Entity-wise CRUD Operation in Laravel

Since our target software is University Learning Management System (LMS), therefore few prominent entities include: Student, Department, Teacher, Course, Admission Office, and others.

This section would help you to implement in Laravel the Student entity. Other entities can be implemented in the same way. Next sections would guide you how to create one-to-many (1xM) and many-to-many (MxM) Relationships.

NOTE: The Laravel commands used below need to be typed in the Terminal Window of VS Code which can be opened using the shortcut key Ctrl + ~ (called control console) or by clicking the VS Code menu: Terminal > New Terminal.

Laravel uses the Model View Controller (MVC) design pattern

- Model interacts with database
- View (or web page) interacts with user
- Controller contains functions which connect a View with its Model

For an entity, a one-to-many (1xM) relation or a many-to-many (MxM) join-table relation, we provide Create, Read, Update and Delete (CRUD) operations. The MVC design pattern is applied to each part (i.e. for Create, Read, Update and Delete) of the CRUD operation.

To provide CRUD for an entity or relation, we first implement Create, then Read, then Update and finally Delete operation. While implementing each operation:

- we will first work with the Model layer
- then with the View layer
- and finally, with the Controller layer

Let's implement CRUD operation for the Student entity. Then implement CRUD for the 1xM Department-Student relation and finally implement CRUD for MxM Course-Student join-relation.

1. Applying MVC on the Create operation of Student

Summary of Model-View-Controller design pattern

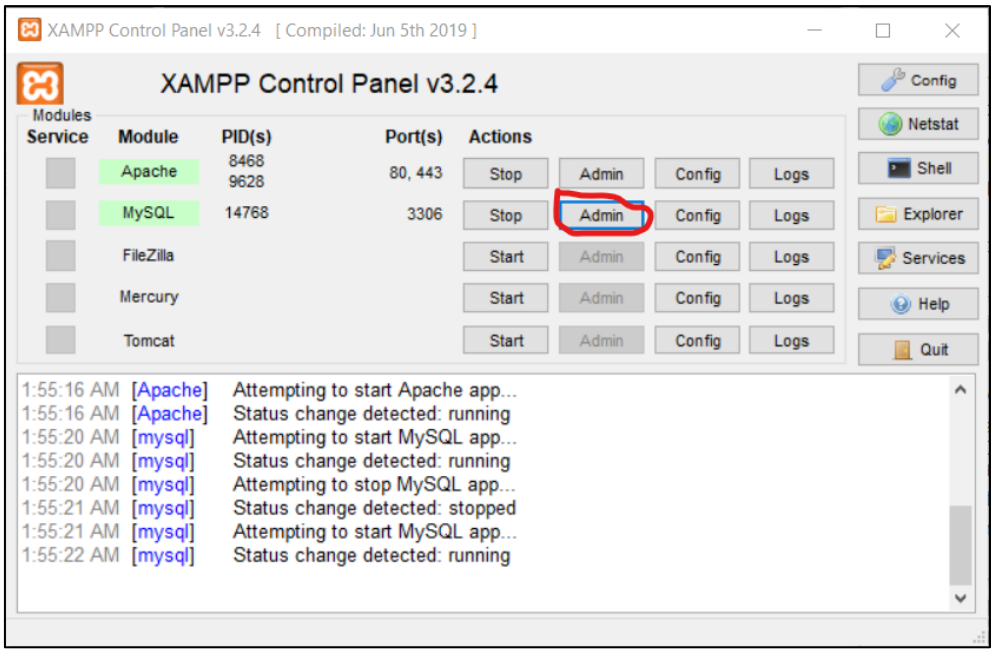
A. **Model** in Laravel is an object that performs read and update operations on a database table. For each database table, there is an associated Model. After creating model, we write migrations and execute them.
B. Create webpages (**Views**) to perform Create, Read, Update and Delete (CRUD) operations on the entity. Define route for each page which is a linkage between the webpage URL typed in the browser and a function. Thus, associated function is called when a user types a webpage URL in the browser.
C. Define one or more associated functions (**Controller Function**) of each webpage.

Detail

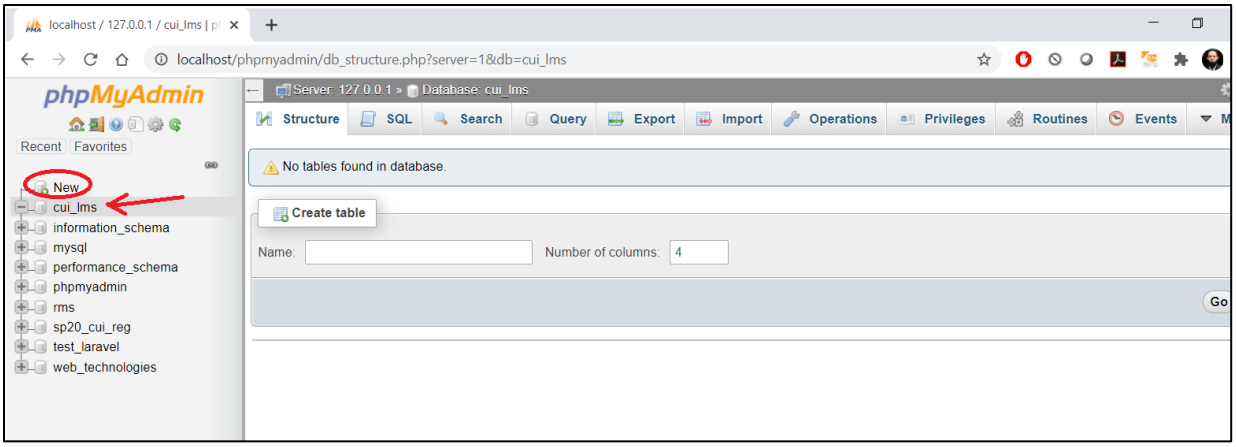
A. Create the Model, Student

a) Create an empty database in MySQL

i. Open XAMPP Control. Start Apache and MySQL
ii. Click **Admin** in front of MySQL to open **phpMyAdmin** in the browser



iii. In **phpMyAdmin**, create a new empty database. We created database named **cui_lms**



b) Set database name in Laravel

i. Open the Laravel project folder in VS Code. It is **~\htdocs\CUI_LMS** in my case

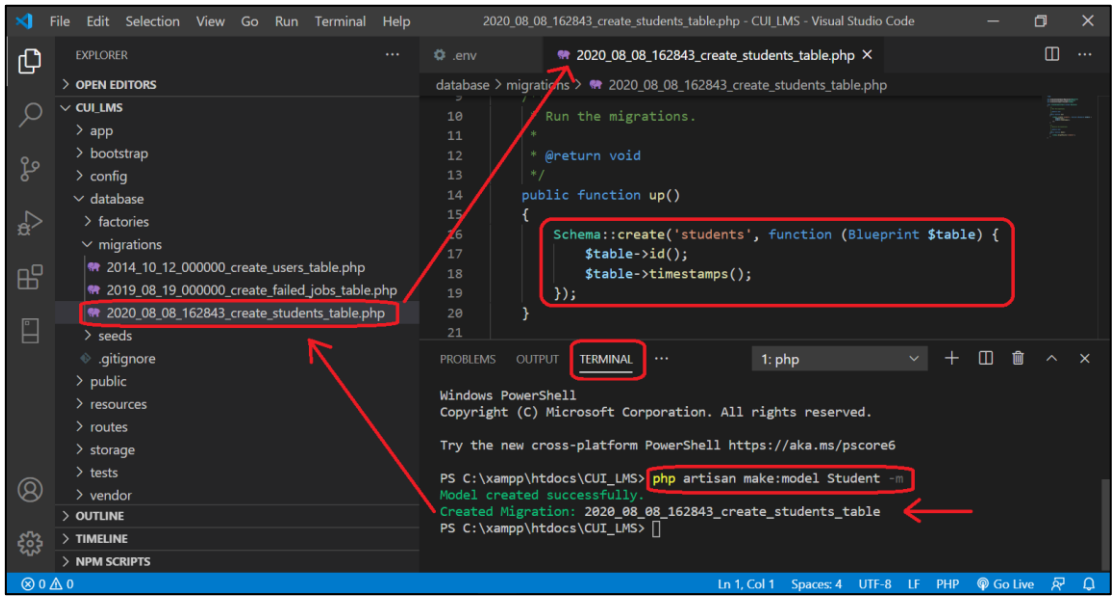
ii. Go to **.env file**

iii. Write database name in front of DB_DATABASE=**cui_lms**

c) **Create model and migration of Student**

i. Type following command in the VS Code Terminal window. It will create the model **Student.php** in the **app folder** and its associated migration in the **database\migrations** folder.

- php artisan make:model Student -m



ii. At this time, the table, **students,** has NOT been created in the database.

d) **Update migration script to add more fields in the table students**

i. A default field ***id*** (autoIncrement Primary Key) would be there. Remove it.

ii. add the fields ***cnic***, ***name***, ***address***, ***telNo***, ***age*** and ***marital_status*** as shown below

iii. Make **cnic** as primary key and set the default value of **maritial_status** as **false**

Help • 2020_02_20_000003_create_students_table.php - CRUD - Visual Studio Code

2020_02_20_000003_create_students_table.php

database > migrations > 2020_02_20_000003_create_students_table.php

```php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateStudentsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->string ('cnic', 15)->primary();
            $table->string('name', 35);
            $table->string('address', 75);
            $table->string('telNo', 15);
            $table->double('age');
            $table->boolean('marital_status')->default(false);
          $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
```

iv. **If we make an attribute of our choice as Primary Key, we must explicitly mention in the Model.**

In the Student model, we need to explicitly mention that **<u>cnic</u>** is the Primary Key and that it would be a non-auto incrementing field. If we use the default autoincrementing, id field, as Primary Key, there is no need to perform this step.

```
class Student extends Model
{
   protected $primaryKey = 'cnic'; // Set as primary key
   public $incrementing = false;   // Non auto-incrementing


}
```

v. Type the following command in the Terminal window of VS Code to make changes to the schema of **students** table

- php artisan migrate

vi. The following command reverses last N migrations. Thus, if N=3, last 3 migrations would be reversed

- php artisan migrate:rollback --step=N

vii. The following command drops all tables and performs migrations again

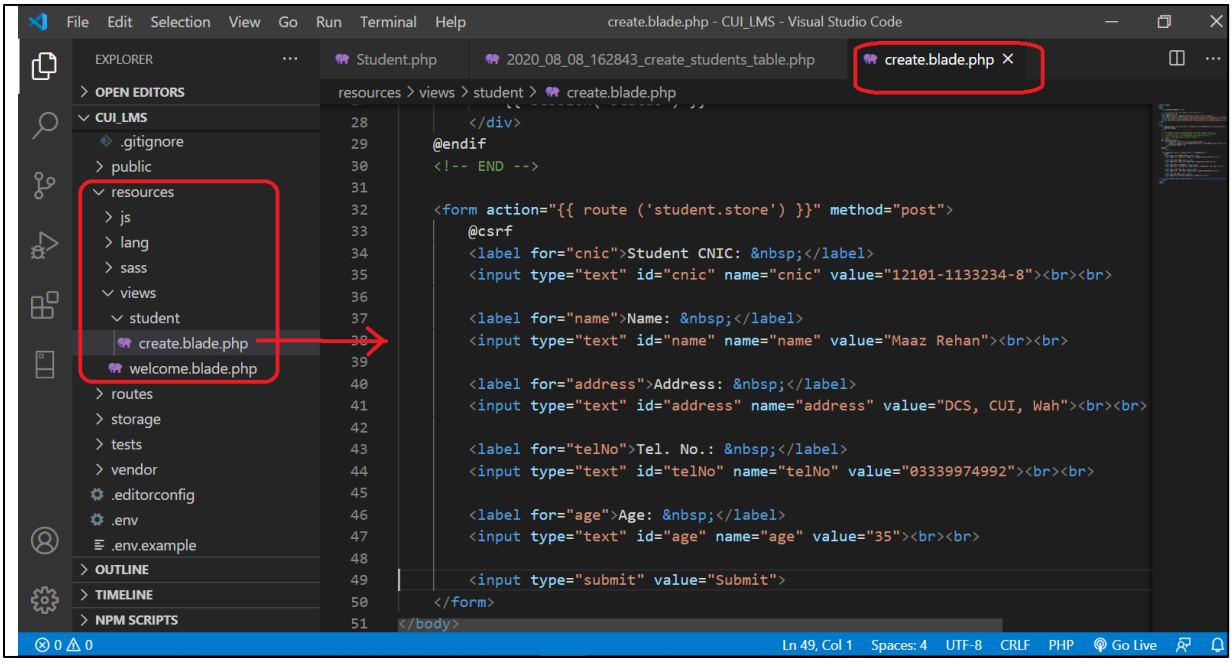
- php artisan migrate:fresh

viii. After the **migrate** command, the table, **students,** would have been created in the database.

B. Write code for the create Webpage (View) and add Routes

a) Create student folder in **resources > views**

b) In student folder, add new file, **create.blade.php**



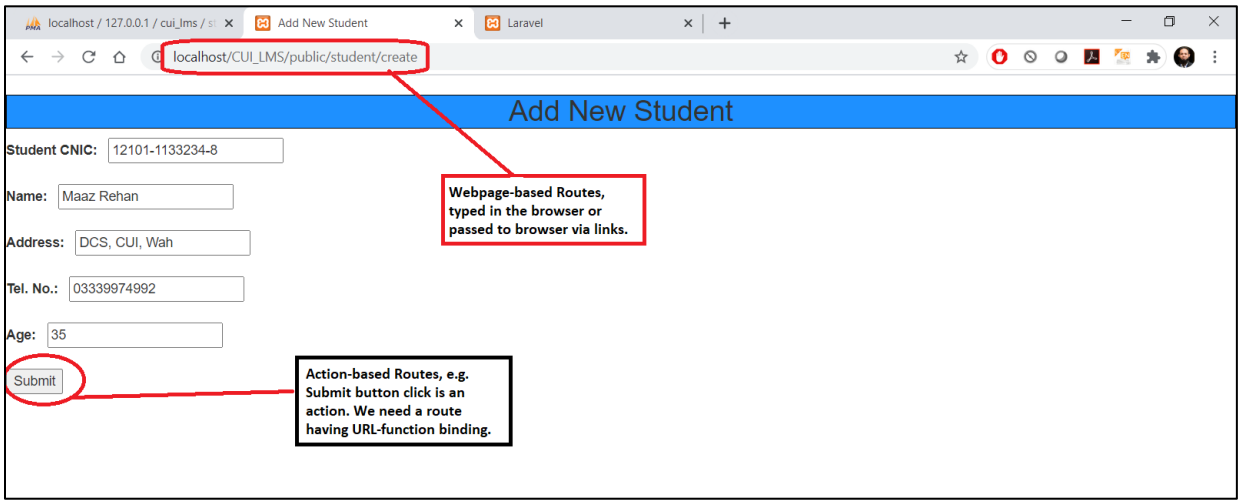
c) Write webpage code inside **create.blade.php**

```
<!DOCTYPE html>
<html>
<head>
    <title>Add New Student</title>

    <!-- For Success alert that appears after deletion -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.
css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>

<body>
    <h2 style="border: 1px solid black; background-color:DodgerBlue; text-align:center;">
        Add New Student
    </h2>

    <!-- For Redirecting With Flashed Session Data when 'Submit' button -->
    <!-- is pressed in the 'create.blade.php' view which calls the relevant -->
    <!-- function 'store' in the StudentController and then this -->
    <!-- view, 'create.blade.php' is again called -->
    <!-- START -->
    @if (session('status'))
        <div class="alert alert-success alert-dismissible">
            <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
            {{ session('status') }}
        </div>
    @endif
    <!-- END -->

    <form action="{{ route ('student.store') }}" method="post">
        @csrf
        <label for="cnic">Student CNIC:  </label>
        <input type="text" id="cnic" name="cnic" value="12101-1133234-8"><br><br>

        <label for="name">Name:  </label>
        <input type="text" id="name" name="name" value="Maaz Rehan"><br><br>

        <label for="address">Address:  </label>
        <input type="text" id="address" name="address" value="DCS, CUI, Wah"><br><br>

        <label for="telNo">Tel. No.:  </label>
        <input type="text" id="telNo" name="telNo" value="03339974992"><br><br>

        <label for="age">Age:  </label>
        <input type="text" id="age" name="age" value="35"><br><br>

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

d) Routes

i. For the above webpage, the following page should open in the browser if the browser is aware of the URL which has been typed in the address bar. In our case the browser is NOT aware. To achieve this in MVC, we need to define a Route.



ii. What is a Route?
A route is a connection between: **(i)** the URL typed in the browser, and **(ii)** the function which is executed once the browser receives URL.

iii. Routes are of two types.
- Web page related routes: First type of routes are those which are defined against web pages. For each web page, there is a route. When URL of a web page is typed in the browser, its associated route is invoked.
- Action related routes: Second type of routes are those which are defined against actions. For each action there is a route. For example, when a button or link is clicked, its associated route is invoked.

iv. The Route of a web page is written in the file **resources > web.php**
v. A route binds a URL with a function which is written inside **Controller**
vi. The function is called/executed when the URL is provided to the address bar of browser

e) Adding Webpage-based Route

i. When user wants to open a webpage, its URL is typed in the browser. In our case, if user types http://localhost/CUI_LMS/public/student/create in the browser, then create webpage should open.
ii. To achieve this, we add the Route::get() function in **resources > web.php** file.

- Route::get('student/create', 'StudentController@create')->name('student.create');

- Here,
  - student/create is the user-defined URL and is associated with the create webpage of student. There can be other create webpages for other entities as well, e.g. teacher.
  - StudentController@create means call the create() function in the StudentController
  - name('student.create') is a user-defined alternate name of the route

f) Adding Action-based Route

i. When user clicks the **Submit button** (shown above), data on the form should store in the database table **students**.

ii. To achieve this, a route related to click (submit) action needs to be defined which calls the store function when user clicks it

- Route::post('student/store', 'StudentController@store')->name('student.store');

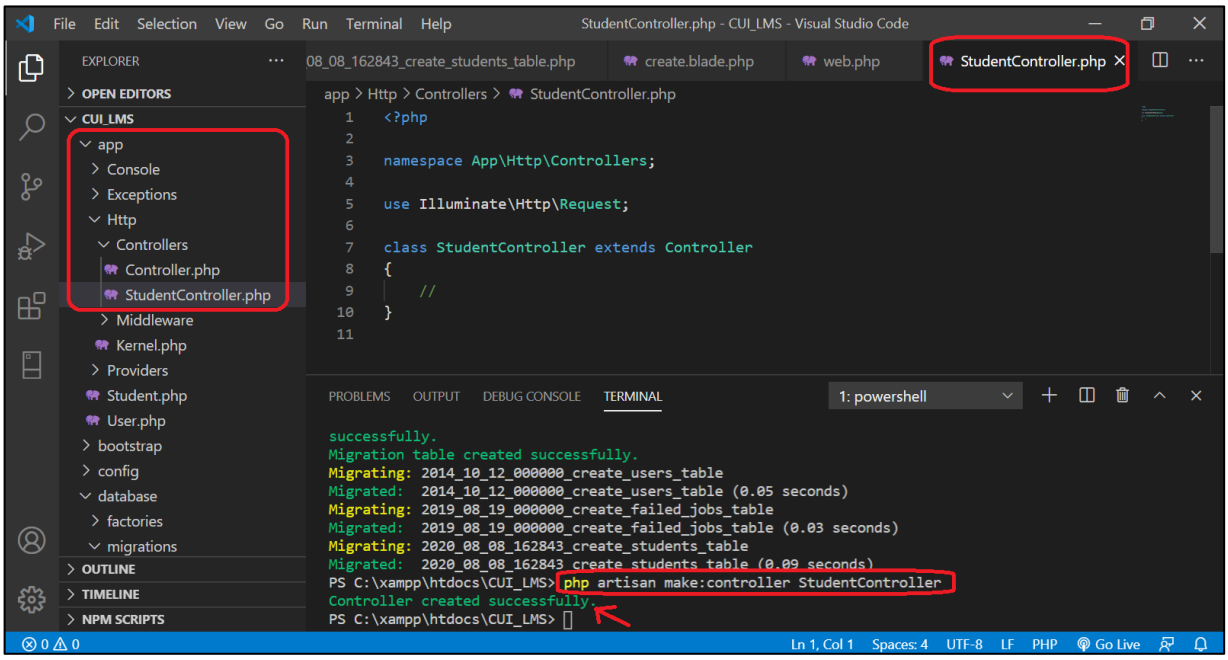
- Here,
  - student/store is the user-defined URL and is associated with the click action of Submit button on Student Form
  - StudentController@store means call the store() function in the StudentController
  - name('student.store') is a user-defined alternate name of route

C. Make StudentController and add functions related to create

a) As shown in bullets (e) and (f) above, there are two routes (Route::get, Route::post) and two functions (create, store) in the controller, StudentController

b) Use following command to create the controller which is shown below

- php artisan make:controller StudentController



c) Include Model name in the Controller. So, we add '**Student' model** in the '**StudentController' controller**.

- use App\Student;

d) The **create()** function is given below which is invoked when the URL is typed in the browser. This function displays the **create** webpage

```
public function create() {
    return view ("student.create");
}
```

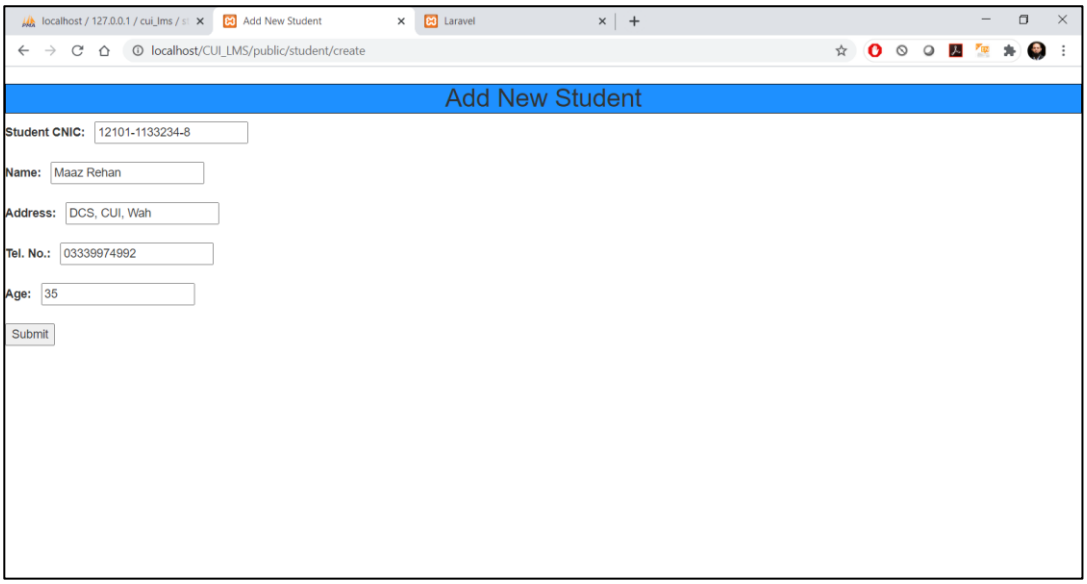
e) The **store()** function is given below which inserts form data into the table **students**

```
public function store(Request $request) {

  $student = new Student;    // Must import the Model file: use App\Student;
  $student->cnic = $request->get('cnic');
```

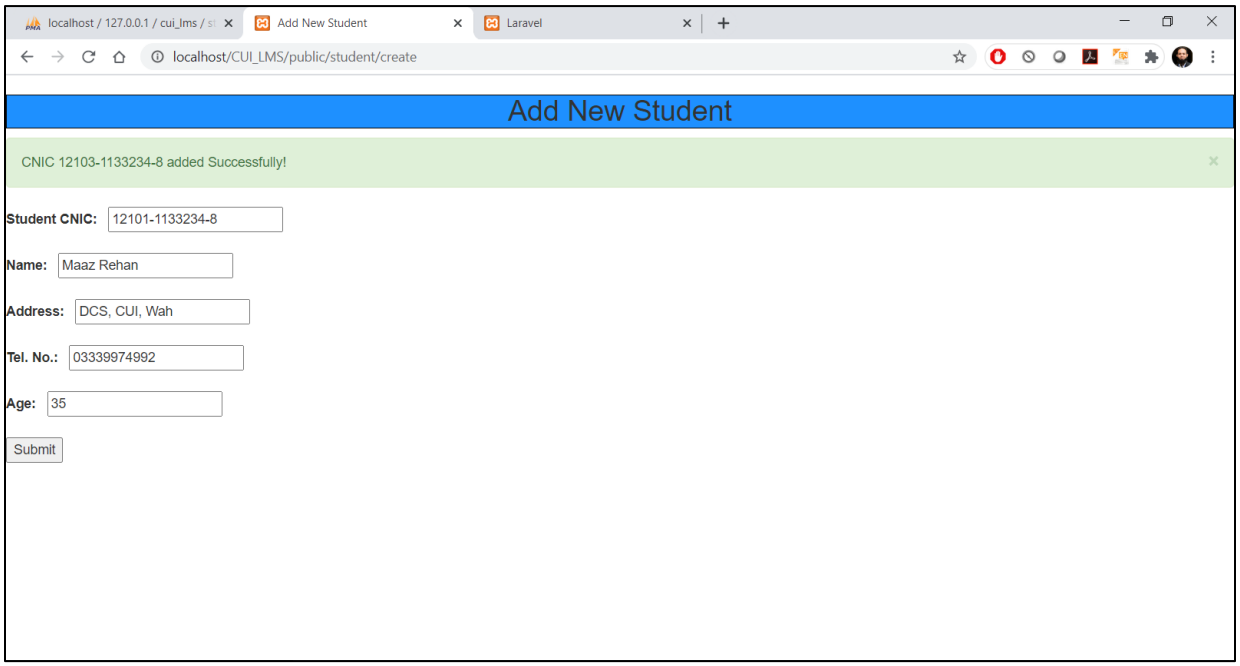
```
    $student->name = $request->get('name');
    $student->address = $request->get('address');
    $student->telno = $request->get('telNo');
    $student->age = $request->get('age');

    // Since the marital_status field has a default value of ZERO,
    // therefore, even if no text is copied from the text box
    // the value ZERO would be stored.
    //$student->marital_status = $request->get('marital_status');

    $student->save();      /* Store data inside the table */

    // -------------------------------------
    // Help on the following code is given at the following URL
    // https://laravel.com/docs/5.8/redirects#redirecting-with-flashed-session-data
    //
    return redirect('student/create')
        ->with('status', 'CNIC '.$student->cnic.' added Successfully!');
    // -------------------------------------
}
```

D. How a webpage opens and data is stored ?

a) When user types the URL http://localhost/CUI_LMS/public/student/create in the browser, , route with the name student.create is searched in web.php file. When found, the StudentController function create() is searched and executed. The create() function displays the Add New Student webpage, as shown below.



i. When user presses the submit button, the route student.store is searched in web.php. When student.store route is found, the StudentController function store() is searched and executed. The store function stores form data in the students table and then displays the create page.
ii. Add few students so that we can view / update / delete them

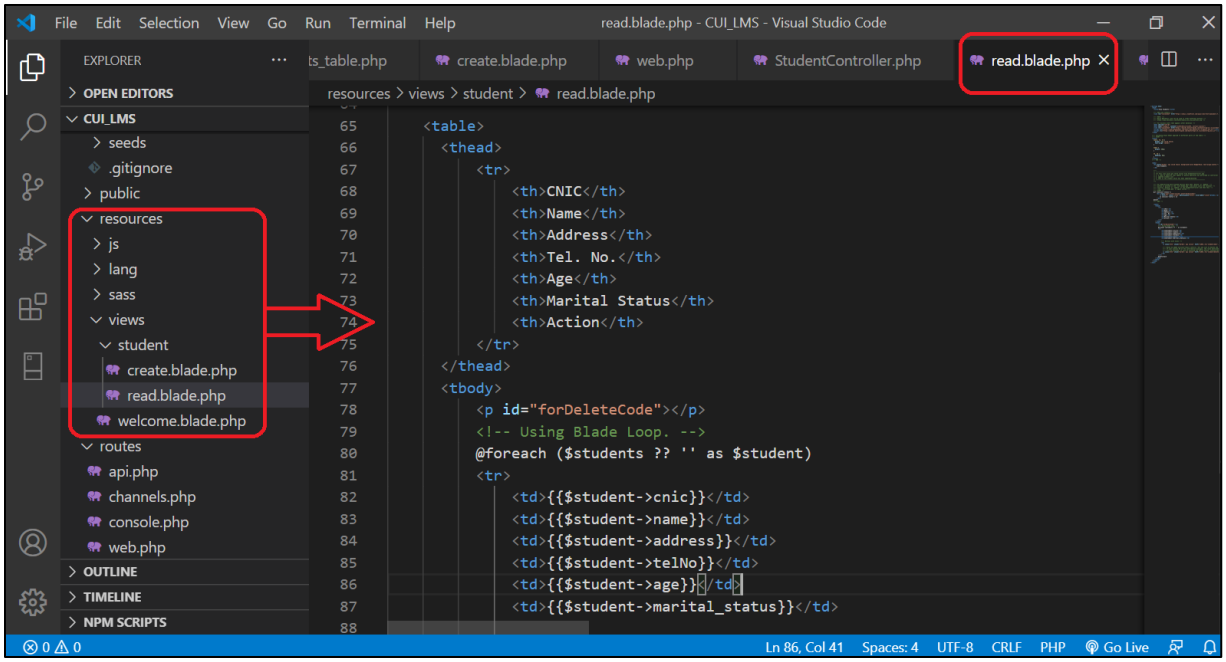


2. Applying MVC on the Read operation of Student

A. Student Model Already Exists. So, skip this step.

B. Write code for the 'read' Webpage (View) and add Routes

a) In the student folder, add new file, **read.blade.php**



b) Write webpage code inside **read.blade.php**

```
<!doctype html>
  <head>
    <title>View Students</title>

    <!-- Add icon library -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
awesome.min.css">

    <!-- NOTE: -->
    <!-- Below W3Schools link can be used to study different buttons -->
    <!-- https://www.w3schools.com/howto/howto_css_icon_buttons.asp -->

    <!-- For Success alert that appears after deletion -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.c
ss">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
  </head>

  <!-- Cascading Style Sheet applied to different parts of the table -->
  <!-- START -->
  <style>
```

```
  table, th, td {
    border: 1px solid black;
    text-align: center;
  }

  table {
    margin: 25px;
  }

  th, td {
    padding: 5px;
  }
</style>
<!-- END -->

<body>
  <h2 style="border: 1px solid black; background-color:DodgerBlue; text-align:center;">
      View Students
  </h2>

  <!--
  // ---------------------------------------------------------------
  // *** This view would get three values from StudentController.php
  // 1. a code to identify that update or delete operation was performed in controller
  // 2. updated student list
  // 3. CNIC of the student which has been updated/deleted
  // ---------------------------------------------------------------
  -->

  <!-- For Redirecting With Flashed Session Data when 'Delete' or 'Update' -->
  <!-- button is pressed in the 'list.blade.php' view which calls the relevant -->
  <!-- function 'delete' or 'update' in the StudentController and then this -->
  <!-- view, 'list.blade.php' is again called -->
  <!-- START -->
  @if (session('status'))
      <div class="alert alert-success alert-dismissible">
          <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
          {{ session('status') }}
      </div>
  @endif
  <!-- END -->

  <table>
    <thead>
        <tr>
```

```
                <th>CNIC</th>
                <th>Name</th>
                <th>Address</th>
                <th>Tel. No.</th>
                <th>Age</th>
                <th>Marital Status</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <p id="forDeleteCode"></p>
            <!-- Using Blade Loop. -->
            @foreach ($students ?? '' as $student)
            <tr>
                <td>{{$student->cnic}}</td>
                <td>{{$student->name}}</td>
                <td>{{$student->address}}</td>
                <td>{{$student->telNo}}</td>
                <td>{{$student->age}}</td>
                <td>{{$student->marital_status}}</td>

                <!-- Buttons with Icons -->
                <td>
                  <a class="btn" style="border: 1px solid;" href="{{URL::to('student/edit', $student-
>cnic)}}" title="Edit -> {{$student->cnic}}"> <i class="fa fa-edit"></i></a>

                  <!-
- Below we added onclick="return confirm ('Are you sure to delete the student {{$student-
>name}} having CNIC {{$student->cnic}}?')"-->
                  <!-- If user presses OK on the confirmation dialogue, the route mentioned in href --
>
                  <!-- will be executed. In case of pressing the Cancle button, nothing happens. -->
                  <a class="btn" style="border: 1px solid;" href="{{URL::to('student/delete', $student
->cnic)}}" onclick="return confirm ('Are you sure to delete the student {{$student-
>name}} having CNIC {{$student->cnic}}?')" title="Delete -> {{$student->cnic}}"> <i class="fa fa-
trash"></i></a>
                </td>
            </tr>
            @endforeach
        </tbody>
    </table>
  </body>
</html>
```

c) Adding Webpage-based Route

i. When user types http://localhost/CUI_LMS/public/student/read in the browser, then read webpage should open.

ii. To achieve this, we add the Route::get() function in **resources > web.php** file.

- Route::get('student/read', 'StudentController@read')->name('student.read');

- Here,
  - student/read is the user-defined URL and is associated with the read webpage of student
  - StudentController@read means call the read() function in the StudentController
  - name('student.read') is a user-defined alternate name of the route

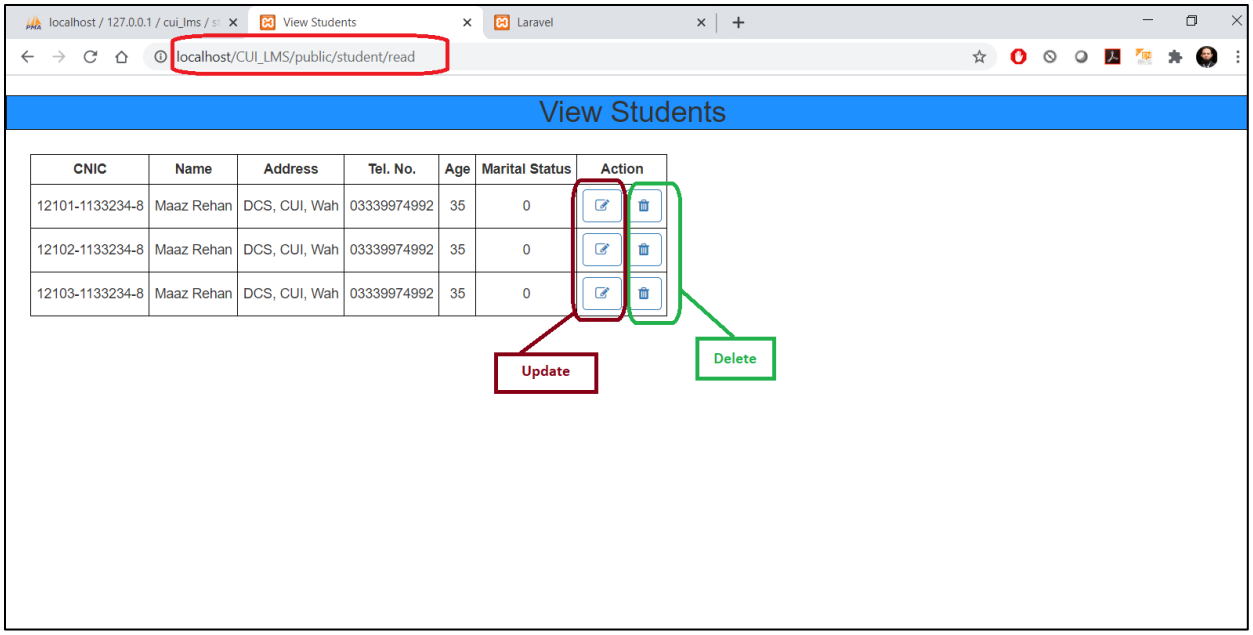
C. StudentController exists, so add functions related to read operation

a) The **read()** function is given below which is invoked when http://localhost/CUI_LMS/public/student/read is typed in the browser. This function lists all students on the webpage

```
public function read() {
    $students = Student::all();         // Load students using the model 'Student'

    // Pass the $students to the view, 'student/read'
    return view('student/read')
        ->with(['students' => $students]);
}
```

D. How read webpage opens and shows data?

a) When user types the URL http://localhost/CUI_LMS/public/student/read in the browser, route with the name student.read is searched in web.php file. When found, the StudentController function read() is searched and executed. The read() function fetches data from the database and displays them as shown below.

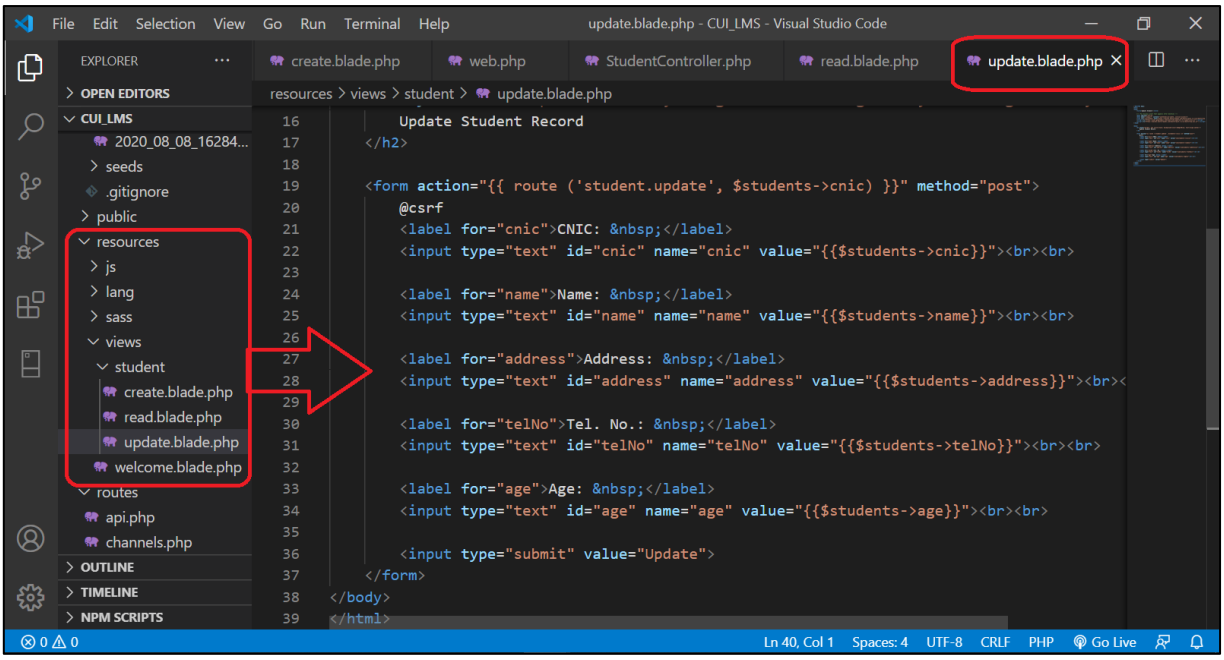


3. Applying MVC on the Update operation of Student

A. Student Model Already Exists. So, skip this step.

B. Write code for the 'update' Webpage (View) and add Routes

a) In the student folder, add new file, **update.blade.php**

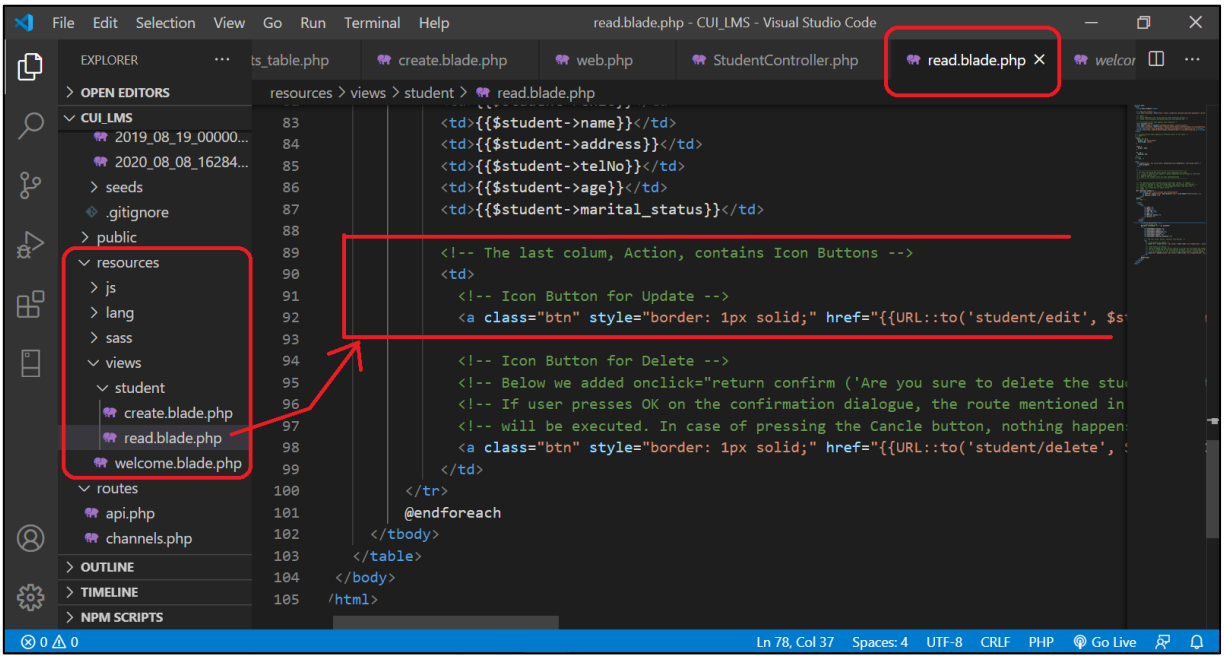


b) Write webpage code inside **update.blade.php**

```
<!DOCTYPE html>
<html>
<head>
    <title>Update Student</title>

    <!-- For Success alert that appears after deletion -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.c
ss">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
```

```
<body>
    <h2 style="border: 1px solid black; background-color:DodgerBlue; text-align:center;">
        Update Student Record
    </h2>

    <form action="{{ route ('student.update', $students->cnic) }}" method="post">
        @csrf
        <label for="cnic">CNIC:  </label>
        <input type="text" id="cnic" name="cnic" value="{{$students->cnic}}"><br><br>

        <label for="name">Name:  </label>
        <input type="text" id="name" name="name" value="{{$students->name}}"><br><br>

        <label for="address">Address:  </label>
        <input type="text" id="address" name="address" value="{{$students->address}}"><br><br>

        <label for="telNo">Tel. No.:  </label>
        <input type="text" id="telNo" name="telNo" value="{{$students->telNo}}"><br><br>

        <label for="age">Age:  </label>
        <input type="text" id="age" name="age" value="{{$students->age}}"><br><br>

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

c) The webpage, **update.blade.php** opens when user clicks an **Edit button** of a student record. The Edit button is in the Action column of **read.blade.php** as shown below. The code lines, **href="{{URL::to('student/edit', $student->cnic)}}"**, state that the route **student/edit** be invoked.



d) Adding Action-based Route for Edit button on the read webpage

i. On the read page, when user clicks the Edit button of a record, the record of student is shown on the **update webpage** with the help of following route

ii. To achieve this, we add the Route::get() function in **resources > web.php** file.

- Route::get('student/edit/{cnic}', 'StudentController@edit')->name('student.edit');
- Here,
  - student/edit/{cnic} is the user-defined URL which takes one argument which is the cnic of the student for whom Update is initiated
  - StudentController@edit means call the edit() function in the StudentController
  - name('student.edit') is a user-defined alternate name of the route

b) Adding Action-based route for the Update button on the update webpage

i. On the update page, when user clicks the Update button, the record of selected student is saved in the database.

ii. To achieve this, we add the Route::post() function in **resources > web.php** file.

- Route::post('student/update/{cnic}', 'StudentController@update')->name('student.update');
- Here,
  - student/update/{cnic} is the user-defined URL which takes one argument which is the cnic of the student for whom Update is done
  - StudentController@ update means call the update() function in the StudentController

- name('student. update') is a user-defined alternate name of the route

C. StudentController exists, so add functions related to update operation

a) The **edit()** function is given below which is invoked when Edit button in the Action column on the read webpage is clicked. This function fetches the information of cnic from database and sends to the webpage **student/update**

```
public function edit($cnic) {

    $students = Student::find($cnic);      // Load students using the model 'Student'

    // Pass the $students to the view, 'student/update'
    // so that user can update.
    return view('student/update')
        ->with(['students' => $students]);
}
```

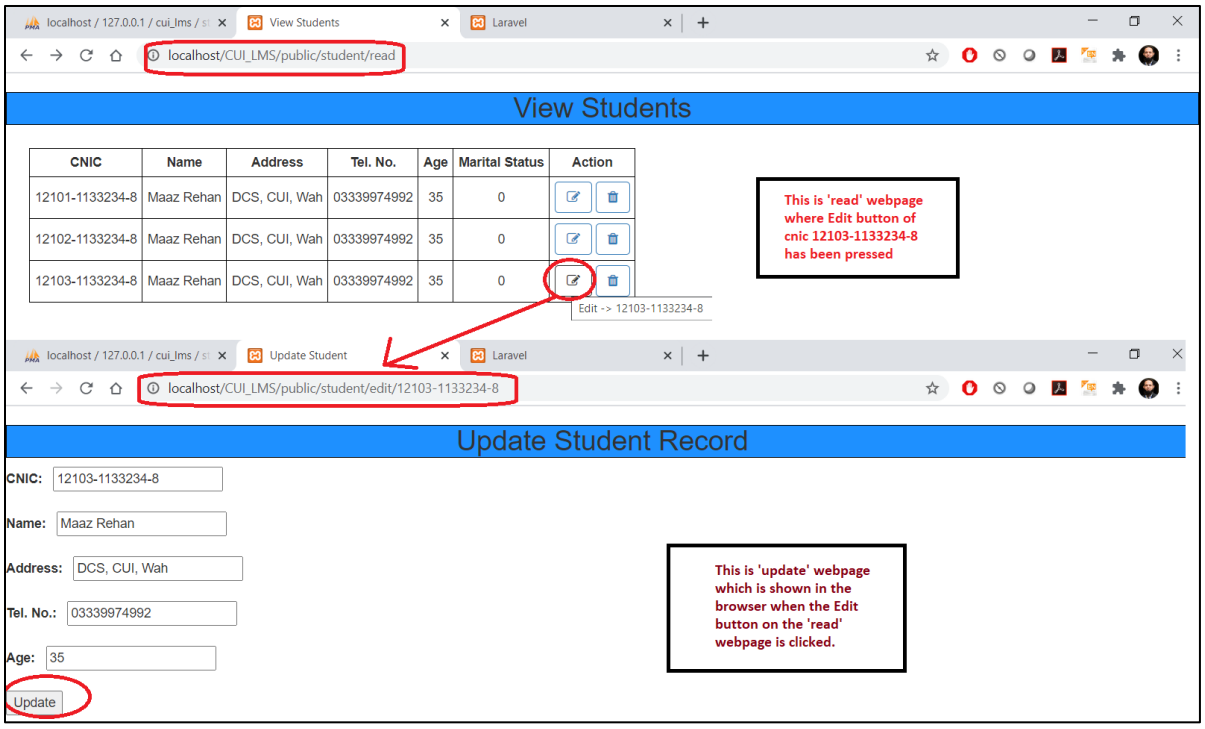
b) The **update()** function is given below which is invoked when Update button on the update webpage is clicked. This function stores the information of cnic in the database and then reloads the read webpage with the new contents of students table.

```
public function update(Request $request, $cnic) {

    // Locate the row of this CNIC so that updated record
    // can be overwritten ONLY on the previous record of this CNIC.
    $student = Student::find($cnic);

    // you can add the chech here whether this student exists or not?

    $student->cnic = $request->get('cnic');   // Copy from textbox and paste on the handler
    $student->name = $request->get('name');
    $student->address = $request->get('address');
    $student->telno = $request->get('telNo');
    $student->age = $request->get('age');

    // Since the marital_status field has a default value of ZERO,
    // therefore, even if no text is copied from the text box
    // the value ZERO would be stored.
    //$student->marital_status = $request->get('marital_status');

    $student->save();       /* Overwrite data on the row pointed by CNIC */

    // ------------------------------------
```

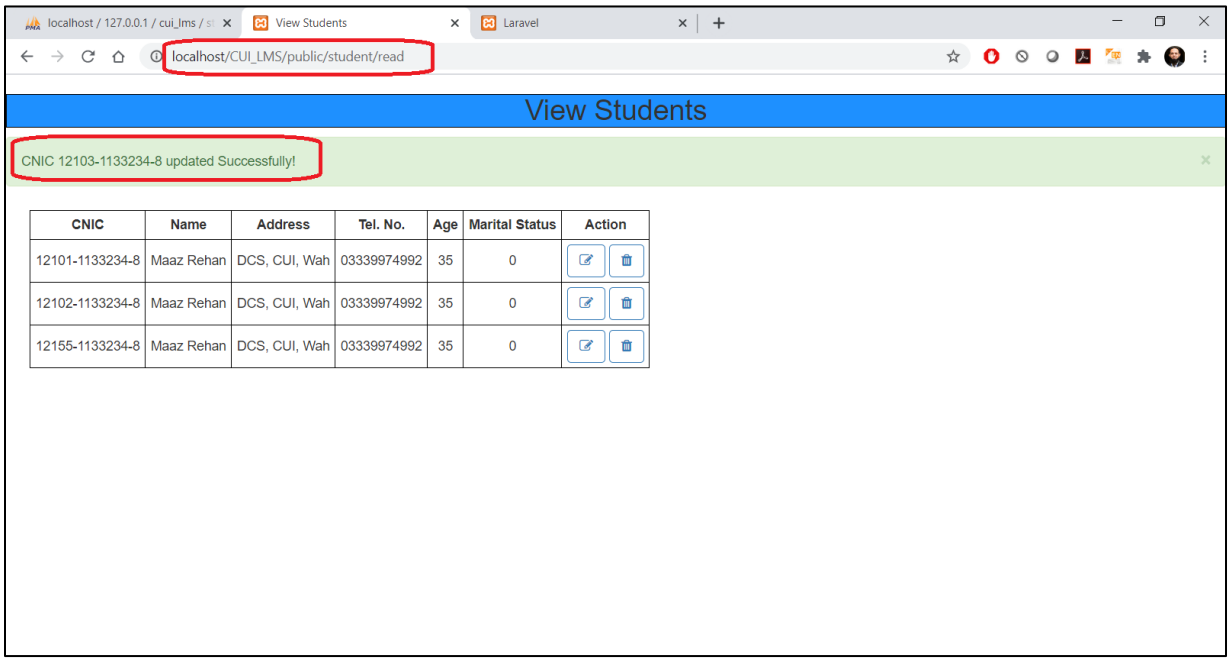
```
        // Help on the following code is given at the following URL
        // https://laravel.com/docs/5.8/redirects#redirecting-with-flashed-session-data
        //
        return redirect('student/read')
            ->with('status', 'CNIC '.$cnic.' updated Successfully!');
        // -------------------------------------
    }
```

D. How update webpage opens and saves data?

a) When user clicks the Edit button of any student record on the read webpage, the cnic is passed to the edit() function of StudentController. The cnic record is fetched from the students table and then passed to the update webpage as shown below.



b. When user clicks the Update button, the record is saved in the **students** table, the updated contents of students table are fetched and passed to the **read webpage** as shown below.



4. Applying MVC on the Delete operation of Student

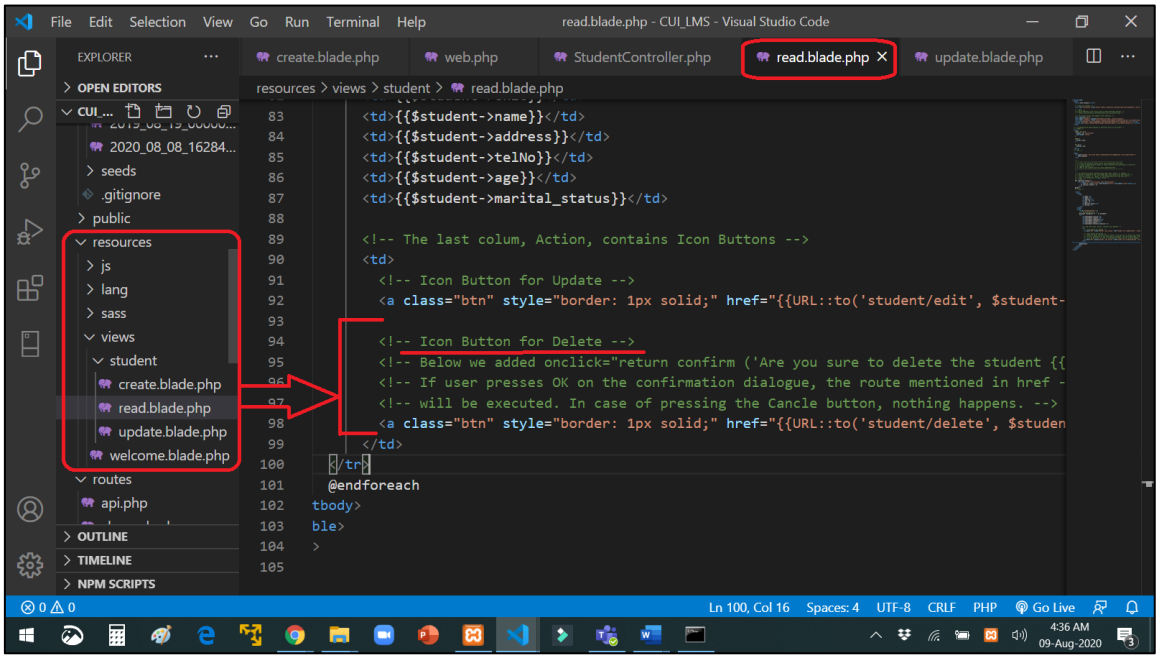
A. Student Model Already Exists. So, skip this step.

B. Write code for the 'delete' Webpage (View) and add Routes

a) The webpage, **delete.blade.php** is NOT required. We can delete an item without creating a new page

b) The **Delete button** is in the Action column of **read.blade.php**

c) The code lines, **href="{{URL::to('student/delete', $student->cnic)}}"**, written in **read.blade.php** state that the route **student/delete** be invoked.



d) Adding Action-based Route for Delete button on the read webpage

iii. On the read page, when user clicks the Delete button of a record, the record of student is deleted, after user confirmation.

iv. To achieve this, we add the Route::get() function in **resources > web.php** file.

- Route::get('student/delete/{cnic}', 'StudentController@delete')->name('student.delete');
- Here,
  - student/delete/{cnic} is the user-defined URL which takes one argument which is the cnic of the student for whom Update is initiated

- StudentController@delete means call the delete() function in the StudentController
- name('student.delete') is a user-defined alternate name of the route

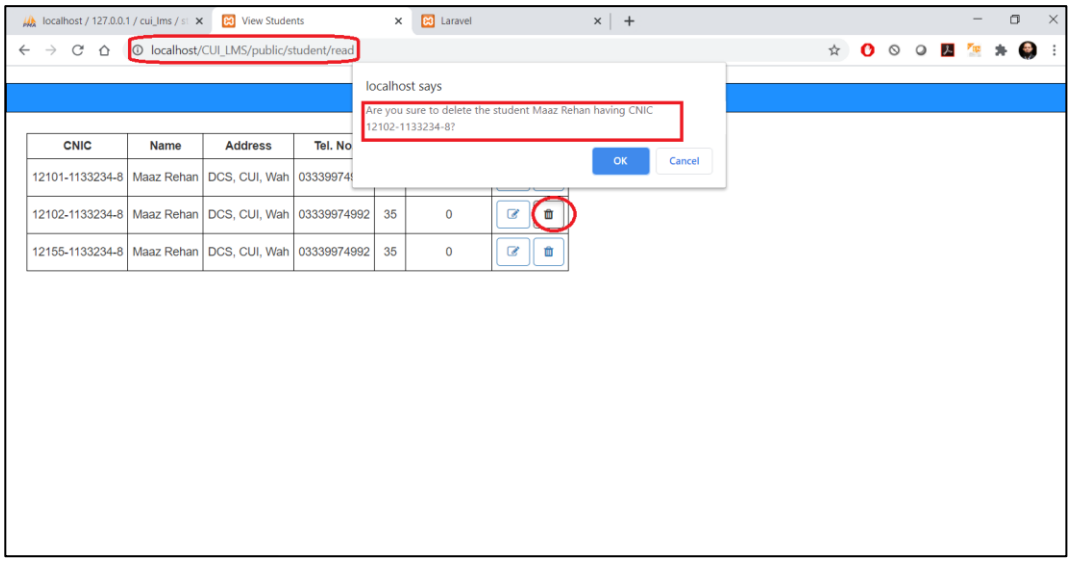
C. StudentController exists, so add functions related to update operation

c) The **delete()** function is given below which is invoked when Delete button in the Action column on the **read webpage** is clicked. This function deletes the record of cnic from **students table** and reloads the read webpage with the new contents of students table

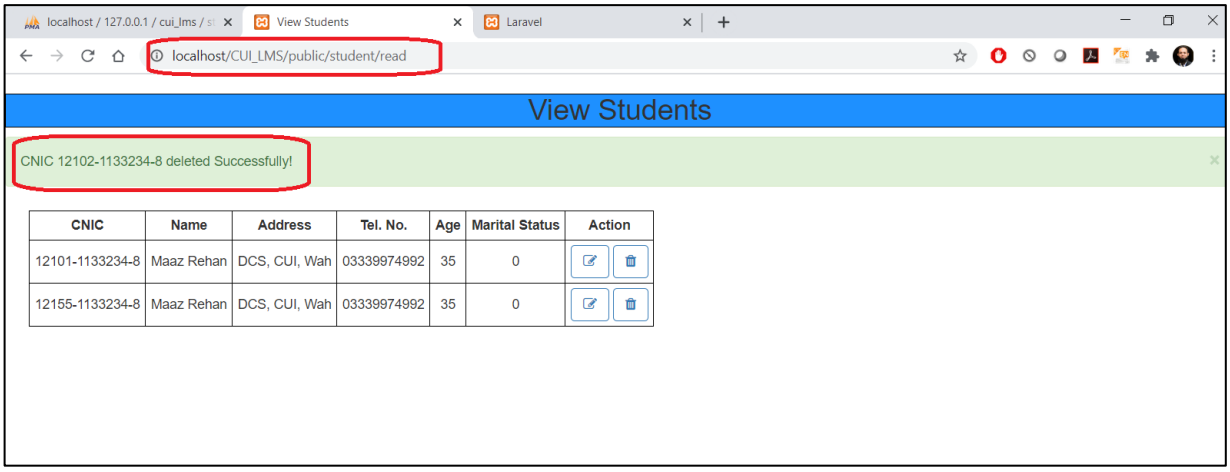
```
public function delete($cnic) {

    // Delete the row pointed to by this CNIC
    Student::destroy($cnic);

    // -------------------------------------
    // Help on the following code is given at the following URL
    // https://laravel.com/docs/5.8/redirects#redirecting-with-flashed-session-data
    //
    return redirect('student/read')
        ->with('status', 'CNIC '.$cnic.' deleted Successfully!');
    // -------------------------------------
}
```

D. How delete operation takes place?

a) When user clicks the Delete button of any student record on the **read webpage**, it asks for confirmation.



b) If confirmed, the cnic is passed to the delete() function of StudentController. The cnic record is deleted from the **students table.** the updated contents of students table are fetched and passed to the **read webpage** as shown below.



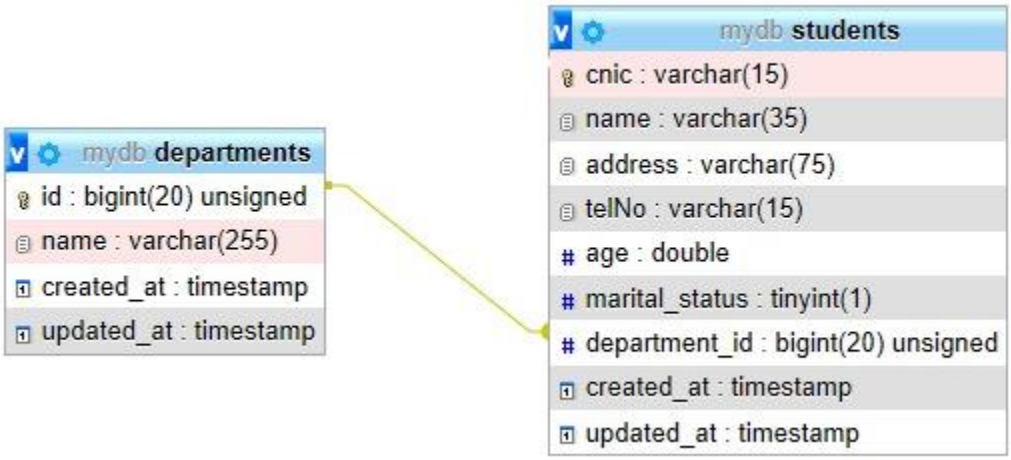
This completes our tutorial on Entity-wise CRUD in Laravel using MVC. Once you make files for the entity Student, copy them and re-use them with some modification for the entities Department, Course, Teacher, Admission_Office, etc.

***

One-to-Many (1xM) Relationships in Laravel

In a one-to-many relationship, one record in a table is normally associated with one or more records in another table.

ER Diagram:



The relation between the entities Department and Student is One-to-Many. A department can have many students while a student has only one department. The primary key('id') in the table **departments** should be foreign key ('department_id') in the table **students.** This is shown in the above ER diagram.

The above diagram speaks that foreign key needs to be added to Student entity and database level relationships needs to be defined on both sides.

Note: Make sure you have created Department entity and written its CRUD operations.

1. Applying MVC on the Create operation of Student

A. Update Student Model

a) **Update migration script to add foreign key in the table students**

i. According to Laravel convention, foreign key name should be <entity_name>_<primay_key>. Thus, in our case, **foreign key** in the **students table** will become **department_id**

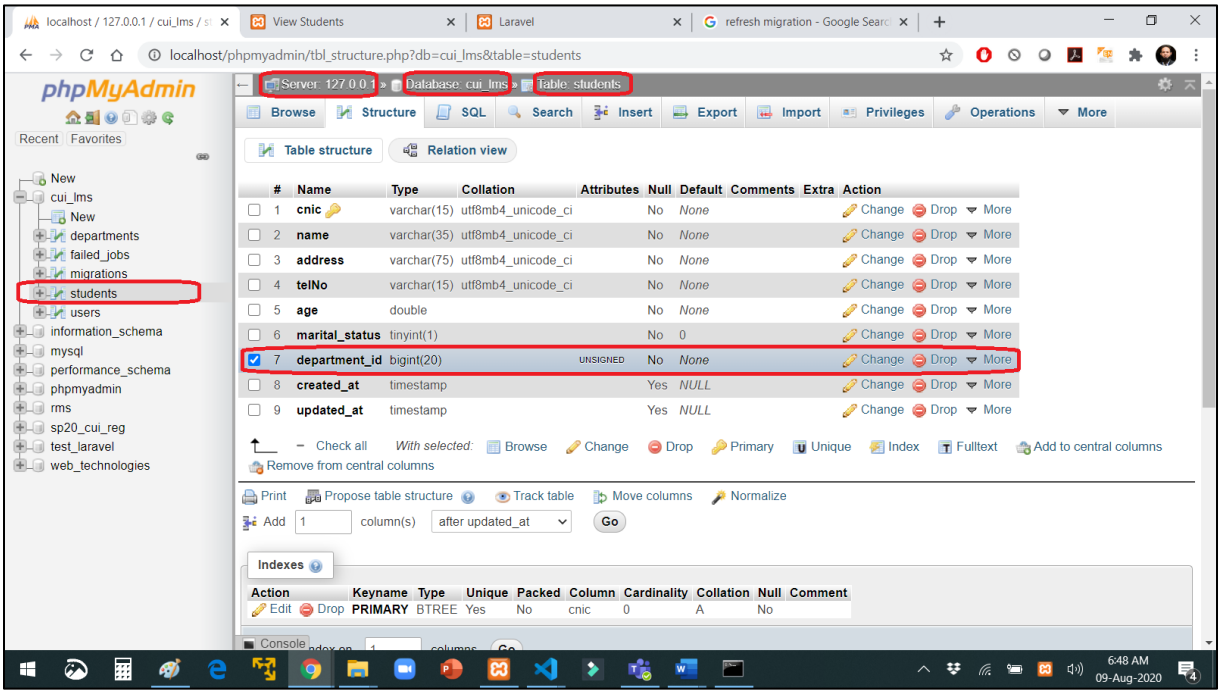
ii. Add the following lines in the up() function of CreatesStudentsTable migration

```
// The departments table MUST exist and MUST have 'id' as Primary key
$table->unsignedbiginteger('department_id');
```

iii. Migrate using the following command

- php artisan migrate:fresh

iv. The snapshot of students table below does have department_id fields but does not have a foreign key symbol



v. **Making foreign key relationship at the database level.** Add the following lines in the up() function of CreatesStudentsTable migration

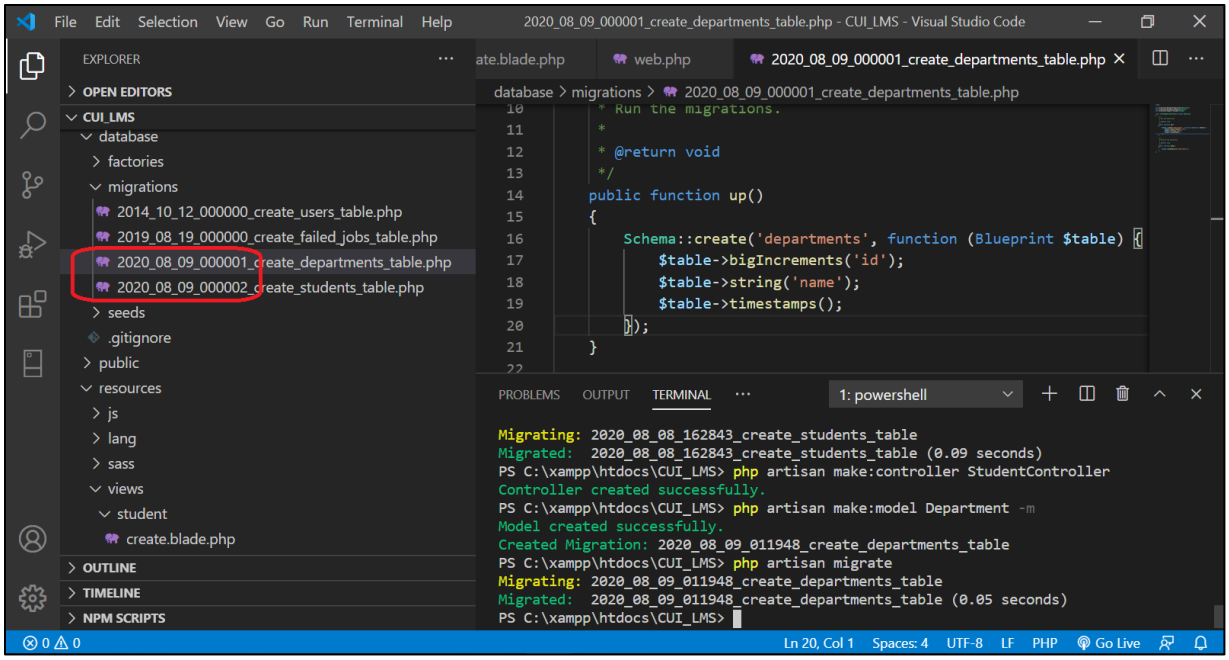
```
        // This will create relationship at the DBMS level.
        // So, a grey colour foreign key must appear in the students table
        // after performing this migration
        $table->foreign('department_id')->references('id')->on('departments')
            ->onDelete('cascade');
```

vi. After adding foreign key and the foreign key constraint, the up() function looks as below

```
public function up()
{
    Schema::create('students', function (Blueprint $table) {
        $table->string ('cnic', 15)->primary();
        $table->string('name', 35);
        $table->string('address', 75);
        $table->string('telNo', 15);
        $table->double('age');
        $table->boolean('marital_status')->default(false);

        // The departments table MUST exist and MUST have 'id' as Primary key
        $table->unsignedbiginteger('department_id');

        // This will create relationship at the DBMS level.
        // So, a grey colour foreign key must appear in the students table
        // after performing this migration
        $table->foreign('department_id')->references('id')->on('departments')
            ->onDelete('cascade');

        $table->timestamps();
    });
}
```

vii. **Rename Migration Files when having Foreign Key constraint**

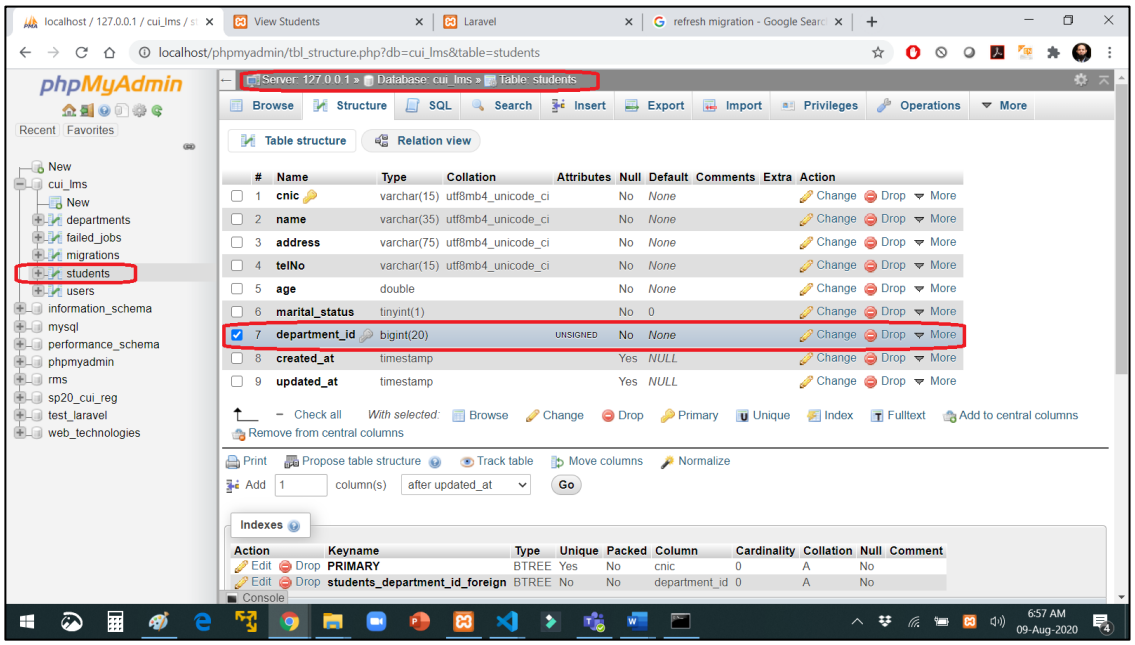
- In my case, Department migration was done later, so the timestamp of student migration is earlier than the timestamp of department migration
- Since **Primary Key of departments table** is **Foreign Key in students table**, therefore, timestamp of Department must be earlier than Student otherwise students table having foreign key would be created earlier than the departments table thereby giving Foreign Key constraint issue



viii. Use the following command to perform migration so that students table can have foreign key constraint at the database level.

- php artisan migrate:fresh

ix. The snapshot of students table below **now has foreign key symbol** which means MySQL database will make sure the foreign key constraint.



x. **Making foreign key relationship at the Laravel level.**

- In the **Department model**, write the function **students()**. It will be read as, a department hasMany students.

```
class Department extends Model
{
    /**
     * Get the students (Many) for the department (One).
     */
    public function students()
    {
        return $this->hasMany(Student::class);
    }
}
```

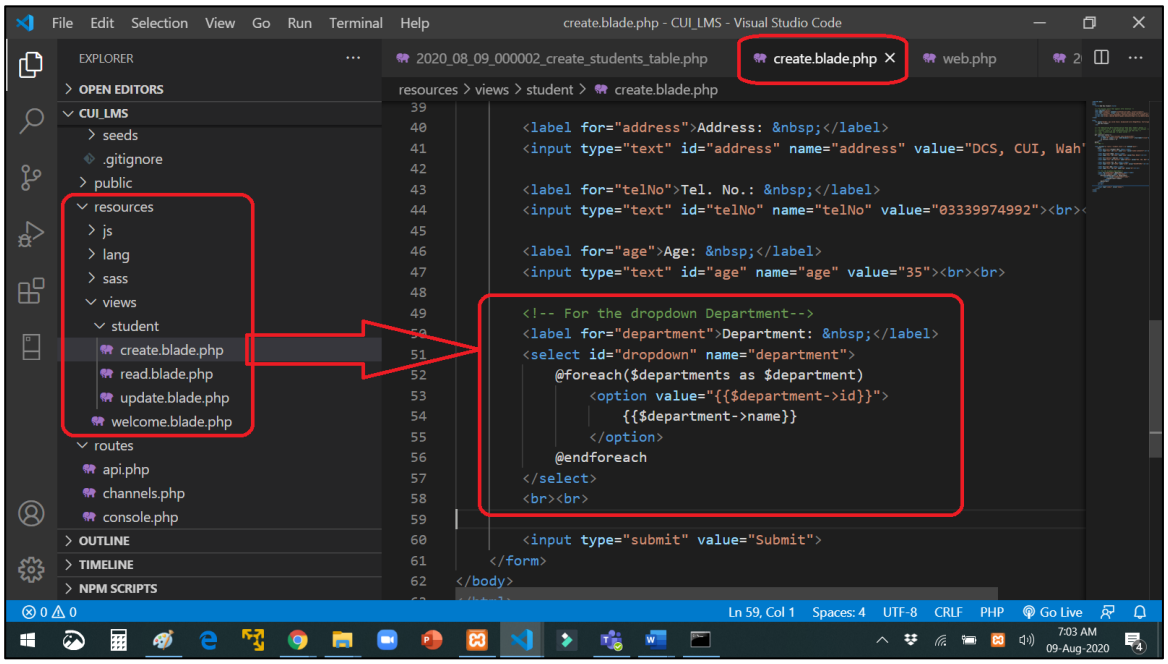
- In the **Student model**, write the function **department()**. It will be read as, one student belongsTo one department.

```
class Student extends Model
{
    protected $primaryKey = 'cnic'; // Set as primary key
    public $incrementing = false;   // Non auto-incrementing

    /**
     * Inverse relationship: Get the Department that owns the Student.
     */
    public function department()
    {
        return $this->belongsTo(Department::class);
    }

}
```

- **Hint:** Use **belongsTo** in the Model whose table contains Foreign Key

B. Write code for the create Webpage (View) and add Routes

a) Add code for Department dropdown in **create.blade.php**



b) Updated code of **create.blade.php**

```
<!DOCTYPE html>
<html>
<head>
    <title>Add New Student</title>

    <!-- For Success alert that appears after deletion -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.c
ss">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>

<body>
    <h2 style="border: 1px solid black; background-color:DodgerBlue; text-align:center;">
        Add New Student
    </h2>

    <!-- For Redirecting With Flashed Session Data when 'Submit' button -->
    <!-- is pressed in the 'create.blade.php' view which calls the relevant -->
    <!-- function 'store' in the StudentController and then this -->
```

```
    <!-- view, 'create.blade.php' is again called -->
    <!-- START -->
    @if (session('status'))
        <div class="alert alert-success alert-dismissible">
            <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
            {{ session('status') }}
        </div>
    @endif
    <!-- END -->

    <form action="{{ route ('student.store') }}" method="post">
        @csrf
        <label for="cnic">Student CNIC:  </label>
        <input type="text" id="cnic" name="cnic" value="12101-1133234-8"><br><br>

        <label for="name">Name:  </label>
        <input type="text" id="name" name="name" value="Maaz Rehan"><br><br>

        <label for="address">Address:  </label>
        <input type="text" id="address" name="address" value="DCS, CUI, Wah"><br><br>

        <label for="telNo">Tel. No.:  </label>
        <input type="text" id="telNo" name="telNo" value="03339974992"><br><br>

        <label for="age">Age:  </label>
        <input type="text" id="age" name="age" value="35"><br><br>

        <!-- For the dropdown Department-->
        <label for="department">Department:  </label>
        <select id="dropdown" name="department">
            @foreach($departments as $department)
                <option value="{{$department->id}}">
                    {{$department->name}}
                </option>
            @endforeach
        </select>
        <br><br>

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

c) All routes in **web.php** for **create webpage** will remain the same. No change. Only mentioning here for the sake of completeness.

- Route::get('student/create}', 'StudentController@create')->name('student.create');
- Route::post('student/store}', 'StudentController@store')->name('student.store');

C. Update create related functions in the StudentController

a) Include the Model Department in StudentController

- use App\Department;

b) Updated **create()** function is given below. It now contains department.

```
public function create() {
    $departments = Department::all();   // Load all departments. To add in the dropdown
    return view ("student/create")
        ->with(['departments' => $departments]);
}
```

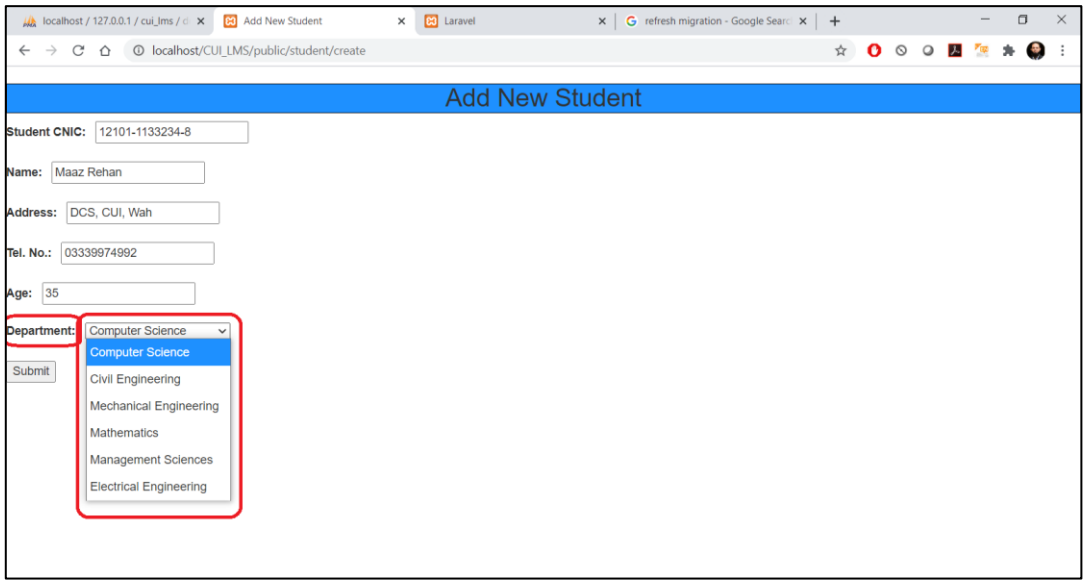
c) Updated **store()** function is given below. It now contains department_id.

```
public function store(Request $request) {

    $student = new Student;     // Must import the Model file: use App\Student;
    $student->cnic = $request->get('cnic');
    $student->name = $request->get('name');
    $student->address = $request->get('address');
    $student->telno = $request->get('telNo');
    $student->age = $request->get('age');

    // This is Foreign Key value in Student Table
    // It is taken from Drop Down list which was
    // populated in Create() from the Department Table
    $student->department_id = $request->get('department');


    // Since the marital_status field has a default value of ZERO,
    // therefore, even if no text is copied from the text box
    // the value ZERO would be stored.
    //$student->marital_status = $request->get('marital_status');
```

```
        $student->save();       /* Store data inside the table */

        // -------------------------------------
        // Help on the following code is given at the following URL
        // https://laravel.com/docs/5.8/redirects#redirecting-with-flashed-session-data
        //
        return redirect('student/create')
            ->with('status', 'CNIC '.$student->cnic.' added Successfully!');
        // -------------------------------------
    }
```

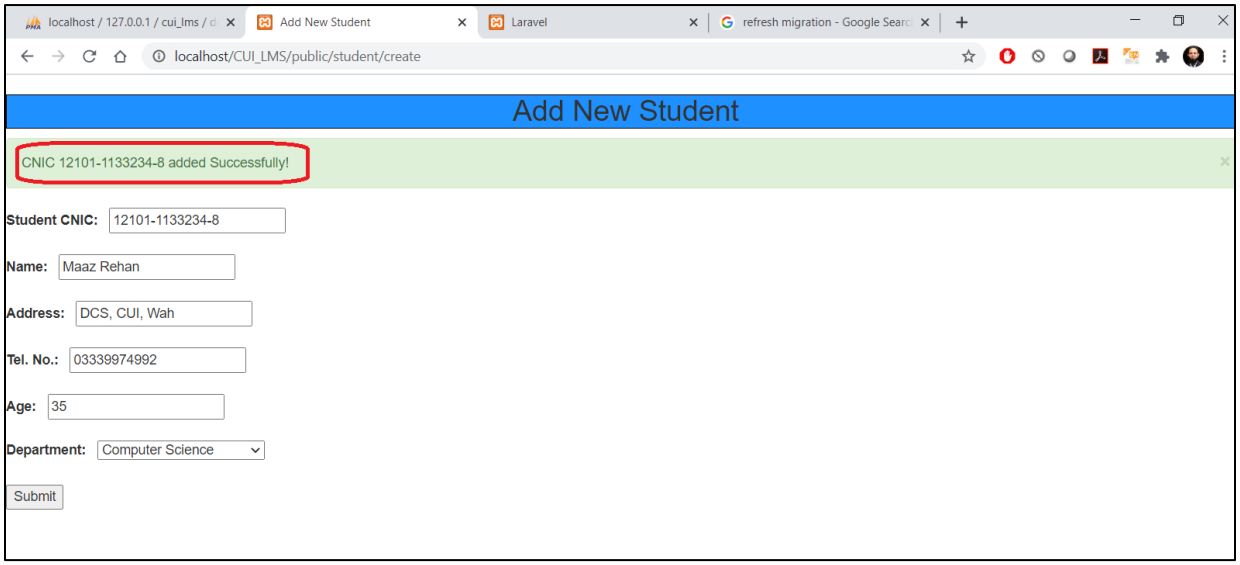
D. How create webpage opens and data is stored ?

a) When user types the URL http://localhost/CUI_LMS/public/student/create in the browser, , route with the name student.create is searched in web.php file. When found, the StudentController function create() is searched and executed. The create() function displays the Add New Student webpage, as shown below.



i. When user presses the submit button, the route student.store is searched in web.php. When student.store route is found, the StudentController function store() is searched and executed. The store function stores form data in the students table and then displays the create page.

ii. Add few students so that we can view / update / delete them


localhost / 127.0.0.1 / cui_lms /
Add New Student
Laravel
refresh migration - Google Searc
localhost/CUI_LMS/public/student/create
Add New Student
CNIC 12101-1133234-8 added Successfully!
Student CNIC: 12101-1133234-8
Name: Maaz Rehan
Address: DCS, CUI, Wah
Tel. No.: 03339974992
Age: 35
Department: Computer Science
Submit

2. Applying MVC on the Read operation of Student

A. Student Model Already Exists. So, skip this step.

B. Update code for the 'read' Webpage (View)

a) Add the following code in **read.blade.php** to support **fetch name from the departments table** based on the **department_id in the students table**

i. Add column Department in the table on read.blade.php

```
<th>Department</th>
```

ii. Add Department name in the each row of the table on read.blade.php

```
<!--
  We shall only get the department id that have been inserted in
  the table Student.
  <td>{{$student->department_id}}</td>

  Using the inverse relation defined in Student.php model,
  laravel helps us fetch any attribute of "departments" table
  based on the Foreign Key stored in "students" table.
  Below, the name of inverse relationship in Student.php model
  is department().
-->
<td>{{$student->department->name}}</td>
```

a) All routes in **web.php** for **read webpage** will remain the same. No change. Only mentioning here for the sake of completeness.

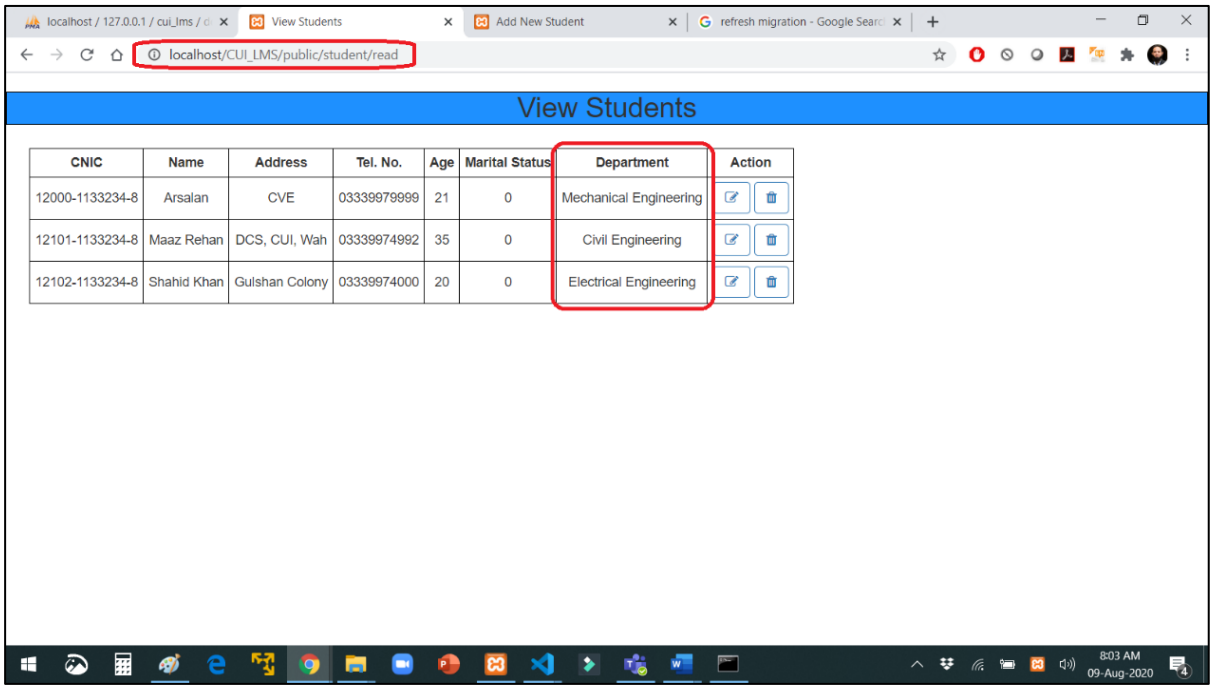
- Route::get('student/read}', 'StudentController@read')->name('student.read');

C. The read() function in StudentController does not need change

D. How read webpage opens and shows data?

a) When user types the URL http://localhost/CUI_LMS/public/student/read in the browser, route with the name student.read is searched in web.php file. When found,

the StudentController function read() is searched and executed. The read() function fetches data from the database and displays them as shown below.



3. Applying MVC on the Update operation of Student

A. Student Model Already Exists. So, skip this step.

B. Add code for the 'update' Webpage (View)

a) Add the following code in **update.blade.php** to accommodate department dropdown

```
<!-- For the Department dropdown -->
<label for="department">Department:  </label>
<select id="dropdown" name="department">
    @foreach($departments as $department)
        <option value="{{$department->id}}">
            {{$department->name}}
        </option>
    @endforeach
</select>
<br><br>
```

b) No change in routes for update operation. Providing routes for the sake of completeness.

- Route::get('student/edit/{cnic}', 'StudentController@edit')->name('student.edit');
- Route::post('student/update/{cnic}', 'StudentController@update')->name('student.update');

C. Adding code to the functions related to update operation

a) After adding code to accommodate Department, new edit() function looks as below.

```
public function edit($cnic) {

    $students = Student::find($cnic);  // Load student using the model 'Student'
    $departments = Department::all();  // Load departments using the model Department'

    // Pass the $students to the view, 'student/update'
    // so that user can update.
    return view('student/update')
```

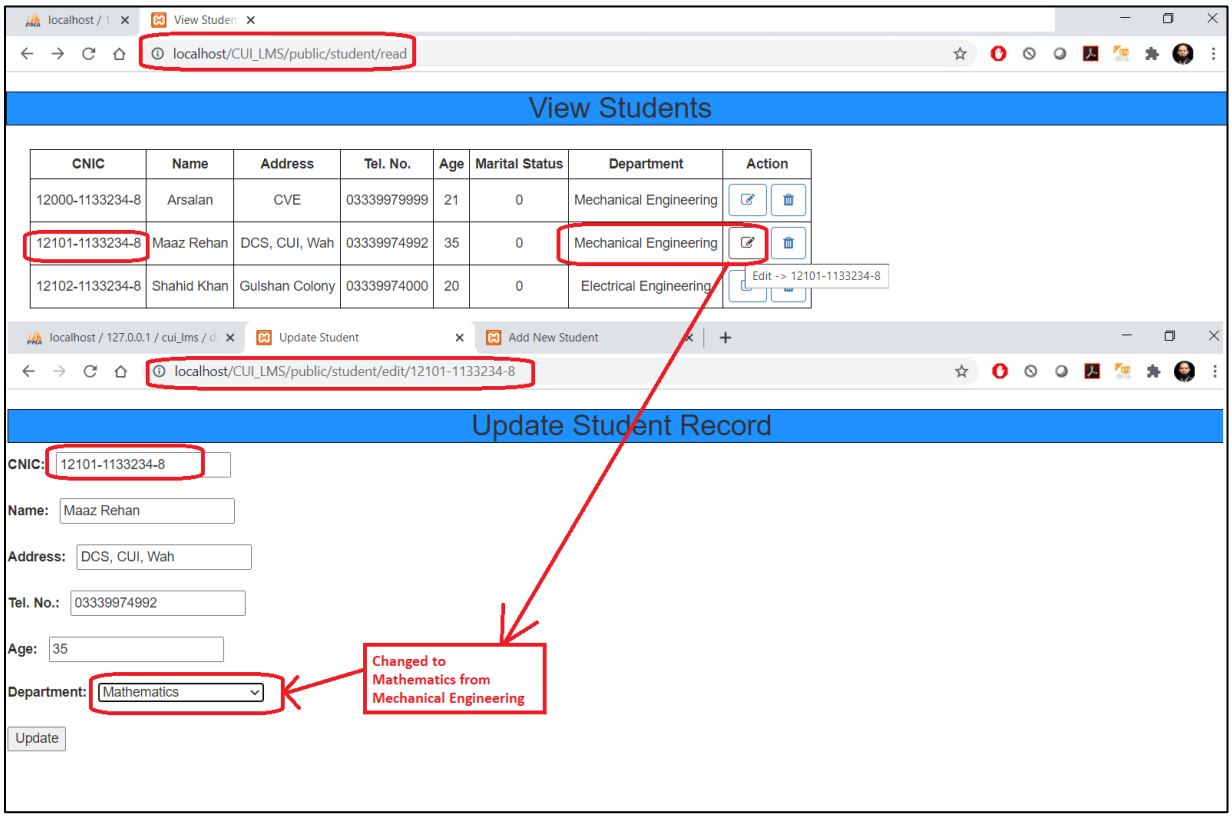
```
            ->with(['students' => $students])
            ->with(['departments' => $departments]);
    }
```

a) Add the following line of code to accommodate department in the update() function.

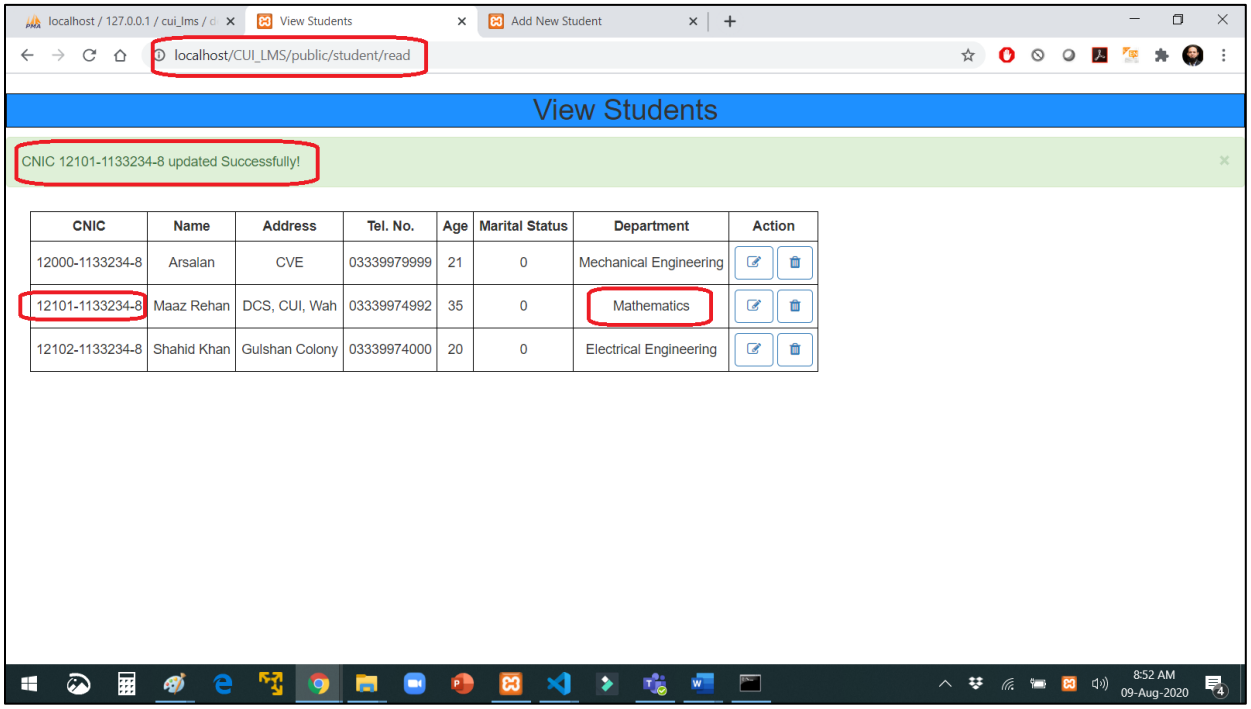
```
        $student->department_id = $request->get('department');
```

D. How update webpage opens and saves data?

b) When user clicks the Edit button of any student record on the read webpage, the cnic is passed to the edit() function of StudentController. The cnic record is fetched from the students table and then passed to the update webpage as shown below.



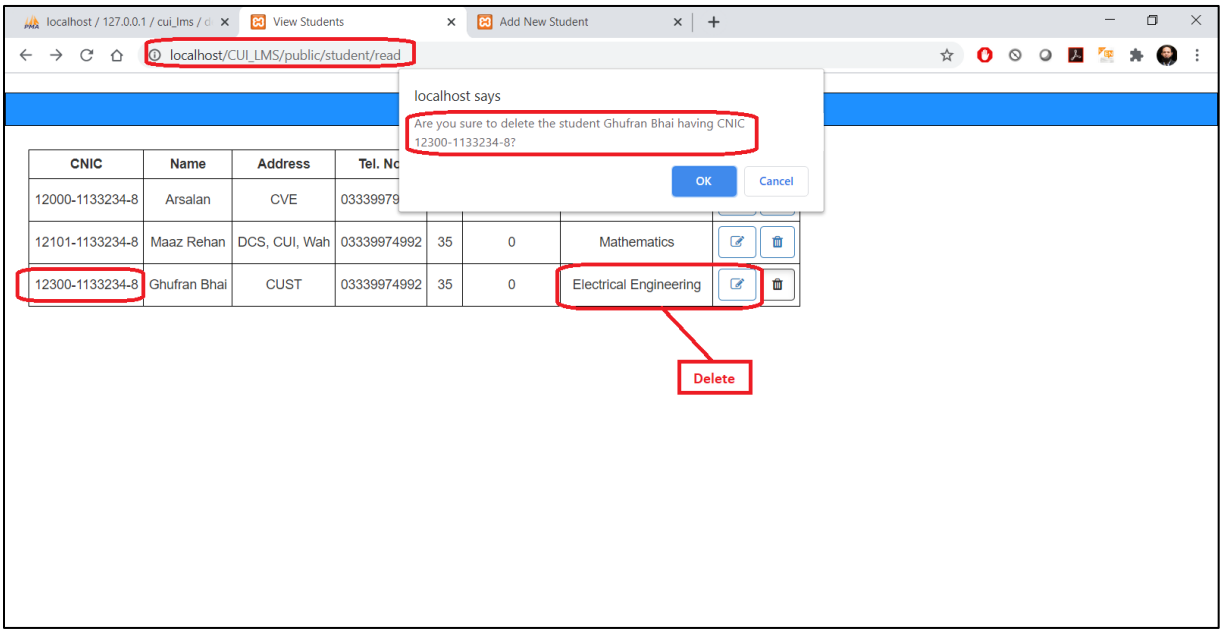
a. When user clicks the Update button, the record is saved in the **students** table, the updated contents of students table are fetched and passed to the **read webpage** as shown below.



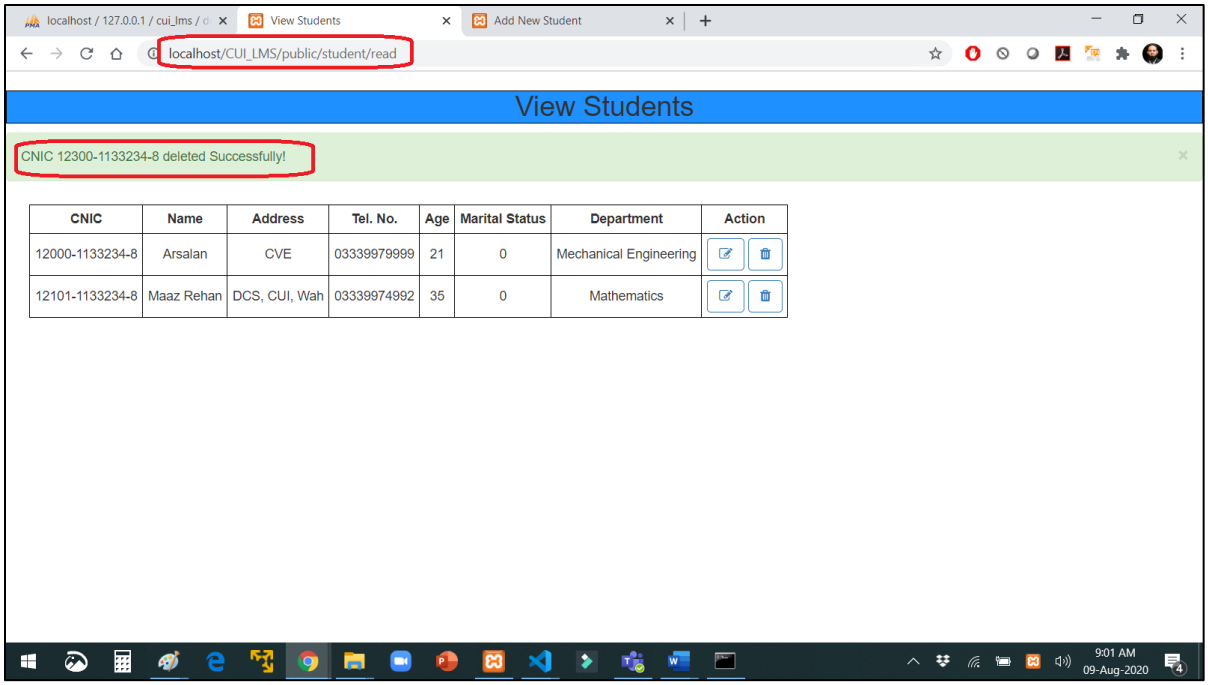
4. Applying MVC on the Delete operation of Student

A. Student Model Already Exists. So, skip this step.
B. No change required in Views. Skip this step.
C. No change required in Controller functions. Skip this step.
D. How delete operation takes place?

a) When user clicks the Delete button of any student record on the **read webpage**, it asks for confirmation.



b) If confirmed, the cnic is passed to the delete() function of StudentController. The cnic record is deleted from the **students table.** the updated contents of students table are fetched and passed to the **read webpage** as shown below.



You can now implement all sort of 1xM relations in Laravel. Next, we guide the implementation of 1xM relation in Laravel

***

Many-to-Many (MxM) Relationships in Laravel

In a many-to-many relationship, one record in a table is normally associated with more than one records in the table and vice-versa. If two entities have MxM relationship, we break it into two 1xM relationships. A new table, called join table or pivot table, is created which connects those two tables. The relation from table 1 and table 2 to pivot table is 1xM, so there are two 1xM relations.

***