📝 Logging with Serilog — محاضرة عملية

> **كيفية تسجيل أحداث التطبيق بشكل احترافي وقابل للـ Analysis**

📄 جدول المحتويات

🎯 مقدمة خفيفة

يعني إيه Logging؟

Logging = تسجيل أحداث التطبيق في ملف أو مكان معين لتحليلها لاحقاً.

```
بدون Logging:
App: "شغال" → حصلت مشكلة → أنت: "شنو اللي حصل؟" 😱

مع Logging:
App: "شغال" → حصلت مشكلة → بيشتغل logs، شوف الـ trace، عرفت المشكلة 🎯
```

ليه Serilog؟

Laravel بتاعه فيها لـ Logging، بس Serilog:

- ✅ Structured Logging (بيانات منظمة، easy to parse)
- ✅ Multiple Sinks (احفظ في ملف + database + cloud)
- ✅ Rich Context (معلومات سياقية قيّمة)
- ✅ Performance (سريع جداً)

🧠 الأساسيات اللي لازم تعرفها

1) Log Levels

المستوى	اللون	المعنى	متى تستخدمه
DEBUG	أزرق	معلومات تفصيلية جداً	في التطوير فقط
INFO	أخضر	حدث عادي	"User logged in"

المستوى	اللون	المعنى	متى تستخدمه
WARNING	برتقالي	حذر	"Disk space low"
ERROR	أحمر	خطأ	Exception حصلت
CRITICAL	أحمر غامق	حرج جداً	Database down

```
\Log::debug('مجرد معلومة');
\Log::info('User logged in');
\Log::warning('Disk space running out');
\Log::error('Payment failed');
\Log::critical('Database connection lost');
```

2) Structured Logging (الفرق)

Normal Logging:

```
2026-01-21 10:30:00 User logged in
// واضح context صعب تحليلها... ما فيش
```

Structured Logging (Serilog):

```
{
  "timestamp": "2026-01-21T10:30:00Z",
  "level": "INFO",
  "message": "User logged in",
  "user_id": 42,
  "ip_address": "192.168.1.1",
  "device": "iPhone",
  "duration_ms": 250
}
// سهل تحليلها + exportable to analytics tools
```

3) Context vs Message

```
// ❌ بدون context
\Log::info('Payment processed');

// ✅ مع context (الـــ structured data)
\Log::info('Payment processed', [
    'user_id' => $user->id,
    'amount' => 100,
    'currency' => 'USD',
    'payment_method' => 'card',
```

```
    'processing_time_ms' => 450,
]);
```

💻 Serilog في Laravel

التطبيق العملي: Payment Logging System

الخطوة 1: Installation

```
# Serilog بتاعة .NET مش PHP
# Laravel uses Monolog, لكن نقدر نستخدم similar approach
# الـ equivalent في Laravel هو تحسين الـ Logging
composer require monolog/monolog
```

الخطوة 2: تحسين الـ Logging Config

```
// config/logging.php
return [
    'default' => env('LOG_CHANNEL', 'stack'),

    'channels' => [
        'stack' => [
            'driver' => 'stack',
            'channels' => ['single', 'daily', 'slack'],
            'ignore_exceptions' => false,
        ],

        'single' => [
            'driver' => 'single',
            'path' => storage_path('logs/laravel.log'),
            'level' => env('LOG_LEVEL', 'debug'),
        ],

        'daily' => [
            'driver' => 'daily',
            'path' => storage_path('logs/laravel.log'),
            'level' => env('LOG_LEVEL', 'debug'),
            'days' => 14, // احفظ آخر 14 يوم
        ],

        'payments' => [
            'driver' => 'daily',
            'path' => storage_path('logs/payments.log'),
            'level' => 'info',
            'days' => 30,
        ],
```

```
        'slack' => [
            'driver' => 'slack',
            'url' => env('LOG_SLACK_WEBHOOK_URL'),
            'level' => 'critical', // بس الـ critical errors
        ],

        'database' => [
            'driver' => 'custom',
            'via' => \App\Logging\DatabaseLogHandler::class,
        ],
    ],
];
```

الخطوة 3: Custom Log Handler (Optional)

```
// app/Logging/DatabaseLogHandler.php
namespace App\Logging;

use Monolog\Handler\AbstractProcessingHandler;
use Monolog\LogRecord;
use App\Models\ApplicationLog;

class DatabaseLogHandler extends AbstractProcessingHandler {
    protected function write(LogRecord $record): void {
        ApplicationLog::create([
            'level' => $record->getLevelName(),
            'message' => $record->getMessage(),
            'context' => json_encode($record->getExtra()),
            'exception' => $record->getFormatted(),
            'created_at' => now(),
        ]);
    }
}
```

```
// app/Models/ApplicationLog.php
namespace App\Models;

class ApplicationLog extends Model {
    protected $fillable = ['level', 'message', 'context', 'exception'];
    public $timestamps = false;
}
```

الخطوة 4: Helper Function للـ Structured Logging

```
// app/Helpers/LogHelper.php (أو حط في AppServiceProvider)
namespace App\Helpers;
```

```php
class LogHelper {
    public static function logPayment(array $context) {
        \Log::channel('payments')->info('Payment processed', $context);
    }

    public static function logApiCall(array $context) {
        \Log::info('API call', array_merge([
            'timestamp' => now(),
            'endpoint' => request()->path(),
            'method' => request()->method(),
            'ip_address' => request()->ip(),
        ], $context));
    }

    public static function logError(\Exception $e, array $context = []) {
        \Log::error('Exception occurred', array_merge([
            'exception_class' => get_class($e),
            'exception_message' => $e->getMessage(),
            'exception_file' => $e->getFile(),
            'exception_line' => $e->getLine(),
            'stack_trace' => $e->getTraceAsString(),
        ], $context));
    }
}
```

الخطوة 5: عملي في Payment Service

```php
// app/Services/PaymentService.php
namespace App\Services;

use App\Helpers\LogHelper;

class PaymentService {
    public function processPayment($user, $amount) {
        $startTime = microtime(true);

        try {
            // قبل ما نبدأ
            \Log::channel('payments')->info('Payment starting', [
                'user_id' => $user->id,
                'user_email' => $user->email,
                'amount' => $amount,
                'currency' => 'USD',
                'payment_method' => 'card',
            ]);

            // المعالجة
            $result = $this->chargeCard($user, $amount);

            // النجاح
```

```
            $duration = microtime(true) - $startTime;
            LogHelper::logPayment([
                'user_id' => $user->id,
                'amount' => $amount,
                'status' => 'success',
                'transaction_id' => $result['id'],
                'processing_time_ms' => round($duration * 1000),
            ]);

            return $result;

        } catch (\Exception $e) {
            // الفشل
            $duration = microtime(true) - $startTime;
            LogHelper::logError($e, [
                'user_id' => $user->id,
                'amount' => $amount,
                'payment_status' => 'failed',
                'processing_time_ms' => round($duration * 1000),
            ]);

            throw $e;
        }
    }

    private function chargeCard($user, $amount) {
        // Stripe API call
        try {
            $charge = \Stripe\Charge::create([
                'amount' => $amount * 100,
                'currency' => 'usd',
                'source' => $user->stripe_token,
            ]);

            return ['id' => $charge->id, 'status' => 'succeeded'];

        } catch (\Stripe\Exception\CardException $e) {
            \Log::warning('Card declined', [
                'user_id' => $user->id,
                'error' => $e->getMessage(),
            ]);
            throw $e;
        }
    }
}
```

الخطوة 6: أمثلة Logging

```
// في أي حتة في الكود

// ✅ معلومة عادية
```

```
\Log::info('User created', ['user_id' => $user->id, 'email' => $user->email]);

// ✅ تحذير
\Log::warning('Slow database query', [
    'query' => 'SELECT * FROM products...',
    'duration_ms' => 5000,
]);

// ✅ خطأ
try {
    $this->doSomething();
} catch (\Exception $e) {
    \Log::error('Something went wrong', [
        'exception' => $e->getMessage(),
        'file' => $e->getFile(),
        'line' => $e->getLine(),
    ]);
}

// ✅ معلومة تفصيلية (بس في التطوير)
\Log::debug('User preferences loaded', [
    'user_id' => $user->id,
    'preferences' => $user->preferences,
]);
```

الخطوة 7: قراءة الـ Logs

```
# شوف آخر 50 سطر
tail -50 storage/logs/laravel.log

# شوف الـ errors بتاعة اليوم
grep "ERROR" storage/logs/laravel-2026-01-21.log

# Real-time monitoring
tail -f storage/logs/laravel.log

# Count entries by level
grep -c "ERROR" storage/logs/laravel.log
```

⚠ الأخطاء الشائعة

1) عدم تسجيل السياق

```
// ❌ سيء: ما فيه context
\Log::info('User logged in');

// ✅ كويس: فيه context
\Log::info('User logged in', [
```

```
    'user_id' => $user->id,
    'ip_address' => request()->ip(),
    'browser' => request()->userAgent(),
]);
```

2) تسجيل Sensitive Data

```
// ❌ خطر: password في الـ log!
\Log::info('User login attempt', [
    'email' => $request->email,
    'password' => $request->password, // ❌❌❌
]);

// ✅ آمن
\Log::info('User login attempt', [
    'email' => $request->email,
    // أبداً password لا تسجل الـ
]);
```

3) Log كتير من غير selector

```
// ❌ إفلاس الـ disk space
if ($debug) {
    \Log::debug('Every single thing', $everythingInMemory);
}

// ✅ اللي مهم فقط log
if ($shouldLog) {
    \Log::debug('Important', $importantData);
}
```

4) نسيان أن الـ logs متراكمة

```
// في config/logging.php

// ❌ ما فيه rotation
'single' => [
    'driver' => 'single',
    'path' => storage_path('logs/laravel.log'),
],

// ✅ مع rotation (يومياً)
'daily' => [
    'driver' => 'daily',
    'path' => storage_path('logs/laravel.log'),
```

```
    'days' => 14, // احذف بعد 14 يوم
],
```

Senior Corner

1) Structured Logging ل Analytics

```
// Log بشكل منظم لتحليل البيانات
\Log::channel('analytics')->info('user_action', [
    'action' => 'purchase',
    'user_id' => $user->id,
    'product_id' => $product->id,
    'amount' => $amount,
    'timestamp' => now(),
    'country' => $user->country,
    'device_type' => $request->header('Device-Type'),
]);

// logs بعدين تقدر تستخدم اداة لتحليل الـ
// (مثل ELK Stack, Splunk, DataDog)
```

2) Context Processor (Automatic Context)

```
// app/Middleware/LogContext.php
namespace App\Middleware;

class LogContext {
    public function handle($request, \Closure $next) {
        // حط context اللي يتكرر في كل request
        \Log::pushProcessor(function ($record) use ($request) {
            $record['extra'] = array_merge($record['extra'] ?? [], [
                'request_id' => $request->header('X-Request-ID',
\Illuminate\Support\Str::uuid()),
                'user_id' => auth()->id(),
                'ip_address' => $request->ip(),
                'method' => $request->method(),
                'path' => $request->path(),
            ]);
            return $record;
        });

        return $next($request);
    }
}

// في kernel.php
protected $middleware = [
    // ...
```

```
    \App\Middleware\LogContext::class,
];
```

3) Performance Monitoring

```
class PerformanceLogger {
    private $startTime;

    public function __construct() {
        $this->startTime = microtime(true);
    }

    public function logPageLoad($page) {
        $duration = (microtime(true) - $this->startTime) * 1000;

        $level = match(true) {
            $duration < 100 => 'info',
            $duration < 500 => 'warning',
            default => 'error',
        };

        \Log::log($level, 'Page load', [
            'page' => $page,
            'duration_ms' => round($duration),
            'memory_mb' => round(memory_get_usage() / 1024 / 1024),
        ]);
    }
}
```

4) Log Rotation + Cleanup

```
# في الـــ cron job (schedule في Laravel)
// app/Console/Kernel.php

protected function schedule(Schedule $schedule) {
    $schedule->command('logs:clean')->daily();
}
```

```
// app/Console/Commands/LogsClean.php
class LogsClean extends Command {
    public function handle() {
        // احذف logs قديمة
        $files = glob(storage_path('logs/*.log'));

        foreach ($files as $file) {
            if (filemtime($file) < strtotime('-30 days')) {
                unlink($file);
```

```
            }
        }
    }
}
```

5) Testing مع Logs

```php
// tests/Feature/PaymentTest.php

public function test_payment_logs_correctly() {
    \Log::spy();

    $this->post('/api/pay', ['amount' => 100]);

    \Log::shouldHaveReceived('info')
        ->with('Payment processed', \Mockery::on(function ($context) {
            return $context['amount'] === 100 && $context['status'] === 'success';
        }));
}
```

🎓 خاتمة + تحدي

تحديك ليك:

اعمل **Request/Response Logger Middleware** بـ:

- ✅ Log الـ كل requests (method, path, status code)
- ✅ Log الـ response time
- ✅ Log الـ request body (بدون sensitive data)
- ✅ Log الـ errors اللي حصلت

الحل:

```php
// app/Http/Middleware/LogRequests.php
namespace App\Http\Middleware;

class LogRequests {
    public function handle($request, \Closure $next) {
        $start = microtime(true);

        $response = $next($request);

        $duration = (microtime(true) - $start) * 1000;

        $context = [
            'method' => $request->method(),
            'path' => $request->path(),
            'status_code' => $response->getStatusCode(),
```

```
            'duration_ms' => round($duration),
            'user_id' => auth()->id(),
            'ip_address' => $request->ip(),
        ];

        $level = $response->getStatusCode() >= 500 ? 'error' : 'info';

        \Log::log($level, 'HTTP Request', $context);

        return $response;
    }
}

// في kernel.php
protected $middleware = [
    \App\Http\Middleware\LogRequests::class,
];
```

Checklist ليك:

- ✅ فهمت الفرق بين normal و structured logging
- ✅ استخدمت multiple log channels
- ✅ صحيح context بـ logged
- ✅ ما سجلت sensitive data
- ✅ عملت log rotation
- ✅ عملت custom handlers

📋 Cheat Sheet

Log Levels

```
\Log::debug('Debug message');
\Log::info('Info message');
\Log::notice('Notice message');
\Log::warning('Warning message');
\Log::error('Error message');
\Log::critical('Critical message');
\Log::alert('Alert message');
\Log::emergency('Emergency message');
```

Common Patterns

```
// Log with context
\Log::info('Action', ['user_id' => 1, 'data' => $data]);

// Log exceptions
\Log::error('Error', ['exception' => $e->getMessage()]);
```

```
// Log to specific channel
\Log::channel('payments')->info('Payment', $data);

// Multiple channels at once
\Log::stack(['single', 'slack'])->error('Error', $data);
```

Configuration

```
// في .env
LOG_CHANNEL=stack
LOG_LEVEL=debug

// في config/logging.php
'daily' => [
    'driver' => 'daily',
    'days' => 14,        // احفظ 14 يوم
    'level' => 'debug',  // احفظ كل الـ levels
],
```

Best Practices

الممارسة	ليه؟
استخدم structured logging	Analytics-friendly
حط context	سهل debugging
ما تسجل secrets	Security
Log errors بـ full trace	Debugging
حط timestamps	Audit trail
استخدم multiple channels	Organization
Rotate logs	Disk space
Monitor logs	Early detection

النهاية 🚀

الفكرة الأساسية:

> Good logging = fast debugging + better insights.

يلا.. ابدأ تسجل! 💪

□□ ♡ □□ □□□□□□ *Backend Instructor*