

# Student Course Management System - Technical Report

**Course:** CSC 640

**Project:** HW4 Part 3 - REST API Development with Database Integration & Laravel Migration

**Date:** December 2025

**Author:** Kevin Deras

# Executive Summary

This report covers my implementation of a REST API for managing students, courses, and enrollments using the Laravel framework. The API handles all the basic CRUD operations and includes secure authentication for all endpoints using Laravel Sanctum. I migrated from my vanilla PHP + PDO implementation to Laravel 10 with Eloquent ORM, following RESTful design principles and implementing token-based authentication.

## What I Built:

- 12 working REST API endpoints (including health check and authentication)
- All endpoints secured with Laravel Sanctum token authentication
- MySQL database integration with Laravel migrations and Eloquent ORM
- Clean MVC architecture following Laravel conventions
- Proper validation using Laravel's built-in validation system

# Table of Contents

1. [Introduction](#)
2. [System Architecture](#)
3. [API Endpoints](#)
4. [Security Implementation](#)
5. [Technical Implementation](#)
6. [Database Design](#)
7. [Testing & Validation](#)
8. [Laravel Migration from Vanilla PHP](#)
9. [Challenges & Solutions](#)
10. [Future Enhancements](#)

# 1. Introduction

## 1.1 Project Overview

The Student Course Management System is a REST API I built using Laravel to handle the basics of managing students and courses. The system lets you:

- **Student Management:** Full CRUD - create, read, update, and delete student records
- **Course Management:** Same deal for courses
- **Enrollment Management:** Let students enroll in courses (with proper authorization, of course)
- **Database Integration:** Full MySQL database with Laravel migrations, Eloquent ORM, and proper relationships

This is a complete migration from my vanilla PHP implementation. Moving to Laravel

## 2. System Architecture

### 2.1 Project Structure

```
student-api-laravel/  
├── app/  
│   ├── Http/  
│   │   ├── Controllers/  
│   │   │   ├── AuthController.php  
│   │   │   ├── StudentController.php  
│   │   │   ├── CourseController.php  
│   │   │   └── EnrollmentController.php  
│   │   └── Middleware/  
│   └── Models/  
│       ├── Student.php  
│       ├── Course.php  
│       ├── Enrollment.php  
│       └── User.php  
├── database/  
│   └── migrations/  
│       ├── create_students_table.php  
│       ├── create_courses_table.php  
│       └── create_enrollments_table.php  
├── routes/  
│   └── api.php          # All API routes defined here  
└── config/  
    ├── auth.php  
    ├── sanctum.php  
    └── cors.php
```

## 3. API Endpoints

### 3.1 Overview

The API provides 12 endpoints across authentication and three resource types plus a health check:

Resource	Total Endpoints	Secure Endpoints
----------	-----------------	------------------

-----	-----	-----
-------	-------	-------

Health	1	0
--------	---	---

Authentication	2	0 (login), 1 (logout)
----------------	---	-----------------------

Students	5	5 (all require auth)
----------	---	----------------------

Courses	5	5 (all require auth)
---------	---	----------------------

Enrollments	3	3 (all require auth)
-------------	---	----------------------

## 3.5 Course Endpoints

### GET /api/courses

**Purpose:** Get all courses

**Authentication:** Bearer Token Required

**Response:** Array of course objects

```
[
  {
    "id": 1,
    "code": "CSC640",
    "title": "Software Engineering",
    "created_at": "2025-12-04T10:30:00.000000Z",
    "updated_at": "2025-12-04T10:30:00.000000Z"
  },
  {
    "id": 2,
    "code": "CSC601",
    "title": "Algorithms"
```

## 3.6 Enrollment Endpoints

### GET /api/enrollments

**Purpose:** Retrieve all enrollments with student and course details

**Authentication:** Bearer Token Required

**Response:** Array of enrollment objects with relationships

```
[
  {
    "id": 1,
    "student_id": 1,
    "course_id": 1,
    "created_at": "2025-12-04T10:35:00.000000Z",
    "updated_at": "2025-12-04T10:35:00.000000Z",
    "student": {
      "id": 1,
      "name": "Kevin",
      "email": "kevin@example.com"
    },
  },
]
```

## 4. Security Implementation

### 4.1 Authentication Strategy

I implemented **Laravel Sanctum** for token-based authentication. This is a huge upgrade from the simple Bearer token in the vanilla PHP version.

#### Key Improvements:

- **Token Management:** Tokens are stored in the database and can be revoked
- **User-Based:** Each token is tied to a specific user account
- **Secure by Default:** Laravel handles token generation, validation, and expiration
- **Logout Support:** Tokens can be invalidated on logout

#### All Endpoints Protected:

Unlike the vanilla PHP version where only 5 endpoints were protected, ALL endpoints

# 5. Technical Implementation

## 5.1 Controllers

Controllers handle HTTP requests and return responses. Much cleaner than the routing logic in `index.php` from the vanilla PHP version.

### StudentController Example:

```
<?php
namespace App\Http\Controllers;

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

class StudentController extends Controller
{
    public function index()
    {
        return response()->json(Student::all());
    }

    public function store(Request $request)
    {
        $data = $request->validate([
            'name' => 'required|string',
            'email' => 'required|email|unique:students,email',
        ]);

        $student = Student::create($data);
        return response()->json($student, 201);
    }

    public function show($id)
    {
        $student = Student::findOrFail($id);
        return response()->json($student);
    }
}
```

## 6. Database Design

### 6.1 Schema Overview

The database consists of three main tables with proper relationships, defined using Laravel migrations:

#### Students Table:

```
Schema::create('students', function (Blueprint $table) {  
    $table->id();  
    $table->string('name');  
    $table->string('email')->unique();  
    $table->timestamps();  
});
```

#### Courses Table:

```
Schema::create('courses', function (Blueprint $table) {  
    $table->id();
```

# 7. Testing & Validation

## 7.1 Testing Strategy

### Primary Testing Tool: Postman

I used Postman extensively to test all endpoints, just like in the vanilla PHP version. The workflow is similar, but now we need to authenticate first.

### Setting Up Postman:

#### 1. Create a New Collection:

- Click "New" → "Collection"
- Name it "Student API Laravel"
- Click "Create"

#### 2. Set Up Environment Variables:

## 8. Laravel Migration from Vanilla PHP

### 8.1 Why Migrate to Laravel?

The vanilla PHP implementation worked, but it had limitations:

#### Issues with Vanilla PHP:

- Manual routing with regex patterns
- Manual PDO connection management
- Manual validation logic
- Manual error handling
- Manual JSON serialization
- Hardcoded authentication token
- No built-in testing framework

## 9. Challenges & Solutions

### 9.1 Challenge: Understanding Laravel Sanctum

**Problem:** Initially, I was confused about how Sanctum tokens work. The vanilla PHP version used a simple hardcoded Bearer token, but Sanctum is more complex.

**Solution:** Read Laravel documentation and realized that:

- Tokens are stored in the database
- Each token is tied to a user
- Tokens can be revoked (logout)
- Middleware handles validation automatically

Much more secure than the hardcoded token approach!

### 9.2 Challenge: All Endpoints Requiring Auth

# 10. Future Enhancements

## 10.1 Security Improvements

### Enhanced Authentication:

- Implement refresh tokens for long-lived sessions
- Add rate limiting to prevent abuse
- Implement role-based access control (RBAC)
- Add IP whitelisting for sensitive operations
- Implement API key authentication for service-to-service communication

### Input Sanitization:

- Add HTML entity encoding for XSS protection
- Implement file upload validation (if file uploads are added)

## 5.x Deployment Scripts (run.sh and setup.sh)

For Stage 2 I added two shell scripts to make deployment repeatable:







- `run.sh` – starts the Laravel Sail environment (or plain PHP server), runs migrations, and launches the API. This script is used for the “Deploy with a shell script” part of the assignment.
- `setup.sh` – builds and runs the Docker containers (Laravel app + MySQL) using `docker compose`. This script is used for the “Deploy with Docker” grading section.

# 11. Conclusion

## 11.1 Mission Accomplished

I successfully migrated my vanilla PHP REST API to Laravel with Eloquent ORM. All 12 endpoints work correctly, and authentication is handled securely with Laravel Sanctum.





### What I Got Done:

-  Migrated from vanilla PHP to Laravel 10
-  Replaced PDO with Eloquent ORM
-  Implemented Laravel Sanctum for secure authentication
-  All 12 REST API endpoints working
-  All endpoints secured with token authentication (except health check and login)
-  MySQL database integration with Laravel migrations

# Appendices

## Appendix A: Complete API Reference

Base URL: `http://localhost/api`

Method	Endpoint	Auth	Description
-----	-----	-----	-----
GET	/api/status	-	Health check
POST	/api/login	-	Authenticate and get token
POST	/api/logout		Invalidate current token
GET	/api/students		List all students
GET	/api/students/{id}		Get student by ID
POST	/api/students		Create student

## End of Report

*This document represents the completed migration of the Student Course Management System REST API from vanilla PHP to Laravel 10 with Eloquent ORM and Laravel Sanctum authentication for CSC 640.*