Tutorial Completo: Desarrollo de una Plataforma Escolar con Laravel 11 y React 19

Introducción

En este tutorial, desarrollaremos Laredu, una plataforma escolar basada en Laravel 11 y React 19 con TypeScript y Tailwind CSS 4.
Esta plataforma permitirá gestionar usuarios, cursos, asignaturas, evaluaciones, tareas, asistencia, eventos del calendario, mensajería interna y roles y permisos.

El objetivo es construir una API REST robusta y segura, junto con un frontend moderno y responsivo, utilizando buenas prácticas y asegurándonos de que todo el stack tecnológico sea compatible.

Este sistema permitirá a los usuarios:

- ✅ **Registrarse e iniciar sesión** de forma segura con **Laravel Sanctum**.
- ✅ **Administrar cursos y asignaturas** para estudiantes y docentes.
- ✅ **Crear y gestionar tareas** con calificaciones.
- ✅ **Administrar eventos del calendario** con **FullCalendar.js**.
- ✅ **Enviar y recibir mensajes privados** entre usuarios.
- ✅ **Asignar y gestionar roles y permisos** dentro de la plataforma.

Tecnologías Utilizadas

Tecnología	Uso en el Proyecto
Laravel 11	Backend y API REST
React 19 + TypeScript	Frontend dinámico
MySQL	Base de datos relacional
Laravel Sanctum	Autenticación API segura
Tailwind CSS 4	Diseño moderno y responsivo
FullCalendar.js	Gestión de eventos y calendario

Instalación de Laravel 11 y Configuración Inicial

📌 **Requisitos Previos**

Antes de instalar Laravel 11, asegúrate de tener:

✅ **PHP 8.2 o superior**
✅ **Composer** (gestor de dependencias de PHP)
✅ **MySQL 8 o MariaDB**
✅ **Node.js y npm** (para manejar el frontend en React)

Si usas Windows, **Laragon** facilita mucho la configuración del entorno.

📌 Instalar Laravel 11

Ejecuta estos comandos para instalar el backend:

```
# Crear el proyecto en la carpeta "backend"
composer create-project laravel/laravel backend


# Moverse a la carpeta del proyecto
cd backend


# Iniciar el servidor de desarrollo
php artisan serve
```

Si todo está correcto, verás en la terminal:

Application running at http://127.0.0.1:8000

📌 Configurar la Base de Datos

Laravel usa un archivo **.env** para la configuración de la base de datos. Modifica las siguientes líneas:

```
DB_CONNECTION=DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laredu
DB_USERNAME=root
DB_PASSWORD=
```

DB_USERNAME=root

DB_PASSWORD=

🔹 **Nota:** Si usas **Laragon**, XAMPP o WAMP, la contraseña puede estar vacía (DB_PASSWORD=).

Definición de la Base de Datos y Creación de Migraciones

La plataforma **Laredu** manejará más de **20 tablas** para gestionar cursos, usuarios, asignaturas, evaluaciones, eventos, asistencia, permisos y más.

A continuación, crearemos **todas las migraciones necesarias** en Laravel 11.

📌 **Listado Completo de Tablas**

Este diseño cubre **gestión de usuarios, cursos, asignaturas, evaluaciones, asistencia, eventos del calendario, roles y permisos, logs de auditoría** y más.

1 Gestión de Usuarios y Roles

Tabla	Descripción
users	Almacena información de los usuarios (estudiantes, profesores, administradores).
roles	Define roles como admin, profesor, estudiante, PAS (Personal Administrativo y de Servicios).
user_roles	Relación entre usuarios y roles (un usuario puede tener múltiples roles).

2 Gestión Académica (Cursos y Asignaturas)

Tabla	Descripción
courses	Almacena los cursos o programas académicos.
subjects	Asignaturas dentro de un curso.
course_user	Relación muchos a muchos entre cursos y usuarios (asigna estudiantes y profesores).

3 Evaluaciones y Entregas

Tabla	Descripción
assignments	Tareas y exámenes creados por los profesores.
submissions	Registra las entregas de tareas por parte de los estudiantes.
grades	Almacena las calificaciones de los estudiantes.

4 Gestión de Horarios y Asistencia

Tabla	Descripción
calendar_events	Eventos del calendario académico (clases, exámenes, reuniones).
attendance	Registra la asistencia de los estudiantes a cada sesión de clase.

5 Sistema de Permisos

Tabla	Descripción
permissions	Lista de permisos específicos dentro de la plataforma.
role_permissions	Relación entre roles y permisos.

6 Sistema de Comunicación y Notificaciones

Tabla	Descripción
messages	Mensajes privados entre usuarios.
notifications	Notificaciones del sistema enviadas a los usuarios.

7 Sistema de Auditoría y Registro de Errores

Tabla	Descripción
logs	Registro de eventos del sistema (acciones realizadas por usuarios).
activity_logs	Guarda cambios realizados dentro de la plataforma.

Creación de Migraciones en Laravel 11

Para generar todas estas tablas, ejecutamos los siguientes comandos:

```
php artisan make:migration create_users_table
php artisan make:migration create_roles_table
php artisan make:migration create_user_roles_table
php artisan make:migration create_courses_table
php artisan make:migration create_subjects_table
php artisan make:migration create_course_user_table
php artisan make:migration create_assignments_table
php artisan make:migration create_submissions_table
php artisan make:migration create_grades_table
php artisan make:migration create_calendar_events_table
php artisan make:migration create_attendance_table
php artisan make:migration create_permissions_table
php artisan make:migration create_role_permissions_table
php artisan make:migration create_messages_table
php artisan make:migration create_notifications_table
php artisan make:migration create_logs_table
php artisan make:migration create_activity_logs_table
php artisan make:migration create_error_logs_table
```

Esto debería ser el orden correcto:

- *2025_02_01_100000_create_users_table.php*
- 2025_02_01_101000_create_roles_table.php
- 2025_02_01_102000_create_user_roles_table.php
- 2025_02_01_103000_create_courses_table.php
- 2025_02_01_104000_create_subjects_table.php
- 2025_02_01_105000_create_course_user_table.php
- 2025_02_01_106000_create_assignments_table.php
- 2025_02_01_107000_create_submissions_table.php
- 2025_02_01_108000_create_grades_table.php
- 2025_02_01_109000_create_calendar_events_table.php
- 2025_02_01_110000_create_attendance_table.php
- 2025_02_01_111000_create_permissions_table.php
- 2025_02_01_112000_create_role_permissions_table.php
- 2025_02_01_113000_create_messages_table.php
- 2025_02_01_114000_create_notifications_table.php
- 2025_02_01_115000_create_logs_table.php
- 2025_02_01_116000_create_activity_logs_table.php
- 2025_02_01_117000_create_error_logs_table.php

Definición de Migraciones en Laravel 11

Ahora que hemos generado las migraciones, vamos a **definir la estructura de cada tabla** en los archivos dentro de **database/migrations/**.

1. Tabla users

La tabla de **usuarios** almacena información de estudiantes, profesores y administradores es la única que se crea automáticamente desde el principio, de momento no hacer ninguna modificación

2. Tabla roles

Los **roles** determinan los permisos de cada usuario (admin, profesor, estudiante, etc.).

📌 **Editar xxxx_xx_xx_create_roles_table.php:**

```
Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique(); // Nombre del rol (admin,
profesor, estudiante)
            $table->timestamps();
        });
```

3. Tabla user_roles

Cada usuario puede tener **uno o varios roles**.

📌 **Editar xxxx_xx_xx_create_user_roles_table.php:**

```
Schema::create('user_roles', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade');
            $table->foreignId('role_id')->constrained()-
>onDelete('cascade');
            $table->timestamps();
        });
```

4. Tabla courses

Define los **cursos o programas académicos** disponibles en la plataforma.

📌 **Editar xxxx_xx_xx_create_courses_table.php:**

```
Schema::create('courses', function (Blueprint $table) {
            $table->id();
            $table->string('name'); // Nombre del curso
            $table->text('description')->nullable(); // Descripción
opcional
            $table->timestamps();
        });
```

5. Tabla subjects

Cada curso tiene **varias asignaturas**.

📌 **Editar xxxx_xx_xx_create_subjects_table.php:**

```
Schema::create('subjects', function (Blueprint $table) {
            $table->id();
            $table->string('name'); // Nombre de la asignatura
            $table->foreignId('course_id')->constrained()-
>onDelete('cascade'); // Relación con curso
            $table->foreignId('teacher_id')->nullable()-
>constrained('users')->onDelete('set null'); // Profesor asignado
            $table->timestamps();
        });
```

6. Tabla course_user

Relaciona **estudiantes y profesores** con sus cursos.

📌 **Editar xxxx_xx_xx_create_course_user_table.php:**

```
Schema::create('course_user', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade');
            $table->foreignId('course_id')->constrained()-
>onDelete('cascade');
            $table->enum('role', ['teacher', 'student']); // Determina si
el usuario es profesor o estudiante
            $table->timestamps();
        });
```

7. Tabla assignments

Define las **tareas y exámenes** creados por los profesores.

📌 **Editar xxxx_xx_xx_create_assignments_table.php:**

```
Schema::create('assignments', function (Blueprint $table) {
            $table->id();
            $table->string('title'); // Título de la tarea
       $table->text('description')->nullable();// Descripción de la tarea
            $table->dateTime('due_date'); // Fecha de entrega
            $table->foreignId('subject_id')->constrained()-
>onDelete('cascade'); // Relación con asignatura
            $table->timestamps();
        });
```

8. Tabla submissions

Registra **las entregas** de tareas por parte de los estudiantes.

📌 **Editar xxxx_xx_xx_create_submissions_table.php:**

```
Schema::create('submissions', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade'); // Estudiante que entrega
            $table->foreignId('assignment_id')->constrained()-
>onDelete('cascade'); // Tarea a entregar
            $table->dateTime('submitted_at')-
>default(DB::raw('CURRENT_TIMESTAMP')); // Fecha de entrega
            $table->decimal('grade', 5, 2)->nullable(); // Calificación
            $table->timestamps();
        });
```

9. Tabla calendar_events

Esta tabla almacena **eventos del calendario**, como clases, reuniones y exámenes.

📌 **Editar xxxx_xx_xx_create_calendar_events_table.php:**

```
Schema::create('calendar_events', function (Blueprint $table) {
            $table->id();
            $table->string('title'); // Nombre del evento
         $table->text('description')->nullable(); // Descripción opcional
            $table->dateTime('start'); // Fecha y hora de inicio
            $table->dateTime('end'); // Fecha y hora de finalización
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade'); // Usuario que crea el evento
            $table->timestamps();
        });
```

10. Tabla attendance

Registra la **asistencia de los estudiantes** a clases.

📌 **Editar xxxx_xx_xx_create_attendance_table.php:**

```
Schema::create('attendance', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade'); // Estudiante
            $table->foreignId('calendar_event_id')->constrained()-
>onDelete('cascade'); // Evento asociado (clase)
            $table->enum('status', ['present', 'absent', 'late']); //
Estado de asistencia
            $table->timestamps();
        });
```

11. Tabla permissions

Lista los **permisos específicos** dentro de la plataforma.

📌 **Editar xxxx_xx_xx_create_permissions_table.php:**

```
Schema::create('permissions', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique(); // Nombre del permiso
            $table->timestamps();
        });
```

12. Tabla role_permissions

Establece la **relación entre roles y permisos**.

📌 **Editar xxxx_xx_xx_create_role_permissions_table.php:**

```
Schema::create('role_permissions', function (Blueprint $table) {
            $table->id();
            $table->foreignId('role_id')->constrained()-
>onDelete('cascade'); // Relación con roles
            $table->foreignId('permission_id')->constrained()-
>onDelete('cascade'); // Relación con permisos
            $table->timestamps();
        });
```

13. Tabla messages

Maneja la **mensajería interna** entre usuarios.

📌 **Editar xxxx_xx_xx_create_messages_table.php:**

```
Schema::create('messages', function (Blueprint $table) {
            $table->id();
            $table->foreignId('sender_id')->constrained('users')-
>onDelete('cascade'); // Usuario remitente
            $table->foreignId('receiver_id')->constrained('users')-
>onDelete('cascade'); // Usuario destinatario
            $table->text('content'); // Contenido del mensaje
            $table->boolean('is_read')->default(false); // Estado del
mensaje
            $table->timestamps();
        });
```

14. Tabla notifications

Maneja las notificaciones del sistema.

📌 Editar xxxx_xx_xx_create_notifications_table.php:

```
Schema::create('notifications', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()-
>onDelete('cascade'); // Usuario que recibe la notificación
            $table->string('type'); // Tipo de notificación
            $table->text('data'); // Datos en formato JSON
            $table->boolean('is_read')->default(false); // Estado de
lectura
            $table->timestamps();
        });
```

15. Tabla logs

Registro general de eventos en la plataforma.

📌 Editar xxxx_xx_xx_create_logs_table.php:

```
Schema::create('logs', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->nullable()->constrained()-
>onDelete('cascade'); // Usuario que realizó la acción
            $table->string('action'); // Acción realizada
            $table->text('details')->nullable(); // Detalles adicionales
            $table->timestamps();
        });
```

16. Tabla activity_logs

Registra modificaciones realizadas en la plataforma.

📌 Editar xxxx_xx_xx_create_activity_logs_table.php:

```
Schema::create('activity_logs', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->nullable()->constrained()-
>onDelete('cascade'); // Usuario responsable
            $table->string('table_name'); // Tabla afectada
            $table->text('changes'); // Descripción de los cambios
            $table->timestamps();
        });
```

17. Tabla error_logs

Registra errores del sistema.

📌 Editar xxxx_xx_xx_create_error_logs_table.php:

```
Schema::create('error_logs', function (Blueprint $table) {
            $table->id();
            $table->text('error_message'); // Mensaje del error
            $table->text('stack_trace'); // Detalles técnicos
            $table->timestamps();
        });
```

Ejecutar todas las migraciones

Ahora que todas las migraciones están definidas, ejecutamos:

```
php artisan migrate
```

Si todo está correcto, Laravel creará **todas las tablas** en la base de datos **MySQL**.

Creación de Modelos en Laravel 11 con Eloquent

En este paso, definiremos los **Modelos** que representan cada tabla en la base de datos y estableceremos las **relaciones** correspondientes. Usaremos **Eloquent**, el ORM de Laravel, para interactuar con MySQL de forma sencilla.

Estructura Básica de un Modelo

En Laravel 11, cada modelo se ubica en la carpeta app/Models/. Cada uno:

1. **Extiende** la clase base Model.
2. **Usa** el trait HasFactory para permitir la generación de factories.
3. Define las **relaciones** Eloquent correspondientes a las tablas.
4. Usa la propiedad $fillable o $guarded para la asignación masiva.

Ejemplo:

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Example extends Model
{
    use HasFactory;

    // $fillable indica qué campos pueden ser asignados en masa
    protected $fillable = ['column1', 'column2'];

    // Aquí definimos las relaciones Eloquent
}
```

Ejecutaremos en la terminal cada comando para generar el archivo base y luego añadiremos el contenido.

```
php artisan make:model Role
php artisan make:model Permission
php artisan make:model Course
php artisan make:model Subject
php artisan make:model Assignment
php artisan make:model Submission
php artisan make:model Grade
php artisan make:model CalendarEvent
php artisan make:model Attendance
php artisan make:model Message
php artisan make:model Notification
php artisan make:model Log
php artisan make:model ActivityLog
php artisan make:model ErrorLog
```

Modelo User

Este modelo gestiona la información de los usuarios (estudiantes, profesores, administradores).

```php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
```

```
            'password' => 'hashed',
        ];
    }

    public function roles()
    {
        return $this->belongsToMany(Role::class, 'user_roles');
    }

    /**
     * Relación con asignaturas (profesor).
     * Un profesor puede tener muchas asignaturas.
     */
    public function subjects()
    {
        return $this->hasMany(Subject::class, 'teacher_id');
    }
    /**
     * Relación con cursos a través de la tabla course_user.
     * Permite asignar estudiantes y profesores a cursos.
     */
    public function courses()
    {
        return $this->belongsToMany(Course::class, 'course_user')-
>withPivot('role');
    }
    /**
     * Relación con entregas de tareas (submissions).
     */
    public function submissions()
    {
        return $this->hasMany(Submission::class);
    }
    /**
     * Relación con eventos del calendario (CalendarEvent).
     */
    public function calendarEvents()
    {
        return $this->hasMany(CalendarEvent::class);
    }
}
```

Modelo Role.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    use HasFactory;

    // Campos que se pueden asignar masivamente
    protected $fillable = [
        'name'
    ];

    // Relación con usuarios a través de user_roles
    public function users()
    {
        // Un rol puede pertenecer a muchos usuarios
        return $this->belongsToMany(User::class, 'user_roles');
    }

    // Relación con permisos a través de role_permissions
    public function permissions()
    {
        // Un rol puede tener varios permisos
        return $this->belongsToMany(Permission::class,
'role_permissions');
    }
}
```

Modelo Permission.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Permission extends Model
{
    use HasFactory;

    protected $fillable = [
        'name'
```

```
    ];

    // Relación con roles a través de role_permissions
    public function roles()
    {
        // Un permiso puede asociarse a varios roles
        return $this->belongsToMany(Role::class, 'role_permissions');
    }
}
```

Etc… continuar a crear todos los modelos

Creación y Uso de Seeders en Laravel 11

La siembra (seeding) nos permite **llenar la base de datos** con datos iniciales o de prueba. Esto es especialmente útil para **roles, usuarios iniciales, cursos y asignaturas** que necesitaremos para probar la plataforma.

Generar Seeders

En **Laravel 11**, utilizamos el comando make:seeder para crear clases de seeder. Por ejemplo:

```
php artisan make:seeder RolesTableSeeder
php artisan make:seeder UsersTableSeeder
php artisan make:seeder CoursesTableSeeder
```

Estos comandos generarán archivos en **database/seeders/**.

Nota: El archivo DatabaseSeeder.php se crea por defecto y es el lugar donde podemos llamar a todos nuestros seeders.

Ejemplo de un Seeder para Roles (RolesTableSeeder)

Aquí tenemos un ejemplo de **RolesTableSeeder**:

```
    public function run()
    {
        // Creamos roles básicos
        Role::create(['name' => 'admin']);
        Role::create(['name' => 'teacher']);
        Role::create(['name' => 'student']);
        Role::create(['name' => 'PAS']);
    }
}
```

Ejemplo de un Seeder para Usuarios (UsersTableSeeder)

Para **usuarios** con diferentes roles:

```
public function run()
{
    // Crear un usuario admin
    $admin = User::create([
        'name' => 'Administrator',
        'email' => 'admin@example.com',
        'password' => Hash::make('admin123'),
    ]);

    // Obtener el rol admin de la base de datos
    $roleAdmin = Role::where('name', 'admin')->first();

    // Asignar el rol admin al usuario
    $admin->roles()->attach($roleAdmin->id);

    // Crear un usuario profesor
    $teacher = User::create([
        'name' => 'John Doe',
        'email' => 'john@example.com',
        'password' => Hash::make('teacher123'),
    ]);

    // Obtener el rol teacher
    $roleTeacher = Role::where('name', 'teacher')->first();
    // Asignar el rol al usuario
    $teacher->roles()->attach($roleTeacher->id);

    // Crear un usuario estudiante
    $student = User::create([
        'name' => 'Jane Student',
        'email' => 'jane@example.com',
        'password' => Hash::make('student123'),
    ]);

    // Obtener el rol student
    $roleStudent = Role::where('name', 'student')->first();
    // Asignar el rol al usuario
    $student->roles()->attach($roleStudent->id);
}
```

Ejemplo de un Seeder para Cursos (CoursesTableSeeder)

```
public function run()
{
    Course::create([
        'name' => 'Curso de Laravel',
        'description' => 'Aprendiendo Laravel 11',
    ]);

    Course::create([
        'name' => 'Curso de React',
        'description' => 'Introducción a React 19',
    ]);

    Course::create([
        'name' => 'Curso de PHP Básico',
        'description' => 'Fundamentos de PHP',
    ]);
}
```

Registrar Seeders en DatabaseSeeder

El archivo **DatabaseSeeder.php**, ubicado en database/seeders/, se encarga de orquestar todos los seeders. Debes **llamar** a cada seeder dentro de su método **run()**:

```
public function run(): void
{
    // Llamamos a nuestros seeders
    $this->call([
        RolesTableSeeder::class,
        UsersTableSeeder::class,
        CoursesTableSeeder::class,
        // Agrega más seeders si los tienes...
    ]);
}
```

6. Ejecutar Seeders

Para **ejecutar** todos los seeders, basta con:

```
php artisan db:seed
```

Si deseas ejecutar un **seeder específico**, puedes usar:

```
php artisan db:seed --class=RolesTableSeeder
```

Atención: Si es necesario limpiar la base de datos antes de sembrar:
Antes de sembrar datos, revisa si tu base de datos está vacía o si deseas hacer ***`php artisan migrate:fresh --seed`*** para re-crear las tablas y poblarlas.

Configuración de Autenticación con Laravel Sanctum (Laravel 11)

En esta sección, integraremos **Laravel Sanctum** en **Laravel 11** para manejar la autenticación por medio de **API Tokens**. Esto permitirá que nuestro **frontend en React** pueda comunicarse de manera segura con el **backend**, protegiendo rutas que requieran un usuario autenticado.

1. Instalación y Configuración de Sanctum

a) Instalar Sanctum

En Laravel 11, **Sanctum** se instala con:

```
composer require laravel/sanctum
```

Luego, ejecutamos las migraciones necesarias para Sanctum:

```
php artisan migrate
```

Esto creará la tabla **personal_access_tokens**, donde se guardarán los tokens.

Nota: A partir de Laravel 11, la configuración de Sanctum es más simple. Ya no editamos Kernel.php como en versiones anteriores.

b) Revisar el Archivo de Configuración (Opcional)

Se generará un archivo config/sanctum.php. Por defecto, ya está configurado para entornos donde el frontend y backend se encuentran en dominios diferentes. Puedes revisar la opción stateful si planeas usar **cookies** con un SPA. Para tokens simples, la configuración por defecto suele ser suficiente.

2. Creación del Controlador de Autenticación

Para manejar registro, login y logout, crearemos un controlador llamado **AuthController**. Ejecuta:

```
php artisan make:controller AuthController --api
```

```php
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\ValidationException;
```

```php
class AuthController extends Controller
{
    /**
     * Register a new user (Registro de un nuevo usuario)
     */
    public function register(Request $request)
    {
        // Validación de campos
        $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8',
        ]);

        // Crear al usuario
        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return response()->json([
            'message' => 'User registered successfully',
            'user' => $user
        ], 201);
    }









   /**
     * Login a user (Iniciar sesión)
     */
    public function login(Request $request)
    {
        // Validación de campos
        $request->validate([
            'email' => 'required|string|email',
            'password' => 'required|string',
        ]);
        // Verificar si el usuario existe
        $user = User::where('email', $request->email)->first();

       if (!$user || !Hash::check($request->password, $user->password)) {
            throw ValidationException::withMessages([
```

```
                'email' => ['The provided credentials are incorrect.'],
            ]);
        }
        // Crear un token para el usuario
        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'message' => 'Login successful',
            'token' => $token,
            'user' => $user
        ], 200);
    }
    /**
     * Logout a user (Cerrar sesión)
     */
    public function logout(Request $request)
    {
        // Revocar todos los tokens del usuario autenticado
        $request->user()->tokens()->delete();

        return response()->json([
            'message' => 'Logout successful'
        ], 200);
    }
    /**
     * Get current user (Obtener datos del usuario autenticado)
     */
    public function me(Request $request)
    {
        return response()->json([
            'user' => $request->user()
        ], 200);
    }
}
```

3. Definir Rutas de Autenticación

En routes/api.php, agregamos las rutas:

```
<?php

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

// Rutas públicas (No requieren token)
Route::post('/register', [AuthController::class, 'register']); //
Registro
Route::post('/login', [AuthController::class, 'login']); // Login
```

```php
// Rutas protegidas (Requieren token)
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']); // Logout
    Route::get('/me', [AuthController::class, 'me']); // Obtener datos
del usuario autenticado
});
```

Nota: Observa que usamos Route::middleware('auth:sanctum') para proteger las rutas. Cualquier petición a estas rutas debe incluir el token en el header **Authorization** como Bearer <TOKEN>.

4. Notas Importantes y Solución de Errores Comunes en Instalaciones Minimalistas de Laravel 11

En esta sección, resumimos varios **errores frecuentes** que pueden aparecer en instalaciones **minimalistas** o no estándar de **Laravel 11**, y cómo resolverlos.

1. Error: Rutas no registradas (Sólo aparecen 4 rutas en php artisan route:list)

En un **proyecto estándar** de Laravel 11, existe un **RouteServiceProvider** que carga routes/api.php y routes/web.php. Sin embargo, en instalaciones minimalistas:

1. **El archivo RouteServiceProvider.php puede no existir.**
2. En vez de config/app.php, hay un archivo bootstrap/providers.php que define los **service providers** de la aplicación.

Solución:

Crear manualmente app/Providers/RouteServiceProvider.php con contenido:

```php
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Route;
use Illuminate\Foundation\Support\Providers\RouteServiceProvider as
ServiceProvider;

class RouteServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     */
    public function boot()
    {
        // Aquí registramos las rutas de la API y las rutas web
        $this->routes(function () {
            // Rutas API
            Route::middleware('api')
                ->prefix('api')
                ->group(base_path('routes/api.php'));

            // Rutas Web
            Route::middleware('web')
                ->group(base_path('routes/web.php'));
        });
    }
}
```

Registrar el RouteServiceProvider en tu archivo bootstrap/providers.php:

```php
<?php

return [
    App\Providers\AppServiceProvider::class,

    // Añadir manualmente el RouteServiceProvider
    App\Providers\RouteServiceProvider::class,
];
```

Limpiar caché:

```
php artisan route:clear
php artisan config:clear
php artisan cache:clear
```

Verificar las rutas con:

```
php artisan route:list
```

Ahora cuando se harán las pruebas con ***postman*** deberían aparecer POST /api/register, POST /api/login, etc.

2. Error: SQLSTATE[42S02]: Base table or view not found: 1146 Table 'laredu.personal_access_tokens' doesn't exist

Este error surge cuando intentas usar **Sanctum** ($user->createToken(...)) pero **no** existe la tabla ***personal_access_tokens***. En instalaciones estándar de Laravel, Sanctum la crea mediante sus migraciones. En una instalación minimalista, no se publican dichas migraciones por defecto.

Solución:

Crear manualmente la migración:

```
php artisan make:migration create_personal_access_tokens_table
```

Editar la migración para definir la tabla (igual que la de Sanctum por defecto):

```php
Schema::create('personal_access_tokens', function (Blueprint $table) {
            $table->id();
            $table->morphs('tokenable'); // Para relacionarse con
diferentes modelos (User, etc.)
            $table->string('name');
            $table->string('token', 64)->unique();
            $table->text('abilities')->nullable();
            $table->timestamp('last_used_at')->nullable();
            $table->timestamp('expires_at')->nullable();
            $table->timestamps();
        });
```

Ejecutar:

```
php artisan migrate
```

Al terminar, la tabla personal_access_tokens estará creada y $user->createToken('auth_token') podrá insertar el token sin fallos.

3. Error: Target class [files] does not exist al ejecutar php artisan route:clear o config:clear

Esto implica que la aplicación no encuentra el binding de files en el contenedor, el cual suele estar definido por el **FilesystemServiceProvider** en instalaciones estándar.

Solución:

Asegúrate de que tu providers incluya la referencia a Illuminate\Filesystem\FilesystemServiceProvider o un provider equivalente.

Verifica tus dependencias con composer.json y que "laravel/framework": "^11.0" esté correctamente instalado:

```
composer install
composer update
```

Si tu proyecto es muy minimal, añade manualmente un service provider que vincule 'files' a la clase Illuminate\Filesystem\Filesystem.

4. Rutas HTTP 404 al usar URLs mezcladas (Apache vs. Artisan Serve)

Caso común: has configurado **Apache** en http://127.0.0.1/laredu/backend/public/, y a la vez usas php artisan serve en http://127.0.0.1:8000. Esto puede generar confusión:

- Si usas php artisan serve, la URL correcta suele ser http://127.0.0.1:8000/api/...
- Si apuntas a la carpeta public/ en Apache, la URL puede ser algo como http://localhost/laredu/backend/public/api/...

Recomendación: en **desarrollo**, usar **solo** php artisan serve o configurar el VirtualHost de Apache para que la raíz sea la carpeta public/. No mezclar ambos.

☑ Resumen

- **Instalaciones minimalistas** de Laravel 11 pueden carecer de ficheros como RouteServiceProvider.php o migraciones de Sanctum.
- **Crear** manualmente el RouteServiceProvider y registrarlo en bootstrap/providers.php para que api.php y web.php se carguen.

- **Crear** manualmente la migración de personal_access_tokens si Sanctum no la provee.
- **Revisar** providers para que la clase files esté disponible y no te cause errores al limpiar cachés.
- **No mezclar** URLs de Apache y Artisan Serve simultáneamente.

5. Probando la Autenticación con Postman

Para probar nuestros *endpoints*:

Registrar un usuario

```
Método: POST
URL: http://127.0.0.1:8000/api/register
Body (JSON):
{
  "name": "Test User",
  "email": "test@example.com",
  "password": "password123"
}
```

Iniciar sesión

```
Método: POST
URL: http://127.0.0.1:8000/api/login
Body (JSON):
{
  "email": "test@example.com",
  "password": "password123"
}
Respuesta exitosa:
{
  "message": "Login successful",
  "token": "1|sjhdgfjhsdfgh..."," // Tu token real
  "user": {
    "id": 1,
    "name": "Test User",
    "email": "test@example.com",
    ...
  }
}
```

Obtener datos del usuario autenticado

```
Método: GET
URL: http://127.0.0.1:8000/api/me
Headers: Authorization: Bearer <TU_TOKEN>
Respuesta esperada:
{
  "user": {
    "id": 1,
    "name": "Test User",
    "email": "test@example.com",
    ...
  }
}
```

Cerrar sesión

```
Método: POST
URL: http://127.0.0.1:8000/api/logout
Headers: Authorization: Bearer <TU_TOKEN>
Respuesta:
{
  "message": "Logout successful"
}
```

5. Integración con el Frontend (CORS)

A partir de **Laravel 11**, la forma recomendada para **publicar** la configuración de CORS es mediante el comando:

```
php artisan config:publish cors
```

Este comando creará un archivo cors.php dentro de tu carpeta config/, donde podrás personalizar los valores de tu configuración **Cross-Origin Resource Sharing**. Luego, **Laravel** responderá automáticamente a las peticiones **OPTIONS** vía el **HandleCors** middleware que se encuentra en la pila de middlewares global.

La documentación oficial de Laravel 11 detalla este flujo en:
https://laravel.com/docs/11.x/routing#cors

Por tanto, cuando configures tu CORS, lo habitual será:

1. **Publicar** la configuración:

```
php artisan config:publish cors
```

Editar el nuevo archivo config/cors.php para ajustar las opciones:

```
return [
    'paths' => ['api/*'],
    'allowed_methods' => ['*'],
    'allowed_origins' => ['http://localhost:3000'],
    'allowed_headers' => ['*'],
    'exposed_headers' => [],
    'max_age' => 0,
    'supports_credentials' => false,
];
```

Reiniciar el servidor de desarrollo o limpiar la caché de configuración si es necesario:

```
php artisan config:clear
```

Así, tu aplicación estará lista para **aceptar peticiones** desde tu **frontend en React 19** o cualquier otro origen que tú especifiques.

Implementación de Controladores CRUD y Rutas API REST

En esta sección, crearemos los **Controladores CRUD** para las distintas entidades de la plataforma **Laredu** en **Laravel 11**. Las rutas serán definidas en api.php para que el **frontend (React 19)** pueda consumir la API REST.

Controlador de Cursos (CourseController)

1 Creación

```
php artisan make:controller CourseController --api
```

Esto generará app/Http/Controllers/CourseController.php. Editamos el archivo para implementar un CRUD sencillo:

```php
<?php

namespace App\Http\Controllers;

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

class CourseController extends Controller
{
    /**
     * Display a listing of courses (Obtener todos los cursos).
     */
```

```
    public function index()
    {
        return response()->json(Course::all(), 200);
    }

    /**
     * Store a newly created course (Crear un nuevo curso).
     */
    public function store(Request $request)
    {
        // Validación simple
        $request->validate([
            'name' => 'required|string|max:255',
            'description' => 'nullable|string',
        ]);

        $course = Course::create($request->all());

        return response()->json([
            'message' => 'Course created successfully',
            'course'  => $course
        ], 201);
    }

    /**
     * Display the specified course (Mostrar curso por ID).
     */
    public function show($id)
    {
        $course = Course::find($id);
        if (!$course) {
            return response()->json(['message' => 'Course not found'],
404);
        }
        return response()->json($course, 200);
    }

    /**
     * Update the specified course (Actualizar curso).
     */
    public function update(Request $request, $id)
    {
        $course = Course::find($id);
        if (!$course) {
            return response()->json(['message' => 'Course not found'],
404);
        }

        $request->validate([
```

```
            'name' => 'nullable|string|max:255',
            'description' => 'nullable|string',
        ]);

        $course->update($request->all());
        return response()->json([
            'message' => 'Course updated successfully',
            'course'  => $course
        ], 200);
    }

    /**
     * Remove the specified course (Eliminar curso).
     */
    public function destroy($id)
    {
        $course = Course::find($id);
        if (!$course) {
            return response()->json(['message' => 'Course not found'],
404);
        }

        $course->delete();
        return response()->json(['message' => 'Course deleted
successfully'], 200);
    }
}
```

2 Definir Rutas

En el archivo routes/api.php, agregamos las rutas para **CourseController**:

```
use App\Http\Controllers\CourseController;

// Rutas públicas (No requieren token)
Route::post('/register', [AuthController::class, 'register']); //
Registro
Route::post('/login', [AuthController::class, 'login']); // Login

// Rutas protegidas (Requieren token)
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']); // Logout
    Route::get('/me', [AuthController::class, 'me']); // Obtener datos
del usuario autenticado
});
```

Atención: Solo usuarios **autenticados** (con token Sanctum) podrán acceder a estos endpoints.

Controlador de Asignaturas (SubjectController)

1 Creación

php artisan make:controller SubjectController --api

Editar app/Http/Controllers/SubjectController.php:

```php
<?php

namespace App\Http\Controllers;

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

class SubjectController extends Controller
{
    /**
     * Get all subjects (Obtener todas las asignaturas).
     */
    public function index()
    {
        return response()->json(Subject::all(), 200);
    }

    /**
     * Create a new subject (Crear una nueva asignatura).
     */
    public function store(Request $request)
    {
        $request->validate([
            'name'       => 'required|string|max:255',
            'course_id'  => 'required|exists:courses,id',
            'teacher_id' => 'nullable|exists:users,id',
        ]);

        $subject = Subject::create($request->all());
        return response()->json([
            'message' => 'Subject created successfully',
            'subject' => $subject
        ], 201);
    }

    /**
     * Show a specific subject (Mostrar asignatura por ID).
     */
    public function show($id)
    {
        $subject = Subject::find($id);
```

```
        if (!$subject) {
            return response()->json(['message' => 'Subject not found'],
404);
        }
        return response()->json($subject, 200);
    }

    /**
     * Update a subject (Actualizar asignatura).
     */
    public function update(Request $request, $id)
    {
        $subject = Subject::find($id);
        if (!$subject) {
            return response()->json(['message' => 'Subject not found'],
404);
        }

        $request->validate([
            'name'       => 'nullable|string|max:255',
            'course_id'  => 'nullable|exists:courses,id',
            'teacher_id' => 'nullable|exists:users,id',
        ]);

        $subject->update($request->all());
        return response()->json([
            'message' => 'Subject updated successfully',
            'subject' => $subject
        ], 200);
    }

    /**
     * Delete a subject (Eliminar asignatura).
     */
    public function destroy($id)
    {
        $subject = Subject::find($id);
        if (!$subject) {
            return response()->json(['message' => 'Subject not found'],
404);
        }

        $subject->delete();
        return response()->json(['message' => 'Subject deleted
successfully'], 200);
    }
}
```

2 Definir Rutas

En routes/api.php, agregamos:

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

Controlador de Tareas y Entregas

En la plataforma **Laredu**, tenemos dos entidades:

- **assignments** (tareas/exámenes)
- **submissions** (entregas)

1 AssignmentController

php artisan make:controller AssignmentController --api

Editar app/Http/Controllers/AssignmentController.php:

```
<?php

namespace App\Http\Controllers;

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

class AssignmentController extends Controller
{
    /**
     * Get all assignments (Obtener todas las tareas).
     */
    public function index()
    {
        return response()->json(Assignment::all(), 200);
    }

    /**
     * Create a new assignment (Crear una nueva tarea).
     */
```

```
    public function store(Request $request)
    {
        $request->validate([
            'title'       => 'required|string|max:255',
            'description' => 'nullable|string',
            'due_date'    => 'required|date',
            'subject_id'  => 'required|exists:subjects,id',
        ]);

        $assignment = Assignment::create($request->all());
        return response()->json([
            'message'    => 'Assignment created successfully',
            'assignment' => $assignment
        ], 201);
    }

    /**
     * Show a specific assignment (Mostrar tarea).
     */
    public function show($id)
    {
        $assignment = Assignment::find($id);
        if (!$assignment) {
            return response()->json(['message' => 'Assignment not
found'], 404);
        }
        return response()->json($assignment, 200);
    }

    /**
     * Update an assignment (Actualizar tarea).
     */
    public function update(Request $request, $id)
    {
        $assignment = Assignment::find($id);
        if (!$assignment) {
            return response()->json(['message' => 'Assignment not
found'], 404);
        }

        $request->validate([
            'title'       => 'nullable|string|max:255',
            'description' => 'nullable|string',
            'due_date'    => 'nullable|date',
            'subject_id'  => 'nullable|exists:subjects,id',
        ]);

        $assignment->update($request->all());
        return response()->json([
```

```
            'message'    => 'Assignment updated successfully',
            'assignment' => $assignment
        ], 200);
    }

    /**
     * Delete an assignment (Eliminar tarea).
     */
    public function destroy($id)
    {
        $assignment = Assignment::find($id);
        if (!$assignment) {
            return response()->json(['message' => 'Assignment not
found'], 404);
        }

        $assignment->delete();
        return response()->json(['message' => 'Assignment deleted
successfully'], 200);
    }
}
```

2 SubmissionController

php artisan make:controller SubmissionController --api

Editar app/Http/Controllers/SubmissionController.php:

```
<?php

namespace App\Http\Controllers;

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

class SubmissionController extends Controller
{
    /**
     * Get all submissions (Obtener todas las entregas).
     */
    public function index()
    {
        return response()->json(Submission::all(), 200);
    }

    /**
     * Create a new submission (Registrar una entrega).
     */
    public function store(Request $request)
```

```
    {
        $request->validate([
            'user_id'       => 'required|exists:users,id',
            'assignment_id' => 'required|exists:assignments,id',
            'submitted_at'  => 'nullable|date',
            'grade'         => 'nullable|numeric|min:0|max:10',
        ]);

        $submission = Submission::create($request->all());
        return response()->json([
            'message'    => 'Submission recorded successfully',
            'submission' => $submission
        ], 201);
    }

    /**
     * Show a specific submission (Mostrar entrega).
     */
    public function show($id)
    {
        $submission = Submission::find($id);
        if (!$submission) {
            return response()->json(['message' => 'Submission not
found'], 404);
        }
        return response()->json($submission, 200);
    }
    /**
     * Update submission (e.g. grading) (Actualizar entrega).
     */
    public function update(Request $request, $id)
    {
        $submission = Submission::find($id);
        if (!$submission) {
            return response()->json(['message' => 'Submission not
found'], 404);
        }
        $request->validate([
            'grade' => 'required|numeric|min:0|max:10',
        ]);

        $submission->update($request->only(['grade']));
        return response()->json([
            'message'    => 'Grade updated successfully',
            'submission' => $submission
        ], 200);
    }
}
```

3 Definir Rutas

En routes/api.php:

```
use App\Http\Controllers\AssignmentController;
use App\Http\Controllers\SubmissionController;
```

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

    // CRUD de submissions
    Route::get('/submissions', [SubmissionController::class, 'index']);
    Route::post('/submissions', [SubmissionController::class, 'store']);
    Route::get('/submissions/{id}', [SubmissionController::class,
'show']);
    Route::put('/submissions/{id}', [SubmissionController::class,
'update']);
});
```

Controlador de Eventos de Calendario (CalendarEventController)

1 Creación

php artisan make:controller CalendarEventController --api

Editar app/Http/Controllers/CalendarEventController.php:

```
<?php

namespace App\Http\Controllers;
use App\Models\CalendarEvent;
use Illuminate\Http\Request;

class CalendarEventController extends Controller
{
    /**
     * Get all calendar events (Obtener todos los eventos).
     */
```

```
    public function index()
    {
        return response()->json(CalendarEvent::all(), 200);
    }

    /**
     * Create a new calendar event (Crear un nuevo evento).
     */
    public function store(Request $request)
    {
        $request->validate([
            'title'       => 'required|string|max:255',
            'description' => 'nullable|string',
            'start'       => 'required|date',
            'end'         => 'required|date|after_or_equal:start',
            'user_id'     => 'required|exists:users,id',
        ]);

        $event = CalendarEvent::create($request->all());
        return response()->json([
            'message' => 'Event created successfully',
            'event'   => $event
        ], 201);
    }

    /**
     * Show a specific event (Mostrar evento).
     */
    public function show($id)
    {
        $event = CalendarEvent::find($id);
        if (!$event) {
            return response()->json(['message' => 'Event not found'],
404);
        }
        return response()->json($event, 200);
    }

    /**
     * Update an event (Actualizar evento).
     */
    public function update(Request $request, $id)
    {
        $event = CalendarEvent::find($id);
        if (!$event) {
            return response()->json(['message' => 'Event not found'],
404);
        }
```

```
        $request->validate([
            'title'       => 'nullable|string|max:255',
            'description' => 'nullable|string',
            'start'       => 'nullable|date',
            'end'         => 'nullable|date|after_or_equal:start',
        ]);

        $event->update($request->all());
        return response()->json([
            'message' => 'Event updated successfully',
            'event'   => $event
        ], 200);
    }

    /**
     * Delete an event (Eliminar evento).
     */
    public function destroy($id)
    {
        $event = CalendarEvent::find($id);
        if (!$event) {
            return response()->json(['message' => 'Event not found'],
404);
        }

        $event->delete();
        return response()->json(['message' => 'Event deleted
successfully'], 200);
    }
}
```

2 Definir Rutas

```
use App\Http\Controllers\CalendarEventController;
```

```
Route::middleware('auth:sanctum')->group(function () {
    // CRUD de eventos del calendario
    Route::get('/calendar', [CalendarEventController::class, 'index']);
    Route::post('/calendar', [CalendarEventController::class, 'store']);
    Route::get('/calendar/{id}', [CalendarEventController::class,
'show']);
    Route::put('/calendar/{id}', [CalendarEventController::class,
'update']);
    Route::delete('/calendar/{id}', [CalendarEventController::class,
'destroy']);
});
```

Controlador de Mensajería Interna (MessageController)

1 Creación

php artisan make:controller MessageController --api

Editar app/Http/Controllers/MessageController.php:

```php
<?php

namespace App\Http\Controllers;

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

class MessageController extends Controller
{
    /**
     * Get messages of the authenticated user (Obtener mensajes del
usuario autenticado).
     */
    public function index(Request $request)
    {
        $userId = $request->user()->id;
        $messages = Message::where('receiver_id', $userId)
                    ->orWhere('sender_id', $userId)
                    ->orderBy('created_at', 'desc')
                    ->get();

        return response()->json($messages, 200);
    }

    /**
     * Send a new message (Enviar mensaje).
     */
    public function store(Request $request)
    {
        $request->validate([
            'receiver_id' => 'required|exists:users,id',
            'content'     => 'required|string',
        ]);

        $message = Message::create([
            'sender_id'   => $request->user()->id,
            'receiver_id' => $request->receiver_id,
            'content'     => $request->content,
        ]);

        return response()->json([
```

```
            'message' => 'Message sent successfully',
            'data'    => $message
        ], 201);
    }

    /**
     * Get conversation between current user and a specific user
(Conversación).
     */
    public function conversation(Request $request, $userId)
    {
        $authId = $request->user()->id;
        $messages = Message::where(function($query) use ($authId,
$userId) {
                            $query->where('sender_id', $authId)
                                  ->where('receiver_id', $userId);
                        })
                        ->orWhere(function($query) use ($authId, $userId)
{
                            $query->where('sender_id', $userId)
                                  ->where('receiver_id', $authId);
                        })
                        ->orderBy('created_at', 'asc')
                        ->get();

        return response()->json($messages, 200);
    }

    /**
     * Mark a message as read (Marcar mensaje como leído).
     */
    public function markAsRead($id)
    {
        $message = Message::find($id);
        if (!$message) {
            return response()->json(['message' => 'Message not found'],
404);
        }

        $message->update(['is_read' => true]);
        return response()->json(['message' => 'Message marked as read'],
200);
    }

    /**
     * Delete a message (Eliminar mensaje).
     */
    public function destroy($id)
    {
```

```
        $message = Message::find($id);
        if (!$message) {
            return response()->json(['message' => 'Message not found'],
404);
        }

        $message->delete();
        return response()->json(['message' => 'Message deleted
successfully'], 200);
    }
}
```

2 Definir Rutas

```
use App\Http\Controllers\MessageController;
```

```
Route::middleware('auth:sanctum')->group(function () {
    // Mensajería
    Route::get('/messages', [MessageController::class, 'index']);
    Route::post('/messages', [MessageController::class, 'store']);
    Route::get('/messages/conversation/{userId}',
[MessageController::class, 'conversation']);
    Route::put('/messages/{id}/read', [MessageController::class,
'markAsRead']);
    Route::delete('/messages/{id}', [MessageController::class,
'destroy']);
});
```

Controlador para Roles y Permisos (RoleController, PermissionController)

Por último, si necesitas **gestión avanzada de roles y permisos**, crea dos controladores:

a) RoleController

php artisan make:controller RoleController --api

```
<?php

namespace App\Http\Controllers;

use App\Models\Role;
use App\Models\User;
use Illuminate\Http\Request;
```

```php
class RoleController extends Controller
{
    /**
     * Get all roles (Obtener todos los roles).
     */
    public function index()
    {
        return response()->json(Role::all(), 200);
    }

    /**
     * Create a new role (Crear un nuevo rol).
     */
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|string|unique:roles,name'
        ]);

        $role = Role::create($request->only(['name']));
        return response()->json([
            'message' => 'Role created successfully',
            'role'    => $role
        ], 201);
    }

    /**
     * Assign a role to a user (Asignar rol a un usuario).
     */
    public function assignRole(Request $request, $userId)
    {
        $request->validate([
            'role_id' => 'required|exists:roles,id'
        ]);

        $user = User::findOrFail($userId);
        $user->roles()->attach($request->role_id);
        return response()->json(['message' => 'Role assigned
successfully'], 200);
    }

    /**
     * Remove a role from a user (Eliminar rol de un usuario).
     */
    public function removeRole(Request $request, $userId)
    {
        $request->validate([
            'role_id' => 'required|exists:roles,id'
        ]);
```

```
        $user = User::findOrFail($userId);
        $user->roles()->detach($request->role_id);
        return response()->json(['message' => 'Role removed
successfully'], 200);
    }
}
```

b) Rutas para Roles

```
use App\Http\Controllers\RoleController;
```

```
Route::middleware('auth:sanctum')->group(function () {
    // Roles
    Route::get('/roles', [RoleController::class, 'index']);
    Route::post('/roles', [RoleController::class, 'store']);
    Route::post('/users/{userId}/assign-role', [RoleController::class,
'assignRole']);
    Route::post('/users/{userId}/remove-role', [RoleController::class,
'removeRole']);
});
```

Si necesitas también un PermissionController, lo creas con lógica similar.

Con estos **controladores CRUD** y **rutas** en api.php, tu plataforma **Laredu** dispondrá de:

- **Cursos (CourseController)**
- **Asignaturas (SubjectController)**
- **Tareas (AssignmentController)**
- **Entregas (SubmissionController)**
- **Eventos del calendario (CalendarEventController)**
- **Mensajería (MessageController)**
- **Roles y permisos (RoleController)**

Pruebas con Postman para los Controladores y Rutas CRUD

A continuación, veremos ejemplos de **pruebas con Postman** para los controladores y rutas que hemos creado. Esto te ayudará a **validar** que tus endpoints funcionan correctamente, además de servir como **guía de uso**.

Pruebas para CourseController

1 Listar cursos

- **Method**: GET
- **URL**: http://127.0.0.1:8000/api/courses
- **Headers**:
  - Authorization: Bearer <TOKEN>
- **Body**: *(vacío)*

Respuesta esperada (HTTP 200):

```
[
  {
    "id": 1,
    "name": "Curso de Laravel",
    "description": "Aprendiendo Laravel 11",
    "created_at": "...",
    "updated_at": "..."
  },
  ...
]
```

2 Crear un curso

- **Method**: POST
- **URL**: http://127.0.0.1:8000/api/courses
- **Headers**:
  - Content-Type: application/json
  - Authorization: Bearer <TOKEN>
- **Body (JSON)**:

```
{
  "name": "Curso de React",
  "description": "React 19 desde cero"
}
```

Respuesta esperada (HTTP 201):

```
{
  "message": "Course created successfully",
  "course": {
    "id": 2,
    "name": "Curso de React",
    "description": "React 19 desde cero",
    "updated_at": "2025-02-02T10:00:00.000000Z",
    "created_at": "2025-02-02T10:00:00.000000Z"
  }
}
```

3 Mostrar un curso específico

- **Method**: GET
- **URL**: http://127.0.0.1:8000/api/courses/1
- **Headers**:
  - Authorization: Bearer <TOKEN>
- **Body**: *(vacío)*

Respuesta esperada (HTTP 200):

```
{
  "id": 1,
  "name": "Curso de Laravel",
  "description": "Aprendiendo Laravel 11",
  "created_at": "...",
  "updated_at": "..."
}
```

4 Actualizar un curso

- **Method**: PUT
- **URL**: http://127.0.0.1:8000/api/courses/1
- **Headers**:
  - Content-Type: application/json
  - Authorization: Bearer <TOKEN>
- **Body (JSON)**:

```
{
  "name": "Curso de Laravel Avanzado",
  "description": "Nivel intermedio-avanzado"
}
```

***Respuesta esperada (HTTP 200)**:*

```
{
  "message": "Course updated successfully",
  "course": {
    "id": 1,
    "name": "Curso de Laravel Avanzado",
    "description": "Nivel intermedio-avanzado",
    "updated_at": "2025-02-02T11:00:00.000000Z",
    "created_at": "..."
  }
}
```

5 Eliminar un curso

- **Method**: DELETE
- **URL**: http://127.0.0.1:8000/api/courses/1
- **Headers**:
  - Authorization: Bearer <TOKEN>
- **Body**: *(vacío)*

Respuesta esperada (HTTP 200):

```
{
  "message": "Course deleted successfully"
}
```

Pruebas para SubjectController

1 Listar asignaturas

- **Method**: GET
- **URL**: http://127.0.0.1:8000/api/subjects
- **Headers**: Authorization: Bearer <TOKEN>

Respuesta esperada (HTTP 200):

```
[
  {
    "id": 1,
    "name": "Programación Backend",
    "course_id": 1,
    "teacher_id": 5,
    "created_at": "...",
    "updated_at": "..."
  },
  ...
]
```

2 Crear una asignatura

- **Method**: POST
- **URL**: http://127.0.0.1:8000/api/subjects
- **Body (JSON)**:

```
{
  "name": "Base de Datos",
  "course_id": 2,
  "teacher_id": 5
}
```

Respuesta esperada (HTTP 201):

```
{
  "message": "Subject created successfully",
  "subject": {
    "id": 2,
    "name": "Base de Datos",
    "course_id": 2,
    "teacher_id": 5,
    "updated_at": "...",
    "created_at": "..."
  }
}
```

3 Mostrar una asignatura

- **Method**: GET
- **URL**: http://127.0.0.1:8000/api/subjects/2
- **Headers**: Authorization: Bearer <TOKEN>

4 Actualizar una asignatura

- **Method**: PUT
- **URL**: http://127.0.0.1:8000/api/subjects/2
- **Body (JSON)**:

```
{
  "name": "Base de Datos Avanzada"
}
```

5 Eliminar una asignatura

- **Method**: DELETE
- **URL**: http://127.0.0.1:8000/api/subjects/2

Pruebas para AssignmentController (Tareas)

1 Listar tareas

- **GET** http://127.0.0.1:8000/api/assignments

2 Crear tarea

- **POST** http://127.0.0.1:8000/api/assignments

```
{
  "title": "Proyecto Final",
  "description": "Crear un sistema con Laravel 11",
  "due_date": "2025-02-28 23:59:59",
  "subject_id": 2
}
```

3 Mostrar tarea

- **GET** http://127.0.0.1:8000/api/assignments/1

4 Actualizar tarea

- **PUT** http://127.0.0.1:8000/api/assignments/1

```
{
  "title": "Proyecto Actualizado",
  "description": "Nueva fecha de entrega",
  "due_date": "2025-03-05 23:59:59"
}
```

5 Eliminar tarea

- **DELETE** http://127.0.0.1:8000/api/assignments/1

Pruebas para SubmissionController (Entregas)

1 Listar entregas

- **GET** http://127.0.0.1:8000/api/submissions

2 Crear entrega

- **POST** http://127.0.0.1:8000/api/submissions

```
{
  "user_id": 3,
  "assignment_id": 1,
  "submitted_at": "2025-02-10 20:00:00",
  "grade": null
}
```

4.3 Mostrar entrega

- **GET** http://127.0.0.1:8000/api/submissions/1

4.4 Actualizar calificación

- **PUT** http://127.0.0.1:8000/api/submissions/1

```
{
  "grade": 8.5
}
```

Pruebas para CalendarEventController (Eventos)

1 Listar eventos

- **GET** http://127.0.0.1:8000/api/calendar

2 Crear evento

- **POST** http://127.0.0.1:8000/api/calendar

```
{
  "title": "Examen Parcial",
  "description": "Examen de la asignatura X",
  "start": "2025-02-20 09:00:00",
  "end": "2025-02-20 10:30:00",
  "user_id": 3
}
```

3 Mostrar evento

- **GET** http://127.0.0.1:8000/api/calendar/1

4 Actualizar evento

- **PUT** http://127.0.0.1:8000/api/calendar/1

```
{
  "title": "Examen Parcial Modificado",
  "end": "2025-02-20 11:00:00"
}
```

5 Eliminar evento

- **DELETE** http://127.0.0.1:8000/api/calendar/1

Pruebas para MessageController (Mensajería Interna)

1 Listar mensajes del usuario autenticado

- **GET** http://127.0.0.1:8000/api/messages

2 Enviar un mensaje

- **POST** http://127.0.0.1:8000/api/messages

```
{
  "receiver_id": 4,
  "content": "¡Hola, qué tal!"
}
```

3 Conversación con un usuario

- **GET** http://127.0.0.1:8000/api/messages/conversation/4

4 Marcar mensaje como leído

- **PUT** http://127.0.0.1:8000/api/messages/10/read

5 Eliminar un mensaje

- **DELETE** http://127.0.0.1:8000/api/messages/10

Pruebas para RoleController (Roles)

1 Listar roles

- **GET** http://127.0.0.1:8000/api/roles

2 Crear un rol

- **POST** http://127.0.0.1:8000/api/roles

```
{
  "name": "coordinator"
}
```

3 Asignar rol a un usuario

- **POST** http://127.0.0.1:8000/api/users/2/assign-role

```
{
  "role_id": 2
}
```

4 Remover rol de un usuario

- **POST** http://127.0.0.1:8000/api/users/2/remove-role

```
{
  "role_id": 2
}
```

Recomendaciones Generales

1. **Siempre** pasar en los Headers:
    - ***Authorization: Bearer <TOKEN>*** es el token que se recibe cuando hacemos el login
    - ***Content-Type: application/json (para peticiones POST/PUT con cuerpo JSON).***
2. **Verifica** el token en me endpoint:
    - **GET** http://127.0.0.1:8000/api/me
    - Para confirmar que estás autenticado.
3. **Observa** los códigos de estado:
    - 200 = OK
    - 201 = CREATED
    - 404 = NOT FOUND
    - 401 = UNAUTHORIZED
4. **Valida** que php artisan route:list muestre todas las rutas. Si falta algo, revisa la configuración de RouteServiceProvider.

Con estos ejemplos de **pruebas en Postman**, puedes comprobar rápidamente cada **CRUD** y la **mensajería**, así como los roles y eventos del calendario.

Entrega de primera parte

Estructura Final del Proyecto

```
laredu/
├── backend/   # Código de Laravel 11
│   ├── app/
│   ├── config/
│   ├── database/
│   ├── public/
│   ├── routes/
│   ├── ...
│   └── .env.example   # Archivo de ejemplo para configuración
├── frontend/  # Código de React 19
│   ├── src/
│   ├── public/
│   ├── ...
│   └── package.json
└── README.md  # Instrucciones de instalación y uso
```

- **laredu/** es la carpeta raíz del proyecto.
- **backend/** contiene toda la aplicación de **Laravel 11**.
- **frontend/** contiene la aplicación en **React 19**.
- **README.md** en la raíz describe cómo instalar y ejecutar ambos entornos.

Configurar Repositorio en GitHub

Crear un repositorio en GitHub:

- Ve a tu cuenta de GitHub.
- Crea un nuevo repositorio con el nombre que desees, por ejemplo, laredu.
- No inicialices el repositorio con un archivo README.md, .gitignore o licencia (a menos que sea necesario).

Inicializar Git en tu carpeta local:

- Abre tu terminal o línea de comandos.
- Navega a la carpeta del proyecto:

```
cd laredu
```

- Inicializa Git en la carpeta:

```
git init
```

Conectar el repositorio local con GitHub:

- Agrega el repositorio remoto de GitHub como origen:

```
git remote add origin
https://github.com/tu_usuario/laredu.git
```

Agregar archivos y hacer el primer commit:

- Agrega todos los archivos al área de preparación (staging):

```
git add .
```

- Realiza tu primer commit con un mensaje descriptivo:

```
git commit -m "Initial commit"
```

Subir los cambios a GitHub:

- Sube los cambios al repositorio remoto (rama main o master):

```
git push -u origin main
```

- Si estás usando una rama diferente, reemplaza main con el nombre de tu rama.

Notas adicionales:

- Asegúrate de tener Git instalado en tu sistema.
- Si es la primera vez que usas Git, configura tu nombre y correo electrónico:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu_correo@example.com"
```

Nota: Asegúrate de tener **.gitignore** configurado para **ignorar** vendor/, node_modules/, y el archivo .env real.

.env.example y No Subir .env

- En el backend de Laravel, **renombra** tu .env a .env.example (o haz copia) y **borra** cualquier credencial sensible (contraseñas, claves API).
- Dentro de .env.example deja **variables ficticias** o comentadas para que otros sepan cómo configurar su entorno.
- El archivo real .env **no** se sube al repositorio. Esto se maneja en tu .gitignore así:

```
# Laravel
/vendor
.env
```

Así, quienes clonen tu repo pueden hacer:

```
cp .env.example .env
```

y luego poner sus credenciales reales.

Base de Datos en Formato SQL

Para compartir tu base de datos, puedes **exportar** la estructura y/o datos:

```
# Con MySQL o MariaDB
mysqldump -u user -p laredu > laredu.sql
```

Luego, en tu repositorio, dentro de laredu/backend/database/, puedes colocar laredu.sql o en la raíz del proyecto. Ten en cuenta:

- Si la base de datos es grande, a veces se prefiere solo exportar la **estructura** y usar seeders.
- Con los **migradores** y **seeders** de Laravel, a menudo no hace falta el .sql, pero puede ser útil si quieres datos de ejemplo más completos.

Informe de Pruebas con Postman

Para **demostrar** que tu API funciona, puedes:

1. **Crea un informe PDF** con la capturas de pantalla de las pruebas
2. **Crear una colección** en Postman con las rutas principales (/register, /login, /courses, etc.).
3. **Exportar** la colección a un archivo .json.
4. **Incluir** dicho archivo en el repositorio (ej. docs/Laredu.postman_collection.json).

En tu README.md, indicas cómo **importar** la colección en Postman y menciona la necesidad de un **token** de Sanctum para rutas protegidas.

README con Instrucciones de Instalación

Ejemplo de un README.md en la carpeta raíz laredu/:

```
# Laredu
Plataforma escolar con **Laravel 11** (backend) y **React 19**
(frontend).
## Requisitos
- PHP ^8.2
- Composer
- Node.js & npm
- MySQL


## Instalación del Backend (Laravel 11)
1. Ir a la carpeta `backend`
2. Copiar `.env.example` a `.env` y configurar la base de datos
3. Ejecutar:
   composer install
   php artisan key:generate
   php artisan migrate --seed
   php artisan serve
```

4. El backend estará disponible en http://127.0.0.1:8000

```
Archivo SQL
```

- En backend/database/laredu.sql se encuentra el backup de la base de datos.
- En el servidor se puedes importarlo manualmente con: mysql -u user -p laredu < laredu.sql

```
Postman Collection
```

- Importar docs/Laredu.postman_collection.json en Postman para probar los endpoints.

```
Licencia
```

- MIT (ejemplo)

```
## 7. Subir Todo a GitHub
1. Asegúrate de que tu `.gitignore` **ignore** `vendor/`
(Laravel), `node_modules/` (React) y `.env`.
2. Añade `frontend/` y `backend/` al repositorio.
3. Haz commit y push:
   git add .
   git commit -m "Add backend and frontend"
   git push
En GitHub, verás tu carpeta laredu con subcarpetas backend y
frontend, junto a README.md.
```

Siguiendo estos pasos, tendrás:

1. **Un repositorio** en GitHub con la carpeta laredu/.
2. **Dos subcarpetas**: backend (Laravel 11) y frontend (React 19).
3. **Un .env.example** en el backend y un **.gitignore** para no subir credenciales ni dependencias.
4. **Un archivo SQL** (opcional) con la DB.
5. **Un README** con instrucciones de instalación, migración, y las pruebas en Postman.