



Laravel

Student CRUD API using Laravel Framework

Presented by: Venkat

Course: Backend Web Development

Instructor: Dr. Sheik Anik

What is a PHP Framework?

A PHP framework is a software framework that provides a basic structure for building web applications using PHP. It offers a standardized way to build and deploy applications, promoting code reuse and adhering to best practices, ultimately streamlining the development process.



Accelerate Development

Frameworks provide pre-built modules and functions, reducing the need to write code from scratch. This significantly speeds up the development process.



Structured Codebase

They enforce architectural patterns (like MVC), making the codebase organized, consistent, and easier to manage, especially in large projects with multiple developers.



Enhanced Security

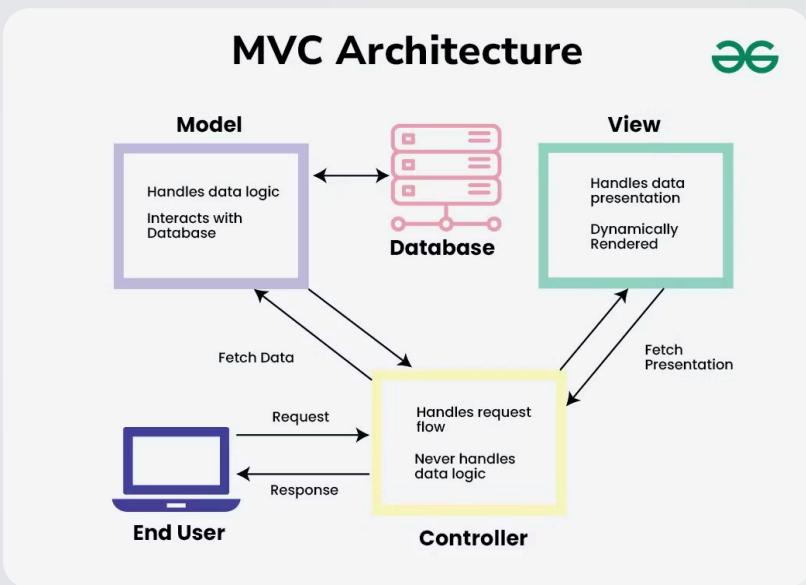
Frameworks often come with built-in security features, protecting against common web vulnerabilities like XSS, CSRF, and SQL injection, making applications more robust.



Easier Maintenance & Scaling

Standardized code and clear structures simplify debugging, updates, and the addition of new features, ensuring long-term maintainability and scalability.

Understanding Laravel's MVC Architecture



Model

Handles database operations and data structure

View

Returns output—JSON data for the API

Controller

Manages application logic and request handling

Laravel is a modern PHP framework based on the MVC architecture, providing elegant syntax and powerful tools for web development.

Framework Setup

01

Install Dependencies

Install XAMPP for PHP and MySQL, and Composer as Laravel's dependency manager

02

Create Laravel Project

```
composer create-project laravel/laravel student-crud-laravel
```

bash

```
composer create-project laravel/laravel student-crud-laravel
```

03

Configure Database

Connect to MySQL database `student_system` using the `.env` file

Database Configuration

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=student_system  
DB_USERNAME=root  
DB_PASSWORD=
```

.env File Settings

```
DB_DATABASE=student_system  
DB_USERNAME=root  
DB_PASSWORD=
```

The .env file stores environment-specific configuration, connecting Laravel to the MySQL database.

Launch Laravel Server

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\sabar> cd C:\projects\student-crud-laravel
PS C:\projects\student-crud-laravel> dir

Directory: C:\projects\student-crud-laravel

Mode                LastWriteTime       Length Name
----                -----       ----- 
d----        19-10-2025 12:42 PM           0 app
d----        19-10-2025 12:42 PM           0 bootstrap
d----        19-10-2025 12:42 PM           0 config
d----        19-10-2025 12:42 PM           0 database
d----        19-10-2025 12:42 PM           0 lang
d----        19-10-2025 03:34 PM           0 public
d----        19-10-2025 12:42 PM           0 resources
d----        19-10-2025 12:42 PM           0 routes
d----        19-10-2025 12:42 PM           0 storage
d----        19-10-2025 12:42 PM           0 tests
d----        19-10-2025 12:44 PM           0 vendor
-a---      31-01-2023 07:05 AM          258 .editorconfig
-a---      19-10-2025 03:06 PM          1127 .env
-a---      31-01-2023 07:05 AM          1069 .env.example
-a---      31-01-2023 07:05 AM          179 .gitattributes
-a---      31-01-2023 07:05 AM          227 .gitignore
-a---      31-01-2023 07:05 AM          1686 artisan
-a---      31-01-2023 07:05 AM          1817 composer.json
-a---      19-10-2025 12:42 PM         301796 composer.lock
-a---      31-01-2023 07:05 AM          286 package.json
-a---      31-01-2023 07:05 AM          1175 phpunit.xml
-a---      31-01-2023 07:05 AM          4158 README.md
-a---      31-01-2023 07:05 AM          263 vite.config.js

PS C:\projects\student-crud-laravel> php artisan serve
INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```



Navigate to Project

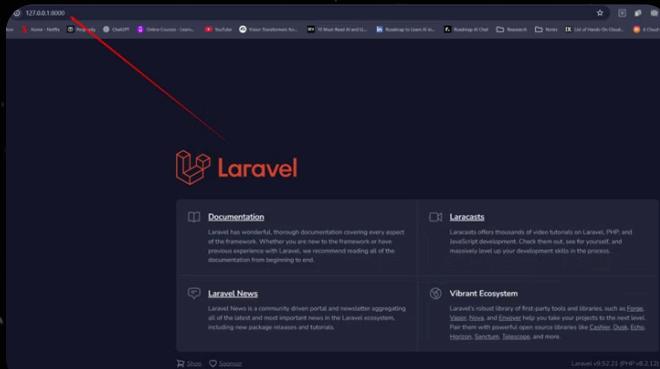
```
cd C:\projects\student-crud-laravel
```

Start Server

```
php artisan serve
```

Access Application

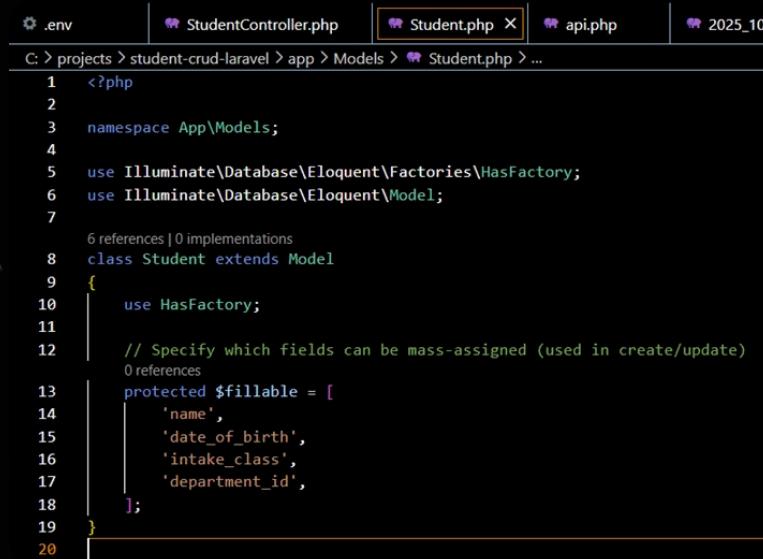
Visit <http://127.0.0.1:8000> to see Laravel welcome page



Building the CRUD Components

Student Model

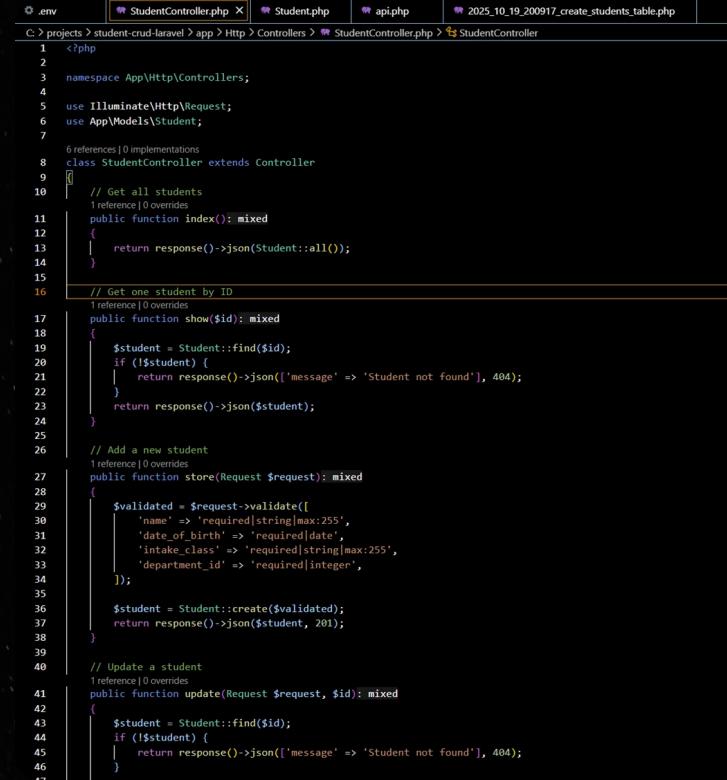
Defines database structure with \$fillable fields: name, date_of_birth, intake_class, department_id



```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Student extends Model
9 {
10     use HasFactory;
11
12     // Specify which fields can be mass-assigned (used in create/update)
13     protected $fillable = [
14         'name',
15         'date_of_birth',
16         'intake_class',
17         'department_id',
18     ];
19 }
```

Student Controller

Contains 5 methods: index(), show(), store(), update(), destroy() for complete CRUD operations



```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Student;
7
8 class StudentController extends Controller
9 {
10     // Get all students
11     public function index(): mixed
12     {
13         return response()->json(Student::all());
14     }
15
16     // Get one student by ID
17     public function show($id): mixed
18     {
19         $student = Student::find($id);
20         if (!$student) {
21             return response()->json(['message' => 'Student not found'], 404);
22         }
23         return response()->json($student);
24     }
25
26     // Add a new student
27     public function store(Request $request): mixed
28     {
29         $validated = $request->validate([
30             'name' => 'required|string|max:255',
31             'date_of_birth' => 'required|date',
32             'intake_class' => 'required|string|max:255',
33             'department_id' => 'required|integer',
34         ]);
35
36         $student = Student::create($validated);
37         return response()->json($student, 201);
38     }
39
40     // Update a student
41     public function update(Request $request, $id): mixed
42     {
43         $student = Student::find($id);
44         if (!$student) {
45             return response()->json(['message' => 'Student not found'], 404);
46         }
47     }
48 }
```

API Routes Configuration



GET /api/students

List all students



GET /api/students/{id}

Show single student



POST /api/students

Create new student



PUT /api/students/{id}

Update student



DELETE /api/students/{id}

Delete student

Routes connect URLs to controller methods. Verify with: `php artisan route:list`

```
.env          api.php X  StudentController.php  Student.php  2025_10
C: > projects > student-crud-laravel > routes > api.php
1  <?php
2
3  use Illuminate\Http\Request;
4  use Illuminate\Support\Facades\Route;
5  use App\Http\Controllers\StudentController;
6
7  Route::get('/students', [StudentController::class, 'index']);
8  Route::get('/students/{id}', [StudentController::class, 'show']);
9  Route::post('/students', [StudentController::class, 'store']);
10 Route::put('/students/{id}', [StudentController::class, 'update']);
11 Route::delete('/students/{id}', [StudentController::class, 'destroy']);
12
```

Laravel Student CRUD API

A complete demonstration of building and testing a Laravel CRUD API using VS Code, Tinker, and browser-based verification.

Opening the Terminal

01

Launch Terminal

Press **Ctrl+`** in VS Code to open the integrated terminal.

02

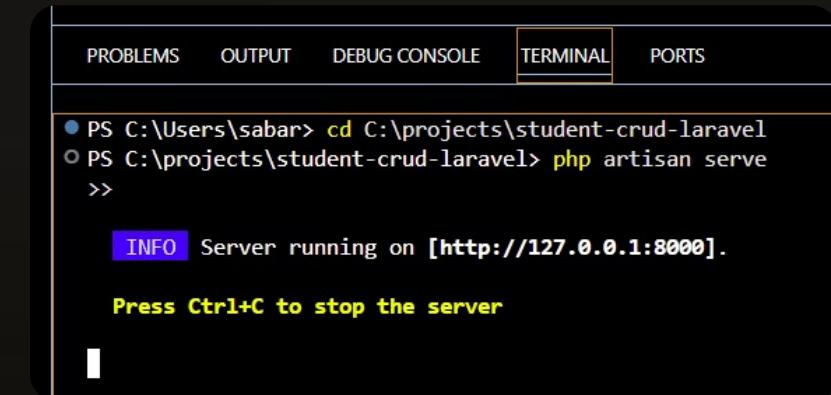
Verify Path

Ensure terminal shows `C:\projects\student-crud-laravel>`. If not, run `cd C:\projects\student-crud-laravel`.

03

Start Server

Execute `php artisan serve` to launch the development server.



A screenshot of the VS Code interface showing the terminal tab selected. The terminal window displays the following output:

```
PS C:\Users\sabar> cd C:\projects\student-crud-laravel
PS C:\projects\student-crud-laravel> php artisan serve
>>
INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

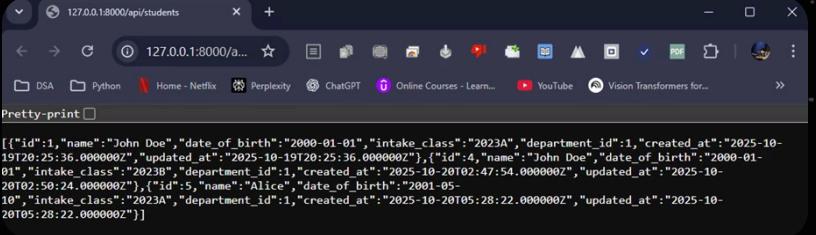
Accessing the Application

Server URL

Open the URL displayed in terminal, typically `http://127.0.0.1:8000`.

API Endpoint

Navigate to `http://127.0.0.1:8000/api/students` to view JSON response in browser.



A screenshot of a web browser window titled "127.0.0.1:8000/api/students". The address bar shows "127.0.0.1:8000/api/students". The page content is a JSON array of student records, with a "Pretty-print" button visible. The JSON data is as follows:

```
[{"id":1,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023A","department_id":1,"created_at":"2025-10-19T20:25:36.000000Z","updated_at":"2025-10-19T20:25:36.000000Z"}, {"id":4,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023B","department_id":1,"created_at":"2025-10-20T02:47:54.000000Z","updated_at":"2025-10-20T02:47:54.000000Z"}, {"id":5,"name":"Alice","date_of_birth":"2001-05-10","intake_class":"2023A","department_id":1,"created_at":"2025-10-20T05:28:22.000000Z","updated_at":"2025-10-20T05:28:22.000000Z"}]
```

Testing Routes

Confirm API endpoints using Laravel's built-in route list command.

```
php artisan route:list
```

This command displays all registered routes, methods, URIs, and corresponding controller actions.

```
PS C:\Users\sabar> cd C:\projects\student-crud-laravel
PS C:\projects\student-crud-laravel> php artisan route:list

GET|HEAD / .....
POST _ignition/execute-solution . ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/students ..... StudentController@index
POST api/students ..... StudentController@store
GET|HEAD api/students/{id} ..... StudentController@show
PUT api/students/{id} ..... StudentController@update
DELETE api/students/{id} ..... StudentController@destroy
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show

Showing [10] routes

PS C:\projects\student-crud-laravel>
```

CRUD Operations in Action

1 CREATE

```
Using Laravel Tinker: Student::create(['name' =>  
    'Alice', 'date_of_birth' => '2001-05-10',  
    'intake_class' => '2023A', 'department_id' => 1])
```

2 READ

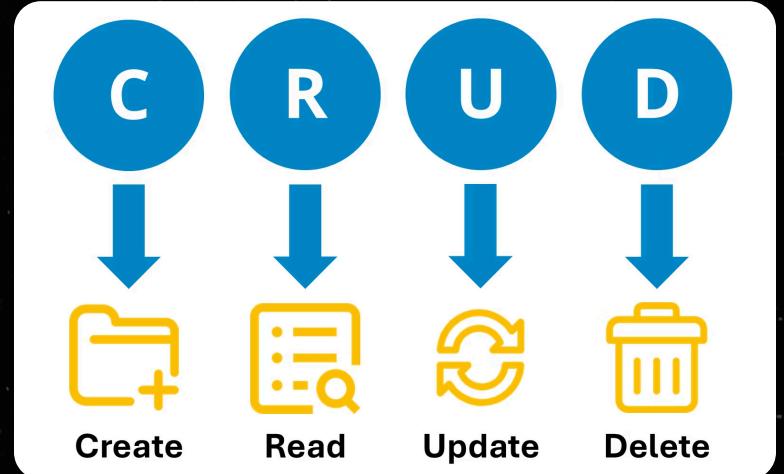
Fetch all: Student::all() or single: Student::find(1).
View at <http://127.0.0.1:8000/api/students>

3 UPDATE

```
Modify data: Student::find(1)->update(['intake_class' => '2023B'])
```

4 DELETE

Remove record: Student::find(1)->delete()



CREATE Operation



Launch Tinker

1

Run `php artisan tinker` to open Laravel's interactive shell.



Insert Record

2

Execute `create` command with student details: `name`, `date_of_birth`, `intake_class`, and `department_id`.



Verify in Browser

3

Visit `/api/students` to see the newly created record in JSON format.

This data is now inserted into my database. Let's check it in the browser.

💻 Go to: <http://127.0.0.1:8000/api/students>

```
[{"id":1,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023A","department_id":1,"created_at":"2025-10-19T20:25:36.000000Z","updated_at":"2025-10-19T20:25:36.000000Z"}, {"id":4,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023B","department_id":1,"created_at":"2025-10-20T02:47:54.000000Z","updated_at":"2025-10-20T02:50:24.000000Z"}, {"id":5,"name":"Alice","date_of_birth":"2001-05-10","intake_class":"2023A","department_id":1,"created_at":"2025-10-20T05:28:22.000000Z","updated_at":"2025-10-20T05:28:22.000000Z"}, {"id":6,"name":"Alice","date_of_birth":"2001-05-10","intake_class":"2023A","department_id":1,"created_at":"2025-10-20T06:12:13.000000Z","updated_at":"2025-10-20T06:12:13.000000Z"}]
```

```
PS C:\projects\student-crud-laravel> php artisan tinker
Psy Shell v0.12.12 (PHP 8.2.12 - cli) by Justin Hileman
> App\Models\Student::create([
    'name' => 'Alice',
    'date_of_birth' => '2001-05-10',
    'intake_class' => '2023A',
    'department_id' => 1
]);
=> App\Models\Student {#5187
    name: "Alice",
    date_of_birth: "2001-05-10",
    intake_class: "2023A",
    department_id: 1,
    updated_at: "2025-10-20 06:12:13",
    created_at: "2025-10-20 06:12:13",
    id: 6,
}
> |
```

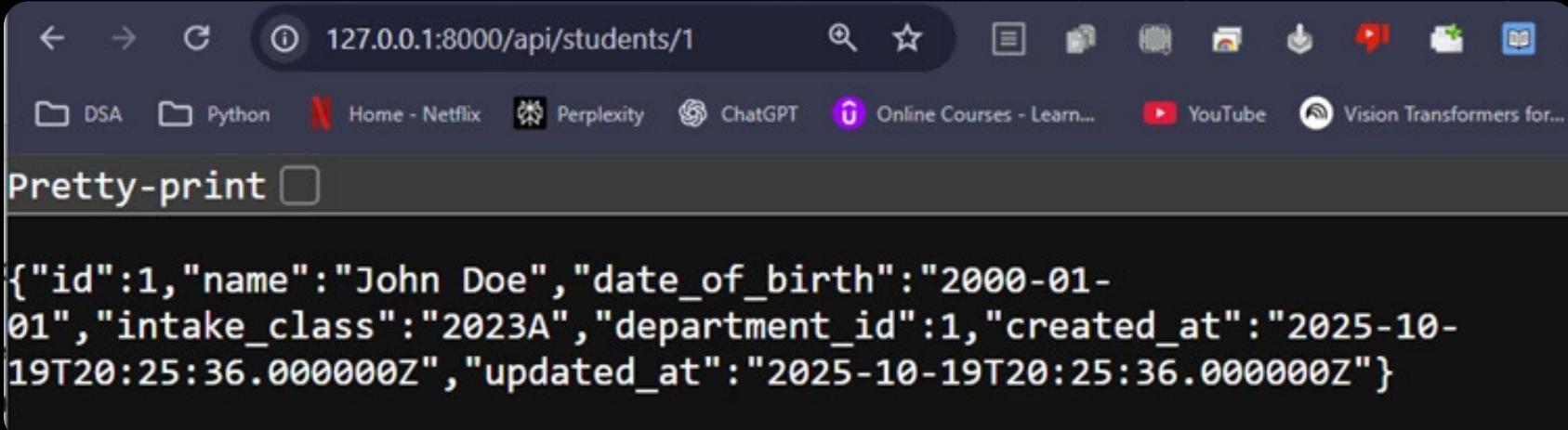
READ Operation

All Students

Access <http://127.0.0.1:8000/api/students> to retrieve all student records.

Single Student

Use <http://127.0.0.1:8000/api/students/1> to fetch a specific student by ID.



A screenshot of a web browser window. The address bar shows the URL <http://127.0.0.1:8000/api/students/1>. The page content is a JSON object representing a student record:

```
{"id":1,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023A","department_id":1,"created_at":"2025-10-19T20:25:36.000000Z","updated_at":"2025-10-19T20:25:36.000000Z"}
```

The browser interface includes a navigation bar with back, forward, and refresh buttons, and a toolbar with various icons. Below the address bar is a search bar and a tab bar with links to DSA, Python, Netflix, Perplexity, ChatGPT, Online Courses, YouTube, and Vision Transformers.

UPDATE Operation



Modify Data

Use Tinker to update student record: Student::find(1)->update(['intake_class' => '2023B'])

```
> App\Models\Student::find(1)->update(['intake_class' => '2023B']);  
= true
```



Verify Changes

Refresh browser to see updated data reflected in the JSON response.

A screenshot of a web browser window. The address bar shows the URL 127.0.0.1:8000/api/students/1. Below the address bar, there are several browser tabs and icons. A dropdown menu labeled "Pretty-print" is open. The main content area of the browser displays a JSON object:

```
{"id":1,"name":"John Doe","date_of_birth":"2000-01-01","intake_class":"2023B","department_id":1,"created_at":"2025-10-19T20:25:36.000000Z","updated_at":"2025-10-20T06:15:30.000000Z"}
```

DELETE Operation

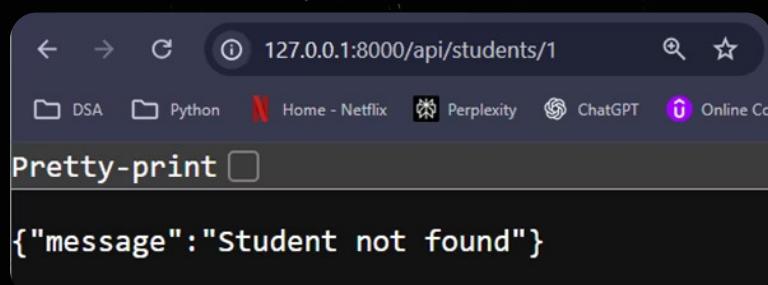
1 Execute Delete

Run `Student::find(1)->delete()` in Tinker to remove the record.

```
> App\Models\Student::find(1)->delete();
= true
```

2 Confirm Removal

Refresh browser endpoint—record no longer appears in JSON output.



Demonstrate CRUD (Using Tinker Only)

CREATE: Add a new student record

Run: `php artisan tinker`

```
App\Models\Student::create([
    'name' => 'Alice',
    'date_of_birth' => '2001-05-10',
    'intake_class' => '2023A',
    'department_id' => 1
]);
```

READ: Retrieve student records

Read All Students

```
>>> App\Models\Student::all();
```

Read a Single Student

```
>>> App\Models\Student::find(1);
```

UPDATE: Modify a student record

Run:

```
App\Models\Student::find(1)->update(['intake_class' => '2023B']);
```

```
>>> App\Models\Student::find(1);
```

DELETE: Remove a student record

Run: App\Models\Student::find(1)->delete();

```
>>> App\Models\Student::find(1);
```

```
Windows PowerShell x Windows PowerShell x + 
> App\Models\Student::find(1)->update(['intake_class' => '2023B']);
= true

> App\Models\Student::find(1)->delete();
= true

> App\Models\Student::all();
= Illuminate\Database\Eloquent\Collection {#5234
  all: [
    App\Models\Student {#5895
      id: 4,
      name: "John Doe",
      date_of_birth: "2000-01-01",
      intake_class: "2023B",
      department_id: 1,
      created_at: "2025-10-20 02:47:54",
      updated_at: "2025-10-20 02:50:24",
    },
    App\Models\Student {#5188
      id: 5,
      name: "Alice",
      date_of_birth: "2001-05-10",
      intake_class: "2023A",
      department_id: 1,
      created_at: "2025-10-20 05:28:22",
      updated_at: "2025-10-20 05:28:22",
    },
    App\Models\Student {#5887
      id: 6,
      name: "Alice",
      date_of_birth: "2001-05-10",
      intake_class: "2023A",
      department_id: 1,
      created_at: "2025-10-20 06:12:13",
      updated_at: "2025-10-20 06:12:13",
    },
  ],
}

> |
```

Database Verification

All CRUD operations reflect live in phpMyAdmin under the **student_system** database and **students** table.

Direct database inspection confirms data integrity and successful API operations.

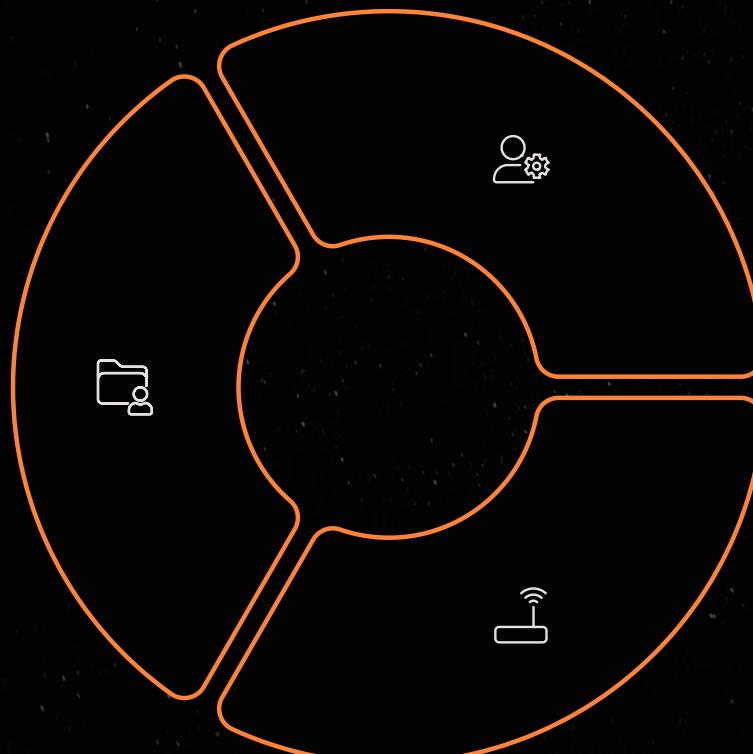
The screenshot shows the phpMyAdmin interface for the 'student_system' database. The 'students' table is selected. The table has columns: id, name, date_of_birth, intake_class, department_id, created_at, and updated_at. There are three rows of data:

	4	John Doe	2000-01-01	2023B	1	2025-10-20 02:47:54	2025-10-20 02:50:24
	5	Alice	2001-05-10	2023A	1	2025-10-20 05:28:22	2025-10-20 05:28:22
	6	Alice	2001-05-10	2023A	1	2025-10-20 06:12:13	2025-10-20 06:12:13

The last two rows (Alice) are highlighted with a red box. The 'id' column contains values 4, 5, and 6. The 'name' column contains 'John Doe' and two entries for 'Alice'. The 'date_of_birth' column contains '2000-01-01', '2001-05-10', and '2001-05-10'. The 'intake_class' column contains '2023B', '2023A', and '2023A'. The 'department_id' column contains '1'. The 'created_at' and 'updated_at' columns both contain the timestamp '2025-10-20 02:47:54', '2025-10-20 05:28:22', and '2025-10-20 06:12:13' respectively.

MVC Architecture in Action

Model
Student model interacts with the database, defining structure and relationships.



Controller

StudentController contains business logic and handles API request processing.

Routes

API routes define endpoints and map HTTP requests to controller methods.

1. Routing

Without Laravel:

```
if ($_SERVER['REQUEST_URI'] == '/hello') {  
    echo "Hello PHP!";  
}
```

With Laravel:

```
Route::get('/hello', function () {  
    return 'Hello Laravel!';  
});
```

👉 Laravel handles routing neatly, no messy conditionals.

2. Database (Eloquent ORM)

Without Laravel:

```
$conn = mysqli_connect('localhost', 'root', '', 'test');  
$result = mysqli_query($conn, "SELECT * FROM users WHERE id=1");  
$user = mysqli_fetch_assoc($result);  
echo $user['name'];
```

With Laravel:

```
$user = User::find(1);  
echo $user->name;
```

👉 No SQL – just object-style database access.



3. Template (View)

Without Laravel:

```
<h1>Hello <?php echo $name; ?></h1>
```

With Laravel (Blade):

```
<h1>Hello, {{ $name }}</h1>
```

👉 Blade is cleaner and prevents XSS automatically.

4. Controller Creation

Without Laravel: Manually create a file and write all logic yourself.

With Laravel:

```
php artisan make:controller PostController
```

👉 One command – ready-to-use controller file.



5. Migration (Database Setup)

Without Laravel: Manually run SQL commands in phpMyAdmin.

With Laravel:

```
php artisan migrate
```

👉 Laravel builds or updates tables automatically.



Deployment (GitHub Prep)

Before uploading your Laravel project to GitHub,
ensure you remove sensitive or unnecessary files:

- vendor/ → too large; can be reinstalled with composer install.
- .env → contains sensitive info (DB passwords, keys).
- storage/logs/ → unnecessary local logs; not needed in repo.



👉 Keep your repository clean and secure by excluding these files.



sabs-27/student-crud-laravel

<https://github.com/sabs-27/student-crud-laravel.git>

