Milestone - 6 Team - 21

IIT Madras
BS Degree

SOFTWARE ENGINEERING PROJECT

SMART SEEK PORTAL

Prepared by :

Sayan Hrik
21f3002833@ds.study.iitm.ac.in

Avijeet Palit
21f1005675@ds.study.iitm.ac.in

Uroosha Rahat
21f1002968@ds.study.iitm.ac.in

Preetam Mukherjee
21f1002436@ds.study.iitm.ac.in

Ayush Singh Rana
21f1005671@ds.study.iitm.ac.in

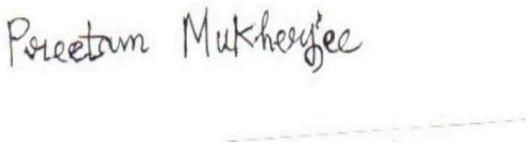
Ramesh Kumar Chandran
21f2000549@ds.study.iitm.ac.in

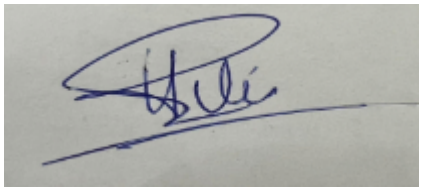
Bodhisatwa Bhattacharya
21f1000270@ds.study.iitm.ac.in

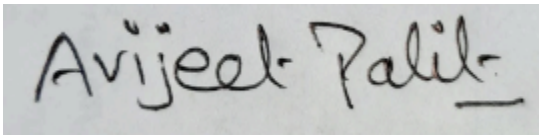
Sachin Kumar
21f2000143@ds.study.iitm.ac.in

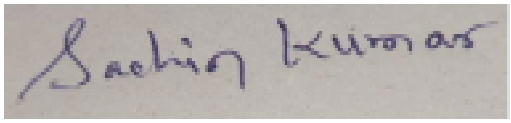
Self-Declaration and Honour Code

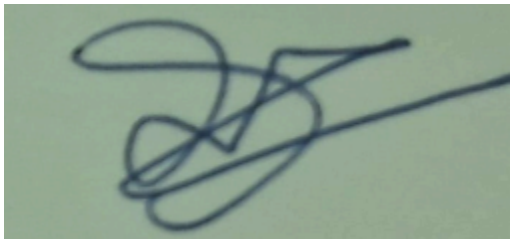
I Sayan Hrik(21f3002833), I Avijeet Palit(21f1005675), I Uroosha Rahat(21f1002968), I Preetam Mukherjee(21f1002436), I Ayush Singh Rana(21f1005671), I Ramesh Kumar Chandran(21f2000549), I Bodhisatwa Bhattacharya(21f1000270), I Sachin Kumar(21f2000143)
declare that I will not use any ideas, writings, code or work that is not my own or my groups with the intention of claiming it is me or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.
I acknowledge that failing in any of the above constitutes plagiarism and in that case, the Institute will take appropriate disciplinary action.











Contents

Problem Statement

Effective integration of Generative AI into programming learning environments

The primary way in which learners engage with content in the IITM BS degree program is through the SEEK portal. The portal contains learning videos, assignments, resources, practice and graded assignments, and programming quizzes. The portal can be described as a learning environment and provides opportunities for self-learning, by enabling learners to learn at their own pace and schedule.

With the rapid advancement of generative AI (GenAI) technologies, there are promising possibilities of effective integration of GenAI into these learning environments. The goal of this term's software engineering project is to examine ways in which GenAI can be effectively integrated into learning environments such as SEEK. There are various ways in which possible integrations can be done. Here are some examples -

1. Providing effective feedback in programming tests - In programming tests, after the results are announced, the feedback usually just mentions the test cases which passed/failed. Can GenAI be used to provide additional effective feedback for the code you have written?

2. Providing support while solving programming problems - Can GenAI support learners while they are solving practice programming problems?

3. Better engagement with content - Can GenAI add interactivity to otherwise static content such as pdfs, documents, and videos?

The above examples are just to give you an idea about the features which are possible. You are free to use/not use the above examples and come up with other innovative ideas.

Project Summary

Smart Seek Portal: Enhancing Learning with AI

In our software engineering project, we developed the Smart Seek Portal—a platform designed to enhance the learning experience for students. Here's how we approached the project:

1. **User Identification:**

   We categorized users into three roles:

   - Primary Users (Students): These are the students who use the platform to access course materials and resources.
   - Secondary Users (Instructors): Instructors upload course materials and manage content.
   - Tertiary Users (Admins): Admins oversee the application's overall functionality.

2. **Requirements Gathering:**
   - Following SMART guidelines, we collected user stories. Interacting with other BS Degree students and researching AI's potential for student-instructor interactions helped us create a comprehensive list of requirements.
3. **Project Management:**
   - We created a work breakdown structure (WBS) and distributed tasks among team members. JIRA served as our project management tool, helping us manage sprints, user stories, and features.
   - Regular scrum meetings (twice a week via Google Meet) kept us updated on project progress.
4. **Design Phase:**
   - We developed both paper and digital prototypes using tools like PowerPoint and Figma. These prototypes guided the definition of app components.

- The database schema and entity-relationship (ER) diagrams were crucial for structuring our data. We identified models, properties, and relationships.

5. **Collaborative Coding:**
    - GitHub facilitated collaborative coding. We started by creating APIs (both general and AI-related).
    - Our YAML file was updated, and we rigorously tested the APIs using Postman and Pytest, designing thorough unit test cases.
    - We then created the frontend using VueJS and VueRoutter and created all the required components and integrated GenAI.

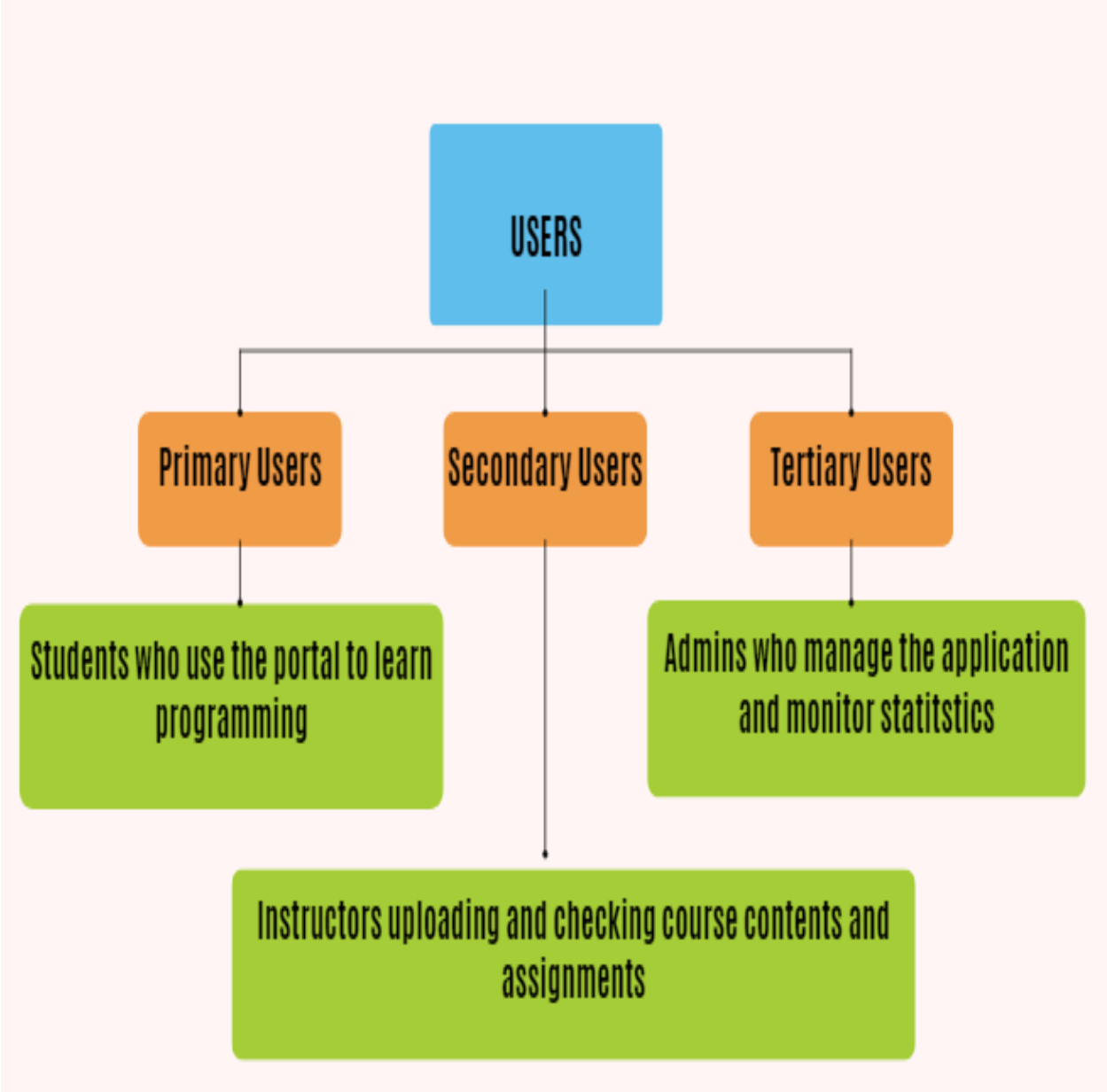
6. **Testing**

    - Finally, we conducted unit and integration testing to ensure the entire application worked seamlessly.
    - We used Pytest to create test cases and facilitate the testing process.
    - We then did Alpha testing where all the members of our group tested our app and compared the working of the features with the initial requirements we had identified.

6. **Deployment:**
    - Following testing we deployed the application so that users can use and interact with it.

Milestone 1

User Identification



1. **Learner Journey Maps**:

1.1 Without GenAI Integration:

1.1.1 Scenario1: **Submitting a Programming Assignment:**

(i) Research and Understanding:

- Student: Searches for resources to understand the assignment requirements.
- Pain Point: Time-consuming to find relevant resources.

(ii) Coding:

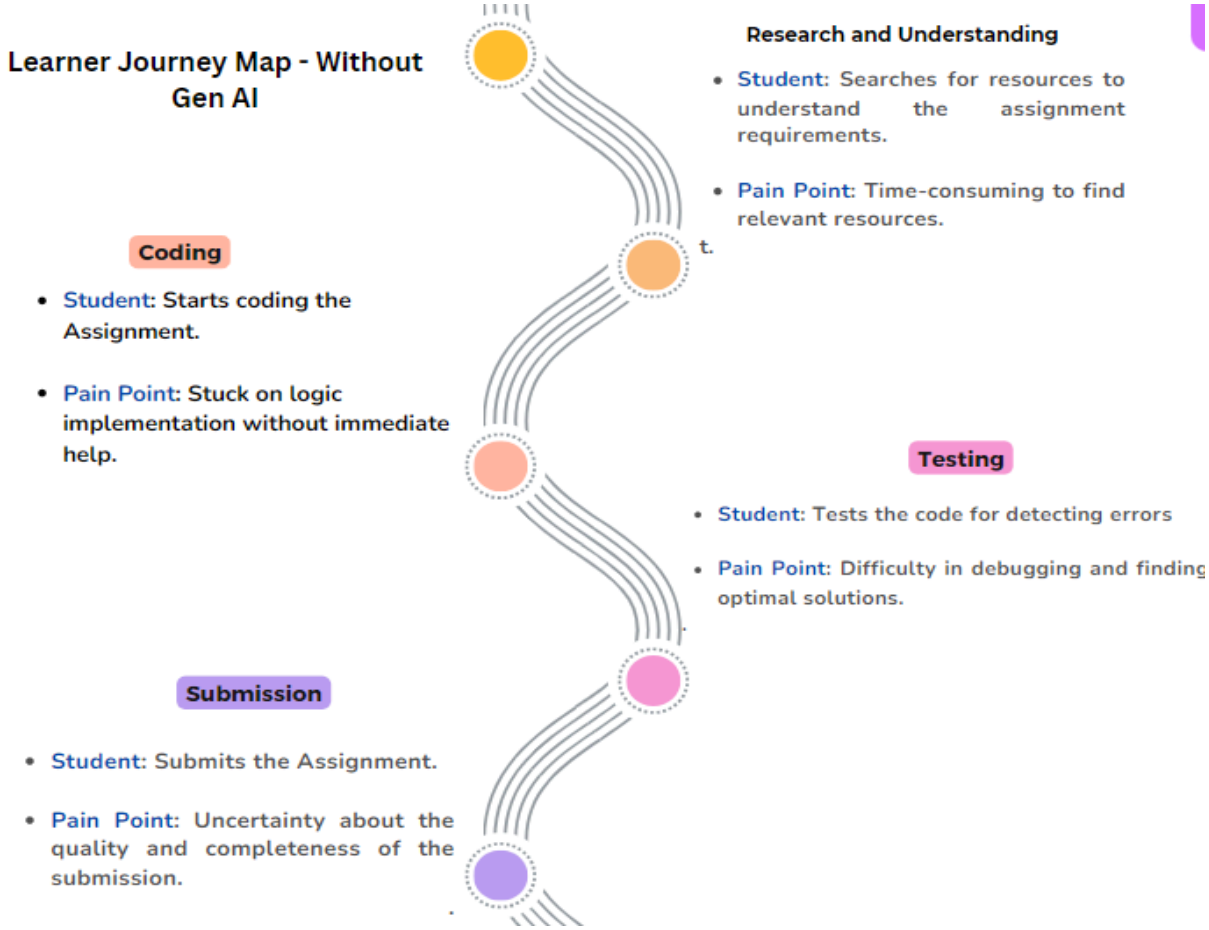
- Student: Starts coding the Assignment.
- Pain Point: Stuck on logic implementation without immediate help.

(iii) Testing:

- Student: Tests the code for detecting errors.
- Pain Point: Difficulty in debugging and finding optimal solutions.

(iv) Submission:

- Student: submits the Assignment.
- Pain Point: Uncertainty about the quality and completeness of the submission.



1.1.2 Scenario2: **Doubts and Feedbacks:**

(i)Access Discourse:

- Student: Goes to the discourse portal and types the doubt.
- Pain Point: Time-consuming to get a response.

(ii) Wants to extend the Conversation:

- Student: Does not understand the solution completely
- Pain Point: More time consuming and a hurdle in the learning journey.
- Instant doubt clearing is not possible.

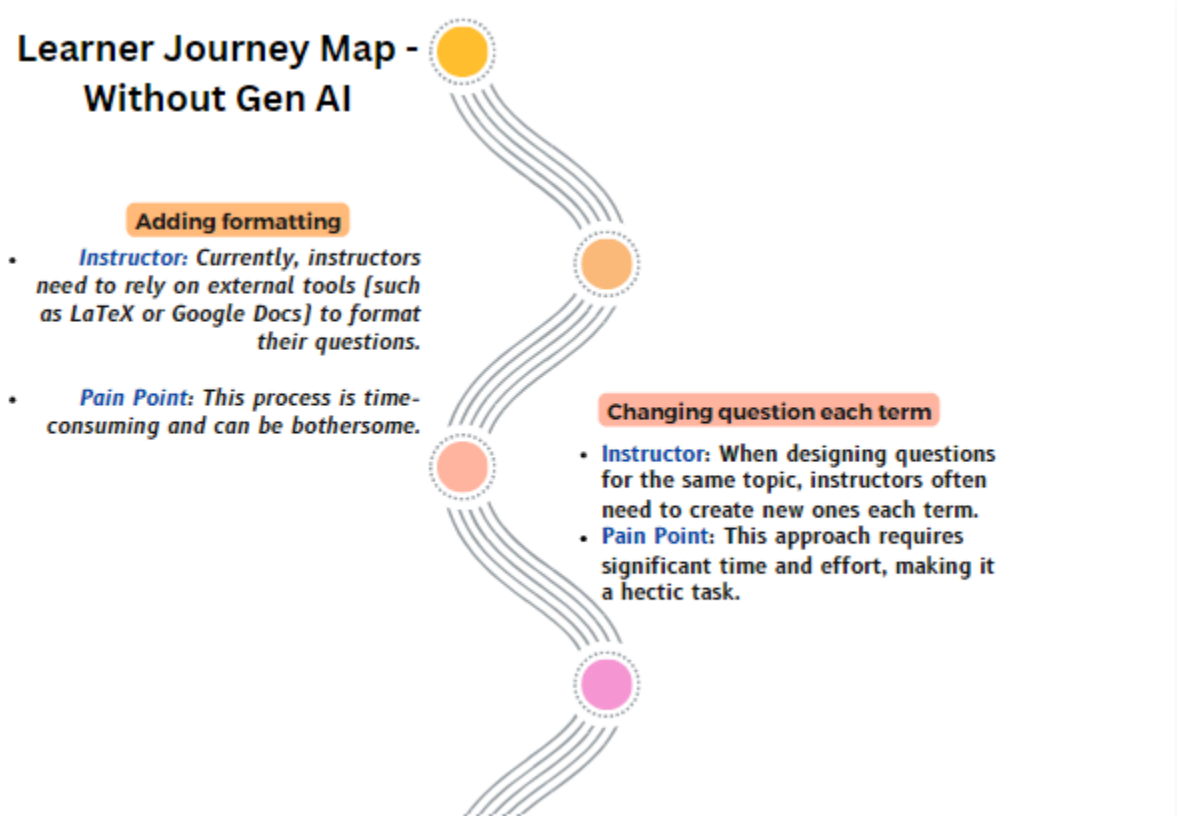
1.1.3 Scenario3: **Question Creation (Instructor):**

(i)Adding formatting:

- Instructor: Currently, instructors need to rely on external tools (such as LaTeX or Google Docs) to format their questions.
- Pain Point: This process is time-consuming and can be bothersome.

(ii)Changing question each term:

- Instructor: When designing questions for the same topic, instructors often need to create new ones each term.
- Pain Point: This approach requires significant time and effort, making it a hectic task.



1.2 With GenAI Integration:

1.2.1 Scenario1: **Submitting a Programming Assignment:** .

(i) Coding:

- Student: Starts coding the Assignment.

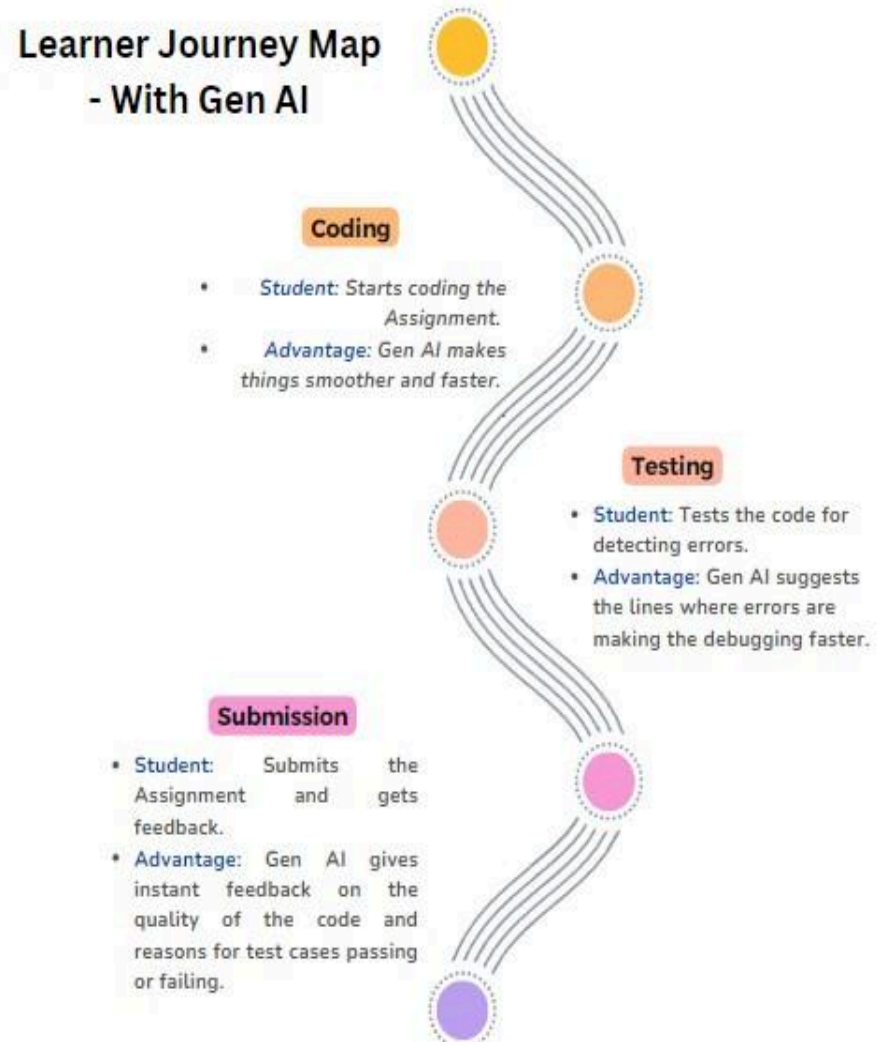
- Advantage: Gen AI makes things smoother and faster.

(iii) Testing:

- Student: Tests the code for detecting errors.
- Advantage: Gen AI suggests the lines where errors are making the debugging faster.

(iv) Submission:

- Student: Submits the Assignment and gets feedback.
- Advantage: Gen AI gives instant feedback on the quality of the code and reasons for test cases passing or failing.



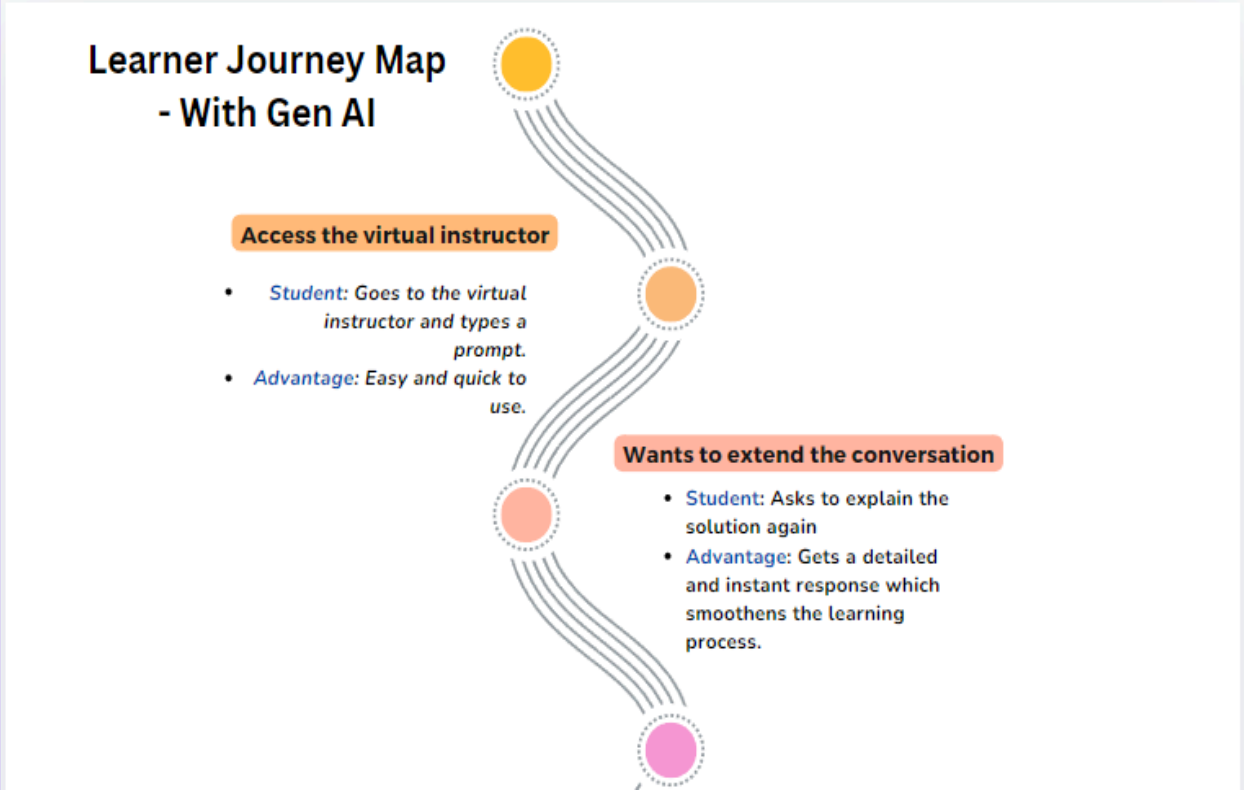
1.2.2 Scenario2: **Doubts and Feedback:**

(i)Access the virtual instructor:

- Student: Goes to the virtual instructor and types a prompt.
- Advantage: Easy and quick to use.

(ii) Wants to extend the conversation:

- Student: Asks to explain the solution again
- Advantage: Gets a detailed and instant response which smoothens the learning process.



1.2.3 Scenario3: **Question Creation (Instructor):**

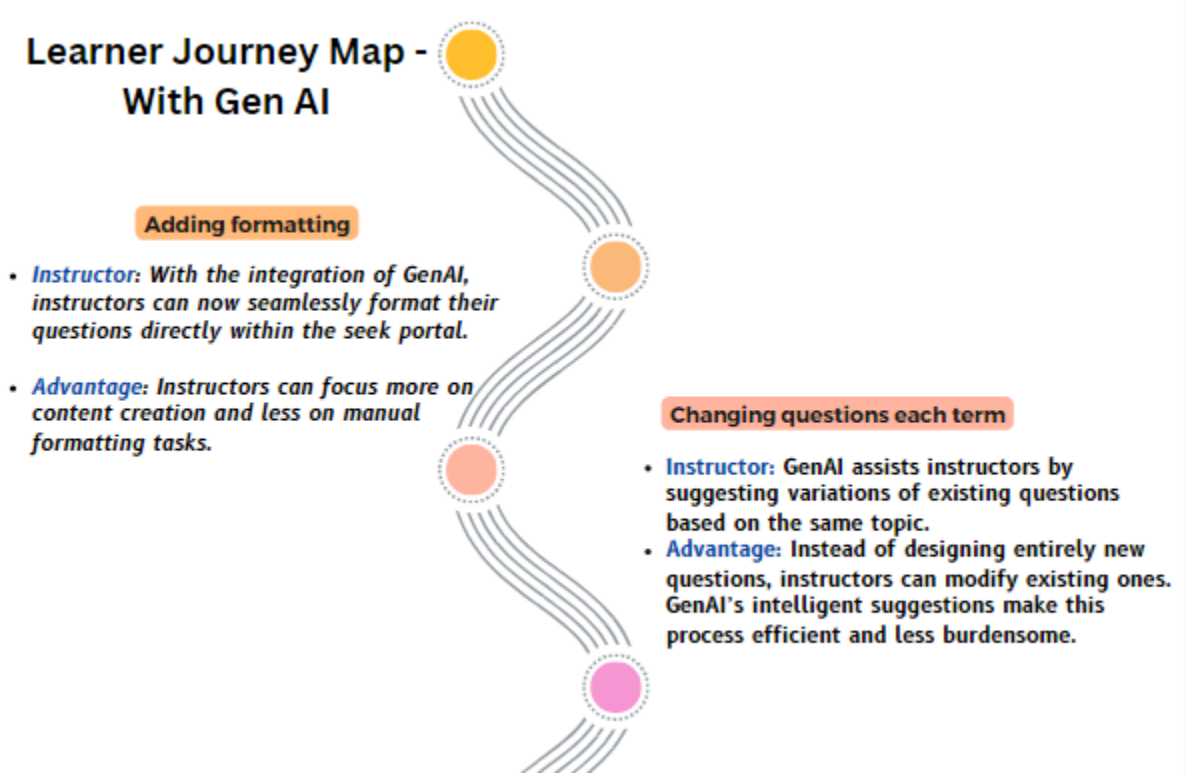
(i)Adding formatting:

- Instructor: With the integration of GenAI, instructors can now seamlessly format their questions directly within the seek portal.

- Advantage:Instructors can focus more on content creation and less on manual formatting tasks.

(ii) Changing questions each term:

- Instructor: GenAI assists instructors by suggesting variations of existing questions based on the same topic.
- Advantage: Instead of designing entirely new questions, instructors can modify existing ones. GenAI's intelligent suggestions make this process efficient and less burdensome.

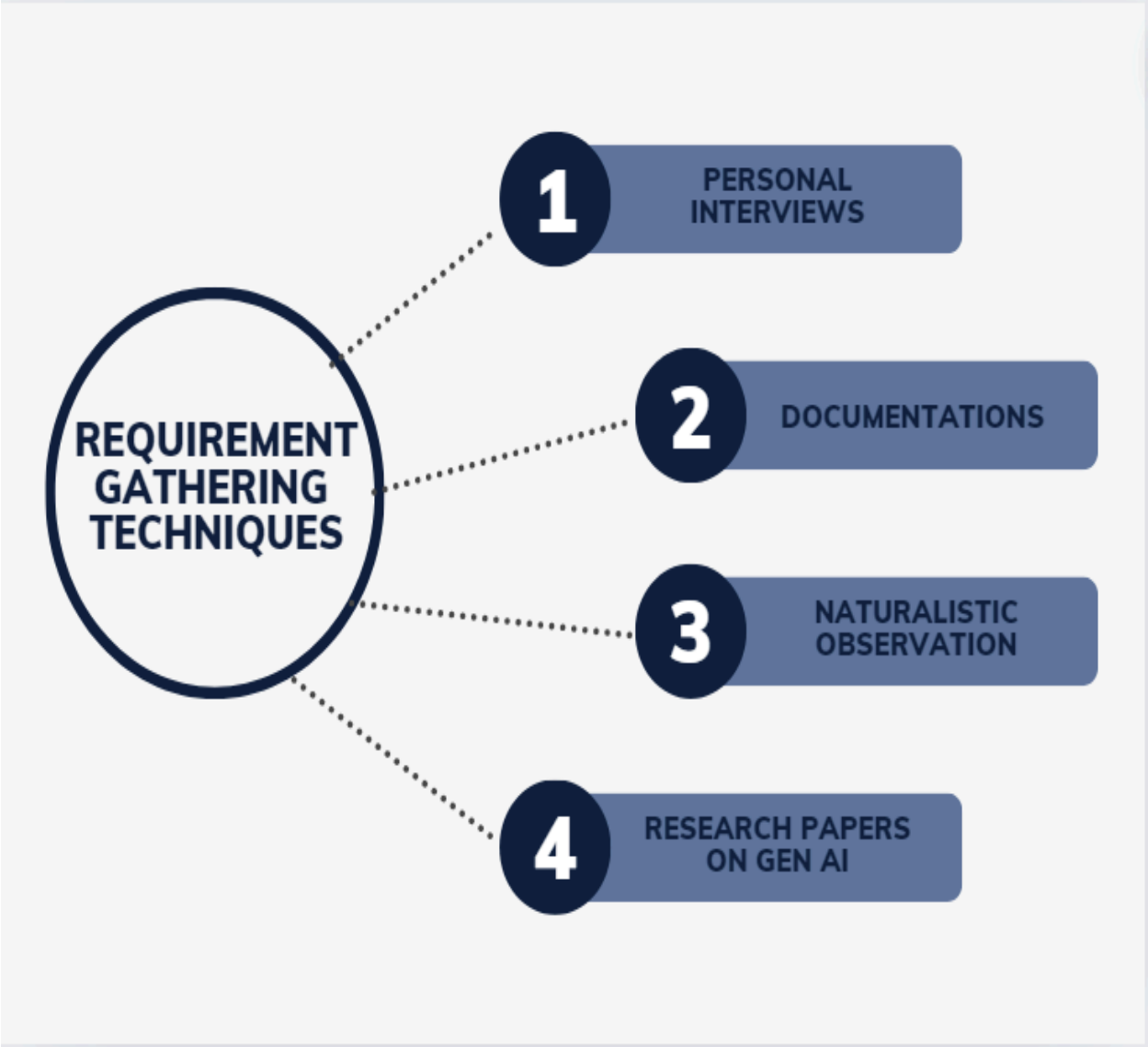


Requirement Gathering Techniques

1. **Personal Interviews** - Our team has interacted with a few peers of our BS Degree Programme. We asked them about the difficulties they face while using the seek portal and what features involving Generative AI will make their learning experience better and more engaging / interactive.

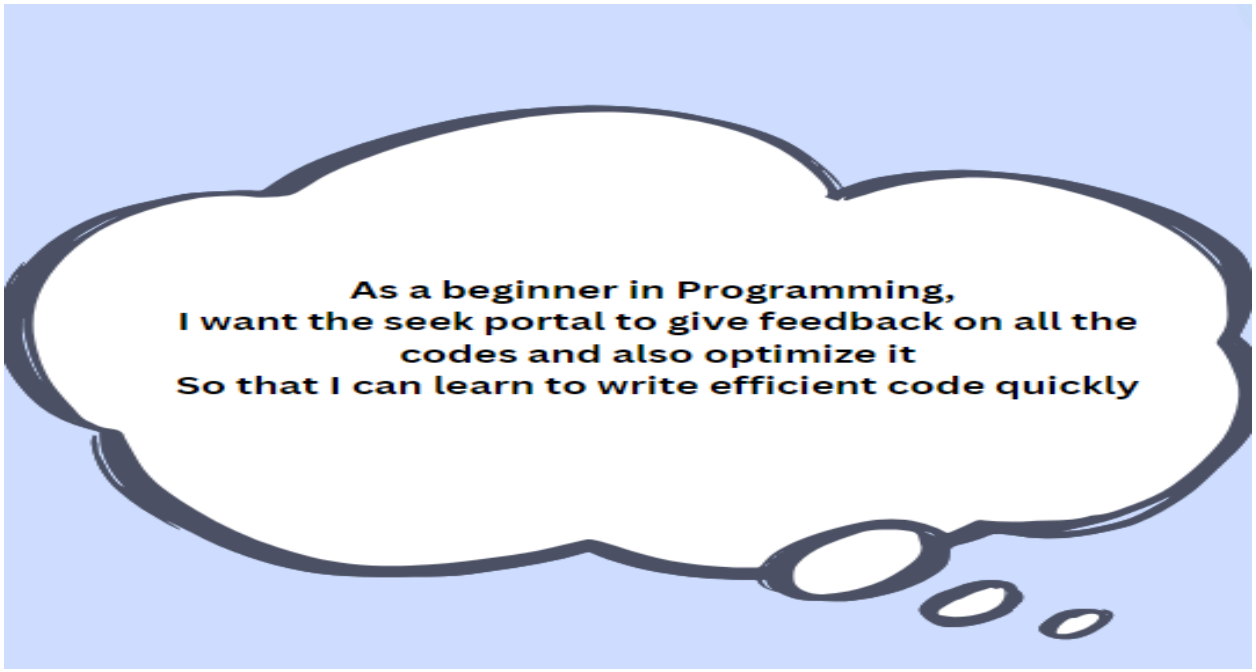
2. **Naturalistic Observations** - As we are students of the BS Degree Programme we have observed each other and our fellow peers through the years and can identify what ways they use GenAI to improve their learning journey and overall experience.

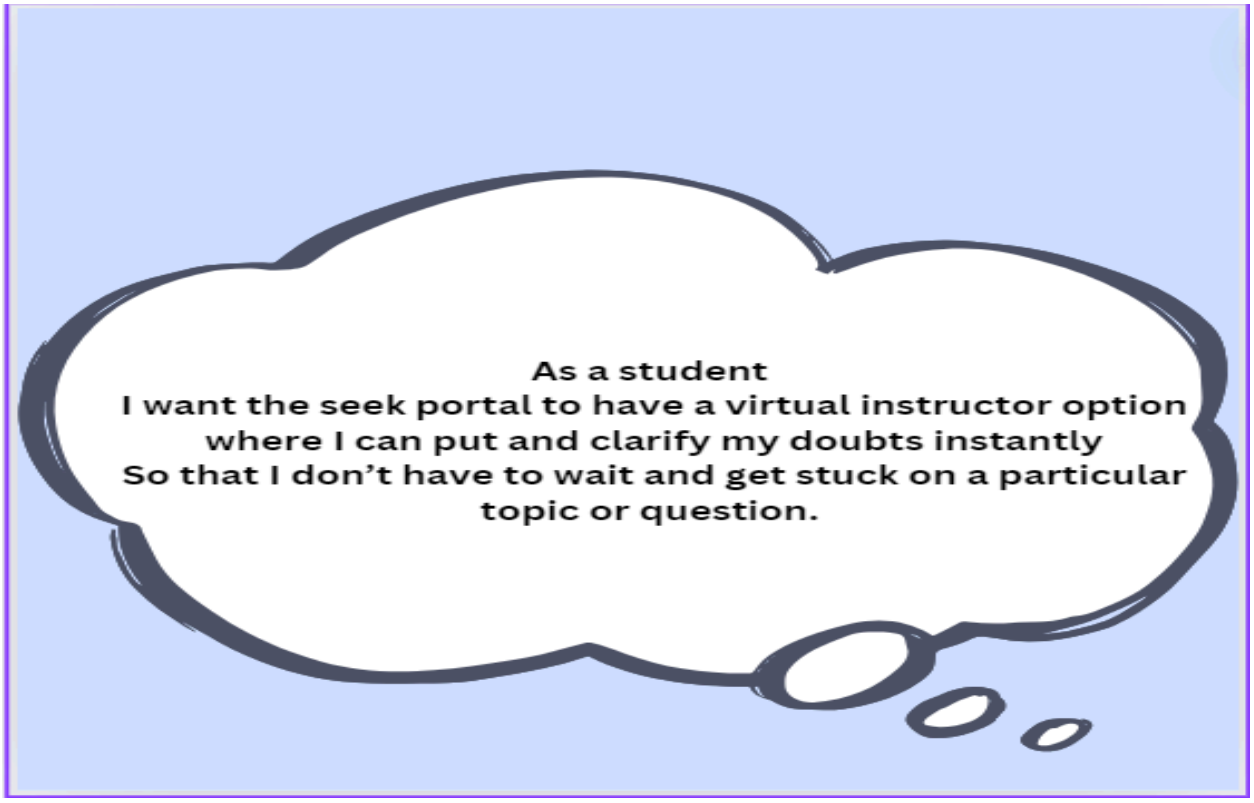
3. **Documentations and Research Papers** - We have gone through some research papers on the advancement of Large Language Models and GenAI and understood their capabilities. This will also help us shape the adequate list of features required in our application to satisfy most of our user requirements.

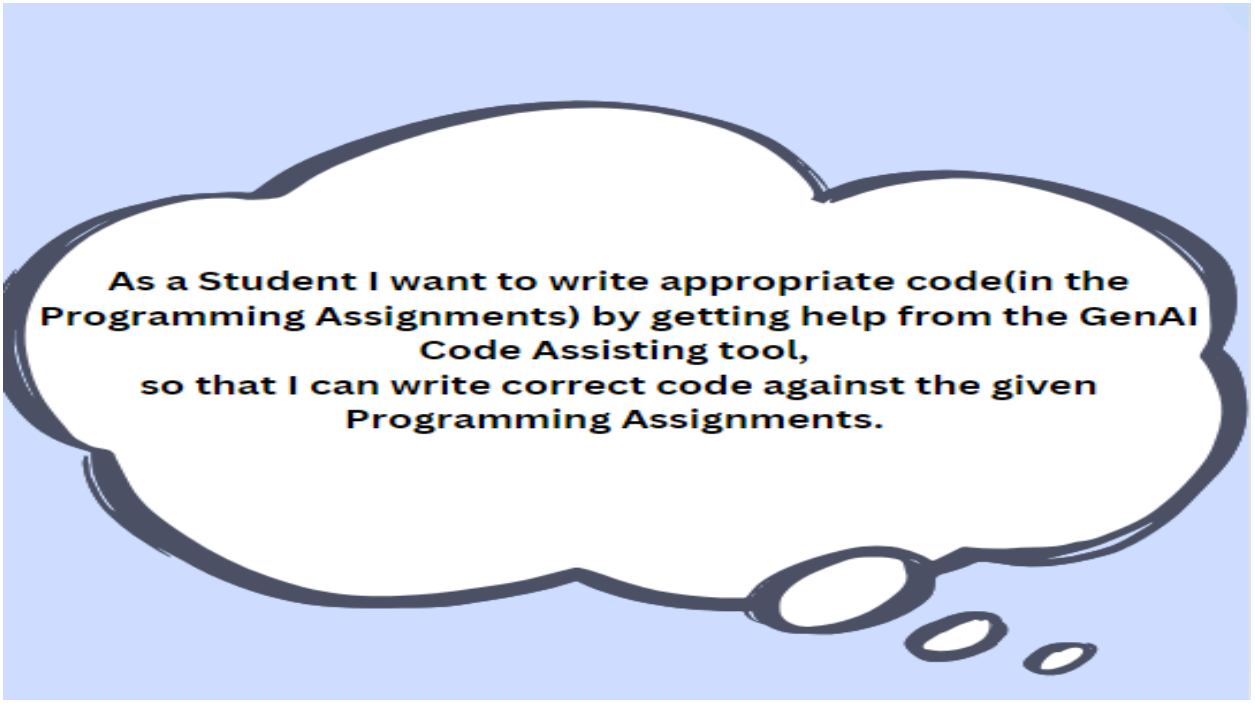


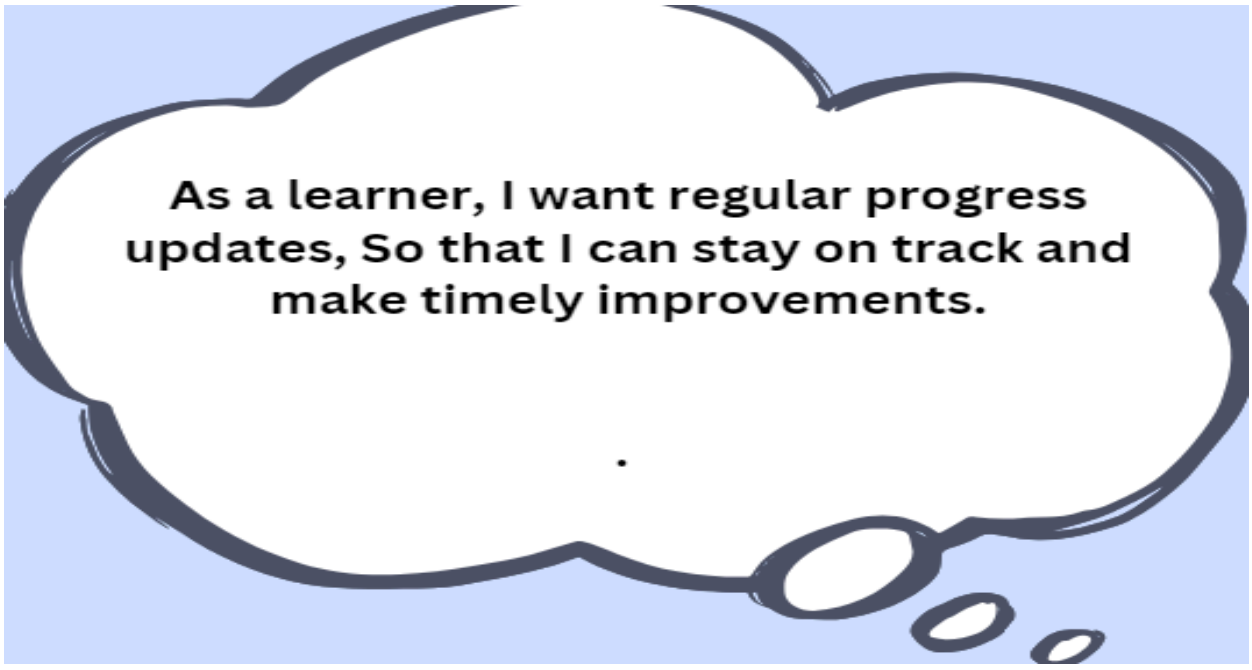
User Stories

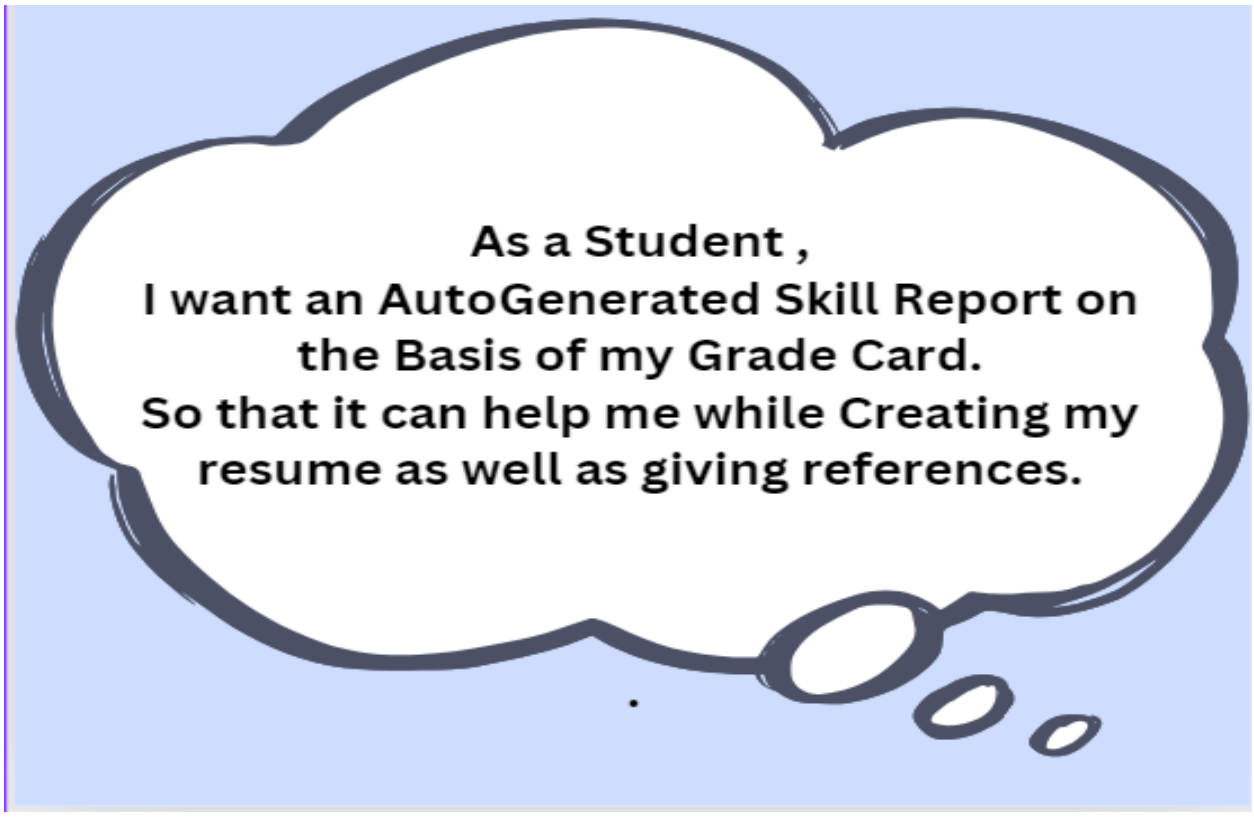
Students - Primary Users



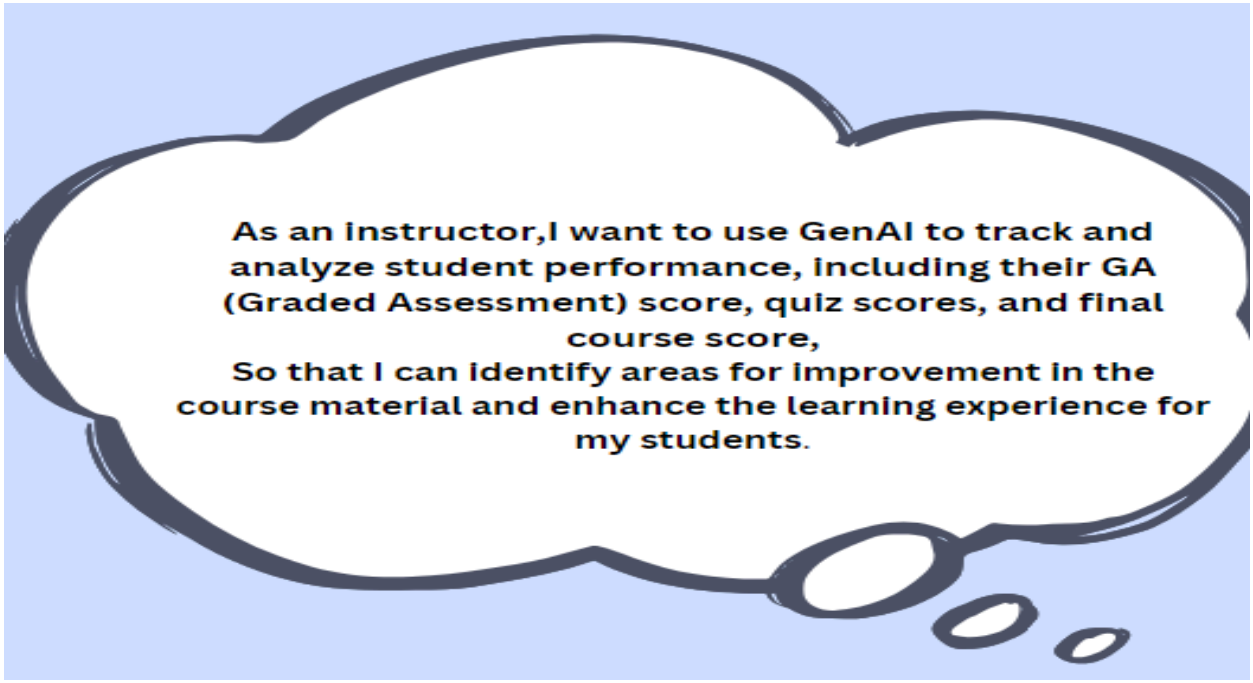

As a student
I want the seek portal to have a virtual instructor option where I can put and clarify my doubts instantly
So that I don't have to wait and get stuck on a particular topic or question.

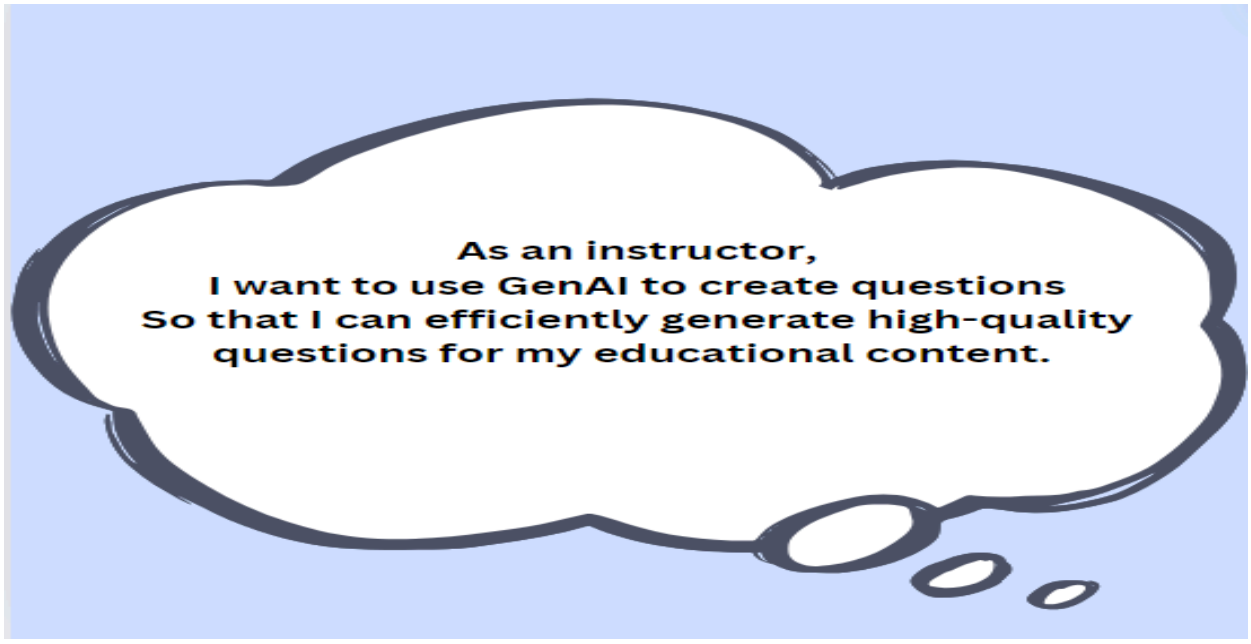

As a Student I want to write appropriate code(in the Programming Assignments) by getting help from the GenAI Code Assisting tool,
so that I can write correct code against the given Programming Assignments.

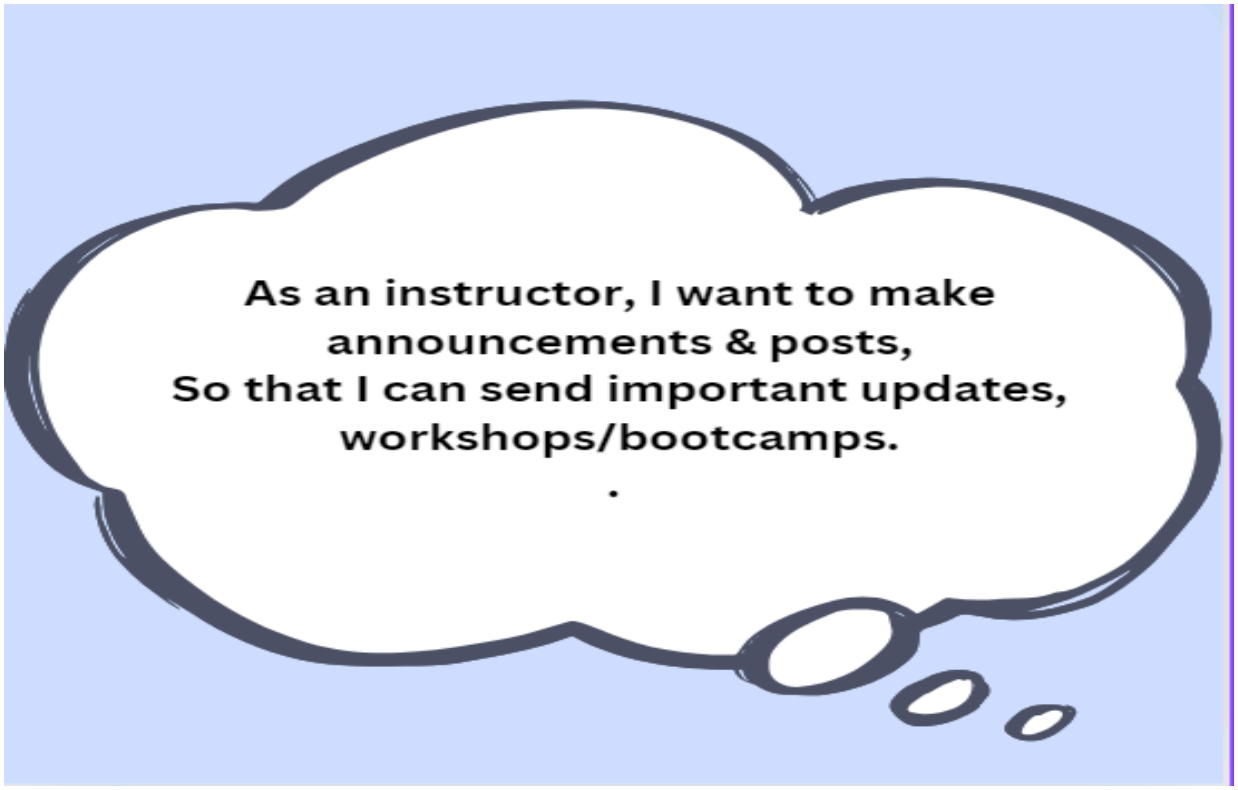

As a learner, I want regular progress updates, So that I can stay on track and make timely improvements.

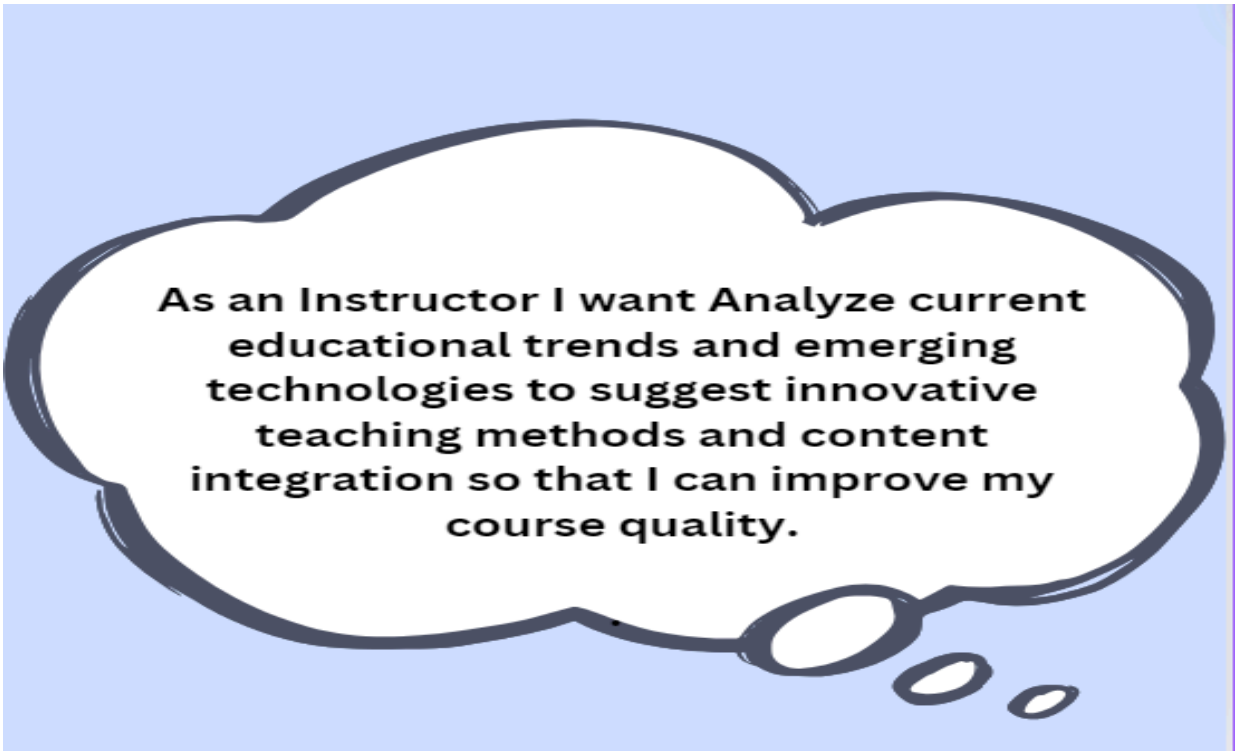

As a Student ,
I want an AutoGenerated Skill Report on the Basis of my Grade Card.
So that it can help me while Creating my resume as well as giving references.

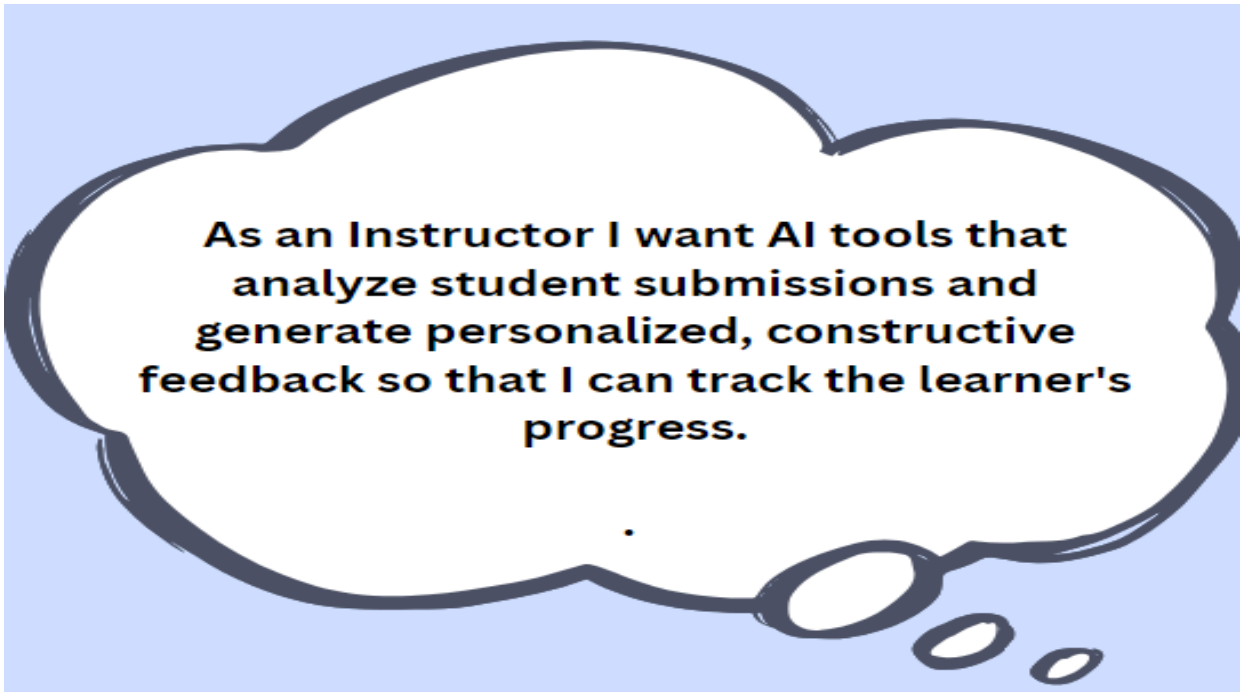
Instructors - SecondaryUsers



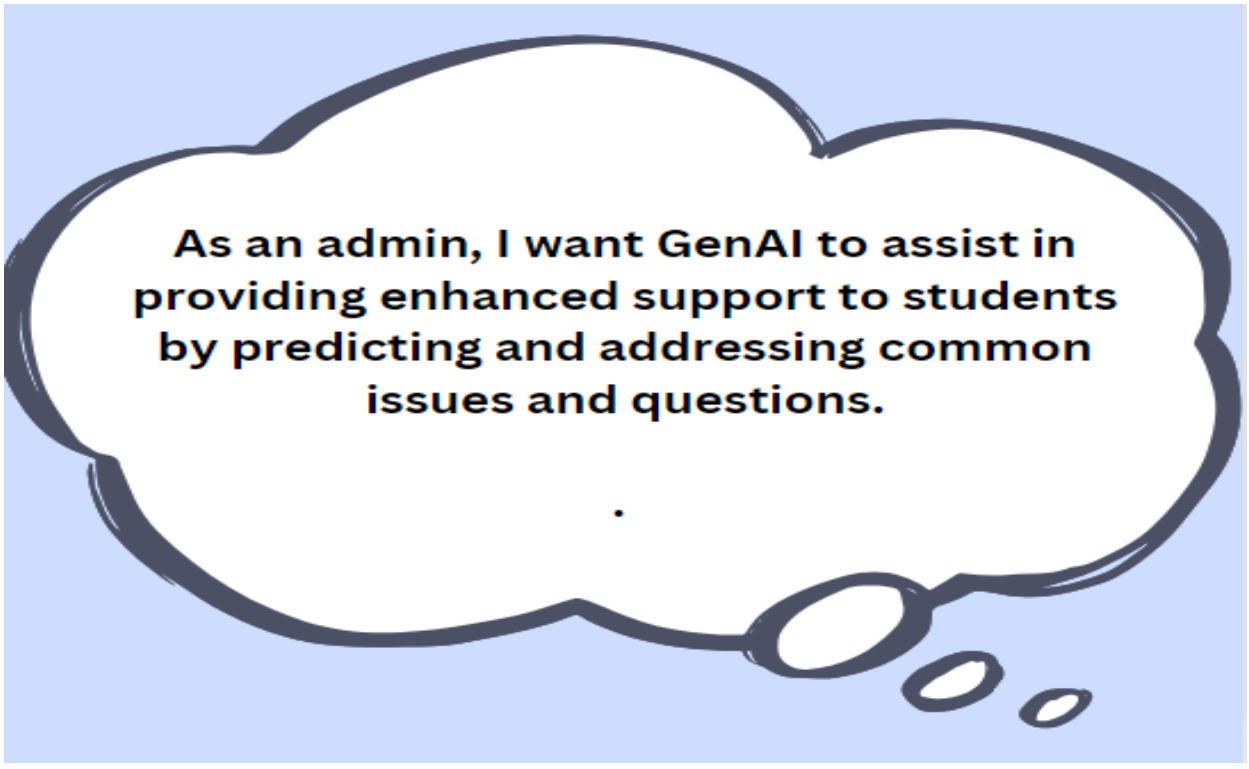


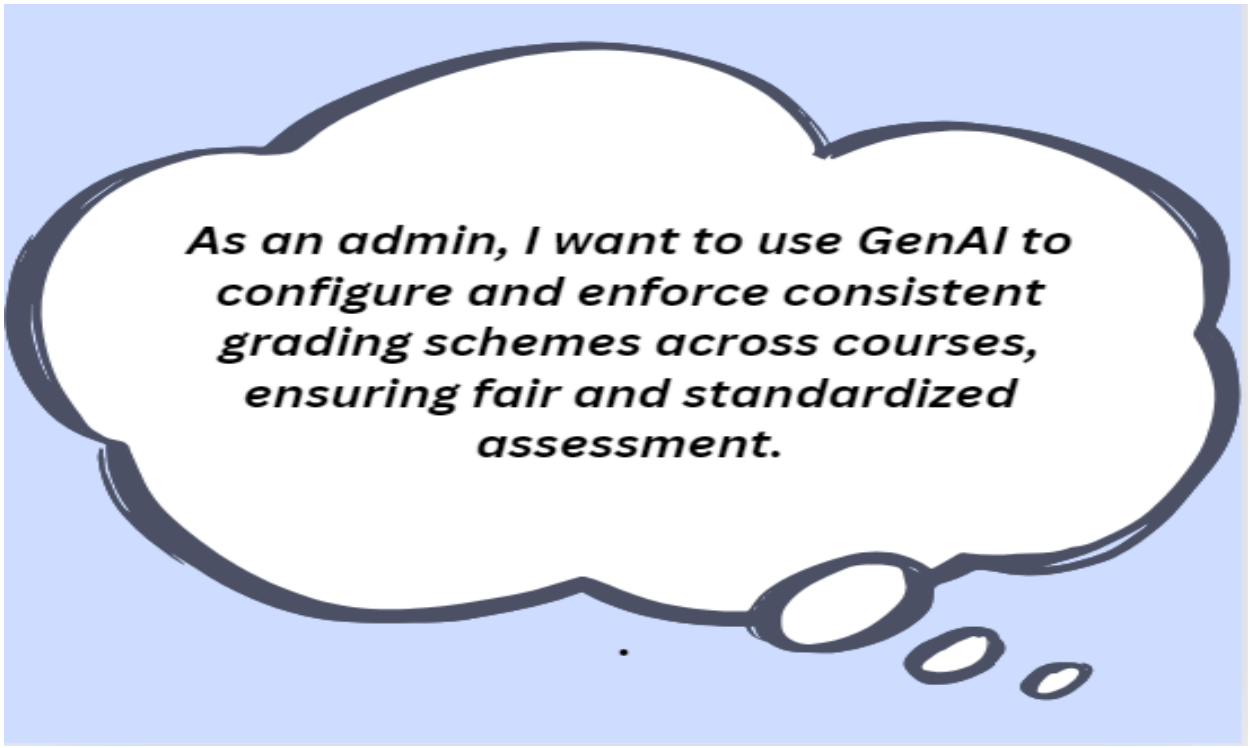

As an instructor, I want to make announcements & posts,
So that I can send important updates, workshops/bootcamps.


As an Instructor I want Analyze current educational trends and emerging technologies to suggest innovative teaching methods and content integration so that I can improve my course quality.



Admins - Tertiary Users




As an admin, I want to use GenAI to configure and enforce consistent grading schemes across courses, ensuring fair and standardized assessment.

Milestone 2

Storyboards

STORY BOARD : ADMIN – INSTRUCTOR

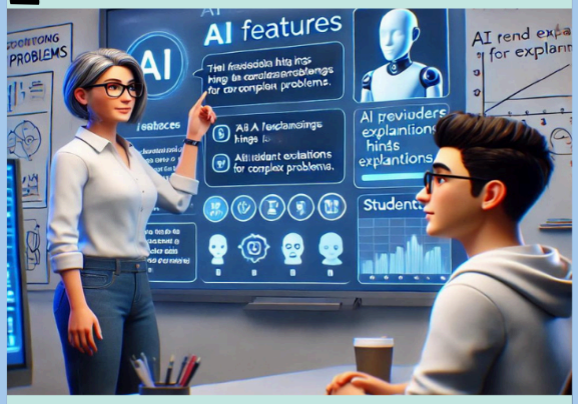
STORY BOARD : INSTRUCTOR - INSTRUCTOR

STORY BOARD : INSTRUCTOR – STUDENT

1

2

Instructor: If you're stuck on a problem, the AI provides hints and even detailed explanations.



Student: So, it customizes the help based on the problem I'm facing?

3

Instructor: Exactly! It assesses your current solution and offers tips to improve or correct it.

Student: That sounds helpful, especially for tougher topics.

4

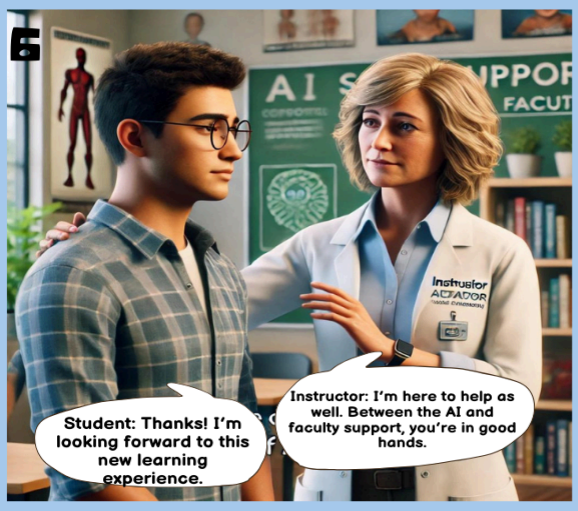
Instructor: It also tracks your progress and suggests review materials based on your performance.

5

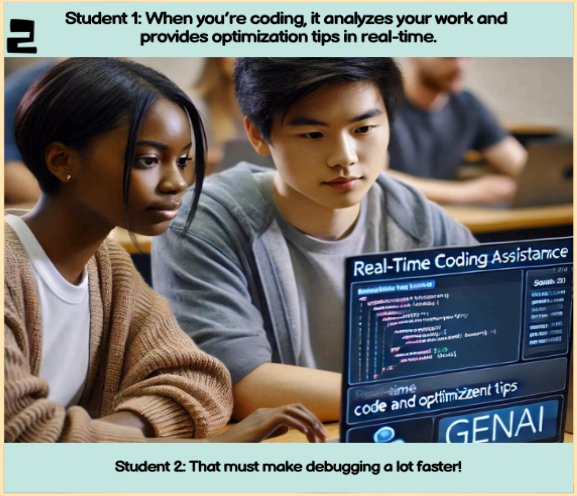
Instructor: It's designed to assist across all courses. Your feedback will be crucial to enhance its capabilities.

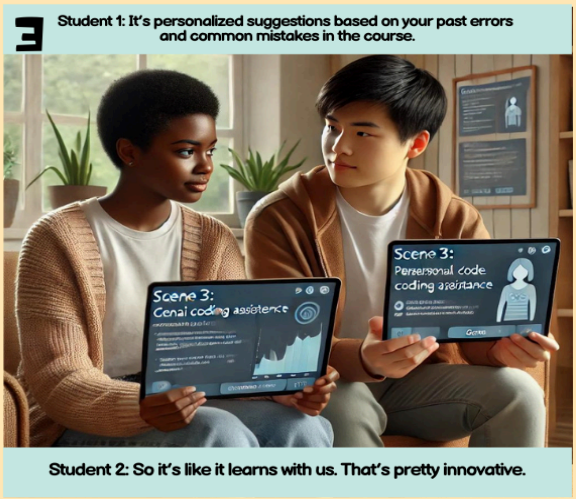
Student: I'll definitely share how it goes. This personalized a aproach might just be what I needed.

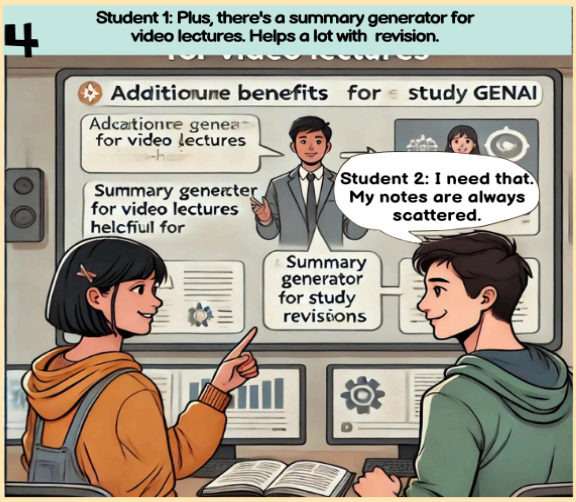
6

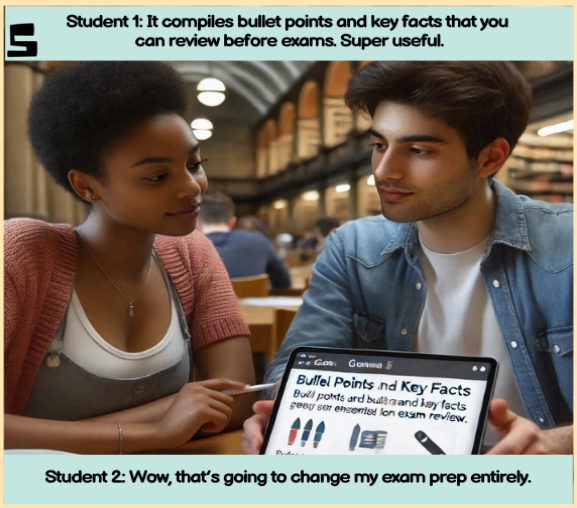


STORY BOARD : STUDENT – STUDENT

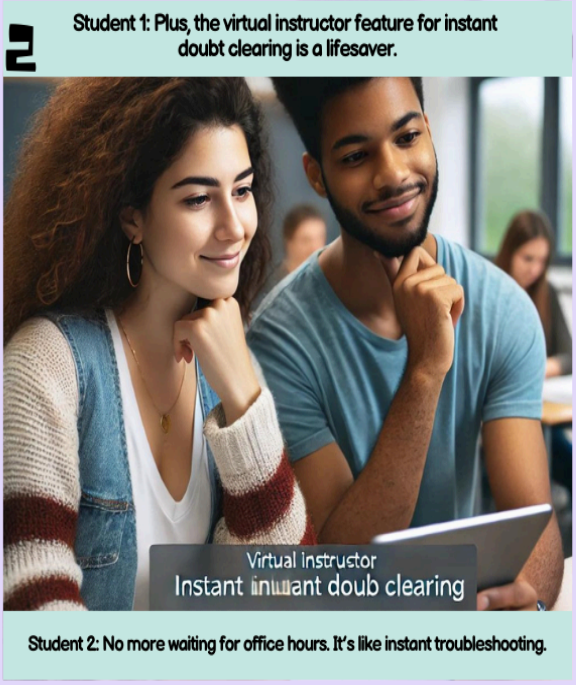






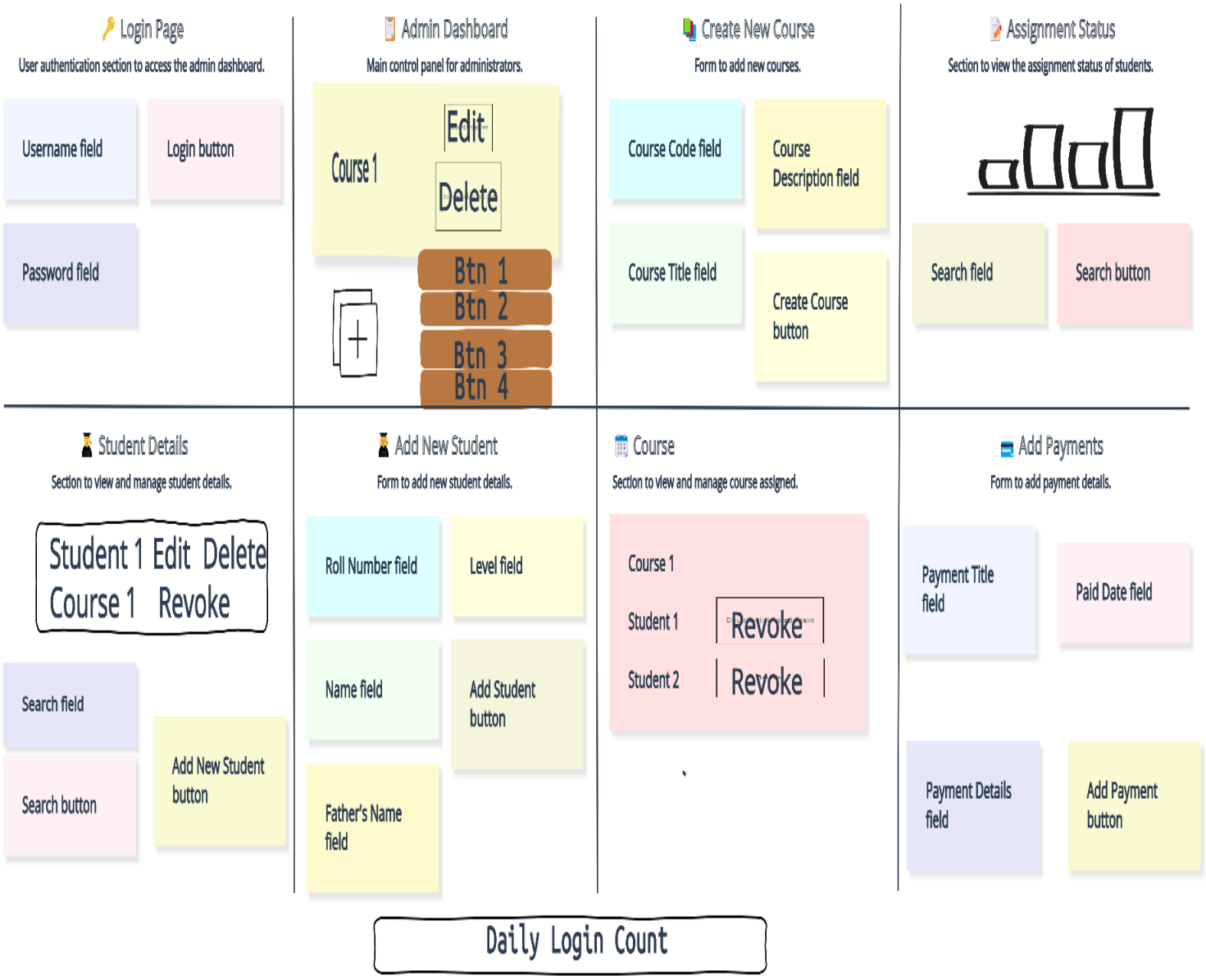


STORY BOARD : STUDENT - STUDENT

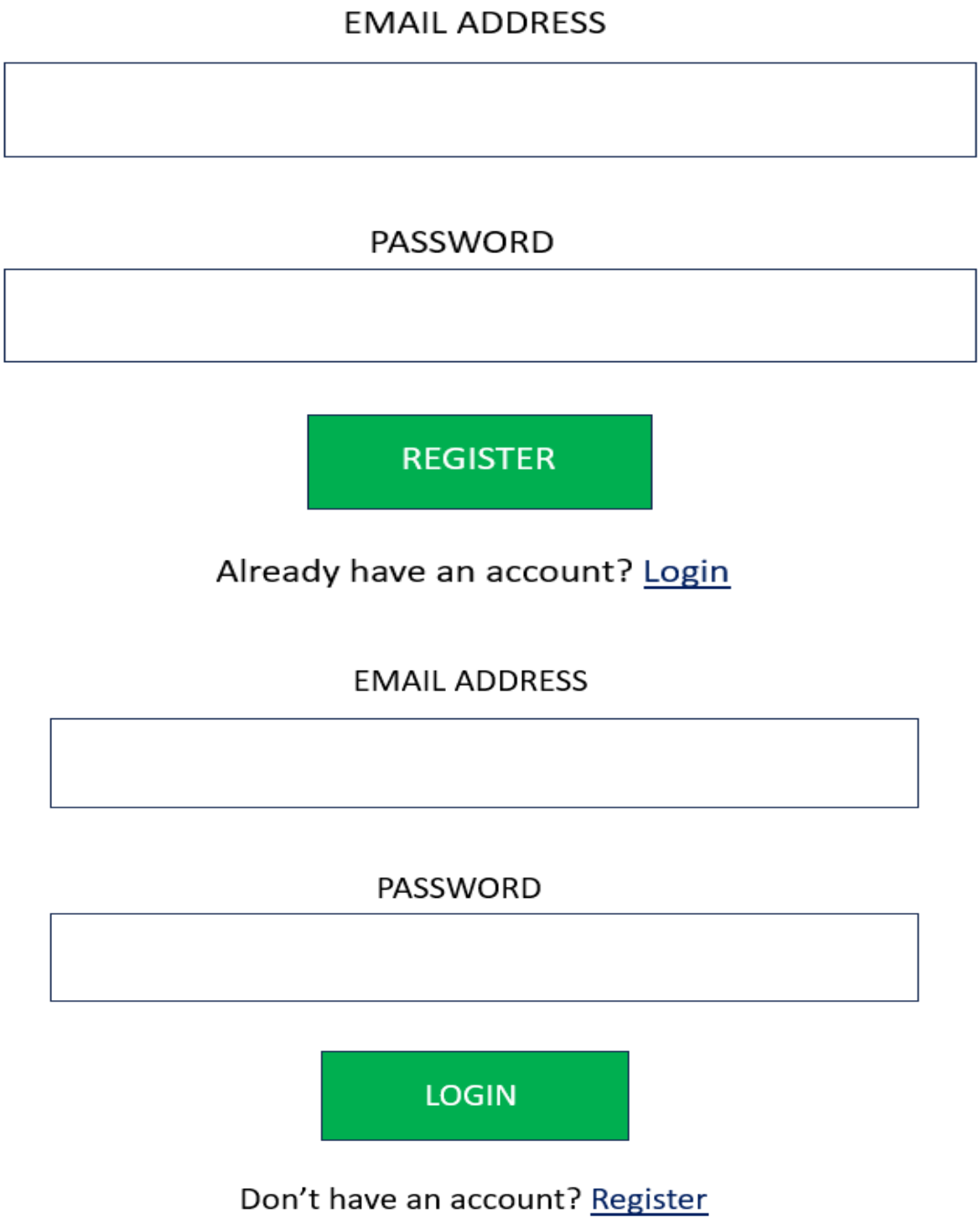


Wireframes

Wireframe - Admin



Wireframe - Student Dashboard



MY COURSE PORTAL

21f400832@ds.study.iitm.ac.in

My Virtual Instructor

Hey there Student !! Having doubts ? Don't worry , I am here to help you out !

Enter you doubt here and get it clarified instantly !

Submit

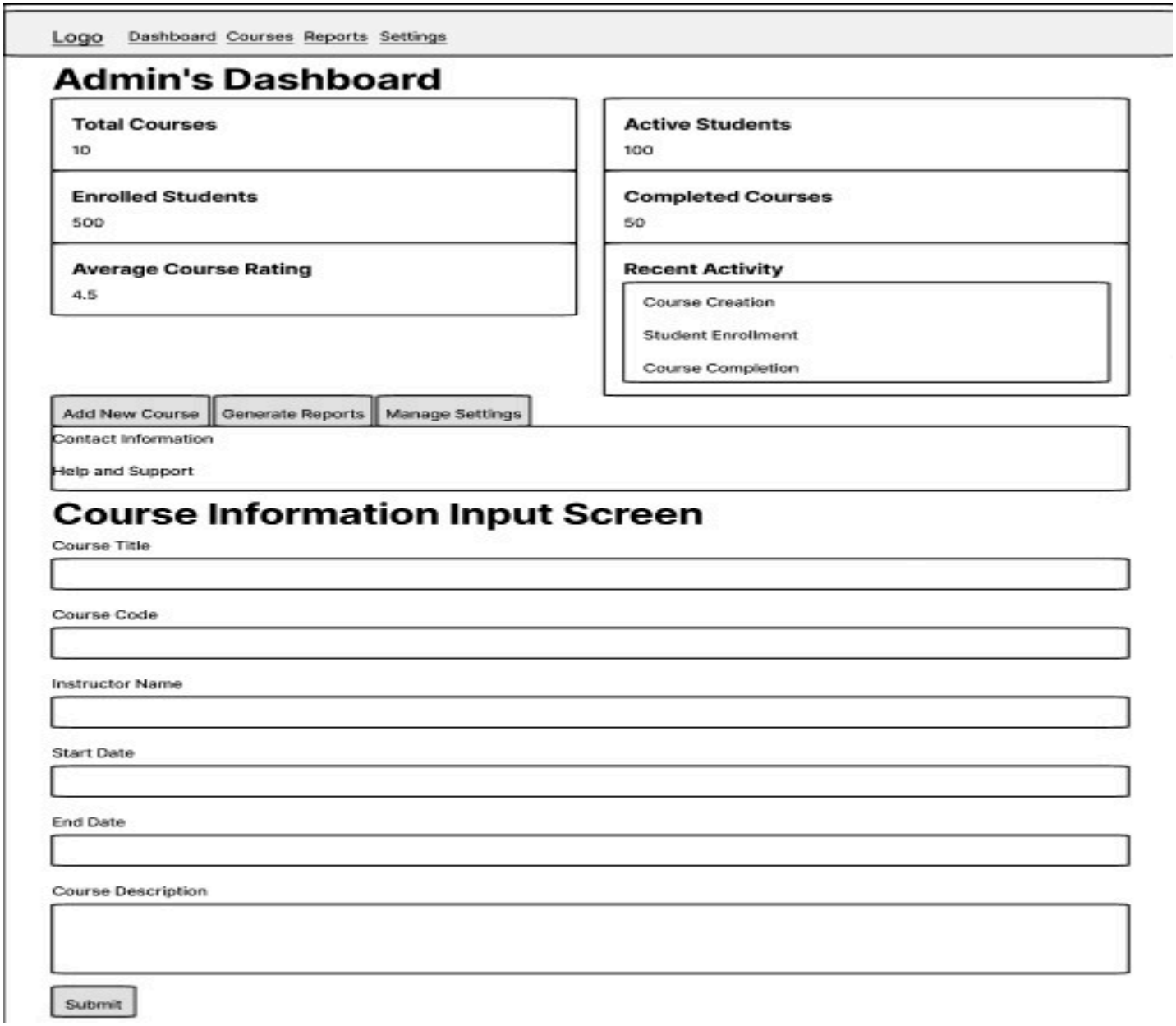
Programming Assignments

Code Feedback

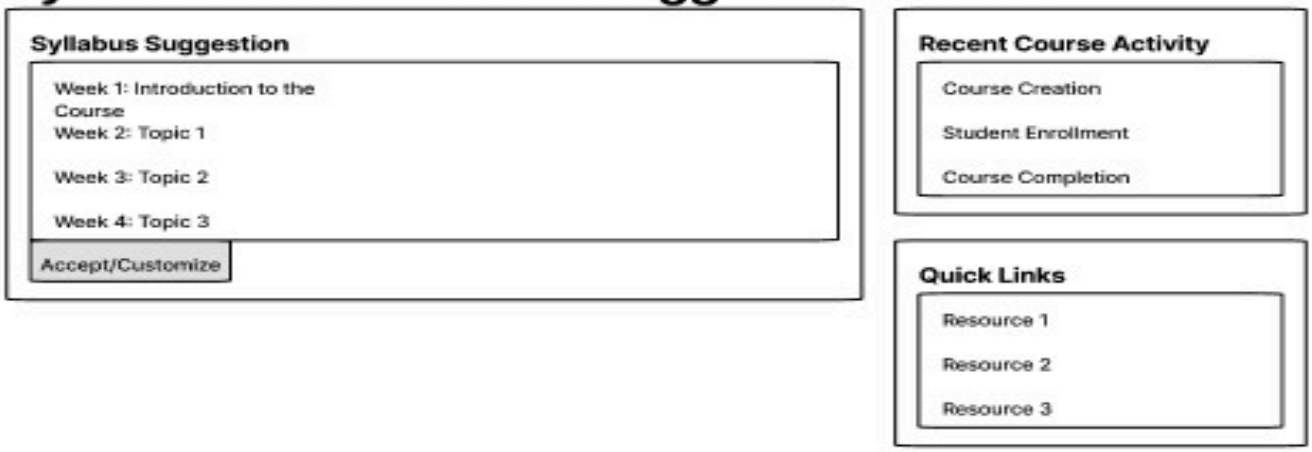
- Your implementation of the iterative binary search looks great! You've correctly divided the array into halves and adjusted the search range based on the comparison with the target value. Here are a few suggestions for improvement:

1. **Comments and Readability**: Consider adding comments to explain each step in your code. This will make it easier for others (and your future self) to understand the logic.
2. **Edge Cases**: Your code handles the general case well, but you might want to consider edge cases:
    1. What if the input array is empty?
    2. What if the target value is not present in the array?

Wireframe - Instructor / Admin

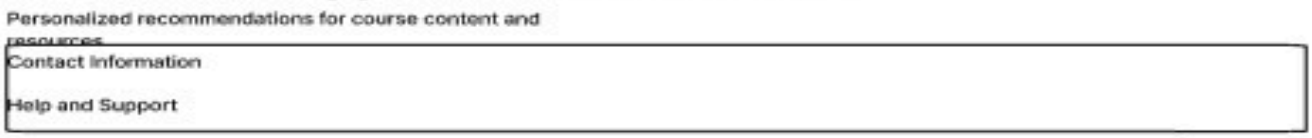


Syllabus and Resources Suggestion Screen

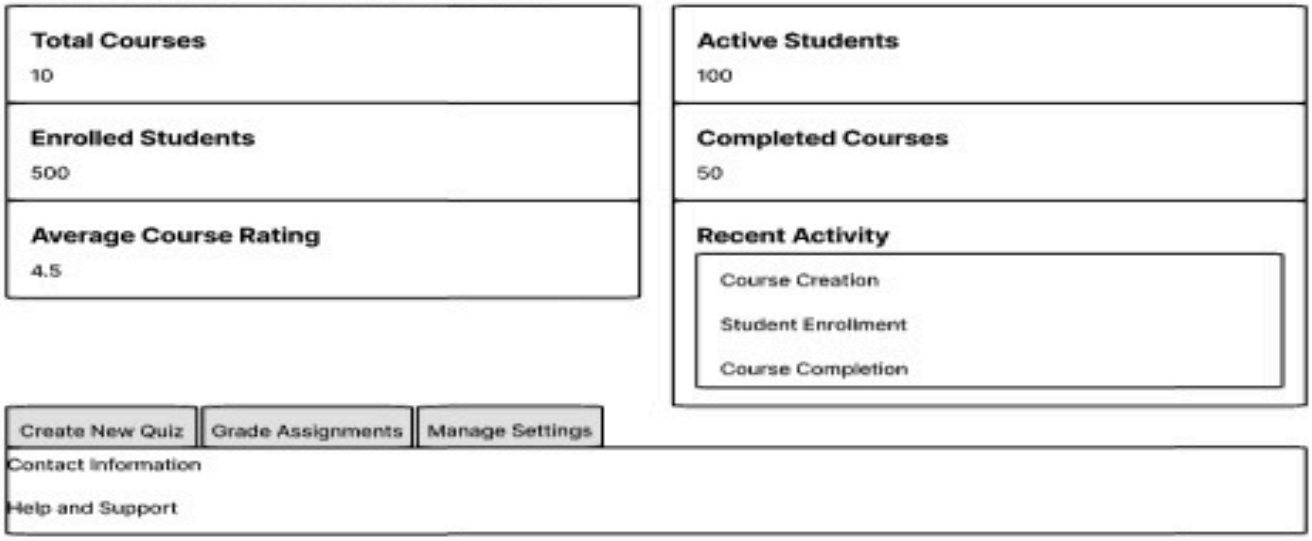


Administrative Tasks Screen

Predictive Support Features Screen



Instructor's Dashboard



Grading Tool Screen

Assignment 1

Assignment 2

Assignment 3

Interactive Quiz Creation Screen

Quiz Title

Quiz Description

Quiz Questions

Submit

Personalized Feedback Screen

Personalized feedback for each student

Plagiarism Detection Screen

Student Assignment 1

Student Assignment 2

Student Assignment 3

Virtual Classroom Setup Screen

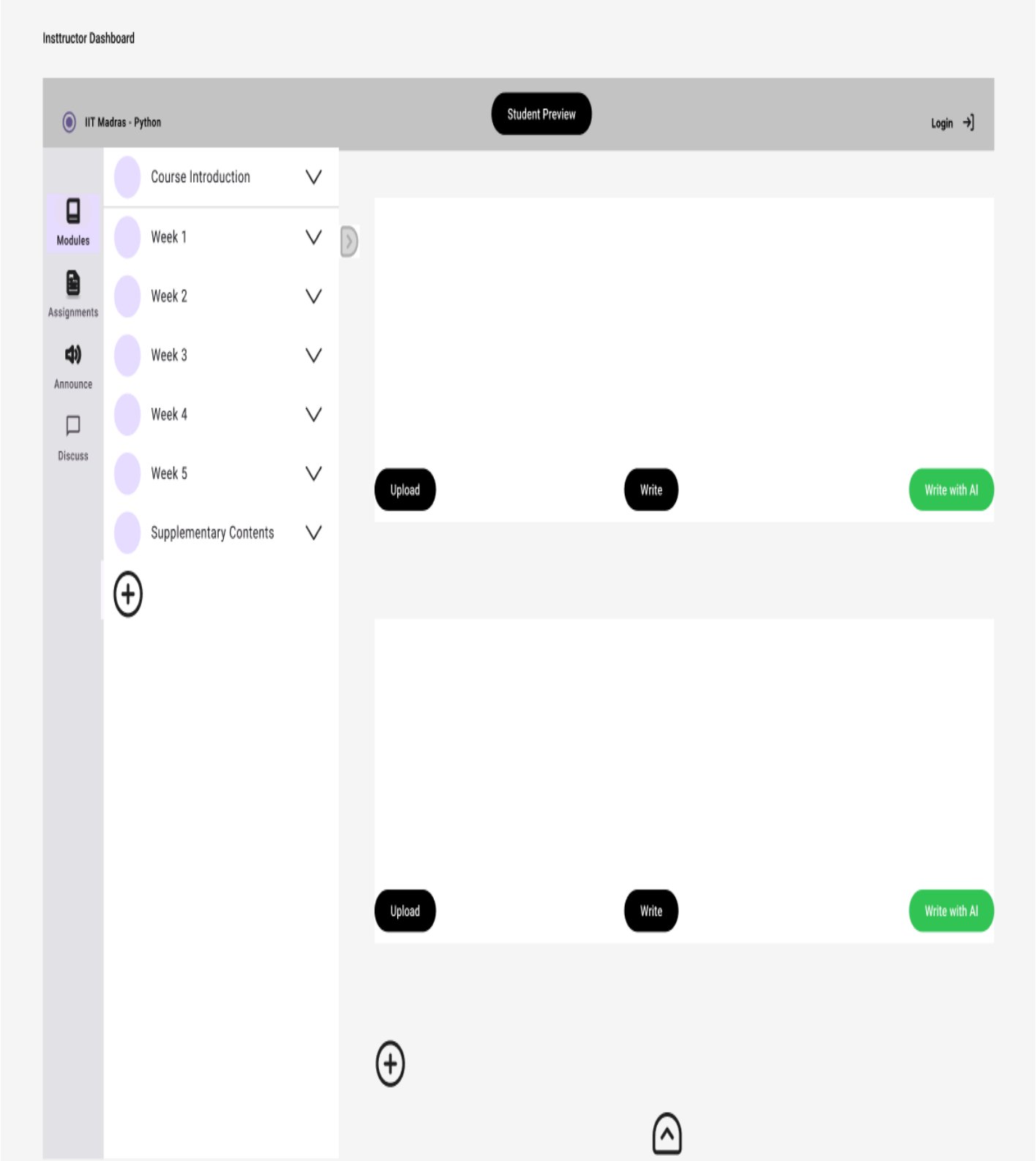
Classroom Title

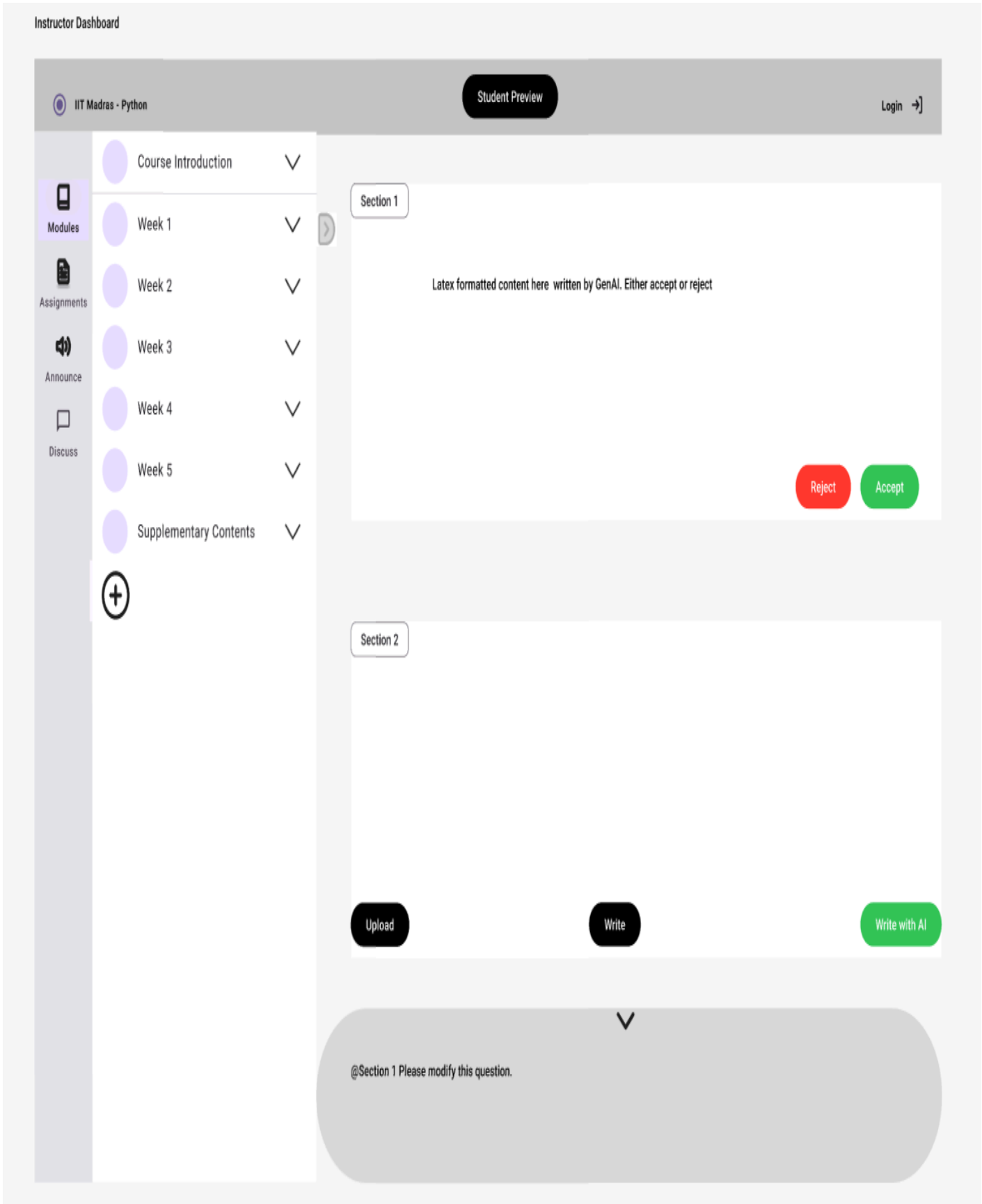
Classroom Description

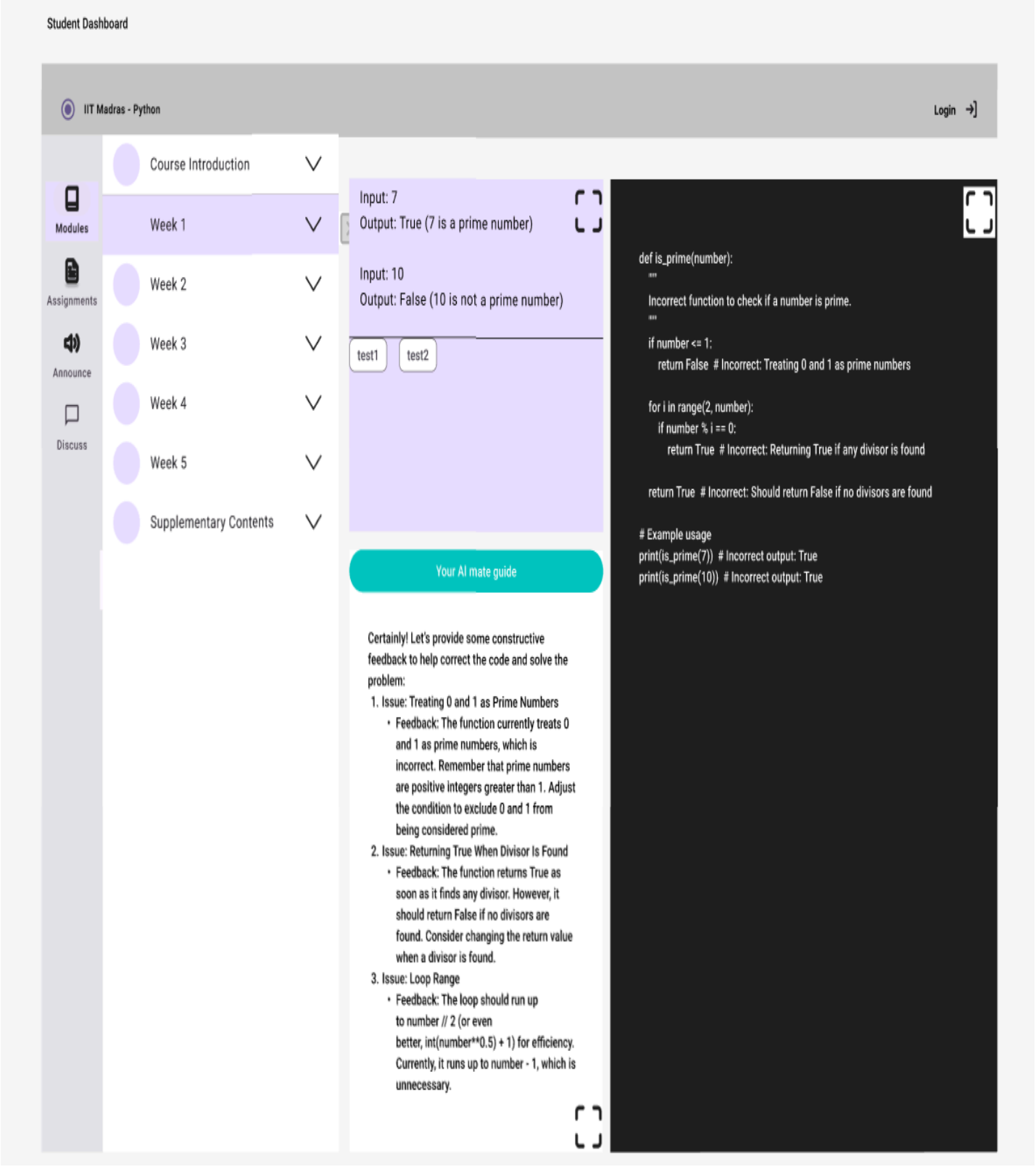
Classroom Settings

Submit

Wireframe - Instructor / Student




Instructor Dashboard
IIT Madras - Python
Student Preview
Login
Course Introduction
Modules
Week 1
Assignments
Week 2
Announce
Week 3
Discuss
Week 4
Week 5
Supplementary Contents
Section 1
Latex formatted content here written by GenAI. Either accept or reject
Reject
Accept
Section 2
Upload
Write
Write with AI
@Section 1 Please modify this question.



Figma Link : [click here]

Milestone 3

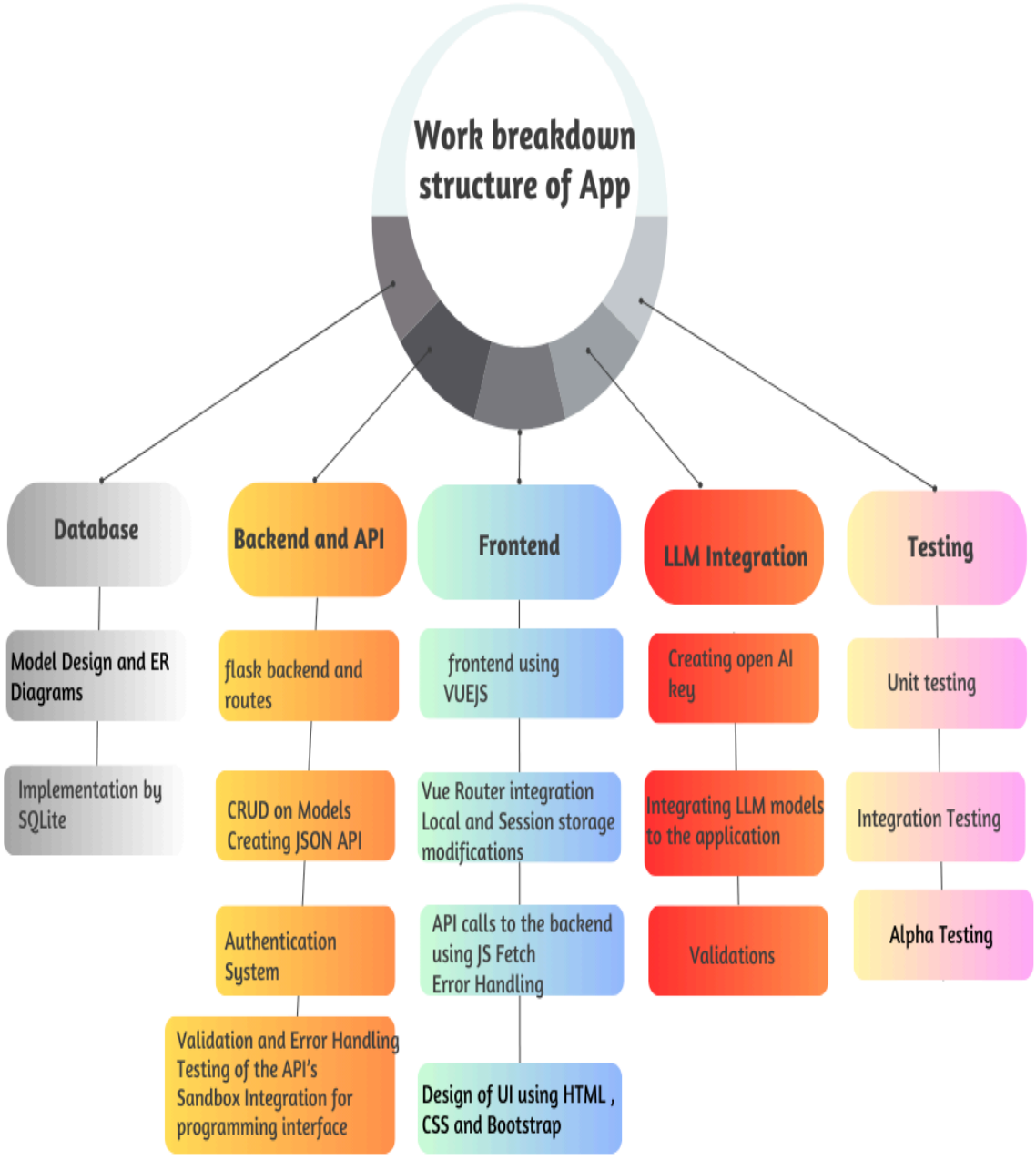
Project Schedule

Task Distribution

Member Name	Component	Sprint	Tasks Assigned
AVIJEET PALIT	• Requirements Gathering - Storyboard • Development - Database	1 3	• Identify User Stories • Create ER Diagrams and Implement DB.
UROOSHA RAHAT	• Wireframes • Development - API	2 4	• Creating wireframes based on user stories. • Working on creating restful API's
SAYAN HRIK	• Task Distribution and Work Breakdown Structure • Component Design • Development - Frontend	3 4	• Distribute project tasks amongst members. • Create and integrate Vue Router and VueJS files to make frontend.
RAMESH KUMAR CHANDRAN	• Project Management / Scrum Master • Testing	4 6	• Create a SPRINT schedule using JIRA • Develop test cases for unit testing.
AYUSH SINGH RANA	• Wireframes • Development - UI Design	2 4	• Creating wireframes based on user stories. • Working on the UI Design using HTML CSS Bootstrap
BODHISATWA BHATTACHARYA	• Learner Journey Maps • Development - Backend	1 4	• Creating learning Journey maps based on use case scenarios. • Creating a flask app for backend.
SACHIN KUMAR	• Project Scheduling • Development - LLM Integration	3 5	• Creating and implementing project schedules on Jira • Integrating LLM's with the APP.
PREETAM MUKHERJEE	Testing Development -	6	• Testing the application (Unit Testing, Integration Testing etc.)

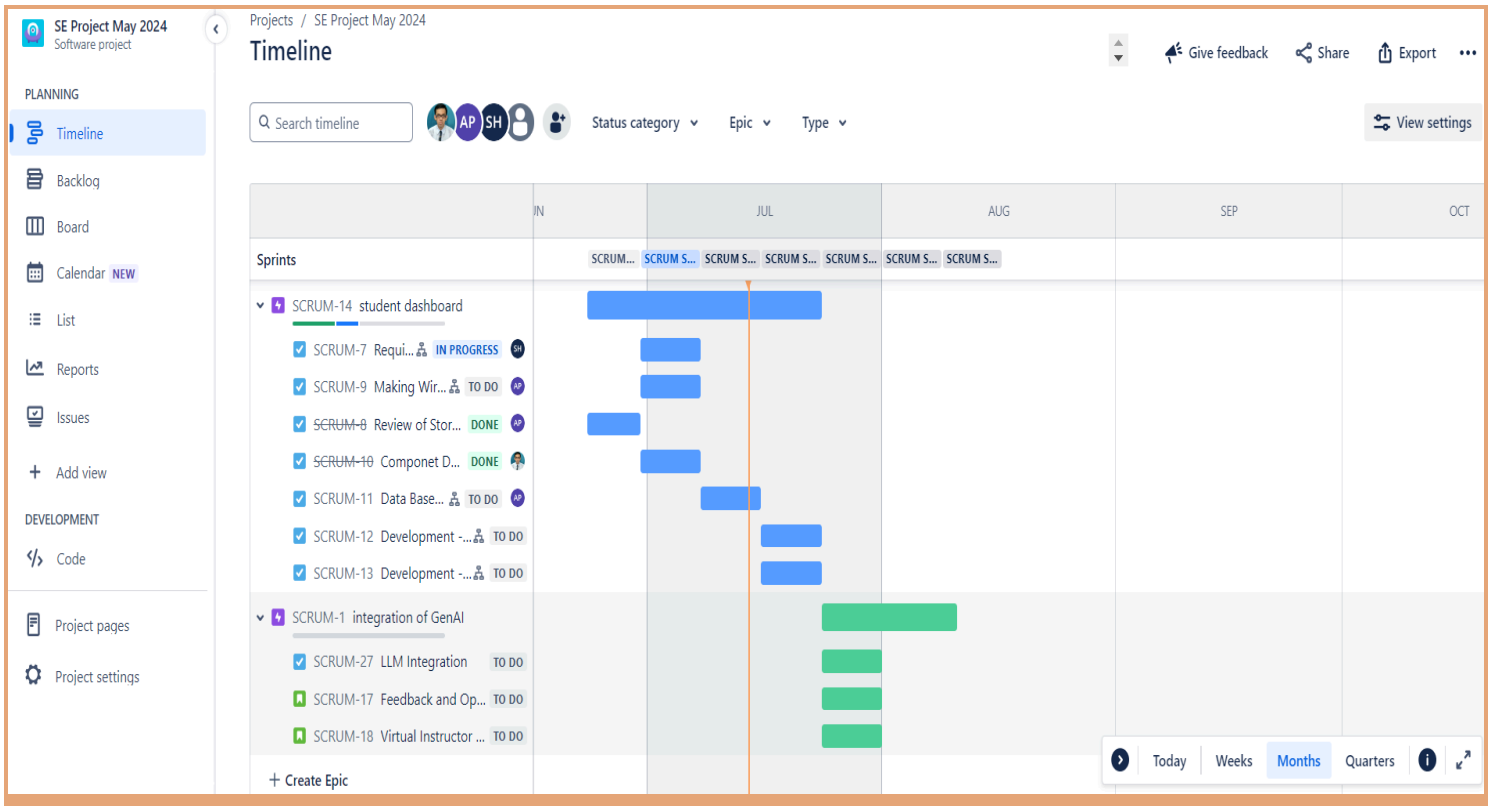
	Authentication	4	• Adding authentication features to the app.

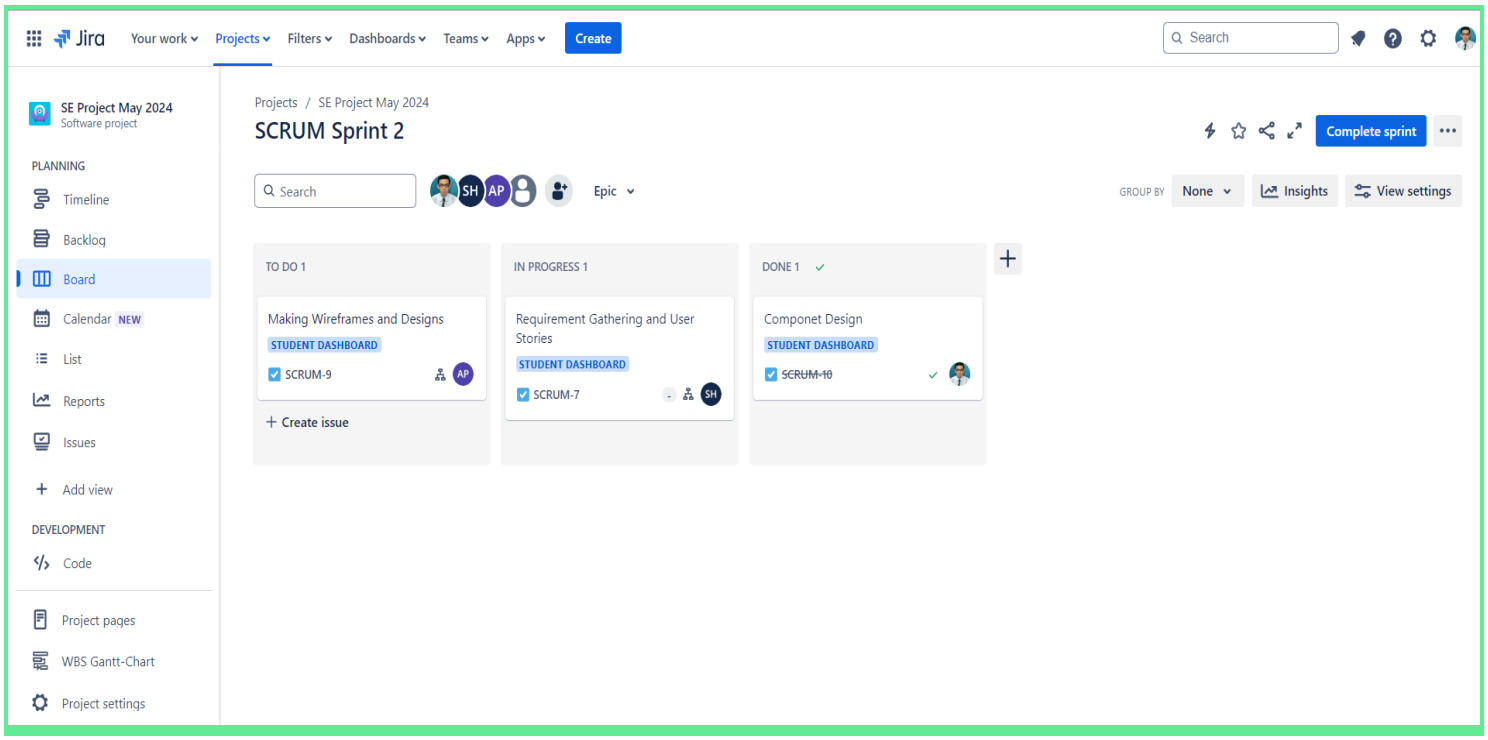
Work Breakdown Structure



Gantt Chart and Scrum Board

Jira Link : [click here]





Components of the Application

Student Components

Authentication (For all Users)

- A login and register form with fields like Email, password for student , instructor and admin login.
- JWT tokens will be used for authentication and RBAC will be used for authorisation.
- Post login each user will be redirected to their specific dashboard.
- A user model will be created to specify roles of each user as Student , Instructor and Admin

Courses Page

- A course page will show the courses the student is taking in the current term.
- Assignment marks for each course will also be shown in the respective course box.

A Course

- Each course will contain its week's modules.
- Each module will contain lectures , notes and assignments.

- Assignments will include theory and programming assignments.

Programming Assignments

- Programming assignments will contain a text editor where students can write code.
- It will have an additional help button along with a submit button which will provide a detailed description of the code written and suggest better / alternative code using Gen AI.

Virtual Instructor

- There will be a separate virtual instructor section in the portal.
- Students can enter a prompt as a doubt and the virtual GEN AI instructor will instantly give detailed explanations.

Resource and Career Planner

Instructor Components

Content Planner

- This section will have a GEN AI planner template where the instructor will enter the course outline and topics and he will get an extensive list of resources for the course materials.
- He will also receive a list of questions he can use for assignments in the course.
- The difficulty level of the questions can be adjusted by the instructor using Gen AI.

Instructor Dashboard

- The instructor will have an extensive dashboard where he can course statistics.
- The stats will include total students enrolled , average marks in assignments and also the total modules currently in the course.

Update Course

- The instructor will be able to update course materials using Gen AI recommendations.
- They will be able to add modules , delete and edit them (CRUD Operations). They will also be able to do the same operations on assignments and notes.

Admin Components

Authentication

- A login and register form with fields like Email, password for admin login.
- JWT tokens will be used for authentication and RBAC will be used for authorisation.
- Post login admin will be redirected to his/her specific dashboard.
- A user model will be created to specify roles of each user as Student , Instructor and Admin.

Add course

- Admin will be able to add the specific course so that the instructor can upload the course contents.
- Admin will be able to perform crud operations on the course details.

Student details

- Admin will be able to add and monitor the details of all the students and courses allotted to them.
- Admin can also revoke the access of the particular course from the student dashboard.
- Admin can perform crud operations on the students details.

Application Statistics

- Admin will be able to see the assignment status of the students , total students enrolled , average CGPA etc using statistical charts.

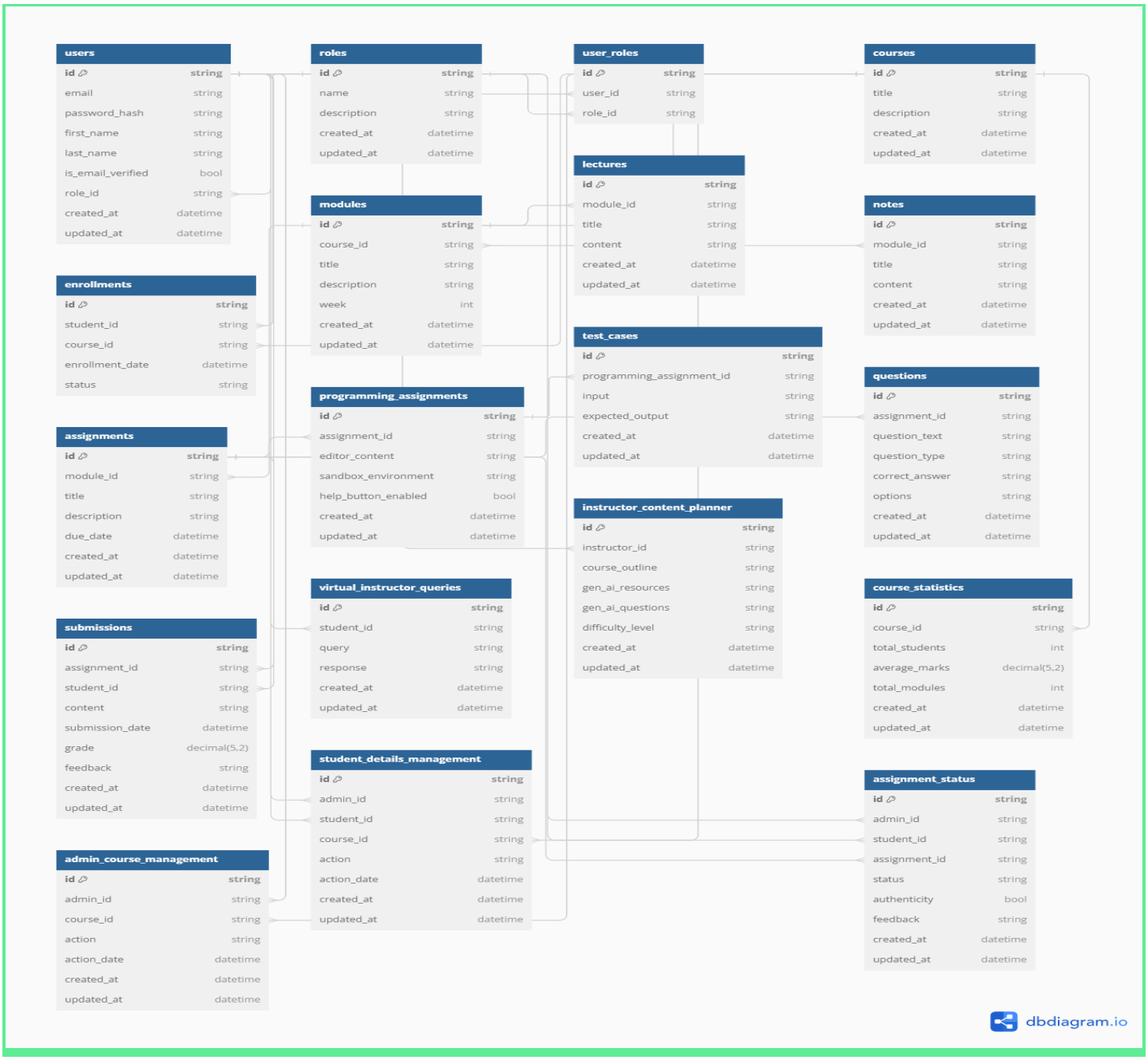
- Admin can also track the legibility and authenticity of the submitted assignments and can take help of GenAI to provide the feedback.

Large Language Models

- LLMs will be used to add Gen AI features to the application.

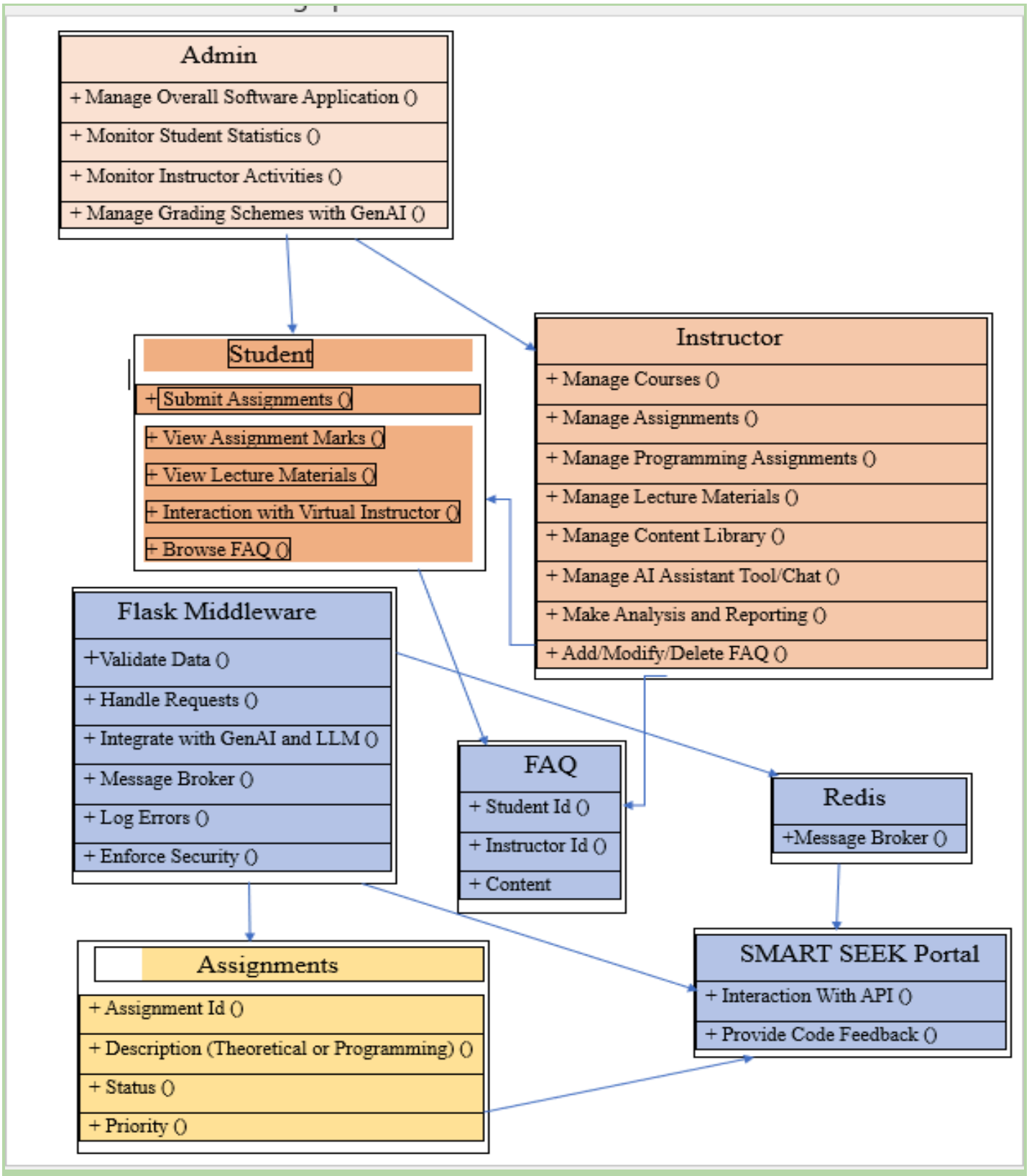
- Gemini's existing models will be integrated using API calls from the backend.

Database Structure and Class Diagrams



Link : https://dbdiagram.io/d/se-project-66936d509939893daedd79ce

Class Diagram


Admin
+ Manage Overall Software Application ()
+ Monitor Student Statistics ()
+ Monitor Instructor Activities ()
+ Manage Grading Schemes with GenAI ()
Student
+ Submit Assignments ()
+ View Assignment Marks ()
+ View Lecture Materials ()
+ Interaction with Virtual Instructor ()
+ Browse FAQ ()
Instructor
+ Manage Courses ()
+ Manage Assignments ()
+ Manage Programming Assignments ()
+ Manage Lecture Materials ()
+ Manage Content Library ()
+ Manage AI Assistant Tool/Chat ()
+ Make Analysis and Reporting ()
+ Add/Modify/Delete FAQ ()
Flask Middleware
+Validate Data ()
+ Handle Requests ()
+ Integrate with GenAI and LLM ()
+ Message Broker ()
+ Log Errors ()
+ Enforce Security ()
FAQ
+ Student Id ()
+ Instructor Id ()
+ Content
Redis
+Message Broker ()
Assignments
+ Assignment Id ()
+ Description (Theoretical or Programming) ()
+ Status ()
+ Priority ()
SMART SEEK Portal
+ Interaction With API ()
+ Provide Code Feedback ()

Minutes of the Meetings (MOM) of few Scrum Meetings

Meeting - 1

Date: July 14, 2024

Time: 1:00 PM - 2:45 PM

Attendees:

- Uroosha
- Avijeet
- Sayan
- Preetam
- Sachin
- Bodhisatwa
- Ayush
- Ramesh

Agenda:

1. **Project Schedule**

   Discuss overall project schedule based on user stories from previous milestones.

   Schedule sprints, iterations, and scrum meetings.

2. **Project Scheduling Tools**

   Selection and use of tools like Trello, Jira for project management.

3. **Design of Components**

   Description of system components based on previous user stories.

4. **Software Design**

   Creation of basic class diagrams for the proposed system.

5. **Scrum Meetings**

   Documenting the minutes of previous scrum meetings.

Minutes:

1. Project Schedule

Sprint Schedule:

Sprint 1: 10/6/2024	18/06/2024: Requirement Gathering and User Stories
Sprint 2: 19/06/2024	25/06/2024: Review of Stories and Storyboard Drafting
Sprint 3: 26/06/2024	30/06/2024: Making Wireframes and Designs
Sprint 4: 01/07/2024	05/07/2024: Component Design
Sprint 5: 03/07/2024	05/07/2024: Database Design and ER Diagrams
Sprint 6: 06/07/2024	11/07/2024: Development - Backend and APIs
Sprint 7: 12/07/2024	17/07/2024: Development - Frontend and API Connections
Sprint 8: 17/07/2024	18/07/2024: Development - LLM Integration
Sprint 9: 18/07/2024	20/07/2024: Development - Frontend and Backend Integration
Sprint 10: 21/07/2024	26/07/2024: Development - Additional Features
Sprint 11: 27/07/2024	05/08/2024: Testing of the Application
Sprint 12: 06/08/2024	12/08/2024: Deployment and Final Report Compilation

Scrum Meetings: Scheduled twice a week on Mondays and Thursdays at 5 PM.

2. Project Scheduling Tools

Trello Board: Used for task assignment and tracking progress.

Jira: Used for managing sprints, issues, and detailed tracking of user stories and tasks.

3. Design of Components

Student:

- **Authentication**: Login and register forms with JWT tokens for authentication and RBAC for authorization.
- **My Courses**: Display courses taken by the student in the current term along with assignment marks.
- **Course Details**: Each course will contain modules, lectures, notes, and assignments.
- **Programming Assignments**: Includes a text editor for coding with Gen AI support for code suggestions.

Instructor:

Content Planner: Gen AI planner for course materials and assignments.

Instructor Dashboard: Displays course statistics and management options.

Update Course: Allows CRUD operations on course materials and assignments.

Admin:

- **Authentication:** Admin login with JWT tokens and RBAC.
- **Add Course:** CRUD operations on course details.
- **Student Details:** Manage student course enrollments and payments.
- **Statistics:** Track assignment status and feedback with Gen AI support.

4. Software Design

Class Diagrams:

- Created basic class diagrams to represent system architecture and interactions among different components.

Meeting 2

Date: July 10, 2024

Time: 5:00 PM 5:30 PM

Attendees:

Uroosha

Avijeet

Sayan

Preetam

Bodhisatwa

Sachin

Ayush

Ramesh

Agenda:

1. **Review of User Stories**

- Confirm completion and accuracy of user stories.

2. **Wireframe and Design Review**
   - Evaluate the wireframes and design drafts.

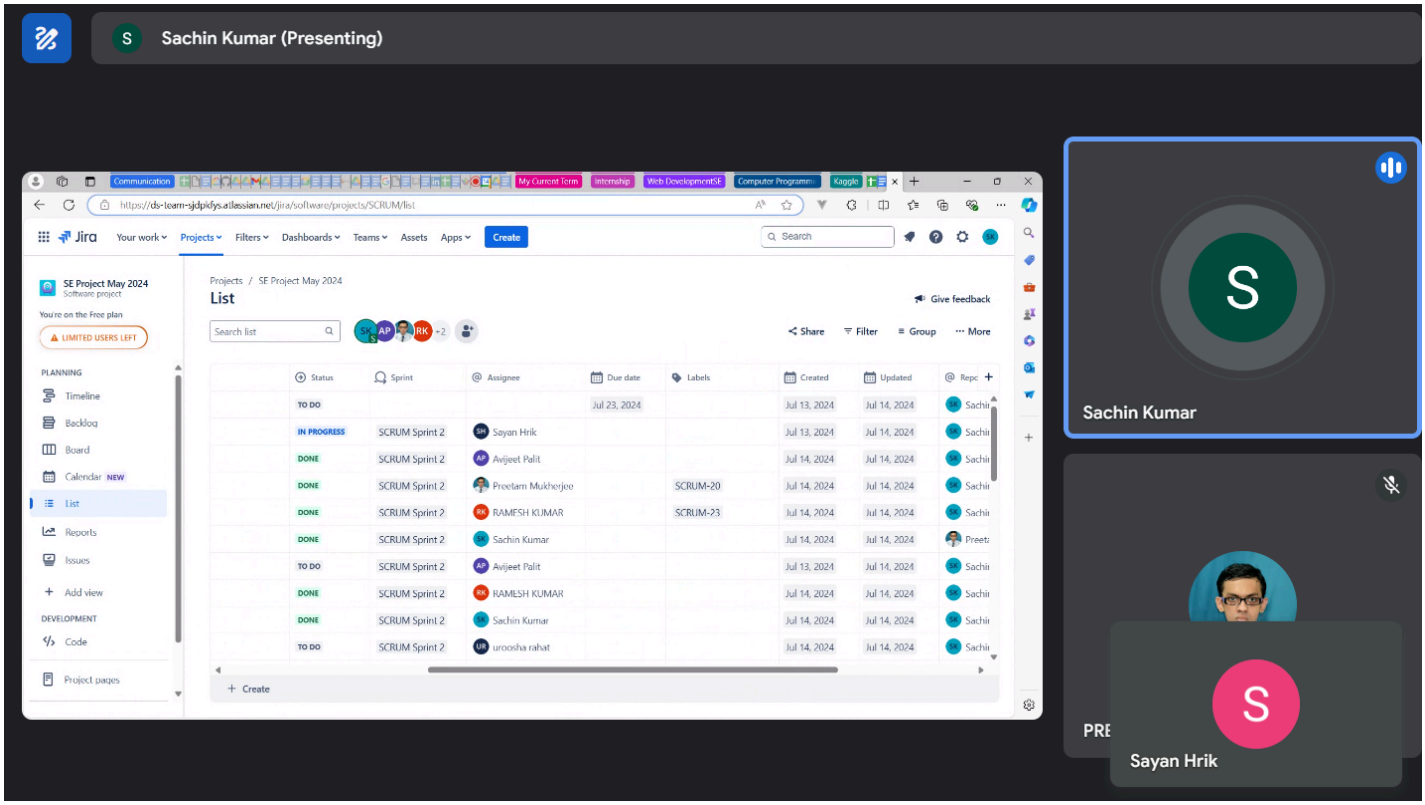
3. **Component Design Discussion**
   - Finalize the design of system components.

Minutes:

- Reviewed and confirmed all user stories are accurate and complete.
- Evaluated wireframes and made minor adjustments based on feedback.
- Finalized component design, ensuring alignment with user stories and system requirements.

Action Items:

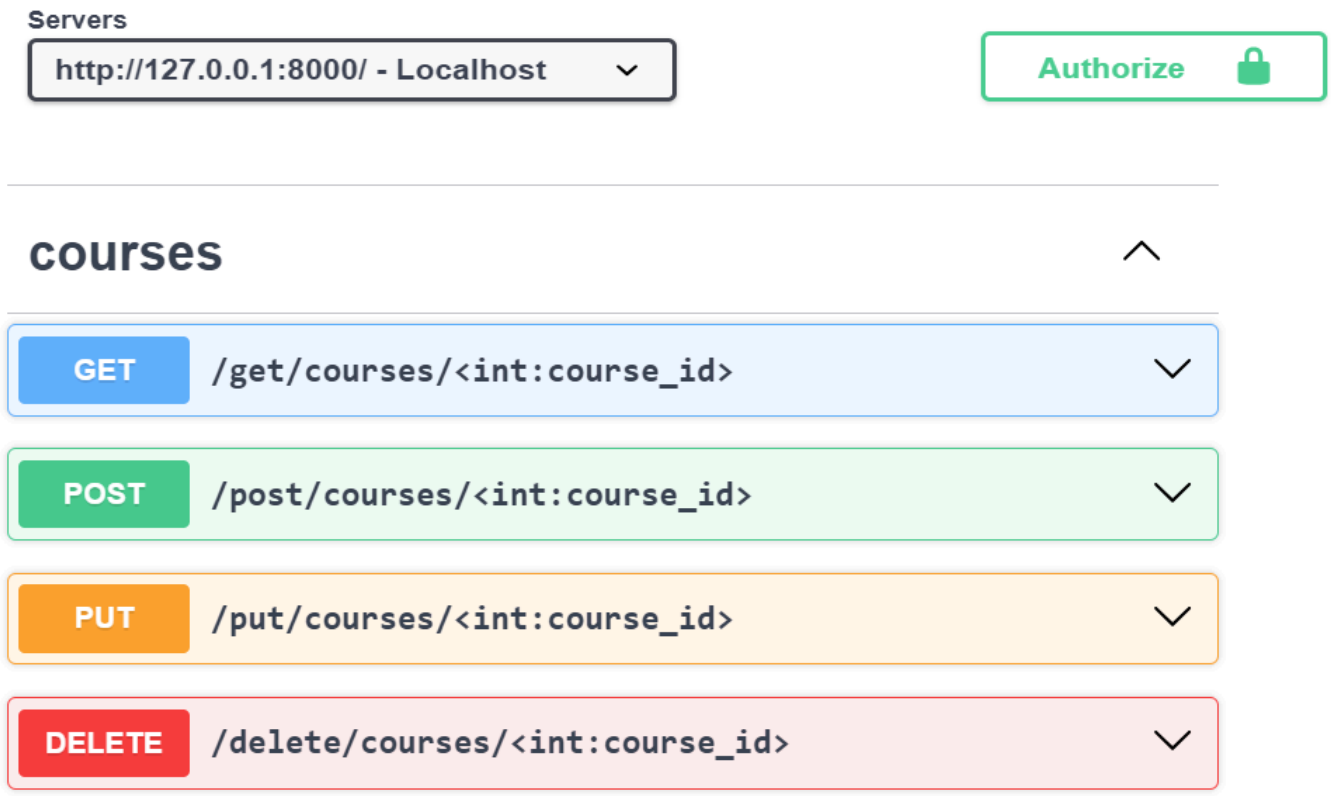
- Avijeet to finalize the database design and ER diagrams by the next meeting.
- Sachin will set up the backend structure and Preetam will do the authentication system.
- Sayan to start integrating Vue Router for frontend.



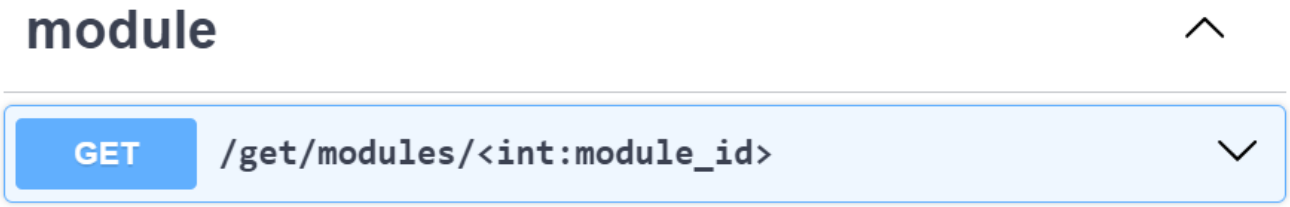
Milestone 4

API Endpoints:

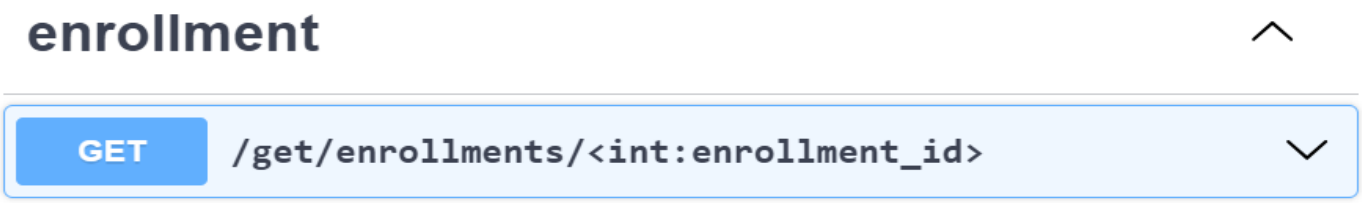
Courses:



Modules:

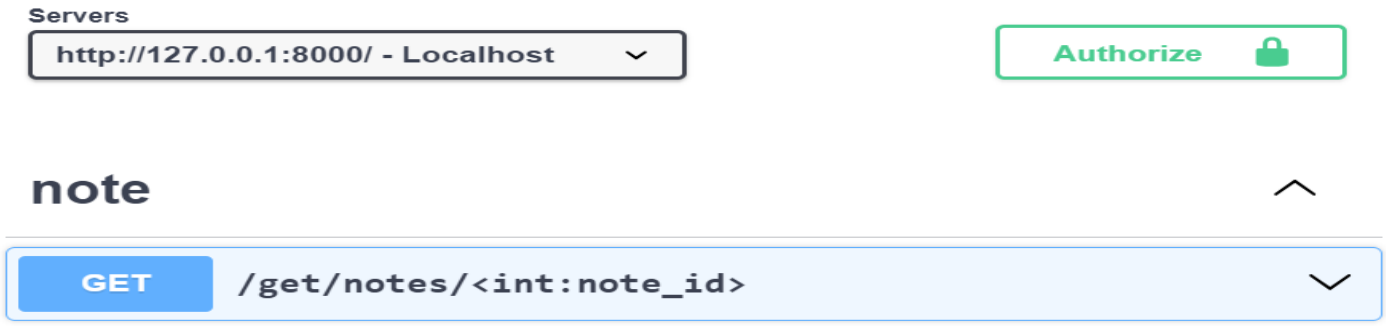


Enrollments:

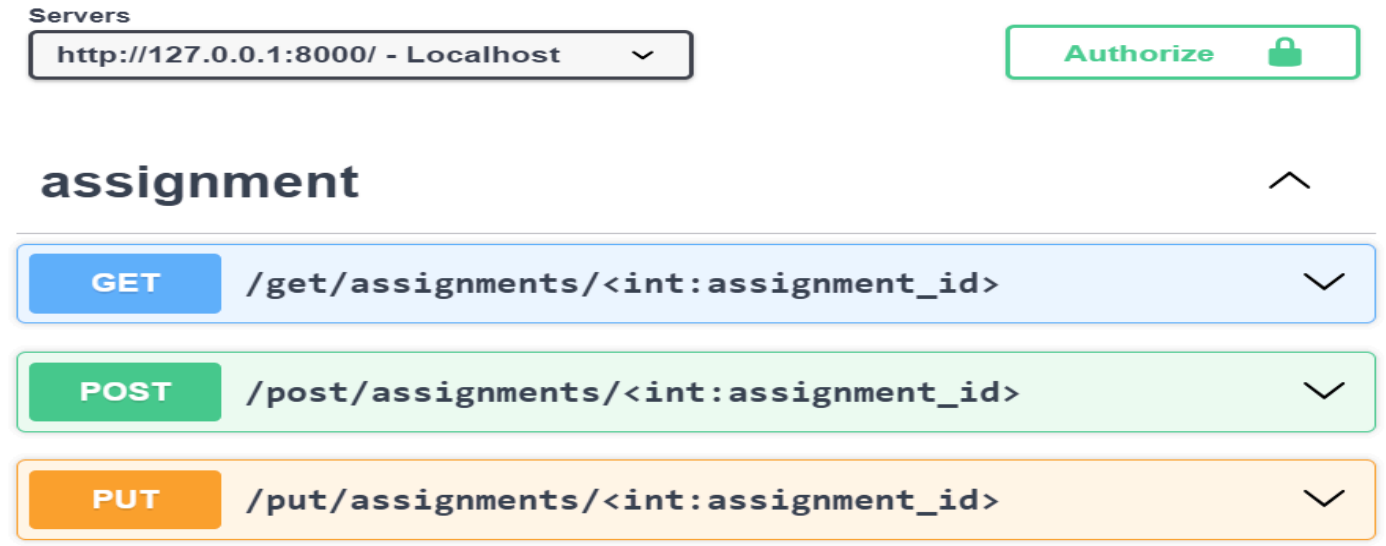


Lectures:

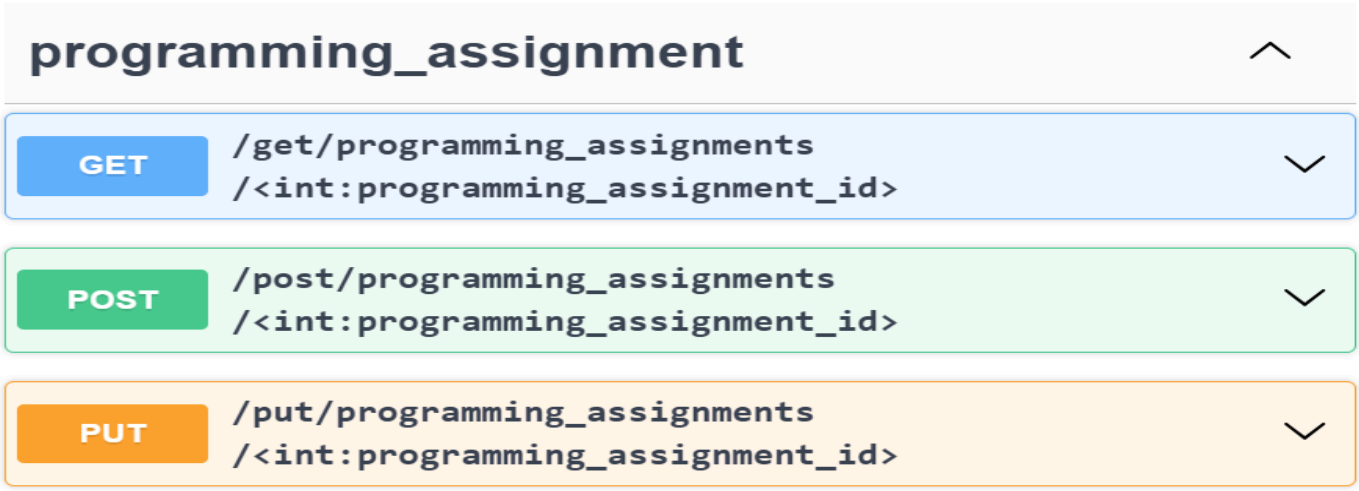
Notes:



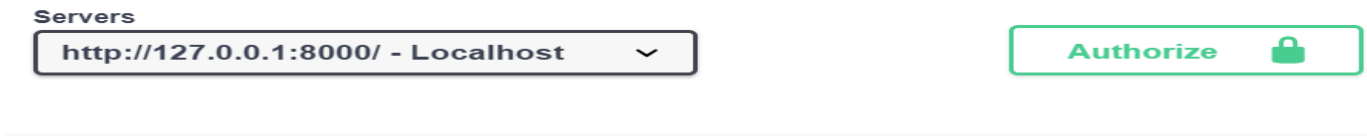
Assignments:



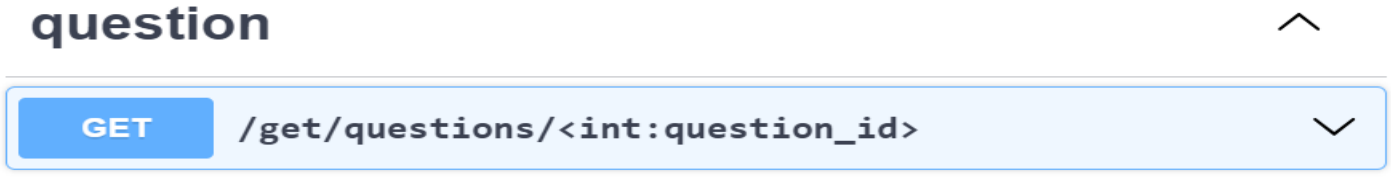
Programming Assignments:



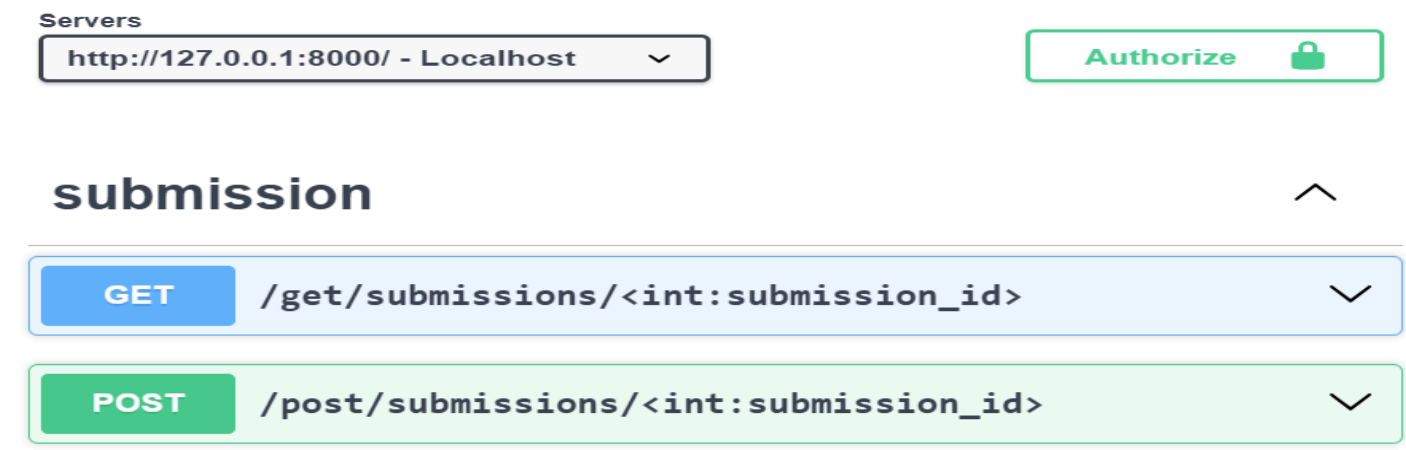
Test Cases:



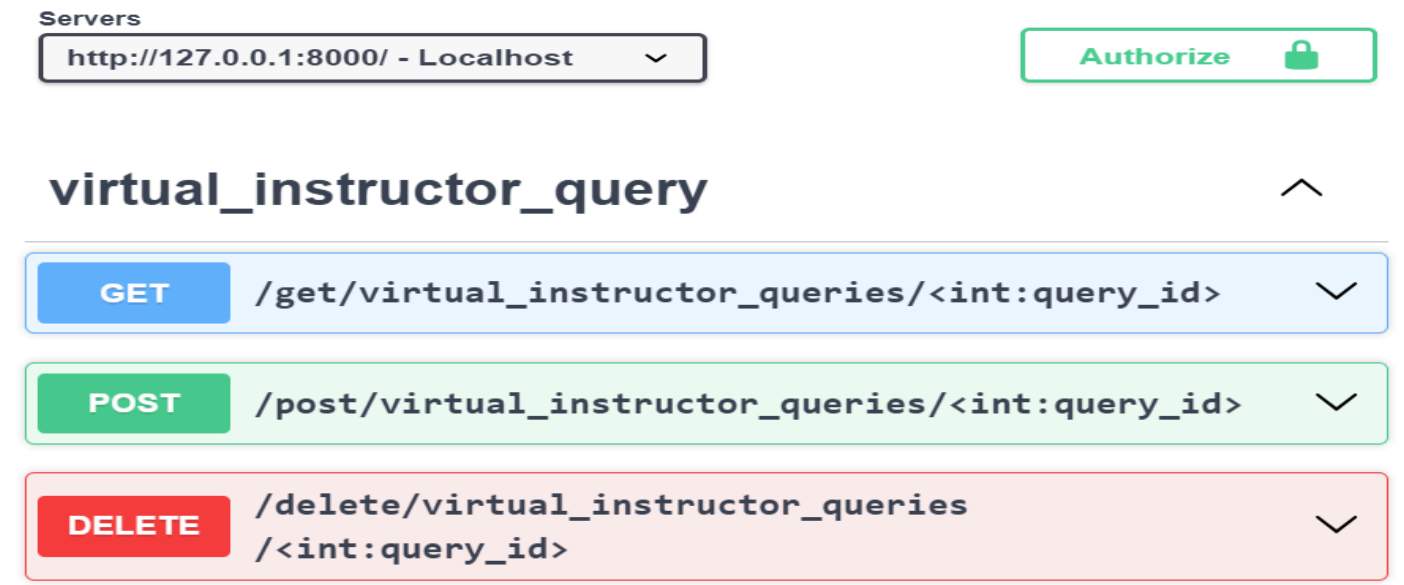
Questions:



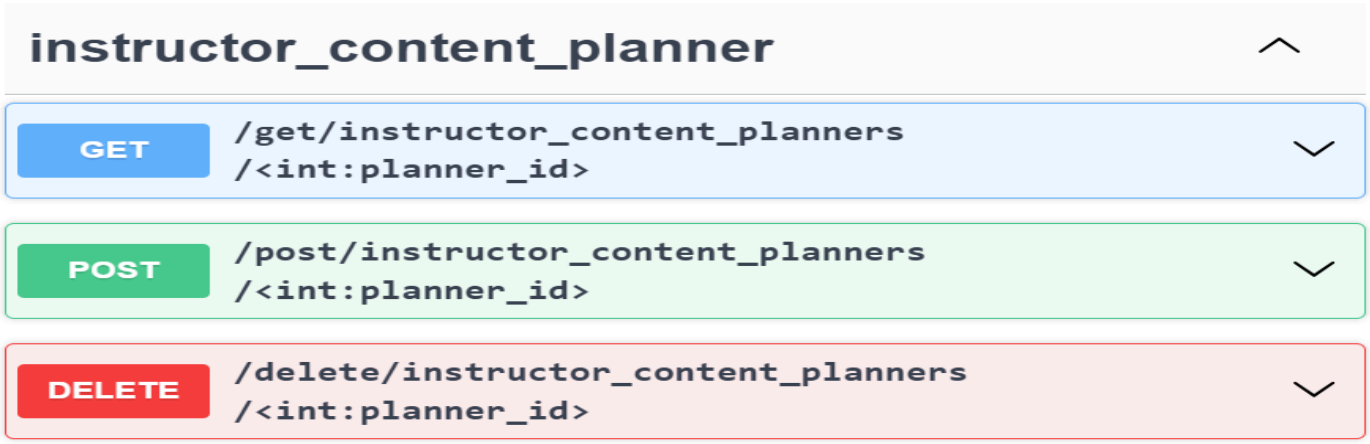
Submissions:



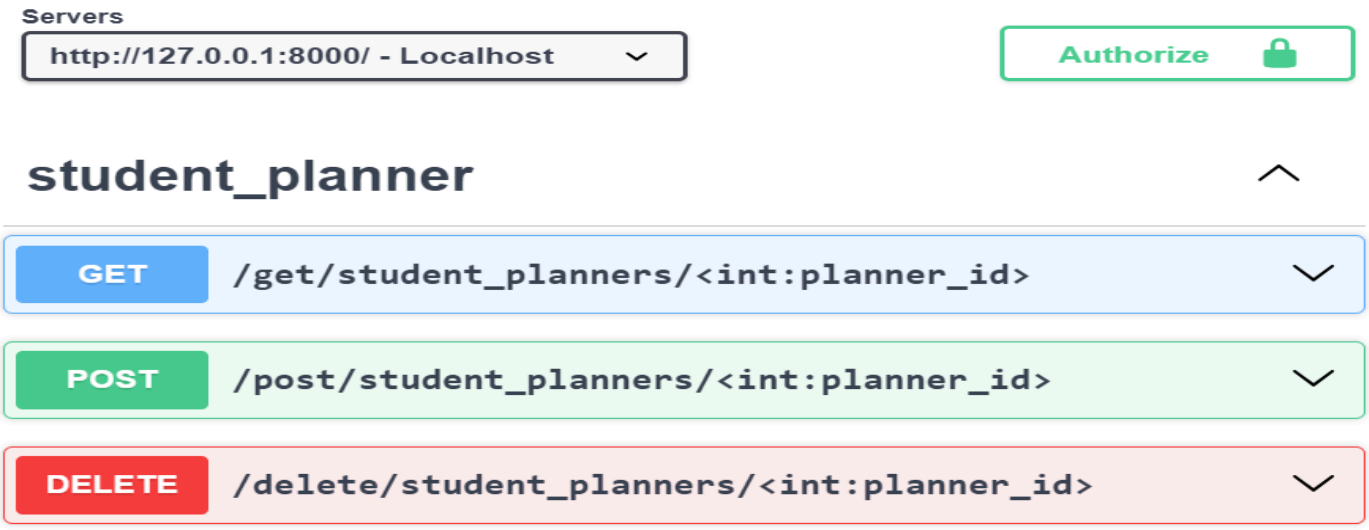
Virtual Instructor Queries:



Instructor Content Planners:

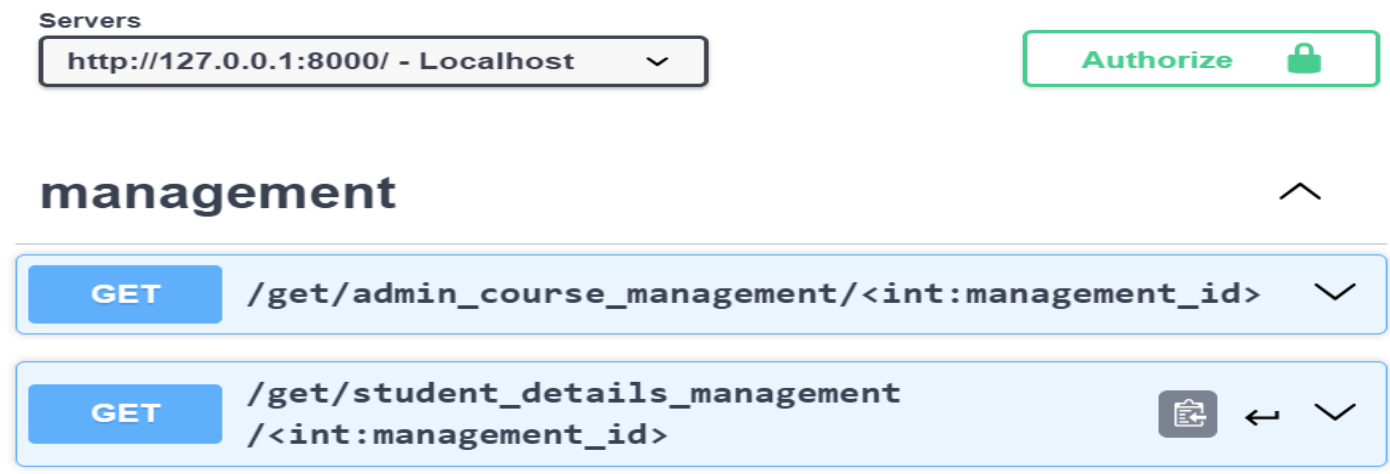


Student Planners:

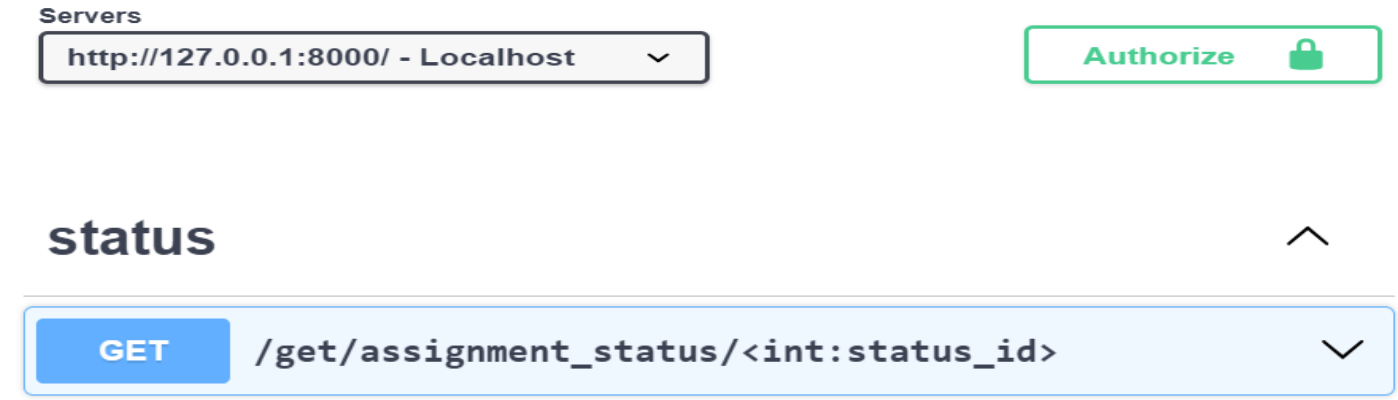


Course Statistics:

Management:


Servers
http://127.0.0.1:8000/ - Localhost
Authorize
management
GET
/get/admin_course_management/<int:management_id>
GET
/get/student_details_management
/<int:management_id>

Status:


Servers
http://127.0.0.1:8000/ - Localhost
Authorize
status
GET
/get/assignment_status/<int:status_id>

Milestone 5

1. Test Cases and Descriptions for Components:

1.1 Student Component Testing:

Test Cases for Authentication

Test Case: Successful Registration of Student

API being tested: /student (POST method)

Inputs:

Student_email: “21f2000143@ds.study.iitm.ac.in”

Expected Output:

Status Code: 200 (OK)

JSON:


```
{
  "success": true,
  "token":
"eyJ2ZXIiOiI1IiwidWlkIjoiZGI5NGI5NTI5MGM4NGQ4NzkzOWVmZDJlMWU4OWM5NjQi
LCJzaWQiOjAsImV4cCI6MH0.ZrYPdA.4VUwnQOFuoyeC4eDW1aLDsbWFAUw",
  "user_data": {
    "email": "21f2000143@ds.study.iitm.ac.in",
    "first_name": "Sachin",
    "last_name": "Kumar"
  }
}
```


Actual Output:

Status Code: 200 (OK)

JSON:


```
{
  "success": true,
  "token":
"eyJ2ZXIiOiI1IiwidWlkIjoiZGI5NGI5NTI5MGM4NGQ4NzkzOWVmZDJlMWU4OWM5NjQi
LCJzaWQiOjAsImV4cCI6MH0.ZrYPdA.4VUwnQOFuoyeC4eDW1aLDsbWFAUw",
  "user_data": {
```

```
    "email": "21f2000143@ds.study.iitm.ac.in",
    "first_name": "Sachin",
    "last_name": "Kumar"
  }
}
```


Result: Success

Test Case: Bad Request - Invalid Input Data
API being tested: /student/<int:student_id> (POST method)
Inputs:

Expected Output:
Status Code: 400
JSON:

```
{
  "error": "Invalid email",
  "success": false
}
```


Actual Output:
Status Code: 400
JSON:

```
{
  "error": "Invalid email",
  "success": false
}
```

Result: Success

Test Cases for Notes

Test Case: Creating / adding notes to a module
API being tested: /notes (POST method)
Inputs:
module_id: “1”
title: “Programming in Python”

content: “This lecture covers the basics of Python programming. ”

Expected Output:
Status Code: 201
JSON:

```
{
   "id": 1,
   "module_id": "CS101",
   "title": "Introduction to Python",
   "content": "This lecture covers the basics of Python programming.",
}
```


Actual Output:
Status Code: 201 (Created)
JSON:

```
{
   "id": 1,
   "module_id": "CS101",
   "title": "Introduction to Python",
   "content": "This lecture covers the basics of Python programming.",
}
```


Result: Success

Test Case: Bad Request for Note creation

API being tested: /student (POST method)
Inputs:
module_id: (missing)
title: “Python Notes”
content: “This note has missing information.”

Expected Output:
Status Code: 400 **(**BAD Request**)**
JSON:

```
{
```

"message": "Invalid input data"
}

Actual Output:
Status Code: 400
JSON:
{
"message": "Invalid input data"
}

Result: Success

Test Case: Deleting Notes
API being tested:/notes/<int:note_id> (DELETE method)
Inputs:
note_id: 1

Expected Output:
Status Code: 200 (OK)
JSON:
{'message': 'Note deleted'}

Actual Output:
Status Code: 200 (OK)
JSON:
{'message': 'Note deleted'}

Result: Success

Test Case: BAD Request (Deleting a Note that does not exist)
API being tested:/notes/<int:note_id> (DELETE method)
Inputs:
note_id: 100 (invalid note id)
Expected Output:
Status Code: 404 (Not Found)

JSON:

```
{
   "message": "Note not found"
}
```

Actual Output:
Status Code: 404 (Not Found)
JSON:

```
{
   "message": "Note not found"
}
```

Result:Success

Test Case For Courses

Test Cases for Course Management

Test Case: Get Details of a Course
API being tested: `/courses/<int:course_id>` (GET method)
Inputs:

`course_id`:1

Expected Output:

Status Code: 200 (OK)

JSON:

```
{
    "course_id": 1,
    "course_name": "Programming in Python",
    "description": "A foundational course in python for data science.",
    "modules": [
        {
            "module_id": "CS101",
            "module_name": "Introduction to Python",
```

```
            "description": "Basics of Python programming."
        },
           ]
}
```


Actual Output:

Status Code: 200 (OK)

JSON:


```
  {

    "course_id": 1,
    "course_name": "Programming in Python",
    "description": "A foundational course in python for data science.",
    "modules": [
        {
            "module_id": "CS101",
            "module_name": "Introduction to Python",
            "description": "Basics of Python programming."
        },
           ]
}
```


Result: Success

Test Case: Bad Request - Course Not Found
API being tested: `/courses/<int:course_id>` (GET method)
Inputs:

`course_id`: 100

Expected Output:

Status Code: 404 (Not Found)

JSON:


```
{
```

```
    "message": "Course not found"
}
```

Actual Output:

Status Code: 404 (Not Found)

JSON:

```
{
    "message": "Course not found"
}
```

Result: Success

Test Case: Add a Course
API being tested: `/courses` (POST method)
Inputs:

```
{
    "course_name": "Programming in Python",
    "description": "An in-depth course on python for data science"
}
```

Expected Output:

Status Code: 201 (Created)

JSON:

```
{
    "course_id": 1,
    "course_name": "Programming in Python",
    "description": "An in-depth course on python for data science."
}
```

Actual Output:

Status Code: 201 (Created)

JSON:

```
{
    "course_id": 1,
    "course_name": "Programming in Python",
    "description": "An in-depth course on python for data science."
}
```

Result: Success

Test Case: Update a Course
API being tested: `/courses/<int:course_id>` **(PUT method)**
Inputs:

```
{
    "course_name": "Python - 2",
    "description": "Changed the course name"
}
```

Expected Output:

Status Code: 200 (OK)

JSON:

```
{
    "course_id": 1,
    "course_name": "Python 2",
    "description": "Changed the course name"
}
```

Actual Output:

Status Code: 200 (OK)

JSON:

```
{
    "course_id": 1,
    "course_name": "Python 2",
    "description": "Changed the course name"
}
```

Result: Success

Test Cases for Modules

Test Case: Add a Module to a Course

API being tested: `/courses/<int:course_id>/modules` **(POST method)**

Inputs:

```
{
    "module_name": "OOPS",
    "description": "Basics of Object Oriented Programming.",
    "course_id": 102
}
```

Expected Output:

Status Code: 201 (Created)

JSON:

```
{
    "module_id": "1",
    "module_name": "OOPS",
    "description": "Basics of object oriented programming.",
    "course_id": 1
}
```

Actual Output:

Status Code: 201 (Created)

JSON:

```
{
    "module_id": "1",
    "module_name": "OOPS",
    "description": "Basics of object oriented programming.",
    "course_id": 1
```

```
}
```

Result: Success

Test Case: Delete a Module from a Course
API being tested: `/courses/<int:course_id>/modules/<int:module_id>` **(DELETE method)**
Inputs:

`module_id`**:** 1

Expected Output:

Status Code: 200 (OK)

JSON:

```
{
    "message": "Module deleted successfully"
}
```

Actual Output:

Status Code: 200 (OK)

JSON:

```
{
    "message": "Module deleted successfully"
}
```

Result: Success

Test Case: Bad Request - Module Not Found
API being tested: `/courses/<int:course_id>/modules/<int:module_id>` **(DELETE method)**
Inputs:

`module_id`: 100 (invalid module id)

Expected Output:

Status Code: 404 (Not Found)

JSON:

```
{
    "message": "Module not found"
}
```

Actual Output:

Status Code: 404 (Not Found)

JSON:

```
{
    "message": "Module not found"
}
```

Result: Success

Test Cases for Lectures

Test Case: Adding a lecture

API being tested: /lectures (POST method)

Inputs:

module_id: "CS101",
title: "Programming in Python",
content: "https://youtu.be/8ndsDXohLMQ?"

Expected Output:

Status Code: 201 (Created)

JSON:

```
{
   "id": 1,
   "module_id": "CS101",
   "title": "Programming in Python",
   "content": "https://youtu.be/8ndsDXohLMQ?"
}
```

Actual Output:

Status Code: 201 (Created)

JSON:


```
{
   "id": 1,
   "module_id": "CS101",
   "title": "Programming in Python",
   "content": "https://youtu.be/8ndsDXohLMQ?"
}
```


Result: Success

Test Case: Successfully deleting a lecture

API being tested: /lectures/<int:lecture_id> (DELETE method)

Inputs:

Expected Output:

Status Code: 200 (OK)

JSON:

{'message': 'Lecture deleted'}

Actual Output:

Status Code: 200 (OK)

JSON:

{'message': 'Lecture deleted'}

Result: Success

Test Case: Bad Request (Deleting a lecture with lecture id that does not exist)

API being tested:/lectures/<int:lecture_id> (DELETE method)

Inputs:

lecture_id : 100

Expected Output:

Status Code: 404 (Not Found)

JSON:


```
{
   "message": "Lecture not found"
}
```

Actual Output:
Status Code: 404 (Not Found)
JSON:

```
{
  "message": "Lecture not found"
}
```


Result: Success

Test Cases for Assignments

Test Case: Accessing an assignment.
API being tested: /assignments/<int:assignment_id> (GET method)
Inputs:
assignment_id: 1

Expected Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "title": "Sample Assignment",
   "description": "This is a sample assignment",
   "due_date": "2024-08-30T00:00:00",
   "created_at": "2024-08-09T10:00:00",
   "updated_at": "2024-08-09T10:00:00"
}
```


Actual Output:
Status Code: 200
JSON:

```
{
  "id": "1",
  "title": "Sample Assignment",
  "description": "This is a sample assignment",
  "due_date": "2024-08-30T00:00:00",
  "created_at": "2024-08-09T10:00:00",
  "updated_at": "2024-08-09T10:00:00"
```

}

Result: Success

Test Case: Successfully creating a new assignment.
API being tested: /assignments (POST method)
Inputs:
title: "New Assignment",
description: "This is a new assignment"

Expected Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "title": "New Assignment",
   "description": "This is a new assignment",
   "due_date": null,
   "created_at": "2024-08-09T10:15:00",
   "updated_at": "2024-08-09T10:15:00"
}
```


Actual Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "title": "New Assignment",
   "description": "This is a new assignment",
   "due_date": null,
   "created_at": "2024-08-09T10:15:00",
   "updated_at": "2024-08-09T10:15:00"
}
```


Result: Success

Test Case: Successfully updating an assignment.

API being tested: /assignments/<int:assignment_id> (PUT method)

Inputs:

assignment_id:1

title: "Updated Assignment",

description: "This assignment has been updated"

Expected Output:

Status Code: 200

JSON:


```
{
   "id": "1",
   "title": "Updated Assignment",
   "description": "This assignment has been updated",
   "due_date": "2024-08-30T00:00:00",
   "created_at": "2024-08-09T10:00:00",
   "updated_at": "2024-08-09T10:20:00"
}
```


Actual Output:

Status Code: 200

JSON:


```
{
   "id": "1",
   "title": "Updated Assignment",
   "description": "This assignment has been updated",
   "due_date": "2024-08-30T00:00:00",
   "created_at": "2024-08-09T10:00:00",
   "updated_at": "2024-08-09T10:20:00"
}
```


Result: Success

Test Cases for Programming Assignments

Test Case: Successfully getting a programming assignment.

APIbeing tested:
/programming_assignments/<int:programming_assignment_id> (GET method)

Inputs:
programming_assignment_id: 1

Expected Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "assignment_id": "2",
   "editor_content": "print('Hello, World!')",
   "sandbox_environment": "Python 3.8",
   "help_button_enabled": true,
   "created_at": "2024-08-09T10:30:00",
   "updated_at": "2024-08-09T10:30:00"
}
```

Actual Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "assignment_id": "2",
   "editor_content": "print('Hello, World!')",
   "sandbox_environment": "Python 3.8",
   "help_button_enabled": true,
   "created_at": "2024-08-09T10:30:00",
   "updated_at": "2024-08-09T10:30:00"
}
```

Result: Success

Test Case: Successfully creating a programming assignment

API being tested: /programming_assignments (POST method)
Inputs:
assignment_id: 2,
language: "Python"

Expected Output:
Status Code: 201 (Created)
JSON:

```
{
   "id": "2",
   "assignment_id": "2",
   "editor_content": null,
   "sandbox_environment": "Python 3.8",
   "help_button_enabled": false,
   "created_at": "2024-08-09T10:35:00",
   "updated_at": "2024-08-09T10:35:00"
}
```

Actual Output:
Status Code: 201 (Created)
JSON:

```
{
   "id": "2",
   "assignment_id": "2",
   "editor_content": null,
   "sandbox_environment": "Python 3.8",
   "help_button_enabled": false,
   "created_at": "2024-08-09T10:35:00",
   "updated_at": "2024-08-09T10:35:00"
}
```


Result: Success

Test Case: Updating a Programming Assignment.

API being tested:
/programming_assignments/<int:programming_assignment_id> (PUT method)
Inputs:
programming_assignment_id: 1,
assignment_id: 2,
language: "Python 3"

Expected Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "assignment_id": "2",
   "editor_content": "print('Updated Content')",
   "sandbox_environment": "Python 3.9",
   "help_button_enabled": true,
   "created_at": "2024-08-09T10:30:00",
   "updated_at": "2024-08-09T10:40:00"
}
```


Actual Output:
Status Code: 200
JSON:

```
{
   "id": "1",
   "assignment_id": "2",
   "editor_content": "print('Updated Content')",
   "sandbox_environment": "Python 3.9",
   "help_button_enabled": true,
   "created_at": "2024-08-09T10:30:00",
   "updated_at": "2024-08-09T10:40:00"
}
```

Result: Success

Test Cases for test cases of Programming Assignments and Submissions

Test Case: Successfully getting a Test Case.

API being tested:/test_cases/<int:test_case_id> (GET method)

Inputs:

test_case_id: 1

Expected Output:

Status Code: 200

JSON:


```
{
 "id": "1",
 "assignment_id": "1",
 "input": "4",
 "expected_output": "16",
 "created_at": "2024-08-09T11:00:00",
 "updated_at": "2024-08-09T11:00:00"
}
```


Actual Output:

Status Code: 200

JSON:


```
{
 "id": "1",
 "assignment_id": "1",
 "input": "4",
 "expected_output": "16",
 "created_at": "2024-08-09T11:00:00",
 "updated_at": "2024-08-09T11:00:00"
}
```

Result: Success

Test Case: Successfully creating a Test Case.

API being tested: /test_cases (POST method)

Inputs:


```
{
 "assignment_id": 1001,
 "input": "5",
 "output": "25"
}
```


Expected Output:

Status Code: 200

JSON:


```
{
 "id": "2",
 "assignment_id": "1",
 "input": "5",
 "expected_output": "25",
 "created_at": "2024-08-09T11:05:00",
 "updated_at": "2024-08-09T11:05:00"
}
```


Actual Output:

Status Code: 200

JSON:


```
{
 "id": "2",
 "assignment_id": "1",
 "input": "5",
 "expected_output": "25",
 "created_at": "2024-08-09T11:05:00",
 "updated_at": "2024-08-09T11:05:00"
}
```


Result: Success

Test Case: Bad Request for test cases

API being tested: /test_cases/<int:test_case_id> (POST method)
Inputs:

{
 "assignment_id": 1,
 "input": "?",
 "output": "error"
}

Expected Output:
Status Code: 400 **(**BAD Request**)**
JSON:

{
 "message": "Bad Request: Input must be a valid string."
}

Actual Output:
Status Code: 400 **(**BAD Request**)**
JSON:

{
 "message": "Bad Request: Input must be a valid string."
}

Result: Success

Test Case: Successfully getting a Submission.
API being tested:/submissions/<int:submission_id> (GET method)
Inputs:
submission_id: 1

Expected Output:
Status Code: 200
JSON:

{
 "id": 1,
 "assignment_id": "1",
 "student_id": "2001",
 "content": "print(4 ** 2)",
 "submission_date": "2024-08-09T11:10:00",
 "grade": null,
 "feedback": null,
 "created_at": "2024-08-09T11:10:00",
 "updated_at": "2024-08-09T11:10:00"
}

Actual Output:
Status Code: 200
JSON:
{
 "id": 1,
 "assignment_id": "1",
 "student_id": "2001",
 "content": "print(4 ** 2)",
 "submission_date": "2024-08-09T11:10:00",
 "grade": null,
 "feedback": null,
 "created_at": "2024-08-09T11:10:00",
 "updated_at": "2024-08-09T11:10:00"
}

Result: Success

Test Case: Creating a submission

API being tested: /post/submissions (POST method)
Inputs:
{
 "assignment_id": 1,
 "student_id": "2001",

```
  "content": "print(5 ** 2)"
}
```


Expected Output:
Status Code: 200
JSON:

```
{
  "id": 2,
  "assignment_id": "1",
  "student_id": "2001",
  "content": "print(5 ** 2)",
  "submission_date": "2024-08-09T11:15:00",
  "grade": null,
  "feedback": null,
  "created_at": "2024-08-09T11:15:00",
  "updated_at": "2024-08-09T10:15:00"
}
```

Actual Output:
Status Code: 200
JSON:

```
{
  "id": 2,
  "assignment_id": "1",
  "student_id": "2001",
  "content": "print(5 ** 2)",
  "submission_date": "2024-08-09T11:15:00",
  "grade": null,
  "feedback": null,
  "created_at": "2024-08-09T11:15:00",
  "updated_at": "2024-08-09T11:15:00"
}
```


Result: Success

Test Cases for Gen AI Features

Test Case: Successfully getting a Virtual Instructor Query.

API being tested:
/virtual_instructor_queries/<string:course_id>/<string:student_id> (GET method)

Inputs:

{ "course_id": "course123", "student_id": "456" }

Expected Output:

Status Code: 200

JSON:

[
  {
    "id": "1",
    "student_id": "456",
    "course_id": "1",
    "gen_query": "Explain the concept of linear regression.",
    len("response") >0
  }
]

Actual Output:

Status Code: 200

JSON:

{
  "id": "1",
  "student_id": "456",
  "course_id": "1",
  "gen_query": "Explain the concept of linear regression.",
  "response": "> Linear regression is a fundamental statistical technique used in data science to model the relationship between a dependent variable (the one you want to predict) and one or more independent variables (the ones you use to make the prediction). \n> \n> Imagine you want to predict the price of a house based on its size. Linear regression assumes a linear relationship: as the size increases, the price increases proportionally. It finds the \"best-fit\" line that minimizes the

distance between the actual house prices and the line's predictions.\n> \n> Mathematically, this line is represented by an equation: y = mx + c, where 'y' is the predicted price, 'x' is the size, 'm' is the slope (how much the price changes per unit increase in size), and 'c' is the intercept (the price when the size is zero). \n> \n> Linear regression helps us understand the relationship between variables, make predictions, and identify important factors influencing a phenomenon. \n",
  "created_at": "Fri, 09 Aug 2024 23:36:35 -0000"
}
Result: Success

Test Case: Successfully posting a Virtual Instructor Query.
API being tested: /virtual_instructor_queries (POST method)
Inputs:
JSON
{
  "student_id": "456",
  "course_id": "1",
  "gen_query": "Explain the concept of linear regression."
}

Expected Output:
Status Code: 201
JSON:
{
  "id": "1",
  "student_id": "456",
  "course_id": "1",
  "gen_query": "Explain the concept of linear regression.",
  len("response") > 0
}
Actual Output:
Status Code: 201
JSON:
{
  "id": "1",

"student_id": "456",
"course_id": "1",
"gen_query": "Explain the concept of linear regression.",
"response": "> Linear regression is a fundamental statistical technique used in data science to model the relationship between a dependent variable (the one you want to predict) and one or more independent variables (the ones you use to make the prediction). \n> \n> Imagine you want to predict the price of a house based on its size. Linear regression assumes a linear relationship: as the size increases, the price increases proportionally. It finds the \"best-fit\" line that minimizes the distance between the actual house prices and the line's predictions.\n> \n> Mathematically, this line is represented by an equation: y = mx + c, where 'y' is the predicted price, 'x' is the size, 'm' is the slope (how much the price changes per unit increase in size), and 'c' is the intercept (the price when the size is zero). \n> \n> Linear regression helps us understand the relationship between variables, make predictions, and identify important factors influencing a phenomenon. \n",
"created_at": "Fri, 09 Aug 2024 23:36:35 -0000"
}

Result: Success
Inputs:
JSON
{
"student_id": "456",
"course_id": "12",
"gen_query": "Explain the concept of linear regression."
}

Expected Output:
Status Code: 201
JSON:
{
"id": "1",
"student_id": "456",
"course_id": "12",
"gen_query": "Explain the concept of linear regression.",

len("response") > 0
}
Actual Output:
Status Code: 404
JSON:
{
"message": "Course not found"
}

Result: Success
Test Case: Successfully deleting a Virtual Instructor Query.
API being tested: /virtual_instructor_queries/<int:query_id> (DELETE method)
Inputs:
{ "course_id": "course123", "student_id": "student456" }
Expected Output:
Status Code: 200
JSON:
{
"message": "All queries deleted"
}

Actual Output:
Status Code: 200
JSON:
{
"message": "All queries deleted"
}

Result: Success
Inputs:
{ "course_id": "course123", "student_id": "student456" }
Expected Output:
Status Code: 200
JSON:

{
"message": "All queries deleted"
}

Actual Output:
Status Code: 200
JSON:
{
"message": "All queries deleted"
}

Result: Success

Test Case: Successfully getting a Student Planner.
API being tested: /student_planners/<int:planner_id> (GET method)
Inputs:
{ "student_id": "1", "content_type": "question" }
Expected Output:
Status Code: 200
JSON:
[{
"id": 1,
"student_id": "1",
len("response")>0
"level": "intermediate",
"content_type": "question",
"gen_query": "How to become a data scientist."
}]
Actual Output:
Status Code: 200
JSON:
[{
"id": 1,
"student_id": "1",
len("response")>0

"level": "intermediate",
"content_type": "question",
"gen_query": "How to become a data scientist."
}]

Result: Success

Test Case: Successfully posting a Student Planner query.

API being tested: /student_planners (POST method)

Inputs:

{
"student_id": "1",
"level": "intermediate",
"content_type": "question",
"gen_query": "How to become a data scientist."
}

Expected Output:

Status Code: 201

JSON:

{
"id": 1,
"student_id": "1",
len("response")>0
"level": "intermediate",
"content_type": "question",
"gen_query": "How to become a data scientist."
}

Actual Output:

Status Code: 201

JSON:

{
"id": 1,
"student_id": "1",
len("response")>0
"level": "intermediate",

"content_type": "question",
"gen_query": "How to become a data scientist."
}
Result: Success

Inputs:
{
"student_id": "2",
"level": "intermediate",
"content_type": "question",
"gen_query": "How to become a data scientist."
}
Expected Output:
Status Code: 404
JSON:
{
"message": "No enrolled courses found for the student"
}
Actual Output:
Status Code: 404
JSON:
{
"message": "No enrolled courses found for the student"
}
Result: Success

Test Case: Successfully deleting a Student Planner query.
API being tested: /student_planners/<int:query_id> (DELETE method)
Inputs:
{ "query_id": "1" }

Expected Output:
Status Code: 200
JSON:
{

"message": "Curation deleted successfully"
}

Actual Output:

Status Code: 200

JSON:

{
"message": "Curation deleted successfully"
}

Result: Success

Instructor Component Testing:

Test Case: Successfully getting an Instructor Content Planner response.

API being tested: /instructor_content_planners/<int:query_id> (GET method)

Inputs:

{ "instructor_id": "instructor789", "course_id": "1", "content_type": "resource" }

Expected Output:

Status Code: 200

JSON:

[
{
"id": "1",
"instructor_id": "instructor789",
"course_id": "1",
"content_type": "resource",
len("gen_ai_resources") > 0
"gen_ai_questions": null,
"gen_ai_answers": null,
"gen_ai_question_detail": null,
"difficulty_level": "intermediate"
}
]

Actual Output:
Status Code: 200
JSON:

[
{
"id": "1",
"instructor_id": "instructor789",
"course_id": "1",
"content_type": "resource",
len("gen_ai_resources") > 0
"gen_ai_questions": null,
"gen_ai_answers": null,
"gen_ai_question_detail": null,
"difficulty_level": "intermediate"
}
]

Result: Success

Test Case: Successfully submitting an Instructor Content Planner Query.
API being tested: /instructor_content_planners (POST method)
Inputs:

{
"instructor_id": "instructor789",
"course_id": "1",
"gen_query": "Introduction to Machine Learning",
"content_type": "resource",
"difficulty_level": "intermediate"
}

Expected Output:
Status Code: 201
JSON:

{
"id": "1",
"instructor_id": "instructor789",

```
  "course_id": "1",
  "content_type": "resource",
  len("gen_ai_resources") > 0
  "gen_ai_questions": null,
  "gen_ai_answers": null,
  "gen_ai_question_detail": null,
  "difficulty_level": "intermediate"
}
```


Actual Output:
Status Code: 201
JSON:


```
{
  "id": "1",
  "instructor_id": "instructor789",
  "course_id": "1",
  "content_type": "resource",
  len("gen_ai_resources") > 0
  "gen_ai_questions": null,
  "gen_ai_answers": null,
  "gen_ai_question_detail": null,
  "difficulty_level": "intermediate"
}
```


Result: Success

Test Case: Successfully deleting an Instructor Content Planner Query.
API being tested: /instructor_content_planners/<int:planner_id> (DELETE method)
Inputs:

{ "query_id": "1" }

Expected Output:
Status Code: 200
JSON:

{
"message": "Content deleted successfully"
}

Actual Output:
Status Code: 200
JSON:
{
"message": "Content deleted successfully"
}
Result: Success

2. Test Summary and Pytest Screenshots:

2.1 Configuration Testing Component:

(a) Screenshot 1:

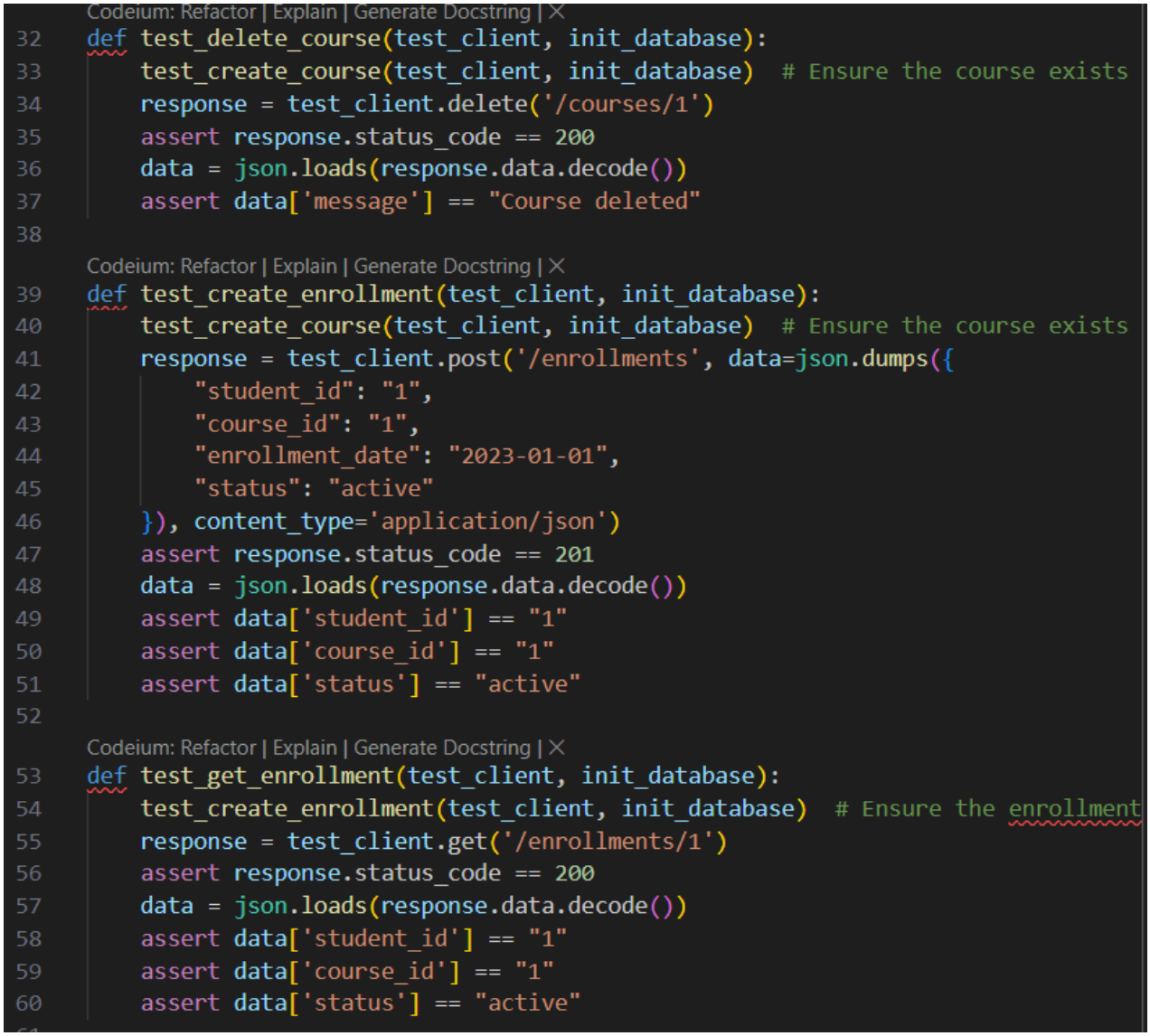
```python
# tests/functional/conftest.py
import pytest
from main import create_app
from application.config import TestingConfig
from application.database import db
from application.models import User, Course

Codeium: Refactor | Explain | Generate Docstring | ×
@pytest.fixture(scope='module')
def test_client():
    app, _ = create_app()
    app.config.from_object(TestingConfig)
    testing_client = app.test_client()

    with app.app_context():
        db.create_all()
        yield testing_client
        db.drop_all()

Codeium: Refactor | Explain | Generate Docstring | ×
@pytest.fixture(scope='module')
def init_database():
    app, _ = create_app()
    app.config.from_object(TestingConfig)

    with app.app_context():
        db.create_all()

        # Create test data
        user1 = User(email="test1@example.com", username="testuser1", password="password")
        user2 = User(email="test2@example.com", username="testuser2", password="password")
        db.session.add(user1)
        db.session.add(user2)
        db.session.commit()

        course1 = Course(title="Test Course 1", description="This is a test course.")
        db.session.add(course1)
        db.session.commit()

        yield db
        db.session.remove()
        db.drop_all()
```

2.2 API Testing Component:

(b) <u>Screenshot 2:</u>

```
Codeium: Refactor | Explain | Generate Docstring | ×
def test_create_course(test_client, init_database):
    response = test_client.post('/courses', data=json.dumps({
        "title": "New Test Course",
        "description": "A new test course description"
    }), content_type='application/json')
    assert response.status_code == 201
    data = json.loads(response.data.decode())
    assert data['title'] == "New Test Course"
    assert data['description'] == "A new test course description"

Codeium: Refactor | Explain | Generate Docstring | ×
def test_get_course(test_client, init_database):
    test_create_course(test_client, init_database)  # Ensure the course exists
    response = test_client.get('/courses/1')
    assert response.status_code == 200
    data = json.loads(response.data.decode())
    assert data['title'] == "New Test Course"
    assert data['description'] == "A new test course description"

Codeium: Refactor | Explain | Generate Docstring | ×
def test_update_course(test_client, init_database):
    test_create_course(test_client, init_database)  # Ensure the course exists
    response = test_client.put('/courses/1', data=json.dumps({
        "title": "Updated Test Course",
        "description": "Updated test course description"
    }), content_type='application/json')
    assert response.status_code == 200
    data = json.loads(response.data.decode())
    assert data['title'] == "Updated Test Course"
    assert data['description'] == "Updated test course description"
```

© **Screenshot 3:**



```
Codeium: Refactor | Explain | Generate Docstring | ×
def test_delete_course(test_client, init_database):
    test_create_course(test_client, init_database)  # Ensure the course exists
    response = test_client.delete('/courses/1')
    assert response.status_code == 200
    data = json.loads(response.data.decode())
    assert data['message'] == "Course deleted"

Codeium: Refactor | Explain | Generate Docstring | ×
def test_create_enrollment(test_client, init_database):
    test_create_course(test_client, init_database)  # Ensure the course exists
    response = test_client.post('/enrollments', data=json.dumps({
        "student_id": "1",
        "course_id": "1",
        "enrollment_date": "2023-01-01",
        "status": "active"
    }), content_type='application/json')
    assert response.status_code == 201
    data = json.loads(response.data.decode())
    assert data['student_id'] == "1"
    assert data['course_id'] == "1"
    assert data['status'] == "active"

Codeium: Refactor | Explain | Generate Docstring | ×
def test_get_enrollment(test_client, init_database):
    test_create_enrollment(test_client, init_database)  # Ensure the enrollment
    response = test_client.get('/enrollments/1')
    assert response.status_code == 200
    data = json.loads(response.data.decode())
    assert data['student_id'] == "1"
    assert data['course_id'] == "1"
    assert data['status'] == "active"
```

(d) Screenshot 4:

```
62  def test_update_enrollment(test_client, init_database):
63      test_create_enrollment(test_client, init_database)  # Ensure the enrollment
64      response = test_client.put('/enrollments/1', data=json.dumps({
65          "student_id": "2",
66          "course_id": "1",
67          "enrollment_date": "2023-01-01",
68          "status": "completed"
69      }), content_type='application/json')
70      assert response.status_code == 200
71      data = json.loads(response.data.decode())
72      assert data['student_id'] == "2"
73      assert data['course_id'] == "1"
74      assert data['status'] == "completed"
75
        Codeium: Refactor | Explain | Generate Docstring | ×
76  def test_delete_enrollment(test_client, init_database):
77      test_create_enrollment(test_client, init_database)  # Ensure the enrollment
78      response = test_client.delete('/enrollments/1')
79      assert response.status_code == 200
80      data = json.loads(response.data.decode())
81      assert data['message'] == "Enrollment deleted"
82
        Codeium: Refactor | Explain | Generate Docstring | ×
83  def test_create_module(test_client, init_database):
84      test_create_course(test_client, init_database)  # Ensure the course exists
85      response = test_client.post('/modules', data=json.dumps({
86          "course_id": "1",
87          "title": "New Test Module",
88          "description": "A new test module description",
89          "week": 1,
90          "created_at": "2023-01-01",
91          "updated_at": "2023-01-01"
92      }), content_type='application/json')
93      assert response.status_code == 201
94      data = json.loads(response.data.decode())
95      assert data['title'] == "New Test Module"
```

(e) Screenshot 5:

```
98  def test_get_module(test_client, init_database):
99      test_create_module(test_client, init_database)  # Ensure the module exists
100     response = test_client.get('/modules/1')
101     assert response.status_code == 200
102     data = json.loads(response.data.decode())
103     assert data['title'] == "New Test Module"
104     assert data['description'] == "A new test module description"
105
        Codeium: Refactor | Explain | Generate Docstring | ×
106 def test_update_module(test_client, init_database):
107     test_create_module(test_client, init_database)  # Ensure the module exists
108     response = test_client.put('/modules/1', data=json.dumps({
109         "course_id": "1",
110         "title": "Updated Test Module",
111         "description": "Updated test module description",
112         "week": 2,
113         "created_at": "2023-01-01",
114         "updated_at": "2023-01-01"
115     }), content_type='application/json')
116     assert response.status_code == 200
117     data = json.loads(response.data.decode())
118     assert data['title'] == "Updated Test Module"
119     assert data['description'] == "Updated test module description"
120
        Codeium: Refactor | Explain | Generate Docstring | ×
121 def test_delete_module(test_client, init_database):
122     test_create_module(test_client, init_database)  # Ensure the module exists
123     response = test_client.delete('/modules/1')
124     assert response.status_code == 200
125     data = json.loads(response.data.decode())
126     assert data['message'] == "Module deleted"
127
```

(f) Screenshot 6:

```
Codeium: Refactor | Explain | Generate Docstring | X
def test_create_lecture(test_client, init_database):
    test_create_module(test_client, init_database)  # Ensure the module exists
    response = test_client.post('/lectures', data=json.dumps({
        "module_id": "1",
        "title": "New Test Lecture",
        "content": "Content for the new test lecture"
    }), content_type='application/json')
    assert response.status_code == 201
    data = json.loads(response.data.decode())
    assert data['title'] == "New Test Lecture"
    assert data['content'] == "Content for the new test lecture"

Codeium: Refactor | Explain | Generate Docstring | X
def test_get_lecture(test_client, init_database):
    test_create_lecture(test_client, init_database)  # Ensure the lecture exist
    response = test_client.get('/lectures/1')
    assert response.status_code == 200
    data = json.loads(response.data.decode())
    assert data['title'] == "New Test Lecture"
    assert data['content'] == "Content for the new test lecture"
```

2.3 Unit Testing Component:

(g) Screenshot 7:

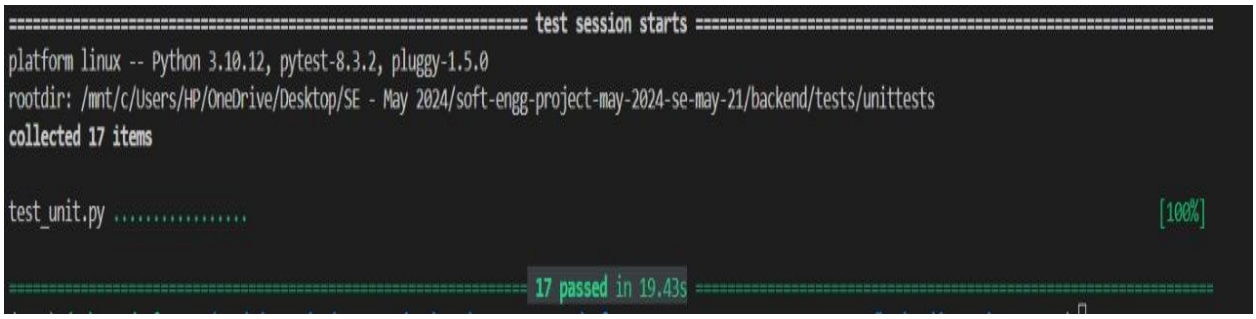
```python
import pytest
from application.models import (
    User, Course, Enrollment, Module, Lecture, Note, Assignment,
    ProgrammingAssignment, Question, Submission,
    VirtualInstructorQuery, InstructorContentPlanner, CourseStatistics,
    AdminCourseManagement, StudentDetailsManagement, AssignmentStatus, StudentCuration
)
from application.database import db

Codeium: Refactor | Explain | Generate Docstring | X
def test_user_model():
    user = User(email="test@example.com", username="testuser", password="password")
    assert user.email == "test@example.com"
    assert user.username == "testuser"
    assert user.password == "password"

Codeium: Refactor | Explain | Generate Docstring | X
def test_course_model():
    course = Course(title="Test Course", description="Test Course Description")
    assert course.title == "Test Course"
    assert course.description == "Test Course Description"

Codeium: Refactor | Explain | Generate Docstring | X
def test_enrollment_model():
    enrollment = Enrollment(student_id="1", course_id="1", enrollment_date="2023-01-01", status="active")
    assert enrollment.student_id == "1"
    assert enrollment.course_id == "1"
    assert enrollment.enrollment_date == "2023-01-01"
    assert enrollment.status == "active"
```

(h) Screenshot 8:

```python
def test_module_model():
    module = Module(course_id="1", title="Test Module", description="Test Module Description", week=1)
    assert module.course_id == "1"
    assert module.title == "Test Module"
    assert module.description == "Test Module Description"
    assert module.week == 1


Codeium: Refactor | Explain | Generate Docstring | X
def test_lecture_model():
    lecture = Lecture(module_id="1", title="Test Lecture", content="Test Lecture Content")
    assert lecture.module_id == "1"
    assert lecture.title == "Test Lecture"
    assert lecture.content == "Test Lecture Content"


Codeium: Refactor | Explain | Generate Docstring | X
def test_note_model():
    note = Note(module_id="1", title="Test Note", content="Test Note Content")
    assert note.module_id == "1"
    assert note.title == "Test Note"
    assert note.content == "Test Note Content"


Codeium: Refactor | Explain | Generate Docstring | X
def test_assignment_model():
    assignment = Assignment(title="Test Assignment", description="Test Assignment Description")
    assert assignment.title == "Test Assignment"
    assert assignment.description == "Test Assignment Description"


Codeium: Refactor | Explain | Generate Docstring | X
def test_programming_assignment_model():
    programming_assignment = ProgrammingAssignment(assignment_id="1", editor_content="Editor content", sandbox_environment="Sandbox environment", he
    assert programming_assignment.assignment_id == "1"
    assert programming_assignment.editor_content == "Editor content"
    assert programming_assignment.sandbox_environment == "Sandbox environment"
    assert programming_assignment.help_button_enabled is True
```

2.4 Result Summary:

(i) Screenshot 9:



Implementation Details and Technologies Used

Tool	Usage
GitHub	Version Control and Collaboration
VS Code	IDE
Google Chrome and Microsoft Edge	Browser
Figma	Prototype Design
Canva	Designing Storyboards and Reports, Project PPT
Swagger	Yaml File Generation
JIRA	Project Management Tool
Docker	Deployment
Ace	Text Editor (In the Programming Portal)

Frontend

- VueJS 3
- Vite
- VueRouter
- Javascript - Async Await and Fetch API

User Interface (UI)

- HTML
- CSS
- Bootstrap

Backend and API

- Flask
- Flask Restful
- Flask SQLAlchemy (Object Relational Mapping)
- Requests

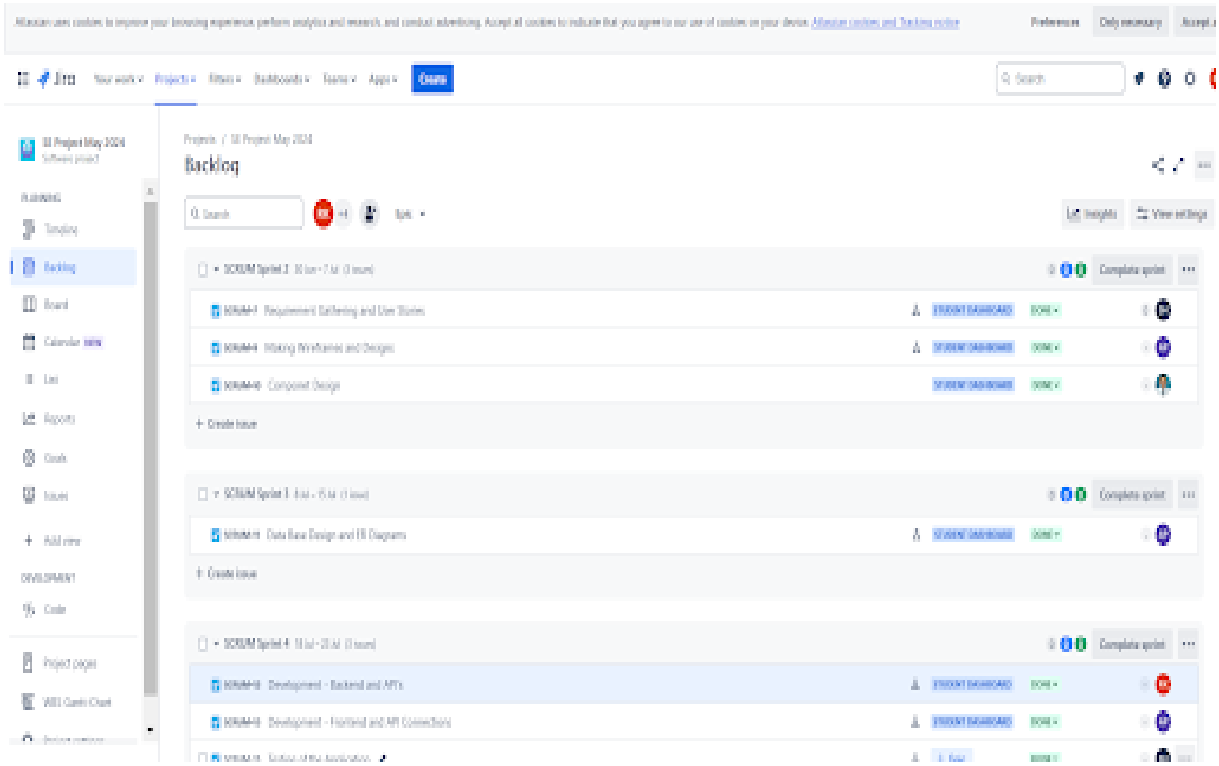
AI API

- Gemminai (LLM)
- Flask and Flask Restful (API Endpoints)

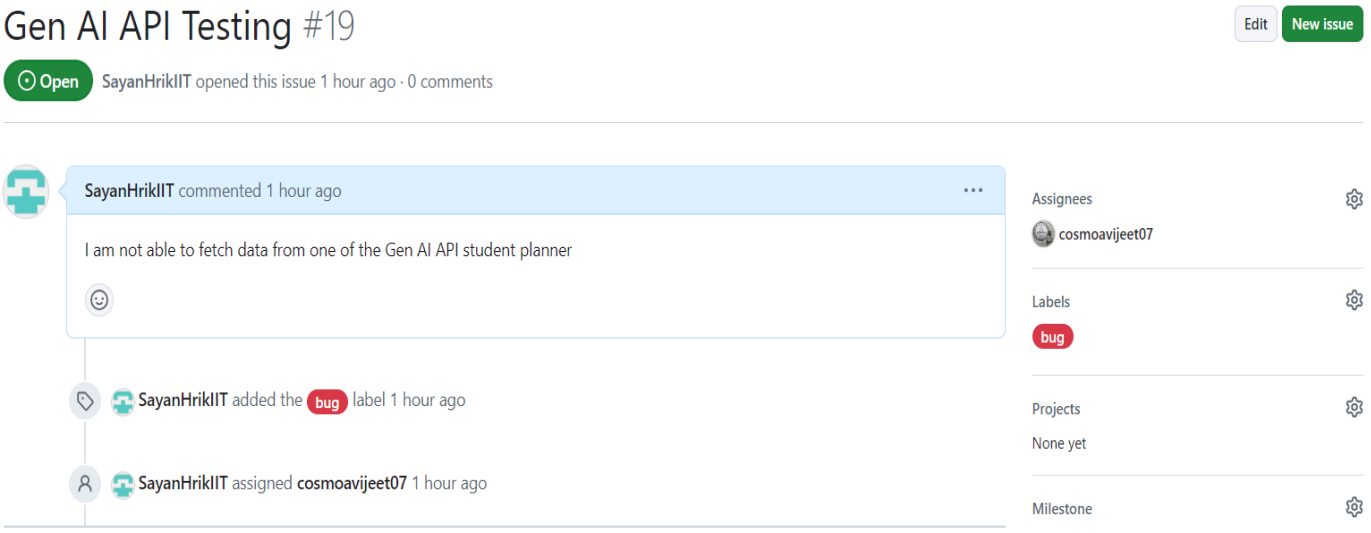
Database

- SQLitte 3 (Relational Database)

Jira Project Management (Final Scrum Board ScreenShot after Project Completion)



Code Review, Issue Reporting and Tracking



Backend API #22

Edit | New issue

Closed — Ramesh21f2000549 opened this issue 6 hours ago · 1 comment

Ramesh21f2000549 commented 6 hours ago

Course API post method not working

Ramesh21f2000549 added the bug label 6 hours ago

Ramesh21f2000549 assigned **21f2000143** and **Ramesh21f2000549** and unassigned **Ramesh21f2000549** 6 hours ago

SayanHrikIIT self-assigned this 3 minutes ago

SayanHrikIIT commented now

Hi Ramesh ! There was a slight syntax issue in the API endpoint I have fixed it. Pl check.
Regards Sayan

SayanHrikIIT closed this as completed now

Assignees: 21f2000143, SayanHrikIIT

Labels: bug

Projects: None yet

Milestone: No milestone

Development: Create a branch for this issue or link a pull request.

Notifications: Customize — Unsubscribe

You're receiving notifications because you're watching this repository.

3 participants

Gitmy3 commented 12 hours ago

I actually can't implement the Nav Bar because several errors are occurring at the Console Tab for creating the Nav Bar for these above mentioned Dashboards.

Gitmy3 added the bug label 12 hours ago

Gitmy3 assigned **21f2000143**, **cosmoavijeet07**, **Ayushrana198**, **SayanHrikIIT**, **uruiitm** and **Ramesh21f2000549** 12 hours ago

Gitmy3 added the help wanted label 12 hours ago

SayanHrikIIT commented now

There was an error with the Vue component configuration here. We have resolved it now Preetam.

SayanHrikIIT closed this as completed now

Assignees: 21f2000143, cosmoavijeet07, Ayushrana198, SayanHrikIIT, uruiitm, Ramesh21f2000549

Labels: bug, help wanted

Projects: None yet

Milestone: No milestone

Development: Create a branch for this issue or link a pull request.

Notifications: Customize

Instructions to run the Application

The same instructions apply for Ubuntu / Windows and MacOS

````
# Project Setup
This guide covers the setup for the complete development
environment including the Vue.js frontend and a Python Flask
backend application.

## Prerequisites
Ensure you have NodeJS, Python ,latest Vite Version and Flask
installed on your machine.
## Installation Steps
First, clone the repository to your local machine and navigate into
the project directory.
### Install Node Modules
```bash
npm install
```
### Run the Vue.js Development Server
```bash
npm run dev
```
### Python Application
Set up the backend by running the following
```bash
. local_setup.sh
```
Then to the app run the following
```bash
. local_run.sh
```
## Additional Information
For more detailed information about each component, visit their
respective documentation or help pages.
````

Project Video Presentation Link

https://drive.google.com/file/d/1c2poNJCSurlGdScHUPnR2E2RhCrhF33c/view

Project PPT Link

https://drive.google.com/file/d/1n5AuvUWoG7-GoXwMkPmt4rquPgySVtK6/view?usp=sharing