Software Engineering

Project

TEAM 27

Prepared By :

Dev Khatri (21f3001150)
Satish Jaiswal (21f2000142)
Prem Kumar (21f1000531)
Srishti Singh (21f1006972)
Shrikanth V (21f1002328)
Shrirang Sapate (21f1002870)
Pranav R (21f1004199)
Shristi Tiwari (21f2000589)

Meet our Team

ACKNOWLEDGEMENT

We, Team 27 of SE Project, May 2024 Term, would like to express our sincere gratitude to all those who have supported us throughout the course of our Software Engineering project.

We are deeply thankful to our professors and mentors for their invaluable guidance, feedback, and encouragement, which have been crucial in the successful completion of this project. We also extend our appreciation to our peers for their collaboration and constructive discussions, which greatly enriched our learning experience.

Finally, we are grateful to our institution, IIT Madras, and the software engineering course team for providing us with the resources and platform to apply our knowledge in a practical and impactful way.

Thank you all for your support and belief in our abilities.

Dev Khatri
21f3001150

Satish Jaiswal
21f2000142

Prem Kumar
21f1000531

Shrirang Sapate
21f1002870

Srishti Singh
21f1006972

Shrikanth V
21f1002328

Pranav R
21f1004199

Shristi Tiwari
21f2000589

About the Project

With the rapid advancement of generative AI (GenAI) technologies, there are promising possibilities of effective integration of GenAI into these learning environments. The goal is to examine ways in which GenAI can be effectively integrated into learning environments such as in the SEEK portal.

- **Learning Environment**
  - Learners can watch video lectures.
  - They can get announcements regarding the course via the portal.
- **Coding Environment**
  - They can code in the given code editor and can get instant output on the code written by them.
  - They can submit practice and graded programming assignments and can verify how accurate their code is.
- **GenAI Support for Learners**
  - Learners can ask the chatbot about the queries they are having in the course.
  - Learners can ask the chatbot about the doubts they are having in the video lectures in the course.
  - Chatbot can give the corrected version of code to the learners in the practice assignments section for learning purposes.

Software Engineering Course

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

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

CONTENTS

MILESTONE 1

Identify User Requirements

Identify User Requirements

1. Student Side

1. **Home:**
   Access point for login and signup.

2. **Login/Signup:**
   Allows users to enter the platform.

3. **Dashboard:**
   Main area where users access various features.

   - A **Landing:**
     Announcements, upcoming deadlines, scheduled lectures.
   - B **Profile/Support (GenAI):**
     User profile, support options.
   - C **Subject:**
     a) Course materials and content(GenAI).
     b) Practice assignments, code editor and GenAI support.

2. Admin Side

1. **Home:**
   Access point for login and signup.

2. **Login/Signup:**
   Allows admins to enter the platform.

3. **Dashboard:**
   Main area where admins upload content.

Envision Different Experiences/Scenarios on SEEK Portal

1. Without GenAI

Scenario 1: Asking video lecture doubts

A A learner watches a video lecture for the course and a doubt in forum.

B Instructor/TA check the doubts and answers the query.

Scenario 2: Accessing Course Support

A A learner needs help with understanding the course rules and regulations. Learners mail the support team.

B Support team navigate to the mail and answers the query.

Scenario 3: Solving a Programming Assignment Problem

A A learner logs into the SEEK portal.

B Navigates to the course page and selects the programming question.

C Reads the problem statement and writes code in the provided editor.

D Submits the code and receives immediate feedback on test cases passed/failed.

E Learner tries to understand why certain test cases failed and attempts to debug and resubmit.

Envision Different Experiences/Scenarios on SEEK Portal

2. With GenAI

Scenario 1: Asking video lecture doubts

A A learner watches a video lecture for the course and ask the doubt to the chatbot.

B The GenAI assistant immediately answers the query of the learner in detail.

Scenario 2: Accessing Support

A A learner needs help with understanding the course rules and regulations. Learners ask the chatbot.

B The GenAI assistant immediately answers the query of the learner in detail.

Scenario 3: Solving a Programming Assignment Problem

A A learner logs into the SEEK portal.

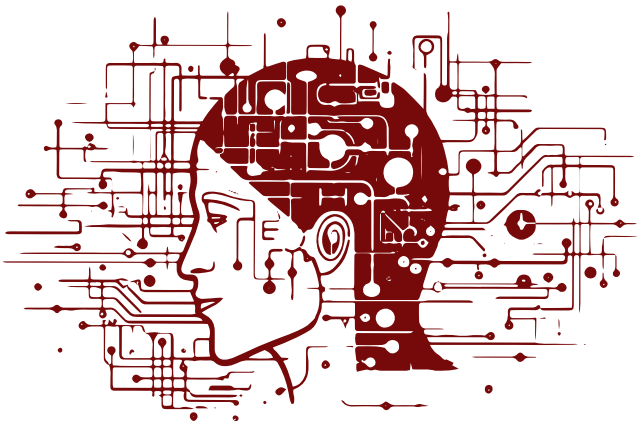
B Navigates to the course page and selects the programming question.

C Reads the problem statement and writes code in the provided editor.

D Submits the code and receives immediate feedback on test cases passed/failed.

E GenAI suggests possible reasons for failed test cases and offers debugging tips.

F Learner attempts to debug and resubmit with the help of GenAI's suggestions.



Learner Journey Maps

1. Without GenAI

Scenario 1: Asking video lecture doubts

A	Login
B	Navigate to Dashboard
C	Watch Course Videos and have doubts.
D	a doubt in forum. Instructor/TA check the check the doubts and answers the query.

Scenario 2: Accessing Course Support

A	Login
B	Navigate to Profile/Support
C	Submit Query
D	Wait for Response

Scenario 3: Solving a Programming Assignment Problem

A	Login
B	Navigate to Course Page
C	Select Programming Question
D	Read Problem Statement
E	Write Code in Editor
F	Submit Code
G	Receive Feedback on Test Cases
H	Debug and Resubmit

2. With GenAI

Scenario 1: Asking video lecture doubts

A Login

B Watch Course Videos and have doubts. Ask the GenAI assistant.

C GenAI answers the query.

Scenario 2: Accessing Course Support

A Login

B Interact with GenAI Chatbot

C Receive Instant Assistance

Scenario 3: Solving a Programming Assignment Problem

A Login

B Navigate to Course Page

C Select Programming Question

D Read Problem Statement

E Write Code in Editor

F Submit Code

G Receive Feedback on Test Cases

H GenAI Suggests Debugging Tips

I Debug and Resubmit with GenAI Assistance

User Stories

Without GenAI

Scenario 1: Asking Doubts Regarding Video Lecture Topics

As a learner, I want a to ask the doubts I will be having in the video lectures, So that I can understand the concepts very well and improve my coding skills.

Scenario 2: Accessing Course Support

As a learner, I want to support, So that I can immediate help with my course related queries and resolve issues quickly.

Scenario 3: Solving a Programming Assignment Problem

As a learner, I want a to solve programming assignment problems and immediate feedback, So that I can understand my mistakes and improve my coding skills.

User Stories

With GenAI

Scenario 1: Asking Doubts Regarding Video Lecture Topics

As a learner, I want a to ask the doubts to the GenAI Assistant I will be having in the video lectures, So that I can understand the concepts very well and improve my coding skills.

Scenario 2: Accessing Course Support

As a learner, I want to interact with a GenAI chatbot for instant support, So that I can immediate help with my course related queries and resolve issues quickly.

Scenario 3: Solving a Programming Assignment Problem

As a learner, I want a GenAI assistant to suggest debugging tips when my code fails test cases, So that I can understand my mistakes better and improve my programming skills more efficiently.

MILESTONE 2

User Interfaces

Userflow for the App

The user flow diagram outlines the primary navigational structure of the application, detailing the pathways for both client and admin users. On the client side, users start at the Home screen, with options to log in or sign up before accessing the Dashboard. The Dashboard serves as a central hub, providing access to key features such as Announcements, Scheduled Lectures, Documents, and various support services powered by GenAI. It also includes a dedicated section for Subjects, where users can monitor their completion percentage, review graded and practice assignments, and access course materials. Additionally, the Profile/Support section allows users to manage their personal details, view current courses, and integrate their social profiles. On the admin side, the flow is streamlined, with the primary functions being Home, Login/Signup, and a Dashboard focused on uploading content. This user flow ensures a cohesive and intuitive experience for both clients and administrators, facilitating efficient navigation and resource management.

Wireframes
(Low-fidelity)

The user flow diagram outlines the primary navigational structure of the application, detailing the pathways for both client and admin users. On the client side, users start at the Home screen, with options to log in or sign up before accessing the Dashboard. The Dashboard serves as a central hub, providing access to key features such as Announcements, Scheduled Lectures, Documents, and various support services powered by GenAI. It also includes a dedicated section for Subjects, where users can monitor their completion percentage, review graded and practice assignments, and access course materials. Additionally, the Profile/Support section allows users to manage their personal details, view current courses, and integrate their social profiles. On the admin side, the flow is streamlined, with the primary functions being Home, Login/Signup, and a Dashboard focused on uploading content. This user flow ensures a cohesive and intuitive experience for both clients and administrators, facilitating efficient navigation and resource management.

User Stories

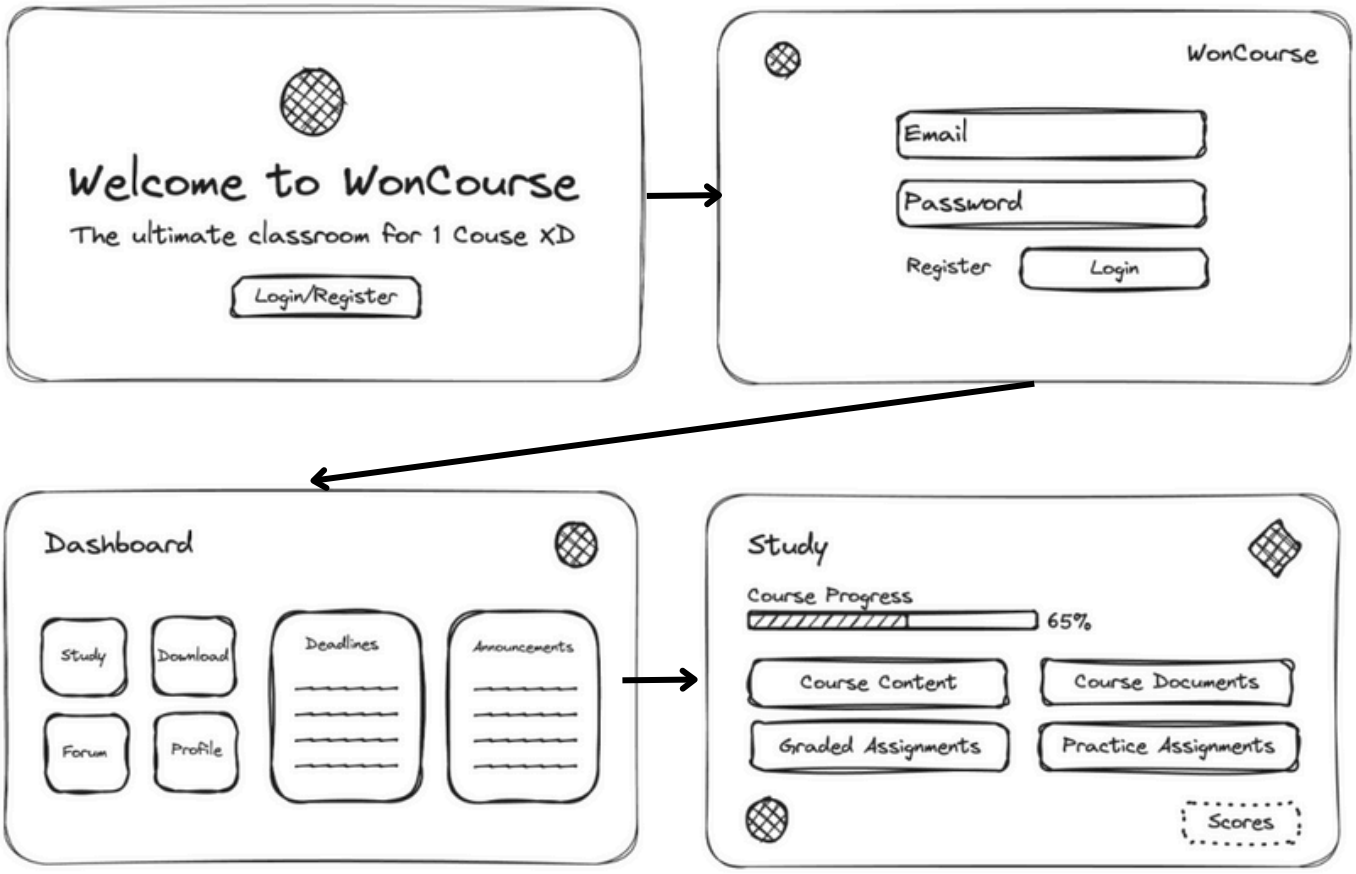
User stories are concise descriptions of a feature from the end user's perspective, capturing their goals and the value We seek. We guide the development process by clearly outlining user needs, ensuring that the final product meets user expectations. The flow of user stories involves defining the user's role, their desired action, and the purpose of that action. When combined with GenAI, user stories become even more impactful. GenAI enhances the understanding of user behavior, refines stories for precision, and automates interactions, providing real-time support and personalized experiences, thereby significantly improving the overall user experience.

User Stories | Scenario 1

Asking video lecture doubts

USERFLOW

HOME –› LOGIN –› DASHBOARD –› STUDY –› COURSE VIDEO


Welcome to WonCourse
The ultimate classroom for 1 Couse XD
Login/Register
WonCourse
Email
Password
Register
Login
Dashboard
Study
Download
Forum
Profile
Deadlines
Announcements
Study
Course Progress
65%
Course Content
Course Documents
Graded Assignments
Practice Assignments
Scores

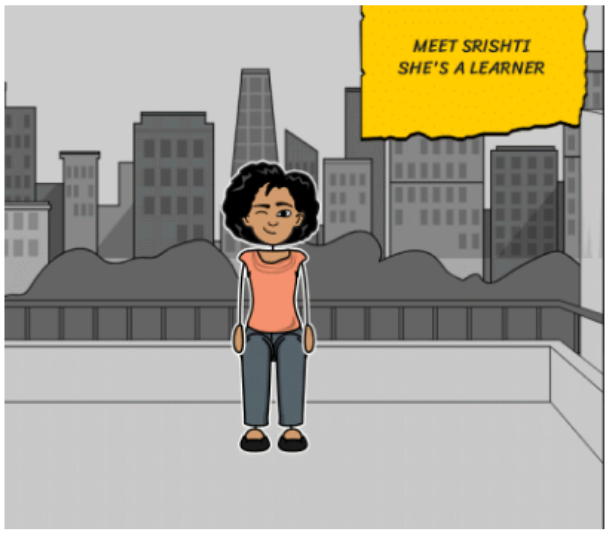

Okay!! I have got the idea of where to watch the lectures but I am still not sure where would I go if I have to ask any doubt regarding the video?!?!?!?!

ENTERS

GEN AI

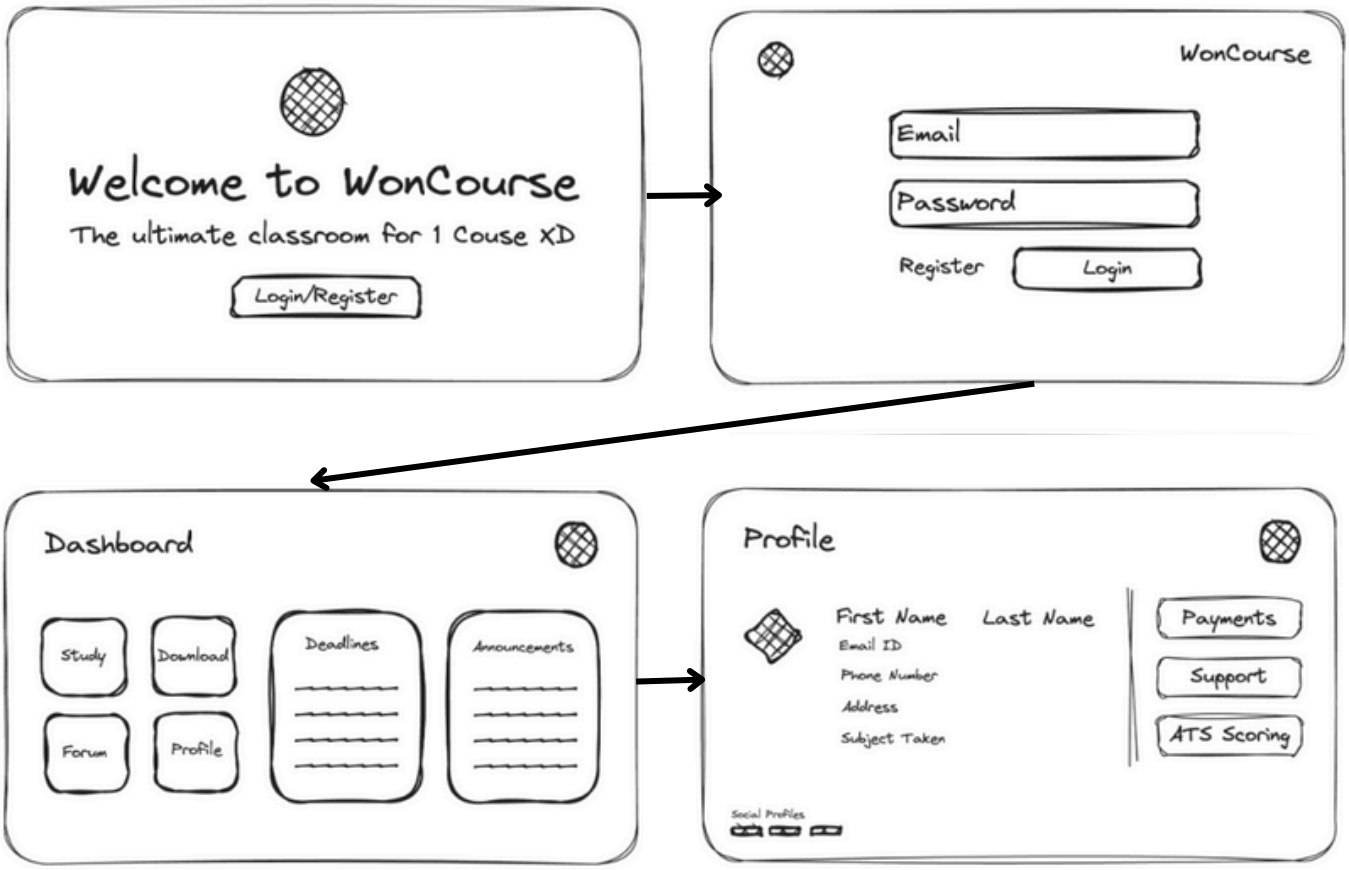
User Stories | Scenario 2

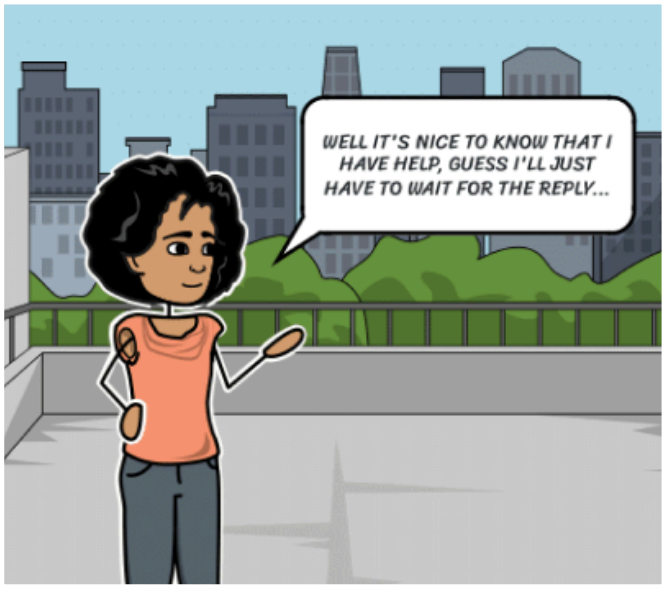
Accessing Course Support



USERFLOW

HOME –› LOGIN –› DASHBOARD –› PROFILE –› SUPPORT


Welcome to WonCourse
The ultimate classroom for 1 Couse XD
Login/Register
WonCourse
Email
Password
Register
Login
Dashboard
Study
Download
Forum
Profile
Deadlines
Announcements
Profile
First Name
Last Name
Email ID
Phone Number
Address
Subject Taken
Payments
Support
ATS Scoring
Social Profiles


WELL IT'S NICE TO KNOW THAT I HAVE HELP, GUESS I'LL JUST HAVE TO WAIT FOR THE REPLY...

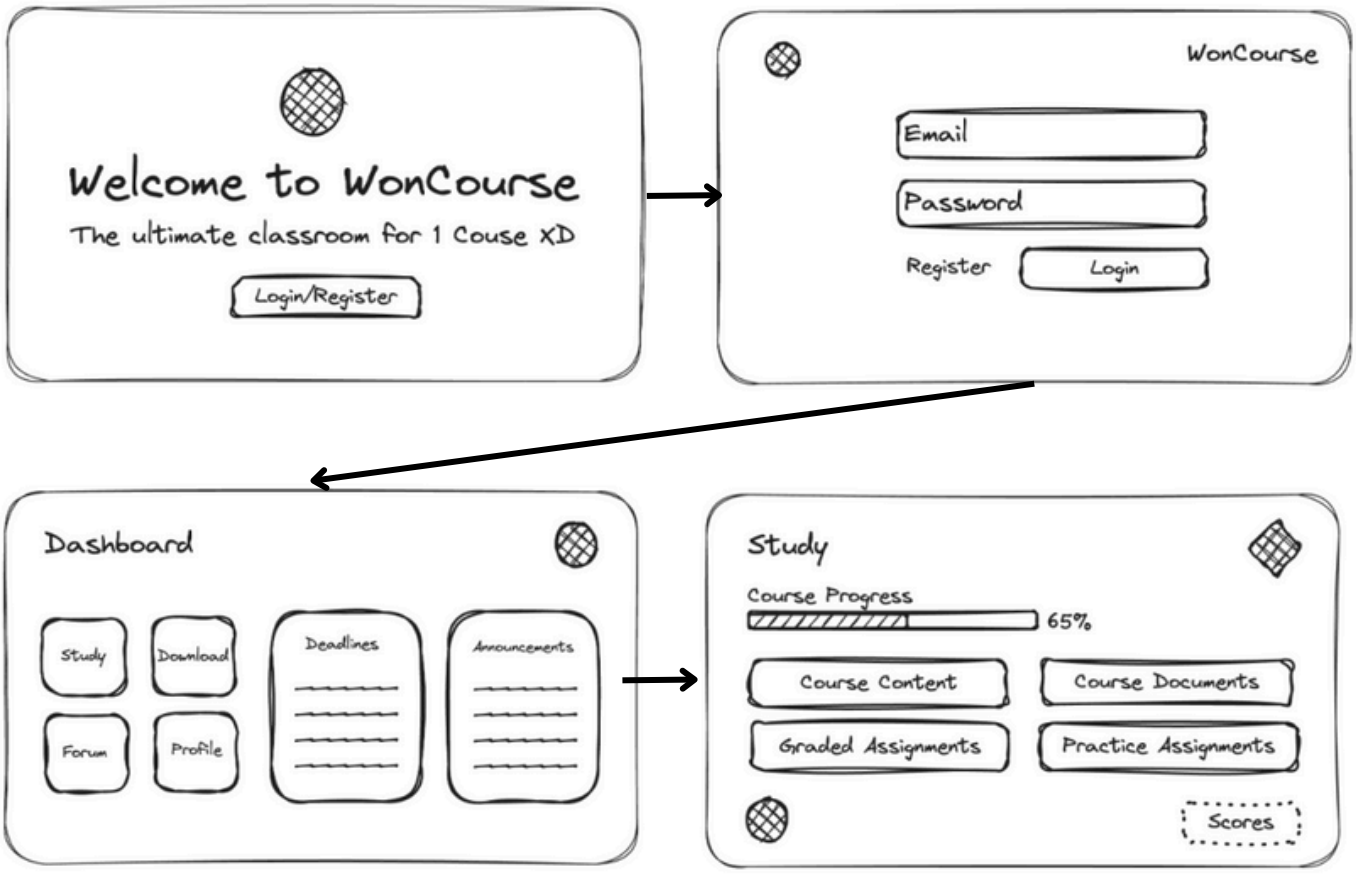
ENTERS

User Stories | Scenario 3

Solving a Practice Programming Question

USERFLOW

HOME –› LOGIN –› DASHBOARD –› STUDY –› PRACTICE ASSIGNMENTS


Welcome to WonCourse
The ultimate classroom for 1 Couse XD
Login/Register
WonCourse
Email
Password
Register
Login
Dashboard
Study
Download
Forum
Profile
Deadlines
Announcements
Study
Course Progress
65%
Course Content
Course Documents
Graded Assignments
Practice Assignments
Scores


I HAVE WRITTEN THE CODE BUT I DON'T KNOW WHAT THIS ERROR MEANS, I GUESS I'LL JUST GOOGLE IT

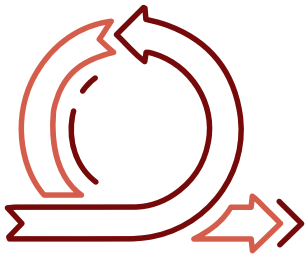
ENTERS

GEN AI

MILESTONE 3

Scheduling & Design

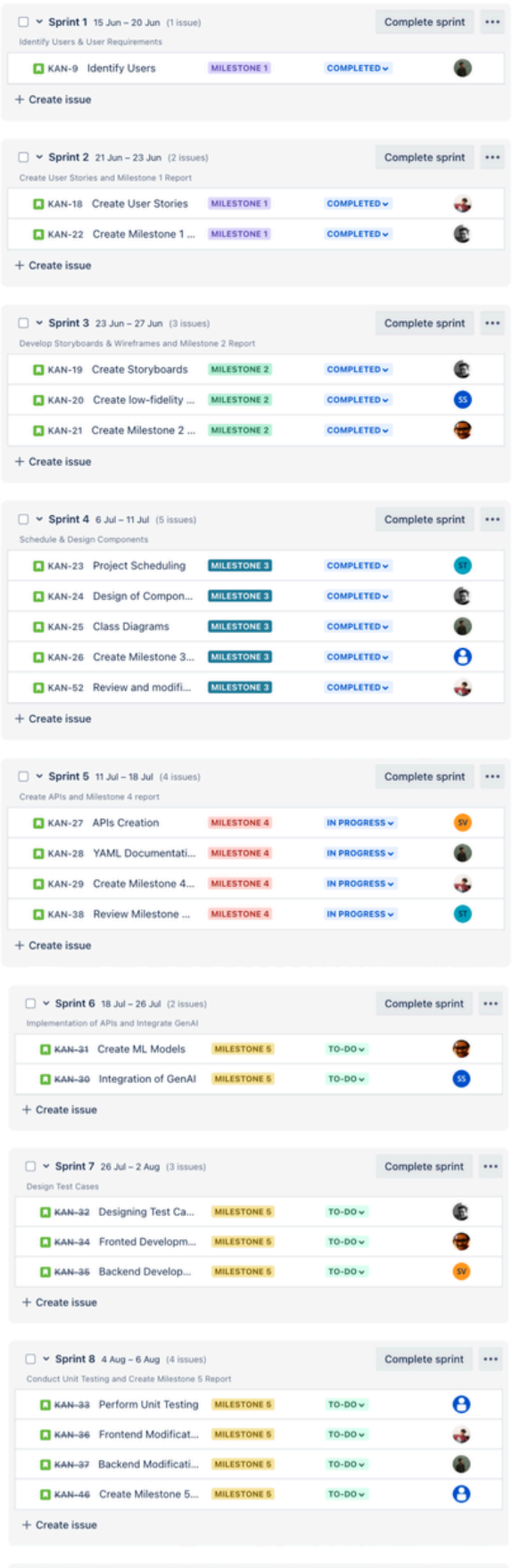
Project Schedule



1. Sprint Schedule

Sprint No.	Topic/Task	Timeline
Sprint 1	Identify Users & Requirements	15 June, 2024 20 June, 2024
Sprint 2	Create User Stories	21 June, 2024 23 June, 2024
Sprint 3	Develop Storyboards & Wireframes	24 June, 2024 27 June, 2024
Sprint 4	Schedule & Design Components	07 July, 2024 11 July, 2024
Sprint 5	Design API & Integrate GenAI	12 July, 2024 18 July, 2024
Sprint 6	Implement API	19 July, 2024 26 July, 2024
Sprint 7	Design Test Cases	27 July, 2024 02 August, 2024
Sprint 8	Conduct Unit Testing	05 August, 2024 07 August, 2024
Sprint 9	Integrate Frontend & Backend	08 August, 2024 18 August, 2024

Sprints as on Jira



2. About the Sprints

- **Sprint 1: Identify Users & Requirements**

Gather and analyse user needs and requirements. Ensure all key stakeholders are identified and their inputs collected.
Dates: 15 June, 2024 20 June, 2024

- **Sprint 2: Create User Stories**

Develop detailed user stories based on gathered requirements. Complete Milestone 1 Vetting and Submission.
Dates: 21 June, 2024 23 June, 2024

- **Sprint 3: Develop Storyboards & Wireframes**

Create visual representations and layouts following usability guidelines. Complete Milestone 2 Vetting and Submission.
Dates: 24 June, 2024 27 June, 2024

- **Sprint 4: Schedule & Design Components**

Develop project timeline and design key components. Complete Milestone 3 Report Vetting and Submission.
Dates: 07 July, 2024 11 July, 2024

- **Sprint 5: Design API & Integrate GenAI**

Plan API architecture and integrate GenAI functionalities. Set the foundation for API implementation.
Dates: 12 July, 2024 18 July, 2024

- **Sprint 6: Implement API**

Develop and deploy the designed API. Complete Milestone 4 Report Vetting and Submission.
Dates: 19 July, 2024 26 July, 2024

- **Sprint 7: Design Test Cases**

Create test cases for validating functionality and performance. Ensure comprehensive coverage for all features.
Dates: 27 July, 2024 02 August, 2024

- **Sprint 8: Conduct Unit Testing**

Perform unit testing to ensure code quality and reliability. Complete Milestone 5 Report Vetting and Submission.
Dates: 05 August, 2024 07 August, 2024

- **Sprint 9: Integrate Frontend & Backend**

Combine frontend and backend components into a cohesive system. Complete Milestone 6 Report Vetting and Submission.
Dates: 08 August, 2024 18 August, 2024

3. Scrum Board

Projects / SE Proj May 2024

All sprints

Search | +3 | Epic | Sprint | GROUP BY None

COMPLETED 11	IN PROGRESS 4	TO-DO 14
Identify Users MILESTONE 1 KAN-9	APIs Creation MILESTONE 4 KAN-27	Create ML Models MILESTONE 5 ~~KAN-31~~
Create User Stories MILESTONE 1 KAN-18	YAML Documentation MILESTONE 4 KAN-28	Integration of GenAI MILESTONE 5 ~~KAN-30~~
Create Milestone 1 Report MILESTONE 1 KAN-22	Create Milestone 4 Report MILESTONE 4 KAN-29	Designing Test Cases MILESTONE 5 ~~KAN-32~~
Create Storyboards MILESTONE 2 KAN-19	Review Milestone 4 Report MILESTONE 4 KAN-38	Fronted Development MILESTONE 5 ~~KAN-34~~
Create low-fidelity wireframes MILESTONE 2 KAN-20		Backend Development MILESTONE 5 ~~KAN-35~~
Create Milestone 2 Report MILESTONE 2 KAN-21		Perform Unit Testing MILESTONE 5 ~~KAN-33~~
Project Scheduling MILESTONE 3 KAN-23		Frontend Modifications MILESTONE 5 ~~KAN-36~~
Design of Components MILESTONE 3 KAN-24		Backend Modifications MILESTONE 5 ~~KAN-37~~
Class Diagrams MILESTONE 3 KAN-25		Create Milestone 5 Report MILESTONE 5 ~~KAN-46~~
Create Milestone 3 Report MILESTONE 3 KAN-26		Frontend and Backend Integration MILESTONE 6 ~~KAN-47~~
Review and modification of Milestone 3 Report MILESTONE 3 KAN-52		Project Code Review MILESTONE 6 ~~KAN-48~~
		Project Report Review MILESTONE 6 ~~KAN-51~~
		Project Presentation Preparation MILESTONE 6 ~~KAN-49~~
		Create Milestone 6 - Final Report MILESTONE 6

Scrum Meeting Minutes/Details

Scrum Meetings Schedule: Every Tuesday, Thursday, and Friday (20:30 22:30 PM)
Mode: Google Meet
Attendees:

- Dev Khatri
- Satish Jaiswal
- Prem Kumar
- Srishti Singh
- Shrikanth V
- Shrirang Sapate
- Pranav R
- Shristi Tiwari

Sprint 1 Scrum meetings minutes/details:
Dates: 15 June, 2024 20 June, 2024
The team will be collaborating to identify and define different Users and User requirements. Each team member will be presenting their own ideas and insights based on their understanding of the project requirements. After identifying the users, each team member will be assigned a task to come up with user stories for the different users.

Sprint 2 Scrum meetings minutes/details:
Dates: 21 June, 2024 23 June, 2024
User stories will be finalised. All team members will be sitting together in a meeting to review the milestone-1 report. The final submission will be made.

Sprint 3 Scrum meetings minutes/details:
Dates: 24 June, 2024 27 June, 2024
The team will be collaborating to create storyboards and wireframes for the project. Each member will be giving their ideas and suggestions regarding the storyboards. A discussion will take place to review the storyboards and wireframes created by the team, providing feedback and suggestions for improvements. After the final review, the Milestone-2 report will be submitted.

Sprint 4 Scrum meetings minutes/details:
Dates: 07 July, 2024 11 July, 2024
The team will be deciding on Project management software (JIRA) to keep track of the project's progress. After a thorough discussion among the team members, the project schedule will be decided, and a Gantt chart will be created. Components will be designed and a class diagram for the application will be made. Following the final review, the Milestone-3 report will be submitted.

Sprint 5 Scrum meetings minutes/details:
Dates: 12 July, 2024 18 July, 2024

The team will be planning the API architecture and integrating GenAI functionalities. We will be setting the foundation for API implementation.

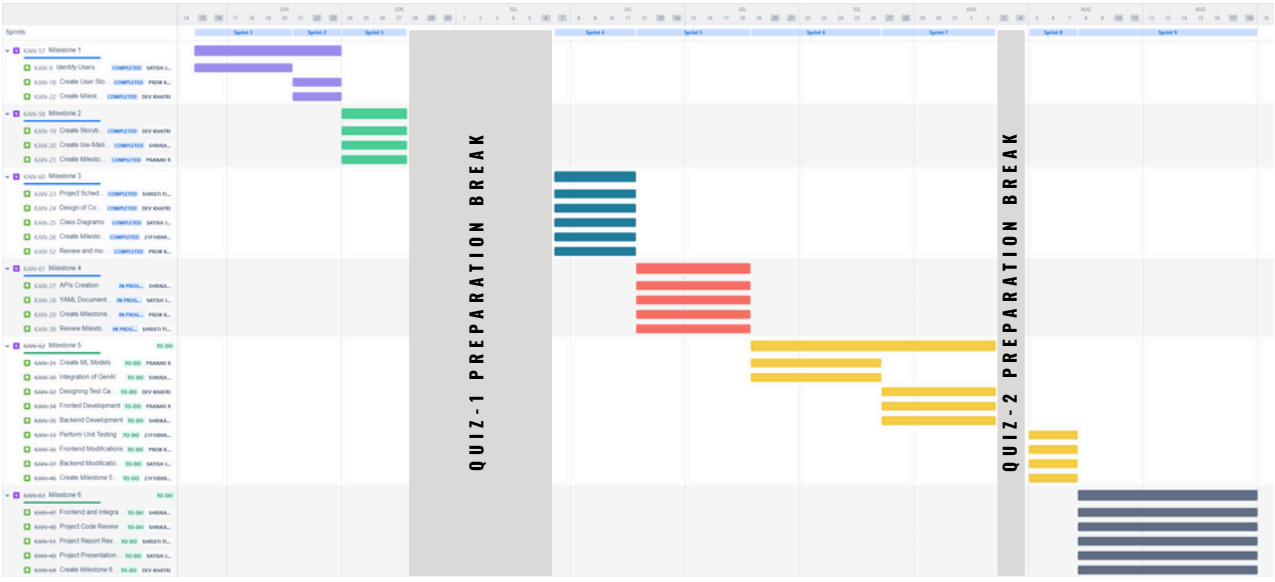
Sprint 6 Scrum meetings minutes/details:
Dates: 19 July, 2024 26 July, 2024
The team will be developing and deploying the designed API. We will be completing Milestone 4 Report Vetting and Submission.

Sprint 7 Scrum meetings minutes/details:
Dates: 27 July, 2024 02 August, 2024
The team will be creating test cases for validating functionality and performance. We will be ensuring comprehensive coverage for all features.

Sprint 8 Scrum meetings minutes/details:
Dates: 05 August, 2024 07 August, 2024
The team will be performing unit testing to ensure code quality and reliability. We will be completing Milestone 5 Report Vetting and Submission.

Sprint 9 Scrum meetings minutes/details:
Dates: 08 August, 2024 18 August, 2024
The team will be combining frontend and backend components into a cohesive system. We will be completing Milestone 6 Report Vetting and Submission.

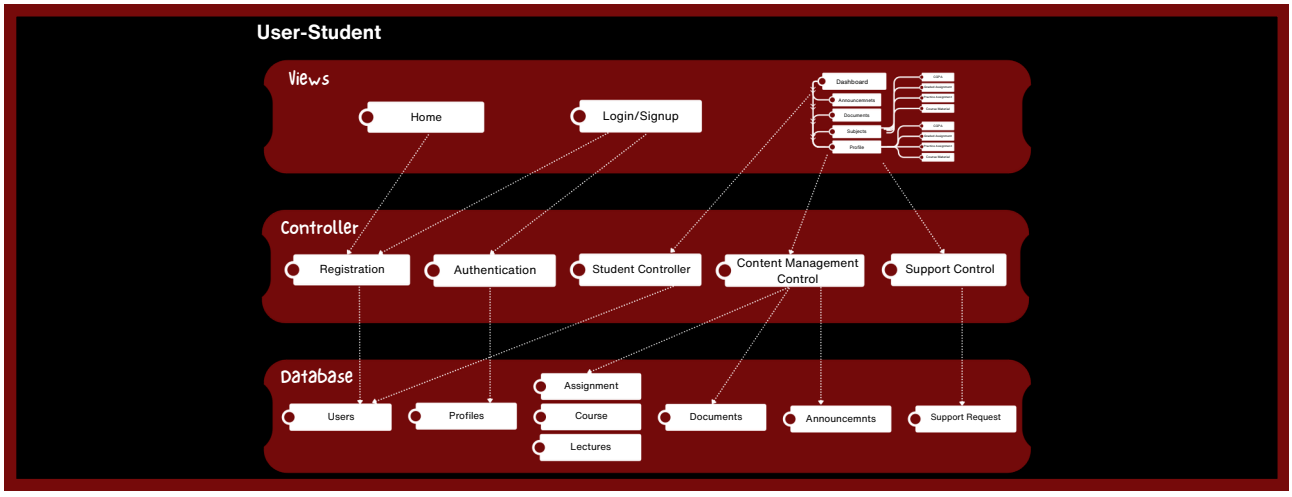
4. Gantt Chart



Component Design

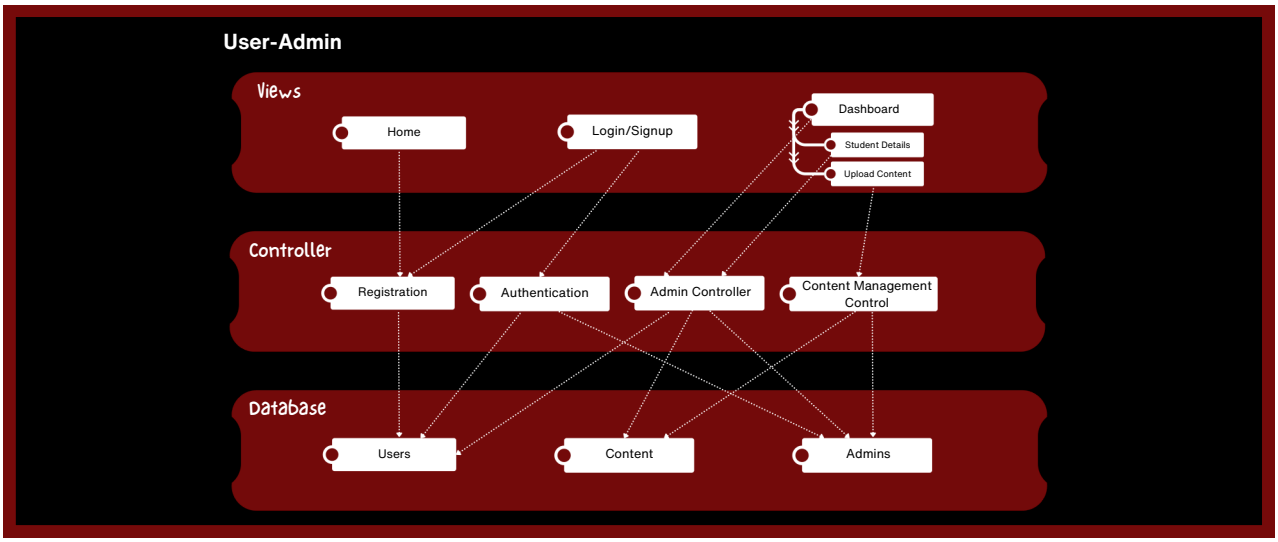
1. Student Component

- **Login/Signup:** This view component uses the Registration and Authentication controller, which interacts with the Users database.
- **Dashboard:** The Dashboard view is managed by the StudentController, which accesses data from the Users, Background, User_courses, and Feedback databases.
- **Courses:** The Courses view is supported by the CourseController, connecting to the Courses database.
- **View Profile:** This component relies on the StudentController, which utilises data from the Users, Background, User_courses, and Feedback databases.
- **Course Details:** The Course Details view is powered by the CourseController, fetching information from the Courses database.
- **Recommendation:** The Recommendation view is influenced by the RecommendationCourse controller, which interacts with data from the Users, Courses, Feedback, and Recommendation_path databases.

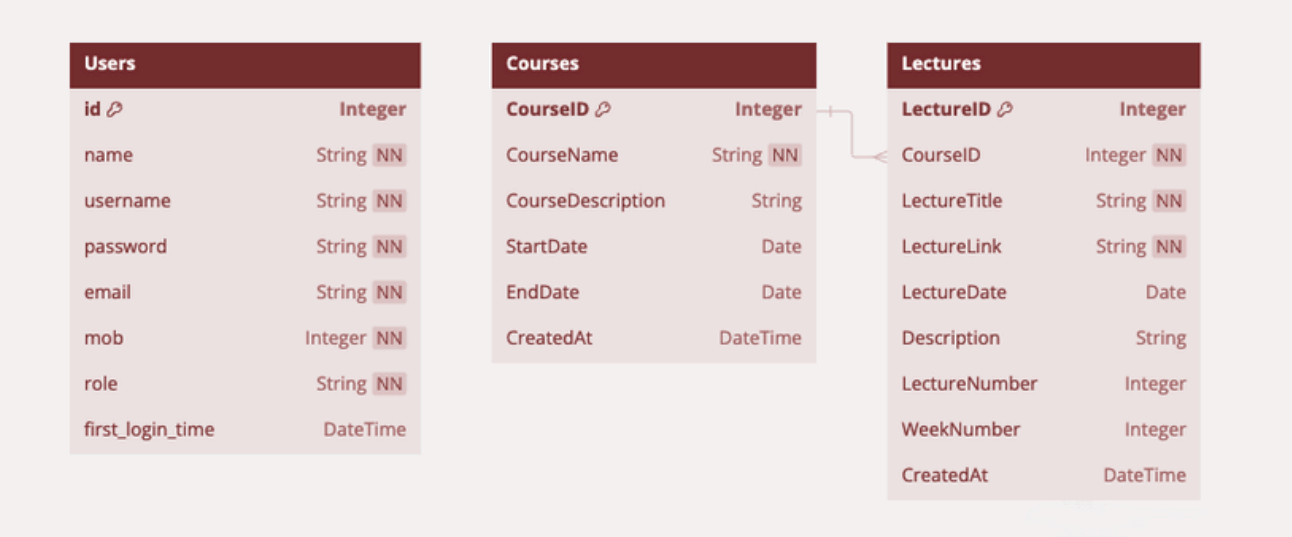


2. Admin Component

- **Login/Signup:** This view component uses the Registration and Authentication controller, which interacts with the Users database.
- **Dashboard:** The Dashboard view for admins is streamlined and focused on uploading content. It is managed by the AdminController, which accesses data from the Users, Courses, and Feedback databases.
- **Upload Content:** This component allows admins to upload and manage course content, assignments, quizzes, etc., and is supported by the AdminController connecting to the Courses and User_Courses databases.
- **Manage Users:** This component enables admins to manage user information and access, interacting with the Users and Background databases.
- **Feedback and Reports:** This component allows admins to view and manage feedback and generate reports, interacting with the Feedback and Recommended_Path databases.

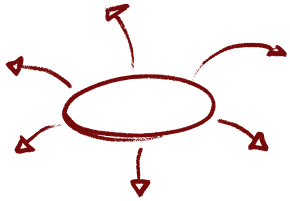


DB Schema Design

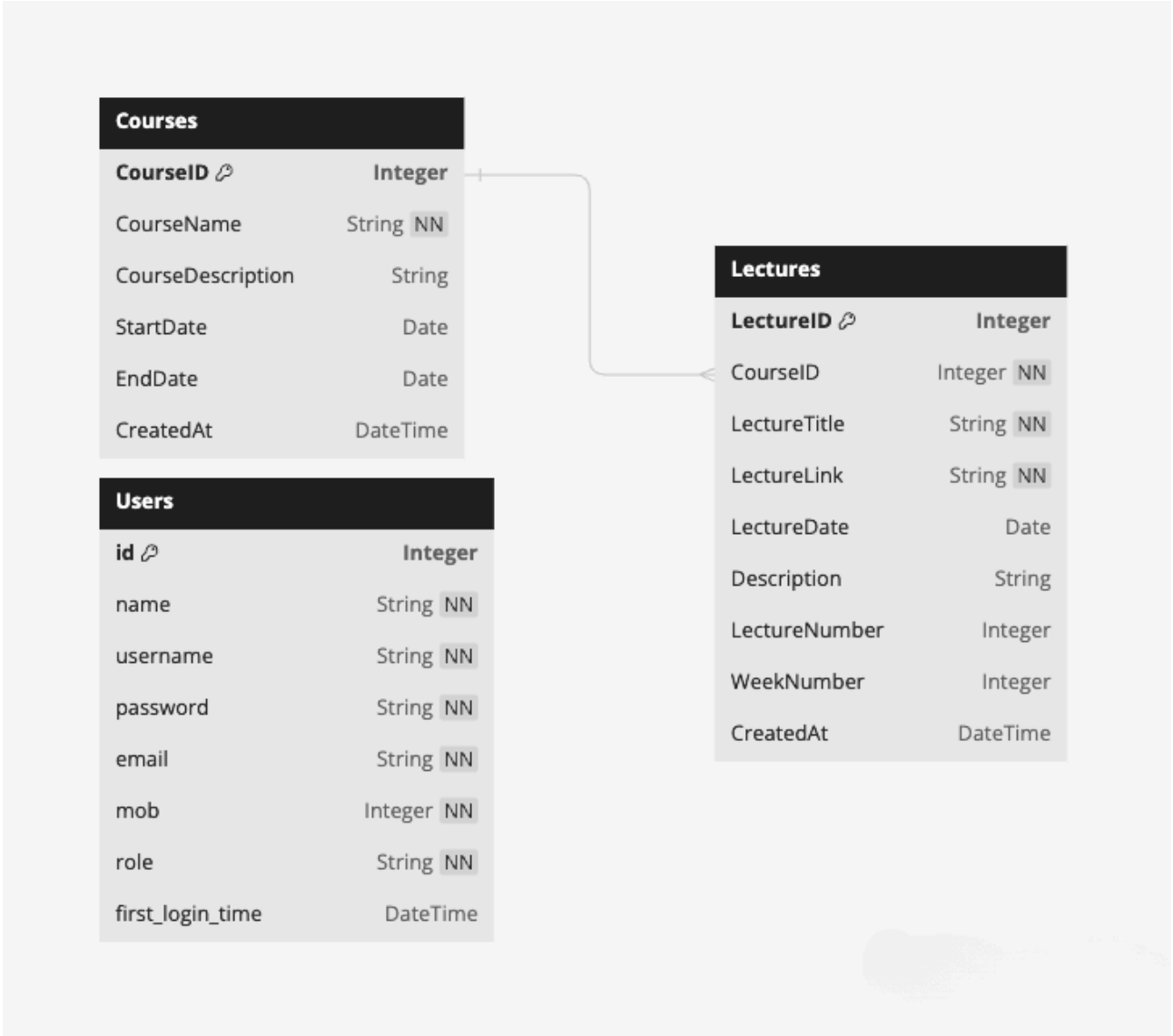


The database schema for the SEEK portal consists of several interconnected tables designed to manage various aspects of the learning environment. The **Users** table stores user information, including credentials and roles, which links to the **Profiles** table containing additional user details such as full name, contact information, and social profile links. The **Admins** table manages administrator-specific data. The **Courses** table outlines course information, including names, descriptions, and schedules, and connects to the **Assignments**, **Lectures**, **Documents**, and **Announcements** tables to handle respective course-related content. Assignments have details such as titles, due dates, and maximum scores. Lectures record information about lecture titles, dates, and times. Documents store paths to course-related files, while Announcements contain course-specific announcements. The **SupportRequests** table logs user requests for support, and the **Content** table holds administrative content, including titles and paths for various content types. This schema provides a structured way to integrate Generative AI for enhancing feedback, support, and interactivity in the SEEK portal.

Class Diagram



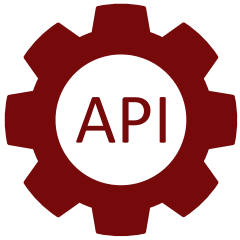
This class diagram depicts the LMS database schema, featuring entities such as Users, Courses, Assignments, Lectures, Announcements, Documents, SupportRequests, Profiles, Content, and Admins. Each entity has specific attributes and primary keys, with foreign keys establishing relationships (e.g., CourseID links Courses to Announcements, Assignments, Lectures, and Documents). Functions for creating, retrieving, and updating records are included, supporting comprehensive management of course materials, user profiles, administrative tasks, and support requests.



MILESTONE 4

API Endpoints

API Endpoints



1. Detailed Descriptions

1. Home Endpoint

- ***/:***
  - This endpoint provides a welcome message to the users. It does not require any request body and responds with a message in JSON format. It is mainly used to check if the application is up and running.

2. User Registration and Authentication

- ***/register:***
  - Registers a new user by taking in user details such as `username`, `password`, `email`, `mobile number`, and `name`. Successful registration returns a 201 status with a success message. If any required fields are missing or if the username already exists, it returns a 400 status. Server errors return a 500 status.
- ***/login:***
  - Allows users to log in with their `username` and `password`. A successful login returns a 200 status, along with a JWT token, user ID, and role. Invalid credentials return a 400 status, and server errors return a 500 status.

- **3. User Management**
- ***/user:***
  - Retrieves a list of users. This endpoint is restricted to Admin users only. It returns a 200 status with a list of user details if successful. If the user is not authorized, it returns a 403 status. Server errors return a 500 status.
- ***/user/{user_id}:***
  - Deletes a user by their user ID. This endpoint is also restricted to Admin users. It returns a 200 status upon successful deletion. If the user is not found, it returns a 404 status, and server errors return a 500 status.

4. Study Content and Courses

- ***/study:***
  - Retrieves information about the course that is available for the user. Admin users can access all course information. It returns a 200 status with course details if successful, a 404 status if the user is not found, and a 500 status for server errors.
- **/study:**
  - Allows the Admin to edit an existing course. The request body should include the course name, description, start date, and end date. A successful update returns a 201 status. Server errors return a 500 status.

5. Lectures Management

- ***/study/lectures:***
  - Retrieves a list of lectures associated with a course. It returns a 200 status with lecture details if successful, and a 500 status for server errors.
- ***/study/lectures:***
  - Allows the creation of a new lecture under a course. The request body should include details like lecture title, link, date, description, week number, and lecture number. A successful creation returns a 201 status. Server errors return a 500 status.
- ***/study/lectures/{lecture_id}:***
  - Deletes a lecture by its lecture ID. It returns a 200 status upon successful deletion. If the lecture is not found, it returns a 404 status, and server errors return a 500 status.

6. Admin Dashboard

- ***/dashboard/admin:***
  - Displays information on the Admin dashboard, including the total number of users, courses, and lectures. This endpoint is restricted to Admin users. It returns a 200 status with the dashboard data if successful, a 403 status if access is forbidden, and a 500 status for server errors.

7. Code Execution

- ***/execute:***
  - Executes a provided code snippet with given test cases. The request body should include the code and test cases. A successful execution returns a 200 status.

7. Code Execution

- ***/execute:***
  - Executes a provided code snippet with given test cases. The request body should include the code and test cases. A successful execution returns a 200 status with test results, while server errors return a 500 status.

8. User Logout

- ***/logout:***
  - Logs out the user by unsetting JWT cookies. It returns a 200 status with a success message. Server errors return a 500 status.

9. Video Chat

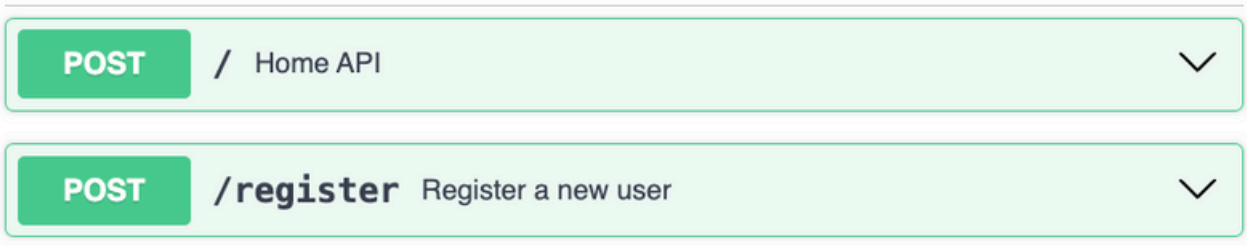
- ***/videochat:***
  - Handles video chat functionality. The request body should include a `message` for the video chat. It returns a 200 status with the chat response, while server errors return a 500 status.

10. Code Chat

- ***/codechat:***
  - Handles code chat functionality. The request body should include a `message` for the code chat. It returns a 200 status with the chat response, while server errors return a 500 status.

11. Support Chat

- ***/supportchat:***
  - Handles support chat functionality. The request body should include a `message` for the support chat. It returns a 200 status with the chat response, while server errors return a 500 status.




POST /login Login user
GET /user Get users
DELETE /user/{user_id} Delete user
GET /study Get study content
PUT /study Edit course
GET /study/lectures Get lectures
POST /study/lectures Add lecture
DELETE /study/lectures/{lecture_id} Delete lecture
GET /dashboard/admin Admin dashboard
POST /execute Execute code
POST /logout Logout
POST /videochat Video Chat
POST /codechat Code Chat
POST /supportchat Support Chat

MILESTONE 5

Test Cases

Introduction

This document outlines the test cases for the GenAI-enhanced programming learning environment project, aimed at integrating Generative AI into educational platforms like the SEEK portal used in the IITM BS degree program. The project seeks to leverage GenAI to enhance learner engagement, provide effective feedback, and support problem-solving activities. Given the importance of ensuring reliable and seamless integration of GenAI technologies, this document provides a comprehensive set of test cases designed to validate the functionality, performance, and reliability of the system components, including APIs, GenAI interactions, and user interface elements. These test cases serve as a roadmap for systematically evaluating the application's capabilities and ensuring a high-quality user experience.

Testing Framework

The testing framework for this project is built upon Python's pytest, a robust and flexible testing tool known for its simplicity and scalability. The framework covers a wide range of tests, including unit tests to verify individual functions, integration tests to ensure components work together as expected, and system tests to validate the application as a whole. Each test case is meticulously designed to assess specific functionalities, such as API responses, data processing, GenAI model integration, and user interactions. The framework also incorporates continuous integration practices to automate testing, allowing for rapid identification and resolution of issues during development. This structured approach ensures that the system not only meets functional requirements but also maintains performance standards under various conditions.

Test Cases

1. Home API

```python
import pytest
from flask import Flask, jsonify
from flask.testing import FlaskClient
from application import app, db
from application.models import User, Course, Assignment, Announcement, Lecture, CourseDocs
from flask_bcrypt import Bcrypt

bcrypt = Bcrypt()

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        with app.app_context():
            db.create_all()
        yield client
        with app.app_context():
            db.drop_all()

def test_home_api(client: FlaskClient):
    response = client.post('/')
    assert response.status_code == 200
    assert response.json == {"message": "Welcome to the App"}
```

Test Case: Home API Welcome Message

- ***API Endpoint:*** */*
- ***Request Method:*** POST
- ***Inputs:*** None
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"message": "Welcome to the App"}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"message": "Welcome to the App"}

2. Registration API

```python
def test_register_new_user(client: FlaskClient):
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student", ##############################################
        "courses": ["python"]
    }
    response = client.post('/register', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "User created successfully", "code": 201}

def test_register_existing_user(client: FlaskClient):
    # First, register a user
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student"
    }
    client.post('/register', json=data)
    response = client.post('/register', json=data)
    assert response.status_code == 400
    assert response.json == {"error": "This username is already taken", "code": 400}
```

Test Case: Register Existing User

- ***API Endpoint:*** /register
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "This username is already taken", "code": 400}
- ***Actual Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "This username is already taken", "code": 400}
- ***Result:*** Success

Test Case: Register New User

- ***API Endpoint:*** /register
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "User created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "User created successfully", "code": 201}
- ***Result:*** Success

3. Login API

```python
def test_valid_login(client: FlaskClient):
    data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123",
        "email": "student@ds.study.iitm.ac.in",
        "mob": "1234567890",
        "name": "Soft Engg Student"
    }
    client.post('/register', json=data)

    # Now, test login
    login_data = {
        "username": "iitmbsstudent",
        "password": "iitmbs123"
    }
    response = client.post('/login', json=login_data)
    assert response.status_code == 200
    assert "token" in response.json

def test_invalid_login(client: FlaskClient):
    login_data = {
        "username": "invalidiitmuser",
        "password": "wrongiitmbspassword"
    }
    response = client.post('/login', json=login_data)
    assert response.status_code == 400
    assert response.json == {"error": "Invalid credentials", "code": 400}
```

Test Case: Valid Login

- ***API Endpoint:*** /login
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"token": "<jwt_token>", "code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"token": "<jwt_token>", "code": 200}
- ***Result:*** Success

Test Case: Invalid Login

- ***API Endpoint:*** /login
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "Invalid credentials", "code": 400}
- ***Actual Output:***
  - HTTP Status Code: 400
  - JSON: {"error": "Invalid credentials", "code": 400}
- ***Result:*** Success

4. Dashboard API

```python
def test_dashboard_api(client: FlaskClient):
    # We are assuming user with id 1 exists
    response = client.get('/dashboard')
    assert response.status_code == 200
    assert "dashboard" in response.json
```

Test Case: Fetch Dashboard Data

- ***API Endpoint:*** /dashboard
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"dashboard": {"user": {"id": 1,"name": "Test User"},"deadlines": [...],"announcements": [...]},"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

5. Study API

```python
def test_study_api(client: FlaskClient):
    response = client.get('/study')
    assert response.status_code == 200
    assert "study" in response.json
```

Test Case: Fetch Study Content

- ***API Endpoint:*** /study
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"study": [{"course_id": 1,"course_name": "Course Name","course_description": "Course Description"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

```python
def test_create_new_course(client: FlaskClient):
    data = {
        "course_name": "Python",
        "course_description": "Python is a Programming Langyage.",
        "start_date": "2024-05-28",
        "end_date": "2024-09-01"
    }
    response = client.post('/study', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "Course created successfully", "code": 201}
```

Test Case: Create New Course

- ***API Endpoint:*** /study
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Course created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Course created successfully", "code": 201}
- ***Result:*** Success

6. Lectures API

```python
def test_fetch_lectures(client: FlaskClient):
    # We are assuming course with id 1 exists
    response = client.get('/study/lectures')
    assert response.status_code == 200
    assert "lectures" in response.json
```

Test Case: Fetch Lectures

- ***API Endpoint:*** /study/lectures
- ***Request Method:*** GET
- ***Expected Output:***
  - HTTP Status Code: 200
  - {"lectures": [{"lecture_id": 1,"lecture_title": "Lecture Title","lecture_link": "http://lecture.link","lecture_date": "2024-01-15","description": "Lecture description"}],"code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: Matches expected output
- ***Result:*** Success

```python
def test_create_new_lecture(client: FlaskClient):
    data = {
        "lecture_title": "Introduction to Python",
        "lecture_link": "https://www.youtube.com/watch?v=8ndsDXohLMQ&list=PLZ2ps__7DhBb2cXAu5PevO_mzgS3Fj3Fs",
        "lecture_date": "2024-05-28",
        "lecture_description": "Introductory lecture to Python Programming Language"
    }
    response = client.post('/study/lectures', json=data)
    assert response.status_code == 201
    assert response.json == {"message": "Lecture created successfully", "code": 201}
```

Test Case: Create New Lecture

- ***API Endpoint:*** /study/lectures
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Lecture created successfully", "code": 201}
- ***Actual Output:***
  - HTTP Status Code: 201
  - JSON: {"message": "Lecture created successfully", "code": 201}
- ***Result:*** Success

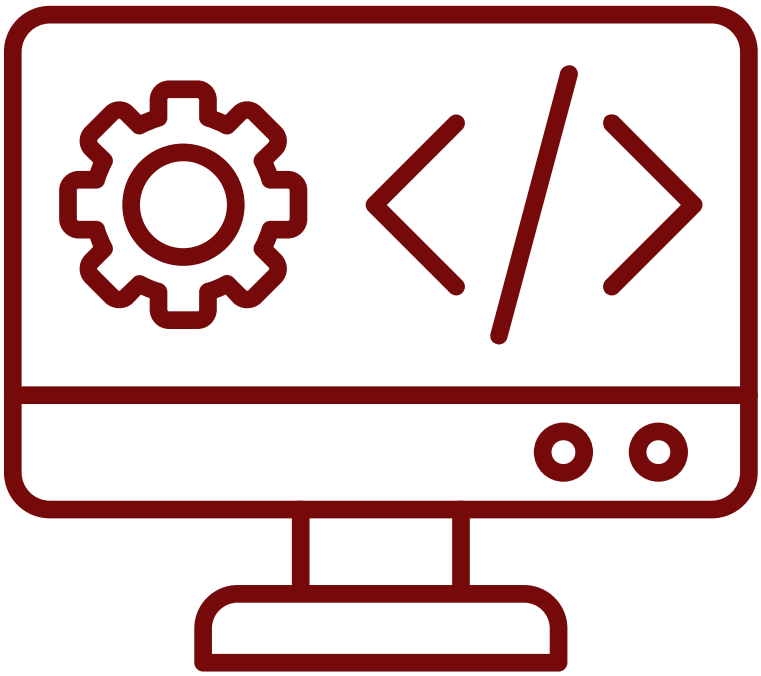
7. Execute Code API

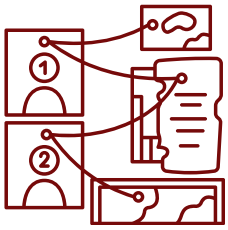
```python
def test_execute_python_code(client: FlaskClient):
    code = {
        "code": "print('Hello, World!')"
    }
    response = client.post('/execute', json=code)
    assert response.status_code == 200
    assert response.json['output'] == "Hello, World!\n"
```

Test Case: Execute Python Code

- ***API Endpoint:*** /execute
- ***Request Method:*** POST
- ***Expected Output:***
  - HTTP Status Code: 200
  - JSON: {"output": "Hello, World!\n", "code": 200}
- ***Actual Output:***
  - HTTP Status Code: 200
  - JSON: {"output": "Hello, World!\n", "code": 200}
- ***Result:*** Success

```
PS C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application> pytest testcases.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.4, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application
plugins: anyio-4.2.0
collected 16 items

testcases.py ................                                                                    [100%]

============================================== warnings summary ===============================================
testcases.py::test_valid_login
  C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application\controllers.py:47: DeprecationWarning: datetime.datetime.utcn
ow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.U
TC).
    first_login_time=datetime.utcnow()

testcases.py::test_valid_login
  C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application\token_validation.py:23: DeprecationWarning: datetime.datetime
.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datet
ime.UTC).
    token=jwt.encode({'id':user.id, 'role': user.role, 'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=100)},app.config['SECRET_KEY'], algori
thm="HS256" )

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 16 passed, 2 warnings in 5.22s ==========================================
PS C:\Users\satis\Desktop\noGenAI\soft-engg-project-may-2024-se-may-27\Code\backend\application>
```





Conclusion

In conclusion, the testing framework and test cases detailed in this document provide a comprehensive approach to validating the integration of GenAI into the SEEK learning environment. Through rigorous testing, we aim to ensure that the application is reliable, efficient, and capable of delivering a seamless and engaging learning experience. By systematically covering all aspects of the application, from individual components to the overall system, we strive to identify and mitigate potential issues early in the development process. This commitment to quality assurance lays the foundation for successful deployment and future enhancements of the GenAI-enhanced learning platform, ultimately transforming how learners interact with educational content and utilize AI-driven insights for improved learning outcomes.

MILESTONE 6

Final Submission

Final Submission

1. Summary of Work Done in Different Milestones

Milestone 1: Identify User Requirements

- Conducted user research to identify key user requirements for the SEEK portal.
- Developed user scenarios both with and without GenAI integration to enhance the portal's usability.

Milestone 2: User Interfaces

- Created detailed user flows and low-fidelity wireframes to visualize the application's structure.
- Outlined user stories to guide the development process, focusing on both client and admin users.

Milestone 3: Scheduling and Design

- Planned and designed the project timeline using a sprint-based approach.
- Developed detailed project components and class diagrams to support implementation.

Milestone 4: API Endpoints

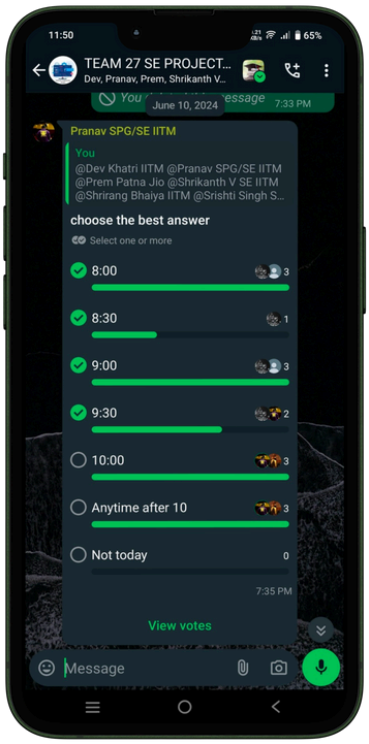
- Designed and implemented API endpoints for key functionalities such as user authentication, course management, and GenAI interactions.

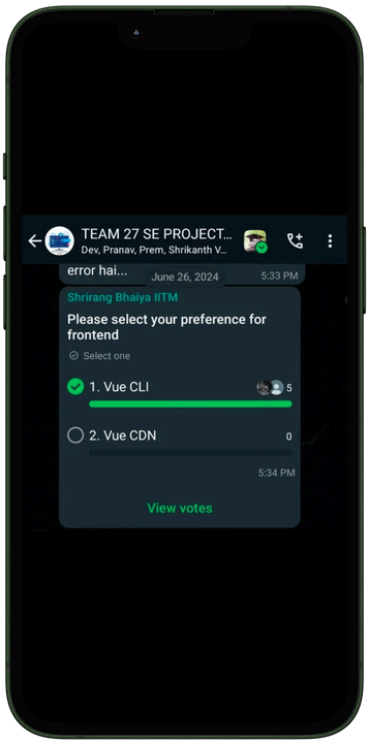
Milestone 5: Test Cases

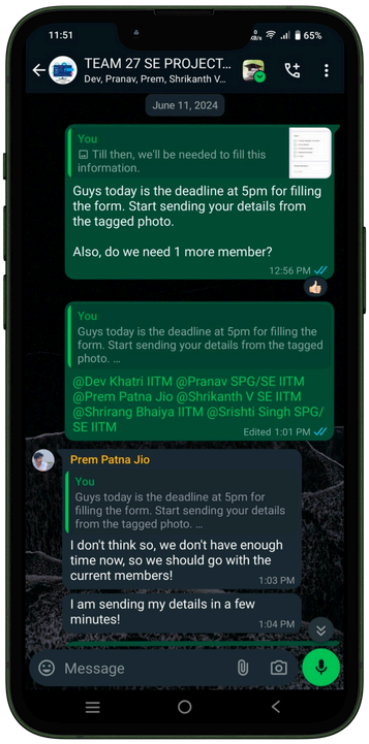
- Developed and executed test cases to ensure the reliability and performance of the implemented features. Covered unit, integration, and system testing.

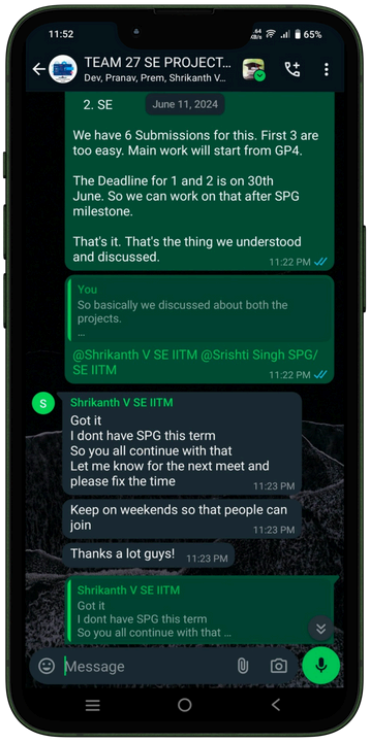
2.1 Issue Tracking and Reporting

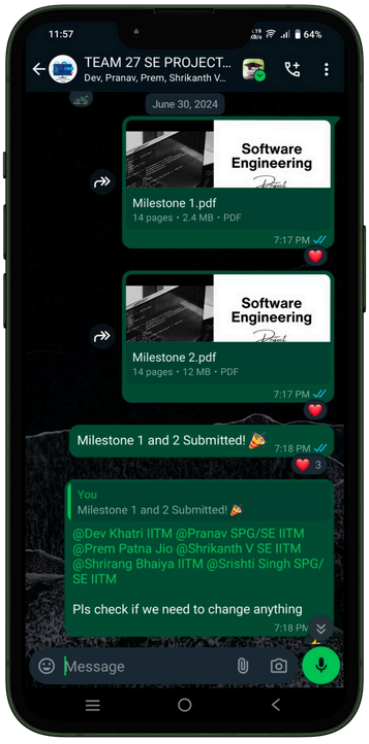
We have created a WhatsApp group for constant communication and easy access to review or code. Below are a few screenshots attached to view.

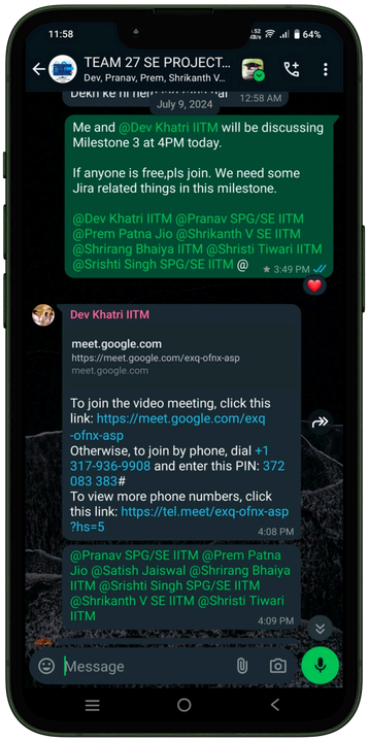




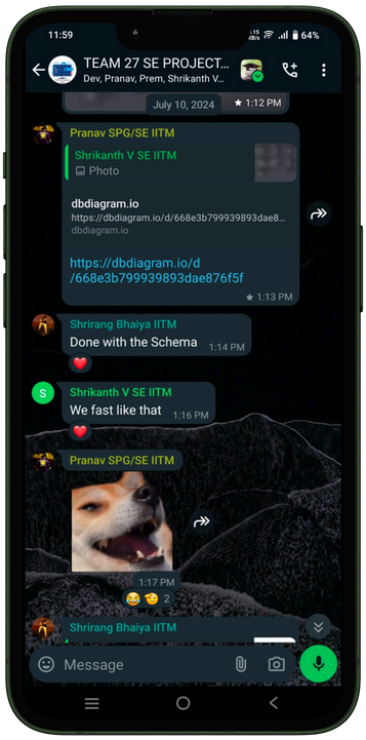


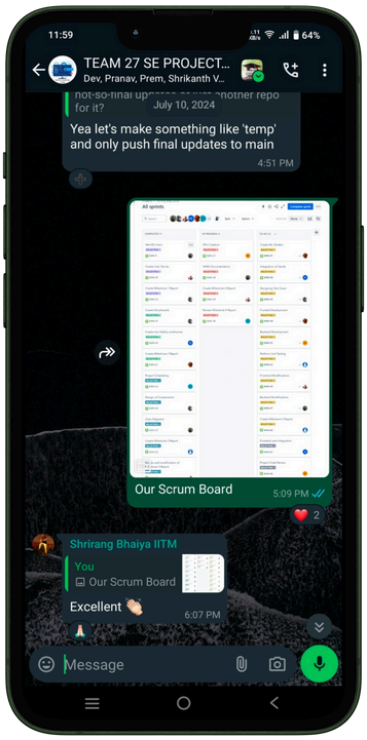


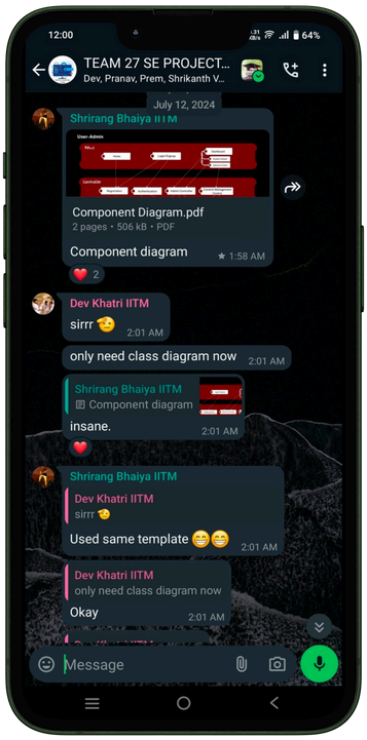


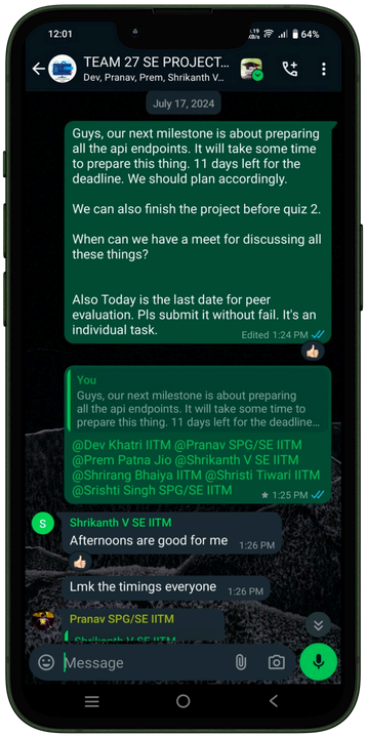


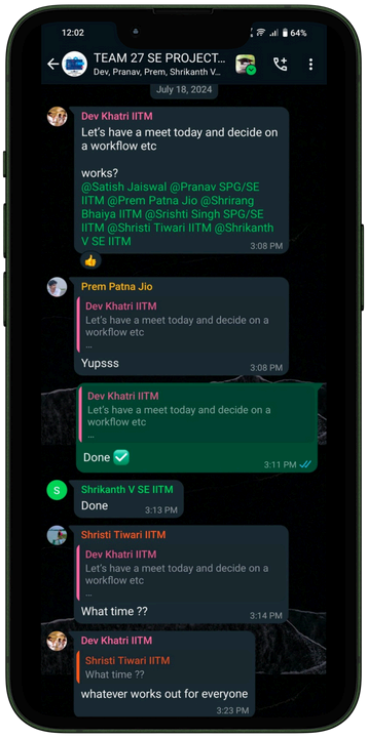
2.2 Issue Tracking and Reporting

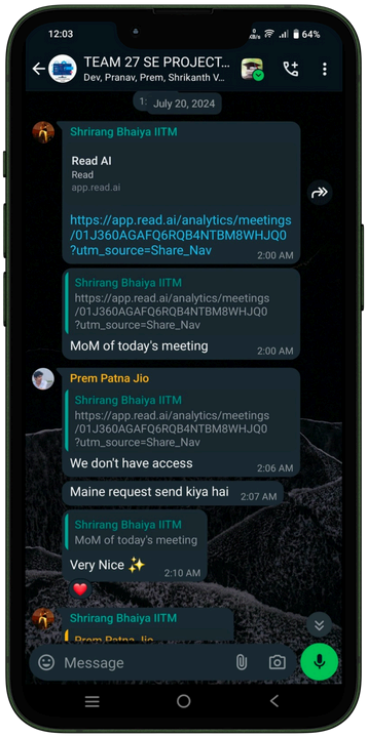




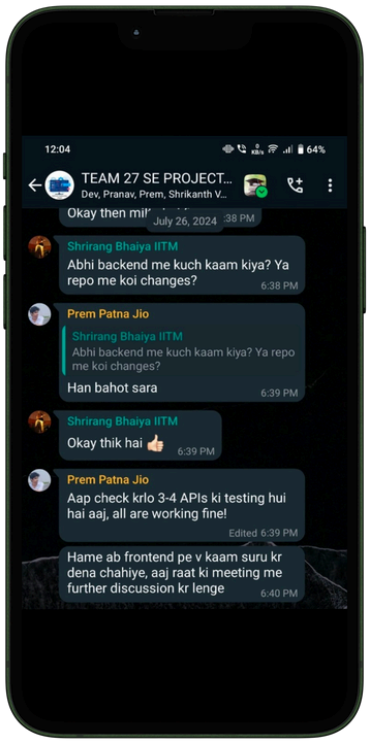


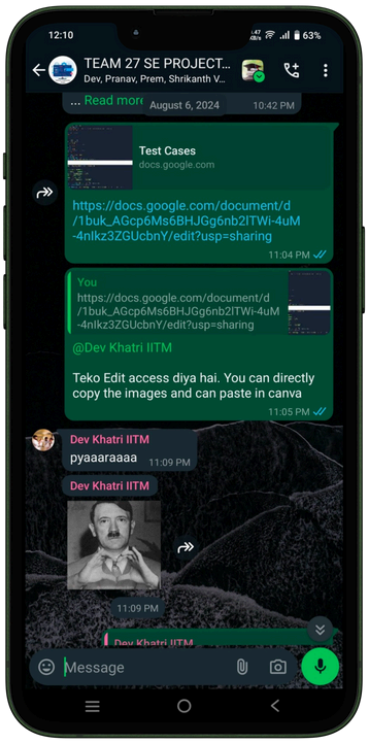


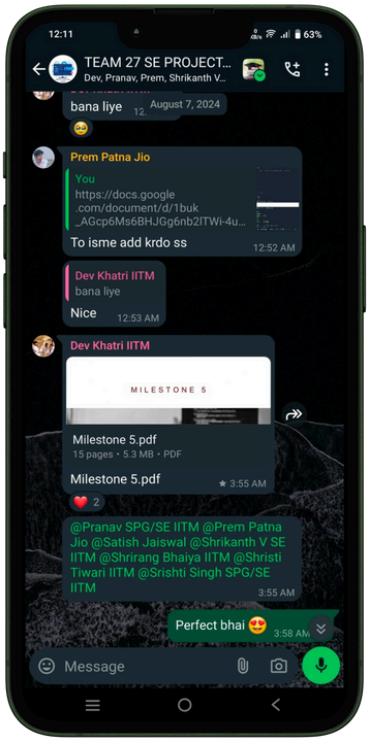


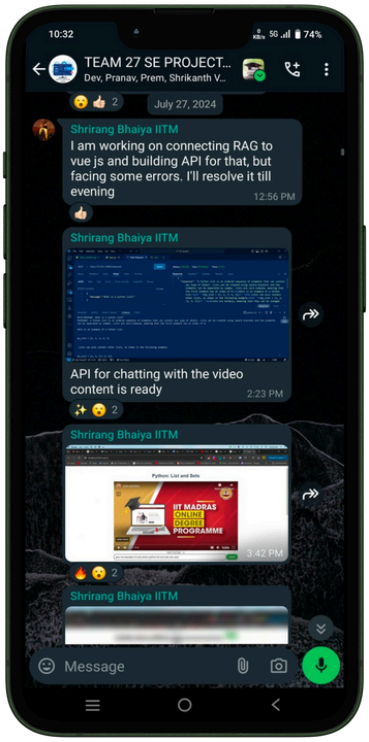


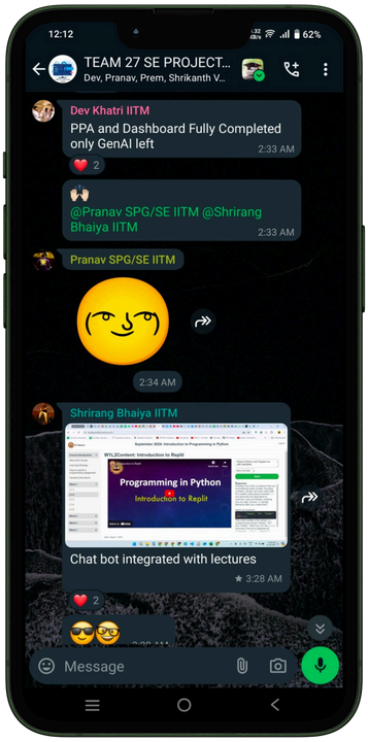
2.3 Issue Tracking and Reporting











3. Implementation Details of the Project

Backend Implementation:

- Developed a RESTful API to manage user authentication, course content, and GenAI interactions.
- Implemented GenAI features such as feedback on programming assignments, study plan generation, and personalized support.

Frontend Implementation:

- Built a user-friendly dashboard that integrates GenAI functionalities for enhanced user interaction.
- Ensured smooth navigation between various components like course content, assignments, and user profiles.

Database Schema:

- Designed a comprehensive database schema to manage users, courses, assignments, and support requests.
- Incorporated relationships between tables to support the GenAI-driven features.

4. Tools and Technologies Used

1. Technologies for the backend:

- Flask, Flask Restful (For creating API endpoints), Flask SQLAlchemy, Pytest (Only for testing), Swagger Editor (Only for API documentation), Thunderclient (For checking API endpoints manually)

2. Technologies for the frontend:

- Vue 3 CLI, Vue Router, Vuex Store, JavaScript, Bootstrap (for aesthetics and styling), HTML and CSS, ESlint to help with debugging the frontend, Axios and Fetch API for requests.

3. General technologies used:

- GitHub (for versioning, code management, tracking, reviewing, issues, etc.), Canva (for making reports), Jira (for project management), draw.io (app.diagrams.net/) (for creating UML diagrams)

5. Instructions to run our application

Getting Started // Prerequisites

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

Python 3.10

Pip

Node.js

A. Creation of Virtual Environment

cd /mnt/c/Users/.....(path to our project folder)

python3 -m venv myenv

B. Start the virtual environment

myenv/scripts/activate

C. Package Installation

Install the required packages inside myenv

pip install -r requirements.txt

D. Install node modules

cd /mnt/c/Users/.....(path to our frontend folder)

npm install

E. Start the frontend

npm run serve

F. Start the backend

cd /mnt/c/Users/.....(path to our backend folder)

python app.py (windows)

python3 app.py (mac)

Open the app in browser by navigating at http://localhost:8080/

Software Engineering

Project

TEAM 27

Project By :

Dev Khatri (21f3001150)
Satish Jaiswal (21f2000142)
Prem Kumar (21f1000531)
Srishti Singh (21f1006972)
Shrikanth V (21f1002328)
Shrirang Sapate (21f1002870)
Pranav R (21f1004199)
Shristi Tiwari (21f2000589)

Thank You!