ReactJS MUI error fix

01/15/2026 22:48:36

Below is a complete, production-ready logging setup for a Node.js +...

Below is a **complete, production-ready logging setup** for a **Node.js + Express** backend using **Winston**.
This is **industry standard**, audit-friendly, and scalable.

No theory — **full working code** 👇

1️⃣ Install dependencies

```
npm install winston winston-daily-rotate-file morgan
```

2️⃣ Logger utility (`utils/logger.js`)

```
import winston from "winston";
import DailyRotateFile from "winston-daily-rotate-file";
import path from "path";

const logDir = path.resolve("logs");

const logFormat = winston.format.combine(
  winston.format.timestamp({ format: "YYYY-MM-DD HH:mm:ss" }),
  winston.format.errors({ stack: true }),
  winston.format.printf(
    ({ timestamp, level, message, stack }) =>
      `${timestamp} [${level.toUpperCase()}]: ${stack || message}`
  )
);

const logger = winston.createLogger({
  level: "info",
  format: logFormat,
  transports: [
    new winston.transports.Console(),

    new DailyRotateFile({
      dirname: logDir,
      filename: "application-%DATE%.log",
```

```
      datePattern: "YYYY-MM-DD",
      maxFiles: "14d",
    }),

    new DailyRotateFile({
      dirname: logDir,
      filename: "error-%DATE%.log",
      datePattern: "YYYY-MM-DD",
      level: "error",
      maxFiles: "30d",
    }),
  ],
});

export default logger;
```

✔ Console logging
✔ Daily rotated files
✔ Separate error logs
✔ Stack traces included

3️⃣ HTTP Request Logging (`middlewares/httpLogger.js`)

```
import morgan from "morgan";
import logger from "../utils/logger.js";

const stream = {
  write: (message) => logger.info(message.trim()),
};

export const httpLogger = morgan(
  ":method :url :status :response-time ms",
  { stream }
);
```

4️⃣ Global Error Handler (`middlewares/errorHandler.js`)

```
import logger from "../utils/logger.js";

export const errorHandler = (err, req, res, next) => {
  logger.error({
    message: err.message,
    method: req.method,
    url: req.originalUrl,
    body: req.body,
  });

  res.status(err.status || 500).json({
    message: err.message || "Internal Server Error",
  });
};
```

✔ Logs request context

✔ Centralized error handling

5️⃣ Use in `app.js` / `server.js`

```
import express from "express";
import { httpLogger } from "./middlewares/httpLogger.js";
import { errorHandler } from "./middlewares/errorHandler.js";
import logger from "./utils/logger.js";

const app = express();

app.use(express.json());
app.use(httpLogger);

logger.info("Server initialized");

// routes
import { CPDSDocumentRouter } from "./routes/CPDSDocumentRouter.js";
CPDSDocumentRouter(app);

// error handler (LAST middleware)
app.use(errorHandler);
```

```
export default app;
```

6️⃣ Logging inside Controllers (IMPORTANT)

❌ Don’t do

```
console.log("Uploaded");
```

✅ Do this

```
import logger from "../utils/logger.js";

export const uploadDocument = async (req, res, next) => {
  try {
    logger.info(`Uploading document for CPDS ${req.body.cpdsSlno}`);

    // business logic

    res.status(201).json({ message: "Document uploaded" });
  } catch (err) {
    logger.error("Upload failed", err);
    next(err);
  }
};
```

7️⃣ Example Log Output

Console

```
2026-01-15 01:24:11 [INFO]: POST /cpds/documents 201 123 ms
```

File (`logs/error-2026-01-15.log`)

```
2026-01-15 01:25:02 [ERROR]: ValidationError: Invalid documentId
```

8️⃣ Environment-based Logging (Optional but Recommended)

```
level: process.env.NODE_ENV === "production" ? "info" : "debug",
```

9️⃣ Folder Structure (Clean)

```
src/
│
├── utils/
│   └── logger.js
├── middlewares/
│   ├── httpLogger.js
│   └── errorHandler.js
├── controllers/
├── routes/
└── app.js
```

✅ Final Verdict

This setup gives you:

✔ Structured logs
✔ Request tracing
✔ Error auditing
✔ Production readiness
✔ Easy DevOps integration

If you want next:

- Correlation IDs
- Request-scoped logging
- ELK / Grafana integration

- CloudWatch compatible logs

Just say 👍