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

Part 1: Prerequisites

```
Install these before starting:
- PHP 8.1+ – Needed for Laravel 10
- Composer – PHP dependency manager
- MySQL – Database for storing students
- IDE – VS Code / PhpStorm
- Postman – For testing REST APIs
```

Part 2: Creating the Project

```
1. Create new project:
   composer create-project laravel/laravel student-api

2. Open in your IDE.

Project Structure (important folders):
- app/Models/ → Models (OOP entities linked to database)
- app/Services/ → Business logic (service layer)
- app/Http/Controllers/ → Controllers (API endpoints)
- database/migrations/ → Database table definitions
- routes/api.php → API routes
- .env → Database credentials
```

Part 3: Database Setup

```
1. Create a MySQL database:
   CREATE DATABASE studentdb;

2. Edit .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 + OOP)

```
Generate model + migration:
   php artisan make:model Student -m

app/Models/Student.php
----------------------
namespace App\Models;

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

class Student extends Model   // Inheritance
{
    use HasFactory;

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

Migration (database/migrations/...)
-----------------------------------
Schema::create('students', function (Blueprint $table) {
```

```
    $table->id('pkStudentID');
    $table->string('name');
    $table->string('course');
    $table->timestamps();
});

Run migration:
   php artisan migrate

Notes:
- Entity → Student model maps to students table.
- Encapsulation → $fillable hides unallowed updates.
- Inheritance → Inherits from Laravel’s Model class.
```

Part 5: Creating the Service Layer (Abstraction)

```
app/Services/StudentService.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 → Controller does not touch database directly, it calls service methods.
```

Part 6: Creating the Controller (Polymorphism)

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

app/Http/Controllers/StudentController.php
-----------------------------------------
namespace App\Http\Controllers;

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

class StudentController extends Controller
{
    protected $studentService;

    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);
    }
}

Notes:
- Polymorphism → Same resource (Student) behaves differently depending on HTTP method (GET, POST, PU
```

Part 7: Defining Routes

```
routes/api.php
---------------
use App\Http\Controllers\StudentController;
Route::apiResource('students', StudentController::class);

This auto-generates all CRUD routes.
```

Part 8: Testing in Postman

```
1. Get all students → GET http://127.0.0.1:8000/api/students
2. Add student → POST http://127.0.0.1:8000/api/students
   { "name": "John Doe", "course": "Computer Science" }
3. Get student by ID → GET http://127.0.0.1:8000/api/students/1
4. Update student → PUT http://127.0.0.1:8000/api/students/1
   { "course": "Information Technology" }
5. Delete student → DELETE http://127.0.0.1:8000/api/students/1
```

Part 9: OOP + Entity Mapping in Laravel

```
Encapsulation → $fillable protects attributes from mass assignment.
Abstraction → Service layer hides database details.
Inheritance → Student extends base Model.
Polymorphism → Controller methods respond differently to HTTP verbs.
Entity → Student model maps to students table (DB entity).
```

Final Flow (Layered Architecture)

```
Client (Postman)
   ↓
Controller (StudentController)
   ↓
Service (StudentService)
   ↓
Model/Entity (Student)
   ↓
Database (MySQL)
```