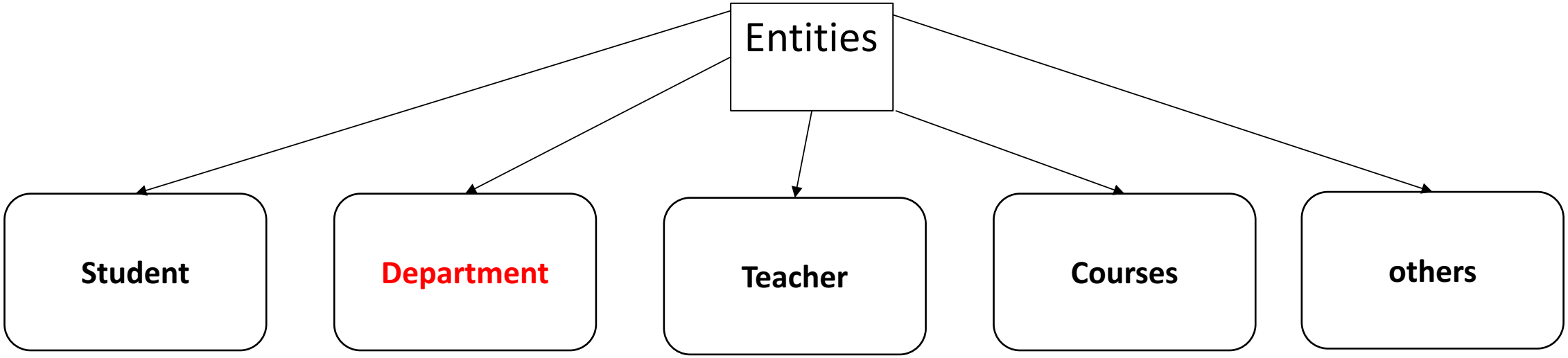
Laravel 1-to-Many, Many-to-Many CRUD Operation and Laravel Authentication

Dr. Majid Mumtaz

CUI, Wah

Courtesy to Dr. Maaz Rehan

Laravel CRUD Operation: Learning Management System



how to create one-to-many (1xM) and many-to-many (MxM) Relationships.

1.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. Create, Read, Update and Delete) of the CRUD operation.

NOTE: The Laravel commands used below needs 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.**

Model View Controller (MVC) and CRUD operation

- **Laravel is based on MVC design model**
- In MVC:
  - **Model** interacts with the database I.e. MySQL, SQLite etc.
  - **View** (or web page) interacts with the user through web clients (Chrome, IE etc.)
  - **Controller** contains functions which connect a View with its Model

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

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

Applying MVC on the Create operation of Department

- **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.

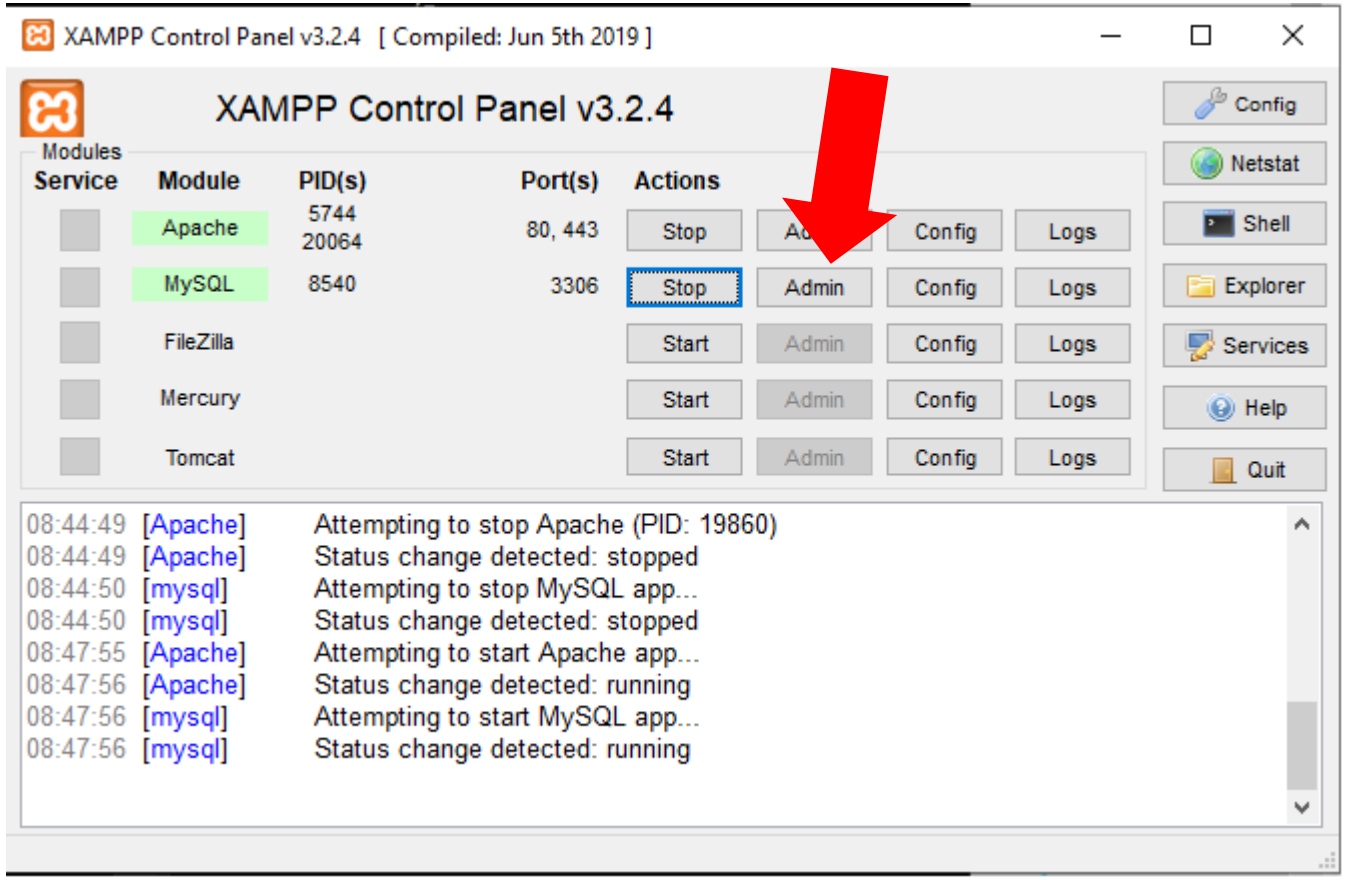
B. Create webpages (**Views**) to perform Create, Read, Update and Delete (CRUD) operations on the entity. Define route for each page which is a linkage between the webpage URL typed in the browser and a function. Thus, associated function is called when a user types a webpage URL in the browser.

C. Define one or more associated functions (**Controller Function**) of each webpage.

Student and Department Model already created

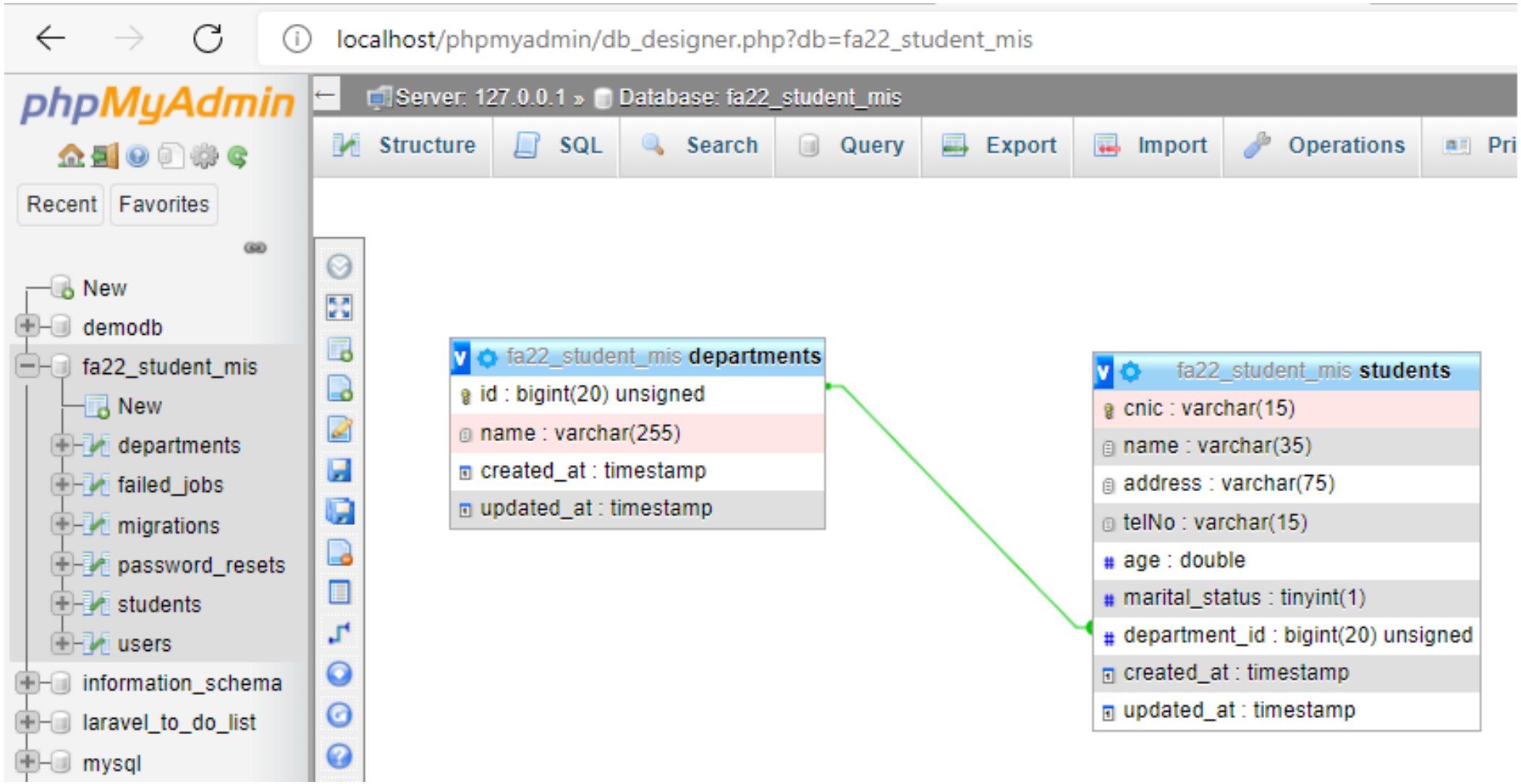
a) Open Fa22_student_mis database in MySQL , for that, follow steps:

1. Open XAMPP Control Penel. Start Apache and MySQL services.
2. Click **Admin** in front of MySQL to open **phpMyAdmin** in the browser.



ER Diagram of Students and Departments entities

3. In **phpMyAdmin**, database **FA22_Student_MIS by clicking on More → Designer,** we can see the ERD diagram as shown below after creating Departments entity model using Laravel framework.



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.

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 ER diagram speaks that foreign key needs to be added to Student entity and database level relationships needs to be defined on both ends.

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.**

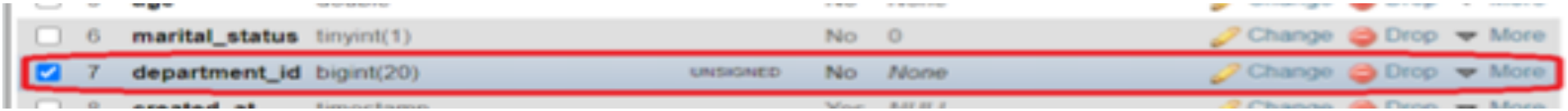
ii. Add the following lines in the up() function of CreatesStudentsTable migration

```
// 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



Applying MVC on the Create operation of Student

A. Update Student Model

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

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');
```

Applying MVC on the Create operation of Student

A. Update Student Model

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

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();
});
}
```

Applying MVC on the Create operation of Student

A. Update Student Model

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

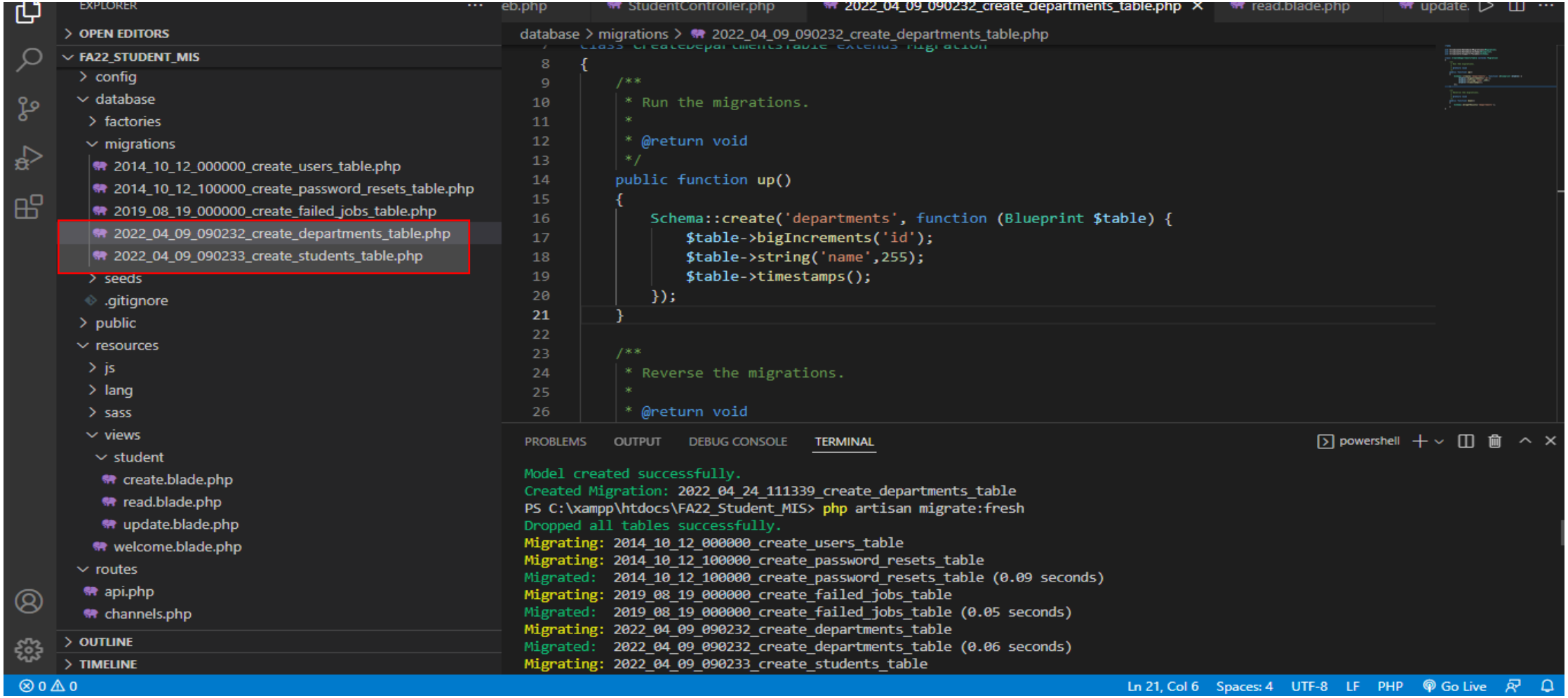
vii. **Rename Migration Files when having Foreign Key constraint**

- 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

See the screenshot on next slide

Applying MVC on the Create operation of Student

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



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

Applying MVC on the Create operation of Student

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

localhost/phpmyadmin/tbl_structure.php?db=fa22_student_mis&table=students

Server: 127.0.0.1 » Database: fa22_student_mis » Table: students

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	cnic	varchar(15)	utf8mb4_unicode_ci		No	None			Change Drop More
2	name	varchar(35)	utf8mb4_unicode_ci		No	None			Change Drop More
3	address	varchar(75)	utf8mb4_unicode_ci		No	None			Change Drop More
4	telNo	varchar(15)	utf8mb4_unicode_ci		No	None			Change Drop More
5	age	double			No	None			Change Drop More
6	marital_status	tinyint(1)			No	0			Change Drop More
7	department_id	bigint(20)		UNSIGNED	No	None			Change Drop More
8	created_at	timestamp			Yes	NULL			Change Drop More
9	updated_at	timestamp			Yes	NULL			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Fulltext Add to central columns

Remove from central columns

Print Propose table structure Track table Move columns Normalize

Add 1 column(s) after updated_at Go

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	cnic	2	A	No	
Edit Drop	students_department_id_foreign	BTREE	No	No	department_id	2	A	No	

phpMyAdmin

Recent Favorites

New
demodb
fa22_student_mis
New
departments
failed_jobs
migrations
password_resets
students
users
information_schema
laravel_to_do_list
mysql
performance_schema
phpmyadmin
student_mis
test
usersdb
wordpressdb

Console

Applying MVC on the Create operation of Student

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);
} }
```

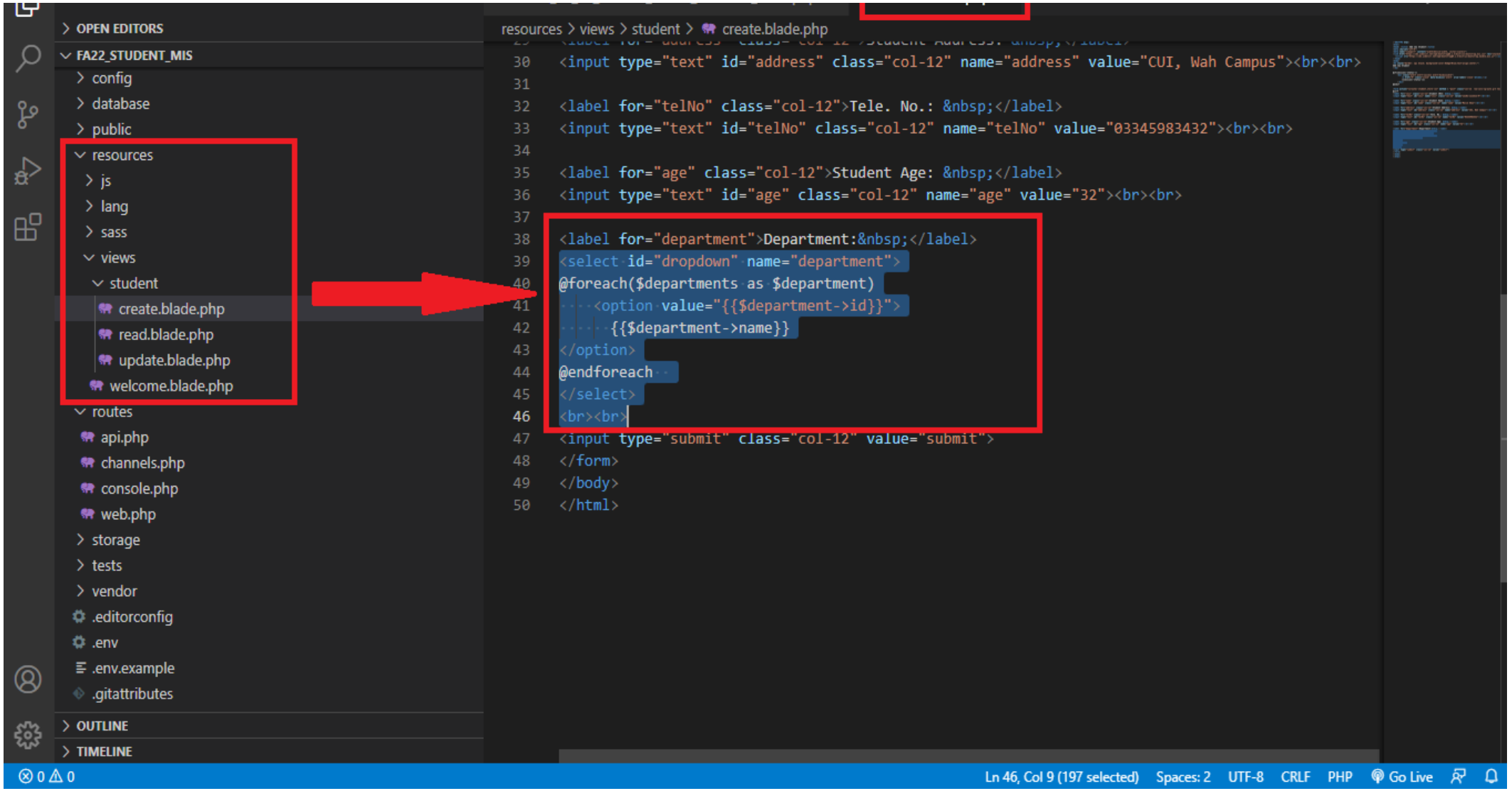
- In the **Student model**, write the function **department()**. It will be read as, one student **belongsTo** one department.

```
class Student extends Model {
        protected $primaryKey = 'cnic';
        public $incrementing = false;

       /* 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

Update create.blade.php Webpage (View) and add Routes

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



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

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>
```

Route Creation

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.

- **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.

- **Routes are of two types.**

  a. **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.
  b. **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.

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

a. Adding Webpage-based Route

1. When user wants to open a webpage, its URL is typed in the browser. In our case, if user types http://localhost/FA22_Student_MIS/public/student/create in the browser, then create webpage should open.
2. 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.

b. Adding Action-based Route

When user clicks the **Submit button**, data on the form should:

1. Store in the database table **students**.
2. To achieve this, a route related to click (submit) action needs to be defined
3. 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

Update create related functions in the StudentController

Update create related functions in the StudentController

a) Include the Model Department in StudentController

```
use App\Department;
```

b) Updated **create()** function in StudentController as given below. It now contains department.

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

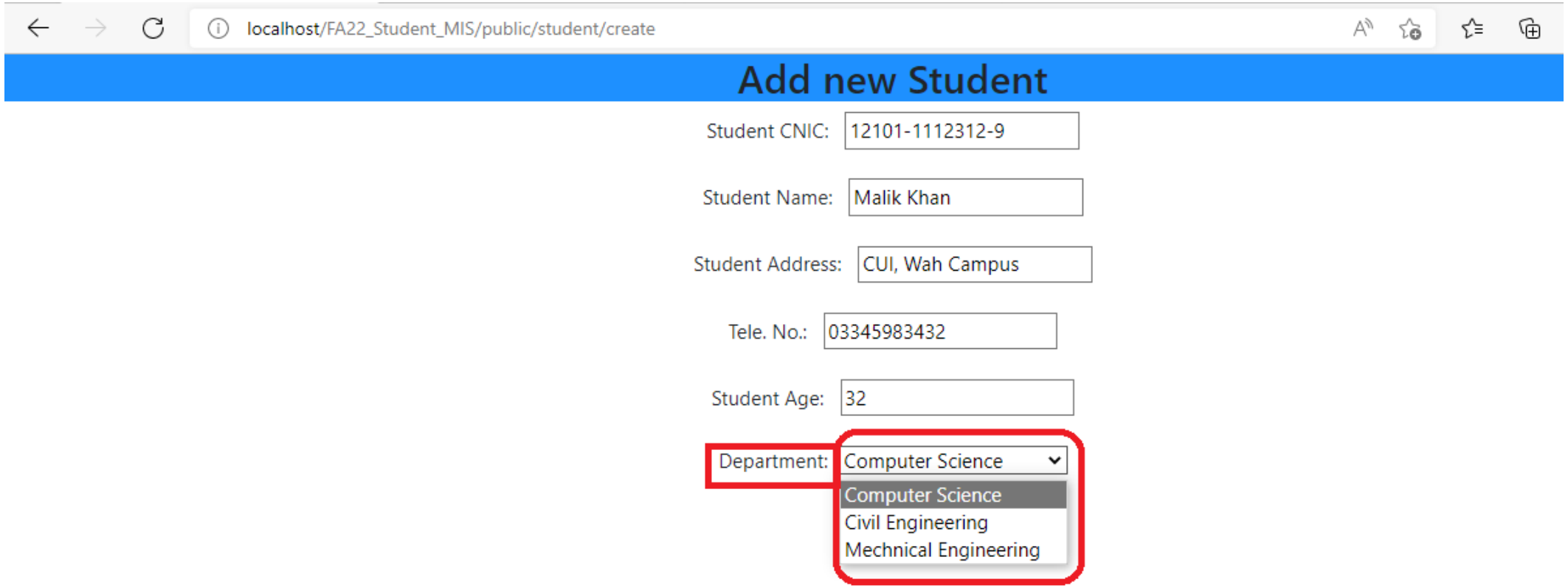
Update store related functions in the StudentController

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

        $student->save();
        return redirect('student/create')
                ->with('status', 'CNIC '.$student->cnic.' added Successfully!');
}
```

How create webpage opens and data is stored ?

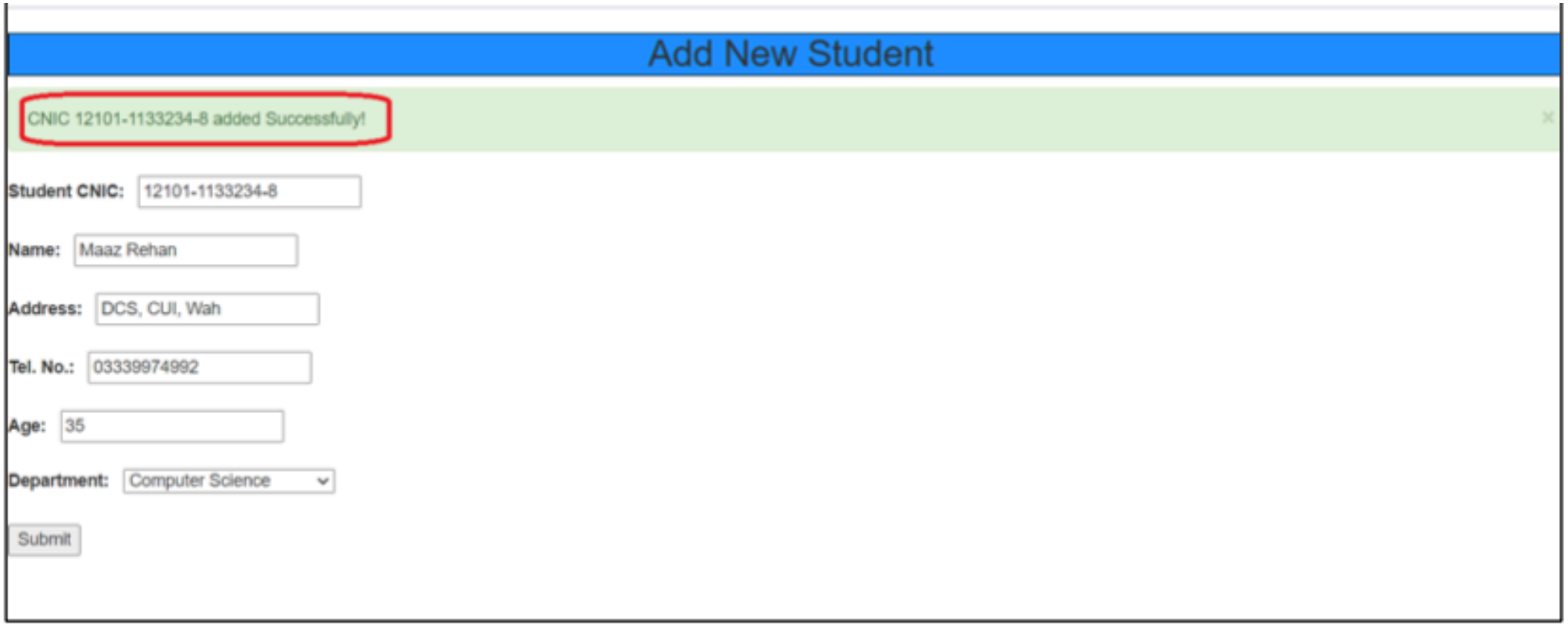
When user types the URL http://localhost/FA22_Student_MIS/public/student/create in the browser, route with the name student.create is searched in web.php file. When found, the StudentController function create() is searched and executed. The create() function displays the Add New Student webpage, as shown below.



How create webpage opens and data is stored ?

- 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.

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



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)

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**

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

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

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>
```

Adding Webpage-based Route

1. When user types http://localhost/FA22_Student_MIS/public/student/read in the browser, then read webpage should open.
2. 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

The read() function in StudentController does not need change

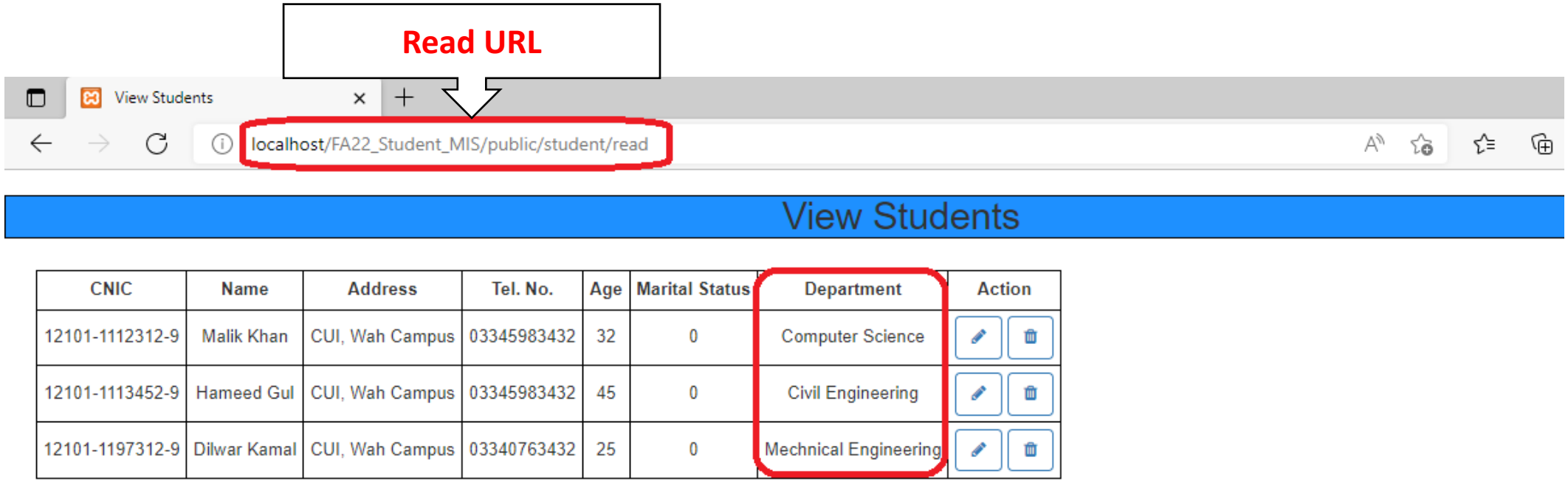
Student controller read function

a) The **read()** function is given below which is invoked when http://localhost/FA22_Student_MIS/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]);
}
```

How read webpage opens and shows data?

When user types the URL http://localhost/FA22_Student_MIS/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.



CNIC	Name	Address	Tel. No.	Age	Marital Status	Department	Action
12101-1112312-9	Malik Khan	CUI, Wah Campus	03345983432	32	0	Computer Science	
12101-1113452-9	Hameed Gul	CUI, Wah Campus	03345983432	45	0	Civil Engineering	
12101-1197312-9	Dilwar Kamal	CUI, Wah Campus	03340763432	25	0	Mechnical Engineering	

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>
```

Applying MVC on the Update operation of Student

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

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

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

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.

Applying MVC on the Update operation of Student

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

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

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

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

Applying MVC on the Update operation of Student

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

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);
     $departments = Department::all();

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

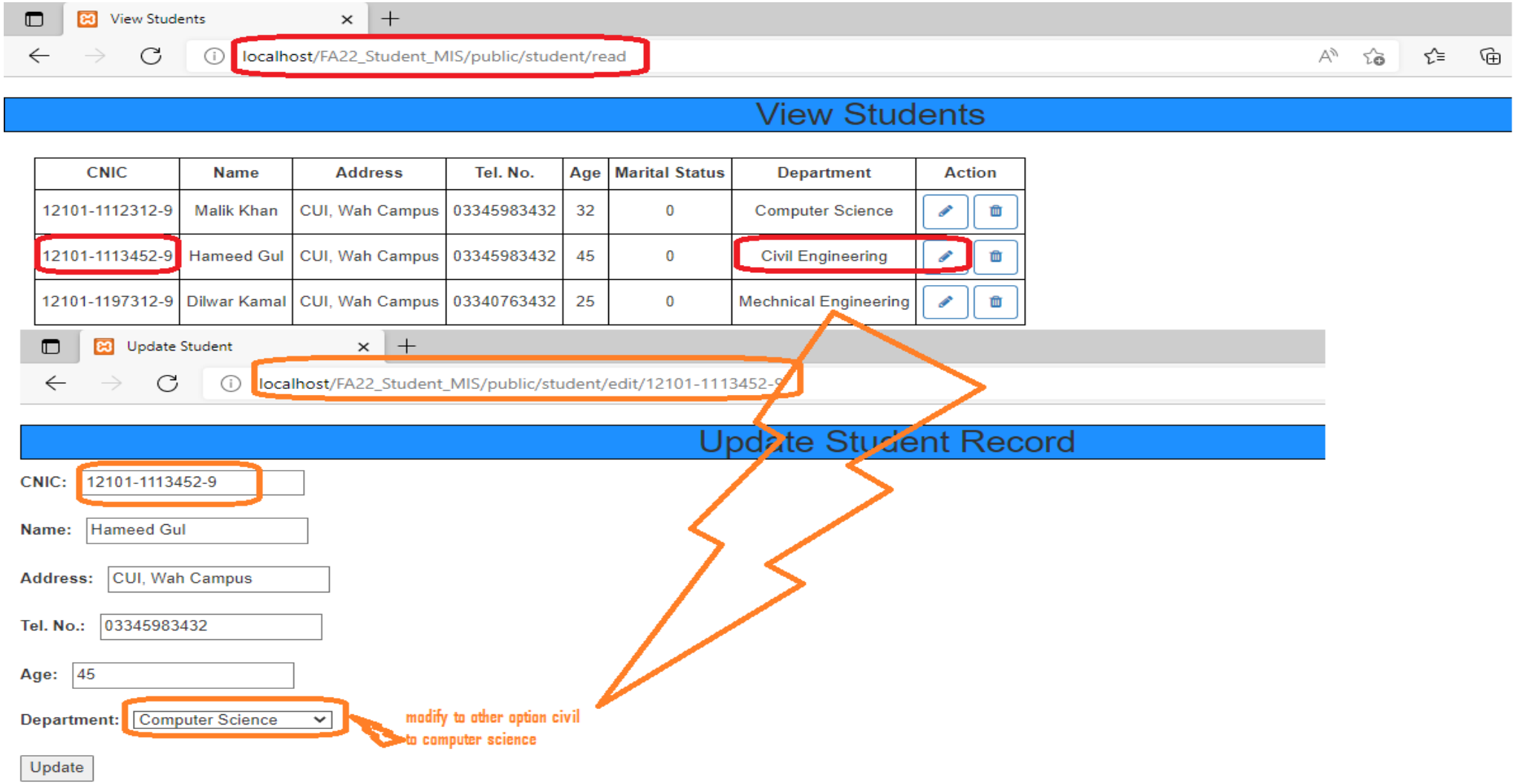
Add the following line of code to accommodate department in the update() function.

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

Applying MVC on the Update operation of Student

D. How update webpage opens and saves data?

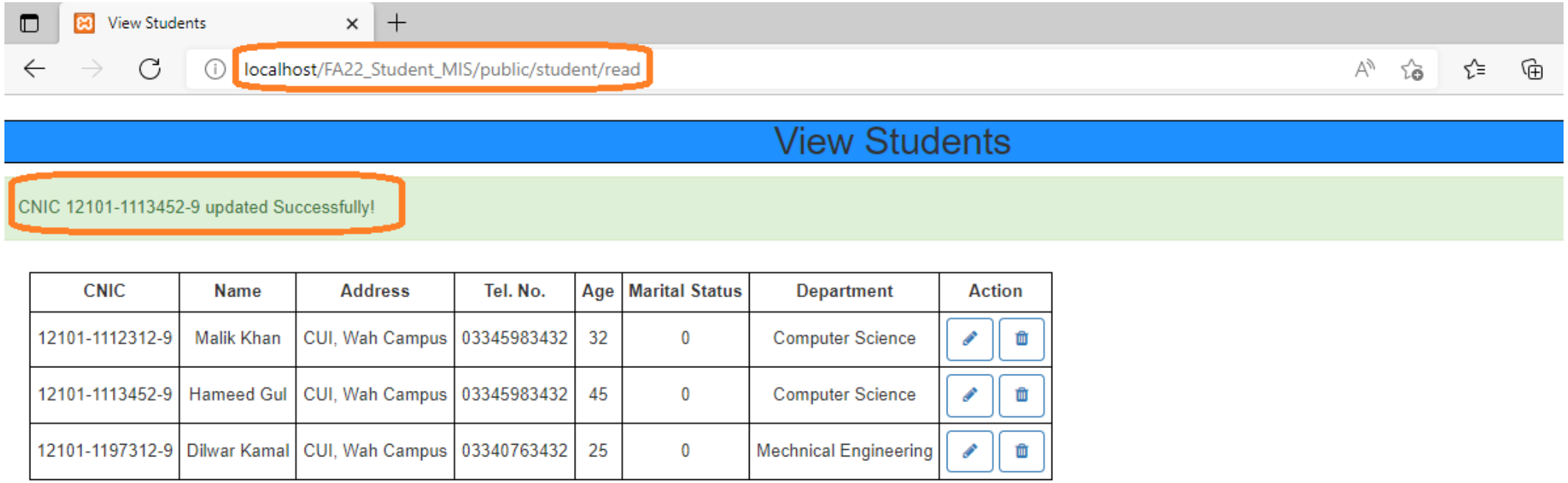
When user clicks the Edit button of any student record on the read webpage, the cnic is passed to the edit() function of StudentController. The cnic record is fetched from the students table and then passed to the update webpage as shown below.



Applying MVC on the Update operation of Student

D. How update webpage opens and saves data?

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.

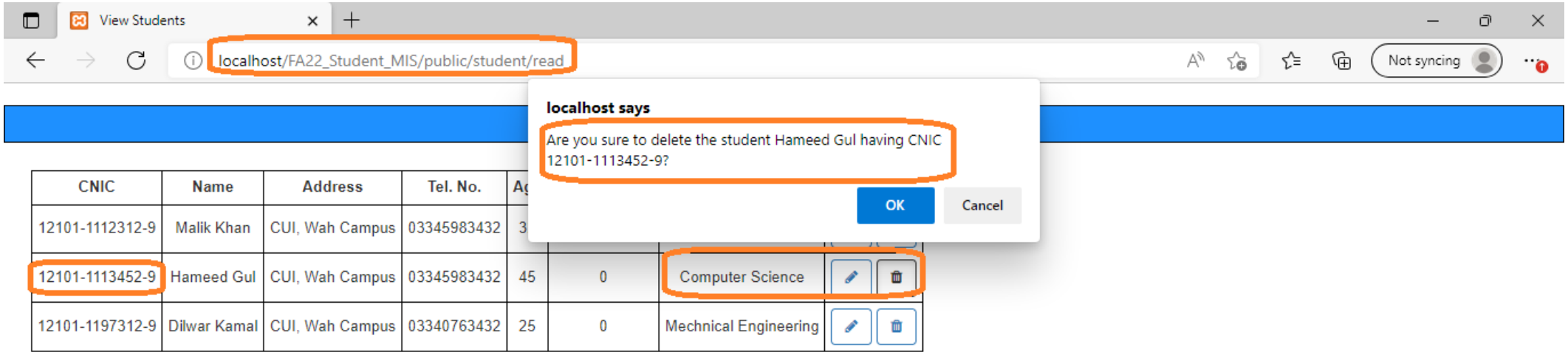


CNIC	Name	Address	Tel. No.	Age	Marital Status	Department	Action
12101-1112312-9	Malik Khan	CUI, Wah Campus	03345983432	32	0	Computer Science	
12101-1113452-9	Hameed Gul	CUI, Wah Campus	03345983432	45	0	Computer Science	
12101-1197312-9	Dilwar Kamal	CUI, Wah Campus	03340763432	25	0	Mechnical Engineering	

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?

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

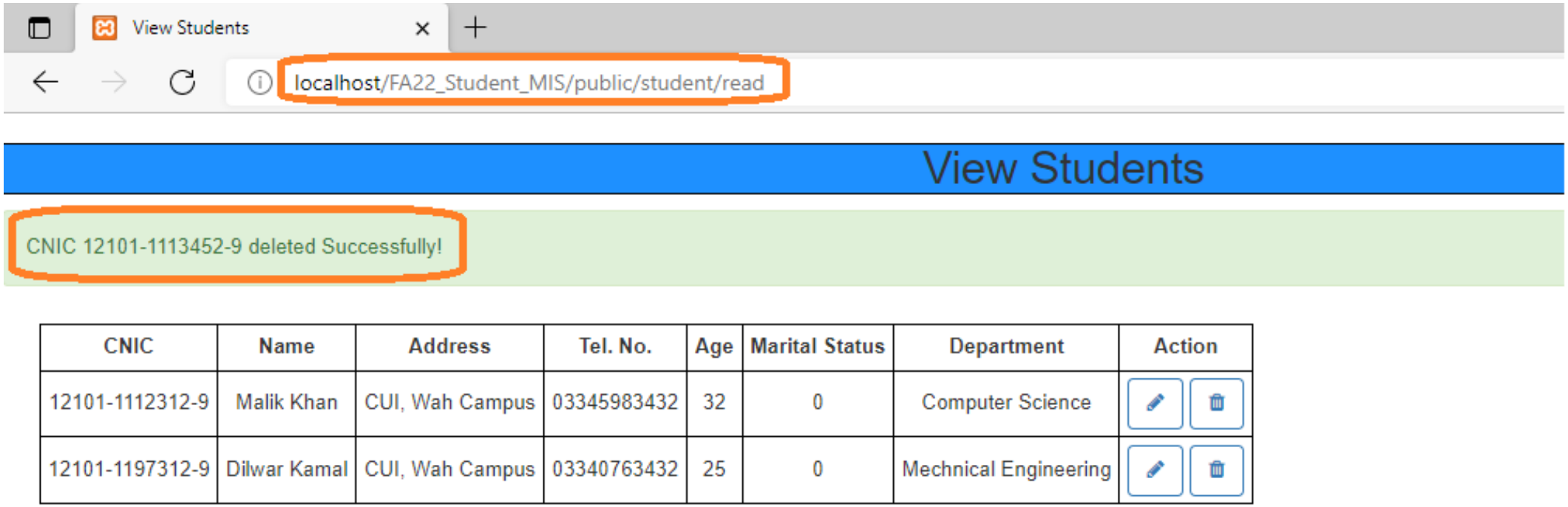


Applying MVC on the Delete operation of Student

D. How delete operation takes place?

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

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.



CNIC	Name	Address	Tel. No.	Age	Marital Status	Department	Action
12101-1112312-9	Malik Khan	CUI, Wah Campus	03345983432	32	0	Computer Science	
12101-1197312-9	Dilwar Kamal	CUI, Wah Campus	03340763432	25	0	Mechnical Engineering	

Example: Relationship based Laravel Project

Step 1: Configure Laravel Project

Create new project by using the following command.

Command Prompt

```
operable program or batch file.

C:\xampp\htdocs>composer create-project laravel/laravel --prefer-dist eloquentrelationshi
p
Creating a "laravel/laravel" project at "./eloquentrelationship"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v7.30.1)
  - Installing laravel/laravel (v7.30.1): Extracting archive
Created project in C:\xampp\htdocs\eloquentrelationship
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 99 installs, 0 updates, 0 removals
  - Locking asm89/stack-cors (v2.1.1)
  - Locking brick/math (0.9.3)
```

Example: Relationship based Laravel Project Step 1: Configure Laravel Project

Then in Command prompt open the newly created project directory and run the following command to start laravel service.

```
php artisan serve
```

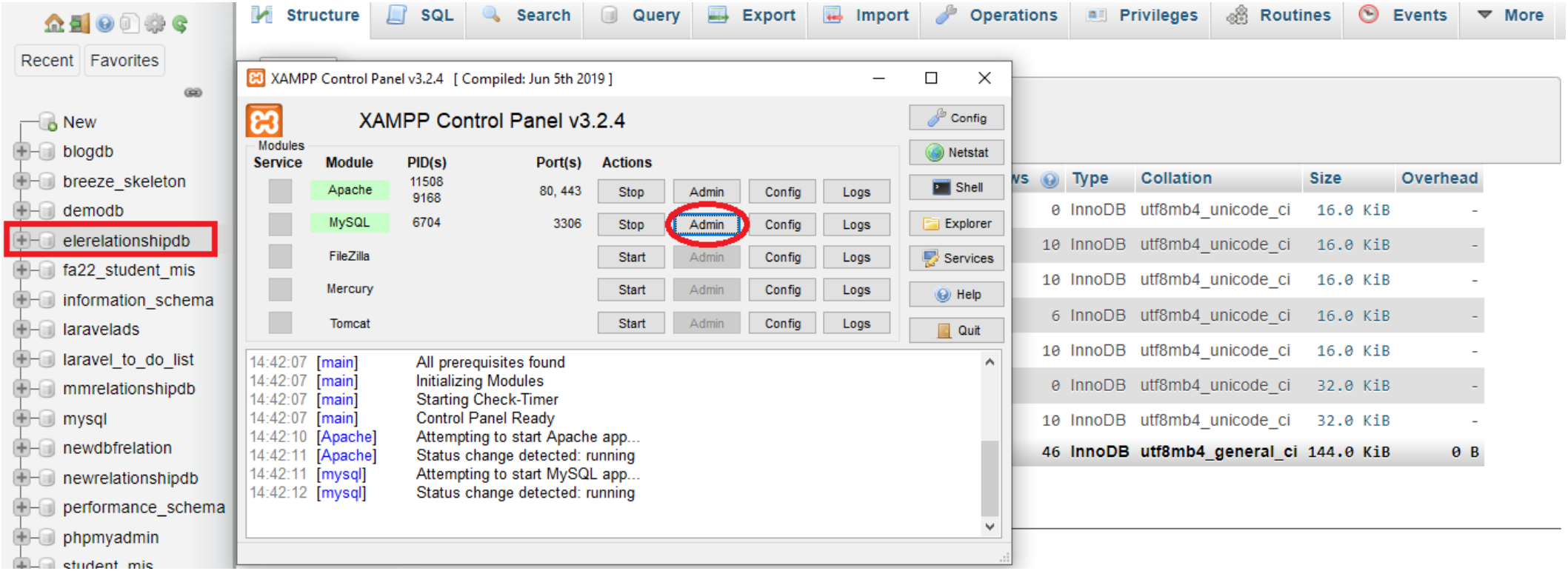
Command Prompt - php artisan serve

```
> @php artisan key:generate --ansi
Application key set successfully.

C:\xampp\htdocs>cd eloquentrelationship

C:\xampp\htdocs\eloquentrelationship>php artisan serve
Laravel development server started: http://127.0.0.1:8000
```

The service is running and laravel project can be browsed on http://127.0.0.1:8000

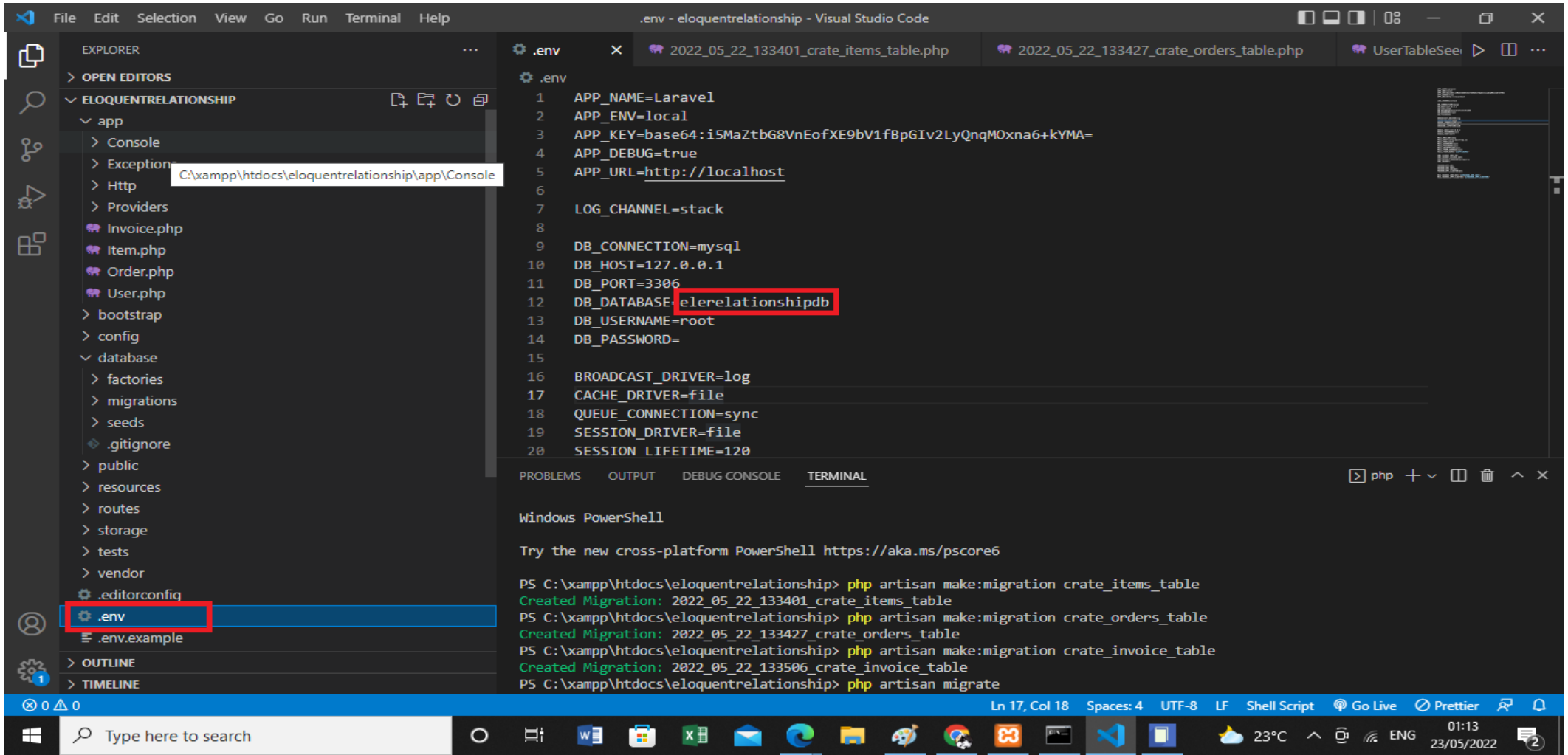
Example: Relationship based Laravel Project
Step 1: Configure Laravel Project

Set up the MySQL database.



Example: Relationship based Laravel Project
Step 1: Configure Laravel Project

Now, edit the .env file.

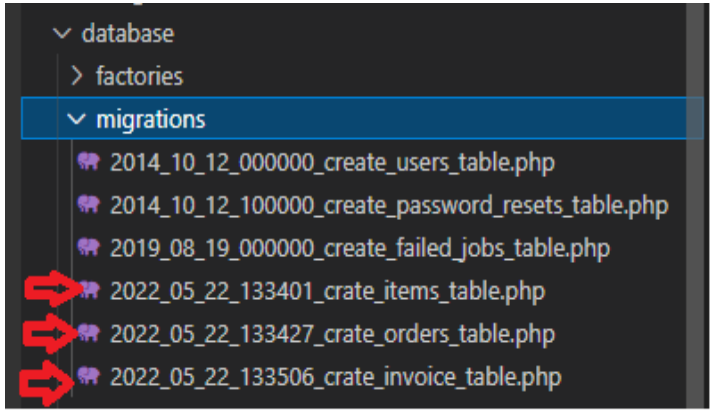


Example: Relationship based Laravel Project
Step 1: Configure Laravel Project

Now, we need to make Three tables to build the relationships between them.

1.Items table
2.Orders table
3.Invoice table

```
php artisan make:migration create_items_table
php artisan make:migration create_orders_table
php artisan make:migration create_invoice_table
```



1. Items Table 2. Orders Table 3. Invoice Table

```
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_items_table
Created Migration: 2022_05_22_133401_crate_items_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_orders_table
Created Migration: 2022_05_22_133427_crate_orders_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_invoice_table
Created Migration: 2022_05_22_133506_crate_invoice_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan migrate
Migration table created successfully.
```

Example: Relationship based Laravel Project
Step 1: Configure Laravel Project

Define the **Schema** of these tables.

database > migrations > 2022_05_22_133401_crate_items_table.php

```php
    */
    public function up()
    {
        Schema::create('items', fu
                $table->increments
                $table->string('it
                $table->integer('p
                $table->timestamps
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('item
```

.env | 2022_05_22_133427_crate_orders_table.php | UserTableSee

database > migrations > 2022_05_22_133427_crate_orders_table.php

```php
function(Blueprint $table){
ts('id');
'user_id');
'item_id');
ps();

ders');
```

.env | 2022_05_22_133506_crate_invoice_table.php

database > migrations > 2022_05_22_133506_crate_invoice_table.php

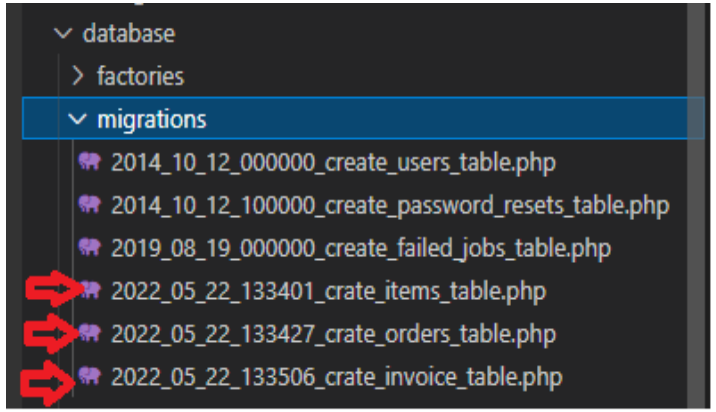
```php
    public function up()
    {
        Schema::create('invoice',function(Blueprint $table){
            $table->increments('id');
            $table->integer('user_id');
            $table->integer('order_id');
            $table->integer('paid_amount');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('invoice');
```

Example: Relationship based Laravel Project
Step 1: Configure Laravel Project

Next, type the following command

```
php artisan migrate
```

It will create all five tables in the database.



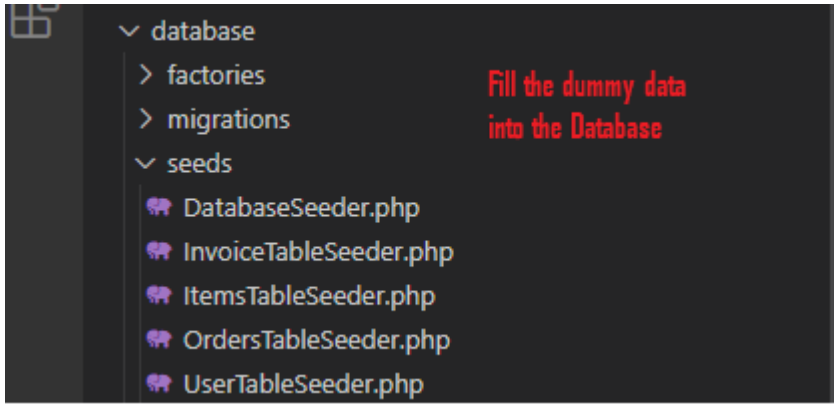
1. Items Table 2. Orders Table 3. Invoice Table

```
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_items_table
Created Migration: 2022_05_22_133401_crate_items_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_orders_table
Created Migration: 2022_05_22_133427_crate_orders_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:migration crate_invoice_table
Created Migration: 2022_05_22_133506_crate_invoice_table
PS C:\xampp\htdocs\eloquentrelationship> php artisan migrate
Migration table created successfully.
```

Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Type the following command to generate the seed files.

```
php artisan make:seeder UsersTableSeeder
php artisan make:seeder ItemsTableSeeder
php artisan make:seeder OrdersTableSeeder
php artisan make:seeder InvoiceTableSeeder
```



Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Feed these tables with the values.

```php
<?php
use Illuminate\Database\Seeder;
class UsersTableSeeder extends Seeder {
/** * Run the database seeds.
* @return void */
public function run()
{
        DB::table('users')->insert([
                'name' => str_random(10),
                'email' => str_random(10).'@gmail.com',
                'password' => bcrypt('secret'),
        ]);
        DB::table('users')->insert([
                'name' => str_random(10),
                'email' => str_random(10).'@gmail.com',
                'password' => bcrypt('secret'),
        ]);
} }
```

Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Feed these tables with the values.

```php
<?php
use Illuminate\Database\Seeder;
class ItemsTableSeeder extends Seeder { /** * Run the database seeds.
* @return void */
 public function run() {
        DB::table('items')->insert([
                'item_name' => 'mobile',
                'price' => 1000
        ]);
        DB::table('items')->insert([
                'item_name' => 'laptop',
                'price' => 2000
        ]);
 }
}
```

Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Feed these tables with the values.

```php
<?php
use Illuminate\Database\Seeder;
class InvoiceTableSeeder extends Seeder {
/** * Run the database seeds. *
* @return void */
 public function run() {
        DB::table('invoice')->insert([
                'user_id' => 1,
                'order_id' => 1,
                'paid_amount' => 1000
        ]);
        DB::table('invoice')->insert([
                'user_id' => 3,
                'order_id' => 2,
                'paid_amount' => 4000
        ]);
 }
}
```

Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Feed these tables with the values.

```php
<?php
use Illuminate\Database\Seeder;
class OrdersTableSeeder extends Seeder
{
/** * Run the database seeds. *
* @return void */
 public function run() {
       DB::table('orders')->insert([
              'user_id' => 1,
              'item_id' => 1
       ]);
       DB::table('orders')->insert([
              'user_id' => 1,
              'item_id' => 5
       ]);
 }
}
```

Example: Relationship based Laravel Project Step 2: Fill the dummy data into the database.

Now, finally, call all these classes in the **DatabaseSeeder.php** file.

```php
<?php
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
/** * Run the database seeds. *
* @return void */
public function run() {
      $this->call(UsersTableSeeder::class);
      $this->call(OrdersTableSeeder::class);
      $this->call(ItemsTableSeeder::class);
      $this->call(InvoiceTableSeeder::class);
}
}
```

Now, run the command below on Terminal, It populates the data in the database tables

```
php artisan db:seed
```

Example: Relationship based Laravel Project
Step 3: Make models for all these three new tables.

Type the following command.

```
php artisan make:model Item
php artisan make:model Order
php artisan make:model Invoice
```

```
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:model Item
Model created successfully.
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:model Order
Model created successfully.
PS C:\xampp\htdocs\eloquentrelationship> php artisan make:model Invoice
Model created successfully.
```

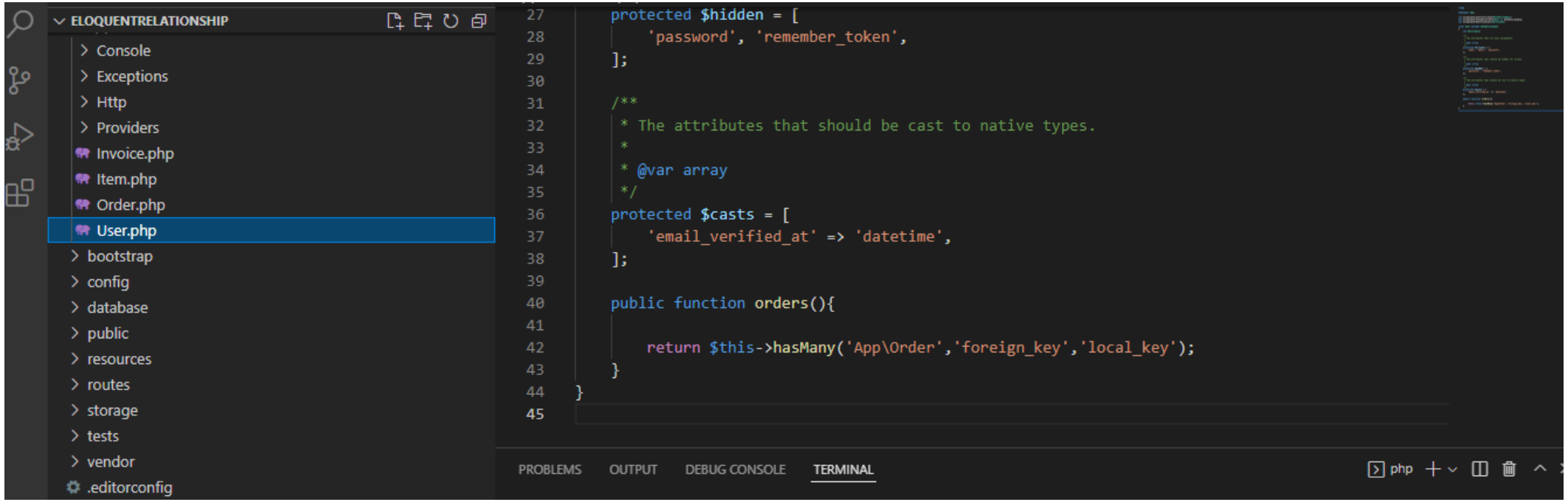
Make Models for all these three new tables

Example: Relationship based Laravel Project
Step 3: One To Many Relationships

A "**one-to-many**" relationship is generally used to define relationships where a single or one model owns any other model.

In our example, the **User** can have multiple **Orders**.

So, in the **User** model, we can write the following functions.




```php
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function orders(){

        return $this->hasMany('App\Order','foreign_key','local_key');
    }
}
```

Example: Relationship based Laravel Project Step 3: One To Many Relationships

Now, go to the terminal and type the following command.

```
php artisan tinker
```

Type the following code in it.

```
$orders = App\User::find(1)->orders;
```

So, it will display the orders whose user_id is 1

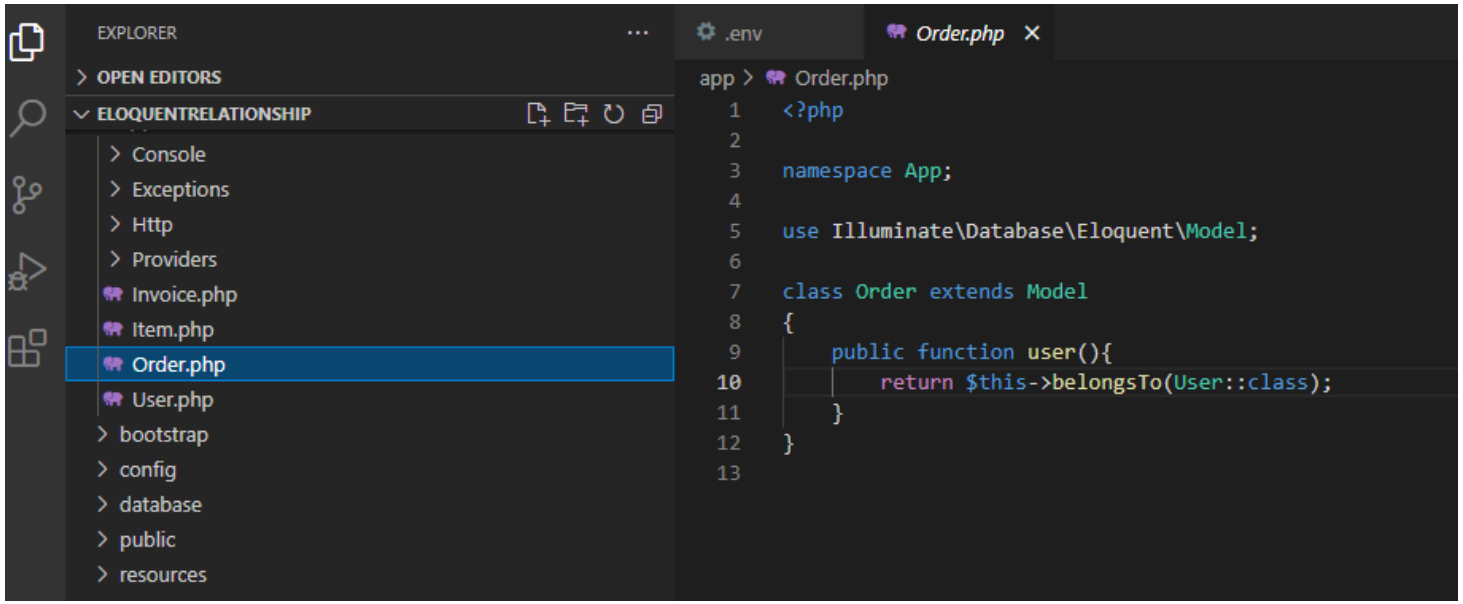
If your table's local Primary Key is different, **id** and Foreign Key are different from **user_id.** Then, you need to specify further arguments like this.

```
<?php
    public function orders() {

        return $this->hasMany('App\Order', 'foreign_key', 'local_key');

}
```

Example: Relationship based Laravel Project
Step 3: One To Many Relationships (Inverse)

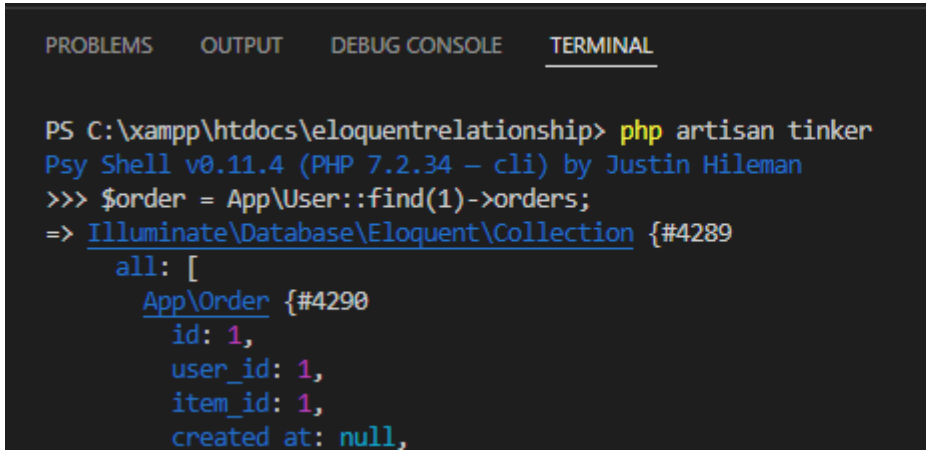
Now that we can access the user that has placed the order.

So, in the **Order.php** model, put the following function in it.



Go to the tinker and type the following code in it.

```
$user = App\Order::find(1)->user;
```



This will find the user, who place the order.

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.

Many to Many Relationship Example

- Let see how to use many to many relationship in Laravel with an example.
- M*M relation is slightly more complicated than **hasOne** and **hasMany** relationships.
- The key in many to many relationship is the join (or pivot) table, which allows the relationship id from one model to be related to many other models and vice-versa. **Many-to-many** relationships are defined by writing a method that returns the result of the **belongsToMany.**
- Let's define the two models.
  - Category and
  - Product
- There exist M*M relation as, **Multiple Categories** have **Multiple Products**, and an inverse relationship will be **Multiple Products** belongs to **Multiple Categories**. Let's do it this example step by step.

Step 1: Install Laravel.

- Use the following command to create new project as follows:

  composer create-project laravel/laravel mmrelationship --prefer-dist

- Now open the project directory
  - Cd mmrelationship
- Open the project in Visual code editor.
- First thing, set up the database. The database name would be **mmrelationshipdb**
- Set the database name in .env file

Step 2: Create a model and migration.

- We are defining two models for this example.
  - **Category**
  - **Product**
- php artisan make:model Category -m
- php artisan make:model Product –m

- It will create products and categories, tables, and models.

Step 2: Create a model and migration (conti.)

- Now, inside **create_categories_table,** define the following schema.

```
// create_categories_table
/** * Run the migrations. *
* @return void */

public function up()
{
        Schema::create('categories', function (Blueprint $table)

        {
                $table->increments('id');
                $table->string('title');
                $table->timestamps();
        });

}
```

Step 2: Create a model and migration (conti.)

- Also, write the following schema inside the **create_products_table.**

```
// create_products_table
/**
* Run the migrations.
*
*
* @return void
*/
public function up()
{
Schema::create('products', function (Blueprint $table)
      {
            $table->increments('id');
            $table->string('name');
            $table->float('price');
            $table->timestamps();
      });
}
```

Now, go to the terminal and create the tables using the migrate command as follows.

php artisan migrate

Step 3: Define random categories manually.

- we make three categories manually inside the database table.

id	title
1	Electronics
2	Mobile
3	Video Games
4	Playstation

Step 4: Define a Pivot table.

- Many-to-many relations require an intermediary table to manage the relationship.
- The most straightforward implementation of the intermediary table, known as a ***pivot table***, would consist of just two columns for storing the foreign keys pointing to each related pair of records.

How to create a Pivot table in Laravel

- **The name of the pivot table** should consist of **singular** names of both tables, separated by underscore symbols, and these names should be arranged in **alphabetical** order, so we have to have **category_product**, not **product_category**.
- **To create a pivot table,** we can create the simple migration with **artisan make:migration** or use the community package **Laravel 5 Generators Extended**. For example, we have the command

  artisan make:migration:pivot.
- **Pivot table fields:** by default, there should be only two fields – the foreign key to each table, in our case **category_id** and **product_id**. You can insert more fields if you need, then you need to add them to the relationship assignment.

Pivot Table

- The **Pivot table** is the relationship between two tables.
- So the **Pivot table** has these columns.
  - id
  - category_id
  - product_id
- Now, we are creating **Many to Many relationships; that** is why many products have categories. For example, to create a migration file, type the following command.

  php artisan make:migration create_category_product_table --create=category_product

Step 5: Define Many To Many relationships.

- Now, Multiple Categories belong to Multiple Products. So inside the **Product.php** file, we can define the **belongsToMany** relationship.

```
// Product.php
<?php namespace App;
use Illuminate\Database\Eloquent\Model; class Product extends Model
{
public function categories() {
return $this->belongsToMany(Category::class); } }
```

Also, the same for the products. Multiple Products belong To Multiple Categories. So inside the **Category.php** file, we can define the **belongsToMany** relationship.

```
// Category.php
<?php namespace App;
use Illuminate\Database\Eloquent\Model; use App\Product;
class Category extends Model {
public function products() {
return $this->belongsToMany(Product::class);
}
}
```

Step 6: Create a Product.

- In a real-time scenario, we create a form, and then through a **POST** request, we insert the **Product** data into the table.
- However, in this example, we will not define any form; we directly store the data into the database because our goal is to use many to many relationships to the complex scenario.
- Now, define a route that saves the Product into the database and assigns the Product to the category using many to many relationships.
- Now, we have four(4) categories. So, we create a product and assign the two categories to one Product.

Step 6: Create a Product.

First, create a ProductController using the following command.

```
php artisan make:controller ProductController
```

The next step, define the route to store the Product.
Now, I am using a GET request for saving the data because we have not created the form, so we take every data manually.

```
// ProductController.php
Route::get('product/create',
'ProductController@create')->name('product.create');
```

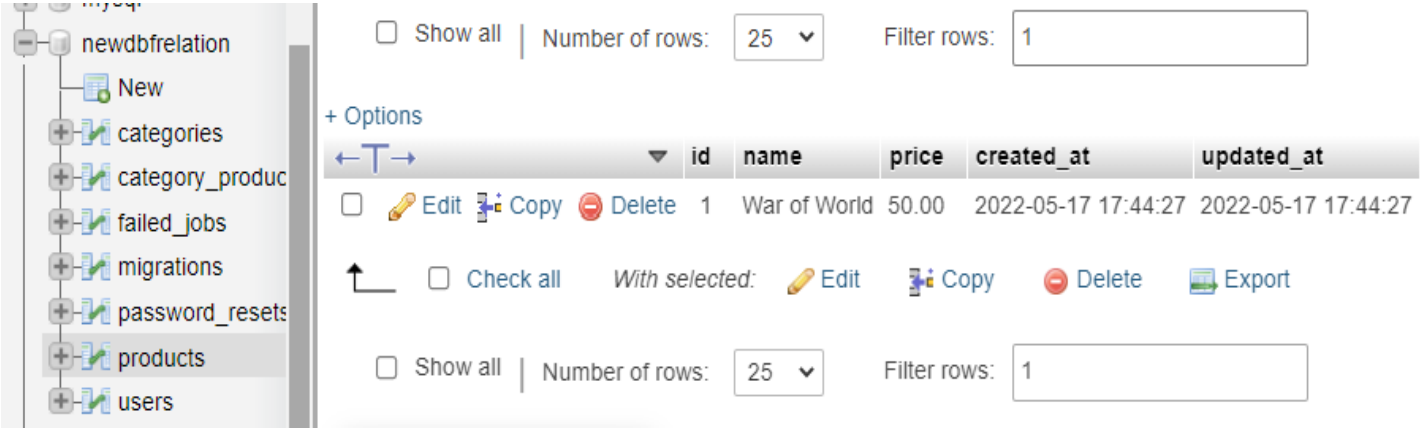
Now, write the following code inside ProductController's create() function.

Step 6: Create a Product (cont.)

```php
// ProductController.php

<?php
namespace App\Http\Controllers;
use App\Category;
use App\Product;
use Illuminate\Http\Request;

class ProductController extends Controller {
    public function create(Request $request)
    {
        $product = new Product;
        $product->name = 'War of World';
        $product->price = 40;
        $product->save();
        $category = Category::find([3, 4]);
        $product->categories()->attach($category);
        return 'Success';
    }
}
```

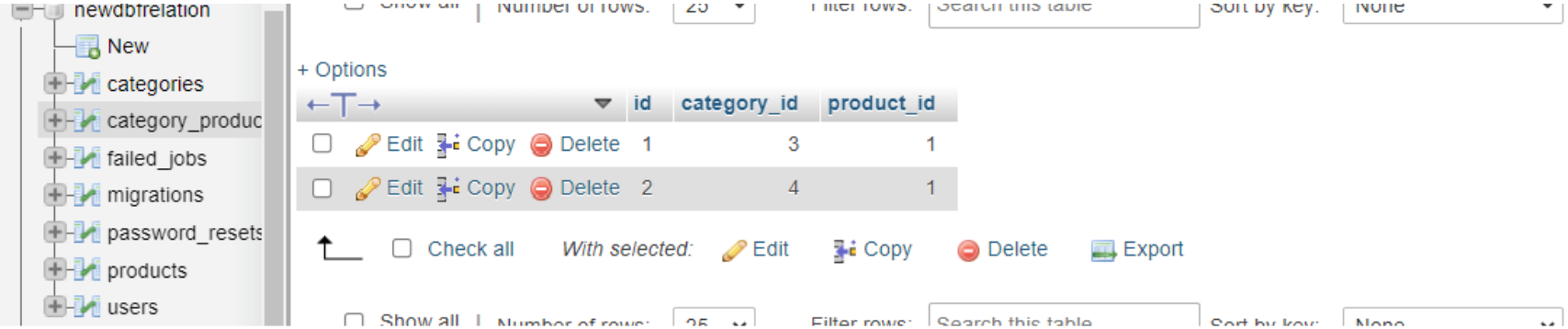
Step 6: Create a Product (cont.)

Now, go to the database and see the **products** table.



Step 6: Create a Product (cont.)

Also, you can check the Pivot Table, which is **a create_product** table. If we have done it all correctly, we can see the two rows inside the table, where product_id is the same 1 for both the rows, but category id's are different, which is **3** and **4.**



In the end we successfully attached the two categories to one Product.

Step 7: Display Product Information.

Define the route that can display all the information.

```
// web.php

Route::get('product/{product}', 'ProductController@show')
     ->name('product.show');
```

Now, define the ProductController's show function. In this function, I am using Routing Model Binding.

```
// ProductController.php public function show(Product
$product) { return view('product.show',
compact('product')); }
```

Create Views

Create a new folder inside the **views** folder called **products** and inside that, create one file called **show.blade.php.**
Write the following code inside the **show.blade.php** file.

```
// show.blade.php

<h2>Product Name: </h2>
<p>{{ $product->name }} || ${{ money_format($product->price, 2) }}</p>

<h3>Product Belongs to</h3>
<ul> @foreach($product->categories as $category)
              <li>{{ $category->title }}</li>

       @endforeach

</ul>
```

Detach() function.

can also delete the relationship between the tables using the detach() function.

```
// ProductController.php
public function removeCategory(Product $product)
{
        $category = Category::find(3); $product->categories()
                ->detach($category);
return 'Success';
}
```

Now, define the following route inside **a web.php** file.

```
// web.php
Route::get('category/product/{product}', 'ProductController@removeCategory')
                ->name('category.product.delete');
```

Laravel Authentication: Registration, Login, Password Reset

- Implementation of Registration, Login, Forgot Password features in Laravel

- Note:

  Php artisan make:auth not working

  Php artisan make:auth in not defined

- We are using Laravel 6 or greater, therefore the process is different than previous version, let see how we proceed

Step 1: New project about Authentication

- Using composer to create new project as follows:

Composer create-project laravel/laravel –prefer-dist laravelauthproject

To creat Authentication in Laravel using built-in controller, blade view and other relevant files.

After creating project, Next to go phpMyAdmin interface and create a database named “laravelauth”

```
C:\xampp\htdocs>composer create-project laravel/laravel --prefer-dist laravelauthproject
Creating a "laravel/laravel" project at "./laravelauthproject"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v7.30.1)
  - Installing laravel/laravel (v7.30.1): Extracting archive
Created project in C:
> @php -r "file_exist
Loading composer repo
Updating dependencies
Lock file operations:
  - Locking asm89/sta
  - Locking brick/mat
  - Locking doctrine/
  - Locking doctrine/
  - Locking doctrine/
  - Locking dragonmar
  - Locking egulias/e
  - Locking facade/fl
  - Locking facade/ig
  - Locking facade/ig
  - Locking fakerphp/
  - Locking fideloper
  - Locking filp/whoo
  - Locking fruitcake
```

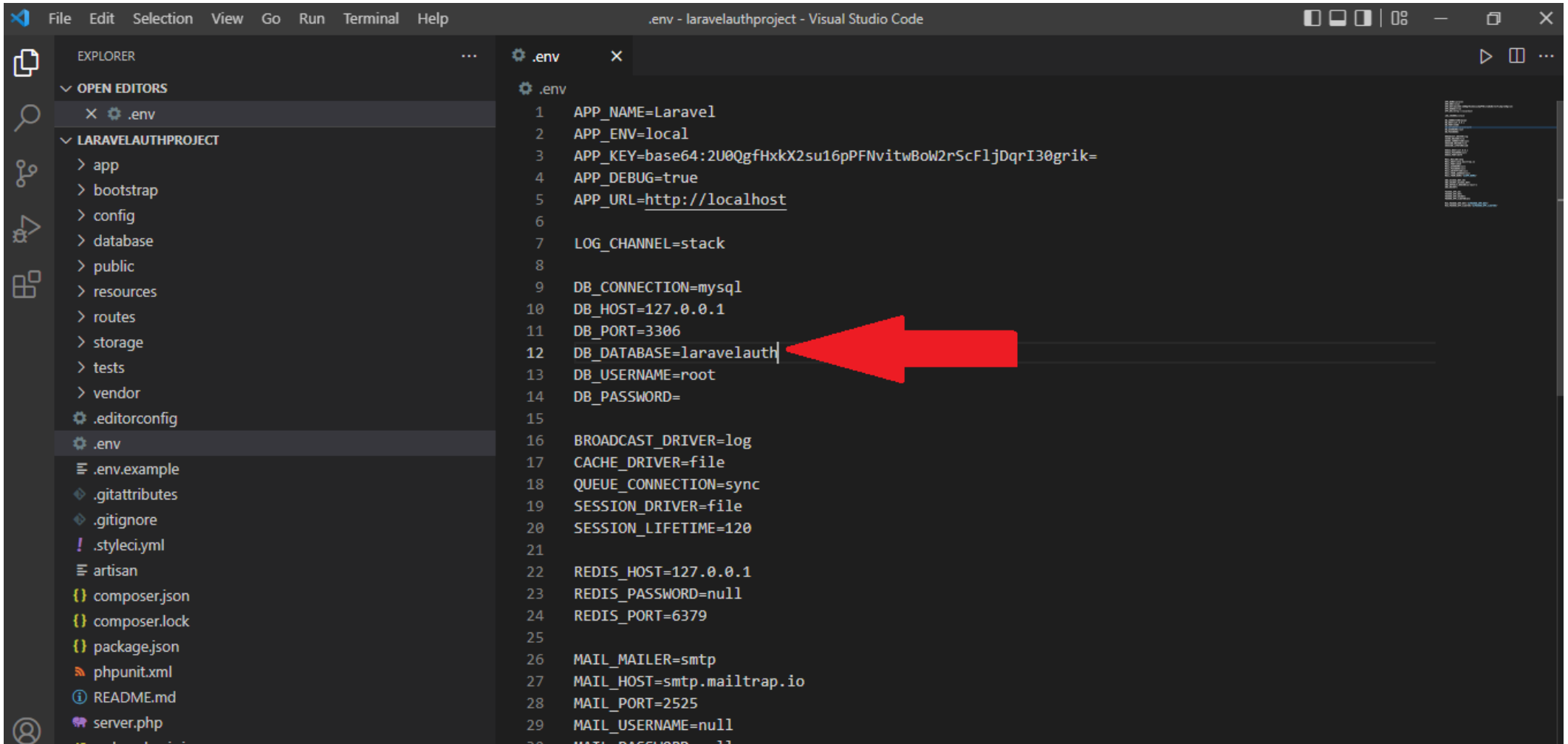
Administrator: Command Prompt

```
  - Installing phar-io/version (3.2.1): Extracting archive
  - Installing phar-io/manifest (2.0.3): Extracting archive
  - Installing myclabs/deep-copy (1.11.0): Extracting archive
  - Installing phpunit/phpunit (8.5.26): Extracting archive
77 package suggestions were added by new dependencies, use `composer suggest` to see details.
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Package phpunit/php-token-stream is abandoned, you should avoid using it. No replacement was suggested.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
69 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan key:generate --ansi
Application key set successfully.

C:\xampp\htdocs>_
```

Type here to search 32°C ENG 16:28 29/05/2022

Step 1: New project about Authentication

- Open project folder in Visual studio code and write db name in.env file



Step2: Implement Authentication

- Composer require laravel/ui: This will integrate necessary Laravel UI files.

```
PS C:\xampp\htdocs\laravelauthproject> composer require laravel/ui
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^2.5 for laravel/ui
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
  - Locking laravel/ui (v2.5.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing laravel/ui (v2.5.0): Extracting archive
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Package phpunit/php-token-stream is abandoned, you should avoid using it. No replacement was suggested.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
69 packages you are using are looking for funding.
Ln 12, Col 24   Spaces: 4   UTF-8   LF   Shell Script   Go Live   Prettier
```

Step2: Implement Authentication

Php artisan ui react/bootstrap/vue –auth

This command will use react/bootstrap/vue for front end designs. (Must use one command at a time)

```
PS C:\xampp\htdocs\laravelauthproject> php artisan ui react --auth
React scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.
PS C:\xampp\htdocs\laravelauthproject>
```

Step2: Implement Authentication

Php artisan migrate

```
PS C:\xampp\htdocs\laravelauthproject> php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (0.09 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (0.13 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (0.08 seconds)
PS C:\xampp\htdocs\laravelauthproject>
```

Also, run two more commands to integrate React UI.

Npm install

```
PS C:\xampp\htdocs\laravelauthproject> npm install
npm WARN deprecated axios@0.19.2: Critical security vulnerability fixed in v0.21.1. For more information, see https://github.co
m/axios/axios/pull/3410
npm WARN deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy
 v1
npm WARN deprecated chokidar@2.1.8: Chokidar 2 does not receive security updates since 2019. Upgrade to chokidar 3 with 15x few
er dependencies
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fseven
ts 2.
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain circum
stances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API in
stead.
npm WARN deprecated source-map-resolve@0.5.3: See https://github.com/lydell/source-map-resolve#deprecated
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
[[redacted].....] \ finalize:kind-of: sill finalize C:\xampp\htdocs\laravelauthproject\node_modules\object-copy\node_module
```

Step2: Implement Authentication

Also, run two more commands to integrate React UI.

Npm run dev

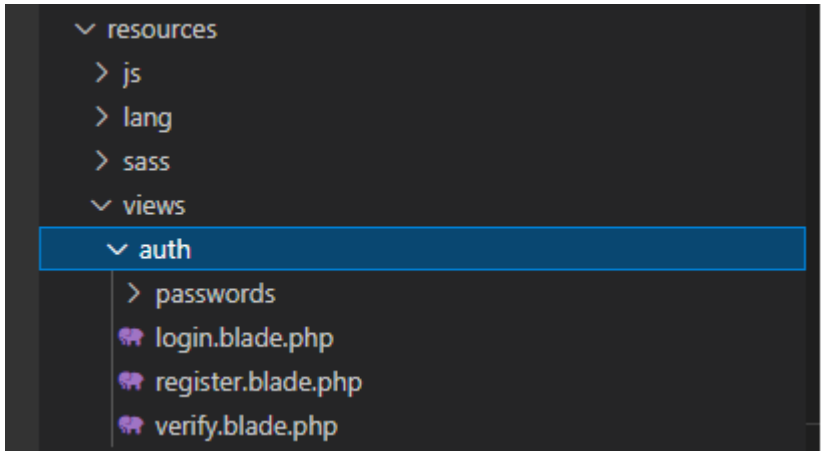
```
PS C:\xampp\htdocs\laravelauthproject> npm run dev

> @ dev C:\xampp\htdocs\laravelauthproject
> npm run development


> @ development C:\xampp\htdocs\laravelauthproject
> cross-env NODE_ENV=development node_modules/webpack/bin/webpack.js --progress --config=node_modules/laravel-mix/setup/webpack
.config.js

98% after emitting SizeLimitsPlugin

 DONE  Compiled successfully in 14331ms                                                                    16:43:02

       Asset     Size  Chunks             Chunk Names
/css/app.css  180 KiB  /js/app  [emitted]  /js/app
  /js/app.js    2 MiB  /js/app  [emitted]  /js/app
Notifications are disabled
Reason: DisabledForUser Please make sure that the app id is set correctly.
Command Line: C:\xampp\htdocs\laravelauthproject\node_modules\node-notifier\vendor\snoreToast\snoretoast-x64.exe -pipeName \\.\
pipe\notifierPipe-a7904f47-0de7-4ea3-9bb2-1363c792b370 -p C:\xampp\htdocs\laravelauthproject\node_modules\laravel-mix\icons\lar
avel.png -m "Build successful" -t "Laravel Mix"
PS C:\xampp\htdocs\laravelauthproject>
```

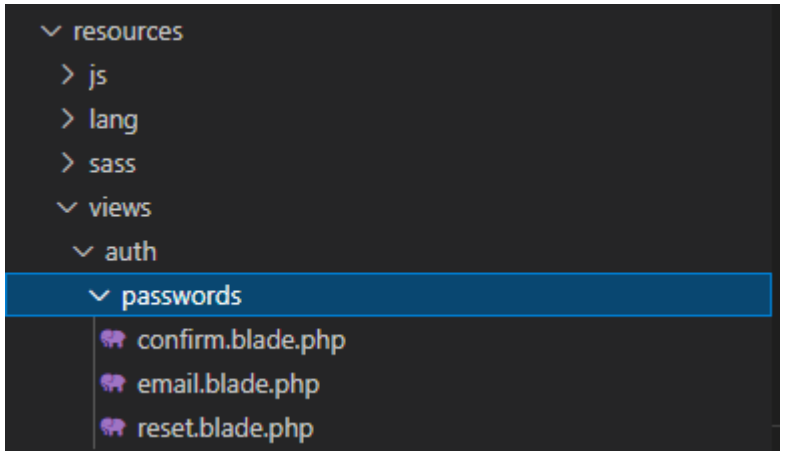
If npm command create some problems then better is to use CMD as an Administrator account privilege.

Step2: implementation Authentication

- When one fired authentication command, the system will generate some files related to authentication and layout as follows:
- Navigate to the **resources->views->auth** folder.
- Inside auth folder, there should be three blade files: login.blade.php, register.blade.php and verify.blade.php
- Also, inside auth folder there is another folder named “passwords”. It contains three blade files:

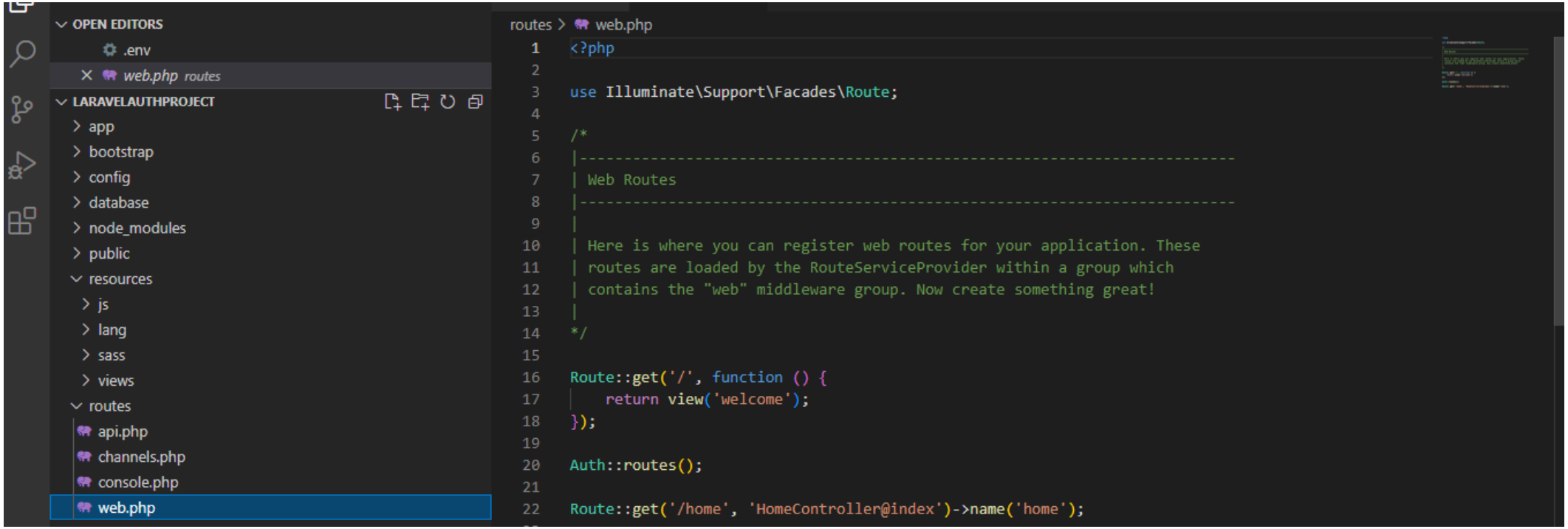
  Confirm.blade.php, email.blade.php and reset.blade.php





Step 3: Routes and their Protection

Now open the routes->web.php file. It contains all the routes defined and protection handle through middleware.



Step 4: Run the laravel project using as:

Administrator: Command Prompt - php artisan serve

```
Application key set successfully.

C:\xampp\htdocs>cd laravelauthproject

C:\xampp\htdocs\laravelauthproject>php artisan serve
Laravel development server started: http://127.0.0.1:8000
```

http://localhost:8000

LOGIN REGISTER

Laravel

DOCS LARACASTS NEWS BLOG NOVA FORGE VAPOR GITHUB

Step 4: Testing the authentication