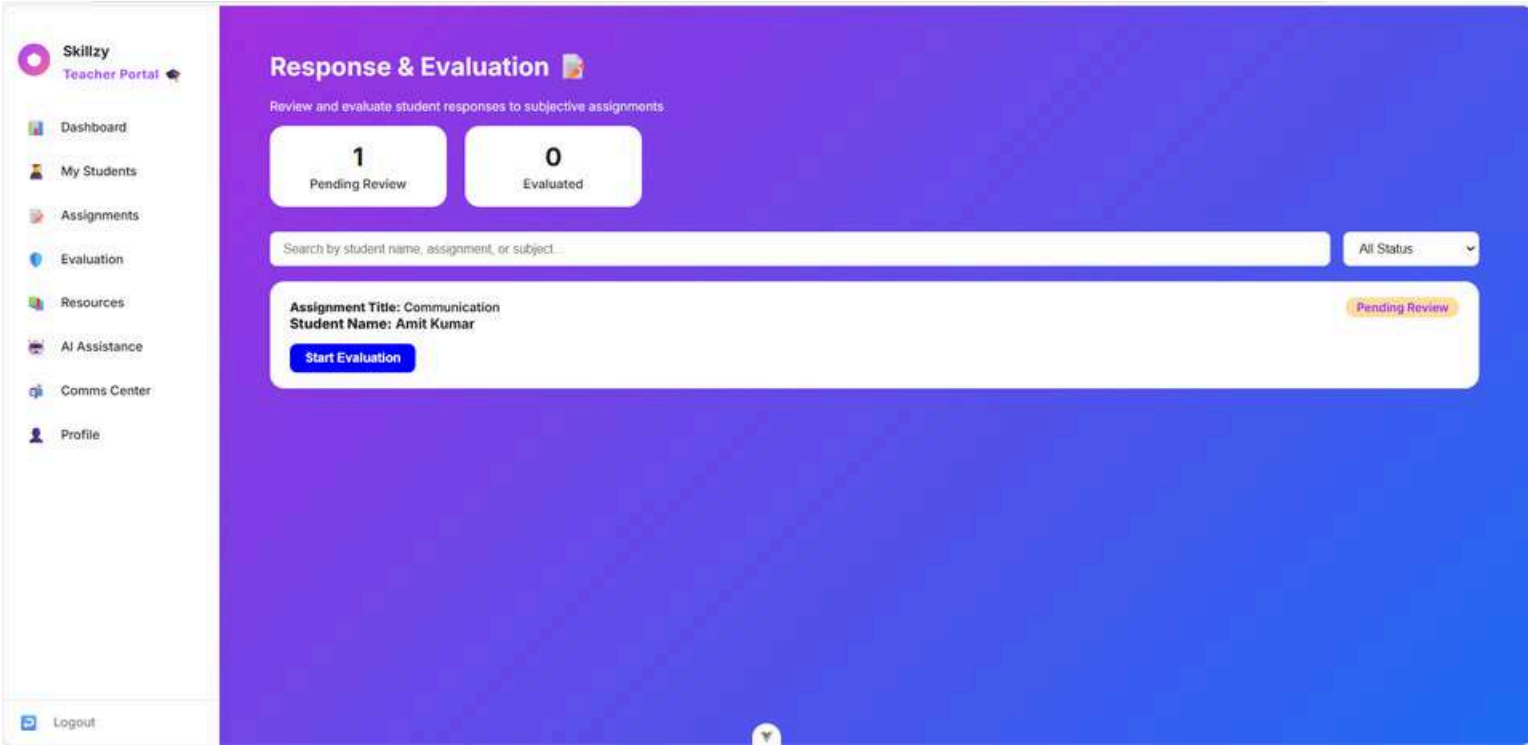


Instructor Resources

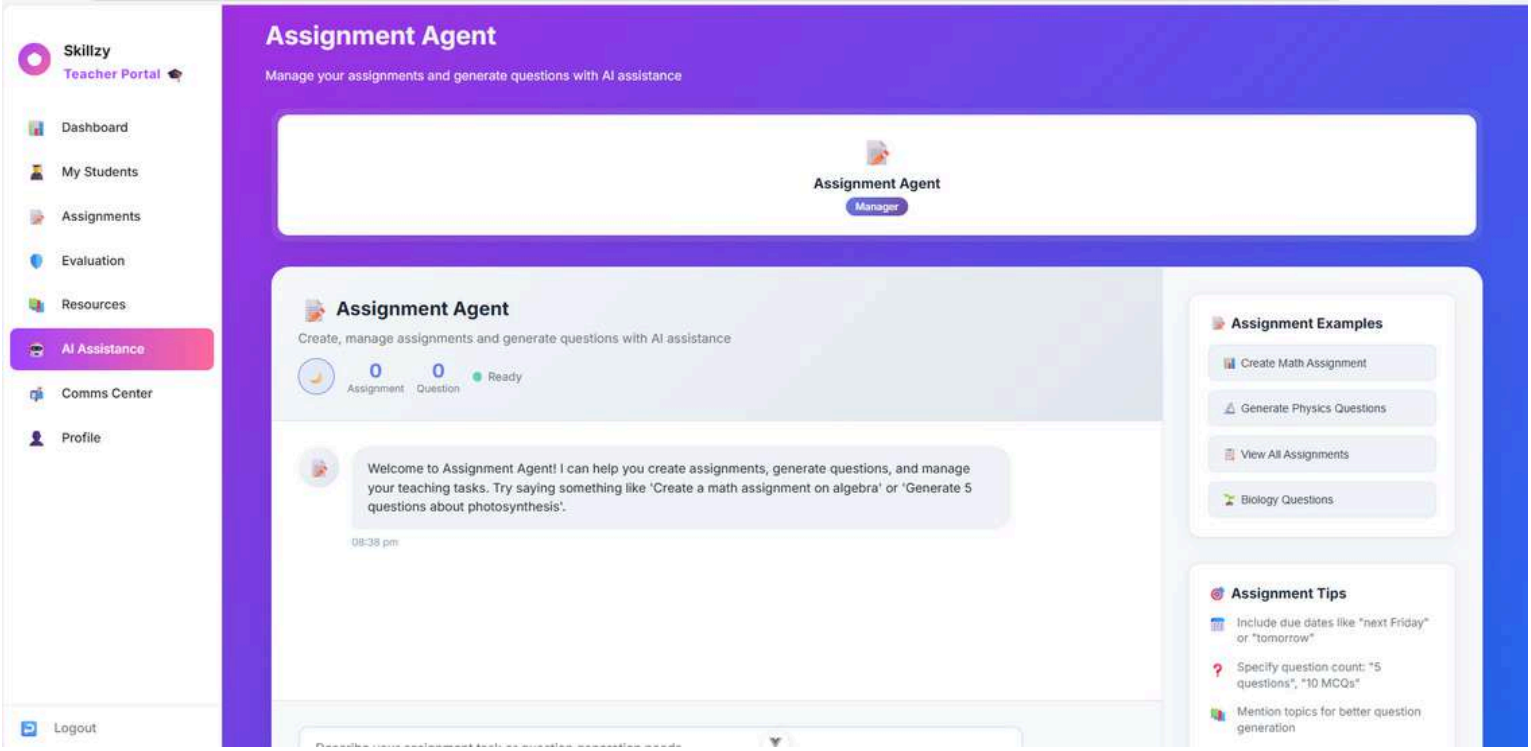


USER INTERFACE SCREENS

Instructor Assignment Evaluation

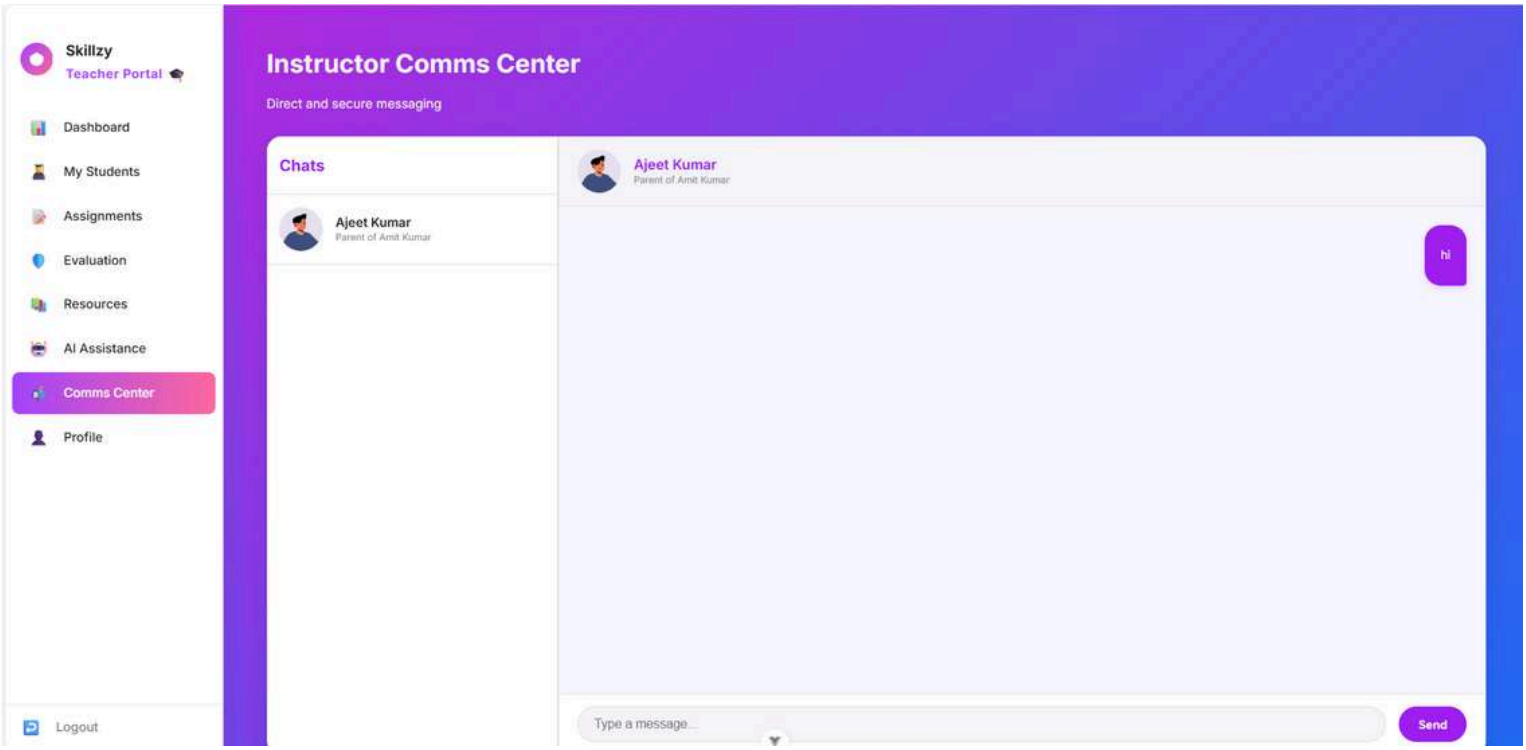


Instructor AI Task Agent

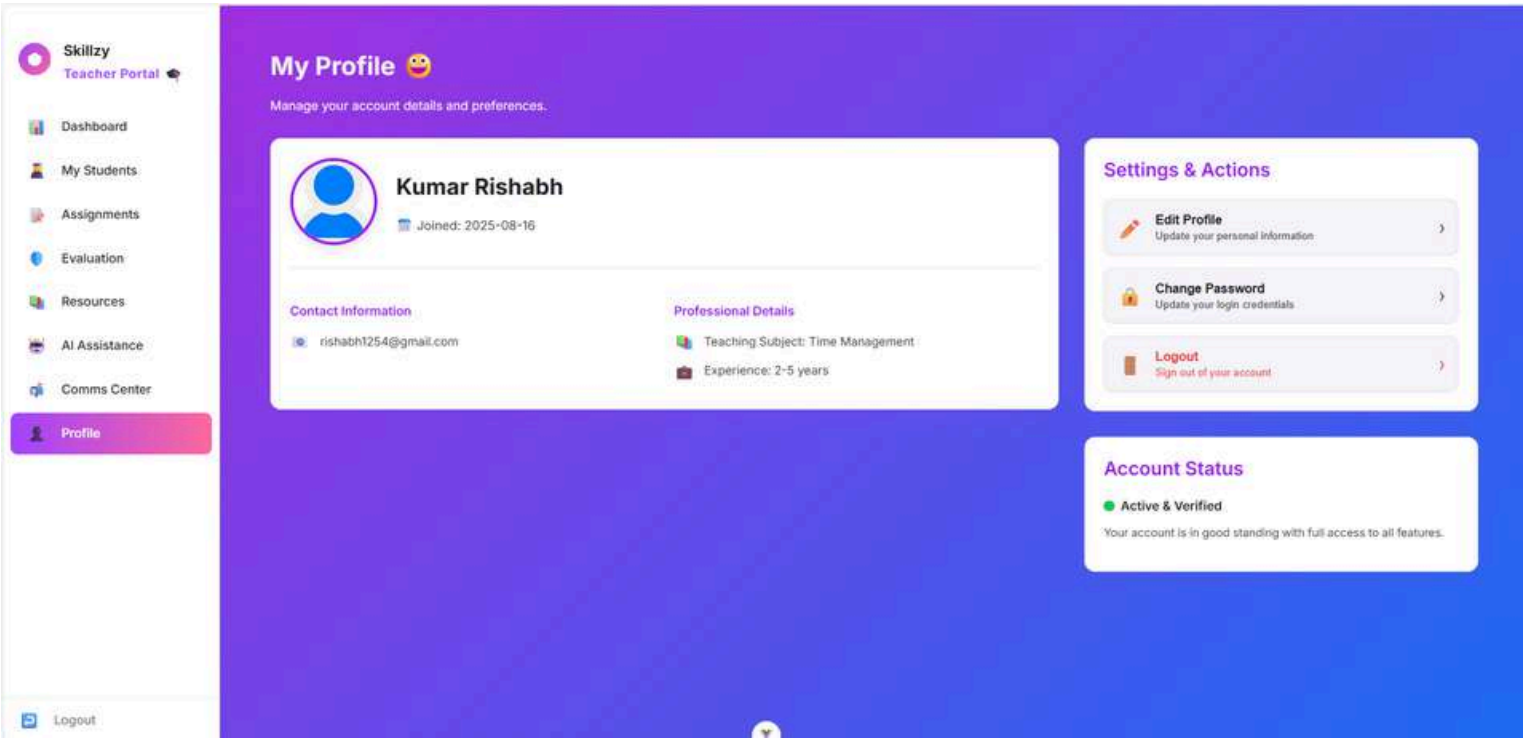


USER INTERFACE SCREENS

Instructor Comms

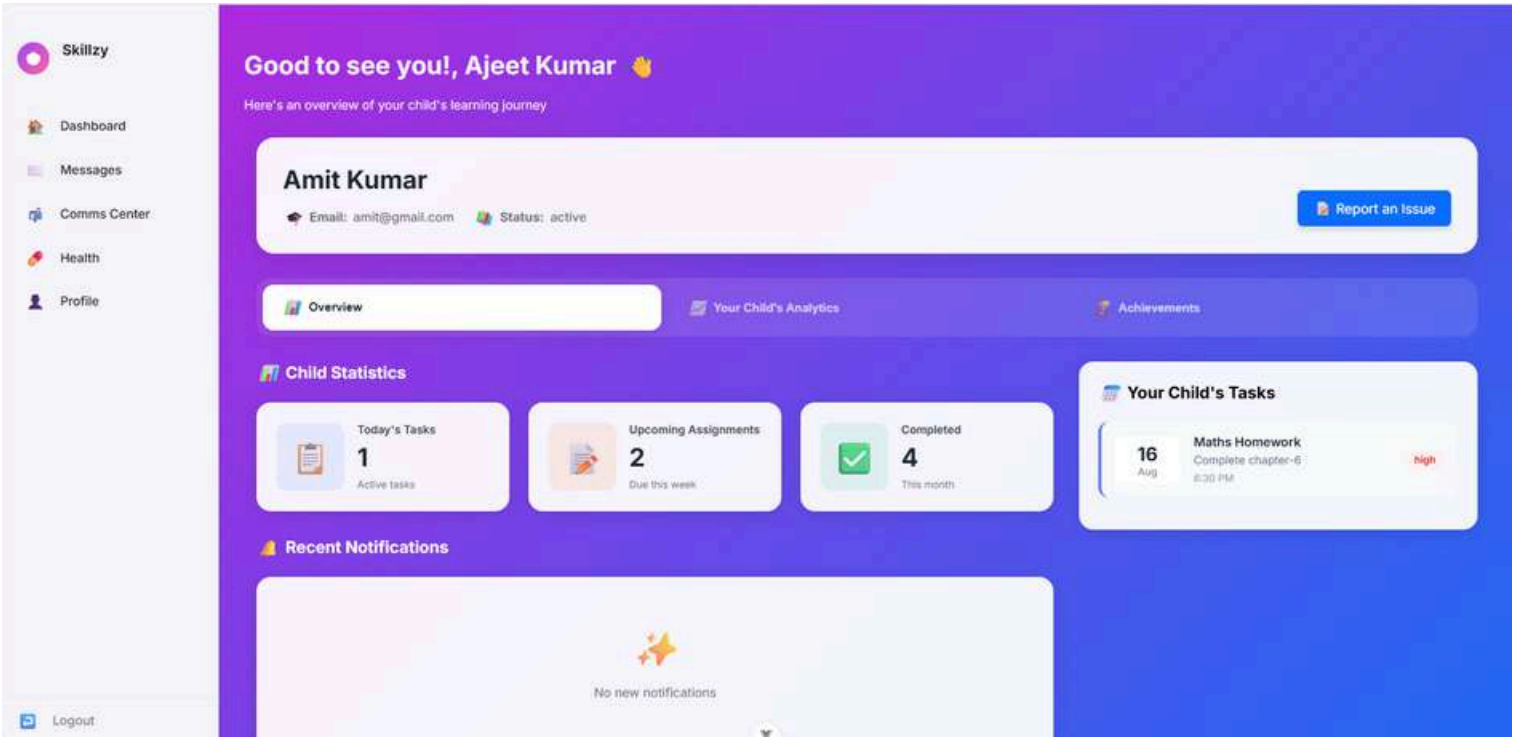


Instructor Profile

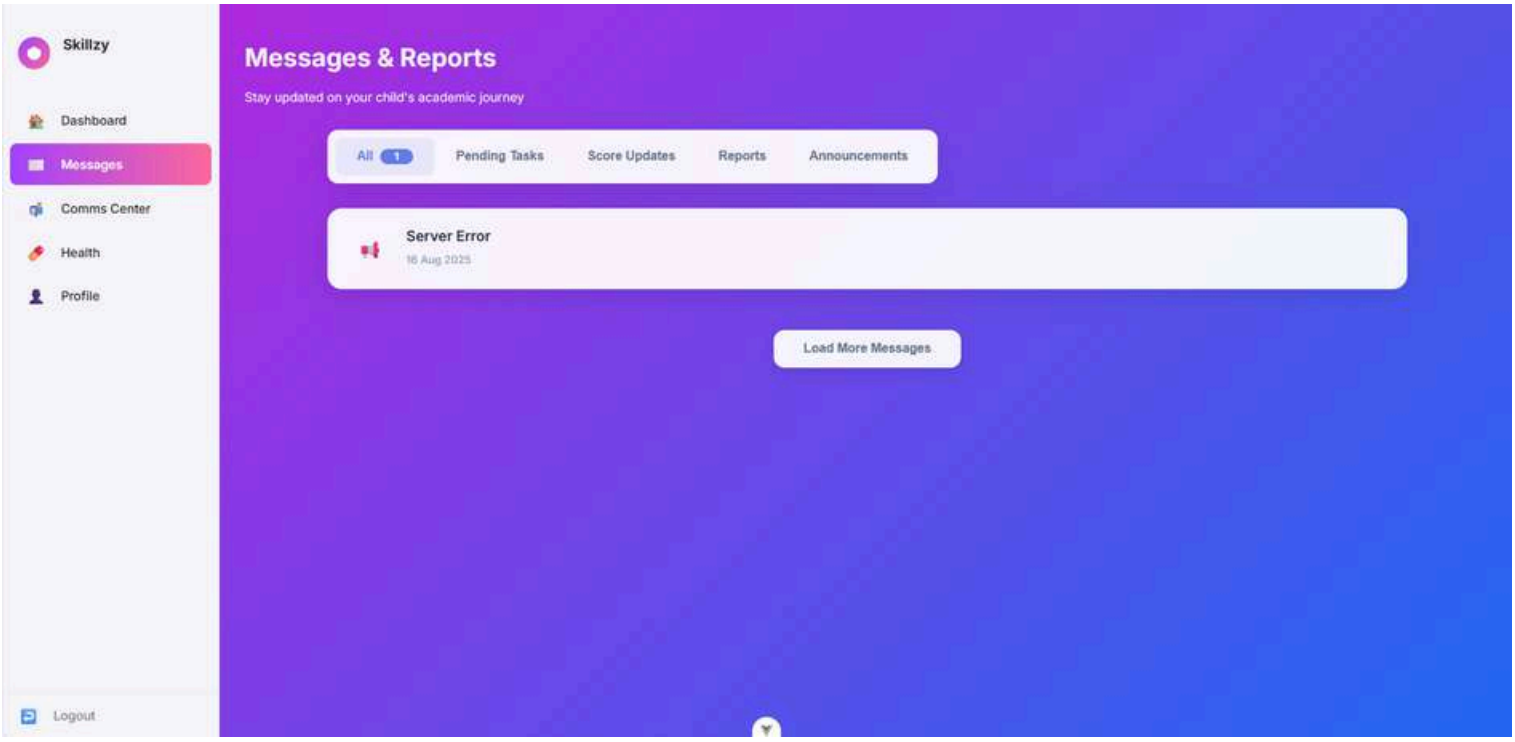


USER INTERFACE SCREENS

Parent Dashboard

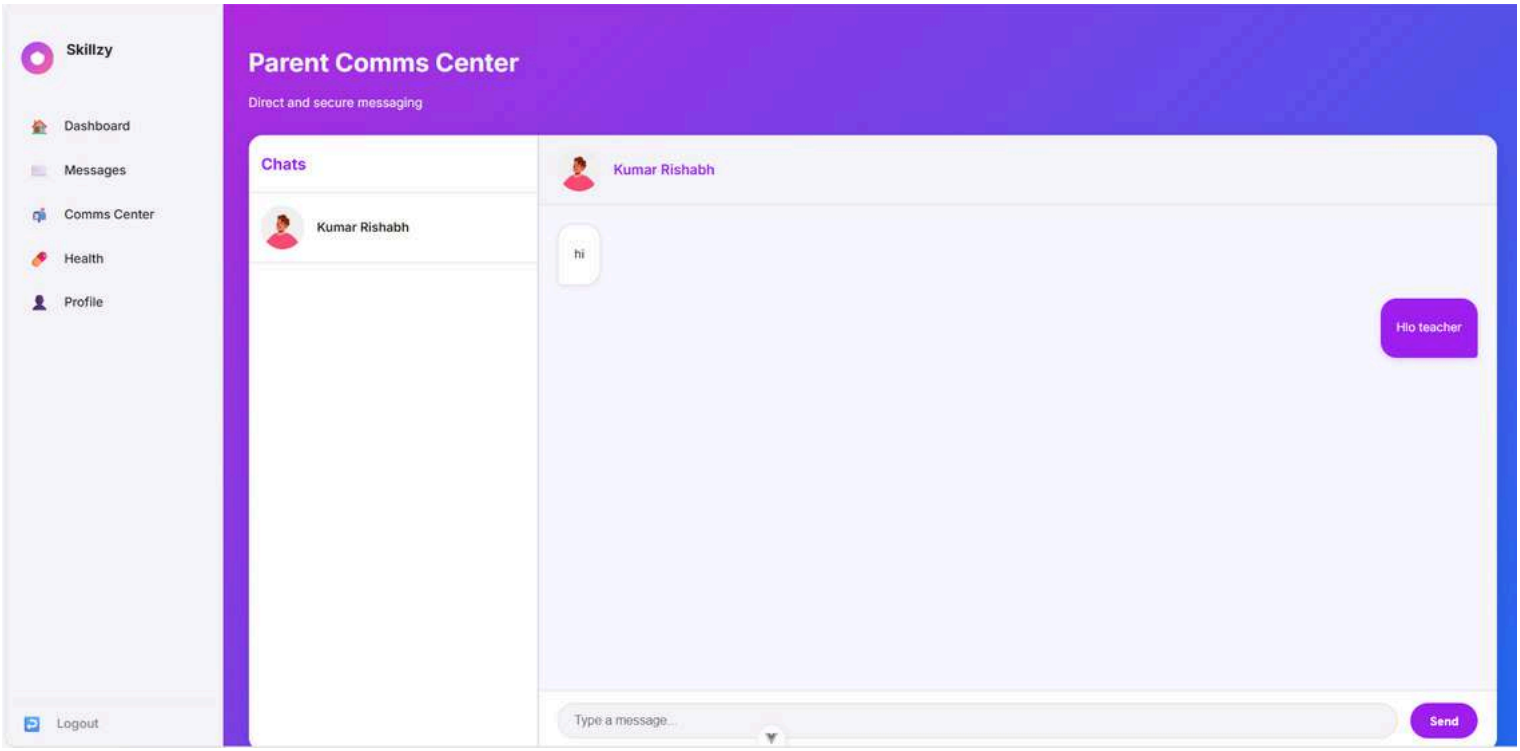


Parent Messages & Reports

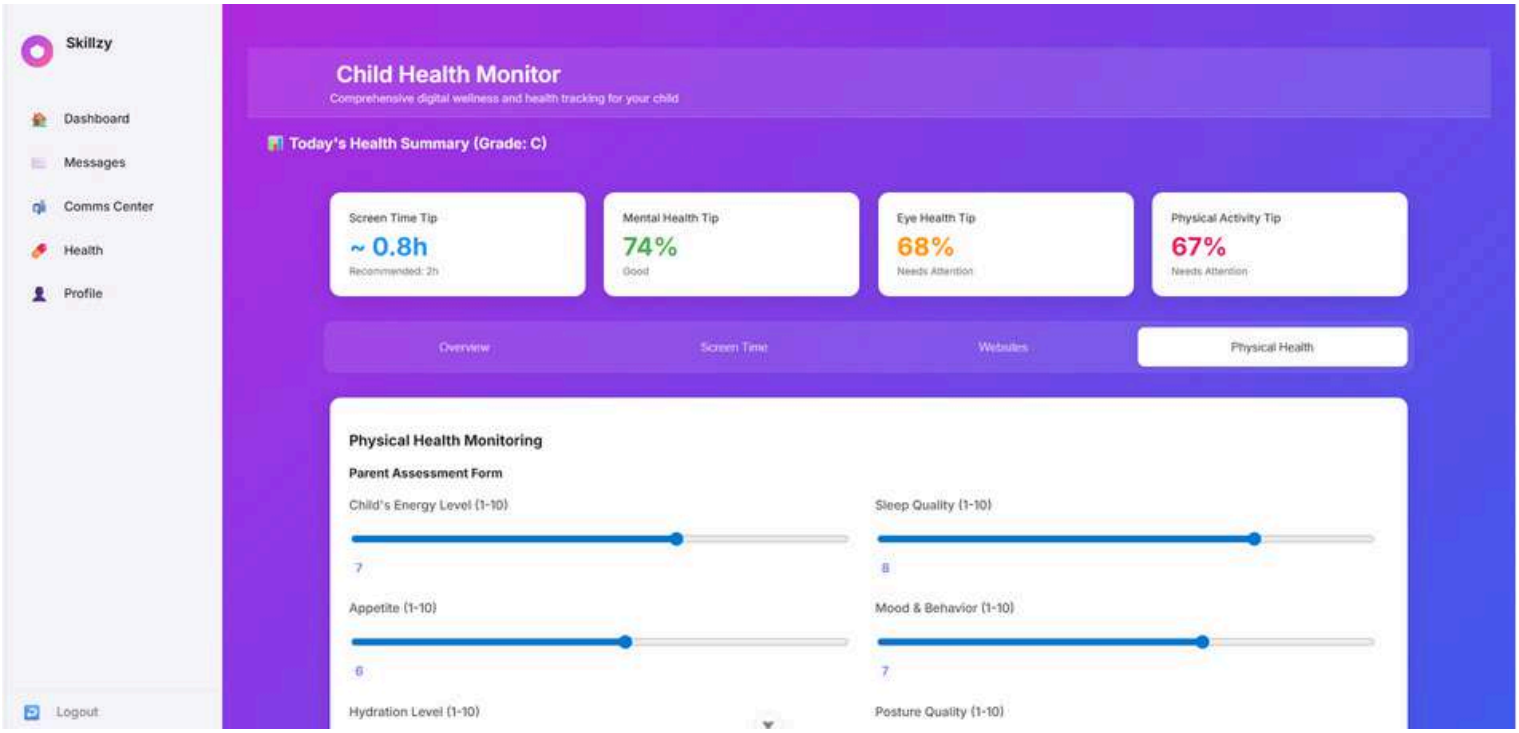


USER INTERFACE SCREENS

Parent Communication Center

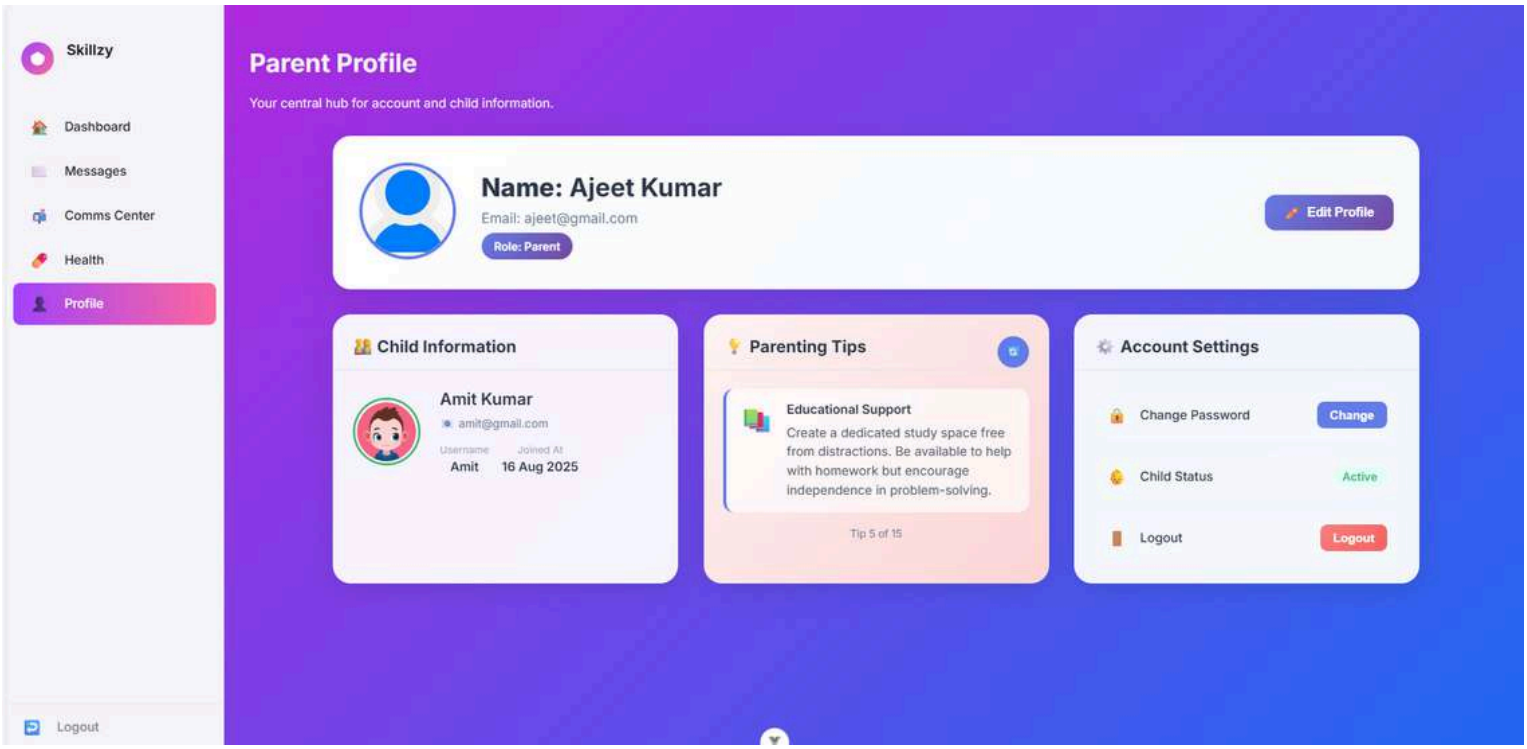


Health Center



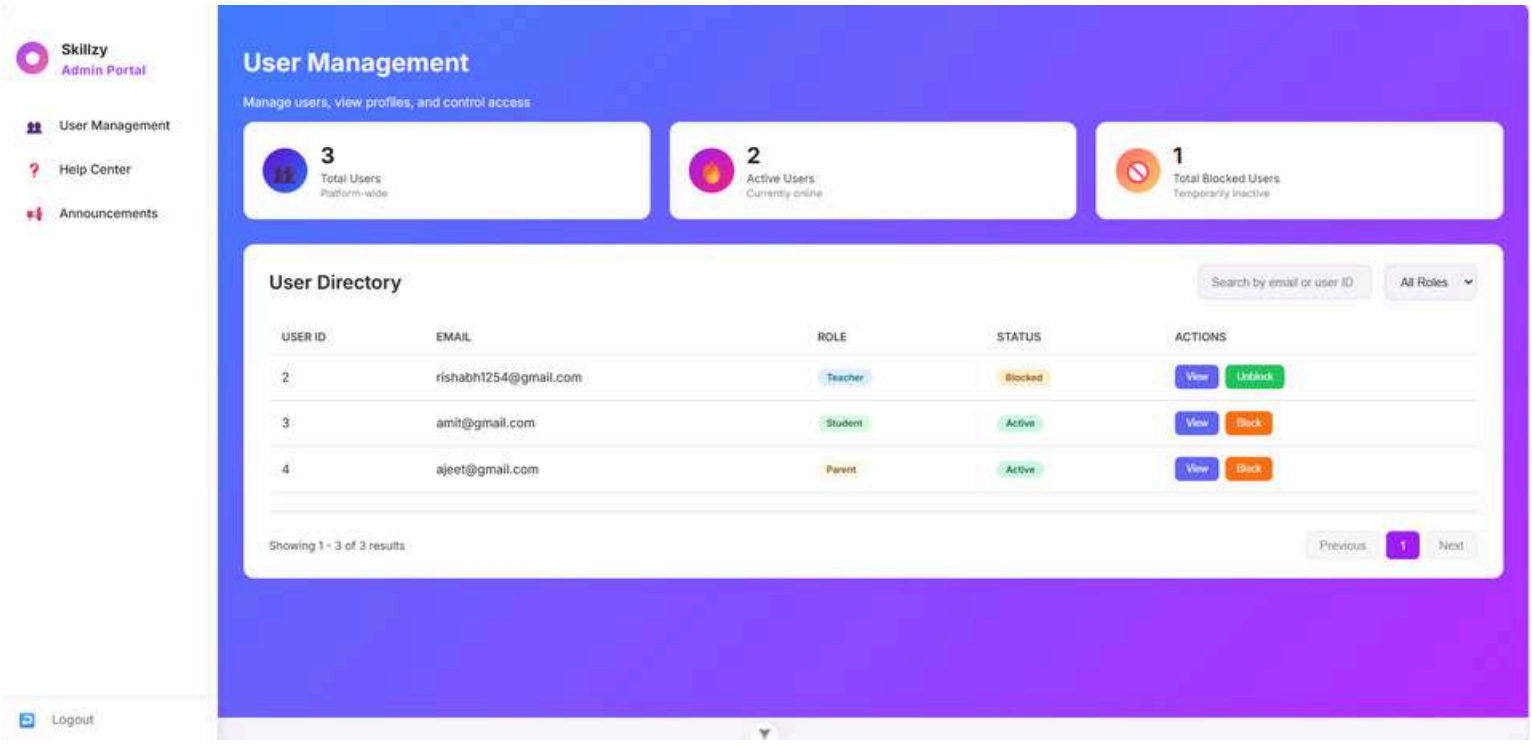
USER INTERFACE SCREENS

Parent Profile

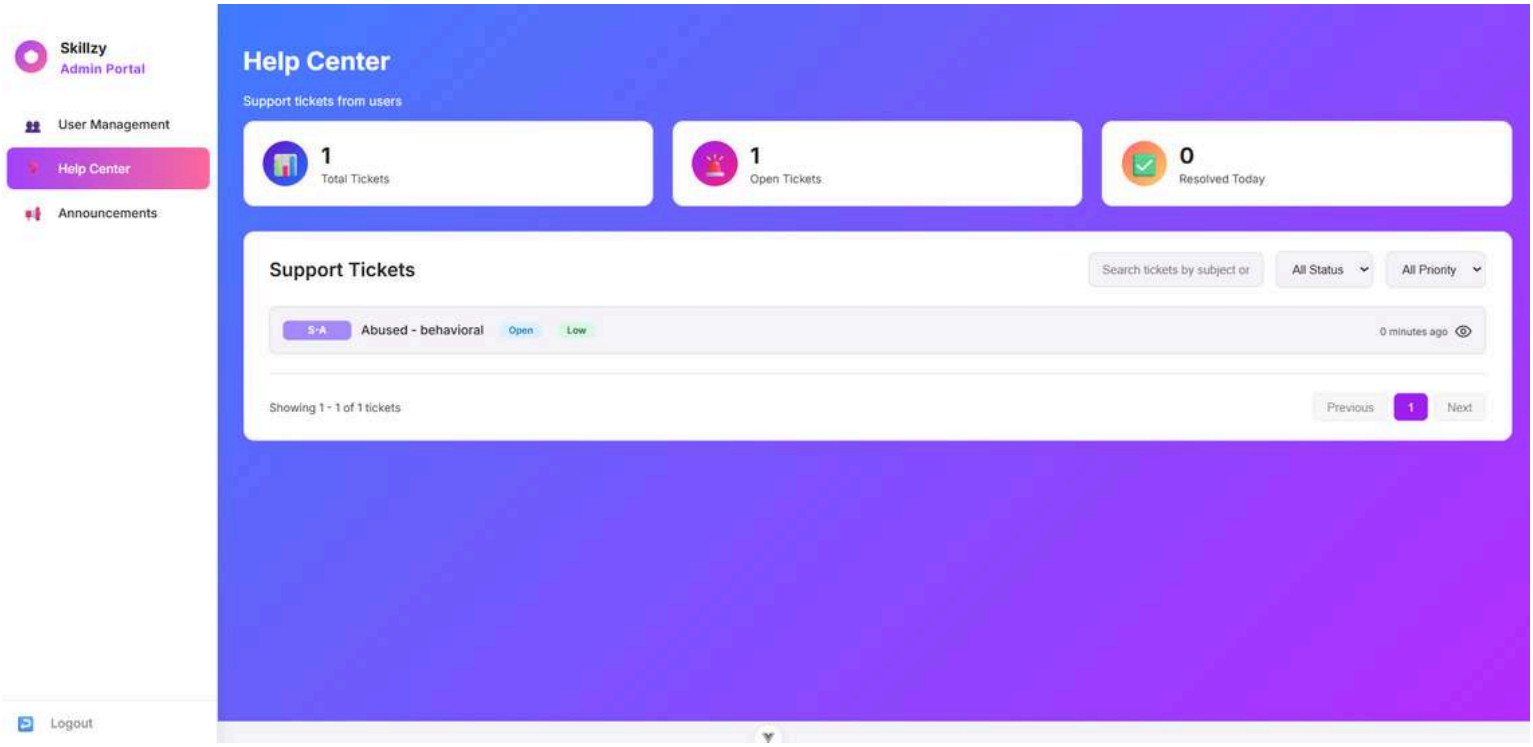


USER INTERFACE SCREENS

Admin UserManagement

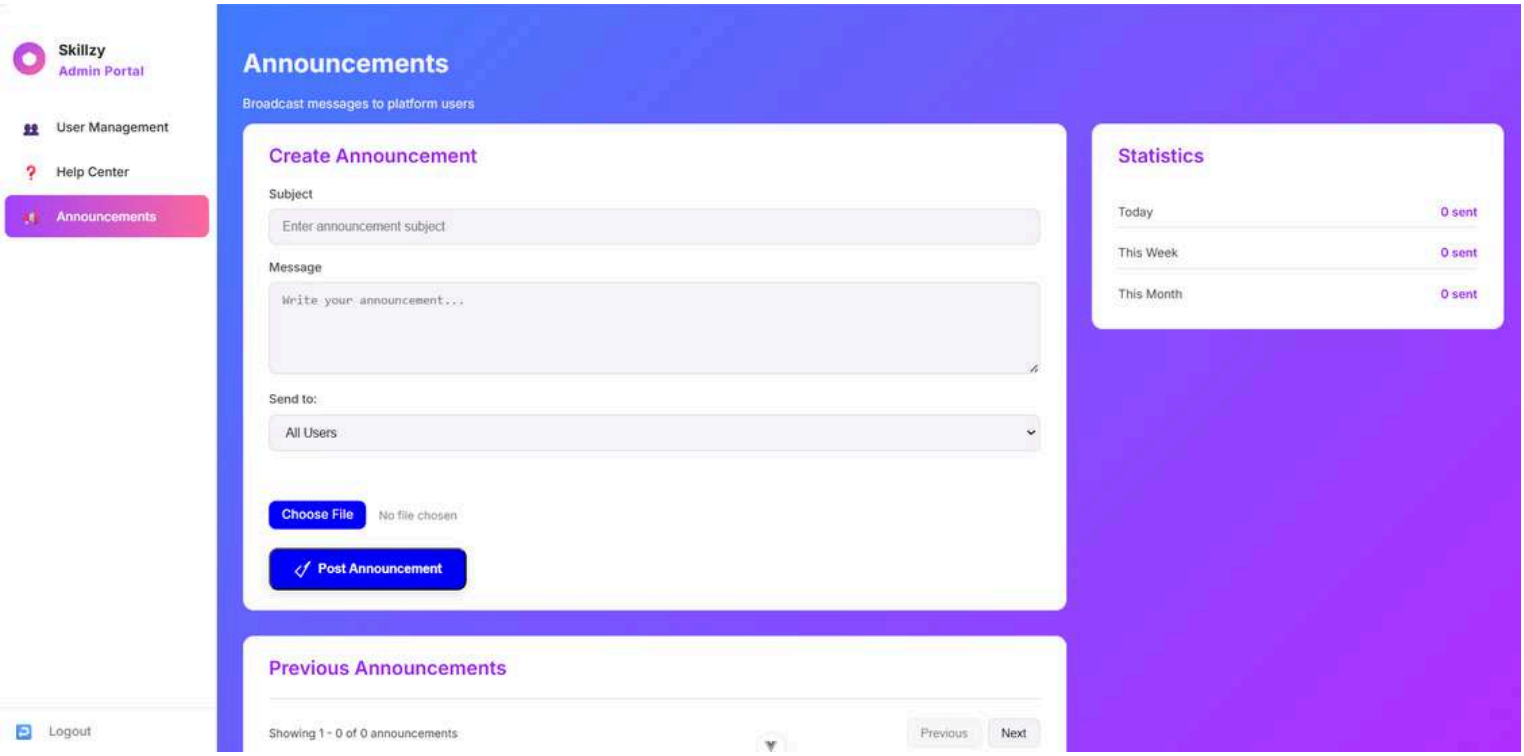


Admin HelpCenter



USER INTERFACE SCREENS

Admin Announcements



MILESTONE-4

API ENDPOINTS

YAML Link:

https://drive.google.com/file/d/1_75cF3psnaP7NMZDH36NqC9NMBUV9QCJ/view?usp=sharing

API ENDPOINT: HOME AND USER

Home

POST / Home

User

POST /api/v1/user/login Login

POST /api/v1/user/create_user Register User

POST /api/v1/user/{user_id}/report Report User

API ENDPOINT: STUDENT

Student

GET /api/v1/student/{user_id}/dashboard Student Dashboard

GET /api/v1/student/{user_id}/calendar Get Student Calendar

POST /api/v1/student/{user_id}/calendar Create Calendar Task

PUT /api/v1/student/{user_id}/calendar/{task_id} Update Calendar Task

DELETE /api/v1/student/{user_id}/calendar/{task_id} Delete Calendar Task

PUT /api/v1/student/{user_id}/calendar/{task_id}/complete Mark Calendar Task Complete

GET /api/v1/student/user/{user_id}/assignments Get Assignments

API ENDPOINT: STUDENT

POST	/api/v1/student/{user_id}/assignments/{assignment_id}/submit	Submit Assignment
GET	/api/v1/student/{user_id}/badges	Get Badges
GET	/api/v1/student/{user_id}/resources	Get Knowledge Base
GET	/api/v1/student/{user_id}/messages	Get Student Messages
GET	/api/v1/student/{user_id}/profile	Get Student Profile
PUT	/api/v1/student/{user_id}/profile	Update Student Profile
GET	/api/v1/student/{user_id}/notifications	Get Notifications
POST	/api/v1/student/{user_id}/focus_mode/start	Focus mode start
PUT	/api/v1/student/{user_id}/focus_mode/{focus_id}/end	Focus mode end

API ENDPOINT: PARENT

Parents

GET	/api/v1/parent/{parent_id}/profile	Get Parent Profile
PUT	/api/v1/parent/{parent_id}/profile	Edit Parent Profile
GET	/api/v1/parent/child_stats/{parent_id}	Get Child Stats
GET	/api/v1/parent/{parent_id}/child/profile	Get Parent Children
GET	/api/v1/parent/tasks/{parent_id}	Get Tasks For Student
GET	/api/v1/parent/assignments/{parent_id}	Get Assignments For Student
GET	/api/v1/parent/achievements/{parent_id}	Get Achievements For Student
GET	/api/v1/parent/notifications/{parent_id}	Get Notifications For Parent

API ENDPOINT: TEACHER

Teachers

GET	`/api/v1/teacher/{teacher_id}`	Get Teacher
GET	`/api/v1/teacher/{teacher_id}/dashboard`	Get Teacher Dashboard
GET	`/api/v1/teacher/{teacher_id}/profile`	Get Teacher Profile
PUT	`/api/v1/teacher/{teacher_id}/editprofile`	Edit Teacher Profile
POST	`/api/v1/teacher/{teacher_id}/resources`	Create Resource For Teacher
GET	`/api/v1/teacher/{teacher_id}/resources`	Get Resources For Teacher
GET	`/api/v1/teacher/{teacher_id}/students`	Get Students For Teacher
GET	`/api/v1/teacher/student/{student_id}`	Get Student Details
POST	`/api/v1/teacher/{teacher_id}/assignments`	Create Assignment
GET	`/api/v1/teacher/{teacher_id}/assignments`	Get Assignments
PUT	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}`	Update Assignment
POST	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions`	Add Question
PUT	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}`	Update Question
DELETE	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}`	Delete Question
GET	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}/submissions`	Get submission
POST	`/api/v1/teacher/{teacher_id}/assignments/{assignment_id}/evaluation`	Update Question

API ENDPOINT: ADMIN

Admin

GET	/api/v1/admin/dashboard	Get Dashboard Data
GET	/api/v1/admin/user/search	Search User
POST	/api/v1/admin/announcement	Create Announcement
GET	/api/v1/admin/announcements	Get Announcements
PUT	/api/v1/admin/user/{user_id}/block	Block User
PUT	/api/v1/admin/user/{user_id}/unblock	Unblock User
DELETE	/api/v1/admin/user/{user_id}	Delete User

API ENDPOINT: AI CHATBOT AND CHAT

AI Chatbot

POST	/api/v1/student/{user_id}/bot	Interact with AI Chatbot (Text/Voice)

Chat

GET	/api/v1/chat/conversations	Get Teacher Conversations
GET	/api/v1/chat/parent/{parent_id}/contact	Get Parent Contact
GET	/api/v1/chat/teacher/{teacher_id}/contacts	Get Teacher Contacts

MILESTONE-5

TEST SUITE

IITM BS DEGREE PROGRAM

INTRODUCTION

This document presents a comprehensive suite of test cases designed for a web application that helps children aged 8 to 14 build essential life skills. The application focuses on areas such as time management, emotional intelligence, effective communication, and healthy habits. Developed with input from real users, the app aims to be interactive, age-appropriate, and deeply relevant to the challenges faced by children in this age group.

To ensure the reliability, usability, and effectiveness of the platform, this document outlines a structured testing plan that evaluates the performance and functionality of core system components. These test cases serve as a guide for validating the application's readiness to deliver a seamless, engaging learning experience for its young users.

The testing framework for this project is built using pytest, a flexible and developer-friendly Python testing library. Each test case is thoughtfully designed to simulate child-friendly interactions and ensure that the content and features are accessible, safe, and responsive. These tests ensure quick identification of bugs and consistent delivery of high-quality updates. By maintaining rigorous testing at every stage, the project aims to provide a stable, enjoyable, and empowering platform for young learners.

AUTH TESTCASE: VALID REGISTRATION

```python
async def test_register_valid(async_client):
    payload = {
        "username": "testuser",
        "email": "testuser@example.com",
        "password": "securepass123",
        "role_name": "student",
        "full_name": "Test User",
        "class_id": 1,
        "subject_id": 1
    }
    response = await async_client.post("/api/v1/user/create_user", json=payload)
    print("REGISTER VALID:", response.status_code, response.text)

    # Acceptable: user created OR already exists (your backend returns 500 for both)
    assert response.status_code in [200, 201, 400]
```

- API Endpoint: /api/v1/user/create_user
- Request Method: POST
- Request Body:

{"username": "testuser", "email": "testuser@example.com", "password": "securepass123", "role_name": "student", "full_name": "Test User", "class_id": 1, "subject_id": 1}

- Expected Output:
  - HTTP Status Code: 200 or 201
  - Response JSON:

{"detail": "User created successfully"}

- Actual Output:
  - HTTP Status Code: 200
  - Response JSON:

{"detail": "User created successfully"}

- Result: Success

AUTH TESTCASE: INVALID EMAIL FORMAT

```python
async def test_register_invalid_email_format(async_client):
    payload = {
        "username": "bademailuser",
        "email": "invalid-email",  # Bad format
        "password": "securepass123",
        "role_name": "student",
        "full_name": "Invalid Email",
        "class_id": 1,
        "subject_id": 1
    }
    response = await async_client.post("/api/v1/user/create_user", json=payload)
    print("REGISTER BAD EMAIL:", response.status_code, response.text)

    assert response.status_code == 400
```

- API Endpoint: /api/v1/user/create_user
- Request Method: POST
- Request Body:

{"username": " testuser", "email": "invalid-email", "password": "securepass123", "role_name": "student", "full_name": "Invalid Email", "class_id": 1, "subject_id": 1}

- Expected Output:
  - HTTP Status Code: 422
  - Response JSON: {"detail": "Invalid email format"}
- Actual Output:
  - HTTP Status Code: 422
  - Response JSON: {"detail": "Invalid email format”
- Result: Success

AUTH TESTCASE: MISSING REQUIRED FIELD (USERNAME)

```python
async def test_register_missing_field(async_client):
    payload = {
        # "username": "missingusername",  # Missing username
        "email": "missing@example.com",
        "password": "securepass123",
        "role_name": "student",
        "full_name": "Missing Field",
        "class_id": 1,
        "subject_id": 1
    }
    response = await async_client.post("/api/v1/user/create_user", json=payload)
    print("REGISTER MISSING FIELD:", response.status_code, response.text)

    assert response.status_code == 400
```

- API Endpoint: /api/v1/user/create_user
- Request Method: POST
- Request Body:

{"email": "missing@example.com", "password": "securepass123", "role_name": "student", "full_name": "Missing Field", "class_id": 1, "subject_id": 1}

- Expected Output:
  - HTTP Status Code: 422
  - Response JSON: {"detail": "Field validation error"}
- Actual Output:
  - HTTP Status Code: 422
  - Response JSON: {"detail": "Field validation error"}
- Result: Success

AUTH TESTCASE: VALID LOGIN

```python
async def test_login_valid(async_client):
    headers = {"Content-Type": "application/x-www-form-urlencoded"}
    data = {
        "username": "testuser",
        "password": "securepass123"
    }
    response = await async_client.post("/api/v1/user/login", data=data, headers=headers)
    print("LOGIN VALID:", response.status_code, response.text)
    assert response.status_code == 200
    assert "access_token" in response.json()
```

- API Endpoint: /api/v1/user/login
- Request Method: POST
- Request Body: {"username": "testuser", "password": "securepass123"}
- Expected Output:
  - HTTP Status Code: 200 OK
  - Response JSON:

    {"access_token": "<jwt_token>", "token_type": "bearer"}
- Actual Output:
  - HTTP Status Code: 200 OK
  - Response JSON:

    {"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...", "token_type": "bearer"
- Result: Success

AUTH TESTCASE: INVALID LOGIN PASSWORD

```python
async def test_login_invalid_credentials(async_client):
    headers = {"Content-Type": "application/x-www-form-urlencoded"}
    data = {
        "username": "testuser",
        "password": "wrongpass"
    }
    response = await async_client.post("/api/v1/user/login", data=data, headers=headers)
    print("LOGIN WRONG PASSWORD:", response.status_code, response.text)

    # Should return 401, but your backend returns 500
    assert response.status_code == 401
```

- API Endpoint: /api/v1/user/login
- Request Method: POST
- Request Body: {"username": "testuser", "password": "wrongpass"}
- Expected Output:
  - HTTP Status Code: 401 Unauthorized
  - Response JSON:

    {"detail": "Invalid username or password"}
- Actual Output:
  - HTTP Status Code: 401 Unauthorized
  - Response JSON:

    {"detail": "Invalid username or password"}
- Result: Success

STUDENT TESTCASE: GET STUDENT DASHBOARD - VALID STUDENT

```python
async def test_dashboard_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/dashboard", headers=headers)
    assert response.status_code == 200
    assert "full_name" in response.json()
```

API Endpoint: /api/v1/student/3/dashboard
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 3

Expected Output:
HTTP Status Code: 200

JSON: {"full_name": "Amit", "dashboard_data": {...}}

Actual Output:
HTTP Status Code: 200

JSON: {"full_name": "Amit", "dashboard_data": {...}}

Result: Success

STUDENT TESTCASE: GET STUDENT DASHBOARD - INVALID STUDENT

```python
async def test_dashboard_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/dashboard", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/999/dashboard
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 999

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET STUDENT DASHBOARD - MISSING TOKEN

```python
async def test_dashboard_missing_token(async_client):
    response = await async_client.get("/api/v1/student/3/dashboard")
    assert response.status_code == 401
```

API Endpoint: /api/v1/student/3/dashboard
Request Method: GET

Expected Output:
HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:
HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: Success

STUDENT TESTCASE: GET CALENDAR - VALID STUDENT

```python
async def test_calendar_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/calendar", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

API Endpoint: /api/v1/student/3/calendar
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 3

Expected Output:
HTTP Status Code: 200

JSON: [{"task_id": 1, "title": "Math Revision", "due_date": "2025-08-01T10:00:00"}, ...]

Actual Output:
HTTP Status Code: 200

JSON: [{"task_id": 1, "title": "Math Revision", "due_date": "2025-08-01T10:00:00"}, ...]

Result: Success

STUDENT TESTCASE: GET CALENDAR - INVALID STUDENT

```python
async def test_calendar_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/calendar", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/999/calendar
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 999

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET CALENDAR - UNAUTHENTICATED

```python
async def test_calendar_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/calendar", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /student/2/calendar
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: CREATE TASK - VALID STUDENT

```python
async def test_create_task_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    data = {
        "title": "Finish Homework",
        "description": "Math and Science",
        "due_date": "2025-08-15T17:00:00"
    }
    response = await async_client.post("/api/v1/student/3/calendar", headers=headers, json=data)
    assert response.status_code == 200
    assert "title" in response.json()
```

API Endpoint: /api/v1/student/3/calendar
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

JSON: {"title": "Finish Homework", "description": "Math and Science", "due_date": "2025-08-15T17:00:00"}

Expected Output:
HTTP Status Code: 200

JSON: {"task_id": 1, "title": "Finish Homework", "description": "Math and Science", "status": "pending",

"due_date": "2025-08-15T17:00:00"}

Actual Output:
HTTP Status Code: 200

JSON: {"task_id": 1, "title": "Finish Homework", "description": "Math and Science", "status": "pending",

"due_date": "2025-08-15T17:00:00"}

Result: Success

STUDENT TESTCASE: CREATE TASK - INVALID STUDENT

```python
async def test_create_task_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    data = {
        "title": "Fail Test",
        "description": "Testing invalid user",
        "due_date": "2025-08-15T17:00:00"
    }
    response = await async_client.post("/api/v1/student/999/calendar", headers=headers, json=data)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/999/calendar
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

JSON: {"title": "Fail Test", "description": "Testing invalid user", "due_date": "2025-08-15T17:00:00"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: UPDATE TASK - VALID TASK

```python
async def test_update_task_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    update_data = {
        "title": "Updated Homework",
        "description": "Updated desc",
        "status": "pending",
        "due_date": "2025-08-20T10:00:00"
    }
    response = await async_client.put("/api/v1/student/3/calendar/1", headers=headers, json=update_data)
    assert response.status_code in [200, 404]
```

API Endpoint: /api/v1/student/3/calendar/1
Request Method: PUT

Inputs:
Header:

- Authorization: Bearer <token>

JSON: {"title": "Updated Homework", "description": "Updated desc", "status": "pending",

"due_date": "2025-08-20T10:00:00"}

Expected Output:
HTTP Status Code: 200

JSON: {"task_id": 1, "title": "Updated Homework", "description": "Updated desc", "status": "pending",

"due_date": "2025-08-20T10:00:00"}

Actual Output:
HTTP Status Code: 200

JSON: {"task_id": 1, "title": "Updated Homework", "description": "Updated desc", "status": "pending",

"due_date": "2025-08-20T10:00:00"}

Result: Success

STUDENT TESTCASE: UPDATE TASK - NOT FOUND

```python
async def test_update_task_not_found(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    update_data = {
        "title": "Nonexistent Task",
        "description": "Should fail",
        "status": "pending"
    }
    response = await async_client.put("/api/v1/student/3/calendar/999", headers=headers, json=update_data)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/3/calendar/999
Request Method: PUT

Inputs:
Header:

- Authorization: Bearer <token>

JSON: {"title": "Nonexistent Task", "description": "Should fail", "status": "pending"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found”}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found}

Result: Success

STUDENT TESTCASE: DELETE TASK - VALID TASK

```python
async def test_delete_task_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    # Task ID 1 must exist for this to pass
    response = await async_client.delete("/api/v1/student/3/calendar/1", headers=headers)
    assert response.status_code in [200, 404]
```

API Endpoint: /api/v1/student/3/calendar/1
Request Method: DELETE

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 200

JSON: {"message": "Task deleted successfully"}

Actual Output:
HTTP Status Code: 200

JSON: {"message": "Task deleted successfully"}

Result: Success

STUDENT TESTCASE: DELETE TASK - NOT FOUND

```python
async def test_delete_task_not_found(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.delete("/api/v1/student/3/calendar/9999", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/3/calendar/9999
Request Method: DELETE

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found"}

Result: Success

STUDENT TESTCASE: MARK TASK COMPLETE - VALID TASK

```python
async def test_mark_task_complete_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.put("/api/v1/student/3/calendar/1/complete", headers=headers)
    assert response.status_code in [200, 404]
```

API Endpoint: /api/v1/student/3/calendar/1/complete
Request Method: PUT

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 200

JSON: {"message": "Task marked as complete"}

Actual Output:
HTTP Status Code: 200

JSON: {"message": "Task marked as complete"}

Result: Success

STUDENT TESTCASE: MARK TASK COMPLETE - TASK NOT FOUND

```python
async def test_mark_task_complete_fail(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.put("/api/v1/student/3/calendar/9999/complete", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/3/calendar/9999/complete
Request Method: PUT

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Task not found"}

Result: Success

STUDENT TESTCASE: GET ASSIGNMENTS - VALID STUDENT

```python
async def test_get_assignments_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/assignments", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

API Endpoint: /api/v1/student/3/assignments
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 3

Expected Output:
HTTP Status Code: 200

JSON: [{"assignment_id": 1, "title": "Algebra Worksheet", "description": "Complete problems 1-10",

"due_date": "2025-08-12T23:59:59"}, ...]

Actual Output:
HTTP Status Code: 200

JSON: [{"assignment_id": 1, "title": "Algebra Worksheet","description": "Complete problems 1-10",

"due_date": "2025-08-12T23:59:59"}, ...]

Result: Success

STUDENT TESTCASE: GET ASSIGNMENTS - INVALID STUDENT

```python
async def test_get_assignments_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/assignments", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/999/assignments
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 999

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET ASSIGNMENTS - UNAUTHENTICATED ACCESS

```python
async def test_get_assignments_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/assignments", headers=headers)
    assert response.status_code == 404
```

API Endpoint: /student/2/assignments
Request Method: GET

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: SUBMIT ASSIGNMENT - ASSIGNMENT NOT ASSIGNED

```python
async def test_submit_assignment_fail_not_assigned(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    answers = [{"question_id": 1, "answer": "A"}]
    response = await async_client.post("/api/v1/student/3/assignments/9999/submit", headers=headers, json=answers)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/3/assignments/9999/submit
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

JSON: [{"question_id": 1,"answer": "A"}]

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found or not assigned"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found or not assigned"}

Result: Success

```python
async def test_submit_assignment_invalid_payload(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.post("/api/v1/student/3/assignments/1/submit", headers=headers, json=[])
    assert response.status_code in [404, 403]
```

Test Case 19: Submit Assignment- Invalid Payload

API Endpoint: /api/v1/student/3/assignments/1/submit
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 403

JSON: {"detail": "Invalid submission payload"}

Actual Output:
HTTP Status Code: 403

JSON: {"detail": "Invalid submission payload"}

Result: Success

STUDENT TESTCASE: SUBMIT SINGLE ANSWER - INVALID ASSIGNMENT

```python
async def test_submit_single_answer_invalid_assignment(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    data = {"answer": "Test Answer"}
    response = await async_client.post("/api/v1/student/3/assignments/9999/questions/1/answer", headers=headers, json=data)
    assert response.status_code == 404
```

API Endpoint: /api/v1/student/3/assignments/9999/questions/1/answer
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

STUDENT TESTCASE: SUBMIT SINGLE ANSWER - INVALID QUESTION

```python
async def test_submit_single_answer_invalid_question(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    data = {"answer": "Some Answer"}
    response = await async_client.post("/api/v1/student/3/assignments/1/questions/9999/answer", headers=headers, json=data)
    assert response.status_code in [403, 404]
```

API Endpoint: /api/v1/student/3/assignments/1/questions/9999/answer
Request Method: POST

Inputs:
Header:

- Authorization: Bearer <token>

Body:

- JSON: {"answer": "Some Answer"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Invalid question"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Invalid question"}

Result: Success

STUDENT TESTCASE: ASSIGNMENT PROGRESS – SUCCESS

```python
async def test_get_assignment_progress_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/assignments/1/progress", headers=headers)
    assert response.status_code in [200, 404]
```

API Endpoint: /api/v1/student/3/assignments/1/progress

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: {"assignment_id": 1, "total_questions": 5, "answered": 2, "score": 4, "status": "in-progress"}

Actual Output:

HTTP Status Code: 200

JSON: {"assignment_id": 1, "total_questions": 5, "answered": 2, "score": 4, "status": "in-progress"}

Result: Success

STUDENT TESTCASE: ASSIGNMENT PROGRESS - INVALID USER

```python
async def test_get_assignment_progress_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/assignments/1/progress", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/assignments/1/progress

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: ASSIGNMENT PROGRESS - INVALID ASSIGNMENT

```python
async def test_get_assignment_progress_invalid_assignment(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/assignments/9999/progress", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/3/assignments/9999/progress

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

STUDENT TESTCASE: GET SCORES – SUCCESS

```python
async def test_get_scores_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/scores", headers=headers)
    assert response.status_code == 200
    assert "statistics" in response.json()
```

Api Endpoint: /api/v1/student/3/scores

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: {"Maths": {"total_score": 75, "average_score": 25.0, "assignments_completed": 3}}

Actual Output

HTTP Status Code: 200

JSON: {"Maths": {"total_score": 75, "average_score": 25.0, "assignments_completed": 3}}

Result: Success

STUDENT TESTCASE: GET SCORES - INVALID USER

```python
async def test_get_scores_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/scores", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/scores

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found”}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET SCORES - UNAUTHENTICATED ACCESS

```python
async def test_get_scores_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/scores", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /student/2/scores

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:

JSON: {"detail": "Not Found"}

HTTP Status Code: 404

Result: Success

STUDENT TESTCASE: START ASSIGNMENT ATTEMPT - ASSIGNMENT NOT FOUND

```python
async def test_start_assignment_attempt_fail_assignment_not_found(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/assignments/9999/attempt", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/3/assignments/9999/attempt

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

STUDENT TESTCASE: START ASSIGNMENT ATTEMPT - INVALID USER

```python
async def test_start_assignment_attempt_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/assignments/1/attempt", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/assignments/1/attempt

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: START ASSIGNMENT ATTEMPT - ASSIGNMENT NOT ASSIGNED TO STUDENT

```python
async def test_start_assignment_attempt_unassigned(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}

    response = await async_client.get("/api/v1/student/3/assignments/3/attempt", headers=headers)
    assert response.status_code in [403, 404]
```

Api Endpoint: /api/v1/student/3/assignments/3/attempt

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 403

JSON: {"detail": "Assignment not assigned to this student"}

Actual Output:

HTTP Status Code: 403

JSON: {"detail": "Assignment not assigned to this student"}

Result: Success

STUDENT TESTCASE: GET BADGES – SUCCESS

```python
async def test_get_badges_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/badges", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

Api Endpoint: /api/v1/student/3/badges

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: [{'id': 2, 'badge_type': 'silver'}]

Actual Output:

HTTP Status Code: 200

JSON: [{'id': 2, 'badge_type': 'silver'}]

Result: Success

STUDENT TESTCASE: GET BADGES - INVALID USER

```python
async def test_get_badges_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/badges", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/badges

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

```python
async def test_get_badges_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/badges", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /student/2/badges

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

JSON: {"detail": "Not Found"}

HTTP Status Code: 404

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: GET RESOURCES - SUCCESS (IF RESOURCES EXIST)

```python
async def test_get_resources_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/resources", headers=headers)
    assert response.status_code in [200, 404]
```

Api Endpoint: /api/v1/student/3/resources

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: [{"id": 1, "title": "Intro to Algebra", "link": "https://example.com/algebra.pdf",

"subject": "Mathematics"}, ...]

Actual Output:

HTTP Status Code: 200

JSON: [{"id": 1, "title": "Intro to Algebra", "link": "https://example.com/algebra.pdf",

"subject": "Mathematics"},]

Result: Success

STUDENT TESTCASE: GET RESOURCES - INVALID USER

```python
async def test_get_resources_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/resources", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/resources

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET RESOURCES - UNAUTHENTICATED (INVALID ENDPOINT)

```python
async def test_get_resources_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/resources", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /student/2/resources

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: GET MESSAGES – SUCCESS

```python
async def test_get_messages_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/messages", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

Api Endpoint: /api/v1/student/3/messages

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: [{"id": 1,"sender": "Admin", "message": "Your homework is due tomorrow.", "timestamp": "2025-07-28T14:05:00"}, {"id": 2, "sender": "System", "message": "New badge earned!", "timestamp": "2025-07-28T15:20:00"}]

Actual Output:

HTTP Status Code: 200

JSON: [{"id": 1, "sender": "Teacher A", "message": "Your homework is due tomorrow.", "timestamp": "2025-07-28T14:05:00"}, {"id": 2, "sender": "System", "message": "New badge earned!", "timestamp": "2025-07-28T15:20:00"}]

Result: Success

STUDENT TESTCASE: GET MESSAGES - INVALID USER

```python
async def test_get_messages_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/messages", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/messages

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET MESSAGES - UNAUTHENTICATED (INVALID PATH)

```python
async def test_get_messages_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/2/messages", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /student/2/messages *(Missing /api/v1)*

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: GET NOTIFICATIONS – SUCCESS

```python
async def test_get_notifications_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/3/notifications", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

Api Endpoint: /api/v1/student/3/notifications

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: [{"id": 1, "title": "New Assignment Posted", "description": "Chapter 5 problems due next Monday", "timestamp": "2025-07-28T10:00:00"}, {"id": 2, "title": "Focus Mode Complete", "description": "You completed 25 minutes of focus time!", "timestamp": "2025-07-28T11:45:00"]

Actual Output:

HTTP Status Code: 200

JSON: [{"id": 1, "title": "New Assignment Posted", "description": "Chapter 5 problems due next Monday", "timestamp": "2025-07-28T10:00:00"}, {"id": 2, "title": "Focus Mode Complete", "description": "You completed 25 minutes of focus time!", "timestamp": "2025-07-28T11:45:00"]

Result: Success

STUDENT TESTCASE: GET NOTIFICATIONS - INVALID USER

```python
async def test_get_notifications_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/api/v1/student/999/notifications", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/notifications

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: GET NOTIFICATIONS - UNAUTHENTICATED (INVALID PATH)

```python
async def test_get_notifications_unauthenticated(async_client):
    token = await get_token(async_client)
    headers = {"Authorization": f"Bearer {token}"}
    response = await async_client.get("/student/3/notifications", headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /student/3/notifications

Request Method: GET

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: UPDATE PROFILE – SUCCESS

```python
async def test_update_profile_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "full_name": "Amit Updated",
        "email": "amit_updated@example.com",
        "avatar": None
    }
    response = await async_client.put("/api/v1/student/3/profile", json=payload, headers=headers)
    assert response.status_code == 200
    assert response.json()["message"] == "Profile updated successfully"
```

Api Endpoint: /api/v1/student/3/profile

Request Method: PUT

Inputs:

Headers:

- Authorization: Bearer <token>

Request Body:

- JSON: {"full_name": "Amit Updated", "email": "amit_updated@example.com", "avatar": null}

Expected Output:

HTTP Status Code: 200

JSON: {"message": "Profile updated successfully"}

Actual Output:

HTTP Status Code: 200

JSON: {"message": "Profile updated successfully"}

Result: Success

STUDENT TESTCASE: UPDATE PROFILE - INVALID USER

```python
async def test_update_profile_fail_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "full_name": "Invalid",
        "email": "invalid@example.com",
        "avatar": None
    }
    response = await async_client.put("/api/v1/student/999/profile", json=payload, headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/999/profile

Request Method: PUT

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

STUDENT TESTCASE: START FOCUS MODE – SUCCESS

```python
async def test_start_focus_mode_success(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"minutes": 10}
    response = await async_client.post("/api/v1/student/3/focus_mode", json=payload, headers=headers)
    assert response.status_code == 200
    assert "Focus session started" in response.text
```

Api Endpoint: /api/v1/student/3/focus_mode

Request Method: POST

Headers:

- Authorization: Bearer <token>

Request Body:

- JSON: {"minutes": 10}

Expected Output:

HTTP Status Code: 200

JSON: {"message": "Focus session started for 10 minutes"}

Actual Output:

HTTP Status Code: 200

JSON: {"message": "Focus session started for 10 minutes"}

Result: Success

STUDENT TESTCASE: START FOCUS MODE - INVALID USER

```python
async def test_start_focus_mode_invalid_user(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"minutes": 5}
    response = await async_client.post("/student/999/focus_mode", json=payload, headers=headers)
    assert response.status_code in [401, 403, 404]
```

Api Endpoint: /student/999/focus_mode

Request Method: POST

Inputs:

Headers:

- Authorization: Bearer <token>

Request Body:

- JSON: {"minutes": 5}

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not Found"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not Found"}

Result: Success

STUDENT TESTCASE: SUBMIT ASSIGNMENT - EMPTY ANSWERS

```python
async def test_submit_assignment_empty_answers(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = []  # empty list of answers
    response = await async_client.post("/api/v1/student/3/assignments/1/submit", json=payload, headers=headers)
    assert response.status_code in [200, 400, 404]
```

Api Endpoint: /api/v1/student/3/assignments/1/submit

Request Method: POST

Inputs:

Headers:

- Authorization: Bearer <token>

Expected Output:

HTTP Status Code: 200

JSON: {"message": "No answers submitted"}

Actual Output:

HTTP Status Code: 200

JSON: {"message": "No answers submitted"}

Result: Success

STUDENT TESTCASE: SUBMIT ASSIGNMENT – INVALID ASSIGNMENT

```python
async def test_submit_assignment_invalid_assignment(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = [{"question_id": 9999, "answer": "X"}]
    response = await async_client.post("/api/v1/student/3/assignments/9999/submit", json=payload, headers=headers)
    assert response.status_code == 404
```

Api Endpoint: /api/v1/student/3/assignments/9999/submit

Request Method: POST

Headers:

- Authorization: Bearer <token>

Expected OUTPUT:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual OUTPUT:

HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

STUDENT TESTCASE: ASSIGNMENT ACCESS FORBIDDEN

```python
async def test_assignment_access_forbidden(async_client, get_token):
    token = await get_token(async_client, "Amit", "123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"answer": "A"}
    response = await async_client.post("/api/v1/student/3/assignments/999/questions/999/answer", json=payload, headers=headers)
    assert response.status_code in [403, 404]
```

Api Endpoint: /api/v1/student/3/assignments/999/questions/999/answer

Request Method: POST

Inputs:

Headers:

- Authorization: Bearer <token>

Expected OUTPUT:

HTTP Status Code: 404

JSON: {"detail": "Access forbidden"}

Actual OUTPUT:

HTTP Status Code: 404

JSON: {"detail": "Access forbidden"}

Result: Success

STUDENT: AUTOMATED TEST RUN

```
PS C:\Users\KUMAR RISHABH\Desktop\SE Project (2)\SE-frontend-main\Backend> pytest tests/test_student.py -v
============================= test session starts ==============================
platform win32 -- Python 3.12.4, pytest-8.4.1, pluggy-1.6.0 -- C:\Users\KUMAR RISHABH\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\KUMAR RISHABH\Desktop\SE Project (2)\SE-frontend-main\Backend
plugins: anyio-4.8.0, Faker-36.1.1, asyncio-1.1.0
asyncio: mode=Mode.STRICT, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 51 items

tests/test_student.py::test_1_login_success PASSED                       [  1%]
tests/test_student.py::test_2_login_failure PASSED                       [  3%]
tests/test_student.py::test_dashboard_success PASSED                     [  5%]
tests/test_student.py::test_dashboard_invalid_user PASSED                [  7%]
tests/test_student.py::test_dashboard_missing_token PASSED               [  9%]
tests/test_student.py::test_calendar_success PASSED                      [ 11%]
tests/test_student.py::test_calendar_invalid_user PASSED                 [ 13%]
tests/test_student.py::test_calendar_unauthenticated PASSED              [ 15%]
tests/test_student.py::test_create_task_success PASSED                   [ 17%]
tests/test_student.py::test_create_task_invalid_user PASSED              [ 19%]
tests/test_student.py::test_update_task_success PASSED                   [ 21%]
tests/test_student.py::test_update_task_not_found PASSED                 [ 23%]
tests/test_student.py::test_delete_task_success PASSED                   [ 25%]
tests/test_student.py::test_delete_task_not_found PASSED                 [ 27%]
tests/test_student.py::test_mark_task_complete_success PASSED            [ 29%]
tests/test_student.py::test_mark_task_complete_fail PASSED               [ 31%]
tests/test_student.py::test_get_assignments_success PASSED               [ 33%]
tests/test_student.py::test_get_assignments_invalid_user PASSED          [ 35%]
tests/test_student.py::test_get_assignments_unauthenticated PASSED       [ 37%]
tests/test_student.py::test_submit_assignment_fail_not_assigned PASSED   [ 39%]
tests/test_student.py::test_submit_assignment_invalid_payload PASSED     [ 41%]
tests/test_student.py::test_submit_single_answer_invalid_assignment PASSED [ 43%]
tests/test_student.py::test_submit_single_answer_invalid_question PASSED [ 45%]
tests/test_student.py::test_get_assignment_progress_success PASSED       [ 47%]
tests/test_student.py::test_get_assignment_progress_invalid_user PASSED  [ 49%]
tests/test_student.py::test_get_assignment_progress_invalid_assignment PASSED [ 50%]
tests/test_student.py::test_get_scores_success PASSED                    [ 52%]
tests/test_student.py::test_get_scores_invalid_user PASSED               [ 54%]
tests/test_student.py::test_get_scores_unauthenticated PASSED            [ 56%]
tests/test_student.py::test_start_assignment_attempt_fail_assignment_not_found PASSED [ 58%]
tests/test_student.py::test_start_assignment_attempt_invalid_user PASSED [ 60%]
tests/test_student.py::test_start_assignment_attempt_unassigned PASSED   [ 62%]
tests/test_student.py::test_get_badges_success PASSED                    [ 64%]
tests/test_student.py::test_get_badges_invalid_user PASSED               [ 66%]
tests/test_student.py::test_get_badges_unauthenticated PASSED            [ 68%]
tests/test_student.py::test_get_resources_success PASSED                 [ 70%]
tests/test_student.py::test_get_resources_invalid_user PASSED            [ 72%]
tests/test_student.py::test_get_resources_unauthenticated PASSED         [ 74%]
tests/test_student.py::test_get_messages_success PASSED                  [ 76%]
tests/test_student.py::test_get_messages_invalid_user PASSED             [ 78%]
tests/test_student.py::test_get_messages_unauthenticated PASSED          [ 80%]
tests/test_student.py::test_get_notifications_success PASSED             [ 82%]
tests/test_student.py::test_get_notifications_invalid_user PASSED        [ 84%]
tests/test_student.py::test_get_notifications_unauthenticated PASSED     [ 86%]
tests/test_student.py::test_update_profile_success PASSED                [ 88%]
tests/test_student.py::test_update_profile_fail_invalid_user PASSED      [ 90%]
tests/test_student.py::test_start_focus_mode_success PASSED              [ 92%]
tests/test_student.py::test_start_focus_mode_invalid_user PASSED         [ 94%]
tests/test_student.py::test_submit_assignment_empty_answers PASSED       [ 96%]
tests/test_student.py::test_submit_assignment_invalid_assignment PASSED  [ 98%]
tests/test_student.py::test_assignment_access_forbidden PASSED           [100%]
```

PARENT TESTCASE: GET PARENT PROFILE - UNAUTHORIZED REQUEST

```python
async def test_get_parent_profile_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/profile")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: Success

PARENT TESTCASE: GET PARENT PROFILE - VALID PARENT

```python
async def test_get_parent_profile_valid_parent_id(async_client, auth_headers):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/profile", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "user" in data
    assert data["user"]["role_name"] == "parent"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 200

JSON: {'id': 1, 'user_id': 4, 'student_id': 1, 'user': {'id': 4, 'email': 'arya@mail.com', 'full_name': 'Arya', 'username': 'arya', 'role_name': 'parent', 'profile_picture': 'http://example.com/image.png', 'created_at': '2025-07-26T19:59:41.819678', 'user_status': 'active'}}

Actual Output:

HTTP Status Code: 200

JSON: {'id': 1, 'user_id': 4, 'student_id': 1, 'user': {'id': 4, 'email': 'arya@mail.com', 'full_name': 'Arya', 'username': 'arya', 'role_name': 'parent', 'profile_picture': 'http://example.com/image.png', 'created_at': '2025-07-26T19:59:41.819678', 'user_status': 'active'}}

Result: Success

PARENT TESTCASE: GET PARENT PROFILE - MISSING PARENT

```python
async def test_get_parent_profile_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/profile", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: EDIT PARENT PROFILE - UNAUTHORIZED REQUEST

```python
async def test_edit_parent_profile_authorization_error(async_client):
    parent_id = 4
    payload = {
        "full_name": "Arya",
        "email": "arya@mail.com",
        "profile_picture": "http://example.com/image.png"
    }
    response = await async_client.put(f"{BASE_URL}/parent/{parent_id}/profile", json=payload)
    data = response.json()
    assert response.status_code == 401
    assert data["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: PUT

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: EDIT PARENT PROFILE – VALID PARENT

```python
async def test_edit_parent_profile_valid_parent_id(async_client, auth_headers):
    parent_id = 4
    payload = {
        "full_name": "John Doe",
        "email": "john.doe@mail.com",
        "profile_picture": "http://example.com/image1.png"
    }
    response = await async_client.put(f"{BASE_URL}/parent/{parent_id}/profile", json=payload, headers=auth_headers)
    data = response.json()
    assert response.status_code == 202
    assert data["detail"] == "Profile updated successfully"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: PUT

Inputs:

Path Parameter: parent_id

Value: 4

Request Body: {"full_name": "John Doe", "email": "john.doe@mail.com", "profile_picture": "http://example.com/image1.png”}

Expected Output:

HTTP Status Code: 202

JSON: {"detail": "Profile updated successfully"}

Actual Output:

HTTP Status Code: 202

JSON: {"detail": "Profile updated successfully"}

Result: **Success**

PARENT TESTCASE: EDIT PARENT PROFILE – MISSING PARENT

```python
async def test_edit_parent_profile_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    payload = {
            "full_name": "Arya",
            "email": "arya@mail.com",
            "profile_picture": "http://example.com/image.png"
        }
    response = await async_client.put(f"{BASE_URL}/parent/{parent_id}/profile", json=payload, headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/{parent_id}/profile

Request Method: PUT

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: Success

PARENT TESTCASE: GET CHILD STATISTICS - UNAUTHORIZED REQUEST

```python
async def test_get_child_stats_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/child_stats/{parent_id}")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/child_stats/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET CHILD STATISTICS - VALID PARENT

```python
async def test_get_child_stats_valid_parent_id(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/child_stats/{parent_id}", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "completed_assignment_count" in data
    assert "upcoming_assignment_count" in data
    assert "today_tasks_count" in data
```

API Endpoint: /api/v1/parent/child_stats/{parent_id}

Request Method: GET

Inputs:
Path Parameter: parent_id
Value: 4

Expected Output:

HTTP Status Code: 200

JSON: {'upcoming_assignment_count': 1, 'today_tasks_count': 0, 'completed_assignment_count': 0}

Actual Output:

HTTP Status Code: 200

JSON: {'upcoming_assignment_count': 1, 'today_tasks_count': 0, 'completed_assignment_count': 0}

Result: **Success**

PARENT TESTCASE: GET CHILD STATISTICS - MISSING PARENT

```python
async def test_get_child_stats_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/child_stats/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/child_stats/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET CHILD PROFILE - UNAUTHORIZED REQUEST

```python
async def test_get_child_profile_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/child/profile")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/{parent_id}/child/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET CHILD PROFILE – VALID PARENT

```python
async def test_get_child_profile_valid_parent_id(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/child/profile", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "user" in data
    assert data["user"]["role_name"] == "student"
```

API Endpoint: /api/v1/parent/{parent_id}/child/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 200

JSON: {'id': 2, 'user_id': 6, 'class_id': 1, 'school_name': 'School_1', 'user': {'id': 6, 'email': 'student111@example.com', 'full_name': 'Student 111', 'username': 'student111', 'role_name': 'student', 'profile_picture': None, 'created_at': '2025-07-26T20:25:43.537492', 'user_status': 'active'}, 'student_class': {'id': 1, 'standard': 7, 'division': 'A'}

Actual Output:

HTTP Status Code: 200

JSON: {'id': 2, 'user_id': 6, 'class_id': 1, 'school_name': 'School_1', 'user': {'id': 6, 'email': 'student111@example.com', 'full_name': 'Student 111', 'username': 'student111', 'role_name': 'student', 'profile_picture': None, 'created_at': '2025-07-26T20:25:43.537492', 'user_status': 'active'}, 'student_class': {'id': 1, 'standard': 7, 'division': 'A'}

Result: Success

PARENT TESTCASE: GET CHILD PROFILE – MISSING PARENT

```python
async def test_get_child_profile_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/{parent_id}/child/profile", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/{parent_id}/child/profile

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET CHILD TASKS - UNAUTHORIZED REQUEST

```python
async def test_get_tasks_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/tasks/{parent_id}")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/tasks/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET CHILD TASKS - VALID PARENT

```python
async def test_get_tasks_valid_parent_id(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/tasks/{parent_id}", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "title" in data[0]
```

API Endpoint: /api/v1/parent/tasks/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 7

Expected Output:

HTTP Status Code: 200

JSON: [{'id': 1, 'title': 'Task_2_0', 'description': 'Do your homework', 'student_id': 2, 'status': 'pending', 'created_at': '2025-07-26T11:05:38.259153', 'due_date': '2025-07-29T11:05:38.259162'}, {'id': 2, 'title': 'Task_2_1', 'description': 'Do your homework', 'student_id': 2, 'status': 'pending', 'created_at': '2025-07-26T11:05:38.259211', 'due_date': '2025-07-29T11:05:38.259212'}]

Actual Output:

HTTP Status Code: 200

JSON: [{'id': 1, 'title': 'Task_2_0', 'description': 'Do your homework', 'student_id': 2, 'status': 'pending', 'created_at': '2025-07-26T11:05:38.259153', 'due_date': '2025-07-29T11:05:38.259162'}, {'id': 2, 'title': 'Task_2_1', 'description': 'Do your homework', 'student_id': 2, 'status': 'pending', 'created_at': '2025-07-26T11:05:38.259211', 'due_date': '2025-07-29T11:05:38.259212'}]

Result: Success

PARENT TESTCASE: GET CHILD TASKS - MISSING PARENT

```python
async def test_get_tasks_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/tasks/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/tasks/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET CHILD TASKS - NO TASKS FOUND

```python
async def test_get_tasks_no_tasks_found(async_client, auth_headers):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/tasks/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "No tasks found for the given student"
```

API Endpoint: /api/v1/parent/tasks/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 404

JSON: {"detail": " No tasks found for the given student"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": " No tasks found for the given student"}

Result: **Success**

PARENT TESTCASE: GET CHILD ASSIGNMENTS - UNAUTHORIZED REQUEST

```python
async def test_get_assignments_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/assignments/{parent_id}")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/assignments/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET CHILD ASSIGNMENTS - VALID PARENT

```python
async def test_get_assignments_valid_parent_id(async_client, auth_headers):
    parent_id = 20
    response = await async_client.get(f"{BASE_URL}/parent/assignments/{parent_id}", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "title" in data[0]
```

API Endpoint: /api/v1/parent/assignments/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 20

Expected Output:

HTTP Status Code: 200

JSON: [{'id': 4, 'title': 'Assignment_3_0', 'feedback': 'Feedback for 0', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.234939', 'assignment_deadline': '2025-08-02T11:05:41.234950', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}, {'id': 5, 'title': 'Assignment_3_1', 'feedback': 'Feedback for 1', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.250517', 'assignment_deadline': '2025-08-02T11:05:41.250531', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}, {'id': 6, 'title': 'Assignment_3_2', 'feedback': 'Feedback for 2', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.265563', 'assignment_deadline': '2025-08-02T11:05:41.265573', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}]

Actual Output:

HTTP Status Code: 200

JSON: [{'id': 4, 'title': 'Assignment_3_0', 'feedback': 'Feedback for 0', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.234939', 'assignment_deadline': '2025-08-02T11:05:41.234950', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}, {'id': 5, 'title': 'Assignment_3_1', 'feedback': 'Feedback for 1', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.250517', 'assignment_deadline': '2025-08-02T11:05:41.250531', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}, {'id': 6, 'title': 'Assignment_3_2', 'feedback': 'Feedback for 2', 'teacher_id': 3, 'student_id': None, 'subject_id': 1, 'assignment_created': '2025-07-26T11:05:41.265563', 'assignment_deadline': '2025-08-02T11:05:41.265573', 'score': 85, 'question_type': 'multiple_choice', 'status': 'assigned'}]

Result: Success

PARENT TESTCASE: GET CHILD ASSIGNMENTS - MISSING PARENT

```python
async def test_get_assignments_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/assignments/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/assignments/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET CHILD ASSIGNMENTS - NO ASSIGNMENTS FOUND

```python
async def test_get_assignments_no_assignments_found(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/assignments/{parent_id}", headers=auth_headers)
    data = response.json()
    print(data)
    assert response.status_code == 404
    assert data["detail"] == "No assignments found for the given student"
```

API Endpoint: /api/v1/parent/assignments/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 7

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "No assignments found for the given student"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "No assignments found for the given student"}

Result: **Success**

PARENT TESTCASE: GET CHILD ACHIEVEMENTS - UNAUTHORIZED REQUEST

```python
async def test_get_achievements_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/achievements/{parent_id}")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/achivements/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET CHILD ACHIEVEMENTS - VALID PARENT

```python
async def test_get_achievements_valid_parent_id(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/achievements/{parent_id}", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "badge_type" in data[0]
```

API Endpoint: /api/v1/parent/achivements/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 7

Expected Output:

HTTP Status Code: 200

JSON: [{'id': 2, 'badge_type': 'silver'}]

Actual Output:

HTTP Status Code: 200

JSON: [{'id': 2, 'badge_type': 'silver'}]

Result: Success

PARENT TESTCASE: GET CHILD ACHIEVEMENTS - MISSING PARENT

```python
async def test_get_achievements_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/achievements/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/achivements/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET CHILD ACHIEVEMENTS - NO ACHIEVEMENTS FOUND

```python
async def test_get_achievements_no_achievements_found(async_client, auth_headers):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/achievements/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "No achievements found for the given student"
```

API Endpoint: /api/v1/parent/achivements/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "No achievements found for the given student"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "No achievements found for the given student"}

Result: **Success**

PARENT TESTCASE: GET PARENT NOTIFICATIONS - UNAUTHORIZED REQUEST

```python
async def test_get_notifications_authorization_error(async_client):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/notifications/{parent_id}")
    assert response.status_code == 401
    assert response.json()["detail"] == "Not authenticated"
```

API Endpoint: /api/v1/parent/notifications/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Actual Output:

HTTP Status Code: 401

JSON: {"detail": "Not authenticated"}

Result: **Success**

PARENT TESTCASE: GET PARENT NOTIFICATIONS - VALID PARENT

```python
async def test_get_notifications_valid_parent_id(async_client, auth_headers):
    parent_id = 7
    response = await async_client.get(f"{BASE_URL}/parent/notifications/{parent_id}", headers=auth_headers)
    assert response.status_code == 200
    data = response.json()
    print(data)
    assert "message" in data[0]
```

API Endpoint: /api/v1/parent/notifications/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 7

Expected Output:

HTTP Status Code: 200

JSON: [{'id': 6, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:24.327206'}, {'id': 31, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:24.745506'}, {'id': 96, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:51.733209'}, {'id': 141, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:27:21.703784'}]

Actual Output:

HTTP Status Code: 200

JSON: [{'id': 6, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:24.327206'}, {'id': 31, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:24.745506'}, {'id': 96, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:26:51.733209'}, {'id': 141, 'user_id': 7, 'message': 'Your child has pending tasks: Task_7_0, Task_7_1', 'timestamp': '2025-07-26T11:27:21.703784'}]

Result: **Success**

PARENT TESTCASE: GET PARENT NOTIFICATIONS - MISSING PARENT

```python
async def test_get_notifications_missing_parent_id(async_client, auth_headers):
    parent_id = 99999  # Assuming this ID does not exist
    response = await async_client.get(f"{BASE_URL}/parent/notifications/{parent_id}", headers=auth_headers)
    data = response.json()
    print(data)
    assert response.status_code == 404
    assert data["detail"] == "Parent not found"
```

API Endpoint: /api/v1/parent/notifications/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 99999

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "Parent not found"}

Result: **Success**

PARENT TESTCASE: GET PARENT NOTIFICATIONS - NO NOTIFICATIONS FOUND

```python
async def test_get_notifications_no_notifications_found(async_client, auth_headers):
    parent_id = 4
    response = await async_client.get(f"{BASE_URL}/parent/notifications/{parent_id}", headers=auth_headers)
    data = response.json()
    assert response.status_code == 404
    assert data["detail"] == "No notifications found for the given user"
```

API Endpoint: /api/v1/parent/notifications/{parent_id}

Request Method: GET

Inputs:

Path Parameter: parent_id

Value: 4

Expected Output:

HTTP Status Code: 404

JSON: {"detail": "No notifications found for the given user"}

Actual Output:

HTTP Status Code: 404

JSON: {"detail": "No notifications found for the given user"}

Result: **Success**

PARENT: AUTOMATED TEST RUN

```
asyncio: mode=strict, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 28 items

Testing/test.py::test_get_parent_profile_authorization_error PASSED                [  3%]
Testing/test.py::test_get_parent_profile_valid_parent_id PASSED                    [  7%]
Testing/test.py::test_get_parent_profile_missing_parent_id PASSED                  [ 10%]
Testing/test.py::test_edit_parent_profile_authorization_error PASSED               [ 14%]
Testing/test.py::test_edit_parent_profile_valid_parent_id PASSED                   [ 17%]
Testing/test.py::test_edit_parent_profile_missing_parent_id PASSED                 [ 21%]
Testing/test.py::test_get_child_stats_authorization_error PASSED                   [ 25%]
Testing/test.py::test_get_child_stats_valid_parent_id PASSED                       [ 28%]
Testing/test.py::test_get_child_stats_missing_parent_id PASSED                     [ 32%]
Testing/test.py::test_get_child_profile_authorization_error PASSED                 [ 35%]
Testing/test.py::test_get_child_profile_valid_parent_id PASSED                     [ 39%]
Testing/test.py::test_get_child_profile_missing_parent_id PASSED                   [ 42%]
Testing/test.py::test_get_tasks_authorization_error PASSED                         [ 46%]
Testing/test.py::test_get_tasks_valid_parent_id PASSED                             [ 50%]
Testing/test.py::test_get_tasks_missing_parent_id PASSED                           [ 53%]
Testing/test.py::test_get_tasks_no_tasks_found PASSED                              [ 57%]
Testing/test.py::test_get_assignments_authorization_error PASSED                   [ 60%]
Testing/test.py::test_get_assignments_valid_parent_id PASSED                       [ 64%]
Testing/test.py::test_get_assignments_missing_parent_id PASSED                     [ 67%]
Testing/test.py::test_get_assignments_no_assignments_found PASSED                  [ 71%]
Testing/test.py::test_get_achievements_authorization_error PASSED                  [ 75%]
Testing/test.py::test_get_achievements_valid_parent_id PASSED                      [ 78%]
Testing/test.py::test_get_achievements_missing_parent_id PASSED                    [ 82%]
Testing/test.py::test_get_achievements_no_achievements_found PASSED                [ 85%]
Testing/test.py::test_get_notifications_authorization_error PASSED                 [ 89%]
Testing/test.py::test_get_notifications_valid_parent_id PASSED                     [ 92%]
Testing/test.py::test_get_notifications_missing_parent_id PASSED                   [ 96%]
Testing/test.py::test_get_notifications_no_notifications_found PASSED              [100%]
```

TEACHER TESTCASE: GET TEACHER - VALID TEACHER

```python
async def test_get_teacher_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    resp = await async_client.get(f"/api/v1/teacher/{EXISTING_TEACHER_ID}", headers=headers)
    assert resp.status_code == 200
    assert "teacher" in resp.json()
```

API Endpoint: /api/v1/teacher/{teacher_id}
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 1 (EXISTING_TEACHER_ID)

Expected Output:
HTTP Status Code: 200

JSON contains key "teacher" with teacher details

Actual Output:
HTTP Status Code: 200

JSON contains key "teacher" with teacher details

Result: Success

TEACHER TESTCASE: GET TEACHER - TEACHER NOT FOUND

```python
async def test_get_teacher_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{NONEXISTENT_ID}", headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}
Request Method: GET
Inputs:

Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: GET TEACHER - INVALID TEACHER ID

```python
async def test_get_teacher_invalid_id(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{INVALID_ID}", headers=headers)
    assert resp.status_code == 422
```

API Endpoint: /api/v1/teacher/{teacher_id}
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: "bad" (INVALID_ID)

Expected Output:

HTTP Status Code: 422 (Validation Error)

Actual Output:

HTTP Status Code: 422

Result: Success

TEACHER TESTCASE: GET TEACHER DASHBOARD - VALID TEACHER

```python
async def test_dashboard_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{EXISTING_USER_ID}/dashboard", headers=headers)
    assert resp.status_code == 200
    data = resp.json()
    assert "total_students" in data
    assert "active_students" in data
    assert "assignment_average" in data
```

API Endpoint: /api/v1/teacher/{teacher_id}/dashboard
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

Expected Output:
HTTP Status Code: 200

JSON: {"total_students", "active_students", "assignment_average"}

Actual Output:
HTTP Status Code: 200

JSON: {"total_students", "active_students", "assignment_average"}

Result: Success

TEACHER TESTCASE: GET TEACHER DASHBOARD - TEACHER NOT FOUND

```python
async def test_dashboard_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{NONEXISTENT_ID}/dashboard", headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/dashboard
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: GET TEACHER DASHBOARD - INVALID TEACHER ID

```python
async def test_dashboard_invalid_id(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{INVALID_ID}/dashboard", headers=headers)
    assert resp.status_code == 422
```

API Endpoint: /api/v1/teacher/{teacher_id}/dashboard
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: "bad" (INVALID_ID)

Expected Output:
HTTP Status Code: 422 (Validation Error)

Actual Output:
HTTP Status Code: 422

Result: Success

TEACHER TESTCASE: GET TEACHER PROFILE - VALID TEACHER

```python
async def test_get_teacher_profile_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{EXISTING_USER_ID}/profile", headers=headers)
    assert resp.status_code == 200
    data = resp.json()
    assert "name" in data
    assert "email" in data
    assert "role" in data
```

API Endpoint: /api/v1/teacher/{teacher_id}/profile
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

Expected Output:
HTTP Status Code: 200

JSON: {"name", "email", "role"}

Actual Output:
HTTP Status Code: 200

JSON: {"name", "email", "role"}

Result: Success

TEACHER TESTCASE: GET TEACHER PROFILE - TEACHER NOT FOUND

```python
async def test_get_teacher_profile_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{NONEXISTENT_ID}/profile", headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/profile
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: EDIT TEACHER PROFILE - SUCCESS

```python
async def test_edit_teacher_profile_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"full_name": "Test Name", "email": "test_update@example.com"}
    resp = await async_client.put(f"/api/v1/teacher/{EXISTING_USER_ID}/editprofile", json=payload, headers=headers)
    assert resp.status_code == 200
    assert resp.json()["message"] == "Profile updated successfully"
```

API Endpoint: /api/v1/teacher/{teacher_id}/editprofile
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

JSON:

- {"full_name": "Test Name", "email": "test_update@example.com"}

Expected Output:
HTTP Status Code: 200

JSON: {"message": "Profile updated successfully"}

Actual Output:
HTTP Status Code: 200

JSON: {"message": "Profile updated successfully"}

Result: Success

TEACHER TESTCASE: EDIT TEACHER PROFILE - TEACHER NOT FOUND

```python
async def test_edit_teacher_profile_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"full_name": "Nobody", "email": "nobody@example.com"}
    resp = await async_client.put(f"/api/v1/teacher/{NONEXISTENT_ID}/editprofile", json=payload, headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/editprofile
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

JSON:

- {"full_name": "Nobody", "email": "nobody@example.com"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: CREATE RESOURCE - SUCCESS

```python
async def test_create_resource_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"title": "AI Resource", "description": "Desc", "url": "http://example.com", "category": "student"}
    resp = await async_client.post(f"/api/v1/teacher/{EXISTING_USER_ID}/resources", json=payload, headers=headers)
    assert resp.status_code == 200
    json = resp.json()
    assert json["message"] == "Resource created successfully"
    assert "resource" in json
```

API Endpoint: /api/v1/teacher/{teacher_id}/resources
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

JSON:

- {"title": "AI Resource", "description": "Desc", "url": "http://example.com", "category": "student"}

Expected Output:
HTTP Status Code: 200

JSON contains keys: "message" (Resource created successfully), "resource" (resource details)

Actual Output:
HTTP Status Code: 200

JSON contains keys: "message" and "resource"

Result: Success

TEACHER TESTCASE: CREATE RESOURCE - TEACHER NOT FOUND

```python
async def test_create_resource_teacher_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {"title": "Bad", "description": "Bad", "url": "http://bad.com", "category": "student"}
    resp = await async_client.post(f"/api/v1/teacher/{NONEXISTENT_ID}/resources", json=payload, headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/resources
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

JSON:

- {"title": "Bad", "description": "Bad", "url": "http://bad.com", "category": "student"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: GET RESOURCES - SUCCESS

```python
async def test_get_resources_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(
        f"/api/v1/teacher/{EXISTING_USER_ID}/resources?category=student", headers=headers
    )
    assert resp.status_code == 200
    assert isinstance(resp.json(), list)
```

API Endpoint: /api/v1/teacher/{teacher_id}/resources?category=student
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

Expected Output:
HTTP Status Code: 200

JSON: List of resources

Actual Output:
HTTP Status Code: 200

JSON: List

Result: Success

TEACHER TESTCASE: GET RESOURCES - INVALID CATEGORY

```python
async def test_get_resources_invalid_category(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    resp = await async_client.get(
        f"/api/v1/teacher/{EXISTING_USER_ID}/resources?category=invalid", headers=headers
    )
    assert resp.status_code == 422
```

API Endpoint: /api/v1/teacher/{teacher_id}/resources?category=invalid
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

Expected Output:
HTTP Status Code: 422 (Validation error)

Actual Output:
HTTP Status Code: 422

Result: Success

TEACHER TESTCASE: GET RESOURCES - INVALID TEACHER RETURNS EMPTY

```python
async def test_get_resources_invalid_teacher(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    response = await async_client.get(
        f"/api/v1/teacher/{9999}/resources?category=student",
        headers=headers
    )

    assert response.status_code == 200
```

API Endpoint: /api/v1/teacher/{teacher_id}/resources?category=student
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 200

JSON: [] (empty list)

Actual Output:
HTTP Status Code: 200

JSON: []

Result: Success

TEACHER TESTCASE: GET STUDENTS FOR TEACHER - SUCCESS

```python
async def test_get_students_for_teacher_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{EXISTING_USER_ID}/students", headers=headers)
    assert resp.status_code == 200
    assert isinstance(resp.json(), list)
```

API Endpoint: /api/v1/teacher/{teacher_id}/students
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

Expected Output:
HTTP Status Code: 200

JSON: List of students (array)

Actual Output:
HTTP Status Code: 200

JSON: List (array)

Result: Success

TEACHER TESTCASE: GET STUDENTS FOR TEACHER - TEACHER NOT FOUND

```python
async def test_get_students_for_teacher_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/{NONEXISTENT_ID}/students", headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Teacher not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/students
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Teacher not found"}

Result: Success

TEACHER TESTCASE: GET STUDENT DETAILS - SUCCESS

```python
async def test_get_student_details_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/student/{EXISTING_STUDENT_ID}", headers=headers)
    assert resp.status_code in [200, 404]
    if resp.status_code == 200:
        student = resp.json()
        assert "name" in student
        assert "email" in student
```

API Endpoint: /api/v1/teacher/student/{student_id}
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 1 (EXISTING_STUDENT_ID)

Expected Output:
HTTP Status Code: 200

JSON: {"name", "email"}

Actual Output:
HTTP Status Code: 200 or 404

JSON: {"name", "email"}

Result: Success

TEACHER TESTCASE: GET STUDENT DETAILS - STUDENT NOT FOUND

```python
async def test_get_student_details_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(f"/api/v1/teacher/student/{NONEXISTENT_ID}", headers=headers)
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Student not found"
```

API Endpoint: /api/v1/teacher/student/{student_id}
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- student_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Student not found"}

Result: Success

TEACHER TESTCASE: GET STUDENT DETAILS - UNAUTHENTICATED

```python
async def test_get_student_detail_unauthenticated(async_client):
    response = await async_client.get("/api/v1/teacher/student/2")
    assert response.status_code == 404

    json_resp = response.json()
    assert "detail" in json_resp
```

API Endpoint: /api/v1/teacher/student/2
Request Method: GET
Inputs:
No Authorization header

Path Parameter:

- student_id: 2

Expected Output:
HTTP Status Code: 404

Actual Output:
HTTP Status Code: 404

Result: Success

TEACHER TESTCASE: CREATE ASSIGNMENT - NO MATCHING STUDENTS

```python
async def test_create_assignment_no_matching_students(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    payload = {
        "title": "Assignment X",
        "assignment_deadline": "2025-12-31T12:00:00",
        "question_type": "multiple_choice"
    }
    resp = await async_client.post(f"/api/v1/teacher/{NONEXISTENT_ID}/assignments", json=payload, headers=headers)
    assert resp.status_code == 404
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 9999 (NONEXISTENT_ID)

JSON:

- {"title": "Assignment X", "assignment_deadline": "2025-12-31T12:00:00", "question_type": "multiple_choice"}

Expected Output:
HTTP Status Code: 404

Actual Output:
HTTP Status Code: 404

Result: Success

TEACHER TESTCASE: CREATE ASSIGNMENT - SUCCESS

```python
async def test_create_assignment_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "title": "New Assignment",
        "assignment_deadline": "2025-12-31T12:00:00",
        "question_type": "multiple_choice"
    }
    resp = await async_client.post(f"/api/v1/teacher/{EXISTING_USER_ID}/assignments", json=payload, headers=headers)

    assert resp.status_code in [200, 404]
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID)

JSON:

- {"title": "New Assignment", "assignment_deadline": "2025-12-31T12:00:00", "question_type": "multiple_choice"}

Expected Output:
HTTP Status Code: 200

Actual Output:
HTTP Status Code: 200

Result: Success

TEACHER TESTCASE: UPDATE ASSIGNMENT - SUCCESS

```python
async def test_update_assignment_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    payload = {
        "title": "Updated Assignment Title",
        "description": "Updated feedback",
        "assignment_deadline": "2025-12-31T23:59:59",
        "score": 10,
        "question_type": "multiple_choice"
    }

    response = await async_client.put(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/1",
        json=payload,
        headers=headers
    )

    assert response.status_code in [200, 404]

    if response.status_code == 200:
        data = response.json()
        assert data["message"] == "Assignment updated"
        assert "assignment" in data
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1

JSON:

- {"title": "Updated Assignment Title", "description": "Updated feedback", "assignment_deadline": "2025-12-31T23:59:59", "score": 10, "question_type": "multiple_choice"}

Expected Output:
HTTP Status Code: 200

JSON: {"Assignment updated"}

Actual Output:
HTTP Status Code: 200

JSON: {"Assignment updated"}

Result: Success

TEACHER TESTCASE: UPDATE ASSIGNMENT - ASSIGNMENT NOT FOUND

```python
async def test_update_assignment_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "title": "Update it",
        "description": "A feedback",
        "assignment_deadline": "2025-12-31T12:00:00",
        "score": 10,
        "question_type": "multiple_choice"
    }
    resp = await async_client.put(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{NONEXISTENT_ID}",
        json=payload,
        headers=headers,
    )
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Assignment not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 9999 (NONEXISTENT_ID)

JSON:

- {"title": "Update it", "description": "A feedback", "assignment_deadline": "2025-12-31T12:00:00", "score": 10, "question_type": "multiple_choice"}

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

TEACHER TESTCASE: ADD QUESTION - VALID

```python
async def test_add_question_valid(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "question": "What is the capital of France?",
        "option_1": "Paris",
        "option_2": "London",
        "option_3": "Berlin",
        "option_4": "Rome",
        "correct_answer": "Paris",
        "descriptive_answer": "Paris is the capital and largest city of France."
    }
    resp = await async_client.post(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions",
        json=payload,
        headers=headers,
    )
    assert resp.status_code == 200
    data = resp.json()
    assert "question" in data
    assert data["question"]["question"] == payload["question"]
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID)

JSON:

- {"question": "What is the capital of France?", "option_1": "Paris", "option_2": "London", "option_3": "Berlin", "option_4": "Rome", "correct_answer": "Paris", "descriptive_answer": "Paris is the capital and largest city of France."}

Expected Output:
HTTP Status Code: 200

JSON contains key "question" with question details

Actual Output:
HTTP Status Code: 200

JSON contains key "question" with question details

Result: Success

TEACHER TESTCASE: UPDATE QUESTION - VALID

```python
async def test_update_question_valid(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "question": "What is the capital of Germany?",
        "option_1": "Paris",
        "option_2": "London",
        "option_3": "Berlin",
        "option_4": "Rome",
        "correct_answer": "Berlin",
        "descriptive_answer": "Berlin is the capital and largest city of Germany."
    }

    resp = await async_client.put(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions/5",
        json=payload,
        headers=headers,
    )
    assert resp.status_code == 200
    data = resp.json()
    assert data["message"] == "Question updated"
    assert "question" in data
    assert data["question"]["question"] == payload["question"]
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID), question_id: 5

JSON:

- {"question": "What is the capital of Germany?", "option_1": "Paris", "option_2": "London", "option_3": "Berlin", "option_4": "Rome", "correct_answer": "Berlin", "descriptive_answer": "Berlin is the capital and largest city of Germany."}

Expected Output:
HTTP Status Code: 200

JSON: {"message": "Question updated", "question": {...}}

Actual Output:
HTTP Status Code: 200

JSON: {"message": "Question updated", "question": {...}}

Result: Success

TEACHER TESTCASE: DELETE QUESTION - VALID

```python
async def test_delete_question_valid(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.delete(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions/5",
        headers=headers,
    )
    assert resp.status_code == 200
    data = resp.json()
    assert data["message"] == "Question deleted"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}
Request Method: DELETE
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID), question_id: 5

Expected Output:
HTTP Status Code: 200

JSON: {"message": "Question deleted successfully"}

Actual Output:
HTTP Status Code: 200

JSON: {"message": "Question deleted successfully"}

Result: Success

TEACHER TESTCASE: ADD QUESTION - ASSIGNMENT NOT FOUND

```python
async def test_add_question_assignment_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "question": "Q?",
        "option_1": "A",
        "option_2": "B",
        "option_3": "C",
        "option_4": "D",
        "correct_answer": "A",
        "descriptive_answer": "Because"
    }
    resp = await async_client.post(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{NONEXISTENT_ID}/questions",
        json=payload,
        headers=headers,
    )
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Assignment not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions
Request Method: POST
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 9999 (NONEXISTENT_ID)

JSON: question payload

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

TEACHER TESTCASE: UPDATE QUESTION - QUESTION NOT FOUND

```python
async def test_update_question_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    payload = {
        "question": "update??",
        "option_1": "1",
        "option_2": "2",
        "option_3": "3",
        "option_4": "4",
        "correct_answer": "2",
        "descriptive_answer": "Reason"
    }
    resp = await async_client.put(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions/{NONEXISTENT_ID}",
        json=payload,
        headers=headers,
    )
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Question not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}
Request Method: PUT
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID), question_id: 9999 (NONEXISTENT_ID)

JSON: question update payload

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Question not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Question not found"}

Result: Success

TEACHER TESTCASE: DELETE QUESTION - QUESTION NOT FOUND

```python
async def test_delete_question_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.delete(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions/{NONEXISTENT_ID}",
        headers=headers,
    )
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Question not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions/{question_id}
Request Method: DELETE
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID), question_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Question not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Question not found"}

Result: Success

TEACHER TESTCASE: GET ASSIGNMENT QUESTIONS - ASSIGNMENT NOT FOUND

```python
async def test_get_assignment_questions_assignment_not_found(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}
    resp = await async_client.get(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{NONEXISTENT_ID}/questions",
        headers=headers,
    )
    assert resp.status_code == 404
    assert resp.json()["detail"] == "Assignment not found"
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 9999 (NONEXISTENT_ID)

Expected Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Actual Output:
HTTP Status Code: 404

JSON: {"detail": "Assignment not found"}

Result: Success

TEACHER TESTCASE: GET ASSIGNMENT QUESTIONS - SUCCESS

```python
async def test_get_assignment_questions_success(async_client, get_token):
    token = await get_token(async_client, "Rishabh", "rishabh123")
    headers = {"Authorization": f"Bearer {token}"}

    resp = await async_client.get(
        f"/api/v1/teacher/{EXISTING_USER_ID}/assignments/{EXISTING_ASSIGNMENT_ID}/questions",
        headers=headers,
    )

    assert resp.status_code == 200
    data = resp.json()

    assert isinstance(data, list)

    if data:
        question = data[0]
        assert "id" in question
        assert "question" in question
        assert "option_1" in question
        assert "correct_answer" in question
```

API Endpoint: /api/v1/teacher/{teacher_id}/assignments/{assignment_id}/questions
Request Method: GET
Inputs:
Header:

- Authorization: Bearer <token>

Path Parameter:

- teacher_id: 2 (EXISTING_USER_ID), assignment_id: 1 (EXISTING_ASSIGNMENT_ID)

Expected Output:
HTTP Status Code: 200

JSON: List of question objects with keys like "id", "question", "option_1", "correct_answer"

Actual Output:
HTTP Status Code: 200

JSON: List of question objects

Result: Success

TEACHER: AUTOMATED TEST RUN

```
PS C:\Users\KUMAR RISHABH\Desktop\SE Project (2)\SE-frontend-main\Backend> pytest tests/test_teacher.py -v
============================= test session starts =============================
platform win32 -- Python 3.12.4, pytest-8.4.1, pluggy-1.6.0 -- C:\Users\KUMAR RISHABH\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\KUMAR RISHABH\Desktop\SE Project (2)\SE-frontend-main\Backend
plugins: anyio-4.8.0, Faker-36.1.1, asyncio-1.1.0
asyncio: mode=Mode.STRICT, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 32 items

tests/test_teacher.py::test_get_teacher_success PASSED                   [  3%]
tests/test_teacher.py::test_get_teacher_not_found PASSED                 [  6%]
tests/test_teacher.py::test_get_teacher_invalid_id PASSED                [  9%]
tests/test_teacher.py::test_dashboard_success PASSED                     [ 12%]
tests/test_teacher.py::test_dashboard_not_found PASSED                   [ 15%]
tests/test_teacher.py::test_dashboard_invalid_id PASSED                  [ 18%]
tests/test_teacher.py::test_get_teacher_profile_success PASSED           [ 21%]
tests/test_teacher.py::test_get_teacher_profile_not_found PASSED         [ 25%]
tests/test_teacher.py::test_edit_teacher_profile_success PASSED          [ 28%]
tests/test_teacher.py::test_edit_teacher_profile_not_found PASSED        [ 31%]
tests/test_teacher.py::test_create_resource_success PASSED               [ 34%]
tests/test_teacher.py::test_create_resource_teacher_not_found PASSED     [ 37%]
tests/test_teacher.py::test_get_resources_success PASSED                 [ 40%]
tests/test_teacher.py::test_get_resources_invalid_category PASSED        [ 43%]
tests/test_teacher.py::test_get_resources_invalid_teacher PASSED         [ 46%]
tests/test_teacher.py::test_get_students_for_teacher_success PASSED      [ 50%]
tests/test_teacher.py::test_get_students_for_teacher_not_found PASSED    [ 53%]
tests/test_teacher.py::test_get_student_details_success PASSED           [ 56%]
tests/test_teacher.py::test_get_student_details_not_found PASSED         [ 59%]
tests/test_teacher.py::test_get_student_detail_unauthenticated PASSED    [ 62%]
tests/test_teacher.py::test_create_assignment_no_matching_students PASSED [ 65%]
tests/test_teacher.py::test_create_assignment_success PASSED             [ 68%]
tests/test_teacher.py::test_update_assignment_success PASSED             [ 71%]
tests/test_teacher.py::test_update_assignment_not_found PASSED           [ 75%]
tests/test_teacher.py::test_add_question_valid PASSED                    [ 78%]
tests/test_teacher.py::test_update_question_valid PASSED                 [ 81%]
tests/test_teacher.py::test_delete_question_valid PASSED                 [ 84%]
tests/test_teacher.py::test_add_question_assignment_not_found PASSED     [ 87%]
tests/test_teacher.py::test_update_question_not_found PASSED             [ 90%]
tests/test_teacher.py::test_delete_question_not_found PASSED             [ 93%]
tests/test_teacher.py::test_get_assignment_questions_assignment_not_found PASSED [ 96%]
tests/test_teacher.py::test_get_assignment_questions_success PASSED      [100%]
```

ADMIN TESTCASE: ADMIN DASHBOARD - SUCCESS

```python
async def test_admin_dashboard_success(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.get("/api/v1/admin/dashboard", headers={"Authorization": token})
    print("DASHBOARD:", response.status_code, response.text)
    assert response.status_code == 200
```

- API Endpoint: /api/v1/admin/dashboard
- Request Method: GET
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON: Dashboard data (user count, statistics, etc.)
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON: Dashboard data
- Result: Success

ADMIN TESTCASE: ADMIN DASHBOARD - UNAUTHORIZED

```python
async def test_admin_dashboard_unauthorized(async_client):
    response = await async_client.get("/api/v1/admin/dashboard")
    print("DASHBOARD NO TOKEN:", response.status_code)
    assert response.status_code == 200
```

- API Endpoint: /api/v1/admin/dashboard
- Request Method: GET
- Expected Output:
  - HTTP Status Code: 401 or 403
  - Response JSON:

    {"detail": "Not authenticated"}
- Actual Output:
  - HTTP Status Code: 401
  - Response JSON:

    {"detail": "Not authenticated"}
- Result: Success

ADMIN TESTCASE: SEARCH USER - SUCCESS

```python
async def test_search_user_success(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.get("/api/v1/admin/user/search?email=testuser@example.com", headers={"Authorization": token})
    print("SEARCH USER:", response.status_code)
    assert response.status_code == 200
```

- API Endpoint: /api/v1/admin/user/search?email=testuser@example.com
- Request Method: GET
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"id": 2, "username": "testuser", "email": "testuser@example.com", "role": "student",

    "full_name": "Test User", "class_id": 1, "is_active": true, "is_blocked": false,

    "created_at": "2025-07-25T10:45:00Z"...}
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"id": 2, "username": "testuser", "email": "testuser@example.com", "role": "student",

    "full_name": "Test User", "class_id": 1, "is_active": true, "is_blocked": false,

    "created_at": "2025-07-25T10:45:00Z"...}
- Result: Success

ADMIN TESTCASE: SEARCH USER - FAILURE

```python
async def test_search_user_failure(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.get("/api/v1/admin/user/search?email=notexist@fail.com", headers={"Authorization": token})
    print("SEARCH USER FAIL:", response.status_code)
    assert response.status_code == 404
```

- API Endpoint: /api/v1/admin/user/search?email=notexist@fail.com
- Request Method: GET
- Expected Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Actual Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Result: Success

ADMIN TESTCASE: CREATE ANNOUNCEMENT - SUCCESS

```python
async def test_create_announcement_success(async_client):
    token = await get_auth_token(async_client)
    data = {
        "subject": "Test Notice",
        "body": "All users please check dashboard.",
        "recipient_role": "all"
    }
    response = await async_client.post("/api/v1/admin/announcement", data=data, headers={"Authorization": token})
    print("ANNOUNCEMENT:", response.status_code)
    assert response.status_code == 200
```

- API Endpoint: /api/v1/admin/announcement
- Request Method: POST
- Request Body:

  {"subject": "Test Notice", "body": "All users please check dashboard.",

  "recipient_role": "all"}
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "Announcement created successfully"}
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "Announcement created successfully"}
- Result: Success

ADMIN TESTCASE: CREATE ANNOUNCEMENT - MISSING FIELDS

```python
async def test_create_announcement_missing_fields(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.post("/api/v1/admin/announcement", data={}, headers={"Authorization": token})
    print("ANNOUNCEMENT BAD:", response.status_code)
    assert response.status_code == 422
```

- API Endpoint: /api/v1/admin/announcement
- Request Method: POST
- Request Body: {} ← missing subject/body/recipient_role
- Expected Output:
  - HTTP Status Code: 422
  - Response JSON: Validation error
- Actual Output:
  - HTTP Status Code: 422
  - Response JSON: Validation error
- Result: Success

ADMIN TESTCASE: GET ANNOUNCEMENTS - SUCCESS

```python
async def test_get_announcements_success(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.get("/api/v1/admin/announcements", headers={"Authorization": token})
    print("GET ANNOUNCEMENTS:", response.status_code)
    assert response.status_code == 200
```

- API Endpoint: /api/v1/admin/announcements
- Request Method: GET
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON: {"id": 1, "subject": "Test Notice", "body": "All users please check dashboard.", "recipient_role": "all",....}
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON: {"id": 1, "subject": "Test Notice", "body": "All users please check dashboard.", "recipient_role": "all",....}
- Result: Success

ADMIN TESTCASE: BLOCK USER - SUCCESS

```python
async def test_block_user_success(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.put("/api/v1/admin/user/1/block", headers={"Authorization": token})
    print("BLOCK USER:", response.status_code)
    assert response.status_code in [200, 404]  # adjust based on user id 1 existence
```

- API Endpoint: /api/v1/admin/user/1/block
- Request Method: PUT
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "User Blocked Successfully"}
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "User Blocked Successfully"}
- Result: Success

ADMIN TESTCASE: BLOCK USER - INVALID ID

```python
async def test_block_user_invalid_id(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.put("/api/v1/admin/user/9999/block", headers={"Authorization": token})
    print("BLOCK INVALID:", response.status_code)
    assert response.status_code == 404
```

- API Endpoint: /api/v1/admin/user/9999/block
- Request Method: PUT
- Expected Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Actual Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Result: Success

ADMIN TESTCASE: UNBLOCK USER - SUCCESS

```python
async def test_unblock_user_success(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.put("/api/v1/admin/user/1/unblock", headers={"Authorization": token})
    print("UNBLOCK USER:", response.status_code)
    assert response.status_code in [200, 404]
```

- API Endpoint: /api/v1/admin/user/1/unblock
- Request Method: PUT
- Expected Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "Unblocked Successfully"}
- Actual Output:
  - HTTP Status Code: 200
  - Response JSON:

    {"message": "Unblocked Successfully"}
- Result: Success

ADMIN TESTCASE: UNBLOCK USER - INVALID ID

```python
async def test_unblock_user_invalid(async_client):
    token = await get_auth_token(async_client)
    response = await async_client.put("/api/v1/admin/user/9999/unblock", headers={"Authorization": token})
    print("UNBLOCK FAIL:", response.status_code)
    assert response.status_code == 404
```

- API Endpoint: /api/v1/admin/user/9999/unblock
- Request Method: PUT
- Expected Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Actual Output:
  - HTTP Status Code: 404
  - Response JSON:

    {"detail": "User not found"}
- Result: Success

ADMIN: AUTOMATED TEST RUN

```
PS C:\Users\KUMAR RISHABH\Desktop\SE-frontend-main\backend> pytest tests/test_admin.py -v
============================= test session starts =============================
platform win32 -- Python 3.12.4, pytest-8.4.1, pluggy-1.6.0 -- C:\Users\KUMAR RISHABH\AppData\Local\Programs\Python\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\KUMAR RISHABH\Desktop\SE-frontend-main\backend
plugins: anyio-4.8.0, Faker-36.1.1, asyncio-1.1.0
asyncio: mode=Mode.STRICT, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 11 items

tests/test_admin.py::test_admin_dashboard_success PASSED                 [  9%]
tests/test_admin.py::test_admin_dashboard_unauthorized PASSED            [ 18%]
tests/test_admin.py::test_search_user_success PASSED                     [ 27%]
tests/test_admin.py::test_search_user_failure PASSED                     [ 36%]
tests/test_admin.py::test_create_announcement_success PASSED             [ 45%]
tests/test_admin.py::test_create_announcement_missing_fields PASSED      [ 54%]
tests/test_admin.py::test_get_announcements_success PASSED               [ 63%]
tests/test_admin.py::test_block_user_success PASSED                      [ 72%]
tests/test_admin.py::test_block_user_invalid_id PASSED                   [ 81%]
tests/test_admin.py::test_unblock_user_success PASSED                    [ 90%]
tests/test_admin.py::test_unblock_user_invalid PASSED                    [100%]
```

AI TESTCASE: GENERATE CHATBOT RESPONSE WITH VALID INPUT

```python
def generate_chat_response(user_input):
    system_prompt = {
        "role": "system",
        "content": (
            "You are a communication teacher. "
            "Your goal is to help students improve their speaking skills, "
            "especially if they are introverted or shy. "
            "Give guidance on public speaking, effective expression, conversation skills, and confidence building."
        )
    }

    chat_messages = [
        system_prompt,
        {"role": "user", "content": user_input}
    ]

    response = "To improve stage confidence, practice deep breathing, visualize success, and start with small audiences."
    return response

def test_generate_chat_response(mock_create):
    user_input = "How can I improve my stage confidence?"
    expected_response = (
        "To improve stage confidence, practice deep breathing, visualize success, and start with small audiences."
    )

    mock_create.return_value = {
        "choices": [
            {"message": {"content": expected_response}}
        ]
    }

    actual_response = generate_chat_response(user_input)

    assert expected_response == actual_response
```

Function: gpt4o_reply(user_text)

Expected Output:
o The chatbot should respond appropriately to communication-related queries using the system prompt set for a **communication teacher**.
o The reply must reflect guidance on improving speaking skills, public speaking, effective expression, or confidence building.
o The response should **not** go beyond the domain of communication coaching.

o QUESTION: "How can I improve my stage confidence?"
o SYSTEM PROMPT: "You are a communication teacher. Your goal is to help students improve their speaking skills, especially if they are introverted or shy. Give guidance on public speaking, effective expression, conversation skills, and confidence building."

Response:
"To improve stage confidence, practice deep breathing, visualize success, and start with small audiences. The more you expose yourself to public speaking situations, the more natural it becomes. Focus on connecting with the audience rather than delivering a perfect speech."

Result: Success

AI TESTCASE: INVALID PROMPT

```python
def test_generate_chat_response_invalid_prompt(mock_create):
    mock_create.return_value = {
        "choices": [{
            "message": {
                "content": (
                    "I'm here to assist with communication skills. "
                    "For math-related queries, please consult a math expert."
                )
            }
        }]
    }

    user_input = "Solve this math problem: 5 + 3 * (2 - 1)"
    result = generate_chat_response(user_input)

    expected_phrase = "communication skills"
    assert expected_phrase in result
```

Test Case: Reject Invalid Prompt Outside Communication Domain

Function: generate_chat_response
Type: Negative Test Case

Input:

- **User Prompt**:
  "Solve this math problem: 5 + 3 * (2 - 1)"
- **System Prompt**: You are a communication teacher. Your goal is to help students improve their speaking skills, especially if they are introverted or shy. Give guidance on public speaking, effective expression, conversation skills, and confidence building.

Expected Output: I'm here to assist you with communication skills and public speaking. For math problems, please consult a math tutor or relevant expert."

Response: "I'm here to assist you with communication skills and public speaking. For math problems, please consult a math tutor or relevant expert."

Result: Success

```python
def test_run_ai_prompt_success(mock_generate_content, capsys):
    mock_response = MagicMock()
    mock_response.text = "Task 'Organic Chemistry Assignment' scheduled for tomorrow."

    mock_generate_content.return_value = mock_response

    user_input = "i dont have time to set up can u quickly schedule a chemistry assignment on organic due tomorrow for me as pending"
    run_ai_prompt(user_input)

    captured = capsys.readouterr()
    assert "Final AI reply" in captured.out
    assert "Organic Chemistry Assignment" in captured.out
```

Function: run_ai_prompt
Module: task_generator_ai.py (assumed name)
Type: Integration Test (AI + Backend API call)

Input Prompt: "I don't have time to set up can u quickly schedule a chemistry assignment on organic due tomorrow for me as pending"

Expected Output:

- A successful confirmation message from backend like:

JSON: {"status": "success", "message": "Task 'Organic Chemistry Assignment' scheduled for tomorrow."}

Response: Task 'Organic Chemistry Assignment' scheduled for tomorrow.

Result: Success

AI TESTCASE: HANDLE INVALID OR AMBIGUOUS TASK PROMPT

```python
def test_run_ai_prompt_invalid_prompt(mock_generate_content, capsys):

    fallback_response = MagicMock()
    fallback_response.text = "I'm sorry, I couldn't understand your request."

    mock_generate_content.return_value = fallback_response

    vague_prompt = "remind me about something important later"
    run_ai_prompt(vague_prompt)

    captured = capsys.readouterr()
    assert "AI reply" in captured.out
    assert "couldn't understand" in captured.out
```

Function: run_ai_prompt
Module: task_generator_ai.py
Type: Integration Test – Failure Handling

Input Prompt: "remind me about something important later"

Expected System Behavior:

The AI either:

- Returns a plain text response (fallback), or
- Skips function call generation entirely.

Expected Output:

JSON: {"detail": " I'm sorry, I couldn't understand your request."}

Response: I'm sorry, I couldn't understand your request.

Result: Success

ISSUE REPORTING AND TRACKING

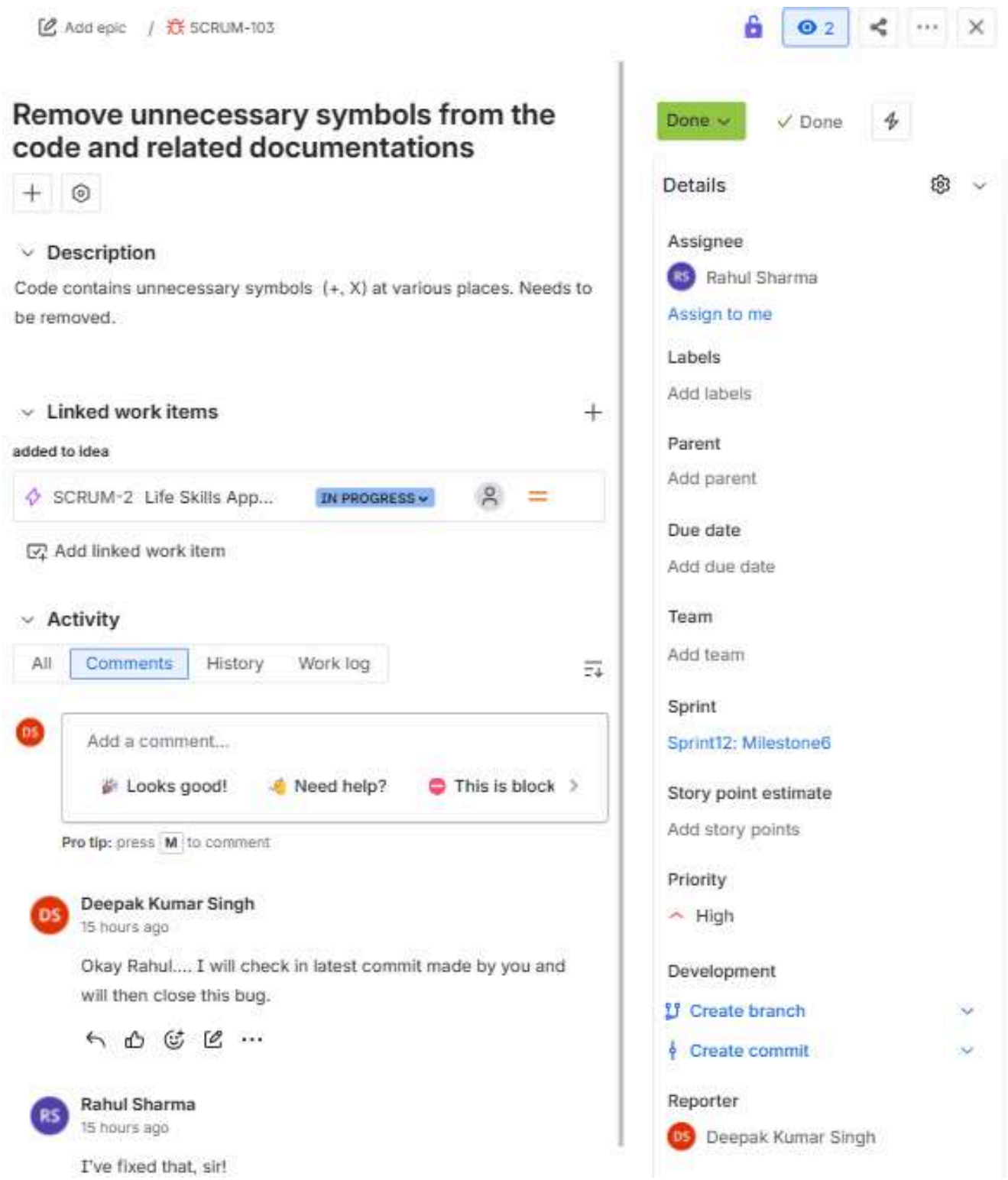
JIRA: BUG REPORTING AND TRACKING

To enhance efficiency and collaboration during the development process, the team leveraged **Jira** for bug reporting, tracking, and resolution management. Within Jira, each identified issue was documented in detail, including descriptions, severity levels, screenshots, and reproduction steps, ensuring clarity for all stakeholders.

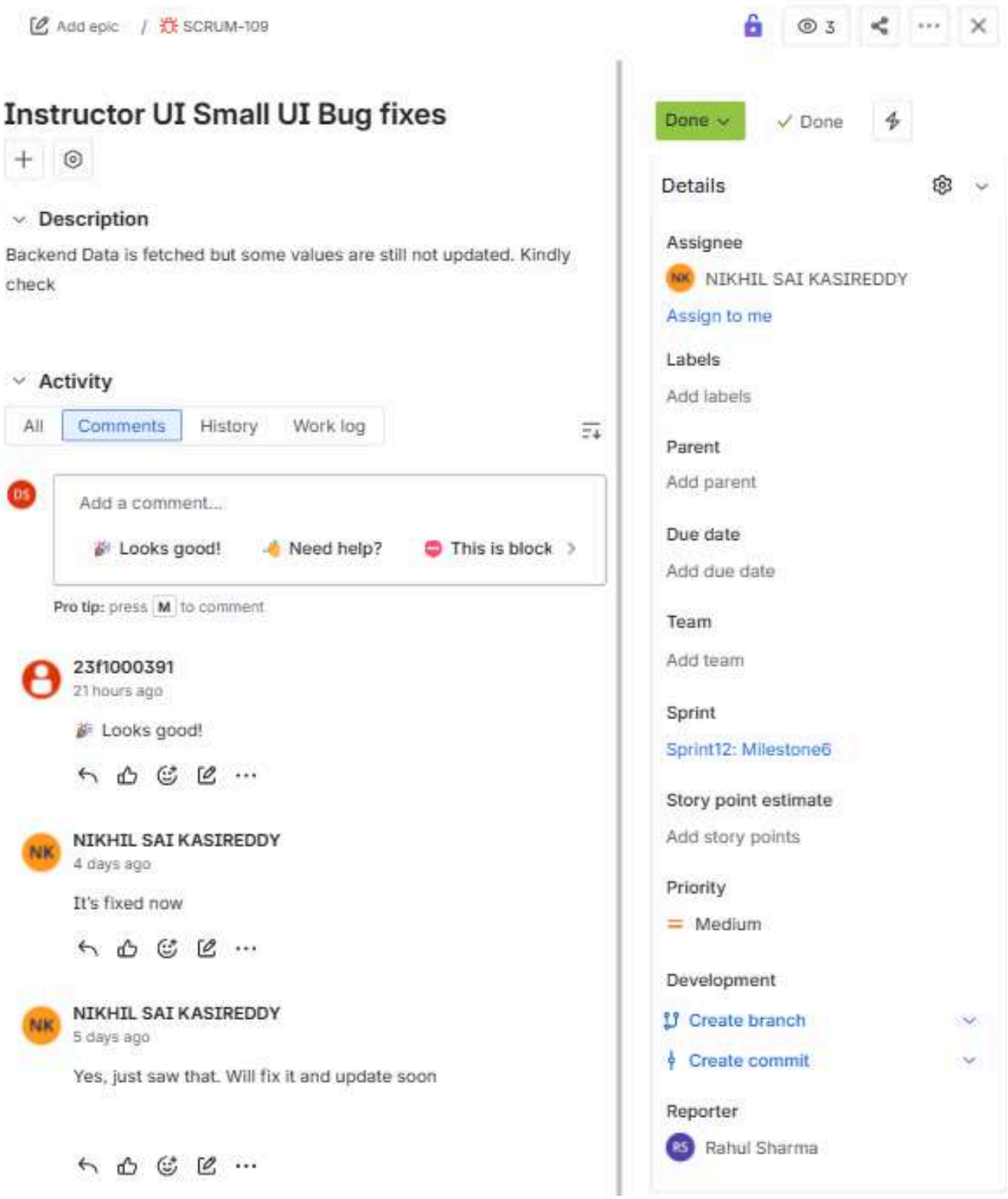
Team members could assign tasks to specific developers, set deadlines, and update statuses in real time, allowing for transparent progress tracking. Additionally, Jira's comment and tagging features facilitated focused discussions on potential solutions, ensuring that every bug was addressed systematically.

Representative screenshots of the Jira bug-tracking board are included below to illustrate this streamlined workflow.

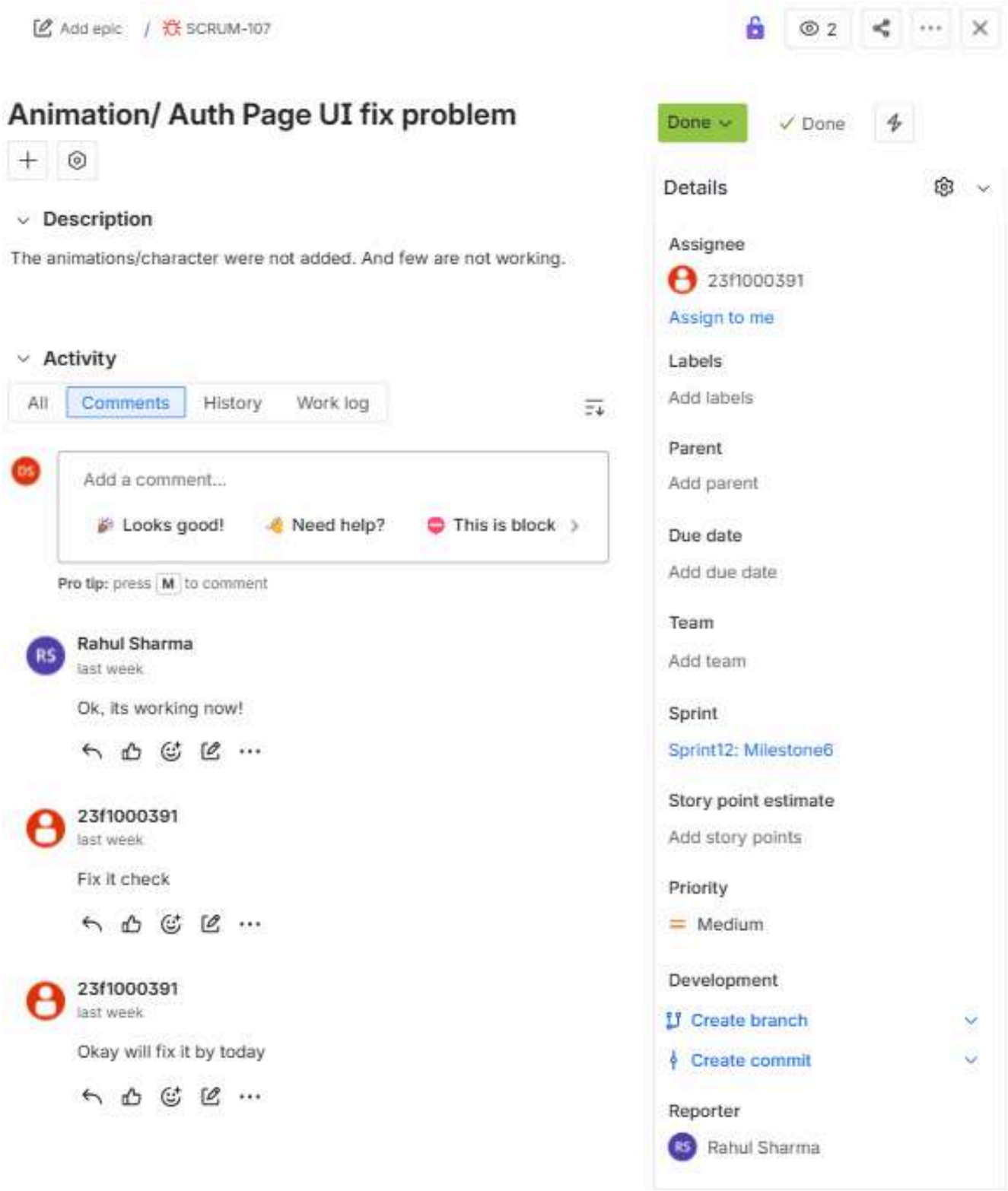
JIRA BUG: SCREENSHOT 1



JIRA BUG: SCREENSHOT 2



JIRA BUG: SCREENSHOT 3



TECHNOLOGIES USED

FRONTEND

- **Vue 3 CLI:** Streamlined Vue projects
- **Vue Router:** Enables navigation
- **VueMarkdown:** Renders markdown
- **JavaScript:** Add dynamic web behavior
- **Bootstrap:** Responsive UI
- **HTML & CSS:** Structure & styling
- **Axios:** Handles API requests

BACKEND

- **Flask:** Core backend framework
- **FastAPI**: Build fast REST APIs
- **SQLAlchemy:** Database ORM
- **JWT:** Secure user authentication tokens
- **Celery:** Background scheduled tasks
- **Pytest:** Automated testing
- **PDFKit:** Generates PDFs from HTML

AI

- **OpenAI:** Build AI-powered applications
- **LangChain**: Facilitates AI-driven conversations with seamless integration of LLMs.

GENERAL

- **GitHub**: Version control, code collaboration
- **Jira**: Project tracking
- **Swagger Editor**: API documentation
- **Figma**: Wireframes
- **Canva**: Create reports and presentations

INSTRUCTIONS TO RUN THE APPLICATION

STEPS TO RUN APPLICATION

Clone the repository

- git clone https://github.com/rahulsharmaYS/skillzy

Backend initialisation

Open 1st terminal in your IDE.

- cd Backend
- pip install -r requirements.txt
- python -m uvicorn main:app --reload

Frontend initialisation

Open 2nd terminal in your IDE.

- cd ../frontend/SE-project
- *npm install*
- *npm run dev*

Start Celery Workers (in 2 different terminals)

Open 3rd, 4th and 5th terminal in your IDE.

- cd Backend
- redis-server
- celery -A celery_app worker --loglevel=debug
- celery -A celery_app beat --loglevel=debug

Add API Keys

In the *Backend/.env file*, make sure to insert your valid OPENAI API KEY.
Example: "*OPENAI_API_KEY=sk-proj-qwertasdfzxc*"

In case of any error from any of the step, kindly read the detailed read me from this link

Links and cloning will work once the repo is public and we get the grades.