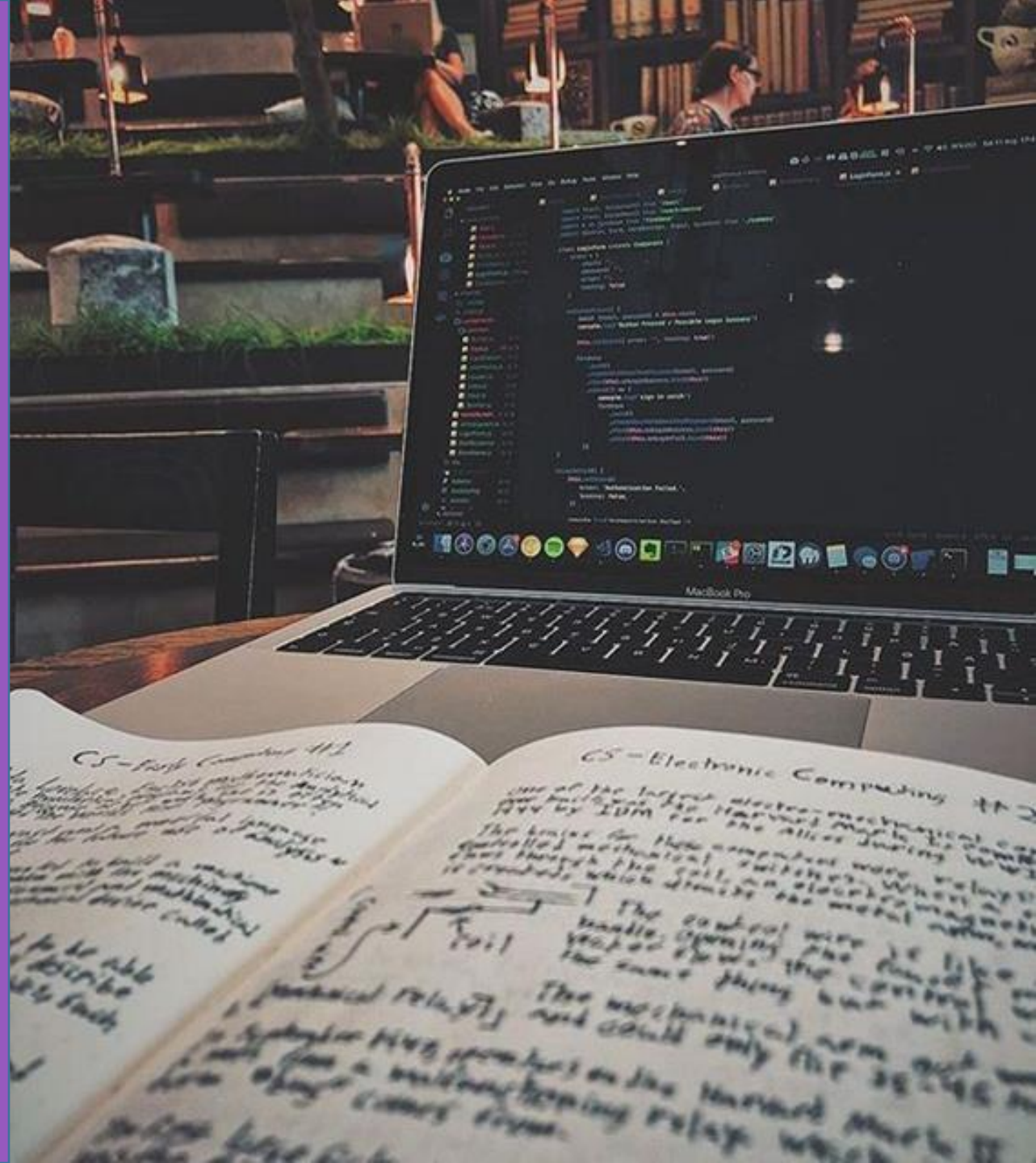
# Clase Nro 2

## Logueo de errores

- Logs
- Aplicaciones
- NLog
- Instalación y configuración de NLog
- ¿Qué debemos loguear?



# Logs o Registros

## Definición

Según Wikipedia, se usa el término **registro**, ***log*** o **historial de *log*** para referirse a la grabación secuencial en un archivo o en una base de datos de todos los acontecimientos (eventos o acciones) que afectan a un proceso particular (aplicación, actividad de una red informática, etc.). De esta forma constituye una evidencia del comportamiento del sistema.

- **logging** son los procedimientos para grabar y guardar los eventos del sistema, aplicación o red.

# Utilidad de los Logs

## Log - Aplicaciones

- Análisis forense.
- Detección de intrusos.
- Depuración de errores.
- Monitorización. Por ejemplo, averiguar los últimos archivos abiertos, últimos archivos modificados, los últimos comandos ejecutados o las últimas páginas web consultadas.
- Cumplir legalidad. Por ejemplo, un proveedor de servicios de Internet tiene que tener por ley un *log* de las conexiones de sus clientes.
- Auditoría

# Ilogger

## ¿Qué es?

.NET cuenta con una API llamada ILogger que admite un registro estructurado de alto rendimiento para ayudar a supervisar el comportamiento de la aplicación y diagnosticar problemas.

```
2 references
private readonly ILogger<HomeController> _logger;
0 references
public HomeController(ILogger<HomeController> logger)
{
    _logger = logger;
}

0 references
public IActionResult Index()
{
    _logger.LogCritical("hola mundo");
    return View();
}
```

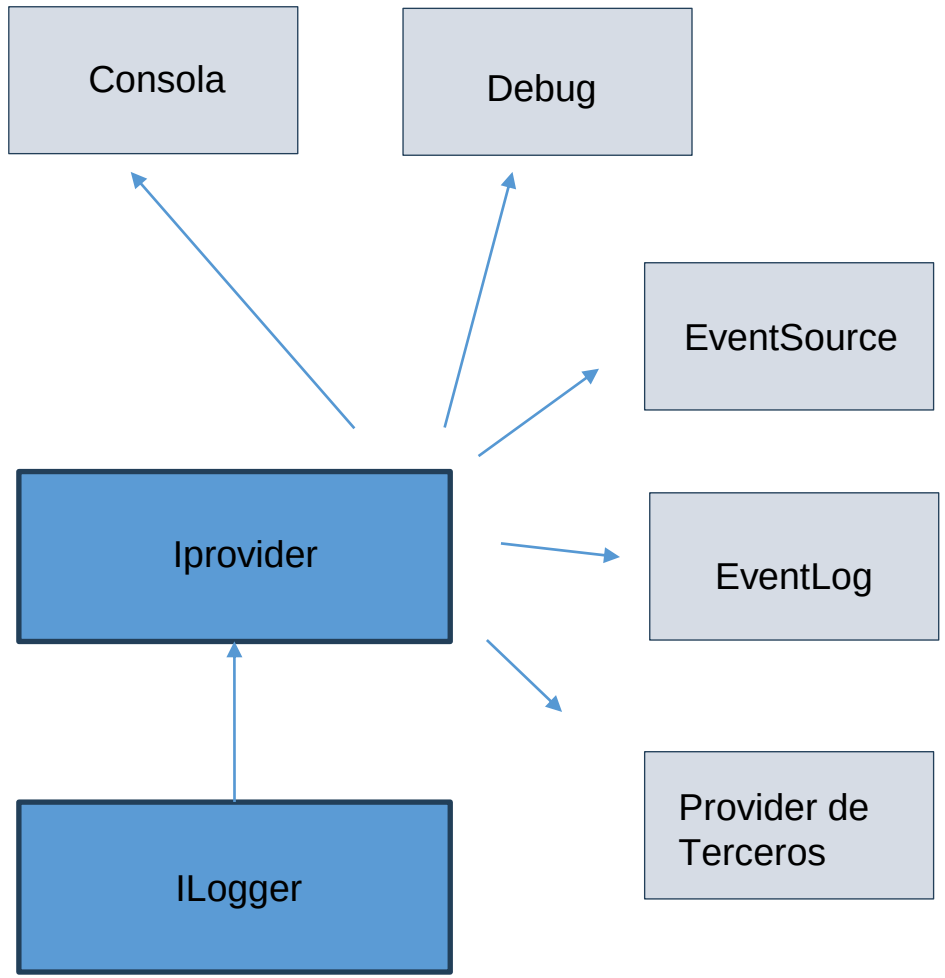
# Ilogger

## Providers

En ASP.NET Core, el sistema de registro (logging) utiliza el concepto de proveedores de registro (logging providers) para determinar dónde se deben enviar los mensajes de registro. Esto permite mostrar o almacenar los registros generados por un sistema en un medio particular, como una consola, un evento de depuración, un registro de eventos, un  trace listener y otros.

Por defecto, ASP.NET Core incluye algunos proveedores de registro comunes, y entre ellos, se encuentra el proveedor de consola (ConsoleLoggerProvider).

ASP.NET Core, se puedes especificar los proveedores de registro que desea utilizar. Si no especificas ninguno, ASP.NET Core utilizará el proveedor de **consola** como predeterminado.



# Ilogger

## Utilizando Ilogger

**El objeto Ilogger llega a nuestros controladores por inyección de dependencias:**

```
private readonly ILogger<HomeController> _logger;

public HomeController(ILogger<HomeController> logger)
{
        _logger = logger;
}
```

**Por último, puedo utilizar el método adecuado para registrar un evento**

```
_logger.LogTrace("Ejemplo de mensaje de trace ");
_logger.LogDebug("Ejemplo de mensaje debug ");
_logger.LogInformation("Ejemplo de mensaje de información");
_logger.LogWarning("Ejemplo de mensaje de Warning");
_logger.LogError("Ejemplo de mensaje de error");
_logger.LogCritical("Ejemplo de mensaje de error Crítico");
```

# Loggin

## Niveles de Logeo

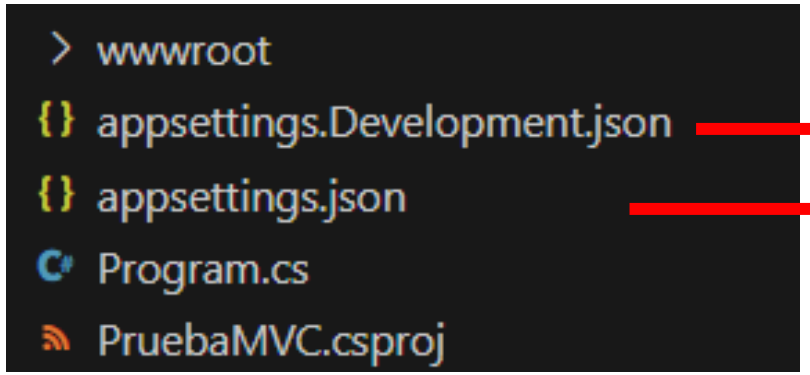
En la tabla, LogLevel aparece de menor a mayor gravedad.

| LogLevel | Valor | Método | Descripción |
|---|---|---|---|
| Trace | 0 | LogTrace | Contienen los mensajes más detallados. Estos mensajes pueden contener datos confidenciales de la aplicación. Están deshabilitados de forma predeterminada y **no** se deben habilitar en un entorno de producción. |
| Debug | 1 | LogDebug | Para depuración y desarrollo. Debido al elevado volumen, tenga precaución cuando lo use en producción. |
| Información | 2 | LogInformation | Realiza el seguimiento del flujo general de la aplicación. Puede tener un valor a largo plazo. |
| Warning | 3 | LogWarning | Para eventos anómalos o inesperados. Normalmente incluye errores o estados que no provocan un error en la aplicación. |
| Error | 4 | LogError | Para los errores y excepciones que no se pueden controlar. Estos mensajes indican un error en la operación o solicitud actual, no un error de toda la aplicación. |
| Critical | 5 | LogCritical | Para los errores que requieren atención inmediata. Ejemplos: escenarios de pérdida de datos, espacio en disco insuficiente. |
| None | 6 | | Especifica que no se debe escribir ningún mensaje |

# Logging

## Configuración de Ilogger

En el caso de las aplicaciones que usan un host, la configuración de registro se proporciona normalmente en la sección "Logging" de appsettings.{Environment}archivos .json.




```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  }
}
```

## Documentación oficial

https://learn.microsoft.com/es-mx/dotnet/core/extensions/logging

# Logging

## Configuración de Ilogger

En el caso de las aplicaciones que usan un host, la configuración de registro se proporciona normalmente en la sección "Logging" de appsettings.{Environment}archivos .json.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
    }
  }
}
```

LogLevel especifica el nivel mínimo que se va a registrar para las categorías seleccionadas.

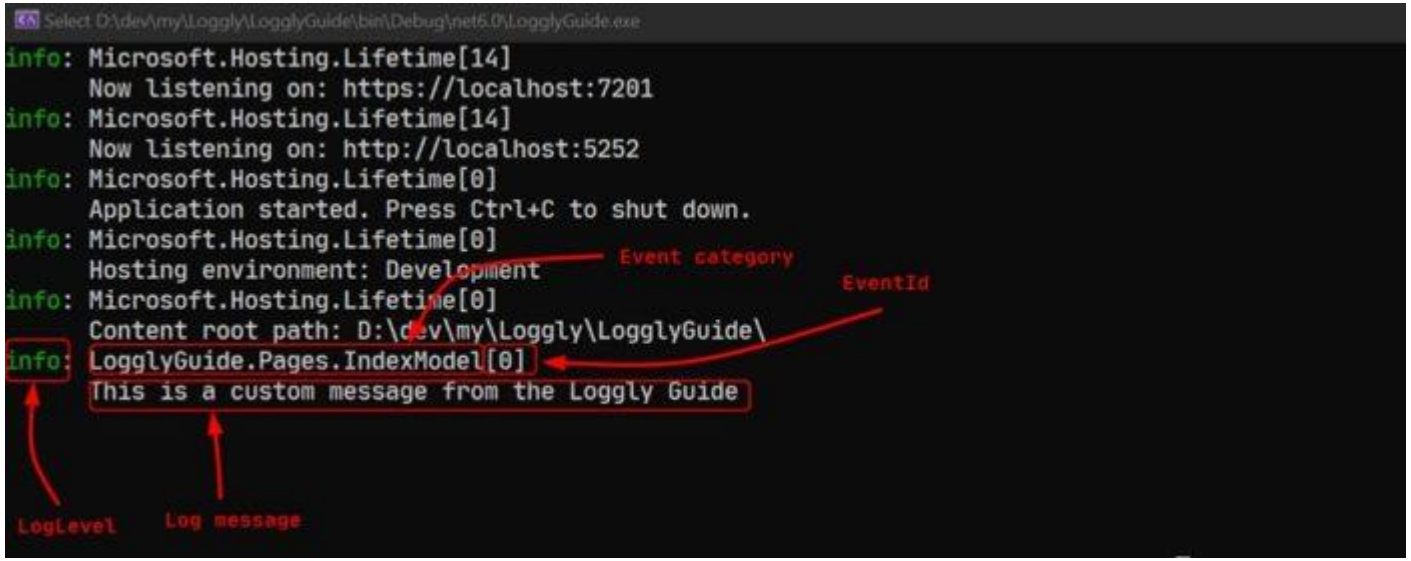
En el documento JSON de la izquierda, se especifican los niveles de registro de “Information” en adelante, todos los niveles que estén por debajo no se registrarán.

En el documento JSON también podemos definir una regla específica para un espacio de nombre específico. Simplemente debemos incluir el espacio de nombre como “clave” y colocar el nivel de registro que necesitemos.

# Utilidad de los Logs

## Log – Salida

Los registros se pueden escribir en distintos destinos mediante la configuración de diferentes proveedores de registro. Los proveedores de registro básicos están integrados y también hay muchos proveedores de terceros disponibles.



¿Donde es conveniente colocar el log en nuestro código?

# Loggin

¿Donde debemos colocar el log?

```
try
{
        // Código que se intenta ejecutar
}
catch (Exception ex)
{
        Ilogger.LogError(ex.tostring());
}
finally
{
        // Codigo que se ejecuta siempre
}
```

De esta forma capturamos cada error y lo insertamos en Nuestro registro de Log.

# Loggin

## Obteniendo información de una excepción

Que información podemos recuperar de una excepción

1. **MessageError**: proporciona una explicación de lo que ha ocurrido.
2. **InnerException**: proporciona información sobre la excepción interna.
3. **StackTrace**: proporciona información de la pila de llamadas antes de la excepción.
4. **Exception.ToString():** Nos provee una gran cantidad e información del error sucedido

# Loggin

¿Donde debemos colocar el log?

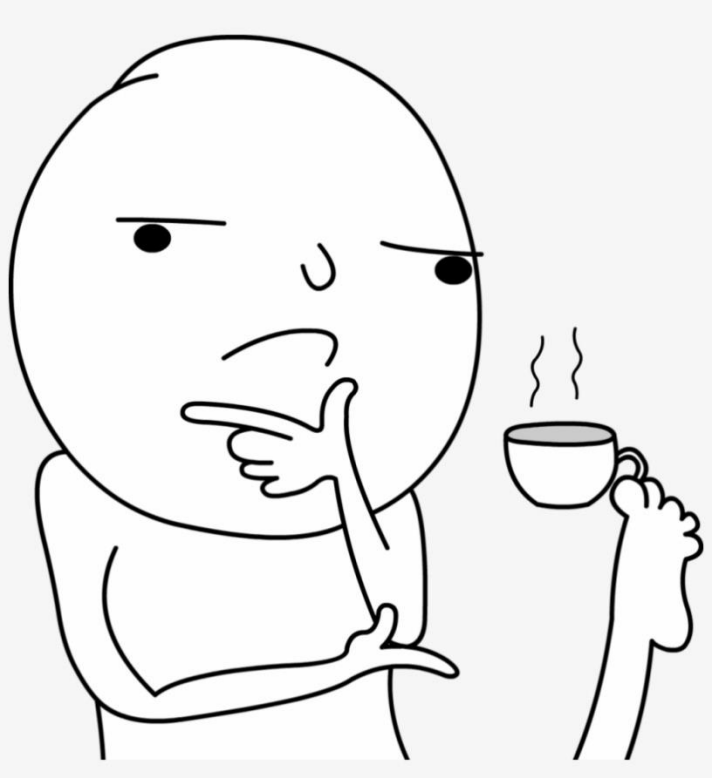
```
try
{
        MetodoQueProvocaUnaExcepcion();
}
catch (Exception ex)
{
   // Qué ha sucedido
   var mensaje = "Error message: " + ex.Message;

   // Información sobre la excepción interna
   if (ex.InnerException != null)
   {
        mensaje = mensaje + " Inner exception: " + ex.InnerException.Message;
   }

   // Dónde ha sucedido
   mensaje = mensaje + " Stack trace: " + ex.StackTrace;

   _ILogger.LogError(mensaje);
}
```

# DEBATE:

¿Que conviene logueamos en nuestra aplicación?

# Logging

¿Solo vamos a loguear los try – catch?



# Logging

¡Si logueamos todo, no se escapa nada!

# Try catch - Loggin

## Conclusiones

- Siempre es conveniente construir un programa considerando capturar excepciones para hacerlo más robusto ante posibles situaciones inesperadas.
- No transferir a los usuario información de errores con la que no podrá lidiar
- Guarda toda la info de las excepciones en un log.
- Exception.ToString() puede brindar suficiente información para guardar en un log

# Bibliografía

https://uniwebsidad.com/libros/algoritmos-python/capitulo-12

https://github.com/nlog/nlog/wiki/Configuration-file

https://albertcapdevila.net/control-excepciones-csharp/

https://www.codeproject.com/Articles/10631/Introduction-to-NLog

Documento, versión 2023-11