BSCCS2003: Week-7 Lab Assignment

This assignment is an extension of Week 5 lab assignment, where your web application must allow CRUD operations for student as well as on the courses with a database model, using flask and flask-SQLAlchemy. We list below the instructions to be followed in preparing and submitting the solution.

General instructions:

- Submit a single .zip file containing all your submission files and folders, the name of which should be "<roll_number>.zip". E.g.: *21f1000000.zip*
- The folder structure inside the zip file should be as follows:
  - The Python program must be written inside a file named "app.py". This file must reside inside the root directory.
  - All the HTML files should be kept inside a folder named "templates" and the images and CSS files (if any) should be kept in "static" folder. Both the "static" and "templates" folder must reside in the root directory.
  - The database file named "week7_database.sqlite3". You are not required to submit this database file with your submission.
- All the endpoints must return 200 as the status code in the case of success.
- You should not keep any code inside the scope of the condition " if __name__ == '__main__' " except run() call.
- You should not use 'create_all()' call in your program, as we will be using a different database in the backend, and using any such call can affect the evaluation.
- Allowed Python packages: jinja2, flask, flask-sqlalchemy, or any standard Python3 package.
- The output pages should be HTML5 compliant, e.g., the file should begin with the declaration <!DOCTYPE html>.

Problem Statement:

- You have to create a database model and must name the database file as "week7_database.sqlite3". The database model must have 3 tables, for which the schema is given below.

- The database URI must be the same as given below:
  app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///week7_database.sqlite3'
  Note: The table names and column names must be the same as given below.

Table 1: student

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	

Table 2: course

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

Table 3: enrollments

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

Using a standard flask template, create an application that: CRUD Operations for Student

1. On the home page (URI = '/'), (when we open it via the browser) displays an index page. The index page must display a table with the list of currently existing students in the database. The HTML table should be the same as given below (Its ID must be "all-students"). It should display an appropriate message if no student exists. It should also have a button labeled "Add student" as shown in the Figure 1.

```
<table id = "all-students">
    <tr>
        <th>SNo</th>
        <th>Roll Number</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Actions</th>
    </tr>
    <tr>
        <td>s_no_1</td>
        <td><a href="/student/<int:student_id>">roll_number_1</a></td>
        <td>first_name_1</td>
        <td>last_name_1</td>
        <td>
          <a href="/student/<int:student_id>/update">Update</a>
          <a href="/student/<int:student_id>/delete">Delete</a>
        </td>
    </tr>
        ......
        ......
        ......
    <tr>
        <td>s_no_n</td>
        <td><a href="/student/<int:student_id>">roll_number_n</a></td>
        <td>first_name_n</td>
        <td>last_name_n</td>
        <td>
          <a href="/student/<int:student_id>/update">Update</a>
          <a href="/student/<int:student_id>/delete">Delete</a>
        </td>
    </tr>
</table>
```

Note: s_no_1, roll_number_1, first_name_1, last_name_1 ..... s_no_n, roll_number_n, first_name_n, last_name_n, $< int : student_id >$ **should be populated accordingly.**

2. If the user clicks the "Add student" button, your flask application should send a GET request to an endpoint "/student/create", which should display an HTML form as shown in the Figure 2. The HTML form should be the same as given below. Its ID must be "create-student-form".

```
<form action="/student/create" method="POST" id="create-student-form">
    <div>
      <label>Roll Number:</label>
      <input type="text" name="roll" required />
    </div>

    <div>
      <label>First Name:</label>
      <input type="text" name="f_name" required />
    </div>

    <div>
      <label>Last Name:</label>
      <input type="text" name="l_name" />
    </div>

    <div>
      <input type="submit" value = "Submit">
    </div>
</form>
```

- The HTML form should not have any other input elements.
- If the user clicks the submit button, the browser should send a POST request to your flask application's "/student/create" URI. The flask application should then create a student object (with attributes roll_number, first_name and last_name) and add it into the database and, it should redirect to the home page (URI = '/') and the student should be added into the table as shown in the Figure 3. Note that the roll number in each row of the table should be clickable.
- If the roll number 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 home page (URI = '/'), as shown in the Figure 4.

3. If the user clicks the "Update" button on the home page (URI = '/'), your flask application should send a GET request to an endpoint "/student/<int:student_id>/update", which should display an HTML form as shown in the Figure 5. The HTML form should be the same as given below. Its ID must be "update-student-form".

```
<form action="/student/<int:student_id>/update" method="POST" id="update-student-form">
    <div>
      <label>Roll Number:</label>
      <input type="text" name="roll" value="current_roll" disabled />
    </div>

    <div>
      <label>First Name:</label>
      <input type="text" name="f_name" value="current_f_name" required />
    </div>

    <div>
      <label>Last Name:</label>
      <input type="text" name="l_name" value="current_l_name"/>
    </div>

   <div>
```

```
            <label for="courses">Select Course: </label>
            <select name="course" id="course">
            <option value="course_1_id">course_1_name</option>
            <option value="course_2_id">course_2_name</option>
            ........
            ........
            <option value="course_n_id">course_n_name</option>
            </select>
        </div>

        <div>
          <input type="submit" value = "Update">
        </div>
    </form>
```

Note: $< int : student_id >$**, current_f_name, current_l_name and current_roll, course_1_id, course_2_id, course_n_id, course_1_name, course_2_name, course_n_name should be populated accordingly.**

- The HTML form should not have any other input elements.
- If the user clicks the submit button, the browser should send a POST request to your flask application's "/student/<int:student_id>/update" URI.
- The flask application should then update the student and corresponding enrollment into the database and redirect to the home page (URI = '/'). Note that the previous enrollment(s) must persist (if any).

4. If the user clicks the "Delete" button on the home page (URI = '/'), your flask application should send a GET request to an endpoint "/student/<int:student_id>/delete", which should delete the student and all the corresponding enrollments from the database and redirect to the home page (URI = '/').

5. If the user clicks on the roll number of any row in the table in the home page of the flask application, the application should send a GET request to an endpoint "/student/<int:student_id>", which should show all the information (student details and enrollment details) in an HTML page. The HTML page should also have a button labelled "Go Back" to navigate back to the home page (URI = '/'), as shown in the Figure 6. There must be 2 HTML tables in this page, one for showing the personal details and the other for displaying the enrollment details. The HTML for showing personal details should be the same as given below. Its ID must be "student-detail".

```
<table id = 'student-detail'>
    <thead>
        <tr>
            <th>Roll Number</th>
            <th>First Name</th>
            <th>Last Name</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>roll_number</td>
            <td>first_name</td>
            <td>last_name</td>
        </tr>
    </tbody>
</table>
```

Note: All the <td> should be populated accordingly.

The HTML for displaying the enrollment details should be the same as given below. Its ID should be "student-enrollments". This table should only exist when there is at least 1 enrollment for that particular student. The table must not exist if there are no enrollments.

```
<table id = "student-enrollments">
    <thead>
      <tr>
        <th>SNo</th>
        <th>Course Code</th>
        <th>Course Name</th>
        <th>Course Description</th>
        <th>Actions</th>
      </tr>
    </thead>

    <tbody>
      <tr>
        <td>s_no_1</td>
        <td>c_code_1</td>
        <td>c_name_1</td>
        <td>c_desc_1</td>
        <td>
          <a href="/student/<int:student_id>/withdraw/<int:course_id>">Withdraw</a>
        </td>
      </tr>
          ........
          ........
          ........
      <tr>
        <td>s_no_n</td>
        <td>c_code_n</td>
        <td>c_name_n</td>
        <td>c_desc_n</td>
        <td>
          <a href="/student/<int:student_id>/withdraw/<int:course_id>">Withdraw</a>
        </td>
      </tr>
    </tbody>
</table>
```

Note: All the <td> should be populated accordingly.

6. Every record in the enrollments table should have a "withdraw" button, which when clicked, your flask application should send a GET request to an endpoint "/student/<int:student_id>/withdraw/<int:course_id>", which should remove the course from current enrollments of the student from the database and redirect to the home page (URI = '/').

CRUD Operations for Courses

7. The index page should also have a button "Go to courses", When a user clicks on "Go to Courses", the browser should send a GET request to your flask application's "/courses" URI and the application should navigate the user to the courses page, which should display a table of all the courses currently

available in the database. The HTML table should be the same as given below (It's ID must be "all-courses"). It should display an appropriate message if no course exists in the database. It should also have a button labeled "Add course" as shown in the Figure 7. The HTML page should also have a button "Go to Students". When a user clicks on "Go to Students", the browser should send a GET request to your flask application's "/" URI and the application should navigate the user to the index page of the application.

```
<table id = "all-courses">
    <tr>
        <th>SNo</th>
        <th>Course Code</th>
        <th>Course Name</th>
        <th>Course Description</th>
        <th>Actions</th>
    </tr>
    <tr>
        <td>s_no_1</td>
        <td><a href="/course/<int:course_id>">course_code_1</a></td>
        <td>course_name_1</td>
        <td>course_description_1</td>
        <td>
          <a href="/course/<int:course_id>/update">Update</a>
          <a href="/course/<int:course_id>/delete">Delete</a>
        </td>
    </tr>
        ......
        ......
        ......
    <tr>
        <td>s_no_n</td>
        <td><a href="/course/<int:course_id>">course_code_n</a></td>
        <td>course_name_n</td>
        <td>course_description_n</td>
        <td>
          <a href="/course/<int:course_id>/update">Update</a>
          <a href="/course/<int:course_id>/delete">Delete</a>
        </td>
    </tr>

</table>
```

Note: s_no_1, course_code_1, course_name_1, course_description_1 ..... s_no_n, course_code_n, course_name_n, course_description_n, $< int : course_id >$ **should be populated accordingly.**

8. If the user clicks the "Add course" button, your flask application should send a GET request to an endpoint "/course/create", which should display an HTML form as shown in the Figure 8. The HTML form should be the same as given below. Its ID must be "create-course-form".

```
<form action="/course/create" method="POST" id="create-course-form">
    <div>
      <label>Course Code:</label>
      <input type="text" name="code" required />
    </div>
```

```
    <div>
      <label>Course Name:</label>
      <input type="text" name="c_name" required />
    </div>

    <div>
      <label>Course Description:</label>
      <input type="text" name="desc" />
    </div>

    <div>
      <input type="submit" value = "Submit">
    </div>
</form>
```

- The HTML form should not have any other input elements.
- If the user clicks the submit button, the browser should send a POST request to your flask application's "/course/create" URI. The flask application should 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'), as shown in the Figure 9. Note that the course code in each row of the table should be clickable.
- 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'), as shown in the Figure 10.

9. If the user clicks the "Update" button on the courses page (URI = '/courses'), your flask application should send a GET request to an endpoint "/course/<int:course_id>/update", which should display an HTML form as shown in the Figure 11. The HTML form should be the same as given below. Its ID must be "update-course-form".

```
<form action="/course/<int:course_id>/update" method="POST" id="update-course-form">
    <div>
      <label>Course Code:</label>
      <input type="text" name="code" value="current_code" disabled />
    </div>

    <div>
      <label>Course Name:</label>
      <input type="text" name="c_name" value="current_c_name" required />
    </div>

    <div>
      <label>Course Description:</label>
      <input type="text" name="desc" value="current_desc"/>
    </div>

   <div>
      <input type="submit" value = "Submit">
    </div>
</form>
```

Note: $< int : course_id >$**, current_code, current_c_name and current_desc should be populated accordingly.**

- The HTML form should not have any other input elements.
- If the user clicks the submit button, the browser should send a POST request to your flask application's "/course/<int:course_id>/update" URI.
- The flask application should then update the targeted course (with the given course_id in the URI) in the database and redirect to the courses page (URI = '/courses').

10. If the user clicks the "Delete" button on the courses page (URI = '/courses'), your flask application should send a GET request to an endpoint "/course/<int:course_id>/delete", which should delete the course from the database and redirect to the home page (URI = '/').

11. 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 should send a GET request to an endpoint "/course/<int:course_id>", which should show all the information (course details and students enrolled in the course) in an HTML page. The HTML page should also have a button labelled "Go Back" to navigate back to the courses page (URI = '/courses'), as shown in the Figure 12. There must be 2 HTML tables in this page, one for showing the course details and the other for displaying the enrollment details. The HTML for showing course details should be the same as given below. Its ID must be "course-detail".

```
<table id = "course-detail">
     <thead>
         <tr>
             <th>Course Code</th>
             <th>Course Name</th>
             <th>Course Description</th>
         </tr>
     </thead>
     <tbody>
         <tr>
             <td>c_code</td>
             <td>c_name</td>
             <td>c_desc</td>
         </tr>
     </tbody>
 </table>
```

Note: All the <td> should be populated accordingly.

The HTML for displaying the enrollment details should be the same as given below. Its ID should be "course-table".

```
<table id = "course-table">
    <thead>
      <tr>
        <th>SNo</th>
        <th>Roll Number</th>
        <th>Student First Name</th>
        <th>Student Last Name</th>
      </tr>
    </thead>

    <tbody>
      <tr>
        <td>s_no_1</td>
```

```
          <td>roll_number_1</td>
          <td>first_name_1</td>
          <td>last_name_1</td>
        </tr>
              ......
              ......
              ......
        <tr>
          <td>s_no_n</td>
          <td>roll_number_n</td>
          <td>first_name_n</td>
          <td>last_name_n</td>
        </tr>
      </tbody>
    </table>
```

Note: All the <td> should be populated accordingly.

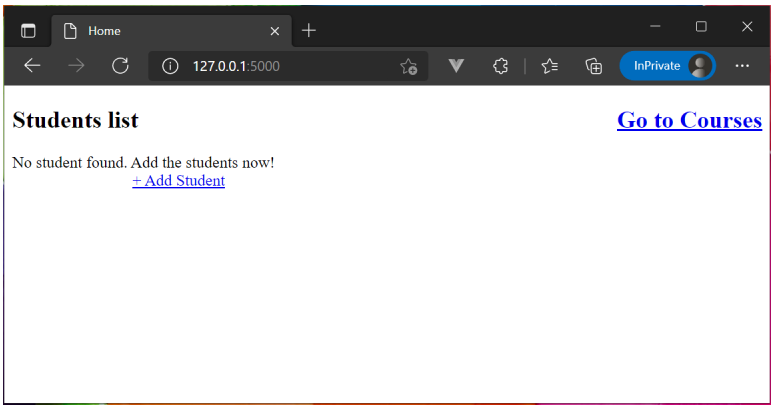

Figure 1: Home Page

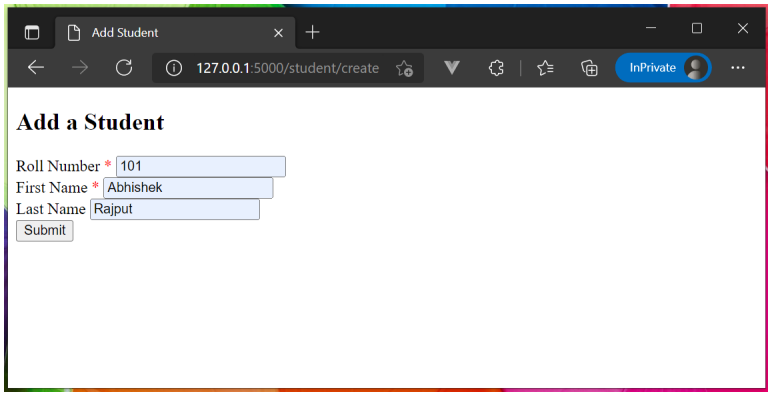

Figure 2: Add Student Form

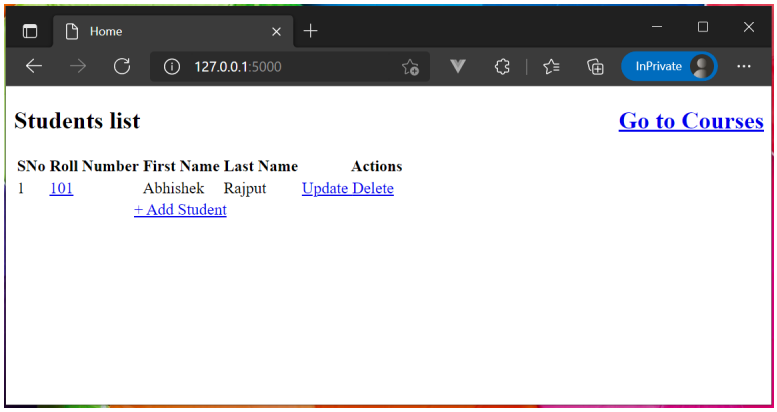

Figure 3: Home Page

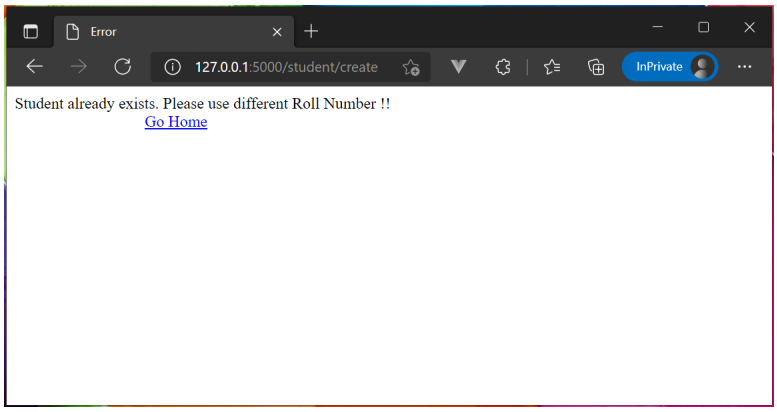

Figure 4: Student Already Exists Page

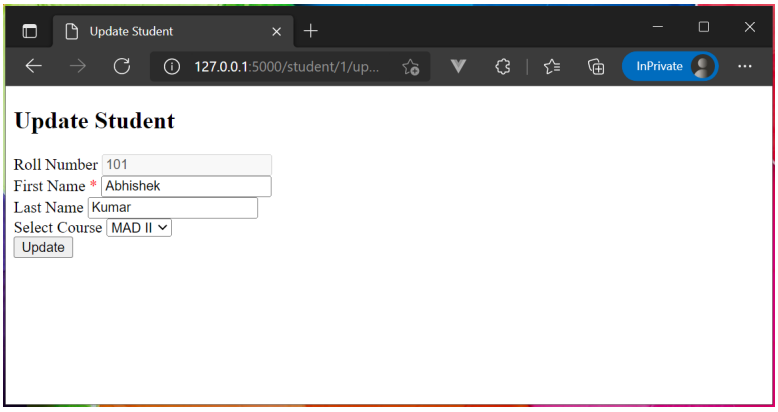

Figure 5: Update Student Form

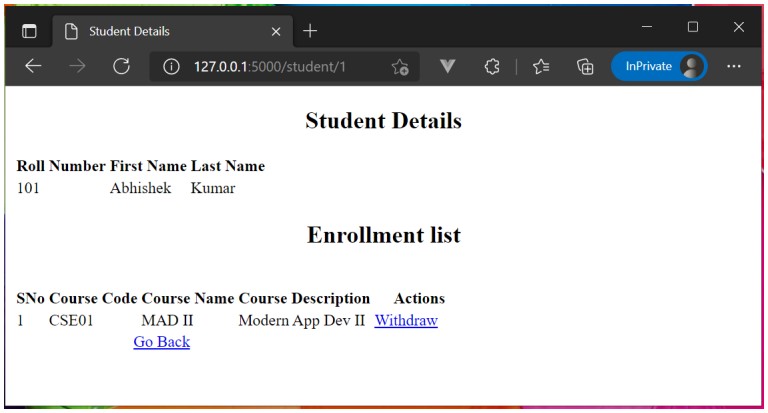

Figure 6: Student Details Page

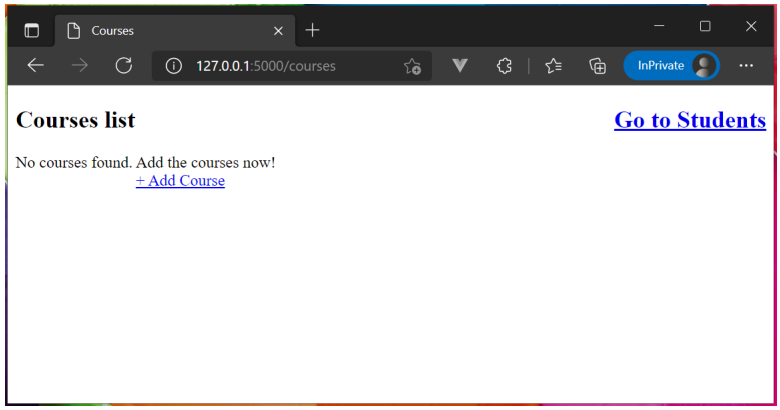

Figure 7: Courses Home Page

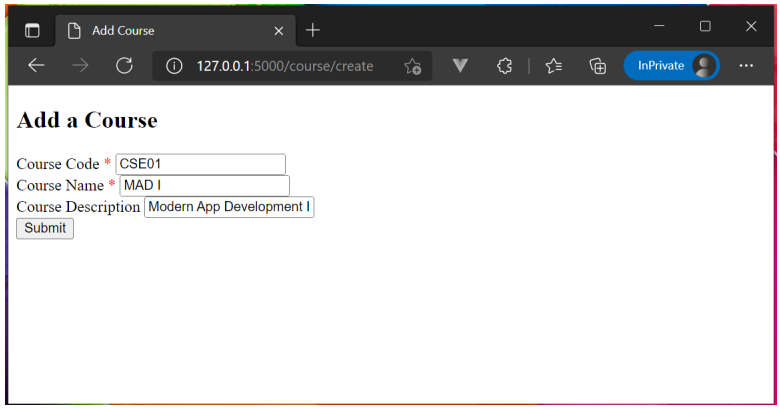

Figure 8: Add Course Form

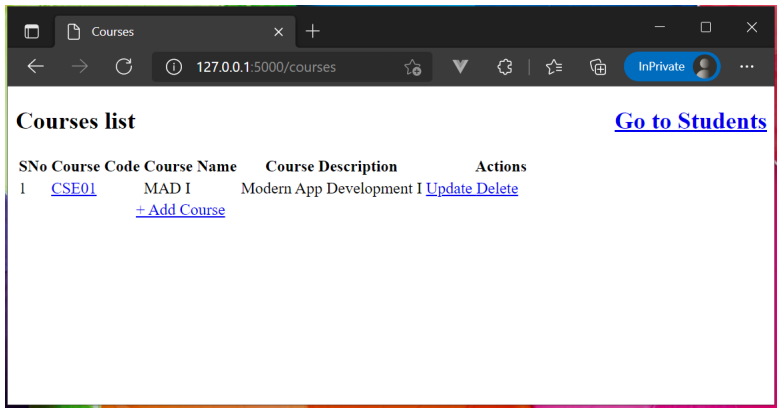

Figure 9: Courses Home Page

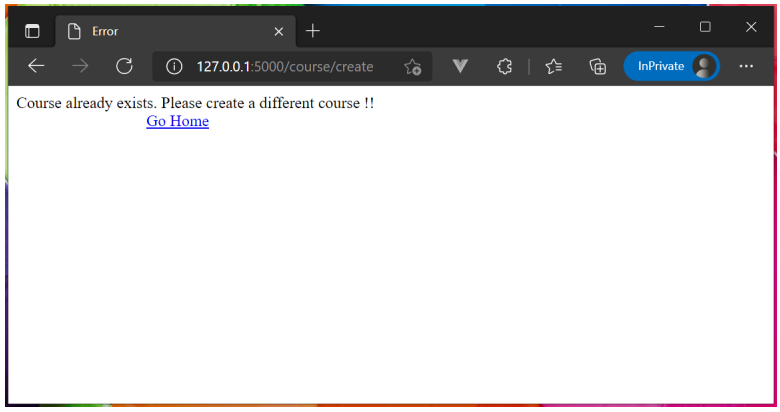

Figure 10: Course Already Exists Page

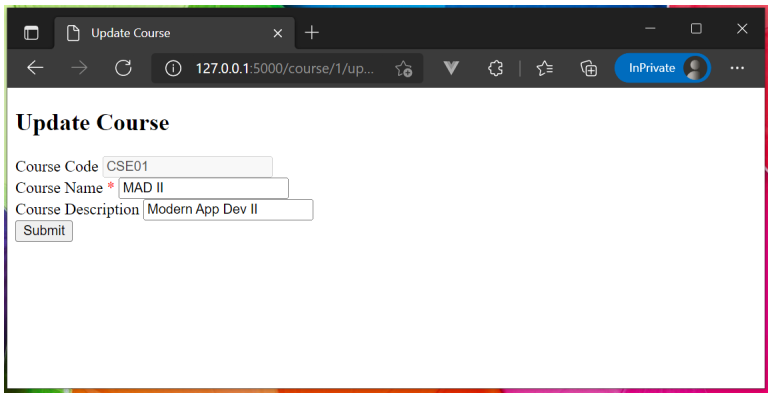

Figure 11: Update Course Form

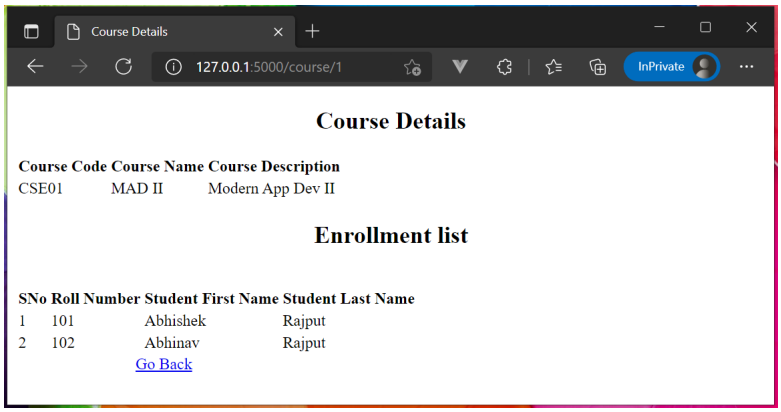

Figure 12: Course Details Page