

# Order Logging Demo (Node.js + Express + MongoDB)

---

## Index

- 1. Project Overview
- 2. Learning Outcomes
- 3. Tech Stack
- 4. Folder Structure
- 5. High-Level Architecture
- 6. Logging Concepts Used in This App
- 7. API Endpoints
- 8. How to Run + Quick Tests
- 9. Common Mistakes
- 10. Debugging Techniques
- Appendix A: Logging-Related Code Snippets
- Appendix B: Full Source Code

## 1. Project Overview

This application demonstrates practical logging for a backend API. The focus is on producing useful logs that help you debug production issues: request/response context, correlation IDs, timing, and consistent log format.

## 2. Learning Outcomes

- Explain why logging is critical for production debugging.
- Add correlation IDs (request IDs) so all logs for one request can be traced.
- Log important context safely (avoid sensitive/PII).
- Use logs to debug failures: timeouts, database errors, and invalid inputs.

## 3. Tech Stack

package.json:

```
{
  "name": "order-logging-demo",
  "version": "1.0.0",
  "type": "commonjs",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.19.2",
    "mongoose": "^8.5.2",
    "uuid": "^9.0.1"
  }
}
```

.env (if present):

```
(none)
```

## 4. Folder Structure

```
- package.json
- server.js
  - correlation.js
  - Order.js
  - orders.js
  - logger.js
```

## 5. High-Level Architecture

Client → Express → logging middleware → routes/controllers → MongoDB → response

### Every request should emit a minimal set of logs:

- Request start: method, path, requestId, timestamp
- Important events: validation failures, DB queries (optional), business events (order created)
- Request end: status code, duration (ms), requestId

## 6. Logging Concepts Used in This App

### Key logging principles:

- Correlation ID: a unique requestId attached to every request; included in every log line.
- Structured logs: logs as JSON-like key/value for easy search in tools (ELK/CloudWatch).
- Log levels: info for normal events, warn for unexpected but handled issues, error for failures.
- Do not log secrets: passwords, tokens, raw card data, personal details.

### Logging-related snippets found in your code:

#### server.js

```
const express = require("express");
const mongoose = require("mongoose");
const orderRoutes = require("../routes/orders");
const correlationMiddleware = require("../middleware/correlation");
const logger = require("../utils/logger");

const app = express();
```

#### middleware/correlation.js

```
const { v4: uuid } = require("uuid");
const logger = require("../utils/logger");

module.exports = function correlationMiddleware(req, res, next) {
  const incomingId = req.header("X-Correlation-Id");
```

#### routes/orders.js

```
const express = require("express");
const Order = require("../models/Order");
const logger = require("../utils/logger");

const router = express.Router();
```

#### utils/logger.js

```
message,
  ...meta
};
console.log(JSON.stringify(logEntry));
}
```

```
module.exports = {
```

## 7. API Endpoints

Base URL: <http://localhost:3000> (or as configured)

Method	Path	Source File	Notes
POST	/orders/	routes/orders.js	
GET	/orders/:id	routes/orders.js	

## 8. How to Run + Quick Tests

### Start MongoDB:

```
docker compose up -d
```

### Run server:

```
npm install
npm run dev
```

### Verify logs by calling an endpoint:

```
curl -i http://localhost:3000/orders
```

### Correlation ID test (example):

```
# If the app supports X-Request-Id, set it and confirm it appears in logs
curl -i http://localhost:3000/orders -H "X-Request-Id: demo-req-001"
```

## 9. Common Mistakes

### Logging too little

You can't reproduce issues without requestId, duration, and error details.

### Logging too much noise

Too many logs hide the signal. Keep logs meaningful and consistent.

### Not including requestId in every log line

You cannot trace a single request across multiple log statements.

### Logging sensitive data

Never log passwords, tokens, or personal data. Mask/omit sensitive fields.

## Inconsistent log format

Mixing formats makes search harder. Prefer structured JSON-like logs.

## 10. Debugging Techniques

- Start with requestId: filter logs by requestId to see the whole timeline.
- Compare 'start log' and 'end log' to detect slow requests (duration spikes).
- On DB errors: log the collection/action and key fields (not full payload).
- On 400 errors: log validation failures with which field failed.
- Use curl -i to see response headers and status codes while reproducing issues.

### Example: log-friendly error response shape:

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "amount must be a positive number",
    "requestId": "demo-req-001"
  }
}
```

## Appendix A: Logging-Related Code Snippets

### server.js

```
const express = require("express");
const mongoose = require("mongoose");
const orderRoutes = require("../routes/orders");
const correlationMiddleware = require("../middleware/correlation");
const logger = require("../utils/logger");

const app = express();
```

### middleware/correlation.js

```
const { v4: uuid } = require("uuid");
const logger = require("../utils/logger");

module.exports = function correlationMiddleware(req, res, next) {
  const incomingId = req.header("X-Correlation-Id");
```

### routes/orders.js

```
const express = require("express");
const Order = require("../models/Order");
const logger = require("../utils/logger");

const router = express.Router();
```

### utils/logger.js

```
message,
  ...meta
};
console.log(JSON.stringify(logEntry));
}

module.exports = {
```

## Appendix B: Full Source Code (as provided in the ZIP)

### package.json

```
{
  "name": "order-logging-demo",
  "version": "1.0.0",
  "type": "commonjs",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.19.2",
    "mongoose": "^8.5.2",
    "uuid": "^9.0.1"
  }
}
```

### server.js

```
const express = require("express");
const mongoose = require("mongoose");
const orderRoutes = require("./routes/orders");
const correlationMiddleware = require("./middleware/correlation");
const logger = require("./utils/logger");

const app = express();
app.use(express.json());

mongoose.connect("mongodb://localhost:27017/order_logging_demo")
  .then(() => logger.info("MongoDB connected"))
  .catch(err => logger.error("MongoDB error", { error: err.message }));

app.use(correlationMiddleware);

app.use("/orders", orderRoutes);

app.use((err, req, res, next) => {
  logger.error("Unhandled error", {
    correlationId: req.correlationId,
    error: err.message
  });
  res.status(500).json({ error: "Internal Server Error" });
});

app.listen(3000, () =>
  logger.info("Order Logging Demo running on http://localhost:3000")
);
```

### middleware/correlation.js

```
const { v4: uuid } = require("uuid");
const logger = require("../utils/logger");
```

```

module.exports = function correlationMiddleware(req, res, next) {
  const incomingId = req.header("X-Correlation-Id");
  const correlationId = incomingId || uuid();

  req.correlationId = correlationId;
  res.setHeader("X-Correlation-Id", correlationId);

  logger.info("Incoming request", {
    correlationId,
    method: req.method,
    path: req.originalUrl
  });

  next();
};

```

## models/Order.js

```

const mongoose = require("mongoose");

const OrderSchema = new mongoose.Schema(
  {
    product: String,
    amount: Number,
    status: String
  },
  { timestamps: true }
);

module.exports = mongoose.model("Order", OrderSchema);

```

## routes/orders.js

```

const express = require("express");
const Order = require("../models/Order");
const logger = require("../utils/logger");

const router = express.Router();

router.post("/", async (req, res, next) => {
  try {
    logger.info("Creating order", { correlationId: req.correlationId });

    const order = await Order.create({
      product: req.body.product,
      amount: req.body.amount,
      status: "CREATED"
    });

    logger.info("Order created", {
      correlationId: req.correlationId,
      orderId: order._id
    });
  } catch (error) {
    next(error);
  }
});

```



```

    });

    res.status(201).json(order);
  } catch (err) {
    logger.error("Order creation failed", {
      correlationId: req.correlationId,
      error: err.message
    });
    next(err);
  }
});

router.get("/:id", async (req, res, next) => {
  try {
    logger.info("Fetching order", {
      correlationId: req.correlationId,
      orderId: req.params.id
    });

    const order = await Order.findById(req.params.id);
    if (!order) {
      logger.error("Order not found", {
        correlationId: req.correlationId,
        orderId: req.params.id
      });
      return res.status(404).json({ error: "Order not found" });
    }

    res.json(order);
  } catch (err) {
    logger.error("Fetch order failed", {
      correlationId: req.correlationId,
      error: err.message
    });
    next(err);
  }
});

module.exports = router;

```

## utils/logger.js

```

function log(level, message, meta = {}) {
  const logEntry = {
    timestamp: new Date().toISOString(),
    level,
    message,
    ...meta
  };
  console.log(JSON.stringify(logEntry));
}

module.exports = {
  info: (msg, meta) => log("INFO", msg, meta),

```

```
error: (msg, meta) => log("ERROR", msg, meta)
};
```