Logging Levels in Detail and Miscellany

Logging Level	What it Records
FATAL[1]	Logs events that are the most severe and that prevent the application from performing its work. Eg - ● Crucial configuration information is missing ● Loss of essential external dependencies or services required for core application operations (such as the database). ● Running out of disk space or memory on the server. Max info should be included in the entry.
ERROR[1]	This level is used to indicate conditions that hinder a specific operation's execution. The application can continue functioning despite the error. Not all occurrences of errors or exceptions in your application should be logged at the ERROR level. Eg - , if an exception is expected behavior and does not result in a degradation of application functionality or performance, it can be logged at a lower level, such as DEBUG. Similarly, errors with a potential for recovery, such as network connectivity issues with automated retry mechanisms, can be logged at the WARN level and then may be elevated to the ERROR level if recovery is unsuccessful after several attempts. Some examples of events that're logged as errors: ● External API or service failures impacting the application's functionality (after automated recovery attempts have failed). ● Network communication errors, such as connection timeouts or DNS resolution failures. ● Failure to create or update a resource in the system. ● An unexpected error, such as the failure to decode a JSON object.

WARN[1]	Events logged at this level indicate that something unexpected has happened. Despite this, the application can continue to function. It is also used to signify conditions that should be promptly addressed before they escalate into problems for the application. Some examples of events that can be logged as WARN entries are: ● Resource consumption nearing predefined thresholds (such as memory, CPU, or bandwidth). ● Outdated configuration settings that are not in line with recommended practices. ● An excessive number of failed login attempts indicating potential security threats. ● External API response times exceed acceptable thresholds. Before we can log a warning, we need to have pre-defined thresholds for conditions. When these conditions are exceeded, then the warning log entry is recorded. Eg - Disk usage, no. of failed login attempts, etc.
INFO[1]	Used to capture events that're part of the application's business logic. Such events are captured to show that the system is operating normally. Used in Production. Some examples of events: ● Changes in the state of an operation. ● The successful completion of scheduled jobs or tasks. ● Starting or stopping a service or application component. ● Records of important milestones or significant events within the application. ● Information about system health checks or status reports.

DEBUG[1]	It is used for logging messages that help developers to detect problems with the program. It isn't used in production; only in development. It should be used with caution as many entries may be logged and this can be resource intensive. DEBUG log entries typically contain detailed technical information. Some examples of events that can be logged as DEBUG: ● Database queries, which can aid in identifying performance bottlenecks or issues related to data retrieval or manipulation. ● The details of external API calls and their responses. ● Configuration values to aid in troubleshooting misconfigured or mismatched settings. ● Timing information such as the duration of specific operations or method executions.
TRACE[1]	It is used for tracing the path of execution of a program. It provides a detailed breakdown of events leading to the program crash and can provide a lot of context for WARN, DEBUG and ERROR entries. Some examples of events that are logged as trace: ● When entering or exiting from a function or method, along with any relevant parameters or return values. ● The iteration details within loops, such as the current index or the values being processed. ● Calculations or operations that produce specific output.

Miscellaneous:

- Currently, I plan to monitor function execution time and response time to http requests. I am looking into monitoring other metrics. Deciding what is INFO, WARN and ERROR will be done according to the number of metrics that we can monitor. We will change the definitions slightly as we can record more metrics.

- Winston can log entries but can't monitor for events [2]. Logging entries is its primary function. It can also profile code execution time and catch uncaught exceptions. [2] However, we aren't using it to catch uncaught exceptions since this is already being done using the global error middleware.

- I am looking into the perf tool of NodeJS which is used to profile code, NodeJS Events and the built-in V8 profiler. [3][4]

- I am also looking into NodeJS Events. [5]

- There is also a middleware to get response times of HTTP requests. The "response time" is defined here as the elapsed time from when a request enters this middleware to when the headers are written out to the client. [6]

REFERENCES

[1] https://betterstack.com/community/guides/logging/log-levels-explained/

[2]https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-winston-and-morgan-to-log-node-js-applications/

[3] https://nodejs.org/api/perf_hooks.html

[4] https://nodejs.org/en/learn/getting-started/profiling

[5] https://nodejs.org/api/events.html#events

[6] https://nodejs.org/en/learn/getting-started/profiling