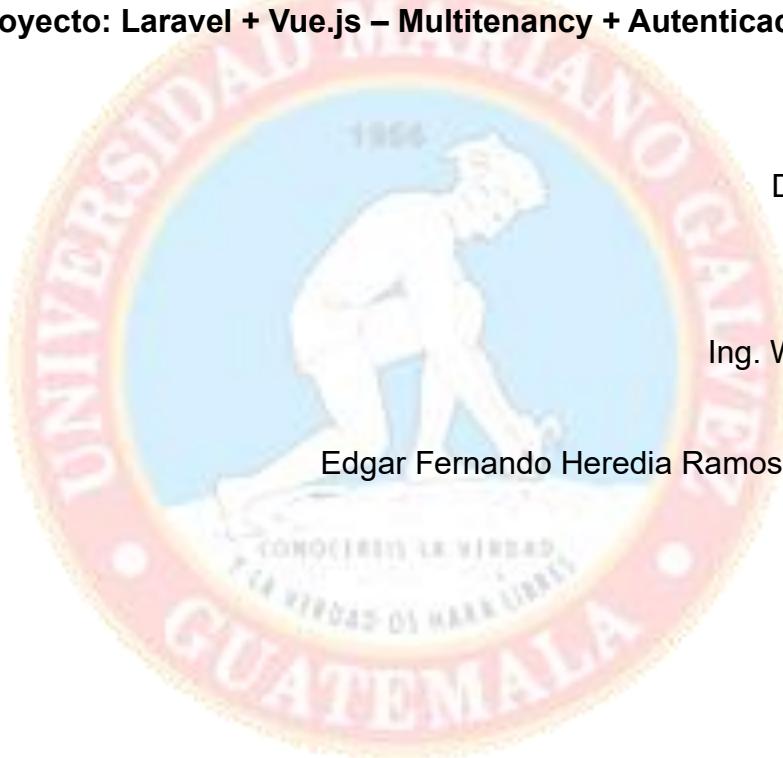


Universidad Mariano Gálvez de Guatemala
Ingeniería en Sistemas de Información
8vo. Semestre
Ciclo 2025

Tema:

Documentación – Segundo Parcial

Proyecto: Laravel + Vue.js – Multitenancy + Autenticación



Desarrollo Web

Ing. Walter Cordova

Edgar Fernando Heredia Ramos 1690-22-2199

Fecha:

14/08/2025

Dolores, Petén

INDICE

1	Introducción	3
2	Marco Teorico	4
2.1	Autenticación de Usuario 	4
2.1.1	Estrategia	4
2.1.2	Implementación	4
2.1.3	Diagrama de Componentes (PlantUML)	6
2.1.4	Diagrama de Secuencia del Login (PlantUML)	7
2.1.5	Funcionamiento del Proyecto:	8
2.2	Arquitectura Multitenant 	9
2.2.1	Concepto	9
2.2.2	Configuración en Laravel	9
2.2.3	Migraciones separadas	10
2.2.4	Diagrama de Componentes	11
2.2.5	Diagrama de Secuencia	12
2.2.6	Cómo se provisiona un nuevo tenant	12
3	Conclusiones	13

1 INTRODUCCIÓN

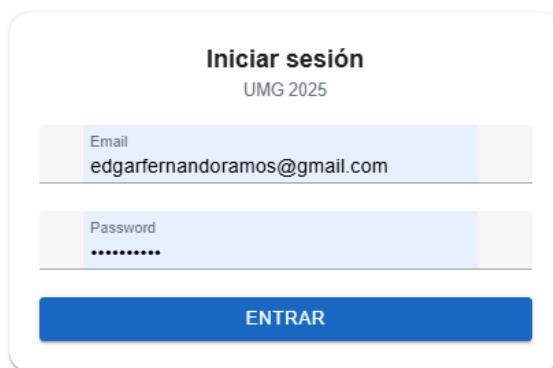
Este proyecto tiene como propósito aplicar buenas prácticas de seguridad y escalabilidad en una aplicación Laravel + Vue.js.

Se realizaron dos grandes mejoras:

1. **Autenticación de usuarios** mediante Laravel Sanctum.
2. **Implementación de arquitectura Multitenant** con bases de datos independientes por cliente (tenant).

La entrega incluye tanto la parte técnica (código y configuración) como la documentación de problemas y soluciones encontrados en el proceso.

Taller Laravel + Vue



Iniciar sesión
UMG 2025

Email
edgarfernandoramos@gmail.com

Password
.....

ENTRAR

2 MARCO TEORICO

2.1 Autenticación de Usuario

2.1.1 Estrategia

La autenticación se implementó usando **Laravel Sanctum**, ya que permite emitir tokens para peticiones API. Esto garantiza que el backend solo responda a usuarios autenticados.

- Protegimos el CRUD con auth:sanctum: si la petición no trae token válido → **401 Unauthorized**.
- El **frontend (Vue)**:
 - Tiene un **form de login** que llama a /api/login.
 - Guarda token y user en localStorage.
 - Un **interceptor Axios** agrega Authorization: Bearer <token> en **todas** las peticiones.

Si el backend responde **401**, limpia sesión y redirige a /login.

2.1.2 Implementación

1. Se instaló Sanctum y se publicaron las migraciones.
2. En el archivo **app/Models/User.php**, se agregó el trait HasApiTokens:

2.1.2.1 Login

1. El usuario envía email y password a POST /api/login.
2. El backend busca el usuario y verifica el hash de la contraseña.
3. Si es correcto, **Sanctum** crea un registro en personal_access_tokens y la API devuelve { usuario, token }.

2.1.2.2 Consumo de APIs protegidas

4. El frontend envía Authorization: Bearer <token>.
5. El middleware auth:sanctum valida el token contra la tabla personal_access_tokens.
6. Si es válido → continúa al controlador. Si no → **401**.

2.1.2.3 Logout

7. La API revoca el **token actual** (o todos) y el frontend borra sesión.

3. Se creó un controlador de autenticación:

```
public function login(Request $request)
{
    $request->validate([
        'email' => 'required|email',
        'password' => 'required|string',
    ]);

    $usuario = Usuario::where('email', $request->email)->first();

    if (! $usuario || ! Hash::check($request->password, $usuario->password)) {
        return response()->json(['message' => 'Credenciales inválidas.'], 401);
    }

    $token = $usuario->createToken('api-token')->plainTextToken;

    return response()->json([
        'message' => 'Login exitoso',
        'usuario' => $usuario,
        'token' => $token,
    ], 200);
}

3 |     'password',
4 | ];
5 | 
```

4. En routes/api.php se protegieron rutas:

```
use App\Http\Controllers\Api\UsuarioController;
use App\Http\Controllers\Api\TareaReporteController;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

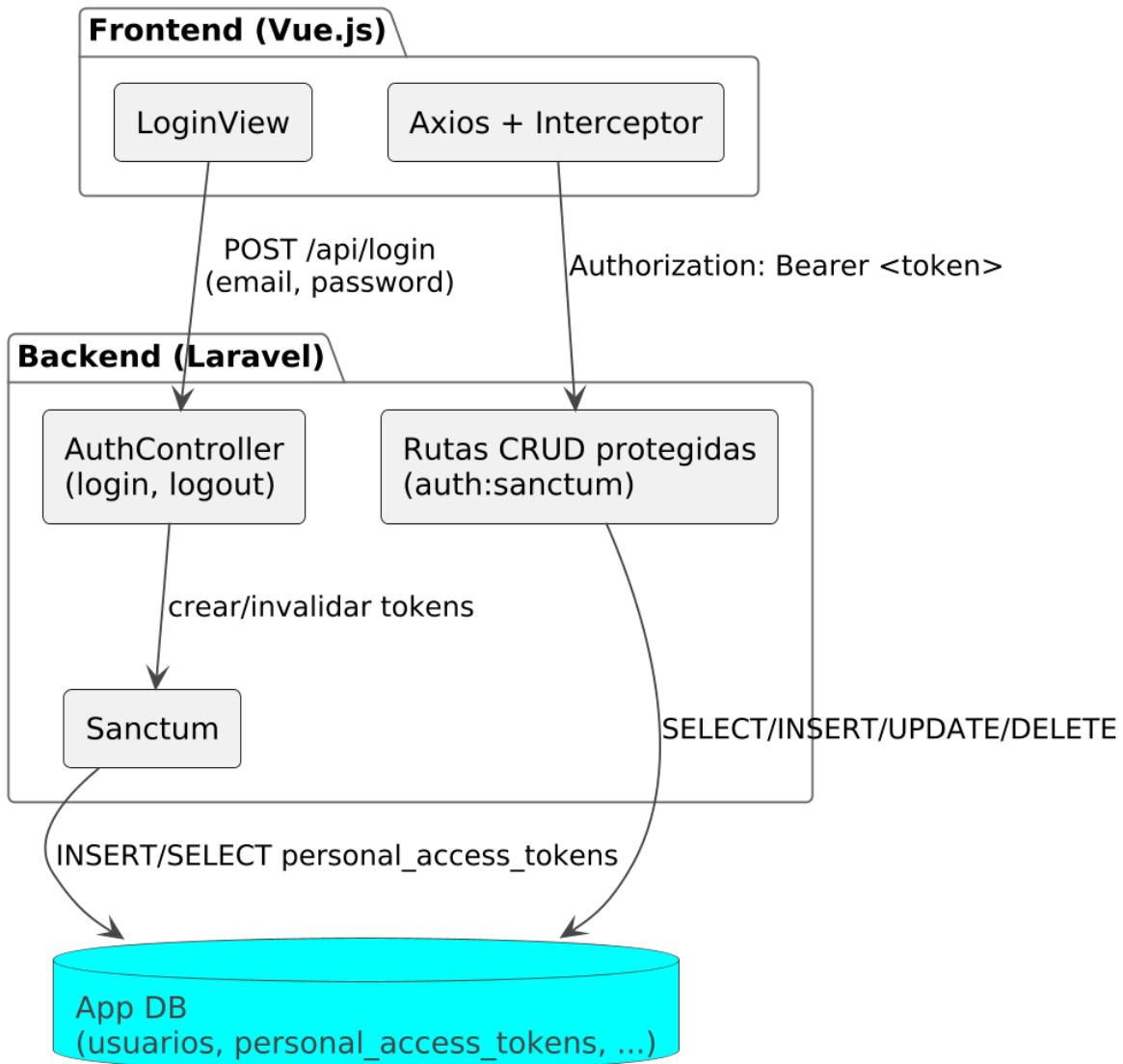
// -----
// LOGIN (público) / LOGOUT (protegido)
// -----
Route::post('/login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);

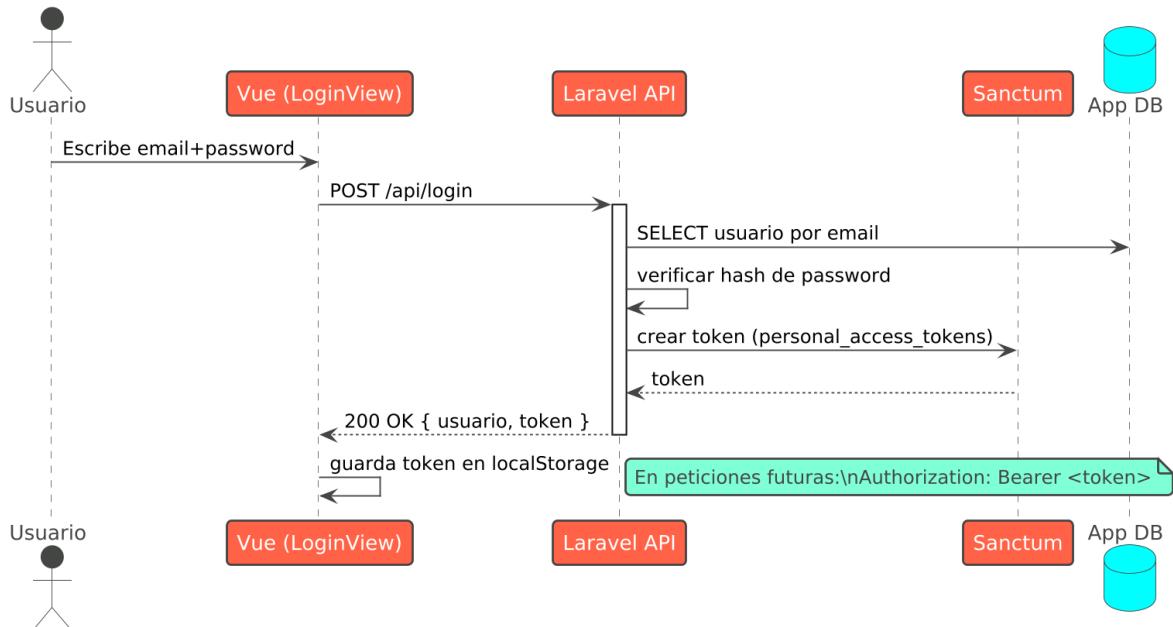
    // -----
    // USUARIOS (CRUD protegido)
    // -----
    Route::prefix('usuarios')->group(function () {
        Route::get('/listUsers', [UsuarioController::class, 'index']);
        Route::post('/addUser', [UsuarioController::class, 'store']);
        Route::get('/getUser/{id}', [UsuarioController::class, 'show']);
        Route::put('/updateUser/{id}', [UsuarioController::class, 'update']);
        Route::delete('/deleteUser/{id}', [UsuarioController::class, 'destroy']);
    });
});

// -----
// TAREAS (protegidas)
// -----
Route::get('/tareas', [TareaController::class, 'index']);
Route::post('/tareas', [TareaController::class, 'store']);
Route::get('/tareas/report', [TareaReporteController::class, 'export']);
Route::get('/tareas/report-csv', [TareaReporteController::class, 'exportCsv']);
});
```

2.1.3 Diagrama de Componentes (PlantUML)



2.1.4 Diagrama de Secuencia del Login (PlantUML)



2.1.5 Funcionamiento del Proyecto:

The screenshot shows a web browser window with the URL `empresa1.localhost:5173/usuarios`. The page title is "Taller Laravel + Vue". A sidebar on the left contains "Acciones" (Actions) and a teal button labeled "TAREAS". Below the sidebar is a search bar with the placeholder "Buscar usuarios". At the bottom of the sidebar is a pink button labeled "CERRAR SESIÓN" (Logout). A message "Sesión: Fernando (usuario)" is displayed. The main content area is titled "Usuarios" and contains a table with two rows of user data. The columns are "Nombre" (Name), "Email", "Rol" (Role), and "Creado" (Created). The first row has "Admin 1" as the name, "admin1@demo.com" as the email, "admin" as the role, and "21/09/2025, 11:33" as the creation date. The second row has "Fernando" as the name, "Fernandoempresa1@gmail.com" as the email, "usuario" as the role, and "21/09/2025, 12:29" as the creation date. Below the table are pagination controls: "Items per page:" followed by a dropdown menu set to "10", and "1-2 of 2".

Nombre	Email	Rol	Creado
Admin 1	admin1@demo.com	admin	21/09/2025, 11:33
Fernando	Fernandoempresa1@gmail.com	usuario	21/09/2025, 12:29

The screenshot shows a web browser window with the URL `empresa2.localhost:5173/usuarios`. The page title is "Taller Laravel + Vue". A sidebar on the left contains "Acciones" (Actions) and a teal button labeled "TAREAS". Below the sidebar is a search bar with the placeholder "Buscar usuarios". At the bottom of the sidebar is a pink button labeled "CERRAR SESIÓN" (Logout). A message "Sesión: Fernando (usuario)" is displayed. The main content area is titled "Usuarios" and contains a table with two rows of user data. The columns are "Nombre" (Name), "Email", "Rol" (Role), and "Creado" (Created). The first row has "Admin 2" as the name, "admin2@demo.com" as the email, "admin" as the role, and "21/09/2025, 11:36" as the creation date. The second row has "Fernando" as the name, "Fernandoempresa2@gmail.com" as the email, "usuario" as the role, and "21/09/2025, 12:32" as the creation date. Below the table are pagination controls: "Items per page:" followed by a dropdown menu set to "10", and "1-2 of 2".

Nombre	Email	Rol	Creado
Admin 2	admin2@demo.com	admin	21/09/2025, 11:36
Fernando	Fernandoempresa2@gmail.com	usuario	21/09/2025, 12:32

2.2 Arquitectura Multitenant

2.2.1 Concepto

El **multitenancy** permite que una misma aplicación atienda múltiples clientes (tenants), manteniendo **aislados** sus **datos**.

En nuestro proyecto:

- landlord_db: controla los tenants registrados.
 - empresa1_db, empresa2_db: contienen los datos de cada cliente.
- Agregamos una **BD central (“landlord”)** con la tabla tenants (name, subdomain, db_*).
- Creamos un **middleware** IdentifyTenant que corre **antes** de la autenticación:
 1. Lee el **subdominio** del host (o X-Tenant en desarrollo).
 2. Busca el tenant en landlord.tenants.
 3. **Cambia la conexión** de la app a la BD del tenant (config dinámica + DB::reconnect).

Todas las consultas (incluyendo **Sanctum**) se ejecutan en la **BD del tenant activo**: empresa1_db, empresa2_db, ... → **aislamiento real de datos y tokens**

2.2.2 Configuración en Laravel

Se modificó **config/database.php** para definir múltiples conexiones:

```
// ===== CONEXIÓN BD CENTRAL (LANDLORD) =====
'landlord' => [
    'driver' => 'mysql',
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'landlord_db'),
    'username' => env('DB_USERNAME', 'root'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([]) : []
],
```

2.2.3 Migraciones separadas

Se creó estructura:

- database/migrations/_landlord/ → tablas globales.
- database/migrations/ → migraciones de cada tenant.

Migración en base central:

```
return new class extends Migration {
    protected $connection = 'landlord';

    public function up(): void
    {
        Schema::connection($this->connection)->create('tenants', function (Blueprint $table) {
            $table->id();
            $table->string('name', 120);
            $table->string('subdomain', 100)->unique();

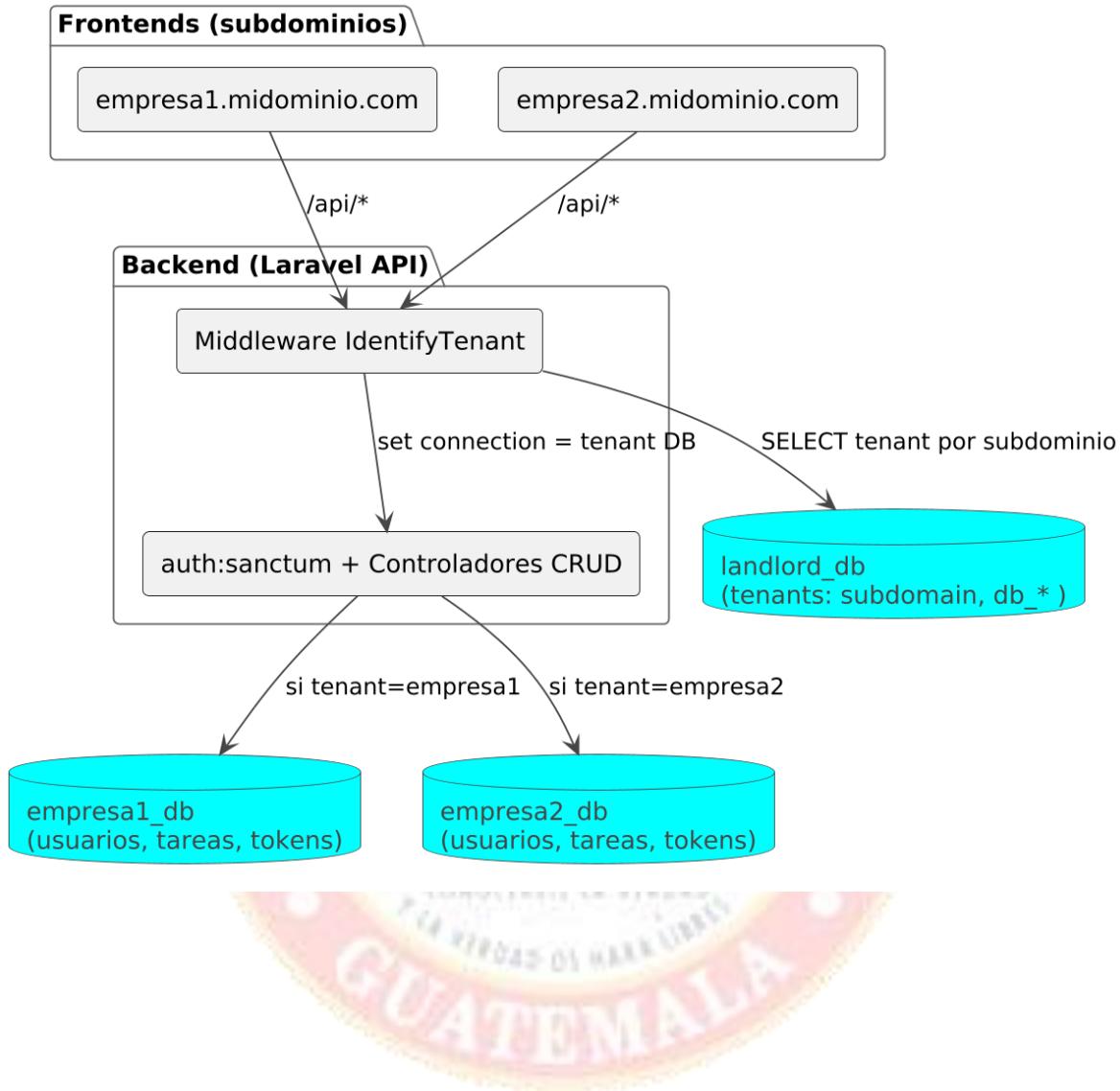
            // Datos conexión (MariaDB/MySQL por base de datos)
            $table->string('db_host')->default(env('DB_HOST', '127.0.0.1'));
            $table->string('db_port')->default(env('DB_PORT', '3306'));
            $table->string('db_database');
            $table->string('db_username')->default(env('DB_USERNAME', 'root'));
            $table->string('db_password')->default(env('DB_PASSWORD', ''));

            $table->string('db_schema')->nullable();
            $table->timestamps();
        });
    }

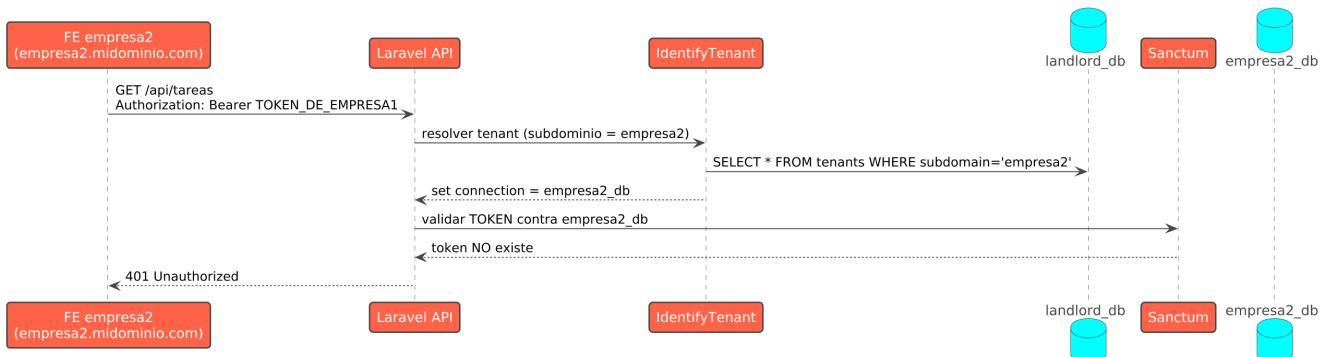
    public function down(): void
    {
        Schema::connection($this->connection)->dropIfExists('tenants');
    }
};
```

1. Llega una petición a <https://empresa1.midominio.com/api/....>
2. IdentifyTenant toma **empresa1**, consulta landlord.tenants, y setea la conexión a empresa1_db.
3. Luego corre auth:sanctum **contra esa BD del tenant** (donde están sus personal_access_tokens).
4. El controlador del CRUD trabaja sobre las tablas del **tenant actual**.
5. Si intentas usar un token de empresa1 en empresa2 → no existe → **401**.

2.2.4 Diagrama de Componentes



2.2.5 Diagrama de Secuencia



2.2.6 Cómo se provisiona un nuevo tenant

1. **Crear BD empresaN_db.**
2. **Insertar la fila en landlord.tenants con subdomain y credenciales de esa BD.**
3. php artisan tenant:migrate para crear tablas del app en empresaN_db.
4. Crear un **admin** en esa BD (Tinker o endpoint temporal).
5. Apuntar **DNS/subdominio** empresaN.midominio.com al mismo backend (Nginx/Apache/ALB).

3 CONCLUSIONES

- Se logró integrar un **sistema de autenticación robusto** con Sanctum.
- Se implementó **multitenancy con bases de datos separadas**, asegurando el aislamiento.
- El proyecto ahora es **seguro, escalable y listo para producción**.
- La experiencia en el despliegue permitió reforzar conocimientos en **servidores Linux, PHP, Laravel y Vue.js**.



CARNET: 1690-22-2199

NOMBRE: EDGAR FERNANDO HEREDIA RAMOS

SECCION: REMOTA "5"

LINK REPOSITORIOS:

BACKEND:

https://github.com/Fernando-20040/Backend_Taller.git

FRONTEND:

https://github.com/Fernando-20040/Frontend_Taller.git

VIDEO EXPLICATIVO:

<https://drive.google.com/drive/folders/1kgb08bSkppUTotvhsvdNuECRpQn82Ejk?usp=sharing>

