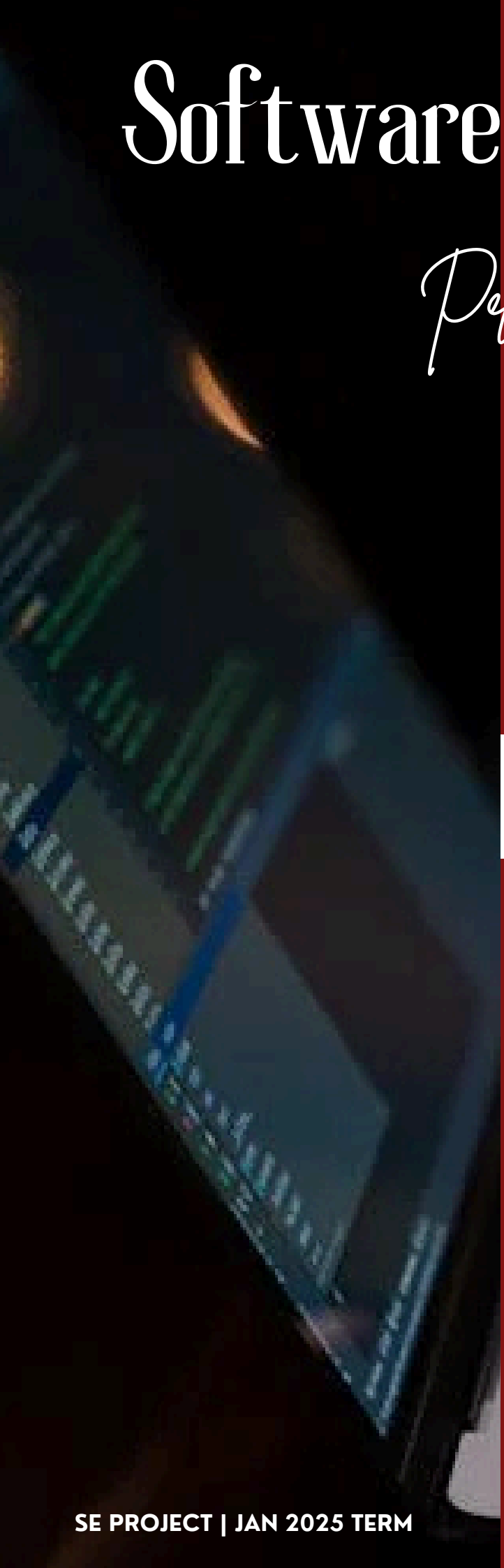


Software Engineering

Project

TEAM 16

Prepared By:

Anish Maity (21f3000417)
Achin Aggarwal (22f1001390)
Atharva Sarbhukan (22f1000533)
Abhishek Darji (22f1000678)
Sarath Sasidharan Pillai (22f1000152)
Rituparno Sen (21f1004275)
Dr. Ambrish Naik (22f2000424)

Meet Our Team

ACKNOWLEDGEMENT

We, Team 16 of the Software Engineering Project, JAN 2025 Term, sincerely thank everyone who supported us throughout this journey.

We are especially grateful to our professors and mentors for their invaluable guidance, feedback, and encouragement, which played a crucial role in our project's success. We also appreciate our peers and teammates for their collaboration and insightful discussions, enriching our learning experience.

A special thanks to IIT Madras and the Software Engineering course team for providing the resources and platform to apply our knowledge in a practical and impactful way.

Thank you all for your support!

ANISH
21f3000417

ACHIN
22f1001390

ATHARVA
22f1000533

ABHISHEK
22f1000678

RITUPARNO
21f1004275

SARATH
22f1000152

Dr. AMBRISH
22f2000424

ABOUT THE PROJECT

The SEEK portal harnesses **Generative AI (GenAI)** to enhance academic guidance by providing **personalized learning support**. It enables students to engage more effectively with course materials through **AI-driven automation and intelligent assistance**, transforming digital education into an interactive and adaptive experience

Learning Environment

The SEEK portal offers an **interactive learning experience** with:

- Video Lectures & Course Materials for easy access to study resources.
- Announcements & Updates on coursework, deadlines, and events.

Coding Environment

The SEEK portal features an interactive coding platform with:

- Live Code Execution for real-time coding, testing, and debugging.
- Automated Evaluation providing instant feedback on assignments

GenAI-Powered Academic Support

The SEEK portal's AI assistant enhances learning by providing:

- Instant Query Resolution for course-related doubts with AI-generated responses.
- Lecture Assistance to clarify concepts from video lectures.
- AI-Powered Code Review offering optimized corrections to improve coding skills.

Software Engineering Course

The Software Engineering course is designed to equip students with the fundamental skills necessary to excel as effective software engineers. The course covers essential concepts in software development, including requirement gathering, software conceptual design, usability, quality evaluation, debugging, testing, and deployment.

Throughout the 12-week course, students will engage in weekly online assignments and participate in two in-person invigilated quizzes and a final invigilated end-term exam. The course provides a comprehensive understanding of the software development lifecycle, from deconstructing the development process to deploying and monitoring software systems.

CONTENTS

IDENTIFY USERS REQUIREMENTS

USER IDENTIFICATION

Primary Users :

Students:

- Students enrolled in the IITM BS program.
- They access the Seek Learning portal to:
  - Register for courses.
  - Watch weekly lesson videos and access resources.
  - Complete practice and graded assignments.
  - Participate in discussions and seek help via the discourse forum.

Secondary Users :

Instructors and Teaching Assistants (TAs):

- Responsible for :
  - Releasing weekly lessons and assignments.
  - Conducting live online sessions.
  - Responding to student queries on the discourse forum.

USER IDENTIFICATION

Tertiary Users :

Professors :

- Occasionally resolve student queries.
- Conduct live sessions.
- Review course materials to ensure quality.

IT Support Staff :

- Provide technical support for the Seek portal.

Administrative Staff:

- Manage tasks such as registration, scheduling, and maintenance of student records.

Course Coordinators:

- Oversee the effective delivery of courses and ensure smooth operations.

USER STORIES

Primary Users

Without GenAi :-

A As a student, I want to navigate the portal to search for course materials so that I can efficiently find the resources needed to complete my assignments.

B As a student, I want to check my course progress and assignment status so that I can track my performance and stay on top of deadlines.

C As a student, I want to ask questions in the discourse forum to seek help from my peers or instructors when I encounter challenges in my coursework.

D As a student, I want to participate in topic-based discussion forums to clarify doubts and exchange ideas with other students.

E As a student, I want to post questions in the discourse forum and wait for responses from peers or instructors, so that I can receive assistance, even if it may take time.

USER STORIES

Primary Users

With GenAi :-

A As a student, I want the GenAI agent to suggest the most relevant study materials based on my specific queries so that I can save time and focus on critical topics.

B As a student, I want the GenAI agent to analyze my progress and provide insights so that I can enhance my learning outcomes and identify areas of improvement.

C As a student, I want the GenAI agent to provide hints or debugging tips for programming assignments so that I can solve problems independently and improve my coding skills.

D As a student, I want the GenAI agent to notify me about upcoming deadlines and new course materials so that I can stay organized and manage my coursework efficiently.

E As a student, I want the GenAI assistant to provide immediate responses to my queries so that I can resolve doubts quickly and stay productive without delays.

USER STORIES

Secondary Users

Without GenAi :-

A As a TA, I want to respond to student queries on the discourse forum so that I can address their doubts and provide timely solutions.

B As an instructor, I want to conduct live sessions using the Seek portal so that I can engage with students and resolve their queries in real-time.

C As an instructor, I want to manually upload and organize assignments for different courses so that students can easily access them without confusion.

USER STORIES

Secondary Users

With GenAi :-

A As a TA, I want the GenAI agent to handle common student queries so that I can focus on resolving more complex issues and provide more in-depth support where needed.

B As a TA, I want the GenAI agent to provide automated hints for practice assignments so that students can solve minor issues independently, improving their learning experience.

C As an instructor, I want the GenAI agent to suggest relevant practice assignments for students so that they can strengthen their understanding of complex topics and improve their overall performance.

USER STORIES

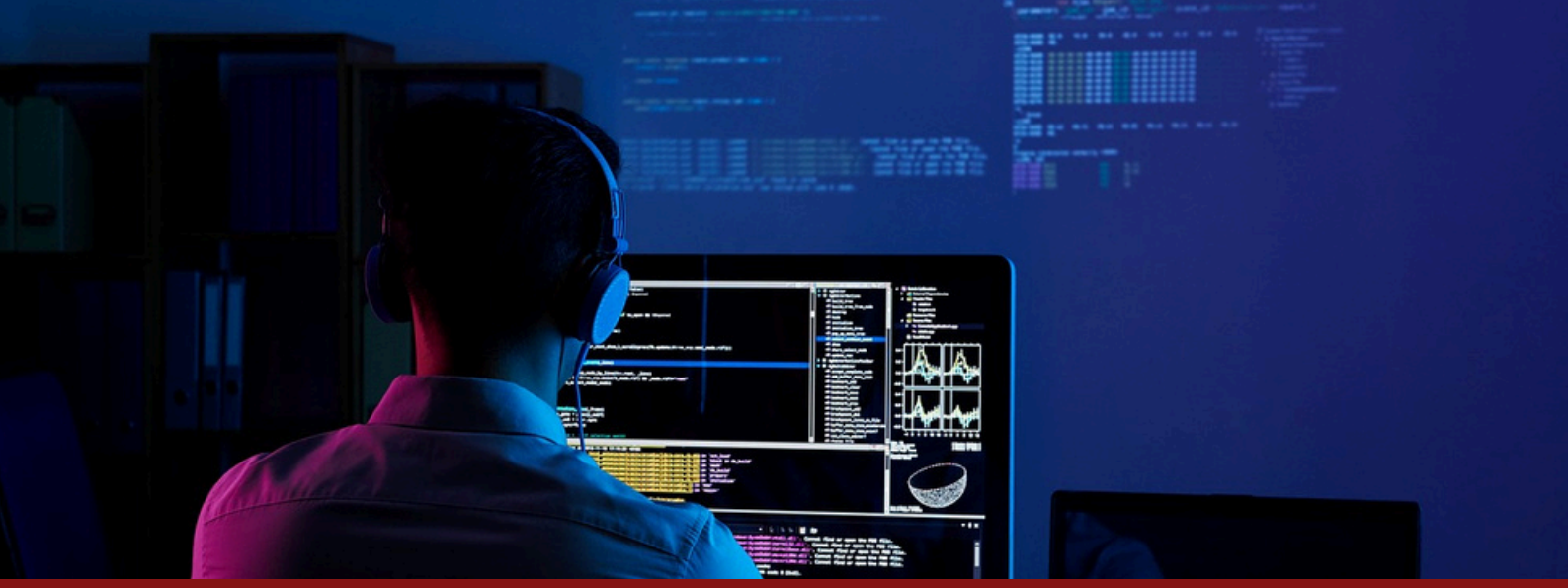
Tertiary Users

Without GenAi :-

A As a professor, I want to occasionally address advanced student queries so that I can help clarify complex topics and deepen their understanding.

B As an admin staff member, I want to manually address student complaints so that I can resolve their issues in a timely and effective manner.

C As a professor, I want to upload new academic materials directly to the portal so that students can easily access the latest updates to the course.



USER STORIES

Tertiary Users

With GenAi :-

A As a professor, I want the GenAI agent to handle initial queries so that I can focus on more advanced academic discussions and support students with deeper questions.

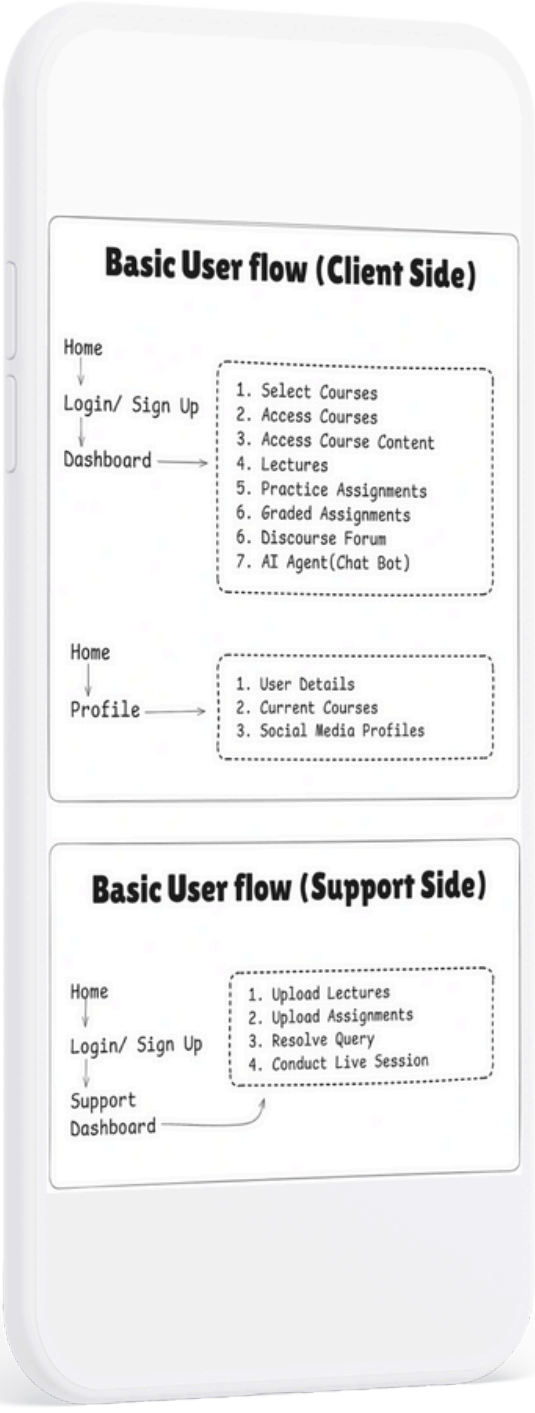
B As an admin staff member, I want the GenAI agent to suggest possible resolutions for common student complaints so that I can address them faster and more efficiently.

C As a professor, I want the GenAI agent to assist me in uploading new academic materials so that students can access the latest course updates in an organized and efficient manner.

MILESTONE 2

USER INTERFACES

APP USERFLOW



SEEK Portal User Workflow

The SEEK Portal simplifies academic activities for students, instructors, and administrators through AI-powered automation, ensuring efficient learning and support.

On the client side, students log in to access the Dashboard, where they can select courses, review content, complete assignments, and track progress. An AI chatbot provides instant query resolution, while deadline alerts help manage submissions.

The Profile section lets students manage details and track courses, with AI-driven insights offering study recommendations and progress tracking.

On the support side, instructors and TAs use the Support Dashboard to upload content, resolve queries, and conduct live sessions. AI automates query resolution and assignment hints, reducing manual workload.

For administrators, AI enhances scheduling, content verification, and issue resolution, streamlining operations and improving efficiency.

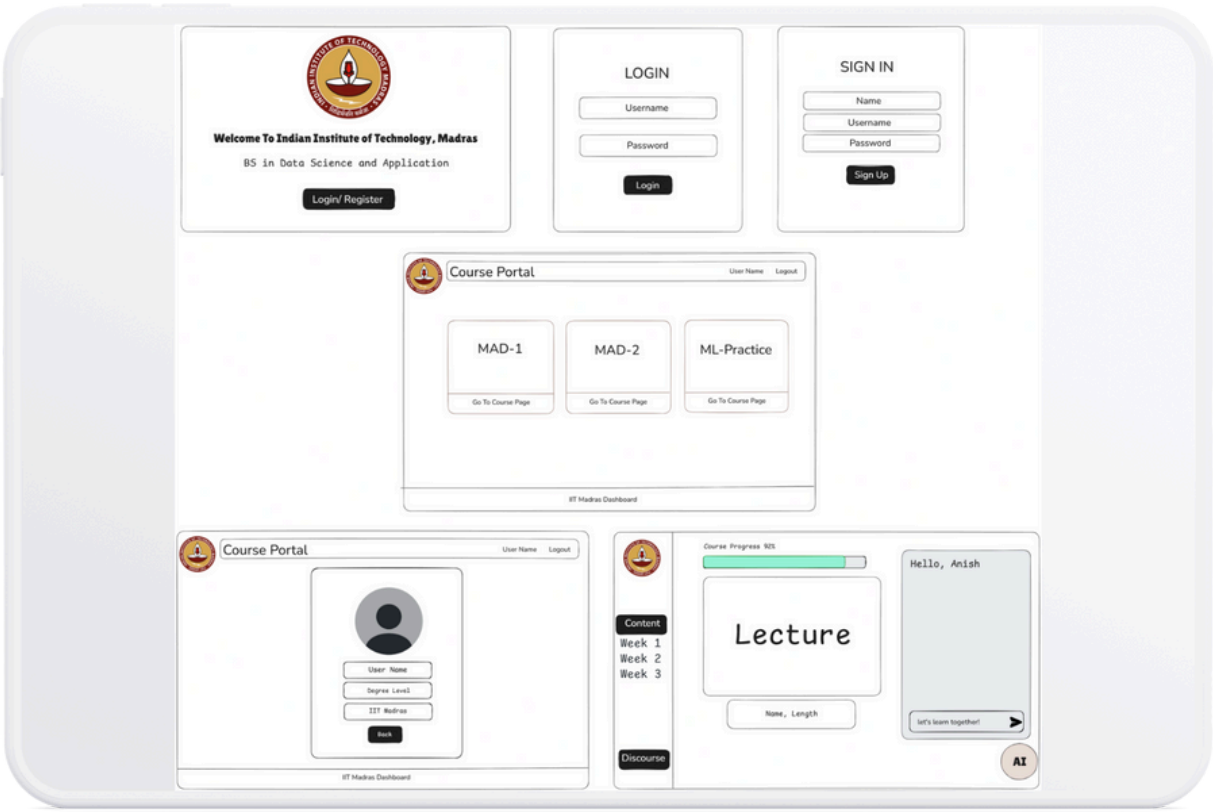
Overall, the SEEK Portal ensures instant support, personalized learning, and efficient faculty assistance, making academic workflows seamless.

WIREFRAMES
(Low Fidelity)

The wireframes for the AI-powered academic guidance system depict an intuitive user journey with GenAI enhancements. Students begin at the Dashboard, where the AI Study Assistant recommends resources based on coursework, while the AI Progress Insights feature tracks performance and suggests areas for improvement. The GenAI Chatbot provides instant query responses, reducing reliance on discourse forums, while the Smart Debugging Assistant offers hints for programming assignments. Deadline Alerts ensure students stay on track with assignments and exams.

For Instructors and TAs, the AI-Powered Dashboard automates query resolution, suggests AI-generated practice assignments, and provides insights into student performance trends. The Assignment Hint System allows GenAI to guide students before TA intervention, while Lecture & Course Insights help identify struggling topics.

Professors, Admin Staff, and IT Support benefit from AI-enhanced content verification, automated query screening, and complaint resolution suggestions, reducing manual workload. AI-driven predictive analytics assist in student enrollment trends, scheduling, and system monitoring, proactively flagging potential issues before they impact users. These AI-powered wireframes ensure a seamless, efficient, and personalized academic experience for all users.



USER STORIES

User stories provide a structured way to define user needs, outlining their roles, actions, and expected outcomes. They help ensure that the development process aligns with user expectations by focusing on real-world scenarios. In the AI-powered academic guidance system, user stories capture how students, instructors, and administrators interact with the SEEK portal to enhance their learning and teaching experiences.

With GenAI integration, user interactions become more efficient and personalized. Students benefit from instant AI-driven support, tailored study recommendations, and real-time progress insights, reducing the effort needed to find relevant resources. Instructors and TAs gain from automated query resolution, AI-generated practice assignments, and performance tracking, making student support more seamless. Professors and administrative staff leverage AI-powered content verification, predictive scheduling, and automated issue resolution, minimizing manual workload and streamlining operations.

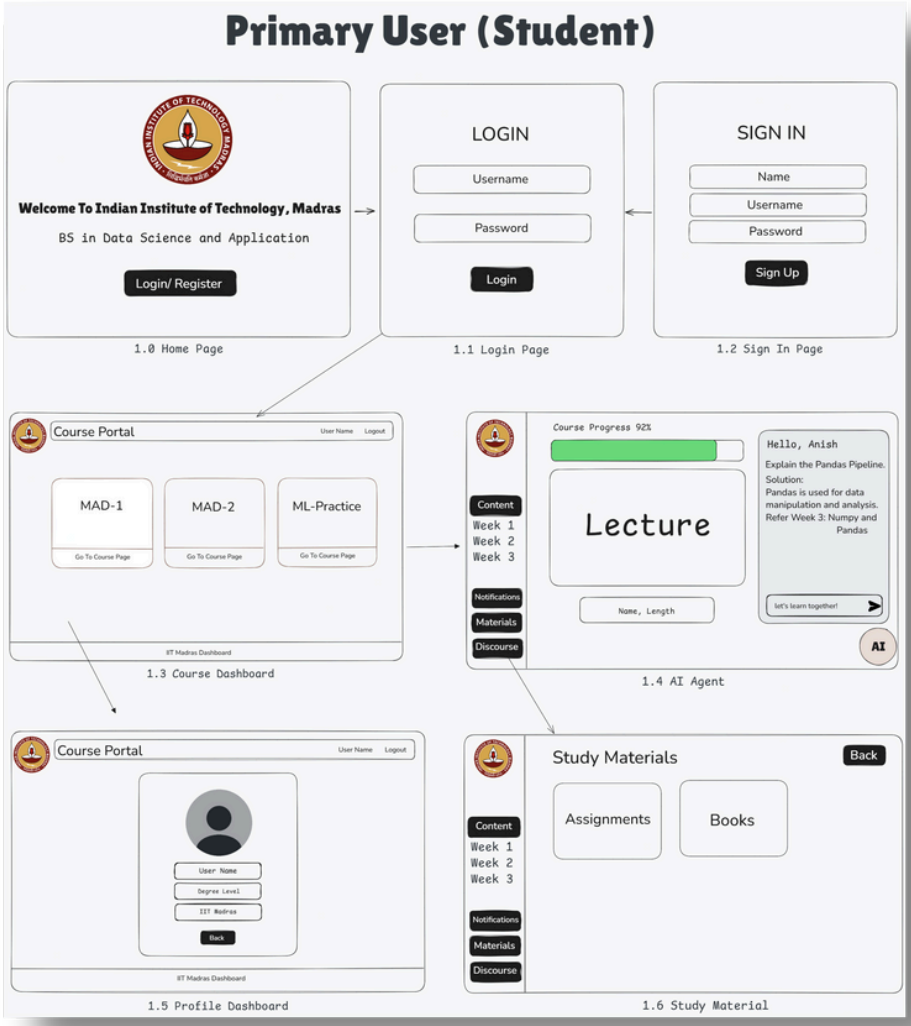
By incorporating GenAI, user stories evolve into intelligent, automated workflows, improving accessibility, efficiency, and engagement. This transformation makes the SEEK portal a more adaptive and student-focused learning environment that enhances the overall academic experience.

User Stories | Primary User | Scenario 1

Searching Course Resources

USERFLOW

HOME → LOGIN → COURSE PORTAL→ COURSES → STUDY MATERIALS


Primary User (Student)
Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
LOGIN
Username
Password
Login
1.1 Login Page
SIGN IN
Name
Username
Password
Sign Up
1.2 Sign In Page
Course Portal
User Name
Logout
MAD-1
MAD-2
ML-Practice
Go To Course Page
IIT Madras Dashboard
1.3 Course Dashboard
Course Progress 92%
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Lecture
Name, Length
Hello, Anish
Explain the Pandas Pipeline.
Solution:
Pandas is used for data manipulation and analysis.
Refer Week 3: Numpy and Pandas
let's learn together!
AI
1.4 AI Agent
Course Portal
User Name
Logout
User Name
Degree Level
IIT Madras
Back
IIT Madras Dashboard
1.5 Profile Dashboard
Study Materials
Back
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Assignments
Books
1.6 Study Material


I found the materials I needed, but searching through the SEEK portal took quite some time.

ENTERS!


GenAI


Wow! GenAI made this so much easier. The instant recommendations saved me a lot of time!

User Stories | Primary User | Scenario 2

Course Progress Performance Insights

USERFLOW

HOME → LOGIN → COURSE PORTAL → COURSES → COURSES PROGRESS


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Course Portal
User Name
Logout
MAD-1
MAD-2
ML-Practice
Go To Course Page
Go To Course Page
Go To Course Page
IIT Madras Dashboard
1.1 Course Dashboard
Course Progress 92%
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Lecture
Name, Length
Hello, Anish
let's learn together!
AI
1.2 AI Agent


I have a basic idea of my progress, but I wish I had more detailed insights to understand where I need to improve.

ENTERS!


GenAI


Wow! GenAI analyzed my progress and gave me detailed insights. Now I know exactly where to focus and how to improve!

User Stories | Primary User | Scenario 3

Interactive Programming Help

USERFLOW

HOME → LOGIN → COURSE PORTAL→ COURSES → CHATBOT


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Course Portal
User Name
Logout
MAD-1
Go To Course Page
MAD-2
Go To Course Page
ML-Practice
Go To Course Page
IIT Madras Dashboard
1.1 Course Dashboard
Course Progress 92%
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Lecture
Name, Length
Hello, Anish
import test_train_split from dataset.
Check Error.
Solutions:
Use Week 3 and Week 9 sklearn.model_selection module.
let's learn together!
AI
1.2 AI Agent


I'm stuck on this programming error, and figuring it out on my own is taking so much time. I wish I had some guidance.

ENTERS!


GenAI

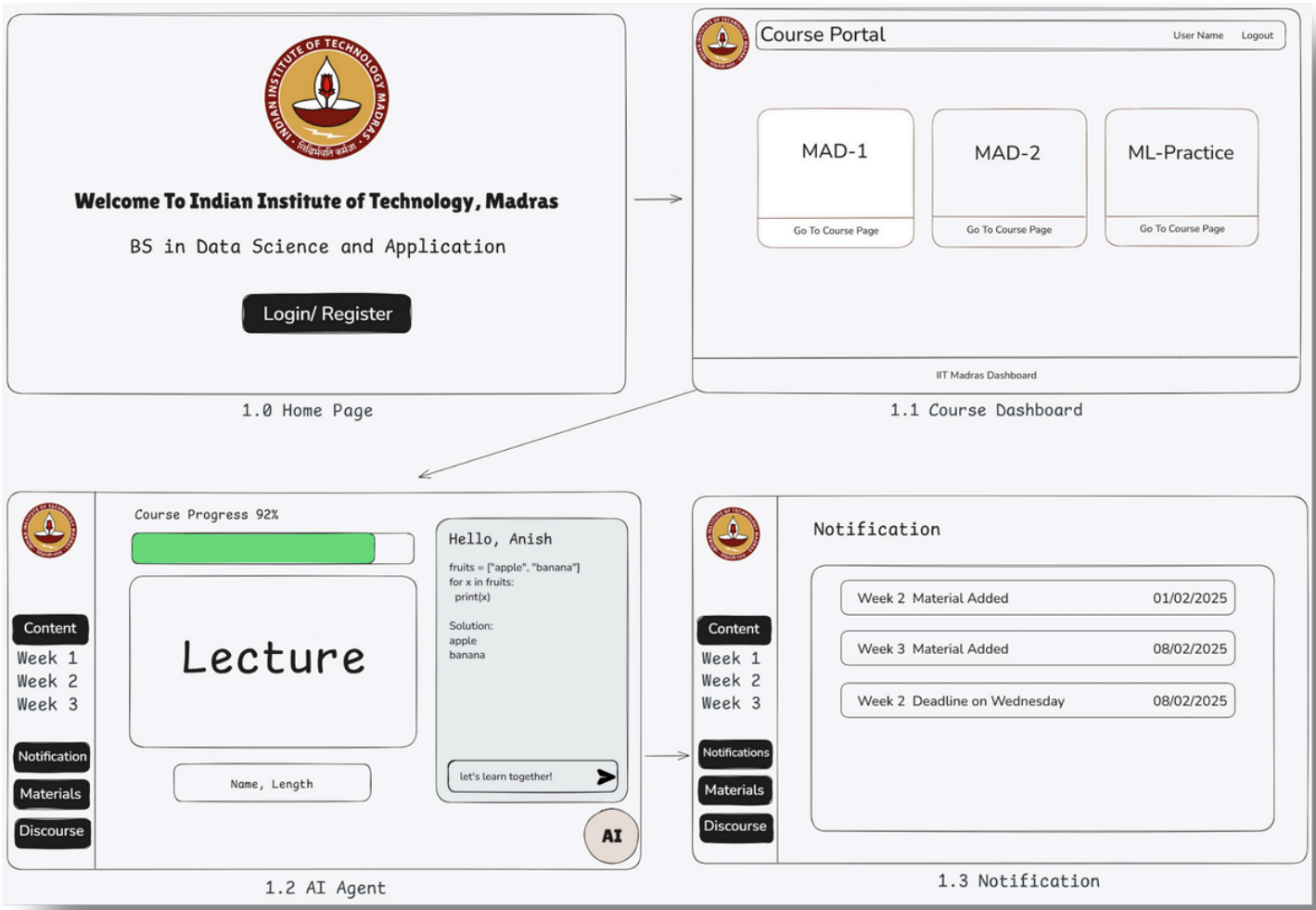

Wow! GenAI helped me debug my code with hints and suggestions. Now I can solve problems faster and improve my coding skills!

User Stories | Primary User | Scenario 4

Checking Proactive Alerts

USERFLOW

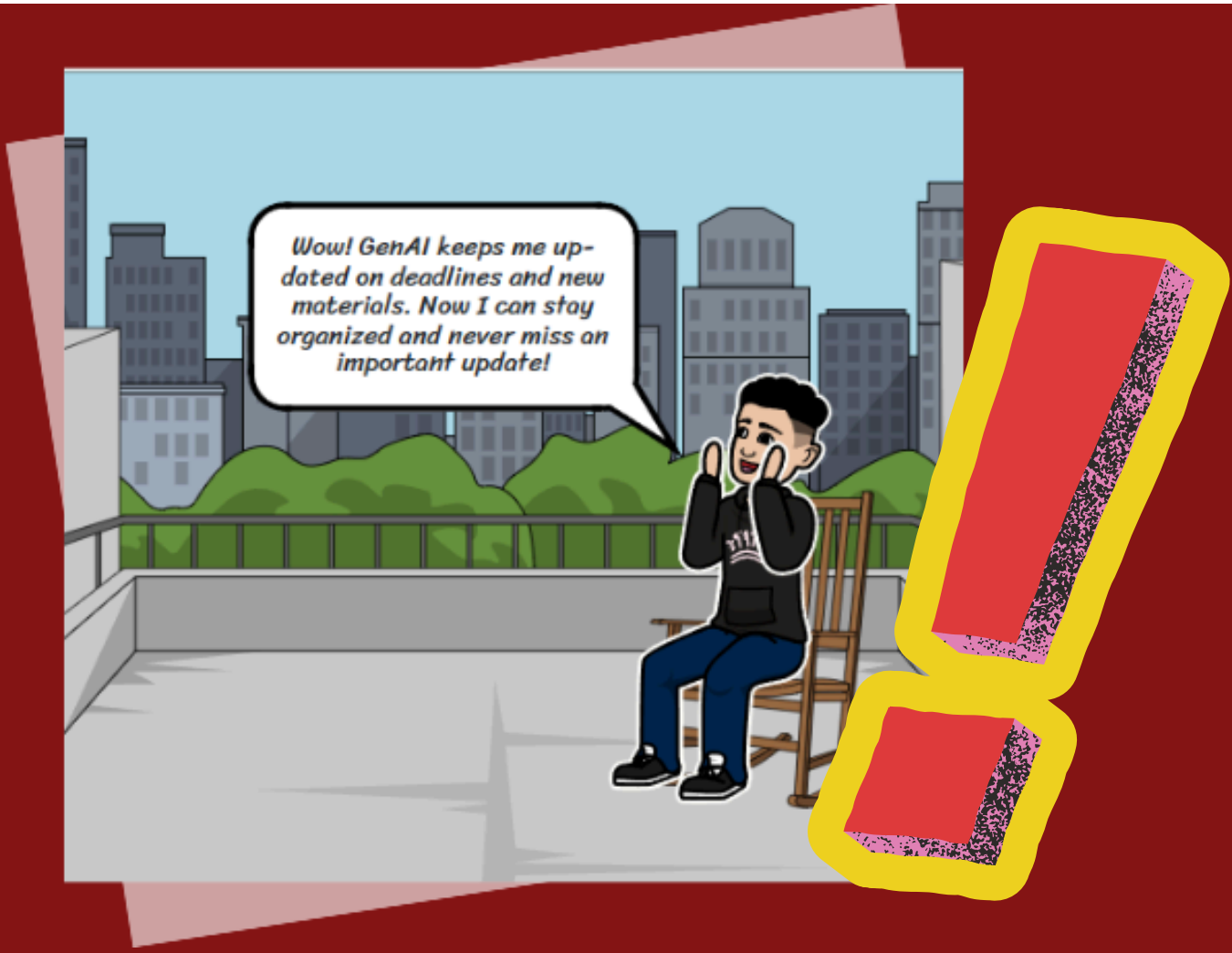
HOME → LOGIN → COURSE PORTAL → COURSE MATERIALS → NOTIFICATIONS


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Course Portal
User Name
Logout
MAD-1
MAD-2
ML-Practice
Go To Course Page
Go To Course Page
Go To Course Page
IIT Madras Dashboard
1.1 Course Dashboard
Course Progress 92%
Content
Week 1
Week 2
Week 3
Notification
Materials
Discourse
Lecture
Name, Length
Hello, Anish
fruits = ["apple", "banana"]
for x in fruits:
print(x)
Solution:
apple
banana
let's learn together!
AI
1.2 AI Agent
Notification
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Week 2 Material Added 01/02/2025
Week 3 Material Added 08/02/2025
Week 2 Deadline on Wednesday 08/02/2025
1.3 Notification


I try to keep track of deadlines, but sometimes I miss important updates. I wish there was a better way to stay organized.

ENTERS!


GenAI

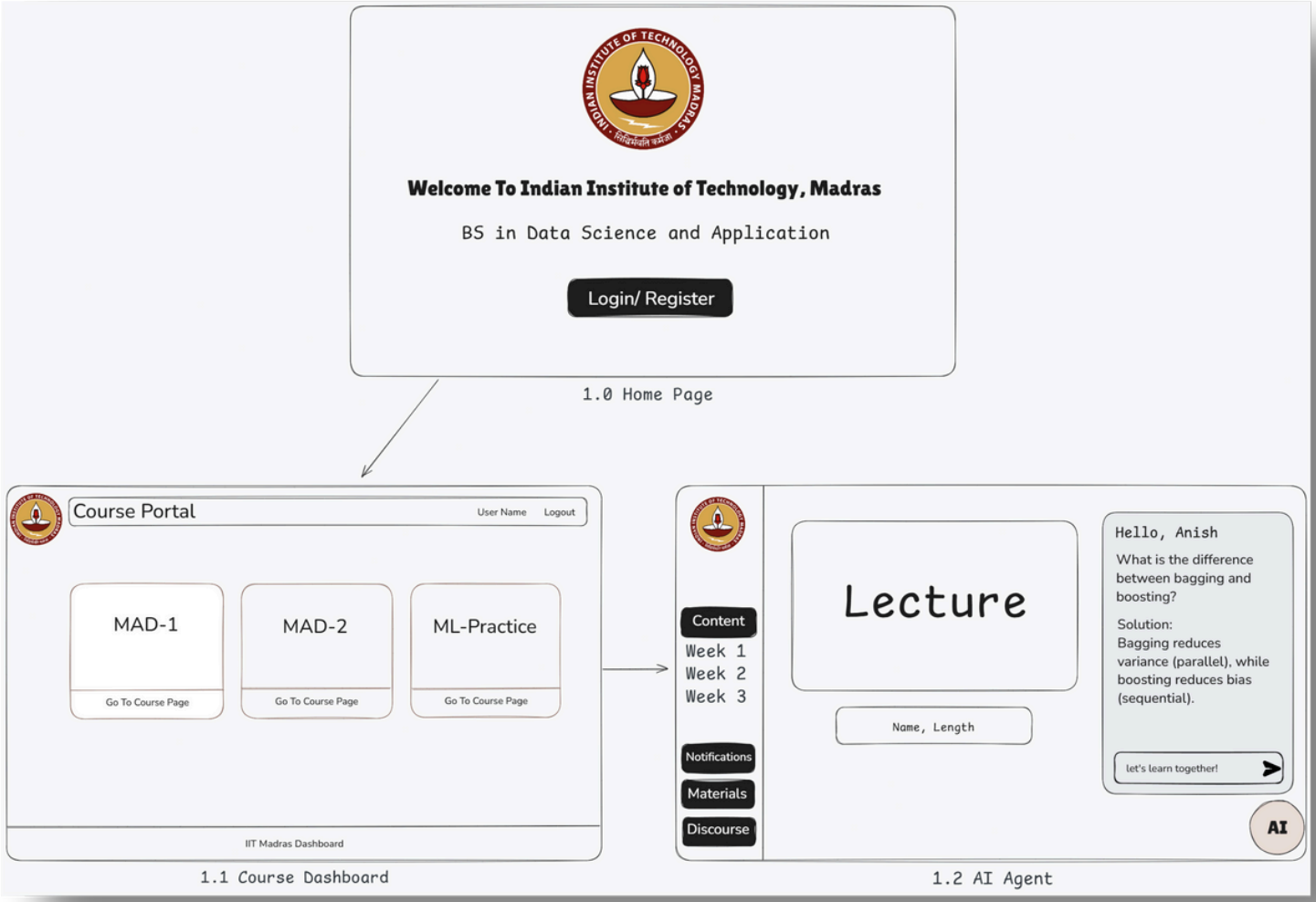

Wow! GenAI keeps me up-dated on deadlines and new materials. Now I can stay organized and never miss an important update!

User Stories | Primary User | Scenario 5

Quick Responses to Queries

USERFLOW

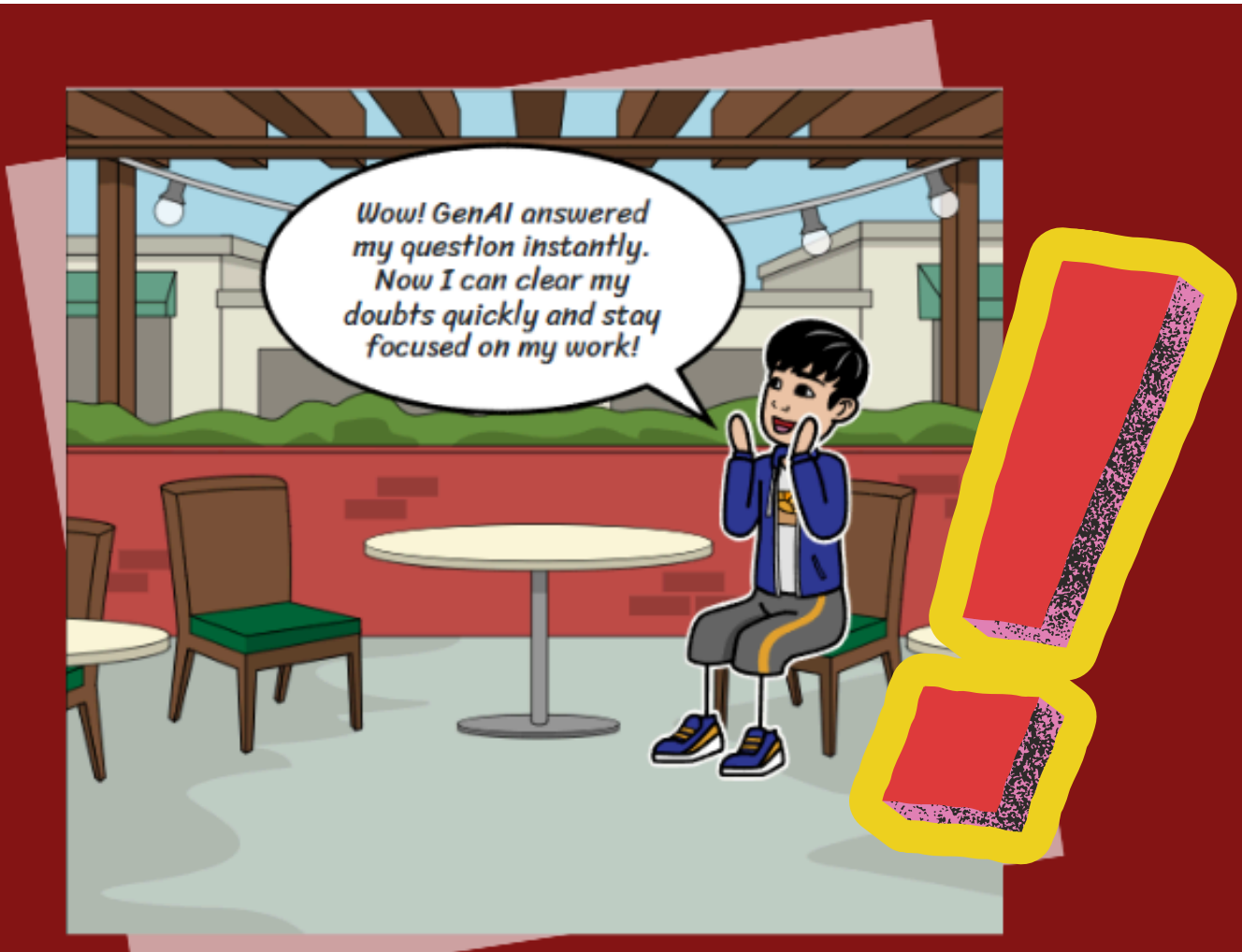
HOME → LOGIN → COURSE PORTAL → COURSE MATERIALS → CHATBOT


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Course Portal
User Name
Logout
MAD-1
Go To Course Page
MAD-2
Go To Course Page
ML-Practice
Go To Course Page
IIT Madras Dashboard
1.1 Course Dashboard
Content
Week 1
Week 2
Week 3
Notifications
Materials
Discourse
Lecture
Name, Length
Hello, Anish
What is the difference between bagging and boosting?
Solution:
Bagging reduces variance (parallel), while boosting reduces bias (sequential).
let's learn together!
AI
1.2 AI Agent


I posted my question, but now I have to wait for a response. I wish I could get answers instantly.

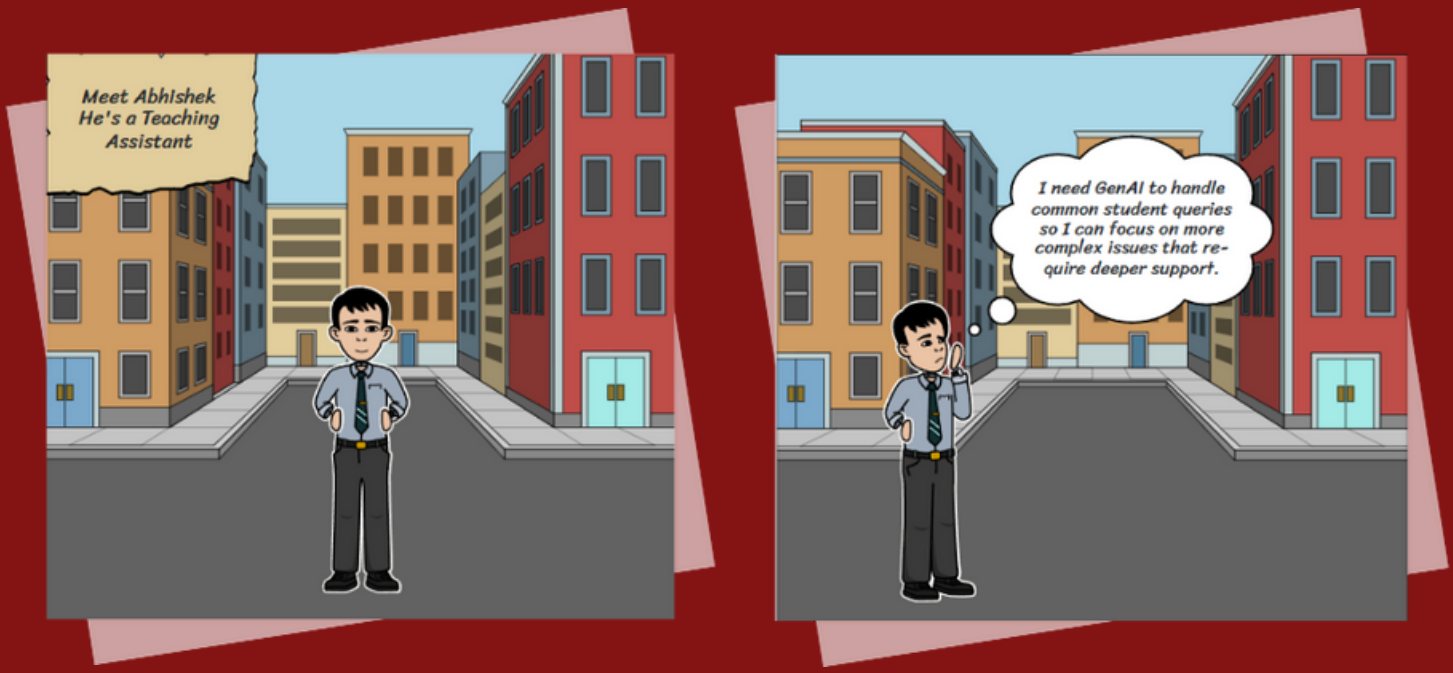
ENTERS!


GenAI


Wow! GenAI answered my question instantly. Now I can clear my doubts quickly and stay focused on my work!

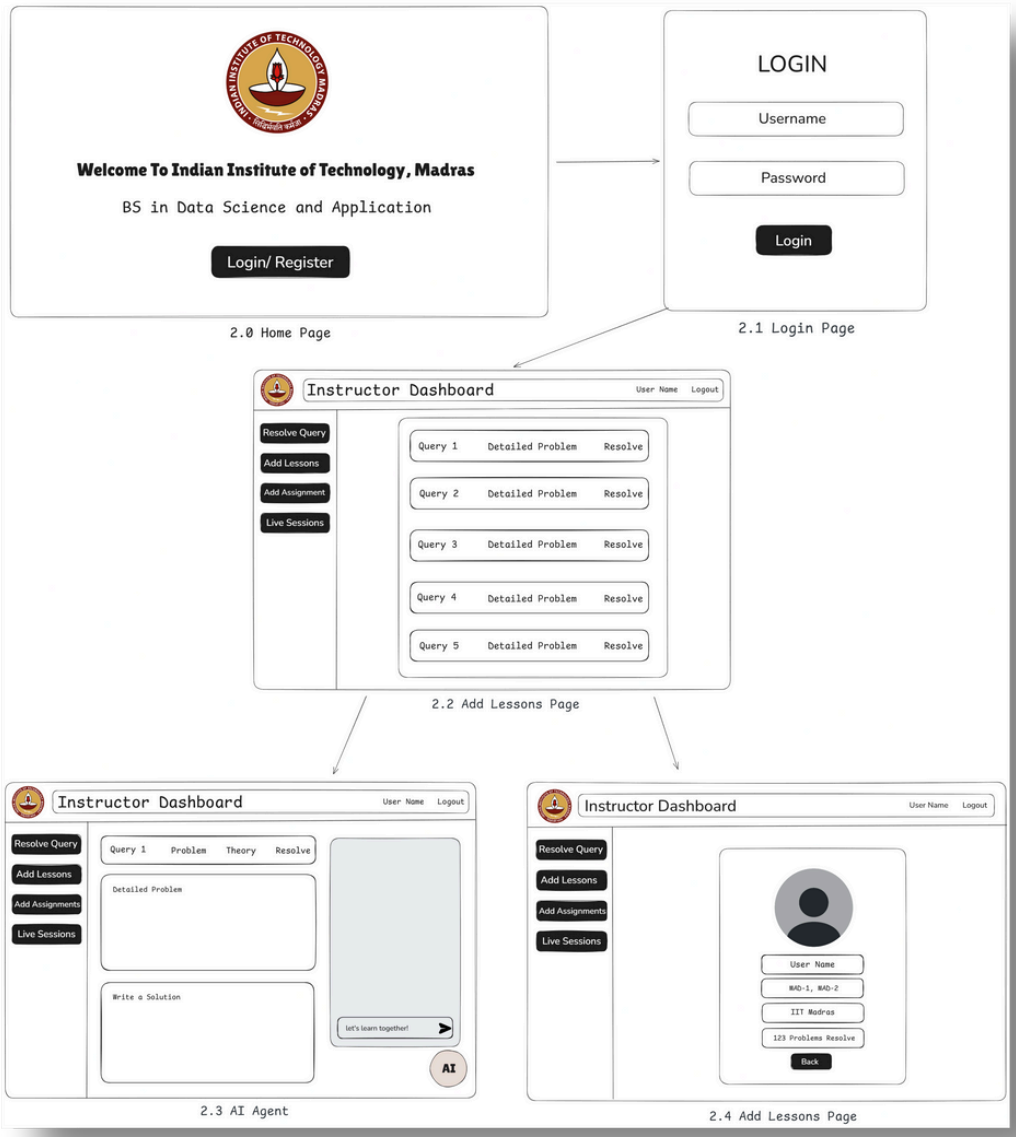
User Stories | Secondary User | Scenario 1

Automated Query Resolution



USERFLOW

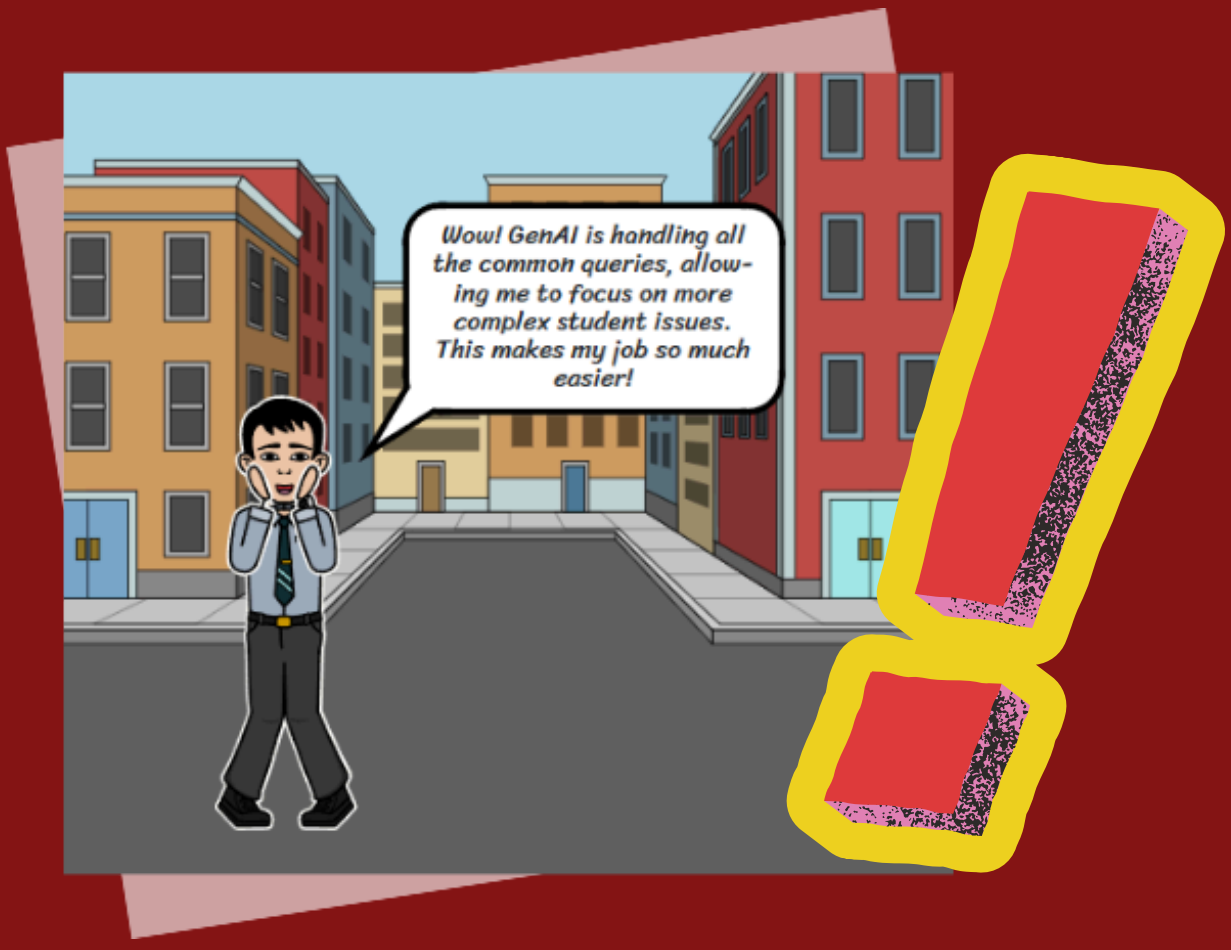
HOME → LOGIN → INSTRUCTOR DASHBOARD → QUERIES → RESOLVE QUERY


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
2.0 Home Page
LOGIN
Username
Password
Login
2.1 Login Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignment
Live Sessions
Query 1 Detailed Problem Resolve
Query 2 Detailed Problem Resolve
Query 3 Detailed Problem Resolve
Query 4 Detailed Problem Resolve
Query 5 Detailed Problem Resolve
2.2 Add Lessons Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignments
Live Sessions
Query 1 Problem Theory Resolve
Detailed Problem
Write a Solution
let's learn together!
AI
2.3 AI Agent
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignments
Live Sessions
User Name
MAD-1, MAD-2
IIT Madras
123 Problems Resolve
Back
2.4 Add Lessons Page


I'm spending so much time answering repet-itive questions when I could be helping with more complex issues.

ENTERS!


GenAI


Wow! GenAI is handling all the common queries, allowing me to focus on more complex student issues. This makes my job so much easier!

User Stories | Secondary User | Scenario 2

Assignment Assistance

USERFLOW

HOME → LOGIN → INSTRUCTOR DASHBOARD → QUERIES → RESOLVE QUERY


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
2.0 Home Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignment
Live Sessions
Query 1 Detailed Problem Resolve
Query 2 Detailed Problem Resolve
Query 3 Detailed Problem Resolve
Query 4 Detailed Problem Resolve
Query 5 Detailed Problem Resolve
2.1 Add Lessons Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignments
Live Sessions
Query 1 Problem Code Resolve
num1 = 1.5
num2 = 6.3
# Add two numbers
sum = num1 + num2
# Display the sum
print(num1 + num2)
Add a Solutions
let's learn together!
AI
2.2 AI Agent


Students keep asking for hints on practice assignments, and I have to explain the same concepts repeatedly. I wish there was a way for them to get quick guidance.

ENTERS!


GenAI

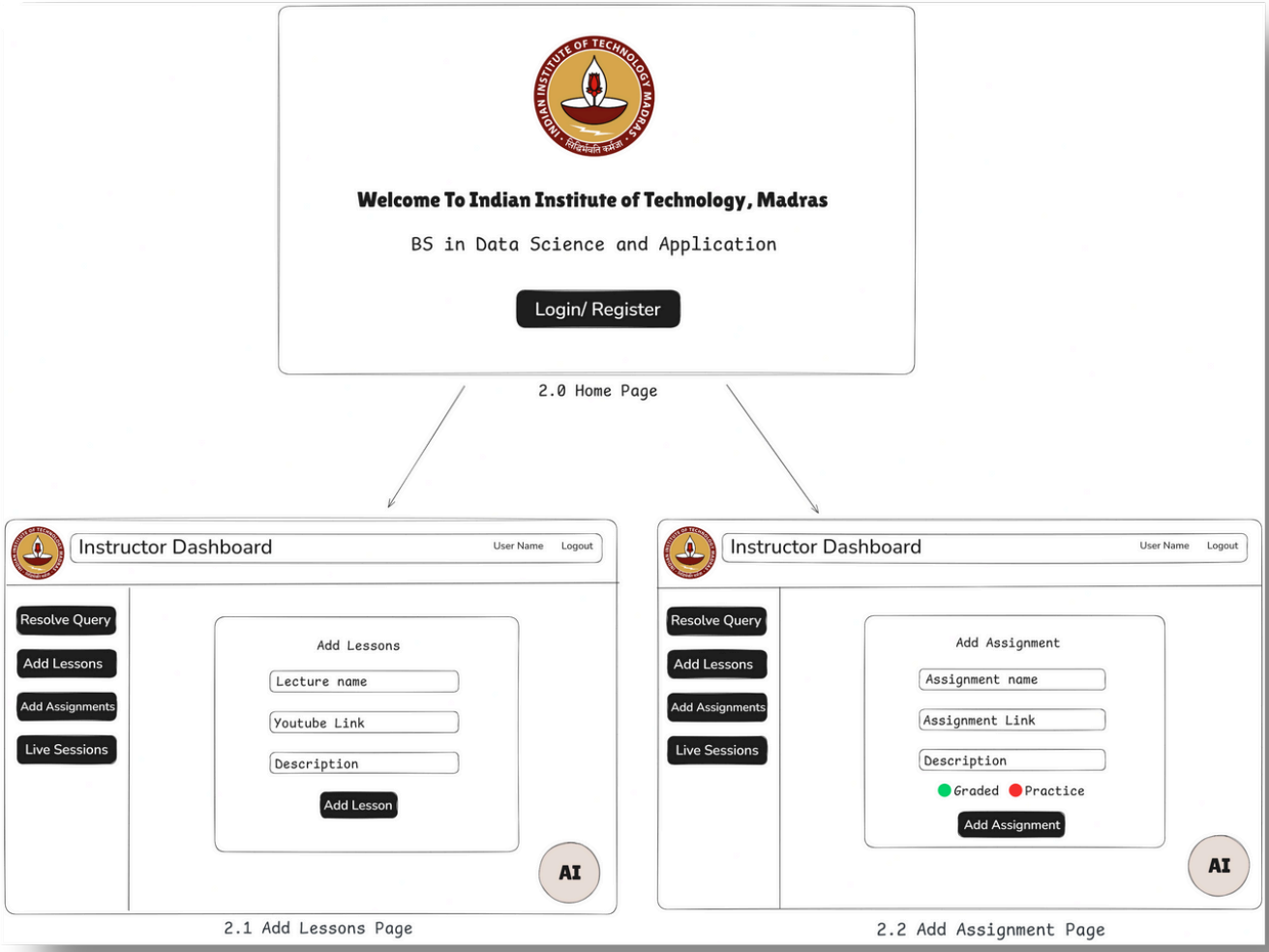

Wow! GenAI is providing students with instant hints, allowing them to solve minor issues on their own. This frees me up to focus on more complex questions!

User Stories | Secondary User | Scenario 3

Assignment Recommendations

USERFLOW

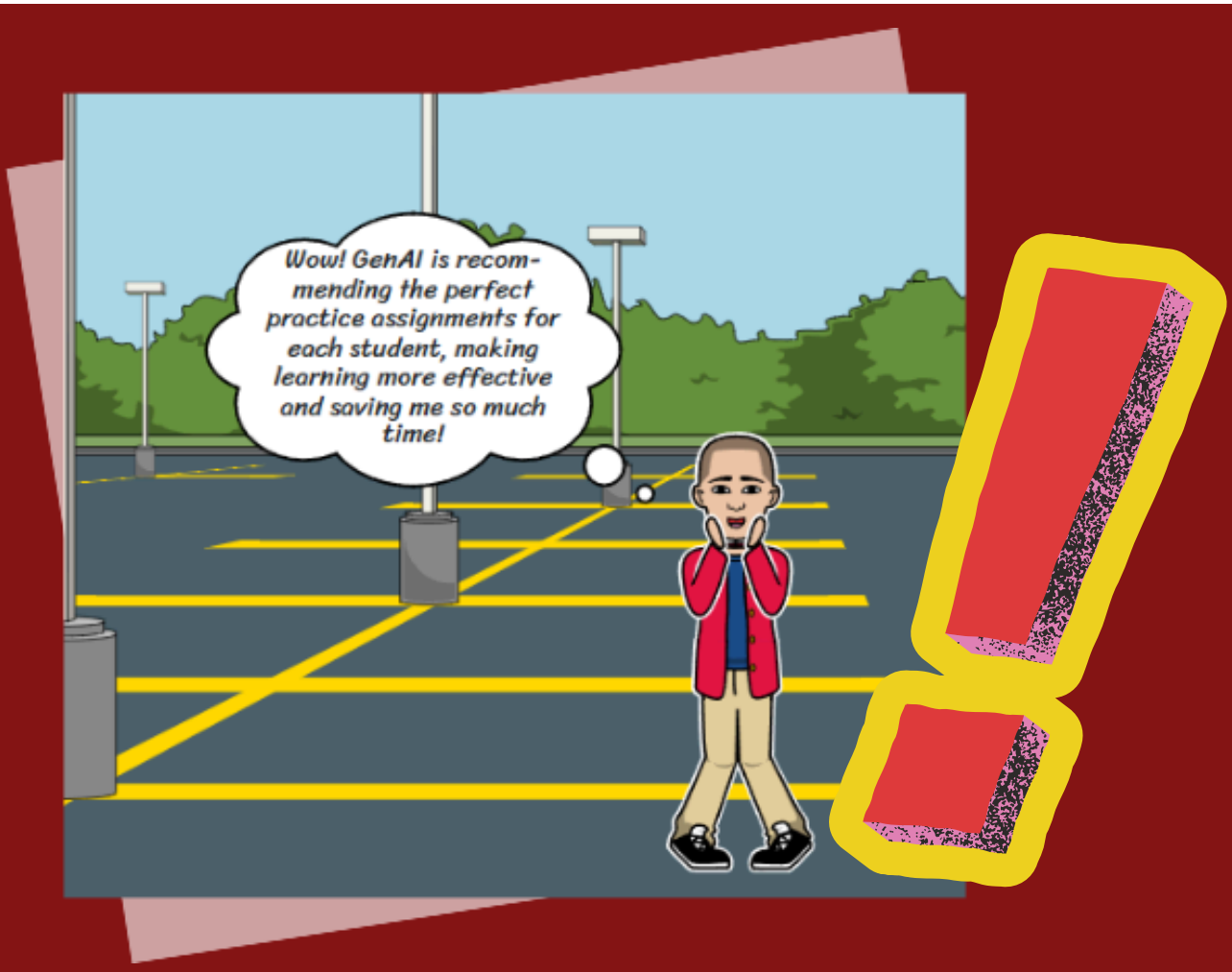
HOME → LOGIN → INSTRUCTOR DASHBOARD → ADD LESSONS → ADD ASSIGNMNETS


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
2.0 Home Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignments
Live Sessions
Add Lessons
Lecture name
Youtube Link
Description
Add Lesson
AI
2.1 Add Lessons Page
Instructor Dashboard
User Name
Logout
Resolve Query
Add Lessons
Add Assignments
Live Sessions
Add Assignment
Assignment name
Assignment Link
Description
Graded
Practice
Add Assignment
AI
2.2 Add Assignment Page


I have to manually select practice assignments for students, and it's challenging to ensure everyone gets the right level of difficulty.

ENTERS!


GenAI

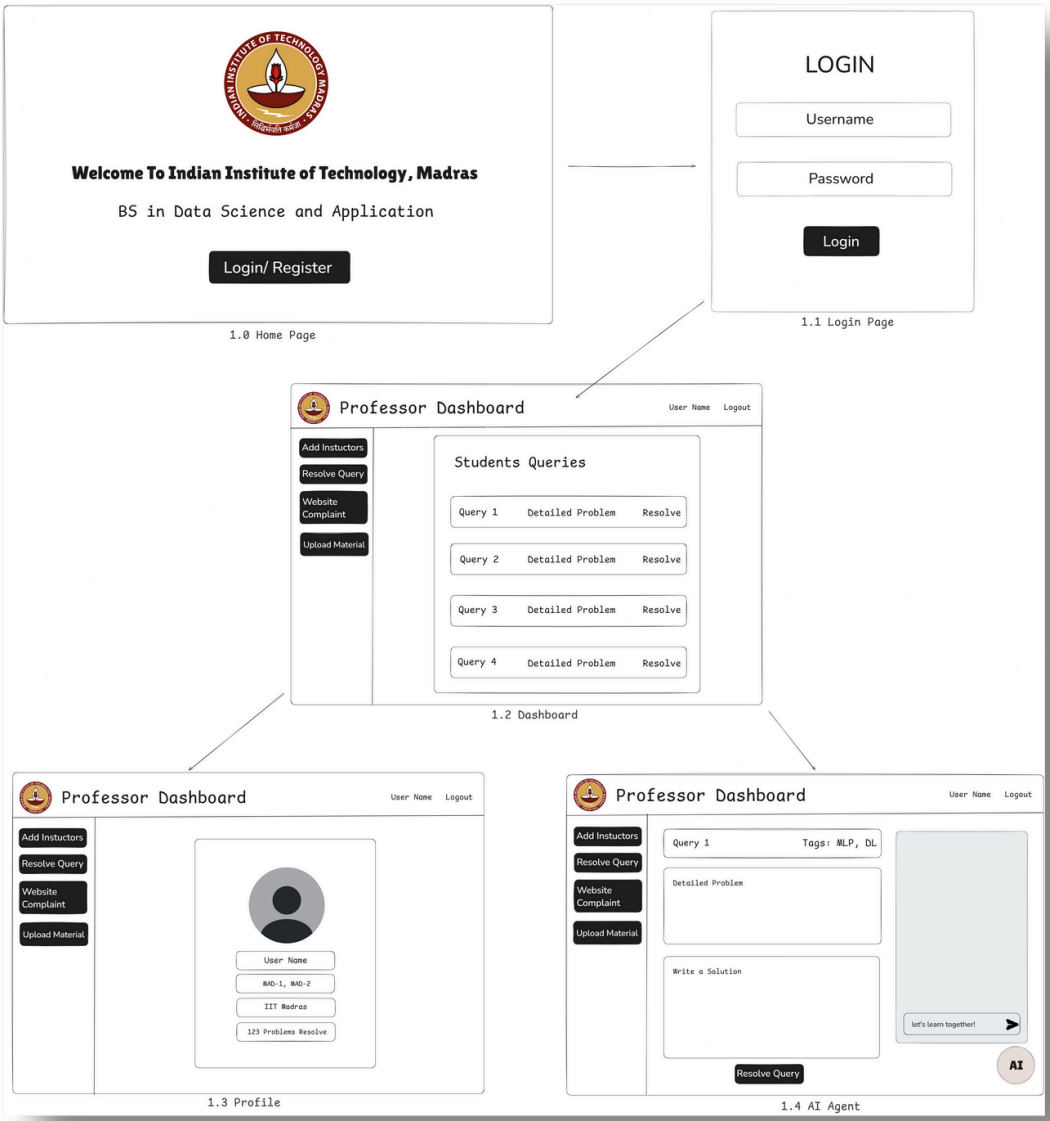

Wow! GenAI is recommending the perfect practice assignments for each student, making learning more effective and saving me so much time!

User Stories | Tertiary User | Scenario 1

Advanced Query Escalation

USERFLOW

HOME → LOGIN → PROFESSOR DASHBOARD → QUERIES → CHATBOT


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
LOGIN
Username
Password
Login
1.1 Login Page
Professor Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
Students Queries
Query 1 Detailed Problem Resolve
Query 2 Detailed Problem Resolve
Query 3 Detailed Problem Resolve
Query 4 Detailed Problem Resolve
1.2 Dashboard
Professor Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
User Name
MAD-1, MAD-2
IIT Madras
123 Problems Resolve
1.3 Profile
Professor Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
Query 1
Tags: MLP, DL
Detailed Problem
Write a Solution
Resolve Query
let's learn together!
AI
1.4 AI Agent


I'm spending a lot of time answering basic questions, leaving me with less time to engage in deeper academic discussions.

ENTERS!


GenAI


Wow! GenAI is handling all the basic queries, allowing me to focus on advanced discussions and provide deeper academic insights!

User Stories | Tertiary User | Scenario 2

Complaint Resolution Suggestions

USERFLOW

HOME → LOGIN → ADMIN DASHBOARD → WEBSITE COMPLAINT → RESOLVE QUERY


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Admin Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
Complaint About Website
Query 1 Detailed Problem Resolve
Query 2 Detailed Problem Resolve
Query 3 Detailed Problem Resolve
Query 4 Detailed Problem Resolve
1.1 Home Page
Admin Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
Query 1
Tags: Website
Detailed Problem
Write a Solution
Resolve Query
let's learn together!
AI
1.2 AI Agent


I'm handling the same student complaints re-peatedly, and finding the right solutions takes a lot of time.

ENTERS!


GenAI

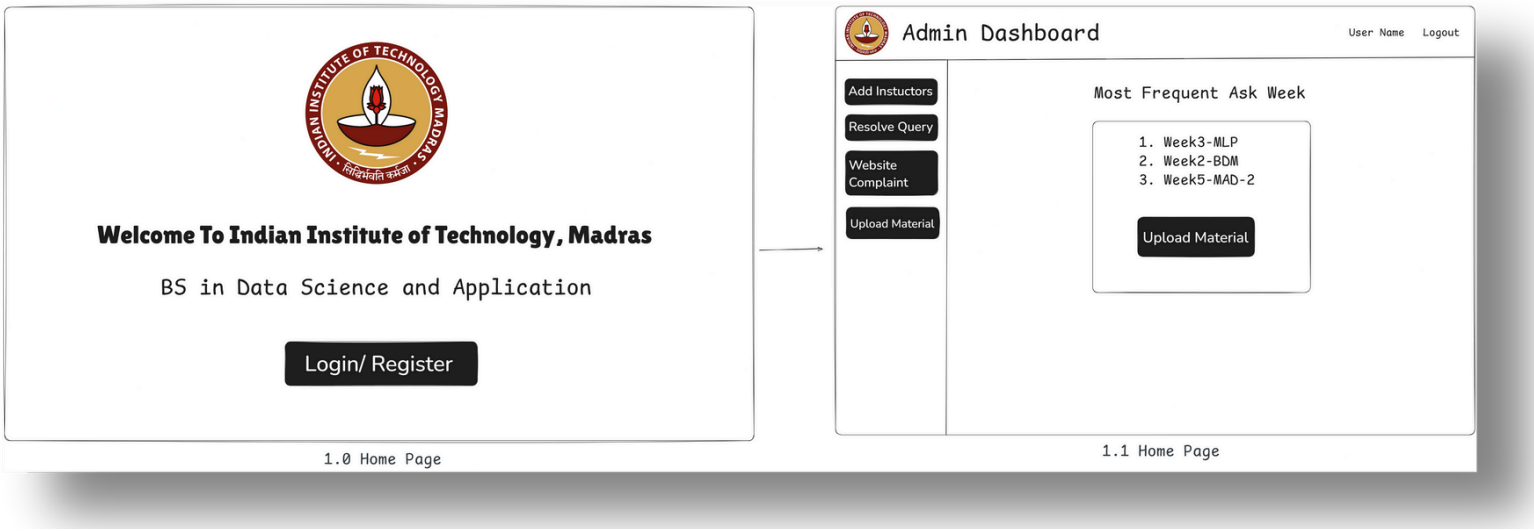

Wow! GenAI is suggesting solutions for common complaints, making it so much easier to resolve issues quickly and efficiently!

User Stories | Tertiary User | Scenario 3

Content Uploading

USERFLOW

HOME → LOGIN → ADMIN DASHBOARD → UPLOAD MATERIAL


Welcome To Indian Institute of Technology, Madras
BS in Data Science and Application
Login/ Register
1.0 Home Page
Admin Dashboard
User Name
Logout
Add Instuctors
Resolve Query
Website Complaint
Upload Material
Most Frequent Ask Week
1. Week3-MLP
2. Week2-BDM
3. Week5-MAD-2
Upload Material
1.1 Home Page


Uploading and organizing new academic materials manually takes a lot of time, and ensuring everything is structured properly is challenging.

ENTERS !


GenAI


Wow! GenAI made uploading and organizing academic materials so much easier. Now students can access updates instantly and without any confusion!

SCHEDULING AND DESIGN

PROJECT SCHEDULE

1. Sprint Schedule

Sprint No.	Topic/Task	Timeline
Sprint 1	Identifying User & Requirements	14 Jan, 2024 - 28 Jan, 2024
Sprint 2	User Interfaces	29 Jan, 2024 - 9 Feb, 2024
Sprint 3	Scheduling and Design	10 Feb, 2024 - 20 Feb, 2024
Sprint 4	API Endpoints	24 Feb, 2024 - 2 Mar, 2024
Sprint 5	Testing All the Test Cases	3 Mar, 2024 - 12 Mar, 2024
Sprint 6	Integrate Frontend & Backend	19 Mar, 2024 - 30 Mar, 2024

2. About the Sprints

- **Sprint 1 :- Identifying User & Requirements**

Identify and categorize users (primary, secondary, and tertiary) by analyzing their roles and interactions within the system. Gather inputs from key stakeholders to ensure comprehensive requirement collection.

Develop detailed user stories for new features and integrations, following SMART guidelines to ensure clarity, feasibility, and alignment with project goals. Complete Milestone 1 Report Vetting and Submission.

Dates: 14 Jan, 2024 - 28 Jan, 2024

- **Sprint 2 :- User Interfaces**

Develop a storyboard to illustrate the user journey and create low-fidelity wireframes for user stories, applying usability guidelines and heuristics, and complete Milestone 2 Report Vetting and Submission.

Dates: 29 Jan, 2024 - 9 Feb, 2024

- **Sprint 3 :- Scheduling and Design**

Develop a project schedule, including sprint timelines, scrum meeting plans, task distribution, and Trello Board, or Gantt charts using Jira tool.

Design system components, create basic class diagrams, develop most UI pages, and document scrum meeting details. Complete Milestone 3 Report Vetting and Submission.

Dates: 10 Feb, 2024 - 20 Feb, 2024

- **Sprint 4 :- API Endpoints**

Design and implement API architecture by creating new endpoints and integrating relevant APIs based on user stories.

Document API details, including integrated and custom-built endpoints, and submit YAML specifications along with implementation code. Complete Milestone 4 Report Vetting and Submission.

Dates: 24 Feb, 2024 - 2 Mar, 2024

- **Sprint 5 :- Testing All the Test Cases**

Design comprehensive test cases for each API endpoint, covering inputs, expected outputs, actual results, and success/failure validation.

Perform unit testing to ensure functionality, performance, and reliability of the system. Complete Milestone 5 Report Vetting and Submission.

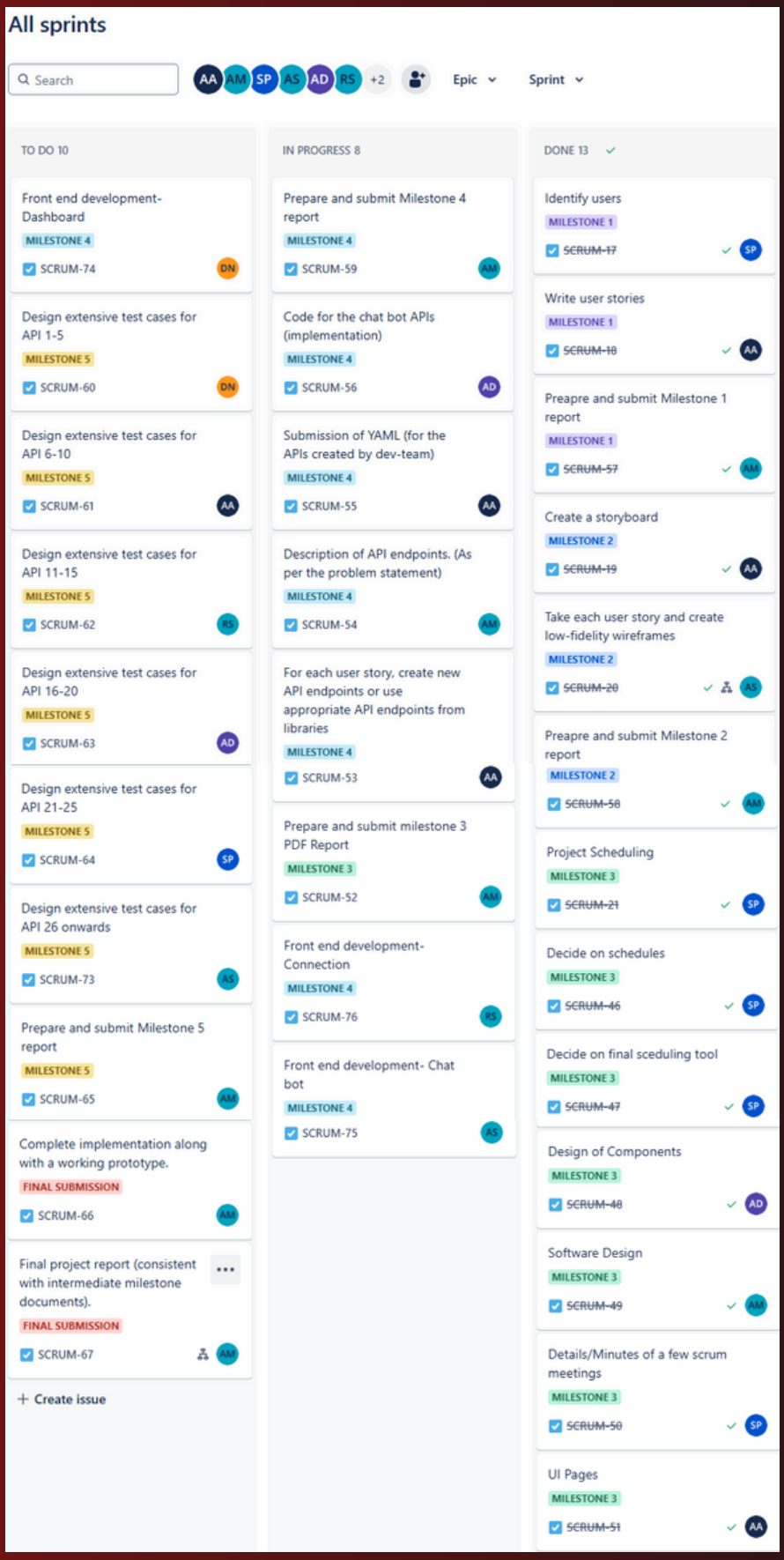
Dates: 3 Mar, 2024 - 12 Mar, 2024

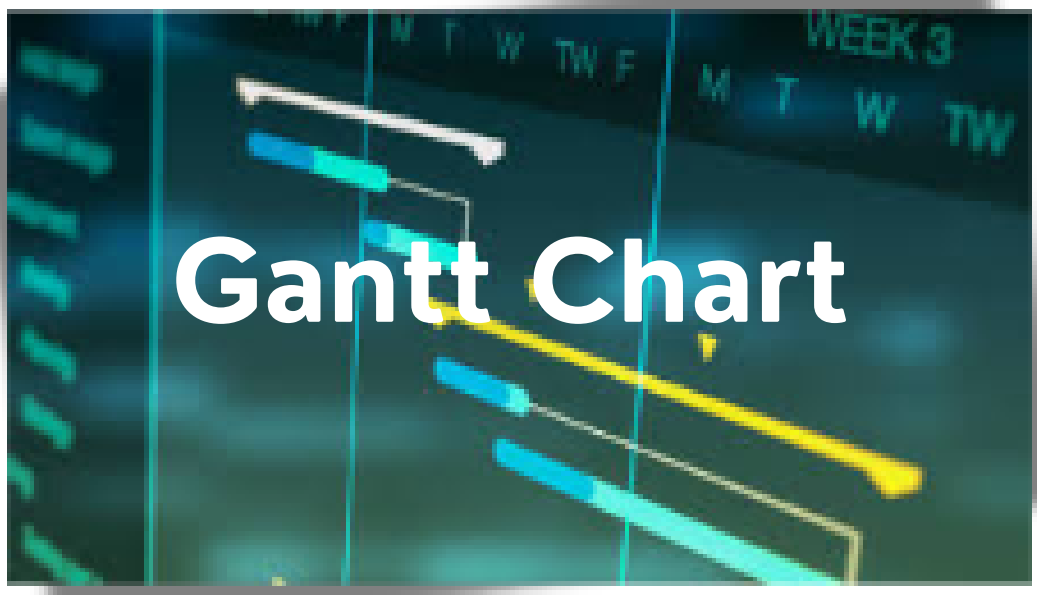
- **Sprint 6 :- Integrate Frontend & Backend**

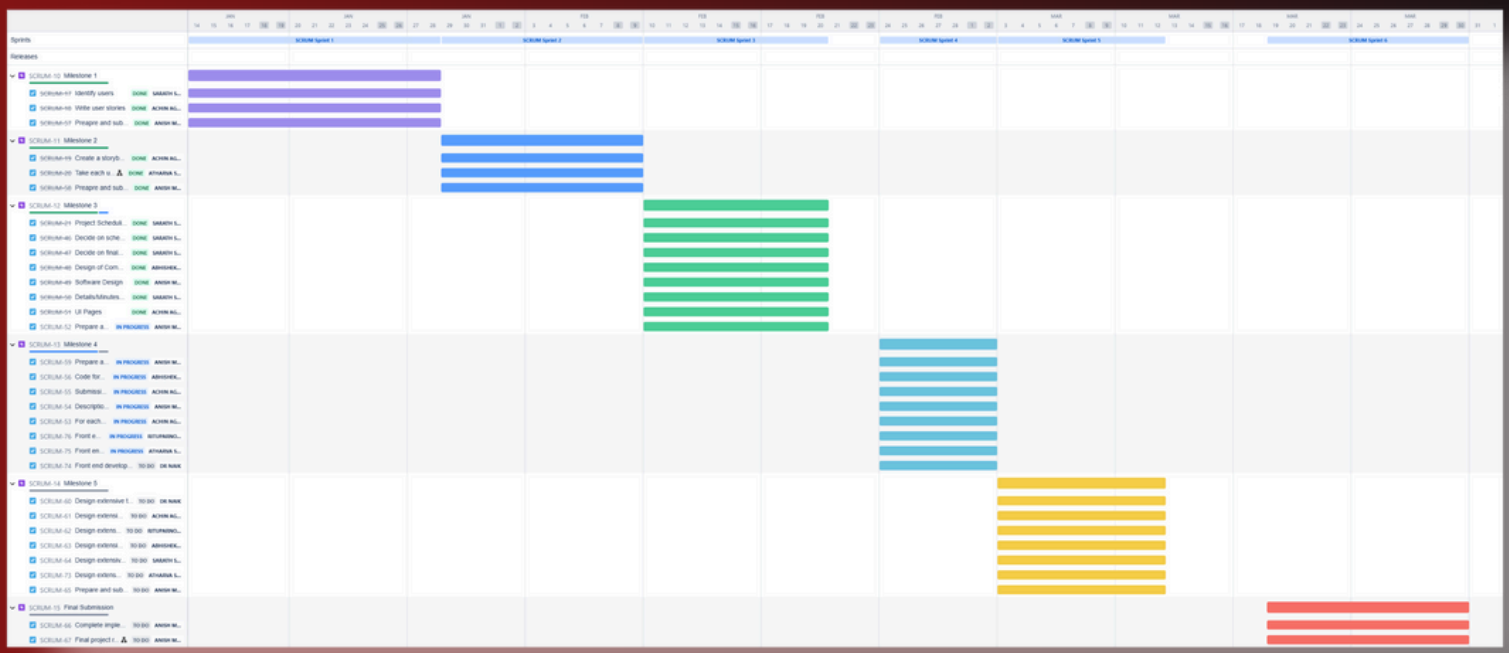
Combine frontend and backend components into a cohesive system. Complete Milestone 6 Report Vetting and Submission.

Dates: 19 Mar, 2024 - 30 Mar, 2024

SCRUM BOARD

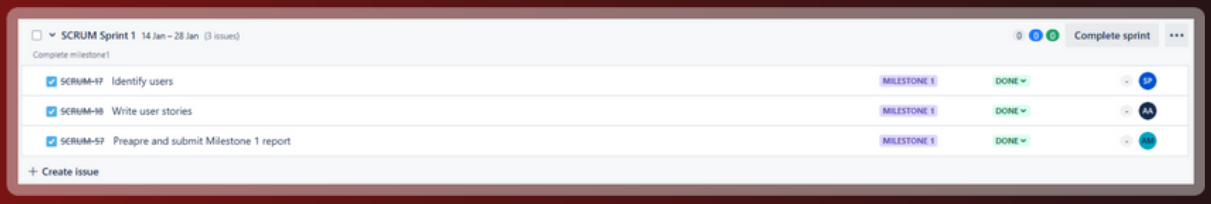

All sprints
Search
Epic
Sprint
TO DO 10
Front end development- Dashboard
MILESTONE 4
SCRUM-74
Design extensive test cases for API 1-5
MILESTONE 5
SCRUM-60
Design extensive test cases for API 6-10
MILESTONE 5
SCRUM-61
Design extensive test cases for API 11-15
MILESTONE 5
SCRUM-62
Design extensive test cases for API 16-20
MILESTONE 5
SCRUM-63
Design extensive test cases for API 21-25
MILESTONE 5
SCRUM-64
Design extensive test cases for API 26 onwards
MILESTONE 5
SCRUM-73
Prepare and submit Milestone 5 report
MILESTONE 5
SCRUM-65
Complete implementation along with a working prototype.
FINAL SUBMISSION
SCRUM-66
Final project report (consistent with intermediate milestone documents).
FINAL SUBMISSION
SCRUM-67
Create issue
IN PROGRESS 8
Prepare and submit Milestone 4 report
MILESTONE 4
SCRUM-59
Code for the chat bot APIs (implementation)
MILESTONE 4
SCRUM-56
Submission of YAML (for the APIs created by dev-team)
MILESTONE 4
SCRUM-55
Description of API endpoints. (As per the problem statement)
MILESTONE 4
SCRUM-54
For each user story, create new API endpoints or use appropriate API endpoints from libraries
MILESTONE 4
SCRUM-53
Prepare and submit milestone 3 PDF Report
MILESTONE 3
SCRUM-52
Front end development- Connection
MILESTONE 4
SCRUM-76
Front end development- Chat bot
MILESTONE 4
SCRUM-75
DONE 13
Identify users
MILESTONE 1
SCRUM-17
Write user stories
MILESTONE 1
SCRUM-18
Preapre and submit Milestone 1 report
MILESTONE 1
SCRUM-57
Create a storyboard
MILESTONE 2
SCRUM-19
Take each user story and create low-fidelity wireframes
MILESTONE 2
SCRUM-20
Preapre and submit Milestone 2 report
MILESTONE 2
SCRUM-58
Project Scheduling
MILESTONE 3
SCRUM-21
Decide on schedules
MILESTONE 3
SCRUM-46
Decide on final sceduling tool
MILESTONE 3
SCRUM-47
Design of Components
MILESTONE 3
SCRUM-48
Software Design
MILESTONE 3
SCRUM-49
Details/Minutes of a few scrum meetings
MILESTONE 3
SCRUM-50
UI Pages
MILESTONE 3
SCRUM-51

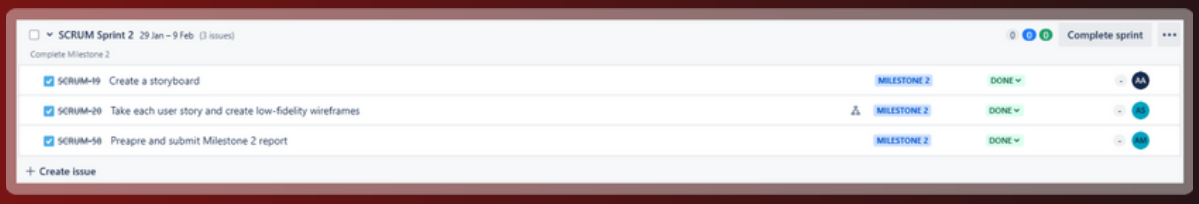

WEEK 3
Gantt Chart

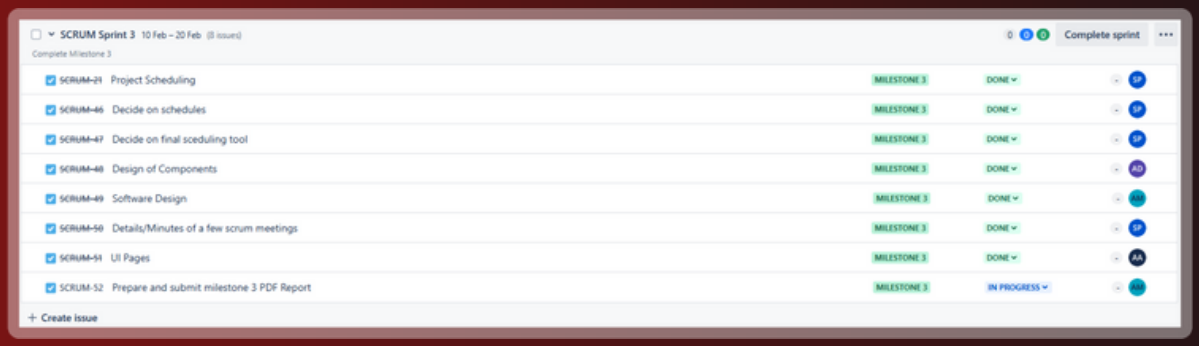


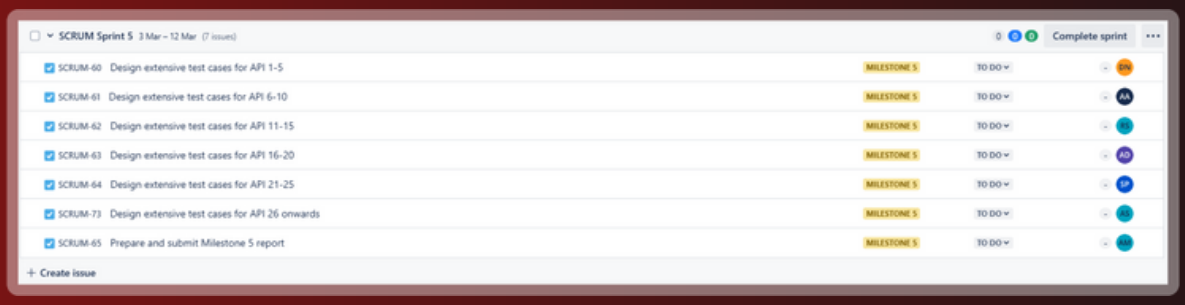
Project Scheduling

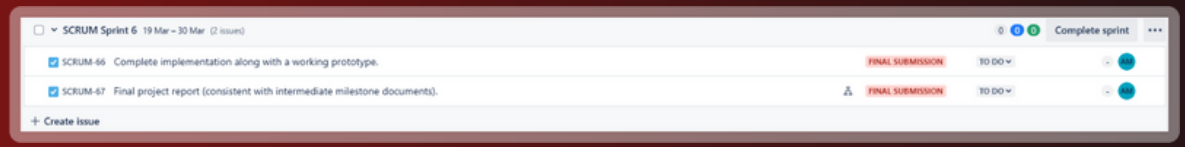
Tool - Jira











Scrum Meeting

Minutes/Details

Scrum Meetings Schedule: Every Monday, Saturday (21:00 - 23:30 PM)
Mode: Google Meet
Attendees:

- Achin Aggarwal
- Anish Maity
- Atharva Sarbhukan
- Abhishek Darji
- Sarath Sasidharan Pillai
- Rituparno Sen
- Dr. Ambrish Naik

- **Sprint 1 Scrum meetings minutes/details:**

The team will focus on identifying and defining different types of users (primary, secondary, and tertiary) based on their roles and interactions within the system. Each team member will contribute by gathering insights and inputs from key stakeholders to ensure a thorough requirement collection.

Once the users are identified, the team will divide tasks to create detailed user stories for the new features and integrations, following SMART guidelines for clarity and feasibility.

In the meeting, we will collaboratively review and finalize the user stories, ensuring alignment with the project goals. Additionally, all members will participate in reviewing the Milestone 1 Report. Once the report is finalized, the team will complete the vetting process and proceed with the final submission.

Dates: 14 Jan, 2024 - 28 Jan, 2024

- **Sprint 2 Scrum meetings minutes/details:**

The team will collaborate to develop a storyboard illustrating the user journey and create low-fidelity wireframes for each user story. Each member will contribute ideas and suggestions to ensure the wireframes align with usability guidelines and heuristics.

A discussion will take place to review the storyboard and wireframes, where feedback will be provided for improvements. The team will finalize the designs based on the discussion and ensure they meet project requirements.

After the final review, the Milestone 2 Report will be vetted and submitted.

Dates: 29 Jan, 2024 - 9 Feb, 2024

- **Sprint 3 Scrum meetings minutes/details:**

The team will collaborate to finalize the project schedule, including sprint timelines, scrum meeting plans, and task distribution. Project management tools like Jira, Trello, or Gantt charts will be used to track progress effectively.

A discussion will be held to design key system components and create basic class diagrams to define system structure. Additionally, most UI pages will be developed, ensuring alignment with project requirements.

After the final review, the Milestone 3 Report will be vetted and submitted.

Dates: 10 Feb, 2024 - 20 Feb, 2024

- **Sprint 4 Scrum meetings minutes/details:**

The team will collaborate to design and implement the API architecture, ensuring that new endpoints are created and relevant APIs are integrated based on user stories. Each team member will contribute to defining the API structure and functionality.

A discussion will take place to document API details, including integrated and custom-built endpoints, along with YAML specifications. The team will then proceed with the development and deployment of the designed APIs.

Following the final review, the Milestone 4 Report will be vetted and submitted.

Dates: 24 Feb, 2024 - 2 Mar, 2024

- **Sprint 5 Scrum meetings minutes/details:**

The team will collaborate to design and document comprehensive test cases for each API endpoint, ensuring functionality, performance, and reliability. Each test case will include inputs, expected outputs, actual results, and success/failure validation.

Unit testing will be performed to verify the quality and robustness of the system. Any identified issues will be addressed through debugging and re-testing.

After the final validation, the Milestone 5 Report will be vetted and submitted.

Dates: 3 Mar, 2024 - 12 Mar, 2024

- **Sprint 6 Scrum meetings minutes/details:**

The team will collaborate to integrate the frontend and backend components, ensuring seamless communication between the UI and the API endpoints. Each module will be tested to verify proper data flow and functionality.

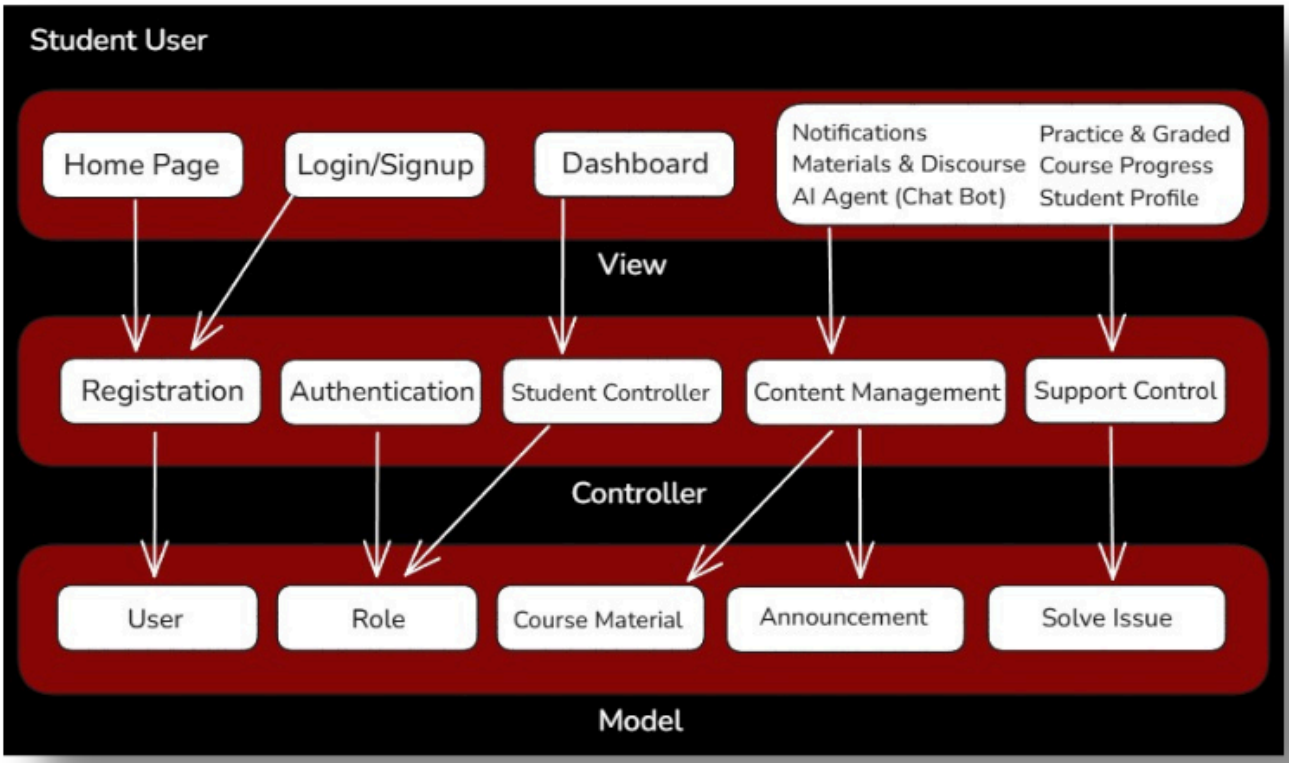
A discussion will be held to address any integration issues and optimize performance for a smooth user experience. Necessary debugging and refinements will be made to ensure stability.

Following successful integration, the Milestone 6 Report will be vetted and submitted.

Dates: 19 Mar, 2024 - 30 Mar, 2024

COMPONENT DESIGN

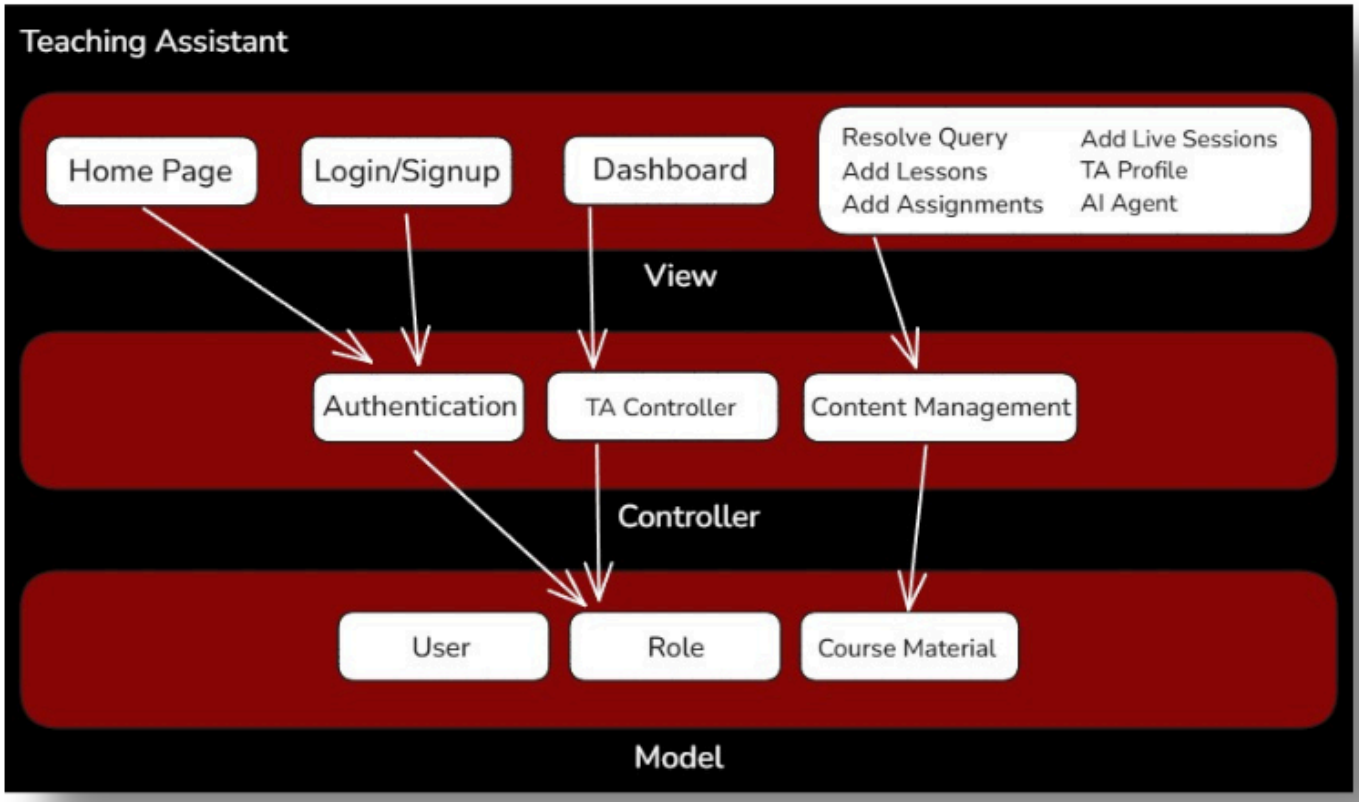
1. Student Component



- **Home Page:** Acts as the entry point, connecting to Registration and Authentication controllers for access management.
- **Login/Signup:** Handles user authentication via the Authentication Controller, interacting with the User and Role models.
- **Dashboard:** Displays student-related data, managed by the Student Controller and fetching information from Course Material and User models.
- **Registration & Authentication:** Manages user sign-ups and logins, ensuring security through the User and Role models.
- **Student Controller:** Oversees student interactions, retrieving study resources and tracking progress from the Course Material model.
- **Content Management:** Handles learning materials, discussions, and announcements, updating the Announcement model as needed.
- **Support Control:** Manages issue reporting and resolution by interacting with the Solve Issue model.

COMPONENT DESIGN

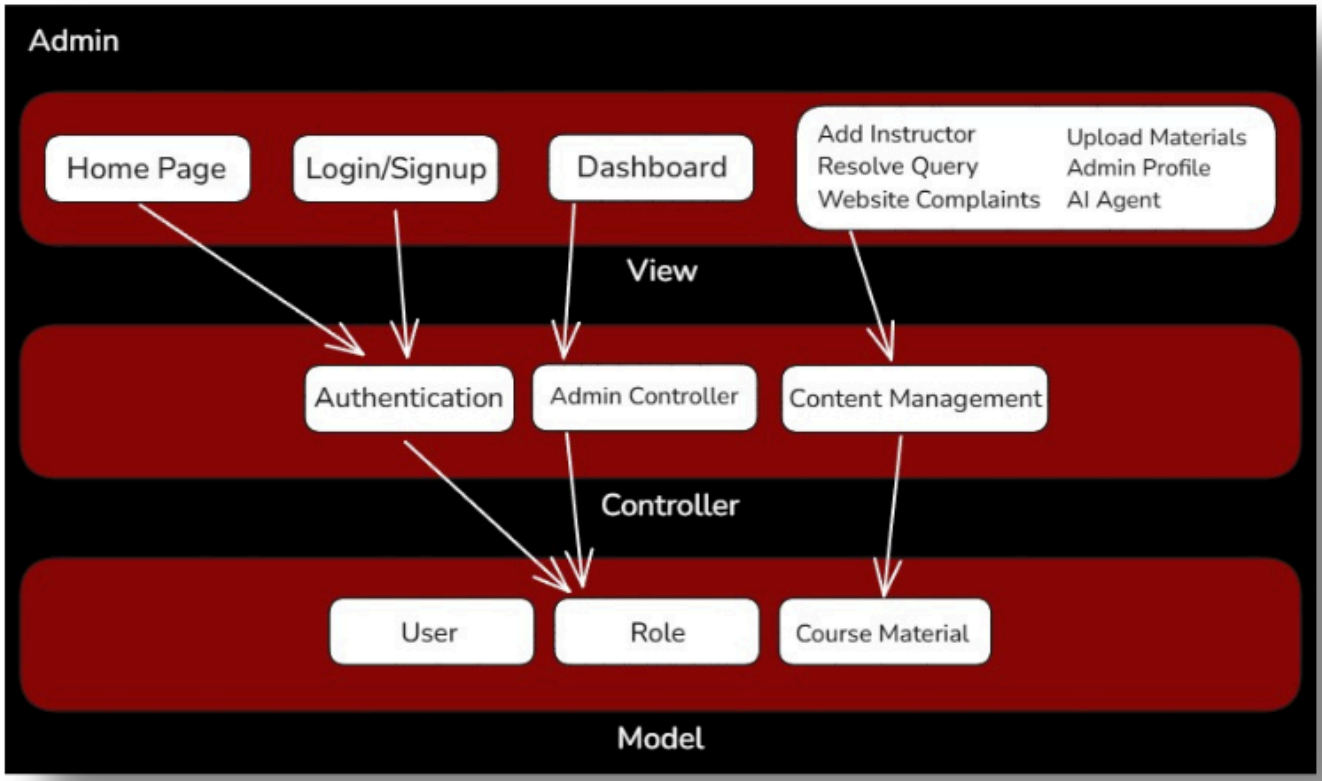
2. Teaching Assistant Component



- **Home Page:** Serves as the entry point, directing users to authentication and dashboard functionalities.
- **Login/Signup:** Manages TA authentication through the Authentication Controller, interacting with the User and Role models.
- **Dashboard:** Provides access to query resolution, lesson creation, assignments, live sessions, and TA profile via the TA Controller.
- **Authentication:** Ensures secure login and role-based access control using the User and Role models.
- **TA Controller:** Manages TA-specific actions such as resolving queries, adding lessons, and handling assignments.
- **Content Management:** Oversees course materials, assignments, and live sessions, interacting with the Course Material model.

COMPONENT DESIGN

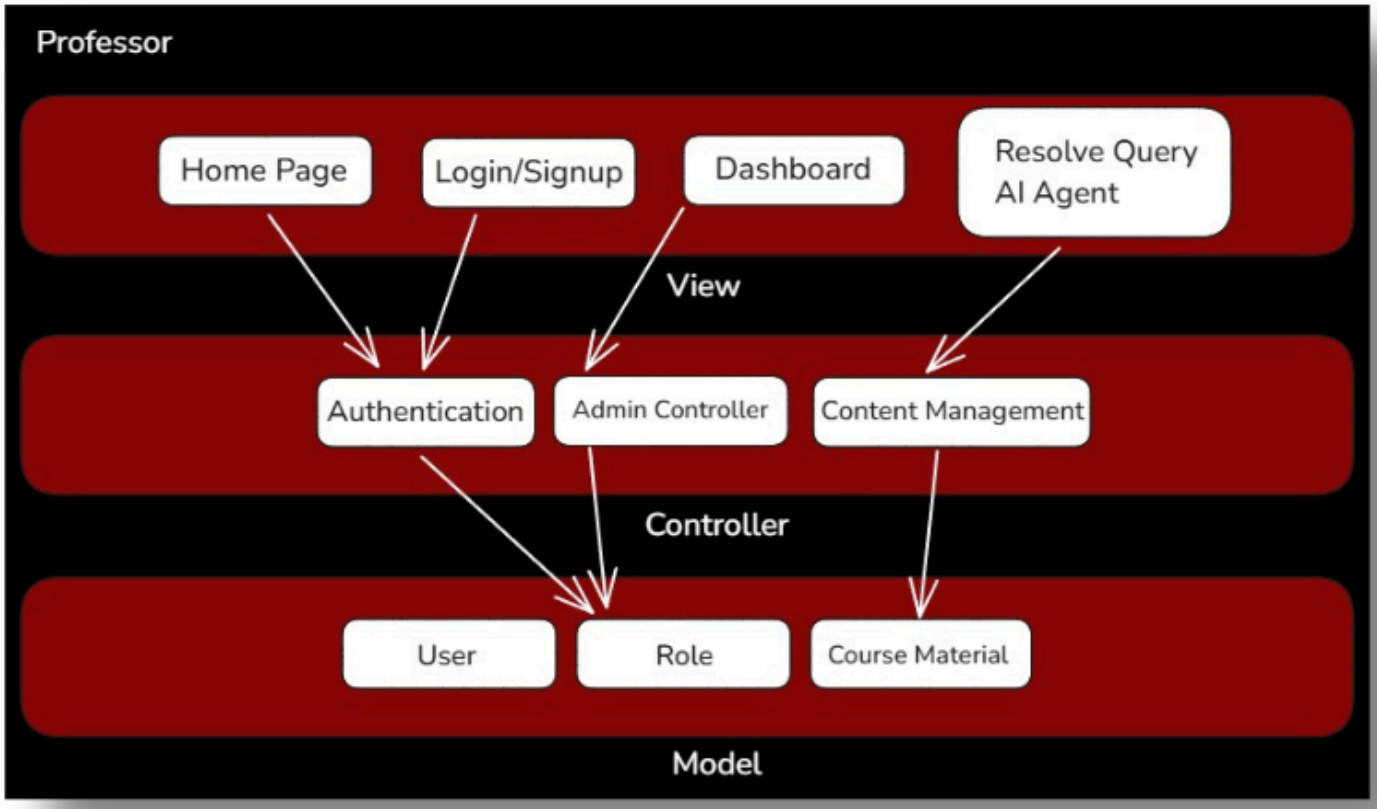
3. Admin Component



- **Home Page:** Entry point directing admins to authentication.
- **Login/Signup:** Manages secure admin authentication and access control.
- **Dashboard:** Provides an overview of admin functions like user management and AI interactions.
- **Authentication:** Ensures secure login and role-based access.
- **Admin Controller:** Handles instructor management, queries, and complaints.
- **Content Management:** Manages course materials and updates.
- **User Model:** Stores admin data and credentials.
- **Role Model:** Defines admin permissions and access levels.
- **Course Material Model:** Stores and retrieves uploaded materials.

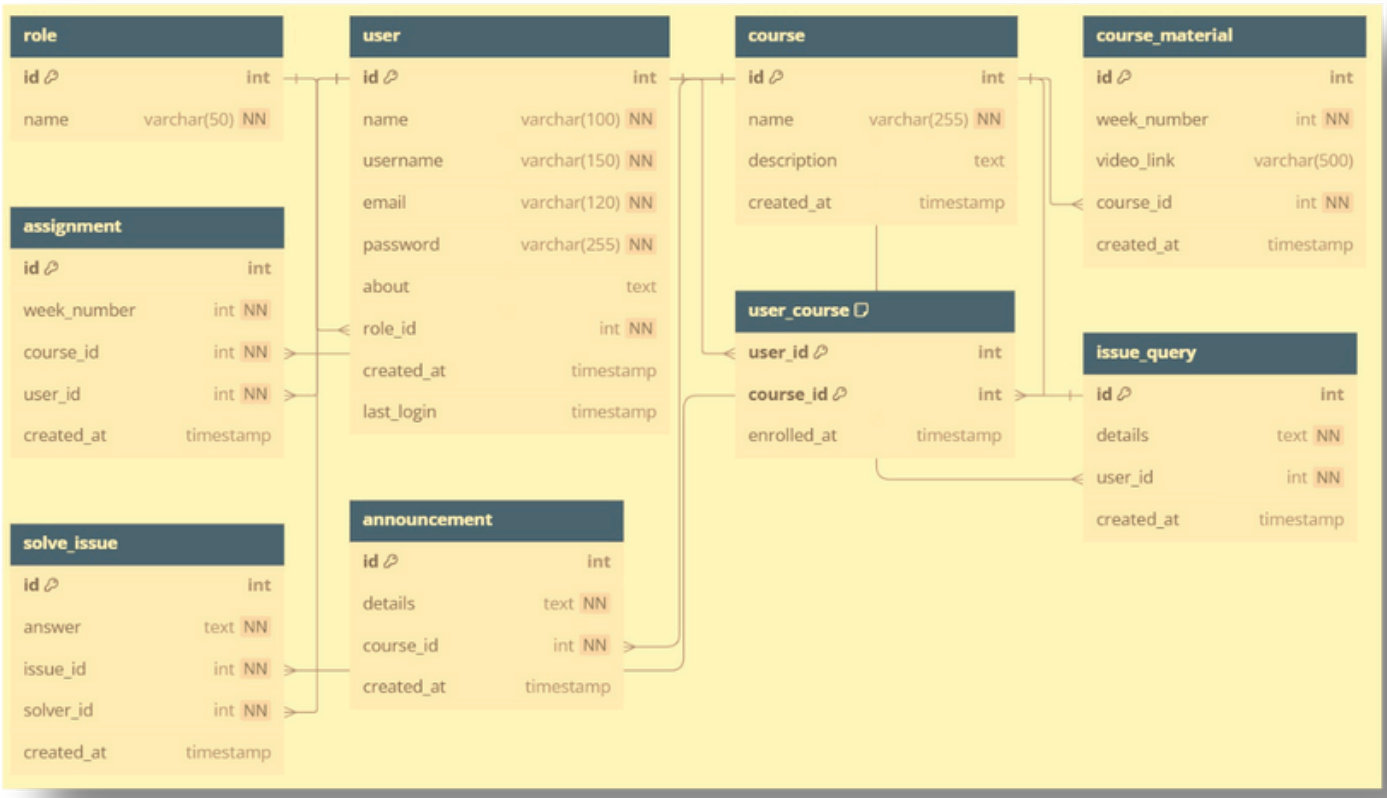
COMPONENT DESIGN

4. Professor Component



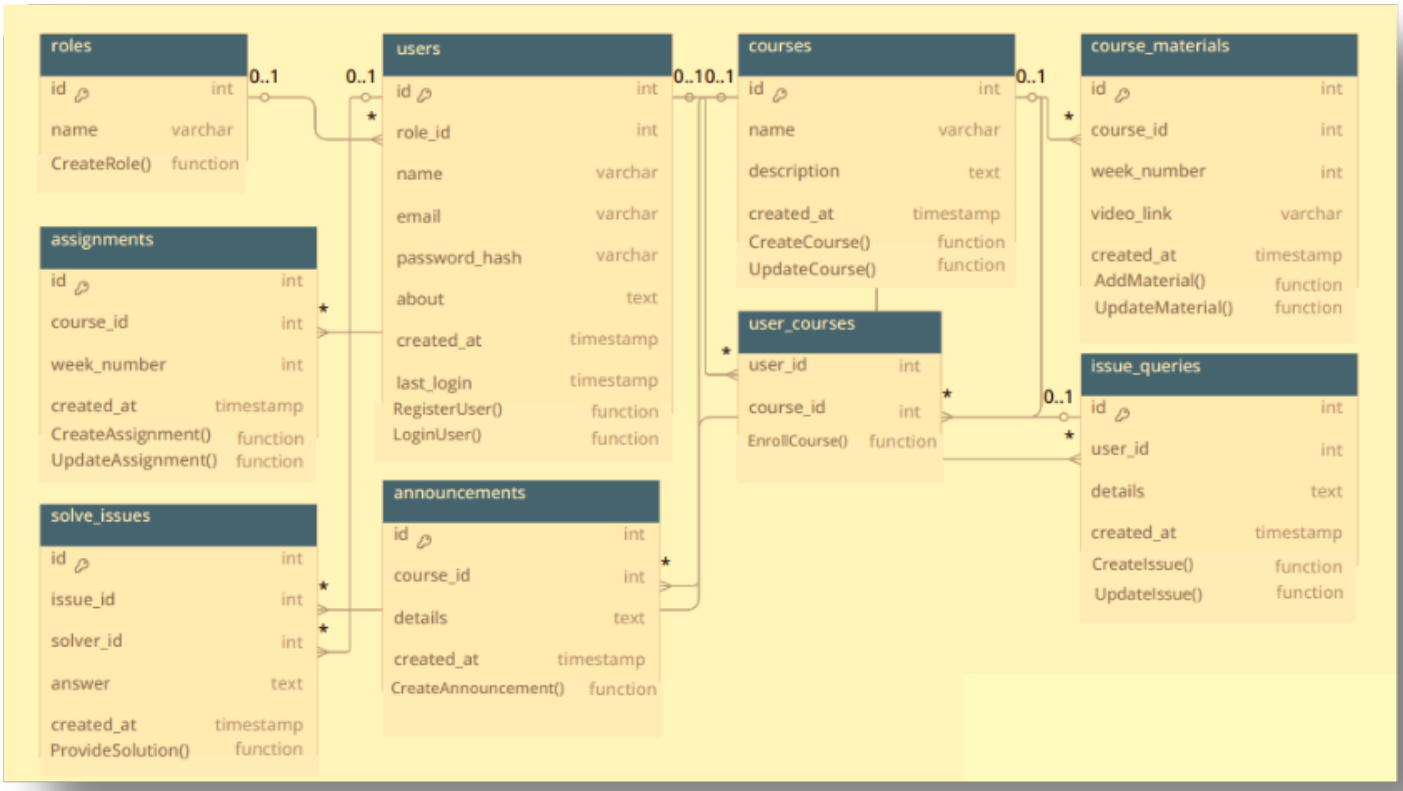
- **Home Page:** Entry point directing professors to authentication.
- **Login/Signup:** Manages secure professor authentication and access control.
- **Dashboard:** Provides an overview of teaching resources and interactions.
- **Resolve Query:** Allows professors to address student inquiries.
- **AI Agent:** Assists with automated responses and support.
- **Authentication:** Ensures secure login and role-based access.
- **Admin Controller:** Manages professor-related administrative tasks.
- **Content Management:** Handles course materials and academic resources.
- **User Model:** Stores professor data and credentials.
- **Role Model:** Defines permissions and teaching access levels.
- **Course Material Model:** Stores and retrieves learning materials.

DB Schema Design



This database schema is designed for an AI-driven academic guidance system, managing users, courses, and interactions between them. The **role** table defines different user roles, while the **user** table stores user details, including login credentials and role associations. The **course** table holds course information, and **course_material** links videos and materials to specific weeks. The **assignment** table tracks assignments submitted by users for each course. The **announcement** table stores course-related updates. The **user_course** table records user enrollments. The **issue_query** and **solve_issue** tables enable students to raise academic queries and receive solutions. Relationships between tables ensure efficient tracking of progress, resources, and interactions, forming a structured foundation for an AI agent to provide academic guidance.

CLASS DIAGRAM



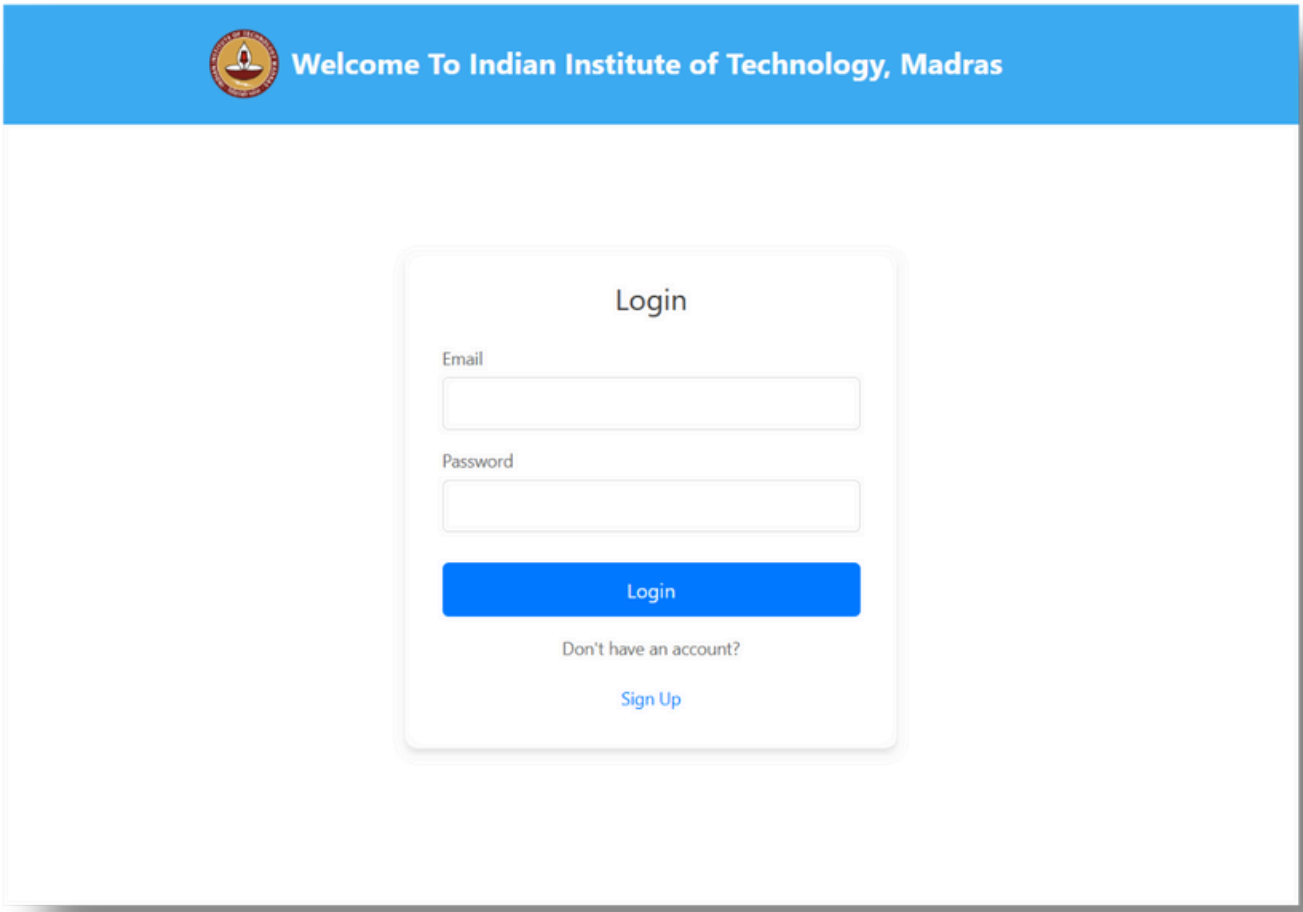
This class diagram represents the LMS database schema, including key entities such as Users, Roles, Courses, Assignments, Course Materials, Issue Queries, Solve Issues, and Announcements.

- **Users:** Manages user details and authentication.
- **Roles:** Defines permissions for different user types.
- **Courses:** Stores course details and enrollment.
- **Assignments:** Tracks assignment creation and updates.
- **Course Materials:** Manages learning resources.
- **Issue Queries:** Handles user-raised concerns.
- **Solve Issues:** Manages query resolution.
- **Announcements:** Stores course-related updates.

Entities are linked via foreign keys, ensuring structured data flow. Core functions like RegisterUser(), CreateCourse(), and ProvideSolution() enable smooth system operations.

UI PAGES

1. Login Page



The **Login Page** is a crucial entry point for users accessing the **Indian Institute of Technology, Madras** online platform. It ensures secure authentication by verifying user credentials before granting access to personalized dashboards.

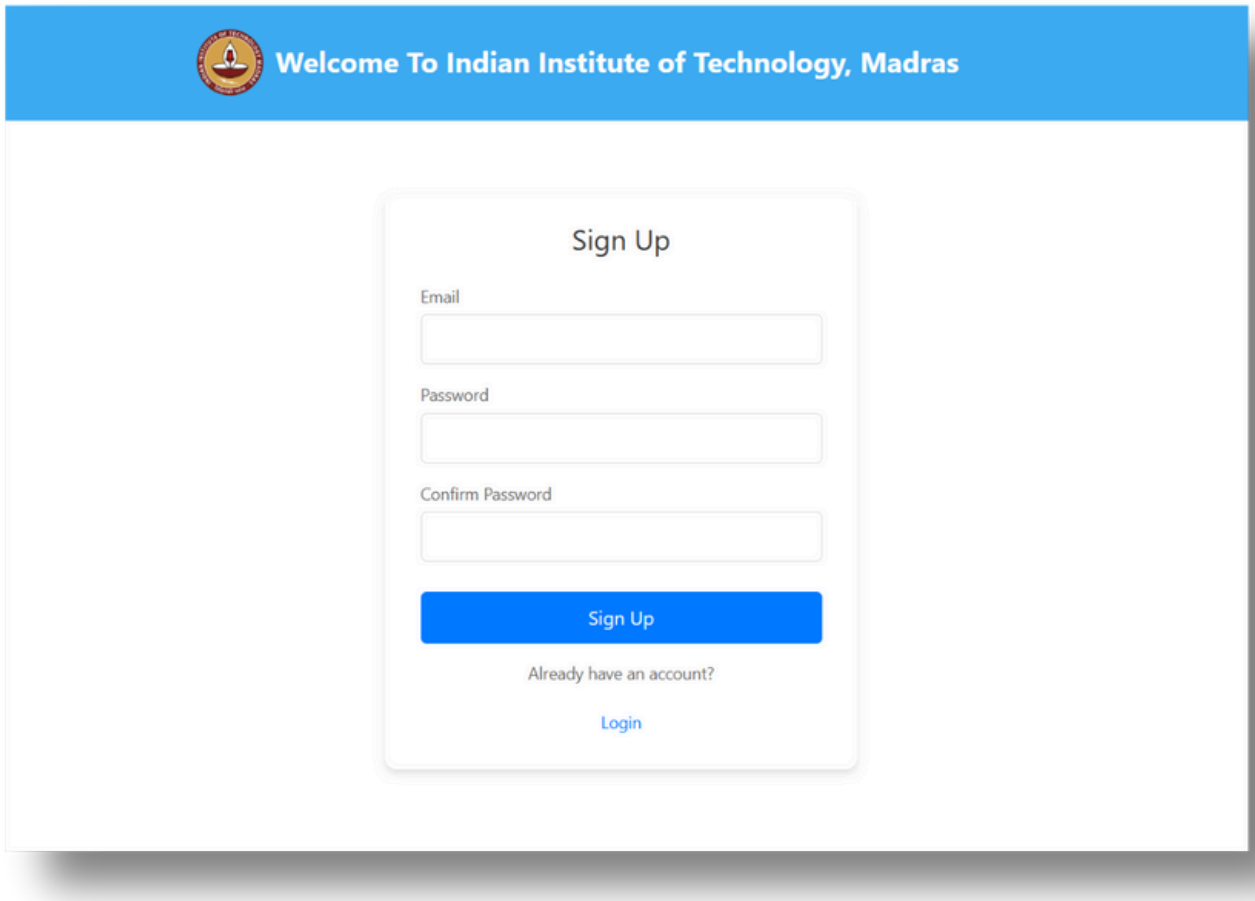
Key Features:

- **Institution Branding:** Displays the IIT Madras logo and welcome text.
- **User Authentication:** Requires Email and Password for secure login.
- **Call to Action:** A prominent Login button for user access.
- **New User Registration:** A Sign Up link for users without an existing account.

This page enhances usability and security, ensuring a seamless experience for students and faculty members.

UI
PAGES

2. Sign Up Page



The Sign-Up Page allows new users to create an account on the Indian Institute of Technology, Madras online platform. It ensures secure user registration before accessing the system.

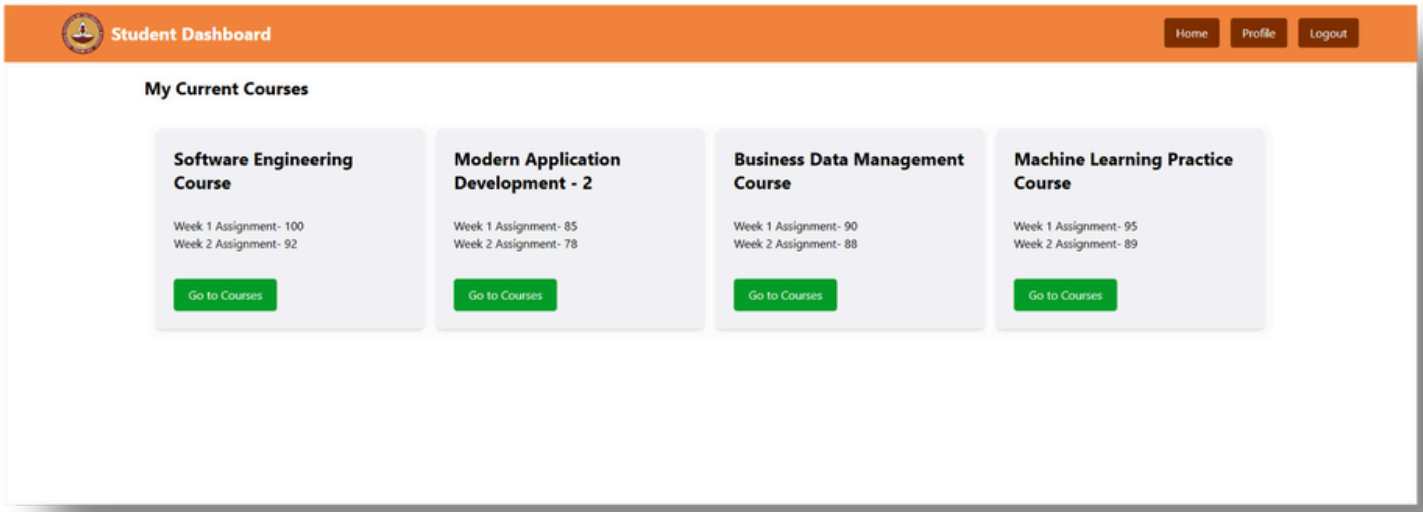
Key Features:

- Institution Branding: Displays the IIT Madras logo and welcome text.
- User Registration: Requires Email, Password, and Confirm Password fields.
- Call to Action: A prominent Sign-Up button to complete the registration.
- Existing User Redirect: A Login link for users who already have an account.

This page facilitates easy and secure user onboarding, ensuring a smooth registration process.

UI
PAGES

3. Student Dashboard Page



The Student Dashboard provides an organized view of the enrolled courses and recent assignment scores. It serves as the central hub for students to manage their coursework efficiently.

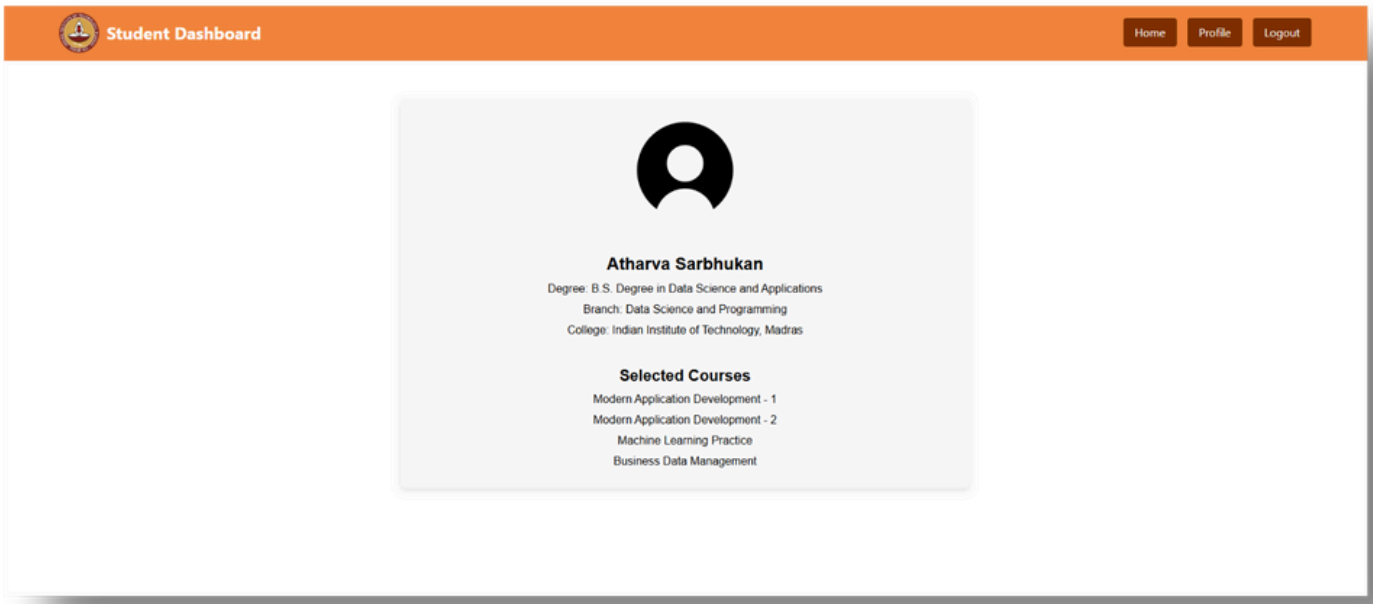
Key Features:

- **Institution Branding:** Displays the IIT Madras logo and a "Student Dashboard" heading.
- **Course Overview:** Displays currently enrolled courses, Shows assignment scores for recent weeks, Provides a "Go to Courses" button for quick access.
- **Navigation Menu: Home -** Redirects to the homepage, **Profile -** Allows students to view/edit their profile details, **Logout -** Logs out the student from the system.

This dashboard ensures easy access to coursework, streamlined navigation, and performance tracking for students.

UI PAGES

4. Profile Page



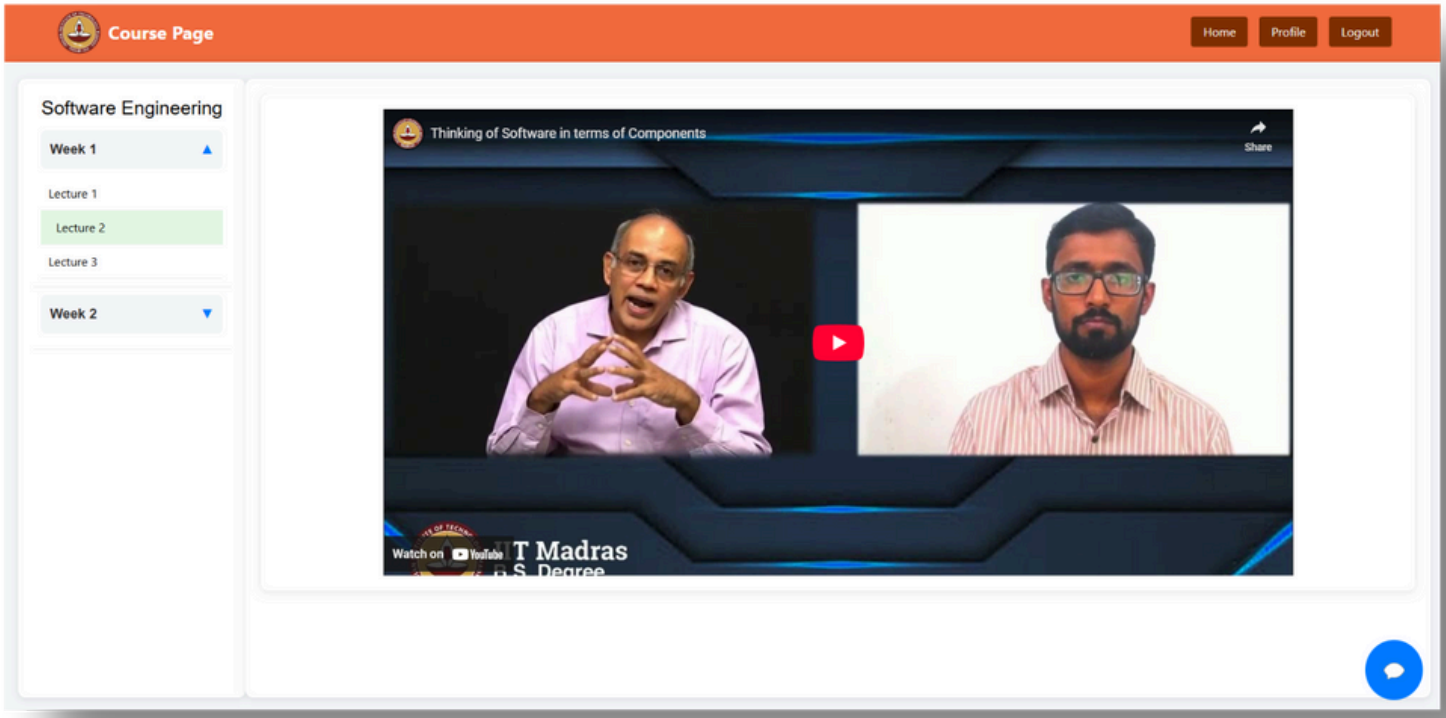
The **Profile Page** provides a summary of the student's academic details and enrolled courses. It acts as a personalized hub for users to view their information at a glance.

Key Features:

- **User Info:** Displays name, degree (B.S. Data Science), branch, and institution (IIT Madras).
- **Selected Courses:** Lists enrolled courses (e.g., Modern App Dev, ML Practice, Business Data Management).
- **Navigation:** Home, Profile, Logout for easy access.

UI
PAGES

5. Course Page



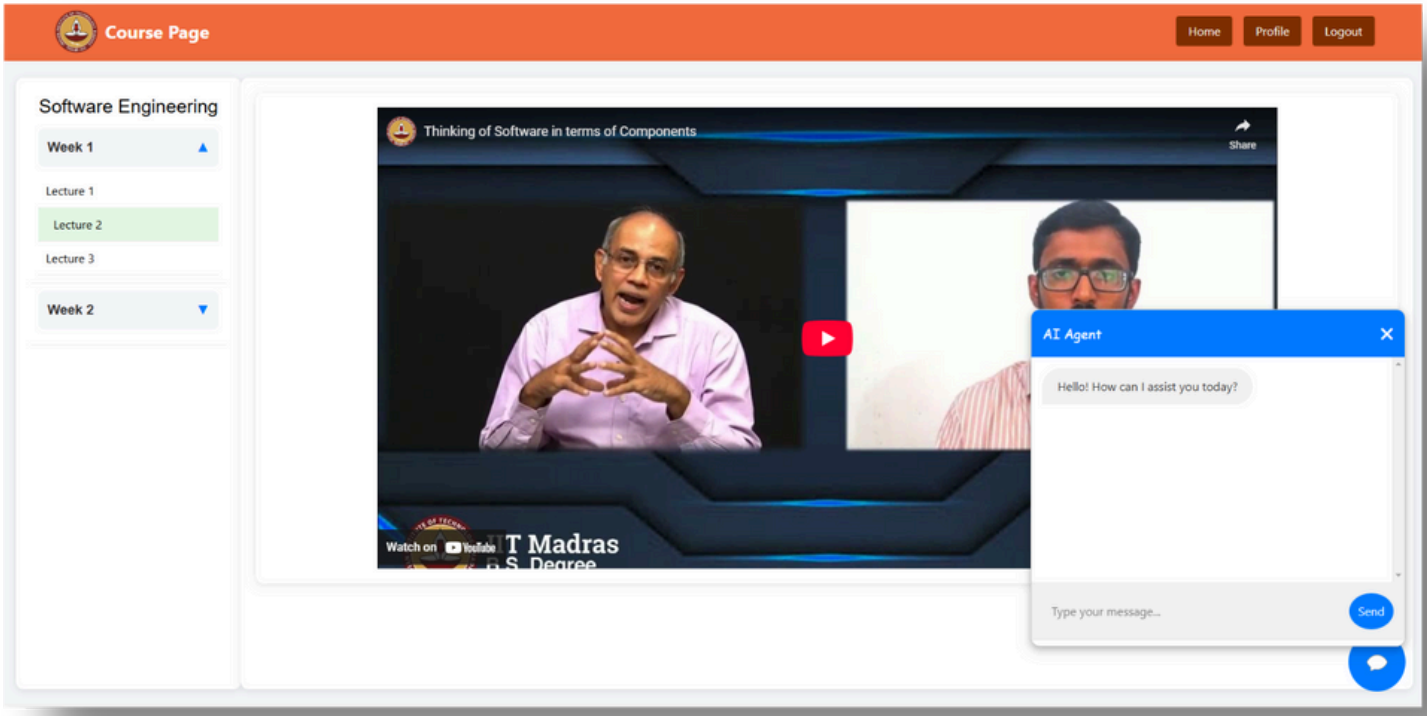
The course page provides an interactive platform for students to access lectures and learning materials efficiently.

Key Features:

- **Course Structure:** The left sidebar displays the course title "Software Engineering" with collapsible sections for weeks and lectures, Users can navigate between weeks and lectures easily.
- **Lecture Video:** The main section features an embedded YouTube lecture titled "Thinking of Software in Terms of Components" with two instructors, Users can play, pause, or expand the video.
- **User Interface:** Sidebar Navigation for switching lectures, Top Navigation Bar with Home, Profile, and Logout buttons, Chat Support Button for assistance.

UI
PAGES

6. Course ChatBot Page



This Course ChatBot page provides a structured layout for easy navigation and learning.

Key Features:

- **Course Navigation:** Sidebar with collapsible sections for weeks and lectures, Users can switch between topics easily.
- **Lecture Video Player:** Embedded YouTube lecture for seamless viewing, Playback and fullscreen options available.
- **User Interface Enhancements:** Top Navigation: Home, Profile, Logout buttons.
- **AI Chatbot:** Provides instant assistance.

Frontend Source Code Link :-

https://drive.google.com/drive/folders/1WimCKGDVJy6ICqCFlTYRmSMxEqXf0lVD

API ENDPOINTS

API ENDPOINTS

Detailed Descriptions

1. Authentication

- **/signup :**

The Signup handles user registration by accepting details such as name, username, email, password, about, and role (either "Student" or "Instructor"). It validates that all required fields are provided, checks for duplicate usernames or emails, and ensures the specified role exists in the database. The password is securely hashed before being stored, and the user is added to the database with a timestamp for account creation. If the user is an instructor, a pending approval request is created. The method returns a success message based on the role, or an error message if any validation fails, such as missing fields or invalid roles.

- **/login :**

The Login handles user login by accepting the username (or email) and password. It first checks if the provided credentials match any user in the database. If no matching user is found or the password is incorrect, it returns an "Invalid credentials" message. For users with the "Instructor" role, the system checks if their approval is pending by looking up their status in the InstructorRequest table. If the instructor is not approved, the login is blocked with a message stating "Instructor approval pending." If the credentials are valid and the instructor is approved (if applicable), the method updates the user's last_login timestamp and generates both an access_token and a refresh_token for the user. The response includes these tokens along with the username and a success message indicating successful login.

- **/token_refresh :**

The RefreshToken allows users to obtain a new access_token using a valid refresh_token. The @jwt_required(refresh=True) decorator ensures that only requests with a valid refresh token can access this endpoint. Upon receiving the refresh_token, the method decodes it to extract the user's user_id, role, and username from the payload. Then, a new access_token is generated using these details. The response includes the newly generated access_token, enabling the user to continue their session without needing to log in again.

- **/logout :**

The Logout handles user logout functionality. It uses the @jwt_required() decorator to ensure that only authenticated users (those with a valid JWT token) can access this endpoint. When a user sends a POST request to log out, the method returns a success message: "Logout successful". If an exception occurs during the process, the method catches the error and returns an error message with the exception details. However, since the logout process generally involves invalidating the user's session or token on the client side (which is not explicitly handled in this code), the response mainly indicates a successful logout action.

2. Admin

- **/topquery :**

The TopSupportQueries retrieves the top 5 most asked queries in the past 7 days from the ChatbotHistory table, requiring user authentication with the @jwt_required() decorator. It calculates the date for 7 days ago, queries the database to count how often each query was asked, groups and orders the queries by count in descending order, and limits the result to the top 5. The queries are formatted into a list of dictionaries with query text and count, and the response is returned as a JSON object containing the top 5 queries.

- **/query_detail/<int:query_id> :**

The QueryDetail retrieves detailed information about a specific query by its query_id. The @jwt_required() decorator ensures the user is authenticated before accessing this resource. The method fetches the query from the IssueQuery table using the query_id. If the query is not found, it returns a 404 error with a "Query not found" message. If the query exists, it retrieves the associated student's name from the User table (or "Unknown Student" if no student is found) and returns a JSON response containing the query details such as the query text, student name, and timestamp.

- **/solve_query/<int:query_id> :**

The SolveQuery allows an admin to solve a specific query by providing an answer. The @jwt_required() decorator ensures the user is authenticated. It first checks if the logged-in user is an admin by verifying their role. If the user is not an admin, it returns a 403 Unauthorized message. Then, the method fetches the query using the provided query_id. If the query is not found, it returns a 404 error. The user must provide an answer for the query; if not, it returns a 400 error. Upon receiving a valid answer, it creates a new SolveIssue entry to store the solution, commits it to the database, and returns a success message with details of the solved query. If an error occurs during the database operation, it rolls back the transaction and returns a 500 error with the exception details.

- **/add_course :**

The AddCourse allows an admin to add a new course. It checks if the user is an admin, returning a 403 error if not. The method then validates that both the course name and description are provided. If any are missing, a 400 error is returned. Upon successful validation, a new course is created, added to the database, and committed. A success message is returned, or a 500 error is returned if an issue occurs during the process.

- **/edit_course/<int:course_id> :**

The EditCourse allows an admin to update a course. It first checks if the user is an admin, returning a 403 error if not. Then, it verifies the course exists, returning a 404 error if not found. The method updates the course's name and description based on the provided data, leaving existing values unchanged if not provided. After committing the changes to the database, it returns a success message, or a 500 error if an issue occurs during the process.

- **/add_course/<int:course_id>/material :**

The AddCourseMaterial allows an admin to add new materials to a course. It checks if the user is an admin, returning a 403 error if not. Then, it verifies that the course exists, returning a 404 error if not found. The method validates that both the title and material link are provided; otherwise, it returns a 400 error. Upon successful validation, a new course material is added to the database. A success message is returned, or a 500 error is returned if an issue occurs during the process.

- **/edit_course/material/<int:material_id> :**

The EditCourseMaterial allows an admin to update course materials. It first checks if the user is an admin, returning a 403 error if not. Then, it verifies the material exists, returning a 404 error if not found. The method updates the material's title and link based on the provided data, leaving existing values unchanged if not provided. After committing the changes to the database, a success message is returned, or a 500 error is returned if there is an issue during the update process.

3. Student

- **/student_profile :**

The StudentProfile retrieves a student's profile information. It first authenticates the user with jwt_required(), then fetches the student's details from the database using their ID. If the student is not found, it returns a 404 error. Otherwise, it returns a JSON response containing the student's ID, name, username, email, role, and last login timestamp.

- **/student_dashboard :**

The StudentDashboard retrieves a student's enrolled courses, assignments, course materials, chatbot history, and issue queries using jwt_required() for authentication. It fetches student details from User, retrieves enrolled courses from UserCourse, and gathers related assignments and materials. Additionally, it collects chatbot interactions from ChatbotHistory and issue queries from IssueQuery. The code could be optimized by reducing individual database queries using join(), applying lazy='joined' in SQLAlchemy relationships, and limiting chatbot history and queries to recent entries for better performance.

4. Instructor

- **/add_suplementary :**

The LessonResource allows authenticated users to add supplementary materials to a course using jwt_required(). It validates the presence of course_id, material_type, and content before creating a new SupplementaryMaterial entry in the database. The lesson is then added and committed to the database. Optimization can be achieved by handling bulk inserts efficiently, implementing validation for material types, and indexing frequently queried fields like course_id to enhance performance.

- **/add_assigments :**

The AssignmentResource enables authenticated users to add assignments to a course using jwt_required(). It validates the presence of course_id, week_number, assignment_link, and description before creating a new Assignment entry in the database. Once validated, the assignment is added and committed. Optimization can be achieved by ensuring unique constraints on week_number per course, implementing validation for assignment links, and indexing course_id for efficient querying.

- **/add_livesession :**

The LiveSessionResource class allows authenticated users to add live sessions to a course using jwt_required(). It checks for required fields (course_id, yt_link, description) before creating a LiveSession entry in the database. The session is then added and committed. Optimization can include validating yt_link format, enforcing unique constraints on sessions per course, and indexing course_id for faster lookups.

5. Professor

- **/pending_instructor :**

The PendingInstructors retrieves all pending instructor requests by first verifying the identity of the professor via jwt_required(). It then checks if the professor exists in the database and fetches pending instructor requests from InstructorRequest based on their status being PENDING. The response is formatted to return the request details in JSON format, including the instructor ID, status, and creation timestamp. For optimization, you could consider limiting the number of results returned (pagination), indexing status for better query performance, and adding error handling for potential database issues.

- **/approve_instructor :**

The ApproveInstructor allows a professor to approve or reject an instructor request by changing its status. It first validates the professor's identity via JWT authentication. Then, it retrieves the status from the request payload and checks if it's valid (either "Approved" or "Rejected"). If the status is valid, it fetches the instructor request from the InstructorRequest model. If the request exists, it updates the status and commits the change to the database. A success message is returned upon completion. For optimization, consider adding error handling for database failures, validating the professor's role to ensure they are authorized to approve/reject, and checking if the instructor request has already been processed.

- **/solved_issues :**

The SolvedIssues retrieves solved issues by joining the IssueQuery and SolveIssue tables on the issue ID, returning both the issue details and its solution. The retrieved data is processed into a list containing the issue ID, details, creation timestamp, solver ID, solution answer, and the solved time. The class then returns a JSON response with the count of solved issues and the corresponding list. Optimizations could include using with_entities() to select only necessary columns, adding pagination for large data sets, and improving error handling when no solved issues are present.

- **/pending_issues :**

The PendingIssues retrieves issues that are not yet solved by performing a subquery to get all issue IDs from the SolveIssue table and then filtering the IssueQuery table to exclude those IDs. The resulting pending issues are processed into a list containing the issue ID, details, and creation timestamp. The class returns a JSON response with the count of pending issues and the corresponding list. To improve performance, consider using pagination for large result sets and optimizing query execution by selecting only necessary fields using with_entities().

6. ChatBot

- **/chat :**

The ChatbotAPI class handles incoming queries by first verifying if the query is empty. It then retrieves relevant documents from the database using a similarity search, generates context from the documents, and prepares messages for the chatbot by providing the context and the user's query. The chatbot response is generated and returned as part of the JSON response. Additionally, document references (subject, week, document type) are included to guide the user with useful resources. The class has a provision to store chat history in the database, though that part is currently commented out. Error handling is in place to return an error message if any issue occurs during the process. To optimize, consider caching results for frequently asked queries and refining document retrieval to improve response time.

SWAGGER


Authentication
POST /login User Login
POST /signup User Signup
POST /refresh_token Refresh Access Token
POST /logout User Logout
Admin
GET /top-support-queries Get top support queries in the past 7 days
GET /query-detail/{query_id} Get details of a specific query
POST /solve-query/{query_id} Solve a specific query
POST /add-course Add a new course
PUT /edit-course/{course_id} Edit a specific course
POST /add-course-material/{course_id} Add course material
PUT /edit-course-material/{material_id} Edit course material
Student
GET /student-profile Retrieve student profile details
GET /student-dashboard Retrieve student dashboard details
Instructor
POST /lesson Add a new lesson (Supplementary Material)
POST /assignment Add a new assignment
POST /livesession Add a new live session
Professor
GET /pending-instructors Retrieve all pending instructor requests
PUT /approve-instructor/{request_id} Approve or reject an instructor request
GET /solved-issues Retrieve all solved issues
GET /pending-issues Retrieve all pending issues
Chatbot
POST /chatbot Interact with the chatbot for educational hints and resources

PyTest TESTING

INTRODUCTION

This document outlines the testing strategy for the AI-based academic guidance system using pytest, a powerful testing framework for Python. The system is designed to assist students by providing academic guidance, answering queries, and recommending relevant resources.

Given the AI-driven nature of the system, testing is crucial to ensure accuracy, reliability, and efficiency. Our approach involves unit testing, integration testing, and functional testing to validate different components, including:

- **AI Model Accuracy:** Ensuring responses are contextually correct and relevant.
- **API Integrations:** Verifying seamless communication with external data sources.
- **User Query Handling:** Testing various input scenarios for robustness.
- **Performance & Edge Cases:** Checking response times and system behavior under stress.

By leveraging pytest, we automate test execution, generate reports, and streamline debugging, ensuring the system meets quality standards before deployment.

The testing framework for this project is built upon Python's pytest, a powerful and flexible testing tool known for its ease of use and scalability. This framework is designed to ensure the accuracy, reliability, and efficiency of the AI-based academic guidance system.

The testing strategy includes the following:

- **Unit Tests:** Verifying individual functions, such as text processing, query classification, and API interactions.
- **Integration Tests:** Ensuring seamless interactions between system components, including the AI model, database, and external APIs.
- **System Tests:** Validating the entire application workflow, from user input to AI-generated responses.
- **Performance Testing:** Evaluating response times and system behavior under high query loads.
- **Edge Case Handling:** Testing how the system responds to incomplete, ambiguous, or unexpected inputs.

To streamline testing, the framework integrates with continuous integration (CI) pipelines, allowing for automated test execution and early detection of issues. This ensures that the AI agent consistently delivers accurate and relevant academic guidance while maintaining high performance across different scenarios.

TEST CASES

1. ADMIN API's

a.) Admin Login (SUCCESS)

ENDPOINT : POST /login

- **Test Case Name :- test_admin_login**

INPUT :-

```
{
  "username": "admin@example.com",
  "password": "admin123"
}
```

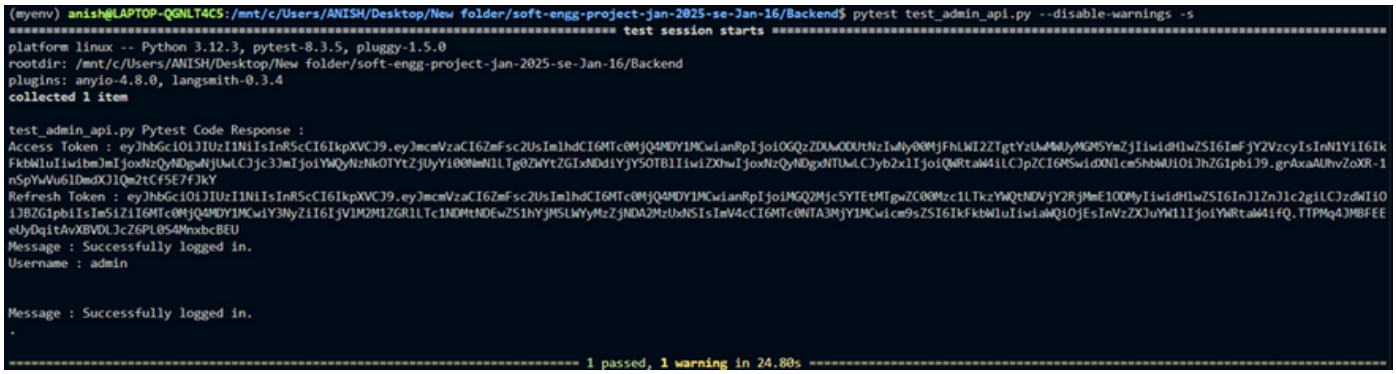
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Successfully loged in." }

PyTest Code :-

```python
def test_admin_login(client):
    """Test admin login API."""
    response = client.post("/login", json={"username": "admin@example.com", "password": "admin123"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response : ')
    print('Access Token :',json_data['access_token'])
    print('Refresh Token :' ,json_data['refresh_token'])
    print('Message :',json_data['message'])
    print('Username :',json_data['username'])
    assert "access_token" in json_data
    assert json_data["message"] == "Successfully logged in."
    print('\n')
    print('Message :',json_data['message'])
```

Actual Output :-



b.) Admin Login (Missing Fields)

ENDPOINT : POST /login

- **Test Case Name :- test_login_invalid_password**

INPUT :-

```
{
  "username": "admin@example.com",
  "password": "wrongpassword"
}
```

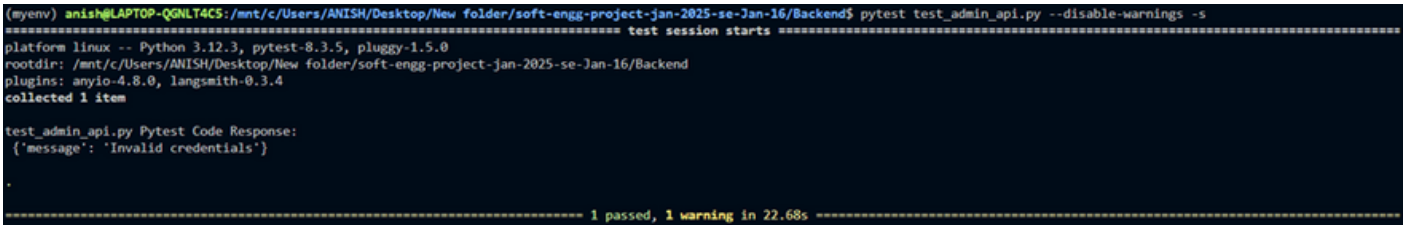
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Invalid Credentials." }

PyTest Code :-

```python
def test_login_invalid_password(client):
    """Test login with incorrect password."""
    response = client.post("/login", json={"username": "admin@example.com", "password": "wrongpassword"})

    assert response.status_code == 200  # Your API returns 200 even for invalid credentials
    json_data = response.get_json()
    print('Pytest Code Response:\n',json_data ,'\n')
    assert json_data["message"] == "Invalid credentials"
```

Actual Output :-



c.) Admin Login (Missing Fields)

ENDPOINT : POST /login

- **Test Case Name :- test_login_nonexistent_user**

INPUT :-

```
{
  "username": "nonexistent@example.com",
  "password": "wrongpassword"
}
```

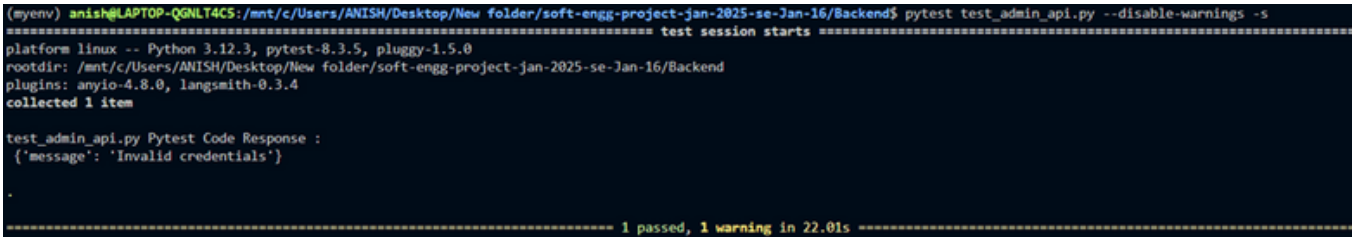
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Invalid Credentials." }

PyTest Code :-

```python
def test_login_nonexistent_user(client):
    """Test login with a non-existing username."""
    response = client.post("/login", json={"username": "nonexistent@example.com", "password": "admin123"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response :\n',json_data ,'\n')
    assert json_data["message"] == "Invalid credentials"
```

Actual Output :-



d.) Admin Login (Missing Fields)

ENDPOINT : POST /login

- **Test Case Name :- test_login_missing_username**

INPUT :-

```
{
   "password": "wrongpassword"
}
```

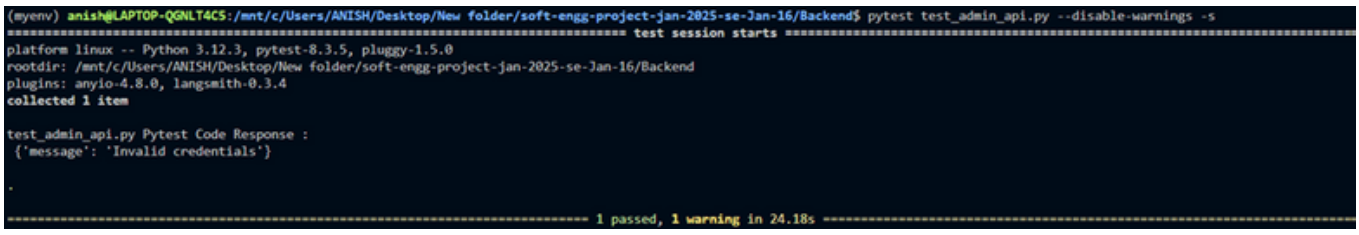
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Invalid Credentials." }

PyTest Code :-

```python
def test_login_missing_username(client):
    """Test login with missing username."""
    response = client.post("/login", json={"password": "admin123"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response :\n',json_data ,'\n')
    assert json_data["message"] == "Invalid credentials"
```

Actual Output :-



e.) Admin Login (Missing Fields)

ENDPOINT : POST /login

- **Test Case Name :- test_login_empty_credentials**

INPUT :-

{ }

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Invalid Credentials." }

PyTest Code :-

```python
def test_login_empty_credentials(client):
    """Test login with empty credentials."""
    response = client.post("/login", json={})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response:\n',json_data ,'\n')
    assert json_data["message"] == "Invalid credentials"
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend$ pytest test_admin_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_admin_api.py Pytest Code Response:
 {'message': 'Invalid credentials'}

.

======================== 1 passed, 1 warning in 28.62s =========================
```

f.) Admin Login (Missing Fields)

ENDPOINT : POST /login

- **Test Case Name :- test_login_invalid_json**

INPUT :-

data = “invalid_json”, content_type = “application/json”

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Invalid Credentials." }

PyTest Code :-

```python
def test_login_invalid_json(client):
    """Test login with invalid JSON format."""
    response = client.post("/login", data="invalid_json", content_type="application/json")

    assert response.status_code == 400  # Assuming Flask handles bad JSON format errors
    json_data = response.get_json()

    print('Pytest Code Response : \n',json_data ,'\n')
    assert "message" in json_data
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend$ pytest test_admin_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_admin_api.py Pytest Code Response:
 {'message': 'Invalid credentials'}

.

======================== 1 passed, 1 warning in 28.62s =========================
```

g.) Admin Details (SUCCESS)

ENDPOINT : GET /admin_details

- **Test Case Name :- test_admin_details**

EXPECTED OUTPUT :-
HTTP 200

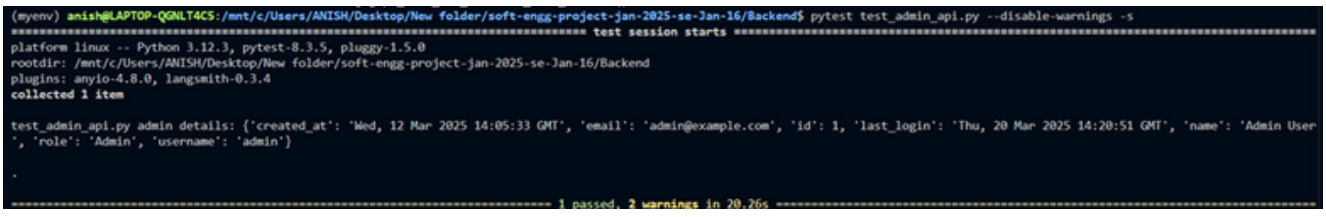
PyTest Code :-

```python
def test_admin_details(client, auth_header):
    """Test fetching admin details API."""
    response = client.get("/admin_details", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print('admin details:',json_data,'\n')

    assert json_data["username"] == "admin"
    assert json_data["email"] == "admin@example.com"
    assert json_data["role"].lower() == "admin"
```

Actual Output :-



h.) Admin Details (SUCCESS)

ENDPOINT : GET /admin_details

- **Test Case Name :- test_admin_details_unauthorized**

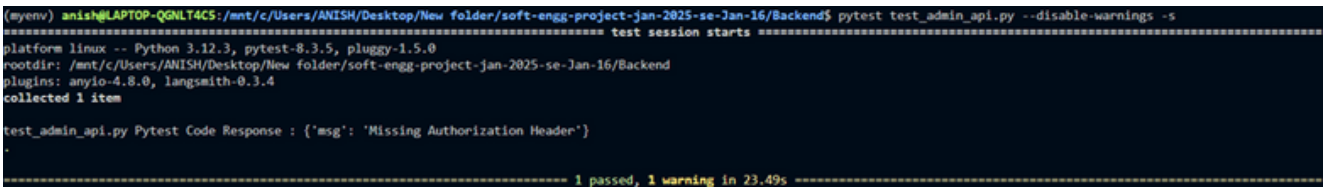
EXPECTED OUTPUT :-
HTTP 401 , JSON { "message": "Missing Authorization Header." }

PyTest Code :-

```python
def test_admin_details_unauthorized(client):
    """Test fetching admin details without authentication."""
    response = client.get("/admin_details")  # No auth header provided

    assert response.status_code == 401  # Expected: Unauthorized
    json_data = response.get_json()
    print("Pytest Code Response :",json_data)
    assert "msg" in json_data  # Typical Flask-JWT error response
    assert json_data["msg"] == "Missing Authorization Header"
```

Actual Output :-



i.) Admin Details (Missing Fields)

ENDPOINT : GET /admin_details

- **Test Case Name :- test_admin_details_invalid_token**

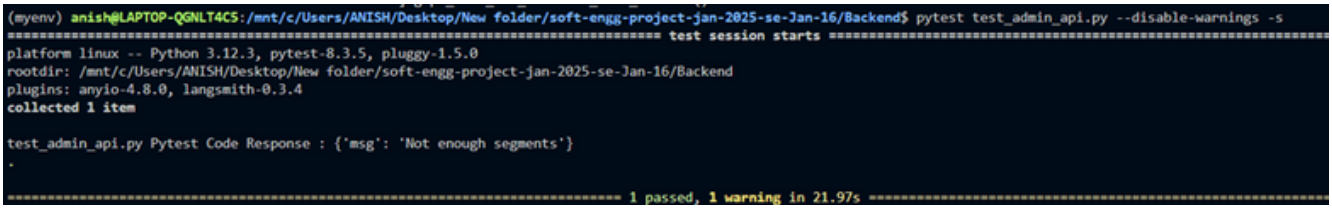
EXPECTED OUTPUT :-
HTTP 422 , JSON { "message": "Not enough segments." }

PyTest Code :-

```python
def test_admin_details_invalid_token(client):
    """Test fetching admin details with an invalid token."""
    response = client.get("/admin_details", headers={"Authorization": "Bearer invalid_token"})

    assert response.status_code == 422  # Expected: Unauthorized
    json_data = response.get_json()
    print("Pytest Code Response :", json_data)
```

Actual Output :-



j.) Admin Details (SUCCESS)

ENDPOINT : GET /admin_courses

- **Test Case Name :- test_get_all_courses**

EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_get_all_courses(client, auth_header):
    """Test fetching all courses API."""
    response = client.get("/admin_courses", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print("Pytest Code Response :", json_data,'\n')

    assert "courses" in json_data
    assert isinstance(json_data["courses"], list)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend$ pytest test_admin_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_admin_api.py Pytest Code Response : {'courses': [{'created_at': 'Wed, 12 Mar 2025 14:13:17 GMT', 'description': 'This is a machine learning course.', 'id': 1, 'name': 'MLP', 'total_
assignments': 0, 'total_materials': 3, 'total_students': 1}, {'created_at': 'Wed, 12 Mar 2025 14:13:47 GMT', 'description': 'This is a Business management course.', 'id': 2, 'name': 'BDM
', 'total_assignments': 0, 'total_materials': 0, 'total_students': 1}, {'created_at': 'Wed, 12 Mar 2025 16:10:58 GMT', 'description': 'aaaaaaaaaaa', 'id': 3, 'name': 'aaaaaaa', 'total_as
signments': 0, 'total_materials': 0, 'total_students': 0}, {'created_at': 'Tue, 18 Mar 2025 11:12:37 GMT', 'description': 'h', 'id': 4, 'name': 'd', 'total_assignments': 0, 'total_materi
als': 0, 'total_students': 0}, {'created_at': 'Tue, 18 Mar 2025 11:21:56 GMT', 'description': 'gggggg', 'id': 5, 'name': 'ddddddddddd', 'total_assignments': 0, 'total_materials': 2, 'tot
al_students': 0}, {'created_at': 'Tue, 18 Mar 2025 11:40:07 GMT', 'description': 'cccc', 'id': 6, 'name': 'Advanced Python', 'total_assignments': 0, 'total_materials': 2, 'total_students
': 0}, {'created_at': 'Thu, 20 Mar 2025 11:49:03 GMT', 'description': 'Intro to Python', 'id': 7, 'name': 'Python 101', 'total_assignments': 0, 'total_materials': 0, 'total_students': 0)
, {'created_at': 'Thu, 20 Mar 2025 11:49:18 GMT', 'description': 'Intro to Python', 'id': 8, 'name': 'Python 101', 'total_assignments': 0, 'total_materials': 0, 'total_students': 0}, {'c
reated_at': 'Thu, 20 Mar 2025 11:54:11 GMT', 'description': 'Intro to Python', 'id': 9, 'name': 'Python 101', 'total_assignments': 0, 'total_materials': 0, 'total_students': 0}, {'create
d_at': 'Thu, 20 Mar 2025 13:03:46 GMT', 'description': 'Intro to Python', 'id': 10, 'name': 'Python 101', 'total_assignments': 0, 'total_materials': 0, 'total_students': 0}, {'created_at
': 'Thu, 20 Mar 2025 13:10:58 GMT', 'description': 'Intro to Python', 'id': 11, 'name': 'Python 101', 'total_assignments': 0, 'total_materials': 0, 'total_students': 0}]}

.

============================== 1 passed, 1 warning in 21.25s ===============================
```

k.) Admin Details (SUCCESS)

ENDPOINT : GET /admin_students

- **Test Case Name :- test_get_all_students**

EXPECTED OUTPUT :-
HTTP 200

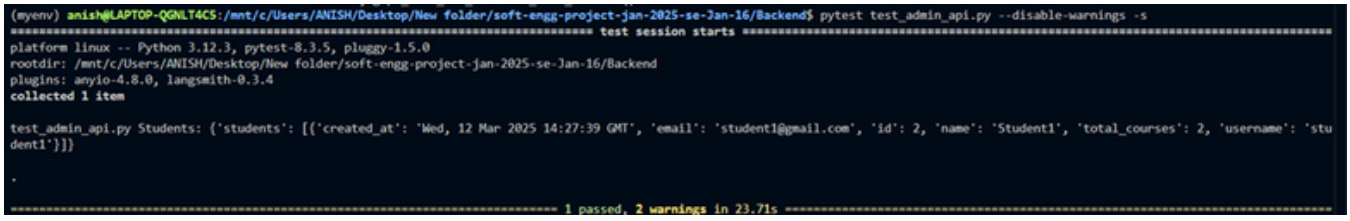
PyTest Code :-

```python
def test_get_all_students(client, auth_header):
    """Test fetching all students API (admin access only)."""
    response = client.get("/admin_students", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print("Students:", json_data,'\n')

    assert "students" in json_data
    assert isinstance(json_data["students"], list)
```

Actual Output :-



I.) Admin Details (SUCCESS)

ENDPOINT : POST /add_course

- **Test Case Name :- test_add_course**

INPUT :-

```
{
  "name": "Python 101",
  "description": "Intro to Python"
}
```

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Course added successfully!" }

PyTest Code :-

```python
def test_add_course(client, auth_header):
    """Test adding a new course."""
    response = client.post("/add_course", json={"name": "Python 101", "description": "Intro to Python"}, headers=auth_header)
    print("Pytest Response :",response.get_json(),'\n')
    assert response.status_code == 201
    assert response.get_json()["message"] == "Course added successfully!"
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend$ pytest test_admin_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_admin_api.py Pytest Response : {'message': 'Course added successfully!'}

.

======================== 1 passed, 2 warnings in 20.28s ========================
```

m.) Admin Details (SUCCESS)

ENDPOINT : PUT /edit_course/<int:course_id>

- **Test Case Name :- test_add_course**

INPUT :-

```
{
  "name": "Advanced Python Hack"
}
```

EXPECTED OUTPUT :-

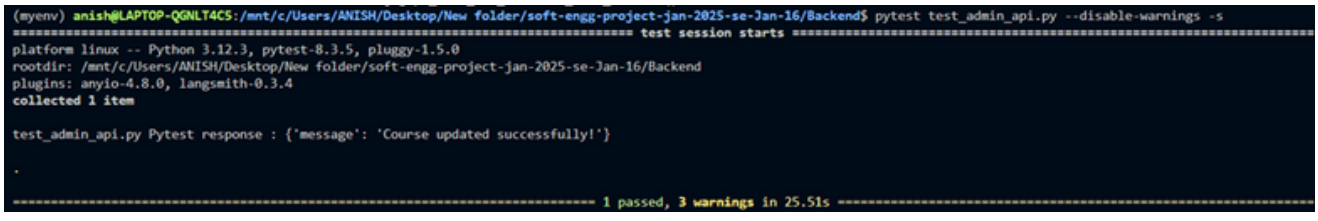
HTTP 200 , JSON { "message": "Course updated successfully!" }

PyTest Code :-

```python
def test_edit_course(client, auth_header):
    """Test editing an existing course."""
    response = client.put("/edit_course/6", json={"name": "Advanced Python Hack"}, headers=auth_header)
    print("Pytest response :",response.get_json(),'\n')

    assert response.status_code == 200
    assert response.get_json()["message"] == "Course updated successfully!"
```

Actual Output :-



n.) Admin Details (SUCCESS)

ENDPOINT : POST /add_course/<int:course_id>/material

- Test Case Name :- test_add_course

INPUT :-

```
{
  "week_number": 2,
  "title": "Introduction",
  "material_link": "http://example.com/video1"
}
```

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Course material added successfully!" }

PyTest Code :-

```python
def test_add_course_material(client, auth_header):
    response = client.post("/add_course/5/material", json={"week_number": 2, "title": "Introduction", "material_link": "http://example.com/video1"}, headers=auth_header)
    print("Pytest response:",response.get_json(),'\n')

    assert response.status_code == 200
    assert response.get_json()["message"] == "Course material added successfully!"
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend$ pytest test_admin_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/Backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_admin_api.py Pytest response: {'message': 'Course material added successfully!'}

.

======================== 1 passed, 3 warnings in 23.26s ========================
```

2. STUDENT API's

a.) Student SignUp (SUCCESS)

ENDPOINT : POST /signup

- **Test Case Name :- test_successful_student_signup**

INPUT :-

```
{
  "name": "John",
  "email": "johndoe1@example.com",
  "password": "password123",
  "role": "Student"
}
```

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Student registered successfully!" }

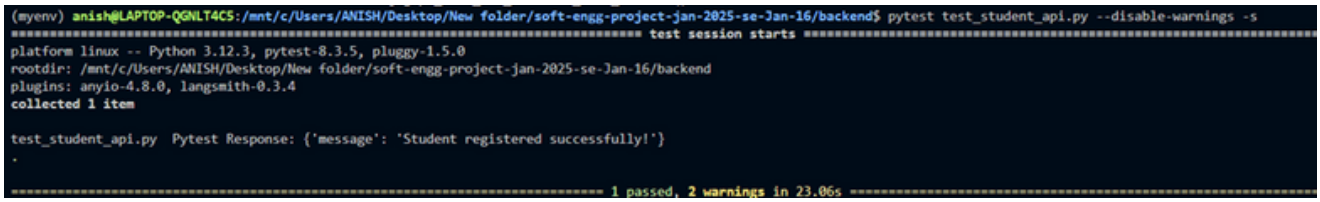
PyTest Code :-

```python
def test_successful_student_signup(client, db_session):
    """Test successful student signup"""
    payload = {
        "name": "John",
        "email": "johndoe1@example.com",
        "password": "password123",
        "role": "Student"
    }

    response = client.post("/signup", json=payload)
    print( " Pytest Response:", response.get_json())

    assert response.status_code == 200
    assert response.get_json()["message"] == "Student registered successfully!"
```

Actual Output :-



b.) Student SignUp (Missing Fields)

ENDPOINT : POST /signup

- **Test Case Name :- test_signup_missing_fields**

INPUT :-

```
{
  "email": "user@example.com",
  "password": "testpass",
  "role": "Student"
}
```

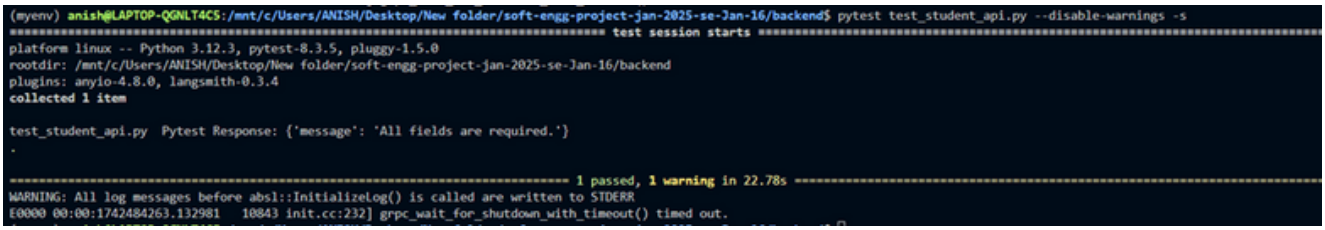
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "All fields are required." }

PyTest Code :-

```python
def test_signup_missing_fields(client):
    """Test signup with missing required fields"""
    payload = {
        "email": "user@example.com",
        "password": "testpass",
        "role": "Student"
    }

    response = client.post("/signup", json=payload)
    print( " Pytest Response:", response.get_json())
    assert response.status_code == 200
    assert response.get_json()["message"] == "All fields are required."
```

Actual Output :-



c.) Student SignUp (Invalid Role)

ENDPOINT : POST /signup

- **Test Case Name :- test_signup_invalid_role**

INPUT :-

```
{
  "name": "Unknown Role",
  "email": "unknown@example.com",
  "password": "testpass",
  "role": "Manager"
}
```

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Role 'Manager' not found in database " }

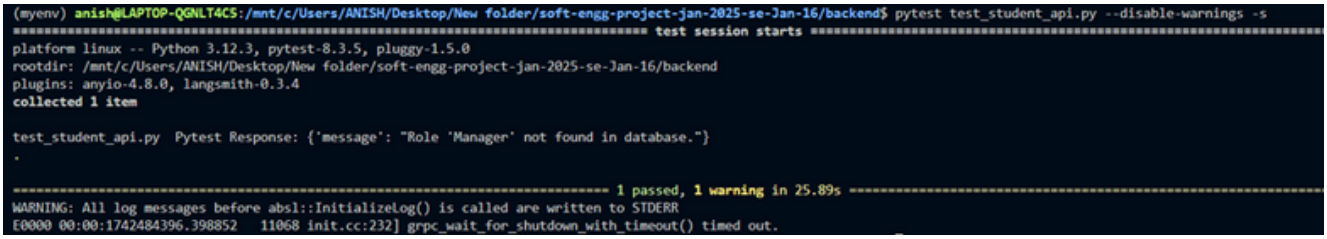
PyTest Code :-

```python
def test_signup_invalid_role(client):
    """Test signup with an invalid role"""
    payload = {
        "name": "Unknown Role",
        "email": "unknown@example.com",
        "password": "testpass",
        "role": "Manager"
    }

    response = client.post("/signup", json=payload)
    print( " Pytest Response:", response.get_json())

    assert response.status_code == 200
    assert response.get_json()["message"] == "Role 'Manager' not found in database."
```

Actual Output :-



d.) Student LogIn (SUCCESS)

ENDPOINT : POST /login

- **Test Case Name :- test_student_login**

INPUT :-

```
{
  "username": "student1@gmail.com",
  "password": "123456"
}
```

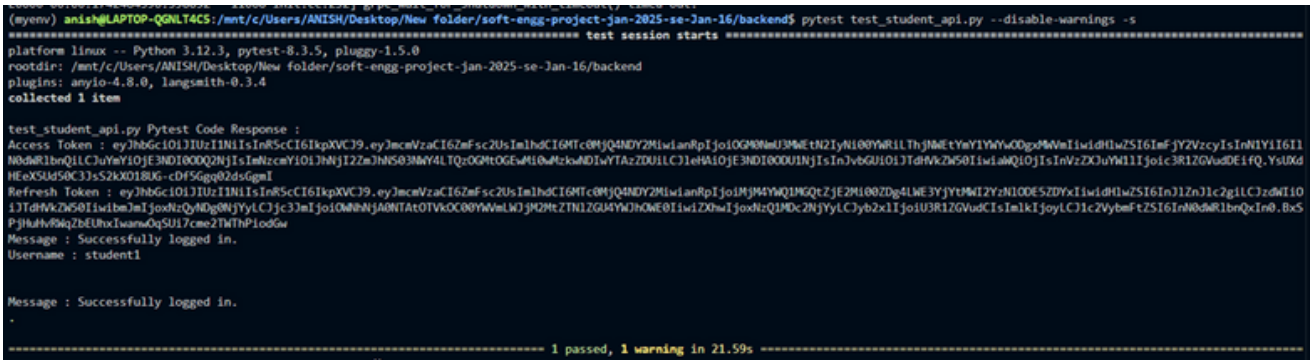
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Successfully logged in." }

PyTest Code :-

```python
def test_student_login(client):
    """Test admin login API."""
    response = client.post("/login", json={"username": "student1@gmail.com", "password": "123456"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response : ')
    print('Access Token :',json_data['access_token'])
    print('Refresh Token :' ,json_data['refresh_token'])
    print('Message :',json_data['message'])
    print('Username :',json_data['username'])
    assert "access_token" in json_data
    assert json_data["message"] == "Successfully logged in."
    print('\n')
    print('Message :',json_data['message'])
```

Actual Output :-



e.) Student Profile (SUCCESS)

ENDPOINT : GET /student_profile

- **Test Case Name :- test_student_profile**

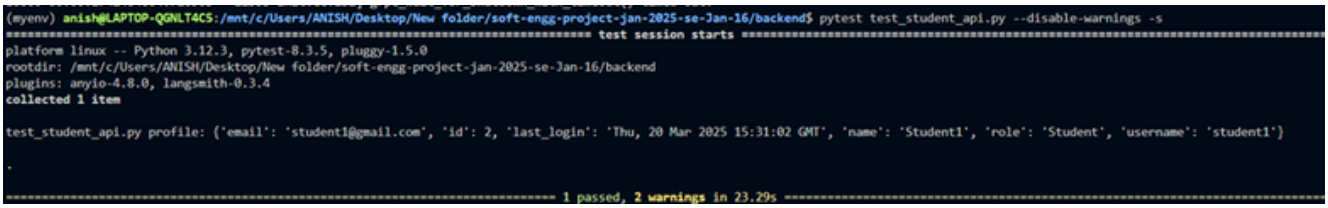
EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_student_profile(client, auth_header):
    """Test the Student Profile API."""
    response = client.get("/student_profile", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print("profile:",json_data ,'\n')
```

Actual Output :-



f.) Student Dashboard (SUCCESS)

ENDPOINT : GET /student_dashboard

- **Test Case Name :- test_student_dashboard**

EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_student_dashboard(client, auth_header):
    """Test the Student Dashboard API."""
    response = client.get("/student_dashboard", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print("dashbord",json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_student_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_student_api.py dashbord {'courses_data': [{'assignments': [], 'created_at': 'Wed, 12 Mar 2025 14:13:17 GMT', 'description': 'This is a machine learning course.', 'id': 1, 'live_sess
ions': [], 'materials': [{'created_at': 'Wed, 12 Mar 2025 14:20:40 GMT', 'id': 1, 'title': 'Pandas Series and DataFrame 1', 'video_link': 'https://youtu.be/MdnmbjKM7a0', 'week_number': 1
}, {'created_at': 'Wed, 12 Mar 2025 14:20:40 GMT', 'id': 2, 'title': 'Pandas Series and DataFrame 1', 'video_link': 'https://youtu.be/MdnmbjKM7a0', 'week_number': 1}, {'created_at': 'Wed
, 12 Mar 2025 14:26:36 GMT', 'id': 3, 'title': 'Pandas Series and DataFrame 1', 'video_link': 'https://youtu.be/MdnmbjKM7a0', 'week_number': 1}], 'name': 'MLP', 'supplementary_materials'
: []}, {'assignments': [], 'created_at': 'Wed, 12 Mar 2025 14:13:47 GMT', 'description': 'This is a Business management course.', 'id': 2, 'live_sessions': [], 'materials': [], 'name': '
BDM', 'supplementary_materials': []}]}
.

============================== 1 passed, 4 warnings in 24.52s ==============================
```

g.) Student Refresh Token (SUCCESS)

ENDPOINT : POST /token_refresh

- **Test Case Name :- test_refresh_token**

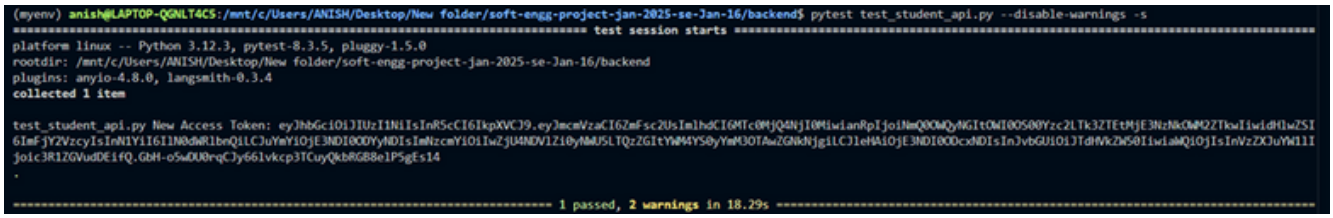
EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```
def test_refresh_token(client,auth_header):
    """Test the Refresh Token API."""
    response = client.post("/token_refresh", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    assert "access_token" in json_data
    print("New Access Token:", json_data["access_token"])
```

Actual Output :-



h.) Student LogOut (SUCCESS)

ENDPOINT : POST /logout

- **Test Case Name :- test_refresh_token**

EXPECTED OUTPUT :-
HTTP 200, JSON { "message": "Logout successful" }

PyTest Code :-

```python
def test_logout(client, auth_header):
    """Test the Logout API."""
    response = client.post("/logout", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    assert json_data["message"] == "Logout successful"
    print("Logout Response:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_student_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_student_api.py Logout Response: {'message': 'Logout successful'}
.

======================== 1 passed, 1 warning in 21.21s =========================
```

3. PROFESSOR API's

a.) Professor LogIn (SUCCESS)

ENDPOINT : POST /login

- **Test Case Name :- test_professor_login**

INPUT :-

```
{
  "username": "professor1@gmail.com",
  "password": "123456"
}
```

EXPECTED OUTPUT :-

HTTP 200 , JSON { "message": "Successful logged in." }

PyTest Code :-

```python
def test_professor_login(client):
    """Test admin login API."""
    response = client.post("/login", json={"username": "professor1@gmail.com", "password": "123456"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response : ')
    print('Access Token :',json_data['access_token'])
    print('Refresh Token :' ,json_data['refresh_token'])
    print('Message :',json_data['message'])
    print('Username :',json_data['username'])
    assert "access_token" in json_data
    assert json_data["message"] == "Successfully logged in."
    print('\n')
    print('Message :',json_data['message'])
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_professor_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_professor_api.py Pytest Code Response :
Access Token : eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc0MjQ4ODAwOCwianRpIjoiN2YyNjVmODktOTcwNy00YWE0LTgyNmItODM0ZWQzMDM1YjAwIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IlByb2Zlc3NvciIsIm5iZiI6MTc0MjQ4ODAwOCwiY3NyZiI6ImJmMTRlMGE0LTY5NzQtNDRiMi04NjgSLTE2MWIwNTE5NjQ4NSIsImV4cCI6MTc0MjQ4ODkwOCwicm9sZSI6IlByb2Zlc3NvciIsImlkIjozLCJ1c2VybmFtZSI6InByb2Zlc3NvcjEifQ.L2H0pyvzsjhZpIwdqLFJ0GvSN8SlhXmtGKniYY-qIXk
Refresh Token : eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc0MjQ4ODAwOCwianRpIjoiNTk5NWFiY2MtNjQ5OS00NjlmLThhYTMtMjJlNTk5MzVjNDFhIiwidHlwZSI6InJlZnJlc2giLCJzdWIiOiJQcm9mZXNzb3IiLCJuYmYiOjE3NDI0ODgwMDgsImNzcmYiOiJiMTM0Y2UwZi04ZWQzLTQ5NTAtODFjZC04ZjBjMmE5YjA5NDkiLCJleHAiOjE3NDUwODA4MDgsInJvbGUiOiJQcm9mZXNzb3IiLCJpZCI6MywidXNlcm5hbWUiOiJwcm9mZXNzb3IxIn0.WzpGXbLbwRb5MMUvSA-eF-uGrrAWp2OpqS1LFC0-Sk4
Message : Successfully logged in.
Username : professor1


Message : Successfully logged in.
.

============================== 1 passed, 1 warning in 18.97s ===============================
```

b.) Professor Details (SUCCESS)

ENDPOINT : GET /professor_details

- **Test Case Name :- test_professor_details**

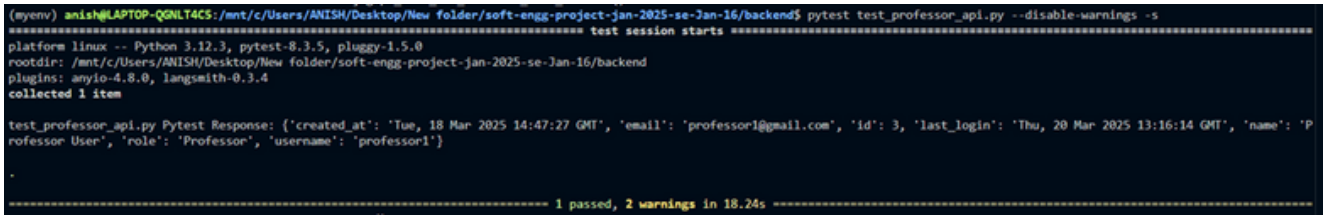
EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_professor_details(client, auth_header):
    """Test fetching admin details API."""
    response = client.get("/professor_details", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Response:',json_data,'\n')
```

Actual Output :-



c.) Professor - Pending Instructors (SUCCESS)

ENDPOINT : GET /pending_instructors

- **Test Case Name :- test_pending_instructors**

EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_pending_instructors(client, auth_header):
    """Test the Pending Instructors API."""
    response = client.get("/pending_instructor", headers=auth_header)

    assert response.status_code in [200, 404]  # Either success or professor not found
    json_data = response.get_json()

    if response.status_code == 200:
        assert isinstance(json_data, list)  # The response should be a list
        if json_data:  # If there are pending instructors
            for req in json_data:
                assert "id" in req
                assert "instructor_id" in req
                assert "status" in req
                assert "created_at" in req
        print("Pending Instructors:", json_data)
    else:
        assert json_data["message"] == "Professor not found"
        print("Professor Not Found:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_professor_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_professor_api.py Pending Instructors: [{'created_at': 'Thu, 20 Mar 2025 15:12:42 GMT', 'id': 6, 'instructor_id': 13, 'status': 'Pending'}, {'created_at': 'Thu, 20 Mar 2025 15:21:35
GMT', 'id': 7, 'instructor_id': 16, 'status': 'Pending'}]
.

======================== 1 passed, 2 warnings in 18.52s ========================
```

d.) Professor - Approve Instructors (SUCCESS)

ENDPOINT : PUT /approve_instructor/<int:request_id>'

- **Test Case Name :- test_approve_instructor**

INPUT :-

```
{
  "status": "Approved"
}
```

EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Instructor request approved successfully " }

PyTest Code :-

```python
def test_approve_instructor(client, auth_header):
    """Test the Approve Instructor API."""
    payload = {"status": "Approved"}  # Change to "Rejected" to test rejection flow
    response = client.put(f"/approve_instructor/6", json=payload, headers=auth_header)

    assert response.status_code in [200, 400, 404]  # Expect success, bad request, or not found
    json_data = response.get_json()

    if response.status_code == 200:
        assert "message" in json_data
        assert json_data["message"] in [f"Instructor request approved successfully",
                                        f"Instructor request rejected successfully"]
        print("Approve Instructor Response:", json_data)

    elif response.status_code == 400:
        assert json_data["message"] == "Invalid status, use 'Approved' or 'Rejected'"
        print("Invalid Status Error:", json_data)

    elif response.status_code == 404:
        assert json_data["message"] in ["Professor not found", "Instructor request not found"]
        print("Not Found Error:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_professor_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_professor_api.py Approve Instructor Response: {'message': 'Instructor request approved successfully'}
.

============================== 1 passed, 3 warnings in 21.18s ==============================
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1742488620.626662   16592 init.cc:232] grpc_wait_for_shutdown_with_timeout() timed out.
```

e.) Professor - Add Lesson (SUCCESS)

ENDPOINT : POST /add_suplementary

- **Test Case Name :- test_add_lesson**

INPUT :-

```
{
  "course_id": 1,
  "material_type": "Video",
  "content": "https://example.com/lesson.mp4"
}
```

EXPECTED OUTPUT :-

HTTP 200 , JSON { "message": "Lesson added successfully " }

PyTest Code :-

```python
def test_add_lesson(client, auth_header):
    """Test the Add Lesson API."""
    payload = {
        "course_id": 1,  # Replace with a valid course_id from your test database
        "material_type": "Video",  # Example material type
        "content": "https://example.com/lesson.mp4"  # Example content (URL or text)
    }

    response = client.post("/add_suplementary", json=payload, headers=auth_header)

    assert response.status_code in [201, 400]  # Expect success or bad request
    json_data = response.get_json()

    if response.status_code == 201:
        assert json_data["message"] == "Lesson added successfully"
        print("Lesson Added Successfully:", json_data)

    elif response.status_code == 400:
        assert json_data["message"] == "Missing required fields"
        print("Missing Fields Error:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_professor_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_professor_api.py Lesson Added Successfully: {'message': 'Lesson added successfully'}
.

============================== 1 passed, 1 warning in 16.84s ===============================
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1742488883.093717   17115 init.cc:232] grpc_wait_for_shutdown_with_timeout() timed out.
```

4. INSTRUCTOR API's

a.) Instructor SignUp (SUCCESS)

ENDPOINT : POST /signup

- **Test Case Name :- test_successful_instructor_signup**

INPUT :-

```
{
  "name": "Jane Smith1",
  "email": "janesmit1h@example.com",
  "password": "securepass",
  "role": "Instructor"
}
```

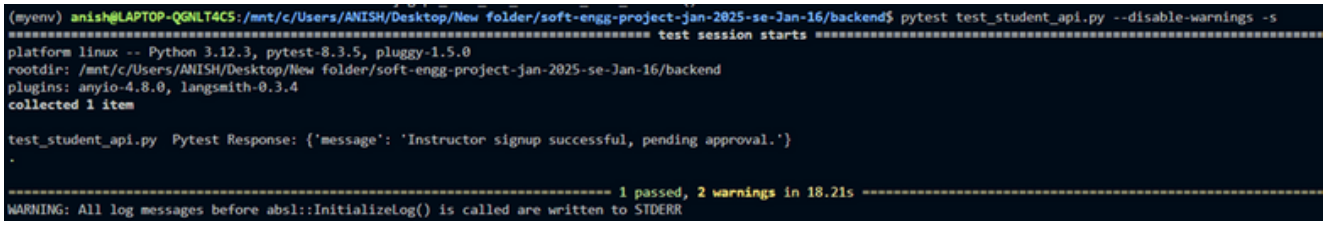
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Instructor signup successful, pending approval." }

PyTest Code :-

```python
def test_successful_instructor_signup(client, db_session):
    """Test successful instructor signup (pending approval)"""
    payload = {
        "name": "Jane Smith1",
        "email": "janesmit1h@example.com",
        "password": "securepass",
        "role": "Instructor"
    }

    response = client.post("/signup", json=payload)
    print( " Pytest Response:", response.get_json())
    assert response.status_code == 200
    assert response.get_json()["message"] == "Instructor signup successful, pending approval."
```

Actual Output :-



b.) Instructor LogIn (SUCCESS)

ENDPOINT : POST /login

- **Test Case Name :- test_inst_login**

INPUT :-

```
{
  "username": "int5@gmail.com",
  "password": "123456"
}
```

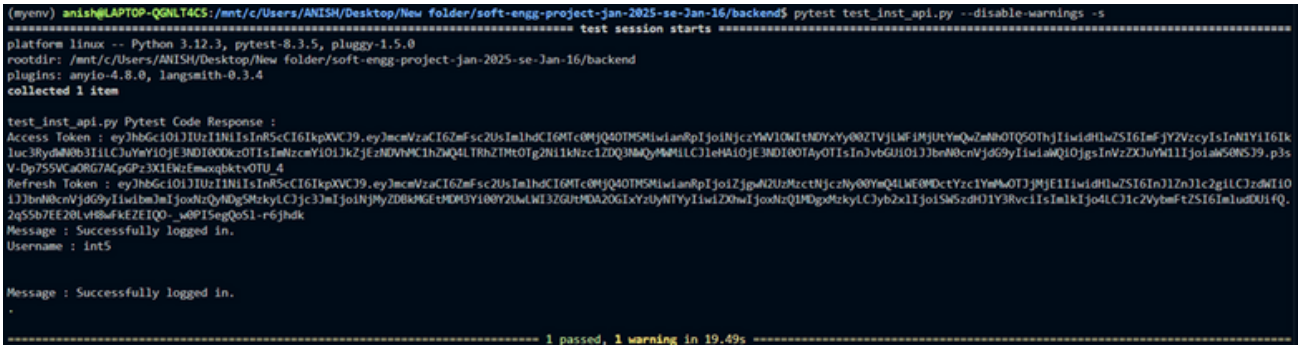
EXPECTED OUTPUT :-
HTTP 200 , JSON { "message": "Successfully logged in ." }

PyTest Code :-

```python
def test_inst_login(client):
    """Test admin login API."""
    response = client.post("/login", json={"username": "int5@gmail.com", "password": "123456"})

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Code Response : ')
    print('Access Token :',json_data['access_token'])
    print('Refresh Token :' ,json_data['refresh_token'])
    print('Message :',json_data['message'])
    print('Username :',json_data['username'])
    assert "access_token" in json_data
    assert json_data["message"] == "Successfully logged in."
    print('\n')
    print('Message :',json_data['message'])
```

Actual Output :-



c.) Instructor Details (SUCCESS)

ENDPOINT : GET /instructor_details

- **Test Case Name :- test_inst_details**

EXPECTED OUTPUT :-

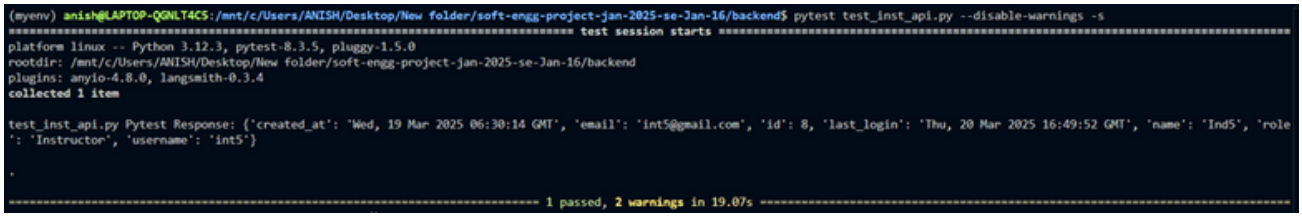
HTTP 200

PyTest Code :-

```python
def test_inst_details(client, auth_header):
    """Test fetching admin details API."""
    response = client.get("/instructor_details", headers=auth_header)

    assert response.status_code == 200
    json_data = response.get_json()
    print('Pytest Response:',json_data,'\n')
```

Actual Output :-



d.) Instructor - Top Support Queries (SUCCESS)

ENDPOINT : GET /topquery

- **Test Case Name :- test_top_support_queries**

EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_top_support_queries(client, auth_header):
    """Test the Top Support Queries API."""
    response = client.get("/topquery", headers=auth_header)

    assert response.status_code in [200, 404]  # Expect success or no queries found
    json_data = response.get_json()

    if response.status_code == 200:
        assert isinstance(json_data, list)  # The response should be a list of queries
        assert all("query_text" in query and "count" in query for query in json_data)
        print("Top Support Queries Response:", json_data)

    elif response.status_code == 404:
        assert json_data["message"] == "No queries found"
        print("No Queries Found:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_inst_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_inst_api.py Top Support Queries Response: [{'count': 3, 'query_text': 'axios'}, {'count': 2, 'query_text': 'what is axios'}, {'count': 2, 'query_text': 'how to print in python'}, {'count': 1, 'query_text': 'what is profit'}, {'count': 1, 'query_text': 'what is decorator function'}]
.

============================== 1 passed, 1 warning in 18.96s ===============================
```

e.) Instructor - Add Live Lessons (SUCCESS)

ENDPOINT : POST /add_livesession

- Test Case Name :- test_top_support_queries

INPUT :-

```
{
  "course_id": 1,
  "yt_link": "https://www.youtube.com/watch?v=dQw4w9WgXcQ",
  "description": "Live session on Python Basics"
}
```

EXPECTED OUTPUT :-
HTTP 200, JSON { "message": "Live session added successfully" }

PyTest Code :-

```python
def test_add_live_session(client, auth_header):
    """Test the Add Live Session API."""
    payload = {
        "course_id": 1,
        "yt_link": "https://www.youtube.com/watch?v=dQw4w9WgXcQ",
        "description": "Live session on Python Basics"
    }

    response = client.post("/add_livesession", json=payload, headers=auth_header)

    assert response.status_code in [201, 400]  # Expect success or missing fields error
    json_data = response.get_json()

    if response.status_code == 201:
        assert json_data["message"] == "Live session added successfully"
        print("Live Session Added:", json_data)

    elif response.status_code == 400:
        assert json_data["message"] == "Missing required fields"
        print("Missing Fields Error:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4C5:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_inst_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_inst_api.py Live Session Added: {'message': 'Live session added successfully'}
.

============================== 1 passed, 1 warning in 18.54s ===============================
```

5. CHATBOT API's

a.) ChatBot (SUCCESS)

ENDPOINT : POST /chat

- **Test Case Name :- test_chatbot_api**

INPUT :-

```
{
  "query": "What is recursion?"
}
```

EXPECTED OUTPUT :-
HTTP 200

PyTest Code :-

```python
def test_chatbot_api(client, auth_header):
    """Test the Chatbot API."""
    payload = {"query": "What is recursion?"}

    response = client.post("/chat", json=payload, headers=auth_header)

    assert response.status_code in [200, 400, 500]  # Expect success, bad request, or server error
    json_data = response.get_json()

    if response.status_code == 200:
        assert json_data["query"] == payload["query"]
        assert "response" in json_data
        assert "references" in json_data
        assert isinstance(json_data["references"], list)
        print("Chatbot Response:", json_data)

    elif response.status_code == 400:
        assert json_data["error"] == "Query cannot be empty"
        print("Empty Query Error:", json_data)

    elif response.status_code == 500:
        print("Server Error:", json_data)
```

Actual Output :-

```
(myenv) anish@LAPTOP-QGNLT4CS:/mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend$ pytest test_student_api.py --disable-warnings -s
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0
rootdir: /mnt/c/Users/ANISH/Desktop/New folder/soft-engg-project-jan-2025-se-Jan-16/backend
plugins: anyio-4.8.0, langsmith-0.3.4
collected 1 item

test_student_api.py Chatbot Response: {'query': 'What is recursion?', 'references': [{'doc_type': 'Transcript', 'subject': 'MAD2', 'week': 'Week-2'}, {'doc_type': 'Transcript', 'subject'
: 'MAD1', 'week': 'Week-5'}, {'doc_type': 'Lecture', 'subject': 'MAD1', 'week': 'Week-6'}], 'response': 'Hint:  Recursion is a programming concept where a function calls itself within it
s own definition.\n\nResource Guide: Check MAD1 Week-5 for more details.  The provided transcript discusses functions, return values, and mentions a "recursive function without a base ca
se," which suggests this week\'s material might offer a more complete explanation of recursion.'}
.

============================== 1 passed, 2 warnings in 24.17s ==============================
```

MILESTONE 6

FINAL SUBMISSION

FINAL SUBMISSION

1. Summary of Work Done in Different Milestones

Milestone 1: Identify User Requirements

To ensure the SEEK portal effectively meets users' needs, we conducted comprehensive user research. This involved:

- **User Research & Feedback Collection:** Gathered insights from students, faculty, and academic advisors to understand their expectations, pain points, and desired features for the AI-driven guidance system.
- **Scenario Development:** Created detailed user scenarios, illustrating interactions with and without GenAI integration, to assess the impact of AI on usability and efficiency.
- **Needs Assessment:** Identified essential features, such as course recommendations, academic planning assistance, and query resolution, to align with user expectations.

These findings provided a strong foundation for designing an intuitive and effective AI agent for academic guidance.

Milestone 2: User Interfaces

To design an intuitive and user-friendly experience for the SEEK portal, we focused on structuring the application's interface through:

- **User Flow Design:** Mapped out detailed user flows to visualize how students, faculty, and administrators interact with the system, ensuring a seamless navigation experience.
- **Wireframing & Prototyping:** Developed low-fidelity wireframes to outline key interface elements and page layouts, providing a blueprint for UI/UX design.

- **User Story Development:** Defined user stories covering both student and admin perspectives, ensuring that the interface caters to diverse user needs while maintaining efficiency and accessibility.

These steps helped create a clear vision for the SEEK portal's interface, enhancing usability and engagement.

Milestone 3: Scheduling and Design

To ensure a structured and efficient development process, we focused on meticulous planning and system design through:

- **Sprint-Based Project Timeline:** Established a structured development schedule using an agile sprint-based approach, ensuring iterative progress and timely deliverables.
- **Component & System Design:** Developed detailed project components, including functional modules and interactions, to streamline the implementation process.
- **Class Diagrams & Architecture Planning:** Created comprehensive class diagrams to define the system's architecture, data flow, and relationships between key components.

These steps provided a clear roadmap for the SEEK portal's development, ensuring a well-organized and efficient implementation process.

Milestone 4: Scheduling and Design

To enable seamless interaction between the SEEK portal's frontend and backend, we designed and implemented essential API endpoints, including:

- **Designed and implemented core API endpoints for** user authentication, course management, and GenAI-driven interactions to enable seamless system functionality.
- **Optimized data handling and security measures** to ensure efficient request processing, secure access control, and scalable system performance.

These API endpoints serve as the backbone of the SEEK portal, enabling smooth functionality and scalable integrations.

Milestone 5: Test Cases

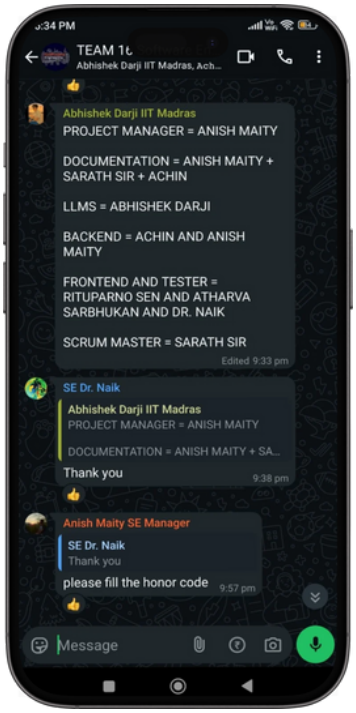
To ensure the SEEK portal functions reliably and efficiently, we developed and executed comprehensive test cases, covering various testing levels:

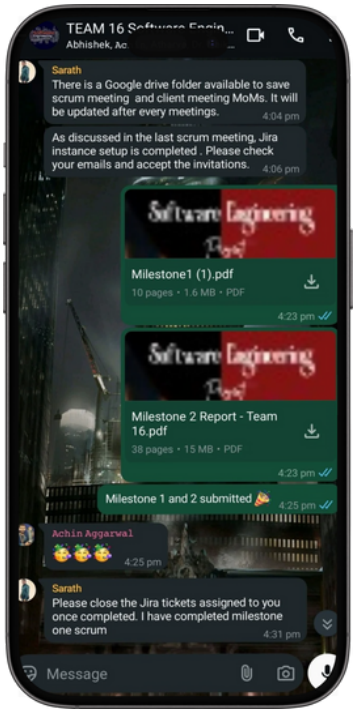
- **Unit Testing:** Verified individual components and functions to ensure they perform as expected.
- **Integration Testing:** Tested API interactions between frontend and backend to confirm seamless data flow.
- **System Testing:** Assessed the entire SEEK portal for performance, usability, and security.
- **Error Handling & Optimization:** Identified and resolved bugs, ensuring robustness and reliability.

These tests ensured that all implemented features function correctly and provide a smooth user experience.

2. Code Review, Issue Tracking and Reporting

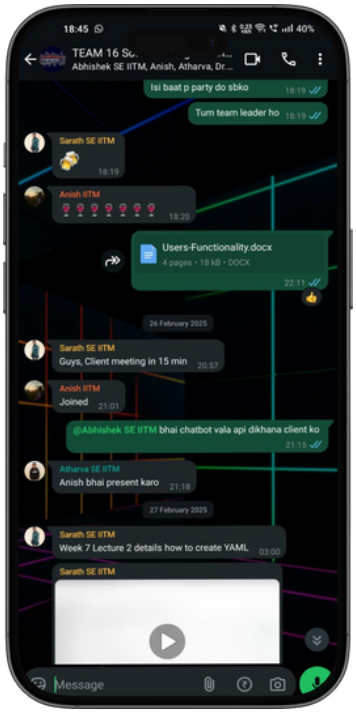
We have created a WhatsApp group for constant communication and easy access to review or code. Below are a few screenshots attached to view

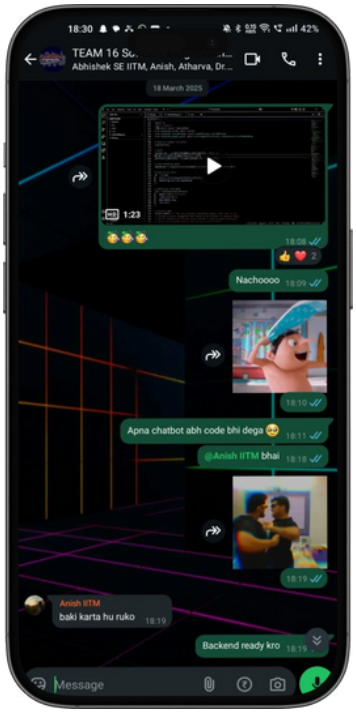




2. Code Review, Issue Tracking and Reporting

We have created a WhatsApp group for constant communication and easy access to review or code. Below are a few screenshots attached to view





3. Implementation Details of the Project

Backend Implementation:

- Developed a **RESTful API** using **Flask** to handle key functionalities, including user authentication, course management, and GenAI-powered interactions.
- Implemented **role-based access control (RBAC)** to differentiate access levels for students, faculty, and admins.
- Integrated **GenAI-powered functionalities**, such as:
  a. Automated feedback on programming assignments using NLP-based analysis.
  b. Personalized study plan generation based on user preferences and past performance.
  c. Context-aware academic support, allowing students to ask course-related queries and receive AI-driven responses.
- Ensured **secure API interactions** using authentication mechanisms like **JWT/OAuth and HTTPS encryption**.

Frontend Implementation:

- Designed and developed a **reactive, user-friendly dashboard** using Vue to improve accessibility and usability.
- Integrated **real-time AI-based assistance**, enabling dynamic responses to user queries without page reloads.
- Created an **intuitive course browsing experience**, allowing students to access syllabi, assignments, and learning materials efficiently.
- Implemented a **notification system** to alert students about deadlines, feedback, and updates.
- Ensured **responsive design** for compatibility across desktops, tablets, and mobile devices.

Database Schema:

- Designed a **normalized relational database MySQL** to efficiently manage:
  a. **Users** (students, faculty, admins) with role-based attributes.
  b. **Courses** with structured modules, assignments, and materials.
  c. **Assignment submissions** and AI-generated feedback logs.
  d. **User interaction history** for AI-driven personalized recommendations.
- Optimized **database indexing and relationships** to ensure fast query execution and support AI-driven insights.
- Implemented **backup and recovery mechanisms** to prevent data loss and ensure reliability.

This robust implementation ensures that the SEEK portal remains scalable, efficient, and AI-driven, providing an enhanced academic guidance experience.

4. Tools and Technologies Used :-

Backend Technologies:

- **Flask & Flask-RESTful:** Used for developing RESTful API endpoints to handle core functionalities.
- **Flask SQLAlchemy:** Implemented as the ORM for efficient database management.
- **Pytest:** Used exclusively for unit and integration testing to ensure API reliability.
- **Swagger Editor:** Utilized for API documentation, enabling clear and structured endpoint references.
- **PostMan:** A lightweight API testing tool used for manually verifying API responses and debugging issues.

Frontend Technologies:

- **Vue 3 CLI:** Chosen as the primary frontend framework for building a reactive.
- **JavaScript:** The primary scripting language for frontend logic and interactions.
- **Bootstrap:** Applied for UI styling, ensuring a responsive and visually appealing design.
- **HTML & CSS:** Used for structuring and styling the user interface.

- **Axios & Fetch API:** Employed for making asynchronous HTTP requests to the backend API.

General Technologies Used:

- **GitHub:** Utilized for version control, code management, tracking issues, reviewing changes, and collaborative development.
- **Canva:** Used for designing reports, presentations, and documentation visuals.
- **Jira:** Implemented as the project management tool for tracking tasks, sprint planning, and progress monitoring.
- **draw.io (app.diagrams.net):** Used for creating UML diagrams, system architecture designs, and flowcharts.

These tools and technologies were carefully selected to ensure a scalable, efficient, and maintainable implementation of the SEEK portal.

5. Instructions to run our application :-

Getting Started // Prerequisites

To run our application on your local device, you will need to have the following installed:

Python 3.10, Pip, Node.js

A. Creation of Virtual Environment

cd /mnt/c/Users/.....(path to our project folder)
cd soft-engg-project-jan-2025-se-Jan-16/
cd Backend/
python3 -m venv venv

B. Start the virtual environment

source venv/bin/activate

C. Package Installation

Install the required packages inside venv
pip install -r requirements.txt

D. Install node modules

cd /mnt/c/Users/.....(path to our frontend folder)
cd soft-engg-project-jan-2025-se-Jan-16/
cd Frontend/
npm install

E. Start the frontend

npm run dev

F. Start the backend

cd /mnt/c/Users/.....(path to our backend folder)
cd soft-engg-project-jan-2025-se-Jan-16/
cd Backend/
python3 main.py (windows)
python3 main.py (mac)

Open the app in browser by navigating at http://localhost:5173/

Software Engineering

Project

TEAM 16

PROJECT BY

Anish Maity (21f3000417)
Achin Aggarwal (22f1001390)
Atharva Sarbhukan (22f1000533)
Abhishek Darji (22f1000678)
Sarath Sasidharan Pillai (22f1000152)
Rituparno Sen (21f1004275)
Dr. Ambrish Naik (22f2000424)

Thank YOU !