

# Orders Bulkhead Demo (Node.js + Express)

---

*Resilience demo: Bulkhead pattern to isolate slow/failing dependencies and protect core API*

## Index

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

## 1. Project Overview

This application demonstrates the Bulkhead resilience pattern: limiting concurrency and isolating resource usage so one slow component (e.g., payment, SMS, email, 3rd-party API) does not take down the entire Orders API.

### Scenario

Under load, slow dependencies cause in-flight requests to grow. Eventually the server runs out of resources and becomes unresponsive. A bulkhead caps in-flight work and optionally queues a small number of requests; excess requests are rejected quickly.

## 2. Learning Outcomes

- Explain bulkhead using concurrency limits.
- Apply bulkhead around slow/unstable dependencies.
- Understand fast reject vs queue vs timeout behavior.
- Debug saturation and verify healthy endpoints remain responsive.

## 3. Tech Stack

package.json:

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

## 4. Folder Structure

```
- bulkhead.js
- package.json
- server.js
  - Order.js
  - orders.js
```

## 5. High-Level Architecture

Client → Express routes → Bulkhead limiter → slow service/dependency → response

Bulkhead protects CPU/memory and external connection pools by controlling concurrency.

## 6. Bulkhead Pattern in This App

Bulkhead is like having compartments on a ship: if one compartment floods, the ship stays afloat. In APIs, we isolate expensive/slow work behind a concurrency limit so the rest of the server stays healthy.

### Typical strategies:

- Semaphore (concurrency cap): allow only N in-flight calls.
- Queue + cap: wait up to M requests; reject beyond that.
- Fast reject: return 429/503 quickly when saturated.
- Timeout: cap how long a request can wait/run.

### Bulkhead implementation files detected:

- bulkhead.js

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

#### bulkhead.js

```
// Simple Bulkhead implementation
class Bulkhead {
  constructor(limit) {
    this.limit = limit;
    this.active = 0;
  }
}
```

#### package.json

```
{
  "name": "orders-bulkhead-demo",
  "version": "1.0.0",
  "type": "commonjs",
  "scripts": {
```

#### server.js

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

app.listen(3000, () => {
  console.log("Orders Bulkhead Demo running on http://localhost:3000");
});
```

#### routes/orders.js

```
const express = require("express");
const Order = require("../models/Order");
const Bulkhead = require("../bulkhead");

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

## 7. API Endpoints

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

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

## 8. How to Run + Quick Tests

### Install and run:

```
npm install
npm run dev
```

### Smoke test:

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

### Saturation test idea:

```
# Fire many parallel requests and observe behavior.
# With bulkhead, you should see fast rejects or bounded queueing instead of
server hanging.
# PowerShell example (Windows):
# 1..30 | % { Start-Job { iwr http://localhost:3000/<bulkhead-endpoint> } }
```

## 9. Common Mistakes

### No bulkhead limit

Slow dependency can consume all in-flight capacity; other requests starve.

### Limit too high

Still exhausts resources; behaves like no limit under load.

### No timeout/queue cap

Waiting requests pile up; memory grows; latency explodes.

### Not returning consistent errors

Clients need 429/503 and clear messages to back off.

### Putting bulkhead on every endpoint

Reduces throughput unnecessarily; apply to risky/slow paths.

## 10. Debugging Techniques

- Log in-flight count / queue length (if implemented) to confirm saturation.

- Add timing logs around the slow dependency to prove the bottleneck.
- Verify healthy endpoints remain responsive during a load test.
- Check Node event-loop lag: bulkhead helps only for I/O waits, not CPU blocking.

## Appendix B: Full Source Code

### bulkhead.js

```
// Simple Bulkhead implementation
class Bulkhead {
  constructor(limit) {
    this.limit = limit;
    this.active = 0;
  }

  async execute(task) {
    if (this.active >= this.limit) {
      throw new Error("Bulkhead limit exceeded");
    }

    this.active++;
    try {
      return await task();
    } finally {
      this.active--;
    }
  }
}

module.exports = Bulkhead;
```

### package.json

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

### server.js

```
const express = require("express");
const mongoose = require("mongoose");
```

```

const orderRoutes = require("../routes/orders");

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

mongoose.connect("mongodb://localhost:27017/orders_bulkhead_demo")
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error(err));

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

app.listen(3000, () => {
  console.log("Orders Bulkhead Demo running on http://localhost:3000");
});

```

## 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 Bulkhead = require("../bulkhead");

const router = express.Router();

// Bulkhead allowing only 2 concurrent operations
const paymentBulkhead = new Bulkhead(2);

// Simulated slow service
function fakePaymentService(orderId) {
  return new Promise((resolve) => {
    console.log("Processing payment for", orderId);
    setTimeout(() => resolve("PAID"), 3000);
  });
}

// CREATE ORDER (Bulkhead protected)
router.post("/", async (req, res) => {
  try {
    const order = await Order.create({

```

```

        product: req.body.product,
        amount: req.body.amount,
        status: "CREATED"
    });

    await paymentBulkhead.execute(async () => {
        const status = await fakePaymentService(order._id);
        order.status = status;
        await order.save();
    });

    res.status(201).json(order);
} catch (err) {
    res.status(429).json({
        error: err.message,
        message: "Too many concurrent requests. Please retry later."
    });
}
});

// READ ORDER
router.get("/:id", async (req, res) => {
    const order = await Order.findById(req.params.id);
    if (!order) return res.status(404).json({ error: "Order not found" });
    res.json(order);
});

module.exports = router;

```