SOFTWARE ENGINEERING PROJECT Term : Sept 2023 - Dec 2023

Milestone – 6 Final Project Report

Contributors:

JS Ranjith (21f1002171)
Mohd. Ikram (21f2000712)
Dhruv Gupta (21f1004011)
Bimlesh KR Kanth (21f1005524)
Rahul Chakraborty (21f1002786)

Detail final report of course recommender

From Milestone 1 to Milestone 6

Milestone 1 Submission:

Honor Code Submission

1. Bimlesk Kr Kanth

I, BIMLESH KR KANTH with roll no. 21f1005524, declare that I will not use any ideas, writings, code or work that is not my own or my group's with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: BIMLESH KR KANTH
Date: 12-10-2023

2. JS Ranjith

I, JS RANJITH with roll no. 21f1002171, declare that I will not use any ideas, writings, code or work that is not my own or my group's with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: JS RANJITH
Date: 12-10-2023

3. Rahul Chakraborty

I, RAHUL CHAKRABORTY with roll no. 21f1002786, declare that I will not use any ideas, writings, code or work that is not my own or my group's with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: RAHUL CHAKRABORTY
Date: 12-10-2023

4. Dhruv Gupta

I, Dhruv Gupta with roll no. 21f1004011, declare that I will not use any ideas, writings, code or work that is not my own or my group's with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: Dhruv Gupta
Date: 12-10-2023

5. Mohd Ikram

I, Mohd Ikram with roll no. 21f2000712, declare that I will not use any ideas, writings, code or work that is not my own or my group's with the intention of claiming it my or my group's work. For all the work that I will submit as part of this project, I will not share it outside my group with anybody directly or indirectly or upload it to any of the public forums on the internet.

I acknowledge that failing in any of the above constitutes plagiarism and in that case, the institute will take appropriate disciplinary action.

Sign: Mohd Ikram
Date: 12-10-2023

Application Users:

User Type	User Role Name
Primary Users	Students
Secondary Users	Admin
Tertiary Users	System

Primary User - Student - 9 User Stories

User Story 1

Feature: Login and Sign up
As a Student **I want to** register in the portal **So that** I can login into the portal
Acceptance Criteria: Student should be able to sign-up and login in the portal

User Story 2

Feature : Password Reset
As a Student **I want to** reset password in the portal **So that** I can set new password if I forget my existing password
Acceptance Criteria: Student should be able to reset a new password

User Story 3

Feature: Profile update

As a Student **I want to** update my profile in the portal **So that** I can keep my profile up-to-date
Acceptance Criteria: Student should be able to edit existing profile

User Story 4

Feature: Course List Display
As a Student **I want to** be able to see list of pending courses in the portal **So that** I can select courses for upcoming academic term
Acceptance Criteria: Student should be able to see a list of available courses which are pending in the course for him/her.

User Story 5

Feature: Display Course Feedbacks
As a Student **I want to** be able to see past feedbacks for a course in the portal **So that** I can make an informed decision about course selection.
Acceptance Criteria: Student should be able to see feedbacks against a particular course and its difficulty level.

User Story 6

Feature: Input Course Feedbacks
As a Student **I want to** be able to provide feedbacks for my completed courses in the portal **So that** I can make meaningful contributions in the course recommendation application.
Acceptance Criteria: Student should be able to write feedbacks against a particular course which he/she has completed successfully.

User Story 7

Feature: Edit Course Feedback
As a Student **I want to** be able to edit my feedbacks for a particular completed course in the portal **So that** I can update feedback .
Acceptance Criteria: Student should be able to write feedbacks against a particular course which he/she has completed successfully.

User Story 8

Feature: Update Academic Details
As a Student **I want to** be able to update enrolled course status and result obtained in the portal **So that** It will help others take informed decisions
Acceptance Criteria: Student should be able to see add/update enrolled courses and performance scores.

User Story 9

Feature: Course Recommendation
As a Student **I want to** be able to enter details regarding time dedication and other commitments in the portal **So that** I can view the course recommendation provided by the portal
Acceptance Criteria: Student should be able to enter weekly time duration dedication for studies and also whether full time student or having other commitments as well in the recommendation form input.

Secondary User - Admin - 4 User Stories

User Story 10

Feature: Login and Sign up

As an Admin **I want to** sign-up and login into the portal **So that** I can view and perform admin role
Acceptance Criteria: List of admins should be able to sign-up and login in the portal

User Story 11

Feature: Admin Dashboard
As an Admin **I want to** be able to view admin dashboard in the portal **So that** I am able to view metrics regarding the website.
Acceptance Criteria: Admin should be able to view metrics with charts

User Story 12

Feature: New Student List
As an Admin **I want to** able to view the new student registration and verify them **So that** the application has legit data of students only
Acceptance Criteria: Admin should be able to approve or decline the students

User Story 13

Feature: Course Adding/ modifiying

As an Admin **I want to** be able to create / modify course details in the portal **So that** I can keep the course details up to date
Acceptance Criteria: Admin should be able to add/update/delete courses in the portal

Tertiary User - System - 1 User Story

User Story 14

Feature: Course Recommendation
As a System **I want to** be able to use the past student and enrollment data **So that** I can make good recommendations for students
Acceptance Criteria: System have some equation / model based on data to generate recommendation

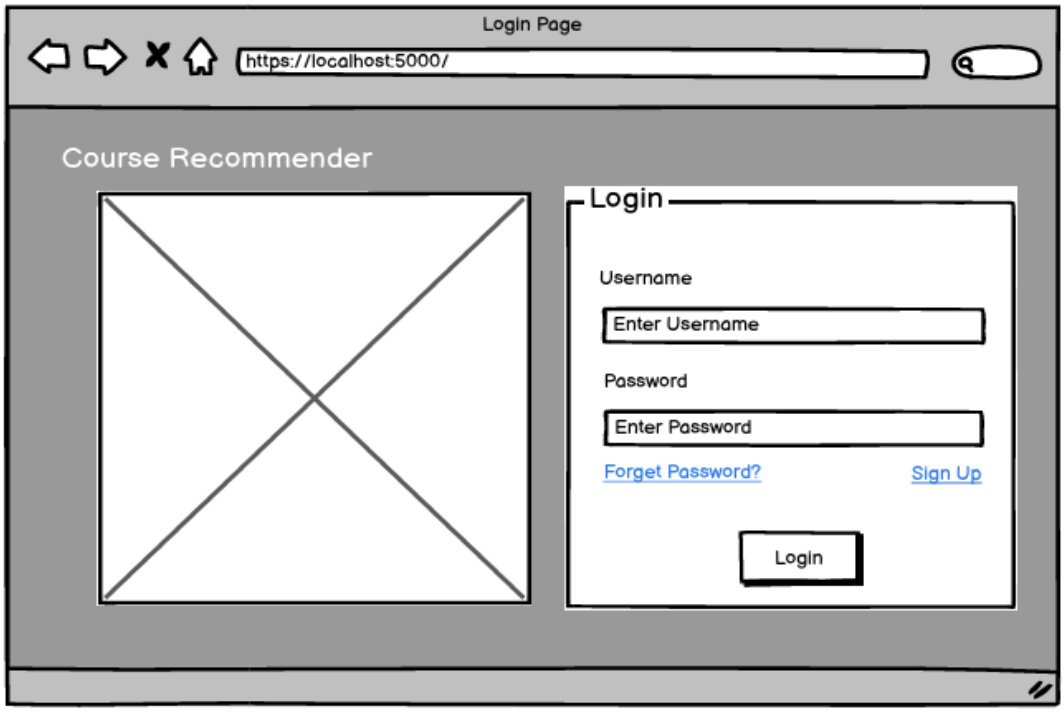
Milestone 2 Submission

Storyboard: Link of Pdf : Click

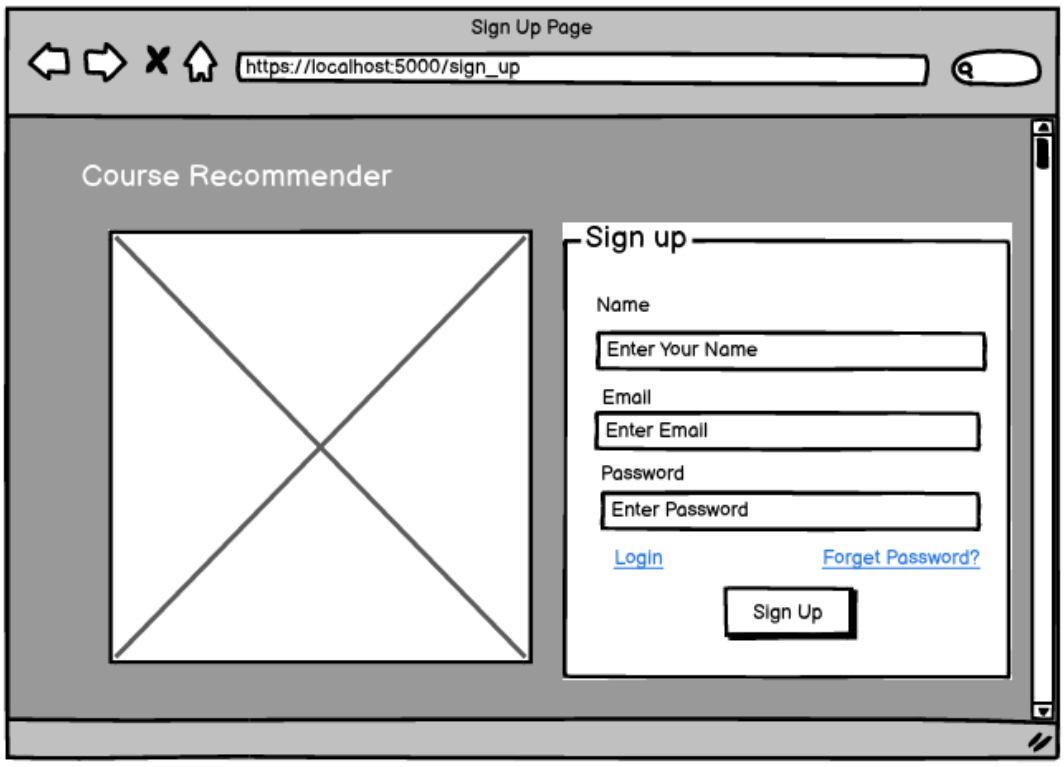
Storyboard animation video: For link : Click here

Wireframes:

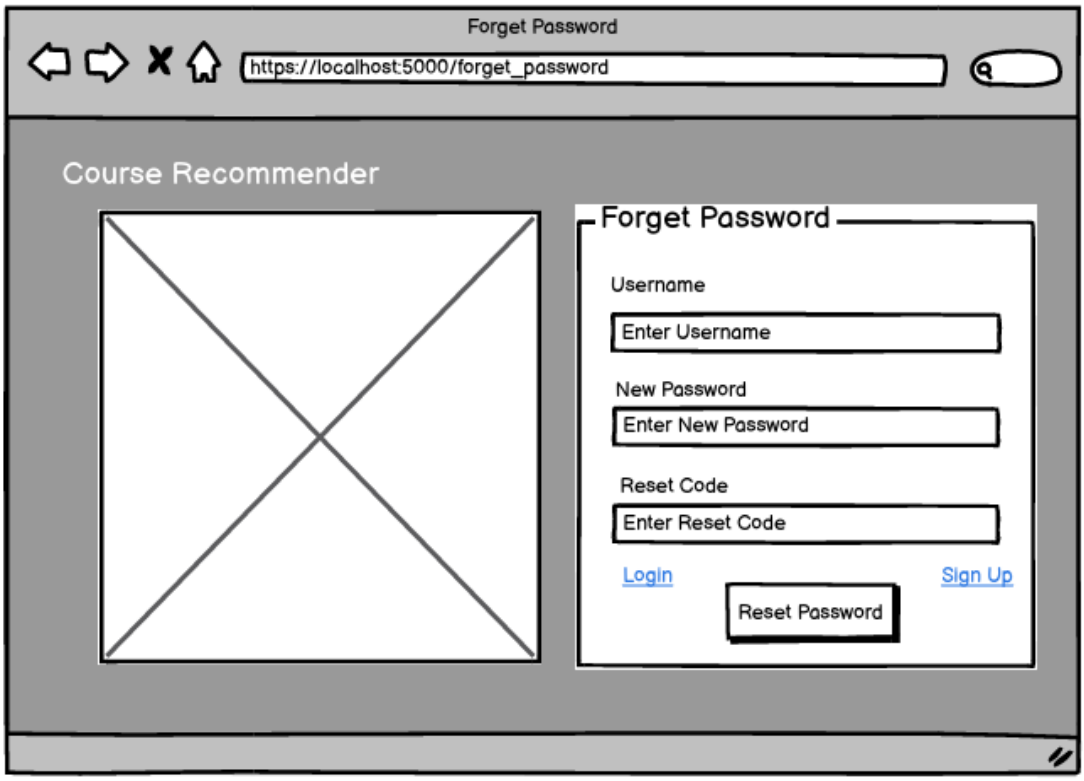
1. **Login Page:**



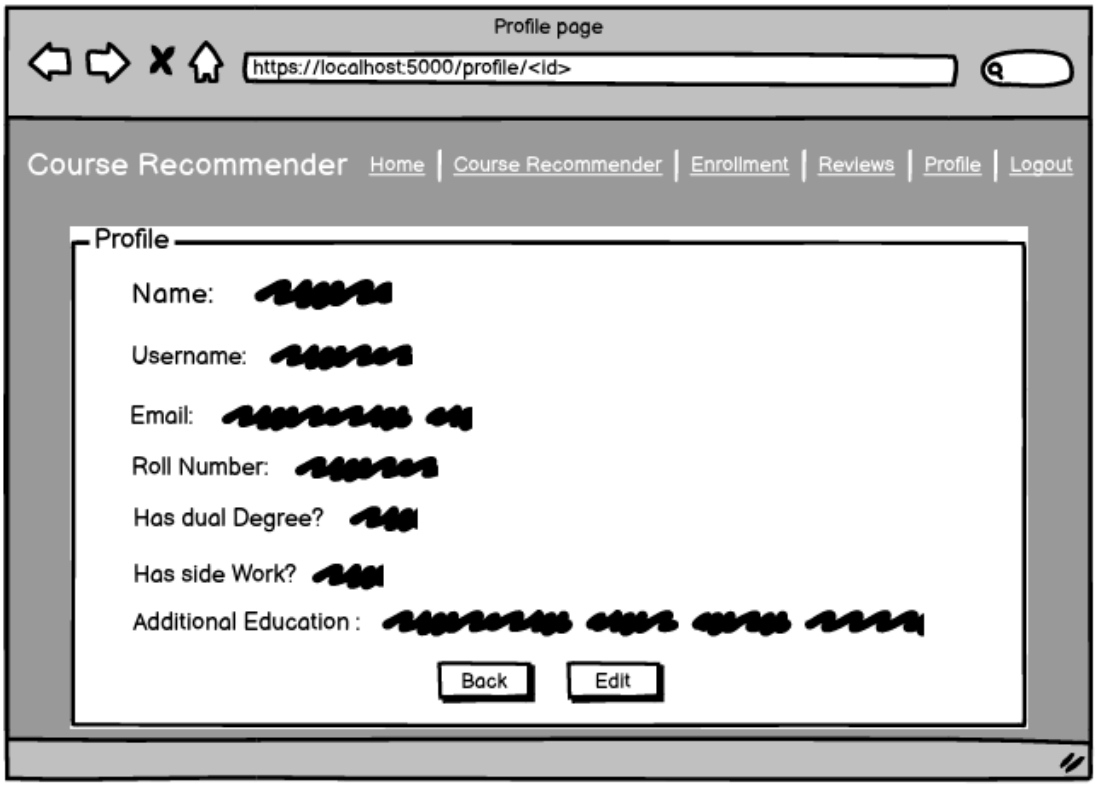
2. **Sign Up Page**



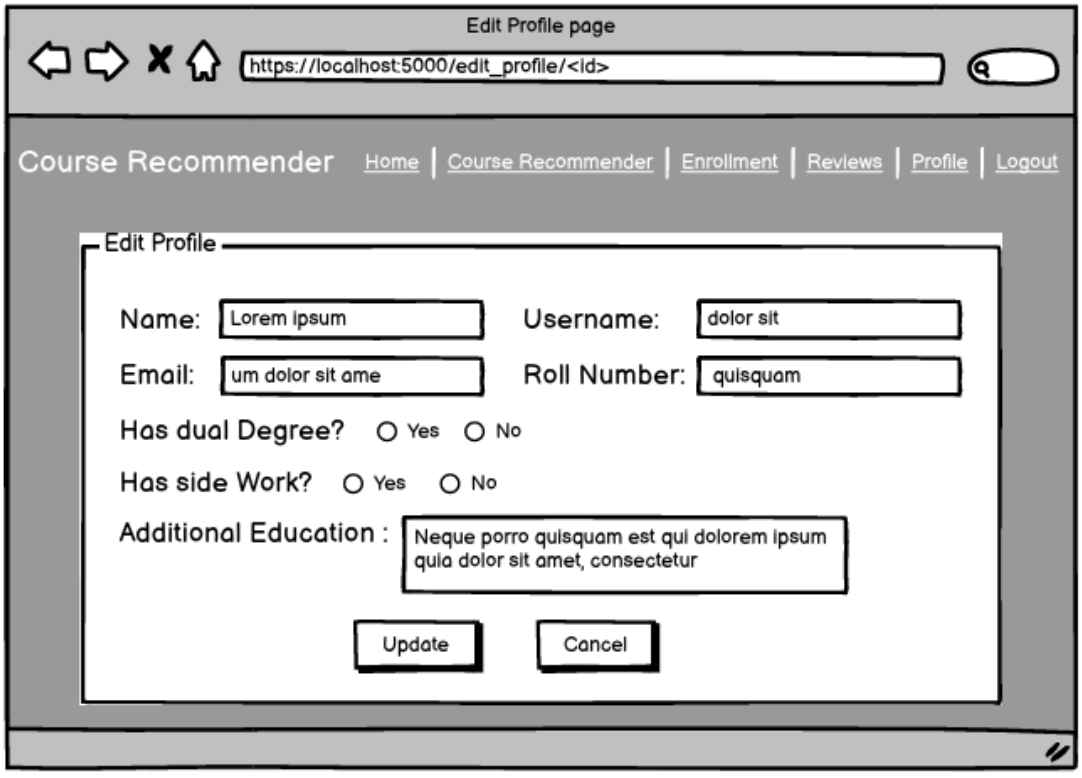
3. **Forget Password Page**



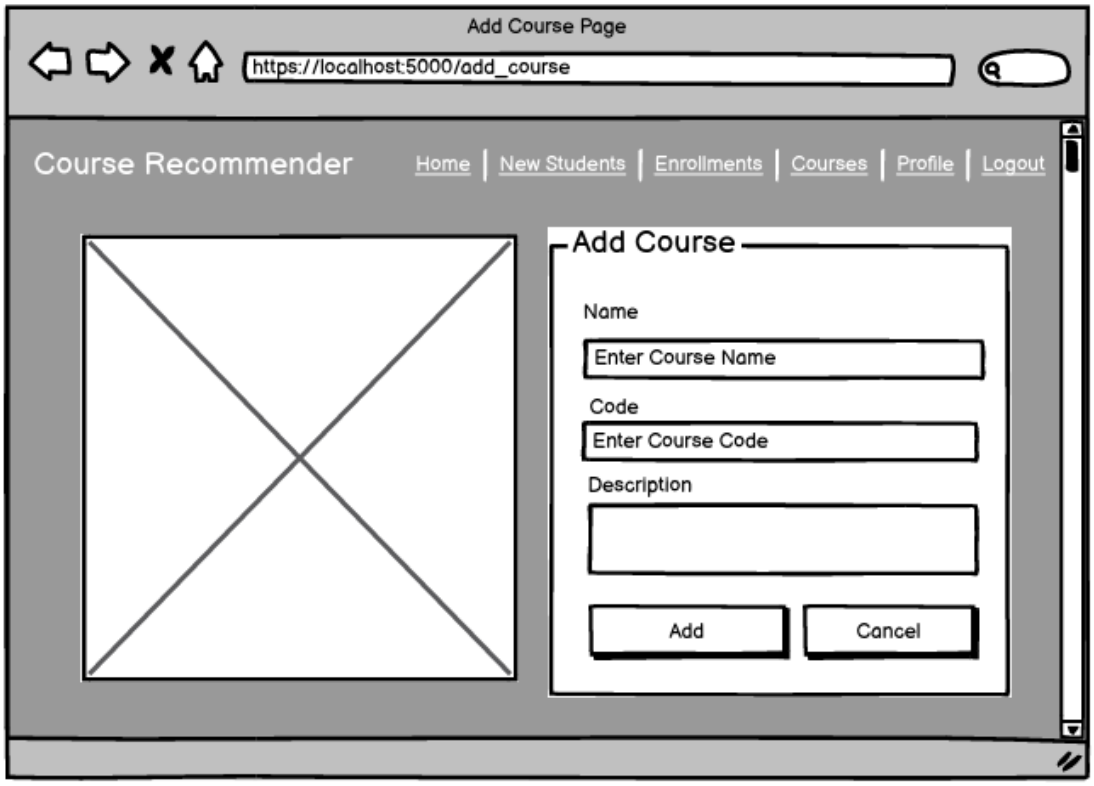
4. **Profile Page**



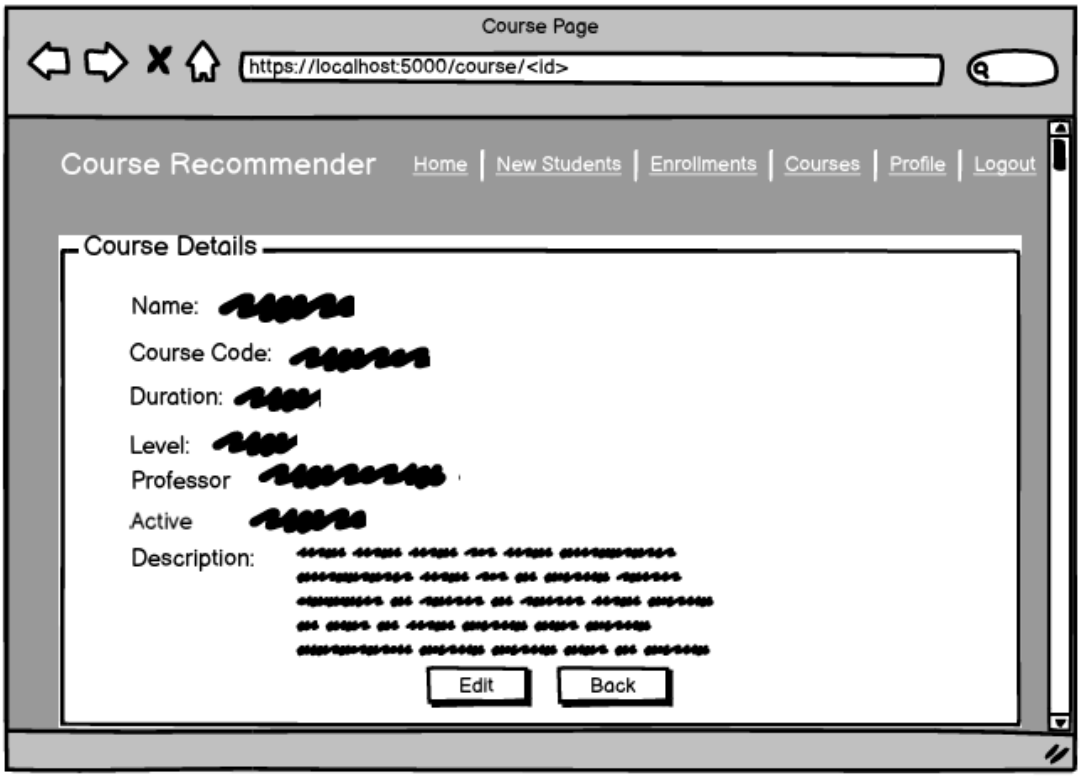
5. **Edit Profile Page**



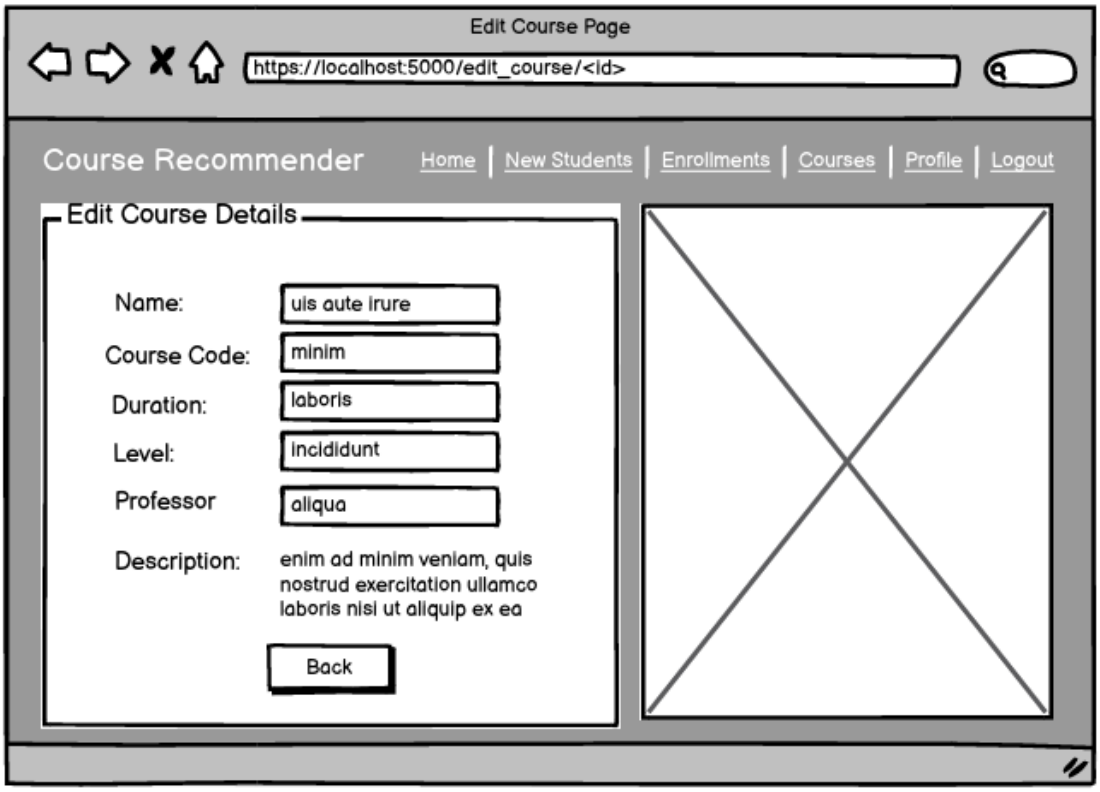
6. Add Course Page



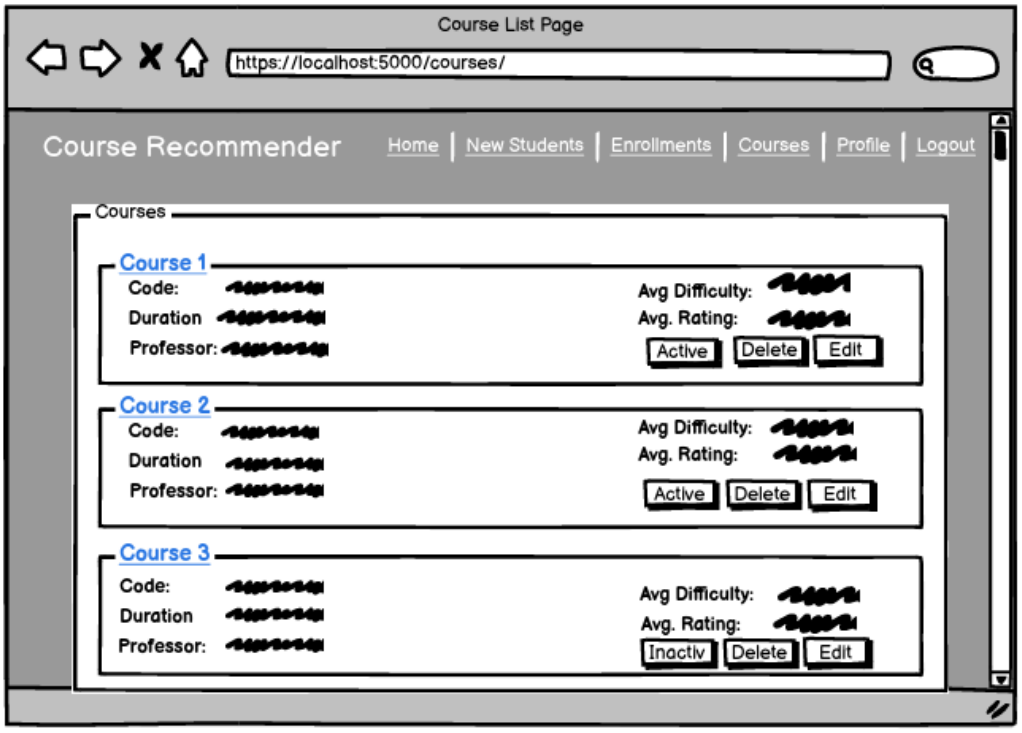
7. Course Details Page



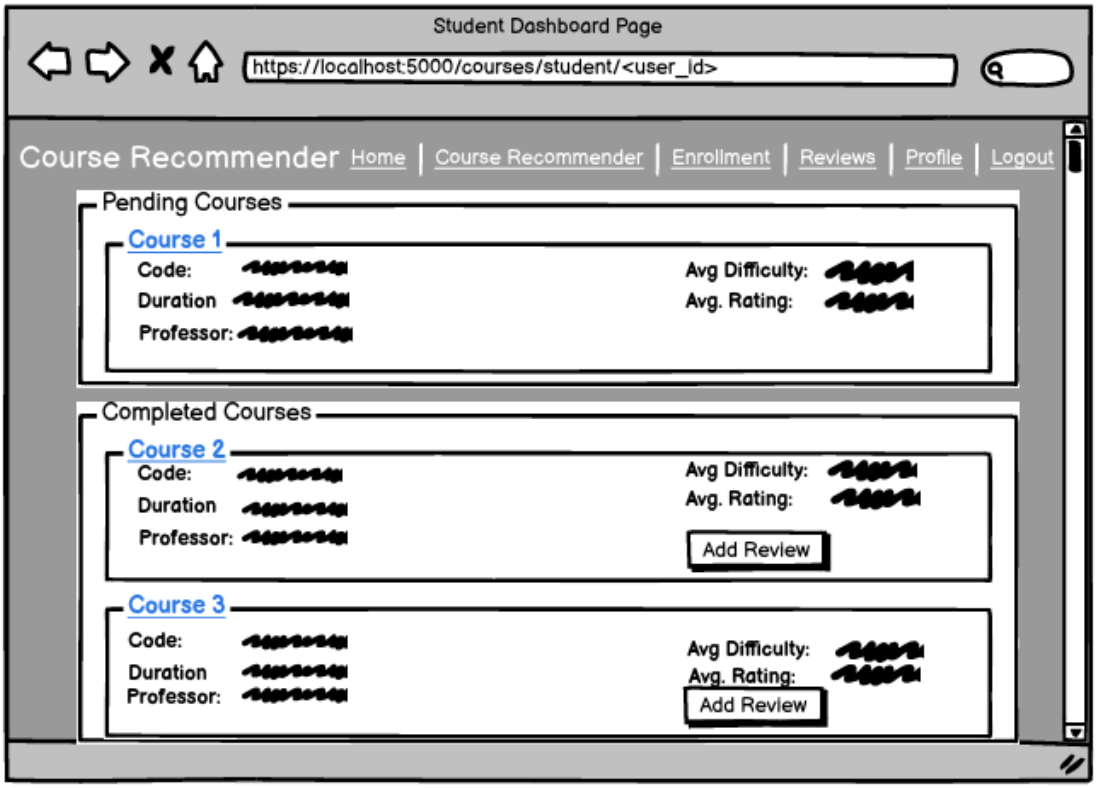
8. Edit Course Details Page



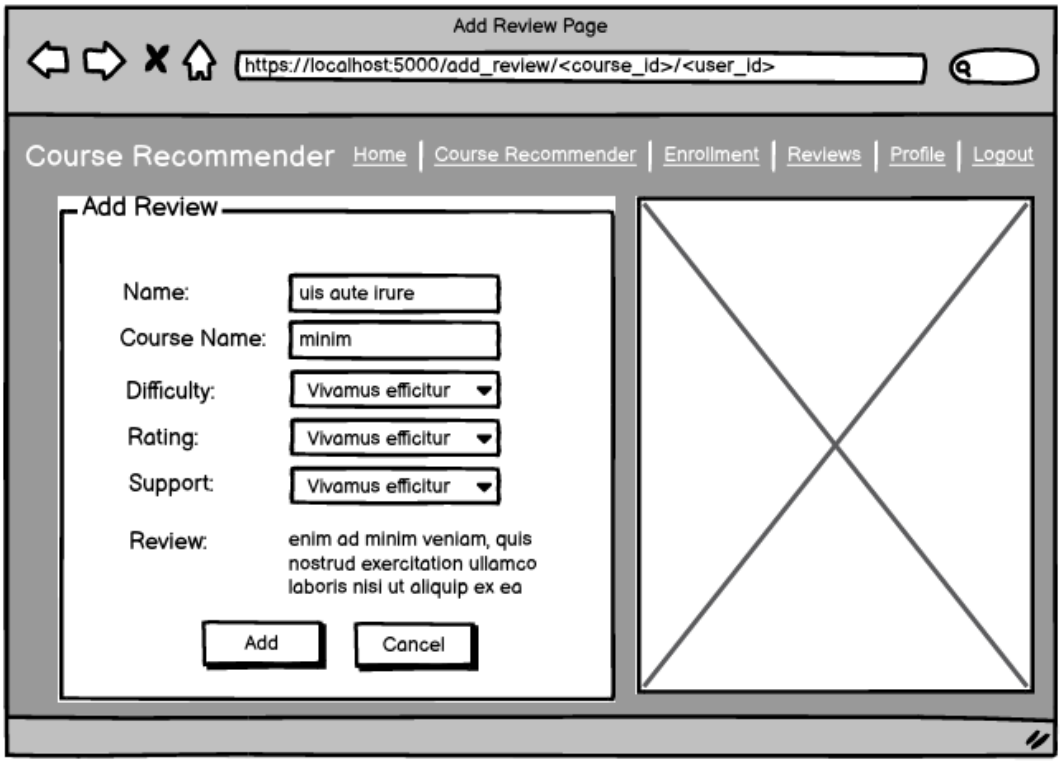
9. Course List Page



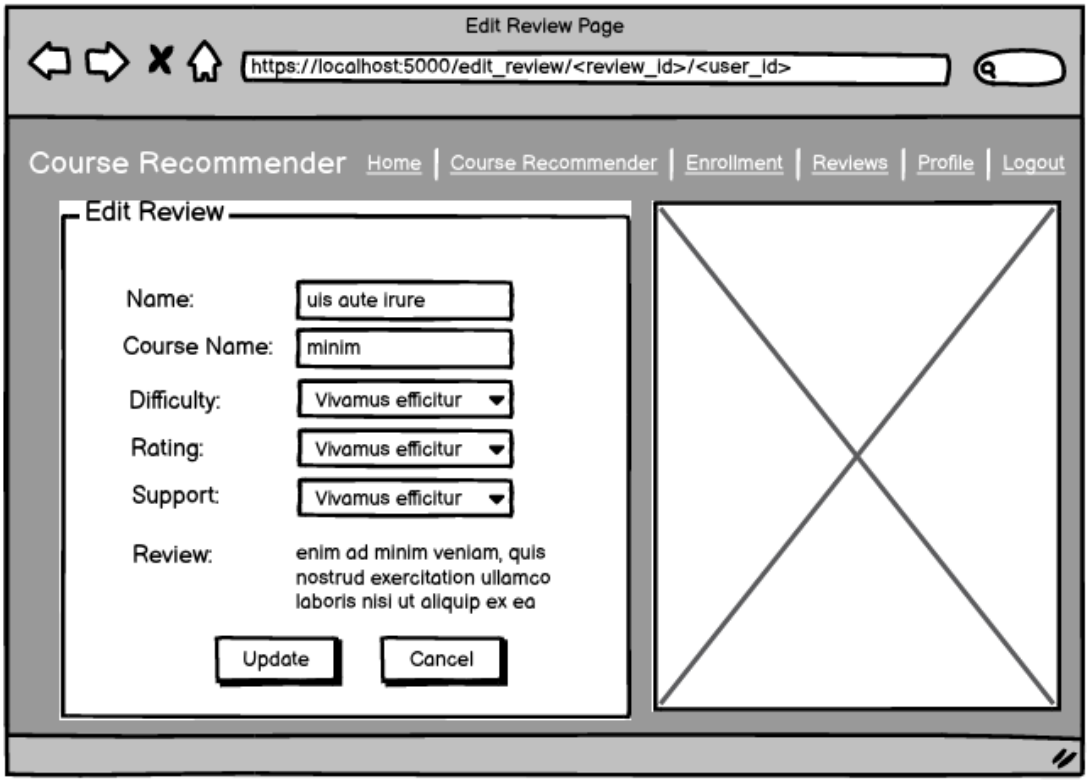
10. Student Dashboard



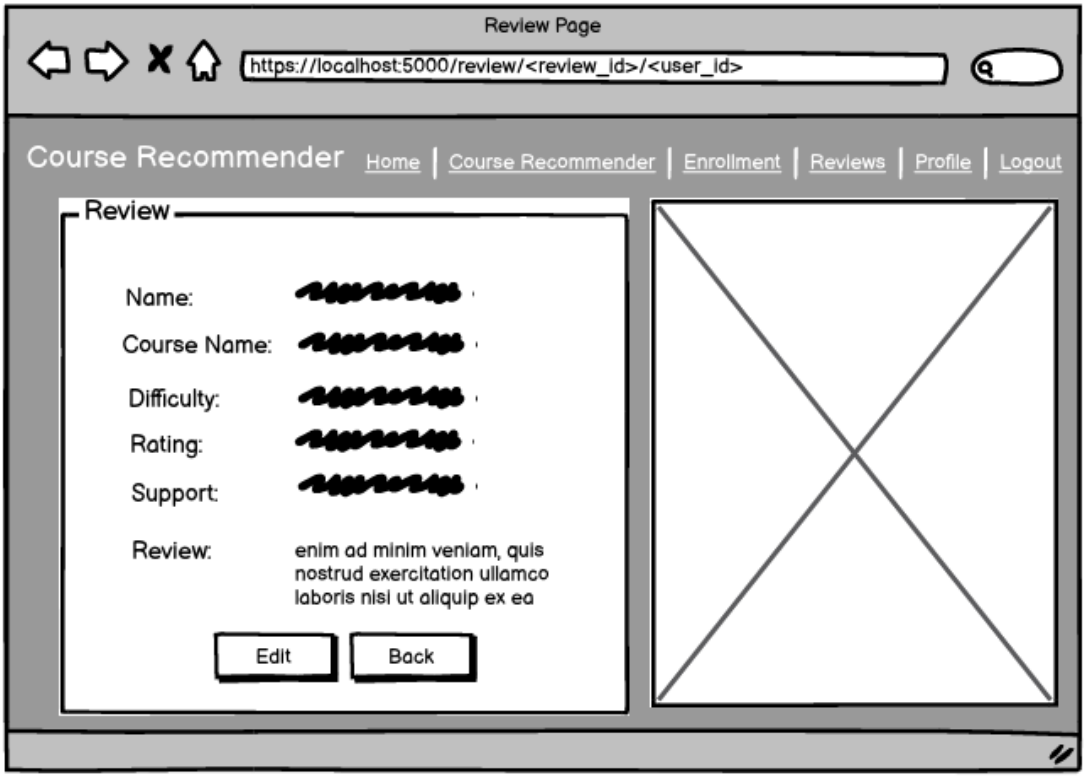
11. Add Review Page



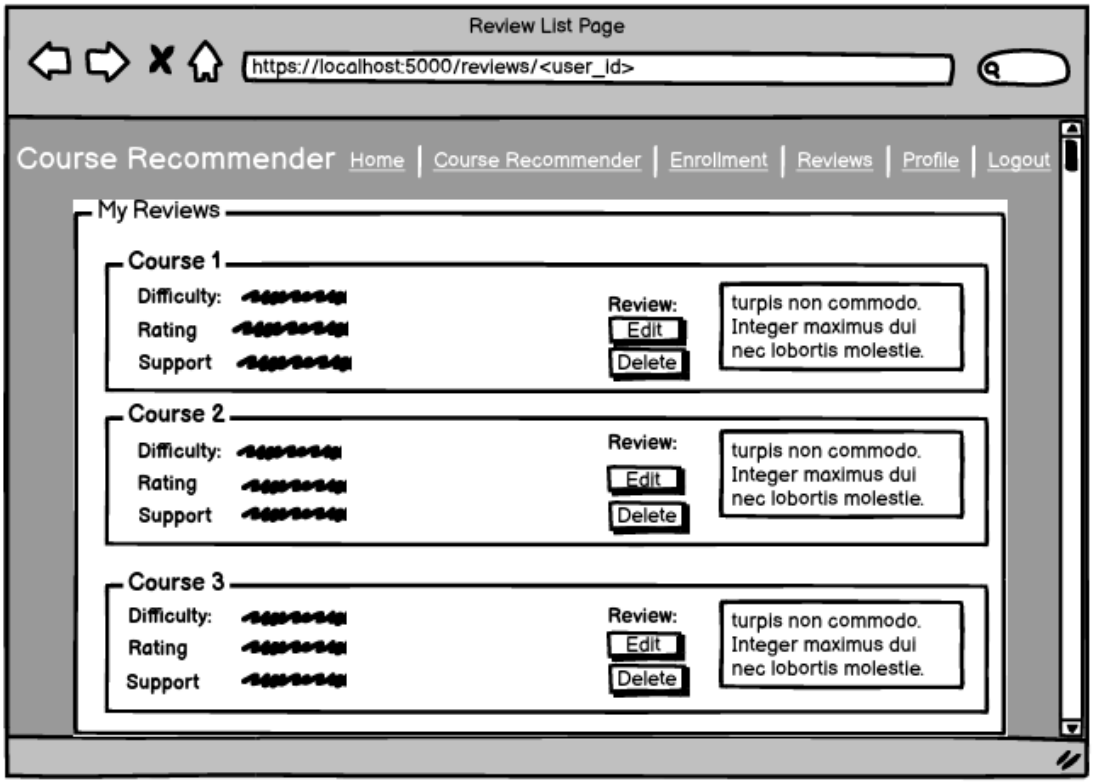
12. Edit Review Page



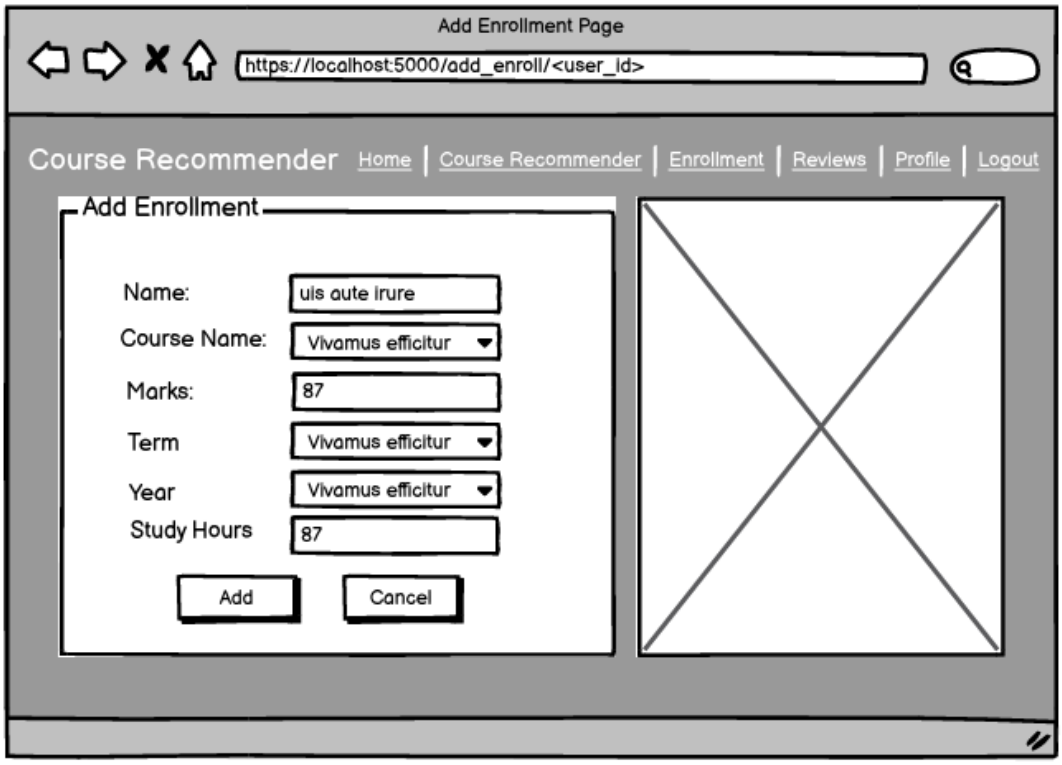
13. Review Details Page



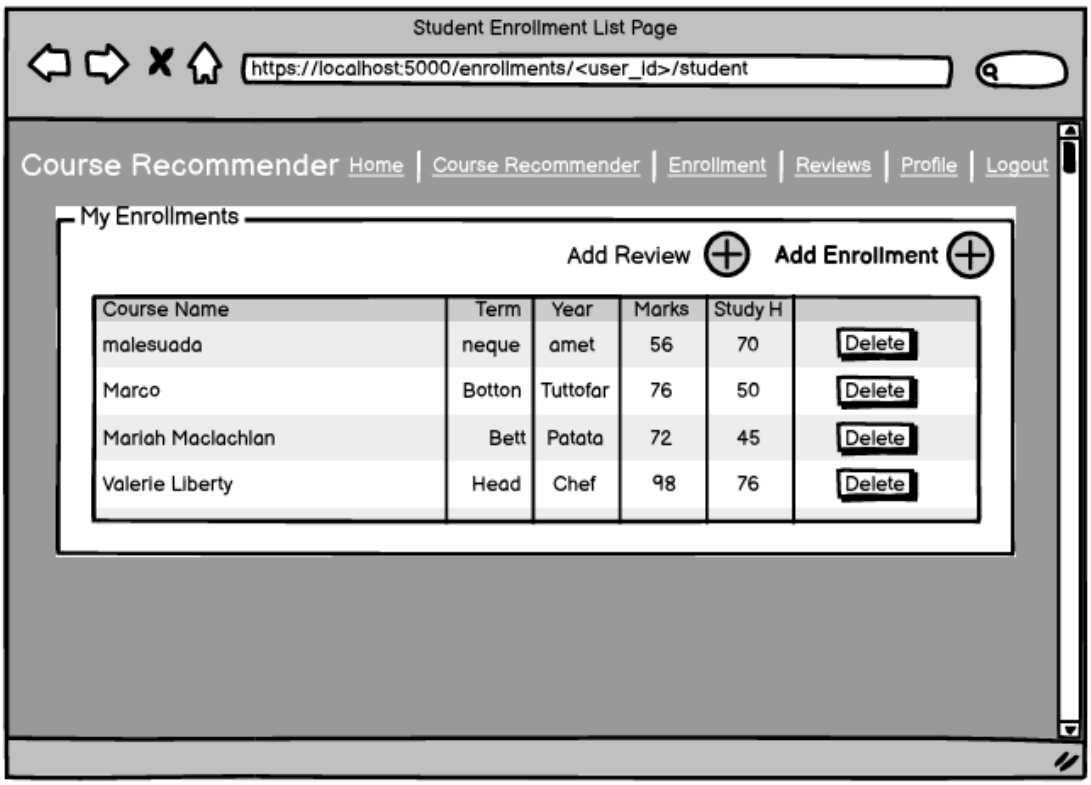
14. Review List Page



15. Add Enrollment Page

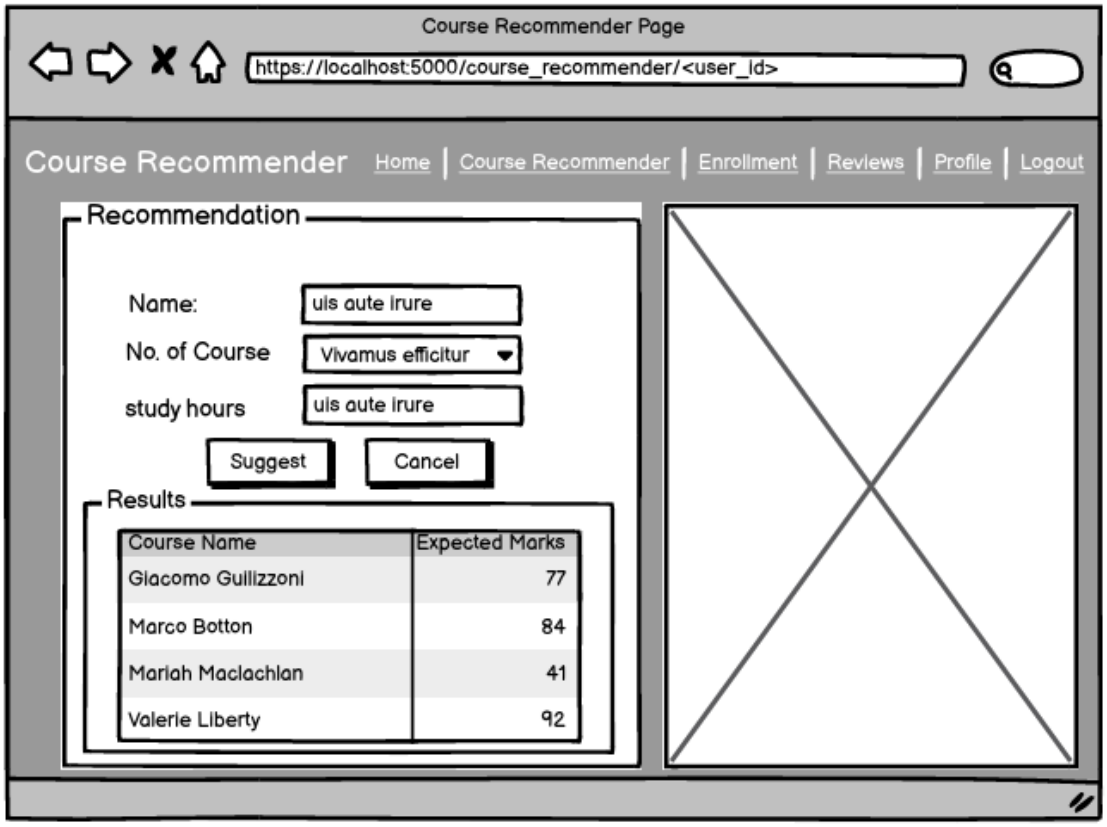


16. Student Enrollment List Page

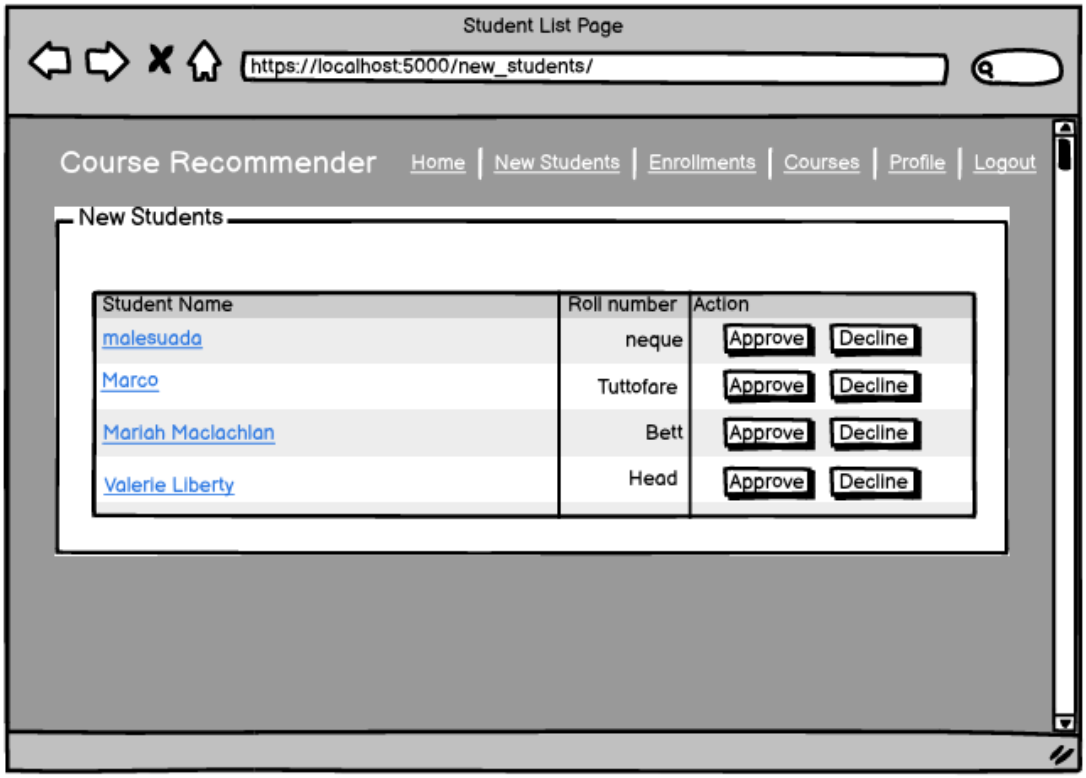


17. Admin Enrollment List Page

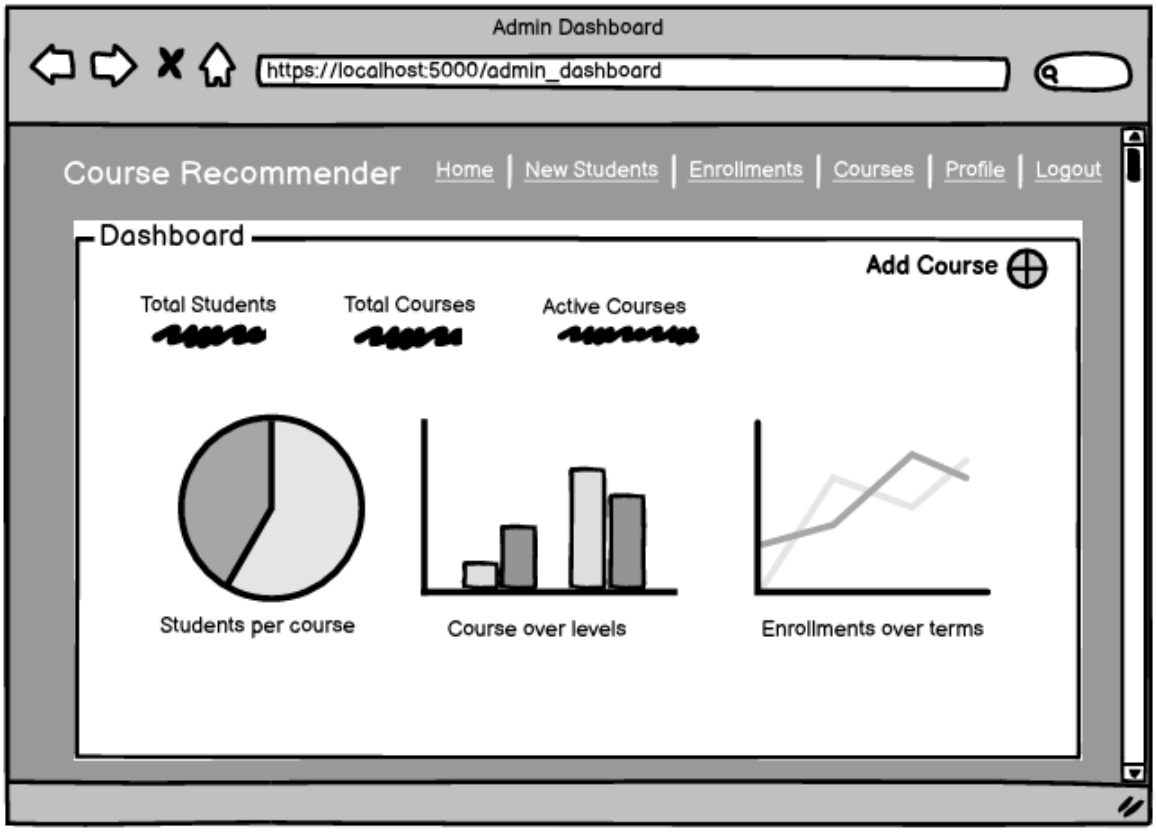
18. Course Recommender



19. New Student Page List



20. Admin Dashboard



Milestone 3 Submission

Sprint Schedule:

Sprint	Duration
1. Requirement Gathering and Designing	01/10/2023 to 10/10/2023
2. Project Scheduling and Designing	15/10/2023 to 24/10/2023
3. Application Development	31/10/2023 to 14/11/2023
4. Application Testing	15/11/2023 to 24/11/2023
5. Final Submission	25/11/2023 to 30/11/2023

Scrum Meeting Schedule:

Sprint Review	Tuesday	9:30 PM
Daily Stand-up call	Monday-Friday	9:30 PM
Backlog Refinement Call	Monday	9:30 PM
Sprint Planning Call	Wednesday	9:30 PM
Sprint Retrospective call	Monday	9:30 PM
Sprint Review	Tuesday	9:30 PM

Few important Minutes of Meeting:

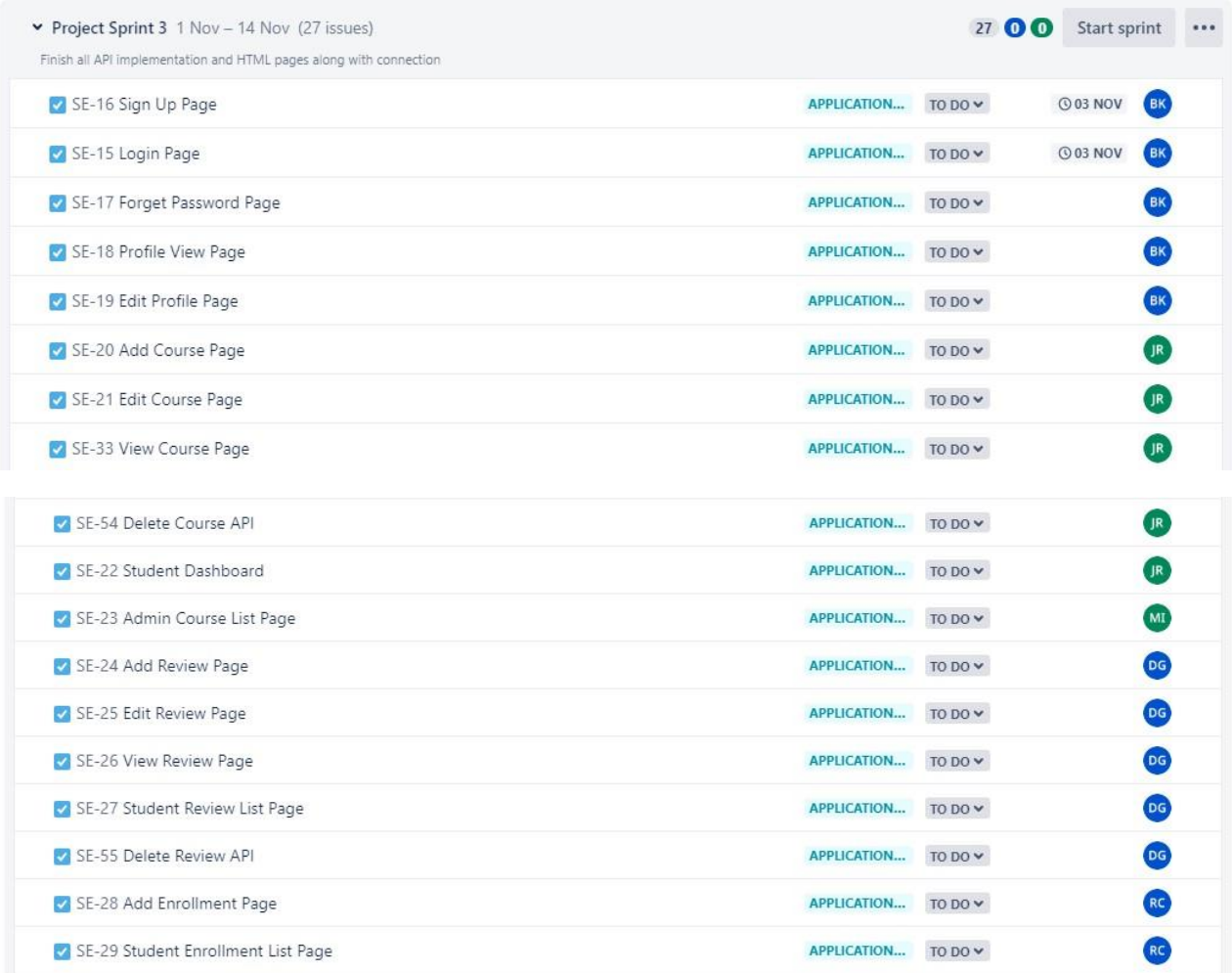
Date	Meeting Type	Description
2/10/2023	Daily Stand-up	Github accounts to be update, Jira account to be created for collaboration
4/10/2023	Sprint Planning	Work distribution among team members. Ikram will be creating the data base design and implementation, Rahul, and Dhruv will be creating Wireframes, Bimlesh/Ranjith will make user-stories, and storyboard
9/10/2023	Backlog Refinement	Discussion about the progress made in the tasks allocated.
11/10/2023	Daily Stand-up	Discussion of Sprint 2
10/10/2023	Sprint Review	Meeting to give the final touches to the final submission, updating honor code and submission of milestone 1
18/10/2023	Sprint Planning	Task allocation for the project. Ranjith will be building Storyboard and python tests for 4 of the wireframes, Dhruv will be taking care of the minutes of meeting, Rahul will be working on the report, Ikram will be working on database design schema
24/10/2023	Sprint Review	Team reviewed the work done under sprint 2.

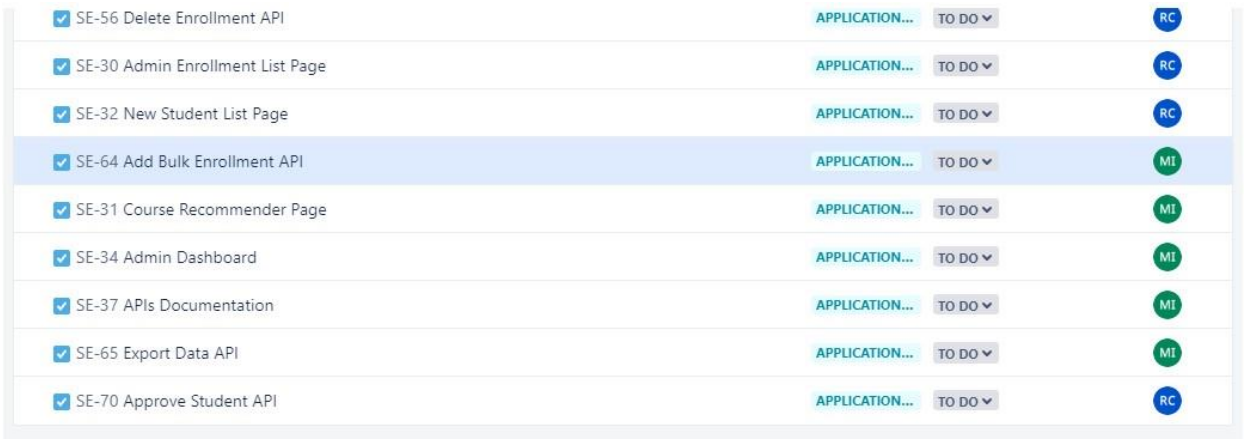
Members of Jira workspace:

Acronym	Names
MI	Mohd Ikram
RC	Rahul Chakraborty
BK	Bimlesh KR Kanth
JR	JS Ranjith
DG	Dhruv Gupta

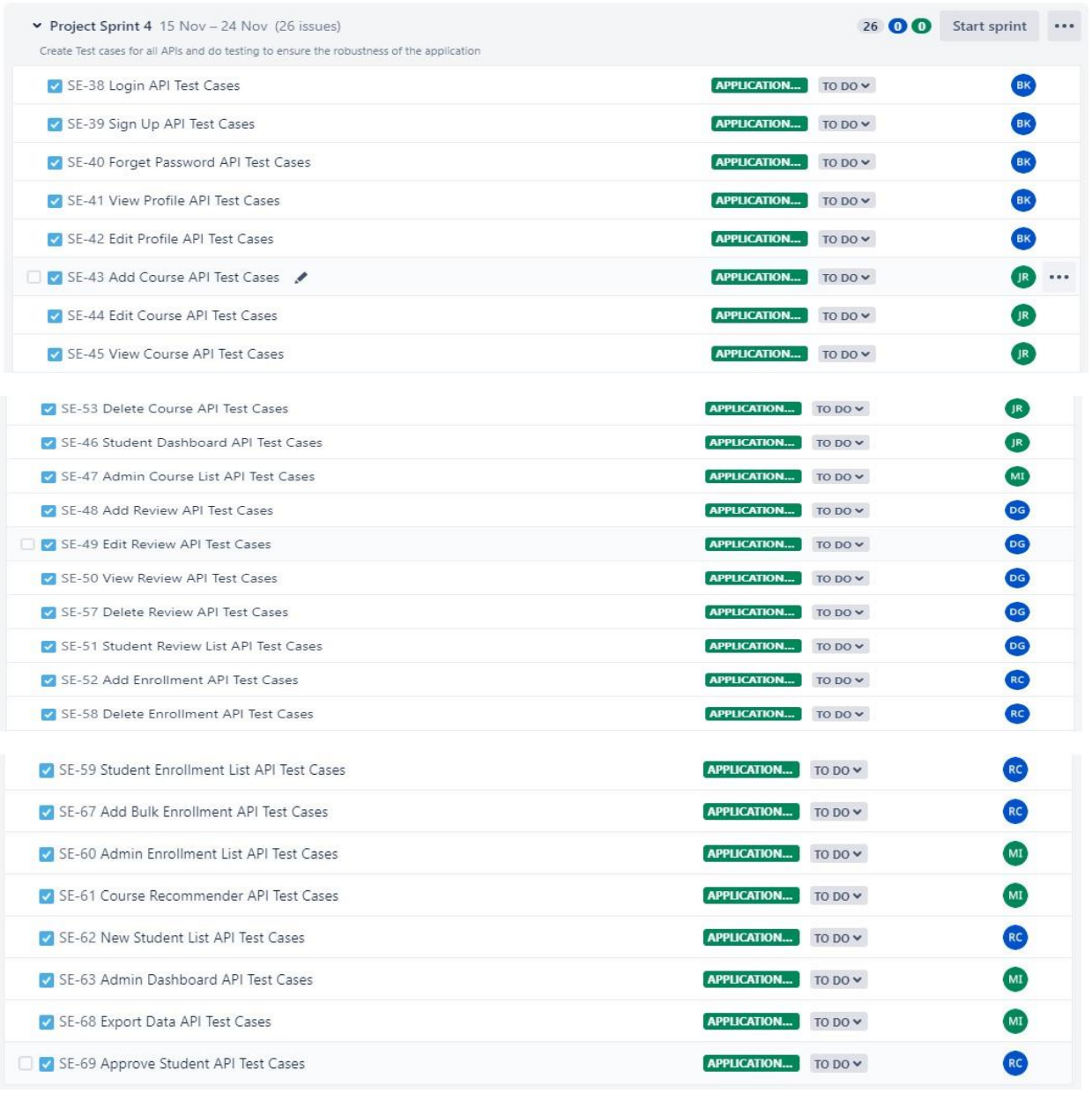
Scheduled Upcoming Tasks Under Sprints:

Sprint 3 (Application Development):

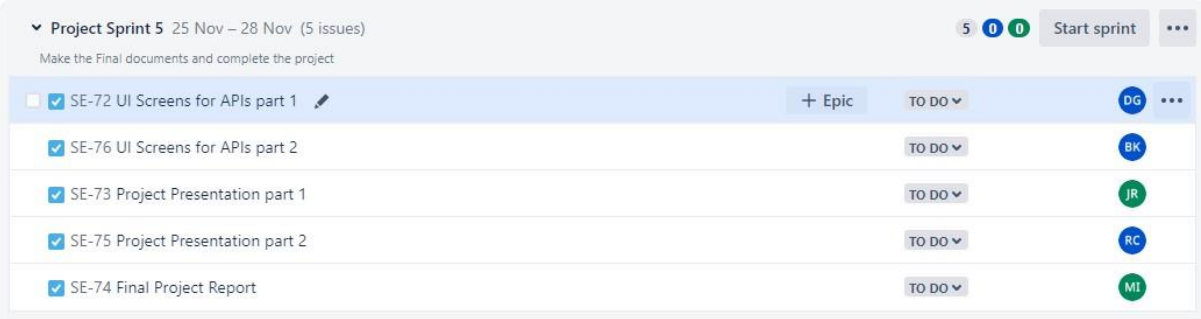




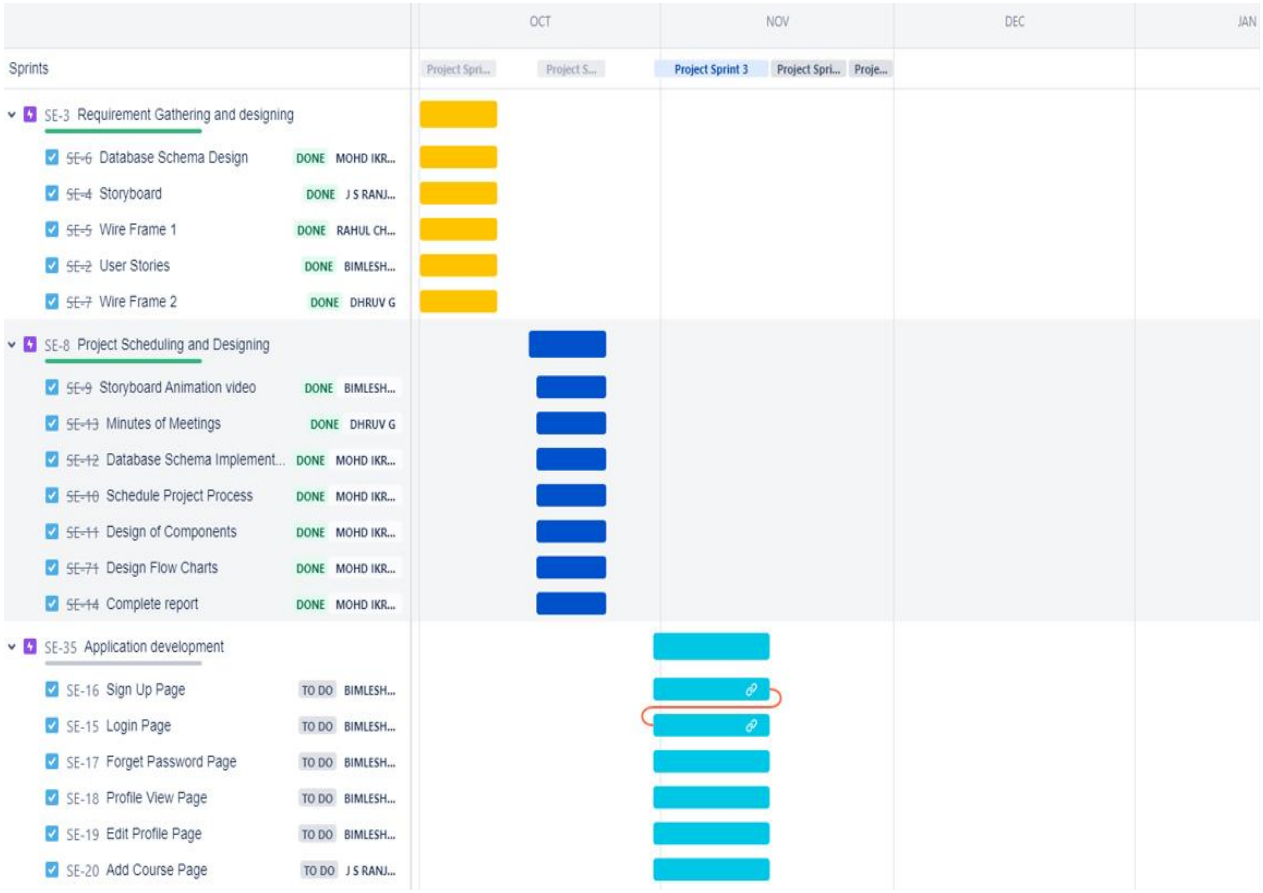
Sprint 4 (Application Testing):

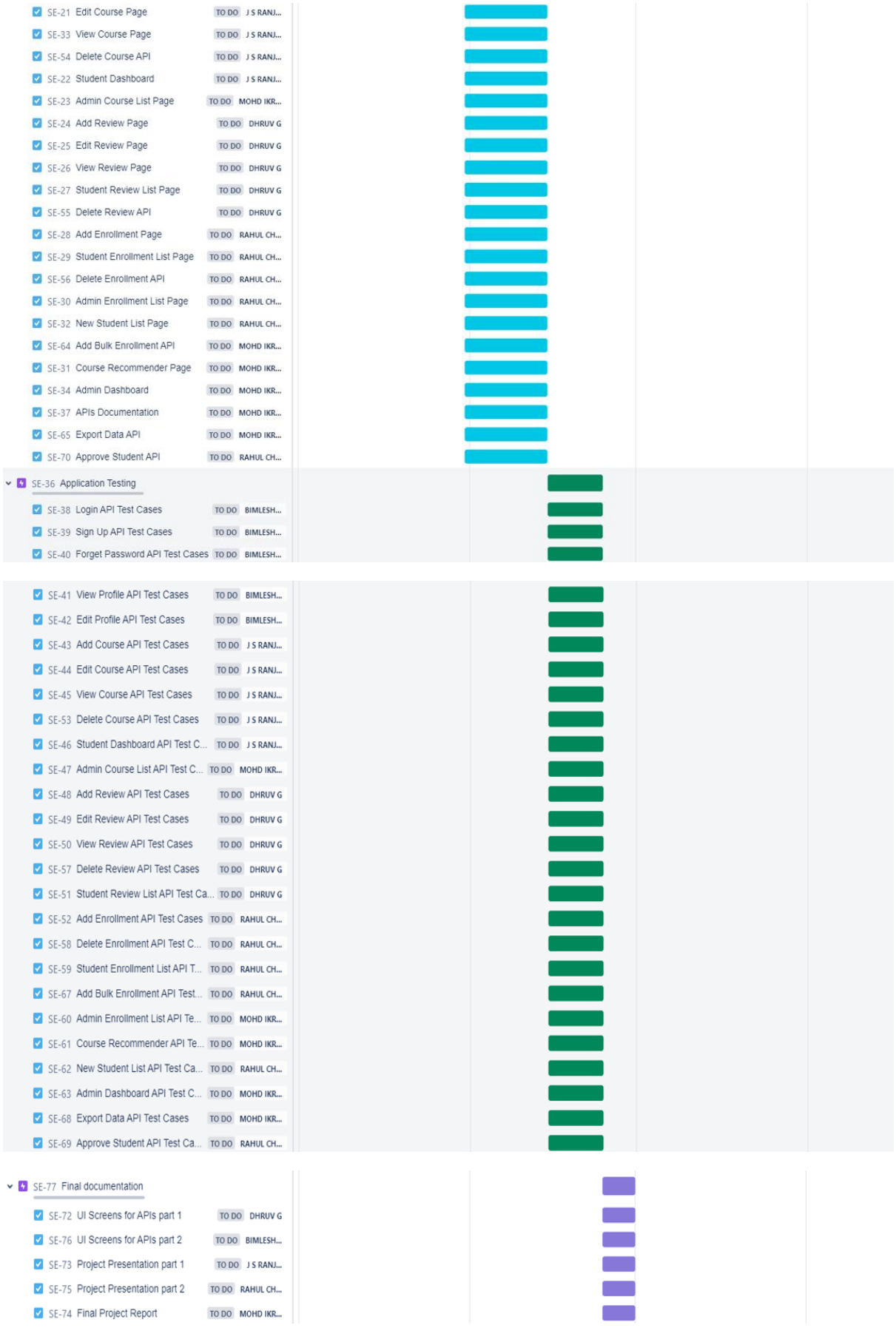


Sprint 5 (Final):

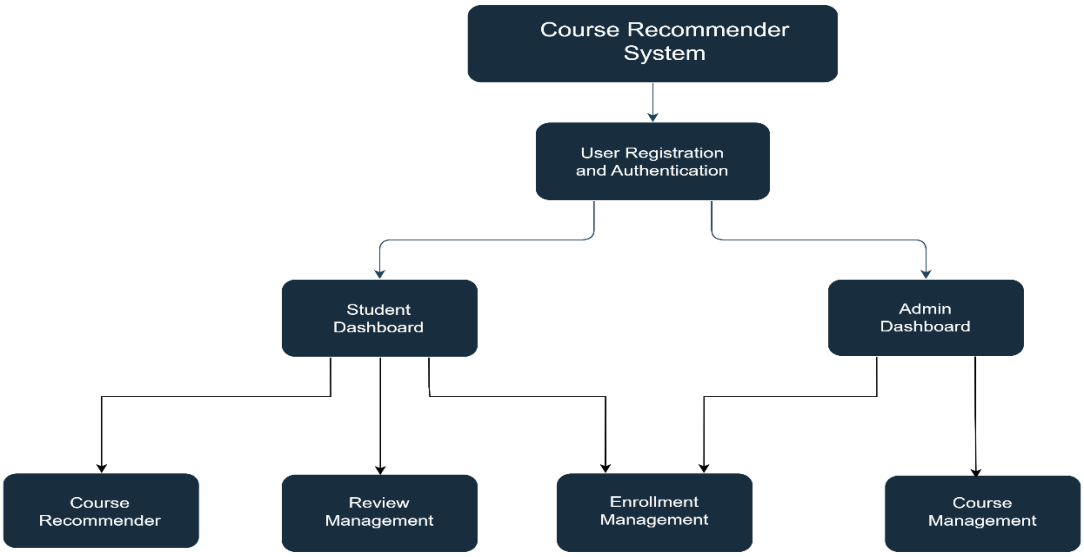


Gantt Chart:

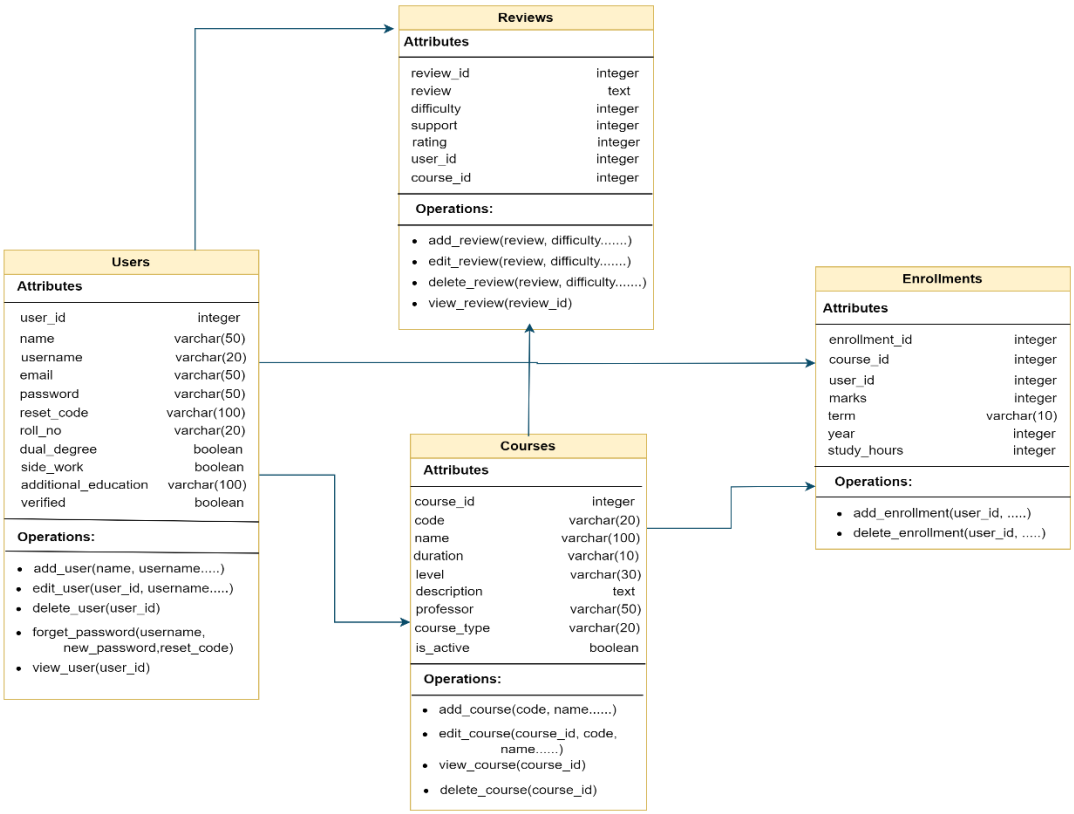



SE-21 Edit Course Page TO DO J S RANJ...
SE-33 View Course Page TO DO J S RANJ...
SE-54 Delete Course API TO DO J S RANJ...
SE-22 Student Dashboard TO DO J S RANJ...
SE-23 Admin Course List Page TO DO MOHD IKR...
SE-24 Add Review Page TO DO DHRUV G
SE-25 Edit Review Page TO DO DHRUV G
SE-26 View Review Page TO DO DHRUV G
SE-27 Student Review List Page TO DO DHRUV G
SE-55 Delete Review API TO DO DHRUV G
SE-28 Add Enrollment Page TO DO RAHUL CH...
SE-29 Student Enrollment List Page TO DO RAHUL CH...
SE-56 Delete Enrollment API TO DO RAHUL CH...
SE-30 Admin Enrollment List Page TO DO RAHUL CH...
SE-32 New Student List Page TO DO RAHUL CH...
SE-64 Add Bulk Enrollment API TO DO MOHD IKR...
SE-31 Course Recommender Page TO DO MOHD IKR...
SE-34 Admin Dashboard TO DO MOHD IKR...
SE-37 APIs Documentation TO DO MOHD IKR...
SE-65 Export Data API TO DO MOHD IKR...
SE-70 Approve Student API TO DO RAHUL CH...
SE-36 Application Testing
SE-38 Login API Test Cases TO DO BIMLESH...
SE-39 Sign Up API Test Cases TO DO BIMLESH...
SE-40 Forget Password API Test Cases TO DO BIMLESH...
SE-41 View Profile API Test Cases TO DO BIMLESH...
SE-42 Edit Profile API Test Cases TO DO BIMLESH...
SE-43 Add Course API Test Cases TO DO J S RANJ...
SE-44 Edit Course API Test Cases TO DO J S RANJ...
SE-45 View Course API Test Cases TO DO J S RANJ...
SE-53 Delete Course API Test Cases TO DO J S RANJ...
SE-46 Student Dashboard API Test C... TO DO J S RANJ...
SE-47 Admin Course List API Test C... TO DO MOHD IKR...
SE-48 Add Review API Test Cases TO DO DHRUV G
SE-49 Edit Review API Test Cases TO DO DHRUV G
SE-50 View Review API Test Cases TO DO DHRUV G
SE-57 Delete Review API Test Cases TO DO DHRUV G
SE-51 Student Review List API Test Ca... TO DO DHRUV G
SE-52 Add Enrollment API Test Cases TO DO RAHUL CH...
SE-58 Delete Enrollment API Test C... TO DO RAHUL CH...
SE-59 Student Enrollment List API T... TO DO RAHUL CH...
SE-67 Add Bulk Enrollment API Test... TO DO RAHUL CH...
SE-60 Admin Enrollment List API Te... TO DO MOHD IKR...
SE-61 Course Recommender API Te... TO DO MOHD IKR...
SE-62 New Student List API Test Ca... TO DO RAHUL CH...
SE-63 Admin Dashboard API Test C... TO DO MOHD IKR...
SE-68 Export Data API Test Cases TO DO MOHD IKR...
SE-69 Approve Student API Test Ca... TO DO RAHUL CH...
SE-77 Final documentation
SE-72 UI Screens for APIs part 1 TO DO DHRUV G
SE-76 UI Screens for APIs part 2 TO DO BIMLESH...
SE-73 Project Presentation part 1 TO DO J S RANJ...
SE-75 Project Presentation part 2 TO DO RAHUL CH...
SE-74 Final Project Report TO DO MOHD IKR...

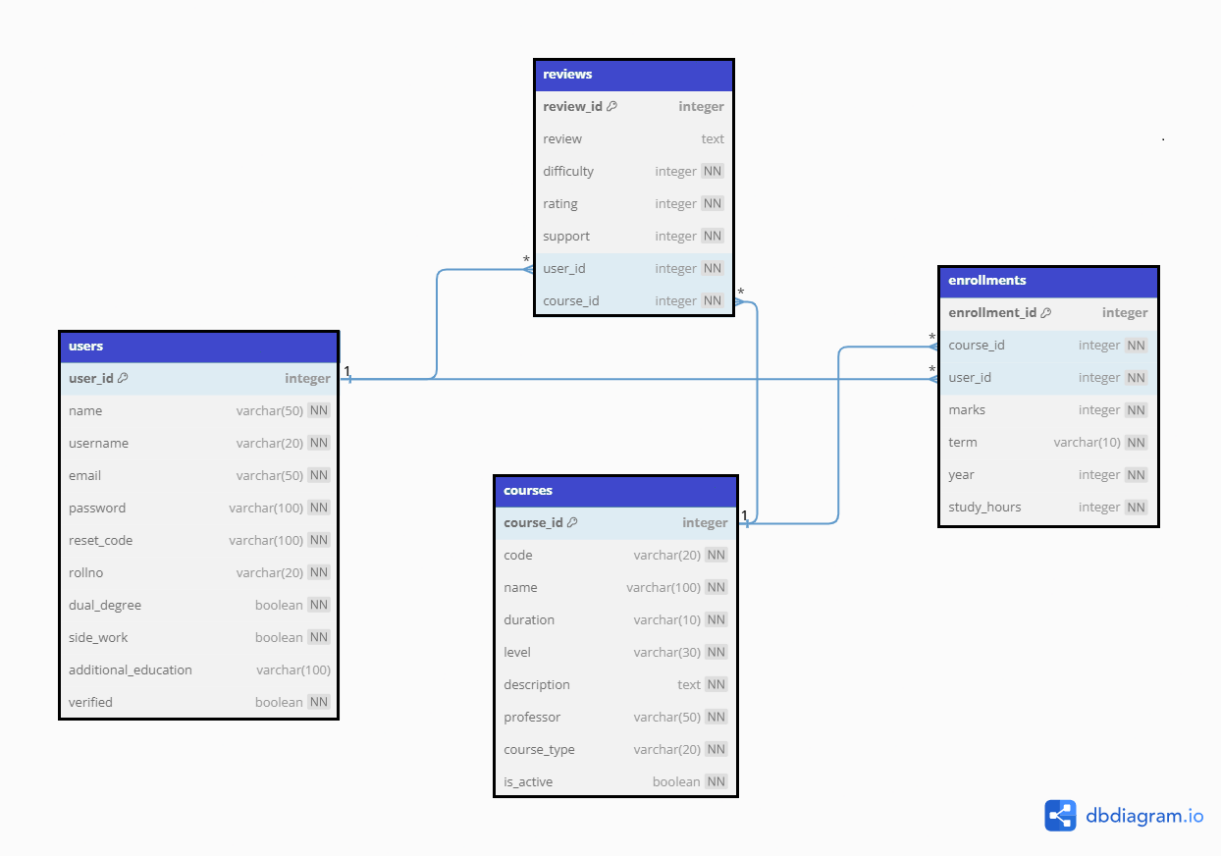
Design Of Components:



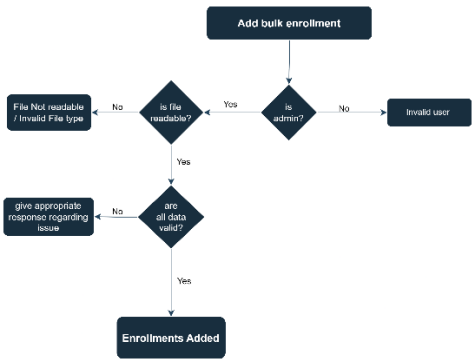
Class Diagram:



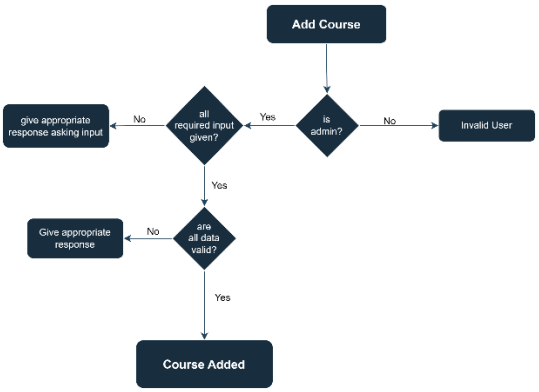
Database Schema:



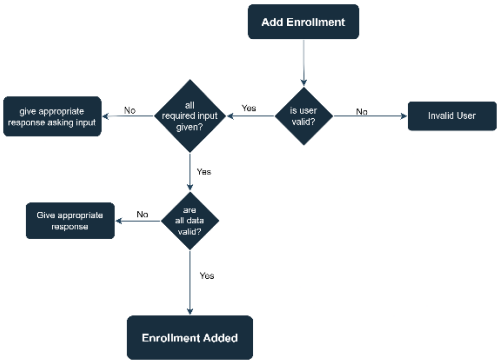
Flowcharts:



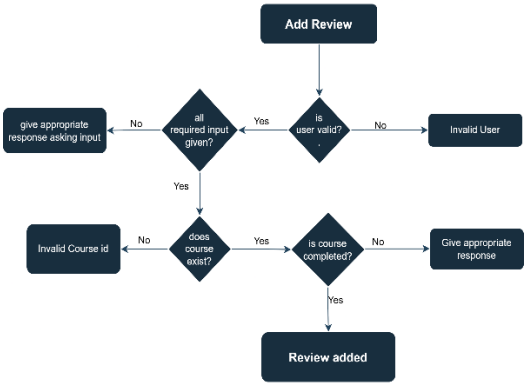
Add Bulk enollment



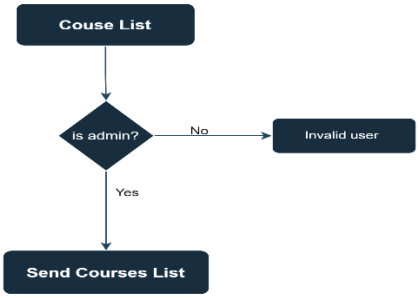
Add Course



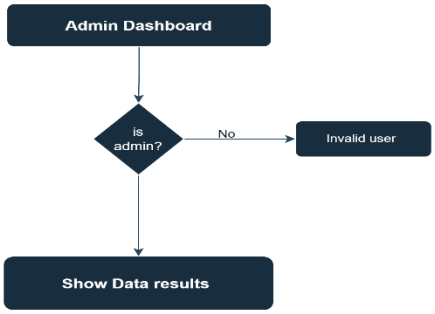
Add Enrollment



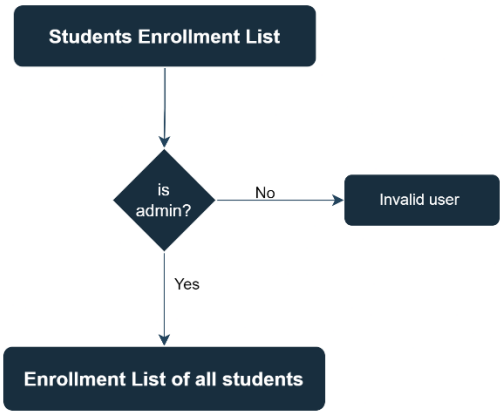
Add Review



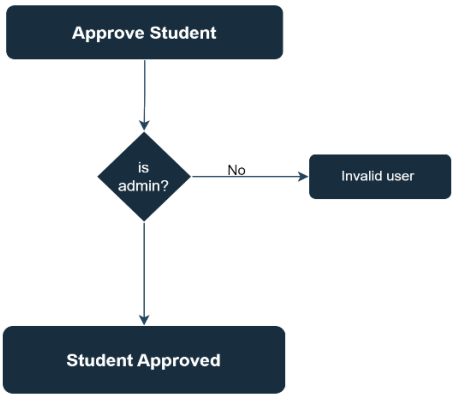
Course List



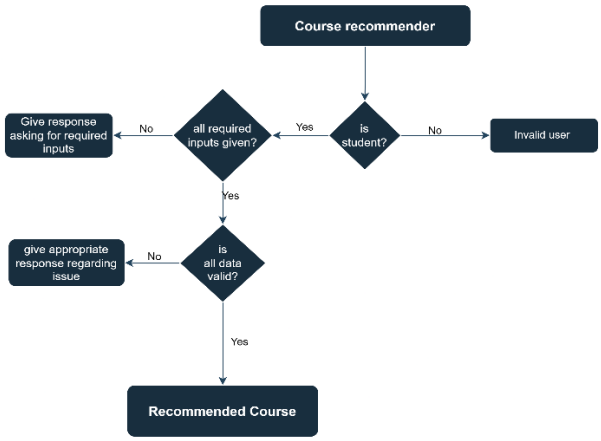
Add Dashboard



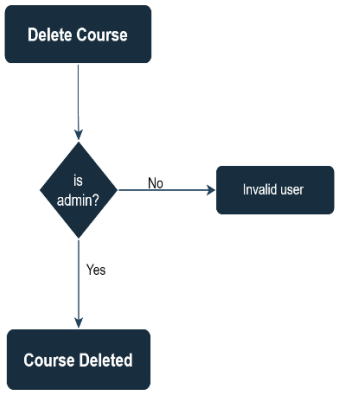
Admin Students Enrollments



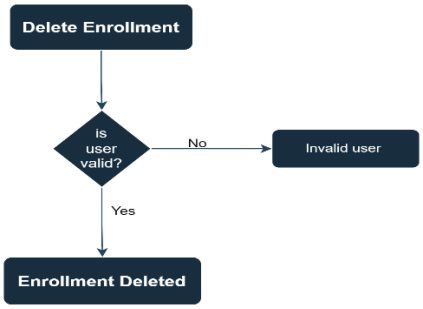
Approve Student



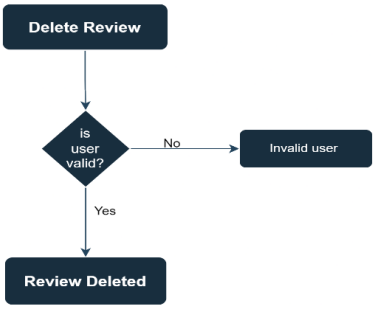
Course Recommender



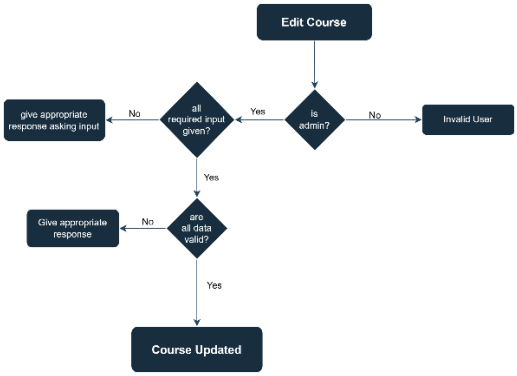
Delete Course



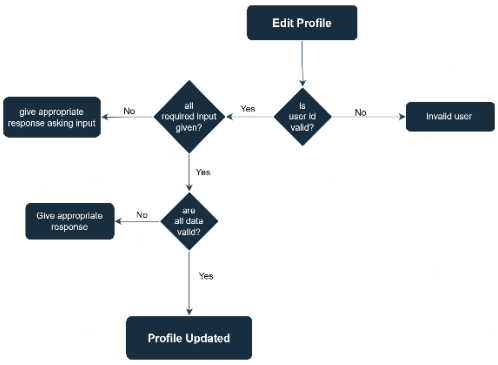
Delete Enrollment



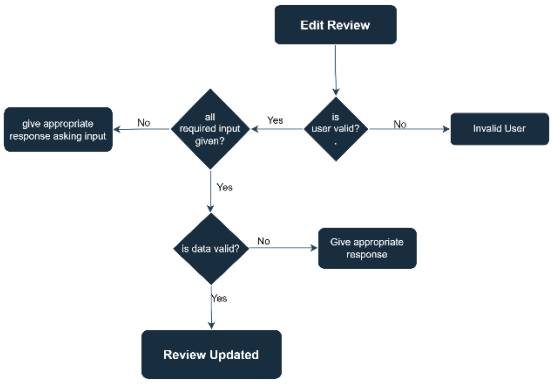
Delete Review



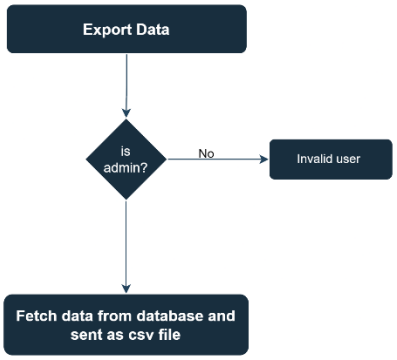
Edit Course



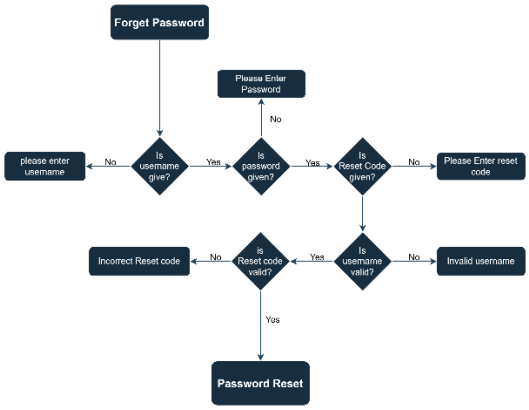
Edit Profile



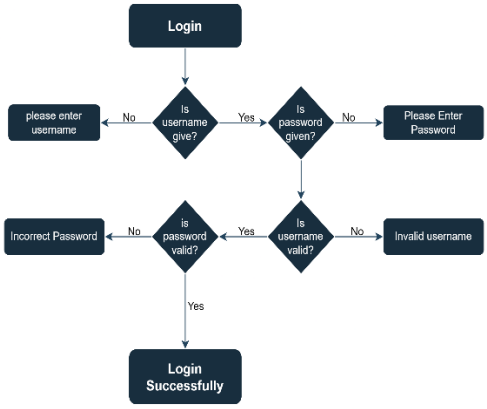
Edit Review



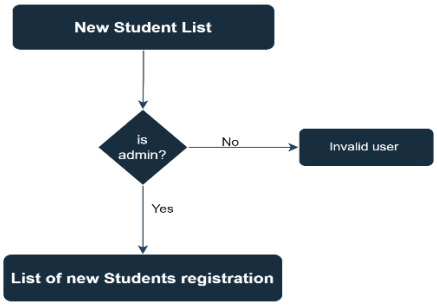
Export Data



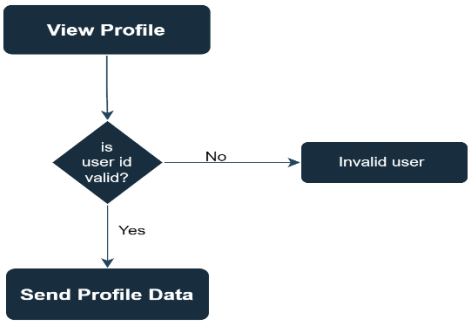
Forget Password



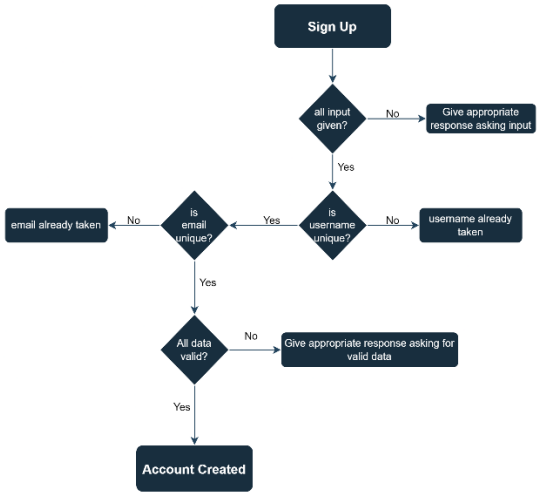
Login



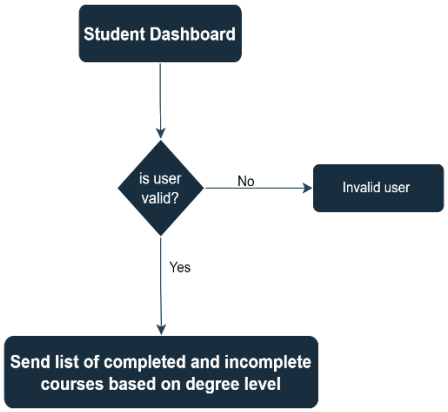
New Students List



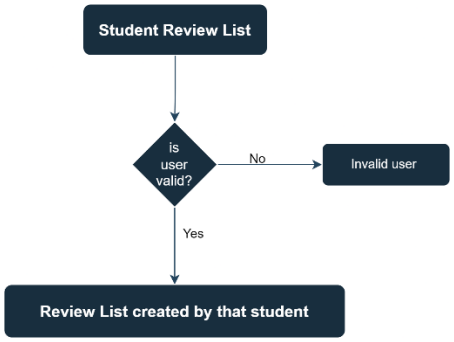
View Profile



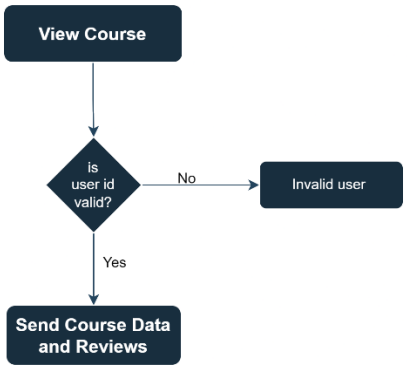
Sign Up



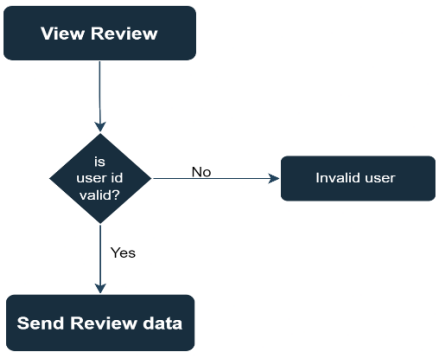
Student Dashboard



Add review

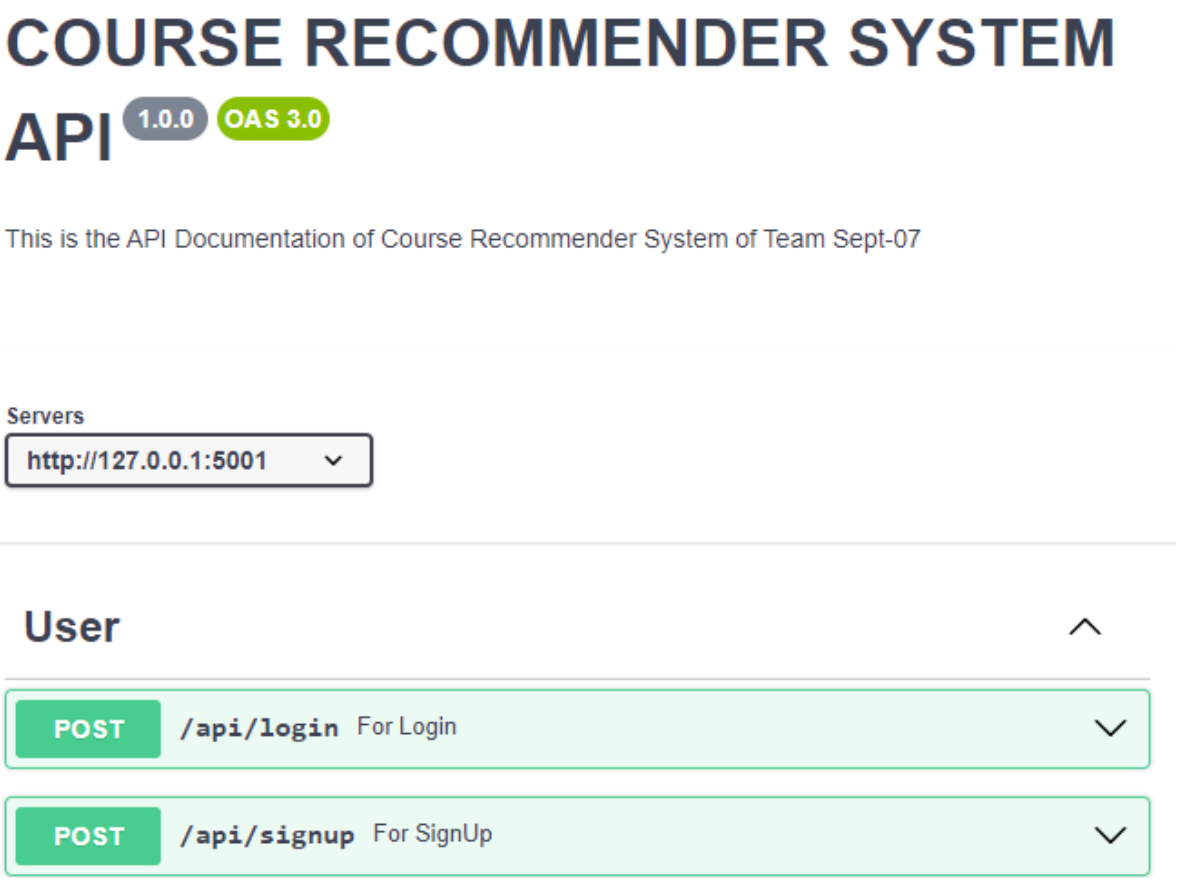


View Course



View Review

Milestone 4 Submission



POST `/api/forget_password` For Forget Password

POST `/api/profile` For creating new User

GET `/api/profile/{user_id}` For getting User data

PUT `/api/profile/{user_id}` For Updating User data

Course

GET `/api/course/{id}` For Getting Course data

PUT `/api/course/{id}` For updating Course data

DELETE `/api/course/{id}` For deleting Course data

POST `/api/course` For creating new Course

GET	/api/courses/student/{user_id}	For getting courses details for the student
POST	/api/courses/student	to get User Enrolled or unenrolled courses

Enrollment

POST	/api/enrollments	For creating new enrollment
GET	/api/enrollments/{enroll_id}	For getting enrollment details
PUT	/api/enrollments/{enroll_id}	For updating enrollment details
DELETE	/api/enrollments/{enroll_id}	For deleting enrollment
GET	/api/enrollments/student/{user_id}	To get Enrollments of the student

Review

POST	/api/review	To create new review
GET	/api/review/{review_id}	To get review data
PUT	/api/review/{review_id}	To updating review data
DELETE	/api/review/{review_id}	To delete review
GET	/api/reviews /student/{user_id}	To get reviews given by student

Admin

PUT	/api/student/approve	For approving new students
GET	/api/enrollments/admin	To get all enrollments

GET	/api/new_students	To get the new students of application
GET	/api/admin_courses_list	To get list of all courses
GET	/api/export_data	Get all the data of application as excel workbook
PUT	/api /update_course_status	For updating course status
GET	/api /admin_dashboard	For admin dashboard data

Additionals

GET	/get_user_name/{user_id}	Get brief data of user
POST	/api/course_recommender	Gives recommended courses

Milestone 5 Submission

Test Case Documentation:

Objective: To login with correct credentials

API being tested: http://127.0.0.1:5001/api/login

Input:

- Request Method : POST
- JSON: {"username": "mdikram888", "password": "12345678"}

Expected Output:

- HTTP Status Code : 200
- JSON : {"message": “Login successful”, “user_id”: 1}

Actual Output:

- HTTP Status Code : 200
- JSON : {"message": “Login successful” ,“user_id”: 1}

Result: Success

```
# login api - correct login credentials
def test_login_correct_credentials(self, client):
    payload = {
        "username": "mdikram888",
        "password": "12345678"
    }
    response = client.post('/api/login', json=payload)
    # Check that the response is as expected
    assert response.status_code == 200
    assert response.json == {
                            "message": "Login successful",
                            "user_id": 7
                        }
```

Objective: Try new user signup with missing roll number

API being tested: http://127.0.0.1:5001/api/signup

Input:

- Request Method : POST
- JSON: { "username": "username4","email": "stone388019@gmail.com", "password": "acdb1234", "name": "bimleshkanth","reset_code": "202001","dual_degree": False, "side_work": False,"additional_education": "PhD"}

Expected Output:

- HTTP Status Code : 404
- JSON : {"message": " 'roll_no' is required"}

Actual Output:

- HTTP Status Code : 404
- JSON : {"message": " 'roll_no' is required"}

Result: Success

```
# signup api - missing roll no
def test_signup_missing_rollno(self, client):
    payload = {
        "username": "username4",
        "email": "stone388019@gmail.com",
        "password": "acdb1234",

        "name": "bimleshkanth",
        "reset_code": "202001",
        "dual_degree": False,
        "side_work": False,
        "additional_education": "PhD"
    }
    response = client.post('/api/signup', json=payload)
    # Check that the response is as expected
    assert response.status_code == 404
    assert response.json == {
                                "message": "'roll_no' is required"
                            }
```

Objective: Try entering non-alphanumeric username while doing a password reset

API being tested: http://127.0.0.1:5001/api/forget_password

Input:

- Request Method : POST
- JSON : { "username": "username4@#", "new_password": "newpwd1234", "reset_code": "202001"}

Expected Output:

- HTTP Status Code : 400
- JSON : {"message": " Invalid username format, Only Alphanumeric is allowed"}

Actual Output:

- HTTP Status Code : 400
- JSON : {"message": " Invalid username format, Only Alphanumeric is allowed"}

Result: Success

```
# password api - no alphanumeric username passed
def test_pwd_reset_no_alphanumeric_username(self, client):
    payload = {
        "username": "username4@#",
        "new_password": "newpwd1234",
        "reset_code": "202001",
    }
    response = client.post('/api/forget_password', json=payload)
    # Check that the response is as expected
    assert response.status_code == 400
    assert response.json == {
                                "message": "Invalid username format, Only Alphanumeric is allowed"
                            }
```

Objective: Try updating user profile email with an existing email in database

API being tested: http://127.0.0.1:5001/api/profile/8

Input:

- Request Method : PUT
- JSON: { "email": "game388019@gmail.com", "password": "acdbe1234", "username": "username4", "name": "bimleshkanth", "reset_code": "202001","dual_degree": False, "side_work": False, "additional_education": "PhD"}

Expected Output:

- HTTP Status Code : 400
- JSON : {"message": " Email is already taken, please try another"}

Actual Output:

- HTTP Status Code : 400
- JSON : {"message": Email is already taken, please try another"}

Result: Success

```
def test_update_user_email_existing(self, client):
    payload = {

        "email": "game388019@gmail.com",
        "password": "acdbe1234",
        "username" : "username4",
        "name": "bimleshkanth",
        "reset_code": "202001",
        "dual_degree": False,
        "side_work": False,
        "additional_education": "PhD"
    }
    response = client.put('/api/profile/8', json=payload)
    # Check that the response is as expected
    assert response.status_code == 400
    assert response.json == {"message": "Email is already taken, please try another"}
```

Objective: To approve new student success

API being tested: http://127.0.0.1:5001 /api/student/approve

Input:

- Request Method : PUT
- JSON: {"user_id": 1}

Expected Output:

- HTTP Status Code : 200
- JSON : {"message": "User approved"}

Actual Output:

- HTTP Status Code : 400
- JSON : {"message": Email is already taken, please try another"}

Result: Success

```
def test_approve_student_success(self, client):
    payload = {"user_id": 1}
    response = client.put('/api/student/approve', json=payload)
    assert response.status_code == 200
    assert response.json == {"message": "User approved"}
```

Objective: To get the Course Details

API being tested: http://127.0.0.1:5000/api/course/-1

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid Course id"}

Actual Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid Course id"}

Result: Success

```
def test_getCourseFailure(self,client):
    response = client.get('api/course/-1')

    assert response.status_code == 404
    assert response.json == {"message": "Invalid Course id"}
```

Objective: To add a new course with already taken course code

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

Input:

- Request Method : POST
- JSON : {

"code": "CS001215", "course_id": 2,

"course_type": "Diploma in Programming",

"description": "Testing API POST", "duration": "8 weeks",

"is_active": True, "level": "1",

"name": "Python", "professor": "It is Me"

}

Expected Output:

- HTTP Status Code : 400
- JSON : {"message": "Course Code is already taken"}

Actual Output:

- HTTP Status Code : 400
- JSON : {"message": "Course Code is already taken"}

Result: Success

```
def test_postCourseFailure_course_code_already_taken(self,client):
    payload = {
        "code": "CS001215",
        "course_id": 2,
        "course_type": "Diploma in Programming",
        "description": "Testing API POST",
        "duration": "8 weeks",
        "is_active": True,
        "level": "1",
        "name": "Python",
        "professor": "It is Me"
    }

    response = client.post('/api/course',json=payload)

    assert response.status_code == 400
    assert response.json == {"message": "Course Code is already taken"}
```

Objective: Delete a course

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

Input:

- Request Method : DELETE

Expected Output:

- HTTP Status Code : 200
- JSON : {"message": "Course deleted Successfully"}

Actual Output:

- HTTP Status Code : 200
- JSON : {"message": "Course deleted Successfully"}

Result: Success

```
def test_deleteCourseSuccess(self,client):

    response = client.delete('/api/course/1')

    assert response.status_code == 200
    assert response.json == {"message": "Course deleted Successfully"}
```

Objective: Update Course details

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

Input:

- Request Method : PUT
- JSON: {

  "code": "CS123456", "course_id": 2,

  "course_type": "Diploma in Programming",

  "description": "Testing API POST",

  "duration": "8 weeks", "is_active": True,

  "level": "1", "name": "SE",

  "professor": "It is Ranjith"

  }

Expected Output:

- HTTP Status Code : 200
- JSON : {"message": "Course Updated Successfully"}

Actual Output:

- HTTP Status Code : 200
- JSON : {"message": "Course Updated Successfully"}

Result: Success

```
def test_putCourseSuccess(self,client):
    payload = {
        "code": "CS123456",
        "course_id": 2,
        "course_type": "Diploma in Programming",
        "description": "Testing API POST",
        "duration": "8 weeks",
        "is_active": True,
        "level": "1",
        "name": "SE",
        "professor": "It is Ranjith"
    }

    response = client.put('/api/course/2',json=payload)

    assert response.status_code == 200
    assert response.json == {"message": "Course Updated Successfully"}
```

Objective: Get all courses registered by certain student

API being tested: http://127.0.0.1:5000/courses/student/1

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200

Actual Output:

- HTTP Status Code : 200

Result: Success

```
def test_getUserCoursesSuccess(self,client):

    response = client.get('/api/courses/student/1')

    assert response.status_code == 200
    assert response.json["username"] == "M.Ikramm"
    assert len(response.json["Enrolled"]) == 1
    assert len(response.json["Unenrolled"]) == 3
```

Objective : To get the enrollments of a particular student which does not exist

API being tested: http://127.0.0.1:5000/api/enrollments/student/1000

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 404
- JSON : { “message” : “user_id is invalid” }

Actual Output:

- HTTP Status Code : 404
- JSON : { “message” : “user_id is invalid” }

Result: Success

```
def test_get_enrollment_student_not_found(self, client):
    response = client.get('api/enrollments/student/1000')
    assert response.status_code == 404
    assert response.json == {"message": "user_id is invalid"}
```

Objective : To get the enrollments of all students

API being tested: http://127.0.0.1:5000/api/enrollments/admin

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200
- JSON : { “data” : [{“course_name” : “Java” , “user_name” : “M.Ikramm”} , {“course_name” : “SE” , “user_name” : “Mohd Ikram”} , {“course_name” : “FastX” , “user_name” : “M.Ikramm”} , “course_name” : “SE” , “user_name” : “Rahul”] }

Actual Output:

- HTTP Status Code : 200
- JSON : { “data” : [{“course_name” : “Java” , “user_name” : “M.Ikramm”} , {“course_name” : “SE” , “user_name” : “Mohd Ikram”} , {“course_name” : “FastX” , “user_name” : “M.Ikramm”} , “course_name” : “SE” , “user_name” : “Rahul”] }

Result: Success

```
def test_all_enrollment(self, client):
    response = client.get('api/enrollments/admin')
    assert response.status_code == 200
    assert "data" in response.json
    assert type(response.json["data"]).__name__ == 'list'
```

Objective : To update enrollment details of a pre-existing enrollment.

API being tested: http://127.0.0.1:5000/api/enrollments/5

Input:

- Request Method : PUT
- Payload : { “marks” : 70, “term” : “Sept”, “year” : 2022, “study_hours” : 30 }

Expected Output:

- HTTP Status Code : 202
- JSON : { “message” : “Enrollment Updated Successfully” }

Actual Output:

- HTTP Status Code : 404
- JSON : { “message” : “Enrollment Updated Successfully” }

Result: Success

```
def test_put_enrollment_id_success(self, client):
    payload = {
                    "marks": 70,
                    "term": "Sept",
                    "year": 2022,
                    "study_hours": 30
                }
    response = client.put('api/enrollments/5', json=payload)
    assert response.status_code == 202
    assert response.json == {"message": "Enrollment Updated Successfully"}
```

Objective : To add enrollment details of a user with invalid course ID.

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

Input:

- Request Method : POST
- Payload : { “course_id” : 200, “user_id” : 3, “marks”: 70, “term” : “Sept”, “year” : 2022, “study_hours” : 20 }

Expected Output:

- HTTP Status Code : 404
- JSON : { “message” : “Invalid course_id” }

Actual Output:

- HTTP Status Code : 404
- JSON : { “message” : “Invalid course_id” }

Result: Success

```
def test_post_enrollment_case_3(self, client):
    payload = {
                    "course_id": 200,
                    "user_id": 3,
                    "marks": 70,
                    "term": "Sept",
                    "year": 2022,
                    "study_hours": 20
                }
    response = client.post('/api/enrollments', json=payload)
    assert response.status_code == 404
    assert response.json == {"message": "Invalid course_id"}
```

Objective : To add enrollment details of a user with a new course.

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

Input:

- Request Method : POST
- Payload : { “course_id” : 1, “user_id” : 3, “marks”: 100, “term” : “Sept”, “year” : 2023, “study_hours” : 40 }

Expected Output:

- HTTP Status Code : 201
- JSON : { “message” : “Enrollment Added Successfully” }

Actual Output:

- HTTP Status Code : 201
- JSON : { “message” : “Enrollment Added Successfully” }

Result: Success

```
def test_post_enrollment_case_5(self, client):
    payload = {"course_id": 1, "user_id": 3, "marks": 100, "term": "Sept", "year": 2023, "study_hours":
    response = client.post('/api/enrollments', json=payload)
    assert response.status_code == 201
    assert response.json == {"message": "Enrollment Added Successfully"}
```

Objective : To delete an enrollment.

API being tested: http://127.0.0.1:5000/api/enrollments/6

Input:

- Request Method : DELETE

Expected Output:

- HTTP Status Code : 200
- JSON : { “message” : “Enrollment deleted Successfully” }

Actual Output:

- HTTP Status Code : 200
- JSON : { “message” : “Enrollment deleted Successfully” }

Result: Success

```
def test_delete_enrollment_id_success(self, client):
    response = client.delete('/api/enrollments/6')
    assert response.status_code == 200
    assert response.json == {"message": "Enrollment deleted Successfully"}
```

Objective : Add review Successfully.

API being tested: http://127.0.0.1:5001/api/review

Input:

- Request Method : POST
- JSON: {“user_id”: 1, “course_id”: 1, “difficulty”: 5, “support”: 3, “rating”: 4,

  “review”: “blab blabalba”

Expected Output:

- HTTP Status Code : 201
- JSON : {"message": "Review added successfully"

  }

Actual Output:

- HTTP Status Code : 201
- JSON : {"message": "Review added successfully"

  }

Result: Success

```
    def test_post_review_success(self,client):
        payload = {
    "user_id": 1,
    "course_id": 1,
    "difficulty": 5,
    "support": 3,
    "rating": 4,
    "review": "blab blabalba"
}

        response = client.post('/api/review', json = payload)
        assert response.status_code == 201
        assert response.json == {
    "message": "Review Added Successfully"
}
```

Objective: Get Review - to retrieve review from the database Successfully

API being tested: 127.0.0.1:5001/api/review/2

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200
- JSON : { "data": { "course_id": 1, "course_name": "Java", "difficulty": 1,

"rating": 5, "review": "23", "review_id": 2,
"support": 4,

"user_id": 5 } }

Actual Output:

- HTTP Status Code : 200
- JSON : { "data": { "course_id": 1, "course_name": "Java", "difficulty": 1,

"rating": 5, "review": "23", "review_id": 2, "support": 4,
"user_id": 5} }

- **Result:** Success

```
def test_get_review_success(self, client):
    # Simulate a GET request to the '/' route
    response = client.get('/api/review/2')
    json_data = response.json
    assert response.status_code == 200
    assert "data" in json_data
    assert type(json_data["data"]).__name__ == "dict"
    assert "course_id" in json_data["data"]
    assert type(json_data["data"]["course_id"]).__name__ == "int"
    assert "course_name" in json_data["data"]
    assert type(json_data["data"]["course_name"]).__name__ == "str"
```

Objective: Edit Review Success

API being tested: 127.0.0.1:5001/api/review

Input:

- Request Method : PUT
- JSON: { "review_id": 2, "difficulty": 1, "support": 4, "rating": 5, "review": "Wonderful"}

Expected Output:

- HTTP Status Code : 202
- JSON : { "message": "Review Successfully Updated" }

Actual Output:

- HTTP Status Code : 202
- JSON : { "message": "Review Successfully Updated"}
- **Result:** Success

```
    new *
    def test_put_review_success(self,client):
        payload = {
    "review_id": 2,
    "difficulty": 1,
    "support": 4,
    "rating": 5,
    "review": "Wonderful"
}

        response = client.put('/api/review', json = payload)
        assert response.status_code == 202
        assert response.json == {
    "message": "Review Successfully Updated"
}
```

Objective: Get Student Review Success

API being tested: 127.0.0.1:5001/api/reviews/student/1

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200
- JSON :{ "data": [

{"course_id": 2, "course_name": "SE","difficulty": 5, "rating": 4, "review": "blab blabalba", "review_id": 3, "support": 3, "user_id": 1}

,

{ "course_id": 1, "course_name": "Java", "difficulty": 5, "rating": 4,

"review": "blab blabalba", "review_id": 5, "support": 3, "user_id": 1

}]}

Actual Output:

- HTTP Status Code : 200
- JSON : { "data": [

{"course_id": 2, "course_name": "SE","difficulty": 5, "rating": 4, "review": "blab blabalba", "review_id": 3, "support": 3, "user_id": 1}

,

{ "course_id": 1, "course_name": "Java", "difficulty": 5, "rating": 4,

"review": "blab blabalba", "review_id": 5, "support": 3, "user_id": 1

}]}

- **Result:** Success

```python
new *
def test_get_student_review_success(self, client):
    # Simulate a GET request to the '/' route
    response = client.get('/api/reviews/student/1')
    json_data = response.json
    assert response.status_code == 200
    assert "data" in json_data
    assert type(json_data["data"]).__name__ == "list"
    assert "course_id" in json_data["data"][0]
    assert type(json_data["data"][0]["course_id"]).__name__ == "int"
    assert "course_name" in json_data["data"][0]
    assert type(json_data["data"][0]["course_name"]).__name__ == "str"
```

Objective: Delete review Success/Failure

API being tested: 127.0.0.1:5001/api/review/5

Input:

- Request Method : DELETE

Expected Output:

- HTTP Status Code : 200
- JSON : {"message": "Review Deleted Successfully"}

Actual Output:

- HTTP Status Code : 200
- JSON : {"message": "Review Deleted successfully" }

Result: Success

```
    new *
    def test_del_review_success(self,client):



        response = client.delete('/api/review/5')
        assert response.status_code == 200
        assert response.json == {
    "message": "Review Deleted Successfully"
}
```

Objective: To get the Admin Course List

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

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200
- JSON : {“data”: [{
"avg_difficulty": 3.0,
"avg_rating": 2.5,
"code": "CS001214",
"course_id": 1,
"course_type": "Diploma in Programming",
"description": "bla bla bla bla4",
"duration": "12 weeks",
"is_active": True,
"level": "1",
"name": "Java",
"professor": "Someone",
"support": 2.0
}] }

- **Actual Output:**
- HTTP Status Code : 200
- JSON : { “data” : [{
"avg_difficulty": 3.0,
"avg_rating": 2.5,
"code": "CS001214",
"course_id": 1,
"course_type": "Diploma in Programming",
"description": "bla bla bla bla4",
"duration": "12 weeks",
"is_active": True,
"level": "1",
"name": "Java", "professor": "Someone", "support": 2.0 }] }

Result: Success

```
def test_admin_course_list_success(self, client):
    response = client.get('/api/admin_courses_list')

    json_data = response.json
    assert response.status_code == 200
    assert "data" in json_data
    assert type(json_data["data"]).__name__ == 'list'
    assert type(json_data["data"][0]).__name__ == 'dict'
    assert json_data["data"][0] == {
"avg_difficulty": 3.0,"avg_rating": 2.5,"code": "CS001214",
"course_id": 1,"course_type": "Diploma in Programming","description": "bla bla bla bla4","duration": "12
"is_active": True,"level": "1", "name": "Java","professor": "Someone", "support": 2.0
}
```

Objective: To Update Course Status by admin

Page being tested: http://127.0.0.1:5000/api/update_course_status

Input:

- Request Method : PUT
- JSON : {"course_id": -1}

Expected Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid Course Id"}

Actual Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid Course Id"}

Result: Success

```
def test_update_course_status_without_course_id(self, client):
    payload = {}
    response = client.put('/api/update_course_status', json=payload)

    assert response.status_code == 404
    assert response.json == {"message": "course_id is required"}
```

Objective: Get basic user Info

Page being tested: http://127.0.0.1:5000/api/get_user_name/2

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid User Id"}

Actual Output:

- HTTP Status Code : 404
- JSON : {"message": "Invalid User Id"}

Result: Success

```
def test_get_user_name_invalid_id(self, client):
    response = client.get('/api/get_user_name/2')

    assert response.status_code == 404
    assert response.json == {"message": "Invalid User Id"}
```

Objective: To Use Course Recommender

Page being tested: http://127.0.0.1:5000/api/course_recommender

Input:

- Request Method : POST
- JSON: {
- "user_id": 1,
- "no_of_courses": 1,
- "study_hour": 20,
- "level": 1
- }

Expected Output:

- HTTP Status Code : 200
- JSON : { "data": [{ "course_name": "Java", "estimated_marks": 90.0 }] }

Actual Output:

- HTTP Status Code : 200
- JSON : { "data": [{ "course_name": "Java", "estimated_marks": 90.0 }] }

Result: Success

```
def test_course_recommender_success(self, client):
    payload = {"user_id": 1,"no_of_courses": 1,"study_hour": 20,"level": 1}
    response = client.post('/api/course_recommender', json=payload)

    json_data = response.json
    assert response.status_code == 200
    assert "data" in json_data
    assert type(json_data["data"]).__name__ == 'list'
    assert len(json_data["data"]) == payload["no_of_courses"]
    assert "course_name" in json_data["data"][0]
    assert "estimated_marks" in json_data["data"][0]
    assert type(json_data["data"][0]["course_name"]).__name__ == 'str'
    assert type(json_data["data"][0]["estimated_marks"]).__name__ == 'float'
    assert json_data == {"data": [{"course_name": "Java","estimated_marks": 90.0}]}
```

Objective: Get all courses registered by certain student

Page being tested: http://127.0.0.1:5000/courses/student/1

Input:

- Request Method : GET

Expected Output:

- HTTP Status Code : 200

Actual Output:

- HTTP Status Code : 200

Result: Success

```
def test_getUserCoursesSuccess(self,client):

    response = client.get('/api/courses/student/1')

    assert response.status_code == 200
    assert response.json["username"] == "M.Ikramm"
    assert len(response.json["Enrolled"]) == 1
    assert len(response.json["Unenrolled"]) == 3
```

Milestone 6 Submission

- **Full code structure pushed to github repository.**
- **Presentation file**
- **Presentation video, demonstration of backend, frontend and hosting videos**
- **Complete Presentation video : click here**
- **Backend Demonstration: click here**
- **Frontend Demonstration: click here**
- **Hosting Video: click here**

Technologies and Tools Used: Overall Development Process

- As specified in the problem statement, version control software (git) was used throughout the entire project's construction.
- Bash was extensively used to perform version control with git.
- Visual Studio Code and pycharm was the primary IDE used for developing the app.
- We have also used postman for testing the APIs and swagger for API documentation
- Google Chrome and Microsoft Edge were the browsers used throughout the development process.
- During the development and testing phase, data was inserted, removed, and edited in the SQLite database in the backend using DB Browser for SQLite.

Technologies and Tools Used: Frontend

- Flask - Jinja2 templating was used to build the frontend. A lot of Jinja2's features, such as dynamic web page generation, extensions, template inheritance, etc., were utilised.
- The entire application made use of Bootstrap for styling. A variety of Bootstrap elements, including buttons, tables, cards, and more, were used for design.

Technologies and Tools Used: Backend

- Python-based frameworks were used to develop the backend
- The backend server was built using the Flask web framework. Many Flask sub-components were used, including Flask-CORS (to enable cross-origin queries) and Flask-RESTful (for the API)
- The ORM used to enable interactions between the application's backend Flask API and the SQLite database was Flask-SQLAlchemy.

Technologies and Tools Used: Hosting

- Azure cloud is used for hosting the website on the internet
- Virtual Machine was created and setup with network settings on cloud to run the controller and API server online.

Deviations from initial milestone submissions

1. Slight deviations from the initial wireframes (Milestone 2): There have been some minor deviations from the initial wireframes that have been submitted, such as placement of buttons, slight changes in the UI elements, layout, colours, etc

Where is our application hosted?

1. Our application (i.e the API server and the frontend itself) is hosted and live on the cloud. It runs on Azure virtual machine setup to receive requests on port 5001 for API request and 5000 for controller (i.e the frontend) and can be access on the below link:

   API: http://course-recommender.northcentralus.cloudapp.azure.com:5001/

   Frontend: http://course-recommender.northcentralus.cloudapp.azure.com:5000/

2. It also runs locally on host “127.0.0.1”, on port “5000” for frontend server and host “127.0.0.1”, on port “5001” for API server.

3. The codebase is also on Github repository.

   https://github.com/mdikram88/Software_Engineering_Project

Steps to run our application Locally:

1. Download the application from the Github project repository, either as a ZIP file or by cloning the repository.

2. Open Terminal and run “pip install-r requirements.txt” in the terminal. This will install all the necessary requirements for the app.

3. Run “python main.py” for starting API server and then start another terminal and run “cd ./application/Controller” followed by “python user_controller.py” for starting frontend server.

4. The API server is now running at http://127.0.0.1:5001 and frontend server is running at http://127.0.0.1:5000. Open a new tab in your browser and hit respective link to use the application.

Challenges we faced: Lessons

- **Time Management -** Navigating through project timelines, team members' schedules, and academic commitments to meet project milestones. Addressing challenges such as quiz schedules and balancing academic work with project deadlines.

- **Clear Vision -** Establishing a clear project vision that aligns with the problem and utilizes the team's expertise. This involves defining achievable goals and objectives for everyone to understand and work towards.

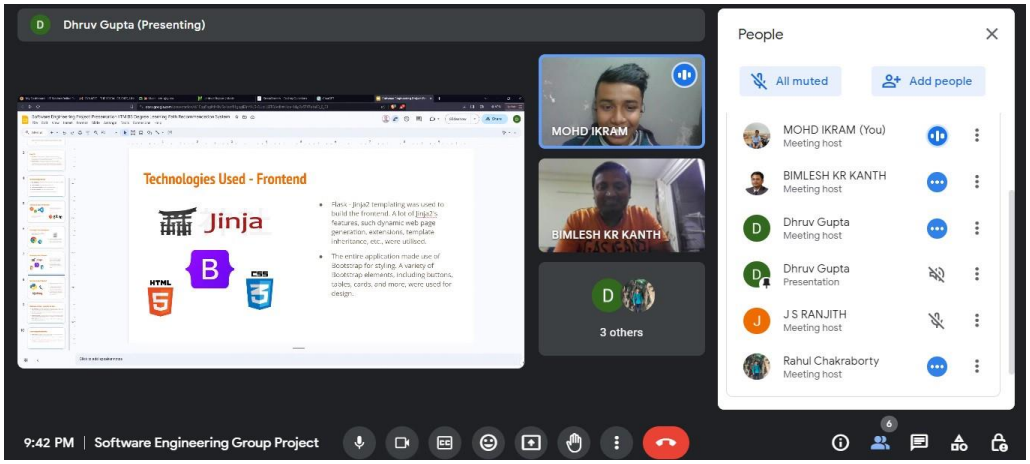
- **Scheduling Coordination -** Facilitating regular team meetings at a specified time remained a persistent challenge. Despite fixing a daily meeting time, coordinating the schedules of all team members proved to be a continuous effort.

- **GitHub Collaboration Issues -** Encountering accessibility challenges on GitHub while collaborating on the project. Overcoming obstacles related to version control and ensuring a smooth workflow for all team members.

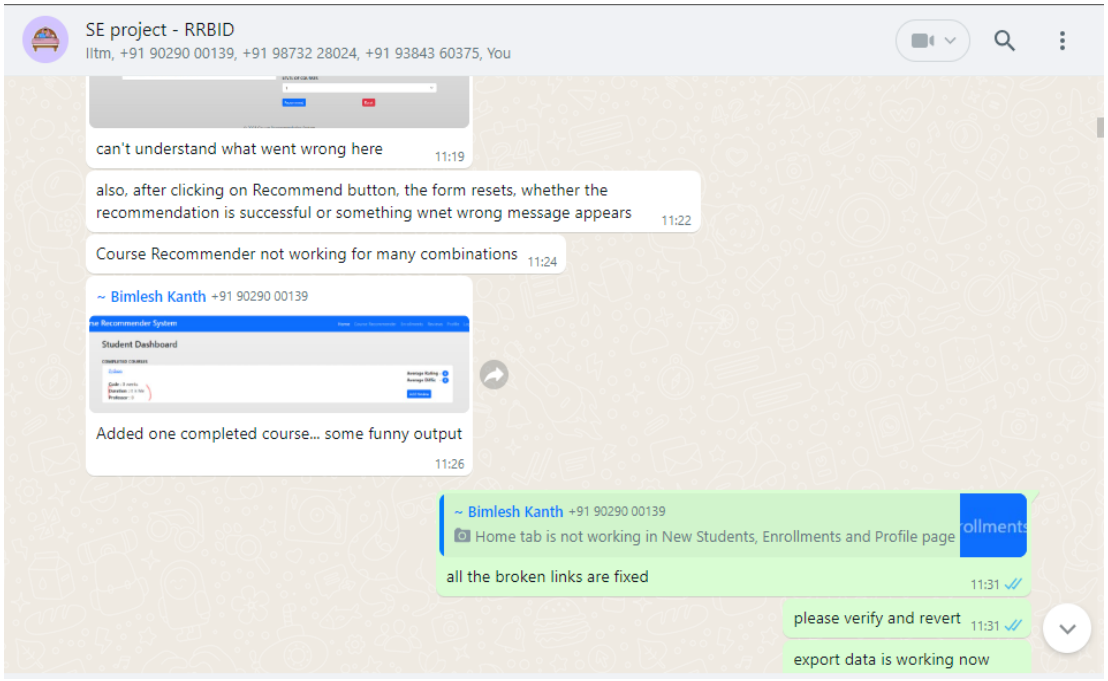
Code Review & Issue Tracking Screenshots

The following screenshots demonstrate the various code review, issue reporting, and issue tracking actions that we've undertaken.

Google meet calls



WhatsApp chats:



Thank You

Team-07 Sept 2023