CMH Challenge

Submitted By: Pavan Prabhakar Bhat (pxb8715@rit.edu)

Documentation:

Software Requirement

Software required to run the project:

1. Python 3.5 or above
2. curl (For accessing REST endpoints through command-line)

IDE used:

1. PyCharm 2016 or above

Support Libraries needed in Python:

1. flask
2. flask_bootstrap
3. flask_mysqldb

*For Windows and Linux users make sure you have pip installed before you can install the required support libraries.

Installation

First **install Python 3.x**

Windows:

First download and install the Python 3.5 or above from https://www.python.org/downloads/release/python-352/

Linux (Ubuntu):

First download and install the Python 3.5 or above from https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-ubuntu-16-04

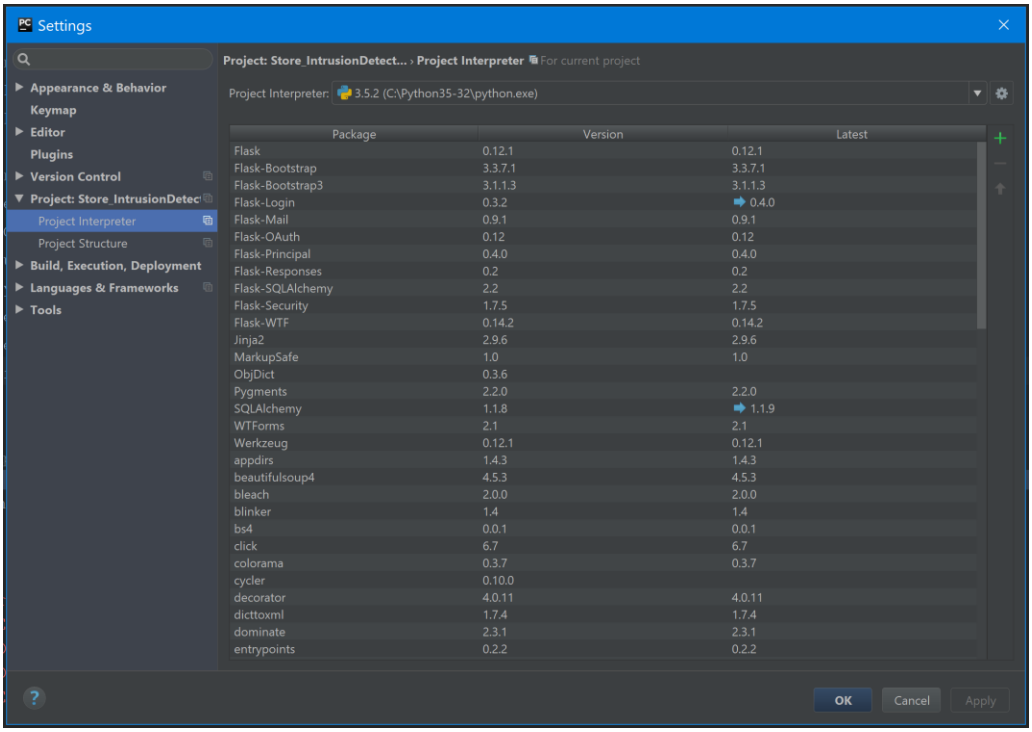
(Optional Step: But highly recommended for convenience)

Next, **install PyCharm** which is freely available from the link given below.

https://www.jetbrains.com/pycharm-edu/quickstart/installation.html

A descriptive installation steps are provided in the above link for both Windows, Mac and Linux users.

Once Pycharm has been installed load the project, Next, Visit File > Settings. The following window will pop up.

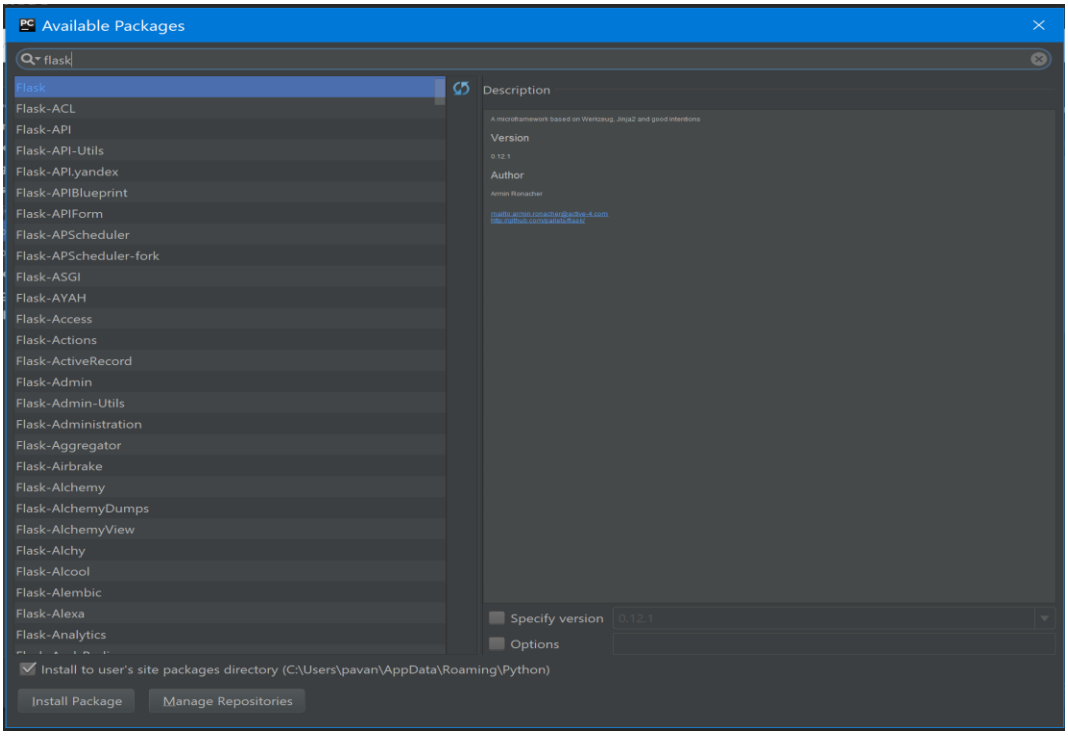


- If **PyCharm is not installed** and this project needs to be directly run through python then for each of the support packages or libraries needed would have to be added through a pip installation. (pip needs to be installed for this, installation link on the left)

  For instance, use the command in the following way:

```
pip install flask
```

Next choose the appropriate Interpreter and install the support libraries listed by clicking the green plus sign on the right. For illustration, use the image shown below.



Type the name of the support package and click on install package. Once all packages have been installed. Click on the apply option.

Next, **install cURL** which is freely available from the link given below.

https://curl.haxx.se/download.html

This will allow us access to the REST end-points through command-line in windows.

Usage

After successful installation,

Load the Project in PyCharm and Run the **student_enrollment.py** file on PyCharm.

If PyCharm is not installed: Go to the source folder directly through command-line where the files are located and Run the file using the following command,

```
python student_enrollment.py
```

Then visit the following url: http://127.0.0.1:5000/

REST end points corresponding to each requirement:

1. GET All courses → http://127.0.0.1:5000/courses
2. GET All majors → http://127.0.0.1:5000/majors
3. GET All students → http://127.0.0.1:5000/students
4. GET Courses available to a student →

   http://127.0.0.1:5000/courses/students/<student_id>

   Here, **<student_id>** should be an integer corresponding to id in the Student table. For e.g.,

   http://127.0.0.1:5000/courses/students/1

5. GET Students enrolled in a course →

   http://127.0.0.1:5000/students/courses/<course_id>

   Here, **<course_id>** should be an integer corresponding to id in the Courses table. For e.g.,

   http://127.0.0.1:5000/students/courses/1

6. POST Student enrollment into a valid course
   Condition:
   - A student can only be enrolled into a course associated with their major
   - A student can only be enrolled once in a course

   http://127.0.0.1:5000/students/enroll/

   Here, while attaching the payload **<student_id>** should be an integer corresponding to id in the Student table and the **<course_id>** should be an integer corresponding to the id of a Course the student wants to enroll into. For e.g., the payload needs to be in the following format:

   ```
   {"Student_id":1,"Course_id":5}
   ```

   This method can't be used on an address bar directly and must be done using command-line through cURL.

Each of these end points can be accessed through command-line using curl in the following manner:

Use the keyword curl in the prompt followed by the URL of the REST end-point
> **curl** http://localhost:5000/students

NOTE: localhost and 127.0.0.1 can be used interchangeably but the version of curl I installed requires commands to be given in a certain format or else it would raise exceptions. The following are a few illustrations of the above-mentioned commands.

1. GET All courses > `curl http://127.0.0.1:5000/courses`

```
Command Prompt
Microsoft Windows [Version 10.0.16179]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\pavan>curl http://127.0.0.1:5000/courses
[{"course": {"cname": "Design Patterns", "id": 1, "major_id": 1}}, {"course": {"cname": "Software Architecture", "id"
: 2, "major_id": 1}}, {"course": {"cname": "Rich Media", "id": 3, "major_id": 2}}, {"course": {"cname": "Data Visuali
zation", "id": 4, "major_id": 2}}, {"course": {"cname": "Compilers", "id": 5, "major_id": 3}}, {"course": {"cname": "
Data Structures", "id": 6, "major_id": 3}}, {"course": {"cname": "Web & Mobile", "id": 7, "major_id": 4}}, {"course":
 {"cname": "User Experience", "id": 8, "major_id": 4}}]
C:\Users\pavan>
```

2. GET All majors > `curl http://127.0.0.1:5000/majors`

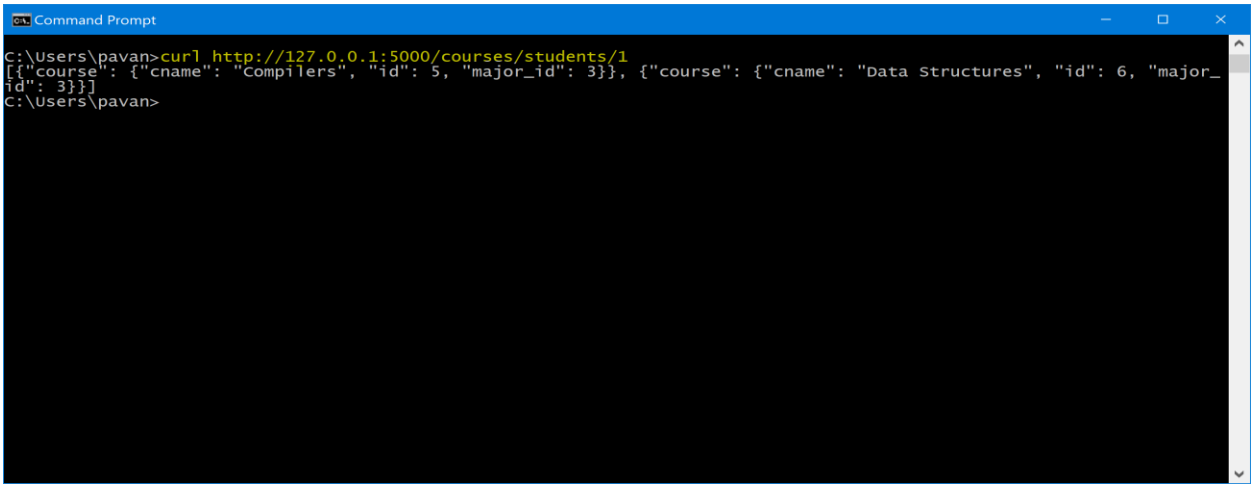
```
Command Prompt

C:\Users\pavan>curl http://127.0.0.1:5000/majors
[{"major": {"id": 1, "mname": "Software Engineering"}}, {"major": {"id": 2, "mname": "New Media"}}, {"major": {"id":
3, "mname": "Computer Science"}}, {"major": {"id": 4, "mname": "Information Technology"}}]
C:\Users\pavan>
```

3. GET All students > `curl http://127.0.0.1:5000/students`

```
Command Prompt

C:\Users\pavan>curl http://127.0.0.1:5000/students
[{"student": {"fname": "Vidit", "id": 1, "lname": "Organisciak", "major_id": 3}}, {"student": {"fname": "Kenneth", "i
d": 2, "lname": "Kenny", "major_id": 3}}, {"student": {"fname": "Ian", "id": 3, "lname": "Olbrich", "major_id": 4}},
{"student": {"fname": "Joey", "id": 4, "lname": "LaMarca", "major_id": 4}}, {"student": {"fname": "Nick", "id": 5, "l
name": "Eckert", "major_id": 2}}, {"student": {"fname": "Keila", "id": 6, "lname": "Oliva", "major_id": 2}}, {"studen
t": {"fname": "Kristen", "id": 7, "lname": "Beatty", "major_id": 2}}, {"student": {"fname": "Jason", "id": 8, "lname"
: "Singhal", "major_id": 1}}, {"student": {"fname": "Brenden", "id": 9, "lname": "Minnoe", "major_id": 1}}, {"student
": {"fname": "Lexi", "id": 10, "lname": "Greaves", "major_id": 1}}, {"student": {"fname": "GG", "id": 11, "lname": "W
P", "major_id": 3}}]
C:\Users\pavan>
```

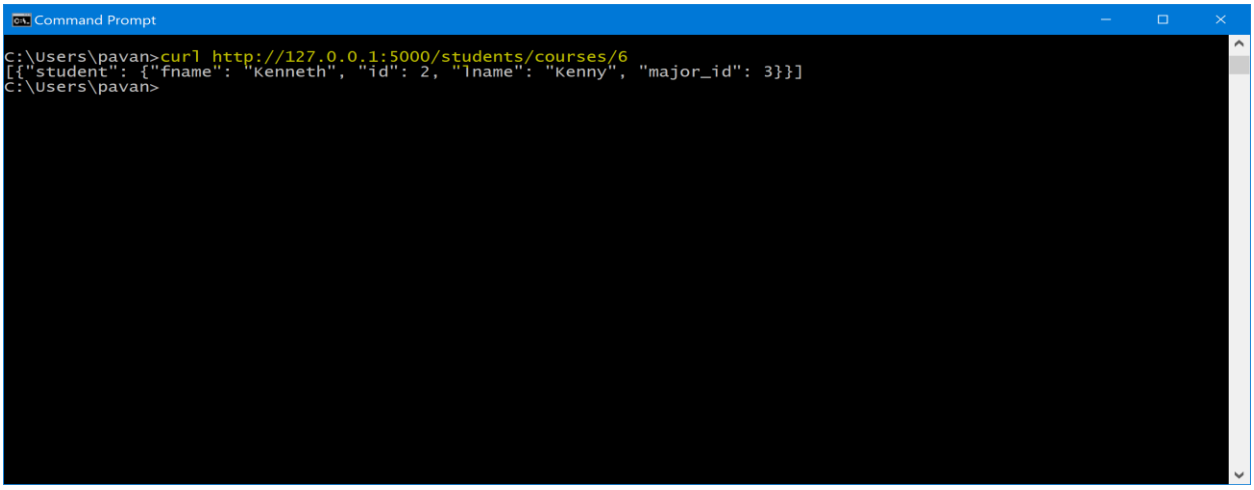
4. GET Courses available to a student

```
> curl http://127.0.0.1:5000/courses/students/1
```



5. GET Students enrolled in a course

```
> curl http://127.0.0.1:5000/students/courses/6
```



6. POST Student enrollment into a valid course

```
> curl -i -H "Content-Type: application/json" -X POST -d
"{"""Student_id""":1,"""Course_id""":5}"
http://127.0.0.1:5000/students/enroll
```

Here,
-i : Information on the POST operation
-H : Headers for the POST operation (it is important to specify the type of content being attached as payload which is JSON)
-X : type of operation i.e. POST
-d : Payload / Data (Here the format of the JSON object or array to be used should be as shown above. This is to ensure that the Key used as Student_id is a String and so needs to be used in triple double quotes.

```
Command Prompt
Microsoft Windows [Version 10.0.16184.1001]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\pavan>curl -i -H "Content-Type: application/json" -X POST -d "{"""Student_id""":1,"""Course_id""":5}" http://lo
calhost:5000/students/enroll
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 17
Server: Werkzeug/0.12.1 Python/3.5.2
Date: Thu, 04 May 2017 00:37:09 GMT

Student Enrolled!
C:\Users\pavan>
```

7. POST Student enrollment into the same course twice (other cases)

```
Command Prompt

C:\Users\pavan>curl -i -H "Content-Type: application/json" -X POST -d "{"""Student_id""":1,"""Course_id""":5}" http://lo
calhost:5000/students/enroll
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 44
Server: Werkzeug/0.12.1 Python/3.5.2
Date: Thu, 04 May 2017 01:20:16 GMT

Student already enrolled in the same course!
C:\Users\pavan>
```

8. POST Student enrollment into an already added course (other cases)

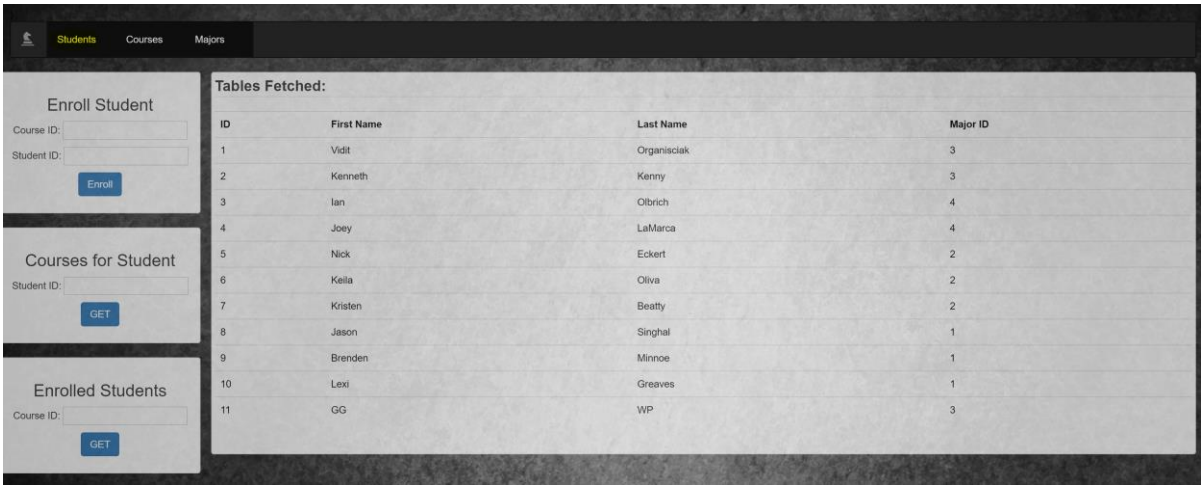
```
Command Prompt

C:\Users\pavan>curl -i -H "Content-Type: application/json" -X POST -d "{"""Student_id""":1,"""Course_id""":6}" http://lo
calhost:5000/students/enroll
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 39
Server: Werkzeug/0.12.1 Python/3.5.2
Date: Thu, 04 May 2017 01:23:19 GMT

Student already enrolled in one course!
C:\Users\pavan>
```

9. POST Student enrollment into an unknown course

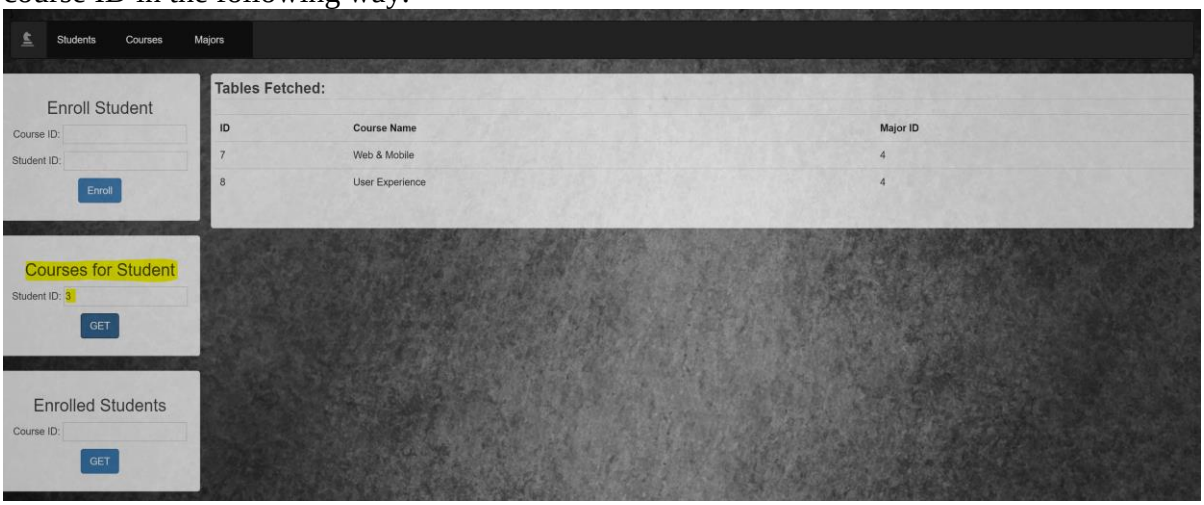
Additional Feature: Front-End Interface

This interface is hosted at http://127.0.0.1:5000/ and can be accessed once the **student_enrollment.py** file is run. By clicking on **Students**, **Courses** or **Majors** on the menu will fetch the respective tables in the "**Tables Fetched**" area.

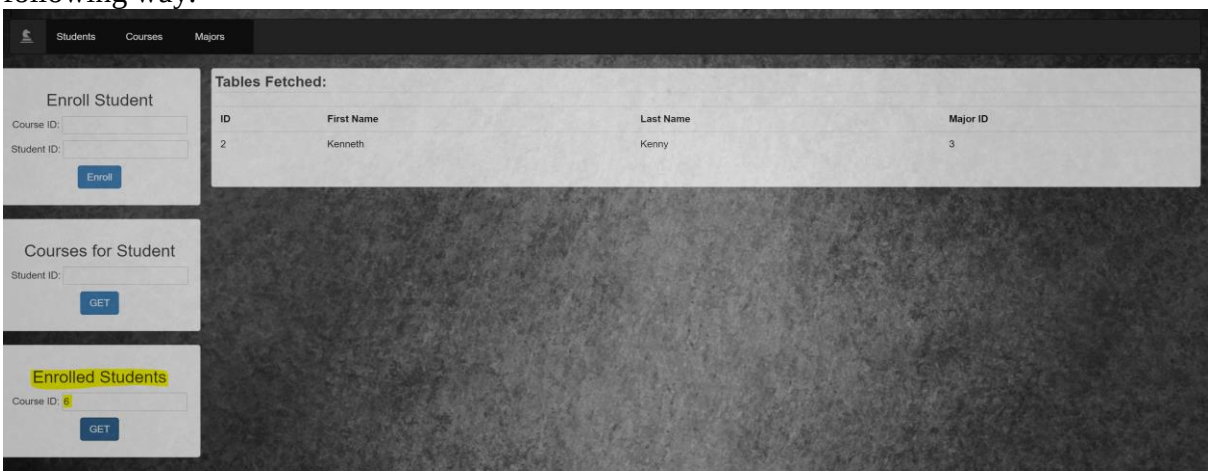
For illustration, on clicking Students tab we get the following:



Similarly, in order to get courses corresponding to a given student's major, enter a valid course ID in the following way:



Now, in order to get the enrolled students in a given course enter a course number in the following way:



Finally, the POST operation can be performed by entering a valid student ID and a valid course ID for the student who wants to enroll into in the following way:

High Level Approach

The database has been hosted on Amazon RDS using MariaDB (MySql) and the data at rest can be accessed through different REST end-points using various clients such as an Internet browser or a command-line interface using cURL or an app like Advanced REST Client (ARC). Using these clients, data can be either retrieved through a GET operation or can be posted through a POST operation which would in turn query the database to obtain the required data. In order to facilitate these services, I have used Flask to manage the server side code in python. Using flask routing from one REST end-point to the other has become feasible and easy to use. Running the flask application loads the server and launches the service which can then be served to the clients using methods discussed above. Each module is independent and has been designed in a modular fashion for effective usage. One of the features of using a payload (data in JSON format) while posting data is that it also supports multiple student enrollment at one go. Each operation performed is bound by a request-response cycle.

During a GET operation => Data flows from the Database deployed on AWS to Client.

During a POST operation => Data flows from the Client to the Database deployed on AWS and receives an acknowledgement once the data is successfully posted.

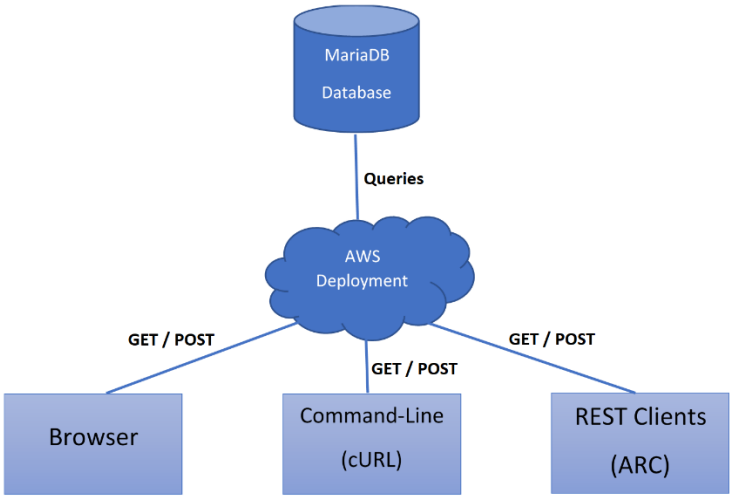


Fig: Software Architecture with clients and services in use

Additional Feature: I have also created a front-end interface using HTML5, CSS3, Bootstrap, jQuery, AJAX for a better user interaction with the inline service. This would help the server side processing to be abstracted for the user and only display whatever operation the user would like to perform in order to perform student enrollment.