Introducción Teórica

Un CRUD es una aplicación que permite realizar cuatro operaciones básicas en una base de datos:

- **C**reate: Crear registros.
- **R**ead: Leer registros.
- **U**pdate: Actualizar registros.
- **D**elete: Eliminar registros.

En Laravel, el CRUD se construye de manera sencilla gracias a su potente sistema de **modelos**, **controladores** y **vistas**, que trabajan en conjunto siguiendo el patrón **MVC (Modelo-Vista-Controlador)**.

¿Qué vamos a construir?

Crearemos una plataforma básica escolar que gestione:

1. **Usuarios:** Los usuarios del sistema. Laravel ya incluye el modelo User, que reutilizaremos.
2. **Proyectos-Asignatura:** Cada proyecto pertenece a un usuario y contiene información como nombre, descripción y fecha límite.

Esta aplicación incluirá:

- **Vistas Blade:** Una versión del CRUD que utiliza las plantillas predeterminadas de Laravel.
- **Vistas React:** Una versión moderna del CRUD, donde React se encarga de la interfaz gráfica.

¿Qué es Laravel?

Laravel es un framework de PHP que facilita la creación de aplicaciones web modernas, proporcionando herramientas y características como:

- **Eloquent ORM:** Para interactuar con la base de datos utilizando objetos.
- **Routing:** Para gestionar las URLs de nuestra aplicación.
- **Blade:** Un motor de plantillas simple y efectivo.
- **Artisan CLI:** Una línea de comandos para generar código y realizar tareas comunes.

Preparación del Entorno

Antes de comenzar con el desarrollo del CRUD, necesitamos configurar nuestro entorno de trabajo. Esta sección incluye los pasos necesarios para instalar y configurar Laravel

---

Requisitos previos

Antes de seguir, asegúrate de tener lo siguiente instalado en tu sistema:

- **PHP (>= 8.1)**
- **Composer** (gestor de dependencias para PHP)
- **Node.js** y **npm/yarn** (para React)
- **Laragon** (opcional, pero recomendado para principiantes)
- **Editor de texto** (como VSCode)

Crear un nuevo proyecto de Laravel

1. **Crear el proyecto:** Abre una terminal y escribe el siguiente comando para crear un nuevo proyecto llamado school_platform:

   ```
   composer create-project laravel/laravel school_platform
   ```

2. **Acceder al directorio del proyecto:**

   ```
   cd school_platform
   ```

3. **Configurar el servidor local de Laravel:** Ejecuta el servidor local para verificar que Laravel se instaló correctamente:

   ```
   php artisan serve
   ```

//***** **Nota:** check en el archivo config/session.php: *****//

```
'driver' => 'file', /*env('SESSION_DRIVER', 'database'),*/
```

Laravel estará disponible en http://127.0.0.1:8000. Abre esta URL en tu navegador y verifica que la página predeterminada de Laravel se cargue correctamente.

Configurar la base de datos

1. **Crear la base de datos:** Accede a tu gestor de bases de datos (como phpMyAdmin o HeidiSQL) y crea una nueva base de datos llamada school_platform.
2. **Configurar .env:** Abre el archivo .env en la raíz de tu proyecto y actualiza las credenciales de la base de datos:

```
DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=school_platform

DB_USERNAME=root

DB_PASSWORD= # Deja esto vacío si usas Laragon
```

3. **Migrar la tabla users:** Laravel incluye un modelo y una migración predeterminados para users. Ejecuta las migraciones con el comando:

```
php artisan migrate
```

   Esto creará la tabla users en tu base de datos.

Implementación del CRUD con Vistas Blade

En esta sección, construiremos el CRUD para gestionar proyectos-asignatura utilizando las vistas Blade de Laravel. Reutilizaremos el modelo de User y crearemos un nuevo modelo para los **proyectos**. Esta versión del CRUD será completamente funcional utilizando las herramientas integradas de Laravel.

Crear el modelo y la migración para los proyectos

1. **Generar el modelo:**

   Ejecuta el siguiente comando para crear el modelo Project con su migración y un controlador:

```
php artisan make:model Project -mcr
```

Esto generará:

- El modelo en app/Models/Project.php.
- La migración en database/migrations/{timestamp}_create_projects_table.php.
- El controlador en app/Http/Controllers/ProjectController.php.

2. **Configurar la migración:**

   Abre el archivo de migración generado en database/migrations y define las columnas de la tabla:

```php
    public function up()
     {
         Schema::create('projects', function (Blueprint $table) {
             $table->id(); // ID del proyecto
             $table->string('name'); // Nombre del proyecto
             $table->text('description')->nullable(); // Descripción del
proyecto
             $table->date('deadline'); // Fecha límite del proyecto
             $table->foreignId('user_id')->constrained()-
>onDelete('cascade'); // Relación con el usuario
             $table->timestamps(); // Timestamps para created_at y
updated_at
         });
     }
```

3. **Ejecutar la migración:**

   Ejecuta el comando para aplicar los cambios en la base de datos:

   ```
   php artisan migrate
   ```

   Ahora, la tabla projects estará creada en la base de datos.

Configurar el modelo Project

1. **Definir la relación con el modelo User:**

   Abre app/Models/Project.php y agrega lo siguiente:

```php
class Project extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'description', 'deadline', 'user_id'];
// Campos rellenables

    // Relación con el modelo User
    public function user()
    {
        // Un proyecto pertenece a un usuario
        return $this->belongsTo(User::class); // Relación de uno a muchos
    }
}
```

2. **Actualizar el modelo User:**

Agrega la relación inversa en app/Models/User.php:

```php
    public function projects()
    {
        // Un usuario tiene muchos proyectos
        return $this->hasMany(Project::class);
    }
```

Configurar el controlador

1. **Definir las funciones CRUD en el controlador:**

   Abre app/Http/Controllers/ProjectController.php y actualiza el controlador con las siguientes funciones:

```php
<?php

namespace App\Http\Controllers;

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

class ProjectController extends Controller
{
    // Mostrar la lista de proyectos
    public function index()
    {
        // Obtener todos los proyectos con el usuario relacionado
        $projects = Project::with('user')->get();
   return view('projects.index', compact('projects'));// Pasar a la vista
    }

    // Mostrar el formulario de creación
    public function create()
    {
        // Mostrar el formulario de creación
        return view('projects.create');
    }

    // Guardar un nuevo proyecto
    public function store(Request $request)
{
    $request->validate([
        'name' => 'required|string|max:255',
        'description' => 'nullable|string',
        'deadline' => 'required|date',
    ]);

    // Verificamos si existe al menos un usuario en la base de datos
    $defaultUser = \App\Models\User::first();

    if (!$defaultUser) {
        return redirect()->route('projects.index')->with('error', 'No
default user found. Please create a user first.');
    }

    Project::create([
```


```
        'name' => $request->name,
        'description' => $request->description,
        'deadline' => $request->deadline,
        'user_id' => $defaultUser->id, // Usuario predeterminado
    ]);

    return redirect()->route('projects.index')->with('success', 'Project
created successfully!');
}

    // Mostrar un proyecto específico
    public function show(Project $project)
    {
        return view('projects.show', compact('project'));
    }

    // Mostrar el formulario de edición
    public function edit(Project $project)
    {
        return view('projects.edit', compact('project'));
    }

    // Actualizar un proyecto
    public function update(Request $request, Project $project)
    {
        // Validar los datos del formulario
        $request->validate([
            'name' => 'required|string|max:255',
            'description' => 'nullable|string',
            'deadline' => 'required|date',
        ]);

        $project->update($request->only(['name', 'description',
'deadline']));

        return redirect()->route('projects.index')->with('success',
'Project updated successfully!');
    }

    // Eliminar un proyecto
    public function destroy(Project $project)
    {
        $project->delete();

        return redirect()->route('projects.index')->with('success',
'Project deleted successfully!');
    }
}
```

Nota: Cada función incluye comentarios y valida los datos para evitar errores comunes.

Crear las vistas Blade

1. **Crear los archivos de vistas:**

   En Laravel, podemos reutilizar una estructura común para todas las vistas mediante un componente de Blade. Creamos un archivo `layout.blade.php` que incluirá el header, estilos básicos, y un área de contenido dinámico.

   Crea un directorio `layouts` dentro de `resources/views` y agrega el siguiente archivo:

   `layout.blade.php (layout principal)`

   Crea un directorio `projects` dentro de `resources/views` y agrega los siguientes archivos:

   `index.blade.php (lista de proyectos)`

   `create.blade.php (formulario de creación)`

   `edit.blade.php (formulario de edición)`

   `show.blade.php (detalles del proyecto)`

Diseñar las vistas Blade

1. Crear el layout principal

Crea el archivo **layout.blade.php** en **resources/views/layouts/**:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>@yield('title', 'School Platform')</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min
.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container">
            <a class="navbar-brand" href="{{ route('projects.index')
}}">School Platform</a>
            <div class="collapse navbar-collapse">
                <ul class="navbar-nav ms-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{
route('projects.index') }}">Projects</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{
route('projects.create') }}">Create Project</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    <div class="container mt-4">
        @yield('content')
    </div>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundl
e.min.js"></script>
</body>
</html>
```

2. **Diseñar la vista i `resources/views/index.blade.php`:**

```
@extends('layouts.layout')
@section('title', 'Projects List')
@section('content')
    <h1>Projects List</h1>
    @if (session('success'))
        <div class="alert alert-success">
            {{ session('success') }}
        </div>
    @endif
    <a href="{{ route('projects.create') }}" class="btn btn-primary mb-
3">Create New Project</a>
    <table class="table table-bordered">
        <thead>
            <tr>
                <th>Name</th>
                <th>Description</th>
                <th>Deadline</th>
                <th>User</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            @forelse ($projects as $project)
                <tr>
                    <td>{{ $project->name }}</td>
                    <td>{{ $project->description }}</td>
                    <td>{{ $project->deadline }}</td>
                    <td>{{ $project->user->name }}</td>
                    <td>
                        <a href="{{ route('projects.show', $project) }}"
class="btn btn-info btn-sm">View</a>
                        <a href="{{ route('projects.edit', $project) }}"
class="btn btn-warning btn-sm">Edit</a>
                        <form action="{{ route('projects.destroy',
$project) }}" method="POST" style="display:inline;">
                            @csrf
                            @method('DELETE')
                            <button type="submit" class="btn btn-danger
btn-sm">Delete</button>
                        </form>
                    </td>
                </tr>
            @empty
                <tr>
                    <td colspan="5" class="text-center">No projects
found.</td>
```

```
                </tr>
            @endforelse
        </tbody>
    </table>
@endsection
```

3. Diseñar la vista i `resources/views/create.blade.php:`

Esta vista muestra un formulario para crear un nuevo proyecto.

```
@extends('layouts.layout')

@section('title', 'Create Project')

@section('content')
    <h1>Create a New Project</h1>
    @if ($errors->any())
        <div class="alert alert-danger">
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    <form action="{{ route('projects.store') }}" method="POST">
        @csrf
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
value="{{ old('name') }}" required>
        </div>
        <div class="mb-3">
            <label for="description" class="form-
label">Description</label>
            <textarea class="form-control" id="description"
name="description">{{ old('description') }}</textarea>
        </div>
        <div class="mb-3">
            <label for="deadline" class="form-label">Deadline</label>
            <input type="date" class="form-control" id="deadline"
name="deadline" value="{{ old('deadline') }}" required>
        </div>
        <button type="submit" class="btn btn-primary">Create</button>
```


```
    </form>
@endsection
1.  Vista edit.blade.php:
Similar a create.blade.php, pero con valores cargados del proyecto.
html
Copiar código
@extends('layouts.layout')

@section('title', 'Edit Project')

@section('content')
    <h1>Edit Project</h1>
    @if ($errors->any())
        <div class="alert alert-danger">
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    <form action="{{ route('projects.update', $project) }}"
method="POST">
        @csrf
        @method('PUT')
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
value="{{ $project->name }}" required>
        </div>
        <div class="mb-3">
            <label for="description" class="form-
label">Description</label>
            <textarea class="form-control" id="description"
name="description">{{ $project->description }}</textarea>
        </div>
        <div class="mb-3">
            <label for="deadline" class="form-label">Deadline</label>
            <input type="date" class="form-control" id="deadline"
name="deadline" value="{{ $project->deadline }}" required>
        </div>
        <button type="submit" class="btn btn-primary">Update</button>
    </form>
@endsection
```

4. **Diseñar la vista i `resources/views/ show.blade.php:`**

```
@extends('layouts.layout')

@section('title', 'Project Details')

@section('content')
    <h1>{{ $project->name }}</h1>
    <p><strong>Description:</strong> {{ $project->description }}</p>
    <p><strong>Deadline:</strong> {{ $project->deadline }}</p>
    <p><strong>User:</strong> {{ $project->user->name }}</p>
    <a href="{{ route('projects.index') }}" class="btn btn-
secondary">Back to List</a>
@endsection
```

Finalizar las rutas y datos de la base de datos

Configuración de rutas (routes/web.php)

```
<?php

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

Route::get('/welcome', function () {
    return view('welcome');
});

// Ruta de inicio, redirige a la lista de proyectos
Route::get('/', function () {
    return redirect()->route('projects.index');
});

// Rutas del CRUD para Project
Route::resource('projects', ProjectController::class);
```

Nota: La ruta raíz (/) redirige a la lista de proyectos para una navegación más amigable.

Población inicial de datos (seeds y factories)

Para simplificar las pruebas y tener datos iniciales, utilizaremos **factories** para generar registros ficticios en las tablas users y projects.

1. **Configurar un factory para Project:**

Laravel incluye un factory para User. Creamos uno para Project en `database/factories/ProjectFactory.php`:

```php
public function definition()
{
    return [
        'name' => $this->faker->sentence(3),
        'description' => $this->faker->paragraph,
        'deadline' => $this->faker->date(),
        'user_id' => User::factory(), // Relación con usuario
    ];
}
```

2. **Configurar un seeder para usuarios y proyectos:**

Creamos un archivo de seeder para poblar la base de datos. En database/seeders/DatabaseSeeder.php:

```php
public function run(): void
{
    // User::factory(10)->create();

    User::factory()->create([
        'name' => 'Test User',
        'email' => 'test@example.com',
    ]);
}
```

3. **Ejecutar los seeders:**

Para aplicar estos cambios, ejecutamos el siguiente comando en la terminal:

```
php artisan db:seed
```

Esto generará usuarios y proyectos ficticios en la base de datos.

Configuración adicional para la base de datos

1. **Verificar las migraciones:**

Confirma que las tablas users y projects estén creadas. Las columnas deben coincidir con las definidas en las migraciones.

2. **Verificar conexión con la base de datos:**

Usa el comando siguiente para probar la conexión y listar registros:

```
php artisan tinker
```

Dentro de Tinker, prueba los siguientes comandos:

```
\App\Models\User::all(); // Lista de usuarios
```

```
\App\Models\Project::all(); // Lista de proyectos
```

Si todo funciona correctamente, deberías ver los datos generados por los seeders.
