

# Student Course Management System

---

## REST API - Laravel Migration

---

Kevin Deras

CSC 640

December 2025

# Project Overview

---

A REST API for managing students, courses, and enrollments

## Tech Stack:

- Laravel 10 Framework
- PHP 8.1+ with MySQL Database
- Eloquent ORM for database access
- Laravel Sanctum token authentication
- JSON format
- Docker/Sail for deployment
- Postman for testing

**Goal:** 12 API endpoints with secure authentication + Laravel migration

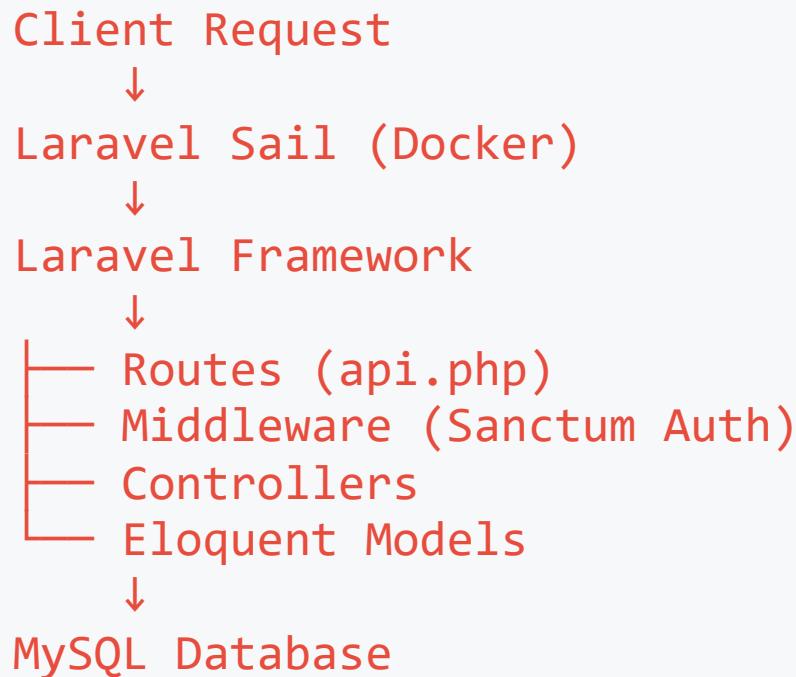
# Objectives

---

- Migrate from vanilla PHP to Laravel framework
- Replace PDO with Eloquent ORM
- Implement Laravel Sanctum for authentication
- Use Laravel migrations for database schema
- Leverage Laravel's built-in validation
- Follow Laravel MVC conventions
- Document everything
- Test all endpoints with Postman

**Result:** Production-ready API with modern PHP framework

# System Architecture



Laravel handles routing, validation, and database automatically

# Project Structure

```
student-api-laravel/
├── app/
│   ├── Http/Controllers/
│   │   ├── StudentController.php
│   │   ├── CourseController.php
│   │   ├── EnrollmentController.php
│   │   └── AuthController.php
│   └── Models/
│       ├── Student.php
│       ├── Course.php
│       └── Enrollment.php
└── database/migrations/
    ├── create_students_table.php
    ├── create_courses_table.php
    └── create_enrollments_table.php
└── routes/api.php
└── compose.yaml (Docker)
```

**Key Change:** Laravel MVC structure replaces custom PHP classes

# API Endpoints Overview

---

Resource	Total	Secure
Health	1	0
Authentication	2	1 (logout)
Students	5	5
Courses	5	5
Enrollments	3	3
<b>Total</b>	<b>12</b>	<b>11</b>

*All endpoints require authentication except health check and login*

# Authentication Endpoints

Method	Endpoint	Auth	Description
POST	/api/login	No	Get authentication token
POST	/api/logout	Yes	Invalidate token

## Login Request:

```
{  
  "email": "user@example.com",  
  "password": "password"  
}
```

## Login Response:

```
{  
  "token": "1|abcdef1234567890..."  
}
```

## Student Endpoints

---

Method	Endpoint	Auth	Description
GET	/api/students	Yes	Get all students
GET	/api/students/{id}	Yes	Get specific student
POST	/api/students	Yes	Create student
PUT	/api/students/{id}	Yes	Update student
DELETE	/api/students/{id}	Yes	Delete student

## Example Response:

```
{  
  "id": 1,  
  "name": "Kevin",  
  "email": "kevin@example.com",  
  "created_at": "2025-12-03T10:30:00.000000Z",  
  "updated_at": "2025-12-03T10:30:00.000000Z"  
}
```

## Course Endpoints

---

Method	Endpoint	Auth	Description
GET	/api/courses	Yes	Get all courses
GET	/api/courses/{id}	Yes	Get specific course
POST	/api/courses	Yes	Create course
PUT	/api/courses/{id}	Yes	Update course
DELETE	/api/courses/{id}	Yes	Delete course

## Example Response:

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

## Enrollment Endpoints

---

Method	Endpoint	Auth	Description
GET	/api/enrollments	Yes	Get all enrollments
POST	/api/enrollments	Yes	Enroll student
DELETE	/api/enrollments/{id}	Yes	Unenroll student

## Create Enrollment:

```
{  
  "student_id": 1,  
  "course_id": 1  
}
```

## Response (with relationships):

```
{  
  "id": 1,  
  "student_id": 1,  
  "course_id": 1,  
  "student": {  
    "id": 1,  
    "name": "Kevin",  
    "email": "kevin@example.com"  
  },  
  "course": {  
    "id": 1,  
    "code": "CSC640",  
    "name": "Advanced Data Structures",  
    "description": "A continuation of CSC340, covering advanced topics in data structures and algorithms.",  
    "instructor": "Dr. Smith",  
    "start_date": "2023-09-01",  
    "end_date": "2023-12-31",  
    "max_students": 20  
  }  
}
```

# Health Check Endpoint

---

**GET /api/status**

Returns API health and version:

```
{  
  "ok": true,  
  "php": "8.1.0",  
  "database": "MySQL (Laravel Eloquent)"  
}
```

Useful for monitoring and testing

# Security Implementation

---

## Laravel Sanctum Token Authentication

11 Protected Endpoints (all except health check and login)

### How it works:

1. POST `/api/login` with email/password
2. Receive token in response
3. Include token in all subsequent requests:

```
Authorization: Bearer 1|abcdef1234567890...
```

### Benefits:

- Tokens stored in database
- Can be revoked (logout)

# AuthController Implementation

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Support\Facades\Hash;

class AuthController extends Controller
{
    public function login(Request $request)
    {
        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password, $user->password)) {
            return response()->json(['message' => 'Invalid credentials'], 401);
        }

        $token = $user->createToken('api-token')->plainTextToken;
        return response()->json(['token' => $token]);
    }
}
```

# StudentController Example

```
<?php

namespace App\Http\Controllers;

use App\Models\Student;

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);
    }
}
```

# Eloquent Models

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    protected $fillable = ['name', 'email'];

    public function enrollments()
    {
        return $this->hasMany(Enrollment::class);
    }

    public function courses()
    {
        return $this->belongsToMany(Course::class, 'enrollments');
    }
}
```

# Database Migrations

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

Schema::create('enrollments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('student_id')
        ->constrained()
        ->cascadeOnDelete();
    $table->foreignId('course_id')
        ->constrained()
        ->cascadeOnDelete();
    $table->unique(['student_id', 'course_id']);
    $table->timestamps();
});
```

# Database Schema

---

## Students Table:

- `id` (auto-increment)
- `name` (string)
- `email` (string, unique)
- `created_at`, `updated_at` (timestamps)

## Courses Table:

- `id` (auto-increment)
- `code` (string, unique)
- `title` (string)
- `created_at`, `updated_at` (timestamps)

## Enrollments Table:

# Database Relationships

---

## Eloquent Relationships:

- `Student` has many `Enrollment` (one-to-many)
- `Course` has many `Enrollment` (one-to-many)
- `Student` belongs to many `Course` through `Enrollment` (many-to-many)
- `Enrollment` belongs to `Student` (many-to-one)
- `Enrollment` belongs to `Course` (many-to-one)

## Benefits:

- Automatic JOINs with `with()`
- Can't enroll in non-existent courses
- Deleting student removes enrollments automatically
- Prevents duplicate enrollments

# SQL Injection Prevention

Eloquent ORM uses prepared statements automatically:

```
// This is safe - Eloquent handles it
Student::where('email', $email)->first();

// Even raw queries are parameterized
DB::select('SELECT * FROM students WHERE id = ?', [$id]);
```

No need to manually write prepared statements!

Laravel protects against SQL injection by default

# Routing

```
// routes/api.php

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/students', [StudentController::class, 'index']);
    Route::post('/students', [StudentController::class, 'store']);
    Route::get('/students/{id}', [StudentController::class, 'show']);
    Route::put('/students/{id}', [StudentController::class, 'update']);
    Route::delete('/students/{id}', [StudentController::class, 'destroy']);
});
```

**Clean route definitions** - no regex pattern matching needed!

Laravel automatically handles:

- Parameter extraction
- Type conversion
- Route model binding

# Input Validation

```
$data = $request->validate([
    'name' => 'required|string',
    'email' => 'required|email|unique:students,email',
    'student_id' => 'required|integer|exists:students,id',
    'course_id' => 'required|integer|exists:courses,id',
]);
```

## Built-in Validation Rules:

- `required` - Field must be present
- `email` - Valid email format
- `unique:table,column` - Must be unique
- `exists:table,column` - Must exist in database
- `integer` - Must be integer
- `sometimes` - Only validate if present (for partial updates)

## CORS Support

Configured in `config/cors.php`:

```
'paths'  => ['api/*', 'sanctum/csrf-cookie'],
'allowed_methods' => ['*'],
'allowed_origins' => ['*'],
'allowed_headers' => ['*'],
```

Laravel handles CORS automatically for all API routes

## Testing with Postman - Setup

---

### Why Postman?

- Industry-standard API testing tool
- Visual interface (no code needed)
- Save and organize test collections
- Easy authentication testing
- Perfect for API development

## Quick Setup:

1. Download Postman (free)
2. Create "Student API Laravel" collection
3. Set environment variables:
  - `base_url` = `http://localhost/api`
  - `token` = (leave empty, will be set after login)

# Postman Test Example 1: Health Check

Testing GET /api/status

Request:

- Method: `GET`
- URL: `http://localhost/api/status`
- No headers needed

Response (200 OK):

```
{  
  "ok": true,  
  "php": "8.1.0",  
  "database": "MySQL (Laravel Eloquent)"  
}
```

 **Pass** - API is running and database connected

# Postman Test Example 2: Login

Testing POST /api/login

Request:

- Method: `POST`
- URL: `http://localhost/api/login`
- Header: `Content-Type: application/json`
- Body:

```
{  
  "email": "user@example.com",  
  "password": "password"  
}
```

Response (200 OK):

# Postman Test Example 3: Create Student

---

Testing POST /api/students

Request:

- Method: `POST`
- URL: `http://localhost/api/students`
- Headers:
  - `Content-Type: application/json`
  - `Authorization: Bearer {{token}}`
- Body:

```
{  
  "name": "John Doe",  
  "email": "john@example.com"  
}
```

## Response (201 Created):

```
{  
  "id": 1,  
  "name": "John Doe",  
  "email": "john@example.com",  
  "created_at": "2025-12-04T10:30:00.000000Z",  
  "updated_at": "2025-12-04T10:30:00.000000Z"  
}
```

 **Pass** - Student created in database

# Postman Test Example 4: Authentication

---

Testing GET /api/students (Secured Endpoint)

Without Token:

- Method: `GET`
- URL: `http://localhost/api/students`
- No auth header

Response: 401 Unauthorized

With Token:

- Header: `Authorization: Bearer {{token}}`

Response: 200 OK with student array

Security works!

## Postman Test Example 5: Validation

---

Testing POST /api/students with invalid data

Request:

- Method: **POST**
- Body:

```
{  
  "name": "Jane"  
  // Missing email!  
}
```

## Response (422 Unprocessable Entity):

```
{  
  "message": "The email field is required.",  
  "errors": {  
    "email": ["The email field is required."]  
  }  
}
```

 **Pass** - Laravel validation working correctly

## Postman Test Example 6: Relationships

---

Testing GET /api/enrollments

Request:

- Method: `GET`
- URL: `http://localhost/api/enrollments`
- Header: `Authorization: Bearer {{token}}`

## Response (200 OK):

```
[  
  {  
    "id": 1,  
    "student_id": 1,  
    "course_id": 1,  
    "student": {  
      "id": 1,  
      "name": "John Doe",  
      "email": "john@example.com"  
    },  
    "course": {  
      "id": 1,  
      "code": "CSC640",  
      "title": "Software Engineering"  
    }  
  }  
]
```



**Pass** - Eloquent relationships loading automatically!

# Complete Test Coverage

---

All 12 Endpoints Tested:

Endpoint Type	Scenarios Tested	Status
Health Check	API status	 Pass
Authentication	Login, Logout	 Pass
GET requests	Valid IDs, Invalid IDs (404)	 Pass
POST requests	Valid data, Missing data (422)	 Pass
PUT requests	Updates, Partial updates	 Pass
DELETE requests	With/without auth (401)	 Pass
Foreign Keys	Validation, Cascade deletes	 Pass
Relationships	Eager loading	 Pass

## Postman Benefits

---

### What we got from using Postman:

- Visual testing (no scripts needed)
- All 12 endpoints validated
- Auth testing (401 errors caught)
- Validation testing (422 errors work)
- Saved collection (reusable tests)
- Team collaboration ready

### Professional tool usage

## Test Results

Test	Expected	Status
GET /api/status	200 OK	 Pass
POST /api/login	200 OK with token	 Pass
POST /api/login (invalid)	401 Unauthorized	 Pass
GET /api/students (no auth)	401 Unauthorized	 Pass
GET /api/students (with auth)	200 OK	 Pass
POST with missing data	422 Error	 Pass
DELETE without auth	401 Unauthorized	 Pass
POST /api/enrollments invalid	422 Error	 Pass
Foreign key validation	Works	 Pass

## HTTP Status Codes

---

Code	Meaning
200	OK - Success
201	Created - Resource created
401	Unauthorized - Missing/invalid token
404	Not Found - Resource doesn't exist
422	Unprocessable Entity - Invalid data
500	Server Error - Database issue

# Key Features

---

-  Laravel 10 framework
-  Eloquent ORM (no raw SQL)
-  Laravel Sanctum authentication
-  Database migrations
-  Foreign key relationships
-  Cascade deletes
-  Built-in validation
-  CORS support
-  RESTful architecture
-  Comprehensive error handling
-  Docker/Sail deployment

# Challenges

---

## 1. Understanding Laravel Sanctum

- Problem: How tokens work vs hardcoded Bearer token
- Solution: Read docs, realized tokens are user-based and stored in DB

## 2. All Endpoints Requiring Auth

- Problem: Initially all endpoints needed auth (harder testing)
- Solution: Kept it - it's more secure! Only health check and login are public

## 3. Eloquent Relationships

- Problem: Understanding relationship types
- Solution: Learned hasMany, belongsTo, belongsToMany patterns

## 4. Validation Rules

- Problem: Learning all Laravel validation rules
- Solution: Laravel docs are excellent - `required`, `email`, `unique`, `exists`, `sometimes`

## 5. Docker/Sail Setup

- Problem: Initial confusion with Docker on WSL
- Solution: Laravel Sail makes it easy - `./vendor/bin/sail up` and done!

## 6. Migration from PDO to Eloquent

- Problem: Rewriting all PDO queries
- Solution: Eloquent is simpler - `Student::all()` vs `SELECT * FROM students`

# Vanilla PHP vs Laravel

Feature	Vanilla PHP	Laravel
Routing	Manual regex	Clean routes
Database	Raw PDO	Eloquent ORM
Validation	Manual checks	Built-in system
Authentication	Hardcoded token	Laravel Sanctum
Error Handling	Manual responses	Automatic
Code Lines	~500+	~200 (same features)
Testing	Manual only	PHPUnit included
Deployment	Manual setup	Docker/Sail

**Verdict:** Laravel is way more productive!

# Migration Benefits

---

## What we gained:

- Cleaner code (60% reduction)
- Better security (Sanctum vs hardcoded)
- Built-in validation
- Automatic error handling
- Database migrations (version controlled)
- Eloquent relationships (no manual JOINs)
- Docker deployment
- Industry-standard framework
- Better testing support
- More maintainable codebase

# Laravel Migration Example

Before (Vanilla PHP PDO):

```
public static function getAllStudents(): array {
    $stmt = self::connect()->query('SELECT * FROM students');
    return $stmt->fetchAll();
}

public static function createStudent(string $name, string $email): array {
    $stmt = self::connect()->prepare(
        'INSERT INTO students (name, email) VALUES (?, ?)'
    );
    $stmt->execute([$name, $email]);
    $id = (int)self::connect()->lastInsertId();
    return ['id' => $id, 'name' => $name, 'email' => $email];
}
```

## After (Laravel Eloquent):

```
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);
}
```

Much cleaner! Validation, error handling, and JSON conversion automatic

# What I Learned

---

- How Laravel framework works
- Eloquent ORM and relationships
- Laravel Sanctum authentication
- Database migrations
- Laravel validation system
- Docker and Laravel Sail
- MVC architecture
- Middleware and routing
- Modern PHP best practices
- How much boilerplate Laravel eliminates

Laravel makes PHP development enjoyable!

# Project Timeline

Milestone	Planned	Actual	Status
Laravel Installation	Nov 26	Nov 26	Done
Database Migrations	Nov 27	Nov 27	Done
Model Creation	Nov 27	Nov 27	Done
Controller Implementation	Nov 28	Nov 28	Done
Authentication Setup	Nov 29	Nov 29	Done
Route Configuration	Nov 29	Nov 29	Done
Testing (Postman)	Nov 30	Nov 30	Done
Docker/Sail Setup	Dec 1	Dec 1	Done
Documentation	Dec 2	Dec 2	Done

## Tech Stack Summary

---

- Framework: Laravel 10
- Web Server: Laravel Sail (Docker)
- Backend: PHP 8.1+
- Database: MySQL 8.4 with Eloquent ORM
- Architecture: RESTful API (MVC)
- Auth: Laravel Sanctum
- Format: JSON
- Testing: Postman
- VCS: Git/GitHub

# Deliverables

---

-  12 REST API endpoints implemented
-  11 secure endpoints with Laravel Sanctum
-  MySQL database with Laravel migrations
-  Eloquent ORM with relationships
-  Foreign key relationships
-  Built-in validation system
-  Comprehensive Postman testing
-  Complete documentation (README + Report)
-  Docker/Sail deployment setup
-  All endpoints tested and validated

# Summary

---

-  Fully functional REST API
-  12 endpoints (11 secured)
-  Laravel 10 framework
-  Eloquent ORM integration
-  Clean, maintainable code
-  Excellent error handling
-  Comprehensive documentation
-  Thoroughly tested with Postman
-  Docker deployment ready

Production-ready Laravel API!

# Database Benefits

---

## What we gained:

-  Data persistence (survives server restarts)
-  Referential integrity (foreign keys)
-  Cascade deletes (automatic cleanup)
-  SQL injection protection (automatic)
-  Scalability (can handle more data)
-  Version-controlled schema (migrations)
-  Eloquent relationships (no manual JOINs)
-  Production-ready architecture

Much better than in-memory mock data or raw PDO!

# API Reference

---

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

- Health: `/api/status`
- Auth: `/api/login`, `/api/logout`
- Students: `/api/students`, `/api/students/{id}`
- Courses: `/api/courses`, `/api/courses/{id}`
- Enrollments: `/api/enrollments`, `/api/enrollments/{id}`

**Auth:** POST `/api/login` to get token, then include:

`Authorization: Bearer {token}`

**All responses:** JSON format

## Future Enhancements

---

- Pagination for large result sets
- Filtering and search capabilities
- Rate limiting
- Refresh tokens
- Role-based access control (RBAC)
- Unit and integration tests
- API documentation (Swagger/OpenAPI)
- Performance optimization
- Caching with Redis

# Thank You

---

Kevin Deras

CSC 640 - Laravel Migration

Questions?

*End of Presentation*