Error Logging

An error log is a file that contains detailed records of error conditions a computer software encounters when it's running. [1] There are many error logging libraries for Node.JS. We will be looking at some of them over here.

Winston:

- It is the most popular logging library for NodeJS. [2][3]
- It decouples log levels, formatting and storage making it more flexible. [2][3]
- It uses **NodeJS Streams** to minimize the performance impact of logging. [2]
- The recommended way of using winston is by creating a logger by calling the **createLogger** method, after which we can start logging messages by using the **logger** object (Eg - logger.warn, logger.error, logger.info, etc.) [2][3]
- Winston writes those messages to the log that are >= to the level set using the **level** property. This allows winston to be run at different levels at different points of time. [3]
- The default value of the **level** property is **info** [2][3]. It should be set using an environmental variable. [3]
- The default log levels are defined in **winston.config.npm.levels**. [3]
- It is also possible to use custom log levels. This is done by defining a JS object logLevels with the custom levels and then assigning it to the **levels** property.
- The default format in which winston outputs logs is **JSON**. The formats are accessible using the **winston.format** object. The format can be set by using the **format** property. [3]
- The timestamp format can be changed by passing a timestamp format string to the **format** property in the **timestamp** object. [3]
- A **transport** in winston refers to the device (path) where the log file will be stored. Some transports available in winston by default are: **console**, **file**, **http** and **stream**.
- It is a good practice to have messages of each level in its own file. [3]
- Log Rotation is the process of restricting log file size and create new ones based on some pre-defined criteria. [3]
- Winston provides the **winston-daily-rotate-file module** for log rotation. [3]
- Contextual messages can be added to a logger using the **default-meta** property..
- Winston can catch uncaught exceptions and promise rejections. To do this, we need to specify the transports for the exception handler and the promise handler respectively. [3]
- Winston can profile application code through the profile() method. [3]
- It is possible to create multiple loggers, each having different settings. This is useful for logging different areas of the application. [3]

Advantages over Pino:

- Winston can handle uncaught exceptions, while pino can't. [3]
- Winston provides a module for log-rotation while with Pino we need to rely on a log rotation utility.
- Winston can catch uncaught exceptions and rejections while Pino can't.
- Winston can profile application code which Pino can't.
- Winston provides the ability to create multiple loggers, each with different settings independent of others, that can profile different parts of the application code, which Pino can't.

Pino:
Pino is a very fast (low overhead) logger for JS frameworks.
Here, we will be going through some of the key features of **Pino**. [4]

- The default log levels in pino are: **trace**, **debug**, **info**, **warn**, **error**, and **fatal** (in ascending order of severity). [4]
- Pino logs are in JSON by default. This makes them machine-parsable. A utility called **jq** can make these logs more readable. [4]
- **pino-pretty** is a package that pretty-prints the JSON output from pino. [4]
- The minimum log level in pino is **info** and it is set at the time of creating the logger. This means that only events of this severity and below will be logged. [4]
- Custom log levels can be created and assigned using the **customLevels** property in **pino**. [4]
- Pino binds two extra properties to each log entry by default: the program's process ID (pid), and the current machine's hostname. These can be changed by changing the **bindings** property. [4]
- It is possible to add **contextual messages** to our logs. This is done through the **message** parameter on a level method. [4]
- Contextual messages about where a log message is being produced (function, endpoint, etc) can be added using **child loggers**. [4]
- It is possible to transport **pino** logs to multiple destinations by creating a **targets** array and placing all the transports in that array. [4]
- It is possible to keep sensitive data out of logs through using **log redaction.** Pino uses the **fast-redaction** package to provide log redaction for identifying and removing sensitive data from logs. The fields to be redacted are placed in the **redact** property. [4]
- Pino can be used to log **HTTP** requests using the **pino-http** package in **express**. [4]

Morgan:
It is a middleware for logging HTTP requests. It works well with NodeJS applications built using the Express library. [5]

- Morgan provides several predefined log formats. These are: **combined** (apache-style, has referrers and agents); common (standard apache format with referrers and user agents); **dev** (A concise color coded output); **short** (A shorter version of the common format); **tiny (**This includes only req info & status code). [5]

- Log outputs can be customized in morgan. This is done by creating **tokens**. A token can be created using the **token()** function. It takes a string as a token name and the the data to be returned (Eg - request id, request header, etc). It also has built-in tokens such as **:method, :url, :status, :response-time,** etc. [5]

- It is also possible to define custom formats using a function that takes (**tokens, req, res**) and returns a string. This method involves passing a method to morgan with these arguments that returns a JSON with various keys. [5]

- Morgan can be used with winston and pino loggers. [4]

	Logs What	Logs Where	Morgan Compatible	ELK	Loki
Winston	Errors, warnings, failures, traces, debugs	Console, File in Local Store / External Device, Database (MongoDB, Cassandra, MySQL, DynamoDB)	Yes	Yes with winston-elasticsearch	Yes with winston-loki
Pino	Errors, warnings, failures, traces, debugs	Console, File in Local Store / External Device, Database (MongoDB)	Yes (But not needed).	Yes with pino-elasticsearch	Yes with pino-loki

REFERENCES

[1] https://www.crowdstrike.com/en-us/cybersecurity-101/observability
[2] https://github.com/winstonjs/winston/
[3] https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-winston-and-morgan-to-log-node-js-applications/
[4] https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-pino-to-log-node-js-applications/
[5] https://betterstack.com/community/guides/logging/morgan-logging-nodejs/