Idempotency App (Node.js + Express + MongoDB)

Base reliability demo: idempotent order creation with a mock payment provider

Index

1. Project Overview

This application demonstrates a reliability pattern: idempotency for write APIs. The scenario is creating an order and charging a mock payment provider while preventing accidental double-charges when clients retry requests.

2. Learning Outcomes

- Explain idempotency for HTTP write operations.
- Persist idempotency keys in MongoDB with a unique compound index.
- Detect conflicting retries (same key but different body).
- Replay cached responses for completed requests.

3. Tech Stack

package.json (dependencies):

```
{
  "name": "reliability-demo",
  "version": "1.1.0",
  "main": "server.js",
  "scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js"
  },
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mongoose": "^8.4.0",
    "uuid": "^9.0.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.4"
  }
}
```

Environment (.env):

```
PORT=3000
MONGODB_URI=mongodb://localhost:27017/reliability_demo
```

Docker compose (MongoDB):

```
services:
  mongo:
    image: mongo:7
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
```

```
volumes:
  mongo_data:
```

4. Folder Structure

```
- .env
- docker-compose.yml
- package-lock.json
- package.json
- server.js
    - db.js
    - idempotency.js
    - IdempotencyKey.js
    - Order.js
    - admin.js
    - orders.js
    - paymentProvider.js
```

5. High-Level Architecture

Client → /orders (route) → idempotency middleware → MongoDB (orders + idempotencykeys) → mock payment provider → response

Retry path: same Idempotency-Key → return stored response (no duplicate effects).

Core components

Item	Details
server.js	Bootstraps Express, JSON parsing, request timing log, mounts routers, connects MongoDB.
src/routes/orders.js	Create order + list orders + get order by id.
src/middleware/idempotency.js	Stores request hash and response for idempotent replay.
src/models/Order.js	Order schema (status, payment details).
src/models/IdempotencyKey.js	Idempotency key store + unique compound index.
src/services/paymentProvider.js	Mock payment provider (delay + failure rate).
src/routes/admin.js	Admin endpoints to tune payment behavior.

6. Data Models

Order model

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

const PaymentSchema = new mongoose.Schema({
  provider: String,
  paymentId: String,
  amount: Number,
  currency: String,
  chargedAt: String
}, { _id: false });

const OrderSchema = new mongoose.Schema({
  userId: String,
  amount: Number,
  currency: String,
  status: {
    type: String,
    enum: ["CREATED", "PAID", "PAYMENT_FAILED"],
    default: "CREATED"
  },
  payment: PaymentSchema
}, { timestamps: true });

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

IdempotencyKey model

Unique index scope: (key, method, path, userId).

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

const IdempotencyKeySchema = new mongoose.Schema({
  key: String,
  method: String,
  path: String,
  userId: String,
  requestHash: String,
  status: { type: String, enum: ["IN_PROGRESS", "COMPLETED"], default:
"IN_PROGRESS" },
  responseStatus: Number,
  responseBody: mongoose.Schema.Types.Mixed
}, { timestamps: true });

IdempotencyKeySchema.index(
  { key: 1, method: 1, path: 1, userId: 1 },
  { unique: true }
);

module.exports = mongoose.model("IdempotencyKey", IdempotencyKeySchema);
```

7. API Endpoints (Orders + Admin)

Base URL: http://localhost:3000

Orders

Method	Path	Purpose	Body	Notes
POST	/orders	Create an order + attempt	JSON { userId, amount, currency }	Send Idempotency-Key header

		payment (idempotent)		
GET	/orders	List all orders	None	Returns { orders: [...] }
GET	/orders/:id	Get order by id	None	404 if not found

Admin (payment simulator)

Method	Path	Purpose	Body	Notes
GET	/admin/payment-behavior	Get payment provider behavior	None	Shows failureRate/delay
POST	/admin/payment-behavior	Set payment provider behavior	JSON fields	Use for failure/retry demos

Orders route code

```
const express = require("express");
const Order = require("../models/Order");
const { charge } = require("../services/paymentProvider");
const idempotency = require("../middleware/idempotency");

const router = express.Router();

router.post("/", idempotency(), async (req, res, next) => {
  try {
    const { userId, amount, currency } = req.body;

    const order = await Order.create({ userId, amount, currency });

    try {
      const payment = await charge({ orderId: order._id, amount, currency });
      order.status = "PAID";
      order.payment = payment;
      await order.save();
      res.status(201).json(order);
    } catch (e) {
      order.status = "PAYMENT_FAILED";
      await order.save();
      res.status(502).json({ error: "Payment failed", order });
    }
  } catch (err) {
    next(err);
  }
});

router.get("/", async (req, res) => {
  const orders = await Order.find();
  res.json({ orders });
});

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

module.exports = router;
```

Idempotency middleware

```
const crypto = require("crypto");
const IdempotencyKey = require("../models/IdempotencyKey");

function stableStringify(obj) {
  if (obj === null || typeof obj !== "object") return JSON.stringify(obj);
  if (Array.isArray(obj)) return "[" + obj.map(stableStringify).join(",") +
"]";
  const keys = Object.keys(obj).sort();
  return "{" + keys.map(k =>
JSON.stringify(k)+":"+stableStringify(obj[k])).join(",") + "}";
}

function hash(body) {
  return
crypto.createHash("sha256").update(stableStringify(body)).digest("hex");
}

module.exports = function idempotency() {
  return async function(req, res, next) {
    const key = req.header("Idempotency-Key");
    if (!key) return res.status(400).json({ error: "Missing Idempotency-Key"
});

    const filter = {
      key,
      method: req.method,
      path: req.baseUrl + req.path,
      userId: req.body?.userId || null
    };

    const reqHash = hash(req.body);
    let record = await IdempotencyKey.findOne(filter);

    if (record) {
      if (record.requestHash !== reqHash)
        return res.status(409).json({ error: "Payload mismatch for
idempotency key" });

      if (record.status === "COMPLETED")
        return res.status(record.responseStatus).json(record.responseBody);

      return res.status(202).json({ status: "IN_PROGRESS" });
    }

    await IdempotencyKey.create({ ...filter, requestHash: reqHash });

    const originalJson = res.json.bind(res);
    res.json = async (body) => {
      await IdempotencyKey.updateOne(
        filter,
        { status: "COMPLETED", responseStatus: res.statusCode, responseBody:
body }
      );
      return originalJson(body);
    };

    next();
  };
};
```

8. Idempotency Workflow (How it Works)

The middleware hashes the request body using a stable stringify + SHA-256. On the first request, it stores (key, method, path, userId, requestHash) as IN_PROGRESS. It wraps res.json so the first response is saved as COMPLETED (status + body). Retries return the stored response without re-running side effects.

Note: Some segments in the provided ZIP use '...' abbreviations; they are included verbatim.

9. How to Run + Quick Tests

Start MongoDB:

```
docker compose up -d
```

Run server:

```
npm install
npm run dev
```

Retry test:

```
curl -X POST http://localhost:3000/orders   -H "Content-Type:
application/json"   -H "Idempotency-Key: demo-key-123"   -d
'{"userId":"u1","amount":500,"currency":"INR"}'

curl -X POST http://localhost:3000/orders   -H "Content-Type:
application/json"   -H "Idempotency-Key: demo-key-123"   -d
'{"userId":"u1","amount":500,"currency":"INR"}'
```

10. Common Mistakes

No Idempotency-Key header

Same key reused for different actions

Body changed on retry

No unique index

Crash before caching response

11. Debugging Techniques

- Log the Idempotency-Key, method, path, and derived userId used for the filter.
- Inspect MongoDB collections: orders and idempotencykeys.
- Check status transitions: IN_PROGRESS → COMPLETED.
- Force payment failures via /admin/payment-behavior to observe retry behavior.

Mongo queries:

```
# mongosh
use reliability_demo
db.idempotencykeys.find().sort({createdAt:-1}).limit(5).pretty()
db.orders.find().sort({createdAt:-1}).limit(5).pretty()
```

Appendix A: Full Source Code (as provided)

.env

```
PORT=3000
MONGODB_URI=mongodb://localhost:27017/reliability_demo
```

docker-compose.yml

```
services:
  mongo:
    image: mongo:7
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
```

package.json

```
{
  "name": "reliability-demo",
  "version": "1.1.0",
  "main": "server.js",
  "scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js"
  },
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mongoose": "^8.4.0",
    "uuid": "^9.0.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.4"
  }
}
```

server.js

```
require("dotenv").config();
const express = require("express");
const connectDB = require("./src/config/db");

const ordersRouter = require("./src/routes/orders");
const adminRouter = require("./src/routes/admin");

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

app.use((req, res, next) => {
  const start = Date.now();
  res.on("finish", () => {
    console.log(`${req.method} ${req.originalUrl} -> ${res.statusCode}
(${Date.now() - start}ms)`);
  });
  next();
});

app.get("/health", (req, res) => res.json({ ok: true }));

app.use("/orders", ordersRouter);
app.use("/admin", adminRouter);

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

(async () => {
  await connectDB();
  app.listen(process.env.PORT || 3000, () =>
    console.log("Server running")
  );
})();
```

src/config/db.js

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

async function connectDB() {
  await mongoose.connect(process.env.MONGODB_URI);
  console.log("MongoDB connected");
}

module.exports = connectDB;
```

src/middleware/idempotency.js

```
const crypto = require("crypto");
const IdempotencyKey = require("../models/IdempotencyKey");

function stableStringify(obj) {
  if (obj === null || typeof obj !== "object") return JSON.stringify(obj);
  if (Array.isArray(obj)) return "[" + obj.map(stableStringify).join(",") +
"]";
  const keys = Object.keys(obj).sort();
  return "{" + keys.map(k =>
JSON.stringify(k)+":"+stableStringify(obj[k])).join(",") + "}";
}

function hash(body) {
  return
crypto.createHash("sha256").update(stableStringify(body)).digest("hex");
}

module.exports = function idempotency() {
  return async function(req, res, next) {
    const key = req.header("Idempotency-Key");
    if (!key) return res.status(400).json({ error: "Missing Idempotency-Key" 
});

    const filter = {
      key,
      method: req.method,
      path: req.baseUrl + req.path,
      userId: req.body?.userId || null
    };

    const reqHash = hash(req.body);
    let record = await IdempotencyKey.findOne(filter);

    if (record) {
      if (record.requestHash !== reqHash)
        return res.status(409).json({ error: "Payload mismatch for 
idempotency key" });

      if (record.status === "COMPLETED")
        return res.status(record.responseStatus).json(record.responseBody);

      return res.status(202).json({ status: "IN_PROGRESS" });
    }

    await IdempotencyKey.create({ ...filter, requestHash: reqHash });

    const originalJson = res.json.bind(res);
    res.json = async (body) => {
      await IdempotencyKey.updateOne(
        filter,
        { status: "COMPLETED", responseStatus: res.statusCode, responseBody: 
body }
      );
      return originalJson(body);
    };

    next();
  };
};
```

src/models/IdempotencyKey.js

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

const IdempotencyKeySchema = new mongoose.Schema({
  key: String,
  method: String,
  path: String,
  userId: String,
  requestHash: String,
  status: { type: String, enum: ["IN_PROGRESS", "COMPLETED"], default: 
"IN_PROGRESS" },
  responseStatus: Number,
  responseBody: mongoose.Schema.Types.Mixed
}, { timestamps: true });

IdempotencyKeySchema.index(
  { key: 1, method: 1, path: 1, userId: 1 },
  { unique: true }
);

module.exports = mongoose.model("IdempotencyKey", IdempotencyKeySchema);
```

src/models/Order.js

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

const PaymentSchema = new mongoose.Schema({
  provider: String,
  paymentId: String,
  amount: Number,
  currency: String,
  chargedAt: String
}, { _id: false });

const OrderSchema = new mongoose.Schema({
  userId: String,
  amount: Number,
  currency: String,
  status: {
    type: String,
    enum: ["CREATED", "PAID", "PAYMENT_FAILED"],
    default: "CREATED"
  },
  payment: PaymentSchema
}, { timestamps: true });

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

src/routes/admin.js

```
const express = require("express");
const { getBehavior, setBehavior } = require("../services/paymentProvider");

const router = express.Router();

router.get("/payment-behavior", (req, res) => {
  res.json(getBehavior());
});

router.post("/payment-behavior", (req, res) => {
  setBehavior(req.body || {});
  res.json(getBehavior());
});

module.exports = router;
```

src/routes/orders.js

```
const express = require("express");
const Order = require("../models/Order");
const { charge } = require("../services/paymentProvider");
const idempotency = require("../middleware/idempotency");

const router = express.Router();

router.post("/", idempotency(), async (req, res, next) => {
  try {
    const { userId, amount, currency } = req.body;

    const order = await Order.create({ userId, amount, currency });

    try {
      const payment = await charge({ orderId: order._id, amount, currency });
      order.status = "PAID";
      order.payment = payment;
      await order.save();
      res.status(201).json(order);
    } catch (e) {
      order.status = "PAYMENT_FAILED";
      await order.save();
      res.status(502).json({ error: "Payment failed", order });
    }
  } catch (err) {
    next(err);
  }
});

router.get("/", async (req, res) => {
  const orders = await Order.find();
  res.json({ orders });
});

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

module.exports = router;
```

src/services/paymentProvider.js

```
const state = { failureRate: 0, minDelayMs: 50, maxDelayMs: 150 };

function sleep(ms) { return new Promise(r => setTimeout(r, ms)); }

async function charge({ orderId, amount, currency }) {
  const delay = Math.floor(Math.random() * (state.maxDelayMs -
state.minDelayMs + 1)) + state.minDelayMs;
  await sleep(delay);
  if (Math.random() < state.failureRate) {
    const err = new Error("Payment provider failure");
    err.code = "PAYMENT_PROVIDER_DOWN";
    throw err;
  }

  return {
    provider: "MockPay",
    paymentId: `pay_${orderId}_${Date.now()}`,
    amount,
    currency,
    chargedAt: new Date().toISOString()
  };
}

function setBehavior(b) { Object.assign(state, b); }
function getBehavior() { return state; }

module.exports = { charge, setBehavior, getBehavior };
```