Orders App - Simple CRUD (Base Project)

Index

- 1. Project overview
- 2. Learning outcomes
- 3. Scope
- 4. Tech stack & versions
- 5. Folder structure
- 6. High-level request flow
- 7. Data model (Order)
- 8. API contract
- 9. Setup & run
- 10. How the code works (file-by-file)
- 11. Common mistakes
- 12. Debugging techniques
- 13. Appendix A - Full source code

1. Project overview

This is a starter Express + MongoDB application that demonstrates a simple Orders API with plain CRUD operations. It is intended to be a clean base for future add-ons, without changing the fundamentals.

2. Learning outcomes

- Create a REST API using Express routers.
- Connect an app to MongoDB using Mongoose.
- Implement CRUD endpoints using GET, POST, PUT, PATCH, DELETE.
- Understand PUT (full update) vs PATCH (partial update).
- Practice status codes and error handling.
- Use a controllable fake payment provider to simulate success/failure.

3. Scope

In-scope:

- Health check endpoint: GET /health
- Orders API: /orders (CRUD)
- Admin API: /admin/payment-behavior (control fake payment provider)
- MongoDB via Docker Compose
- Minimal request timing log middleware

Out-of-scope (future add-ons):

- Authentication/authorization
- Advanced validation layer (Joi/Zod), DTOs
- Pagination/filtering/sorting
- Rate limiting, security headers, production logging
- Automated tests

4. Tech stack & versions

Framework	Express 4
Database	MongoDB (docker image mongo:7)
ODM	Mongoose 8
Config	dotenv
Dev	nodemon

5. Folder structure

```
Orders-App/
  server.js
  package.json
```

```
docker-compose.yml
.env
src/
  config/db.js
  models/Order.js
  routes/orders.js
  routes/admin.js
  services/paymentProvider.js
```

6. High-level request flow

1. Client sends HTTP request (JSON).
2. Express parses JSON using express.json().
3. Router handles route (/orders or /admin).
4. Mongoose reads/writes Order documents in MongoDB.
5. POST /orders attempts a fake payment charge and updates status.
6. Response returns status code + JSON body.

7. Data model (Order)

userId	String
amount	Number
currency	String
status	Enum: CREATED, PAID, PAYMENT_FAILED (default CREATED)
payment	Embedded doc set on success
createdAt / updatedAt	Auto timestamps

8. API contract

Base URL: http://localhost:3000

Method	Route	Purpose	Notes
GET	/health	Health check	{ ok: true }
GET	/orders	List orders	Order.find(); no pagination
GET	/orders/:id	Get order	404 if not found
POST	/orders	Create order + attempt payment	201 or 502
PUT	/orders/:id	Full update	Requires userId, amount, currency
PATCH	/orders/:id	Partial update	Allows userId, amount, currency, status
DELETE	/orders/:id	Delete order	{ message: 'Deleted' }
GET	/admin/payment-behavior	View payment provider behavior	failureRate/delay

POST	/admin/payment-behavior	Update payment provider behavior	Send JSON body

Examples

```
curl -X POST http://localhost:3000/orders \
  -H "Content-Type: application/json" \
  -d '{ "userId": "u1", "amount": 499, "currency": "INR" }'

curl -X PATCH http://localhost:3000/orders/<id> \
  -H "Content-Type: application/json" \
  -d '{ "amount": 799 }'

curl -X POST http://localhost:3000/admin/payment-behavior \
  -H "Content-Type: application/json" \
  -d '{ "failureRate": 1 }'
```

9. Setup & run

7. npm install
8. docker compose up -d
9. verify .env (MONGODB_URI)
10. npm run dev (or npm start)
11. curl http://localhost:3000/health

```
docker compose up -d
npm run dev
curl http://localhost:3000/health
```

10. How the code works (file-by-file)

server.js	Bootstraps Express, mounts routers, global error handler, starts server after DB connect.
src/config/db.js	MongoDB connection using MONGODB_URI.
src/models/Order.js	Order schema + embedded payment schema.
src/routes/orders.js	CRUD for /orders + payment attempt on create.
src/routes/admin.js	Controls paymentProvider behavior.
src/services/paymentProvider.js	Fake provider: random delay + failureRate.

11. Common mistakes

- Missing Content-Type: application/json (req.body empty).
- MongoDB not running or wrong MONGODB_URI.
- Invalid ObjectId in /orders/:id causing CastError (often seen as 500).
- PUT used like PATCH (PUT requires core fields here).
- Not handling 502 from POST /orders (payment failure).
- Invalid status value not in enum.
- Not awaiting async calls.

- Assuming delete is always safe (future audit logging may change).
- Not checking status codes in client code.
- PORT already in use.

12. Debugging techniques

Use this checklist in order:

12. Confirm MongoDB is running and accessible on 27017.
13. Confirm server boot logs: 'MongoDB connected' then 'Server running on port ...'.
14. Call GET /health.
15. Reproduce the failing call using curl/Postman and inspect status + JSON.
16. Verify ObjectId is valid and order exists for /orders/:id.
17. Temporarily log req.body to verify payload.
18. Use /admin/payment-behavior to force payment failure and test client handling.
19. Check MongoDB data with mongosh/Compass.

```
docker ps
mongosh "mongodb://localhost:27017/reliability_demo"
db.orders.find().limit(5)
```

13. Appendix A - Full source code

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" });
});

const PORT = process.env.PORT || 3000;
```

```
(async () => {
  await connectDB();
  app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
})();
```

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/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/orders.js

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

const router = express.Router();

router.post("/", 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);
});

// PUT = full update (client sends all fields)
router.put("/:id", async (req, res) => {
  try {
    const { userId, amount, currency, status } = req.body;

    // keep it simple: require the core fields
    if (!userId || amount === undefined || !currency) {
      return res.status(400).json({
        error: "userId, amount, currency are required for PUT",
      });
    }

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

    order.userId = userId;
    order.amount = amount;
    order.currency = currency;
    if (status !== undefined) order.status = status;

    await order.save();
    res.json(order);
  } catch (err) {
    res.status(500).json({ error: "Server error" });
  }
});

// PATCH = partial update (client sends only changed fields)
router.patch("/:id", async (req, res) => {
  try {
    const allowed = ["userId", "amount", "currency", "status"];
    const updates = {};

    for (const key of allowed) {
      if (req.body[key] !== undefined) updates[key] = req.body[key];
    }

    const order = await Order.findByIdAndUpdate(req.params.id, updates, {
      new: true,
      runValidators: true,
    });

    if (!order) return res.status(404).json({ error: "Not found" });
    res.json(order);
  } catch (err) {
    res.status(500).json({ error: "Server error" });
  }
});

router.delete("/:id", async (req, res) => {
  try {
    const order = await Order.findByIdAndDelete(req.params.id);
    if (!order) return res.status(404).json({ error: "Not found" });
    res.json({ message: "Deleted" });
  } catch (err) {
    res.status(500).json({ error: "Server error" });
  }
});

module.exports = router;
```

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/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 };
```

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": "1.0.0",
  "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"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

.env

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