

CSE 3100: Web Programming Laboratory

# Lab 8: Laravel Migration & Database Designing

## Create a Laravel Project

- Open Terminal in your VS Code . Shortcut: *Ctrl+`* or View>Terminal
- Go to the htdocs folder in the terminal :

```
cd C:\xampp\htdocs
```

- Create a Laravel project: *composer create-project laravel/laravel test-app*

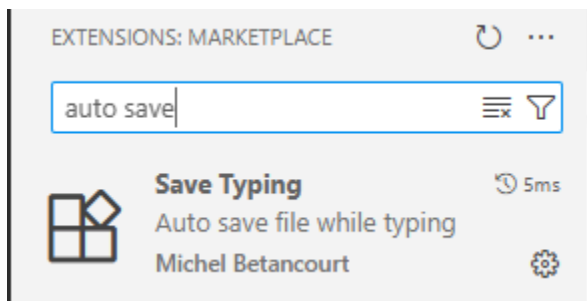
```
PS C:\Users\hp> cd C:\xampp\htdocs
PS C:\xampp\htdocs> composer create-project laravel/laravel test-app
Creating a "laravel/laravel" project at "test-app"
```

- Open the project in VS Code:

```
cd test-app
code .
```

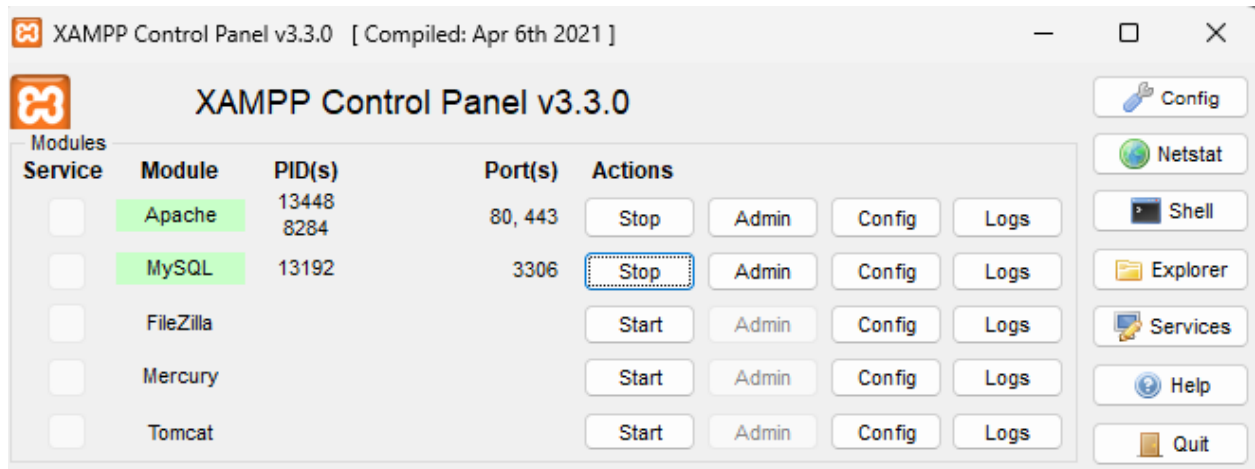
```
PS C:\xampp\htdocs> cd test-app
PS C:\xampp\htdocs\test-app> code .
PS C:\xampp\htdocs\test-app> █
```

- Start Laravel built-in server: *php artisan serve*
- Suggestion: Add any autosave extension in your VS code project



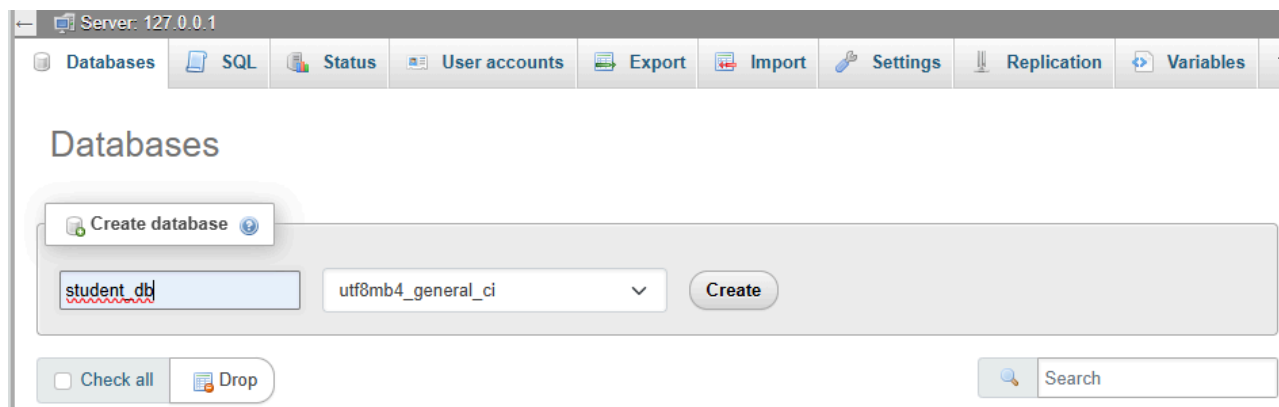
## Apache & MySQL

First open your XAMPP Control Panel and turn MySQL services on along with the Apache server.



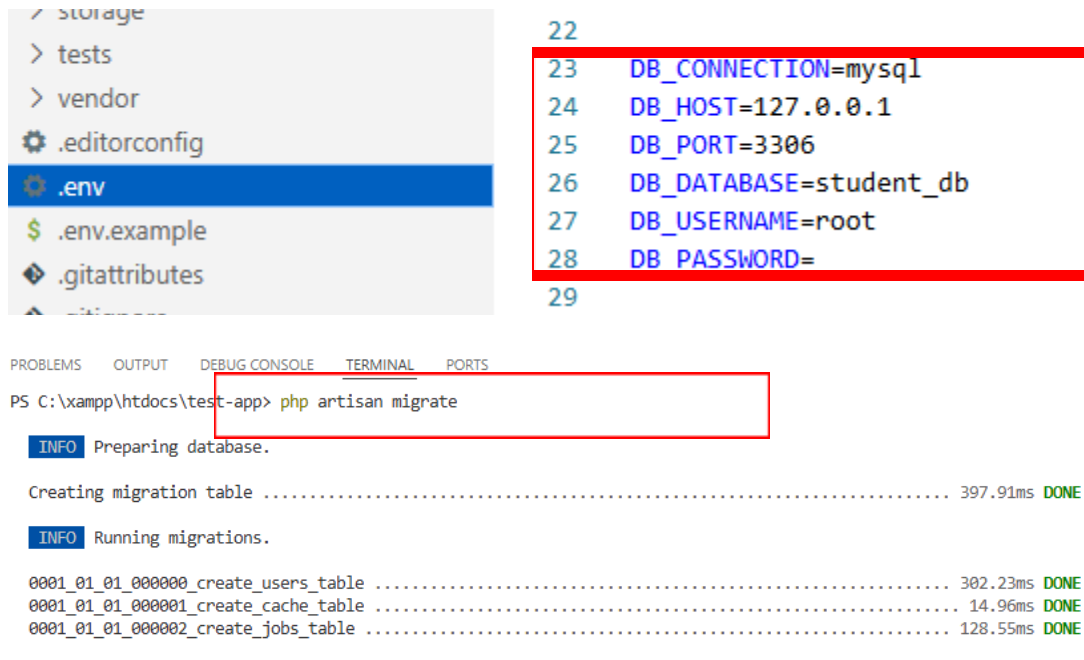
## Create Database

Paste <http://localhost/phpmyadmin> URL in your browser to open phpMyAdmin. Create a database with any name i.e. `student_db`. We do not need to add any table manually. We will add tables through artisan commands. First we need to connect our database to the project.



## Connect Laravel App with Database

Now, copy your database name and go back to your project in VS Code and open the `.env` file. Change the `DB_CONNECTION` to your RDBMS name i.e. `mysql`, `sqlite`, `mongodb`. Then comment out the other variables and change `DB_DATABASE` to your created database's name. Then create a new terminal in VS Code and write the following artisan command: `php artisan migrate`



The image shows a VS Code editor with the `.env` file selected in the file explorer. The `.env` file contains the following database configuration:

```
22
23 DB_CONNECTION=mysql
24 DB_HOST=127.0.0.1
25 DB_PORT=3306
26 DB_DATABASE=student_db
27 DB_USERNAME=root
28 DB_PASSWORD=
29
```

Below the editor, the terminal window shows the command `php artisan migrate` being executed. The output indicates that the database is being prepared and migrations are being run successfully.

```
PS C:\xampp\htdocs\test-app> php artisan migrate

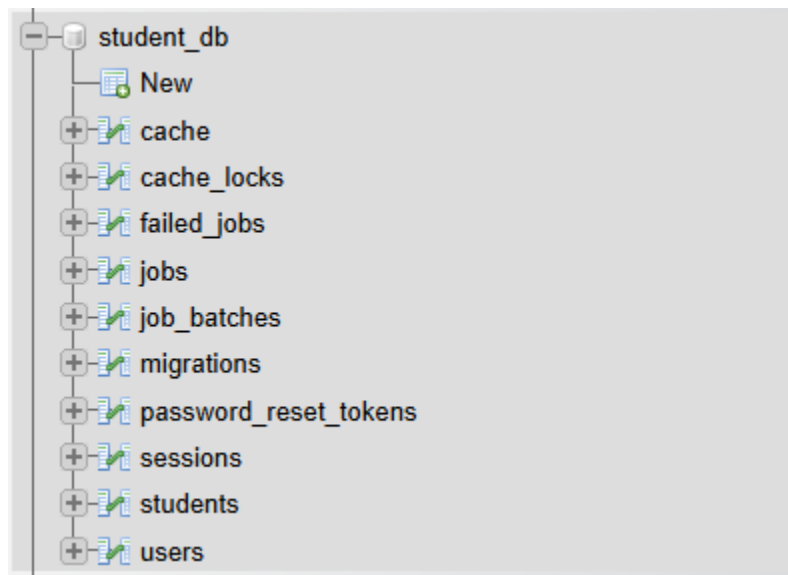
INFO: Preparing database.

Creating migration table ..... 397.91ms DONE

INFO: Running migrations.

0001_01_01_000000_create_users_table ..... 302.23ms DONE
0001_01_01_000001_create_cache_table ..... 14.96ms DONE
0001_01_01_000002_create_jobs_table ..... 128.55ms DONE
```

If you go to the database in phpmyadmin, you will find some newly created tables. These are default tables of Laravel. Now we will create our own table students or today's lab task.



## Create a New Table in Database using Migration File

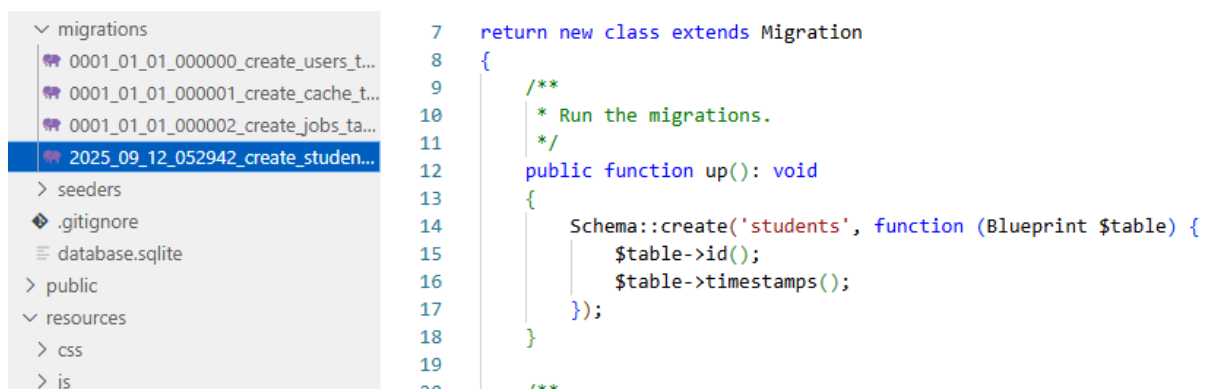
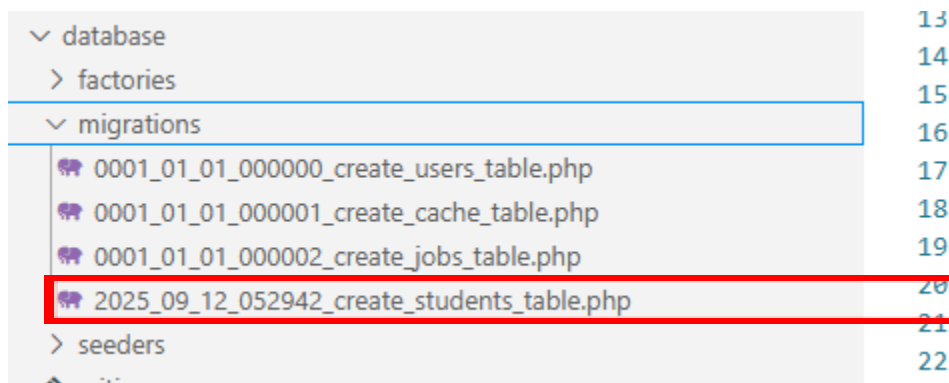
We will create a table using migration [<https://laravel.com/docs/12.x/migrations>]. Write the artisan command: `php artisan make:migration create_students_table`.

```
PS C:\xampp\htdocs\test-app> php artisan make:migration create_students_table
```

```
INFO Migration [C:\xampp\htdocs\test-app\database\Migrations\2025_09_12_052942_create_students_table.php] created successfully.
```

Then a table will be created under the Folder Database>migrations. We will now edit the **create\_students\_table.php** and add our required columns or the table. Check the available column types from [Laravel Documentation](#). After adding all the columns, go to the terminal and write

`php artisan migrate` to create the table.



Now we will add columns to the table.

```

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->integer('age');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
}

```

After adding all the columns, go to the terminal and write *php artisan migrate* to create the table.

```

PS C:\xampp\htdocs\test-app> php artisan migrate

```

**INFO** Running migrations.

2025\_09\_11\_110043\_create\_students\_table .....

The screenshot shows a database management interface. On the left, a tree view displays the database structure, including schemas like 'information\_schema', 'mysql', and 'performance\_schema', and a database named 'student\_db' containing tables like 'cache', 'cache\_locks', 'failed\_jobs', 'jobs', 'job\_batches', 'migrations', 'password\_reset\_tokens', 'sessions', and 'students'. The 'students' table is selected. On the right, the 'Query results' pane shows a successful query: 'SELECT \* FROM `students`'. The result is an empty set. Below the query, the table structure is displayed with columns: 'id', 'name', 'email', 'age', 'created\_at', and 'updated\_at'. The 'Query results operations' section includes a 'Create view' button. The 'Bookmark this SQL query' section has a 'Label' input field and a checkbox 'Let every user access this bookmark'.

## Create Laravel Model

We will use Artisan Command to generate a new model:

*php artisan make:model Student*

We will think of each Eloquent model in your application as a "resource" and create a resource controller for each of the models.

For better understanding visit: <https://laravel.com/docs/12.x/eloquent>

```
PS C:\xampp\htdocs\test-app> php artisan make:model Student

INFO Model [C:\xampp\htdocs\test-app\app\Models\Student.php] created successfully.

PS C:\xampp\htdocs\test-app>
```

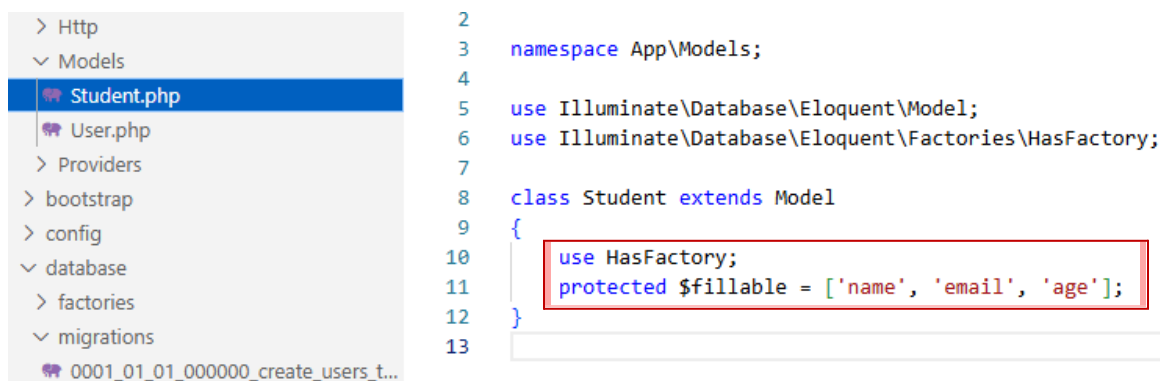
After creating the model, you will find it under `app>models>Student.php` . Open the `Student.php` file.

We will use the *create* method to "save" a new model using a single PHP statement while inserting data in the database through `StudentController` and `Student` model. However, before using the *create* method, you will need to specify either *a fillable or guarded property* on your model class. These properties are required because all Eloquent models are protected against **mass assignment vulnerabilities** by default. Hence, we will add fillable property in our `Student.php` model.

We will first import `HasFactory` in order to use `$fillable` . Then we will specify the column names that we want to fill.

```
use Illuminate\Database\Eloquent\Factories\HasFactory;

protected $fillable = ['name', 'email', 'age'];
```



```
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6 use Illuminate\Database\Eloquent\Factories\HasFactory;
7
8 class Student extends Model
9 {
10     use HasFactory;
11     protected $fillable = ['name', 'email', 'age'];
12 }
13
```

# Create Laravel Controller

We will use Resource Controllers for our Eloquent Model Student.

If you think of each Eloquent model in your application as a "resource", it is typical to perform the same sets of actions against each resource in your application. For example, imagine your application contains a Photo model and a Movie model. It is likely that users can create, read, update, or delete these resources.

Because of this common use case, Laravel resource routing assigns the typical create, read, update, and delete ("CRUD") routes to a controller with a single line of code.

For better understanding visit: <https://laravel.com/docs/12.x/controllers#main-content>

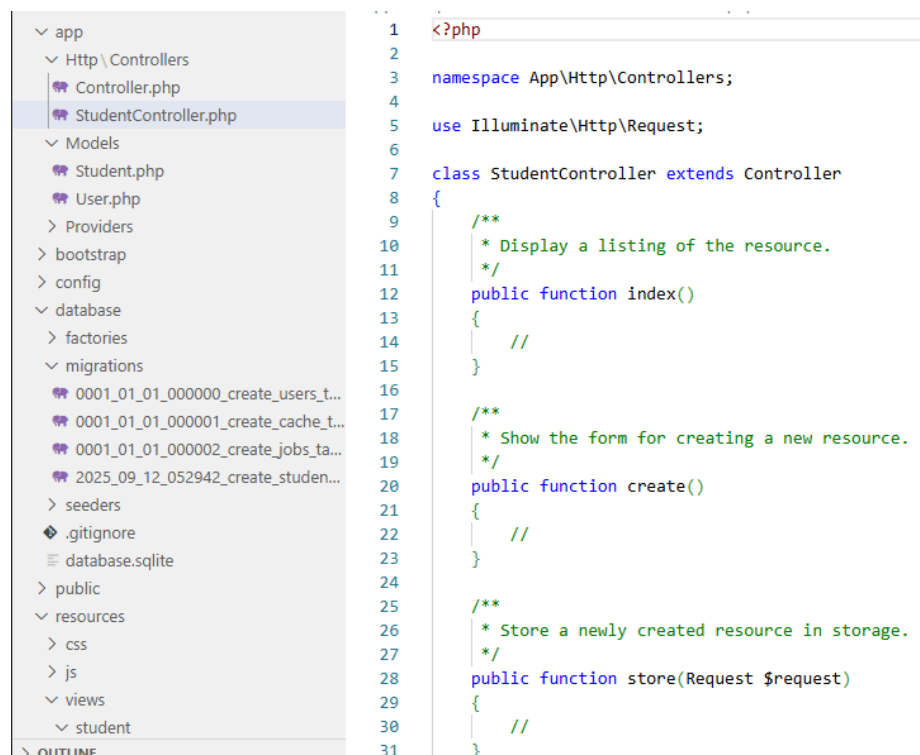
Now, we will write artisan command to create a resource controller

*php artisan make:controller StudentController --resource*

```
PS C:\xampp\htdocs\test-app> php artisan make:controller StudentController --resource

INFO Controller [C:\xampp\htdocs\test-app\app\Http\Controllers\StudentController.php] created successfully.
```

Now, go to your StudentController.php file under app>Http>Controllers. You will see some methods already created by the controller to handle basic CRUD operations.





As we will use the Student.php model for our CRUD operation, we firstly need to add it to our namespace. Add `use App\Models\Student;` after `namespace App\Http\Controllers;`

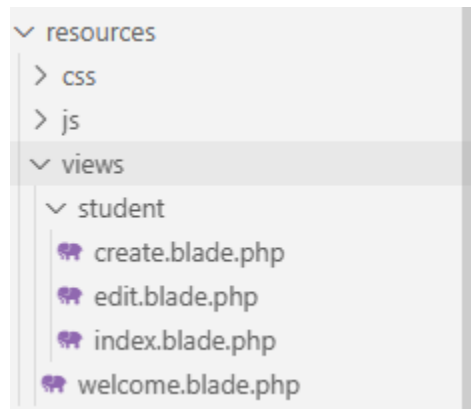
```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Student;

class StudentController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        //
    }
}
```

## View files

Create a folder under views named “student”. Now, generate create.blade.php , edit.blade.php , index.blade.php files in the folder. We can use routes to view these pages. Go to web.php and give the following route to see the create.blade.php and index.blade.php files. Then write *php artisan serve* in VS code terminal to start Laravel built-in server..



## Create Student

Name:

Email:

Age:

127.0.0.1:8000/student/list

## Student List

[Add New Student](#)

ID	Name	Email	Age	Update	Delete
				<a href="#">Edit</a>	<a href="#">Delete</a>

```

routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::get('/', function () {
6      return view('welcome');
7  });
8  Route::get('student/test', function () {
9      return view('student.create');
10 })->name('student.create');
11 Route::get('student/list', function () {
12     return view('student.index');
13 });

```

## Create Route using Controller

Now, we will do routing using the controller class. First we need to add the controller class namespace in our web.php file. Then, we will map a URL to our controller's specific method. Let's start with showing the create.blade.php page through the controller method. Add the following namespace to access StudentController.

To learn more about Routes in Laravel: <https://www.cloudways.com/blog/routing-in-laravel/> , <https://laravel.com/docs/12.x/routing>, <https://laravel.com/docs/12.x/routing#named-routes>

```
use App\Http\Controllers\StudentController;
```

Then write the route:

```
Route::get('student/create', [StudentController::class, 'create'])->name('student.create');
```

Here, first we are mentioning the method which is “get” then the url that we will use and the associated controller class and the function inside the controller class that is used to view student.create page.

After mentioning the route in web.php , we will go to StudentController.php file to define the create () function.

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\StudentController;
5
6  Route::get('/', function () {
7      return view('welcome');
8  });
9  Route::get('student/test', function () {
10     return view('student.create');
11 }->name('student.create');
12 Route::get('student/list', function () {
13     return view('student.index');
14 });
15 Route::get('student/create', [StudentController::class, 'create'])->name('student.create');
```

StudentController.php

- Models
- Student.php
- User.php
- Providers
- bootstrap
- config

```
20
21
22 public function create()
23 {
24     //view form create
25     return view('student.create');
26 }
27
```

You can use *php artisan serve* command to start the project in built-in server and search student/create

```
PS C:\xampp\htdocs\test-app> php artisan serve
```

```
INFO Server running on [http://127.0.0.1:8000].
```

```
Press Ctrl+C to stop the server
```



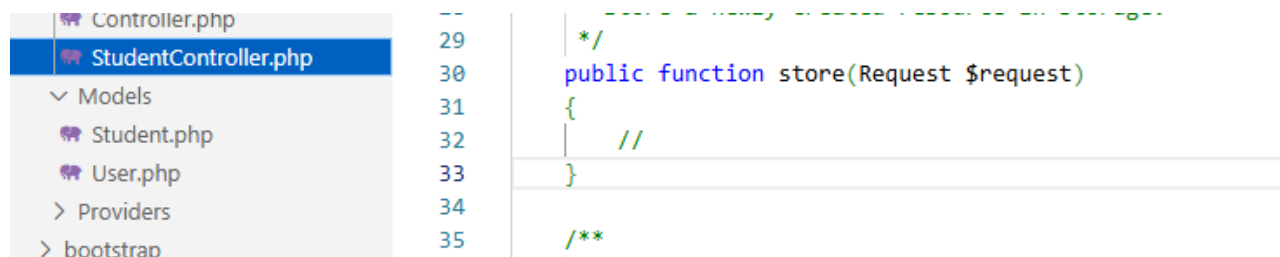
The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/student/create`. The page title is "Create Student". Below the title, there are three input fields labeled "Name:", "Email:", and "Age:". At the bottom of the form, there are two buttons: "Submit" and "Back".

## Store/Insert Data

For better understanding: <https://laravel.com/docs/master/eloquent#inserting-and-updating-models>

To insert a new record into the database, you should instantiate a new model instance and set attributes on the model. We will use the built-in *create* method, we have already specified a fillable property on our Student model class for this purpose.

Now, go to the StudentController.php file, and let's start defining the function *store* for storing data in the database. **public function store** has a Http Request class as a parameter. To learn more about HTTP Request Handling: <https://laravel.com/docs/12.x/requests#retrieving-input>



for security purposes, we will first validate the values submitted in the form using `$request->validate` To learn more about validating form inputs: <https://laravel.com/docs/12.x/validation#quick-writing-the-validation-logic>

```

26
27
28  /**
29   * Store a newly created resource in storage.
30   */
31  public function store(Request $request)
32  {
33      //
34      $data = $request->validate([
35          'name' => 'required|string|max:255',
36          'email' => 'required|email|unique:students,email',
37          'age' => 'required|integer|min:0',
38      ]);
39
40      Student::create($data);
41      return redirect(route('student.create'))->with('success', 'Student created successfully.');
```

After validating we will use a *create* method to "save" a new Student model using a single PHP statement. `Student::create($data);`

You can skip validation and directly use `Student::create($request->all());` to store data.

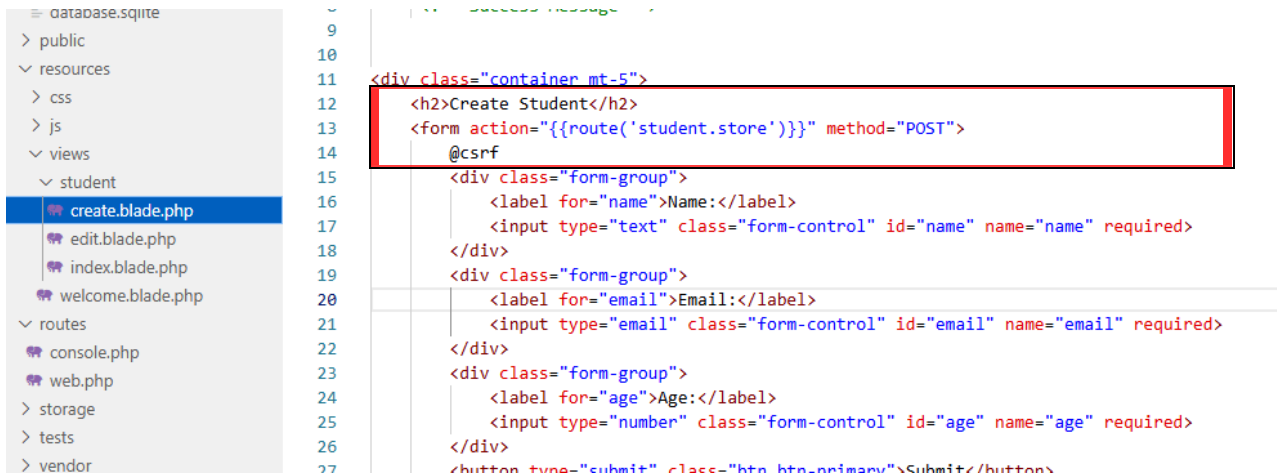
Finally, redirect the route to student.create page after a successful store operation

```
return redirect(route('student.create'));
```

```

26
27
28  /**
29   * Store a newly created resource in storage.
30   */
31  public function store(Request $request)
32  {
33      //
34      $data = $request->validate([
35          'name' => 'required|string|max:255',
36          'email' => 'required|email|unique:students,email',
37          'age' => 'required|integer|min:0',
38      ]);
39
40      Student::create($data);
41      return redirect(route('student.create'))->with('success', 'Student created successfully.');
```

Now, we need to mention this store method route in our view file and web.php file. First go to create.blade.php file where in the form action attribute we will mention the route student.store.



Also for the success message , we need to add a session key to show the message.

```

<div>

    <!-- Success Message -->

    @if(session()->has('success'))

        <div class="alert alert-success">

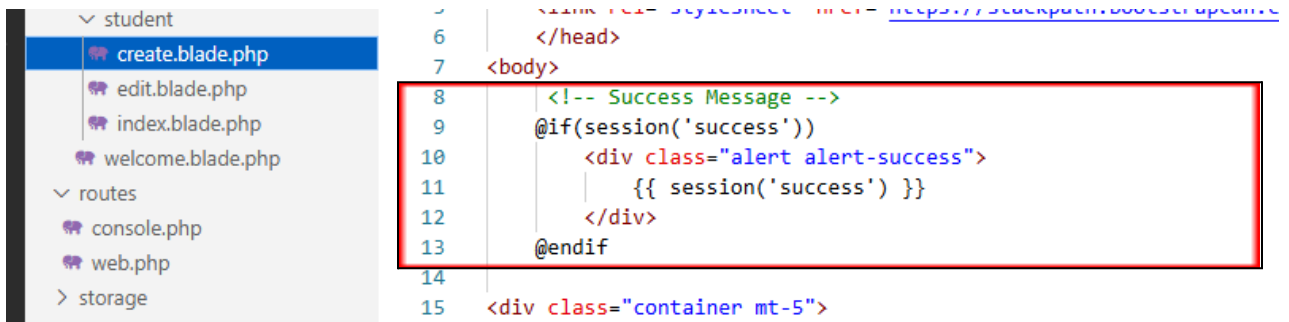
            {{ session('success') }}

        </div>

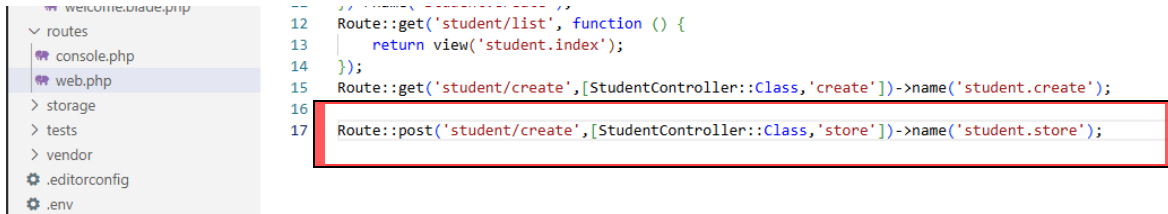
    @endif

</div>

```



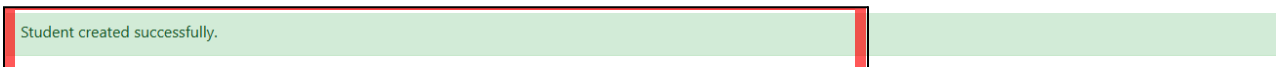
After modifying the create.blade.php file, we will modify the route in the web.php file. As method='POST' is used for submitting forms, the route will as follows:



Here, we have mentioned the ,method = 'post' and StudentController function 'store'

Now, start the server using: php artisan serve and go to /student/create

After a successful store operation check your database in phpmyadmin

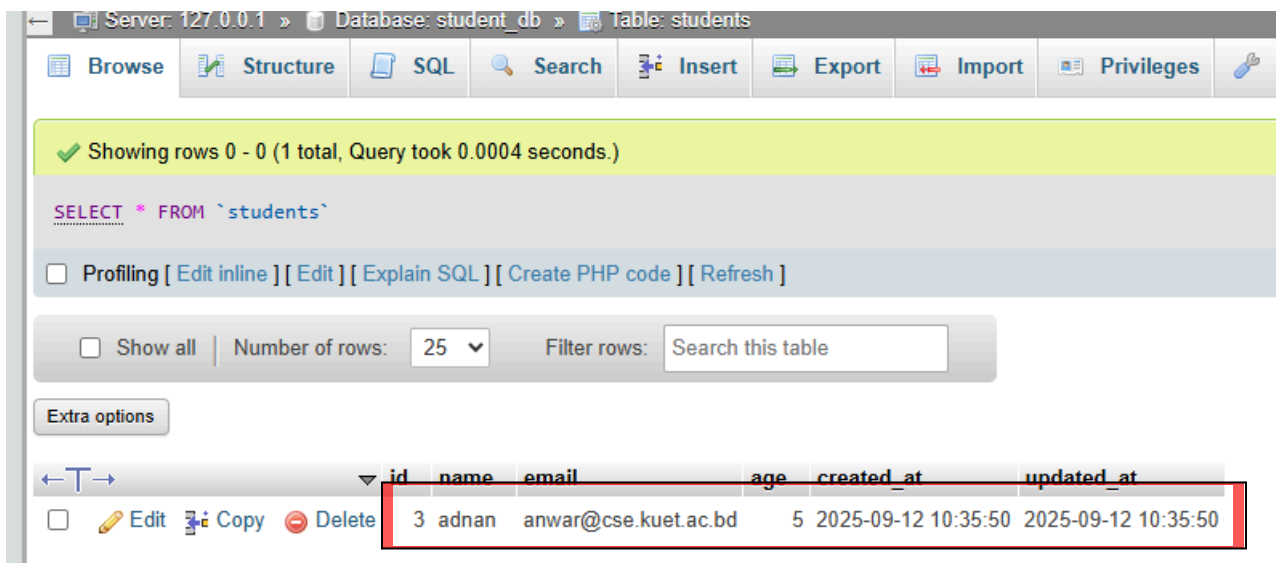


## Create Student

Name:

Email:

Age:



## Read Data

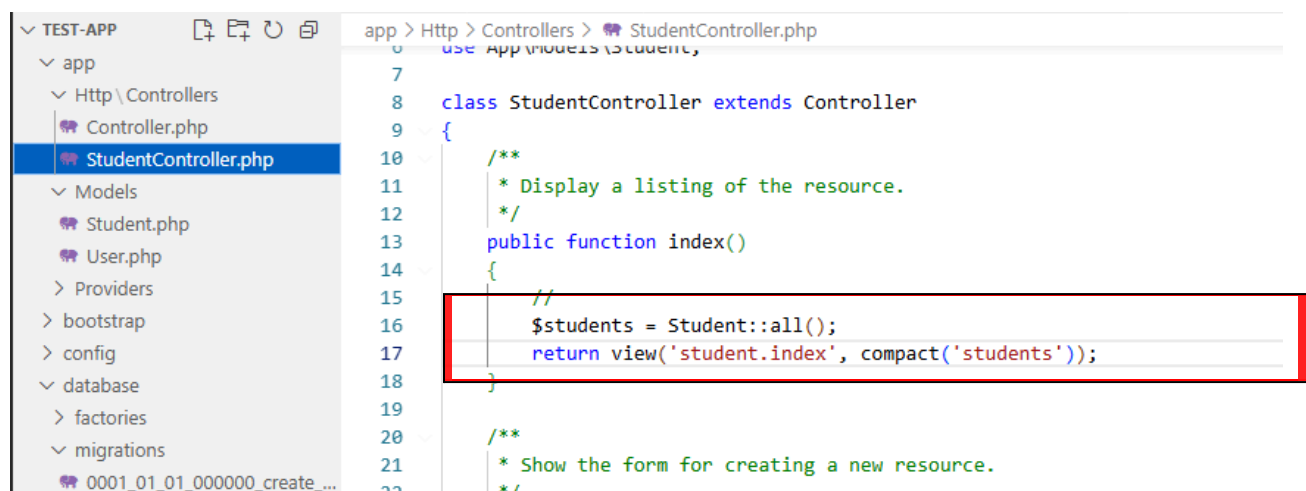
Once you have created a model and its associated database table, you are ready to start retrieving data from your database. You can think of each Eloquent model as a powerful query builder allowing you to fluently query the database table associated with the model.

The model's *all* method will retrieve all of the records from the model's associated database table. The Eloquent *all* method will return **all of the results** in the model's table.

Eloquent methods like *all* and *get* to retrieve multiple records from the database. However, these methods don't return a plain PHP array. Instead, an instance of `Illuminate\Database\Eloquent\Collection` is returned.

For better understanding: <https://laravel.com/docs/master/eloquent#retrieving-models>

Now, we will start retrieving data from our database table students. We will create a function in `StudentController.php` named public function `index()`. Here, we are retrieving the records using *all* methods and redirecting them to the `student.index` page along with the records in a compact array form/associative array.



```
app > Http > Controllers > StudentController.php
1 use App\Models\Student;
2
3
4
5
6
7
8 class StudentController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         //
16         $students = Student::all();
17         return view('student.index', compact('students'));
18     }
19
20     /**
21      * Show the form for creating a new resource.
22      */
23 }
```

Now we need to modify the `index.blade.php` and `web.php` files.

First, let's modify the `index.blade.php` file. We will use `@foreach...@endforeach` loop to iterate through all the values that were read from the database and stored in the student array then print those values in respective `<td></td>` tags.



```

resources > views > student > index.blade.php
7  <body>
8  <div class="container mt-5">
15 <table class="table table-bordered">
26 <tbody>
27 <!-- Looping through student array -->
28
29 <tr>
30 <td></td>
31 <td></td>
32 <td></td>
33 <td></td>
34 <td>
35 <a href="#" >Edit</a>
36 </td>
37 <td>
38
39 <form action="" method="POST" style="display:inline;">
40 @csrf
41 @method('delete')
42 <button type="submit" class="btn btn-danger btn-sm">Delete</button>
43 </form>
44 </td>
45 </tr>
46
47 </tbody>
48 </table>
49
50 </div>

```

```

> css
> js
> views
> student
> create.blade.php
> edit.blade.php
> index.blade.php
> welcome.blade.php
> routes
> console.php
> web.php
> storage
> tests
> vendor
> .editorconfig
> .env
> .env.example
> .gitattributes
> .gitignore
> artisan
> composer.json
> composer.lock
> package.json

15 <table class="table table-bordered">
26 <tbody>
27 <!-- Looping through student array -->
28 @foreach($students as $student)
29 <tr>
30 <td>{{ $student->id }}</td>
31 <td>{{ $student->name }}</td>
32 <td>{{ $student->email }}</td>
33 <td>{{ $student->age }}</td>
34 <td>
35 <a href="#" >Edit</a>
36 </td>
37 <td>
38
39 <form action="" method="POST" style="display:inline;">
40 @csrf
41 @method('delete')
42 <button type="submit" class="btn btn-danger btn-sm">Delete</button>
43 </form>
44 </td>
45 </tr>
46 @endforeach
47 </tbody>
48 </table>
49
50 </div>

```

Now, we will update the routes in web.php files and add the following route.

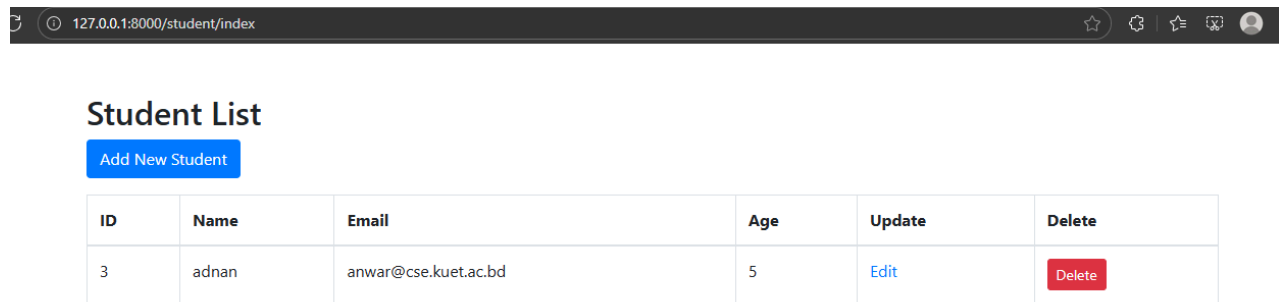
```

> routes
> console.php
> web.php
> storage
> tests
> vendor
> .editorconfig
> .env
> .env.example
> .gitattributes

15 Route::get('student/create',[StudentController::Class,'create'])->name('student.create');
16
17 Route::post('student/create',[StudentController::Class,'store'])->name('student.store');
18
19 Route::get('student/index',[StudentController::Class,'index'])->name('student.index');

```

Now start the server using *php artisan serve* and go to student/index to see the following table



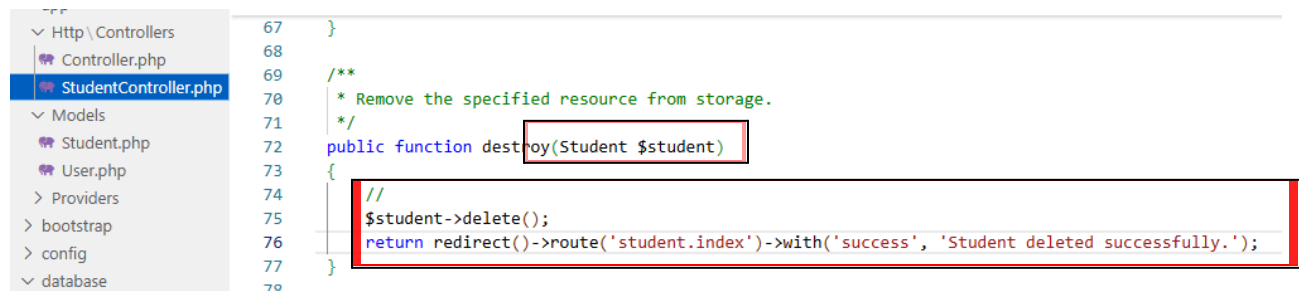
ID	Name	Email	Age	Update	Delete
3	adnan	anwar@cse.kuet.ac.bd	5	<a href="#">Edit</a>	<a href="#">Delete</a>

## Delete Data

To delete a model, you may call the *delete* method on the model instance. Here, we are retrieving the model from the database before calling the *delete* method. However, if you know the primary key of the model, you may delete the model without explicitly retrieving it by calling the *destroy* method. The *destroy* method loads each model individually and calls the *delete* method so that the deleting and deleted events are properly dispatched for each model.

for better understanding visit: <https://laravel.com/docs/master/eloquent#deleting-models>

Now, go StudentController.php and define the function public function destroy(Student \$student). We will use *delete* method. So we need to modify the function parameter to Student \$student.



```
67 }
68
69 /**
70  * Remove the specified resource from storage.
71  */
72 public function destroy(Student $student)
73 {
74     //
75     $student->delete();
76     return redirect()->route('student.index')->with('success', 'Student deleted successfully.');
```

Now we need to edit the index.blade.php file and web.php file.

In the index.blade.php file we need to modify the form action attribute where we not only need to add route but also pass the appropriate model's values. We also need to mention the method [@method\('delete'\)](#) or this route.

for details about : <https://www.cloudways.com/blog/routing-in-laravel/>

<https://laravel.com/docs/12.x/routing>

```

38
39
40 <form action="{{route('student.destroy',$student)}}" method="POST" style="display:inline;">
41 @csrf
42 @method('delete')
43 <button type="submit" class="btn btn-danger btn-sm">Delete</button>
44 </form>
45 </td>
46 <tr>
47 <td>

```

To show the message “successfully deleted”, we need to add the following session keys

```
<!-- Success Message -->
```

```

@if(session('success'))

    <div class="alert alert-success">

        {{ session('success') }}

    </div>

@endif

```

```

12 <!-- Success Message -->
13 @if(session('success'))
14     <div class="alert alert-success">
15         {{ session('success') }}
16     </div>
17 @endif
18
19 <table class="table table-bordered">
20     <thead>
21         <tr>

```

You can also add an alert box while pressing delete button in the following manner:

```

43
44 <form action="{{route('student.destroy',$student)}}" method="POST" style="display:inline;">
45 @csrf
46 @method('delete')
47 <button type="submit" class="btn btn-danger btn-sm" onclick="return confirm('Are you sure?')">Delete</button>
48 </form>
49 </td>
50 <tr>

```

Now, we need to update the routes in web.php file

```

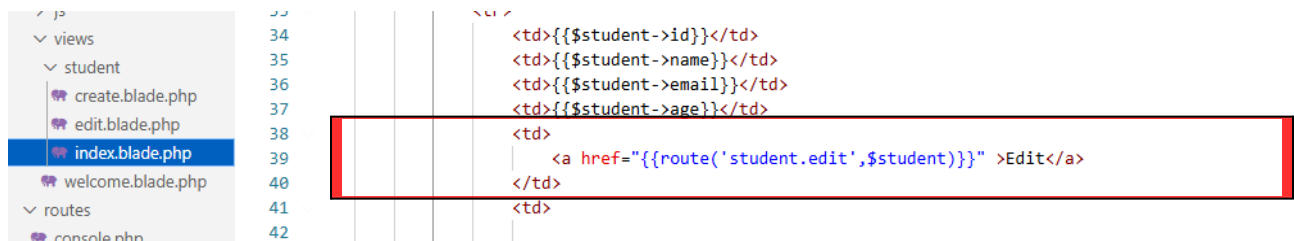
18
19 Route::get('student/index',[StudentController::Class,'index']->name('student.index'));
20
21 Route::delete('student/{student}',[StudentController::Class,'destroy']->name('student.destroy'));

```

## Update Data

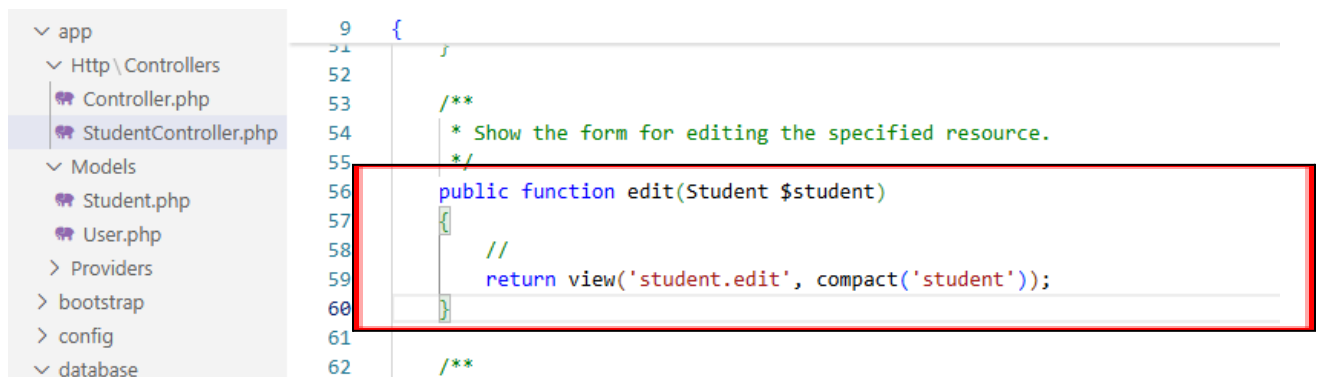
Updating data can be divided into two steps: 1. Editing the existing data 2. Storing the updated data

When a user submits/presses the edit button in index.blade.php file, it will be redirected to the edit.blade.php page along with the \$student model's information. Go to index.blade.php file and add the route to student.edit.



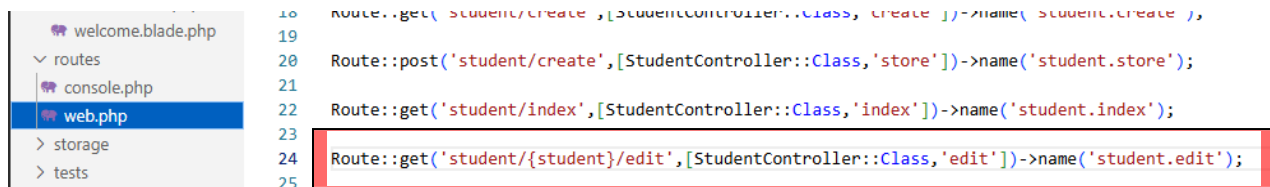
```
34 <td>{{ $student->id }}</td>
35 <td>{{ $student->name }}</td>
36 <td>{{ $student->email }}</td>
37 <td>{{ $student->age }}</td>
38 <td>
39     <a href="{route('student.edit',$student)}" >Edit</a>
40 </td>
41 </tr>
42 </tbody>
```

Now, go to StudentController.php and define public function edit to redirect it to student.edit page



```
51 {
52
53     /**
54      * Show the form for editing the specified resource.
55      */
56     public function edit(Student $student)
57     {
58         //
59         return view('student.edit', compact('student'));
60     }
61
62     /**
```

Then also update the route in web.php file



```
18 Route::get('student/create', [StudentController::class, 'create'])->name('student.create');
19
20 Route::post('student/create', [StudentController::class, 'store'])->name('student.store');
21
22 Route::get('student/index', [StudentController::class, 'index'])->name('student.index');
23
24 Route::get('student/{student}/edit', [StudentController::class, 'edit'])->name('student.edit');
25
```

Now, we want to read the values from the database and show them in the edit.blade.php form so that users can see the existing **values** and edit it. Now go to edit.blade.php file and we will add value attribute to each of the inputs in the form



```

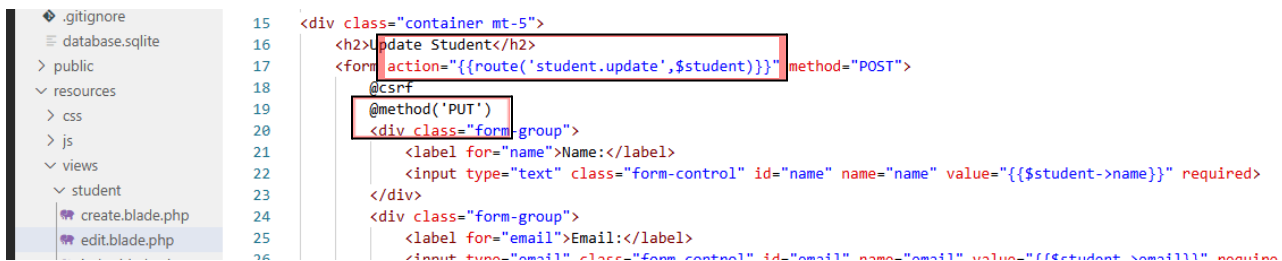
15 <div class="container mt-5">
16 <h2>Update Student</h2>
17 <form action="" method="POST">
18 @csrf
19 <div class="form-group">
20 <label for="name">Name:</label>
21 <input type="text" class="form-control" id="name" name="name" value="{{ $student->name }}" required>
22 </div>
23 <div class="form-group">
24 <label for="email">Email:</label>
25 <input type="email" class="form-control" id="email" name="email" value="{{ $student->email }}" required>
26 </div>
27 <div class="form-group">
28 <label for="age">Age:</label>
29 <input type="number" class="form-control" id="age" name="age" value="{{ $student->age }}" required>
30 </div>
31 <button type="submit" class="btn btn-primary">Update</button>
32 <a href="" class="btn btn-secondary">Back</a>
33 </form>

```

Now, our second task is to update the values. We will create a public function update in StudentController.php , then give routes to update functions in the web.php file and also add the route in the form action attribute of our edit.blade.php file and mention the `@method('put')` specifically for update database purpose.

First add the route in the form action and `@method('put')` in the edit.blade.php file.

For more about routing: <https://www.cloudways.com/blog/routing-in-laravel/>

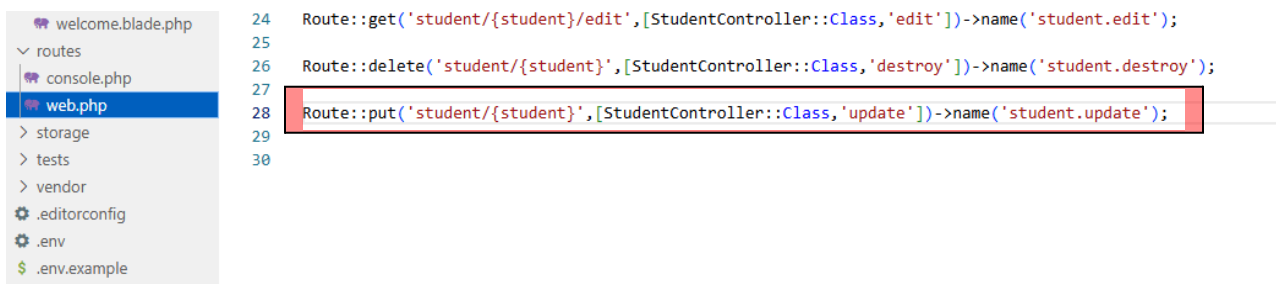


```

15 <div class="container mt-5">
16 <h2>Update Student</h2>
17 <form action="{{ route('student.update', $student) }}" method="POST">
18 @csrf
19 @method('PUT')
20 <div class="form-group">
21 <label for="name">Name:</label>
22 <input type="text" class="form-control" id="name" name="name" value="{{ $student->name }}" required>
23 </div>
24 <div class="form-group">
25 <label for="email">Email:</label>
26 <input type="email" class="form-control" id="email" name="email" value="{{ $student->email }}" required>
27 </div>
28 <div class="form-group">
29 <label for="age">Age:</label>
30 <input type="number" class="form-control" id="age" name="age" value="{{ $student->age }}" required>
31 </div>
32 <button type="submit" class="btn btn-primary">Update</button>
33 <a href="" class="btn btn-secondary">Back</a>
34 </form>

```

Now, go to the web.php file and add the route to the student.update file.



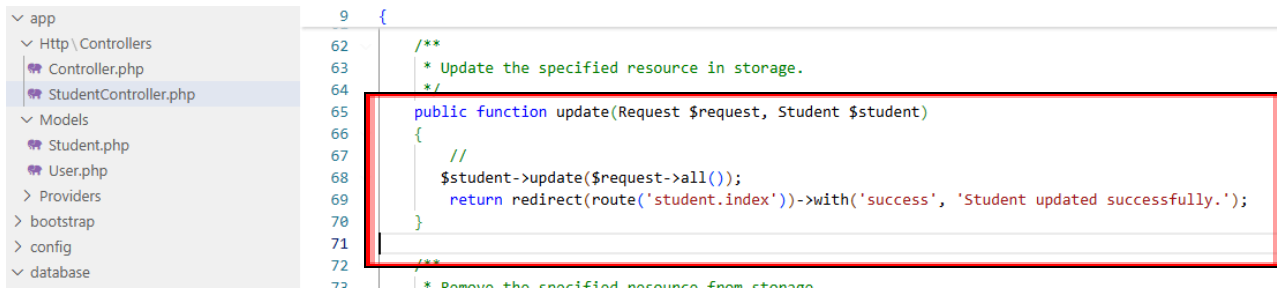
```

24 Route::get('student/{student}/edit', [StudentController::class, 'edit'])->name('student.edit');
25
26 Route::delete('student/{student}', [StudentController::class, 'destroy'])->name('student.destroy');
27
28 Route::put('student/{student}', [StudentController::class, 'update'])->name('student.update');
29
30

```

And Finally we will now define the function for update in StudentController. We will use *update* method to update the data of the student model. More details:

<https://laravel.com/docs/master/eloquent#updates>



If you want you can also add from validation the same as **public function store** in the StudentController.php

## CSRF Protection

All the post, put, and delete requests require the CSRF token to be sent along with the request. Otherwise, the request will be rejected. We need to use `@csrf` inside the form tag. In Laravel, `@csrf` is used inside Blade form to protect it from CSRF attacks (Cross-Site Request Forgery). Without `@csrf` you will get an error.

## Mass Assignment

You may use the *create* method to "save" a new model using a single PHP statement. However, before using the *create* method, you will need to specify either a *fillable* or *guarded* property on your *model class*. These properties are required because all Eloquent models are protected against mass assignment vulnerabilities by default.

A mass assignment vulnerability occurs when a user passes an unexpected HTTP request field and that field changes a column in your database that you did not expect. For example, a malicious user might send an `is_admin` parameter through an HTTP request, which is then passed to your model's *create* method, allowing the user to escalate themselves to an administrator.

So, to get started, you should define which model attributes you want to make mass assignable. You may do this using the *\$fillable* property on the model.

## Form Method Spoofing

The use of `@method(...)` is called form method spoofing in Laravel and is a requirement as HTML forms do not support PUT, PATCH, or DELETE actions. The value of the `@method` is sent as part of the form request and used by Laravel to determine how to process the form submission.

So, when defining PUT, PATCH, or DELETE routes that are called from an HTML form, you will need to add a hidden `_method` field to the form. The value sent with the `_method` field will be used as the HTTP request method. For convenience, we have used the `@method` Blade directive to generate the `_method` input field.

## Interacting With the Session & Redirecting With Flashed Session Data:-

- **Flash Data:** Sometimes you may wish to store items in the session for the next request. You may do so using the flash method. Data stored in the session using this method will be available immediately and during the subsequent HTTP request. After the subsequent HTTP request, the flashed data will be deleted. Flash data is primarily useful for short-lived status messages.
- **Redirecting With Flashed Session Data:** Redirecting to a new URL and flashing data to the session are usually done at the same time. Typically, this is done after successfully performing an action when you flash a success message to the session. For convenience, you may create a RedirectResponse instance and flash data to the session in a single, fluent method chain:

Example: The success messages that we have sent throughout today's work is done by session

```
//  
$student->update($request->all());  
return redirect(route('student.index'))->with('success', 'Student updated successfully.');
```

- After the user is redirected, you may display the flashed message from the session. For our case it was the following Blade syntax:

```
<!-- Success Message -->  
@if(session('success'))  
    <div class="alert alert-success">  
        {{ session('success') }}  
    </div>  
@endif
```

## HTTP Redirects

### Resources:

- <https://medium.com/@zulfikarditya/model-attribute-casting-in-laravel-complete-guide-to-accessors-and-mutators-b8e67e8c00df>
- <https://laravel.com/docs/master/eloquent#updates>
- <https://www.cloudways.com/blog/routing-in-laravel/>
- <https://laravel.com/docs/12.x/routing>
- <https://laravel.com/docs/12.x/controllers>
- <https://laravel.com/docs/master/eloquent#retrieving-models>
- <https://laravel.com/docs/master/eloquent#deleting-models>
- <https://laravel.com/docs/master/queries>

- <https://laravel.com/docs/master/eloquent#mass-assignment>
- <https://laravel.com/docs/12.x/csrf>
- <https://laravel.com/docs/11.x/redirects#redirecting-with-flash-data>
- <https://laravel.com/docs/11.x/redirects#redirecting-with-flashed-session-data>
- <https://laravel.com/docs/12.x/validation#quick-writing-the-validation-logic>
- Repopulating forms: <https://laravel.com/docs/12.x/validation#repopulating-forms>