🌐 GUÍA COMPLETA: CONECTAR API EXTERNA EN LARAVEL

🎯 OBJETIVO

Implementar una conexión completa a una API externa en Laravel, incluyendo:

- Configuración de credenciales
- Servicios para manejar las llamadas
- Controladores para exponer endpoints
- Manejo de errores y respuestas
- Caché para optimizar rendimiento

📁 ESTRUCTURA DE CARPETAS Y ARCHIVOS

```
app/
├── Http/
│   ├── Controllers/
│   │   └── ApiController.php          # Controlador principal
│   └── Requests/
│       └── ApiRequest.php             # Validación de requests
├── Services/
│   ├── ExternalApiService.php         # Servicio principal
│   └── ApiResponseService.php         # Manejo de respuestas
├── Models/
│   └── ApiLog.php                     # Modelo para logs (opcional)
└── Exceptions/
    └── ApiException.php               # Excepciones personalizadas

config/
└── services.php                       # Configuración de APIs

routes/
└── api.php                           # Rutas de la API

database/
└── migrations/
    └── create_api_logs_table.php     # Migración para logs (opcional)

storage/
└── logs/
    └── api.log                       # Logs de la API externa
```

📋 PASO 1: CONFIGURACIÓN INICIAL

1.1 Configurar credenciales

Archivo: `config/services.php`

```php
<?php

return [
    // ... otras configuraciones

    'external_api' => [
        'base_url' => env('EXTERNAL_API_BASE_URL', 'https://api.ejemplo.com'),
        'api_key' => env('EXTERNAL_API_KEY'),
        'timeout' => env('EXTERNAL_API_TIMEOUT', 30),
        'retry_attempts' => env('EXTERNAL_API_RETRY_ATTEMPTS', 3),
    ],
];
```

1.2 Variables de entorno

Archivo: `.env`

```
EXTERNAL_API_BASE_URL=https://api.ejemplo.com
EXTERNAL_API_KEY=tu_api_key_aqui
EXTERNAL_API_TIMEOUT=30
EXTERNAL_API_RETRY_ATTEMPTS=3
```

🔧 PASO 2: CREAR SERVICIO PRINCIPAL

2.1 Servicio para manejar la API externa

Archivo: `app/Services/ExternalApiService.php`

```php
<?php

namespace App\Services;

use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Log;
use App\Exceptions\ApiException;

class ExternalApiService
{
    private $baseUrl;
    private $apiKey;
    private $timeout;
    private $retryAttempts;
```

```
    public function __construct()
    {
        $this->baseUrl = config('services.external_api.base_url');
        $this->apiKey = config('services.external_api.api_key');
        $this->timeout = config('services.external_api.timeout');
        $this->retryAttempts = config('services.external_api.retry_attempts');
    }

    /**
     * Realizar petición GET a la API externa
     */
    public function get(string $endpoint, array $params = []): array
    {
        return $this->makeRequest('GET', $endpoint, $params);
    }

    /**
     * Realizar petición POST a la API externa
     */
    public function post(string $endpoint, array $data = []): array
    {
        return $this->makeRequest('POST', $endpoint, [], $data);
    }

    /**
     * Realizar petición PUT a la API externa
     */
    public function put(string $endpoint, array $data = []): array
    {
        return $this->makeRequest('PUT', $endpoint, [], $data);
    }

    /**
     * Realizar petición DELETE a la API externa
     */
    public function delete(string $endpoint): array
    {
        return $this->makeRequest('DELETE', $endpoint);
    }

    /**
     * Método principal para realizar peticiones
     */
    private function makeRequest(string $method, string $endpoint, array $params =
[], array $data = []): array
    {
        $url = $this->baseUrl . '/' . ltrim($endpoint, '/');

        $attempt = 0;
        do {
            try {
                $response = Http::timeout($this->timeout)
                    ->withHeaders([
                        'Authorization' => 'Bearer ' . $this->apiKey,
```

```
                        'Accept' => 'application/json',
                        'Content-Type' => 'application/json',
                    ])
                    ->when($method === 'GET', function ($request) use ($params) {
                        return $request->withQueryParameters($params);
                    })
                    ->when(in_array($method, ['POST', 'PUT', 'PATCH']), function
($request) use ($data) {
                        return $request->withBody(json_encode($data),
'application/json');
                    })
                    ->$method($url);

                // Log de la petición
                $this->logRequest($method, $url, $params, $data, $response);

                if ($response->successful()) {
                    return $response->json();
                }

                // Manejar errores HTTP
                $this->handleHttpError($response);

            } catch (\Exception $e) {
                $attempt++;

                if ($attempt >= $this->retryAttempts) {
                    throw new ApiException(
                        "Error en petición a API externa después de {$this-
>retryAttempts} intentos: " . $e->getMessage(),
                        500
                    );
                }

                // Esperar antes de reintentar (backoff exponencial)
                sleep(pow(2, $attempt));
            }
        } while ($attempt < $this->retryAttempts);

        throw new ApiException('Error inesperado en la API externa', 500);
    }

    /**
     * Manejar errores HTTP
     */
    private function handleHttpError($response): void
    {
        $statusCode = $response->status();
        $body = $response->body();

        switch ($statusCode) {
            case 401:
                throw new ApiException('No autorizado - Verificar API key', 401);
            case 403:
```

```
                throw new ApiException('Acceso prohibido', 403);
            case 404:
                throw new ApiException('Recurso no encontrado', 404);
            case 422:
                throw new ApiException('Datos inválidos: ' . $body, 422);
            case 429:
                throw new ApiException('Límite de peticiones excedido', 429);
            case 500:
                throw new ApiException('Error interno del servidor externo', 500);
            default:
                throw new ApiException("Error HTTP {$statusCode}: {$body}",
$statusCode);
        }
    }

    /**
     * Log de peticiones y respuestas
     */
    private function logRequest(string $method, string $url, array $params, array
$data, $response): void
    {
        Log::channel('api')->info('API Externa - Petición', [
            'method' => $method,
            'url' => $url,
            'params' => $params,
            'data' => $data,
            'status' => $response->status(),
            'response_size' => strlen($response->body()),
        ]);
    }
}
```

🎮 PASO 3: CREAR CONTROLADOR

3.1 Controlador principal

Archivo: app/Http/Controllers/ApiController.php

```
<?php

namespace App\Http\Controllers;

use App\Services\ExternalApiService;
use App\Http\Requests\ApiRequest;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Cache;

class ApiController extends Controller
{
    private $apiService;
```

```php
    public function __construct(ExternalApiService $apiService)
    {
        $this->apiService = $apiService;
    }

    /**
     * Obtener datos de la API externa
     */
    public function getData(ApiRequest $request): JsonResponse
    {
        try {
            $params = $request->validated();

            // Usar caché para optimizar rendimiento
            $cacheKey = 'external_api_data_' . md5(json_encode($params));

            $data = Cache::remember($cacheKey, 300, function () use ($params) {
                return $this->apiService->get('data', $params);
            });

            return response()->json([
                'success' => true,
                'data' => $data,
                'message' => 'Datos obtenidos correctamente'
            ]);

        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => $e->getMessage()
            ], $e->getCode() ?: 500);
        }
    }

    /**
     * Crear datos en la API externa
     */
    public function createData(ApiRequest $request): JsonResponse
    {
        try {
            $data = $request->validated();

            $result = $this->apiService->post('data', $data);

            // Limpiar caché relacionada
            Cache::forget('external_api_data_*');

            return response()->json([
                'success' => true,
                'data' => $result,
                'message' => 'Datos creados correctamente'
            ], 201);
```

```
        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => $e->getMessage()
            ], $e->getCode() ?: 500);
        }
    }

    /**
     * Actualizar datos en la API externa
     */
    public function updateData(ApiRequest $request, $id): JsonResponse
    {
        try {
            $data = $request->validated();

            $result = $this->apiService->put("data/{$id}", $data);

            // Limpiar caché relacionada
            Cache::forget('external_api_data_*');

            return response()->json([
                'success' => true,
                'data' => $result,
                'message' => 'Datos actualizados correctamente'
            ]);

        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => $e->getMessage()
            ], $e->getCode() ?: 500);
        }
    }

    /**
     * Eliminar datos de la API externa
     */
    public function deleteData($id): JsonResponse
    {
        try {
            $this->apiService->delete("data/{$id}");

            // Limpiar caché relacionada
            Cache::forget('external_api_data_*');

            return response()->json([
                'success' => true,
                'message' => 'Datos eliminados correctamente'
            ]);

        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
```

```
                'message' => $e->getMessage()
            ], $e->getCode() ?: 500);
        }
    }
}
```

☑ PASO 4: CREAR VALIDACIÓN DE REQUESTS

4.1 Request personalizado

Archivo: app/Http/Requests/ApiRequest.php

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class ApiRequest extends FormRequest
{
    /**
     * Determinar si el usuario está autorizado
     */
    public function authorize(): bool
    {
        return true; // O implementar lógica de autorización
    }

    /**
     * Reglas de validación
     */
    public function rules(): array
    {
        $method = $this->method();

        $rules = [
            'name' => 'sometimes|string|max:255',
            'email' => 'sometimes|email|max:255',
            'description' => 'sometimes|string|max:1000',
        ];

        // Reglas específicas según el método HTTP
        switch ($method) {
            case 'POST':
                $rules['name'] = 'required|string|max:255';
                $rules['email'] = 'required|email|max:255';
                break;

            case 'PUT':
            case 'PATCH':
```

```
                $rules['name'] = 'sometimes|required|string|max:255';
                $rules['email'] = 'sometimes|required|email|max:255';
                break;
        }

        return $rules;
    }

    /**
     * Mensajes de error personalizados
     */
    public function messages(): array
    {
        return [
            'name.required' => 'El nombre es obligatorio',
            'name.string' => 'El nombre debe ser texto',
            'name.max' => 'El nombre no puede exceder 255 caracteres',
            'email.required' => 'El email es obligatorio',
            'email.email' => 'El email debe tener un formato válido',
            'email.max' => 'El email no puede exceder 255 caracteres',
            'description.string' => 'La descripción debe ser texto',
            'description.max' => 'La descripción no puede exceder 1000
caracteres',
        ];
    }
}
```

🚨 PASO 5: CREAR EXCEPCIONES PERSONALIZADAS

5.1 Excepción para errores de API

Archivo: `app/Exceptions/ApiException.php`

```
<?php

namespace App\Exceptions;

use Exception;

class ApiException extends Exception
{
    protected $context;

    public function __construct(string $message = "", int $code = 0, array
$context = [])
    {
        parent::__construct($message, $code);
        $this->context = $context;
    }
```

```
    /**
     * Obtener contexto adicional del error
     */
    public function getContext(): array
    {
        return $this->context;
    }

    /**
     * Reportar el error
     */
    public function report(): void
    {
        \Log::error('API Exception: ' . $this->getMessage(), [
            'code' => $this->getCode(),
            'file' => $this->getFile(),
            'line' => $this->getLine(),
            'context' => $this->context,
        ]);
    }
}
```

🛣️ PASO 6: CONFIGURAR RUTAS

6.1 Rutas de la API

Archivo: `routes/api.php`

```
<?php

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

// Rutas para la API externa
Route::prefix('external-api')->group(function () {
    // Obtener datos
    Route::get('/data', [ApiController::class, 'getData']);

    // Crear datos
    Route::post('/data', [ApiController::class, 'createData']);

    // Actualizar datos
    Route::put('/data/{id}', [ApiController::class, 'updateData']);
    Route::patch('/data/{id}', [ApiController::class, 'updateData']);

    // Eliminar datos
    Route::delete('/data/{id}', [ApiController::class, 'deleteData']);
});

// Ruta de prueba
```

```php
Route::get('/test-external-api', function () {
    return response()->json([
        'message' => 'API externa configurada correctamente',
        'timestamp' => now(),
    ]);
});
```

PASO 7: CONFIGURAR LOGS

7.1 Configuración de logs

Archivo: `config/logging.php`

```php
'channels' => [
    // ... otros canales

    'api' => [
        'driver' => 'daily',
        'path' => storage_path('logs/api.log'),
        'level' => env('LOG_LEVEL', 'debug'),
        'days' => 14,
    ],
],
```

PASO 8: MODELO PARA LOGS (OPCIONAL)

8.1 Modelo ApiLog

Archivo: `app/Models/ApiLog.php`

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class ApiLog extends Model
{
    protected $fillable = [
        'method',
        'url',
        'request_data',
        'response_data',
        'status_code',
        'response_time',
        'error_message',
    ];
```

```
    protected $casts = [
        'request_data' => 'array',
        'response_data' => 'array',
        'response_time' => 'float',
    ];
}
```

8.2 Migración para logs

Archivo: database/migrations/create_api_logs_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('api_logs', function (Blueprint $table) {
            $table->id();
            $table->string('method');
            $table->text('url');
            $table->json('request_data')->nullable();
            $table->json('response_data')->nullable();
            $table->integer('status_code');
            $table->float('response_time');
            $table->text('error_message')->nullable();
            $table->timestamps();
        });
    }

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

🔧 FUNCIONES CLAVE

Configuración de servicios

```php
// Obtener configuración
$baseUrl = config('services.external_api.base_url');
$apiKey = config('services.external_api.api_key');
```

Realizar peticiones HTTP

```
// GET
$data = $apiService->get('endpoint', ['param' => 'value']);

// POST
$result = $apiService->post('endpoint', ['data' => 'value']);

// PUT
$result = $apiService->put('endpoint/1', ['data' => 'value']);

// DELETE
$result = $apiService->delete('endpoint/1');
```

Manejo de caché

```
// Guardar en caché
Cache::remember('key', 300, function () {
    return $apiService->get('data');
});

// Limpiar caché
Cache::forget('key');
Cache::flush();
```

Logs

```
// Log de información
Log::channel('api')->info('Mensaje', ['data' => $data]);

// Log de error
Log::channel('api')->error('Error', ['error' => $e->getMessage()]);
```

PROBLEMAS COMUNES Y SOLUCIONES

1. Timeout en peticiones

Problema: Las peticiones tardan mucho o fallan por timeout **Solución:** Ajustar el timeout en la configuración y implementar reintentos

2. Errores de autenticación

Problema: Error 401 - No autorizado **Solución:** Verificar que la API key esté correcta y no haya expirado

3. Límite de peticiones

Problema: Error 429 - Too Many Requests **Solución:** Implementar rate limiting y caché

4. Errores de red

Problema: Errores de conexión **Solución:** Implementar reintentos con backoff exponencial

5. Respuestas inconsistentes

Problema: La API externa devuelve formatos diferentes **Solución:** Implementar validación y normalización de respuestas

---

☑ CHECKLIST FINAL

- ☐ Configuración en `config/services.php`
- ☐ Variables de entorno en `.env`
- ☐ Servicio `ExternalApiService` creado
- ☐ Controlador `ApiController` implementado
- ☐ Request `ApiRequest` con validación
- ☐ Excepción `ApiException` personalizada
- ☐ Rutas configuradas en `routes/api.php`
- ☐ Logs configurados en `config/logging.php`
- ☐ Modelo y migración para logs (opcional)
- ☐ Pruebas realizadas con diferentes endpoints
- ☐ Manejo de errores implementado
- ☐ Caché configurado para optimización

💡 CONSEJOS ADICIONALES

- **Usar caché** para mejorar el rendimiento
- **Implementar rate limiting** para evitar sobrecargar la API externa
- **Logs detallados** para debugging
- **Validación robusta** de requests y responses
- **Manejo de errores** con mensajes claros
- **Documentación** de los endpoints disponibles
- **Tests unitarios** para el servicio
- **Monitoreo** de la salud de la API externa

🎉 ¡CONEXIÓN A API EXTERNA COMPLETADA!

Ahora tu aplicación Laravel tiene una conexión completa y robusta a una API externa con manejo de errores, caché, logs y validación.

📄 EXPORTAR A PDF

Opción 1: Con Pandoc

```
pandoc guia_api_externa_laravel.md -o guia_api_externa_laravel.pdf
```

Opción 2: Herramientas online

- **StackEdit** (stackedit.io)
- **Dillinger** (dillinger.io)
- **Markdown to PDF** (markdowntopdf.com)

Opción 3: Editores de código

- **VS Code** con extensión "Markdown PDF"
- **Typora** (editor Markdown con exportación a PDF)
- **Obsidian** (con plugin de exportación)