SOFTWARE ENGINEERING PROJECT

JAN 2025 TERM - TEAM 1

MILESTONE - 6

PRESENTED BY:

Amit Kulkarni (23f1001947)
Anjali Galav (22f3002299)
Jyotiraditya Saha (21f2000759)
Kajol Singh (21f1001886)
Saima Zainab Shroff (21f3002151)
Sandeep Kumar (21f3002365)
Siddharth Umathe (22f2001536)

CONTENTS

ABOUT THE PROJECT

With the rapid advancement of generative AI technologies, there is a need to integrate these technologies into the SEEK portal to enhance the learning experience for students. The current system lacks personalized guidance and support, which can hinder students' ability to navigate course material effectively, improve study habits, and maintain academic integrity.

The objective of this project is to develop an AI agent that serves as a virtual guide for students enrolled in the IITM BS program.

THE TEAM

Siddharth Umathe

Anjali Galav

Amit Kulkarni

Sandeep Kumar

Kajol Singh

Jyotiraditya Saha

Saima Shroff

MILESTONE - 1

IDENTIFYING USER REQUIREMENTS

IDENTIFIED USERS

01 Primary Users

Students: The primary audience who use the AI agent to get guidance on course materials, assignments, and quizzes. They rely on the agent to enhance their study strategies and locate relevant resources without receiving direct answers, fostering self-learning.

02 Secondary Users

Course Instructors: Course instructors indirectly interact with the AI agent by relying on its ability to guide students effectively. Instructors benefit from the agent's role in addressing routine queries and nudging students toward effective study practices, which supports their teaching efforts.

Teaching Assistants (TAs): TAs may refer students to the AI agent for routine academic guidance, allowing them to focus on more complex or personalized student needs.

IDENTIFIED USERS

03 Tertiary Users

Professors: Professors deliver lectures and play an essential role in designing and overseeing the course content but do not interact with the SEEK portal directly.

Support Team: The support team manages technical issues and addresses students' non-academic concerns related to the AI agent or SEEK portal.

Developers and Maintainers of the AI Agent: The developers and maintainers are responsible for building, testing, and ensuring the ongoing functionality of the AI agent. They design the system to align with the institution's objectives, integrate it seamlessly into the SEEK portal, and ensure it enforces academic integrity principles while providing valuable guidance to students.

USER STORIES

01

As a student,
I want the AI agent to summarize key concepts from lecture videos into bullet points,
so that I can review and understand the main ideas without re-watching the entire content.

02

As a student,
I want the AI agent to provide a comprehensive summary of all the content covered during the week, including key concepts and activities,
so that I can review the entire week's material at once without needing to revisit individual lectures or resources.

03

As a student,
I want the AI agent to persistently remember my last 5 queries and maintain the context of our conversations,
so that I can have seamless and coherent conversations without repeating information and avoid redundant discussions.

04

As a student,
I want the AI agent to answer academic queries related to specific topics,
so that I can quickly access relevant course material and deepen my understanding without wasting time searching through multiple resources.

USER STORIES

05

As a student,
I want the AI agent to generate 10 practice questions tailored to the specific topics I request,
so that I can effectively test my understanding and address my weak areas with focused practice.

06

As a student,
I want the AI agent to generate concise notes with bullet points for the specific topics I request,
so that I can quickly and effectively review key concepts to prepare for assignments and quizzes.

07

As a student,
I want the AI agent to create mock quizzes or end-term exam simulations tailored to the course content,
so that I can practice under exam-like conditions and identify areas for improvement.

08

As a student,
I want the AI agent to recommend specific topics to review if my performance in a mock quiz is below 80%,
so that I can focus on weak areas and improve my understanding quickly.

USER STORIES

09

As a student,
I want the AI agent to analyze my programming assignment errors and explain the underlying issues,
so that I can understand what went wrong and learn how to fix them effectively.

10

As a teaching assistant,
I want the AI agent to handle routine academic queries from students,
so that I can focus on providing personalized support for more complex issues.

11

As a course instructor,
I want the AI agent to generate mock quizzes and end-term exams tailored to the course syllabus and difficulty levels,
so that students can effectively practice and improve their readiness for the actual exams.

12

As a support team member,
I want to track and analyze usage statistics for the AI agent's features,
so that I can monitor feature adoption, identify user preferences, and investigate underutilized functionalities.

MILESTONE - 2

USER INTERFACES

STORYBOARD

Meet **Anjali** – a student juggling endless lectures, assignments, and deadlines. With time slipping away and a mountain of content to review, she’s searching for a smarter way to study.


This lecture was so long. I can't afford to spend another hour rewatching this. There has to be a better way to review!

1.2: Scikit-Learn

Summarize Lecture

Lecture Summary

Scikit-learn is a Python library for machine learning, built on NumPy, SciPy, and Matplotlib. It supports tasks like classification, regression, clustering, and preprocessing through a simple API. Key steps include:

1. Install: pip install scikit-learn.

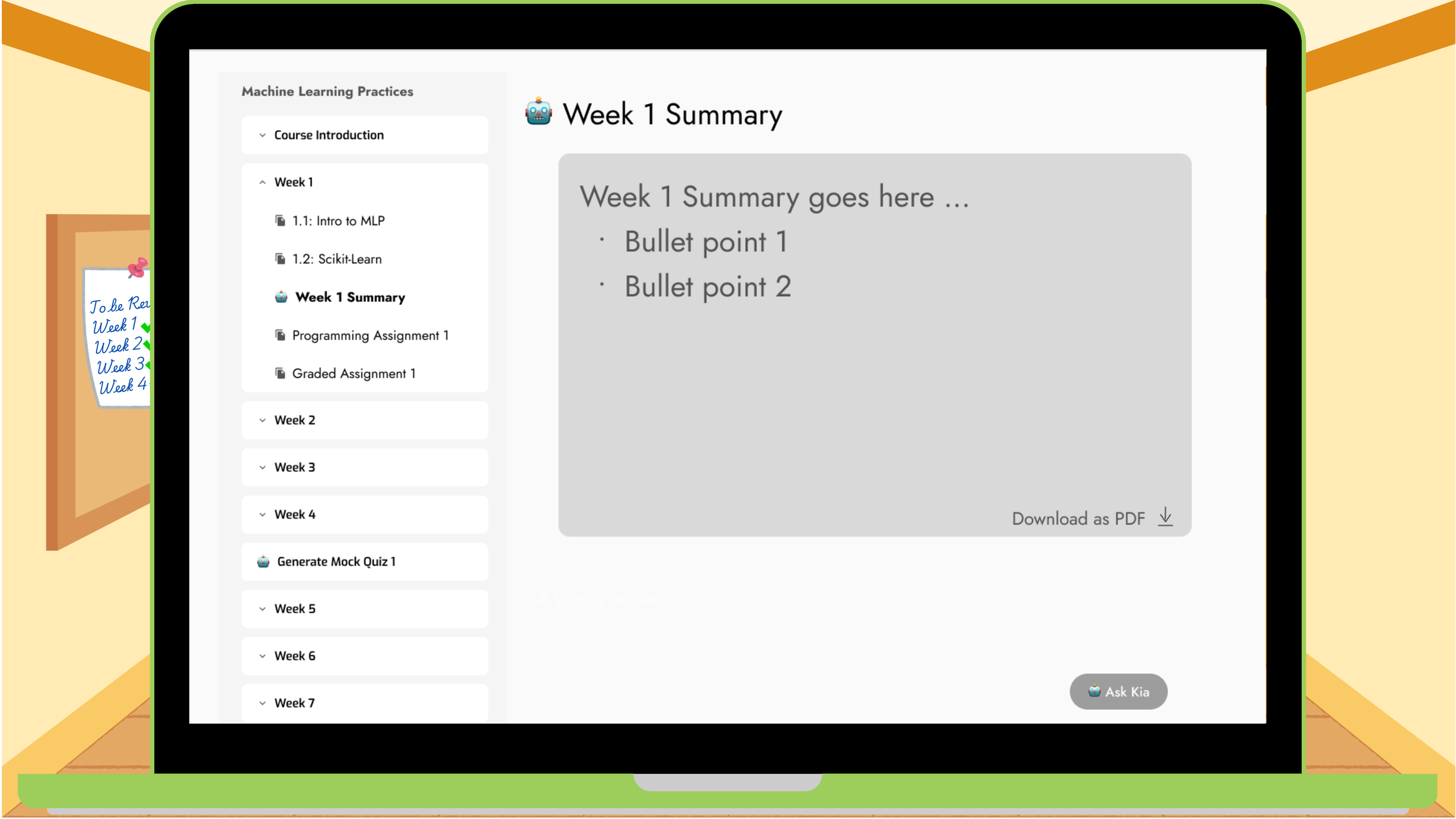
Copy Text

Ask Kia


This saved me so much time! I understand the main points now, and I'm ready for the assignment.

Anjali is drowning in lecture videos the night before the assignment deadline. The clock ticks louder than her racing thoughts... until a smarter way to revise emerges.


To be Revised
Week 1
Week 2
Week 3
Week 4
So many lecture videos to revisit before Quiz 1!


Machine Learning Practices
Course Introduction
Week 1
1.1: Intro to MLP
1.2: Scikit-Learn
Week 1 Summary
Programming Assignment 1
Graded Assignment 1
Week 2
Week 3
Week 4
Generate Mock Quiz 1
Week 5
Week 6
Week 7
Week 1 Summary
Week 1 Summary goes here ...
Bullet point 1
Bullet point 2
Download as PDF
Ask Kia


To be Revised
Week 1
Week 2
Week 3
Week 4
Whoa! This is exactly what I needed!
Finally, a way to catch up without losing sleep!"


To be Revised
Week 1
Week 2
Week 3
Week 4
Am I really ready for the Quiz?

Machine Learning Practices

- Course Introduction
- Week 1
- Week 2
- Week 3
- Week 4
- Generate Mock Quiz 1
- Week 5
- Week 6
- Week 7
- Week 8
- Generate Mock Quiz 2
- Week 9
- Week 10

Mock Quiz 1

1. What is the full form of ML? 1 point

- Machine Language
- Machine Learning
- Macro Learning

2. Which of the following is a type of supervised learning? 1 point

- Clustering
- Regression
- Dimensionality Reduction
- Association Rule Mining

Check Score

Your Assignment Score Is 1/2

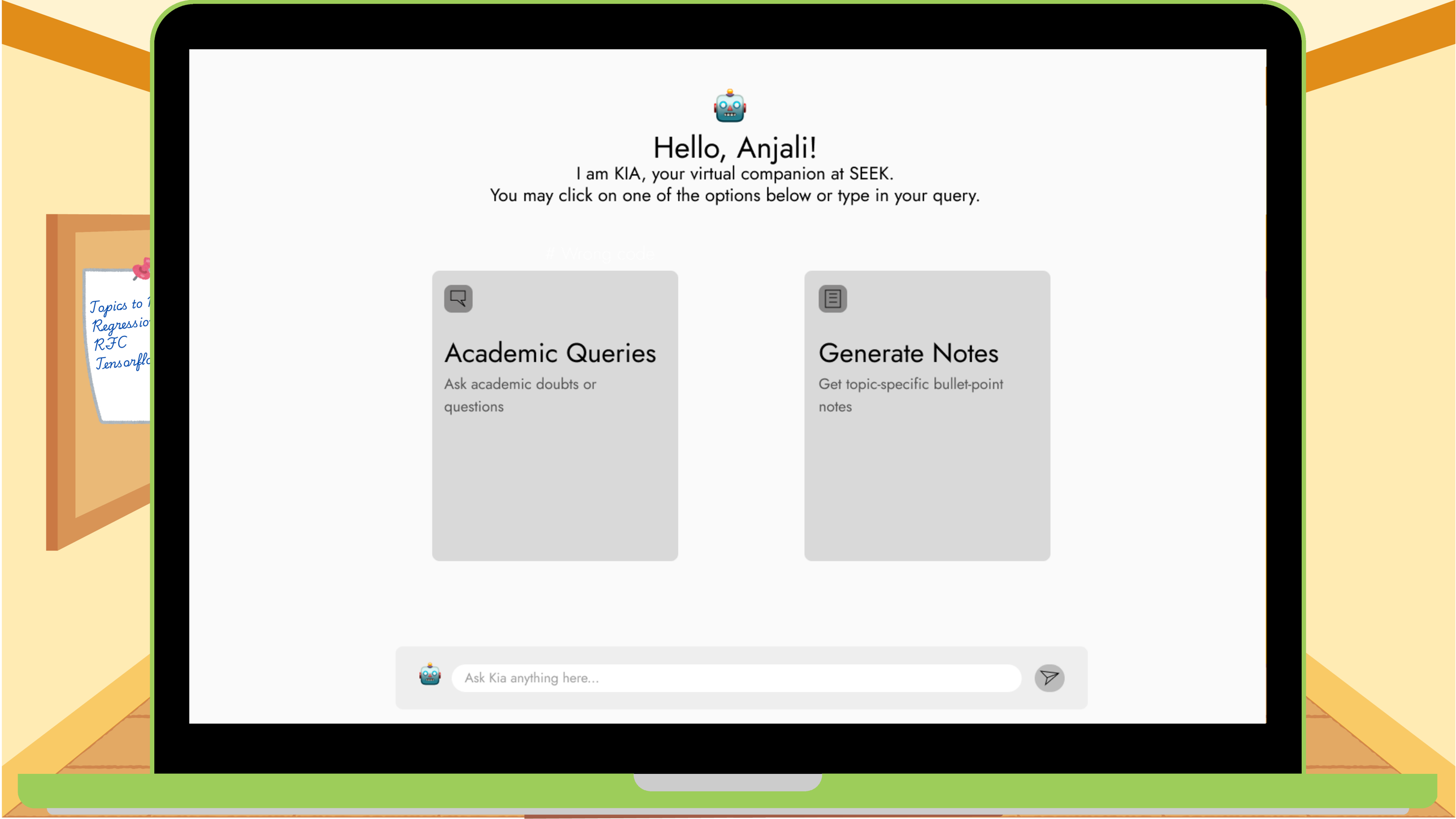
Topics For Review:

Kia's Feedback Here

Download Report

Ask Kia


Topics to Review
Regression
RFC
Tensorflow
Perfect! Now I know where to focus my studying.


Topics to
Regressio
RFC
Tensorflo
Wrong code
Hello, Anjali!
I am KIA, your virtual companion at SEEK.
You may click on one of the options below or type in your query.
Academic Queries
Ask academic doubts or questions
Generate Notes
Get topic-specific bullet-point notes
Ask Kia anything here...

Generate Topic-Specific Notes

Regres

Search

- Regression
- Auto-regressive model
- Linear Regression
- Probabilistic Regression Model
- Logistic Regression

Reset Session

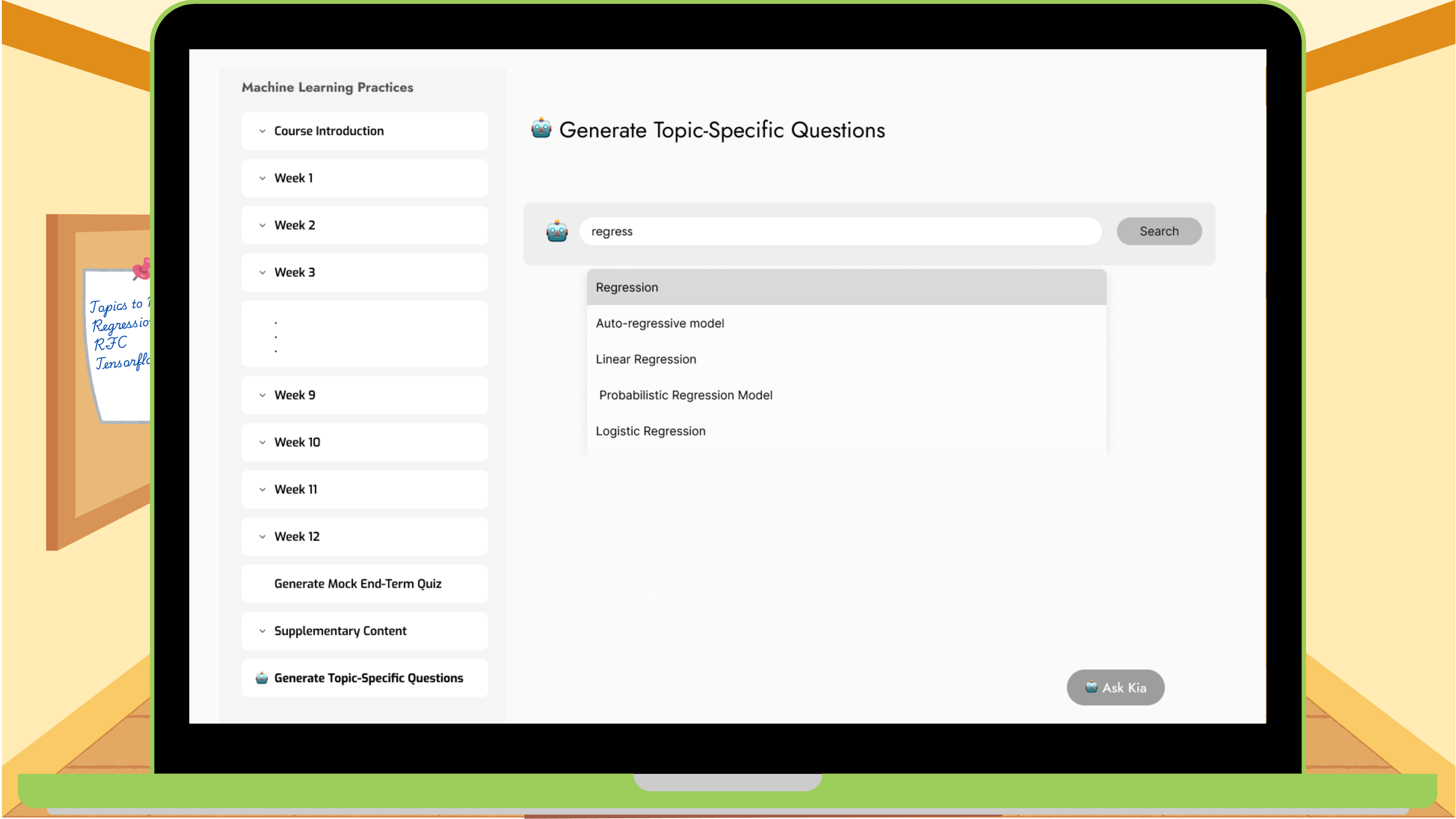
Notes on Regression

Topic notes go here...

- Bullet point 1
- Bullet point 2

Download as PDF

Reset Session


Machine Learning Practices
Course Introduction
Week 1
Week 2
Week 3
Week 9
Week 10
Week 11
Week 12
Generate Mock End-Term Quiz
Supplementary Content
Generate Topic-Specific Questions
Generate Topic-Specific Questions
regress
Search
Regression
Auto-regressive model
Linear Regression
Probabilistic Regression Model
Logistic Regression
Ask Kia

Machine Learning Practices

- Course Introduction
- Week 1
- Week 2
- Week 3
- .
- .
- .
- Week 9
- Week 10
- Week 11
- Week 12
- Generate Mock End-Term Quiz
- Supplementary Content
- Generate Topic-Specific Questions

Practice Questions on **Regression**

1. Which of the following is NOT a type of regression?

- ○ Polynomial Regression
- ○ K-Means Regression
- ○ Random Forest Regression

2. Which of the following statements about Linear Regression are true?

- ☐ It assumes a linear relationship between independent and dependent variables
- ☐ It can be affected by multicollinearity among features
- ☐ It works well with categorical target variables
- ☐ It minimizes the sum of squared errors

Check Score

Ask Kia

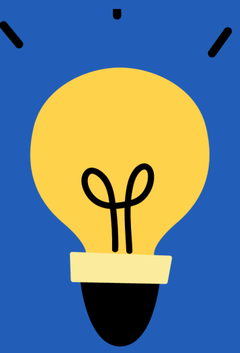
I can't believe how much I got done today with Kia! Usually, the night before an exam is a complete disaster.

First, the weekly content summarizers... saved me hours of re-watching lectures!

The mock quiz? That was genius. Found my weak spots before the real test.

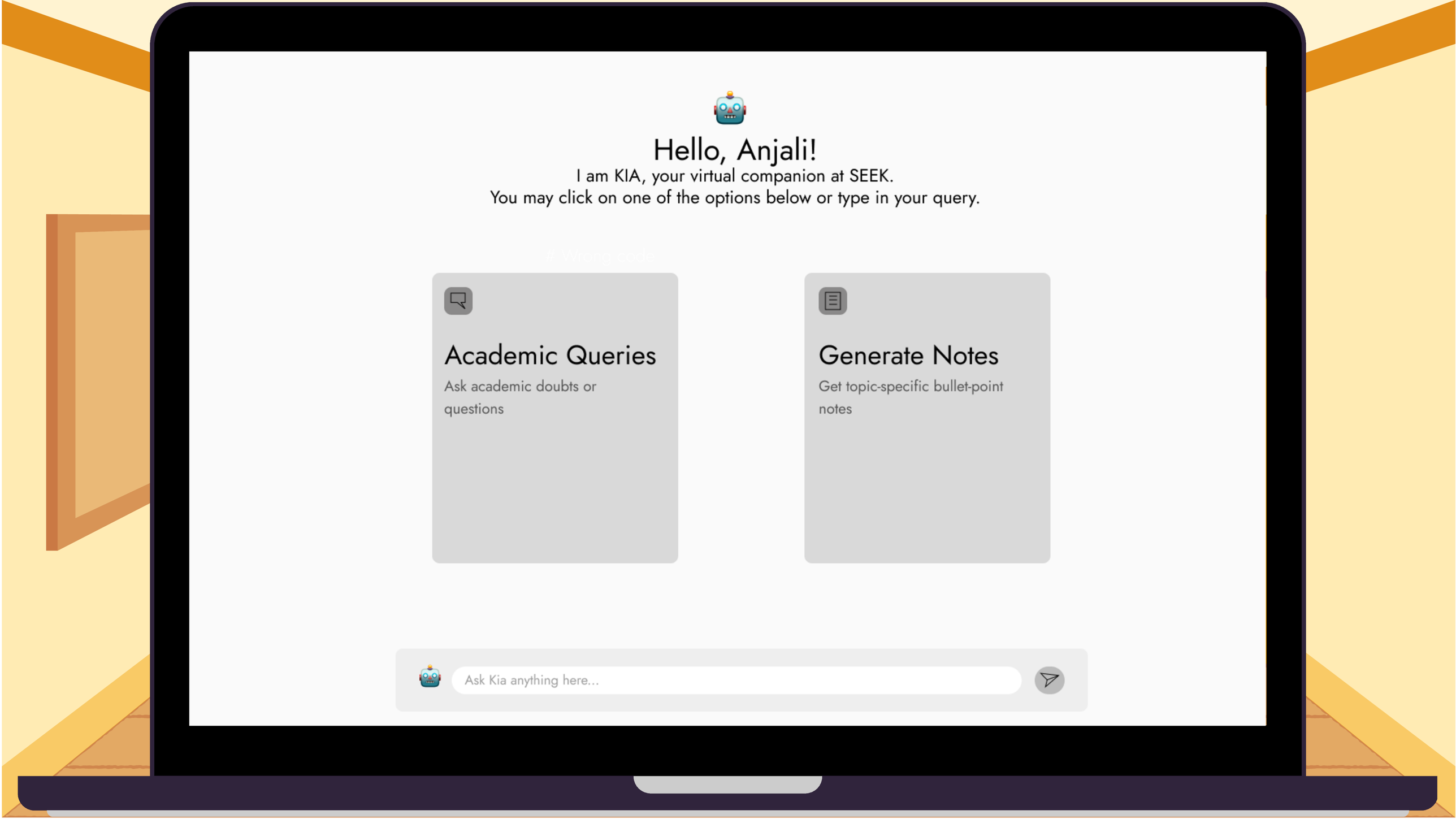
I didn't even need to run anywhere for topic-specific resources and questions. Thanks to Kia!

No more all-nighters, no more cramming. Just...sleep. And tomorrow, I ace that exam!



Trapped in a maze of formulae and scattered lecture slides, **Anjali** struggles as multiple regression clouds her understanding. She seeks **Kia** for help, who remembers more than just equations.


I don't fully understand how multiple regression works. Searching through all these lecture slides is taking forever!
If only I could get a quick and clear explanation without digging through multiple resources.


Hello, Anjali!
I am KIA, your virtual companion at SEEK.
You may click on one of the options below or type in your query.
Wrong code
Academic Queries
Ask academic doubts or questions
Generate Notes
Get topic-specific bullet-point notes
Ask Kia anything here...

Academic Query Support

Hey There! How can I help you today?

When should I use simple versus multiple regression?

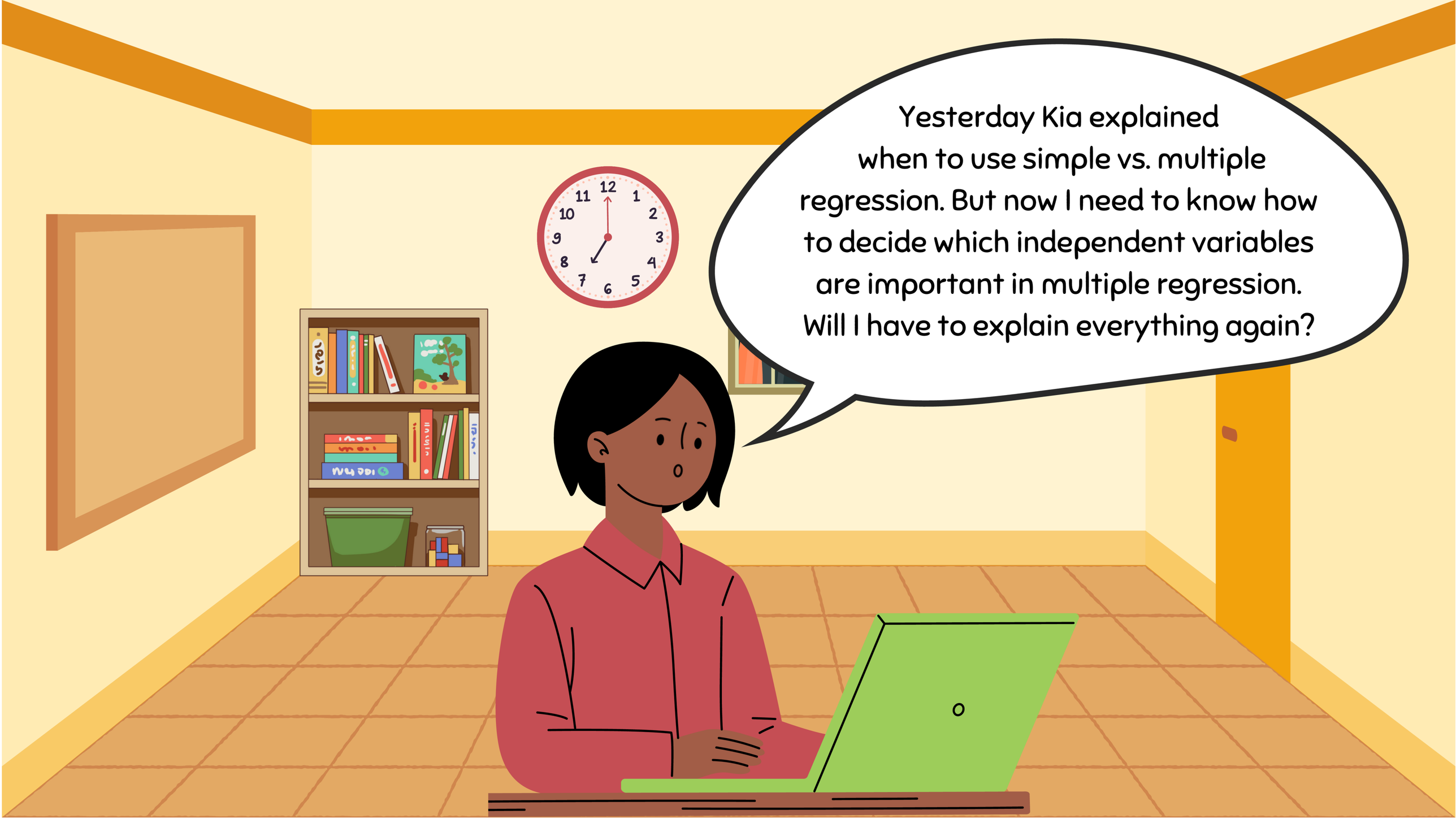
Simple and multiple regression are both techniques used to model relationships between variables, but they differ in complexity and purpose. Here's when to use each:

When to Use Simple Regression

- **Single Predictor Focus:**
  Use simple regression when you believe that one independent variable is the primary driver of your

Ask a question

Reset Session


Yesterday Kia explained when to use simple vs. multiple regression. But now I need to know how to decide which independent variables are important in multiple regression. Will I have to explain everything again?

Academic Query Support

Hey There! How can I help you today?

How do I decide which independent variables to include in multiple regression?

Since you **previously asked** about when to use simple and multiple regression, let me build on that.
To select the right independent variables for multiple regression, consider:

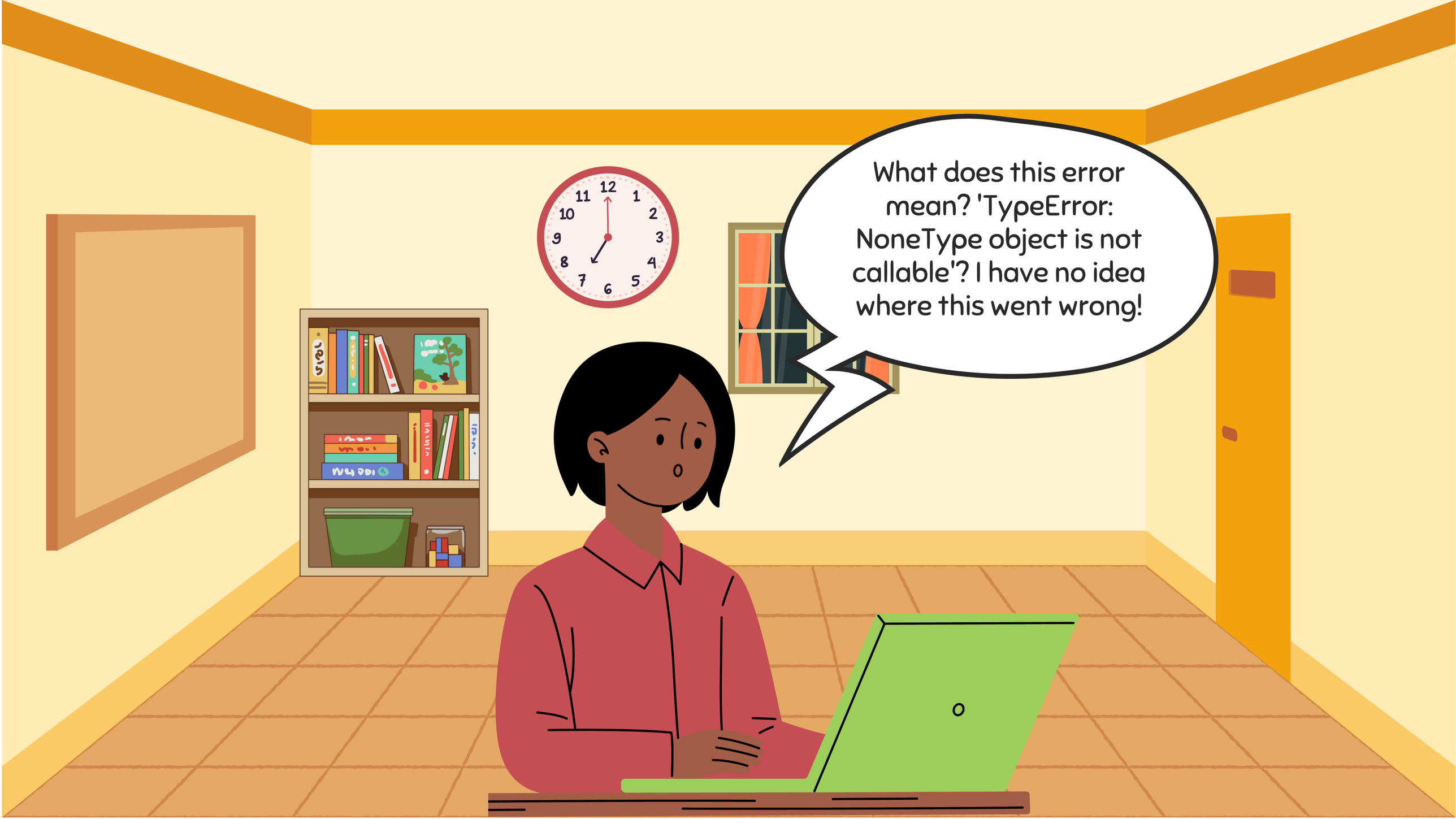
- **Correlation Strength:** Use variables that show a

Ask a question

Reset Session


Kia is just amazing! I don't have to keep re-explaining my doubts, it keeps track of what I ask.
From now on, I'll use this for all my study queries. No redundant discussions!

Anjali is working on a programming assignment but encounters a complicated error. Frustration builds, but **Kia** steps in, simplifying the error and guiding her to a solution.


What does this error mean? 'TypeError: NoneType object is not callable'? I have no idea where this went wrong!

Machine Learning Practices

- Course Introduction
- Week 1
  - 1.1: Intro to MLP
  - 1.2: Scikit-Learn
  - Week 1 Summary
  - **Programming Assignment 1**
  - Graded Assignment 1
- Week 2
- Week 3
- Week 4
- Generate Mock Quiz 1
- Week 5
- Week 6
- Week 7

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Wrong code
```

0/4 Test cases passed

Submit

Output

```
# Wrong output
```

Explain Error

Ask Kia

Machine Learning Practices

- Course Introduction
- Week 1
  - 1.1: Intro to MLP
  - 1.2: Scikit-Learn
  - Week 1 Summary
  - **Programming Assignment 1**
  - Graded Assignment 1
- Week 2
- Week 3
- Week 4
- Generate Mock Quiz 1
- Week 5
- Week 6
- Week 7

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Wrong code
```

0/4 Test cases passed

Submit

Output

```
TypeError: NoneType object is not callable
```

Explain Error

Here's what this error means:

- A function or variable in your code is being treated as something that can be called, but it's actually **None.**
- This usually happens when you accidentally overwrite a function with a variable of the same name or forget to return a value from a function.

Ask Kia

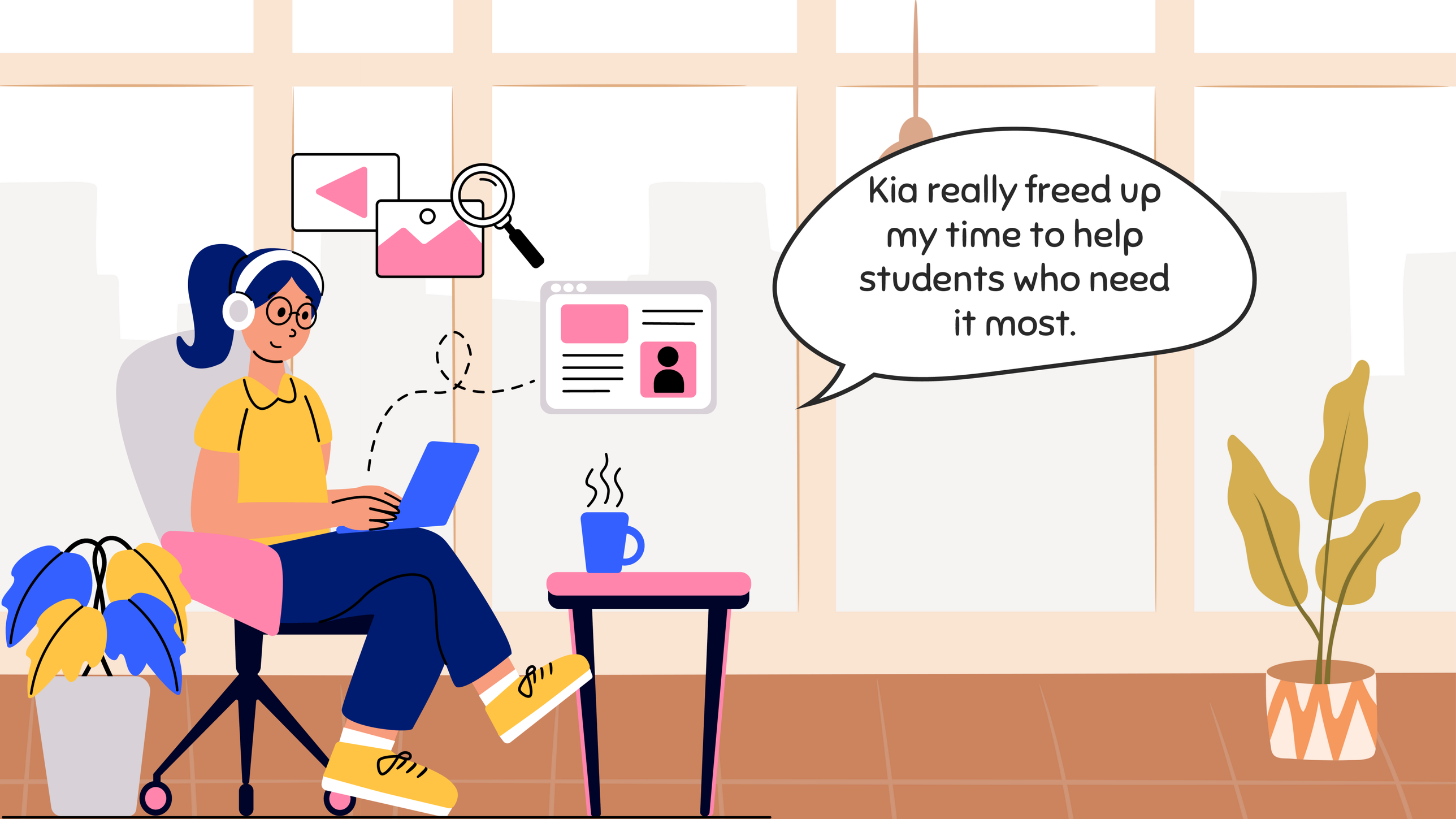

Finally! Now I know what this error means and how to debug it if I see it again.
Kia makes error messages so much easier to understand. It's like having a tutor who explains things in simple English!

Teaching Assistant **Soumya's** inbox overflows, and **Prof. Ashish** becomes overburdened as office hours begin. But this term, **Kia** handles the tidal wave – freeing human expertise for where it matters most.


Office hours... here we go. Hope I can answer everyone's questions in time.

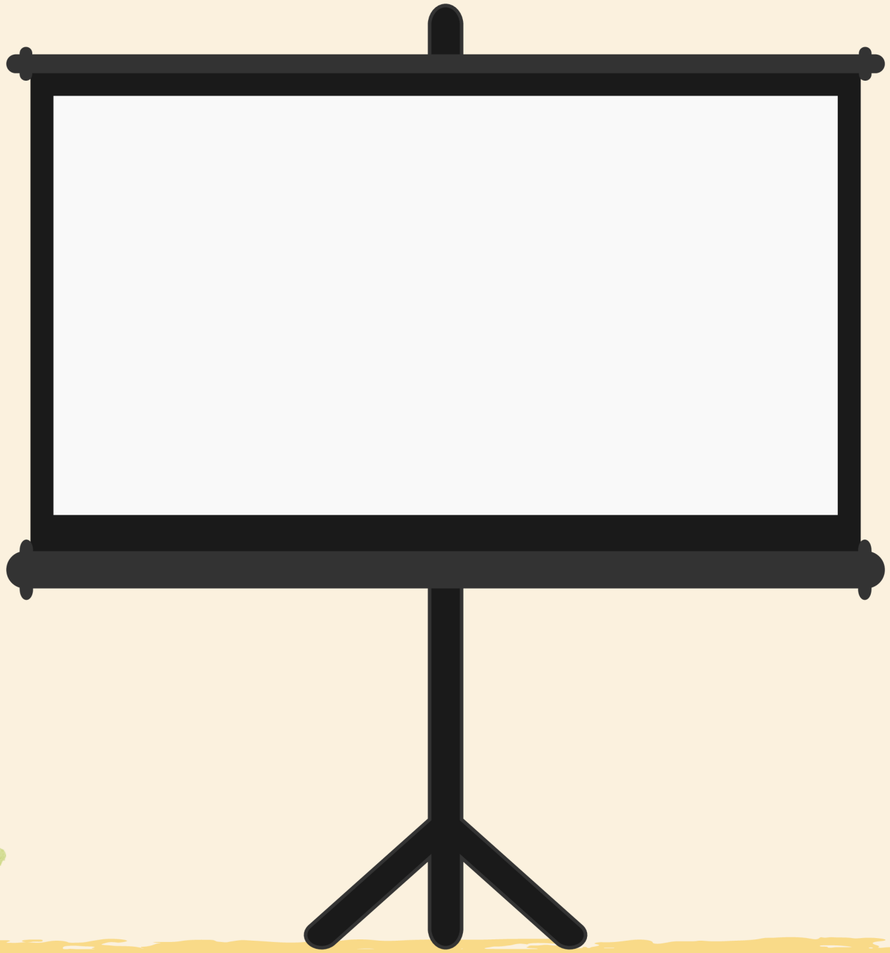

Wow, Kia is handling the routine academic queries perfectly!

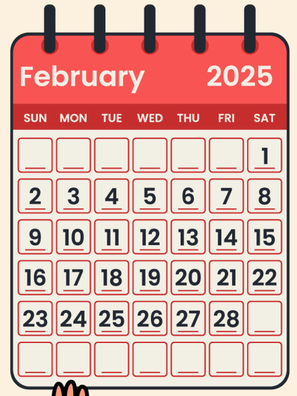

Okay, this student is struggling with a core concept... now I can actually focus on helping them.

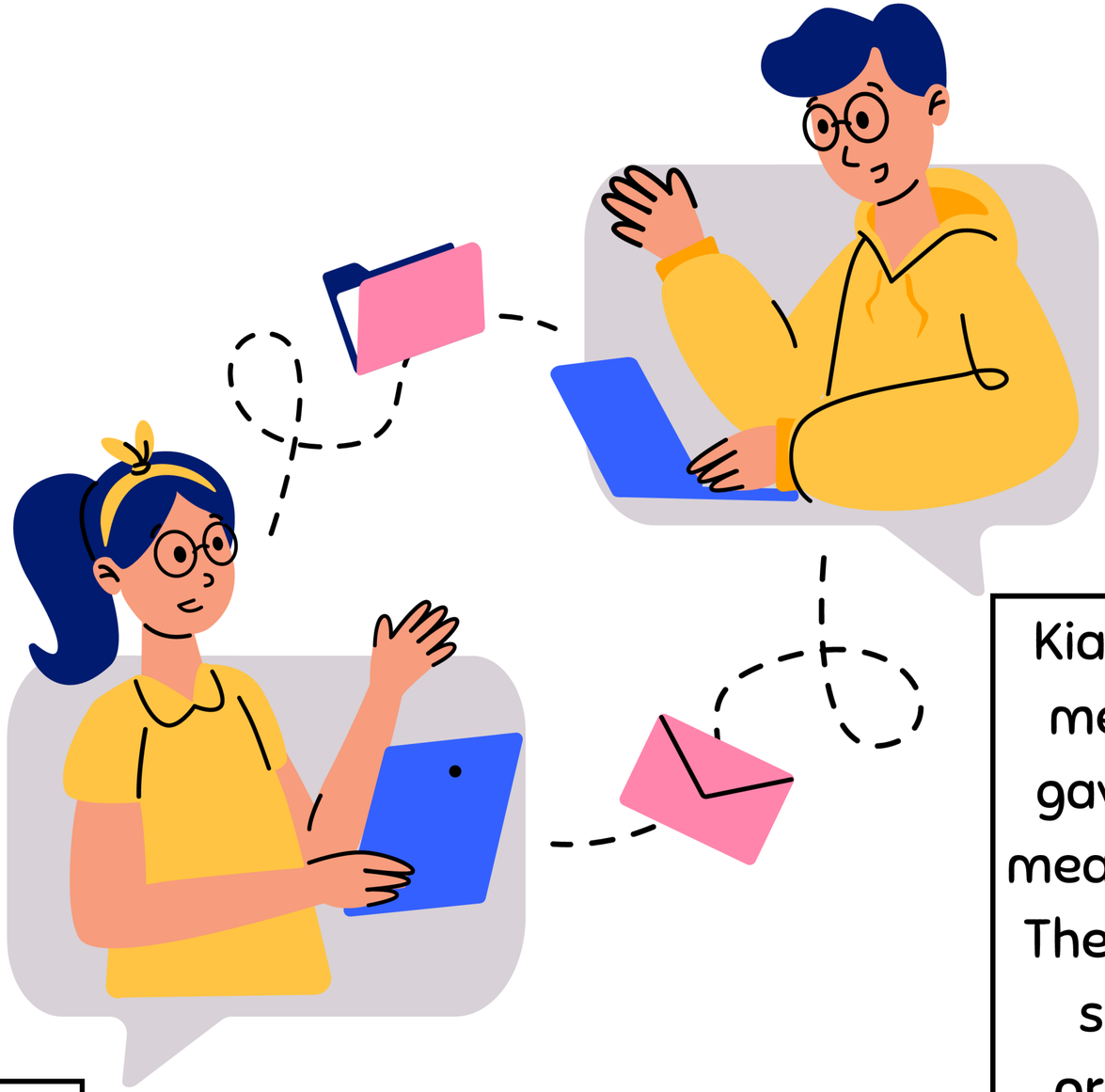

Kia really freed up my time to help students who need it most.


I need to create mock quizzes for the end-term exam that challenge my students appropriately. But where do I find the time?
February 2025
SUN MON TUE WED THU FRI SAT
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28

Prof. Ashish learns about Kia's ability to generate mock quizzes...







Kia not only saved me time but also gave my students meaningful practice. They're going to be so much more prepared for the finals!

Finally, a way to provide better guidance support without getting overwhelmed.

As a support team member, **Priyam's** mission is to uncover how users are engaging with **Kia's** features and identify opportunities for improvement.

Are users engaging with all the features, or are some being ignored?

Let's see... timestamps, feature calls, user IDs... everything I need is here

Using log analysis tools, Priyam transforms raw server data into actionable insights ...

Feature adoption trends... engagement rates... even anomalies. This is gold!

Looks like the 'Topic-Specific Questions Generation' feature isn't getting much traction. Why?

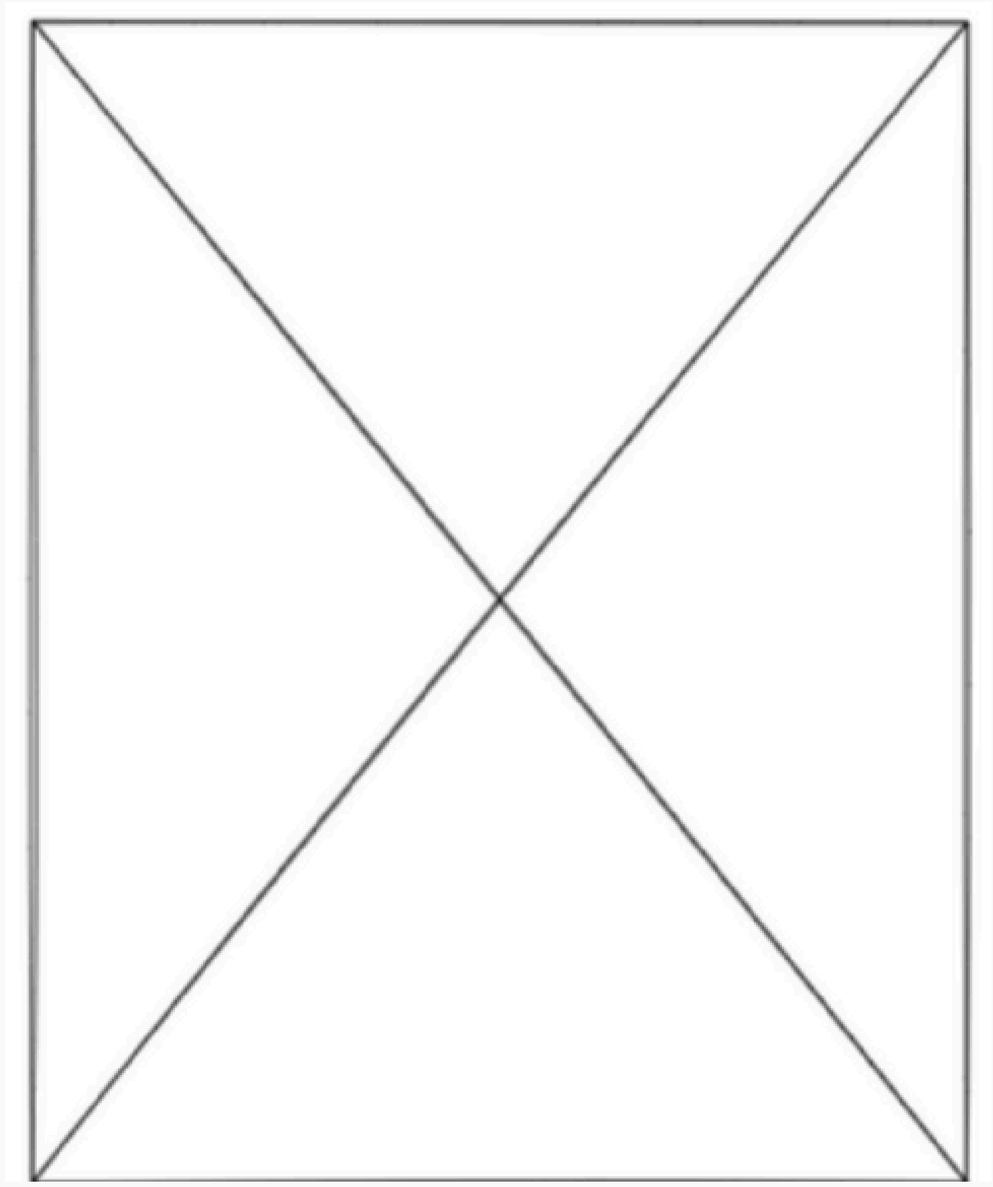
Could it be too complex or just not well-promoted? I should flag this for the product team.

Armed with insights, Priyam collaborates with the product team to improve feature visibility and usability.

WIREFRAMES

SEEK

Register



Login

Email

Enter your Email

Password

Enter your Password

Login

Sign In With Google

New User? Register

SEEK

Create An Account

Full Name

Enter your Full Name

Email

Enter your Email

Password

Enter your Password

Confirm Password

Re-enter your Password

Login

Sign In With Google

Existing User? Login

1.2: Scikit-Learn

Summarize Lecture

Ask Kia

1.2: Scikit-Learn

Summarize Lecture

Video content summary goes here ...

Copy Text

Ask Kia

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Write your Python code here ...
```

Submit

Ask Kia

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Right code
```

2/2 test cases passed

Submit

Output

```
# Right output
```

Ask Kia

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Wrong code
```

0/2 test cases passed

Submit

Output

```
# Wrong output
```

Explain Error

Ask Kia

Programming Assignment 1: Train an SVM Classifier

Write a code in Python to train an SVM Classifier with specefic parameters and datapoints.

```
# Wrong code
```

0/2 test cases passed

Run & Submit

Output

```
# Wrong output
```

Explain Error

Error explanation goes here ...

Ask Kia

🤖 Week 1 Summary

Week 1 Summary goes here ...

- Bullet point 1
- Bullet point 2

Download as PDF

Wrong output

🤖 Ask Kia

IIT Madras

Profile

Machine Learning Practices

- Course Introduction
- Week 1
- Week 2
- Week 3
- Week 4
- 🤖 Generate Mock Quiz 1
- Week 5
- Week 6
- Week 7
- Week 8
- 🤖 Generate Mock Quiz 2
- Week 9
- Week 10

🤖 Mock Quiz 1

1) What is the full form of ML? 1 point

- Machine Language
- Machine Learning
- Macro Learning

2. Which of the following is a type of supervised learning? 1 point

- Clustering
- Regression
- Dimensionality Reduction
- Association Rule Mining

Check Score

🤖 Mock Quiz 1

1. What is the full form of ML? 1 point

- Machine Language
- Machine Learning
- Macro Learning

2. Which of the following is a type of supervised learning? 1 point

- Clustering
- Regression
- Dimensionality Reduction
- Association Rule Mining

Check Score

Your Assignment Score Is 1/2

🤖 Topics For Review:

Kia's Feedback Here

🤖 Download Report

🤖 Ask Kia

IIT Madras

Profile

Machine Learning Practices

- Course Introduction
- Week 1
- Week 2
- Week 3
- .
- .
- .
- Week 9
- Week 10
- Week 11
- Week 12
- 🤖 Generate Mock End-Term Quiz
- Supplementary Content
- 🤖 Generate Topic-Specific Questions

🤖 Generate Topic-Specific Questions

🤖 regress Search

Regression

Auto-regressive model

Linear Regression

Probabilistic Regression Model

Logistic Regression

Wrong output

🤖 Ask Kia

Practice Questions on **Regression**

1. Which of the following is NOT a type of regression?

- ○ Polynomial Regression
- ○ K-Means Regression
- ○ Random Forest Regression

2. Which of the following statements about Linear Regression are true?

- ☐ It assumes a linear relationship between independent and dependent variables
- ☐ It can be affected by multicollinearity among features
- ☐ It works well with categorical target variables
- ☐ It minimizes the sum of squared errors

Check Score

Ask Kia

IIT Madras

Profile

🤖

Hello, User!

I am KIA, your virtual companion at SEEK.
You may click on one of the options below or type in your query.

```
# Wrong code
```

Academic Queries

Ask academic doubts or questions

Generate Notes

Get topic-specific bullet-point notes

🤖 Ask Kia anything here...

IIT Madras

Profile

🤖 Academic Query Support

🤖 Hey There! How can I help you today?

Wrong code

User Query

🤖 Response

Ask a question

Reset Session

IIT Madras

Profile

🤖 Generate Topic-Specific Notes

Regres

Search

- Regression
- Auto-regressive model
- Linear Regression
- Probabilistic Regression Model
- Logistic Regression

Reset Session

🤖 Notes on Regression

Topic notes go here...

- Bullet point 1
- Bullet point 2

Download as PDF

Reset Session

MILESTONE - 3

SHEDULING AND DESIGN

PROJECT SCHEDULE

1.1 Task Distribution

- The milestones of the project are divided into smaller, manageable tasks. Each task is assigned to team members based on their expertise, availability, and workload to ensure a balanced and efficient workflow.

Milestone	Tasks	Start Date	End Date	Assigned To
Milestone 1	Identify Users	12th January	14th January	Kajol
	Define User Stories	15th January	20th January	Amit, Jyotiraditya
	Draft and Submit Milestone 1 Report	21st January	26th January	Jyotiraditya, Saima
Milestone 2	Design Storyboards	27th January	3rd February	Anjali, Siddharth
	Design Low-Fidelity Wireframes	27th January	3rd February	Saima, Jyotiraditya
	Draft and Submit Milestone 2 Report	4th February	9th February	Jyotiraditya, Saima

PROJECT SCHEDULE

Milestone	Tasks	Start Date	End Date	Assigned To
Milestone 3	Project Scheduling using Gantt Chart and Kanban Board	10th February	12th February	Sandeep
	Component Design	13th February	17th February	Amit
	Software Design - Class Diagram	10th February	13th February	Amit
	Frontend Development	10th February	19th February	Anjali, Amit
	Draft and Submit Milestone 3 Report	17th February	20th February	Jyotiraditya, Saima
Milestone 4	Design APIs	21st February	27th February	Kajol, Anjali
	Create YAML File	28th February	2nd March	Sandeep
	Integrate GenAI Features	21st February	27th February	Amit, Siddharth
	Draft and Submit Milestone 4 Report	28th February	2nd March	Jyotiraditya, Saima

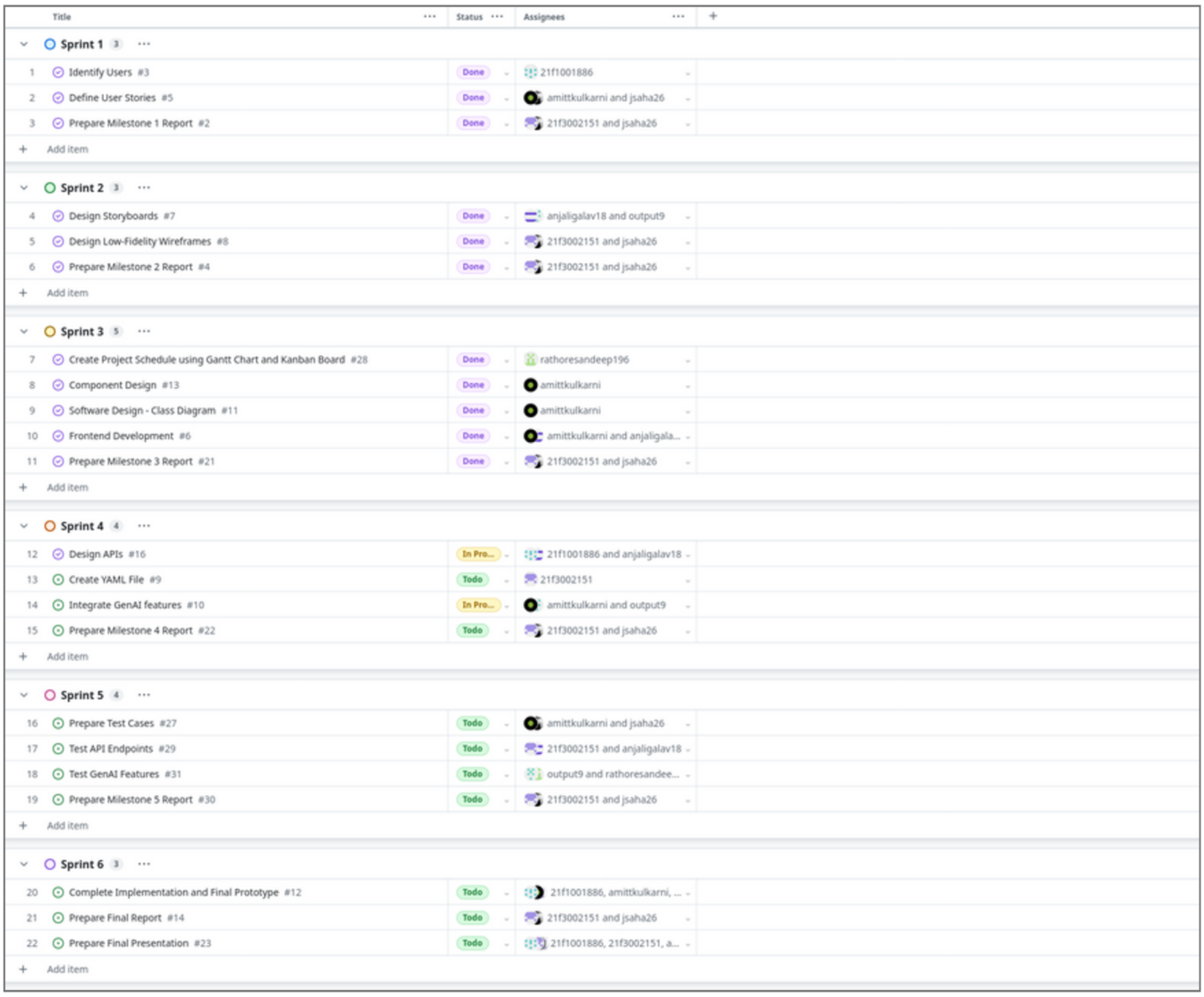
PROJECT SCHEDULE

Milestone	Tasks	Start Date	End Date	Assigned To
Milestone 5	Prepare Test Cases for each API Endpoint	3rd March	7th March	Amit, Jyotiraditya
	Test API endpoints	8th March	10th March	Saima, Anjali
	Test GenAI Functionalities	10th March	12th March	Siddharth, Sandeep
	Draft and submit Milestone 5 Report	13th March	16th March	Jyotiraditya, Saima
Milestone 6	Complete Implementation and Final Prototype	17th March	23rd March	Kajol, Anjali, Amit
	Draft and Submit Final Report	23rd March	26th March	Saima, Jyotiraditya
	Prepare and Submit Final Presentation	27th March	30th March	Entire Team

PROJECT SCHEDULE

1.2 Sprint Backlog

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

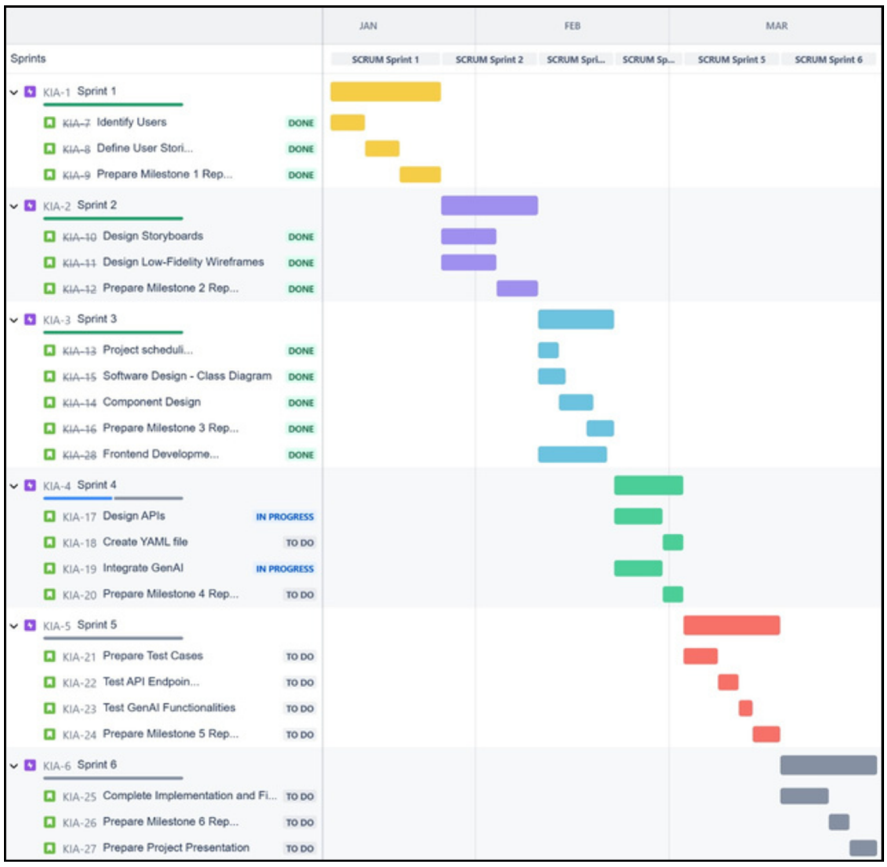


	Title	Status	Assignees
	Sprint 1 3		
1	Identify Users #3	Done	21f1001886
2	Define User Stories #5	Done	amittkulkarni and jsaha26
3	Prepare Milestone 1 Report #2	Done	21f3002151 and jsaha26
+	Add item		
	Sprint 2 3		
4	Design Storyboards #7	Done	anjaligalav18 and output9
5	Design Low-Fidelity Wireframes #8	Done	21f3002151 and jsaha26
6	Prepare Milestone 2 Report #4	Done	21f3002151 and jsaha26
+	Add item		
	Sprint 3 5		
7	Create Project Schedule using Gantt Chart and Kanban Board #28	Done	rathoresandeep196
8	Component Design #13	Done	amittkulkarni
9	Software Design - Class Diagram #11	Done	amittkulkarni
10	Frontend Development #6	Done	amittkulkarni and anjaligala...
11	Prepare Milestone 3 Report #21	Done	21f3002151 and jsaha26
+	Add item		
	Sprint 4 4		
12	Design APIs #16	In Pro...	21f1001886 and anjaligalav18
13	Create YAML File #9	Todo	21f3002151
14	Integrate GenAI features #10	In Pro...	amittkulkarni and output9
15	Prepare Milestone 4 Report #22	Todo	21f3002151 and jsaha26
+	Add item		
	Sprint 5 4		
16	Prepare Test Cases #27	Todo	amittkulkarni and jsaha26
17	Test API Endpoints #29	Todo	21f3002151 and anjaligalav18
18	Test GenAI Features #31	Todo	output9 and rathoresandee...
19	Prepare Milestone 5 Report #30	Todo	21f3002151 and jsaha26
+	Add item		
	Sprint 6 3		
20	Complete Implementation and Final Prototype #12	Todo	21f1001886, amittkulkarni, ...
21	Prepare Final Report #14	Todo	21f3002151 and jsaha26
22	Prepare Final Presentation #23	Todo	21f1001886, 21f3002151, a...
+	Add item		

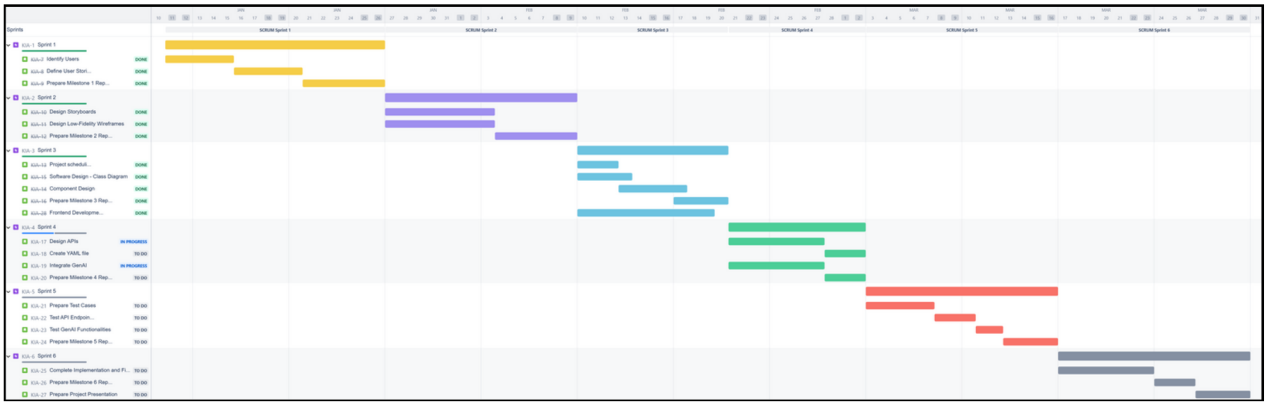
PROJECT SCHEDULE

1.3 Gantt Chart

- A Gantt chart was plotted using Jira for representing the timeline and progress of the project divided into multiple sprints. This helped in visually organizing tasks, their durations, dependencies, and completion statuses, ensuring that all team members are aligned with the milestones and deadlines.



(BY MONTH)

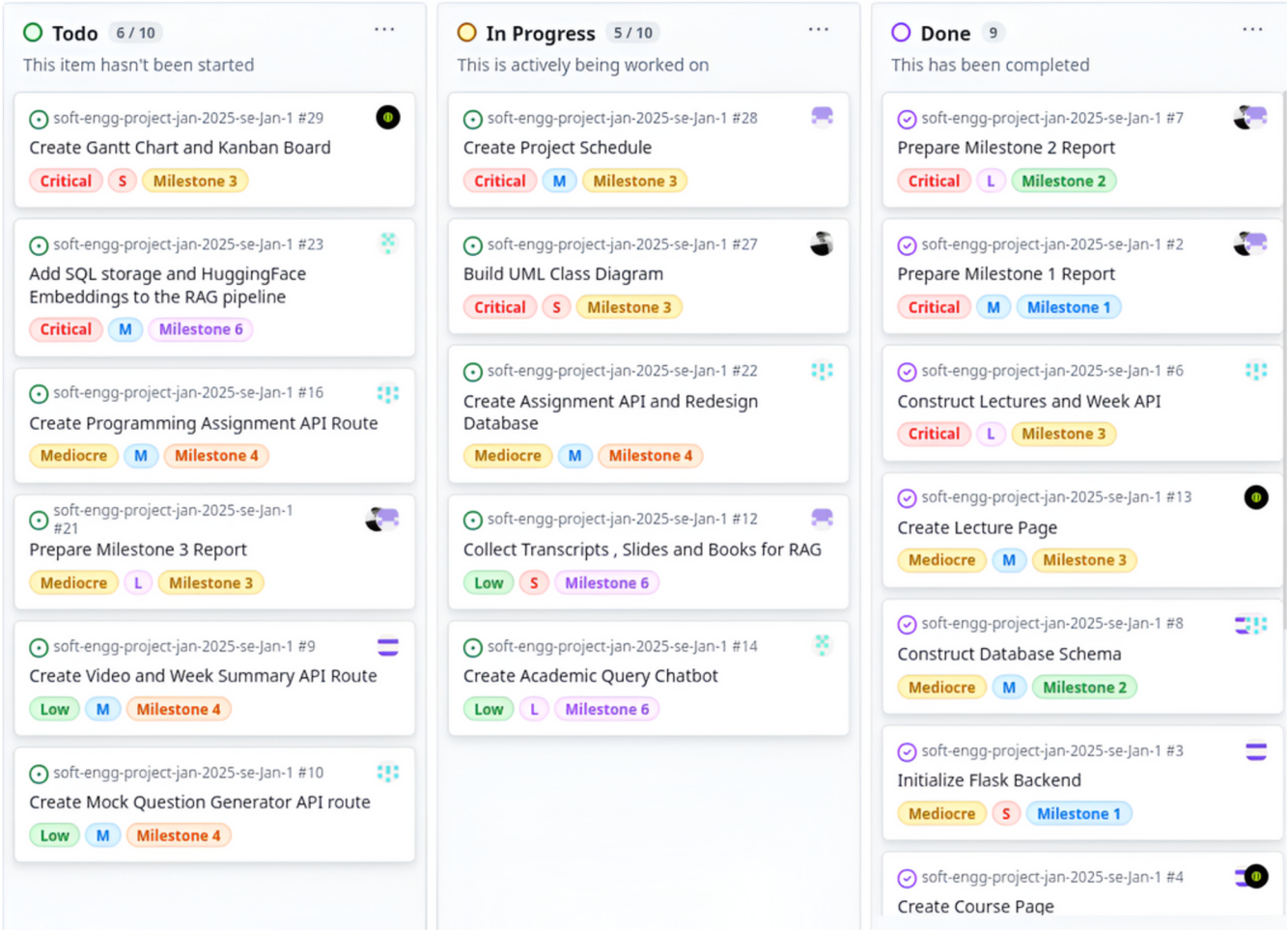


(BY WEEK)

PROJECT SCHEDULE

1.4 Kanban Board

- The Kanban board was made using GitHub to provide a real-time, task-oriented view of the project's progress. It organizes tasks into three categories: To Do, In Progress, and Done, ensuring clear visibility of the workflow. The board also highlights task priorities (Critical, Medium, Low) and their association with specific milestones. This structure helps our team track progress, identify bottlenecks, and manage workload effectively.



SCRUM MEETINGS

2.1 (SCRUM-1 to SCRUM-3) User Requirements

- **Understanding Project Scope and Objectives**
  - The project scope and objectives were explained, focusing on developing an AI agent for enhancing learning experience through the SEEK portal.

- **Communication and Collaboration Setup**
  - The team agreed on the following tools for communication and collaboration:
    - **WhatsApp**: For daily updates
    - **GitHub**: For code collaboration, version control, and task management
    - **Google Meet**: For weekly meetings scheduled every Sunday at 10:00 PM

- **User Stories and SMART Guidelines**
  - The team designed and reviewed user stories to adhere to SMART criteria based on client recommendations.

- **Action Items:**
  - Finalize role-specific responsibilities by January 20, 2025 (Sandeep Kumar).
  - Set up GitHub repositories for collaboration (Entire Team).

SCRUM MEETINGS

2.2 (SCRUM-4 to SCRUM-5) User Interfaces

- **UI Wireframes Presentation**
  - Jyotiraditya and Saima presented various UI wireframes, and the team provided feedback.

- **Database Schema Presentation**
  - Kajol presented the database schema, which was reviewed and refined through team discussions.

- **Storyboard and Wireframes**
  - The storyboard for the application was reviewed and finalized.
  - Low-fidelity wireframes were created for user stories, incorporating usability design guidelines.

- **GitHub Workflow Implementation**
  - The team discussed GitHub best practices for collaboration, including branch creation, regular commits, and submitting pull requests for task completion.

- **Frontend Flow Discussion**
  - The frontend flow was finalized with corrections:
    - Post-login navigation will direct users to the course page instead of a launchpad.
    - Chat interface behavior will include option to reset session, and a simplistic theme.

- **Action Items:**
  - Finalize UI wireframes by February 5, 2025 (Jyotiraditya, Saima).
  - Improve chatbot functionalities by February 12, 2025 (Siddharth).

SCRUM MEETINGS

2.3 (SCRUM-6) Scheduling & Design

- **Role Allocation Matrix Finalization**
  - The finalized role allocation matrix is as follows:

Role	Members
Scrum Master	Sandeep Kumar
Frontend Development	Amit Kulkarni, Anjali Galav, Sandeep Kumar
AI Development	Amit Kulkarni, Siddharth Umathe
Backend Development	Anjali Galav, Kajol Singh, Saima Zainab Shroff
Documentation	Jyotiraditya Saha, Saima Zainab Shroff

- **Feature Development Priority List**
  - The team discussed and agreed on the priority order for feature development based on user stories.

- **Scheduling Activities**
  - The team came up with a proper schedule for sprints and iterations using Kanban board, Gantt chart

- **ER Diagram Presentation**
  - Anjali presented an ER diagram for database design that was reviewed by the team.

- **UML Class Diagram Discussion**
  - The team discussed classes and methods for designing the UML class diagram.

SCRUM MEETINGS

- **Progress Review for Milestone 3 Completion**
  - The team reviewed current progress and outlined next steps for completing Milestone 3.

- **Action Items**
  - Document feature priority list by February 16, 2025 (Saima).
  - Refine ER diagram by February 18, 2025 (Anjali, Kajol).
  - Draft initial UML class diagram by February 18, 2025 (Amit, Jyotiraditya).
  - Work on Milestone 3 report by February 19, 2025 (Jyotiraditya, Saima).

SOFTWARE DESIGN - SYSTEM COMPONENTS

3.1 Frontend

3.1.1 User Interface Components

- **Navigation Bar (AppNavbar)**
  - Positioned at the top of the screen, it includes branding, facilitates navigation between different sections, and provides a logout option.

- **Sidebar (AppSidebar)**
  - A side menu offering quick access to pages such as Course Details, Lecture Videos, Mock Quizzes, and the AI Agent (KIA).

3.1.2 Pages

- **Course Details Page**
  - Displays course-related information, including the syllabus, instructors, and reference materials.
  - Provides links to course-specific resources.
  - Includes an integrated chat feature to ask KIA for guidance.

- **Assignments and Mock Quizzes Pages**
  - Allows students to view and submit assignment responses.
  - Automatically calculates scores and provides improvement suggestions.
  - Enables students to download a detailed performance report.

SOFTWARE DESIGN - SYSTEM COMPONENTS

- **Topic-Specific Question Generator Page**
  - Generates practice tests based on selected topics.
  - Suggests topics as users type in the search bar.
  - Displays scores and AI-generated improvement tips after clicking the “Check Score” button.

- **AI Agent Page (KIA)**
  - Serves as the primary interface for interacting with KIA.
  - Offers options to generate weekly summaries and topic-specific notes.
  - Supports markdown rendering for generated content.
  - Allows downloading notes as PDF files for offline access.

- **Lecture Page**
  - Displays video lectures with additional features, including:
  - An embedded YouTube video player.
  - Video summarization functionality.
  - A copy-to-clipboard feature for summaries.

- **Programming Assignment Page**
  - Provides an environment for coding assignments and exercises.
  - Displays problem statements with input/output examples.
  - Includes an integrated code editor (AceEditor) for writing and testing code.
  - Shows test results and provides AI-powered explanations for errors.

SOFTWARE DESIGN - SYSTEM COMPONENTS

3.1.3 Chat Integration

- **Chat Window**
  - A floating chat interface for interacting with KIA at any time.
  - Maintains previous conversations for reference.
  - Allows users to reset chat history when needed.

3.1.2 Code Editor

- **Code Writing Tool (AceEditor)**
  - Enables users to write and test Python code within the application.
  - Highlights syntax errors and enhances readability with color-coded text.

3.2 Backend

3.2.1 APIs

- **User Authentication APIs:**
  - `/api/signup`:
    Registers a new user with a hashed password.
  - `/api/login`:
    Authenticates users and generates tokens.
- **Course Management APIs:**
  - `/api/weeks`:
    Performs CRUD operations for course weeks.
  - `/api/lectures`:
    Manages lectures within a course week.

SOFTWARE DESIGN - SYSTEM COMPONENTS

- **Video Management API:**
  - `/api/get-video-summary`:
    Generates summaries for video lectures.
- **Assignment Management APIs:**
  - `/api/assignments`:
    CRUD operations for assignments
- **Programming Assignment APIs:**
  - `/api/programming-assignments (POST)`:
    Create new coding assignment
  - `/api/programming-assignments/{id} (GET)`:
    Get assignment details
  - `/api/programming-assignments/{id}/submit (POST)`:
    Submit code solution
  - `/api/programming-assignments/{id}/test-results (GET)`:
    Retrieve test results
  - `/api/programming-assignments/{id}/explain-error (GET)`:
    Get simplified explanation for coding errors

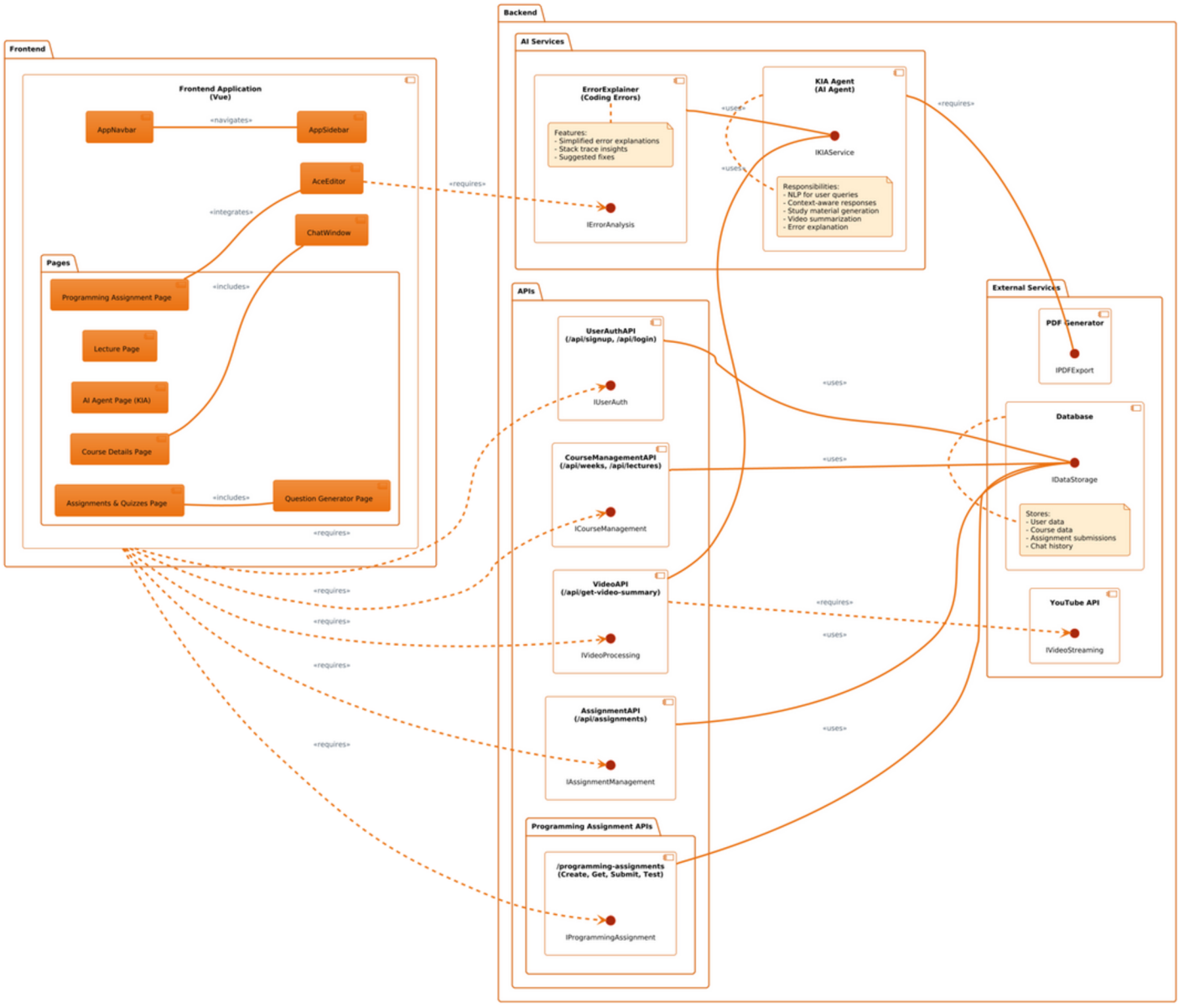
3.2.2 AI Integration

- **KIA (AI Agent)**
  - Powers AI-driven features across the application.
  - **Core Capabilities**:
    - Natural language processing for user queries.
    - Context-aware responses based on course content.
    - Automated content generation.

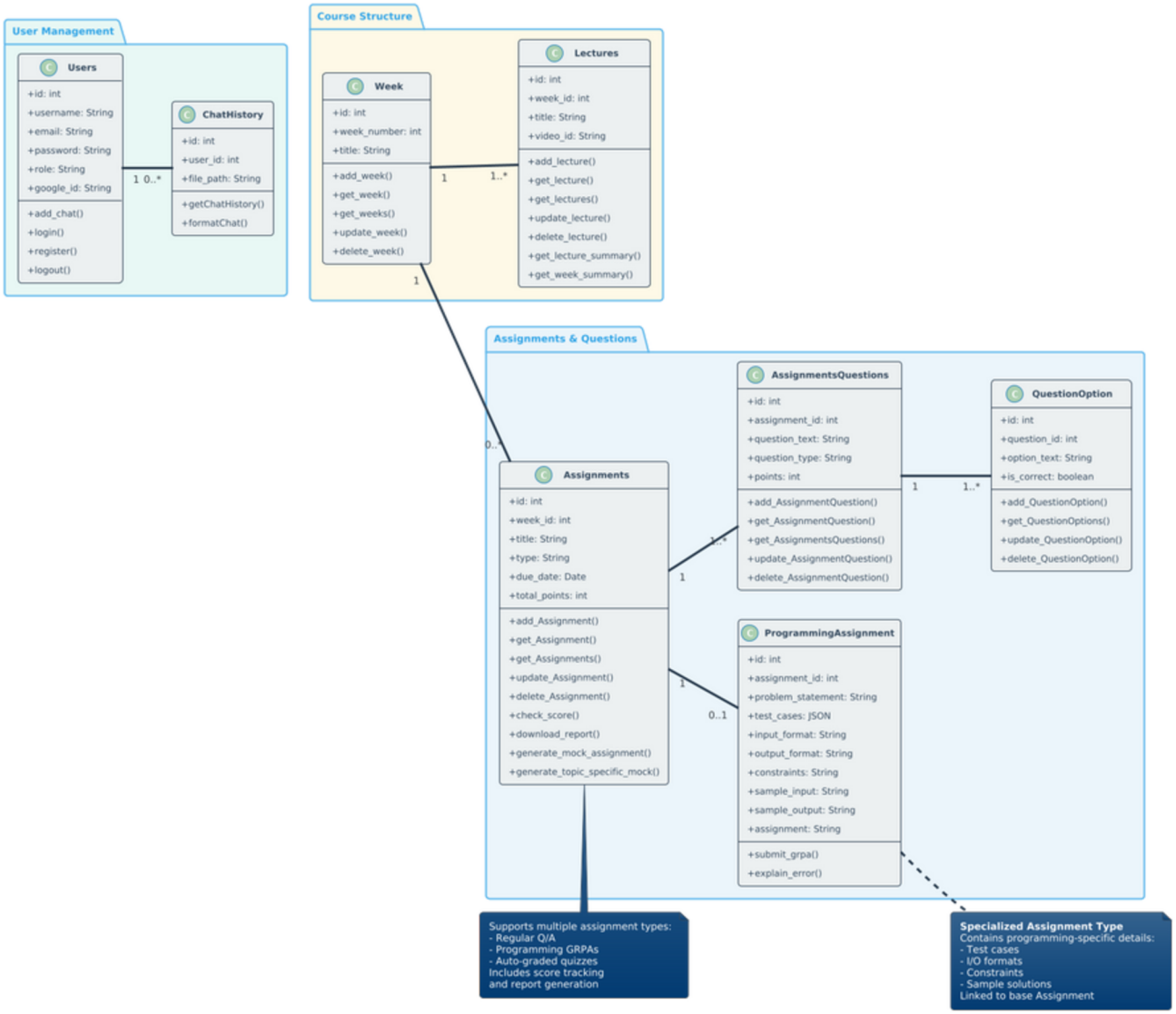
SOFTWARE DESIGN - SYSTEM COMPONENTS

- **Functional Components**:
  - **Video Content Summarization**: Processes video content and generates concise summaries.
  - **Error Explanation**:
    - Provides simplified explanations for coding errors
    - Analyzes error messages and stack traces to generate human-readable insights and suggested fixes.
  - **Generation of study materials**, including topic-specific notes, practice questions, weekly summaries
- **Operational Features:**
  - Asynchronous request handling
  - Markdown rendering support
  - Conversation history management

SOFTWARE DESIGN - SYSTEM COMPONENTS

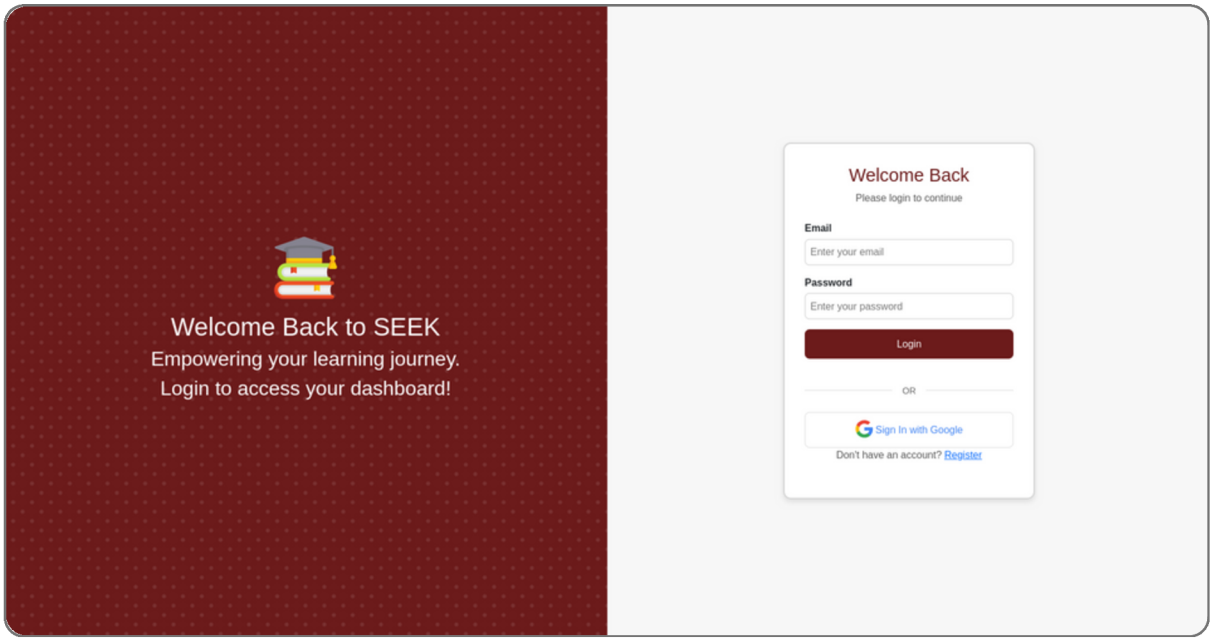


CLASS DIAGRAM

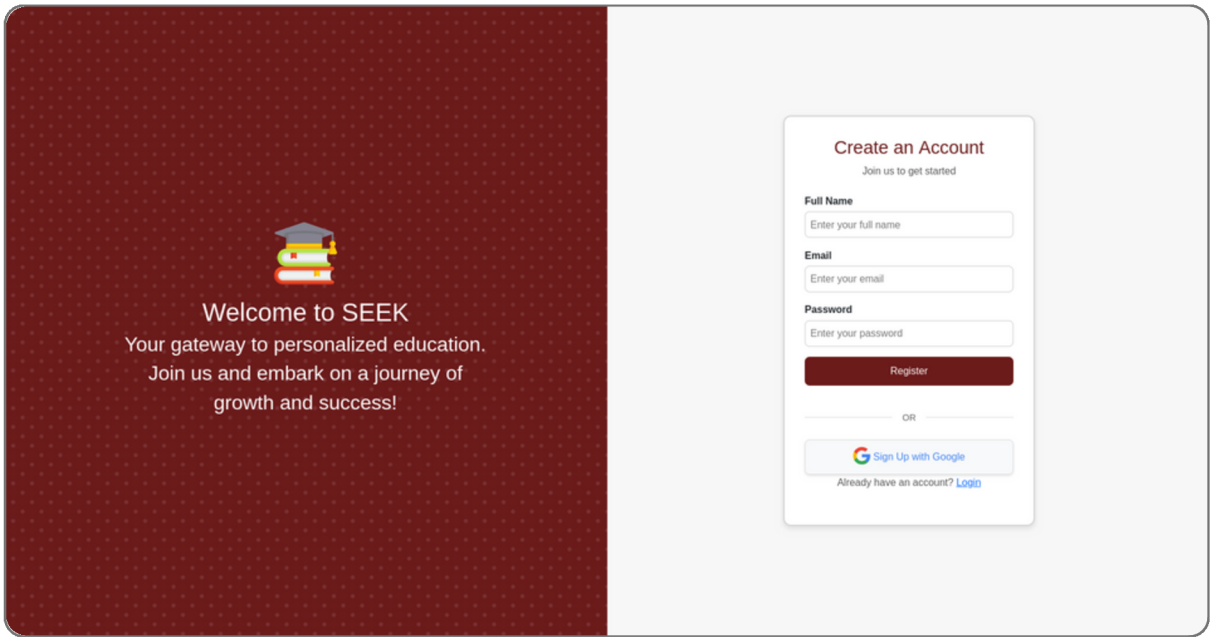


UI PAGES

Login Page

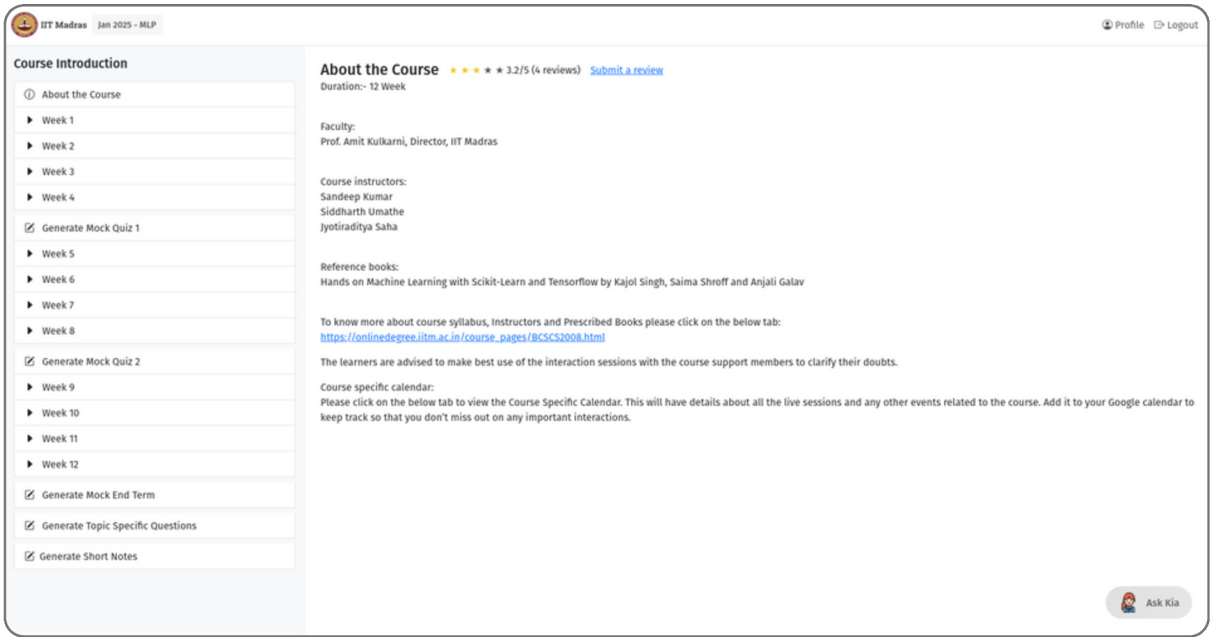


Register Page

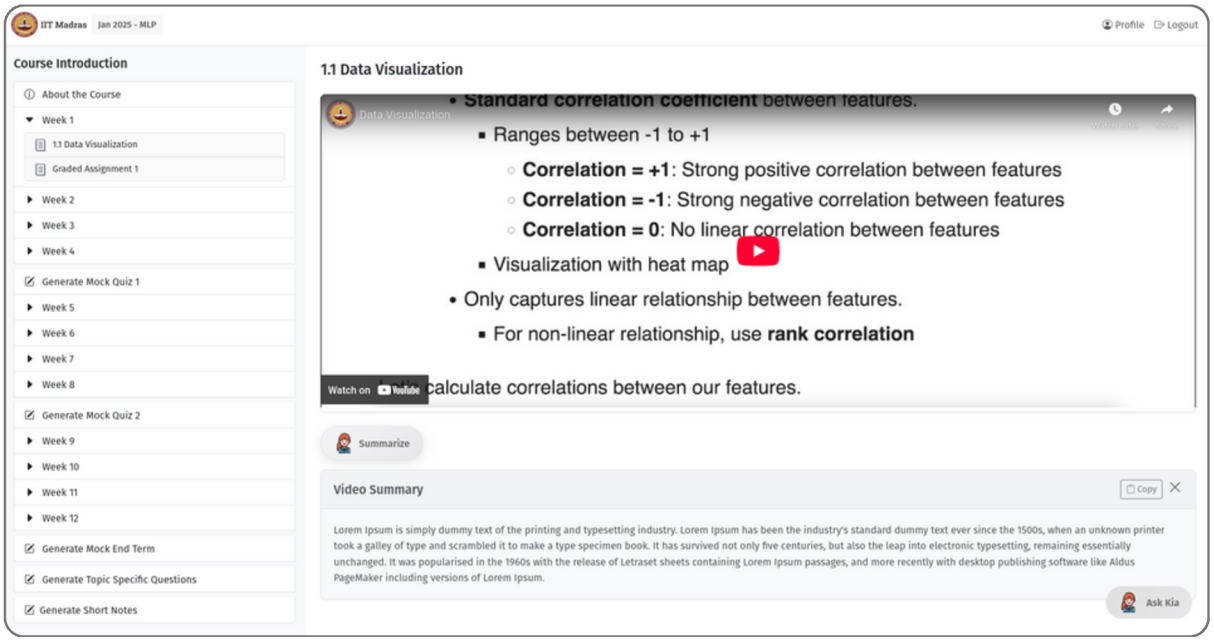


UI PAGES

Course Page

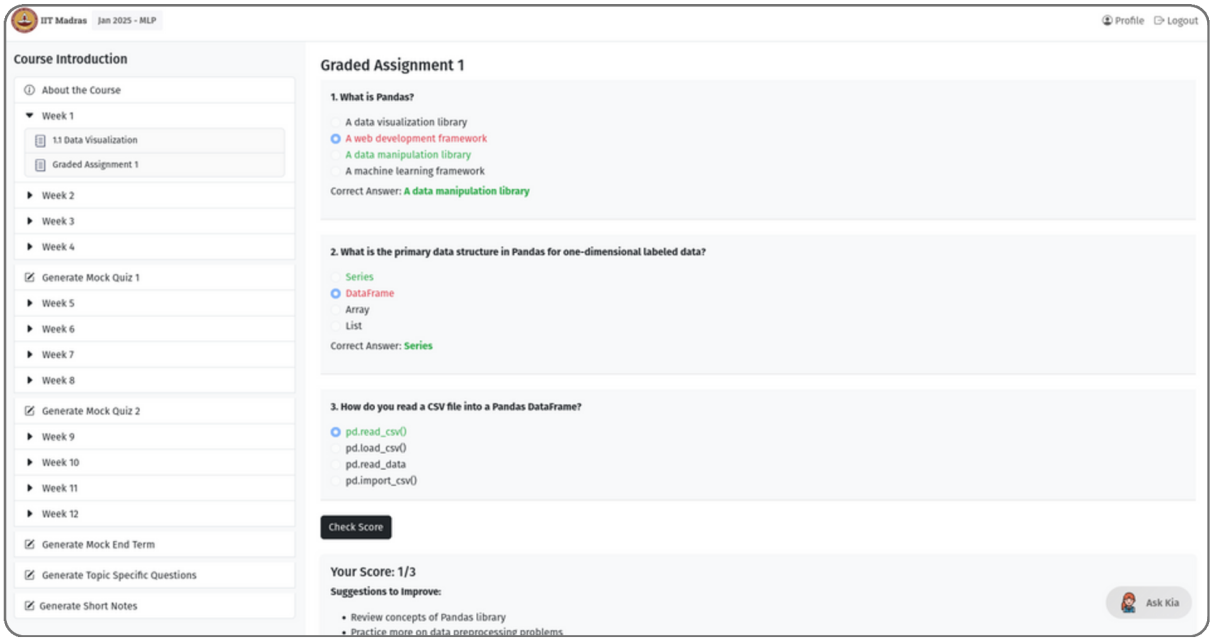


Lecture Page

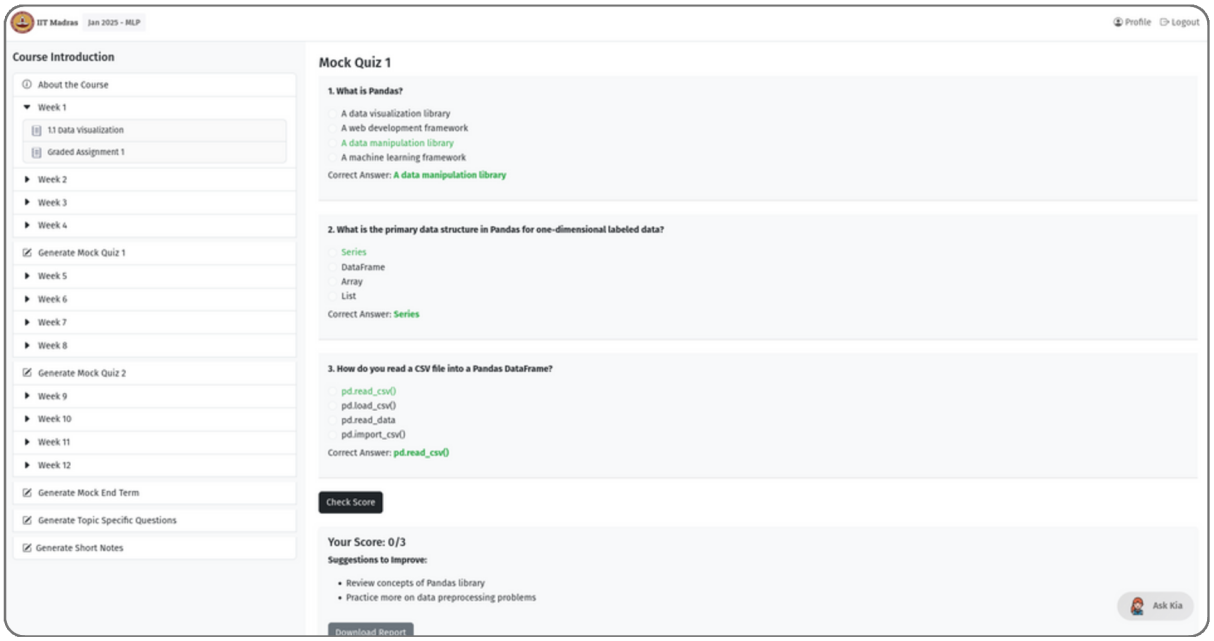


UI PAGES

Graded Assignment Page

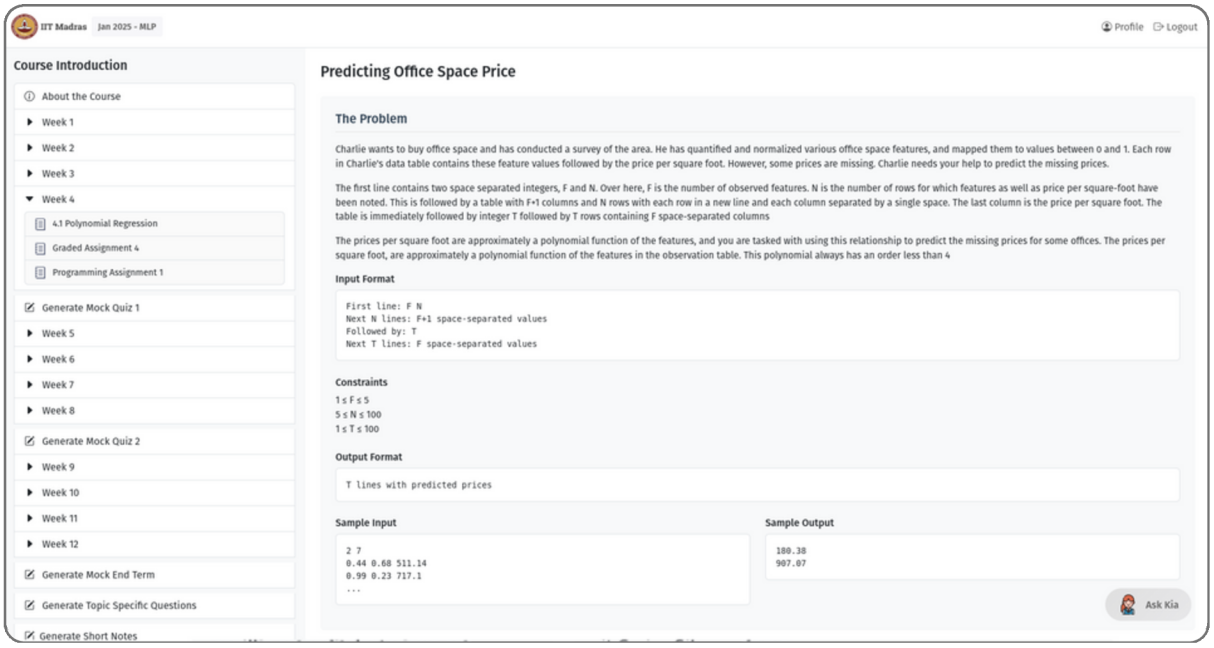


Mock Quiz Page

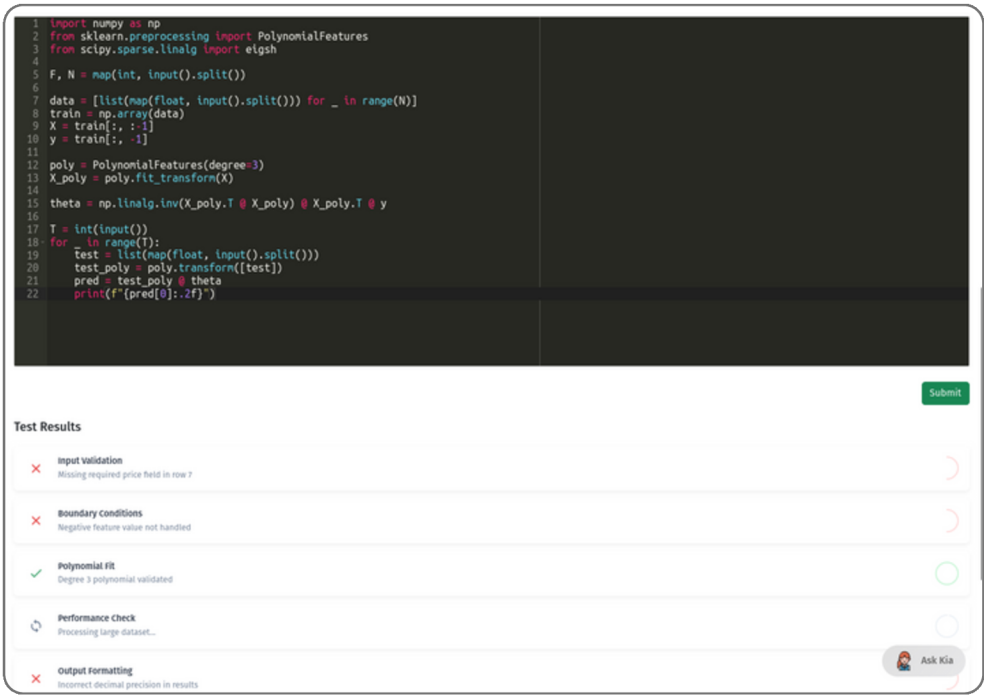


UI PAGES

Programming Assignment Page

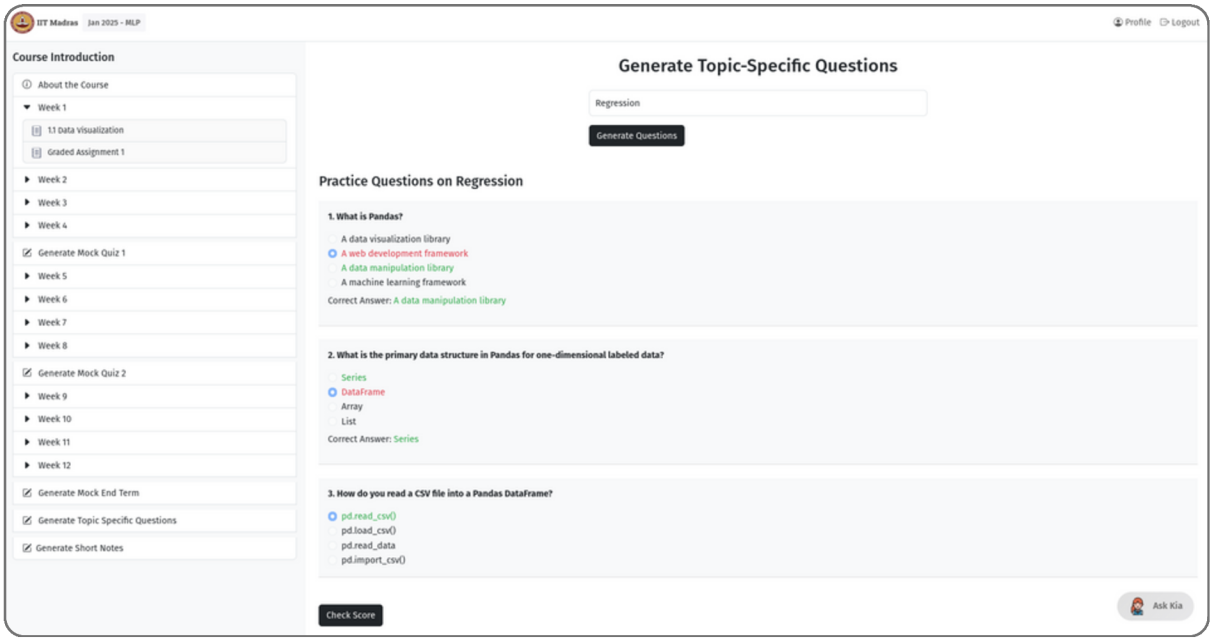


Ace Editor

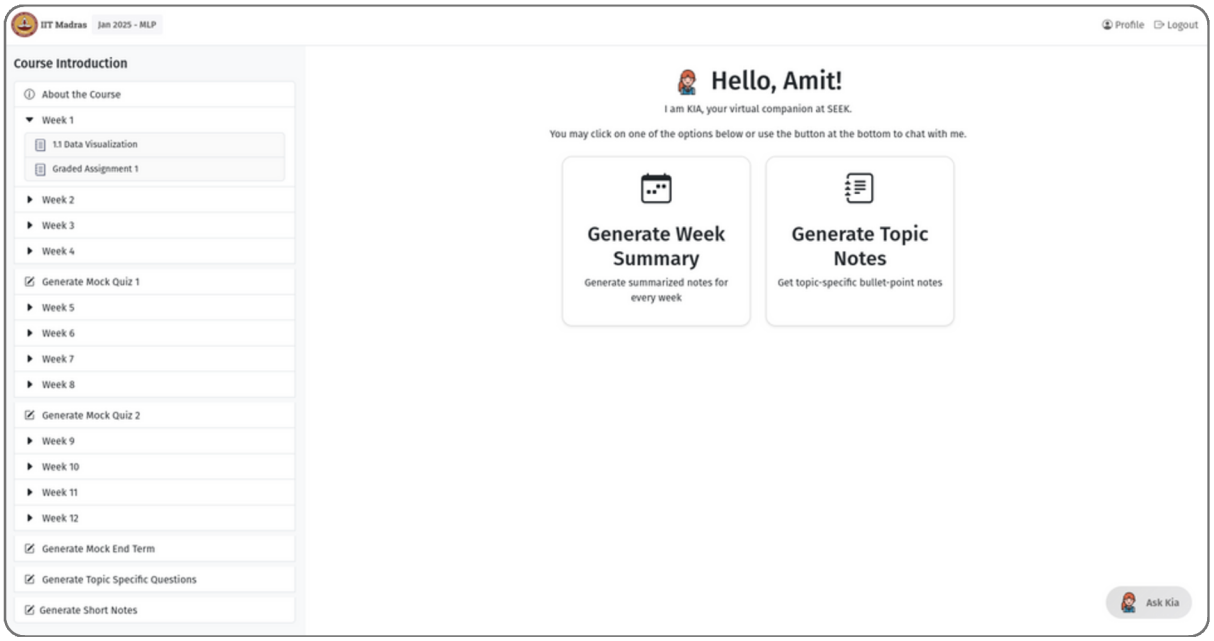


UI PAGES

Topic-Specific Questions Page



Kia Page



UI PAGES

Topic-Specific Notes Page

UI PAGES

Week Summary Page

MILESTONE - 4

API ENDPOINTS

API ENDPOINTS

Base URL: localhost:5000/api

01 User Registration and Authentication

Signup (**/signup**, POST)

- Registers a new user with details such as `username`, `email`, `password`and `role`.

- **Responses**:
  - 201: Registration successful
  - 400: Validation error or duplicate
  - 500: Internal server error

Login (**/login**, POST)

- Authenticates a user with `email` and `password`.

- **Responses**:
  - 200: Login successful with access token
  - 400/401/403: Invalid credentials or Google sign-in required
  - 500: Internal server error

Google Signup (**/google_signup**, POST)

- Registers a user using Google ID token.

- **Responses**:
  - 201: Google signup successful
  - 400/500: Invalid token or server error

API ENDPOINTS

Google Login (**/google_login**, POST)

- Logs in a user via Google authentication.

- **Responses**:
  - 200: Login successful
  - 400/404/500: Validation error or user not registered

02 Week Management

/weeks (GET, POST)

- Retrieves all weeks or creates a new week with `week_number` and `title`.

- **Responses**:
  - 200: Weeks retrieved
  - 201: Week created
  - 400/409/500: Validation or server error

/weeks/{week_id} (GET, PUT, DELETE)

- Manages a specific week's data.
  - GET: Returns week details including `lectures` and `assignments`
  - PUT: Updates `week_number` or `title`
  - DELETE: Removes a specific week

- **Responses**:
  - 200: Success
  - 404/500: Not found or server error

API ENDPOINTS

03 Lecture Management

/lectures (GET, POST)

- Retrieves all lectures or adds a new lecture with `title`, `video_id`, and `week_id`.

/lectures/{lecture_id} (GET, PUT, DELETE)

- Performs operations on a specific lecture.

- **Responses**:
  - 200: Success
  - 404: Lecture not found
  - 500: Server error

04 Assignment Management

/assignments (GET, POST)

- Retrieves all assignments or creates one using `week_id`, `title`, `assignment_type`, and `due_date`.

/assignments/{assignment_id} (GET, PUT, DELETE)

- Manages a specific assignment, including `questions` and `total_points`.

- **Responses**:
  - 200: Success
  - 404/500: Not found or server error

API ENDPOINTS

05 Assignment Questions

/assignment_questions (GET, POST)

- Retrieves or creates questions under an assignment, including `question_text`, `type`, and `points`.

/assignment_questions/{question_id} (GET, PUT, DELETE)

- Retrieves, updates, or deletes a specific question and its options.

- **Responses**:
  - 200: Success
  - 404/500: Not found or server error

06 Question Options

/options (GET, POST)

- Retrieves all options or adds one to a question. Requires `option_text` and `is_correct`.

/options/{option_id} (GET, PUT, DELETE)

- Manages a specific option's content.

- **Responses**:
  - 200: Success
  - 404/500: Not found or server error

API ENDPOINTS

07 Programming Assignments

/programming_assignments (POST)

- Creates a coding assignment using fields like `problem_statement`,`input_format`,`output_format and `test_cases`.

- **Responses:**
  - 201: Created
  - 400/409/500: Validation or server error

/programming_assignments/{assignment_id}/execute (POST)

- Executes submitted Python code against a set of predefined test cases for a given programming assignment.

06 Gen-AI Functionalities

/explain_error (POST)

- Explains programming errors in user-submitted code using GenAI.

/chat_history (GET, POST)

- Manages chat queries and retains last five conversational entries.

/topic_recommendation (POST)

- Recommends weak topics based on quiz performance.

API ENDPOINTS

/video_summarizer (POST)

- Summarizes lecture video into digestible bullet points.

/generate_notes (POST)

- Generates structured notes from video content.

/generate_week_summary (POST)

- Provides a holistic summary of the week's course content.

/generate_mock (POST)

- Generates mock questions for practice.

/generate_topic_specific_questions (POST)

- Creates practice questions for a specific topic.

06 Other Functionalities

/download_report (POST)

- Generates a personalized PDF report summarizing mock quiz performance.

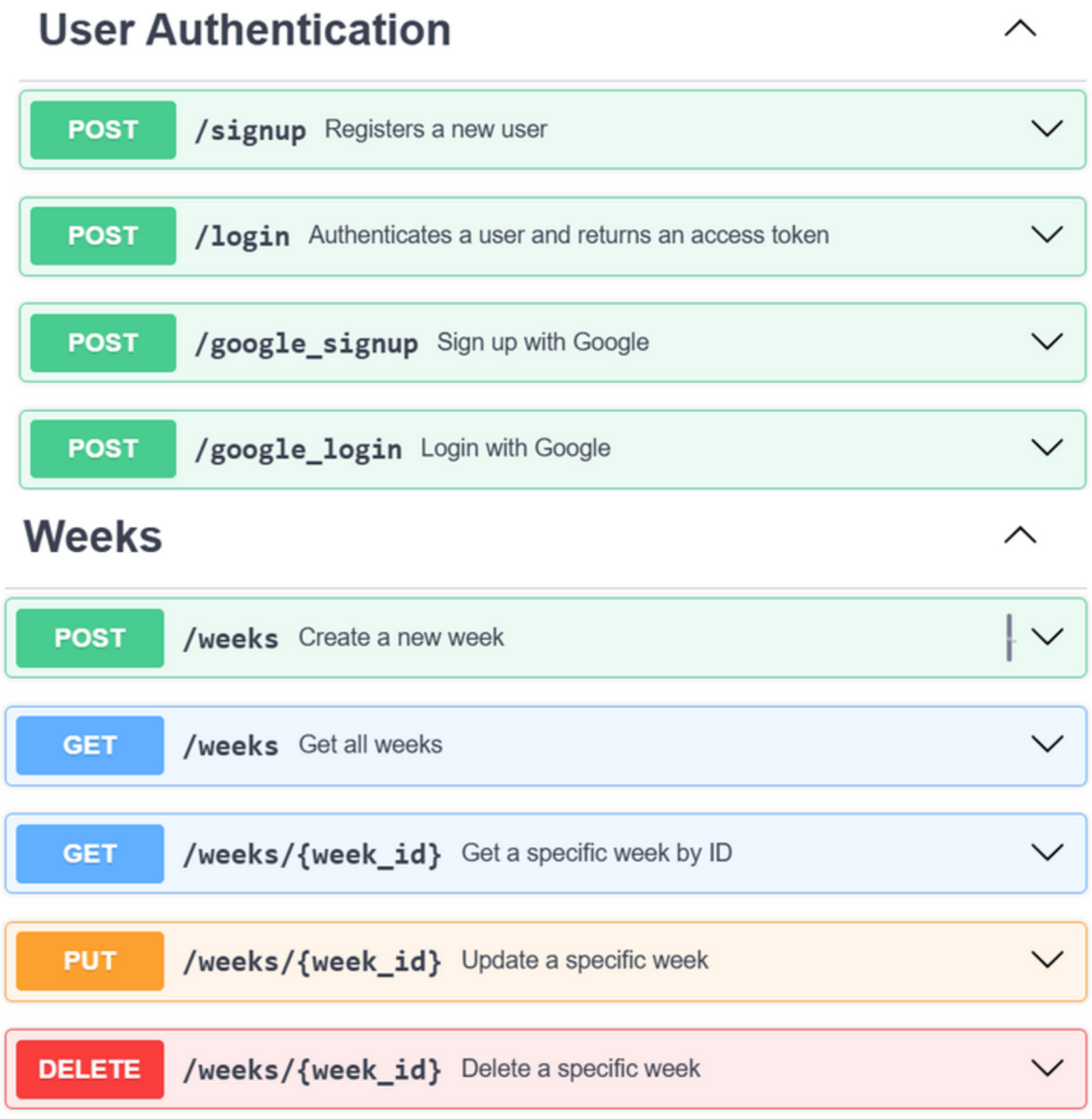
- **Responses:**
  - 200: PDF report downloaded successfully
  - 400: Missing required fields
  - 500: PDF generation or file sending failed

API ENDPOINTS

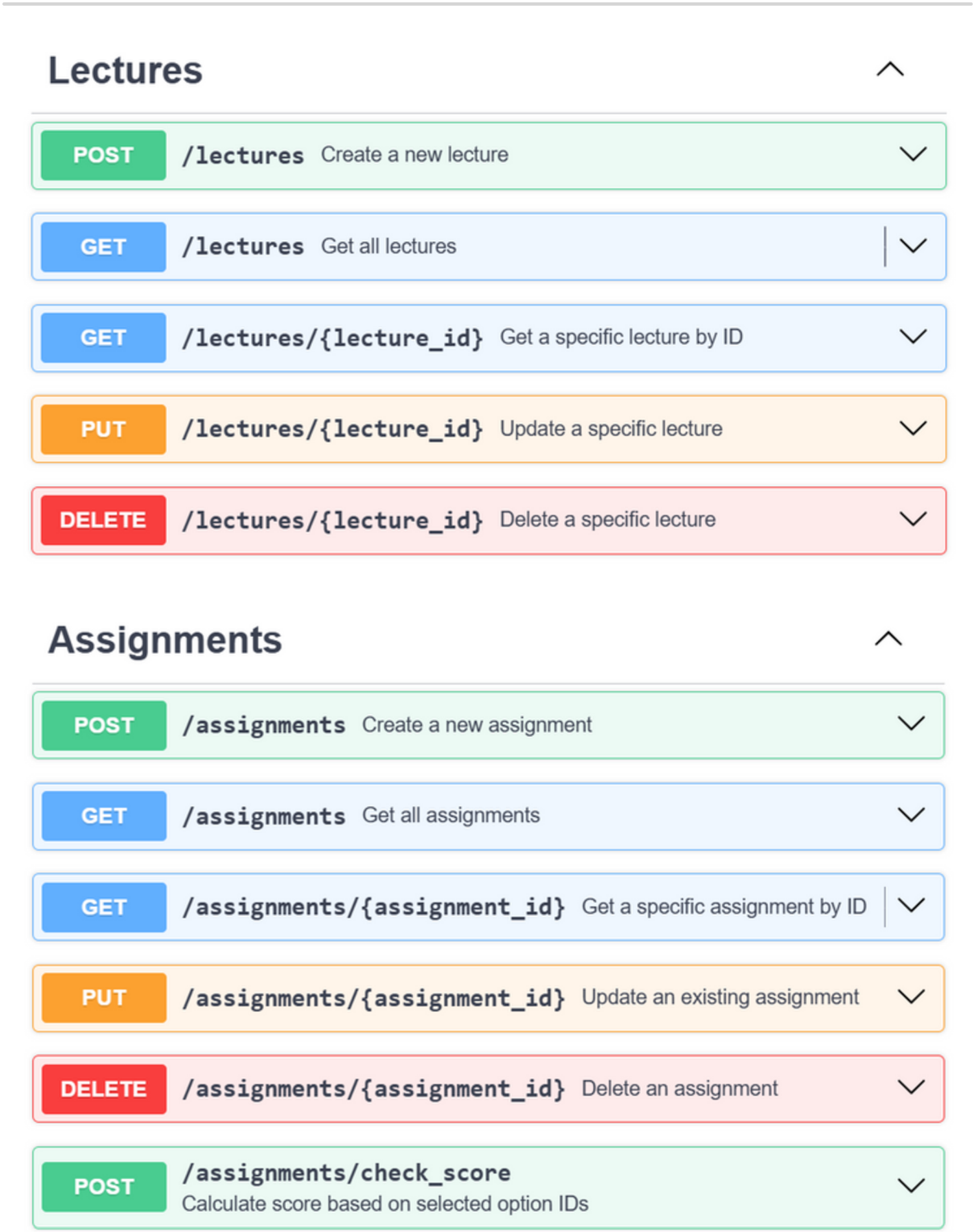
/download_markdown_pdf (POST)

- Converts markdown content (including LaTeX expressions) into a styled PDF and returns it for download.

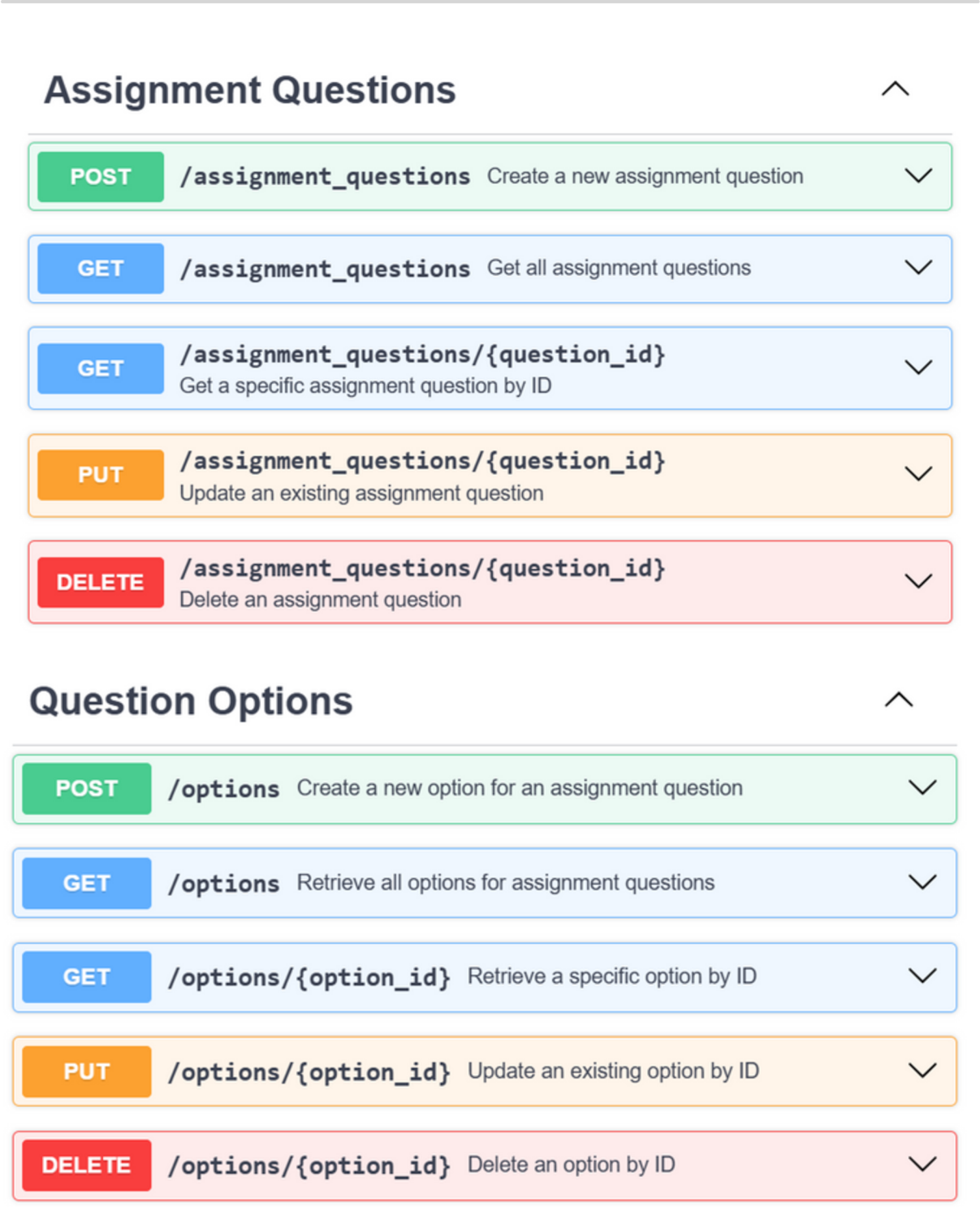
- **Responses:**
  - 200: PDF generated and sent successfully
  - 400: Missing markdown content
  - 500: PDF generation failed (includes stack trace)



API ENDPOINTS

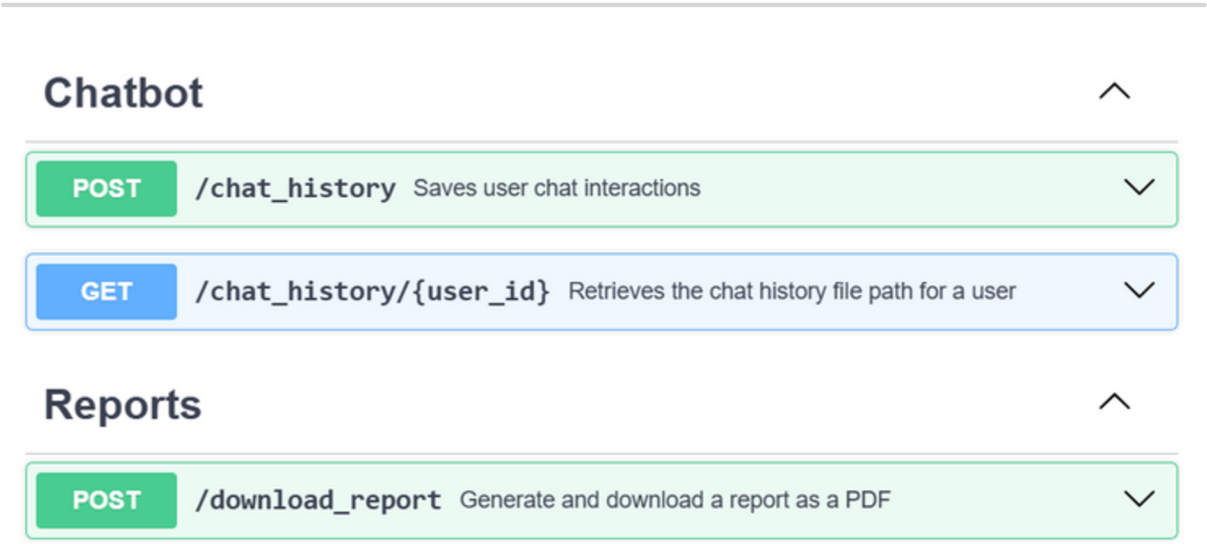


API ENDPOINTS



API ENDPOINTS

API ENDPOINTS



MILESTONE - 5

TEST SUITE

AI AGENT APIs

/api/generate_topic_specific_questions

METHOD: POST

- **Test Case: Generate practice questions for a particular topic - default**

```
def test_generate_topic_specific_questions_success(client):
    """Test successful generation of topic-specific questions."""
    payload = {
        "topic": "Data Visualization Libraries",
        "num_questions": 3
    }
    response = client.post('/generate_topic_specific_questions',
json=payload)
    data = response.get_json()

    assert response.status_code == 200
    assert data['success'] is True
    assert 'questions' in data
    assert len(data['questions']) == 3
    assert 'question' in data['questions'][0]
    assert 'options' in data['questions'][0]
    assert 'answer' in data['questions'][0]
    assert 'explanation' in data['questions'][0]
```

Input Data

• JSON
{
"topic": "Data Visualization Libraries",
"num_questions": 3
}

Expected Output	Actual Output
• Status Code: 200 • JSON { "success": true, "questions": [{ "question": "...,	• Status Code: 200 • JSON { "success": true, "questions": [{ "question": "...,

AI AGENT APIs

"options": ["...", "...", "..."], "answer": "...", "explanation": "..." }] }	"options": ["...", "...", "..."], "answer": "...", "explanation": "..." }] }
Result: **Success**	

- **Test Case: Generate practice questions for a particular topic with invalid question count (zero)**

```
def test_generate_topic_specific_questions_zero_questions(client):
    """Test API with zero questions requested."""
    payload = {"topic": "Artificial Intelligence", "num_questions": 0}
    response = client.post('/generate_topic_specific_questions',
json=payload)
    data = response.get_json()

    assert response.status_code == 400
    assert data['success'] is False
    assert data['message'] == 'Number of questions must be at least 1'
```

Input Data	

• JSON
{
"topic": "Artificial Intelligence",
"num_questions": 0
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Number of questions must be at least 1"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Number of questions must be at least 1"
	}

Result: **Success**	

AI AGENT APIs

- **Test Case: Generate practice questions for a particular topic with invalid question count (negative number)**

```
def test_generate_topic_specific_questions_negative_questions(client):
    """Test API with negative value for `num_questions`."""
    payload = {"topic": "Cyber Security", "num_questions": -3}
    response = client.post('/generate_topic_specific_questions',
json=payload)
    data = response.get_json()

    assert response.status_code == 400
    assert data['success'] is False
    assert data['message'] == 'Number of questions must be at least 1'
```

Input Data	

• JSON
{
"topic": "Cyber Security",
"num_questions": -3
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Number of questions must be at least 1"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Number of questions must be at least 1"
	}

Result: **Success**	

- **Test Case: Generate practice questions for a particular topic topic is missing in the input payload**

```
def test_generate_topic_specific_questions_missing_topic(client):
    """Test API when topic is missing in the payload."""
    payload = {"num_questions": 3}
    response = client.post('/generate_topic_specific_questions',
json=payload)
```

AI AGENT APIs

```
    data = response.get_json()

    assert response.status_code == 400
    assert data['success'] is False
    assert data['message'] == 'Topic is required'
```

Input Data	

• JSON
{
"num_questions": 3
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Topic is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Topic is required"
	}

Result: **Success**	

- **Test Case: Generate practice questions for a particular topic with invalid data types in the input payload**

```
def test_generate_topic_specific_questions_invalid_data_type(client):
    """Test API with invalid data types in the payload."""
    payload = {"topic": 12345, "num_questions": "three"}
    response = client.post('/generate_topic_specific_questions',
json=payload)
    data = response.get_json()

    assert response.status_code == 400
    assert data['success'] is False
    assert data['message'] == 'Invalid data type for num_questions'
```

AI AGENT APIs

Input Data	

• JSON
{
"topic": 12345,
"num_questions": "three"
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Invalid data type for num_questions"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Invalid data type for num_questions"
	}

Result: **Success**	

/api/video_summarizer

METHOD: POST

- **Test Case: Generate video summary with a valid lecture ID - default**

```
def test_video_summarizer_success(client):
    """Test successful video summarization with valid lecture_id."""

    payload = {"lecture_id": "2"}
    response = client.post('/video_summarizer', json=payload)
    data = response.get_json()

    assert response.status_code == 200
    assert data['success'] is True
```

AI AGENT APIs

Input Data	

• JSON
{
"lecture_id": "2"
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Summary generated successfully",
"summary": "..."
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Summary generated successfully",
	"summary": "..."
	}

Result: **Success**	

- **Test Case: Generate video summary with missing lecture ID**

```python
def test_video_summarizer_missing_lecture_id(client):
    """Test API when `lecture_id` is missing in the payload."""
    payload = {}
    response = client.post('/video_summarizer', json=payload)
    data = response.get_json()

    assert response.status_code == 400
    assert data['success'] is False
    assert data['message'] == 'lecture_id is required'
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "lecture_id is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "lecture_id is required",
	}

Result: **Success**	

AI AGENT APIs

- **Test Case: Generate video summary with non-existing lecture ID**

```python
def test_video_summarizer_lecture_not_found(client):
    """Test API when the requested lecture_id is not found."""

    payload = {"lecture_id": "99_99"}
    response = client.post('/video_summarizer', json=payload)
    data = response.get_json()

    assert response.status_code == 404
    assert data['success'] is False
    assert data['message'] == 'Lecture not found'
```

Input Data	

• JSON
{
"lecture_id": "99_99"
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Lecture not found"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Lecture not found",
	}

Result: Success	

AI AGENT APIs

/api/chat_history

METHOD: POST

- **Test Case: Chat history stored successfully**

```
def test_chatbot_response(client):
    """Test if chatbot returns a valid response"""
    input_data = {
        "user_id": 1,
        "query": "What is machine learning?",
        "response": "Machine learning is a subfield of artificial
intelligence."
    }
    response = client.post('/chat_history', json=input_data)

    assert response.status_code == 201

    data = response.get_json()

    assert data["success"] is True
    assert data["message"] == "Chat history saved successfully"
    assert data["user_id"] == 1
```

Input Data	

• JSON
{
"user_id": 1,
"query": "What is machine learning?",
"response": "Machine learning is a subfield of artificial intelligence."
}	

Expected Output	**Actual Output**

• Status Code: 201
• JSON
{
"success": true,
"message": "Chat history saved successfully",
"file_path": "..."
}
	• Status Code: 201
	• JSON
	{
	"success": true,
	"message": "Chat history saved successfully",
	"file_path": "..."
	}

Result: Success	

AI AGENT APIs

- Test Case: Query not provided

```python
def test_chatbot_missing_query(client):
    """Test chatbot response when the query is missing"""
    input_data = {
        "user_id": 2  # No query provided
    }
    response = client.post('/chat_history', json=input_data)

    assert response.status_code == 400

    data = response.get_json()

    assert data["success"] is False
    assert data["message"] == "Missing required fields"
```

Input Data	

• JSON
{
"user_id": 1,
"query": "What is machine learning?",
"response": "Machine learning is a subfield of artificial intelligence."
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Missing required fields"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Missing required fields"
	}

Result: **Success**	

AI AGENT APIs

/api/explain_error

METHOD: POST

- **Test Case: Error explanation generated successfully**

```
def test_explain_error_success(client):
    """Test API with a valid code snippet that contains an error."""
    input_data = {
        "code_snippet": "print(1/0)"  # Causes ZeroDivisionError
    }
    response = client.post('/explain_error', json=input_data)

    assert response.status_code == 200
    assert response.json["success"] is True
    assert response.json["message"] == "Error explanation generated
successfully"
    assert "explanation" in response.json
    assert "divide by zero" in response.json["explanation"]
```

Input Data	

• JSON
{
"code_snippet": "print(1/0)"
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Error explanation generated successfully",
"explanation": "..."
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Error explanation generated successfully",
	"explanation": "..."
	}

Result: Success	

AI AGENT APIs

- Test Case: Missing code snippet

```python
def test_explain_error_missing_code_snippet(client):
    """Test API when the request body is missing the 'code_snippet'
key."""
    input_data = {}
    response = client.post('/explain_error', json=input_data)

    assert response.status_code == 400
    assert response.json["success"] is False
    assert response.json["message"] == "Code snippet is required"
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Code snippet is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Code snippet is required"
	}

Result: **Success**	

/api/generate_week_summary

METHOD: POST

- Test Case: Week summary generated successfully

```python
#Test case for successful summary generation when the week exists.
def test_generate_week_summary_success(client):
    response = client.post('/generate_week_summary', json={"week_id": 2})
    assert response.status_code == 200
    assert response.json['success'] is True
    assert 'summary' in response.json
```

AI AGENT APIs

Input Data	

• JSON
{
"week_id": 2
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Week summary generated successfully",
"week_id": 2,
"summary": "..."
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Week summary generated successfully",
	"week_id": 2,
	"summary": "..."
	}

Result: **Success**	

- **Test Case: Missing week ID**

```python
#Test case when `week_id` is missing in the request body.
def test_generate_week_summary_missing_week_id(client):
    response = client.post('/generate_week_summary', json={})

    assert response.status_code == 400
    assert response.json['success'] is False
    assert response.json['message'] == 'week_id is required'
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "week_id is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "week_id is required"
	}

Result: **Success**	

AI AGENT APIs

- Test Case: Missing week ID

```
#Test case when `week_id` does not exist in the database.
def test_generate_week_summary_non_existent_week(client):
    response = client.post('/generate_week_summary', json={"week_id":
999})

    assert response.status_code == 404
    assert response.json['success'] is False
    assert response.json['message'] == 'Week not found'
```

Input Data	

• JSON
{
"week_id": 999
}	

Expected Output	**Actual Output**

• Status Code: 404
• JSON
{
"success": false,
"message": "Week not found"
}
	• Status Code: 404
	• JSON
	{
	"success": false,
	"message": "Week not found"
	}

Result: **Success**	

/api/generate_mock

METHOD: POST

- Test Case: Mock test generated successfully

```
#Test case for successful generation of a mock test
def test_generate_mock_success(client):
    response = client.post('/generate_mock', json={'quiz_type': 'quiz1',
'num_questions': 10})

    assert response.status_code == 200
    data = response.get_json()
    assert data['success'] is True
    assert 'questions' in data
```

AI AGENT APIs

Input Data	

• JSON
{
"quiz_type": "quiz1",
"num_questions": 10
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"message": "Mock test generated successfully for quiz1",
"success": true,
"quiz_type": "quiz1",
"num_questions": 10,
"questions": [
{
"question": "...",
"options": [".."],
"correct_answer": "..."
}
]
}
	• Status Code: 200
	• JSON
	{
	"message": "Mock test generated successfully for quiz1",
	"success": true,
	"quiz_type": "quiz1",
	"num_questions": 10,
	"questions": [
	{
	"question": "...",
	"options": [".."],
	"correct_answer": "..."
	}
	]
	}

Result: **Success**	

- **Test Case: Missing quiz type field**

```python
# Test case: Missing quiz_type field
def test_generate_mock_missing_quiz_type(client):
    response = client.post('/generate_mock', json={
        "num_questions": 5
    })

    assert response.status_code == 400
    data = response.get_json()
    assert data['success'] is False
    assert data['message'] == 'quiz_type is required'
```

AI AGENT APIs

Input Data	

• JSON
{
"num_questions": 5
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "quiz_type is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "quiz_type is required"
	}

Result: **Success**	

- **Test Case: Invalid quiz type**

```python
# Test case: Non-existent quiz_type
def test_generate_mock_non_existent_quiz_type(client):
    response = client.post('/generate_mock', json={
        'quiz_type': 'unknown_quiz',
        'num_questions': 5
    })

    assert response.status_code == 404
    data = response.get_json()
    assert data['success'] is False
    assert 'message' in data
```

Input Data	

• JSON
{
"quiz_type": "unknown_quiz",
"num_questions": 5
}	

Expected Output	**Actual Output**
• Status Code: 404	• Status Code: 404

AI AGENT APIs

• JSON
{
"success": false,
"message": "..."
}
	• JSON
	{
	"success": false,
	"message": "..."
	}

Result: **Success**	

/api/generate_notes

METHOD: POST

- **Test Case: Notes generated successfully**

```python
#Test Case: Generate Notes Successfully
def test_generate_notes_success(client):
    response = client.post('/generate_notes', json={"topic":
"Reinforcement Learning"})

    assert response.status_code == 200
    assert response.json['success'] is True
    assert response.json['message'] == 'Notes generated successfully for
topic "Reinforcement Learning"'
    assert 'notes' in response.json
```

Input Data	

• JSON
{
"topic": "Reinforcement Learning"
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Notes generated successfully for topic "Reinforcement Learning"",
"notes": "..."
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Notes generated successfully for topic "Reinforcement Learning"",
	"notes": "..."
	}

Result: **Success**	

AI AGENT APIs

- Test Case: Missing topic

```python
#Test Case: missing topic
def test_generate_notes_missing_topic(client):
    response = client.post('/generate_notes', json={})

    assert response.status_code == 400
    assert response.json['success'] is False
    assert response.json['message'] == 'topic is required'
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "topic is required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "topic is required"
	}

Result: **Success**	

/api/topic_recommendation

METHOD: POST

- Test Case: Topic recommendation based on incorrect quiz responses

```python
# Test case for successful recommendation generation based on incorrect answers
def test_topic_recommendation_success(client):
    # Mock data for incorrect answers
    submitted_answers = [
        {"question_id": 1, "selected_option_id": 2},
        {"question_id": 2, "selected_option_id": 5}
    ]

    # Make the request
```

AI AGENT APIs

```
    response = client.post('/topic_recommendation',
                           json={"answers": submitted_answers})

    # Assertions
    assert response.status_code == 200
    assert response.json['success'] is True

    # Check structure exists
    assert 'message' in response.json
    assert 'suggestions' in response.json

    # Check suggestions has expected structure
    assert 'overall_assessment' in response.json['suggestions']
    assert 'topic_suggestions' in response.json['suggestions']
    assert 'general_tips' in response.json['suggestions']

    # Check content presence rather than exact equality
    assert len(response.json['suggestions']['topic_suggestions']) > 0
    assert len(response.json['suggestions']['general_tips']) > 0

    # For text fields, check that key phrases are present
    assert 'programming' in
response.json['suggestions']['overall_assessment'].lower()
```

Input Data	

• JSON
{
"answers": [
{"question_id": 1, "selected_option_id": 2},
{"question_id": 2, "selected_option_id": 5}
]
}	

Expected Output	**Actual Output**
• Status Code: 200 • JSON { "success": true, "message": "All answers	• Status Code: 200 • JSON { "success": true, "message": "All answers are

AI AGENT APIs

are correct! Great job!",
"suggestions": {
"overall_assessment":
"All questions were answered
correctly. Excellent
performance!",
"topic_suggestions":
[],
"general_tips": []
}
}
	correct! Great job!",
	"suggestions": {
	"overall_assessment":
	"All questions were answered
	correctly. Excellent
	performance!",
	"topic_suggestions":
	[],
	"general_tips": []
	}
	}

Result: **Success**	

- **Test Case: Topic recommendation when all answers are correct**

```
# Test case for when all answers are correct
def test_topic_recommendation_all_correct(client):
    # Mock data for correct answers
    submitted_answers = [
        {"question_id": 1, "selected_option_id": 3},
        {"question_id": 2, "selected_option_id": 6}
    ]

    # Make the request
    response = client.post('/topic_recommendation',
                           json={"answers": submitted_answers})

    # Assertions
    assert response.status_code == 200
    assert response.json['success'] is True
    assert "All answers are correct! Great job!" in
response.json['message']
    assert response.json['suggestions']['overall_assessment'] == "All
questions were answered correctly. Excellent performance!"
    assert len(response.json['suggestions']['topic_suggestions']) == 0
```

AI AGENT APIs

Input Data	

• JSON
{
"answers": [
{"question_id": 1, "selected_option_id": 3},
{"question_id": 2, "selected_option_id": 6}
]
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Topic suggestions generated successfully",
"suggestions": {
"overall_assessment": "You need to improve in programming basics",
"topic_suggestions": ["Variables", "Data Types"],
"general_tips": ["Practice daily"]
}
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Topic suggestions generated successfully",
	"suggestions": {
	"overall_assessment": "You need to improve in programming basics",
	"topic_suggestions": ["Variables", "Data Types"],
	"general_tips": ["Practice daily"]
	}
	}

Result: **Success**	

- **Test Case: Topic recommendation when answers are missing**

```python
# Test case for missing answers
def test_topic_recommendation_missing_answers(client):
    # Request with empty answers array
    response = client.post('/topic_recommendation', json={"answers": []})

    # Assertions
    assert response.status_code == 400
    assert response.json['success'] is False
    assert response.json['message'] == 'Answers are required'

    # Request with missing answers field
```

AI AGENT APIs

```
    response = client.post('/topic_recommendation', json={})

    # Assertions
    assert response.status_code == 400
    assert response.json['success'] is False
    assert response.json['message'] == 'Answers are required'
```

Input Data	

• JSON
{
"answers": []
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success": false,
"message": "Answers are required"
}
	• Status Code: 400
	• JSON
	{
	"success": false,
	"message": "Answers are required"
	}

Result: **Success**	

/api/download_report

METHOD: POST

- **Test Case: Notes generated successfully**

```
# Test case for successful report generation and download
def test_download_report_success(client):
    # Test data with all required fields
    test_data = {
        "username": "testuser",
        "score": 85,
        "total": 100,
        "suggestions": ["Study more", "Practice regularly"],
        "questions": [{"question": "What is 2+2?", "answer": "4"}]
    }

    response = client.post('/download_report', json=test_data)
```

AI AGENT APIs

```
    assert response.status_code == 200
    # send_file was successfully called and returned our mock response
```

Input Data

• JSON
{
"username": "testuser",
"score": 85,
"total": 100,
"suggestions": [
"..."
],
"questions": [
{
"question_text": "...",
"options": [
"..."
],
"correct_answer": "..."
}
]
}

Expected Output	Actual Output

• Status Code: 200
• JSON
{
"success": true,
"message": "PDF generated and downloaded successfully",
"file_path": "..."
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "PDF generated and downloaded successfully",
	"file_path": "..."
	}

Result: **Success**	

- **Test Case: Missing required fields**

```
# Test case for missing required fields
def test_download_report_missing_fields(client):
    # Test with missing username
    response = client.post('/download_report', json={"score": 85, "total":
100})
```

```
    assert response.status_code == 400
    assert response.json['success'] is False
    assert response.json['message'] == "Invalid input: 'username',
'score', and 'total' are required fields."
```

Input Data	

• JSON
{
"score": 85,
"total": 100
}	

Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"success":false,
"message": "PDF generated and downloaded successfully""Invalid input: 'username', 'score', and 'total' are required fields."
}
	• Status Code: 400
	• JSON
	{
	"success":false,
	"message": "PDF generated and downloaded successfully""Invalid input: 'username', 'score', and 'total' are required fields."
	}

Result: **Success**	

AUTHENTICATION APIs

/api/google_signup

METHOD: POST

- **Test Case: Signup with Google account authentication (New User) - default**

```
# Test case for successful signup (new user)
def test_google_signup_success(client):
    response = client.post('/google_signup', json={"access_token":
"valid_token"})

    assert response.status_code == 201
    assert response.json['Success'] is True
    assert response.json['access_token'] == 'mock_jwt_token'
    assert response.json['message'] == 'User registered successfully'
```

Input Data	

• JSON
{
"access_token": "valid_token"
}	

Expected Output	**Actual Output**

• Status Code: 201
• JSON
{
"Success": true,
"access_token": "mock_jwt_token",
"message": "User registered successfully"
}
	• Status Code: 201
	• JSON
	{
	"Success": true,
	"access_token": "mock_jwt_token",
	"message": "User registered successfully"
	}

Result: Success	

AUTHENTICATION APIs

- **Test Case: Signup with Google account authentication with missing access token**

```
# Test case for missing access token
def test_google_signup_missing_token(client):
    response = client.post('/google_signup', json={})

    assert response.status_code == 400
    assert response.json['Success'] is False
    assert response.json['message'] == 'Google access token is required'
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"Success": false,
"message": "Google access token is required"
}
	• Status Code: 400
	• JSON
	{
	"Success": false,
	"message": "Google access token is required"
	}

Result: **Success**	

/api/google_login

METHOD: POST

- **Test Case: Login with Google account authentication - default**

```
# Test case for successful login
def test_google_login_success(client):
    response = client.post('/google_login', json={"access_token":
"valid_token"})

    assert response.status_code == 200
    assert response.json['Success'] is True
    assert response.json['access_token'] == 'mock_jwt_token'
    assert response.json['message'] == 'Login successful'
```

AUTHENTICATION APIs

Input Data	

• JSON
{
"access_token": "valid_token"
}	

Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"Success": true,
"access_token":
"mock_jwt_token",
"message": "Login
successful"
}
	• Status Code: 200
	• JSON
	{
	"Success": true,
	"access_token":
	"mock_jwt_token",
	"message": "Login
	successful"
	}

Result: **Success**	

- **Test Case: Login with Google account authentication with missing access token**

```python
# Test case for missing access token
def test_google_login_missing_token(client):
    response = client.post('/google_login', json={})

    assert response.status_code == 400
    assert response.json['Success'] is False
    assert response.json['message'] == 'Google access token is required'
```

Input Data	
• JSON {}	
Expected Output	**Actual Output**

• Status Code: 400
• JSON
{
"Success": false,
"message": "Google access
token is required"
}
	• Status Code: 400
	• JSON
	{
	"Success": false,
	"message": "Google access token
	is required"
	}

Result: **Success**	

WEEK APIs

/api/weeks

METHOD: GET

- **Test Case: Retrieve details of all weeks**

```python
def test_get_weeks_with_data(client)
    """Test GET /weeks when there are weeks in the database."""
    response = client.get('/weeks')
    assert response.status_code == 200
    data = response.get_json()

    assert data["success"] is True
    assert data["message"] == "Weeks retrieved successfully"
    assert isinstance(data["weeks"], list)
```

Expected Output	Actual Output

• Status Code: 201
• JSON
{
"success": true,
"message": "Weeks retrieved successfully",
"weeks": [
{
"id": 1,
"week_number": 1,
"title": "..."
},
// Additional Weeks...
]
}
	• Status Code: 201
	• JSON
	{
	"success": true,
	"message": "Weeks retrieved successfully",
	"weeks": [
	{
	"id": 1,
	"week_number": 1,
	"title": "..."
	},
	// Additional Weeks...
	]
	}

Result: **Success**	

WEEK APIs

/api/weeks/{week_id)

METHOD: GET

- **Test Case: Retrieve details of a specific week by its ID**

```python
def test_get_week_details_valid_id(client):
    #Test GET /weeks/<week_id> with a valid existing week ID.
    response = client.get('/weeks/1')

    assert response.status_code == 200
    data = response.get_json()

    assert data["success"] is True
    assert data["message"] == "Week details retrieved successfully"
    assert "week" in data
    assert data["week"]["id"] == 1
    assert "lectures" in data["week"]
    assert "assignments" in data["week"]
```

Endpoint	
GET /api/weeks/1	
Expected Output	**Actual Output**

● Status Code: 200
● JSON
{
"success": true,
"message": "Week details retrieved successfully",
"week": {
"id": 1,
"week_number": 1,
"title": "...",
"lectures": [
{
"id": 1,
"title": "...",
"video_id": "..."
},
// Additional Lectures
],
"assignments": [
{
"id": 1,
"title": "...",
"assignment_type":
	● Status Code: 200
	● JSON
	{
	"success": true,
	"message": "Week details retrieved successfully",
	"week": {
	"id": 1,
	"week_number": 1,
	"title": "...",
	"lectures": [
	{
	"id": 1,
	"title": "...",
	"video_id": "..."
	},
	// Additional Lectures
	],
	"assignments": [
	{
	"id": 1,
	"title": "...",
	"assignment_type":

WEEK APIs

"..."
},
// Additional Assignments
]
}
}
	"..."
	},
	// Additional Assignments
	]
	}
	}

Result: **Success**	

- **Test Case: Retrieve details of a non-existent week by its ID**

```python
def test_get_week_details_invalid_id(client):
    #Test GET /weeks/<week_id> with an ID that does not exist.
    response = client.get('/weeks/9999')

    assert response.status_code == 404
    data = response.get_json()

    assert data["success"] is False
    assert data["message"] == "Week not found"
```

Endpoint	
GET /api/weeks/9999	
Expected Output	**Actual Output**

• Status Code: 404
• JSON
{
"success": false,
"message": "Week not found"
}
	• Status Code: 404
	• JSON
	{
	"success": false,
	"message": "Week not found"
	}

Result: **Success**	

ASSIGNMENT APIs

/api/assignments

METHOD: GET

- **Test Case: Retrieve details of all the existing assignments**

```
# Test when assignments exist
def test_get_assignments_with_data(client):
    """
    Test case for GET /assignments when assignments are present in the
database.
    Expects a 200 status code with a list containing assignment data.
    """
    response = client.get('/assignments')  # Make the GET request
    assert response.status_code == 200  # Check for success status code

    json_data = response.get_json()  # Parse the response as JSON
    assert json_data['success'] is True # Ensure the success flag is True
    assert json_data['message'] == 'Assignments retrieved successfully'
# Validate the response message
    assert len(json_data['assignments']) > 0  # Confirm at least one
assignment is returned
```

Expected Output	Actual Output

• Status Code: 200
• JSON
{
"success": true,
"message": "Assignments retrieved successfully",
"assignments": [
{
"id": 1,
"title": "...",
"due_date": "...",
"description": "...",
}
// Additional assignments...
]
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Assignments retrieved successfully",
	"assignments": [
	{
	"id": 1,
	"title": "...",
	"due_date": "...",
	"description": "...",
	}
	// Additional assignments...
	]
	}

Result: **Success**	

ASSIGNMENT APIs

/api/assignments/{assignment_id}

METHOD: GET

- **Test Case: Retrieve details of a specific assignment by its ID**

```
# Test when assignments exist
def test_get_assignments_with_data(client):
    """
    Test case for GET /assignments when assignments are present in the
database.
    Expects a 200 status code with a list containing assignment data.
    """
    response = client.get('/assignments')  # Make the GET request
    assert response.status_code == 200  # Check for success status code

    json_data = response.get_json()  # Parse the response as JSON
    assert json_data['success'] is True # Ensure the success flag is True
    assert json_data['message'] == 'Assignments retrieved successfully'
# Validate the response message
    assert len(json_data['assignments']) > 0  # Confirm at least one
assignment is returned
```

Endpoint	
GET /api/assignments/1	
Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": true,
"message": "Assignment retrieved successfully",
"assignment": {
"id": 1,
"title": "...",
"due_date":"...",
"description":"..."
}
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"message": "Assignment retrieved successfully",
	"assignment": {
	"id": 1,
	"title": "...",
	"due_date":"...",
	"description":"..."
	}
	}

Result: Success	

ASSIGNMENT APIs

- **Test Case: Retrieve details of a non-existent assignment by its ID**

```
# Test when the assignment does not exist
def test_get_assignment_not_found(client):
    """
    Test case for GET /assignments/<assignment_id> when the assignment ID
does not exist.
    Expects a 404 status code with an 'Assignment not found' message.
    """
    response = client.get('/assignments/999')  # Using a non-existent
assignment ID
    assert response.status_code == 404  # Check for not found status code

    json_data = response.get_json()  # Parse the response as JSON
    assert json_data['success'] is False  # Ensure the success flag is
False
    assert json_data['message'] == 'Assignment not found'  # Validate the
not found message



    assert len(json_data['assignments']) > 0  # Confirm at least one
assignment is returned
```

Endpoint	
GET /api/assignments/999	
Expected Output	**Actual Output**

• Status Code: 404
• JSON
{
"success": false,
"message": "Assignment not found"
}
	• Status Code: 404
	• JSON
	{
	"success": false,
	"message": "Assignment not found"
	}

Result: Success	

PROGRAMMING ASSIGNMENT APIs

/api/programming_assignments/{question.id}

METHOD: GET

- **Test Case: Retrieve details of a specific programming assignment by its ID**

```python
# Test case for retrieving a programming assignment
def test_get_programming_assignment(client):

    question = ProgrammingAssignment.query.filter_by(id = 3).first()
    response = client.get(f'/programming_assignments/{question.id}')
    assert response.status_code == 200
    assert response.json['success'] == True
    assert response.json['data']['assignment_id'] == 101
```

Endpoint	
GET /api/programming_assignments/3	
Expected Output	**Actual Output**

• Status Code: 200
• JSON
{
"success": True,
"data": {
"assignment_id": 101,
"problem_statement": "...",
"input_format": "...",
"output_format": "...",
"constraints": "...",
"sample_input": "...",
"sample_output": "...",
"test_cases": [...]
}
}
	• Status Code: 200
	• JSON
	{
	"success": True,
	"data": {
	"assignment_id": 101,
	"problem_statement": "...",
	"input_format": "...",
	"output_format": "...",
	"constraints": "...",
	"sample_input": "...",
	"sample_output": "...",
	"test_cases": [...]
	}
	}

Result: Success	

PROGRAMMING ASSIGNMENT APIs

- **Test Case: Retrieve details of a non-existent programming assignment by its ID**

```python
# Test case for non-existent assignment retrieval
def test_get_non_existent_assignment(client):
    response = client.get('/programming_assignments/999')
    assert response.status_code == 404
    assert response.json['success'] == False
    assert "Programming assignment not found" in response.json['message']
```

Endpoint	
GET /api/programming_assignments/999	
Expected Output	**Actual Output**

• Status Code: 404
• JSON
{
"success": false,
"message": "Programming assignment not found"
}
	• Status Code: 404
	• JSON
	{
	"success": false,
	"message": "Programming assignment not found"
	}

Result: **Success**	

PROGRAMMING ASSIGNMENT APIs

/api/programming_assignments/{assignment_id}/execute

METHOD: POST

- **Test Case: Execute a programming assignment successfully**

```
# Test case for successful code execution with all test cases passing
def test_execute_solution_success(client):
    response = client.post('/programming_assignments/1/execute',
                           json={
                               "code": '''
                               def is_prime(N):
                                   if N <= 1:
                                   return 'NO'
                                   for i in range(2, int(N**0.5) + 1):
                                       if N % i == 0:
                                           return 'NO'
                                   return 'YES'
                                N = int(input())
                                print(is_prime(N))
                           '''})

    assert response.status_code == 200
    assert response.json['success'] is True
    assert response.json['score'] == 100
    assert response.json['passed_count'] == 4
    assert response.json['total_cases'] == 4
```

Input Data

- JSON

```
{
    "code": '''
            def is_prime(N):
              if N <= 1:
                 return 'NO'
              for i in range(2, int(N**0.5) + 1):
                 if N % i == 0:
                   return 'NO'
                 return 'YES'
            N = int(input())
            print(is_prime(N))'''
}
```

PROGRAMMING ASSIGNMENT APIs

Expected Output	Actual Output

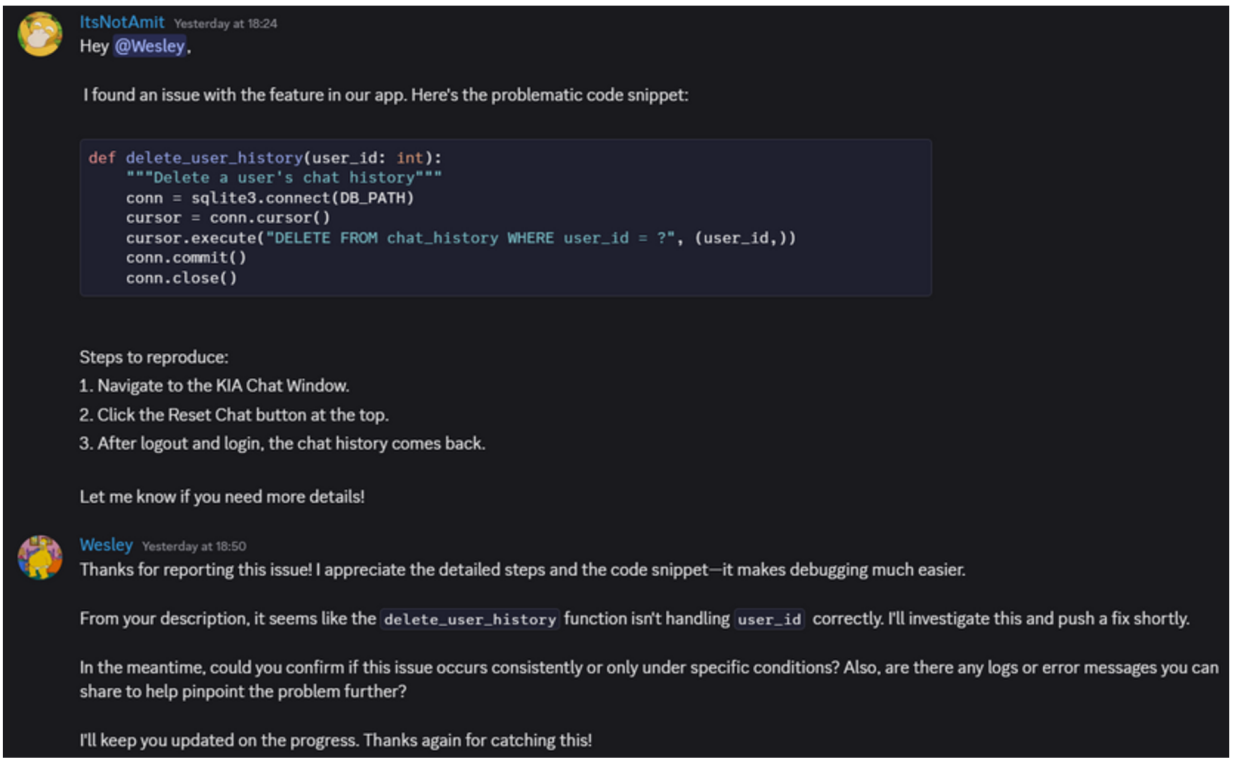
• Status Code: 200
• JSON
{
"success": true,
"score": 100,
"passed_count": 4,
"total_cases": 4
}
	• Status Code: 200
	• JSON
	{
	"success": true,
	"score": 100,
	"passed_count": 4,
	"total_cases": 4
	}

Result: **Success**	

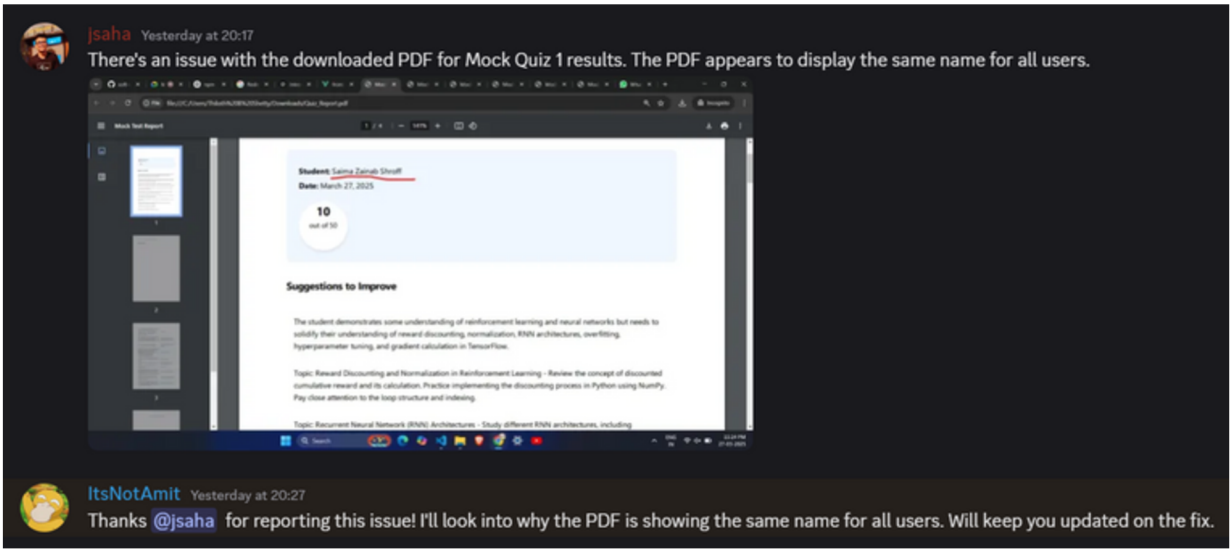
ISSUE REPORTING AND TRACKING

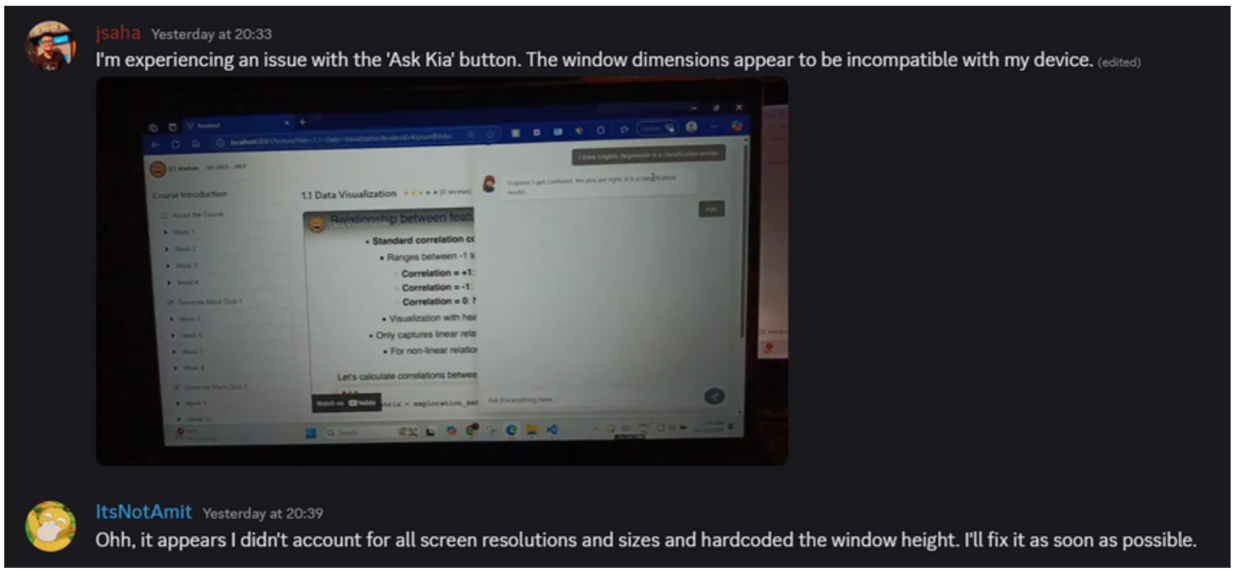
ISSUE REPORTING AND TRACKING

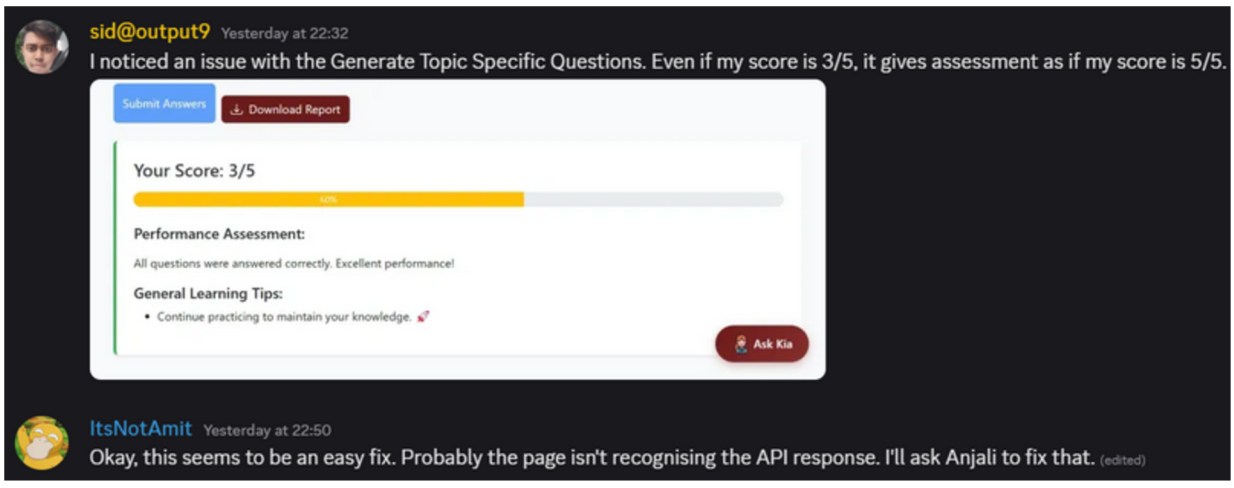
To streamline collaboration during development, the team used a dedicated Discord channel exclusively for reporting and tracking bugs. This channel served as a centralized space where team members could log issues in real time, discuss potential fixes, and follow up on resolutions. A few representative screenshots of this bug report channel have been attached below.



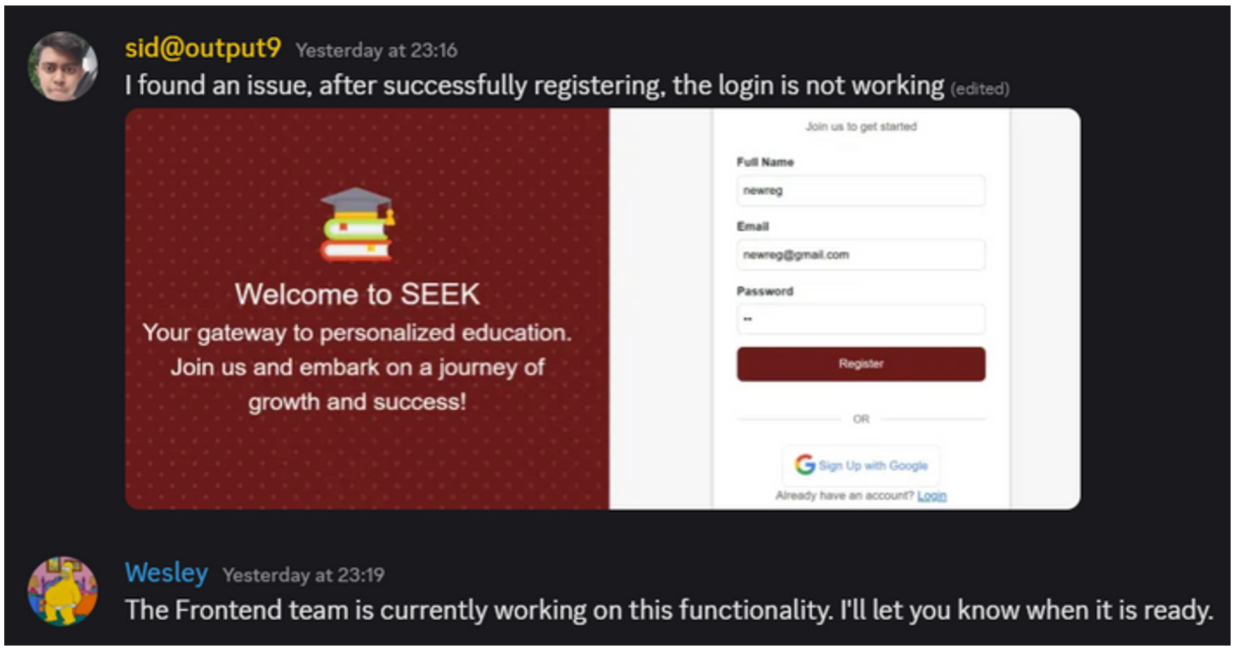
ISSUE REPORTING AND TRACKING

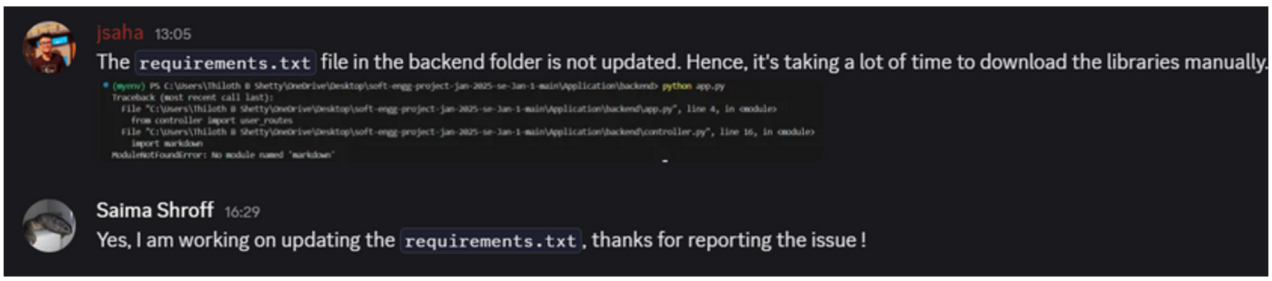


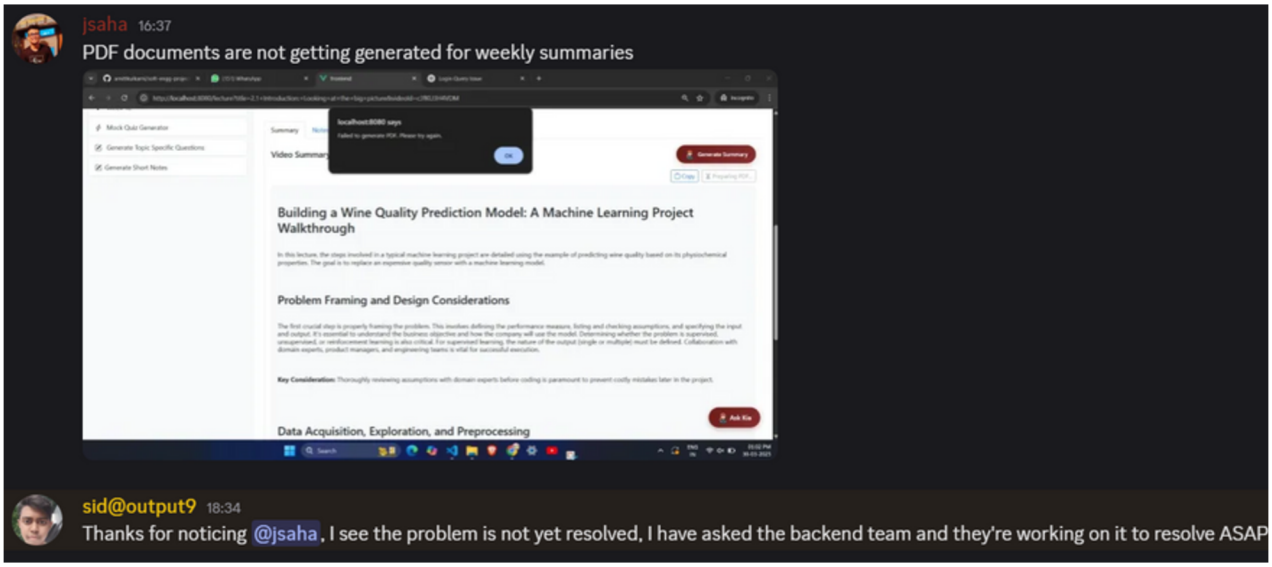




ISSUE REPORTING AND TRACKING







TECHNOLOGIES USED

TECHNOLOGIES USED

Backend

- Flask – Core backend framework.
- SQLAlchemy – Database ORM.
- JWT-Authentication – Secure authentication.
- Werkzeug Security – Password hashing.
- Google OAuth2 – User authentication.
- PDFKit – Generates PDFs.
- Pytest – Automated testing.
- Thunderclient – API testing tool.

Frontend

- Vue 3 CLI – Streamlined Vue.js setup.
- Vue Router – Enables navigation.
- VueMarkdown – Renders markdown.
- JavaScript – Powers interactivity.
- Bootstrap – Responsive UI.
- HTML & CSS – Structure & styling.
- ESLint – Enforces code quality.
- Axios – Handles API requests.

TECHNOLOGIES USED

GenAI

- LangChain – Facilitates AI-driven conversations with seamless integration of LLMs.
- Gemini 1.5 Flash – Powers advanced natural language processing for smart responses.
- HuggingFace – Provides access to various AI models for text generation and analysis.
- ChromaDB – A vector database that enhances retrieval-augmented generation (RAG).

General

- GitHub – Version control, project tracking & collaboration.
- Jira – Project tracking.
- Figma– Wireframes.
- PlantUML – UML diagrams.
- Swagger Editor – API documentation.

INSTRUCTIONS TO RUN THE APPLICATION

INSTRUCTIONS TO RUN THE APPLICATION

To execute the application on a local development environment, the following setup procedure must be followed. The system is built using a Flask backend and a Vue.js frontend, both of which need to be configured and run independently. All steps have been clearly outlined below for ease of use, including commands for Linux, macOS, and Windows platforms.

BACKEND SETUP

Step 1: Navigate to the backend directory

Open a terminal or command prompt and run the following command:

```
cd backend
```

Step 2: Create and activate a virtual environment

For Linux/macOS:

```
python3 -m venv venv
source venv/bin/activate
```

For Windows:

```
python -m venv venv
venv\Scripts\activate
```

INSTRUCTIONS TO RUN THE APPLICATION

Step 3: Install required Python dependencies
Once the virtual environment is activated, install all required packages by running:

```
pip install -r requirements.txt
```

Step 4: Set up environment variables
Linux/macOS:

```
cp .env.example .env
```

Windows:

```
copy .env.example .env
```

This command copies the environment variables template to a new `.env` file.

Step 5: Add required keys to the `.env` file

- Generate `SECRET_KEY` and `JWT_SECRET_KEY` from https://jwtsecret.com/generate, with length set to 128.
- Obtain a `GOOGLE_API_KEY` from the Google Cloud Console.
- Add all three keys to the `.env` file under their respective names.

INSTRUCTIONS TO RUN THE APPLICATION

Step 6: Initialize the database

Run the following commands to set up and migrate the database:

```
flask db init
flask db migrate -m "Initial migration"
flask db upgrade
```

Step 7: Seed the database with initial data

Execute the seeding script to populate the database with sample content:

```
python seed_data.py
```

FRONTEND SETUP

Step 1: Open a new terminal window

This allows running the frontend server in parallel with the backend.

Step 2: Navigate to the frontend directory

Run:

```
cd frontend
```

Step 3: Install frontend dependencies

Use npm to install all necessary frontend packages:

```
npm install
```

INSTRUCTIONS TO RUN THE APPLICATION

RUNNING THE APPLICATION

Step 1: Start the backend server

For Linux/macOS:

```
cd backend
source venv/bin/activate
python app.py
```

For Windows:

```
cd backend
venv\Scripts\activate
python app.py
```

The backend server will be running at:
http://localhost:5000/

Step 2: Start the frontend server

Open a new terminal and run:

```
cd frontend
npm run serve
```

The frontend interface will be available at:
http://localhost:8080/

This completes the local setup and launch process for the application.