LN05 – API logger nodejs

Contents

1. Requirement & Dependencies 3
1.1. Requirement 3
1.2. New dependencies 3
2. Log request and response step 3
2.1. Create a logger.js file in the utils folder 3
2.2. Create a new logger object and export it 3
2.3. Use the logger in API GET film/:id 4
3. Log level 5
4. Log storage. 6
4.1. Log to file 6
4.2. 3rd party storage. 7
5. Search log in file 10
6. Rotation 11

1. **Requirement & Dependencies**

   1.1. **Requirement**

   - All APIs implementation, database and dependencies in week1.
   - A papertrail account in papertrailapp.com.

   1.2. **New dependencies**

   - winston: a logging library with support for multiple transports. (transports are likely paths to where we store logs).
   - winston-daily-rotate-file: a library that support Winston to rotate log file.
   - winston-syslog: asynchronously transmitting events from Node.js, use the winston-syslog transport (this is going to be used in the 3rd party).

   Use npm install <package name> to install those.

2. **Log request and response step**

   2.1. **Create a logger.js file in the utils folder**

   First, we import some feature

```
import { createLogger, format, transports } from "winston";
```

   2.2. **Create a new logger object and export it.**

```
const logger = createLogger({
  format: format.combine(
    format.timestamp(),
    format.json()
  ),
  transports: [
    new transports.Console({
      level: "info",
    })
  ]
});

export default logger;
```

   Any number of formats may be combined into a single format using format.combine.
   For future query in file, format.json is needed.
   Format.timestamp is needed when we need to search by a range of time.

2.3. Use the logger in API GET film/:id

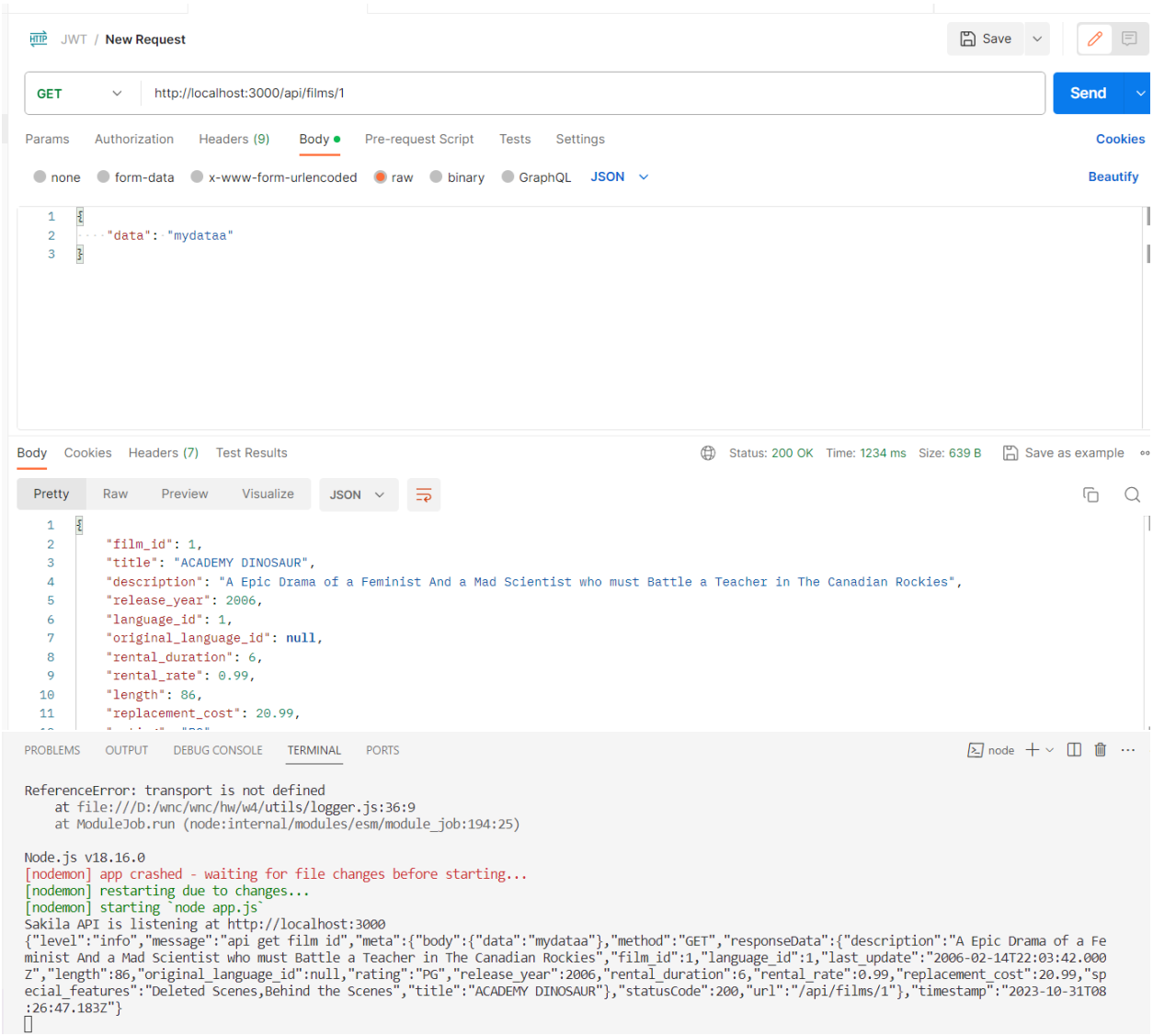
```
router.get("/:id", async function (req, res) {
  const id = req.params.id || 0;
  const info = {
    level: "info",
    message: "api get film id",
    meta: {
      method: req.method,
      url: req.url,
      body: req.body,
    },
  };
  const film = await filmModel.findById(id);
  if (film === null) {
    info.meta.response = "not found";
    info.meta.statusCode = 204;
    return res.status(204).end();
  }
  info.meta.responseData = film
  info.meta.statusCode = 200
  logger.log(info);
  res.json(film);
});
```

Before we use the logger; remember to import it!

We can log the data by the code logger.log(info) where info must have the level and message attribute, the rest attribute is meta data (`const { level, message, ...meta } = info`)

In this meta data, we store response data, response code, some request information for further log searching and timestamp that we just config in the format above.

Result:



3. Log level

By default, Winston follows the severity ordering specified by RFC5424.

```
const levels = {
  error: 0,
  warn: 1,
  info: 2,
  http: 3,
  verbose: 4,
  debug: 5,
  silly: 6
};
```

Looking at the transport.Console above, we specify it level is info which means that any log with the severity level from http to silly will not log. For example, if we set info.level = 'http', then the log will

not appear on the console by the logger object. And on the other hand, any level from info to error can be logged by the logger object.

You can config your log level by assign the levels attribute of a logger:

```
const myCustomLevels = {
  levels: {
    foo: 0,
    bar: 1,
    baz: 2,
    foobar: 3
  },
  colors: {
    foo: 'blue',
    bar: 'green',
    baz: 'yellow',
    foobar: 'red'
  }
};

const customLevelLogger = winston.createLogger({
  levels: myCustomLevels.levels
});
```

4. Log storage.

4.1. Log to file.

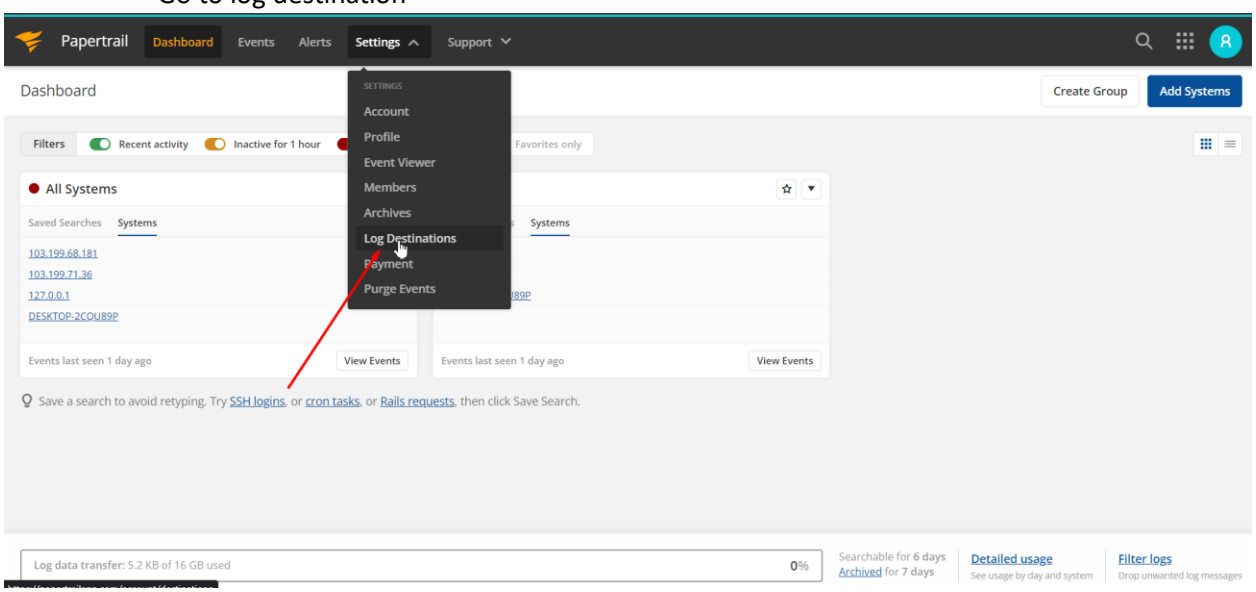
Now, We go back to the logger object and add a new file transport with a specific filename.

```
const logger = createLogger({
  format: format.combine(
    format.timestamp(),
    format.json()
  ),
  transports: [
    new transports.Console({
      level: "info",
    }),
    new transports.File({
      filename: './log/logToFile.txt'
    })
  ]
});
```

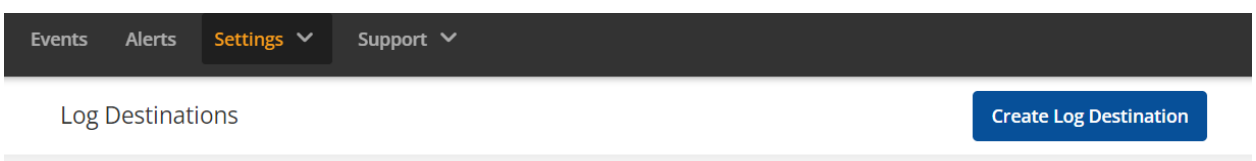
And that how we log to file, the info variable will log to both console and logToFile.txt
So when you want some log only print to a file or somewhere else, you can create another logger object for each purpose.

4.2. 3rd party storage.

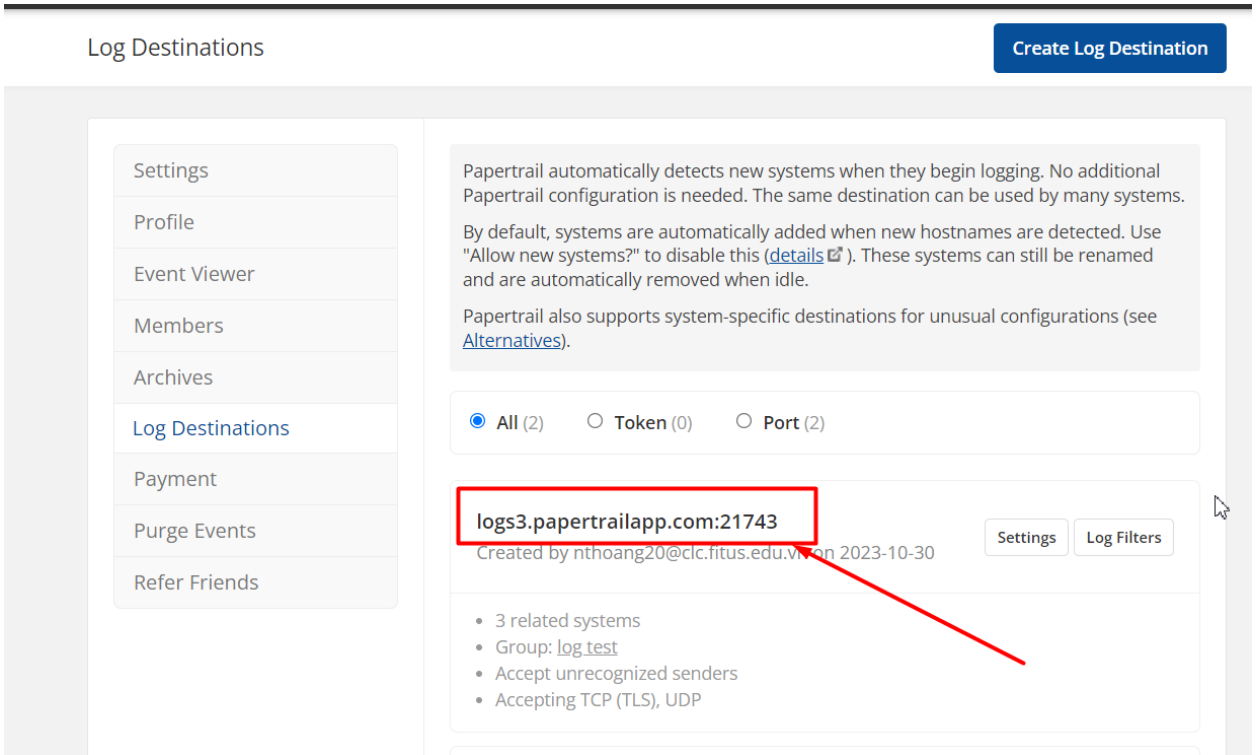
- Log in to papertrail.
- Go to log destination



- Click create log destination and hit create



- Then you will see the host and port of the log destination



- Now, we go to the logger.js file
  - Import the Winston-syslog

```
import "winston-syslog";
```

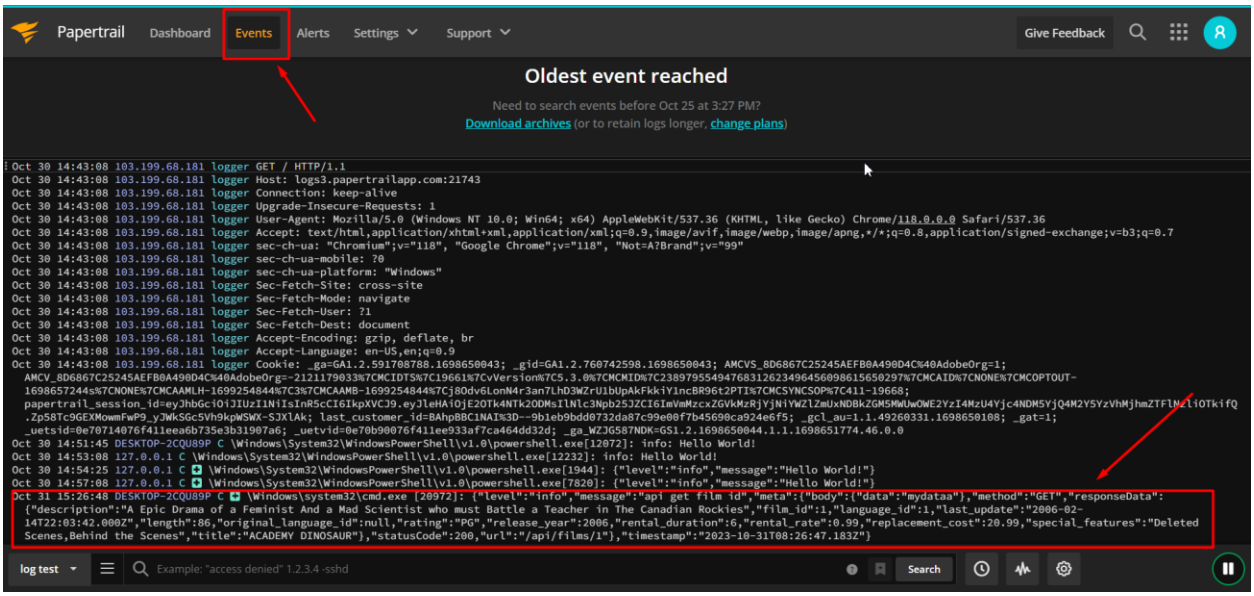
  - Create a syslog transport with host and port you just get

```
const papertrail = new transports.Syslog({
  host: 'logs3.papertrailapp.com',
  port: 21743,
  protocol: 'tls4',
  localhost: os.hostname(),
  eol: '\n',
});
```

    The localhost can be any name you like, this one, I just got the os hostname, protocol and eol are followed by this instruction (https://www.papertrail.com/help/configuring-centralized-logging-from-nodejs-apps/)
  - Add that transport to the logger

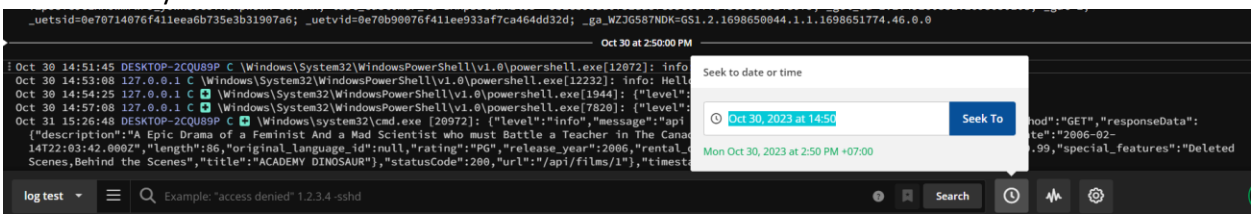
```
const logger = createLogger({
  format: format.combine(
    format.timestamp(),
    format.json()
  ),
  transports: [
    new transports.Console({
      level: "info",
    }),
    new transport.File({
      filename: './log/logToFile.txt'
    }),
    papertrail
  ]
});
```

- Next, we run the backend server and call GET films/:id.
- Then, we go to the events tab and see the log



- In here, you can search by keywords

- Or by time



5. **Search log in file**

We can add a new file call logQuery.js and create a logger which it can log to a file we want to search. For more convenient, I import the logger in the utils folder.
winston supports querying of logs with Loggly-like options.

```
JS logQuery.js > logger.query() callback
import logger from "./utils/logger.js";

var options = {
    from:   new Date - 48 *60 * 60 * 1000,
    until:  new Date,
    limit:  10,
    start:  0,
    order:  'asc',
    fields: ['meta', 'timestamp']
};
logger.query(options, function (err, results) {
    if (err) {
        throw err;
    }
    results = results.file.filter(result => result.meta.method === "GET")
    console.log(results);
});
```

This option will help us search log from today to two days before, starting from log number 0 to number 9 and only the fields meta and timestamp will be returned as an array in the file attribute of the result object. To find result in more specific case, we can use some function in javascript like filter.
The result:

```
PS D:\wnc\wnc\hw\w4> node logQuery.js
[
  {
    meta: {
      body: [Object],
      method: 'GET',
      responseData: [Object],
      statusCode: 200,
      url: '/api/films/1'
    },
    timestamp: '2023-10-30T16:40:04.299Z'
  },
  {
    meta: {
      body: [Object],
      method: 'GET',
      responseData: [Object],
      statusCode: 200,
      url: '/api/films/1'
    },
    timestamp: '2023-10-31T08:26:47.183Z'
  }
]
```

You can stringify the json to see full data of each Object.

One most important thing I just say somewhere above, the query of Winston works if the data in the file is json format. In addition, winston query only work on some transport: File, Couchdb, Redis, Loggly, Nssocket, and Http.

6. **Rotation**

When we don't have enough space for storing log, rotation is an option for us to deal with it. We can rotate the log data weekly, daily,… or based on size. In this document, I will rotate by minute and by size.

First, we back to the logger.js file and import Winston-daily-rotate-file library.

```
import { createLogger, format, transport
import os from 'os';
import "winston-daily-rotate-file";
import "winston-syslog";
```

Then, we create a daily rotate file transport

```
const rotateTransport = new transports.DailyRotateFile({
  frequency: "3m",
  filename: "./log/application-%DATE%.log",
  datePattern: "mm",
  maxSize: "0.578k",
  maxFiles: "2",
});
```

- datePattern: A string representing the moment.js date format to be used for rotating. The meta characters used in this string will dictate the frequency of the file rotation. For

example, if your datePattern is simply 'HH' you will end up with 24 log files that are picked up and appended to every day. (default: 'YYYY-MM-DD').

- Frequency: A string representing the frequency of rotation. This is useful if you want to have timed rotations, as opposed to rotations that happen at specific moments in time. Valid values are '#m' or '#h' (e.g., '5m' or '3h'). Leaving this null relies on datePattern for the rotation times. (default: null).
- maxSize: Maximum size of the file after which it will rotate. This can be a number of bytes, or units of kb, mb, and gb. If using the units, add 'k', 'm', or 'g' as the suffix. The units need to directly follow the number. (default: null)
- maxFile: Maximum number of logs to keep. If not set, no logs will be removed. This can be a number of files or number of days. If using days, add 'd' as the suffix. It uses auditFile to keep track of the log files in a json format. It won't delete any file not contained in it. It can be a number of files or number of days (default: null)

So, this transport will rotate every 3 minutes and only store 2 files. For example, if it's 15:15, then we got application-15.log and after 3 minutes to 6 minutes, we'll get application-18.log if this transport is trigger by the API at that time. When we get application-21.log, the application-15.log will be removed because maxFiles is 2. If the application-21.log exceed 0.578kb in the time from 15:21:00 to 15:23:59, the next rotation will remove that file and create another log file called application-21.log.1.

And all this process will be manage by an audit file, which has a long name of hash character. In addition, each transport creates its own audit file.

Next, we add that transport to the createLogger

```
const logger = createLogger({
  format: format.combine(
    format.timestamp(),
    format.json()
  ),
  transports: [
    new transports.Console({
      level: "info",
    }),
    new transports.File({
      filename: './log/logToFile.txt'
    }),
    rotateTransport
    //papertrail
  ]
});
```

Finally, we run the app, call the GET films/1 and get the log in the log file.

- log
  - .77e1d5dfbea8f3f19e6f5366c59cd05f8db1e332-a...
  - application-12.log
  - logToFile.txt
- models
- node_modules


```
1 {"level":"info","message":"api get film id","meta":{"body":{"data":"mydataa"},"method":"GET","responseData":{"description":"A
  Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies","film_id":1,"language_id":1,
  "last_update":"2006-02-14T22:03:42.000Z","length":86,"original_language_id":null,"rating":"PG","release_year":2006,
  "rental_duration":6,"rental_rate":0.99,"replacement_cost":20.99,"special_features":"Deleted Scenes,Behind the Scenes",
  "title":"ACADEMY DINOSAUR"},"statusCode":200,"url":"/api/films/1"},"timestamp":"2023-10-31T13:12:22.419Z"}
2
```