simulación de un *sprint* de desarrollo industrial.

El objetivo es construir una aplicación web con **Renderizado del Lado del Servidor (SSR)** utilizando **Laravel**. Vamos a enfocarnos en por qué SSR sigue siendo crítico para el SEO, el rendimiento inicial (First Contentful Paint) y la accesibilidad.

Proyecto: "Plataforma de Reseñas de Cursos (CourseReview)"

Construiremos una plataforma donde los usuarios pueden ver una lista de cursos (públicamente) y, una vez autenticados, pueden dejar reseñas (calificación y comentario) de esos cursos.

El Principio SSR Clave: La totalidad del HTML de las vistas públicas (lista de cursos, detalles del curso) debe ser generada por el servidor (Laravel/Blade) **antes** de llegar al navegador. No usaremos React o Vue para el *frontend* principal; confiaremos en la arquitectura clásica y robusta de Laravel.

Fase 0: Configuración del Entorno (El "Boilerplate")

Aquí establecemos las fundaciones. En la industria, un *setup* incorrecto genera deuda técnica inmediata.

1. **Instalación de Laravel:**
    - Asegúrense de tener un entorno LAMP/LEMP funcional (PHP 8.1+, Composer, Node.js).
    - Ejecuten: composer create-project laravel/laravel course-review
2. **Configuración de la Base de Datos:**
    - Creen una base de datos (ej. course_review_db).
    - Configuren su archivo .env con las credenciales correctas (DB_DATABASE, DB_USERNAME, DB_PASSWORD).
3. **Andamiaje de Autenticación (Scaffolding):**
    - Vamos a usar el *starter kit* más ligero y enfocado en SSR: **Laravel Breeze** (en modo Blade).
    - composer require laravel/breeze --dev
    - php artisan breeze:install
        - Cuando pregunte, seleccionen **Blade** (Opción 0 o 1, dependiendo de la versión).
    - npm install && npm run dev

- php artisan migrate
- **Verificación:** Levanten el servidor (php artisan serve) y verifiquen que pueden registrarse e iniciar sesión.

Fase 1: El Modelo de Datos (La "Arquitectura de Negocio")

El modelo de datos es el esqueleto de la aplicación. Si esto falla, todo falla.

1. **Crear Modelo y Migración del Curso (Course):**
    - php artisan make:model Course -m
    - Abran el archivo de migración (..._create_courses_table.php).
    - Definan la estructura:
        - $table->string('title');
        - $table->string('slug')->unique(); (Esencial para URLs amigables y SEO).
        - $table->text('description');
        - $table->string('instructor');
2. **Crear Modelo y Migración de la Reseña (Review):**
    - php artisan make:model Review -m
    - Abran el archivo de migración (..._create_reviews_table.php).
    - Definan la estructura:
        - $table->foreignId('user_id')->constrained()->onDelete('cascade'); (Relación con el usuario).
        - $table->foreignId('course_id')->constrained()->onDelete('cascade'); (Relación con el curso).
        - $table->unsignedTinyInteger('rating'); (Calificación del 1 al 5).
        - $table->text('comment');
        - $table->timestamps();
3. **Definir Relaciones (Eloquent ORM):**

- En app/Models/User.php:

  //PHP

  public function reviews() {

```
    return $this->hasMany(Review::class);
}
```

- En app/Models/Course.php:

```
//PHP

public function reviews() {
    return $this->hasMany(Review::class);
}

// (Opcional pero recomendado)
public function getRouteKeyName() {
    return 'slug'; // Para usar el slug en la URL en lugar del ID
}
```

- En app/Models/Review.php:

```
//PHP

public function user() {
    return $this->belongsTo(User::class);
}

public function course() {
    return $this->belongsTo(Course::class);
}
```

4. **Ejecutar Migración:**

- php artisan migrate

Fase 2: Módulo de Administración (CRUD de Cursos)

Solo los usuarios autenticados (eventualmente "administradores", pero por ahora basta con autenticados) pueden gestionar los cursos.

1. **Crear el Controlador:**
    - Usaremos un controlador tipo *Resource* para seguir las convenciones de Laravel.
    - php artisan make:controller CourseController --resource --model=Course

2. **Definir Rutas (Protegidas):**
    - En routes/web.php, agreguen las rutas del CRUD dentro del *middleware* de autenticación.

//PHP

```
Route::middleware(['auth'])->group(function () {
   // ... (rutas de profile de Breeze) ...

   // Rutas para administrar cursos
   Route::resource('courses', CourseController::class)->except(['index', 'show']);
});
```

1.
    - (Noten que index y show las dejaremos públicas en la siguiente fase).
2. **Implementar Lógica del Controlador (CourseController):**
    - **create():** Muestra el formulario (resources/views/courses/create.blade.php).
    - **store():** Valida los datos (¡crítico!) y guarda el nuevo curso. Usen *Form Requests* para la validación (php artisan make:request StoreCourseRequest).
    - **edit(Course $course):** Muestra el formulario de edición (resources/views/courses/edit.blade.php).
    - **update(UpdateCourseRequest $request, Course $course):** Valida y actualiza el curso.
    - **destroy(Course $course):** Elimina el curso.
3. **Crear las Vistas (Blade):**
    - Diseñen los formularios (create.blade.php, edit.blade.php) usando los componentes de Breeze/Tailwind para mantener la consistencia visual.

Fase 3: Vistas Públicas (El Corazón del SSR)

Aquí es donde demostramos el SSR. Estas páginas deben cargar instantáneamente con todo el contenido, sin llamadas a APIs desde el *frontend*.

1. **Controlador Público:**
    - php artisan make:controller PublicCourseController
2. **Definir Rutas Públicas:**
    - En routes/web.php (fuera del *middleware* auth):

```
//PHP

use App\Http\Controllers\PublicCourseController;

// Página de inicio (Listado de cursos)
Route::get('/', [PublicCourseController::class, 'index'])->name('home');

// Vista de detalle del curso
Route::get('/curso/{course}', [PublicCourseController::class, 'show'])->name('courses.show');
```

3. **Implementar Lógica del Controlador Público:**

- En PublicCourseController.php:

```
// PHP

public function index() {
   // Obtenemos los cursos (paginados)
   $courses = Course::latest()->paginate(10);

   // Renderizamos la vista Blade y le pasamos los datos
   return view('home', ['courses' => $courses]);
}

public function show(Course $course) {
   // Cargamos el curso y sus reseñas (Eager Loading para optimizar queries)
```

```
    $course->load('reviews.user');

    // Renderizamos la vista de detalle
    return view('courses.show', ['course' => $course]);
}
```

1. **Crear Vistas Públicas (Blade):**
    - resources/views/home.blade.php:
        - Debe iterar sobre la variable $courses (pasada desde el controlador) y mostrar la lista.
        - Usen @foreach($courses as $course)... @endforeach.
        - Usen $courses->links() para mostrar la paginación.
    - resources/views/courses/show.blade.php:
        - Debe mostrar los detalles de $course->title, $course->description, etc.
        - Debe iterar sobre $course->reviews para mostrar las reseñas existentes.

Fase 4: Módulo de Reseñas (Interacción del Usuario)

Los usuarios autenticados pueden dejar reseñas en la página de detalle del curso.

1. **Controlador de Reseñas:**
    - php artisan make:controller ReviewController
2. **Ruta de Almacenamiento:**
    - En routes/web.php (dentro del *middleware* auth):

```
// PHP

Route::post('/curso/{course}/reviews', [ReviewController::class, 'store'])
    ->name('reviews.store');
```

1. **Lógica de store() en ReviewController:**
    - Validar los datos (usando *Form Request*): course_id (del parámetro de ruta), rating (requerido, numérico, 1-5), comment (requerido, string).

- Asignar el user_id desde el usuario autenticado: auth()->id().
- Crear la reseña.
- Redirigir de vuelta a la página del curso: return back()->with('success', 'Reseña enviada.');

2. **Formulario en la Vista (courses/show.blade.php):**
   - Usen @auth... @endauth para mostrar el formulario de reseña solo si el usuario está logueado.
   - Usen @guest para mostrar un enlace de "Inicia sesión para dejar una reseña".
   - El formulario debe hacer POST a la ruta reviews.store.

Fase 5: Pruebas (El Sello de Calidad de Capcom)

Un sistema que no se prueba, no funciona. En su nivel, las pruebas de *feature* (integración) son obligatorias.

1. **Pruebas Públicas (Pest/PHPUnit):**
   - php artisan make:test PublicViewTest
   - test_home_page_loads_successfully(): (Verifica un código 200 en /).
   - test_home_page_displays_courses(): (Verifica que un curso creado aparece en la vista).
   - test_course_detail_page_loads(): (Verifica el detalle del curso).
2. **Pruebas de Autenticación:**
   - php artisan make:test CourseManagementTest
   - test_guest_cannot_create_course(): (Verifica redirección al login).
   - test_authenticated_user_can_create_course(): (Simula un login y verifica la creación).
3. **Pruebas de Reseñas:**
   - test_authenticated_user_can_submit_review()
   - test_guest_cannot_submit_review()

Criterios de Evaluación y Ponderados (lo que voy a calificar y como lo hare)

1. **Repositorio Git (Obligatorio):**

- Un repositorio en GitHub o GitLab.
- Espero *commits* atómicos y mensajes claros. No quiero un solo *commit* que diga "proyecto final".

2. **Demostración Funcional:**

- La aplicación debe estar desplegada (pueden usar servicios gratuitos como Railway, Heroku, o un VPS si lo tienen).

3. **Calidad del Código (se tomara como examen):**

- **SSR (40%):** ¿Las vistas públicas se renderizan en el servidor? (Verificable viendo el "código fuente" en el navegador; debe contener el HTML del contenido).
- **Funcionalidad (30%):** ¿Funcionan el CRUD de cursos y el sistema de reseñas?
- **Pruebas (20%):** Cobertura y calidad de las pruebas PEST/PHPUnit.
- **Buenas Prácticas (10%):** Uso de Form Requests, Eager Loading (N+1), convenciones de Laravel.

Los avances se dividirán de la siguiente manera para la unidad 3:

Módulo 1: Fundamentos y Modelo de Datos

El objetivo es asegurar que todos tengan un entorno idéntico y entiendan la arquitectura de la base de datos.

- **Práctica 1 (En Sesión/Laboratorio): Instalación y Autenticación.**
    - **Objetivo:** Levantar el *scaffolding*.
    - **Pasos Guiados:**
        1. Instalación de Laravel 11.
        2. Configuración del .env y conexión a la base de datos local.
        3. Instalación de laravel/breeze (modo Blade).

4. Ejecución de php artisan migrate.

- **Evaluación de Práctica:** Al final de la sesión, deben mostrarme su pantalla con el sistema de login y registro de Breeze funcionando. Esto es un *check* de "completado/no completado".

- **Tarea 1: Arquitectura del Modelo de Datos.**
  - **Objetivo:** Demostrar competencia en el ORM (Eloquent) y la estructura de la base de datos.
  - **Requisitos:**
    1. Entregar los archivos de **Migración** (...create_courses_table.php y ...create_reviews_table.php) con todos los campos especificados en el plan.
    2. Entregar los archivos de **Modelo** (User.php, Course.php, Review.php) con todas las relaciones Eloquent (hasMany, belongsTo) correctamente definidas.
  - **Cómo lo Reviso:** Clonaré su repositorio, ejecutaré php artisan migrate. Si falla, el entregable está incompleto. Luego revisaré manualmente los 3 archivos de modelo.

Módulo 2: El Backend (CRUD de Cursos)

Aquí evaluamos la lógica de negocio del lado del servidor y la gestión de datos.

- **Práctica 2 (En Sesión/Laboratorio): Validación y Creación (El 'C' del CRUD).**
  - **Objetivo:** Entender cómo procesar formularios de manera segura.
  - **Pasos Guiados:**
    1. Juntos crearemos la ruta courses.create y courses.store (protegida por auth).
    2. Crearemos el CourseController.
    3. Crearemos la vista courses/create.blade.php usando componentes de Breeze.
    4. **Enfoque principal:** Crearemos juntos el StoreCourseRequest (php artisan make:request) y definiremos las reglas de validación.
  - **Evaluación de Práctica:** Deben mostrarme que el formulario guarda un curso y, más importante, que la validación falla (con mensajes de error) si intentan enviar datos incorrectos.

- **Tarea2: Módulo de Administración (CRUD Completo).**

- **Objetivo:** Completar la gestión de recursos.
- **Requisitos:**
    1. Implementar la funcionalidad completa de **Editar**, **Actualizar** y **Eliminar** para los cursos.
    2. Crear una vista simple de "dashboard" (courses.index) donde solo los usuarios autenticados puedan ver la lista de cursos para gestionar (con sus botones de editar/eliminar).
    3. **Requisito de Calidad:** Deben usar *Route Model Binding* (ej. public function edit(Course $course)) y *Form Requests* separados para Update.
- **Cómo lo Reviso:** Inciaré sesión, crearé un curso, lo editaré, y lo eliminaré. Revisaré su CourseController para asegurar que está limpio y que la validación está en sus propios *Request*.

Los avances se dividirán de la siguiente manera para la unidad 4:

Módulo 3: El Núcleo SSR (Vistas Públicas)

Este es el módulo más importante para el concepto de SSR. Aquí se demuestra la diferencia entre SSR y CSR.

- **Práctica 3 (En Sesión/Laboratorio): La Portada SSR (El 'R' de Read).**
    - **Objetivo:** Renderizar una lista de datos públicos desde el servidor.
    - **Pasos Guiados:**
        1. Crearemos el PublicCourseController y la ruta pública para el *home* (/).
        2. Implementaremos el método index, obteniendo los cursos con paginación (Course::paginate(10)).
        3. Crearemos la vista home.blade.php y usaremos @foreach para iterar $courses.
        4. Añadiremos $courses->links() para la paginación.
    - **Evaluación de Práctica:** Les pediré que "Vean el código fuente" (View Page Source) de su *home* en el navegador. Deben **mostrarme** el HTML renderizado de sus cursos en el código fuente, probando que el servidor lo envió.
- **Entregable 3 (Tarea): Vista de Detalle y Optimización (N+1).**
    - **Objetivo:** Demostrar la carga de datos relacionados (la página de detalle).

- **Requisitos:**
    1. Implementar la ruta pública y el método show en PublicCourseController.
    2. La vista courses/show.blade.php debe mostrar el detalle del curso y la lista de reseñas asociadas.
    3. **Requisito de Rendimiento (Crítico):** Deben resolver el problema de consultas N+1. Espero ver *Eager Loading* en su controlador (ej. Course::with('reviews.user')->findOrFail(...)).
- **Cómo lo Reviso:** Instalaré **Laravel Debugbar**. Visitaré la página de detalle de un curso con 10 reseñas. Si Debugbar me reporta más de 3-4 consultas a la base de datos (en lugar de 11), sé que implementaron mal el *Eager Loading* y serán penalizados.

Módulo 4: Interacción y Aseguramiento de Calidad (QA)

Cerramos el ciclo con la interacción del usuario y validamos que nada se haya roto.

- **Práctica 4 (En Sesión/Laboratorio): Interacción del Usuario (Formulario de Reseña).**
    - **Objetivo:** Manejar la lógica de un formulario para usuarios autenticados en una vista pública.
    - **Pasos Guiados:**
        1. En courses/show.blade.php, usaremos las directivas @auth y @guest de Blade.
        2. @guest mostrará un enlace para "Iniciar sesión".
        3. @auth mostrará el formulario para enviar una reseña.
        4. Crearemos el ReviewController y su ruta store (protegida) para procesar el formulario.
    - **Evaluación de Práctica:** Deben mostrarme la vista de detalle: una vez como invitado (solo viendo) y otra como usuario logueado (viendo el formulario).
- **Entregable 4 (Tarea Final): Pruebas (El Sello de Calidad de Capcom).**
    - **Objetivo:** Demostrar que la aplicación es robusta y está probada.
    - **Requisitos:**
        1. Entregar un mínimo de **4 pruebas de *feature*** (Feature Tests) en PEST o PHPUnit.
        2. **Prueba 1 (Pública):** test_home_page_is_accessible() (Verifica un código 200).

3. **Prueba 2 (Pública):** test_course_detail_page_displays_course_title() (Verifica que se ve contenido de la BD).
4. **Prueba 3 (Seguridad):** test_guest_cannot_access_create_course_page() (Verifica redirección al login).
5. **Prueba 4 (Funcional):** test_authenticated_user_can_create_a_course() (Simula un login y la creación de un curso).

- **Cómo lo Reviso:** Simplemente ejecutaran php artisan test. Si sus pruebas no pasan, o si no cubren estos 4 escenarios, el entregable no está completo.