Final Project Report

Links: Demo video Presentation slides

Milestone 1 submission

Types of users :

Primary users:

- Students

Secondary users:

- Admin

Tertiary users:

- Course Professors or Instructors
- IITM BS academics team or student affairs team
- IITM BS Portal management team

Users Stories :

Primary Users

User Story 1: Getting input from students to recommend personalized learning paths

As a student,
I want to input my academic interests, learning goals, schedules, and commitments,

Final Project Report

so that the system considers my current preferences and constraints to recommend a personalized learning path.

User Story 2: Considering past performances

As a student,
I want the system to consider my past performance in previous courses,
so that the recommendations align with my academic strengths and weaknesses.

User Story 3: Sharing course feedback

As a student,
I want to share feedback on each course I have taken or finished,
so that the system can consider my sentiments and ratings to enhance the recommendation engine.

User Story 4: Feedback on the learning paths

As a student,
I want to provide feedback on the recommended learning paths,
so that the system can utilize my sentiments and ratings in the refinement of the recommendation engine.

User Story 5: Enhancing recommendations through feedback

As a student,
I want the system to consider both individual and peer feedback during course recommendations,
so that my learning experience is personalized and benefits from collective wisdom.

User Story 6: Providing multiple learning paths

As a student,
I want to view multiple learning paths based on the same input provided,
so that I have diverse options to explore and choose from.

Final Project Report

User Story 7: Exploring popular learning paths

As a student,
I want to explore different popular learning paths taken by my peers who are similar to me,
so that I can gain insights, broaden my academic horizons, and make informed decisions about my own learning journey.

User Story 8: Secure student interface

As a student,
I want to prevent other students from accessing my personal data,
so that I have confidence in the confidentiality and integrity of my data.

Secondary Users

User Story 1: Enrollment Data Management (CRUD)

User story 1.1 Upload Enrollment Data (create)

As an admin,
I want to upload enrollment data, such as student's learning profile, including their past academic performance, interests, and goals from previous terms,
so that the system can use it to generate learning path recommendations.

User story 1.2 Verify Enrollment Data (Read)

As an admin,
I want to view or download a summary report of the uploaded enrollment data,
so that I can verify its accuracy.

Final Project Report

User story 1.3 Manage Enrollment Data (Update and Delete)

As an admin,
I want to have the option to edit and delete uploaded enrollment data in case of errors,
so that I can maintain data accuracy and reliability.

User Story 2: Data security and privacy

As an admin,
I want the access control feature to ensure secure access to enrollment data and prevent unauthorized entry.
so that data integrity is maintained.

User Story 3: Analytics

As an admin,
I want to access student engagement and satisfaction metric on the effectiveness of learning path recommendations,
so that I can make informed decisions for system improvement.

User Story 4: Considering course prerequisites and corequisites

As an admin,
I want to efficiently manage courses, including their prerequisites, corequisites, and required minimum hours of study, by performing actions such as inserting, updating, and deleting,
so that learning path recommendations align with academic requirements and regulations.

User Story 5: Admin feedback for learning path reinforcement

As an admin,
I want to provide feedback or ratings on recommended learning paths,
so that the recommendation engine can be reinforced and fine-tuned for improved suggestions.

Final Project Report

Tertiary Users

User Story 1: Assessing course effectiveness

As a course professor or instructor,
I want to get alerted if the rating of my course is less than 3,
so that I can evaluate the effectiveness of my teaching methods and make improvements.

User Story 2: Evaluating recommendation engine efficiency

As a member of the student affairs or academic team,
I want to access data on student engagement with recommended learning paths,
so that I can verify the efficiency and effectiveness of the recommendation engine.

Milestone 2 submission

- **StoryBoard Animation: link**
- **StoryBoard PPT: link**
- **StoryBoard Video: link**
- **Wireframe: link**

Final Project Report

Milestone 3 submission

Design of Components

1. Login Screen(Admin and Student)

a. Create New user/Sign-Up page
b. Create Sign-in Page

2. Admin Dashboard

a. View Summary of Student and Courses
b. View Uploaded Student Log data
c. Upload Student data
    - Upload Performance, Interest and Goal of student from previous Term
    - Uploaded Enrollment Data
    - Download Summary Report of Uploaded Enrollment Data
    - View Summary Report of Uploaded Enrollment Data
d. Course Data
    - View Course data with prerequisites, corequisites, and required minimum hours of study
    - Create Course data with prerequisites, corequisites, and required minimum hours of study
    - Delete Course data with prerequisites, corequisites, and required minimum hours of study
    - Edit Course data with prerequisites, corequisites, and required minimum hours of study

Final Project Report

e. Learning Paths
    - View all Generated each Learning path
    - Add Rating and Feedback for each Learning path
    - Edit Rating and Feedback for each Learning path
    - Create Rating and Feedback for each leaning path

3. Student Dashboard

a. Show Current Learning Path
b. Show and add Feedback on Completed Course
c. Show upcoming Course
d. Show all Learning Path generated by Recommendation system
e. Add Feedback and Rating on generated by Recommendation system
f. Show Previous Recommended Learning Path
    - Add new learning Path
    - Add Academic interests.
    - Learning Goal
    - Schedule
    - Commitments

4. Common

a. Data Preparation for Each Table
b. Course professor or instructor will get alert if course rating is less than 3
    - Dynamic Report Template creation
    - Email formatting
    - Write Schedule JOB to send Email if Rating of Course Feedback in Less than 3
    - Email JOB scheduler
    -

Final Project Report

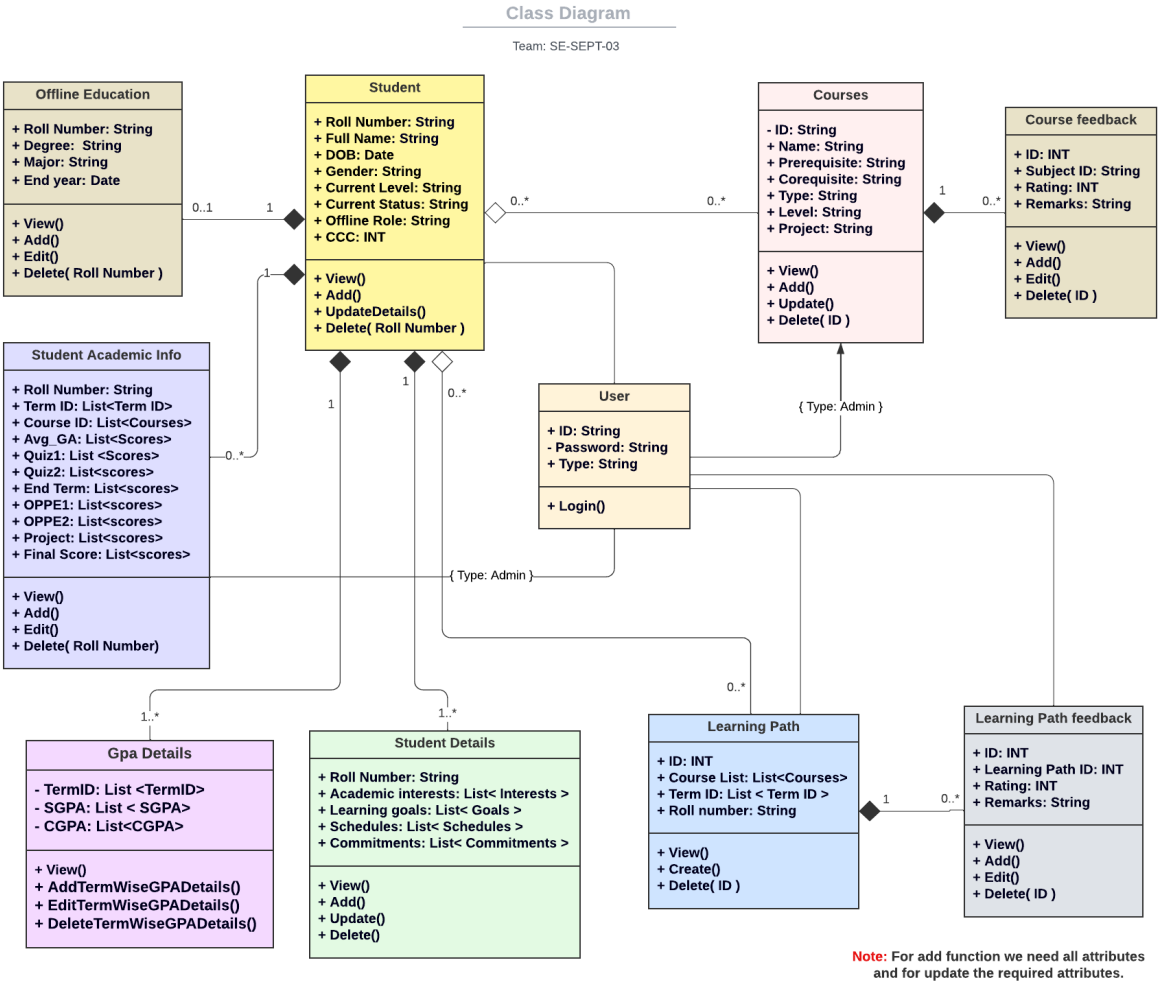
- Email Template for Notification on Performance on Recommendation Engine

c. ML model for Recommendation Engine

- Data Cleaning
- Feature Engineering
- Model Architecture
- Model Training
- Model Evaluation
- Hyper parameter Tuning
- Model Deployment
- Saving Model checkpoint into Google Drive Storage
- API Integration with Application

Final Project Report

Class Diagram



Final Project Report
Sprint Schedules

Sprint 1 (28-sep to 12-oct)

- Identify Users and write user stories

Sprint 2 (13-oct to 26-oct)

- Create a storyboard for the application
- Create low-fidelity wireframes

Sprint 3 (28-oct to 2-Nov)

- Project Scheduling and Jira setup
- Create Class diagram
- Create Components

Sprint 4 (1-Nov to 8-Nov)

- Create Academic interests, Learning goals, schedules, and commitments
- Edit a existing Create Academic interests, Learning goals, schedules, and commitments
- Delete Academic interests, Learning goals, schedules, and commitments
- Create View to read data during Model preparation
- View the Term and Subject Wise score in Student Dashboard
- View all the courses registered till now
- Create Feedback for each course
- Edit Feedback for each course
- Delete Feedback for each course

Final Project Report

Sprint 5 (9-Nov to-16 Nov)

- Data Cleaning
- Feature Engineering
- Model Architecture
- Model Training
- Model Evaluation
- Hyper parameter Tuning
- Model Deployment
- Saving Model checkpoint into Google Drive Storage
- API Integration with Application

Sprint 6 (17-Nov to 24-Nov)

- View uploaded Enrollment Data
- Edit uploaded Enrollment Data
- Delete uploaded Enrollment Data
- Create Signup Page for Login
- Database validation for Access
- View all Generate Learning Path
- Compare Generate Path with Student actually selected Path with Score
- Create Course data with prerequisites, corequisites, and required minimum hours of study
- Edit Course data with prerequisites, corequisites, and required minimum hours of study
- Delete Course data with prerequisites, corequisites, and required minimum hours of study
- View Course data with prerequisites, corequisites, and required minimum hours of study

Final Project Report

Sprint 7 (25-Nov to 2-Dec)

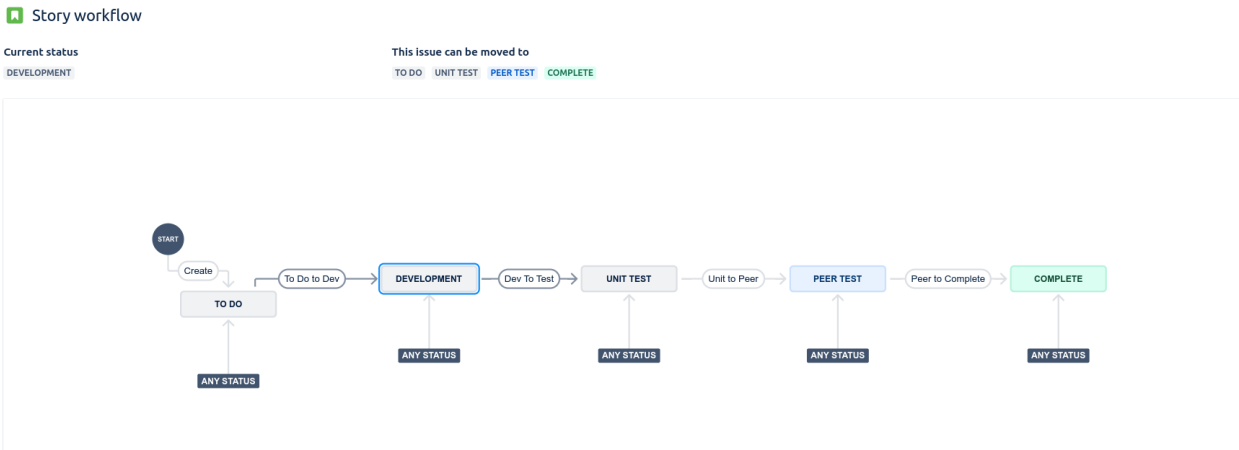
- Uploaded Enrollment Data
- Upload Performance, Interest and Goal of student from previous Term
- View Summary Report of Uploaded Enrollment Data
- Download Summary Report of Uploaded Enrollment Data
- Create Rating and Feedback for each leaning path
- Edit Rating and Feedback for each Learning path
- Edit Rating and Feedback for each Learning path
- Create Sign-in Page
- Create New user/Sign-Up page
- Database validation for Admin and Student during login

Sprint 8 (3 Dec to 10 Dec)

- Write Schedule JOB to send Email if Rating of Course Feedback in Less than 3
- Email formatting
- Dynamic Report template creation
- Email Template for Notification on Performance on Recommendation Engine
- Email JOB scheduler

[Link To Detail Scheduling]

WorkFlow



Final Project Report

Scrum Meeting Schedule and Minutes

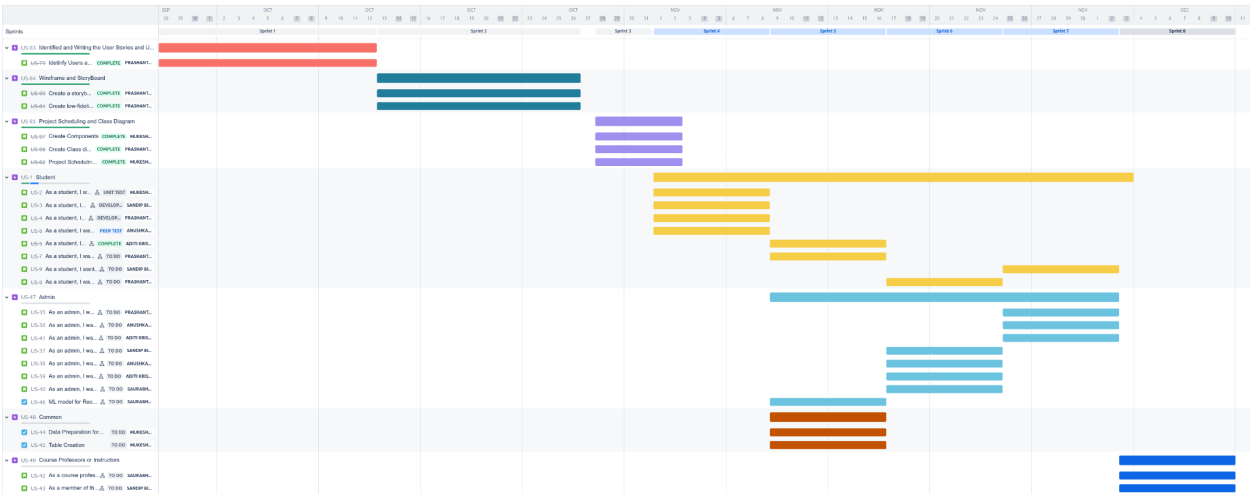
Sprint	Date	Description	Resp	Deadline
Sprint 1	28/09/2023	Go through first two weeks of lectures	Everyone	1/10/2023
Sprint 1	28/09/2023	Read the problem statement and imagine user stories in the suggested format	Everyone	5/10/2023
Sprint 1	28/09/2023	Explore online tools for preparing user stories	Sandip	6/10/2023
Sprint 1	7/10/2023	Preparation of User Stories	Prashant, Aditi	07/10/2023
Sprint 1	7/10/2023	Going through Each user stories and evaluate according to SMART Guidelines	Everyone	9/10/2023
Sprint 1	11/10/2023	Going through Data upload templates to decide user requirements	Everyone	11/10/2023
Sprint 1	11/10/2023	Finalize submission for Milestone-1 and make submission	Prashant	13/10/2023
Sprint 1	12/10/2023	Discussion about logic behind recommendation engine	Everyone	13/10/2023
Sprint 2	23/10/2023	Go through Milestone 2 requirements	Everyone	23/10/2023
Sprint 2	23/10/2023	Review tools to be used for generating user story boards	Everyone	24/10/2023
Sprint 2	23/10/2023	Generate user story boards and wireframe	Prashant	24/10/2023
Sprint 2	23/10/2023	Review user story boards and make necessary changes	Everyone	25/10/2023
Sprint 2	23/10/2023	Prepare submission file for Milestone-2 and make submission	Prashant	26/10/2023
Sprint 3	30/10/2023	Go through milestone-3 requirements and plan future meetings and preparations.	Everyone	30/10/2023
Sprint 3	30/10/2023	Initial discussion on models to be considered for back end and iron out attributes and methods for each model	Everyone	30/10/2023
Sprint 3	31/10/2023	Initial scheduling for the project using JIRA and creating tasks and sprints	Mukesh	02/11/2023

Final Project Report

Sprint 3	31/10/2023	Class diagram to be prepared	Prashant	02/11/2023
Sprint 3	2/10/2023	Reviewing the Class Diagram	Everyone	02/11/2023
Sprint 3	2/10/2023	Component Diagram Discussion	Everyone	02/11/2023
Sprint 3	2/10/2023	Task assignment	Mukesh	02/11/2023
Sprint 3	3/10/2023	Review and Discuss Milestone 3 submission files	Everyone	03/11/2023
Sprint 3	3/10/2023	Finalize submission for Milestone-3 and make submission	Prashant	03/11/2023

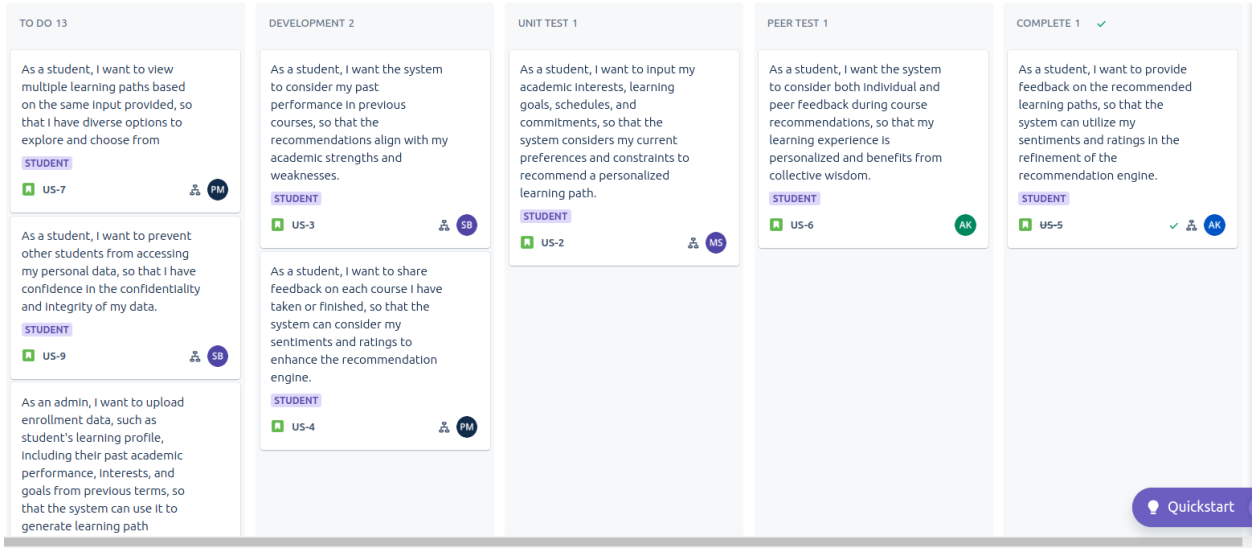
Final Project Report

GANTT Chart



[View full chart]

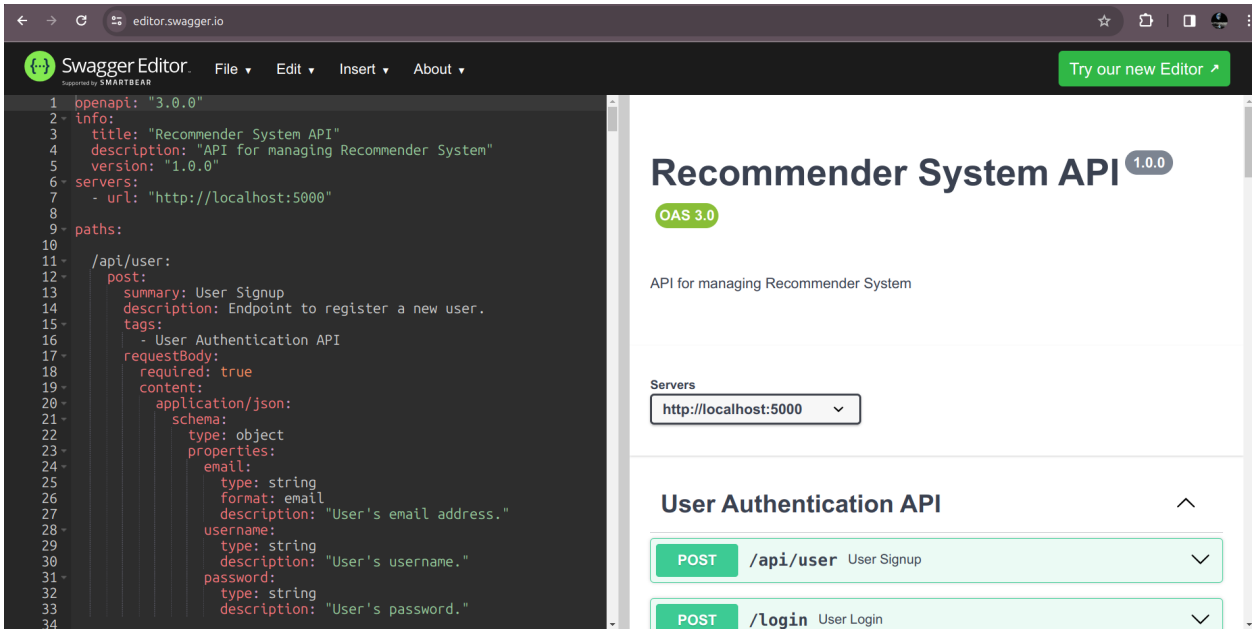
Partial Kanban Board



Final Project Report

Milestone 4 submission

API Documentation



[YAML Documentation Link]

Milestone 5 submission

TESTING

User Story :- Secure student interface

API being tested : http://127.0.0.1:5000/api/user

Inputs :

Request Mode :- **POST**

Json Body as

```
{
    "email": "testuser@example.com",
    "username": "testuser",
    "password1": "testpassword",
}
```

Final Project Report

Expected output:

HTTP Status Code =200

Return Json Result as

```
{
   "username": "testuser",
   "email": "testuser@example.com"
}
```

Actual Output :

HTTP Status Code =200

Return Json Result as

```
{
   "username": "testuser",
   "email": "testuser@example.com"
}
```

Result- Success

```python
def test_user_signup():
    api_url = 'http://127.0.0.1:5000/api/user'
    data = {
        "email": "testuser@example.com",
        "username": "testuser",
        "password1": "testpassword",
    }
    # Make the POST request
    response = requests.post(api_url, json=data)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Providing multiple learning paths

API being tested : http://127.0.0.1:5000/api/learning_path/1

Inputs :

Request Mode :- **POST**

Json Body as

```
{
    "term_1": ["BDM", "MAD1", "DBMS", "MAD1 Project"],
    "term_2": ["PDSA", "MLF", "MAD2", "MAD2 Project"],
    "term_3": ["BA", "MLT", "Java", "TDS"],
    "term_4": ["MLP", "SC", "MLT Project", "MLP Project"]
}
```

Expected output:

HTTP Status Code =200

Return Json Result as

```
{
  "image_path": "learning_path_20231206113952.png"
}
```

Actual Output :

HTTP Status Code =200

Return Json Result as

```
{
  "image_path": "learning_path_20231206113952.png"
}
```

Result- Success

```python
def test_generate_learning_path():
    api_url = 'http://127.0.0.1:5000/api/learning_path/1'
    course_data = {
        "term_1": ["BDM", "MAD1", "DBMS", "MAD1 Project"],
        "term_2": ["PDSA", "MLF", "MAD2", "MAD2 Project"],
        "term_3": ["BA", "MLT", "Java", "TDS"],
        "term_4": ["MLP", "SC", "MLT Project", "MLP Project"]
    }
    data = {
        "course_data": course_data
    }
    response = requests.post(api_url, json=data)
    assert response.status_code == 201
```

Final Project Report

User Story :- Providing multiple learning paths
API being tested : http://127.0.0.1:5000/api/learning_path/1
Inputs :
Request Mode :- **POST**
Json Body as
{ }
Expected output:
HTTP Status Code =400
{
"message": "Request must contain a valid JSON payload"
}
Actual Output :
HTTP Status Code =400
{
"message": "Request must contain a valid JSON payload"
}

Result- Success

```
def test_generate_learning_path_errors():
    api_url = 'http://127.0.0.1:5000/api/learning_path/1'
    # Intentionally omitting the course_data to trigger an error
    data = {}
    # Make the POST request.
    response = requests.post(api_url, json=data)
    # Assert the status code is 400
    assert response.status_code == 400
```

User Story :- Providing multiple learning paths
API being tested : http://127.0.0.1:5000/api/learning_path/1
Inputs :
Request Mode :- **GET**
User ID :- 1
Expected output:
HTTP Status Code =200
Return Json Result as
{

"image_path": "learning_path_20231206113952.png"
}

Actual Output :

HTTP Status Code =200
Return Json Result as
{
"image_path": "learning_path_20231206113952.png"
}

Result- Success

```
def test_current_learning_path():
    api_url = 'http://127.0.0.1:5000/api/learning_path/1'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 200
    assert response.status_code == 200
```

User Story :- Providing multiple learning paths

API being tested : http://127.0.0.1:5000/api/learning_path/x

Inputs :

Request Mode :- **GET**
User ID :- 1

Expected output:

HTTP Status Code =404
{
"message": "No learning path found for the user"
}

Actual Output :

HTTP Status Code =404
{
"message": "No learning path found for the user"
}

Result- Success

Final Project Report

```
def test_current_learning_path_error():
    api_url = 'http://127.0.0.1:5000/api/learning_path/x'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 404
    assert response.status_code == 404
```

User Story :- Exploring popular learning paths
API being tested : http://127.0.0.1:5000/api/top_rated_learning_paths
Inputs :
Request Mode :- **GET**
Expected output:
HTTP Status Code =200
Return Json Result as
[
{"image_path": "learning_path_20231206113952.png"},
{"image_path": "learning_path_20231206113953.png"},
{"image_path": "learning_path_20231206113954.png"},
{"image_path": "learning_path_20231206113955.png"},
{"image_path": "learning_path_20231206113956.png"},
{"image_path": "learning_path_20231206113957.png"}
]

Actual Output :
HTTP Status Code =200
Return Json Result as
[
{"image_path": "learning_path_20231206113952.png"},
{"image_path": "learning_path_20231206113953.png"},
{"image_path": "learning_path_20231206113954.png"},
{"image_path": "learning_path_20231206113955.png"},
{"image_path": "learning_path_20231206113956.png"},
{"image_path": "learning_path_20231206113957.png"}
]
Result- Success

Final Project Report

```
def test_top_rated_learning_paths():
    api_url = 'http://127.0.0.1:5000/api/top_rated_learning_paths'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Exploring popular learning paths
API being tested : http://127.0.0.1:5000/api/learning_path/1
Inputs :
Request Mode :- **GET**
Expected output:
HTTP Status Code =404
Return Json Result as
{
"message": "No learning paths found"
}

Actual Output :
HTTP Status Code =404
Return Json Result as
{
"message": "No learning paths found"
}

Result- Success

```
def test_top_rated_learning_paths_error():
    api_url = 'http://127.0.0.1:5000/api/top_rated_learning_paths'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 404 (Not Found)
    assert response.status_code == 404
```

User Story :- Feedback on the learning paths
API being tested : http://127.0.0.1:5000/api/all_learning_paths

Final Project Report

Inputs :

Request Mode :- **GET**

Expected output:

HTTP Status Code =200

Return Json Result as

{
[
{"image_path": "learning_path_20231206113952.png"},
{"image_path": "learning_path_20231206113953.png"},
{"image_path": "learning_path_20231206113954.png"},
{"image_path": "learning_path_20231206113955.png"},
{"image_path": "learning_path_20231206113956.png"},
{"image_path": "learning_path_20231206113957.png"}
]
}

Actual Output :

HTTP Status Code =200

Return Json Result as

{
[
{"image_path": "learning_path_20231206113952.png"},
{"image_path": "learning_path_20231206113953.png"},
{"image_path": "learning_path_20231206113954.png"},
{"image_path": "learning_path_20231206113955.png"},
{"image_path": "learning_path_20231206113956.png"},
{"image_path": "learning_path_20231206113957.png"}
]
}

Result- Success

```
def test_all_learning_paths():
    api_url = 'http://127.0.0.1:5000/api/all_learning_paths'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Feedback on the learning paths

Final Project Report

API being tested : http://127.0.0.1:5000/api/feedbacks/1
Inputs :
Request Mode :- **GET**
User ID :- 1
Expected output:
HTTP Status Code =200
Return Json Result as

{
"id": 1,
"learning_path_id": 101,
"rating": 5,
"remarks": "Excellent content!",
"created": "2023-12-06T11:39:52Z"
}

Actual Output :
HTTP Status Code =200
Return Json Result as

{
"id": 1,
"learning_path_id": 101,
"rating": 5,
"remarks": "Excellent content!",
"created": "2023-12-06T11:39:52Z"
}

Result- Success

```
def test_learning_path_feedbacks(learning_path_id):
    api_url = f'http://127.0.0.1:5000/feedbacks/1'
    # Make the GET request
    response = requests.get(api_url)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Feedback on the learning paths
API being tested : http://127.0.0.1:5000/api/feedbacks/1
Inputs :
Request Mode :- **POST**

Final Project Report

User ID :- 1
Jason data as
{
'rating': 5,
'remarks': 'Great learning experience!',
}

Expected output:

HTTP Status Code =201
Return Json Result as
{
"learning_path_id": 101,
"rating": 5,
"remarks": "Excellent content!",
"created": "2023-12-06T11:39:52Z"
}

Actual Output :

HTTP Status Code =200
Return Json Result as
{
"learning_path_id": 101,
"rating": 5,
"remarks": "Excellent content!",
"created": "2023-12-06T11:39:52Z"
}

Result- Success

Final Project Report

```python
def test_post_learning_path_feedbacks(learning_path_id):
    api_url = 'http://127.0.0.1:5000/feedbacks/1'
    # Define the data you want to send in the POST request
    data = {
        'rating': 5,
        'remarks': 'Great learning experience!',
    }
    # Make the POST request
    response = requests.post(api_url, json=data)
    # Assert the status code is 201 (Created)
    assert response.status_code == 201
```

User Story :- Feedback on the learning paths
API being tested : http://127.0.0.1:5000/api/feedbacks/1
Inputs :

Request Mode :- **POST**
User ID :- 1
Jason data as
{
'rating': -1,
'remarks': 'Great learning experience!',
}

Expected output:

HTTP Status Code =201
Return Json Result as
{
"learning_path_id": 101,
"rating": 5,
"remarks": "Excellent content!",
"created": "2023-12-06T11:39:52Z"
}

Actual Output :

Error throws in pytest

Result- Fail

Final Project Report

```python
def test_post_learning_path_feedbacks_error(learning_path_id):
    api_url = 'http://127.0.0.1:5000/feedbacks/1'
    # Define the data with missing required fields for error case
    data = {
        'remarks': 'Great learning experience!',
    }
    # Make the POST request
    response = requests.post(api_url, json=data)
    # Assert the status code is 400 (Bad Request)
    assert response.status_code == 400
```

User Story :- Considering course prerequisites and corequisites
API being tested : http://127.0.0.1:5000/api/courses
Inputs :
Request Mode :- **GET**
Expected output:
HTTP Status Code =200
Return Json Result as

{
[
{
"id": 1,
"name": "BDM",
"prerequisite": "",
"corequisite": "",
"type_": "Data Science",
"level": "2",
"project": "BDM Project"
},
{
"id": 2,
"name": "MAD1",
"prerequisite": "",
"corequisite": "",
"type_": "Programming",
"level": "2",
"project": "MAD1 Project"
},

{
"id": 3,
"name": "MAD2",
"prerequisite": "MAD1",
"corequisite": "",
"type_": "Programming",
"level": "2",
"project": "MAD2 Project"
},
]
}

Actual Output :

HTTP Status Code =200

Return Json Result as

{
[
{
"id": 1,
"name": "BDM",
"prerequisite": "",
"corequisite": "",
"type_": "Data Science",
"level": "2",
"project": "BDM Project"
},
{
"id": 2,
"name": "MAD1",
"prerequisite": "",
"corequisite": "",
"type_": "Programming",
"level": "2",
"project": "MAD1 Project"
},
{
"id": 3,
"name": "MAD2",
"prerequisite": "MAD1",
"corequisite": "",

"type_": "Programming",
"level": "2",
"project": "MAD2 Project"
},
]
}

Result- Success

```
def test_get_courses():
    # Make the GET request
    api_url = 'http://127.0.0.1:5000/api/courses'
    response = requests.get(api_url)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Considering course prerequisites and corequisites

API being tested : http://127.0.0.1:5000/api/courses

Inputs :

Request Mode :- **POST**

Json as

{
'course_id': 4,
'course_name': 'New Course',
'pre_req': 'Prerequisite',
'co_req': 'Corequisite',
'type_': 'Programming',
'level': '4',
'project': 'Project',
}

Expected output:

HTTP Status Code =201

Return Json Result as

{
'course_id': 4,
'course_name': 'New Course',
'pre_req': 'Prerequisite',
'co_req': 'Corequisite',
'type_': 'Programming',

'level': '4',
'project': 'Project',
}

Actual Output :

HTTP Status Code =201

Return Json Result as

{
'course_id': 4,
'course_name': 'New Course',
'pre_req': 'Prerequisite',
'co_req': 'Corequisite',
'type_': 'Programming',
'level': '4',
'project': 'Project',
}

Final Project Report

Result- Success

```
def test_post_course():
    api_url = 'http://127.0.0.1:5000/api/course'
    data = {
        'course_id': 'CS004',
        'course_name': 'New Course',
        'pre_req': 'Prerequisite',
        'co_req': 'Corequisite',
        'type': 'Programming',
        'level': '4',
        'project': 'Project',
    }
    # Make the POST request
    response = requests.post(api_url, json=data)
    # Assert the status code is 201 (Success)
    assert response.status_code == 201
```

User Story :- Considering course prerequisites and corequisites
API being tested : http://127.0.0.1:5000/api/courses/CS004
Inputs :

Request Mode :- **PUT**
Course id:- CS004

Json as

```
{
    'course_name': 'Course',
    'pre_req': 'Updated Prerequisite',
    'co_req': 'Updated Corequisite',
    'type_': 'Data Science',
    'level': '2',
    'project': 'Updated Project',
```

}

Expected output:

HTTP Status Code =200

Return Json Result as

{
'course_name': 'Course',
'pre_req': 'Updated Prerequisite',
'co_req': 'Updated Corequisite',
'type_': 'Data Science',
'level': '2',
'project': 'Updated Project',
}

Actual Output :

HTTP Status Code =200

Return Json Result as

{
'course_name': 'Course',
'pre_req': 'Updated Prerequisite',
'co_req': 'Updated Corequisite',
'type_': 'Data Science',
'level': '2',
'project': 'Updated Project',
}

Result- Success

Final Project Report

```
def test_put_course():
    api_url = 'http://127.0.0.1:5000/api/course/CS004'
    # Replace {id} with the actual course_id
    data = {
        'course_name': 'Course',
        'pre_req': 'Updated Prerequisite',
        'co_req': 'Updated Corequisite',
        'type': 'Data Science',
        'level': '2',
        'project': 'Updated Project',
    }
    response = requests.put(api_url, json=data)
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Considering course prerequisites and corequisites

API being tested : http://127.0.0.1:5000/api/course/CS004

Inputs :

Request Mode :- **PUT**

Course id:- CSS04

Expected output:

HTTP Status Code =400

Actual Output :

HTTP Status Code =400

Result- Success

```
def test_put_course_error():
    api_url = 'http://127.0.0.1:5000/api/course/CSS04'
    # Intentially giving wrong course id
    data = {}
    response = requests.put(api_url, json=data)
    # Assert the status code is 404 (Not Found)
    assert response.status_code == 404
```

Final Project Report

User Story :- Considering course prerequisites and corequisites
API being tested : http://127.0.0.1:5000/api/course/CS004
Inputs :
Request Mode :- **DELETE**
Course id:- CS004
Expected output:
HTTP Status Code =200
Actual Output :
HTTP Status Code =200
Result- Success

```
def test_delete_course():
    api_url = 'http://127.0.0.1:5000/api/courses'
    course_id = 4
    # Make the DELETE request
    response = requests.delete(f'{api_url}/{course_id}')
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

User Story :- Considering course prerequisites and corequisites
API being tested : http://127.0.0.1:5000/api/course/CS004
Inputs :
Request Mode :- **DELETE**
Course id:- 4
Expected output:
HTTP Status Code =500
Actual Output :
HTTP Status Code =500
Result- Success

```
def test_delete_course_error():
    api_url = 'http://127.0.0.1:5000/api/course/CSS004'
    # Make the DELETE request
    response = requests.delete(api_url)
    # Assert the status code is 500 (Success)
    assert response.status_code == 500
```

Final Project Report

User Story :- Enhancing recommendations through feedback
API being tested : http://127.0.0.1:5000/api/CourseFeedback/1
Inputs :

Request Mode :- **GET**
User Id =1

Expected output:

HTTP Status Code =200
Return Json Result as

{
""id"": 1,
""user_id"": 1,
""subject_id"": ""2"",
""rating"": 4,
""remarks"": ""Very good update""
}"

Actual Output :

HTTP Status Code =200
Return Json Result as

{
""id"": 1,
""user_id"": 1,
""subject_id"": ""2"",
""rating"": 4,
""remarks"": ""Very good update""
}

Result- Success

```
def test_api_call_success():
    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/1'

    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 200
    assert response.status_code == 200
```

Final Project Report

User Story :- Enhancing recommendations through feedback
API being tested : http://127.0.0.1:5000/api/CourseFeedback/10
Inputs :

Request Mode :- **GET**
User Id =10

Expected output:

HTTP Status Code = 404 NOT FOUND
Return Json Result as
{
"message": "No feedback for given by user"
}

Actual Output :

HTTP Status Code = 404 NOT FOUND
Return Json Result as
{
"message": "No feedback for given by user"
}

Result- Success

```
def test_api_call_fail():
    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/10'

    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 403
    assert response.status_code == 404
```

User Story :- Enhancing recommendations through feedback
API being tested : http://127.0.0.1:5000/api/CourseFeedback/10000000000000000000
Inputs :

Request Mode :- **GET**
User Id =10000000000000000000

Expected output:

HTTP Status Code = 403 FORBIDDEN
Return Json Result as
{
"message": "Internal Server Error"
}

Actual Output :

HTTP Status Code = 403 FORBIDDEN
Return Json Result as
{
"message": "Internal Server Error"
}

Result- Success

```
def test_api_call_fail_1():
    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/10000000000000000000'

    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 500
    assert response.status_code == 500
```

User Story :- Enhancing recommendations through feedback
API being tested : http://127.0.0.1:5000/api/CourseFeedback/
Inputs :

Request Mode :- **POST**
User Id =1
Json Body
{
"subject_id": 4,
"rating": 5,
"remarks": "Very good"
}

Expected output:

HTTP Status Code = 201 CREATED
{
"id": 15,
"user_id": 2,
"subject_id": "4",
"rating": 5,
"remarks": "Very good"
}

Final Project Report

Actual Output :

HTTP Status Code = 201 CREATED

{
"id": 15,
"user_id": 2,
"subject_id": "4",
"rating": 5,
"remarks": "Very good"
}

Result- Success

```
def test_post_request():

    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/1'

    # Define the data you want to send in the POST request
    data = {
        "subject_id": '4',
        "rating": '5',
        "remarks": "Very good"
    }

    # Make the POST request. PLEASE use JSON so that data will pass as Json
    response = requests.post(api_url, json=data)

    # Assert the status code is 201 or any other expected status code
    assert response.status_code == 201
```

User Story :- Enhancing recommendations through feedback

API being tested : http://127.0.0.1:5000/api/CourseFeedback/1

Inputs :

Request Mode :- **POST**

User Id =1

Json Body

{
"subject_id": ,
"rating": 5,
"remarks": "Very good"
}

Final Project Report

Expected output:

HTTP Status Code = 403 FORBIDDEN
Return Json Result as
{
"message": "Internal Server Error"
}

Actual Output :

Error throws in pytest

Result- Fail

```
def test_post_request_fail():

    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/1'

    # Define the data you want to send in the POST request
    data = {
        "subject_id": ,
        "rating": 4,
        "remarks": "Very good"
    }

    # Make the POST request. PLEASE use JSON so that data will pass as Json
    response = requests.post(api_url, json=data)

    # Assert the status code is 201 or any other expected status code
    assert response.status_code == 201
```

User Story :- Enhancing recommendations through feedback

API being tested : http://127.0.0.1:5000/api/CourseFeedback/1/1

Inputs :

Request Mode :- **PUT**
User Id =1
Subject ID =1
Json Body
{
"rating": 4,
"remarks": "Very good update"
}

Final Project Report

Expected output:

HTTP Status Code = 201 CREATED

{
"id": 1,
"user_id": 1,
"subject_id": "2",
"rating": 4,
"remarks": "Very good update"
}

Actual Output :

HTTP Status Code = 201 CREATED

{
"id": 1,
"user_id": 1,
"subject_id": "2",
"rating": 4,
"remarks": "Very good update"
}

Result- Success

```
def test_put_request():

    api_url = 'http://127.0.0.1:5000/api/CourseFeedback/1/2'

    # Define the data you want to send in the POST request
    data = {
        "rating": 5,
        "remarks": "Very good"
    }

    # Make the POST request. PLEASE use JSON so that data will pass as Json
    response = requests.put(api_url, json=data)

    # Assert the status code is 201 or any other expected status code
    assert response.status_code == 201
```

Final Project Report

User Story :- Feedback on the learning paths

API being tested : http://127.0.0.1:5000/api/StudentAcademicInfo/1/F3-2024

Inputs :

Request Mode :- **GET**

User Id =1

Subject ID =1

Json Body

{
"rating": 4,
"remarks": "Very good update"
}

Expected output:

HTTP Status Code = 200 OK

{
"roll_number": "1",
"term_id": "F3-2024",
"course_id": "100",
"avg_ga": 90.0,
"quiz1": 10.0,
"quiz2": 10.0,
"end_term": 10.0,
"oppe1": 10.0,
"oppe2": 10.0,
"project": 10.0,
"final_score": 10.0
}

Actual Output :

HTTP Status Code = 200 OK

{
"roll_number": "1",
"term_id": "F3-2024",
"course_id": "100",
"avg_ga": 90.0,
"quiz1": 10.0,
"quiz2": 10.0,
"end_term": 10.0,
"oppe1": 10.0,
"oppe2": 10.0,
"project": 10.0,
"final_score": 10.0

}

Result- Success

```
def test_StudentAcademic_api_call_success():
    api_url = 'http://127.0.0.1:5000/api/StudentAcademicInfo/1/F3-2024'

    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 200
    assert response.status_code == 200
```

User Story :- Feedback on the learning paths

API being tested : http://127.0.0.1:5000/api/StudentProfile

Inputs :

Request Mode :- **POST**

{
"roll_number":2,
"academic_interests": "Higher study",
"learning_goals" : "DP",
"schedules" : "3 month",
"commitments" :"Higher study+1"
}

Expected output:

HTTP Status Code = 201 CREATED

{
"roll_number": "2",
"academic_interests": "Higher study",
"learning_goals": "DP",
"schedules": "3 month",
"commitments": "Higher study+1"
}

Actual Output :

HTTP Status Code = 201 CREATED

{
"roll_number": "2",
"academic_interests": "Higher study",
"learning_goals": "DP",
"schedules": "3 month",

"commitments": "Higher study+1"
}

Result- Success

```
def test_Studentprofile_post_call_success():
    api_url = 'http://127.0.0.1:5000/api/StudentProfile'

    # Define the data you want to send in the POST request
    data = {
    "roll_number":4,
    "academic_interests": "Higher study",
    "learning_goals" : "DP",
    "schedules" :    "3 month",
    "commitments" :"Higher study+1"
}

    # Make the POST request. PLEASE use JSON so that data will pass as Json
    response = requests.post(api_url, json=data)

    # Assert the status code is 201 or any other expected status code
    assert response.status_code == 201
```

User Story :- Feedback on the learning paths
API being tested : http://127.0.0.1:5000/api/StudentProfile/2
Inputs :
Request Mode :- **GET**
RollNumberParam =2
Expected output:
HTTP Status Code = 200 OK
{
"roll_number": "2",
"academic_interests": "Higher study",
"learning_goals": "DP",
"schedules": "3 month",
"commitments": "Higher study+1"
}

Final Project Report

Actual Output :

HTTP Status Code = 200 OK

{
"roll_number": "2",
"academic_interests": "Higher study",
"learning_goals": "DP",
"schedules": "3 month",
"commitments": "Higher study+1"
}

Result- Success

```
def test_Studentprofile_get_call_success():
    api_url = 'http://127.0.0.1:5000/api/StudentProfile/2'


    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 200
    assert response.status_code == 200
```

User Story :- Feedback on the learning paths

API being tested : http://127.0.0.1:5000/StudentProfile/**2999999999999**

Inputs :

Request Mode :- **GET**

RollNumberParam =2999999999999

Expected output:

HTTP Status Code = 404 NOT FOUND

Return Json Result as

{
"message": "No student profile found"
}

Actual Output :

HTTP Status Code = 404 NOT FOUND

Return Json Result as

{
"message": "No student profile found"
}

Result- Success

Final Project Report

```python
def test_Studentprofile_get_call_fail():
    api_url = 'http://127.0.0.1:5000/api/StudentProfile/2999999999999'


    # Make the API call
    response = requests.get(api_url)

    # Assert the status code is 200
    assert response.status_code == 404
```

User Story :- Upload Student Academic Info
API Used: http://127.0.0.1:5000/api/add_student_acamdemic_info
Inputs :

Request Mode :- **POST**

{
"roll_number": "S013",
"term_id": 1,
"course_id": "CS101",
"avg_ga": 85.5,
"quiz1": 75.0,
"quiz2": 88.5,
"end_term": 92.0,
"oppe1": 90.0,
"oppe2": 87.5,
"project": 95.0,
"final_score": 89.7
}

Expected output:

HTTP Status Code = 201 CREATED

{
"roll_number": "S013",
"term_id": 1,
"course_id": "CS101",
"avg_ga": 85.5,
"quiz1": 75.0,
"quiz2": 88.5,
"end_term": 92.0,
"oppe1": 90.0,
"oppe2": 87.5,
"project": 95.0,
"final_score": 89.7

}

Actual Output :

HTTP Status Code = 201 CREATED

{
"roll_number": "S013",
"term_id": 1,
"course_id": "CS101",
"avg_ga": 85.5,
"quiz1": 75.0,
"quiz2": 88.5,
"end_term": 92.0,
"oppe1": 90.0,
"oppe2": 87.5,
"project": 95.0,
"final_score": 89.7
}

Result- Success

```
def test_add_student_academic_info_api_success():
    api_url = 'http://127.0.0.1:5000/api/add_student_acamdemic_info'

    data = {
  "roll_number": "S040",
  "term_id": 1,
  "course_id": "CS101",
  "avg_ga": 85.5,
  "quiz1": 75.0,
  "quiz2": 88.5,
  "end_term": 92.0,
  "oppe1": 90.0,
  "oppe2": 87.5,
  "project": 95.0,
  "final_score": 89.7
}
    response = requests.post(api_url, json=data)
    assert response.status_code == 201
```

Final Project Report

User Story :- Upload Student GPA Details
API Used: http://127.0.0.1:5000/api/add_gpa
Inputs :

Request Mode :- **POST**

```
{
   "roll_number": "S037",
   "term_id": 3,
   "sgpa": 7.8,
   "cgpa": 7.6
}
```

Expected output:

HTTP Status Code = 201 CREATED

```
{
   "roll_number": "S037",
   "term_id": 3,
   "sgpa": 7.8,
   "cgpa": 7.6
}
```

Actual Output :

HTTP Status Code = 201 CREATED

```
{
   "roll_number": "S037",
   "term_id": 3,
   "sgpa": 7.8,
   "cgpa": 7.6
}
```

Result- Success

Final Project Report

```python
def test_add_gpa_api_success():
    api_url = 'http://127.0.0.1:5000/api/add_gpa'

    data = {
    "roll_number": "S040",
    "term_id": 3,
    "sgpa": 7.8,
    "cgpa": 7.6
}
    response = requests.post(api_url, json=data)
    assert response.status_code == 201
```

User Story :- Upload Student Offline Education Details
API Used: http://127.0.0.1:5000/api/add_offline_education
Inputs :

Request Mode :- **POST**

```
{
  "degree": "Bachelor of Science",
  "major": "Computer Science",
  "end_year": "2022"
}
```

Expected output:

HTTP Status Code = 201 CREATED

```
{
  "degree": "Bachelor of Science",
  "major": "Computer Science",
  "end_year": "2022"
}
```

Actual Output :

HTTP Status Code = 201 CREATED

```
{
  "degree": "Bachelor of Science",
  "major": "Computer Science",
  "end_year": "2022"
}
```

Result- Success

Final Project Report

```
def test_add_offline_education_api_success():
    api_url = 'http://127.0.0.1:5000/api/add_offline_education'

    data = {
  "roll_number": "S040",
  "degree": "Bachelor of Science",
  "major": "Computer Science",
  "end_year": "2022"
}
    response = requests.post(api_url, json=data)
    assert response.status_code == 201
```

User Story :- Analytics
API Used: http://127.0.0.1:5000/api/admin
Inputs :

Request Mode :- **GET**

Email:- admin@admin.com

Expected output:

HTTP Status Code = 200


```
{
  "tstudents": [
    {"id": 1, "name": "Student1", "roll_number": "21f1000123", "email": "student1@example.com"},
    {"id": 2, "name": "Student2", "roll_number": "21f1000124", "email": "student2@example.com"},
    {"id": 3, "name": "Student3", "roll_number": "21f1000125", "email": "student3@example.com"}
    // ... (data for all students)
  ],
  "tcourses": 16,  // Total number of courses
  "counts": {
    "1": 10,  // Number of learning paths with a rating of 1
    "2": 15,  // Number of learning paths with a rating of 2
    "3": 20,  // Number of learning paths with a rating of 3
    "4": 25,  // Number of learning paths with a rating of 4
    "5": 30   // Number of learning paths with a rating of 5
  },
  "StudentsDataSummaryReport": {},
  "StudentsDataSummaryReportImage": "static/IMG/StudentSatisfactionMetric.png"
}
```

Final Project Report

Actual Output :

HTTP Status Code = 200
HTTP Status Code = 200
{
"tstudents": [
{"id": 1, "name": "Student1", "roll_number": "21f1000123", "email": "student1@example.com"},
{"id": 2, "name": "Student2", "roll_number": "21f1000124", "email": "student2@example.com"},
{"id": 3, "name": "Student3", "roll_number": "21f1000125", "email": "student3@example.com"}
// ... (data for all students)
],
"tcourses": 16, // Total number of courses
"counts": {
"1": 10, // Number of learning paths with a rating of 1
"2": 15, // Number of learning paths with a rating of 2
"3": 20, // Number of learning paths with a rating of 3
"4": 25, // Number of learning paths with a rating of 4
"5": 30 // Number of learning paths with a rating of 5
},
"StudentsDataSummaryReport": {},
"StudentsDataSummaryReportImage": "static/IMG/StudentSatisfactionMetric.png"
}

Result- Success

```
def test_admin_data():
    # Replace {email} with the actual admin email
    api_url = 'http://127.0.0.1:5000/api/admin'
    email = 'admin@example.com'
    # Make the GET request
    response = requests.get(f'{api_url}/{email}')
    # Assert the status code is 200 (Success)
    assert response.status_code == 200
```

Final Project Report

Implementation Details of the Project

Technologies and tools used

Technologies for the Backend

- Flask== 3.0.0
- Flask-Cors==3.0.10
- Flask-Login==0.6.3
- Flask-Principal==0.4.0
- Flask-RESTful==0.3.9 (For creating API endpoints)
- Flask-SQLAlchemy==3.0.3
- Jinja2==3.1.2
- MarkupSafe==2.1.3
- requests==2.28.1
- SQLAlchemy==2.0.23 (SQL toolkit and Object Relational Mapper)
- typing_extensions==4.8.0
- Werkzeug==3.0.1
- Flask-Security-Too == 5.3.2 (Secure Login-Logout Interface)
- matplotlib ==3.8.2 (Path Generation)
- bcrypt==4.0.1
- blinker==1.7.0
- bleach == 6.1.0
- Sqlite3 (For database)

Final Project Report

Technologies for the Frontend

- Vue 3 CLI
- JavaScript
- Vue Router
- Bootstrap v5
- Font awesome

Technologies for the Recommender system

- Streamlit
- Python

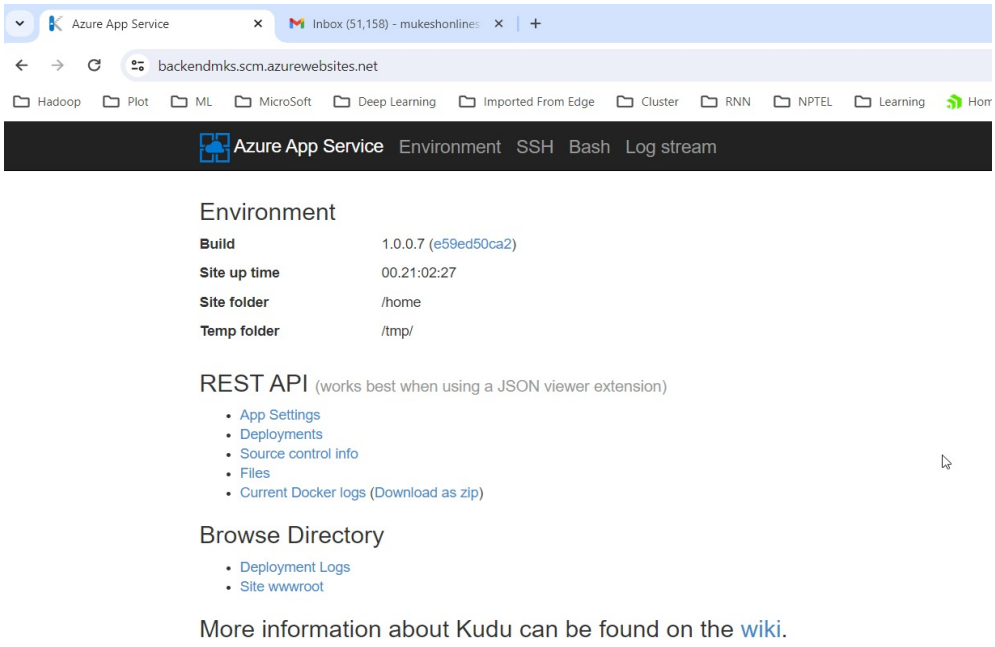
General technologies used

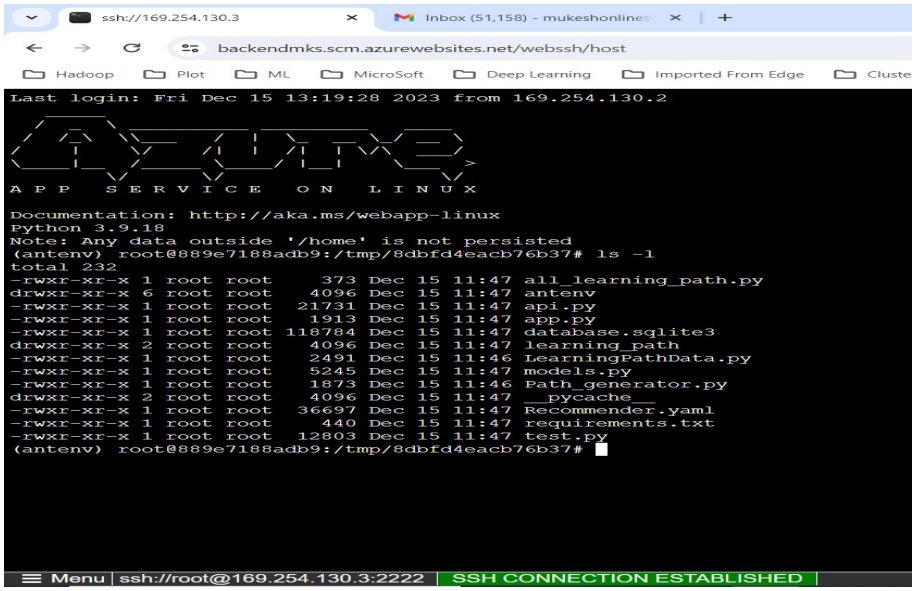
- GitHub (for versioning, code management, tracking, reviewing, issues, etc.).
- Jira (for project management).
- Google Calendar (For Scheduling Meetings)
- Whatsapp Group (For internal communication)
- IDE :- VS code
- Lucid (For UML diagrams and Wireframe)
- Animaker (For making animated video)
- Canva and Slides (PPT making)
- Excel sheets (MOM, Data management)
- Chatgpt (Getting idea for synthetic data generation)

Final Project Report

Hosting:

- API Hosting on - Azure App service
- Recommendation system - Huggingface spaces.





Final Project Report

Instructions to run the application

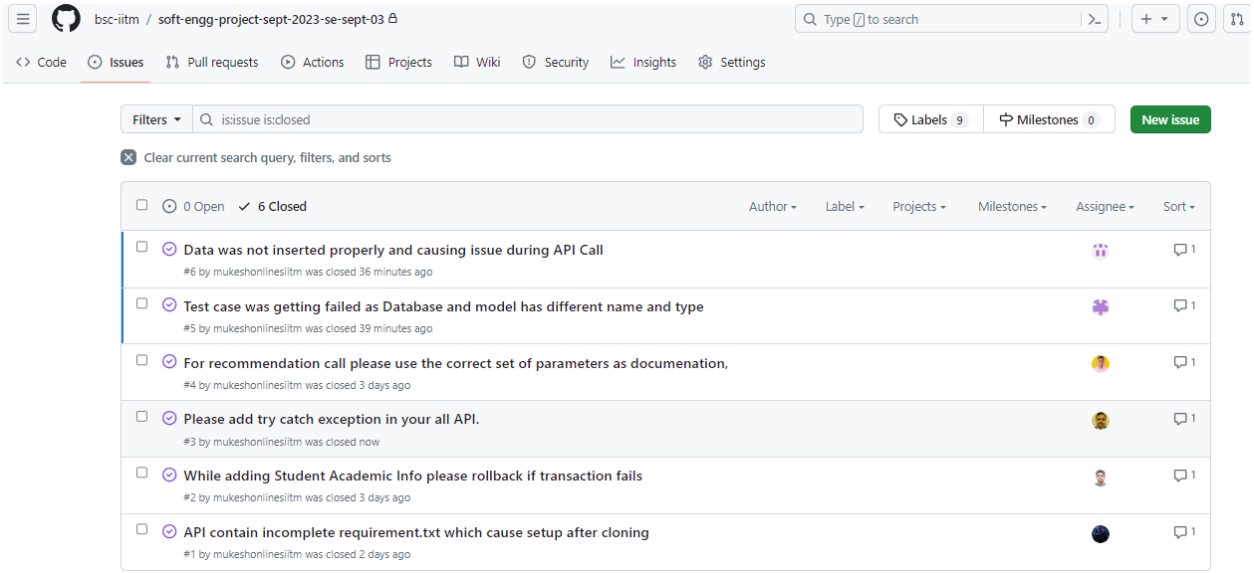
On Ubuntu/Linux/MAC/Windows OS:

- Git clone the repository.
- Change the directory to the “backend” directory inside the “Milestone-6-Final-Submission” directory using the command :
  cd .\Milestone-6-Final-Submission\Code\backend
- Activate the virtual environment using the command :
  source env/bin/activate
- Install the requirements using the command :
  pip3 install -r requirements.txt
- Run the app using command :
  python3 app.py
- Open new terminal change the directory to the “frontend” directory inside the “Milestone-6-Final-Submission” directory using the command :
  cd .\Milestone-6-Final-Submission\Code\frontend
- First make sure that node and npm are installed in your system.
- Install Vue-cli using command :
  npm install -g @vue/cli
- Install font-awesome using command :
  npm install --save @fontawesome/fontawesome-free
- Now run the vue application using the command:
  npm run serve
- Now navigate to this link http://localhost:8080/
- And use the application as shown in the **Demo video**

Final Project Report

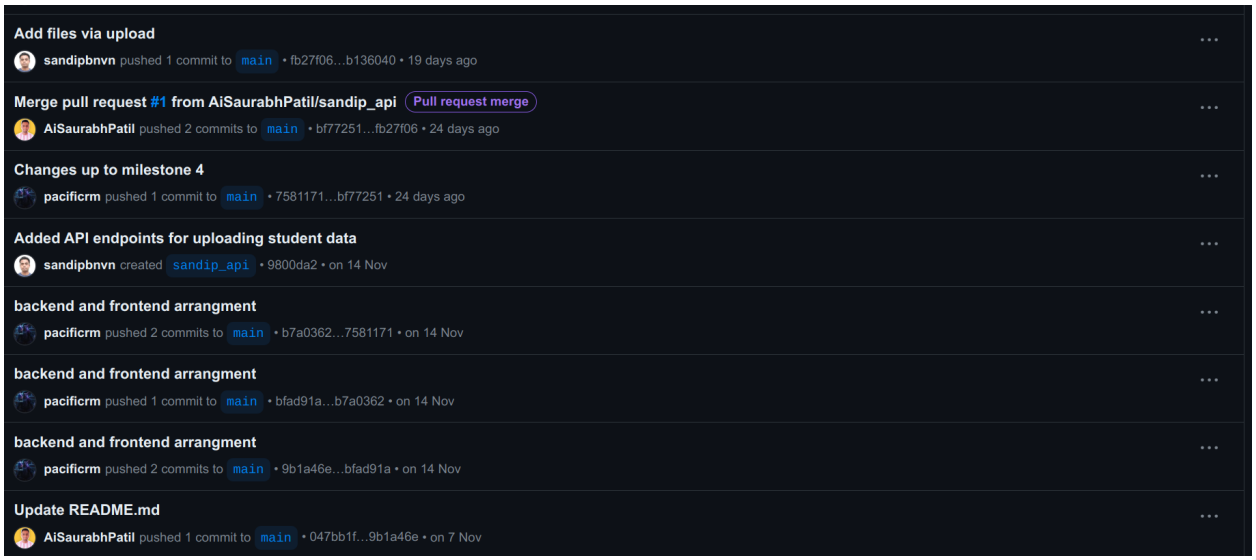
Code review issue reporting and tracking

Issue tracker



Final Project Report

Github Activities / Pull requests



Final Project Report

Code review