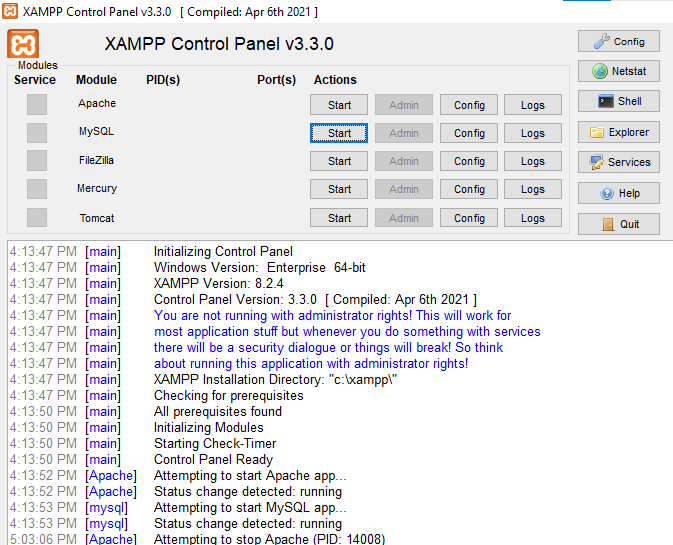
**Purpose of web app:**

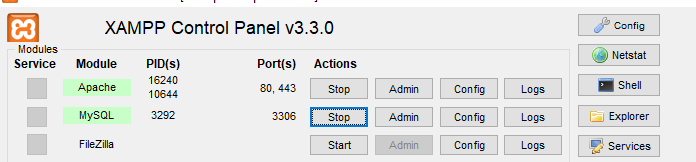
The purpose of the given assignment was to créate a new full stack web application using Python framework Flask using MySQL database. The app should be able to perform all CRUD operations.All Steps are listed below to perform CRUD operation in Flask framework of Python using MySQL database.

**Setting the ZAMPP server for database management:**

1. First and foremost, before moving to the Flask, we need the local web server for MySQL database management. Download the XAMPP server depending upon your machine and install it. It has username = root, and no password set by default. After Installing, following screen will be open when we open the XAMPP.



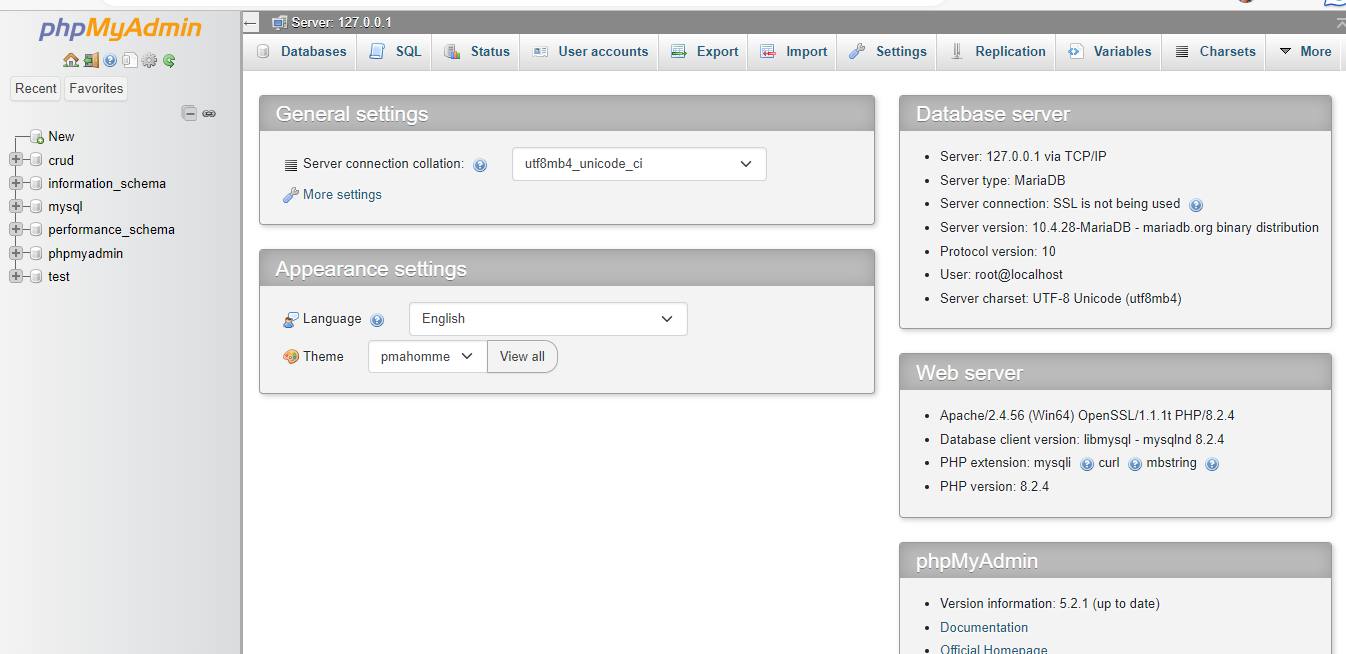
1. After step 01, and whenever we will be interacting with our web page backed in Flask, we always click on the start button of Apache and MySQL module and then will minimize the screen. If these modules are not running, our web app will not work as MySQL database will not get access when these are stopped. When we click these modules, they turned green and port no will be assigned to them as follow.



1. After this, for creating the database in MySQL and managing it, we’ve to open phpMyAdmin of local host. Open with following address in browser.

<http://localhost/phpmyadmin>

Following screen will show to you.



1. In the top-left side, click on New to add new database named “crud”. And in crud database, add table of students having 4 columns. i.e. id, name, email, phone.

**CRUD operation in Flask:**

Before going to perform CRUD operations in Flask, we’ve to install some python libraries to backend the web app with flask, database connection, and testing using pip.

Running following command in your terminal:

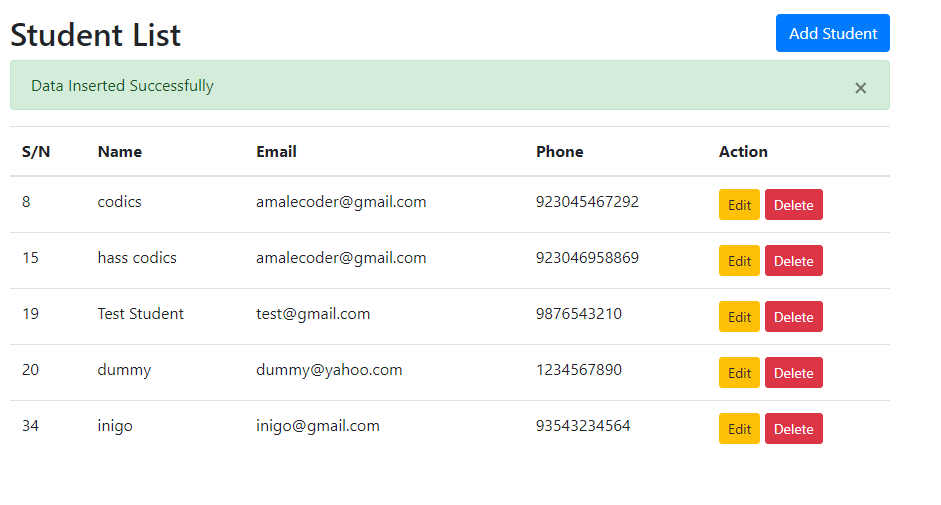
1. pip install Flask
2. pip install flask-mysqldb

Everything is explained in app.py file where CRUD operations are being performed on students table created in step 04. Then we’ve index.html that has code for web page where user can read, add, update, and delete student’s record.

Here is the web page.



Adding student;

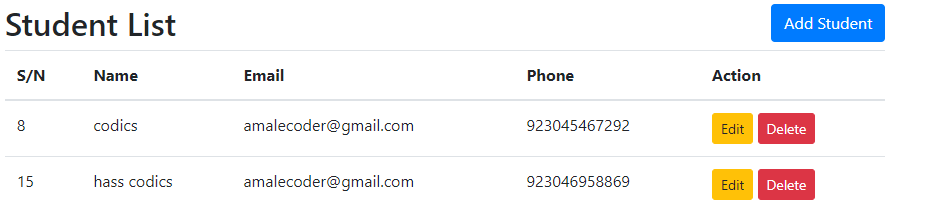


Updating student record;

A screenshot of a computer

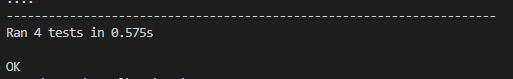
Description automatically generated

Deleting student record;

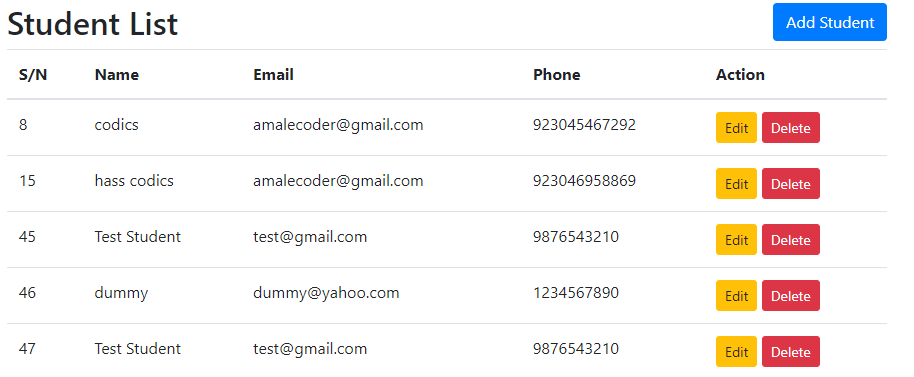


**Creating tests in tests.py**

In tests.py, I’ve created different tests to make sure that all opertions in CRUD are being performed well. For this purpose, I’ve used unit testing framework that test every function that either they work well or not. These inludes test\_create\_student to handle for adding new student, test\_read\_students to read all the students in database, test\_update\_student to update existing record of student, and test\_delete\_student to delete the record of student from database. Here is the results of tests functions after running tests.py file.



After this, we will check that either the changes reflected in our database to make sure that our tests are correct.



**Anti-hacking security measures:**

Implementing anti-hacking security measures is crucial to protect the web app from potential security vulnerabilities and attacks.

Below are the security measures that are added in my code:

* **SQL Injection Prevention**
* **Cross-Site Scripting (XSS) Prevention**

**SQL Injection Prevention:**

SQL injection is a common hacking technique where attackers inject malicious SQL code into the application's input fields to manipulate the database. To prevent SQL injection, I used parameterized queries when interacting with the database. In the code provided, the usage of parameterized queries can be seen in the **insert** and **update** functions as below:

**cur.execute("INSERT INTO students (name, email, phone) VALUES (%s, %s, %s)", (name, email, phone))**

**cur.execute("UPDATE students SET name=%s, email=%s, phone=%s WHERE id=%s", (name, email, phone, id\_data))**

**Cross-Site Scripting (XSS) Prevention:**

XSS is an attack that injects malicious scripts into web pages, allowing attackers to steal sensitive information. To prevent XSS attacks,I ensure that user input is properly validated, sanitized, and escaped before displaying it on web pages. In the provided code, Jinja2 templating engine automatically escapes the data from the database, making it safe for rendering in HTML templates.

For example:

                    {% for row in students %}

                    <!-- Loop through students and display their information in the table -->

                    <tr>

                        <td>{{ row.0 }}</td>

                        <td>{{ row.1 }}</td>

                        <td>{{ row.2 }}</td>

                        <td>{{ row.3 }}</td>

                        <td>