Software Engineering Project (BSCSS3001)

Milestone 6

Submitted by Team-10

- **Hari Prapan-** 21f3002087
- **OJASV SINGHAL-** 22f3002350
- **Mahak Thakre-** 21f2000845
- **Bindu Yadav-** 21f2000511
- **Pragya Singh-** 21f3001204
- **Santosh Kumar Verma-** 21f1000243
- **Anujj Prem-** 21f3000815
- **Kavya Dwivedi-** 21f3002442

IITM BS Degree Program,
Indian Institute of Technology, Madras, Chennai, Tamil Nadu, India, 600036

Table of contents

1. Project Statement: Integrating Generative AI in Programming Learning Environments

1.1 Objective

Enhance the IITM BS degree program's SEEK portal with generative AI (GenAI) technologies.

1.2 Current Portal Features

- Learning videos
- Assignments
- Resources
- Programming quizzes
- Self-paced learning

1.3 Potential GenAI Integrations

1. **Enhanced Feedback**: Use GenAI to provide detailed feedback on programming tests beyond pass/fail results.

2. **Problem-Solving Support**: Implement GenAI assistance for learners tackling programming problems.

3. **Interactive Content**: Add interactivity to static content like PDFs, documents, and videos using GenAI.

1.4 Project Scope

- Develop a GenAI-enhanced SEEK portal clone for one course, including learning materials, assignments, and quizzes.

- Utilize existing libraries, templates, and APIs (e.g., ACE for code editing, online compiler APIs, free/open-source LLM models like Ollama).

1.5 Flexibility

- Innovate beyond provided examples with other GenAI integration ideas.

2. Various Users

2.1 Learners journey map

Scenario 1: Lecture Summaries Without GenAI Integration.

	STAGE 1: LOGGING IN	STAGE 2: ACCESSING CONTENT	STAGE 3: WATCHING LECTURES	STAGE 4: MAKING NOTES	STAGE 5: REVISION
ACTIONS	Logging into the SEEK portal.	Navigating to the week-wise arranged lectures and assignments.	Watching long video lectures.	Manually making notes from the lectures.	Reviewing the manually taken notes.
EXPERIENCE/ EMOTIONS	Intrigued, motivated	Excited, overwhelmed 	Tired, bored 	Frustated 	Relief from having notes but stress about incomplete understanding 
PAIN POINTS	• Too much content	• Difficulty in finding specific topics quickly.	• Lectures are long and require significant time commitment.	• Time-consuming and often incomplete note-taking.	• Reviewing notes is less effective due to potential gaps.
SOLUTIONS	Main topics covered in lectures easily visible in the lecture summary.	Lecture specific summary	Lecture summary available for every lecture.	• Providing note-taking templates or guides.	• Offering summary sheets or quick revision guides.

Scenario 1: Lecture Summaries With GenAI Integration.

	STAGE 1: LOGGING IN	STAGE 2: ACCESSING CONTENT	STAGE 3: USING GEN-AI TOOL TO MAKE NOTES	STAGE 4: SAVING NOTES	STAGE 5: REVISION
ACTIONS	Logging into the SEEK portal.	Navigating to the week-wise arranged lectures and assignments.	Asking Gen-AI tool to generate lecture notes and highlight key points.	Save the notes that Gen-AI tool generated.	Now i can revise my notes and utilize my time efficiently.
EXPERIENCE/ EMOTIONS	Intrigued, motivated 	Excited, overwhelmed 	Curious, Hopeful 	Happy, Satisfied 	Great to have notes and save time by skipping lengthy lectures. 
PAIN POINTS	• Too much content	• Difficulty in finding specific topics quickly.	• Unsure if the tool is reliable or easy to use.	• Technical difficulties in saving the notes, unclear on where to access the saved notes.	• Reviewing notes is frustrating if it is incomplete and inaccurate.
SOLUTIONS	Main topics covered in lectures easily visible in the lecture summary.	Lecture specific summary	Gen-Ai tool is available for every lecture.	Providing note-taking templates or guides.	Offering summary sheets or quick revision guides.

Scenario 3 : Suggesting New Practice Assignment Questions.

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials.	Completing the provided assignments.	Recognizing the need for more practice.	Searching for additional practice questions online or in textbooks.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Engaged but potentially overwhelmed by limited practice.	Stress due to difficult assignments.	Frustration due to lack of additional practice material. 	Anxiety about finding relevant practice questions.
PAIN POINTS	• Too much content	• Limited practice material in the portal.	• Difficulty in understanding without enough practice.	• Time-consuming process of finding additional materials.	• Inconsistent quality and relevance of found questions.
SOLUTIONS	Having more assignments to practice.	Provide a comprehensive set of practice materials.	Ensure assignments cover a wide range of difficulties.	Offer guidance on where to find additional practice material.	Add a section for recommended practice questions in the portal.

Scenario 3 : Practice Assignment Questions with GenAI

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials and solving assignments.	Recognizing the need for more practice.	Using Gen-AI Integrated tool to find targeted practice assignments.	Gen-AI tool provides targeted practice assignments and their solutions instantly.
EXPERIENCE/ EMOTIONS	Intrigued, motivated to learn. 	Engaged but potentially overwhelmed by limited practice. 	Stress due to difficult assignments. 	Cautious, Intrigued for practice assignments 	Confident, Happy, Satisfied 
PAIN POINTS		• Limited practice material in the portal.	• Difficulty in understanding without enough practice.	• Unsure if the tool provides targeted practice assignments with solutions.	• Inconsistent quality and relevance of questions may be frustrating.
SOLUTIONS		Provide a comprehensive set of practice materials.	Ensure assignments cover a wide range of difficulties.	Gen-AI tool to offer guidance on where to find practice material.	Add a section for recommended practice questions in the portal.

Scenario 4 : Providing/Suggesting Important PYQ Questions

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials.	Searching for previous year question papers online or in libraries.	Practicing with manually found questions.	Reviewing answers and performance.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Engaged but lacking direction on PYQs	Frustration due to difficulty finding PYQs. 	Uncertainty about the relevance of the questions. 	Anxiety about accuracy and relevance of answers.
PAIN POINTS	• Lack of PYQs	• PYQs not available on portal.	• Time-consuming process of finding PYQs	• Inconsistent quality and relevance of found questions.	• Difficulty in self-assessment.
SOLUTIONS	Having a section for PYQs.	GenAI suggesting important question based on previous term questions.	Provide important question for quiz preparations based on PYQs	Recommend a set of reliable previous year questions	Provide answer keys and previous year questions.

Scenario 4 : Providing Imported PYQ Questions with GenAI .

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials.	Recognizing the need to search for previous year question papers.	Using Gen-AI Integrated tool to get previous year question papers with answer key.	Practicing, reviewing answers and evaluating performance.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Engaged but lacking direction on PYQs 	Frustration due to difficulty finding PYQs. 	Cautious, Intrigued for previous year question papers 	Anxiety about accuracy and relevance of answers. 
PAIN POINTS		• PYQs not available on portal.	• Time-consuming process of finding PYQs	• Uncertainty if provided question paper is correct.	• Difficulty in self-assessment.
SOLUTIONS			Provide important question for quiz preparations based on PYQs	Recommend a set of reliable previous year questions	Provide answer keys and previous year questions.

Scenario 5 : Providing More Information About Programming Assignment Errors.

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials.	Completing programming assignments.	Receiving feedback on assignments.	Trying to understand why certain test cases failed.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Engaged but potentially confused by complex material.	Stress from trying to complete assignments without enough feedback.	Frustration due to lack of detailed feedback 	Confusion and frustration over unresolved errors and unpassed private test cases.
PAIN POINTS		• Limited explanations for wrong answers.	• Difficulty understanding assignment requirements.	• Lack of detailed feedback on errors	• Difficulty in identifying the root cause of errors.
SOLUTIONS		Solve programming and practice assignments.	Clear and detailed assignment instructions.	Providing detailed feedback on common errors.	Offering office hours or additional support sessions.

Scenario 5 : Beyond Syntax Errors, Leveraging GenAI for Programming Assignments .

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material.	Studying the provided materials.	Completing programming assignments.	Using Gen-AI Integrated tool to analyze the code upon submission.	Gen-AI provides explanations written in a clear, concise manner, tailored to the student's learning level.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Engaged but potentially confused by complex material. 	Confused and Stressed from a simple "compile error" message. 	Cautious, Intrigued for analysis. 	Empowered and Motivated to improve. 
PAIN POINTS		• Limited explanations for wrong answers.	• Unclear error messages can be frustrating and difficult to decipher.	• Difficulty in understanding complex error explanations or suggested fixes.	• Unsure if all errors were addressed or if new ones were introduced.
SOLUTIONS			Clear and detailed assignment instructions.	Providing detailed feedback on common errors.	Offering office hours or additional support sessions.

Scenario 6 : Analyzing Previous Wrong Answers to Identify Weak Topics

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material	Completing practice assignments and graded assignments.	Manually reviewing wrong answers from assignments.	Identifying weak topics based on manual review.	Searching for additional resources to improve in weak areas.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Mixed emotions, depending on performance	Frustration due to the time-consuming review process. 	Uncertainty about correctly identifying all weak topics.	Determination to improve but uncertainty about resources.
PAIN POINTS		• Stress and anxiety from assessments.	• Time-consuming and inefficient review process	• Difficulty in accurately identifying all weak areas.	• Inconsistent quality of found resources.
SOLUTIONS		Clear feedback on assessments.	Providing structured review guides	Offering tools to help identify weak areas	Recommending high-quality study resources

Scenario 6 : GenAi-Powered Weak Topic Identification from Wrong Answers.

	STAGE 1:	STAGE 2:	STAGE 3:	STAGE 4:	STAGE 5:
ACTIONS	Logging into the SEEK portal and navigating through study material	Completing practice assignments and graded assignments.	Asking Gen-AI to analyze and breakdown correct and incorrect answers.	Identifying weak topics based on breakdown received..	Learning paths provided by Gen-AI to revisit and study the identified weak topics.
EXPERIENCE/ EMOTIONS	Initial motivation to learn. 	Mixed emotions, depending on performance 	Cautious, Intrigued for analysis. 	Focused, determined to do well 	Empowered and Motivated to improve. 
PAIN POINTS		• Stress and anxiety from assessments.	• Uncertainty if provided analysis is correct.	• Difficulty in accurately identifying all weak areas.	• Inconsistent quality of found resources.
SOLUTIONS			Offering Gen-AI Integrated tool to help identify weak areas	Provide a prioritized list of weak topics with clear learning paths.	Provide additional resources and support options, such as connecting with instructors or a learning community forum, to address any lingering difficulties.

2.2. Identifying Various Types of Users

1.1 Primary Users

Students: They are the primary users who interact with the GENAI system to submit queries, access study materials, receive personalized study recommendations, engage in interactive learning sessions, and track their progress.

1.2 Secondary Users

Course Instructors: They have elevated privileges and are responsible for managing students, configuring course settings, creating and updating course content, monitoring student progress, and receiving notifications about student performance and engagement.

1.3 Tertiary Users

Higher Authorities: While not directly interacting with the GENAI system, they receive notifications via webhooks or other communication channels for important updates or escalated issues. They may intervene in critical situations, make high-level decisions, and provide necessary support and resources.

2.2. User Stories

1. As a: Student

I want: The portal to provide detailed feedback on my programming assignments, highlighting errors and suggesting improvements within 5 minutes of the deadline.

So that: I can quickly learn from my mistakes and improve my coding skills.

2. As a: Learner

I want: To use a GenAI tool for note-taking and highlighting key points.

So that: I can focus on understanding the lecture rather than just documenting it.

3. As a: Student

I want: The LMS to generate concise summaries of lecture videos, summarizing key points and concepts in under 200 words.

So that: I can quickly review and understand the main ideas.

4. As a: Student

I want: The LMS to provide a detailed comparison of my coding assignment against the optimal solution, including a similarity score and areas for improvement within 5 minutes of the deadline.

So that: I can understand how to achieve a better solution.

5. As a: Student

I want: Access to a course-specific chatbot within the portal that can provide real-time assistance with a response time of under 30 seconds.

So that: I can resolve issues and learn more efficiently.

6. As a: Student

I want: GenAI to create a dynamic study plan based on my course load and deadlines.

So that: I can manage my time effectively and keep up with all my assignments and exams.

7. As a: Student

I want: GenAI to enhance lecture transcripts in my native language.

So that: I can understand the concepts better.

8. As a: Student

I want: GenAI to provide instant feedback on my practice assignments.

So that: I can immediately understand my mistakes and improve

9. As a: Student

I want: The IDE in the SEEK portal to suggest code completions, improvements and syntax corrections.

So that: I can write code more efficiently and with fewer errors.

10. As a: Student

I want: GenAI to provide hints for fixing issues related to failed hidden test cases in my programming assignment.

So that: I can improve my code and understand the underlying problems without knowing the exact hidden test cases.

11. As a: Student

I want: GenAI to analyze my previous wrong answers and suggest the topics I am weak in.

So that: I can focus my studies on areas where I need improvement.

12. As a: Course Instructor

I want: The portal to auto-grade programming assignments with 90% accuracy and provide comprehensive reports to students within 10 minutes of the deadline

So that: I can reduce my grading workload and ensure timely feedback.

13. As a: Course Instructor

I want: Integrating a chatbot that uses NLP to answer students' questions in real-time.

So that: This can include clarifying doubts, providing additional resources, or guiding through problem-solving steps.

14. As a: Course Instructor

I want: The portal to include a course-specific chatbot that can accurately answer at least 80% of student queries within 30 seconds

So that: I can ensure students have continuous support even outside of office hours.

15. As a: Learner

I want: AI-moderated study groups where students can collaborate on assignments and quiz preparations.

So that: AI can suggest group activities, moderate discussions, and ensure productive collaboration.

16. As a: Learner

I want: Using AI to create gamified elements such as badges, leaderboards, and personalized challenges

So that: It will help in increasing engagement and motivation

17. As a: Learner

I want: Using AI to automate the grading of assignments and quizzes, providing instant feedback

So that: Advanced systems can offer detailed explanations for incorrect answers and suggest resources for improvement.

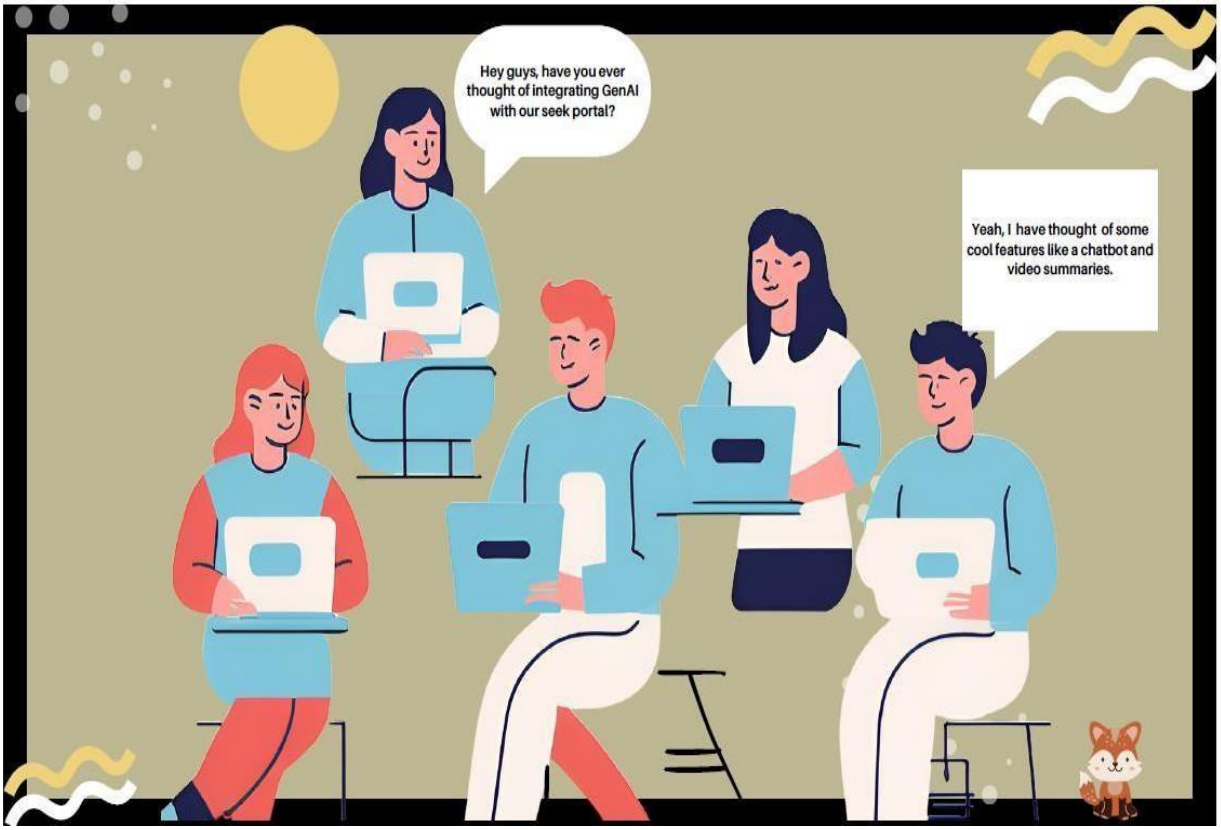
18. As a: Course Instructor

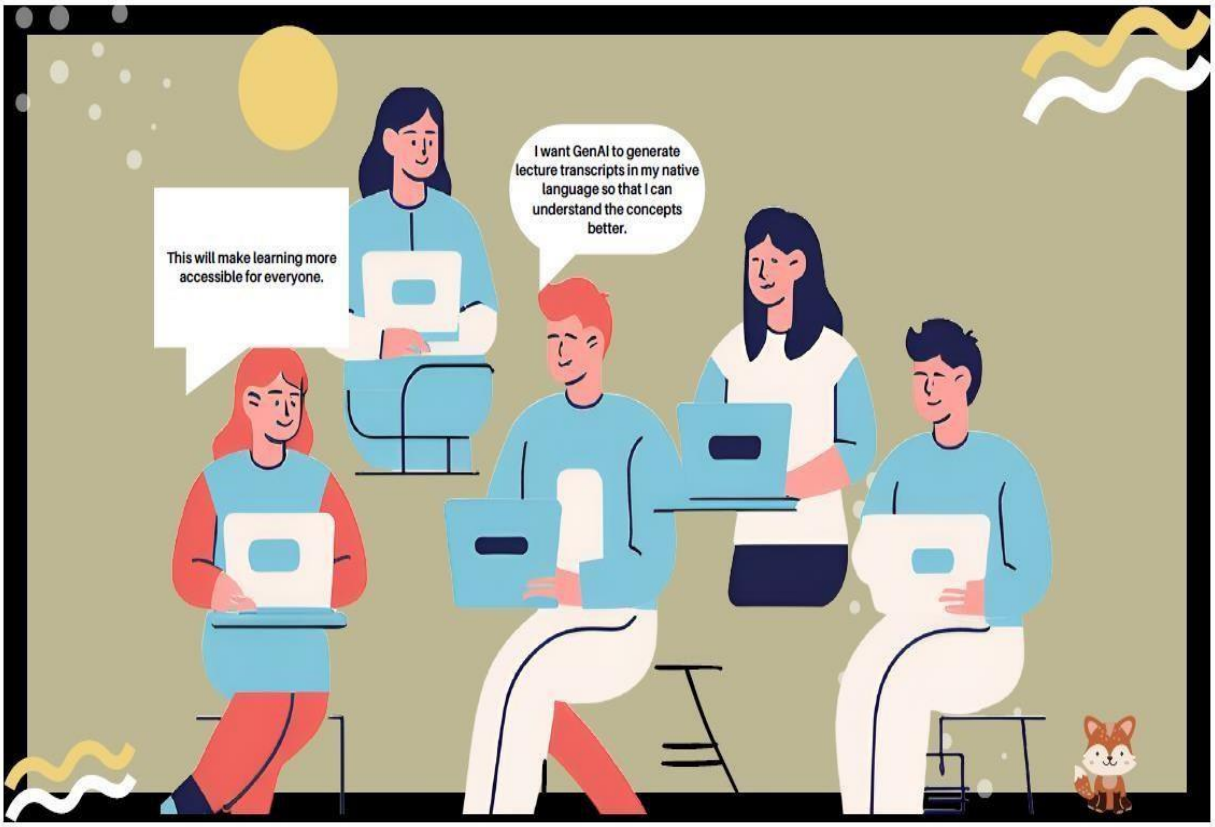
I want: GenAI to only be a tool for small queries and not spoon-feed the answers to students

So that: They can improve their analytical skills and not rely on readymade answers.

3. Wireframes:

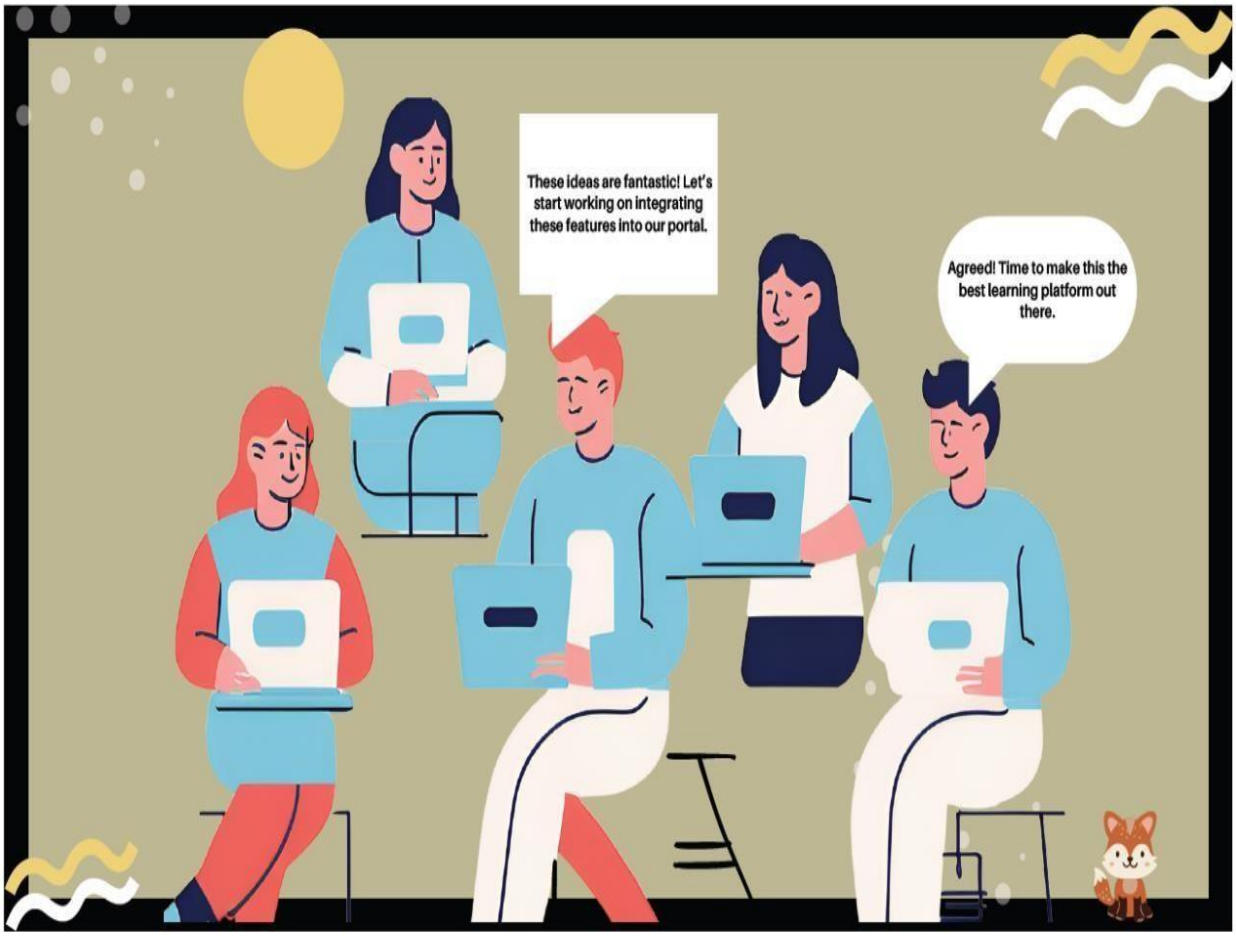
3.1 Storyboard




This will make learning more accessible for everyone.
I want GenAI to generate lecture transcripts in my native language so that I can understand the concepts better.

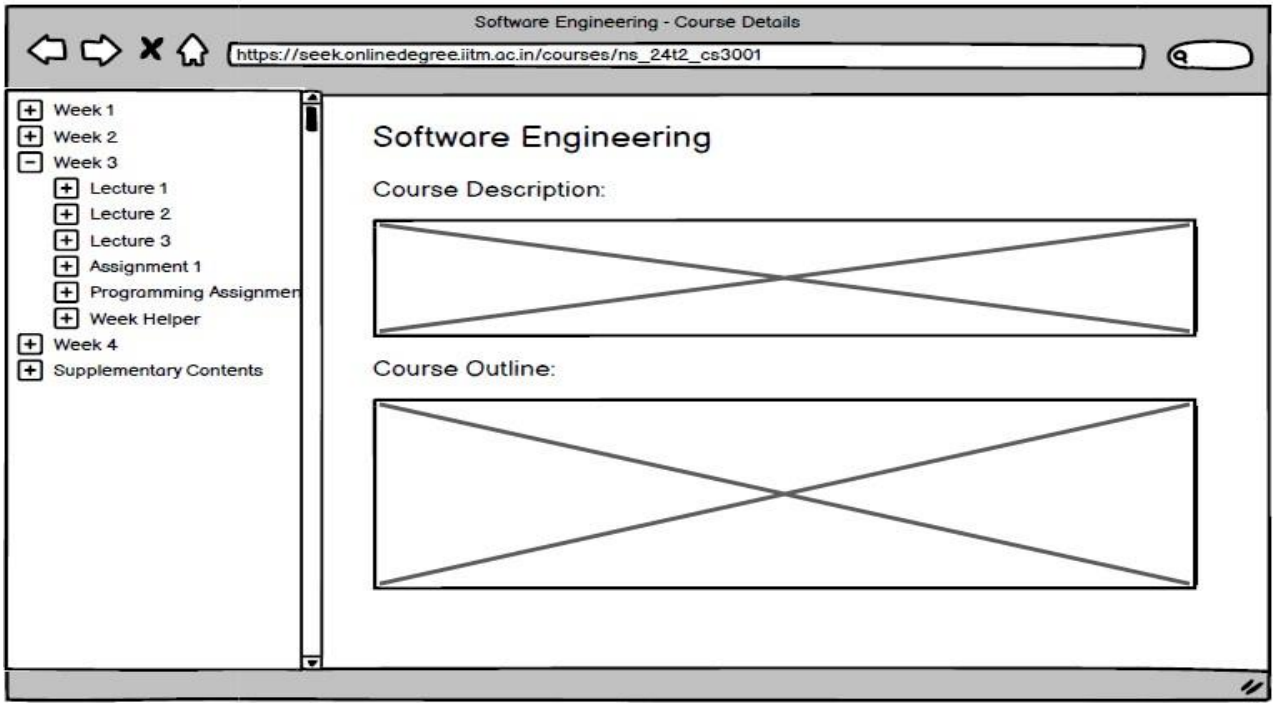

GenAI can create a dynamic study plan based on my course load and deadlines, helping me manage my time effectively.
We can also use AI to create gamified elements like badges, leaderboards, and personalized challenges to increase engagement and motivation.


But it should not spoon-feed us the answers, so that we can improve our analytical skills and learn for ourselves.
And in the IDE, it can suggest code completions, improvements, and syntax corrections to make coding more efficient.

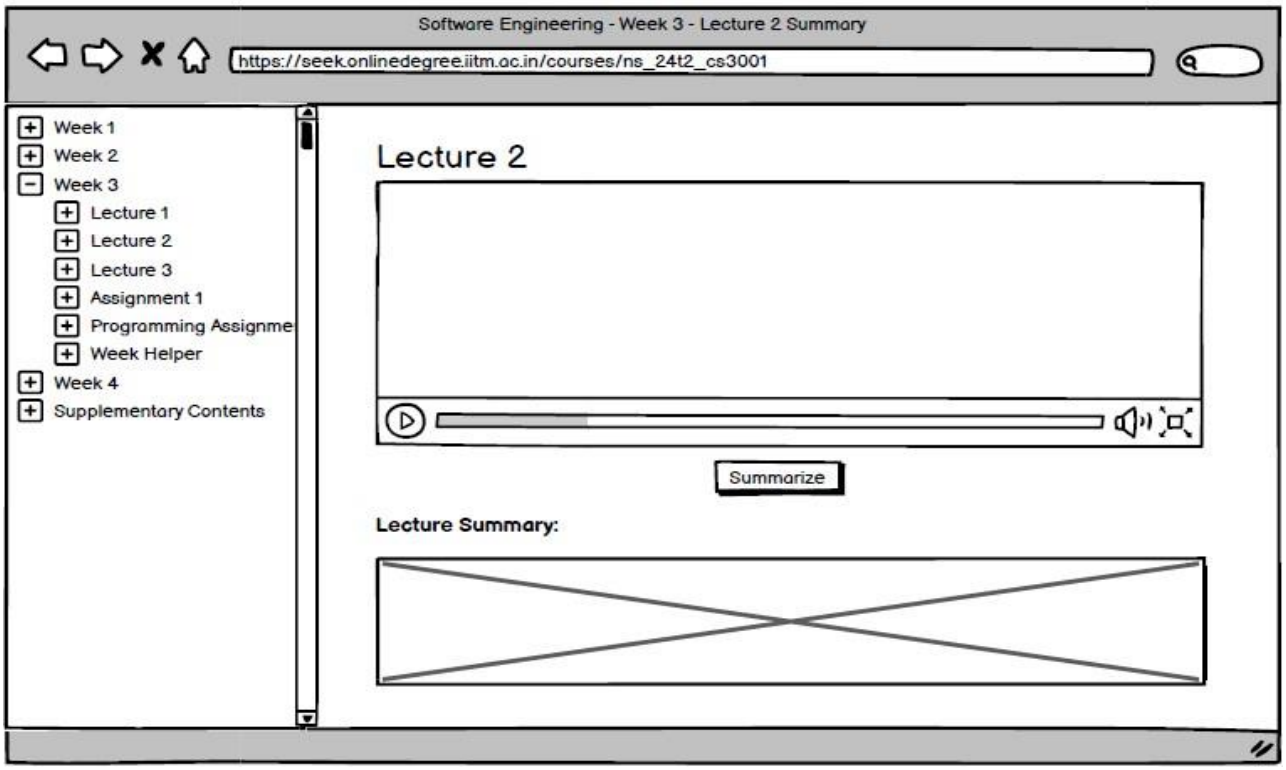

These ideas are fantastic! Let's start working on integrating these features into our portal.
Agreed! Time to make this the best learning platform out there.

3. 2 WireFrames:

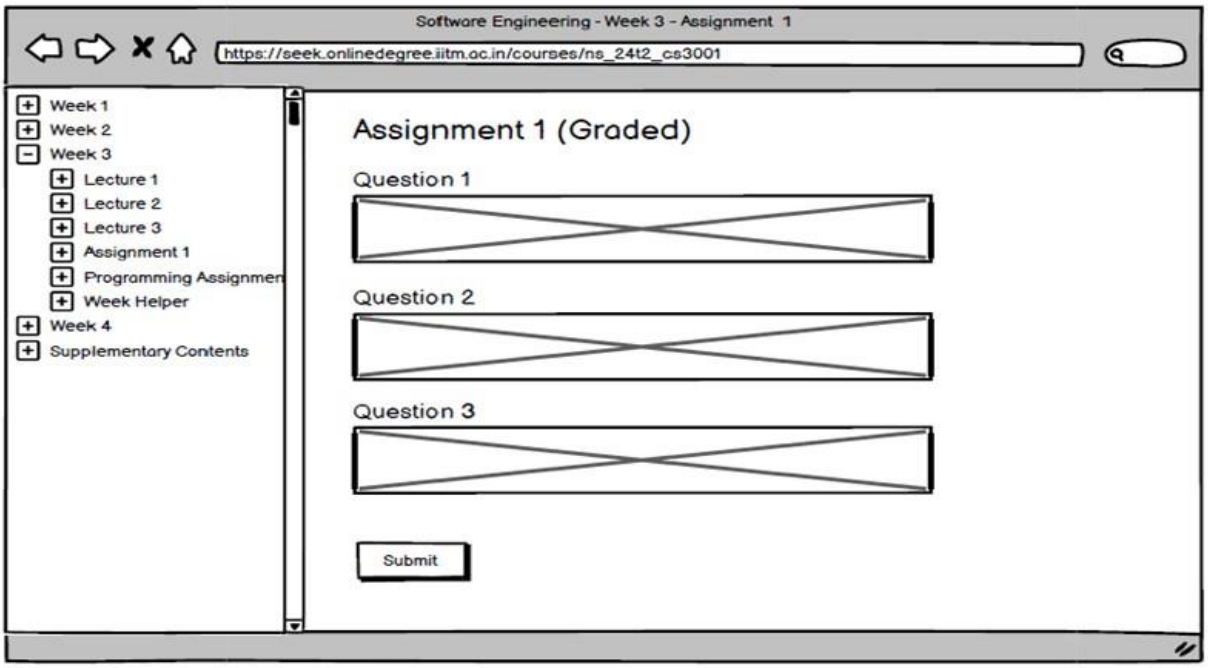
Course Home Page



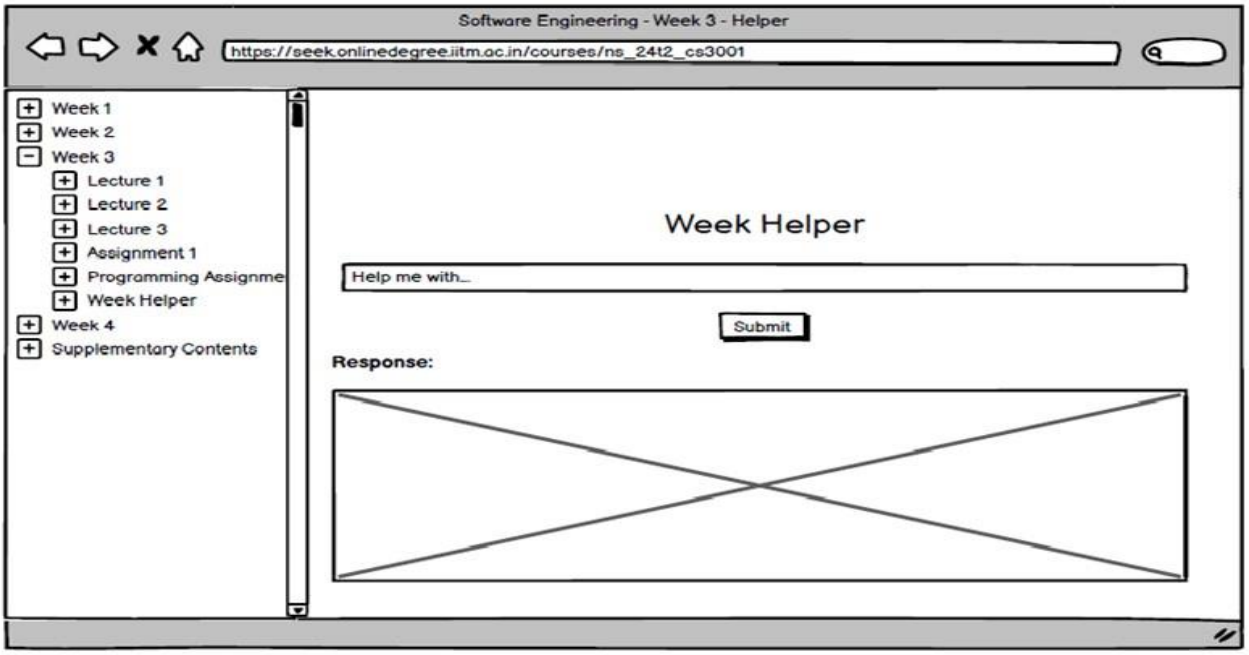
Lecture Summary

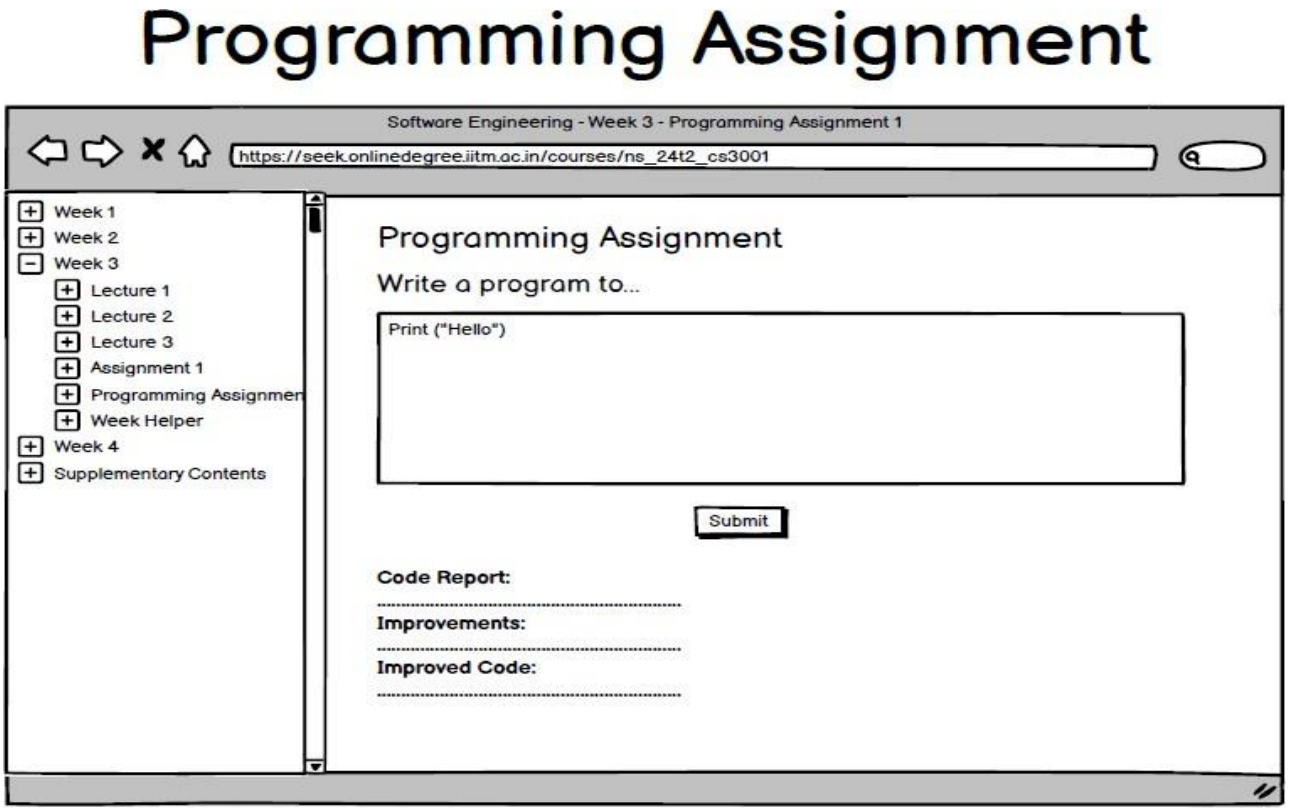


Assignment



Week Helper





3.3 Project Schedule

3.3.1 Sprints Schedule

Sprint 1: June 5 - June 15, 2024

- Define the learner journey map and draft user stories for milestone 1.
- Prepare the milestone 1 report summarizing requirements and user scenarios.

Sprint 2: June 16 - June 30, 2024

- Develop the storyboard and create low-fidelity wireframes.
- Finalize the storyboard and wireframes and prepare the milestone 2 report detailing interface concepts.

Sprint 3: July 1 - July 9, 2024

- Establish the project schedule and design system components.

- Finalize the schedule, com6ponent designs, and software architecture in preparation for the milestone 3 report.

Sprint 4: July 10 - July 25, 2024

- Define and integrate required API endpoints.
- Document all APIs, create necessary endpoints, and submit the YAML file.
- Finalize API documentation for the milestone 4 report.

Sprint 5: July 26 - August 4, 2024

- Design comprehensive test cases covering all project functionalities.
- Develop a test suite to ensure robust testing and prepare the milestone 5 report.

Sprint 6: August 5 - August 17, 2024

- Implement final project details, conduct thorough code reviews, and address any outstanding issues.
- Prepare the project for final submission and complete the milestone 6 report, summarizing implementation details and project outcomes.

3.4 SCRUM Meeting Minutes/Details

3.4.1 SCRUM Board

Jira
Sorted by: Created descending
1–15 of **15** as at: **29/Jun/24 4:04 PM**

T	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Due
	SCRUM-16	FINAL PDF	*Unassigned*	Bindu	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-15	Scrum details	*Unassigned*	Kavya Dwivedi	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-14	Class diag.	*Unassigned*	Santosh Verma	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-13	Design comp.	*Unassigned*	Ojasv Singhal	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-12	Gnt ch. , scrum board	*Unassigned*	Hari Prapan	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-11	Project shed..	*Unassigned*	Hari Prapan	=	TO DO	*Unresolved*	21/Jun/24	21/Jun/24	
	SCRUM-10	Final PDF Report	Hari Prapan	Hari Prapan	=	DONE	Done	21/Jun/24	21/Jun/24	
	SCRUM-9	low-fidelity wireframes	Hari Prapan	Anujj Prem	=	DONE	Done	21/Jun/24	21/Jun/24	
	SCRUM-8	Story Board	Hari Prapan	Pragya Singh	=	DONE	Done	21/Jun/24	21/Jun/24	
	SCRUM-7	Milestone 6	Hari Prapan	Pragya Singh	=	TO DO	*Unresolved*	21/Jun/24	29/Jun/24	10/Aug/24
	SCRUM-6	Milestone 5	Hari Prapan	Kavya Dwivedi	=	TO DO	*Unresolved*	21/Jun/24	29/Jun/24	31/Jul/24
	SCRUM-5	Milestone 4	Hari Prapan	Ojasv Singhal	=	TO DO	*Unresolved*	21/Jun/24	29/Jun/24	15/Jul/24
	SCRUM-4	Milestone 1	Hari Prapan	Hari Prapan	=	DONE	Done	21/Jun/24	21/Jun/24	15/Jun/24
	SCRUM-3	Milestone 3	Hari Prapan	Bindu	=	IN PROGRESS	*Unresolved*	21/Jun/24	21/Jun/24	27/Jun/24
	SCRUM-1	Milestone 2	Mahak Thakre	Mahak Thakre	=	DONE	Done	20/Jun/24	21/Jun/24	21/Jun/24

3.4.2 SCRUM Meeting Schedule

1. **SCRUM Meetings Schedule:** Every Tuesday and Saturday (19:30 - 21:30 PM)

2. **Location:** Google Meet

3. **Attendees:** Kavya Dwivedi, Hari Prapan, Pragya Singh, Ojasv Singhal, Mahak Thakre, Bindu Yadav, Santosh Kumar Verma, Anujj Prem

4. **Sprint 1 SCRUM Meeting Minutes/Details**: The team collaborated and discussed user requirements and potential features for the SEEK portal. Each member shared ideas on how GenAI could enhance the portal. Key milestones and project parts were briefly outlined, and initial responsibilities were assigned. Each member was tasked to write 5 user stories to be completed by the next meeting. Additionally, two members were assigned the task to work on the user journey map. The next meeting's agenda included reviewing progress, discussing challenges, and updating the project timeline.

5. **Sprint 2 SCRUM Meeting Minutes/Details:** The team discussed the user stories submitted through a Google Form, finalizing the best and most unique ones. We also reviewed the user journey maps, provided feedback for changes, and finalized them. The milestone 1 report was reviewed collectively, and the final submission was made.

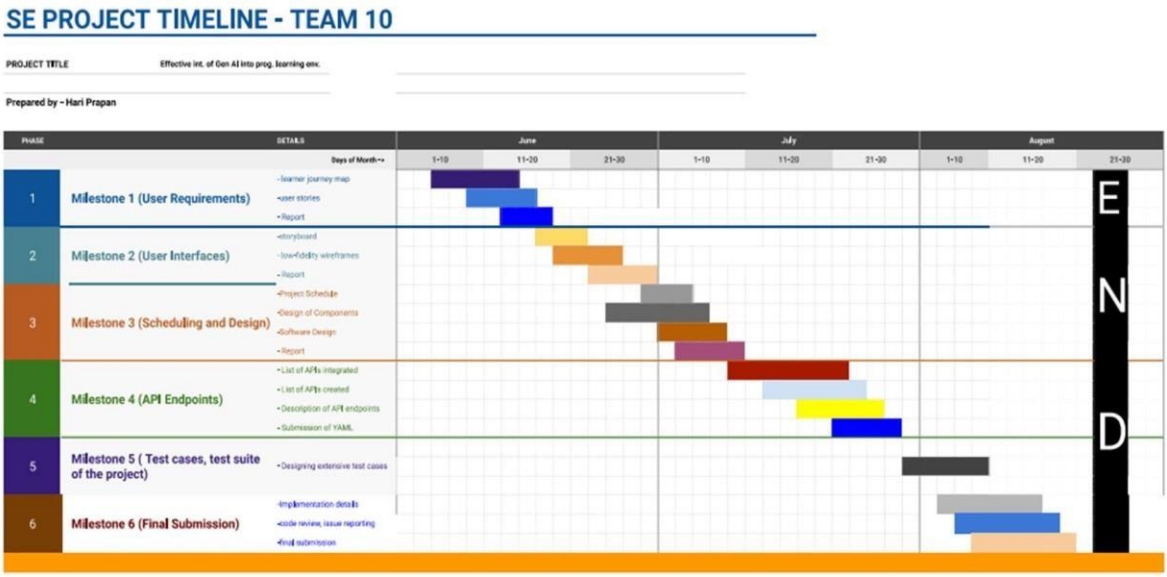
6. **Sprint 3 SCRUM Meeting Minutes/Details:** In Sprint 3, we started working on milestone 2, discussing storyboards and wireframes for the project. Everyone provided inputs and suggestions on how to approach the storyboard. After the discussion, we had a clear picture of our plan for the storyboards and wireframes. Two team members were assigned the task of finalizing the storyboard and wireframe before our next meeting.

7. **Sprint 4 SCRUM Meeting Minutes/Details**: In Sprint 4, we reviewed the storyboards and wireframes, leading to the final milestone 2 submission. We then discussed the project management tool JIRA and its use for tracking project progress and due dates. Following the discussion, we prepared a Gantt chart to schedule future tasks and submissions. Additionally, we designed the components and worked on the class diagram.

8. **Sprint 5 SCRUM Meeting Minutes/Details:** In Sprint 5, we finalized the components and class diagram. The final submission PDF for milestone 3 was prepared and submitted.

3.4.3 Timeline / Gantt Chart

The Gantt chart for the project schedule is displayed below. For a high-resolution version of the chart.



3.5 Design of Components

This application, built with **Flask** and **Vue.js**, provides an interactive learning experience with AI assistance. Below is a breakdown of its components.

I. Backend (Flask)

1. app.py (Main Application)

Routing

- Directs incoming HTTP requests to the correct functions.

API Endpoints

- `/info`: Fetches data for all weeks, including lectures and assignments.
- `/helper`: Processes user queries, using Google Gemini AI to provide answers within the course context.
- `/summary/lectureId`: Generates a summary of a YouTube lecture using Gemini AI.
- `/progassg_suggestions`: Generates a code report leveraging AI for the submitted code for the assignment.
- `/submit_assg`: Provides the score and feedback on the submitted assignment questions using GenAI.

CRUD Endpoints

- Provides endpoints for creating, reading, updating, and deleting weeks,lectures, assignments, questions, and programming assignments.

Database Interaction

- Utilizes SQLAlchemy to manage course data stored in an SQLite database (app.db).

AI Integration

- Leverages Google's generativeai library for AI-powered features like question answering and lecture summarization.

2. models.py (Database Models)

Defines the database structure using SQLAlchemy's ORM:

- **Week**: Represents a week in the course, containing lectures and assignments.
- **Lecture**: Represents a single lecture with a name and YouTube video link.
- **Assignment**: Represents an assignment, potentially with multiple questions.
- **Question**: Represents a multiple-choice question with options and a correct answer.
- **ProgAssg**: Represents a programming assignment with a name, question, and associated week.

3. yt_summary.py (YouTube Transcript Utility)

- `get_transcript(video_url)`: Fetches the transcript of a YouTube video using the youtube_transcript_api library.

II. Frontend (Vue.js)

1. main.js (Entry Point)

- Creates the Vue.js application instance.
- Sets up Vue Router for navigation between views.
- Integrates Vuex store for managing application state.
- Fetches initial data for weeks and programming assignments from the backend.

2. store.js (Vuex Store)

1.State

- Holds the application's data:
  - weeks: Array of week objects fetched from the backend.
  - progassg: Array of programming assignment objects.

2.Mutations

- Functions that directly modify the state (e.g., setWeeks, setProgAssg).

3.Actions

- Asynchronous operations that fetch data and commit mutations (e.g., fetchWeeks, fetchProgAssg).

4. Getters

- Provide computed properties based on the state (e.g., getLectureById, getAssignmentById).

3. Components (Vue.js Single-File Components)

1. **App.vue**: Root component, responsible for overall layout and navigation.
2. **LectureView.vue**: Displays details of a specific lecture.
3. **AssignmentView.vue**: Displays an assignment and its questions.
4. **WeekHelper.vue**: Provides AI assistance for a specific week's content.
5. **HomePage.vue**: The initial view of the application.
6. **ProgAssg.vue**: Displays a programming assignment and likely allows code submission and feedback.
7. **SideBar.vue**: The navigator for the application showing the various week contents.

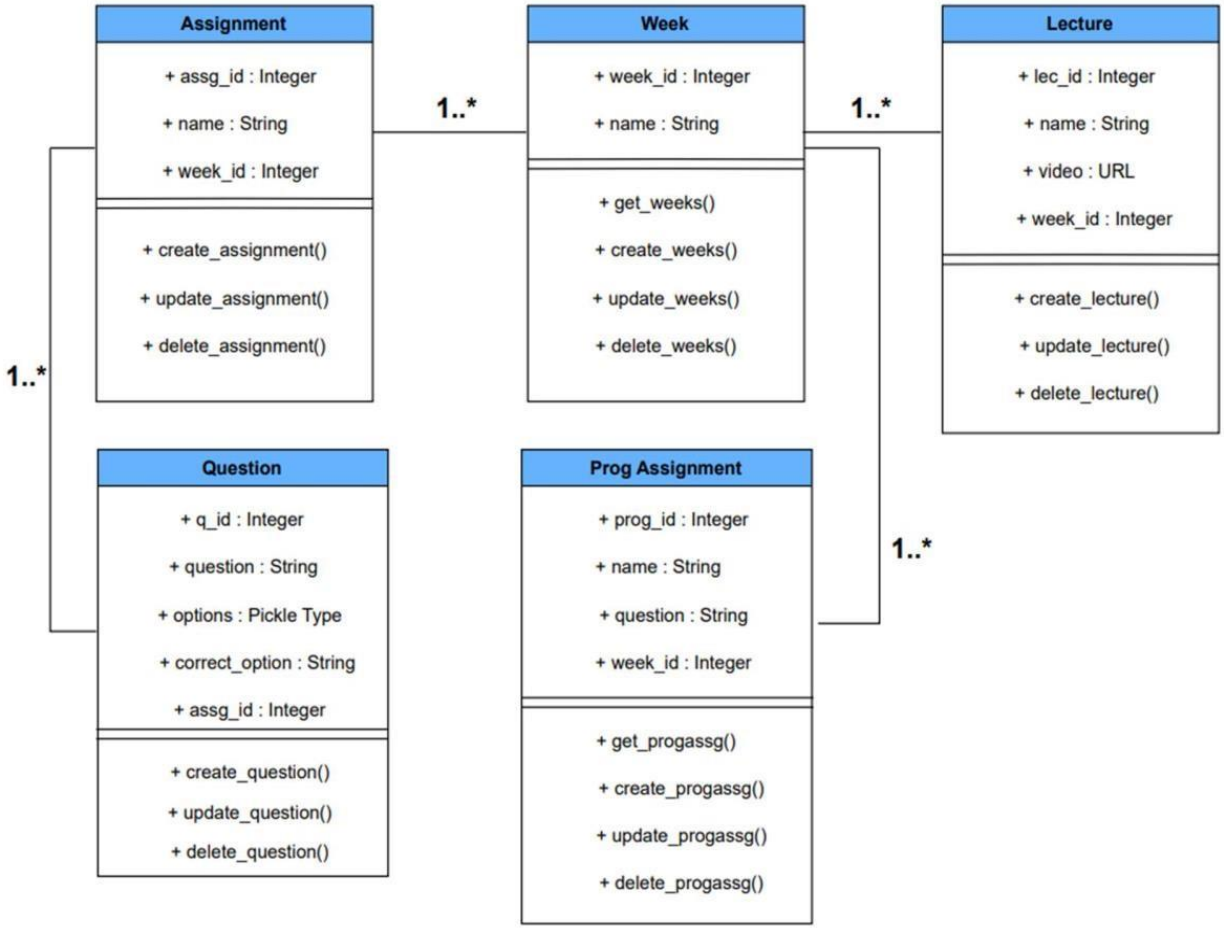
III. Data Flow

1. **User Interaction** : User interacts with the Vue.js frontend.
2. **API Requests** : Frontend sends API requests to the Flask backend.
3. **Backend Processing** : Backend processes requests, interacts with the database, and uses AI models if needed.
4. **Backend Response** : Backend sends responses back to the frontend.
5. **UI Update** : Frontend updates the UI based on the received data.

IV. Key Features

- **AI-Powered Learning :** Google Gemini AI provides contextual help, answers questions, and summarizes lectures.
- **Interactive Learning:** Frontend components display lectures, assignments, and programming exercises.
- **Data Persistence :** Course data is stored in an SQLite database, ensuring data is saved between sessions.

3.6 Class Diagram



4. API Documentation

Task: API Endpoints Documentation

4.1 /api/info

4.1.1 GET

Summary: Get all weeks and their associated data **Responses:**

Code	Description
200	A list of weeks, each containing lectures and assignments
404	Week not found

4.2 /api/helper

4.2.1 POST

Summary: Get AI assistance for a query within a specific week's context **Parameters:**

Name	Located in	Description	Required	Schema
query	body	The user's query	Yes	string
weekId	body	The ID of the week to provide context from	Yes	integer

Responses:

Code	Description
200	AI-generated response in markdown format
404	Week not found

4.3 /api/summary/{lectureId}

4.3.1 GET

Summary: Get a summarized transcript of a YouTube lecture

Parameters:

Name	Located in	Description	Required	Schema
lectureId	path	The ID of the lecture	Yes	integer

Responses:

Code	Description
200	Summarized transcript in markdown format
404	Lecture not found

4.4 /api/week

4.4.1 POST

Summary: Create a new week

Parameters:

Name	Located in	Description	Required	Schema
weekName	body	The name of the new week	Yes	string

Responses:

Code	Description
201	Week created successfully

4.5 /api/week/{id}

4.5.1 PUT

Summary: Update a week

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the week to update	Yes	integer
weekName	body	The updated name of the week	Yes	string

Responses:

Code	Description
200	Week updated successfully
404	Week not found

4.5.2 DELETE

Summary: Delete a week

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the week to delete	Yes	integer

Responses

Code	Description
200	Week deleted successfully
404	Week not found

4.6 /api/lecture

4.6.1 POST

Summary: Create a new lecture

Parameters:

Name	Located in	Description	Required	Schema
Name	body	The name of the lecture	Yes	string
video	body	The YouTube video URL of the lecture	Yes	string
weekId	body	The ID of the week this lecture belongs to	Yes	integer

Responses:

Code	Description
201	Lecture created successfully

4.7 /api/lecture/{id}

4.7.1 PUT

Summary: Update a lecture

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the lecture to update	Yes	integer
name	body	The updated name of the lecture	Yes	string
video	body	The updated YouTube video URL	Yes	string
weekId	body	The updated ID of the week		

Responses:

Code	Description
200	Lecture updated successfully
404	Lecture not found

4.7.2 DELETE

Summary: Delete a lecture

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the lecture to delete	Yes	integer

Responses:

Code	Description
200	Lecture deleted successfully
404	Lecture not found

4.8 /api/assignment

4.8.1 POST

Summary: Create a new assignment

Parameters:

Name	Located in	Description	Required	Schema
name	body	The name of the assignment	Yes	string

Responses:

Code	Description
201	Assignment created successfully

4.9 /api/assignment/{id}

4.9.1 PUT

Summary: Update an assignment

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the assignment to update	Yes	integer
name	body	The updated name of the assignment	Yes	string
weekId	body	The updated ID of the week	Yes	integer

Responses:

Code	Description
200	Assignment updated successfully
404	Assignment not found

4.9.2 DELETE

Summary: Delete an assignment

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the assignment to delete	Yes	integer

Responses:

Code	Description
200	Assignment deleted successfully
404	Assignment not found

4.10 /api/question

4.10.1 POST

Summary: Create a new question

Parameters:

Name	Located in	Description	Required	Schema
question	body	The text of the question	Yes	string
options	body	An array of answer options	Yes	array
correctOption	body	The correct answer option	Yes	string
assignmentId	body	The ID of the assignment this question belongs to	Yes	integer

Responses:

Code	Description
201	Question created successfully

4.11 /api/question/{id}

4.11.1 PUT

Summary: Update a question

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the question to update	Yes	integer
question	body	The updated text of the question	Yes	string
options	body	The updated array of answer options	Yes	array
correctOption	body	The updated correct answer option	Yes	string
assignmentId	body	The updated ID of the assignment	Yes	integer

Responses:

Code	Description
200	Question updated successfully
404	Question not found

4.11.2 DELETE

Summary: Delete a question

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the question to delete	Yes	integer

Responses:

Code	Description
200	Question deleted successfully
404	Question not found

4.12 /api/progassg

4.12.1 GET

Summary: Get all programming assignments

Parameters:

Name	Located in	Description	Required	Schema
None				

Responses:

Code	Description
200	A list of programming assignments

4.12.2 POST

Summary: Create a new programming assignment

Parameters:

Name	Located in	Description	Required	Schema
name	body	The name of the programming assignment	Yes	string
question	body	The description or question for the assignment	Yes	string

weekId	body	The ID of the week this programming assignment belongs to	Yes	integer

Responses:

Code	Description
201	Programming assignment created successfully

4.13 /api/progassg/{id}

4.13.1 PUT

Summary: Update a programming assignment

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the programming assignment to update	Yes	integer
name	body	The updated name of the programming assignment	Yes	string
question	body	The updated description or question for the assignment	Yes	string
weekId	body	The updated ID of the week	Yes	integer

Responses:

Code	Description
200	Programming assignment updated successfully
404	Programming assignment not found

4.13.2 DELETE

Summary: Delete a programming assignment

Parameters:

Name	Located in	Description	Required	Schema
id	path	The ID of the programming assignment to delete	Yes	integer

Responses:

Code	Description

200	Programming assignment deleted successfully
404	Programming assignment not found

4.14 /api/progassg_suggestions

4.14.1 POST

Summary: Get code improvement suggestions and explanations for a programming assignment

Parameters:

Name	Located in	Description	Required	Schema
question	body	The programming assignment question	Yes	string
code	body	The user's code	Yes	string

Responses:

Code	Description
200	AI-generated code improvement suggestions and explanations in markdown format

---

4.15 /api/submit_assg

4.15.1 POST

Summary: Submit answers to an assignment and receive AI-powered feedback and grading

Parameters:

Name	Located in	Description	Required	Schema
questionId1	body	Selected option for question with ID 1	Yes	string
questionId2	body	Selected option for question with ID 2	Yes	string

Responses

Code	Description
200	Assignment submission successful with feedback and grading

5.API Tests

This Markdown file documents tests for an API that manages user data.

The API has endpoints for user retrieval, creation, updating, and deletion.

5.1. /info Endpoints

Test	Description	Example
1.1 test_get_info_successful	Ensures that the /info endpoint returns a 200 status code and contains the expected "weeks" key in the response	If the request is made to /info, the API should return a 200 status code and include a "weeks" key in the response data.
1.1 test_get_info_invalid_endpoint	Verifies that requesting an invalid endpoint returns a non-200 status code.	If the request is made to /info_invalid, the API should return a status code indicating failure (e.g., 404).

5.1.1 Test: test_get_info_successful()

Successful retrieval of all weeks and their associated data

Description: Ensures that the /info endpoint returns a 200 status code and contains the expected "weeks" key in the response.

Test Function:

```
def test_get_info_successful(self,
client):         response =
client.get('/info')         assert
response.status_code == 200         data =
json.loads(response.data)         assert
"weeks" in data
    # Add more assertions to validate the structure of weeks, lectures,
and assignments
```

Expected Output:


```
{
  "status_code": 200,
  "body": [
    {
      "id": 1,
      "name": "Week 1: Introduction to Python",
      "lectures": [
        {
          "id": 1,
          "name": "Lecture 1: Setting Up Your Environment",
          "video": "https://www.youtube.com/watch?v=your-video-id",
          "week_id": 1
        }
      ],
      "assignments": [
        {
          "id": 1,
          "name": "Assignment 1: Basic Python Syntax",
          "week_id": 1
        }
      ]
    }
  ]
}
```


Actual Output:


```
{
  "status_code": 200,
  "body": [
    {
      "id": 1,
      "name": "Week 1: Introduction to Python",
      "lectures": [
        {
          "id": 1,
```

```
          "name": "Lecture 1: Setting Up Your Environment",
          "video": "https://www.youtube.com/watch?v=your-video-id",
"week_id": 1
        }
      ],
      "assignments": [
        {
          "id": 1,
          "name": "Assignment 1: Basic Python Syntax",
          "week_id": 1
        }
      ]
    }
  ]
}
```

Result: Pass

5.1.2 Test: test_get_info_invalid_endpoint()

Invalid endpoint request

Description: Verifies that requesting an invalid endpoint returns a non-200 status code.

Test Function:

```
def test_get_info_invalid_endpoint(self, client):
response = client.get('/invalid_endpoint')
assert response.status_code != 200
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.2. /helper Endpoints

Test	Description	Example
2.1 test_helper_successful	Checks if the /helper endpoint correctly processes a valid query and weekId, returning a 200 status code and a markdown response.	If the request includes a valid query and weekId, the API should return a 200 status code and markdown content.
2.2 test_helper_invalid_week	Ensures that the /helper endpoint returns a 404 status code when an invalid weekId is provided.	If the request includes an invalid weekId, the API should return a 404 status code indicating week not found.
2.3 test_helper_invalid_request	Verifies that the /helper endpoint returns a 400 status code when required parameters are missing.	If the request is missing required parameters, the API should return a 400 status code indicating a bad request.

5.2.1 Test: test_helper_successful()

Successful AI assistance request

Description: Checks if the /helper endpoint correctly processes a valid query and weekId, returning a 200 status code and a markdown response.

Test Function:

```
def test_helper_successful(self, client):
data = {
            "query": "What is a variable in Python?",
            "weekId": 1
        }          response =
client.post('/helper', json=data)
assert response.status_code == 200
        assert isinstance(response.json, str)  # Expecting markdown
response
```

Expected Output:

```
{
  "status_code": 200,
  "body": "```python\nprint('Hello, World!')\n```"
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": "```python\nprint('Hello, World!')\n```"
}
```

Result: Pass

5.2.2 Test: test_helper_invalid_week()

AI assistance request with invalid week

Description: Ensures that the /helper endpoint returns a 404 status code when an invalid weekId is provided.

Test Function:

```
def test_helper_invalid_week(self, client):
        data = {
            "query": "What is a variable in Python?",
            "weekId": 999  # Non-existent week
        }         response =
client.post('/helper', json=data)         assert
response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.2.3 Test: test_helper_invalid_request()

Invalid AI assistance request

Description: Verifies that the /helper endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_helper_invalid_request(self, client):
        data = {
            "query": "What is a variable in Python?"
            # Missing weekId          }
response = client.post('/helper', json=data)
assert response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.3. /summary/{lectureId} Endpoints

Test	Description	Example
3.1 test_get_summary_successful	Checks if the /summary endpoint returns a 200 status code and a markdown summary for a valid lecture ID.	If the request includes a valid lectureId, the API should return a 200 status code and markdown summary content.

3.2 test_get_summary_invalid_lecture	Ensures that the /summary endpoint returns a 404 status code for a non-existent lecture ID.	If the request includes a non-existent lectureId, the API should return a 404 status code indicating lecture not found.

5.3.1 Test: test_get_summary_successful()

Successful retrieval of lecture summary

Description: Checks if the /summary endpoint returns a 200 status code and a markdown summary for a valid lecture ID.

Test Function:

```
def test_get_summary_successful(self, client):         response =
client.get('/summary/1')  # Assuming lecture ID 1 exists
assert response.status_code == 200
        assert isinstance(response.json, str)  # Expecting markdown
summary
```

Expected Outcome:

```
{
  "status_code": 200,
  "body": "This lecture covered the basics of..."
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": "This lecture covered the basics of..."
}
```

Results: Pass

5.3.2 Test: test_get_summary_invalid_lecture()

Retrieval of summary for invalid lecture

Description: Ensures that the /summary endpoint returns a 404 status code for a nonexistent lecture ID.

Test Function:

```
def test_get_summary_invalid_lecture(self, client):
    response = client.get('/summary/999')  # Non-existent lecture ID
assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.4. /week Endpoints

Test	Description	Example
4.1 test_create_week_successful	Verifies that the POST /week endpoint correctly creates a new week and returns a 201 status code with the expected data.	If a valid week creation request is made, the API should return a 201 status code and the created week data.
4.2 test_create_week_invalid_request	Ensures that the POST /week endpoint returns a 400 status code when	If the week creation request is missing required parameters, the API should return a 400

	required parameters are missing.	status code indicating a bad request.
4.3 test_update_week_successful	Checks if the PUT /week/{id} endpoint an correctly updates an API existing week and returns a 200 status code.	If a valid week update request is made for existing week, the should return a 200 status code.
4.4 test_update_week_invalid_id	Verifies that the PUT /week/{id} endpoint returns a 404 status code when trying to update a non-existent week.	If the week update request is made for a nonexistent week, the API should return a 404 status code indicating week not found.
4.5 test_delete_week_successful	Ensures that the DELETE /week/{id} endpoint correctly deletes an existing week and returns a 200 status code.	If a valid delete request is made for an existing week, the API should return a 200 status code.
4.6 test_delete_week_invalid_id	Checks if the DELETE /week/{id} endpoint returns a 404 status code when trying to delete a non-existent week.	If the delete request is made for a non-existent week, the API should return a 404 status code indicating week not found.

5.4.1 Test: test_create_week_successful()

Successful creation of a new week

Description: Verifies that the POST /week endpoint correctly creates a new week and returns a
201 status code with the expected data.

Test Function:

```
def test_create_week_successful(self, client):
data = {
            "weekName": "Week 2: Data Types and
Variables"         }
         response = client.post('/week',
 json=data)         assert response.status_code
== 201         data =
json.loads(response.data)         assert "id"
in data
        assert data["name"] == "Week 2: Data Types and Variables"
```

Expected Outcome:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Week 2: Data Types and Variables",
    "lectures": [],
    "assignments": []
  }
}
```

Actual Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Week 2: Data Types and Variables",
    "lectures": [],
    "assignments": []
  }
}
```

Results: Pass

5.4.2 Test: test_create_week_invalid_request()

Invalid week creation request

Description: Ensures that the POST /week endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_create_week_invalid_request(self, client):
data = {}  # Missing required weekName
response = client.post('/week', json=data)
assert response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.4.3 Test: test_update_week_successful()

Successful update of an existing week

Description: Checks if the PUT /week/{id} endpoint correctly updates an existing week and returns a 200 status code.

Test Function:

```
def test_update_week_successful(self, client):
data = {
            "weekName": "Updated Week 1: Advanced Introduction to
Programming"         }         response = client.put('/week/1',
json=data)  # Assuming week ID 1 exists
        assert response.status_code == 200
```

Expected Outcome:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Week 1: Advanced Introduction to Programming",
    "lectures": [],
    "assignments": []
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Week 1: Advanced Introduction to Programming",
    "lectures": [],
    "assignments": []
  }
}
```

Results: Pass

5.4.4 Test: test_update_week_invalid_id()

Update request for non-existent week

Description: Verifies that the PUT /week/{id} endpoint returns a 404 status code when trying to update a non-existent week.

Test Function:

```
def test_update_week_invalid_id(self, client):
data = {
        "weekName": "Updated Non-existent Week"
    }     response = client.put('/week/999', json=data)  # Nonexistent
week ID     assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.4.5 Test: test_delete_week_successful()

Successful deletion of a week

Description: Ensures that the DELETE /week/{id} endpoint correctly deletes an existing week and returns a 200 status code.

Test Function:

```
def test_delete_week_successful(self, client):
    response = client.delete('/week/1')  # Assuming week ID 1 exists
assert response.status_code == 200
```

Expected Outcome:

```
{
  "status_code": 200
}
```

Actual Output:

```
{
  "status_code": 200
}
```

Results: Pass

5.4.6 Test: test_delete_week_invalid_id()

Deletion request for non-existent week

Description: Checks if the DELETE /week/{id} endpoint returns a 404 status code when trying to delete a non-existent week.

Test Function:

```
def test_delete_week_invalid_id(self, client):
    response = client.delete('/week/999')  # Non-existent week ID
assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.5. /lecture Endpoints

Test	Description	Example
5.1 test_create_lecture_successful	Verifies that the POST /lecture endpoint correctly creates a new lecture and returns a 201 status code with the expected data.	If a valid lecture creation request is made, the API should return a 201 status code and the created lecture data.
5.2 test_create_lecture_invalid_request	Ensures that the POST /lecture endpoint returns a 400 status code when required parameters are missing.	If the lecture creation request is missing required parameters, the API should return a 400 status code indicating a bad request.

5.3 test_update_lecture_successful	Checks if the PUT /lecture/{id} endpoint correctly updates an existing lecture and returns a 200 status code.	If a valid lecture update request is made for an existing lecture, the API should return a 200 status code.
5.4 test_update_lecture_invalid_id	Verifies that the PUT /lecture/{id} endpoint returns a 404 status code when trying to update a non-existent lecture.	If the lecture update request is made for a non-existent lecture, the API should return a 404 status code indicating lecture not found.
5.5 test_delete_lecture_successful	Ensures that the DELETE /lecture/{id} endpoint correctly deletes an existing lecture and returns a 200 status code.	If a valid delete request is made for an existing lecture, the API should return a 200 status code.
5.6 test_delete_lecture_invalid_id	Checks if the DELETE /lecture/{id} endpoint returns a 404 status code when trying to delete a non-existent lecture.	If the delete request is made for a non-existent lecture, the API should return a 404 status code indicating lecture not found.

5.5.1 Test: test_create_lecture_successful()

Successful creation of a new lecture

Description: Verifies that the POST /lecture endpoint correctly creates a new lecture and returns a 201 status code with the expected data.

Test Function:

```
def test_create_lecture_successful(self, client):
data = {
      "name": "Lecture 1: Introduction to Variables",
```

```
        "video": "https://www.youtube.com/watch?v=example",
        "weekId": 1
    }     response = client.post('/lecture', json=data)
assert response.status_code == 201     data =
json.loads(response.data)     assert "id" in data     assert
data["name"] == "Lecture 1: Introduction to Variables"
```

Expected Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Lecture 1: Introduction to Variables",
    "video": "https://www.youtube.com/watch?v=example",
    "week_id": 1
  }
}
```

Actual Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Lecture 1: Introduction to Variables",
    "video": "https://www.youtube.com/watch?v=example",
    "week_id": 1
  }
}
```

Result: Pass

5.5.2 Test: test_create_lecture_invalid_request()

Invalid lecture creation request

Description: Ensures that the POST /lecture endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
    def test_create_lecture_invalid_request(self, client):
        data = {
            "name": "Invalid Lecture",
            "video": "https://www.youtube.com/watch?v=example"
            # Missing weekId         }
response = client.post('/lecture', json=data)
assert response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.5.3 Test: test_update_lecture_successful()

Successful update of an existing lecture

Description: Checks if the PUT /lecture/{id} endpoint correctly updates an existing lecture and returns a 200 status code.

Test Function:

```
 def test_update_lecture_successful(self, client):
        data = {
            "name": "Updated Lecture 1: Advanced Variables",
            "video": "https://www.youtube.com/watch?v=new_example",
            "weekId": 1
        }         response = client.put('/lecture/1', json=data)
# Assuming
lecture ID 1 exists
        assert response.status_code == 200
```

Expected Outcome:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Lecture 1: Advanced Variables",
    "video": "https://www.youtube.com/watch?v=new_example",
    "week_id": 1
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Lecture 1: Advanced Variables",
    "video": "https://www.youtube.com/watch?v=new_example",
    "week_id": 1
  }
}
```

Results: Pass

5.5.4 Test: test_update_lecture_invalid_id()

Update request for non-existent lecture

Description: Verifies that the PUT /lecture/{id} endpoint returns a 404 status code when trying to update a non-existent lecture.

Test Function:

```
    def test_update_lecture_invalid_id(self, client):
data = {
            "name": "Updated Non-existent Lecture",
            "video": "https://www.youtube.com/watch?v=new_example",
            "weekId": 1
```

```
        }         response = client.put('/lecture/999', json=data)
# Non-existent lecture ID         assert
response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.5.5 Test: test_delete_lecture_successful()

Successful deletion of a lecture

Description: Ensures that the DELETE /lecture/{id} endpoint correctly deletes an existing lecture and returns a 200 status code.

Test Function:

```
def test_delete_lecture_successful(self, client):
    response = client.delete('/lecture/1')  # Assuming lecture ID 1
exists     assert response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200
}
```

Actual Output:

```
{
  "status_code": 200
```

}

Result: Pass

5.5.6 Test: test_delete_lecture_invalid_id()

Deletion request for non-existent lecture

Description: Checks if the DELETE /lecture/{id} endpoint returns a 404 status code when trying to delete a non-existent lecture.

Test Function:

```
def test_delete_lecture_invalid_id(self, client):
    response = client.delete('/lecture/999')  # Non-existent lecture
ID     assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.6. /assignment Endpoints

Test	Description	Example

6.1 test_create_assignment_successful	Verifies that the POST /assignment endpoint correctly creates a new assignment and returns a 201 status code with the expected data.	If a valid assignment creation request is made, the API should return a 201 status code and the created assignment data.
6.2 test_create_assignment_invalid_request	Ensures that the POST /assignment endpoint returns a 400 status code when required parameters are missing.	If the assignment creation request is missing required parameters, the API should return a 400 status code indicating a bad request.
6.3 test_update_assignment_successful	Checks if the PUT /assignment/{id} endpoint correctly updates an existing assignment and returns a 200 status code.	If a valid assignment update request is made for an existing assignment, the API should return a 200 status code.
6.4 test_update_assignment_invalid_id	Verifies that the PUT /assignment/{id} endpoint returns a 404 status code when trying to update a nonexistent assignment.	If the assignment update request is made for a nonexistent assignment, the API should return a 404 status code indicating assignment not found.
6.5 test_delete_assignment_successful	Ensures that the DELETE /assignment/{id} endpoint correctly deletes an existing assignment and returns a 200 status code.	If a valid delete request is made for an existing assignment, the API should return a 200 status code.

6.6 test_delete_assignment_invalid_id	Checks if the DELETE /assignment/{id} endpoint returns a 404 status code when trying to delete a nonexistent assignment.	If the delete request is made for a nonexistent assignment, the API should return a 404 status code indicating assignment not found.

5.6.1 Test: test_create_assignment_successful()

Successful creation of a new assignment

Description: Verifies that the POST /assignment endpoint correctly creates a new assignment and returns a 201 status code with the expected data.

Test Function:

```
 def test_create_assignment_successful(self, client):
data = {
        "name": "Assignment 1: Python Basics",
        "weekId": 1
    }    response = client.post('/assignment',
 json=data)    assert response.status_code ==
 201    data = json.loads(response.data)
 assert "id" in data
    assert data["name"] == "Assignment 1: Python Basics"
```

Expected Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Assignment 1: Python Basics",
    "week_id": 1
  }
}
```

Actual Output:

```
{
```

```
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Assignment 1: Python Basics",
    "week_id": 1
  }
}
```

Result: Pass

5.6.2 Test: test_create_assignment_invalid_request()

Invalid assignment creation request

Description: Ensures that the POST /assignment endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_create_assignment_invalid_request(self, client):
    data = {
         "name": "Invalid Assignment"
          # Missing weekId
    }    response = client.post('/assignment',
json=data)    assert response.status_code ==
400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.6.3 Test: test_update_assignment_successful()

Successful update of an existing assignment

Description: Checks if the PUT /assignment/{id} endpoint correctly updates an existing assignment and returns a 200 status code.

Test Function:

```
 def test_update_assignment_successful(self, client):
data = {
        "name": "Updated Assignment 1: Advanced Python Basics",
        "weekId": 1
    }    response = client.put('/assignment/1', json=data)  #
 Assuming
 assignment ID 1 exists
    assert response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Assignment 1: Advanced Python Basics",
    "week_id": 1
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Assignment 1: Advanced Python Basics",
    "week_id": 1
  }
}
```

Result: Pass

5.6.4 Test: test_update_assignment_invalid_id()

Update request for non-existent assignment

Description: Verifies that the PUT /assignment/{id} endpoint returns a 404 status code when trying to update a non-existent assignment.

Test Function:

```
def test_update_assignment_invalid_id(self, client):
data = {
       "name": "Updated Non-existent Assignment",
        "weekId": 1
   }     response = client.put('/assignment/999', json=data)   #
Non-existent assignment ID    assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.6.5 Test: test_delete_assignment_successful()

Successful deletion of an assignment

Description: Ensures that the DELETE /assignment/{id} endpoint correctly deletes an existing assignment and returns a 200 status code.

Test Function:

```
def test_delete_assignment_successful(self, client):
    response = client.delete('/assignment/1')  # Assuming assignment
ID
1 exists      assert
response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200
}
```

Actual Output:

```
{
  "status_code": 200
}
```

Result: Pass

5.6.6 Test: test_delete_assignment_invalid_id()

Deletion request for non-existent assignment

Description: Checks if the DELETE /assignment/{id} endpoint returns a 404 status code when trying to delete a non-existent assignment.

Test Function:

```
def test_delete_assignment_invalid_id(self, client):
    response = client.delete('/assignment/999')  # Non-existent
assignment ID
    assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.7. /question Endpoints

Test	Description	Example
7.1 test_create_question_successful	Verifies that the POST /question endpoint correctly creates a new question and returns a 201 status code with the expected data.	If a valid question creation request is made, the API should return a 201 status code and the created question data
7.2 test_create_question_invalid_request	Ensures that the POST /question endpoint returns a 400 status code when required parameters are missing.	If the question creation request is missing required parameters, the API should return a 400 status code indicating a bad request.
7.3 test_update_question_successful	Checks if the PUT /question/{id} endpoint correctly updates an existing question and returns a 200 status code.	If a valid question update request is made for an existing question, the API should return a 200 status code.
7.4 test_update_question_invalid_id	Verifies that the PUT /question/{id} endpoint returns a 404 status code when trying to update a nonexistent question.	If the question update request is made for a non-existent question, the API should return a 404 status code indicating question not found.
7.5 test_delete_question_successful	Ensures that the DELETE /question/{id} endpoint correctly deletes an existing question and returns a 200 status code.	If a valid delete request is made for an existing question, the API should return a 200 status code.
7.6 test_delete_question_invalid_id	Checks if the DELETE /question/{id} endpoint returns a 404 status code when trying to delete a nonexistent question.	If the delete request is made for a nonexistent question, the API should return a 404 status code indicating question not found.

5.7.1 Test: test_create_question_successful()

Successful creation of a new question

Description: Verifies that the POST /question endpoint correctly creates a new question and returns a 201 status code with the expected data.

Test Function:

```
    def test_create_question_successful(self, client):
data = {
            "question": "What is the correct syntax to print 'Hello,
World!' in Python?",
            "options": ["print('Hello, World!')", "console.log('Hello,
World!')", "System.out.println('Hello, World!')"],
            "correctOption": "print('Hello, World!')",
            "assignmentId": 1
        }         response =
client.post('/question', json=data)         assert
response.status_code == 201         data =
json.loads(response.data)         assert "id" in
data
        assert data["question"] == "What is the correct syntax to
print 'Hello, World!' in Python?"
```

Expected Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "question": "What is the correct syntax to print 'Hello, World!' in
Python?",
    "options": ["print('Hello, World!')", "console.log('Hello,
World!')",
"System.out.println('Hello, World!')"],
    "correctOption": "print('Hello, World!')",
    "assignmentId": 1
  }
}
```

Actual Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "question": "What is the correct syntax to print 'Hello, World!' in
Python?",
    "options": ["print('Hello, World!')", "console.log('Hello,
World!')",
"System.out.println('Hello, World!')"],
    "correctOption": "print('Hello, World!')",
    "assignmentId": 1
  }
}
```

Result: Pass

5.7.2 Test: test_create_question_invalid_request()

Invalid question creation request

Description: Ensures that the POST /question endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_create_question_invalid_request(self, client):
        data = {
            "question": "Invalid Question",
            "options": ["Option 1", "Option 2"]
            # Missing correctOption and assignmentId
        }         response =
client.post('/question', json=data)         assert
response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
```

```
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.7.3 Test: test_update_question_successful()

Successful update of an existing question

Description: Checks if the PUT /question/{id} endpoint correctly updates an existing question and returns a 200 status code.

Test Function:

```
  def test_update_question_successful(self, client):
 data = {
            "question": "Updated: What is the correct syntax to print
'Hello, Python!' in Python?",
            "options": ["print('Hello, Python!')",
"console.log('Hello,
Python!')", "System.out.println('Hello, Python!')"],
            "correctOption": "print('Hello, Python!')",
            "assignmentId": 1
        }         response = client.put('/question/1',
json=data)  # Assuming question ID 1 exists
        assert response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "question": "Updated: What is the correct syntax to print 'Hello,
Python!' in Python?",
    "options": ["print('Hello, Python!')", "console.log('Hello,
Python!')",
```

```
"System.out.println('Hello, Python!')"],
    "correctOption": "print('Hello, Python!')",
    "assignmentId": 1
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "question": "Updated: What is the correct syntax to print 'Hello,
Python!' in Python?",
    "options": ["print('Hello, Python!')", "console.log('Hello,
Python!')",
"System.out.println('Hello, Python!')"],
    "correctOption": "print('Hello, Python!')",
    "assignmentId": 1
  }
}
```

Result: Pass

5.7.4 Test: test_update_question_invalid_id()

Update request for non-existent question

Description: Verifies that the PUT /question/{id} endpoint returns a 404 status code when trying to update a non-existent question.

Test Function:

```
def test_update_question_invalid_id(self, client):
        data = {
            "question": "Updated Non-existent Question",
            "options": ["Option 1", "Option 2"],
            "correctOption": "Option 1",
            "assignmentId": 1
        }         response = client.put('/question/999', json=data)
```

```
# Non-existent question ID
assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.7.5 Test: test_delete_question_successful()

Successful deletion of a question

Description: Ensures that the DELETE /question/{id} endpoint correctly deletes an existing question and returns a 200 status code.

Test Function:

```
def test_delete_question_successful(self, client):
        response = client.delete('/question/1')  # Assuming question
ID
1 exists          assert
response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200
}
```

Actual Output:

```
{
```

```
  "status_code": 200
}
```

Result: Pass

5.7.6 Test: test_delete_question_invalid_id()

Deletion request for non-existent question

Description: Checks if the DELETE /question/{id} endpoint returns a 404 status code when trying to delete a non-existent question.

Test Function:

```
def test_delete_question_invalid_id(self, client):
        response = client.delete('/question/999')  # Non-existent
question ID
        assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
```

Results: Pass

5.8. /progassg Endpoints

Test	Description	Example

8.1 test_get_programming_assignments_successful	Ensures that the GET /progassg endpoint returns a 200 status code and a list of programming assignments.	If the request is made to /progassg, the API should return a 200 status code and a list of programming assignments.
8.2 test_get_programming_assignments_invalid_endpoi n t	Verifies that requesting an invalid endpoint returns a non-200 status code.	If the request is made to /progassg_invalid , the API should return a status code indicating failure (e.g., 404).
8.3 test_create_programming_assignment_successful	Verifies that the POST /progassg endpoint correctly creates a new programming assignment and returns a 201 status code with the expected data.	If a valid programming assignment creation request is made, the API should return a 201 status code and the created assignment data.
8.4 test_create_programming_assignment_invalid_reque st	Ensures that the POST /progassg endpoint returns a 400 status code when required parameters are missing.	If the programming assignment creation request is missing required parameters, the API should return a 400 status code indicating a bad request.

8.5 test_update_programming_assignment_successful	Checks if the PUT /progassg/{id} endpoint correctly updates an existing programming assignment and returns a 200 status code.	If a valid programming assignment update request is made for an existing assignment, the API should return a 200 status code.
8.6 test_update_programming_assignment_invalid_id	Verifies that the PUT /progassg/{id} endpoint returns a 404 status code when trying to update a non-existent programming assignment.	If the programming assignment update request is made for a nonexistent assignment, the API should return a 404 status code indicating assignment not found.
8.7 test_delete_programming_assignment_successful	Ensures that the DELETE /progassg/{id} endpoint correctly deletes an existing programming assignment and returns a 200 status code.	If a valid delete request is made for an existing programming assignment, the API should return a 200 status code.
8.8 test_delete_programming_assignment_invalid_id	Checks if the DELETE /progassg/{id} endpoint returns a 404 status code when trying to delete a non-existent	If the delete request is made for a nonexistent programming assignment, the API should return a 404 status code indicating assignment not found.

	programming assignment.	

5.8.1 Test: test_get_programming_assignments_successful()

Successful retrieval of all programming assignments

Description: Ensures that the GET /progassg endpoint returns a 200 status code and a list of programming assignments.

Test Function:

```
def test_get_programming_assignments_successful(self, client):
        response = client.get('/progassg')
assert response.status_code == 200
data = json.loads(response.data)
assert isinstance(data, list)
        # Add more assertions to validate the structure of programming
assignments
```

Expected Output:

```
{
  "status_code": 200,
  "body": [
    {
      "id": 1,
      "name": "Programming Assignment 1: Calculator",
      "question": "Create a simple calculator program in Python.",
      "week_id": 1
    }
  ]
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": [
    {
      "id": 1,
```

```
      "name": "Programming Assignment 1: Calculator",
      "question": "Create a simple calculator program in Python.",
      "week_id": 1
    }
  ]
}
```

Result: Pass

5.8.2 Test: test_get_programming_assignments_invalid_endpoint()

Invalid endpoint request for programming assignments

Description: Verifies that requesting an invalid endpoint returns a non-200 status code.

Test Function:

```
 def test_get_programming_assignments_invalid_endpoint(self,
client):
        response = client.get('/invalid_endpoint')
assert response.status_code != 200
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.8.3 Test: test_create_programming_assignment_successful()

Successful creation of a new programming assignment

Description: Verifies that the POST /progassg endpoint correctly creates a new programming assignment and returns a 201 status code with the expected data.

Test Function:

```
def test_create_programming_assignment_successful(self, client):
 data = {
            "name": "Programming Assignment 1: Calculator",
            "question": "Create a simple calculator program in
Python.",
            "weekId": 1          }
response = client.post('/progassg', json=data)
assert response.status_code == 201         data =
json.loads(response.data)         assert "id" in
data
        assert data["name"] == "Programming Assignment 1: Calculator"
```

Expected Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Programming Assignment 1: Calculator",
    "question": "Create a simple calculator program in Python.",
    "week_id": 1
  }
}
```

Actual Output:

```
{
  "status_code": 201,
  "body": {
    "id": 2,
    "name": "Programming Assignment 1: Calculator",
    "question": "Create a simple calculator program in Python.",
    "week_id": 1
  }
}
```

Result: Pass

5.8.4 Test: test_create_programming_assignment_invalid_request()

Invalid programming assignment creation request

Description: Ensures that the POST /progassg endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_create_programming_assignment_invalid_request(self,
client):
        data = {
            "name": "Invalid Programming Assignment",
            "question": "Create something."
            # Missing weekId         }
response = client.post('/progassg', json=data)
assert response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.8.5 Test: test_update_programming_assignment_successful()

Successful update of an existing programming assignment

Description: Checks if the PUT /progassg/{id} endpoint correctly updates an existing programming assignment and returns a 200 status code.

Test Function:

```
def test_update_programming_assignment_successful(self,
client):         data = {
            "name": "Updated Programming Assignment 1: Advanced
Calculator",
            "question": "Create an advanced calculator program in
Python with scientific functions.",
            "weekId": 1
        }         response = client.put('/progassg/1',
json=data)  # Assuming programming assignment ID 1 exists
        assert response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Programming Assignment 1: Advanced Calculator",
"question": "Create an advanced calculator program in Python with
scientific functions.",
    "week_id": 1
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "id": 1,
    "name": "Updated Programming Assignment 1: Advanced Calculator",
"question": "Create an advanced calculator program in Python with
scientific functions.",
    "week_id": 1
  }
}
```

Result: Pass

5.8.6 Test: test_update_programming_assignment_invalid_id()

Update request for non-existent programming assignment

Description: Verifies that the PUT /progassg/{id} endpoint returns a 404 status code when trying to update a non-existent programming assignment.

Test Function:

```
 def test_update_programming_assignment_invalid_id(self,
client):         data = {
            "name": "Updated Non-existent Programming Assignment",
            "question": "Do something impossible.",
            "weekId": 1
        }         response = client.put('/progassg/999', json=data)
# Non-existent programming assignment ID
assert response.status_code == 404
```

Expected Output:

```
{
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.8.7 Test: test_delete_programming_assignment_successful()

Successful deletion of a programming assignment

Description: Ensures that the DELETE /progassg/{id} endpoint correctly deletes an existing programming assignment and returns a 200 status code.

Test Function:

```
def test_delete_programming_assignment_successful(self, client):
        response = client.delete('/progassg/1')  # Assuming
programming
assignment ID 1 exists
        assert response.status_code == 200
```

Expected Output:

```
{
  "status_code": 200
}
```

Actual Output:

```
{
  "status_code": 200
}
```

Result: Pass

5.8.8 Test: test_delete_programming_assignment_invalid_id()

Deletion request for non-existent programming assignment

Description: Checks if the DELETE /progassg/{id} endpoint returns a 404 status code when trying to delete a non-existent programming assignment.

Test Function:

```
def test_delete_programming_assignment_invalid_id(self, client):
response = client.delete('/progassg/999')  # Non-existent
programming assignment ID
        assert response.status_code == 404
```

Expected Output:

```
{
```

```
  "status_code": 404
}
```

Actual Output:

```
{
  "status_code": 404
}
```

Result: Pass

5.9. /progassg_suggestions Endpoints

Test	Description	Example
9.1 test_get_programming_suggestions_successful	Checks if the POST /progassg_suggestions endpoint correctly processes a valid request and returns a 200 status code with markdown suggestions.	If a valid request is made to /progassg_suggestions, the API should return a 200 status code and markdown suggestions.
9.2 test_get_programming_suggestions_invalid_request	Ensures that the POST /progassg_suggestions endpoint returns a 400 status code when required parameters are missing.	If the request to /progassg_suggestions is missing required parameters, the API should return a 400 status code indicating a bad request.

5.9.1 Test: test_get_programming_suggestions_successful()

Successful retrieval of programming suggestions

Description: Checks if the POST /progassg_suggestions endpoint correctly processes a valid request and returns a 200 status code with markdown suggestions.

Test Function:

```
def test_get_programming_suggestions_successful(self, client):
data = {
            "question": "Create a simple calculator program in
Python.",
            "code": "def add(a, b):\n return a + b\n\nprint(add(5, 3))"
        }         response =
client.post('/progassg_suggestions', json=data)         assert
response.status_code == 200         assert
isinstance(response.json, str)  # Expecting markdown response
with suggestions
```

Expected Output:

```
{
  "status_code": 200,
  "body": "```\n# Your code is good, but you can improve it
by...\n```"
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": "```\n# Your code is good, but you can improve it
by...\n```"
}
```

Result: Pass

5.9.2 Test: test_get_programming_suggestions_invalid_request()

Invalid request for programming suggestions

Page |

Description: Ensures that the POST /progassg_suggestions endpoint returns a 400 status code when required parameters are missing.

Test Function:

```
def test_get_programming_suggestions_invalid_request(self,
client):
```

```
        data = {
            "question": "Create a simple calculator program in Python."
            # Missing code          }
response =
client.post('/progassg_suggestions',
json=data)         assert response.status_code
== 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

5.10. /submit_assg Endpoints

Test	Description	Example
10.1 test_submit_assignment_successful	Verifies that the POST /submit_assg endpoint correctly processes a valid assignment submission and returns a 200 status code with expected feedback.	If a valid assignment submission is made to /submit_assg, the API should return a 200 status code with feedback.
10.2 test_submit_assignment_invalid_request	Ensures that the POST /submit_assg endpoint returns a 400 status code when an invalid submission is made.	If an invalid assignment submission is made to /submit_assg, the

		API should return a 400 status code with feedback.

5.10.1 Test: test_submit_assignment_successful()

Successful submission of an assignment

Description: Verifies that the POST /submit_assg endpoint correctly processes a valid assignment submission and returns a 200 status code with expected feedback.

Test Function:

```
def test_submit_assignment_successful(self, client):
 data = {
            "questionId1": "Option A",
            "questionId2": "Option B"
        }         response =
client.post('/submit_assg', json=data)
assert response.status_code == 200         data =
json.loads(response.data)         assert "message"
in data         assert "status" in data
assert "score" in data
```

Expected Output:

```
{
  "status_code": 200,
  "body": {
    "message": "You did well on understanding...",
    "status": "success",
    "score": 80
  }
}
```

Actual Output:

```
{
  "status_code": 200,
  "body": {
    "message": "You did well on understanding...",
```

```
    "status": "success",
    "score": 80
  }
}
```

Result: Pass

5.10.2 Test: test_submit_assignment_invalid_request()

Invalid assignment submission request

Description: Ensures that the POST /submit_assg endpoint returns a 400 status code when an invalid submission is made.

Test Function:

```
def test_submit_assignment_invalid_request(self, client):
        data = {
            "invalidQuestionId": "Invalid Option"
        }         response =
client.post('/submit_assg', json=data)         assert
response.status_code == 400
```

Expected Output:

```
{
  "status_code": 400
}
```

Actual Output:

```
{
  "status_code": 400
}
```

Result: Pass

---

6. Implementation Details

6.1 Instructions to Use the Vue.js and Flask Web Application

Prerequisites

Before you start, ensure you have the following installed on your system:

- Node.js (which includes npm)
- Python
- Git
- Virtualenv (optional but recommended)

1. Clone the Repository

First, clone the repository from GitHub to your local machine:

git clone <your-repo-url>
cd <your-repo-directory>

2. Set Up the Backend (Flask)

a. Create a Virtual Environment

It's a good practice to use a virtual environment to manage your project dependencies. Navigate to the backend directory and create a virtual environment:

cd backend
python -m venv venv

Activate the virtual environment:

- On Windows: **venv\Scripts\activate**
- On macOS and Linux**: source venv/bin/activate**

b. Install Dependencies

Install the required Python packages using pip:

pip install -r requirements.txt

c. Configure Environment Variables

Create a .env file in the backend directory and add your environment variables.

d. the Flask Application

Start the Flask server: **flask run**

The backend server should now be running at ***http://127.0.0.1:5000***.

3. Set Up the Frontend (Vue.js)

Navigate to the frontend directory: **cd ../frontend**

a. Install Dependencies

Install the required Node.js packages using npm:

npm install

b. Run the Vue.js Application

Start the Vue.js development server:

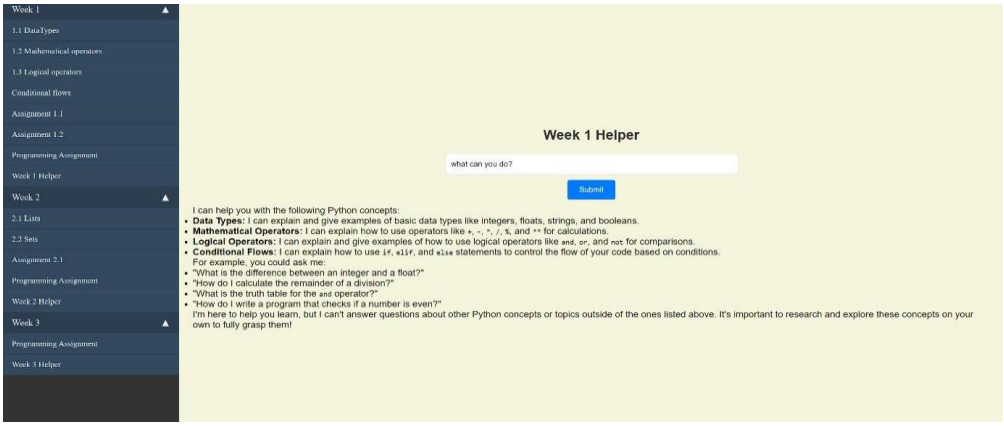
npm run serve

The frontend server should now be running at ***http://localhost:8080***.

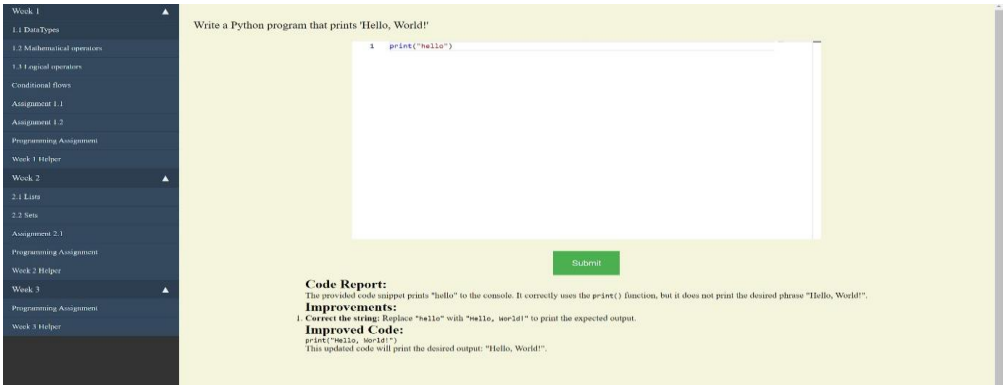
4. Access the Application

Open your web browser and navigate to ***http://localhost:8080***. You should see the frontend of your web application.

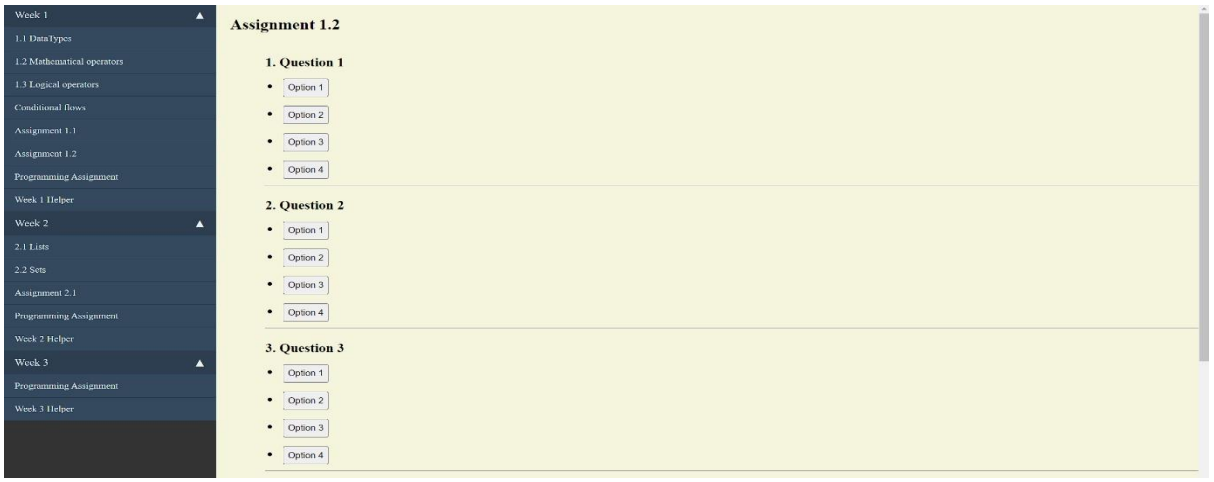
Week Helper: -



Programming assignment: -



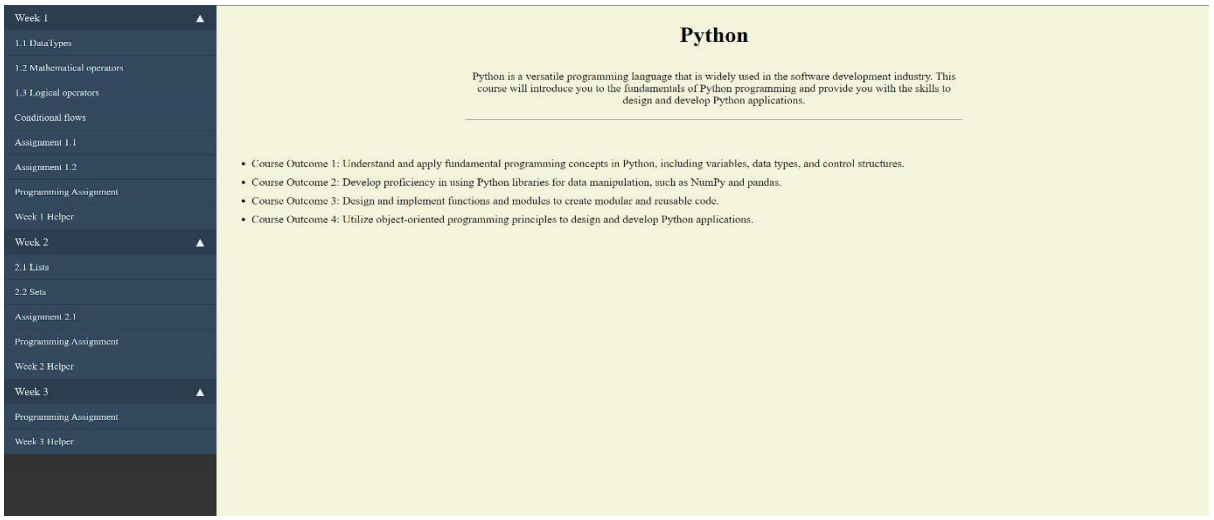
Assignment: -



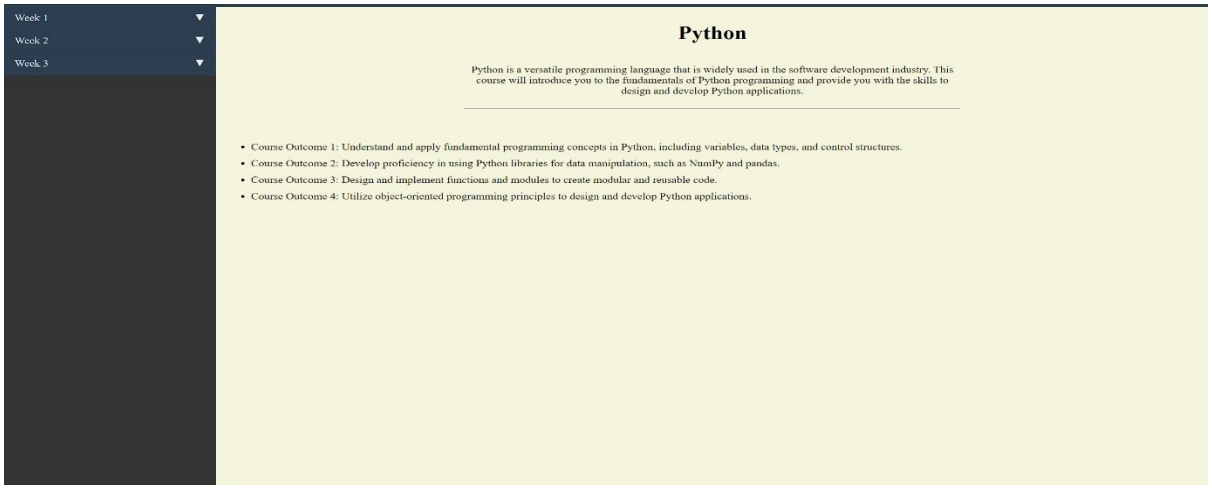
Lecture Summaries: -

Lecture:

Navigation Sidebar: -



Home Page: -



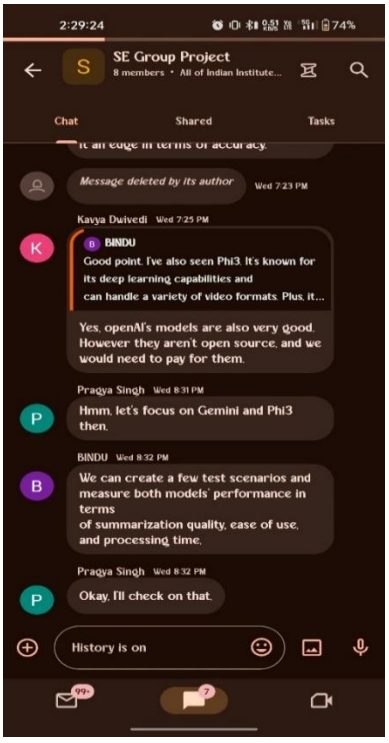
6.2 Screenshots:

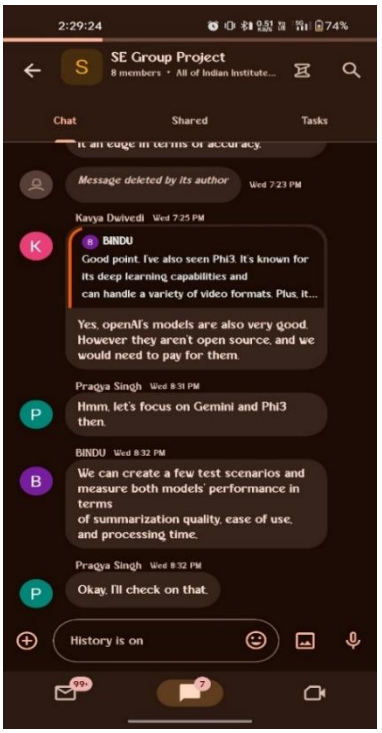
6.2.1 General Discussioin of project:

In these conversations, we focus on:

- Selecting AI Models: Evaluating different AI models for video summarization, including Gemini and Phi3, based on their capabilities, accuracy, and ease of use.
- UI Improvements: Enhancing the user interface by adjusting the background color and tweaking the navigation bar for a better user experience.
- API Testing: Ensuring the APIs for the video summarizer, chatbot, and code summarizer function correctly, and addressing issues with the chatbot API.
- Training Data for Chatbot: Deciding on the appropriate amount and relevance of data to train our chatbot effectively.

Convo-1:

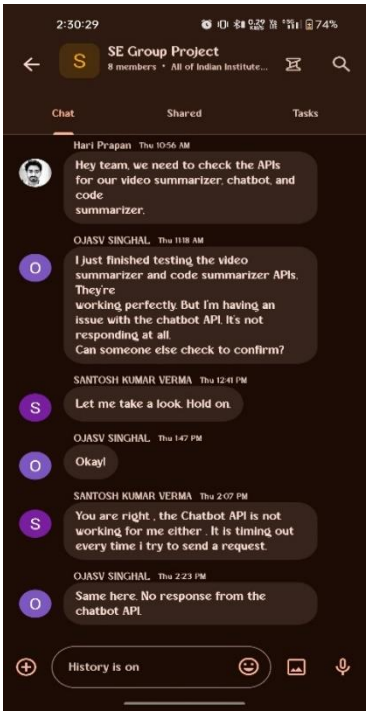


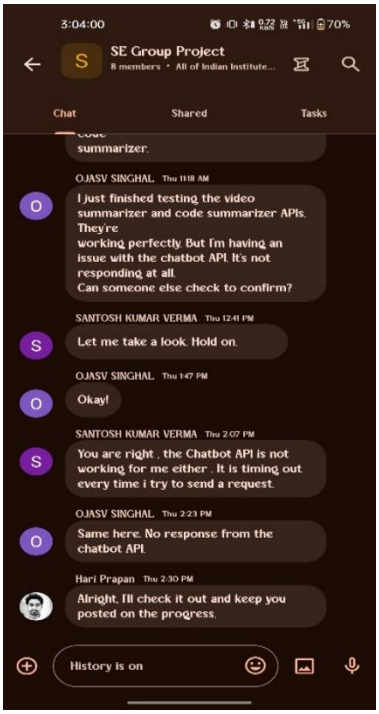


Convo-2:

Convo-3:

Convo-4:





6.2.2 Code Review and Issue Tracking Conversations:

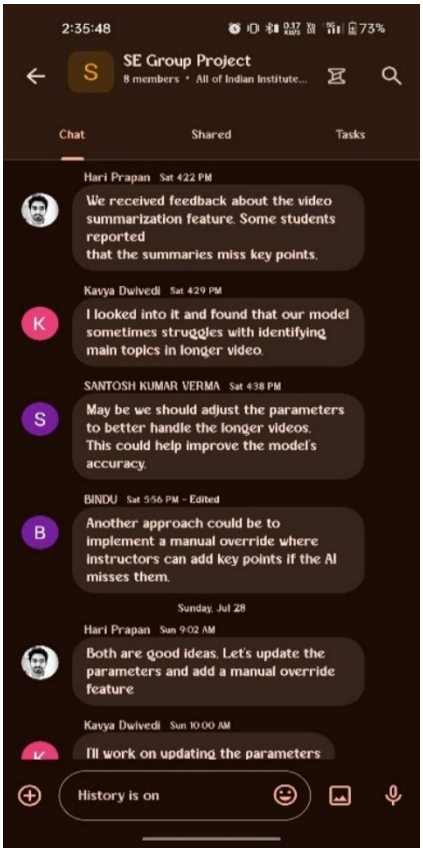
Through these discussions, we focus on:

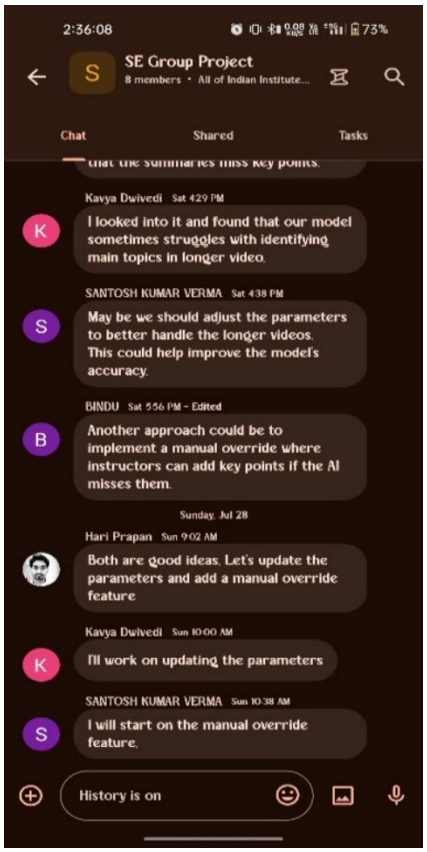
- Enhancing feedback mechanisms for better user experience.
- Addressing key points missed in video summaries.
- Reviewing and improving error categorization and filtering.
- Ensuring smooth post-deployment performance and monitoring.

These steps are essential to ensure our software meets the high standards expected by our users. The screenshots provide a detailed look into our code review and issue tracking process, showcasing our commitment to continuous improvement and teamwork.

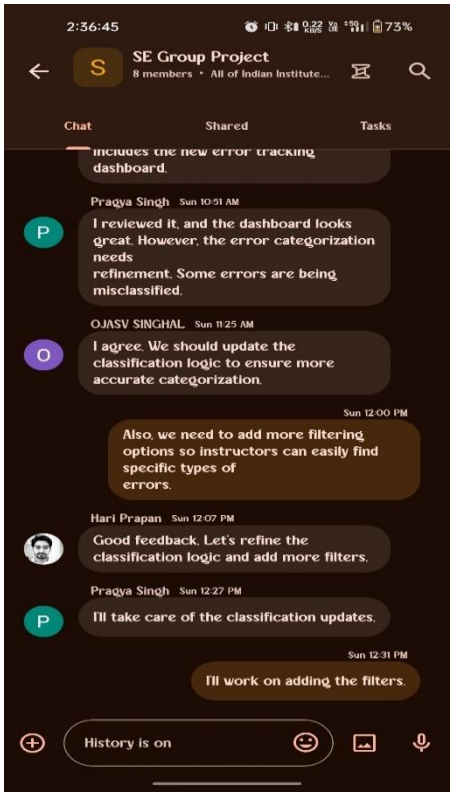
Conversation 1: Code Feedback Review Meeting:

Conversation 2: Video Summarization Issues:





Conversation 3: Reviewing Pull Requests:



Conversation 4: Post-Deployment Review: