

SCORM Model for Murunani

Table of contents

Overview	2
Prompt 1	3
1. SCORM Communication & Data Model.....	3
Core SCORM API Functions.....	3
Essential CMI Data Elements (SCORM 1.2 Example).....	4
2. Laravel Implementation Steps.....	4
A. Database Migration (CMI Data Storage).....	4
B. Define API Routes.....	5
C. Create the API Controller.....	6
Prompt 2	8
🔗 SCORM API Adapter JavaScript.....	8
1. The Core Adapter Script (scorm_api_adapter.js).....	9
2. SCORM Wrapper HTML Example.....	12
3. Key Points & Next Steps.....	13

Overview

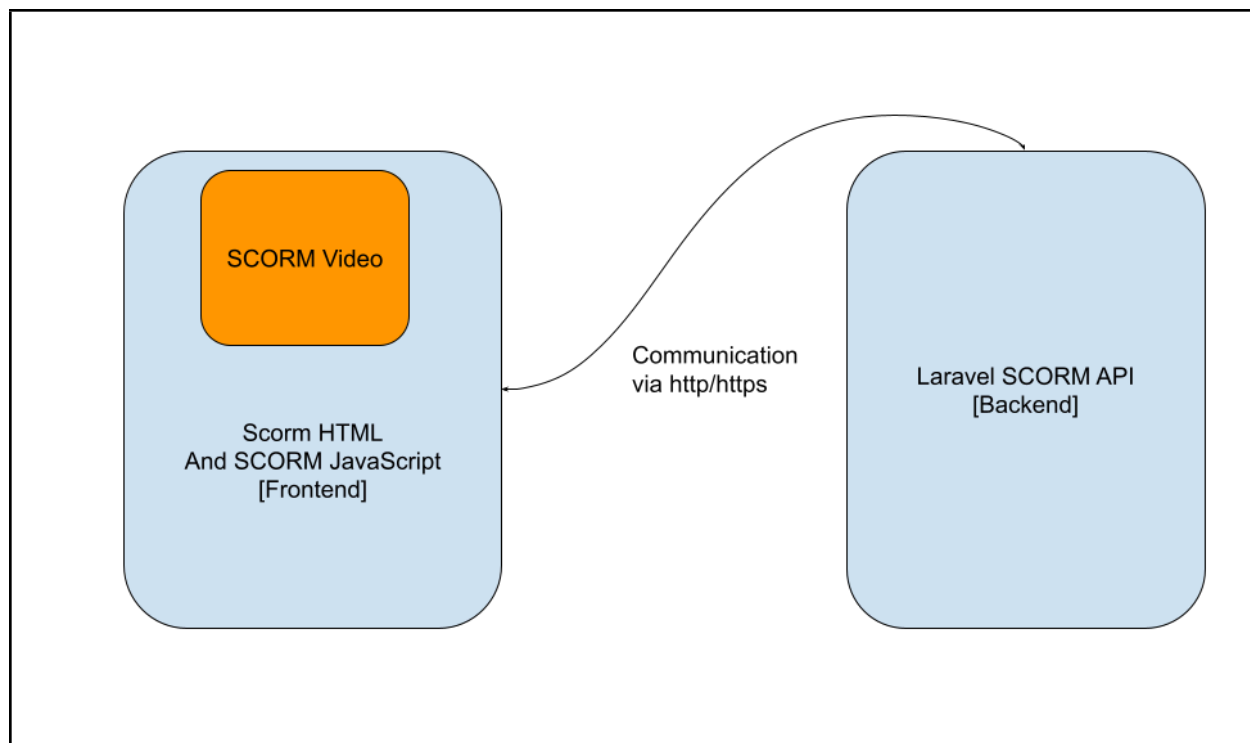


Figure 01 SCORM Model

The scrome model requires three stages as follows.

- a. **SCORM Video:** This stage is already handled by Camtasia for us. This produces a video with JavaScript that needs to be wrapped in HTML, and middle-tier JavaScript that then sends data to an API on the backend.
- b. **SCROM HTML + SCORM JavaScript [middle-tier]:** This points to the backend API URL and defines JavaScript functions that collect data from the SCROM video and send it to the API.
- c. **Laravel SCORM API:** The backend API that now receives, stores and retrieves data from the database. This is written in PHP Laravel.

Below are sample prompts and solutions that I got from Gemini that can further elaborate on what I need.

Prompt 1

Prompt

Create a scorm API based on Laravel. The API should listen for data from vedios being played and other assets that are scorm compliant.

The API stores the details in a database.

Gemini answer

Creating a full **SCORM Run-Time API** from scratch in Laravel to listen for and record data from SCORM-compliant assets is a complex project, as it requires implementing the specific **SCORM API (Application Programming Interface)** functions (like `LMSInitialize`, `LMSSetValue`, `LMSCommit`, etc.) that SCORM content expects to find and communicate with via **JavaScript** in the client's browser.

A more practical and recommended approach for a Laravel application is to use a dedicated **Laravel SCORM package**, or at least use the SCORM specification as a blueprint for your custom API endpoints and database structure.

Here is a conceptual breakdown and a minimal implementation guide for a custom Laravel API to handle SCORM communication, focusing on the data persistence aspect.

1. SCORM Communication & Data Model

SCORM content (e.g., a video player or quiz) runs in an `<iframe>` or a new window and uses a JavaScript object (usually named `API` for SCORM 1.2 or `API_1484_11` for SCORM 2004) to communicate with the Learning Management System (LMS), which your Laravel API is acting as.

Core SCORM API Functions

Your Laravel API needs to handle requests corresponding to these core functions:

- `LMSInitialize()` / `Initialize()`: Starts a communication session.
- `LMSFinish()` / `Terminate()`: Ends the communication session.
- `LMSGetValue(cmi_element)` / `GetValue(cmi_element)`: Retrieves data from the LMS (e.g., learner name, previous status).
- `LMSSetValue(cmi_element, value)` / `SetValue(cmi_element, value)`: Sends data to the LMS (e.g., progress, score, lesson status).
- `LMSCommit()` / `Commit()`: Explicitly saves the changes made by `LMSSetValue()`.

Essential CMI Data Elements (SCORM 1.2 Example)

The data you'll store in the database corresponds to the **CMI (Course/Learner Management Interchange)** data model. Key elements include:

CMI Element	Purpose
cmi.core.student_id	Unique ID of the learner.
cmi.core.lesson_status	Completion status (completed, passed, failed, incomplete).
cmi.core.score.raw	Learner's score.
cmi.core.total_time	Total time spent on the content.
cmi.suspend_data	String for bookmarking/resuming the learner's state.

2. Laravel Implementation Steps

A. Database Migration (CMI Data Storage)

You need a table to store the SCORM progress for each user/content combination.

Bash

```
php artisan make:migration create_scorm_tracks_table
```

```
database/migrations/..._create_scorm_tracks_table.php
```

PHP

```
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;
```

```

use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('scorm_tracks', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->foreignId('content_id')->constrained('scorm_content')->onDelete('cascade'); //
Assuming a content table
            $table->string('lesson_status', 50)->default('not attempted');
            $table->float('score_raw')->nullable();
            $table->string('total_time', 20)->nullable();
            $table->longText('suspend_data')->nullable(); // For bookmarking
            $table->json('cmi_data')->nullable(); // To store all other CMI elements
            $table->timestamps();

            $table->unique(['user_id', 'content_id']);
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('scorm_tracks');
    }
};

```

B. Define API Routes

SCORM content communicates via JavaScript, often making **HTTP POST/GET requests** to the LMS wrapper functions. Your routes will listen for these requests. The SCORM content must be configured to point its communication functions to these URLs.

routes/api.php

PHP

```

use App\Http\Controllers\ScormApiController;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:sanctum')->group(function () {
    // Endpoints SCORM content will call (e.g., for a course with ID: 123)
    Route::post('/scorm/123/initialize', [ScormApiController::class, 'initialize']);
    Route::post('/scorm/123/setvalue', [ScormApiController::class, 'setValue']);
    Route::post('/scorm/123/commit', [ScormApiController::class, 'commit']);
    Route::get('/scorm/123/getvalue', [ScormApiController::class, 'getValue']);
    Route::post('/scorm/123/finish', [ScormApiController::class, 'finish']);
});

```

C. Create the API Controller

The controller handles the incoming SCORM function calls, validates the data, and updates the database.

app/Http/Controllers/ScormApiController.php (Minimal Example)

PHP

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\ScormTrack; // Assuming you create this model

class ScormApiController extends Controller
{
    // Utility to get the current track record
    protected function getTrack(int $contentId)
    {
        return ScormTrack::firstOrCreate(
            ['user_id' => auth()->id(), 'content_id' => $contentId],
            ['lesson_status' => 'not attempted']
        );
    }

    public function initialize(Request $request, int $contentId)
    {
        // 1. Get/Create the tracking record
        $track = $this->getTrack($contentId);

        // 2. SCORM Content expects "true" or "false" string response
        return response('true', 200)->header('Content-Type', 'text/plain');
    }

    public function getValue(Request $request, int $contentId)
    {
        // Example: SCORM content asks for 'cmi.core.lesson_status'
        $element = $request->input('element'); // The CMI element SCORM wants
        $track = $this->getTrack($contentId);

        $value = match ($element) {
            'cmi.core.student_id' => (string)auth()->id(),
            'cmi.core.lesson_status' => $track->lesson_status,
            'cmi.suspend_data' => $track->suspend_data ?? "",
            // Add other CMI elements as needed
            default => "", // Return empty string if not found
        };
    }
}
```

```

    return response($value, 200)->header('Content-Type', 'text/plain');
}

public function setValue(Request $request, int $contentId)
{
    $element = $request->input('element');
    $value = $request->input('value');
    $track = $this->getTrack($contentId);
    $updateData = [];

    // Map the SCORM CMI element to your database fields
    switch ($element) {
        case 'cmi.core.lesson_status':
            $updateData['lesson_status'] = $value;
            break;
        case 'cmi.core.score.raw':
            $updateData['score_raw'] = (float)$value;
            break;
        case 'cmi.suspend_data':
            $updateData['suspend_data'] = $value;
            break;
        // You will need many more case statements for a full implementation

        default:
            // Store all unhandled CMI data in the JSON column
            $cmiData = $track->cmi_data ?? [];
            $cmiData[$element] = $value;
            $updateData['cmi_data'] = $cmiData;
            break;
    }

    $track->update($updateData);

    // SCORM Content expects "true" or "false" string response
    return response("true", 200)->header('Content-Type', 'text/plain');
}

public function commit(Request $request, int $contentId)
{
    // In this simple setup, 'setValue' already updates the DB,
    // so 'commit' just confirms success.
    return response("true", 200)->header('Content-Type', 'text/plain');
}

public function finish(Request $request, int $contentId)
{
    // Logic to finalize the session, perhaps save final timestamps.
    // SCORM Content expects "true" or "false" string response
    return response("true", 200)->header('Content-Type', 'text/plain');
}
}

```

Note on SCORM Player: The content's communication is triggered by a JavaScript adapter embedded in the content itself. For this to work, you need a piece of JavaScript code that acts as the **API adapter** on the client side, which is injected into the HTML page that hosts the SCORM content (usually in an `<iframe>`). This adapter must be what intercepts the standard SCORM calls (`API.LMSSetValue()`) and translates them into **HTTP requests** to your Laravel API endpoints (e.g., `/api/scorm/123/setvalue`). Building this client-side adapter is the other crucial half of the SCORM LMS implementation.

Would you like me to elaborate on the **client-side JavaScript API adapter** necessary to bridge the SCORM content to these Laravel API endpoints?

Prompt 2

Prompt

create A JAVASCRIPT CODE THAT COMMUNICATES WITH THE DEVELOPED API AND THIS JS SHOULD BE ABLE TO wrap around a scorm formatted video

Gemini Answer

That's the most critical piece! The JavaScript API adapter acts as the glue between the SCORM content (your video player and its SCORM wrapper) and your Laravel API endpoints.

This script must be included on the HTML page that hosts the SCORM content (usually in the parent window or frame). It creates the standard SCORM communication object (`API` or `API_1484_11`) and handles sending/receiving data via asynchronous HTTP requests (using `fetch` or `XMLHttpRequest`) to your Laravel routes.

SCORM API Adapter JavaScript

Below is a foundational JavaScript implementation for SCORM 1.2. You'll need to define the `SCORM_API_URL` and ensure the user is authenticated (e.g., by including a CSRF token and the content ID).

1. The Core Adapter Script (scorm_api_adapter.js)

JavaScript

```
/**
 * SCORM 1.2 API Adapter for Laravel LMS
 * This script must be included in the parent window/frame hosting the SCORM content.
 */

// --- Configuration ---
// Base URL for your Laravel API SCORM endpoints
// This should be dynamically set based on the current content/user session.
const CONTENT_ID = 123; // Replace with the actual ID of the SCORM content
const SCORM_API_BASE_URL = `/api/scorm/${CONTENT_ID}/`;

// Placeholder for the tracking data storage
let CMI_DATA = {};
let IS_INITIALIZED = false;

// --- Utility Functions ---

/**
 * Sends a POST request to the Laravel API endpoint.
 * @param {string} endpoint - The route segment (e.g., 'initialize', 'setvalue').
 * @param {object} data - Payload to send to the API.
 * @returns {Promise<string>} The API response text ('true', 'false', or value).
 */
async function apiPost(endpoint, data = {}) {
  const url = SCORM_API_BASE_URL + endpoint;
  const token =
    document.querySelector('meta[name="csrf-token"]')?.getAttribute('content'); // Assuming
    Laravel CSRF token setup

  try {
    const response = await fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-Requested-With': 'XMLHttpRequest',
        // Include authentication if needed (e.g., Bearer token or cookie-based session)
        ...(token && {'X-CSRF-TOKEN': token}),
      },
      body: JSON.stringify(data),
    });

    // SCORM expects 'true', 'false', or a data string
    return await response.text();

  } catch (error) {
    console.error(`SCORM API Error on ${endpoint}:`, error);
    return "false";
  }
}
```

```

}

/**
 * Sends a GET request to the Laravel API endpoint.
 * @param {string} endpoint - The route segment (e.g., 'getvalue').
 * @param {object} params - Query parameters.
 * @returns {Promise<string>} The API response text (the CMI value).
 */
async function apiGet(endpoint, params = {}) {
  const query = new URLSearchParams(params).toString();
  const url = SCORM_API_BASE_URL + endpoint + (query ? `?${query}` : "");

  try {
    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'X-Requested-With': 'XMLHttpRequest',
      },
    });
    return await response.text();
  } catch (error) {
    console.error(`SCORM API Error on ${endpoint}:`, error);
    return ""; // Return empty string for failure in GetValue
  }
}

// --- SCORM 1.2 API Functions ---

const API = {
  // 1. SCORM Initialization
  LMSInitialize: async function(param) {
    if (IS_INITIALIZED) return "true";

    // Call the Laravel Initialize endpoint
    const result = await apiPost('initialize', { param });

    if (result === "true") {
      // After successful initialization, fetch all initial CMI data (Lesson Status, Suspend
      Data)
      CMI_DATA['cmi.core.lesson_status'] = await
      API.LMSGetValue('cmi.core.lesson_status');
      CMI_DATA['cmi.suspend_data'] = await API.LMSGetValue('cmi.suspend_data');

      IS_INITIALIZED = true;
    }
    return result;
  },

  // 2. SCORM Termination
  LMSFinish: async function(param) {
    if (!IS_INITIALIZED) return "true";

```

```

// Ensure any pending data is saved before finishing
const commitResult = await API.LMSCommit("");

// Call the Laravel Finish endpoint
const finishResult = await apiPost('finish', { param });

IS_INITIALIZED = false;
return finishResult;
},

// 3. Getting CMI Data
LMSGetValue: async function(element) {
  if (!IS_INITIALIZED && element !== 'cmi.core.student_id') {
    // SCORM content may check student ID before initialize
    // Only allow student_id or return error if not initialized
    return "";
  }

  // Check local cache first (for speed and consistency during a session)
  if (CMI_DATA[element] !== undefined) {
    return CMI_DATA[element];
  }

  // If not in cache, request from the Laravel API
  const value = await apiGet('getvalue', { element });
  // Cache the retrieved value locally
  CMI_DATA[element] = value;
  return value;
},

// 4. Setting CMI Data
LMSSetValue: async function(element, value) {
  if (!IS_INITIALIZED) return "false";

  // Update local cache immediately
  CMI_DATA[element] = value;

  // The SCORM standard typically requires LMSCommit to send data,
  // but for robustness, we update the DB here and again on Commit.
  // This makes the data loss less likely on immediate tab closure.
  return await apiPost('setvalue', { element, value });
},

// 5. Saving CMI Data (Explicit Save)
LMSCommit: async function(param) {
  if (!IS_INITIALIZED) return "false";
  // Since LMSSetValue immediately updates the DB in our simplified model,
  // LMSCommit just confirms the save with the API.
  return await apiPost('commit', { param });
},

```

```

// 6. Error Handling (Required by SCORM)
LMSGetLastErrorCode: function() {
    return "0"; // Simple success
},
LMSGetErrorString: function(errorCode) {
    return "No error";
},
LMSGetDiagnostic: function(errorCode) {
    return "SCORM 1.2 API adapter is functional.";
}
};

// --- Global Setup ---

// Expose the API object to the global window object.
// The SCORM content (running in an iframe) will look for this object
// in its parent window to establish communication.
window.API = API;

console.log("SCORM 1.2 API Adapter loaded and ready. Content ID:", CONTENT_ID);

```

2. SCORM Wrapper HTML Example

You need to embed your video/SCORM content inside an HTML file that includes the adapter script. This HTML file is what the user opens in the browser.

HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>SCORM Video Wrapper</title>
  <meta name="csrf-token" content="{{ csrf_token() }}">
</head>
<body>

  <h1>SCORM Compliant Video Player</h1>
  <p>This page hosts the SCORM content and provides the API object.</p>

  <iframe id="scormContent"
    src="/path/to/your/scorm/video/index.html"
    width="800"
    height="600"
    allowfullscreen>
  </iframe>

  <script src="scorm_api_adapter.js"></script>

```

```
</body>  
</html>
```

3. Key Points & Next Steps

- **Same-Origin Policy:** The SCORM content running in the `<iframe>` must be served from the same domain as the parent window for the content's internal JavaScript to successfully find the `window.parent.API` object.
- **Authentication:** The `fetch` calls in the adapter must be correctly authenticated. If your Laravel API uses Sanctum for SPA authentication, cookies and session handling should work automatically if the adapter is on the same domain. If using tokens, you must include a `Bearer` token in the headers.
- **CMI Data Mapping:** The `LMSGetValue` and `LMSSetValue` functions are simplified here. A full SCORM 1.2 or 2004 implementation requires robust logic to handle complex CMI elements like `cmi.interactions` (for quizzes) and `cmi.objectives`.
- **Error Reporting:** A production-ready API adapter must implement rigorous SCORM error code reporting via `LMSsetErrorString` and `LMSsetDiagnostic`.