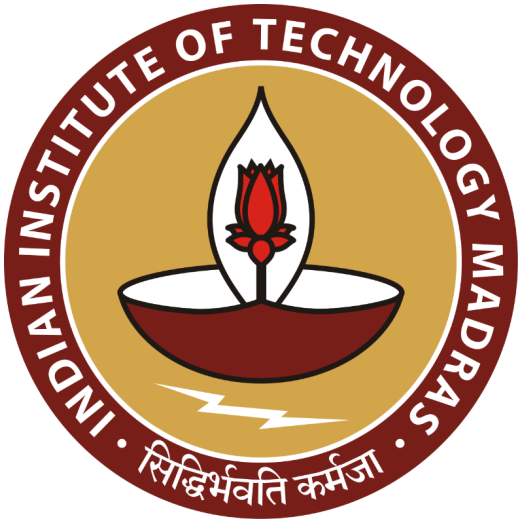
Software Engineering Project

January 2025 Term - Team 24

Milestone - 6



Submitted by:

Rajarshi Roy (21f3000348)

Rushil Gupta (21f1006728)

Prakhar Singh (21f3001320)

Kanishka Sharma (21f2000766)

Poorvi Choudhary (21f3002396)

Soumyadeep Dhali (22f1001762)

Abhinash Kumar Sinha (21f2001350)

IITM BS Degree Program

Milestone - 1

User Requirements

The goal is to identify user requirements for the SEEK learning portal, with and without GenAI integration. This includes understanding primary, secondary, and tertiary users' needs and crafting user stories to guide AI agent development, enhancing learning while promoting academic integrity and independence.

1. User Categories: To build a system that caters to diverse needs, users are categorized into three groups based on their interaction with the portal:

(a) Primary Users - Students: IITM BS Students are the primary users of the Seek portal who register for the courses, access course related materials, and have to submit their assignments.

(b) Secondary Users - Instructors: These are the individuals who coordinate the projects of students as well as provide and release the weekly assignments and weekly supplementary contents on the platform. They also assist students with their difficulties along the duration of the courses.

(c) **Tertiary Users- Administrators:** They have higher privileges and functionalities to handle any urgent issues and maintain the portal. Additionally, also resolving the queries that students might face.

2. User Stories: Below are the user stories that outline the needs, both with and without the integration of GenAI into the SEEK learning portal. These stories describe the functionalities that each user group expects to enhance their experience on the platform.

Without GenAI

(a) Primary Users: (Students)

- **As a Student,** I want to access course materials such as videos, notes, and assignments in an organized manner, so that I can study efficiently.
- **As a Student,** I want to submit assignments and take quizzes online, so that I can be evaluated on my understanding of the course.
- **As a Student,** I want to participate in discussion forums and ask questions, so that I can clarify my doubts and collaborate with peers.
- **As a Student,** I want to receive reminders about upcoming deadlines, so that I can manage my time effectively and avoid missing submissions.

(b) Secondary Users: (Instructors)

(i) Instructors:

- **As an Instructor,** I want to upload and organize weekly assignments and supplementary content, so that students can access them easily.

(c) Tertiary Users: (Administrators)

(i) Administrators

- **As an Administrator,** I want to manage user accounts and permissions, so that the portal remains secure and accessible to authorized users.
- **As an Administrator,** I want to resolve technical queries and issues faced by students, so that their learning experience is not disrupted.

With GenAI

(a) Primary Users: (Students)

- **As a Student,** I want the AI agent to suggest personalized learning strategies based on my progress and performance, so that I can improve on my weaker areas.
- **As a Student,** I want the AI agent to recommend relevant and credible course materials such as videos, tutorials, or documentation without giving me direct answers.

- **As a Student,** I want the AI agent to summarize lectures into bullet points so that I can get a brief overview of the topic.

(b) Secondary Users: (Instructors)

(i) Instructors: As an Instructor, I want the AI agent to provide insights into student performance and common challenges so that I can tailor my teaching approach and provide targeted support.

(c) Tertiary Users: (Administrators)

(i) Administrators

- **As an Administrator,** I want the AI agent to analyze learning patterns and engagement metrics across courses to identify improvement areas and enable data-driven decisions.

Key Features Of The AI Agent

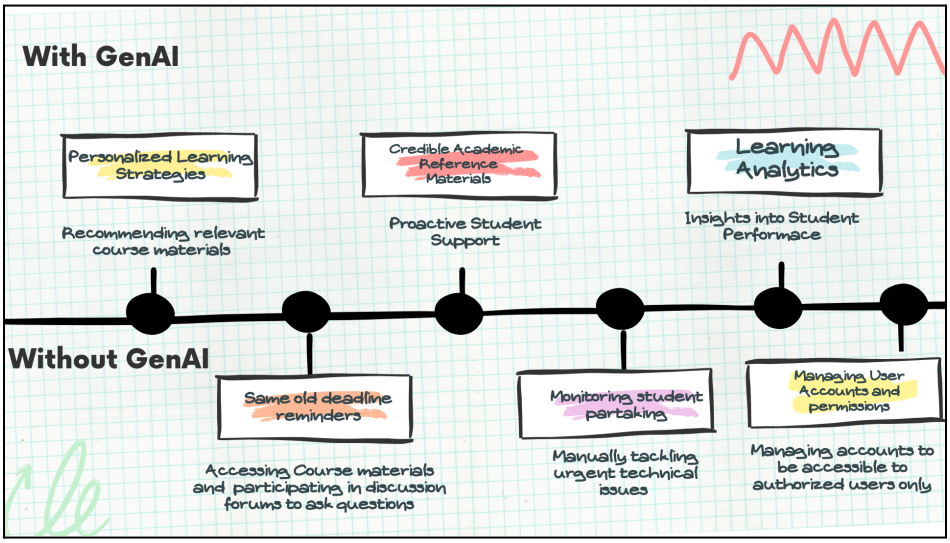


Figure 1: Learner's Journey

Milestone 2

Software Engineering Project

January 2025 Term - Team 24
Milestone - 2

Submitted by:
Abhinash Kumar Sinha (21f2001350)
Kanishka Sharma (21f2000766)
Poorvi Choudhary (21f3002396)
Prakhar Singh (21f3001320)
Rajarshi Roy (21f3000348)
Rushil Gupta (21f1006728)
Soumyadeep Dhali (22f1001762)

Storyboard:

Figure 2.a: Student Perspective

Figure 2.b: Student Perspective

Figure 2.c: Student Perspective

Figure 2.d: Student Perspective

Figure 3.a: Instructor Perspective

Figure 3.b: Instructor Perspective

Figure 4.a: Administrator Perspective

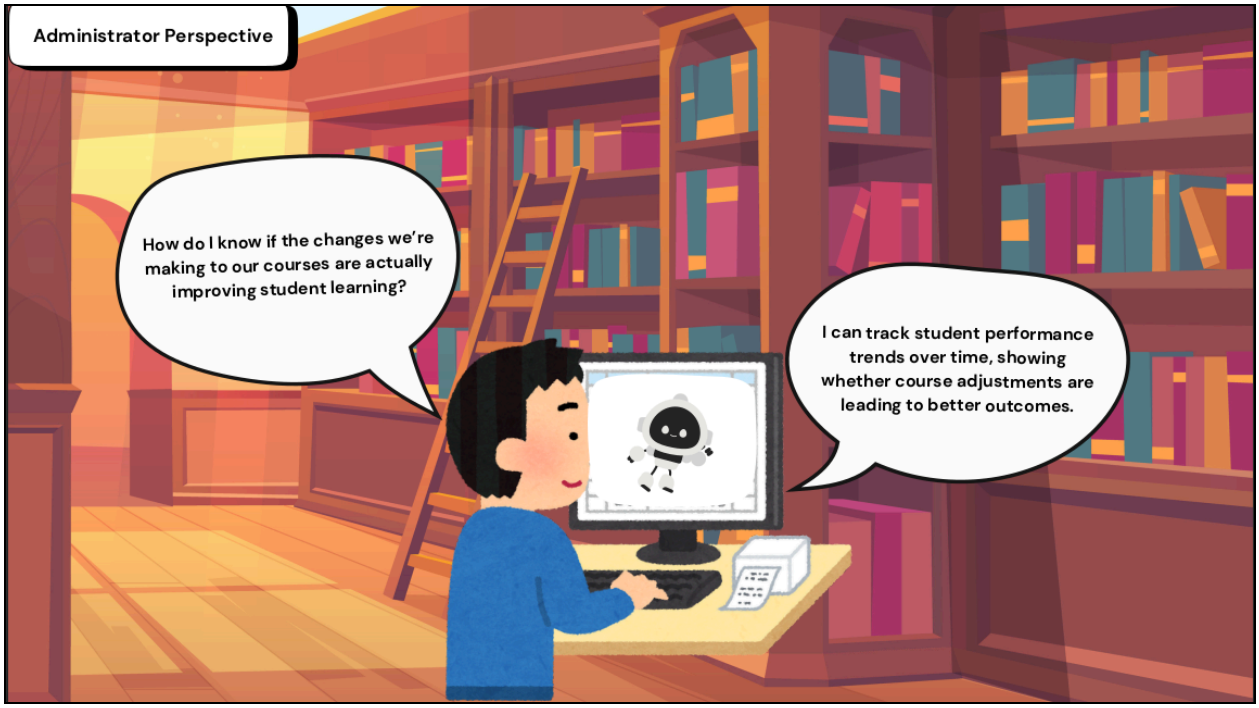


Figure 4.b: Administrator Perspective

Wireframes:

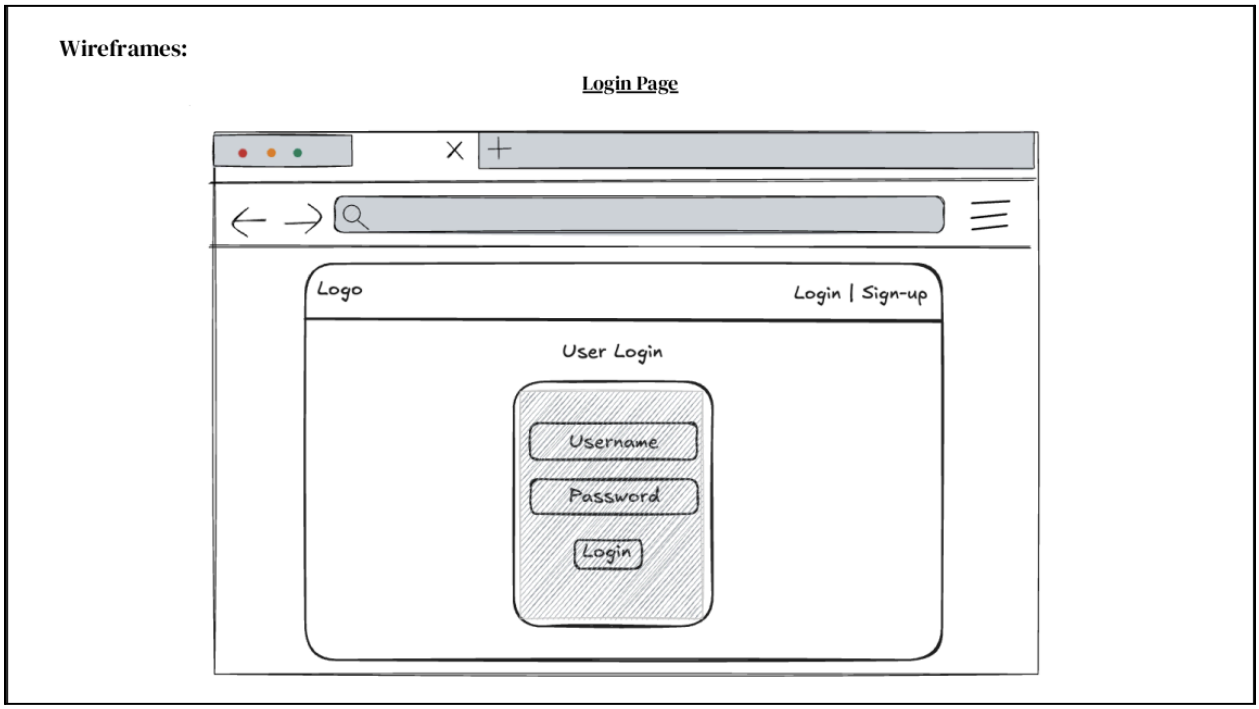


Figure 5.1: Login Page

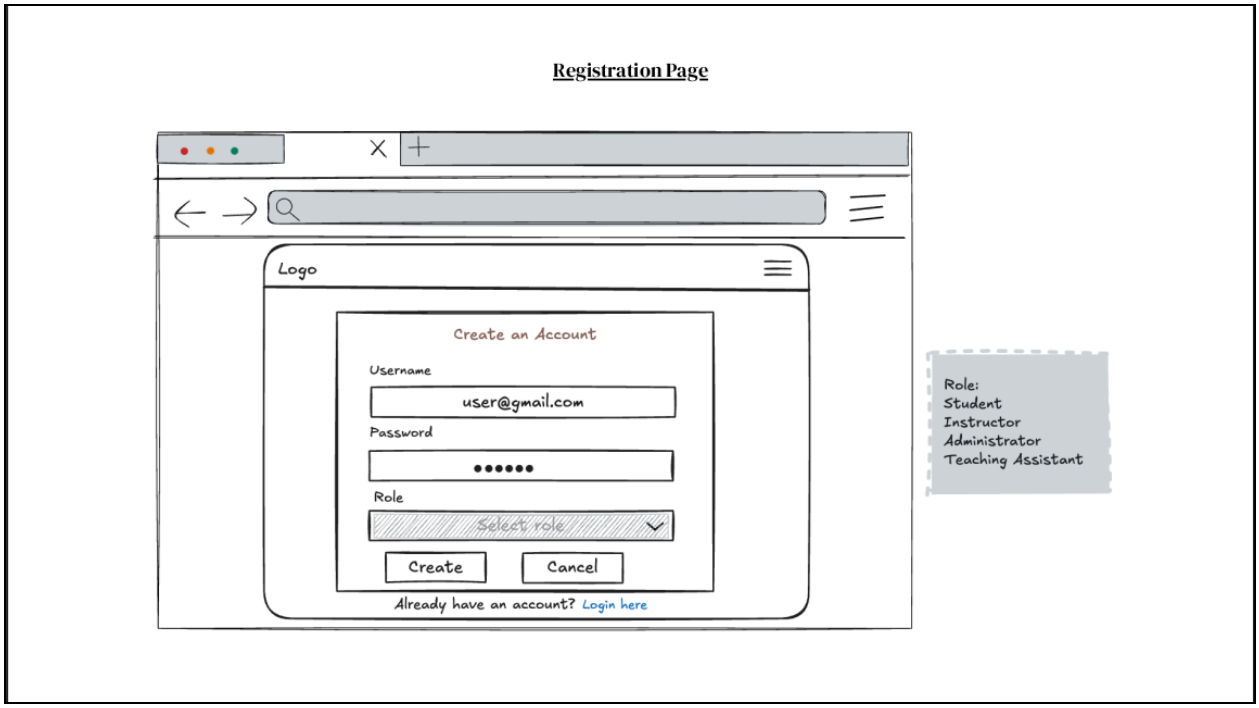


Figure 5.2: Registration Page

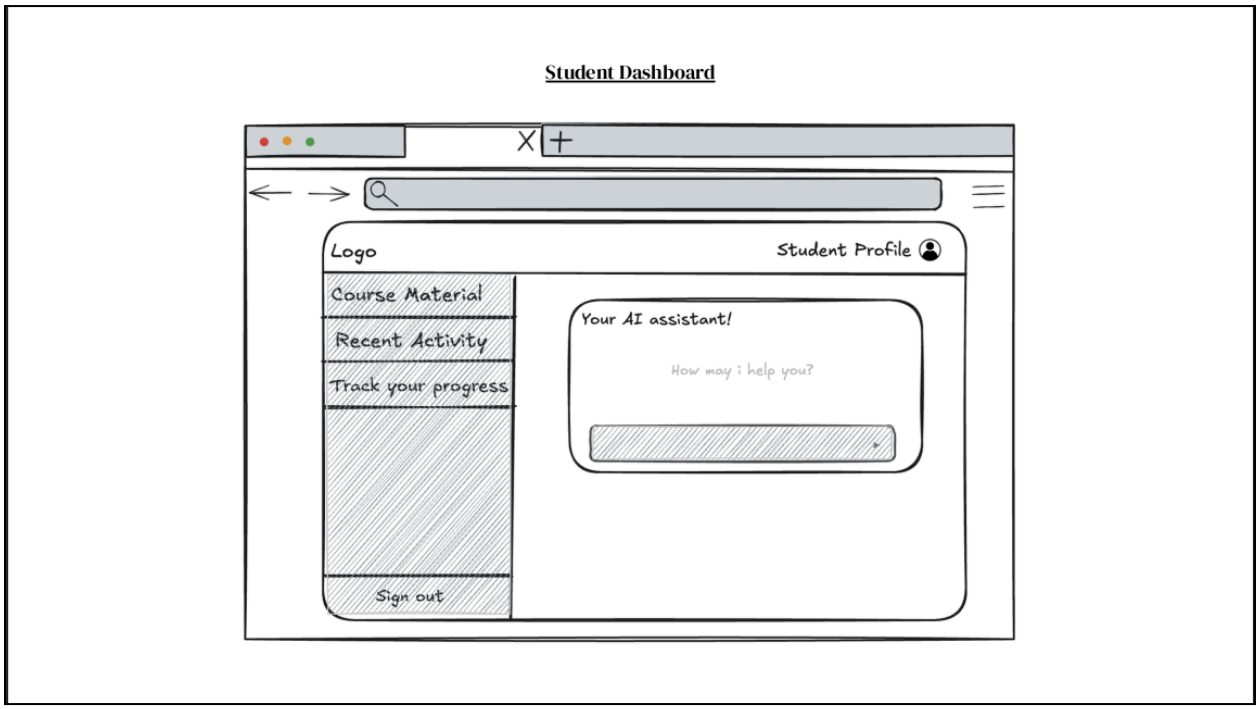


Figure 5.3: Student Dashboard

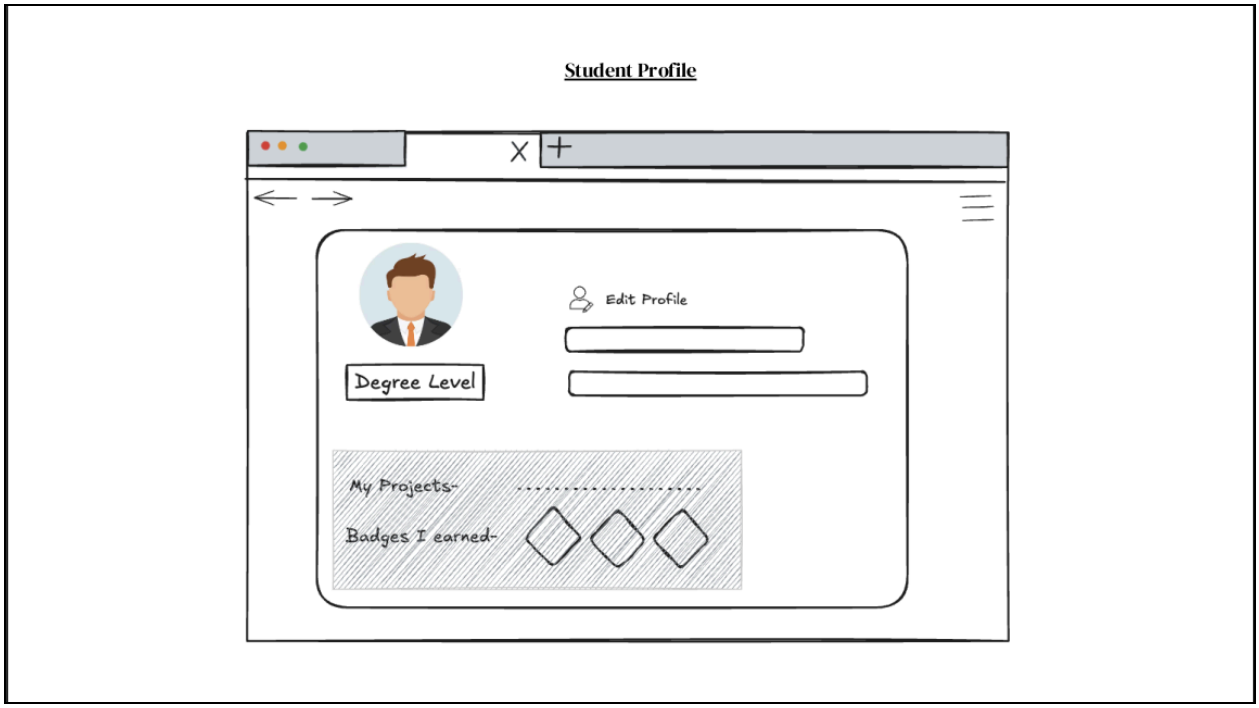


Figure 5.4: Student Profile

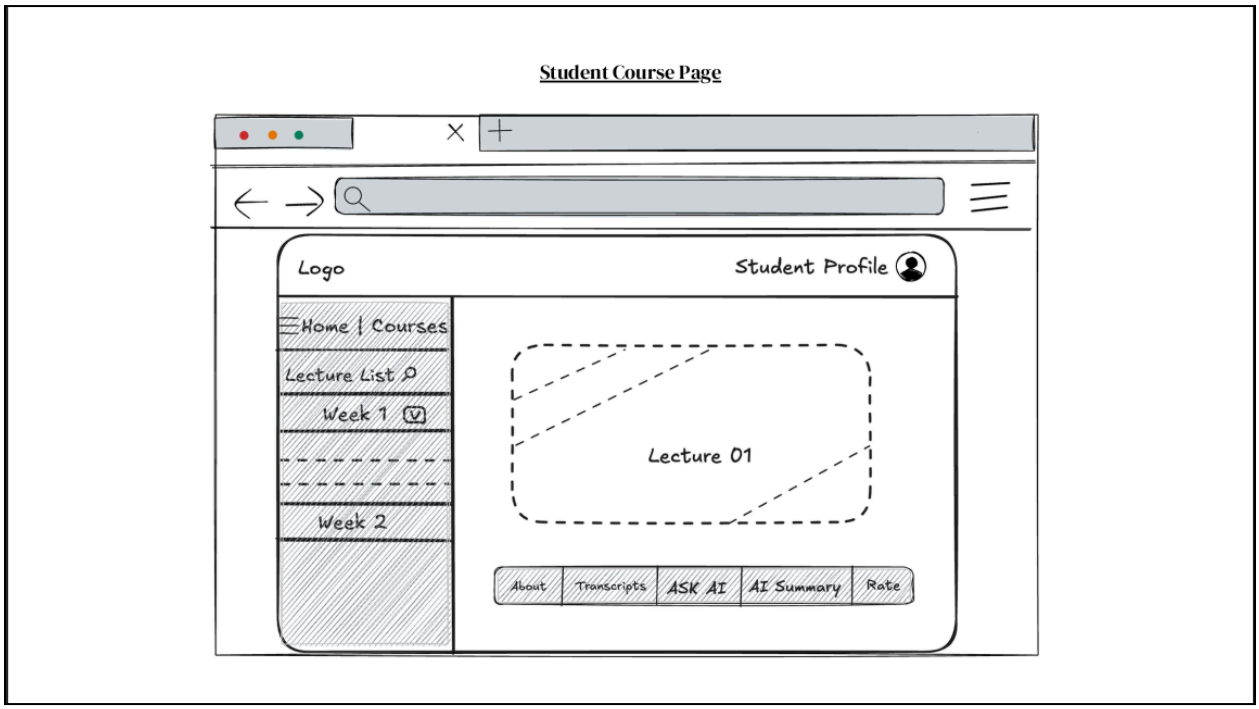


Figure 5.5: Student Course Page

Figure 5.6: Admin Dashboard

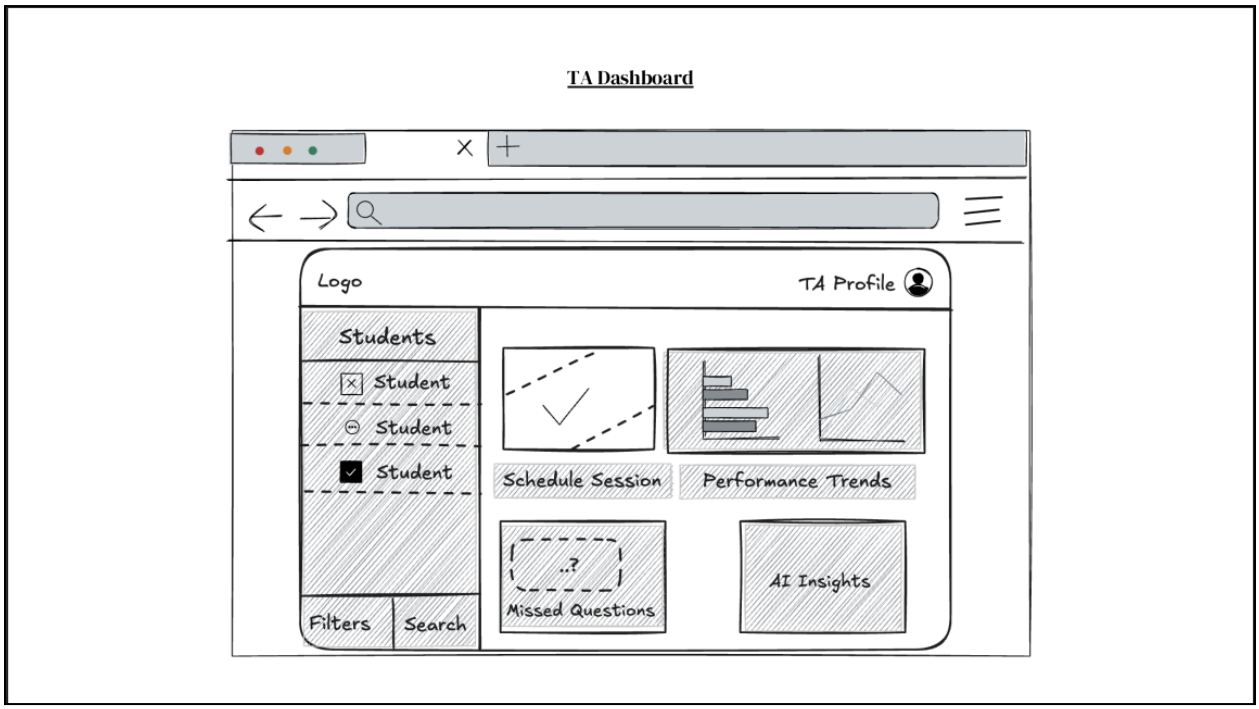


Figure 5.7: TA Dashboard

Milestone 3

1. Project Schedule

1.1 Objective: The objective is to establish a well-structured project timeline that ensures smooth execution and timely completion of all development phases. This involves:

- Breaking down the project into sprints to systematically develop and integrate key features.
- Aligning the development process with user stories identified in previous milestones to ensure functionality meets user needs.
- Implementing project tracking tools to monitor individual and team contributions, task completion, and overall progress.
- Facilitating efficient team collaboration through clear task assignments, milestone deadlines, and structured sprint planning. By following this approach, the project will progress in a phased manner, ensuring continuous testing, refinement, and alignment with the end goals.

1.2 Sprint Schedule:

● **Sprint 1 (Weeks 1-2) (Jan 15–28, 2025):** Team introduction, task distribution for Milestone 1, and initial project management discussions. Setup development environment and implement user authentication.

● **Sprint 2 (Weeks 3-4) (Jan 29–Feb 9, 2025):** Milestone-2 task allocation. UI development by Kanishka, Poorvi, and Abhinash; Wireframing by Rajarshi and Soumyadeep; Backend setup by Rushil and Prakhar. Develop core features for students, including access to lecture materials.

● **Sprint 3 (Weeks 5-6) (Feb 10–20, 2025):** Implement AI-driven features for students, including personalized learning strategies based on progress, recommendations for relevant course materials, and lecture summarization into bullet points. Develop AI-powered insights for instructors to monitor student performance and identify struggling students.

● **Sprint 4 (Weeks 7-8) (Feb 21–Mar 2, 2025):** Enhance AI-driven functionalities by refining

personalized learning recommendations and improving content suggestions. Integrate real-time analytics for instructors to provide targeted support.

- **Sprint 5 (Weeks 9-10) (Mar 3–18, 2025):** Conduct testing and debugging. Implement AI-driven analytics for administrators to track learning patterns, engagement metrics, and areas of improvement.

- **Sprint 6 (Weeks 11-12) (Mar 19–30, 2025):** Final review, deployment readiness, and system-wide refinements to ensure stability and usability. AI features are fine-tuned based on feedback from students, instructors, and administrators.

1.3 Scrum Board and Gantt Chart:

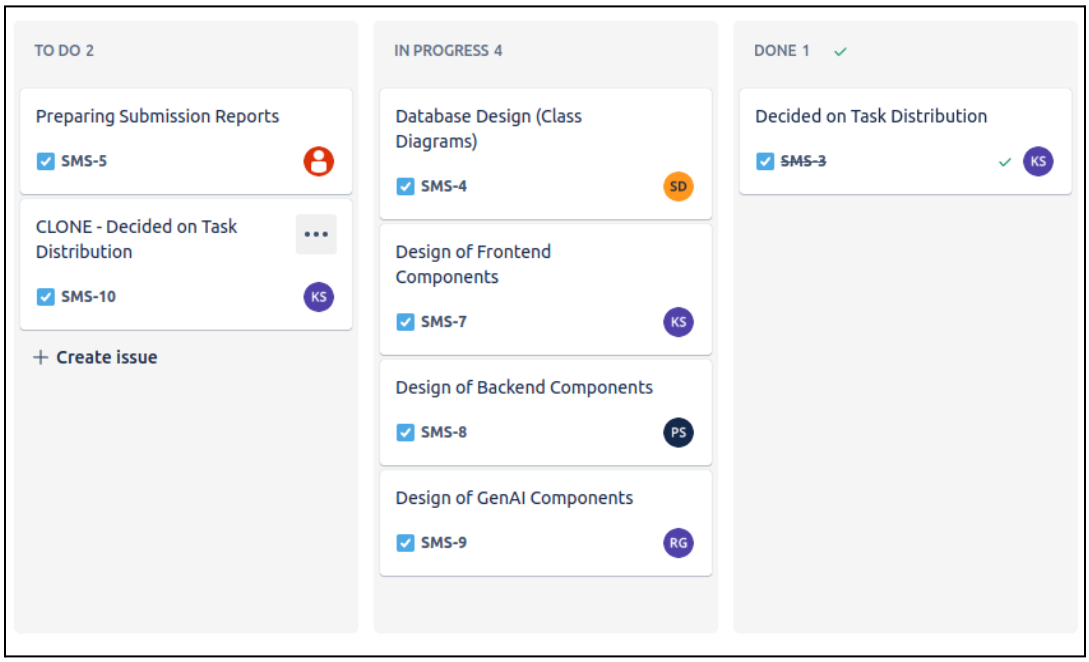


Figure 1.3.a Scrum Board

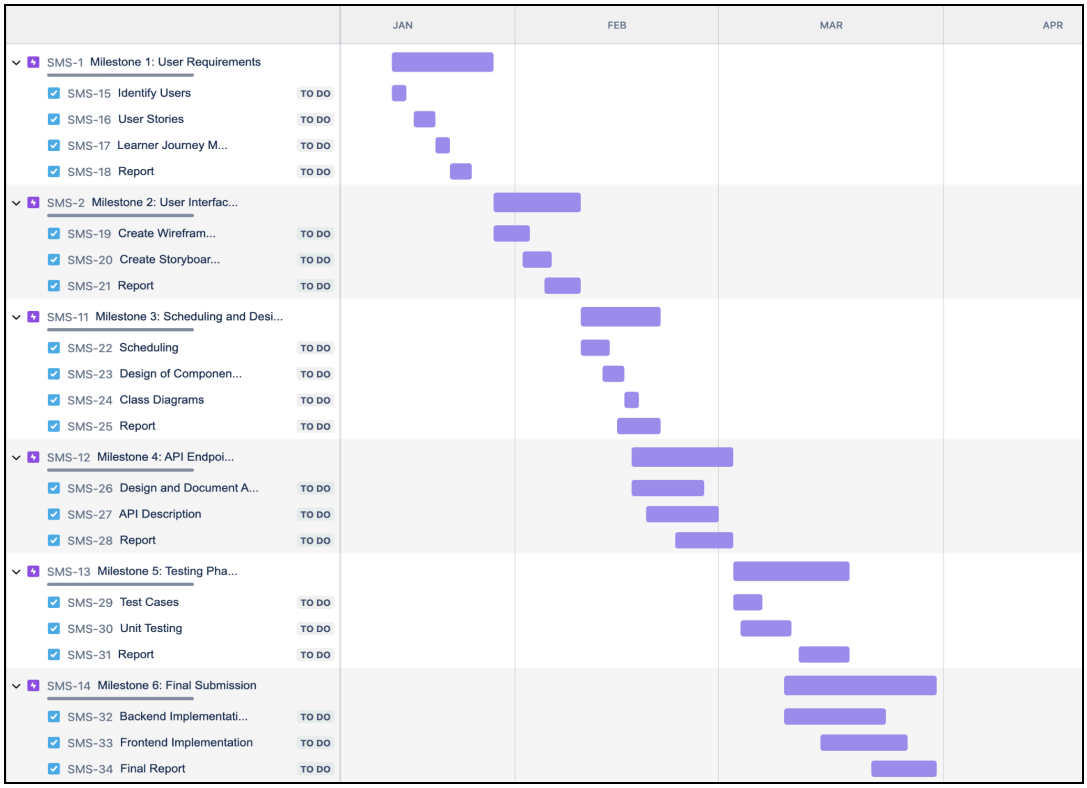


Figure 1.3.b Gantt Chart

1.4 Task Distribution:

SERIAL NO.	MILESTONE	SUB-TASKS	SPRINT	ASSIGNED TO
1	Milestone 1	User Identification	1	Rushil, Prakhar
		User Stories	1	Poorvi, Soumyadeep
		Without/With GenAI	1	Kanishka, Rajarshi
		Learner's Journey Through Diagram	1	Kanishka
2	Milestone 2	Storyboard	2	Rajarshi, Soumyadeep
		Wireframes	2	Kanishka, Poorvi
3	Milestone 3	Project Scheduling & Management	3	Kanishka, Rajarshi
		System Design & Architecture	3	Rushil, Prakhar
		UI Development	3	Rushil, Prakhar, Abhinash
		Documentation	3	Poorvi, Kanishka, Rajarshi, Soumyadeep
4	Milestone 4	API Endpoint Development	4	Rushil, Prakhar, Abhinash
		API List and Documentation	4	Kanishka, Poorvi, Soumyadeep
		Code Implementation	4	Rushil, Rajarshi, Prakhar
5	Milestone 5	Test Case Designs, Unit Testing	5	Abhinash, Prakhar
		API Testing	5	Rushil, Abhinash
		Drafting Report	5	Poorvi, Kanishka
6	Milestone 6	Code Quality and Process	6	Soumyadeep, Rushil, Prakhar
		Presentation Slides	6	Prakhar, Kanishka, Poorvi, Rajarshi
		Presentation Video	6	Rushil

Figure 1.4 Task Distribution

2. Project Scheduling Tools

We are utilizing Jira to efficiently track our project's progress and ensure that all tasks are completed on time. By leveraging Jira's robust task management, sprint planning, and reporting features, we can stay organized, monitor deadlines, and maintain seamless collaboration across the team.

3. Design of Components

3.1 API Components:

(a) User Authentication API: This API manages the login and registration processes, handling authentication requests, verifying user credentials, and ensuring secure system access. It plays a vital role in maintaining the security and integrity of user sessions.

(b) User Support API: Designed to offer real-time assistance, this API processes user queries, interacts with the GenAI system to generate relevant suggestions and solutions, and delivers prompt support to users.

(c) Course Management API: This API facilitates the retrieval and modification of course-related content, including details, assignments, and quizzes. It enables frontend components to access course materials and supports the submission of new or updated resources.

3.2 Frontend Components:

(a) User Authentication (Login & Registration): This module manages user access by providing secure authentication and authorization. It includes forms for new users to sign up and for existing users to log in safely.

(b) User Dashboard: Acting as the central hub, the dashboard presents an overview of enrolled courses, assignments, and progress. It is designed to offer quick navigation to essential features and actions.

(c) AI-Enhanced Content Viewer: This feature transforms traditional course materials into interactive learning experiences using GenAI. It enables dynamic engagement through

interactive quizzes, simulations, and AI-powered insights.

(d) User Profile Management: This feature allows users to update their personal information, customize their profiles, and showcase their work by adding projects. It enhances personalization and provides a space for users to manage their academic or professional portfolio.

3.3 Backend Components:

(a) User Access & Authentication: This module manages user accounts, handling authentication, authorization, and security. It ensures safe storage of user data, enforces role-based access control, and provides essential features like password recovery and profile updates.

(b) Course Content Management: This component oversees the organization and maintenance of course-related materials, including lectures, assignments, and quizzes. It enables users to create, edit, and remove content, ensuring access to the most updated learning resources.

(c) Automated Feedback System: Utilizing GenAI models, this feature generates detailed feedback for programming submissions. It assesses user code, identifies errors, and provides actionable suggestions to enhance learning and debugging.

(d) Intelligent Problem-Solving Assistant: This system offers real-time guidance and support for problem-solving tasks. By leveraging GenAI, it provides hints, step-by-step recommendations, and troubleshooting insights to help users navigate coding challenges effectively.

4. Software Design

The aim is to architect the system components and create the initial design for the software. This involves defining key system components based on the user stories, preparing basic class diagrams, and ensuring the system is modular.

It also covers the development of frontend UI pages using HTML, CSS, and JavaScript (or relevant frameworks such as Vue.js or React.js). These pages are interconnected with proper redirection, and a Readme file will be provided for setting up and running the frontend code.

4.1 Class Diagram: (Click here for drive link)

5. Minutes of Scrum Meetings

SCRUM Meetings: Every Tuesday and Saturday, 21:00 - 22:00

Minutes from Scrum Meeting 1 (January 21, 2025): In the first SCRUM meeting, the team defined user types: primary (students), secondary (instructors, TAs), and tertiary (administrators). They discussed core requirements for the SEEK learning portal, with and without GenAI. Members were assigned to research AI integrations for the next meeting.

Minutes from Scrum Meeting 2 (January 25, 2025): The team refined user stories using SMART guidelines, finalized role-based functionalities, and assigned wireframing tasks. The development environment was set up, and backend structure plans were discussed.

Minutes from Scrum Meeting 3 (February 1, 2025): The team primarily focused on UI wireframing and the selection of suitable tools for designing the storyboards. The initial drafts of the wireframes were reviewed, and specific members were tasked with refining them based on feedback. Tasks were distributed among team members, ensuring that parallel progress could be made in both wireframing and storyboarding.

Minutes from Scrum Meeting 4 (February 4, 2025): The team refined wireframes and explored AI-driven learning enhancements. Core functionalities like course management and the student dashboard were initiated. Wireframes for key pages were finalized, and real-time AI insights for student support were discussed.

Minutes from Scrum Meeting 5 (February 8, 2025): The team decided on a structured project schedule and selected Jira as the primary tool for tracking progress and managing tasks

efficiently. Additionally, the team began working on discussing real-time AI support strategies for students, providing personalized assistance based on their learning patterns.

Minutes from Scrum Meeting 6 (February 11, 2025): The team structured the Milestone 3 report, covering project scheduling, Gantt charts, and task distribution. Front-end and backend development progressed, and Jira logs were reviewed for accurate reporting.

Minutes from Scrum Meeting 7 (February 15, 2025): The team refined software design documentation, gathered UI screenshots for alignment, and verified the development methodology and tracking tools.

Minutes from Scrum Meeting 8 (February 18, 2025): The team finalized the Milestone 3 report, ensuring formatting, consistency, and completeness. They reviewed sections, included project tools, UI snapshots, and design details, and cross-checked Jira logs for accuracy.

6. Snapshots For UI Pages: (Click here for drive link)

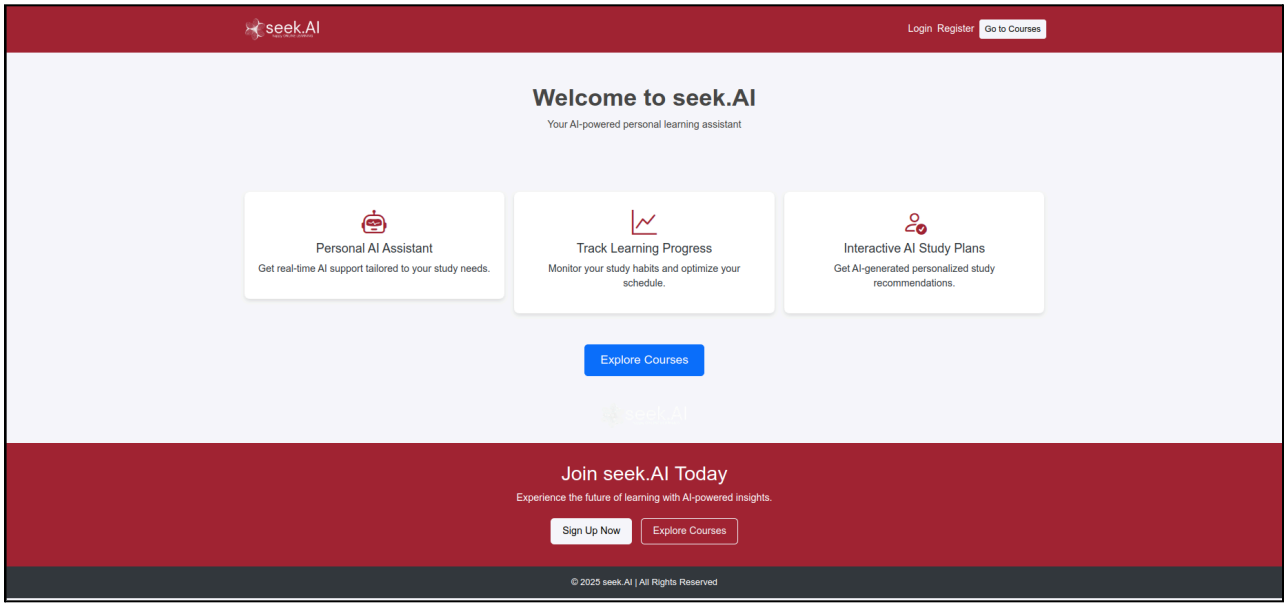


Figure 6.1 Home Page

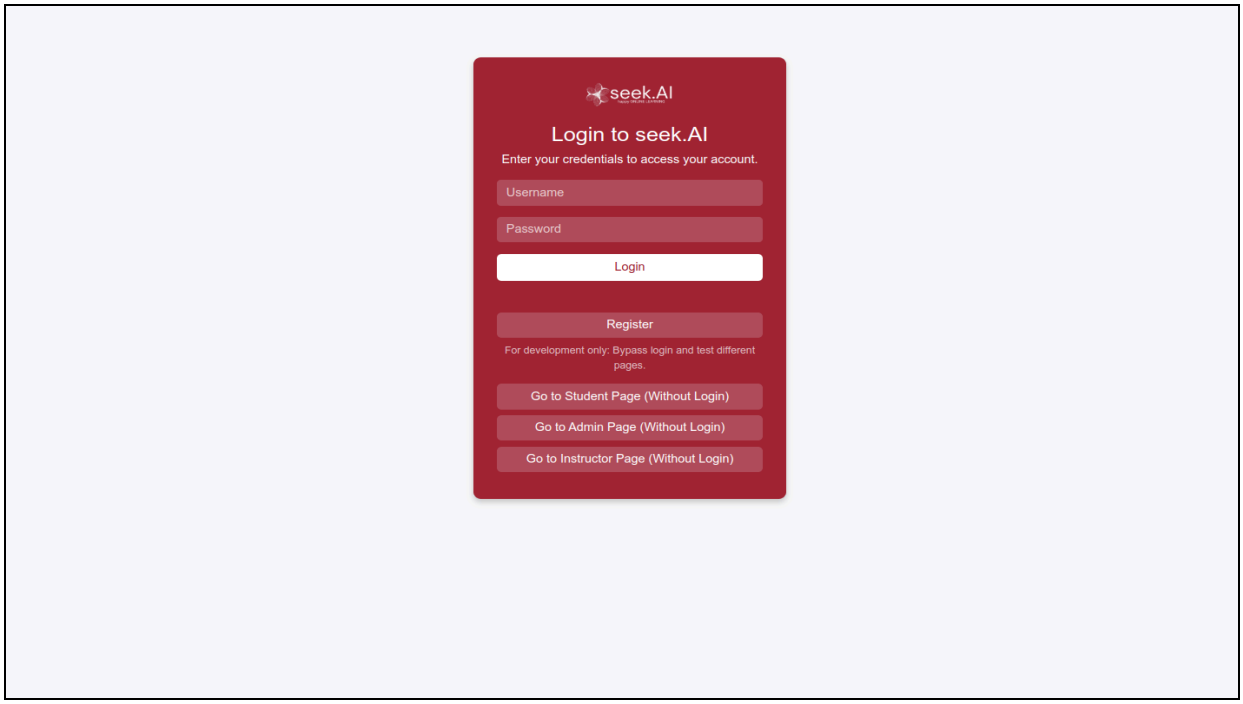


Figure 6.2 Login Page

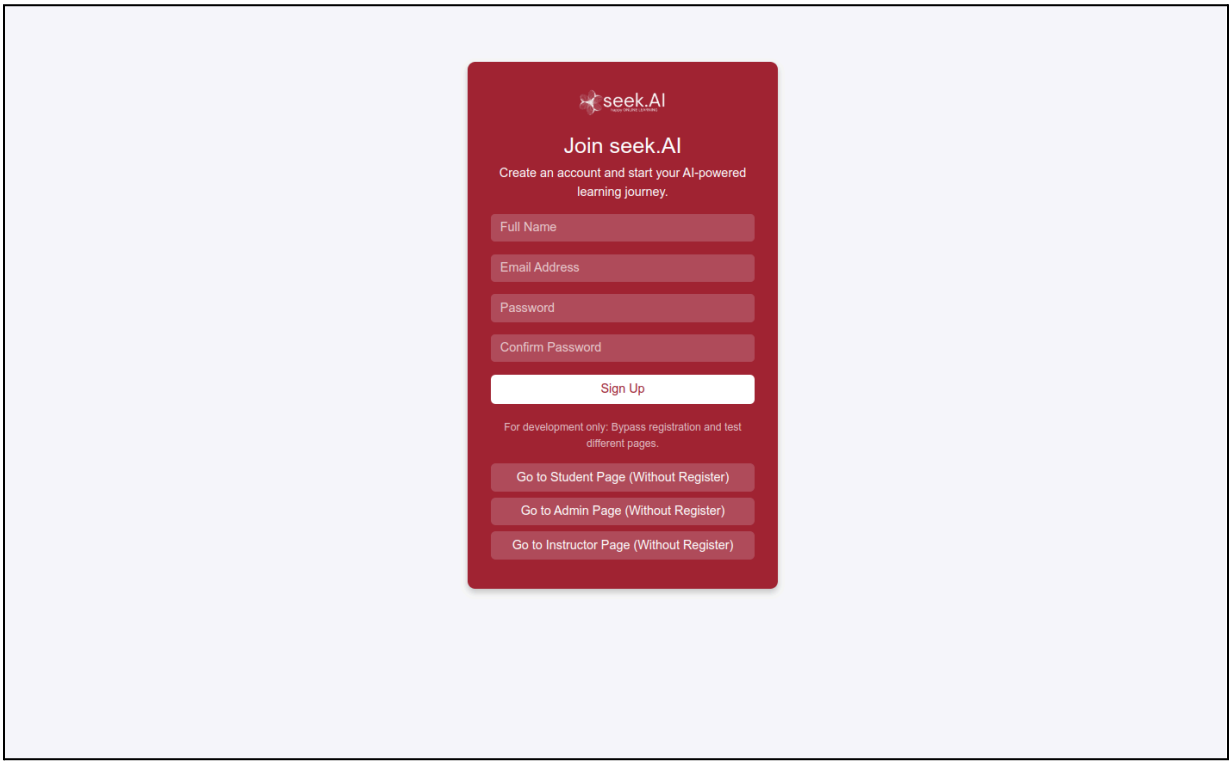


Figure 6.3 Registration Page

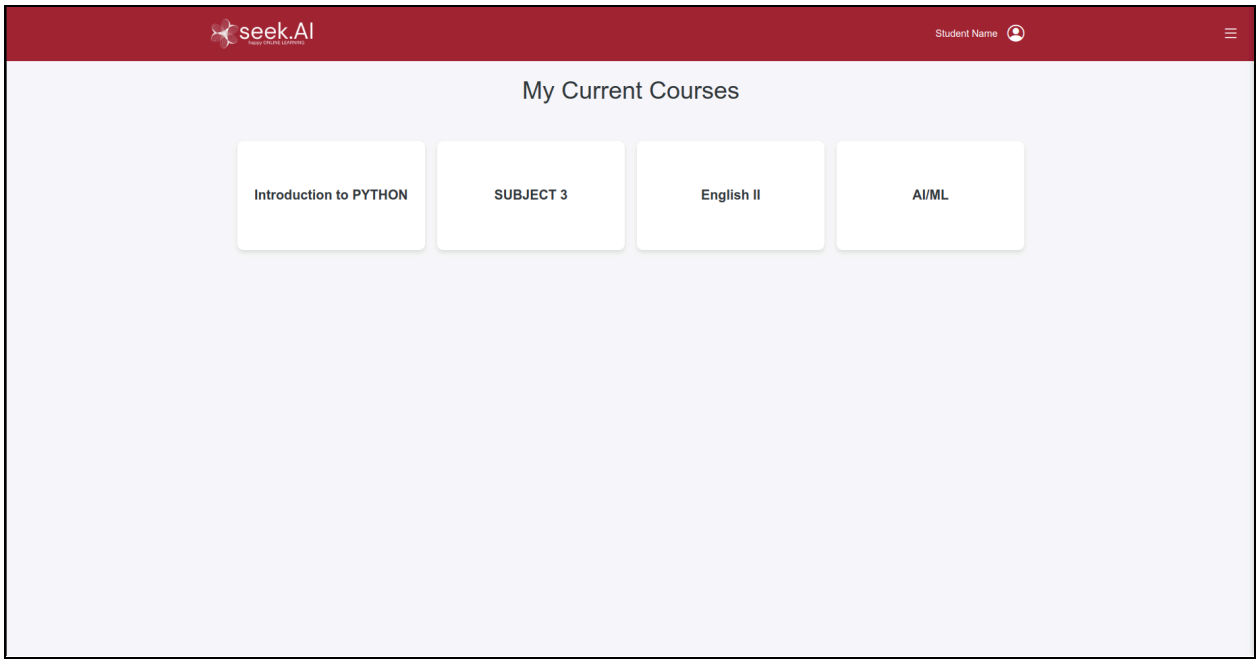


Figure 6.4 Student Dashboard (Index Page)

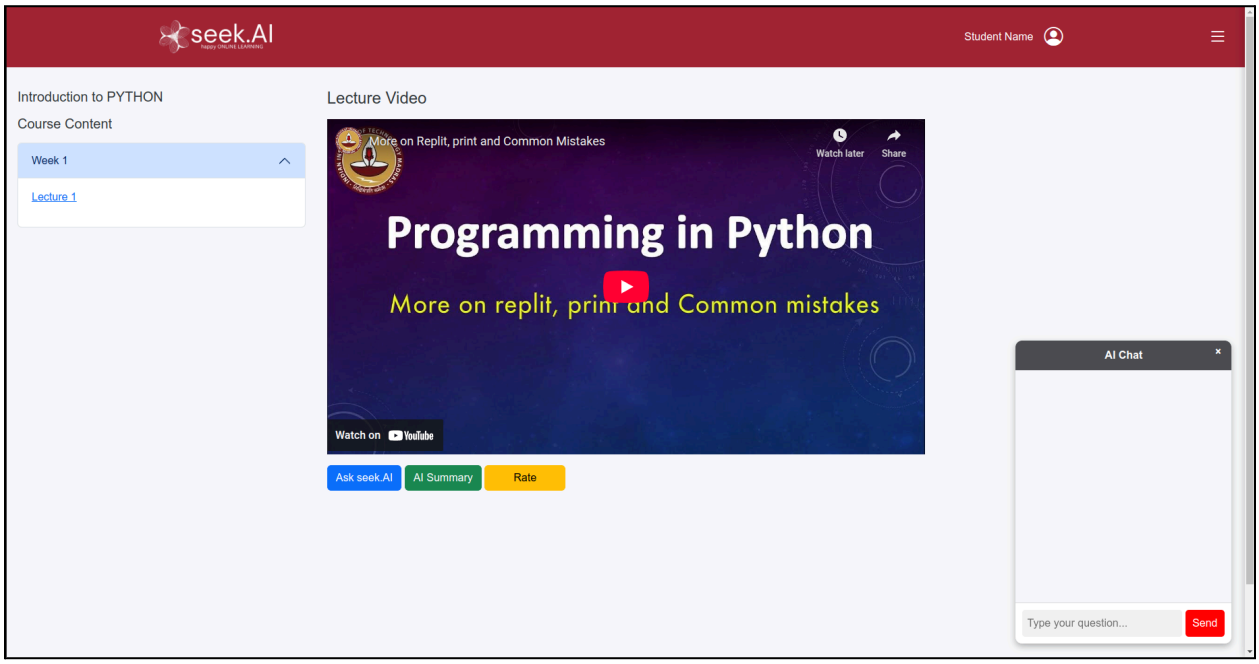


Figure 6.5 Student Course Page

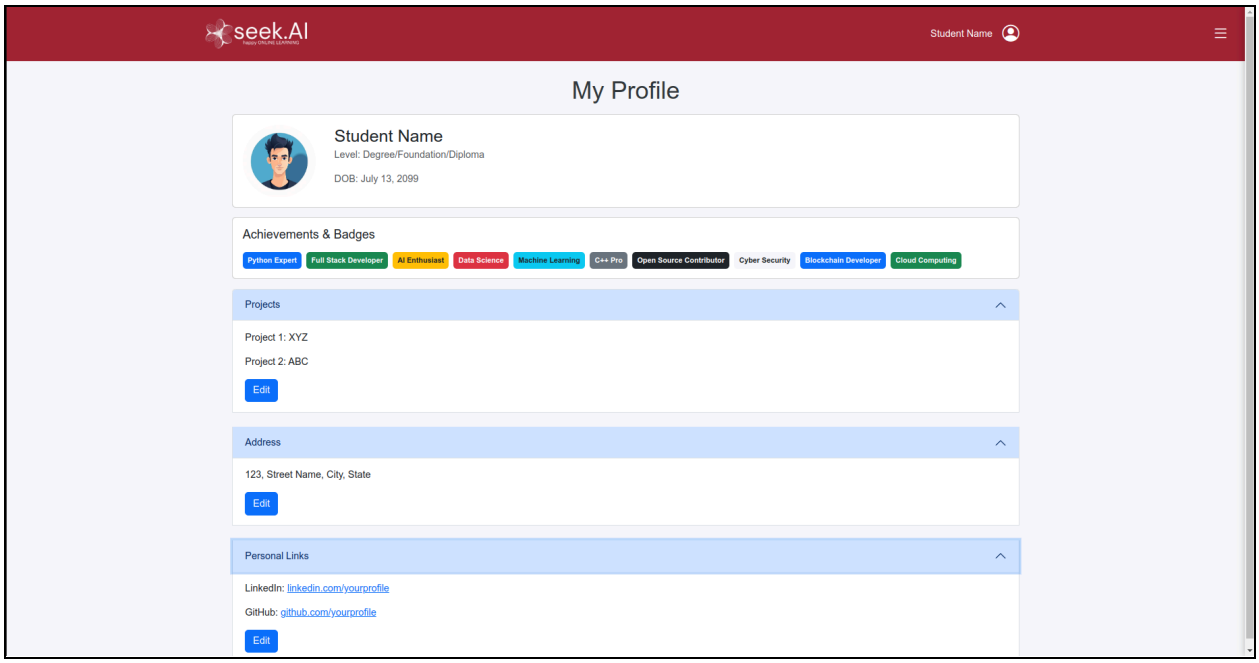


Figure 6.6 Student Profile Page

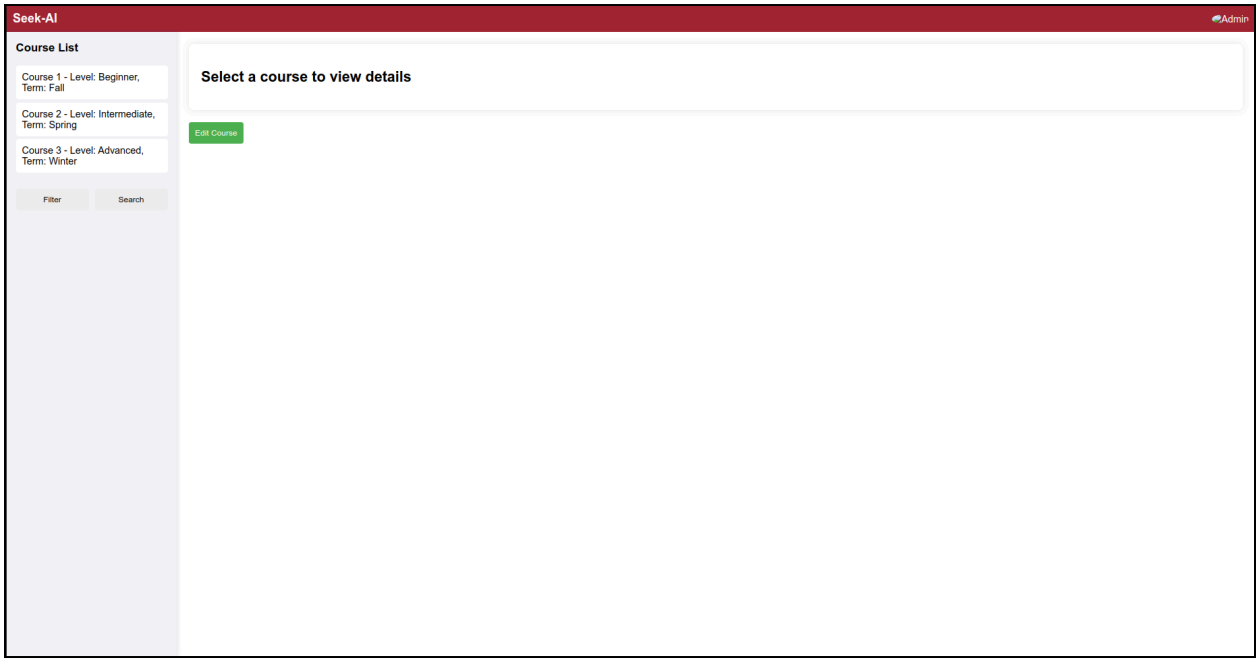


Figure 6.7 Admin Page

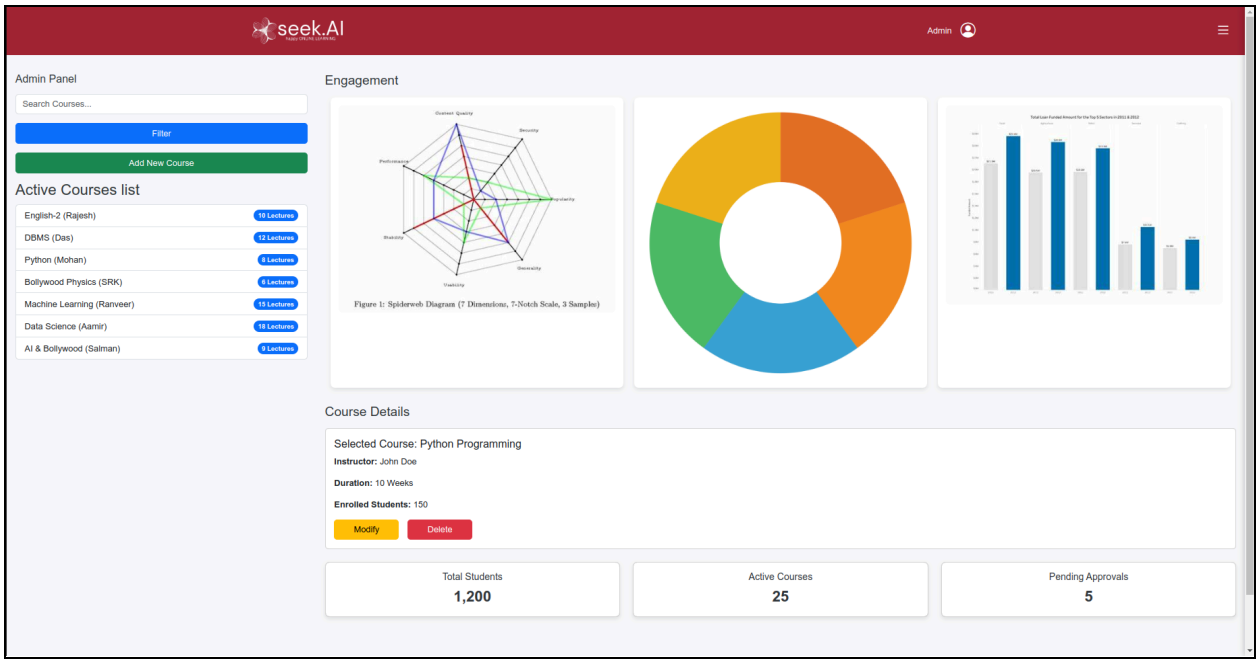


Figure 6.8 Admin Dashboard

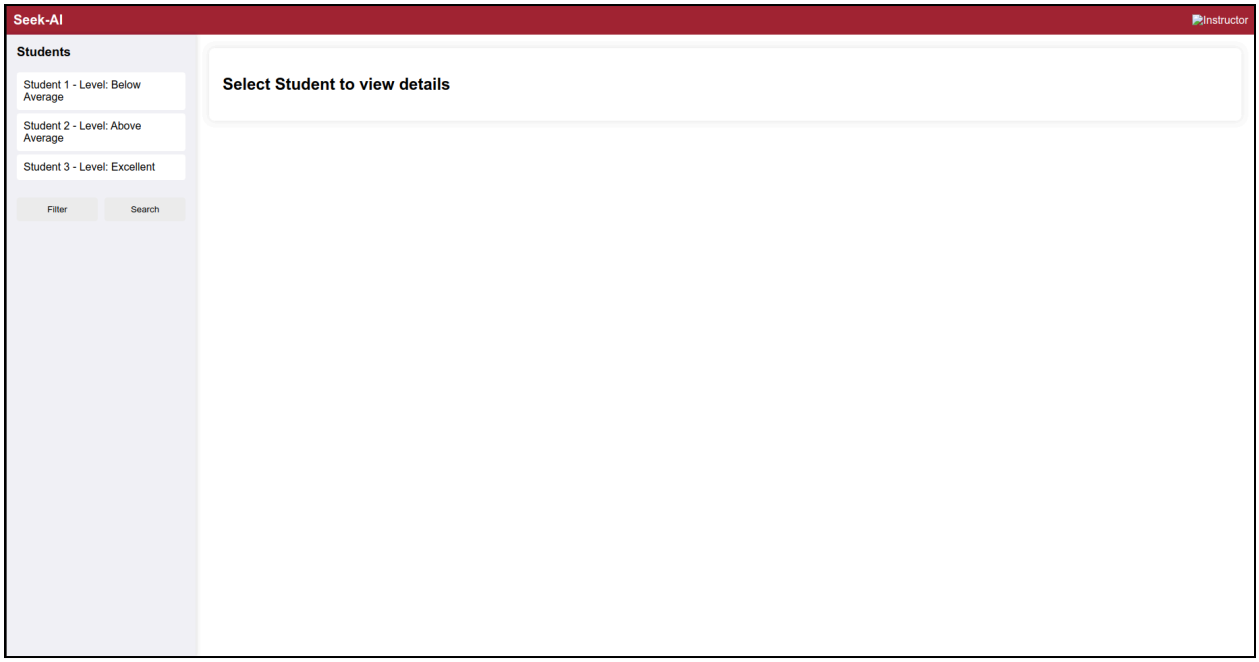


Figure 6.9 Instructor (Home Page)

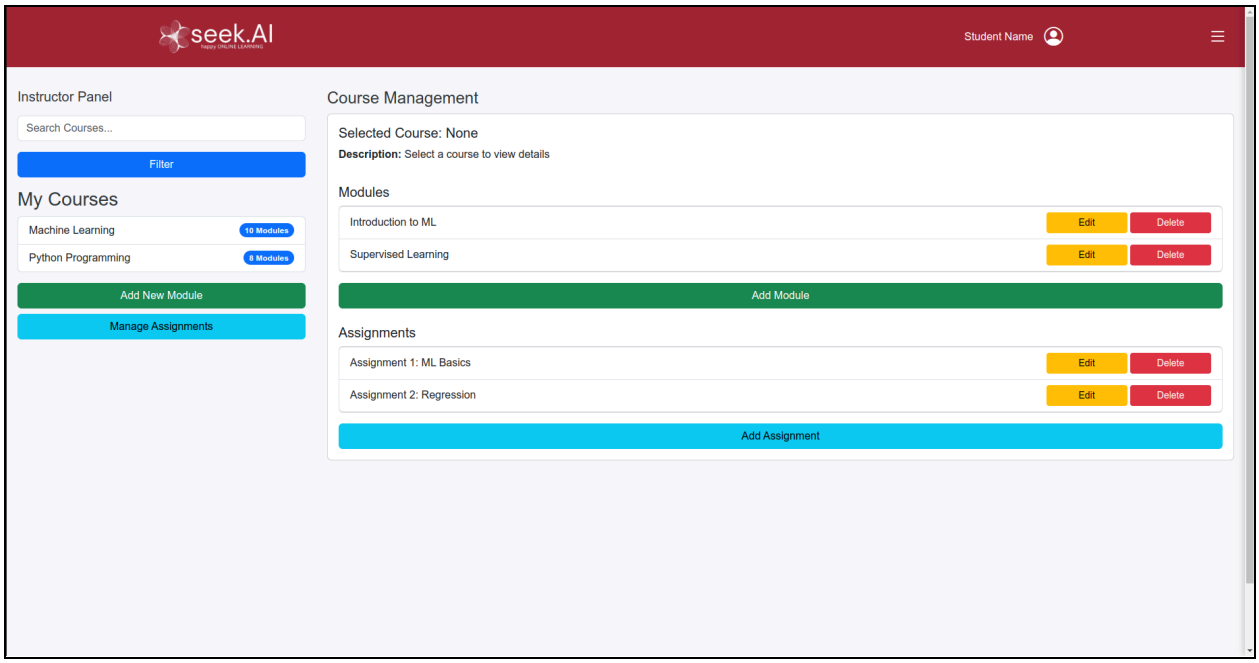


Figure 6.10 Instructor Dashboard

Milestone 4

```
openapi: 3.0.0

info:

 title: Seek AI Platform API

 version: 1.0.0

   description: Comprehensive API for managing courses, students,
instructors, assignments, questions, submissions and AI-powered features



servers:

 - url: /api

   description: Base API path
```

```yaml
components:
 securitySchemes:
   BearerAuth:
     type: http
     scheme: bearer


 schemas:
   Student:
     type: object
     properties:
       student_id:
         type: integer
       level:
         type: string
       enrolled_courses:
         type: object
       total_marks:
         type: number
       weaknesses:
         type: array
         items:
           type: string
       username:
```

```
        type: string
      email:
        type: string

  Instructor:
    type: object
    properties:
      instructor_id:
        type: integer
      specialization:
        type: string
      experience_years:
        type: integer
      qualifications:
        type: array
        items:
          type: string
      username:
        type: string
      email:
        type: string

  Course:
```

```yaml
    type: object
    properties:
      course_id:
        type: integer
      instructor_id:
        type: integer
      title:
        type: string
      name:
        type: string
      created_at:
        type: string
        format: date-time


  Video:
    type: object
    properties:
      video_id:
        type: integer
      week_id:
        type: integer
      video_no:
        type: integer
```

```
      title:
        type: string
      video_url:
        type: string


  Assignment:
    type: object
    properties:
      assignment_id:
        type: integer
      week_id:
        type: integer
      due_date:
        type: string
        format: date-time


  Question:
    type: object
    properties:
      question_id:
        type: integer
      type:
        type: string
```

```
      correct_answer:
        type: string
      assignment_id:
        type: integer

  Submission:
    type: object
    properties:
      submission_id:
        type: integer
      assignment_id:
        type: integer
      user_id:
        type: integer
      submitted_at:
        type: string
        format: date-time
      marks_obtained:
        type: number

  AutoGradeRequest:
    type: object
    properties:
```

```yaml
      submission_id:
        type: integer
      answers:
        type: object
        additionalProperties:
          type: string

  AutoGradeResponse:
    type: object
    properties:
      grade:
        type: number
        description: Percentage score obtained

  AnalysisResponse:
    type: object
    properties:
      analysis:
        type: string
        description: AI-generated analysis or recommendation
      timestamp:
        type: string
        format: date-time
```

```yaml
paths:
 # Student Routes
 /students/course/{course_id}:
   get:
     summary: Get students by course
     security:
       - BearerAuth: []
     parameters:
       - name: course_id
         in: path
         required: true
         schema:
           type: integer
     responses:
       '200':
         description: List of students in course
         content:
           application/json:
             schema:
               type: array
               items:
                 $ref: '#/components/schemas/Student'
```

```yaml
/students/{student_id}:
  get:
    summary: Get student details
    security:
      - BearerAuth: []
    parameters:
      - name: student_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Student details
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Student'


# Instructor Routes
/instructors:
  get:
```

```yaml
    summary: Get all instructors
    security:
      - BearerAuth: []
    responses:
      '200':
        description: List of instructors
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Instructor'


/instructors/{instructor_id}:
  get:
    summary: Get instructor details
    security:
      - BearerAuth: []
    parameters:
      - name: instructor_id
        in: path
        required: true
        schema:
```

```yaml
          type: integer
    responses:
      '200':
        description: Instructor details
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Instructor'


# Course Routes
/courses:
  get:
    summary: Get all courses
    security:
      - BearerAuth: []
    responses:
      '200':
        description: List of courses
        content:
          application/json:
            schema:
              type: array
              items:
```

```
              $ref: '#/components/schemas/Course'


/courses/student/{student_id}:
  get:
    summary: Get courses by student
    security:
      - BearerAuth: []
    parameters:
      - name: student_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of courses for student
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Course'
```

```yaml
/courses/instructor/{instructor_id}:
  get:
    summary: Get courses by instructor
    security:
      - BearerAuth: []
    parameters:
      - name: instructor_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of courses for instructor
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Course'


/courses/{course_id}:
  put:
```

```yaml
    summary: Create or Update course
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              instructor_id:
                type: integer
              title:
                type: string
              name:
                type: string
    responses:
```

```
      '200':
        description: Course updated successfully


  delete:
    summary: Delete course
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Course deleted successfully


# Video Routes
/videos/course/{course_id}:
  get:
    summary: Get videos by course
    security:
      - BearerAuth: []
```

```yaml
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of videos in course
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Video'


/videos/course/{course_id}/week/{week_number}:
  get:
    summary: Get videos by course week
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
```

```
        in: path
        required: true
        schema:
          type: integer
      - name: week_number
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of videos in course week
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Video'


# Assignment Routes
/assignments/course/{course_id}:
  get:
    summary: Get assignments by course
```

```yaml
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of assignments in course
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Assignment'


# Additional Assignment Routes
/assignments/course/{course_id}/week/{week_number}:
  get:
    summary: Get assignments by course week
    security:
```

```
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
      - name: week_number
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of assignments in course week
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Assignment'


/assignments/{assignment_id}:
```

```yaml
  put:
    summary: Create or update assignment
    security:
      - BearerAuth: []
    parameters:
      - name: assignment_id
        in: path
        required: true
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              week_id:
                type: integer
              due_date:
                type: string
                format: date-time
    responses:
```

```
      '200':
        description: Assignment created/updated successfully


  delete:
    summary: Delete assignment
    security:
      - BearerAuth: []
    parameters:
      - name: assignment_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Assignment deleted successfully


# Question Routes
/questions/assignment/{assignment_id}:
  get:
    summary: Get questions by assignment
    security:
      - BearerAuth: []
```

```yaml
    parameters:
      - name: assignment_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of questions in assignment
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Question'


/questions/{question_id}:
  put:
    summary: Create or update question
    security:
      - BearerAuth: []
    parameters:
      - name: question_id
```

```
        in: path
        required: true
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              assignment_id:
                type: integer
              type:
                type: string
              correct_answer:
                type: string
    responses:
      '200':
        description: Question created/updated successfully


  delete:
    summary: Delete question
```

```
    security:
      - BearerAuth: []
    parameters:
      - name: question_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Question deleted successfully


# Submission Routes
/submissions/course/{course_id}/week/{week_number}:
  get:
    summary: Get submissions by course week
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
```

```
          type: integer
      - name: week_number
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of submissions for course week
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Submission'


/submissions/course/{course_id}:
  get:
    summary: Get submissions by course
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
```

```yaml
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of submissions for course
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Submission'


/submissions/{assignment_id}:
  put:
    summary: Create or update submission
    security:
      - BearerAuth: []
    parameters:
      - name: assignment_id
        in: path
        required: true
```

```
        schema:
          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              marks_obtained:
                type: integer
    responses:
      '200':
        description: Submission created/updated successfully
# Gen AI Routes
/genai/chat:
  post:
    summary: Chat with AI
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
```

```
        application/json:
          schema:
            type: object
            properties:
              prompt:
                type: string
    responses:
      '200':
        description: AI response
        content:
          application/json:
            schema:
              type: string

/videos/{video_id}/summary:
  get:
    summary: Get AI-generated video summary
    security:
      - BearerAuth: []
    parameters:
      - name: video_id
        in: path
        required: true
```

```yaml
        schema:
          type: integer
    responses:
      '200':
        description: Video summary
        content:
          application/json:
            schema:
              type: string

/videos/{video_id}/transcript:
  get:
    summary: Get video transcript
    security:
      - BearerAuth: []
    parameters:
      - name: video_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
```

```
        description: Video transcript
        content:
          application/json:
            schema:
              type: array
              items:
                type: object


/questions/{question_id}/hint:
  get:
    summary: Get AI-generated hint for question
    security:
      - BearerAuth: []
    parameters:
      - name: question_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Question hint
        content:
```

```yaml
          application/json:
            schema:
              type: string


/assignments/{assignment_id}/weak-students:
  get:
    summary: Get weak students for assignment
    security:
      - BearerAuth: []
    parameters:
      - name: assignment_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of weak students
        content:
          application/json:
            schema:
              type: array
              items:
```

```
                type: object
                properties:
                  student_id:
                    type: integer
                  username:
                    type: string


/courses/{course_id}/weak-students:
  get:
    summary: Get weak students for course
    security:
      - BearerAuth: []
    parameters:
      - name: course_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: List of weak students
        content:
          application/json:
```

```yaml
            schema:
              type: array
              items:
                type: object
                properties:
                  student_id:
                    type: integer


/videos/{video_id}/resources:
  get:
    summary: Get AI-generated additional resources
    security:
      - BearerAuth: []
    parameters:
      - name: video_id
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Additional resources
        content:
```

```yaml
          application/json:
            schema:
              type: string
/auto-grade:
  post:
    summary: Auto-grade a student's submission
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              submission_id:
                type: integer
              answers:
                type: object
                additionalProperties:
                  type: string
                description: Key-value pairs of question_id and answers
          example:
```

```
            submission_id: 123
            answer_1: "option a"
            answer_2: "true"
    responses:
      '200':
        description: Grading result
        content:
          application/json:
            schema:
              type: object
              properties:
                grade:
                  type: number
                  description: Percentage score
              example:
                grade: 85.5
      '404':
        description: Submission not found


# System-wide performance report
/reports/system:
  get:
    summary: Generate system-wide performance report
```

```
    security:
      - BearerAuth: []
     description: Generates a comprehensive report including submissions
and video watch statistics
    responses:
      '200':
        description: Generated report
        content:
          application/json:
            schema:
              type: string


# Assignment feedback
/assignments/{assignment_id}/feedback:
  get:
    summary: Get AI-generated assignment feedback
    security:
      - BearerAuth: []
    parameters:
      - name: assignment_id
        in: path
        required: true
        schema:
          type: integer
```

```
    responses:
      '200':
        description: AI-generated feedback
        content:
          application/json:
            schema:
              type: string
      '404':
        description: Assignment not found


# Student progress analysis
/students/{student_id}/progress:
  get:
    summary: Get detailed student progress analysis
    security:
      - BearerAuth: []
    parameters:
      - name: student_id
        in: path
        required: true
        schema:
          type: integer
      description: Analyzes student progress across all enrolled courses
using AI
```

```
    responses:
      '200':
        description: Progress analysis
        content:
          application/json:
            schema:
              type: string


 # Course recommendations
 /students/{student_id}/recommended-courses:
   get:
     summary: Get course recommendations for student using AI
     security:
       - BearerAuth: []
     parameters:
       - name: student_id
         in: path
         required: true
         schema:
           type: integer
     description: Course recommendation based on student's performance and
interests
     responses:
       '200':
```

```
        description: Course recommendations
        content:
          application/json:
            schema:
              type: string


 # Personalized learning plan
 /students/{student_id}/learning-plan:
   get:
     summary: Generate personalized learning plan using AI
     security:
       - BearerAuth: []
     parameters:
       - name: student_id
         in: path
         required: true
         schema:
           type: integer
      description: Creates a customized learning plan based on student's
performance
     responses:
       '200':
         description: Personalized learning plan
         content:
```

```
          application/json:
            schema:
              type: string


 # Knowledge gap analysis
 /students/{student_id}/knowledge-gaps:
   get:
     summary: Analyze student's knowledge gaps using AI
     security:
       - BearerAuth: []
     parameters:
       - name: student_id
         in: path
         required: true
         schema:
           type: integer
       description: Identifies strong and weak areas across all enrolled
courses
     responses:
       '200':
         description: Knowledge gap analysis
         content:
           application/json:
             schema:
```

```
                type: string
```

Milestone 5

1. API Testing and Test Suite

Objective: The focus of this milestone is to design and implement a **comprehensive test suite** for the project, ensuring the **accuracy, reliability, and performance** of all API endpoints and functionalities. This includes: Defining **extensive test cases,** Establishing **peer evaluation criteria,** Implementing **basic unit tests** using pytest

Test Cases for API Endpoints

2.1 User Management APIs

Description: These APIs have functionality related to features such as registration, login, and retrieval of user data. They are for normal and other functionalities.

(a) Register user

Endpoint: URL: http://127.0.0.1:5000/register ; **Method:** POST

1. def test_signup_empty_inputs() Tests whether the application correctly rejects invalid empty inputs during user registration

- **Passed Inputs:** `{}`
- **Expected Output:** `HTTP-Status Code: 400`
- **Actual Output:** `HTTP-Status Code: 400`
- **Result:** `Passed`

```
def test_signup_empty_inputs():

    response = requests.post(f"{BASE_URL}/register", json={})
```

```
    assert response.status_code == 400
```

2. def test_register_instructor(client) Tests whether the application correctly rejects invalid empty inputs during user registration

- Passed Inputs: `data_instructor = {"email": "instructor8@example.com","password": "instructor123","role": "instructor","name": "test4"}`

- Expected Output: `HTTP-Status Code: 201`

- Actual Output: `HTTP-Status Code: 201`

- Result: `Passed`

```
def test_register_instructor(client):
    #data = {"email": "instructor3@example.com", "password": "instructor123", "role": "instructor"}
    response = client.post("/register", json=data_instructor)
    assert response.status_code == 201
```

(b) Authenticate User

Endpoint: URL: http: //127.0.0.1:5000/login ; **Method:** POST

1. `def test_login_successful():` Tests whether the app lets the user log in if he gives correct credentials

- Passed Inputs: `{"email": "instructor10@example.com", "password": "instructor123","role": "instructor" }`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_login_success():

    data = {"email": "instructor10@example.com", "password": "instructor123","role": "instructor"}
```

```
    response = requests.post(f"{BASE_URL}/login", json=data)

    assert response.status_code == 200

    assert "token" in response.json()
```

(c) Verify User Email

Endpoint: URL: http://127.0.0.1:5000/register ; **Method:** POST

1. `def test_signup_duplicate_mail(client)` Tests whether the application correctly rejects invalid empty inputs during user registration

- **Passed Inputs:** `data_instructor = {"email": "instructor@example.com","password": "instructor123","role": "instructor","name": "test4"}`

- **Expected Output:** `HTTP-Status Code: 400`

- **Actual Output:** `HTTP-Status Code: 400`

- **Result:** `Passed`

```
def test_signup_duplicate_email():

    data = {"email": "instructor@example.com", "password":
"instructor123", "role": "instructor"}

    requests.post(f"{BASE_URL}/register", json=data)  # First request to
create the user

    response = requests.post(f"{BASE_URL}/register", json=data)  # Second
request (duplicate)

    assert response.status_code == 400
```

(d) Delete User

Endpoint: URL: http://127.0.0.1:5000/delete_user ; **Method:** POST

1. `def test_delete_user(client)` Tests whether the application correctly deleting the user from database

- **Passed Inputs:** `data_instructor = {"email": "instructor8@example.com"}`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_delete_user(client):

    response = client.delete(f"{BASE_URL}/delete_user", json={"email":
"instructor8@example.com"})

    assert response.status_code == 200

    assert response.json["message"] == "Instructor deleted successfully"
```

(e) Update Password

Endpoint: URL: http://127.0.0.1:5000/update_password ; **Method:** POST

1. `def test_update_password(client)` Tests whether the application correctly deleting the user from database

- **Passed Inputs:** `data_instructor = {"email": "instructor@example.com", "old_password": "instructor123", "new_password": "instructor456"}`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_update_password(client):
```

```
      response  =client.put(f"{BASE_URL}/update_password",  json={"email":
"instructor@example.com", "old_password": "instructor123", "new_password":
"instructor456"})
    assert response.status_code == 200
```

2.2 Courses APIs

(a) List Courses:

Endpoint: URL: http://127.0.0.1:5000/api/courses ; **Method:** GET

1. `def test_get_courselist(client)` Tests whether the application correctly fetches the course details.

- Passed Inputs: `course`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
#Course test
def test_get_courselist(client):
    response = client.get(f"{BASE_URL}/api/courses", headers=headers)
    assert response.status_code == 200
    assert len(response.json) > 0Code description
```

(b) Courses by Course ID:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id> ; **Method:** GET

1. `def test_get_course_not_found()` Tests whether the application correctly fetches the course details

- Passed Inputs: `course/123`

- Expected Output: `HTTP-Status Code: 404`

- Actual Output: `HTTP-Status Code: 404`

- Result: `Passed`

```
def test_get_course_not_found():
```

```
    response = requests.get(f"{BASE_URL}/api/course/123")
    assert response.status_code == 404
```

2. `def test_get_course_success()` Tests whether the application correctly fetches the course details by course id.

- Passed Inputs: `course/5`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_get_course_success(client):

    response = requests.get(f"{BASE_URL}/api/course/5")

    assert response.status_code == 200

    data=response.json()

    assert data["course_id"] == 5

    assert data["name"] == "Machine Learning "

    assert data["desc"] == "Introduction to ML"

    assert data["term_name"] == "Spring 2025"

    assert data["course_code"] == "ML0001"
```

(c) Creation / Addition of Courses:

Endpoint: URL: http://127.0.0.1:5000/api/course ; **Method:** POST

1. `def test_create_course(clie`nt) Tests whether the application correctly adds the course or not.

- Passed Inputs: data = {

"name": "Advanced Natural Language Processing","desc": "Text processing, sentiment analysis, and transformers","term_name": "Spring 2025","course_code": "NLP304"

}

- Expected Output: HTTP-Status Code: 200

- Actual Output: HTTP-Status Code: 200

- Result: Passed

```
def test_create_course(client):

def test_create_course(client):

    headers = {

        "Content-Type": "application/json",

        "Authentication-Token":
".eJwNysEKgCAMANB_2VlCTbflz4SbC7pJUQTRv-ftHd4Ltx1QIIODa29DzMTG5FkUk-QoC87N
hBNWUdU04nauvVYogVIMjEw0YRik7MCeDsV_Px3DF1c.Z9epfQ.XGePg0_8PPnWHc_MmgmUKDC
BU4g"

    }

    data = {

        "name": "Advanced Natural Language Processing",

        "desc": "Text processing, sentiment analysis, and transformers",

        "term_name": "Spring 2025",

        "course_code": "NLP304"

    }

    response = client.post("/api/course", json=data, headers=headers)

    assert response.status_code == 201
```

(d) Updation of Courses:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id> ; **Method:** PUT

1. `def test_update_course(client)` Tests whether the application correctly updates the course or not.

- Passed Inputs: data = {

```
"name": "Advanced NLP","desc": "Perfect Guide for Text processing, sentiment analysis, and transformers","term_name": "Spring 2025","course_code": "NLP304"

}
```

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_update_course(client):
    headers = {
        "Content-Type": "application/json",
                                                "Authentication-Token":
".eJwNysEKgCAMANB_2VlCTbflz4SbC7pJUQTRv-ftHd4Ltx1QIIODa29DzMTG5FkUk-QoC87N
hBNWUdU04nauvVYogVIMjEw0YRik7MCeDsV_Px3DF1c.Z9epfQ.XGePg0_8PPnWHc_MmgmUKDC
BU4g"
    }
    data = {
        "name": "Advanced NLP",
          "desc": "Perfect Guide for Text processing, sentiment analysis,
and transformers",
        "term_name": "Spring 2025",
        "course_code": "NLP304"
    }
    response = client.put(f"/api/course/8", json=data, headers=headers)
    assert response.status_code==200
    updated_course= db.session.get(Course,8)
    assert updated_course.name == "Advanced NLP"
```

(e) Deletion of Courses:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id> ; **Method:** DELETE

`1. def test_delete_course(clie`nt) Tests whether the application correctly deletes the course or not.

- Passed Inputs: data = `{11}`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_delete_course(client):

    response = client.delete(f"/api/course/11", headers=headers)
    assert response.status_code ==200
    assert response.json == {"message": "Course deleted."}
```

2.3 Assignment APIs

(a) List Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignments ; **Method:** GET

1. `def test_get_assignmentlist()` Tests whether the application correctly fetching the assignment list

- Passed Inputs: `assignments`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_get_assignmentslist(client):
    response = client.get('/api/assignments')
    assert response.status_code == 200
    assert len(response.json) > 0
```

(b) Create Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignments ; **Method:** POST

`1. def test_create_assignment(client)` Tests whether the application correctly creates the assignment.

- Passed Inputs: `json={`

`"instructor_id": 6,due_date": datetime.fromisoformat('2025-03-30T23:59:59'), #datetime('2025-03-30T23:59:59').isoformat(),"week_no": 4,"course_id": 101}`

- Expected Output: `HTTP-Status Code: 201`

- Actual Output: `HTTP-Status Code: 201`

- Result: `Passed`

```
def test_create_assignment(client):

    response = client.post('/api/assignments', json={

        "instructor_id": 6,

        "due_date": datetime.fromisoformat('2025-03-30T23:59:59'),

        "week_no": 4,

        "course_id": 101

    }, headers=headers)

    assert response.status_code == 201

    assert response.json["message"] == "Assignment created"
```

(c) List Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignments/<assignment_id> ; **Method:** GET

1. `def test_get_assignment_by_ID()` Tests whether the application correctly fetching the assignment by its assignment ID

- **Passed Inputs:** 1

- **Expected Output:** HTTP-Status Code: 200

- **Actual Output:** HTTP-Status Code: 200

- **Result:** Passed

```
def test_get_assignment_by_ID():
    response = requests.get(f"{BASE_URL}/api/assignments/1")
    assert response.status_code == 200
```

2. def test_get_assignment_not_found() Tests whether the application correctly returning the response if there is no assignment

- **Passed Inputs:** 123

- **Expected Output:** HTTP-Status Code: 404

- **Actual Output:** HTTP-Status Code: 404

- **Result:** Passed

```
def test_get_assignment_not_found():
    response = requests.get(f"{BASE_URL}/api/assignments/123")
    assert response.status_code == 404
```

(d) Update Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignments/<assignment_id> ; **Method:** PUT

1. def test_update_assignment(CLIENT) Tests whether the application correctly updating the assignment

- **Passed Inputs:** json={

"instructor_id": 6,"week_no": 5,"course_id": 104}

- **Expected Output:** HTTP-Status Code: 200

- **Actual Output:** HTTP-Status Code: 200

- **Result:** Passed

```
def test_update_assignment(client):
    response = client.put(f'/api/assignments/6', json={
        "instructor_id": 6,
        "week_no": 5,
        "course_id": 104
    }, headers=headers)
    assert response.status_code == 200
    assert response.json["message"] == "Assignment updated"
```

(e) Delete Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignments/<assignment_id> ; **Method:** DELETE

1. `def test_delete_assignment(client)` Tests whether the application correctly deleting the assignment

- **Passed Inputs:** `23`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_delete_assignment(client):
    response = client.delete(f'/api/assignments/23', headers=headers)
    assert response.status_code == 200
    assert response.json["message"] == "Assignment deleted"
```

(f) Assignment by Course:

Endpoint: URL: http://127.0.0.1:5000/api/courses/<coursses_id>/assignments ; **Method:** GET

1. `def test_assignment_by_course(client)` Tests whether the application correctly fetches the assignment by course

- **Passed Inputs:** course_id = `5`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#assignment by course

def test_get_assignment_by_course(client):

        response  =  client.get(f"{BASE_URL}/api/courses/5/assignments",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

(g) Assignment by Course Week:

Endpoint: URL:
http://127.0.0.1:5000/api/courses/<coursses_id>/weeks/<week_id>/assignments ; **Method:** GET

1. `def test_assignment_by_courseweek(client)` Tests whether the application correctly fetches the assignment by course week

- **Passed Inputs:** `course_id =5 & week_id =2`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#assignment by course week

def test_get_assignment_by_courseweek(client):

    response = client.get(f"{BASE_URL}/api/courses/5/weeks/2/assignments",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

2.4 Student APIs

(a) Details of Students by id:

Endpoint: URL: http://127.0.0.1:5000/api/students<student_id> ; **Method:** GET

1. `def test_get_student_details(client)` Tests whether the application correctly fetching the Student details by Student Id

- **Passed Inputs:** `5`

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_get_student_details(client):
    response = client.get(f"{BASE_URL}/api/students/5", headers=headers)
    assert response.status_code == 200
    print(response.status_code, response.json)
```

2. `def test_get_student_not_found(client)` Tests whether the application correctly gives response incase of student not found.

- **Passed Inputs:** `1`

- **Expected Output:** `HTTP-Status Code: 404`

- **Actual Output:** `HTTP-Status Code: 404`

- **Result:** `Passed`

```
def test_get_student_not_found(client):
    response = client.get(f"{BASE_URL}/api/students/1", headers=headers)
    assert response.status_code == 404
```

(b) Details of Students by Course:

Endpoint: URL: http://127.0.0.1:5000/api/courses<course_id> ; **Method:** GET

1. `def test_get_student_course(client)` Tests whether the application correctly fetching the Student details by Student Id

- Passed Inputs: `course_id = 5`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_get_student_by_course(client):
      response = client.get(f"{BASE_URL}/api/courses/5", headers=headers)
#api/courses/<course_id>/students
    assert response.status_code == 200
    print(response.status_code, response.json)
```

(c) Details of Course by Students:

Endpoint: URL: http://127.0.0.1:5000/api/student/<student_id>/courses ; **Method:** GET

1. `def test_get_course_by_student(client)` Tests whether the application correctly fetching the courses by Student

- Passed Inputs: `student_id = 5`

- Expected Output: `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
#course by student

def test_get_course_by_student(client):

   response = client.get(f"{BASE_URL}/api/students/5/courses",

       headers=headers)

   assert response.status_code == 200

   print(response.status_code, response.json)
```

2.5 Instructor APIs

(a) Details of Instructors:

Endpoint: URL: http://127.0.0.1:5000/api/instructors ; **Method:** GET

1. `def test_get_instructor_list` Tests whether the application correctly fetching the all the instructors details

- **Passed Inputs:**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#instructor details
def test_get_instructor_list(client):
    response = client.get('/api/instructors', headers=headers)
    assert response.status_code == 200
    assert len(response.json) > 0
```

(b) Details of Instructors by Instructor ID:

Endpoint: URL: http://127.0.0.1:5000/api/instructors/<instructor_id> ; **Method:** GET

1. `def test_get_instructor_by_ID(client)` Tests whether the application correctly fetching the the instructors details by ID

- **Passed Inputs: instructor_id = 6**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#instructor details by id
def test_get_instructor_details_by_ID(client):
   response = client.get(f"{BASE_URL}/api/instructors/6", headers=headers)
   assert response.status_code == 200
```

```
    print(response.status_code, response.json)
```

2. `def test_get_instructor_not_found(client)` Tests whether the application correctly giving response incase of instructors not found

- **Passed Inputs:** 1

- **Expected Output:** `HTTP-Status Code: 404`

- **Actual Output:** `HTTP-Status Code: 404`

- **Result:** `Passed`

```
#instructor not found
def test_get_instructor_not_found(client):
    response = client.get(f"{BASE_URL}/api/instructors/1", headers=headers)
    assert response.status_code == 404
```

(c) Details of Courses by Instructor:

Endpoint: URL: http://127.0.0.1:5000/api/instructors/<instructor_id>/courses ; **Method:** GET

1. `def test_get_course_by_instructor(client)` Tests whether the application correctly fetches the courses by instructors.

- **Passed Inputs: instructor_id = 6**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#course by instructor

def test_get_course_by_instructor(client):

          response   =   client.get(f"{BASE_URL}/api/instructors/6/courses",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

2.6 Questions APIs

(a) Questions by Assignment:

Endpoint: URL: http://127.0.0.1:5000/api/assignment/<assignment_id>/questions ; **Method:** GET

1. `def test_get_questions_by_assignments(client)` Tests whether the application correctly fetches the questions by assignment.

- **Passed Inputs: assignment_id = 1**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#questions by assignment

def test_get_questions_by_assignment(client):

        response  =  client.get(f"{BASE_URL}/api/assignments/1/questions",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

(b) Creation of Questions:

Endpoint: URL: http://127.0.0.1:5000/api//questions ; **Method:** POST

1. `def test_create_questions(client)` Tests whether the application correctly creates the new questions.

- **Passed Inputs: {"assignment_id": 5, "stmt": "This is the first question to be tested", "tags": "False"}**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- Result: Passed

```
def test_create_questions(client):

    response = client.put(f"{BASE_URL}/api/questions", json={

        "assignment_id": 5,

        "stmt": "This is the first question to be tested",

        "tags": "False"

    }, headers=headers)

    assert response.status_code == 200

    assert response.json["message"] == "Question created"
```

(c) Updation of Questions:

Endpoint: URL: http://127.0.0.1:5000/api//questions ; **Method:** PUT

1. `def test_update_questions(client)` Tests whether the application correctly updates the questions.

- Passed Inputs: {"assignment_id": 5, "stmt": "This is the first question to be tested", "tags": "True"}

- Expected Output: HTTP-Status Code: 200

- Actual Output: HTTP-Status Code: 200

- Result: Passed

```
#updation of questions

def test_update_questions(client):

    response = client.put(f"{BASE_URL}/api/questions/1", json={

        "question_id": 1,

        "assignment_id": 5,

        "stmt": "This is the first question to be tested",
```

```
        "tags": "True"

    }, headers=headers)

    assert response.status_code == 200

    assert response.json["message"] == "Question updated"
```

2.7 Submission APIs

(a) Submission by Course:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id>/submissions ; **Method:** GET

1. `def test_get_submission_by_course(client)` Tests whether the application correctly fetches the submission by course.

- **Passed Inputs: course_id =5**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#submission by course

def test_get_submission_by_course(client):

         response  =  client.get(f"{BASE_URL}/api/courses/5/submissions",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

(b) Submission by Course Week:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id>/week/<week_id>/submissions

Method: GET

1. `def test_get_submission_by_courseweek(client)` Tests whether the application correctly fetches the submission by course week.

- **Passed Inputs:** **course_id = 5 & week_id = 2**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
#submission by course week

def test_get_submission_by_courseweek(client):

    response = client.get(f"{BASE_URL}/api/courses/5/weeks/2/submissions",
headers=headers)

    assert response.status_code == 200

    print(response.status_code, response.json)
```

2.8 Videos APIs

(a) Fetching Video by ID (Video_id):

Endpoint: **URL:** http://127.0.0.1:5000/api/videos/<video_id> ; **Method:** GET

1. `def test_get_videos_by_ID(client)` Tests whether the application correctly fetches the video by ID.

- **Passed Inputs:** **video_id = 1**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_get_videos_by_ID(client):
```

```
    response = client.get('/api/videos/1', headers=headers)

    assert response.status_code == 200

    assert len(response.json) > 0
```

(b) Videos by Course:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id>/videos ; **Method:** GET

1. `def test_get_video_by_course(client)` Tests whether the application correctly fetches the videos by course.

- **Passed Inputs: course_id =5**

- **Expected Output:** `HTTP-Status Code: 200`

- **Actual Output:** `HTTP-Status Code: 200`

- **Result:** `Passed`

```
def test_get_video_by_course(client):

      response      =      client.get(f"{BASE_URL}/api/courses/5/videos",
headers=headers)

      assert response.status_code == 200

      print(response.status_code, response.json)
```

(c) Videos by Course Week:

Endpoint: URL: http://127.0.0.1:5000/api/course/<course_id>/week/<week_id>/videos ; **Method:** GET

1. `def test_get_videos_by_courseweek(client)` Tests whether the application correctly fetches the submission by course week.

- **Passed Inputs: course_id = 5 & week_id = 2**

- **Expected Output:** `HTTP-Status Code: 200`

- Actual Output: `HTTP-Status Code: 200`

- Result: `Passed`

```
def test_get_videos_by_courseweek(client):

        response  =  client.get(f"{BASE_URL}/api/courses/5/weeks/2/videos",
headers=headers)

        assert response.status_code == 200

        print(response.status_code, response.json)

        assert len(response.json) > 0
```

2.9 Gen AI Testing

(a) Loading of PDF:

1. `def test_load_pdf_valid()` Tests whether the application is correctly loading the pdf.

- **Passed Inputs:** `TEST_PDF_PATH = "handbook.pdf"`

- **Expected Output:** `Passed`

- **Actual Output:** `Passed`

- **Result:** `Passed`

```
def test_load_pdf_valid():

    pdf_text = load_pdf(TEST_PDF_PATH)

    assert pdf_text is not None

    assert len(pdf_text) > 0
```

(b) Loading of PDF with invalid path:

1. `def test_load_pdf_invalid_path()` Tests whether the application is correctly giving the response while loading the pdf with invalid path.

- **Passed Inputs:** `INVALID_PDF_PATH = "test_files/invalid.pdf"`

- **Expected Output:** `FileNotFoundError`

- **Actual Output:** `FileNotFoundError`

- **Result:** `Passed`

```
def test_load_pdf_invalid_path():

    with pytest.raises(FileNotFoundError):

        load_pdf(INVALID_PDF_PATH)
```

(c) Creation of Faiss Index when no pickle exist:

1. `def test_create_faiss_index_create_new()` Tests whether the application is correctly creating faiss index.

- **Passed Inputs:** **`"This is a sample text for FAISS."`**

- **Expected Output:** `Passed`

- **Actual Output:** `Passed`

- **Result:** `Passed`

```
Create FAISS index when no pickle exists

def test_create_faiss_index_create_new():

    pdf_text = "This is a sample text for FAISS."

    faiss_index = create_faiss_index(pdf_text)

    assert faiss_index is not None

    assert os.path.exists(FAISS_INDEX_FILE)

    # Load the stored index to verify

    with open(FAISS_INDEX_FILE, "rb") as f:

        loaded_index = pickle.load(f)

    assert loaded_index is not None
```

(d) Creation of Faiss Index when no pickle exist:

1. `def test_create_faiss_index_load_existing()` Tests whether the application is correctly loading the FAISS index from pickle.

- **Passed Inputs:** `"This is a sample text for FAISS."`

- **Expected Output:** `Passed`

- **Actual Output:** `Passed`

- **Result:** `Passed`

```
 Load existing FAISS index from pickle

def test_create_faiss_index_load_existing():

    pdf_text = "This is a sample text for FAISS."

    # Create and save FAISS index initially

    create_faiss_index(pdf_text)

    assert os.path.exists(FAISS_INDEX_FILE)

    # Load the same index to ensure no re-creation

    faiss_index_loaded = create_faiss_index(pdf_text)

    assert faiss_index_loaded is not None
```

(e) Handling of empty text for FAISS:

1. `def test_create_faiss_index_with_empty_text()` Tests whether the application is correctly handling the empty text for FAISS.

- **Passed Inputs:** `""`

- **Expected Output:** `Passed`

- **Actual Output:** `Passed`

- **Result:** `Passed`

```
def test_create_faiss_index_with_empty_text():

    faiss_index = create_faiss_index("")

    assert faiss_index is not None

    assert os.path.exists(FAISS_INDEX_FILE)
```

(f) Generation of Prompt:

1. `def test_generate_promt_basic()` Tests whether the application correctly generates the prompt.

- Passed Inputs: `query = "What is the course duration?"`

`context = "The course duration is 6 months for online mode and 4 months for offline mode."`

- Expected Output: `Your name is seek.AI, an AI assistant for students of IIT Madras. Your job is to guide students by providing references from the available materials.`

`some restrictions to you`

`just answer only the things asked not more not less(suppose name asked then ans only name if work/purpose asked then only purpose),`

`QUESTION: What is the course duration?`

`CONTEXT: The course duration is 6 months for online mode and 4 months for offline mode.`

`Suggest relevant learning strategies and direct students to reference materials.`

- Actual Output: `Same as Expected Output`

- Result: `Passed`

```
def test_generate_prompt_basic():

    query = "What is the course duration?"
```

```
    context = "The course duration is 6 months for online mode and 4 months for offline mode."

    expected_prompt = """

    Your name is seek.AI, an AI assistant for students of IIT Madras. Your job is to guide students by providing references from the available materials.

    some restrictions to you

    just answer only the things asked not more not less(suppose name asked then ans only name if work/purpose asked then only purpose),

    QUESTION: What is the course duration?

    CONTEXT: The course duration is 6 months for online mode and 4 months for offline mode.

    Suggest relevant learning strategies and direct students to reference materials.

    """

    result = generate_prompt(query, context)

    assert result.strip() == expected_prompt.strip()
```

(g) Handling of empty query:

1. `def test_generate_promt_empty()` Tests whether the application correctly handles the empty input

- **Passed Inputs:** `query = ""`

  `context = ""`

- **Expected Output:** `"""`

```
Your name is seek.AI, an AI assistant for students of IIT Madras. Your job is to guide students by providing references from the available materials.

some restrictions to you
```

just answer only the things asked not more not less(suppose name asked then ans only name if work/purpose asked then only purpose),

QUESTION:

CONTEXT:

Suggest relevant learning strategies and direct students to reference materials.

"""

- Actual Output: Same as Expected Output

- Result: Passed

```
def test_generate_prompt_empty():

    query = ""

    context = ""

    expected_prompt = """

    Your name is seek.AI, an AI assistant for students of IIT Madras. Your job is to guide students by providing references from the available materials.

    some restrictions to you

    just answer only the things asked not more not less(suppose name asked then ans only name if work/purpose asked then only purpose),

    QUESTION:

    CONTEXT:

    Suggest relevant learning strategies and direct students to reference materials.

    """

    result = generate_prompt(query, context)
```

```
    assert result.strip() == expected_prompt.strip()
```

(h) Handling of Special characters:

1. `def test_generate_promt_special_chars()` Tests whether the application correctly handles the special characters.

- **Passed Inputs:** `query = "What is AI? Define @ML & DL!"`

```
context = "AI (Artificial Intelligence) refers to machines simulating
human intelligence. ML (Machine Learning) and DL (Deep Learning) are its
subsets."
```

- **Expected Output:** `"""`

```
Your name is seek.AI, an AI assistant for students of IIT Madras. Your
job is to guide students by providing references from the available
materials.

some restrictions to you

just answer only the things asked not more not less(suppose name asked
then ans only name if work/purpose asked then only purpose),

QUESTION: What is AI? Define @ML & DL!

CONTEXT: AI (Artificial Intelligence) refers to machines simulating
human intelligence. ML (Machine Learning) and DL (Deep Learning) are
its subsets.

Suggest relevant learning strategies and direct students to reference
materials.

"""
```

- **Actual Output:** `Same as Expected Output`

- **Result:** `Passed`

```
def test_generate_prompt_special_chars():

    query = "What is AI? Define @ML & DL!"
```

```
    context = "AI (Artificial Intelligence) refers to machines simulating human intelligence. ML (Machine Learning) and DL (Deep Learning) are its subsets."

    expected_prompt = """

    Your name is seek.AI, an AI assistant for students of IIT Madras. Your job is to guide students by providing references from the available materials.

    some restrictions to you

    just answer only the things asked not more not less(suppose name asked then ans only name if work/purpose asked then only purpose),

    QUESTION: What is AI? Define @ML & DL!

     CONTEXT: AI (Artificial Intelligence) refers to machines simulating human intelligence. ML (Machine Learning) and DL (Deep Learning) are its subsets.

    Suggest relevant learning strategies and direct students to reference materials.

    """

    result = generate_prompt(query, context)

    assert result.strip() == expected_prompt.strip()
```

(i) Chat with AI:

1. `def test_chat_with_ai()` Tests whether the application correctly gives the response.

- **Passed Inputs:** `"What is the main content of the PDF?"`

- **Expected Output:** `Passed`

- **Actual Output:** `Passed`

- **Result:** `Passed`

```python
def real_faiss_index():

    """Generate FAISS index from a real PDF file."""

    pdf_text = load_pdf(TEST_PDF_PATH)

    faiss_index = create_faiss_index(pdf_text)

    return faiss_index

def test_chat_with_ai(real_faiss_index):

    query = "What is the main content of the PDF?"

    response_text = chat_with_ai(query, real_faiss_index)

    print("\nAI Response:", response_text) #this is for AI response veri.

    assert response_text is not None


    assert len(response_text) > 0
```

3. Test Result

Test Result Screenshots:

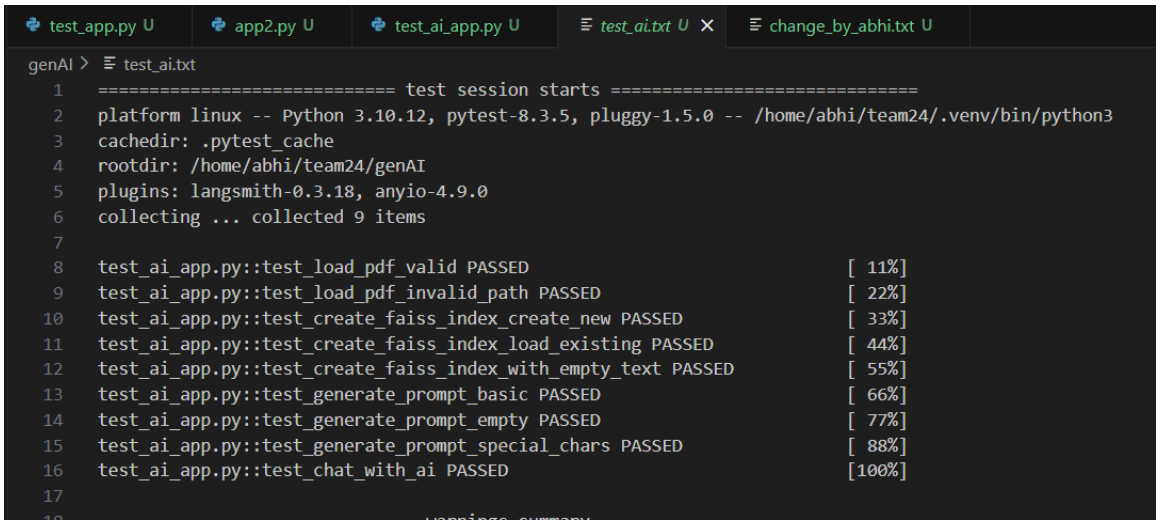


Figure 3.1 AI Test screenshots

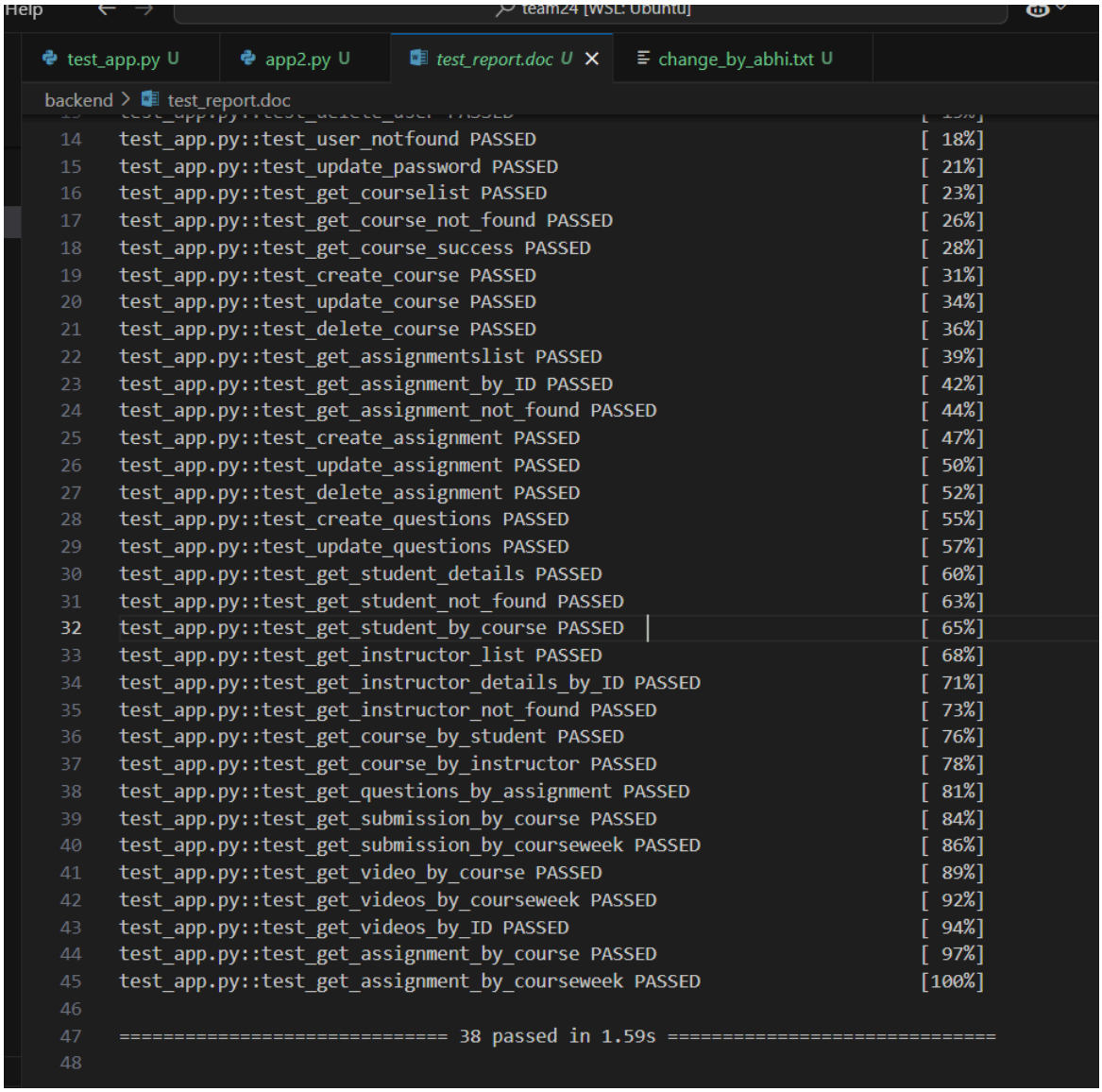


Figure 3.2 General Test Report

Conclusion

All test cases passed with 100% completion, validating API and AI module functionality. User management tests covered signup, login, updates, and error handling. Course, assignment, and question APIs were verified for CRUD operations and retrieval. Student and instructor APIs returned correct details, handling missing records. AI and PDF processing tested FAISS indexing and edge cases. Prompt generation handled various inputs, ensuring relevant AI responses. Video and submission APIs fetched data accurately. Error handling was effectively managed across all endpoints, confirming system robustness and reliability.

Milestone 6

Frontend Development:

- Framework: Vue.js was selected as the primary JavaScript framework for frontend development due to its lightweight nature, reactivity, and component-based architecture. It enables seamless integration with backend APIs while ensuring a dynamic and responsive user interface. The framework's flexibility allowed the team to efficiently build modular components for dashboards, course materials, and AI-driven interactions.

- Languages: JavaScript (ES6+), HTML5, CSS3 Modern JavaScript (ES6+) was used to implement interactive features such as real-time updates, form validations, and API calls. HTML5 and CSS3 ensured semantic structuring and visually appealing designs, with responsive layouts to accommodate different device screens.

- Development Tools: Visual Studio Code (VSCode): The primary Integrated Development Environment (IDE) for writing, debugging, and version-controlling frontend code. Extensions like Vetur and ESLint enhanced productivity.

- Excalidraw: Utilized for creating low-fidelity wireframes during the planning phase, helping visualize UI layouts and user flows. Canva: Employed for designing high-fidelity storyboards to demonstrate user interactions and scenarios.

Backend Development:

- Framework: Flask, a micro web framework in Python, was chosen for its simplicity and scalability in building RESTful APIs. It facilitated rapid development of endpoints for user authentication, course management, and AI integrations while maintaining clean code architecture.

- Database: SQLite served as the lightweight, file-based database during development due to its zero-configuration setup and compatibility with Flask. It efficiently handled data storage for users, courses, assignments, and submissions. For production, considerations include migrating to PostgreSQL for better scalability and concurrent access support.

- API Documentation: Swagger/OpenAPI Swagger was integrated to auto-generate interactive API documentation (OpenAPI 3.0 specification), ensuring clarity in endpoint definitions, request/response schemas, and authentication workflows for developers.

- Testing: Pytest streamlined unit and integration testing for backend logic, covering edge cases in API routes (e.g., user authentication failures, course retrieval). Automated test suites ensured robustness, as evidenced by the extensive test cases listed in the documentation.

General Technologies:

- Core Programming Language: Python's readability and extensive libraries (e.g., Flask, SQLAlchemy) accelerated backend development, while its compatibility with AI/ML frameworks (future-proofing) aligned with the project's GenAI goals.

- Version Control: Git, coupled with platforms like GitHub, enabled collaborative code management, branch-based feature development, and version tracking. Commit histories and pull requests maintained accountability among team members.

- Project Management: Jira's agile boards and sprint planning tools were pivotal in tracking tasks, deadlines, and dependencies. It provided visibility into individual responsibilities (e.g., Rushil/Prakhar/Abhinash on backend, Soumyadeep/Rajarshi/Kanishka/Poorvi on UI) and progress across milestones, as referenced in the project schedule.

Hosting and Deployment:

- Platform (Backend Hosting): Services like Render or Heroku are ideal for deploying Flask APIs due to their support for Python environments, scalability, and seamless CI/CD pipeline integrations.

- Frontend Hosting: Netlify or Vercel offer optimized workflows for Vue.js applications, including automatic deployments from Git repositories and global CDN distribution.

- Database: While SQLite sufficed for development, production environments would leverage PostgreSQL or MySQL for ACID compliance, multi-user support, and enhanced security. Cloud solutions like AWS RDS or Firebase could further streamline database management.

Additional Tools:

- Collaboration: Google Space/Whatsapp for real-time communication during scrums (held biweekly).

- Design: Figma (alternative to Canva) for iterative UI/UX prototyping.

Key Advantages of the Chosen Stack:

- Modularity and Agility: Vue.js and Flask's component-based structures promoted reusability and maintainability. Jira and Git enabled iterative development aligned with sprint goals (e.g., Milestone 3's focus on AI-driven features).

- Scalability: The stack supports future enhancements, such as integrating TensorFlow for advanced AI analytics.

Instructions to Run the Application:

On Ubuntu/Mac OS:

1. Clone the repository
- git clone **https://github.com/21f1006728/seek-AI.git**
- Complete code is present in "main" branch.
2. Change the directory to the "backend" directory inside the directory:
- cd seek-AI
3. Create a Python virtual environment:
- python -m venv env (or python3)
4. Activate the virtual environment:
- source env/bin/activate
5. Install the requirements:
- pip install -r req.txt
6. Run the application inside backend folder:
- python backend/app.py (or python3)
- Open your browser and go to:

http://127.0.0.1:5000/

On Windows:

1. Clone the repository:
- git clone **https://github.com/21f1006728/seek-AI.git**
2. Change the directory to the "backend" directory inside the directory:
- cd seek-AI
3. Create a Python virtual environment:
- python -m venv env
4. Activate the virtual environment:
- env\Scripts\activate
5. Install the requirements:
- pip install -r req.txt
6. Run the application:
- python app.py
- Open your browser and go to:
  http://127.0.0.1:5000/

Issue Reporting and Tracking:

For daily communication, WhatsApp was used including raising issues, obtaining approvals and basic collaboration. GitHub was the repository for the codes and documentation. The code changes were committed and pushed as necessary.

- Daily Communication: A WhatsApp group was created for quick updates, addressing minor issues, and ensuring all team members stayed informed.

- Scheduled Meetings: The Scrum Master organized meetings on scheduled days to track individual progress, identify roadblocks, and facilitate coordination between frontend and backend teams.

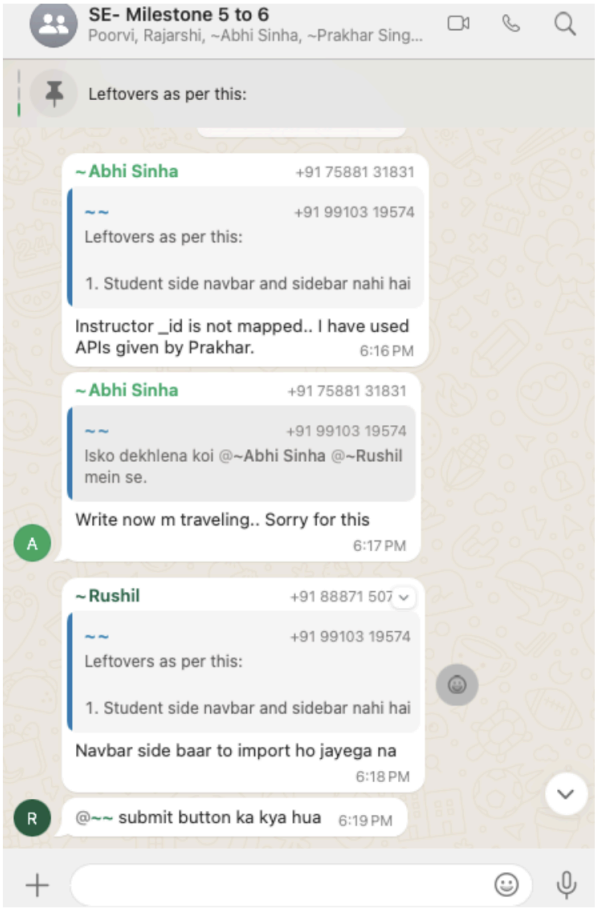
- Development with GitHub: GitHub was utilized as a version control system for collaboratively writing, reviewing, and merging code.

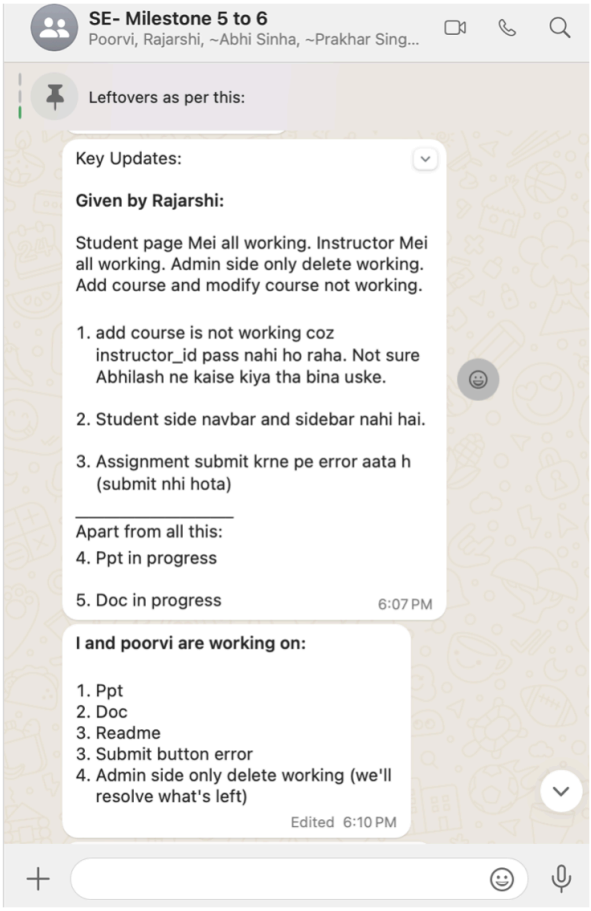
Open 0 **Closed** 20 Author Labels Projects Milestones Assignees Types Newest

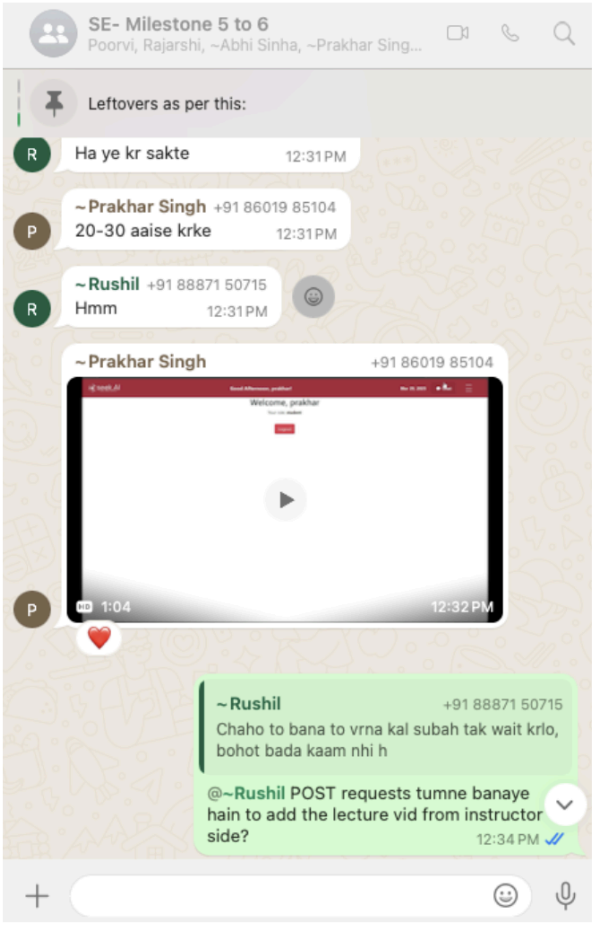
- **Submit assignment button not working**
  #20 · by 21f1006728 was closed 25 minutes ago
- **APIs requirement**
  #19 · by SINHA-IIT was closed 2 days ago — 1
- **CurrentCourse frontend page**
  #18 · by SINHA-IIT was closed 2 days ago
- **Latest router file is not available**
  #17 · by SINHA-IIT was closed 2 days ago
- **editcourse.js file pushed**
  #16 · by SINHA-IIT was closed 2 days ago
- **addcourse.js file pushed to git**
  #15 · by SINHA-IIT was closed 2 days ago
- **Instructor Frontend page**
  #14 · by SINHA-IIT was closed 2 days ago
- **Student frontend page**
  #13 · by SINHA-IIT was closed 2 days ago
- **GenAI Part Have to much restrictions**
  #12 · by prakh-singh was closed 2 days ago — 1
- **some pages lagging with nav bar and side bar component**
  #11 · by prakh-singh was closed 2 days ago — 1
- **Student assignment submission page not created**
  #10 · by prakh-singh was closed 2 days ago — 1
- **APIs**
  #9 · by SINHA-IIT was closed 3 days ago
- **Test cases GenAI**
  #8 · by SINHA-IIT was closed 3 days ago
- **Styling**
  #7 · by SINHA-IIT was closed 3 days ago
- **Frontend pages styling**
  #6 · by SINHA-IIT was closed 3 days ago
- **Frontend pages**
  #5 · by SINHA-IIT was closed 3 days ago
- **Login**
  #4 · by SINHA-IIT was closed 3 days ago
- **Test case**
  #3 · by SINHA-IIT was closed 3 days ago
- **Test cases**
  #2 · by SINHA-IIT was closed 3 days ago
- **Frontend**
  #1 · by SINHA-IIT was closed 3 days ago

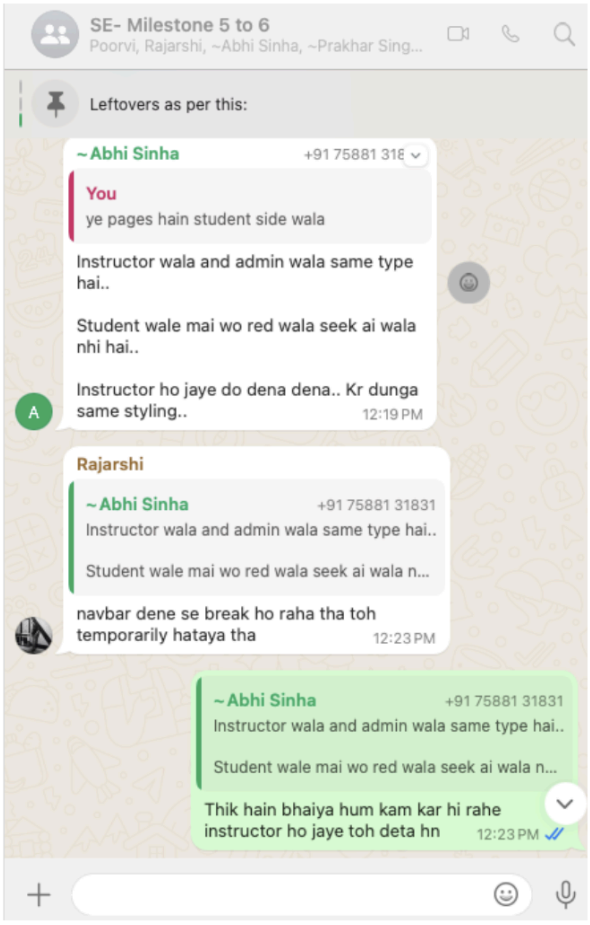
Open 0 **Closed 9** Author Labels Projects Milestones Assignees Types Newest

- **Loading Animation Stuck During AI Response**
  #10 · by soumyadeep-git was closed 5 hours ago
- **Flask Server Crashes Under 50+ Concurrent Users**
  #9 · by soumyadeep-git was closed 5 hours ago
- **Environment Variables Not Loaded in Docker**
  #8 · by soumyadeep-git was closed 5 hours ago
- **Virtual Environment Setup Fails on Windows**
  #7 · by soumyadeep-git was closed 5 hours ago
- **Hardcoded Prompts Fail Non-STEM Courses**
  #6 · by soumyadeep-git was closed 5 hours ago
- **Agent Leaks Assignment Solutions via RAG**
  #5 · by soumyadeep-git was closed 5 hours ago
- **Unoptimized LLM Payloads Increase Costs**
  #4 · by soumyadeep-git was closed 5 hours ago
- **Rate Limiting Not Enforced**
  #3 · by soumyadeep-git was closed 5 hours ago
- **CORS Configuration Not Working**
  #2 · by soumyadeep-git was closed yesterday









Conclusion:

The successful completion of Milestone 6 marks a significant step forward in the project's development, ensuring a well-structured and scalable architecture for both frontend and backend components. By leveraging Vue.js for dynamic UI, Flask for efficient API management, and robust tools for version control, testing, and deployment, the team has built a solid foundation for future enhancements. With streamlined collaboration, agile project management, and well-documented workflows, the application is now well-positioned for further refinements, user testing, and eventual production deployment.