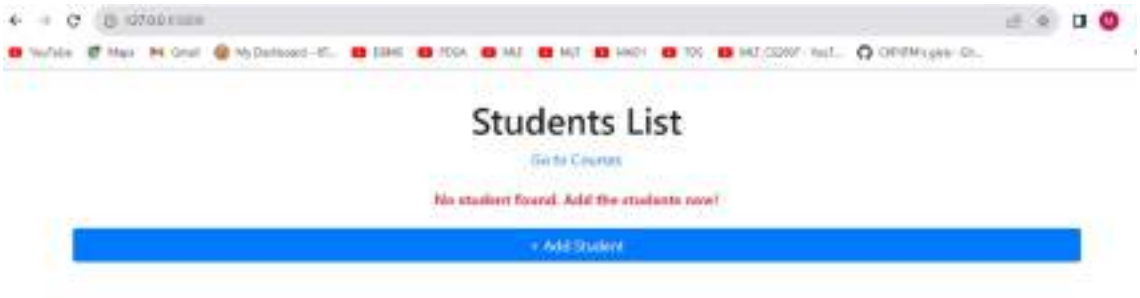
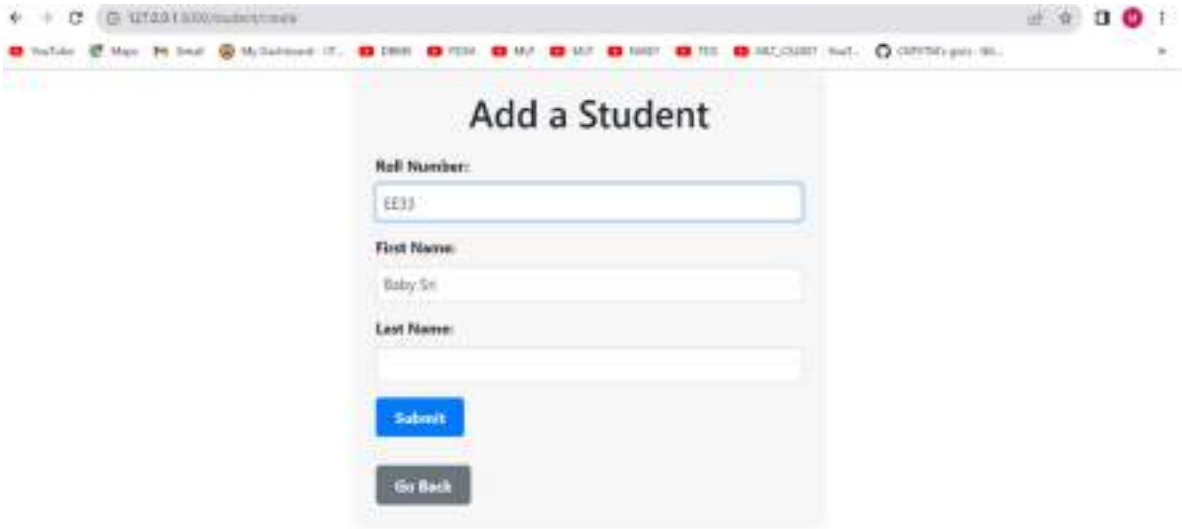
Using a standard flask template, create an application that:

CRUD Operations for Students:

1. On the home page (URI = '/'), (when we open it via the browser) displays an index page. The index page displays a table with the list of currently existing students in the database and displays an appropriate message if no student exists. It also consists a button labeled "Add student".



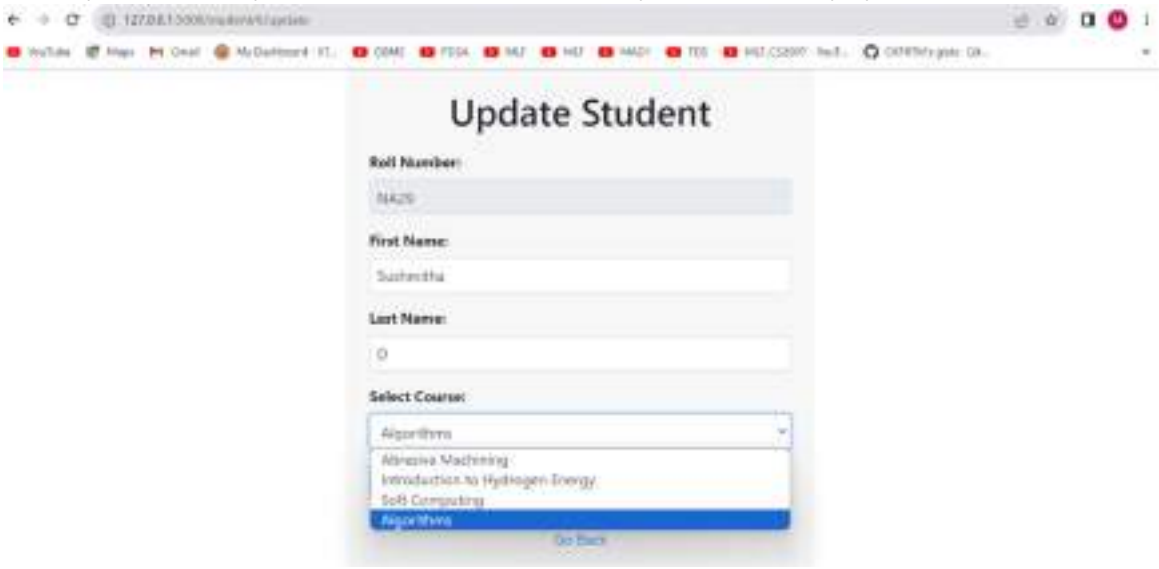
2. If the user clicks the "Add student" button, the flask application will send a GET request to an endpoint "/student/create", which displays an HTML form.



3. If the user clicks the submit button, the browser sends a POST request to flask application's "/student/create" URI. The flask application then creates a student object (with attributes roll number, first name and last name) and add it into the database and, it redirects to the home page (URI = '/') and the student will be added into the table. The roll number in each row of the table is clickable.

4. If the roll number already exists, then, the user will be redirected to an HTML page, which displays an appropriate message and have a button to navigate back to the home page (URI = '/')

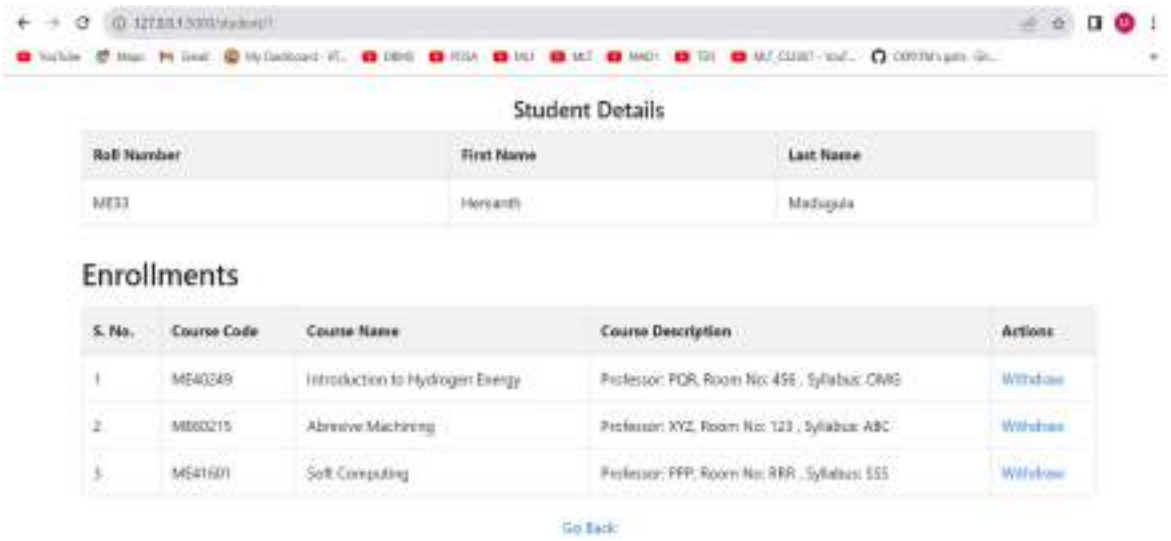
5. If the user clicks the "Update" button on the home page (URI = '/'), the flask application sends a GET request to an endpoint "/student/<int:student id>/update", which displays an HTML form.



6. If the user clicks the submit button, the browser sends a POST request to your flask application's "/student/<int:student id>/update" URI. The flask application then updates the student and corresponding enrollment into the database and redirect to the home page (URI = '/'). The previous enrollment(s) will persist. (if any)

7. If the user clicks the "Delete" button on the home page (URI = '/'), the flask application sends a GET request to an endpoint "/student/<int:student id>/delete", which deletes the student and all the corresponding enrollments from the database and redirected to the home page (URI = '/').

8. If the user clicks on the roll number of any row in the table in the home page of the flask application, the application sends a GET request to an endpoint “/student/<int:student id>”, which shows all the information (student details and enrollment details) in an HTML page. The HTML page also have a button labelled “Go Back” to navigate back to the home page (URI = ‘/’). There are 2 HTML tables in this page, one for showing the personal details and the other for displaying the enrollment details. Every record in the enrollments table have a “withdraw” button, which when clicked, the flask application sends a GET request to an endpoint “/student/<int:student id>/withdraw/<int:course id>”, which removes the course from current enrollments of the student from the database and redirected to the home page (URI = ‘/’).



Student Details

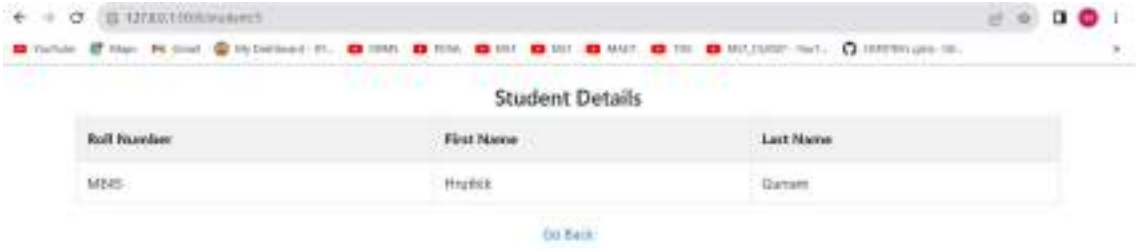
Roll Number	First Name	Last Name
ME33	Hemanth	Madugula

Enrollments

S. No.	Course Code	Course Name	Course Description	Actions
1	ME40249	Introduction to Hydrogen Energy	Professor: POR, Room No: 456 , Syllabus: OMG	Withdraw
2	MI660215	Abrasive Machining	Professor: XYZ, Room No: 123 , Syllabus: ABC	Withdraw
3	ME41601	Soft Computing	Professor: PPP, Room No: RRR , Syllabus: SSS	Withdraw

Go Back

9. This table only exists when there is at least 1 enrollment for that particular student. The table must not exist if there are no enrollments.



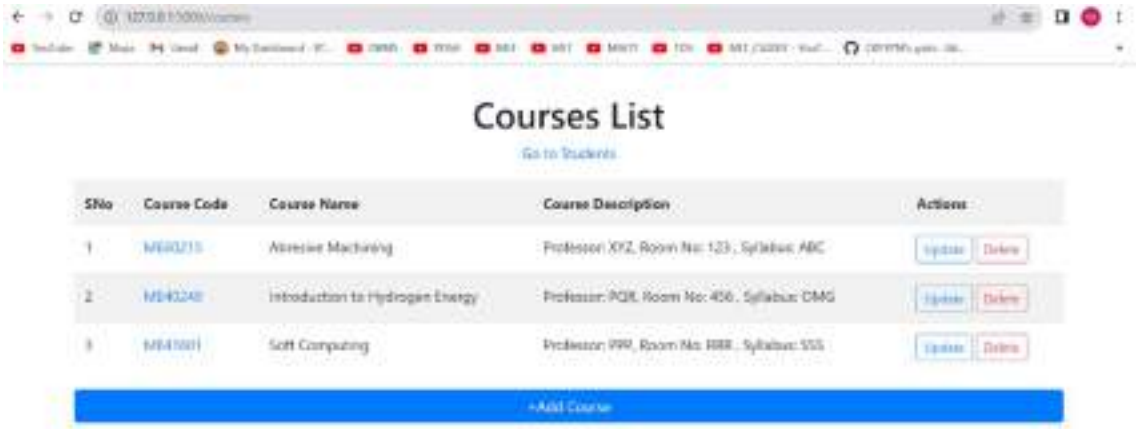
Student Details

Roll Number	First Name	Last Name
ME45	Hrishik	Ganam

Go Back

CRUD Operations for Courses:

1. The index page also consists a button “Go to courses”, When a user clicks on “Go to Courses”, the browser sends a GET request to your flask application’s “/courses” URI and the application navigate the user to the courses page, which displays a table of all the courses currently available in the database. It displays an appropriate message if no course exists in the database. It also consists a button labeled “Add course” . The HTML page also have a button “Go to Students”. When a user clicks on “Go to Students”, the browser sends a GET request to your flask application’s “/” URI and the application navigate the user to the index page of the application.



2. If the user clicks the “Add course” button, flask application sends a GET request to an endpoint “/course/create”, which displays an HTML form. If the user clicks the submit button, the browser sends a POST request to your flask application’s “/course/create” URI. The flask application then create a course object (with attributes course code, course name and course description) and add it into the database and, it should redirect to the courses page (URI = ‘/courses’).Note that the course code in each row of the table can be clickable.

3. If the course code already exists, then, the user should be redirected to an HTML page, which should display an appropriate message and have a button to navigate back to the courses page (URI = '/courses')

4. If the user clicks the "Update" button on the courses page (URI = '/courses') flask application sends a GET request to an endpoint "/course/<int:course id>/update", which displays an HTML form. If the user clicks the submit button, the browser sends a POST request to your flask application's "/course/<int:course id>/update" URI. The flask application then updates the targeted course (with the given course id in the URI) in the database and redirect to the courses page (URI = '/courses').

5.If the user clicks on the course code of any row in the table in the courses page of the flask application (URI = '/courses'), the application sends a GET request to an endpoint "/course/<int:course id>", which shows all the information (course details and students enrolled in the course) in an HTML page. The HTML page also consists a button labelled "Go Back" to navigate back to the courses page (URI = '/courses'. There must be 2 HTML tables in this page, one for showing the course details and the other for displaying the enrollment details.

Created a RESTful API, database models using Flask-RESTful and flask-SQLAlchemy, Postman/Thunderclient

Course Table Schema

Column Name	Column Type	Constraints
course_id	Integer	Primary Key, Auto Increment
course_name	String	Not Null
course_code	String	Unique, Not Null
course_description	String	

Student Table Schema

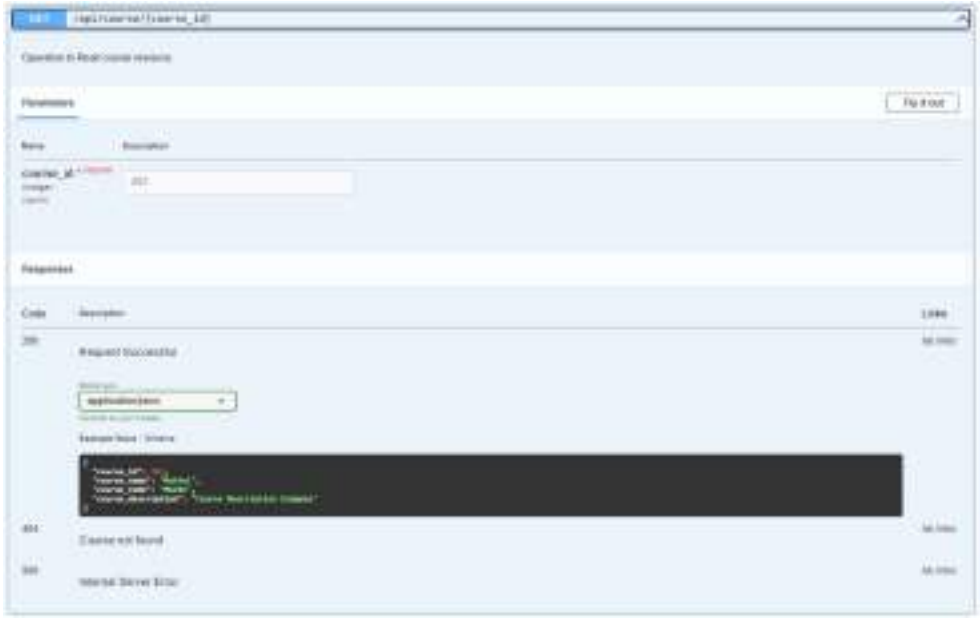
Column Name	Column Type	Constraints
student_id	Integer	Primary Key, Auto Increment
roll_number	String	Unique, Not Null
first_name	String	Not Null
last_name	String	

Enrollment Table Schema

Column Name	Column Type	Constraints
enrollment_id	Integer	Primary Key, Auto Increment
student_id	Integer	Foreign Key (student.student_id), Not Null
course_id	Integer	Foreign Key (course.course_id), Not Null

Error Codes

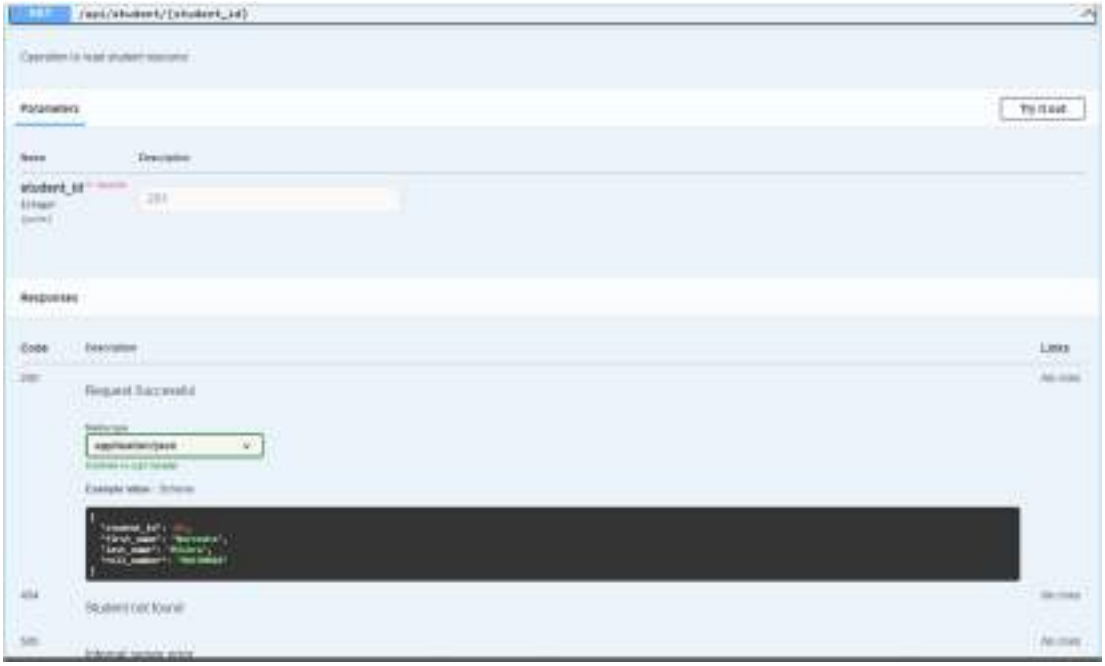
Resource	Error Code	Message
Course	COURSE001	Course Name is required
Course	COURSE002	Course Code is required
Student	STUDENT001	Roll Number required
Student	STUDENT002	First Name is required
Enrollment	ENROLLMENT001	Course does not exist
Enrollment	ENROLLMENT002	Student does not exist

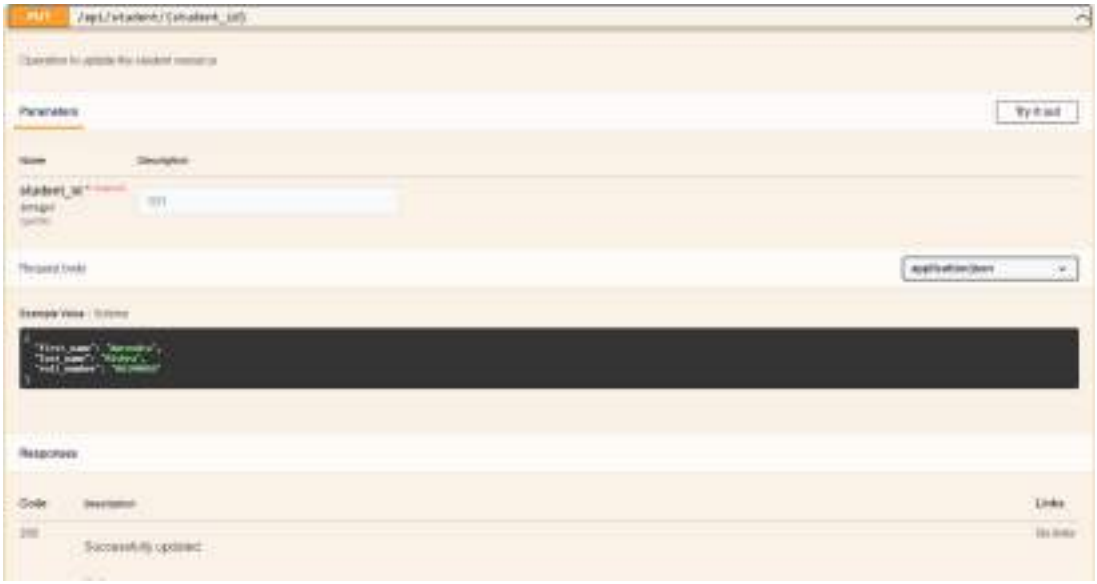


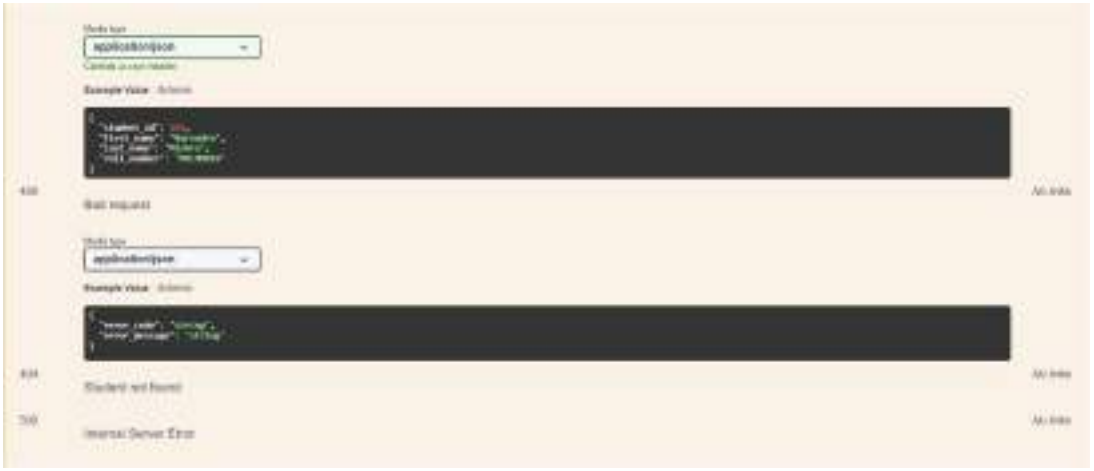

DELETE
/api/course/{course_id}
Parameters
Try it out
Name
Description
course_id
Responses
Code
Description
Links
200
Successfully Deleted
No links
404
Course not found
No links
500
Internal Server Error
No links


POST
/api/course
Parameters
Try it out
No parameters
Request body
application/json
Example Value
Schema
{
"course_name": "Maths",
"course_code": "MA101",
"course_description": "Course description example"
}
Responses
Code
Description
Links
201
Successfully Created
No links
Media type
application/json
Controls Accept header.


Example Value
Schema
{
"course_id": 0,
"course_name": "Maths",
"course_code": "MA101",
"course_description": "Course description example"
}
400
Bad request
No links
Media type
application/json
Example Value
Schema
{
"error_code": "string",
"error_message": "string"
}
409
course_code already exist
No links
500
Internal Server Error
No links



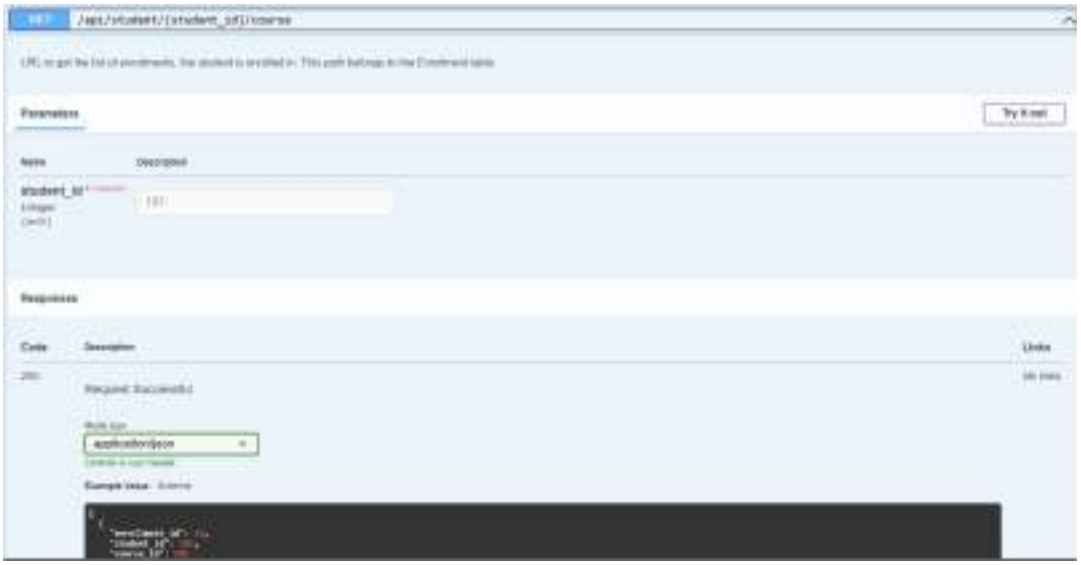


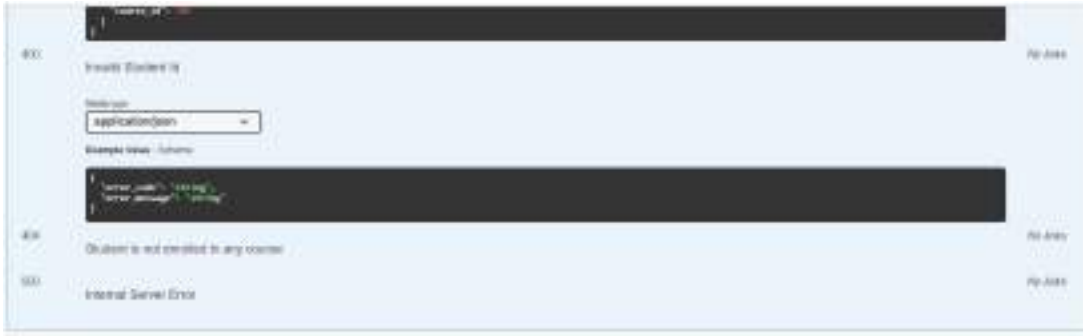


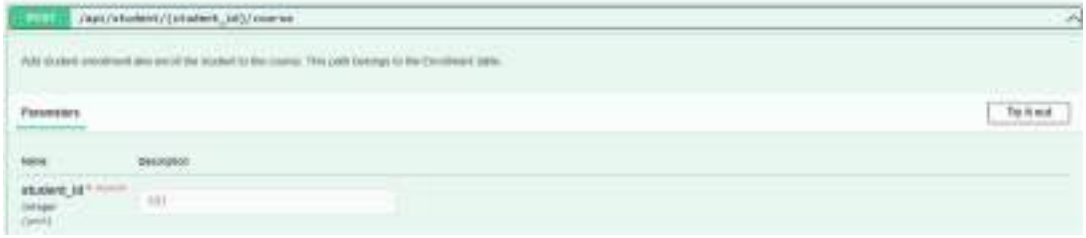

DELETE
/api/student/{student_id}
Parameters
Try it out
student_id
Responses
Code
Description
Links
200
Successfully Deleted
No links
404
Student not found
No links
500
Internal Server Error
No links

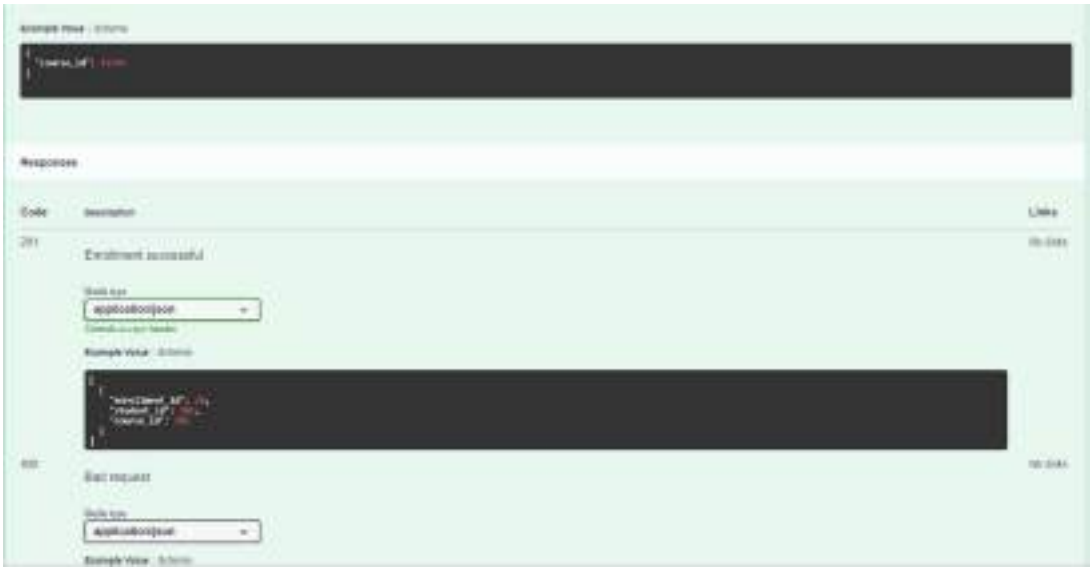

POST
/api/student
Parameters
Try it out
No parameters
Request body
application/json
Example Value
Schema
Responses
Code
Description
Links
201
Successfully Created
Media type
application/json

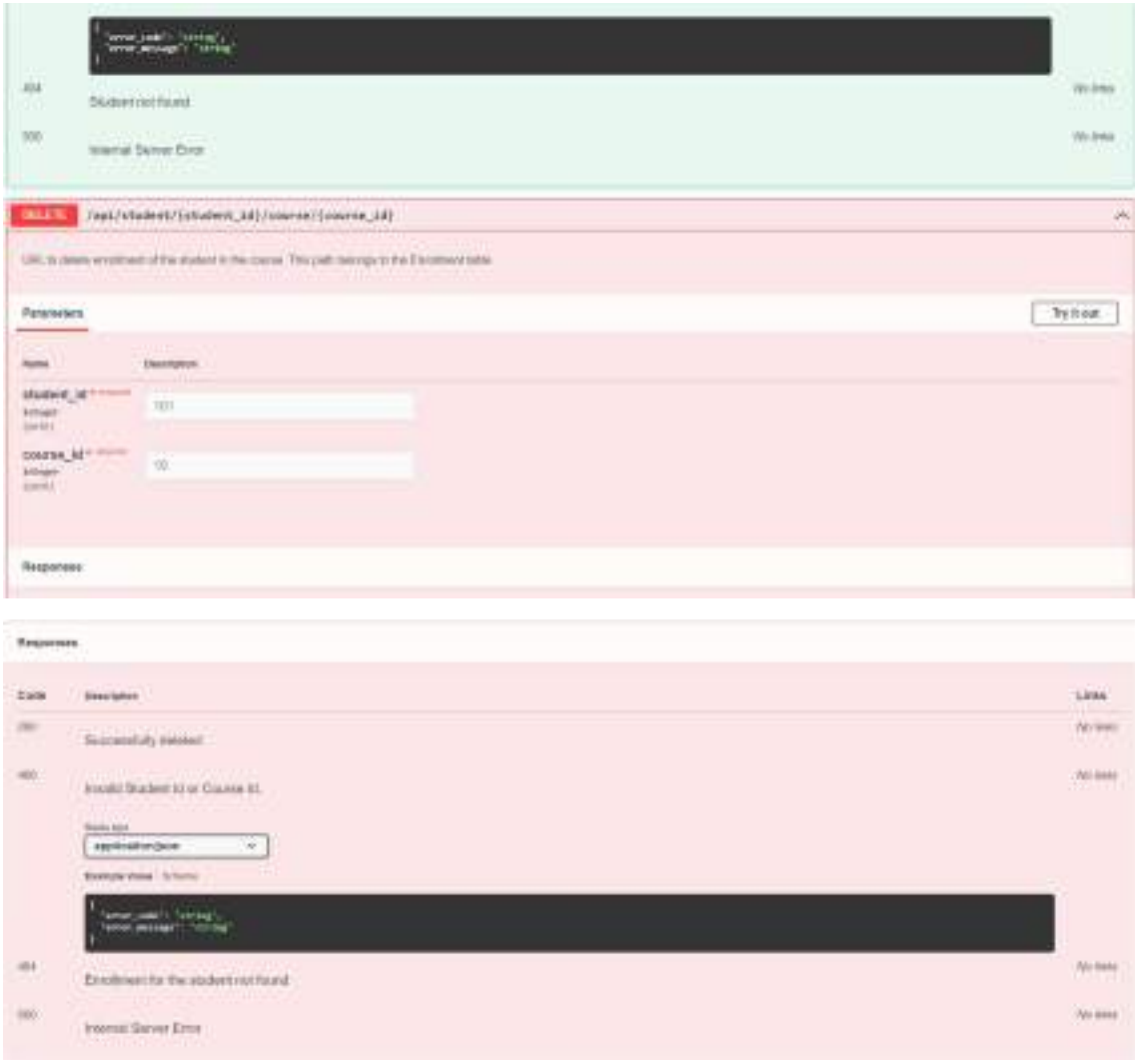

Example Value
Schema
400
Bad request
Media type
application/json
Example Value
Schema
409
Student already exist
500
Internal Server Error


/api/student/{student_id}/course
Parameters
Try it out
Name
Description
student_id
Responses
Code
Description
Links
200
Request successful
application/json
Example Value
Schema


400
Invalid Student Id
application/json
Example Value
Schema
404
Student is not enrolled in any courses
500
Internal Server Error


POST
/api/student/{student_id}/course
Add student enrollment details to the student to the course. This path belongs to the Enrollment table.
Parameters
Try it out
Name
Description
student_id
integer
(path)


Example Value
Schema
Responses
Code
Description
Links
201
Enrollment successful
application/json
Example Value
Schema
400
Bad request
application/json
Example Value
Schema



Reference:

https://onlinedegree.gitlab.io/mad1/week_six_openapi/#/