# Intro

In this guide, we will be creating a simple students table and perform all CRUD operations on it. We will keep it simple and only have three columns namely fname, lname and email address.

Use this repo as a reference: https://github.com/pixelrio/LaravelCRUD

# Install Laravel

To install Laravel, we will use the composer’s create-project command and use Laravel/Laravel to create a Laravel project. The name of the project can be anything you want:

composer create-project laravel/laravel projectName

Once Laravel is installed, open the project in VS Code and edit the .env File. Uncomment the following lines and ensure the Host, Port, Database Name, Username and Password are correctly filled based on your local environment.

# Update .ENV file

DB\_CONNECTION=mysql

# DB\_HOST=127.0.0.1

# DB\_PORT=8889

# DB\_DATABASE=lms

# DB\_USERNAME=root

# DB\_PASSWORD=root

# Run the Migrations

Once you have updated the .ENV file, use the CD command to enter your project directory. You can now run the PHP Artisan commands as you are in the Laravel project directory. Since the new version of Laravel runs some migrations as part of installation, make sure you add the refresh flag with migrate as shown below.

cd projectName/

php artisan migrate:refresh

Once you have run the migrations, you can run the following command to serve the application, leave this tab of the terminal run in the background and all subsequent commands can be run in a new tab so that the server can keep running.

php artisan serve

# PHP Artisan Make

PHP Artisan Make command is used to make Models, Migrations, Seeders, Factories, and Controllers. You can individually make any necessary components as required or you can use the following command which will create everything for you. We will be handling the validation using the requests file so you can add both -a and -requests flag as shown below:

php artisan make:model Student -a --requests

Once you execute the above command, you will see these new files in your project folder:

app\Http\Models\Student.php

app\Http\Controllers\StudentController.php

app\Http\Requests\StoreStudentRequest.php

app\Http\Requests\UpdateStudentRequest.php

database\migrations\create\_students\_table.php

database\factory\studentFactory.php

database\seeders\StudentSeeder.php

# Migration File

Migration file contains the table structure and you can define the columns names and types as desired. In this example, you will see that Laravel automatically adds ID and Timestamps and we have added fname, lname and email as strings.

You can add nullable and unique attributes as required.

We will be using softDeletes which allow us to enable trash, restore and permanenent delete capabilities within our Model.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

/\*\*

\* Run the migrations.

\*/

public function up(): void

{

Schema::create('students', function (Blueprint $table) {

$table->id();

$table->string('fname');

$table->string('lname');

$table->string('email');

$table->softDeletes();

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*/

public function down(): void

{

Schema::dropIfExists('students');

}

};

# Model File

Since we are using softDeletes , ensure that the softDeletes has been added in the imports and then use it in the Student Model as shown below.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\SoftDeletes;

class Student extends Model

{

use HasFactory;

use SoftDeletes;

protected $dates = ['deleted\_at'];

protected $fillable = [

'fname',

'lname',

'email',

];

}

In the above code, we are passing column names to protected **$fillable** array to enable mass assignment of these fields, otherwise Laravel will not add records into **products** table if we use the Laravel Eloquent to add data.

# Factory File

The Factory methos allows us to create some test data, we will be using the Fake method in Laravel to generate random firstName, lastName and email addresses.

<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/\*\*

\* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Student>

\*/

class StudentFactory extends Factory

{

/\*\*

\* Define the model's default state.

\*

\* @return array<string, mixed>

\*/

public function definition(): array

{

return [

'fname' => fake()->firstName(),

'lname' => fake()->lastName(),

'email' => fake()->unique()->safeEmail(),

];

}

}

# Seeder File

We will be calling our Student Factory file in the main seeder file, ensure you have added the Student Model at the top for it to work.

Student::factory(20)->create();

<?php

namespace Database\Seeders;

use App\Models\Student;

use App\Models\User;

// use Illuminate\Database\Console\Seeds\WithoutModelEvents;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Seed the application's database.

\*/

public function run(): void

{

// User::factory(10)->create();

User::factory()->create([

'name' => 'Test User',

'email' => 'test@example.com',

]);

Student::factory(20)->create();

}

}

# Migrate and Seed

Now that we have the seeder ready, we can migrate again and this time add --seed flag along with it. The seed flag runs the seeder along with the migrations which will add test data to our app.

php artisan migrate:refresh --seed

# Routing

For the routing to work, we can add the StudentController on the top and add one single resource route which will enable all basic routes such as create, update, delete etc but since we are using softDeletes, we have to manually define routes for trash, trashed and restore as shown below:

<?php

use app\Models\Student;

use App\Http\Controllers\StudentController;

use Illuminate\Support\Facades\Route;

Route::get('/', function () {

return view('welcome');

});

Route::get(

'students/trash/{id}',

[StudentController::class, 'trash']

)->name('students.trash');

Route::get(

'students/trashed/',

[StudentController::class, 'trashed']

)->name('students.trashed');

Route::get(

'students/restore/{id}',

[StudentController::class, 'trash']

)->name('students.restore');

Route::resource('students', StudentController::class);

You can check all existing routes using command line with the help of the command below:

php artisan route:list

## Creating Files

You can manually create files in Laravel or use the make:view command to generate the files. In the following example, we are creating a main template file called admin under layouts folder and 4 CRUD files under students folder. The period is used to define the hierarchy of folders.

php artisan make:view layouts.admin

php artisan make:view students.index

php artisan make:view students.create

php artisan make:view students.edit

php artisan make:view students.show

## Layout File

The layout file is pretty standard and we are using Bootstrap 5 using CDN and we are also using a library called Toastr for instant notifications. Whereever you want the dynamic content to show up, you can use @yield(‘content’) to create a space where we could inject dynamic content in this template from the individual files that will use this template.

From the toastr library’s documentation, we will add some scaffolding for getting and displaying session information.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Lara LMS</title>

<meta name="csrf-token" content="{{ csrf\_token() }}">

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

<link href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.css" rel="stylesheet">

</head>

<body>

<div class="container-fluid">

<div class="container">

<nav class="navbar navbar-expand-lg bg-body-tertiary">

<div class="container-fluid">

<a class="navbar-brand" href="#">Laravel LMS</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link active" aria-current="page" href="#">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Students</a>

</li>

</ul>

</div>

</div>

</nav>

</div>

</div>

<div class="container-fluid mt-5">

<div class="container">

@yield('content')

</div>

</div>

<script src="https://code.jquery.com/jquery-3.7.1.js" integrity="sha256-eKhayi8LEQwp4NKxN+CfCh+3qOVUtJn3QNZ0TciWLP4=" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

<script>

@if(Session::has('success'))

toastr.success("{{ Session::get('success') }}")

@endif

@if( Session::has('info'))

toastr.info("{{ Session::get('info') }}")

@endif

@if( Session::has('error'))

toastr.info("{{ Session::get('error') }}")

@endif

</script>

</body>

</html>

## Controller File

The controller file is where all the magic happens, the logic for all CRUD operation goes here. In this example, we will be using the Str and Request to support writing to the sessions and retrieving the request data.

<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreStudentRequest;

use App\Http\Requests\UpdateStudentRequest;

use App\Models\Student;

use Illuminate\Support\Str;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Session;

class StudentController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*/

public function index()

{

return view('students.index', [

'students' => Student::all()

]);

}

/\*\*

\* Show the form for creating a new resource.

\*/

public function create()

{

return view('students.create');

}

/\*\*

\* Store a newly created resource in storage.

\*/

public function store(StoreStudentRequest $request)

{

Student::create($request->validated());

Session::flash('success', 'Student added successfully');

return redirect() -> route('students.index');

}

/\*\*

\* Display the specified resource.

\*/

public function show(Student $student)

{

return view('students.show', compact('student'));

}

/\*\*

\* Show the form for editing the specified resource.

\*/

public function edit(Student $student)

{

return view('students.edit', compact('student'));

}

/\*\*

\* Update the specified resource in storage.

\*/

public function update(UpdateStudentRequest $request, Student $student)

{

$student->update($request->validated());

}

/\*\*

\* Remove the specified resource from storage.

\*/

public function trash($id)

{

Student::Destroy($id);

Session::Flash('success', 'Student trashed successfully');

return redirect() -> route('students.index');

}

public function destroy($id)

{

$student = Student::withTrashed() -> where('id', $id) -> first();

$student -> forceDelete();

Session::Flash('success', 'Student deleted successfully');

return redirect() -> route('students.index');

}

public function restore($id)

{

$student = Student::withTrashed() -> where('id', $id) -> first();

$student -> restore();

Session::Flash('success', 'Student restored successfully');

return redirect() -> route('students.trashed');

}

}

If you recall from one of the initial steps where we create the model, we used the --requests flag, it allows us to write the validation logic in separate files. In the file above you will see that we have two files as shown below which will have our data validation logic.

use App\Http\Requests\StoreStudentRequest;

use App\Http\Requests\UpdateStudentRequest;

### StoreStudentRequest File

Laravel supports a variety of validation including checking for unique values across tables, data-types, minimum/maximum values etc.

<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreStudentRequest extends FormRequest

{

/\*\*

\* Determine if the user is authorized to make this request.

\*/

public function authorize(): bool

{

return true;

}

/\*\*

\* Get the validation rules that apply to the request.

\*

\* @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>

\*/

public function rules(): array

{

return [

'fname' => 'required|string',

'lname' => 'required|string',

'email' => 'required',

];

}

}

### UpdateStudentRequest File

<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UpdateStudentRequest extends FormRequest

{

/\*\*

\* Determine if the user is authorized to make this request.

\*/

public function authorize(): bool

{

return false;

}

/\*\*

\* Get the validation rules that apply to the request.

\*

\* @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>

\*/

public function rules(): array

{

return [

'fname' => 'required|string',

'lname' => 'required|string',

'email' => 'required',

];

}

}

# Views

## Index

@extends('layouts/admin')

@section('content')

<div class="row">

<div class="col">

<h1 class="display-2">

View all Students

</h1>

</div>

</div>

<div class="row">

@foreach($students as $student)

<div class="col-md-4 mb-3">

<div class="card" style="width: 18rem;">

<div class="card-body">

<h5 class="card-title">{{ $student -> fname }} {{ $student -> lname }}</h5>

<a href="{{ route('students.edit', $student -> id ) }}" class="card-link">Edit</a>

<a href="{{ route('students.trash', $student -> id )}}" class="card-link">Delete</a>

</div>

</div>

</div>

@endforeach

</div>

@endsection

## Create

@extends('layouts/admin')

@section('content')

<div class="row">

<div class="col">

<h1 class="display-2">

Add a Student Profile

</h1>

</div>

</div>

<div class="row">

<form action="{{ route('students.store') }}" method="post">

@if ($errors->any())

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

{{ csrf\_field() }}

<div class="mb-3">

<label for="fname" class="form-label">First Name</label>

<input type="text" class="form-control" id="fname" name="fname" aria-describedby="fname">

</div>

<div class="mb-3">

<label for="lname" class="form-label">Last Name</label>

<input type="text" class="form-control" id="lname" name="lname" aria-describedby="lname">

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" class="form-control" id="email" name="email" aria-describedby="email">

@error('email')

<span class="text-danger" role="alert">

<strong>{{ $message }}</strong>

</span>

@enderror

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

@endsection

## Edit

@extends('layouts/admin')

@section('content')

<div class="row">

<div class="col">

<h1 class="display-2">

Update Student Profile

</h1>

</div>

</div>

<div class="row">

<form action="{{ route('students.update', $student -> id) }}" method="PUT">

@if ($errors->any())

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

{{ csrf\_field() }}

<div class="mb-3">

<label for="fname" class="form-label">First Name</label>

<input type="text" class="form-control" id="fname" name="fname" aria-describedby="fname" value="{{ $student -> fname }}">

</div>

<div class="mb-3">

<label for="lname" class="form-label">Last Name</label>

<input type="text" class="form-control" id="lname" name="lname" aria-describedby="lname" value="{{ $student -> lname }}">

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" class="form-control" id="email" name="email" aria-describedby="email" value="{{ $student -> email }}">

@error('email')

<span class="text-danger" role="alert">

<strong>{{ $message }}</strong>

</span>

@enderror

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

@endsection

## Show

@extends('layouts/admin')

@section('content')

<div class="row">

<div class="col">

<h1 class="display-2">

Student Profile

</h1>

</div>

</div>

<div class="row">

<div class="col-md-4 mb-3">

<div class="card" style="width: 18rem;">

<div class="card-body">

<h5 class="card-title">{{ $student -> fname }} {{ $student -> lname }}</h5>

<a href="{{ route('students.edit', $student -> id ) }}" class="card-link">Edit</a>

<a href="{{ route('students.trash', $student -> id )}}" class="card-link">Delete</a>

</div>

</div>

</div>

</div>

@endsection

# Adding Authentication

Before using the commands below, ensure that you have the latest version of node and npm installed on your machine. For Authentication, we will be using Breeze which is one of the official Laravel packages, and it can work with both React and Vuejs. Breeze also works with LiveWire which is very similar to Vuejs but it is still in development mode and hasn’t matured as a library yet.

composer require laravel/breeze –dev

php artisan breeze:install

php artisan migrate:refresh --seed

npm install

npm run dev

After you have installed Breeze, you will have new routes in your Laravel App:

`

http://127.0.0.1:8000/register

http://127.0.0.1:8000/login

<http://127.0.0.1:8000/dashboard>

# Relationships

Laravel supports all different kinds of relationships such as one-to-one, one-to-many, many-to-many etc. One-to-one and one-to-many are easy to implement and can be configured within the migration file by adding the appropriate foreign keys as shown below:

In this example, an author can have multiple books:

Schema::create('books', function (Blueprint $table) {

$table->id();

$table->string('title');

$table->text('description');

$table->unsignedBigInteger('author\_id');

$table->foreign('author\_id')->references('id')->on('authors');

$table->timestamps();

});

app/Models/Author.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Author extends Model

{

public function books()

{

return $this->hasMany(Book::class);

}

}

app/Models/Book.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Book extends Model

{

public function author()

{

return $this->belongsTo(Author::class);

}

}

Retrieving Books and Authors:

/ Retrieve books by a specific author (via hasMany)

$author = Author::first();

$books = $author->books;

dump($books);

// Retrieve author of a book (via belongsTo)

$book = Book::first();

$author = $book->author;

dump($author);

The most complicated relationship is the many to many and we will implement this in our LMS app. In our app, one course can belong to many students and one student can belong to many courses and hence the need for a many-to-many relationship. In Laravel, many-to-many relationship has to have a pivot table, whereas all other relationship types can be implemented without the need of a pivot table

In Laravel, a pivot table is created **by checking the alphabetical order of the two-parent tables**using their **singular names**. For example, if we had parent **table “a”** and parent**table “b”**, then the **pivot table would be “a\_b”**.

php artisan make:migration create\_course\_student\_table

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

/\*\*

\* Run the migrations.

\*/

public function up(): void

{

Schema::create('course\_student', function (Blueprint $table) {

$table->id();

$table->integer('course\_id');

$table->integer('student\_id');

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*/

public function down(): void

{

Schema::dropIfExists('course\_student');

}

};

## Course Controller File

<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreCourseRequest;

use App\Http\Requests\UpdateCourseRequest;

use App\Models\Course;

use App\Models\Student;

class CourseController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*/

public function index()

{

return view('courses.index');

}

/\*\*

\* Show the form for creating a new resource.

\*/

public function create()

{

//

}

/\*\*

\* Store a newly created resource in storage.

\*/

public function store(StoreCourseRequest $request)

{

//

}

/\*\*

\* Display the specified resource.

\*/

public function show(Course $course, $id)

{

// dd($id);

$course = Course::find($id);

$student = $course->students;

return view('courses.show', ['course' => $course, 'students' => $student]);

}

/\*\*

\* Show the form for editing the specified resource.

\*/

public function edit(Course $course)

{

//

}

/\*\*

\* Update the specified resource in storage.

\*/

public function update(UpdateCourseRequest $request, Course $course)

{

//

}

/\*\*

\* Remove the specified resource from storage.

\*/

public function destroy(Course $course)

{

//

}

}

## Course Model File

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsToMany;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\SoftDeletes;

class Course extends Model

{

use HasFactory;

use SoftDeletes;

protected $dates = ['deleted\_at'];

protected $fillable = [

'courseName',

'courseID',

'description',

];

public function students(): BelongsToMany

{

return $this->belongsToMany(Student::class);

}

}

## Student Model File

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsToMany;

use Illuminate\Database\Eloquent\SoftDeletes;

class Student extends Model

{

use HasFactory;

use SoftDeletes;

protected $dates = ['deleted\_at'];

protected $fillable = [

'fname',

'lname',

'email',

];

public function courses(): BelongsToMany

{

return $this->belongsToMany(Course::class);

}

}

# Toastr Notifications

To use toastr notifications, use the following boilerplate which includes the CDN for the stylesheet, javascript library as well as jQuery.

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

<link href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.css" rel="stylesheet">

<script src="https://code.jquery.com/jquery-3.7.1.js" integrity="sha256-eKhayi8LEQwp4NKxN+CfCh+3qOVUtJn3QNZ0TciWLP4=" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

Add the following code below the Toastr JS file:

<script>

@if(Session::has('success'))

toastr.success("{{ Session::get('success') }}")

@endif

@if( Session::has('info'))

toastr.info("{{ Session::get('info') }}")

@endif

@if( Session::has('error'))

toastr.info("{{ Session::get('error') }}")

@endif

</script>

The above piece of code might show red underlines which is completely normal as it is not valid JavaScript and depending on what extensions you have installed, VS code might not like it. This piece of code simply checks if certain session messages are set and if yes, they are displayed as Toastr notifications.

For example, in the code below, we are using Laravel Blade conditional within JavaScript to check for a session variable called success and if it exists, it will show a toastr notification.

@if(Session::has('success'))

toastr.success("{{ Session::get('success') }}")

@endif

Toastr.success, Toastr.info, Toastr.error have built in styling. To generate a notification, a session has to be set first which can be done in the controller file once an action has taken place as shown below:

Session::flash('success', 'Student added successfully');

# Error Handling

We can display error messages on the top, as well as show the old value of the field along with manipulating the classes based on errors.

@extends('layouts/admin')

@section('content')

<div class="row">

<div class="col">

<h1 class="display-2">

Add a Student Profile

</h1>

</div>

</div>

<div class="row">

<form action="{{ route('students.store') }}" method="post">

@if ($errors->any())

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

{{ csrf\_field() }}

<div class="mb-3">

<label for="fname" class="form-label">First Name</label>

<input type="text" name="fname" class="form-control @error('fname') is-invalid @enderror" placeholder="First Name" value="{{ old('fname') }}">

@error('fname')

<span class="invalid-feedback" role="alert">

<strong>{{ $message }}</strong>

</span>

@enderror

</div>

<div class="mb-3">

<label for="lname" class="form-label">Last Name</label>

<input type="text" name="lname" class="form-control @error('lname') is-invalid @enderror" placeholder="Last Name" value="{{ old('lname') }}">

@error('lname')

<span class="invalid-feedback" role="alert">

<strong>{{ $message }}</strong>

</span>

@enderror

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="text" name="email" class="form-control @error('email') is-invalid @enderror" placeholder="Email" value="{{ old('email') }}">

@error('email')

<span class="text-danger" role="alert">

<strong>{{ $message }}</strong>

</span>

@enderror

</div>

<div class="mb-3">

<label for="courseName" class="form-label">Course</label>

<select name="course" id="course">

@foreach ($courses as $course)

<option value="{{ $course -> id }}">{{ $course -> courseName }}</option>

@endforeach

</select>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

@endsection