📘 PHP + Laravel: A Simple RESTful API Guide (with Service Layer)

Part 1: Prerequisites

Install these first:

- **PHP 8.1+** – Required for Laravel 10.
- **Composer** – Dependency manager for PHP.
- **MySQL** – Database for student records.
- **IDE (VS Code)** – For writing code.
- **Postman** – For testing APIs.

P.S do not forget to fork the repository

Part 2: Creating the Project

1. **Create a new Laravel project:**

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

Open the project in your IDE.

Project Structure

- **app/Models/ → Models (OOP classes + database entities).**
- **app/Services/ → Service classes (business logic).**
- **app/Http/Controllers/ → Controllers (API endpoints).**
- **database/migrations/ → Defines database tables.**
- **routes/api.php → Defines API routes.**
- **.env → Database credentials.**

Part 3: Database Setup

1. **Create a database in MySQL:**

```sql
sql

CREATE DATABASE studentdb;
```

2. **Update .env:**

```env
env

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

Part 4: Creating the Student Model (Entity + Encapsulation + Inheritance)

Generate a model + migration:

```
php artisan make:model Student -m
```

📌 Notes:

- **Entity** → Student maps to the students table.
- **Encapsulation** → $fillable protects attributes from unwanted updates.
- **Inheritance** → Inherits from Laravel's Model base class.

app/Models/Student.php

```php
namespace App\Models;

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

class Student extends Model   // Inheritance: extends base Model class
{
    use HasFactory;

    // Encapsulation: only these fields can be mass-assigned
    protected $primaryKey = 'pkStudentID';
    protected $fillable = ['name', 'course'];
}
```

Migration (Database Table)

Edit the generated migration in database/migrations/xxxx_create_students_table.php:

```php
public function up(): void
{
    Schema::create('students', function (Blueprint $table) {
        $table->id('pkStudentID'); // Primary Key
        $table->string('name');
        $table->string('course');
        $table->timestamps();
    });
}
```

```
php artisan migrate
```

Part 5: Creating the Service Layer (Abstraction)

Create a folder app/Services/ and add StudentService.php.

app/Services/StudentService.php

```php
namespace App\Services;

use App\Models\Student;

class StudentService
{
    public function getAllStudents()
    {
        return Student::all();
    }

    public function getStudentById($id)
    {
        return Student::find($id);
    }

    public function addStudent($data)
    {
        return Student::create($data);
    }

    public function updateStudent($id, $data)
    {
        $student = Student::find($id);
        if (!$student) return null;

        $student->update($data);
        return $student;
    }

    public function deleteStudent($id)
    {
        $student = Student::find($id);
        if (!$student) return false;

        $student->delete();
        return true;
    }
}
```

📌 Notes:

- **Abstraction** → Service hides database details from controllers.
- Business logic is now centralized here.
- Instead of writing raw SQL, we use **ORM (Object Relational Mapping)**. In Laravel, this is done through **Eloquent**, which provides simple methods that translate into SQL queries behind the scenes. The goal is to make database operations easier, more readable, and secure.

Part 6: Creating the Controller (Polymorphism)
Generate controller:

```
php artisan make:controller StudentController --api
```

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Services\StudentService;

class StudentController extends Controller
{
    protected $studentService;

    // Dependency Injection (like @Autowired in Spring Boot)
    public function __construct(StudentService $studentService)
    {
        $this->studentService = $studentService;
    }

    public function index()
    {
        return response()->json($this->studentService->getAllStudents(), 200);
    }

    public function show($id)
    {
        $student = $this->studentService->getStudentById($id);
        return $student
            ? response()->json($student, 200)
            : response()->json(['message' => 'Student not found'], 404);
    }

    public function store(Request $request)
    {
        $student = $this->studentService->addStudent($request->all());
        return response()->json($student, 201);
    }

    public function update(Request $request, $id)
    {
        $student = $this->studentService->updateStudent($id, $request->all());
        return $student
            ? response()->json($student, 200)
            : response()->json(['message' => 'Student not found'], 404);
    }

    public function destroy($id)
    {
        $deleted = $this->studentService->deleteStudent($id);
        return $deleted
            ? response()->json(['message' => 'Student deleted'], 200)
            : response()->json(['message' => 'Student not found'], 404);
    }
}
```

Part 7: Defining Routes

In routes/api.php:

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

Route::apiResource('students', StudentController::class);
```

Using **Route::apiResource** will automatically generate all the CRUD routes for you. However, you can still define them manually if needed, for example:

```php
Route::get('/students', [StudentController::class, 'index']);
Route::post('/students', [StudentController::class, 'store']);
Route::put('/students/{id}', [StudentController::class, 'update']);
Route::delete('/students/{id}', [StudentController::class, 'destroy']);
```

Part 8: Test in Postman

You do the rest afterwards. Please follow proper Pull Request Title and proper commit message.