Order Pagination Demo (Node.js + Express + MongoDB) Documentation + Source Code

Index

1. Project Overview

This application demonstrates how to paginate large result sets in a REST API using Orders as the example domain. Pagination prevents huge responses, improves performance, and enables stable UX for page navigation and infinite scroll.

2. Learning Outcomes

- Explain why pagination is required for list endpoints.
- Implement offset (page/limit) pagination safely.
- Explain cursor pagination and when it is more stable than offset pagination.
- Avoid common bugs: missing sort, duplicated/missed records, and missing indexes.

3. Tech Stack

package.json:

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

4. Folder Structure

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

5. High-Level Architecture

Client → GET /orders with pagination params → Mongoose query (sort + limit/skip or cursor filter) → MongoDB → response

Key requirement: Always sort deterministically (e.g., createdAt desc, _id desc) so pages are consistent.

6. Pagination Concepts Used in This App

Two common pagination styles:

- Offset pagination (page & limit): easy for page numbers, but can duplicate/miss items if data changes between requests.
- Cursor pagination (nextCursor): better for infinite scroll; uses a filter 'after/before cursor' with deterministic sort.

Sorting rule of thumb:

Sort by a stable field (createdAt) plus a tie-breaker (_id).

Pagination-related snippets found in your code:

routes/orders.js

```
// OFFSET-BASED PAGINATION
// GET /orders/offset?page=1&limit=10
router.get("/offset", async (req, res) => {
  const page = parseInt(req.query.page || "1");
  const limit = parseInt(req.query.limit || "10");
```

7. API Endpoints

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

Method	Path	Source File	Notes
POST	/orders/seed	routes/orders.js	
GET	/orders/offset	routes/orders.js	
GET	/orders/cursor	routes/orders.js	

8. How to Run + Quick Tests

Start MongoDB:

```
docker compose up -d
```

Run server:

```
npm install
npm run dev
```

Offset pagination example:

```
curl -i 'http://localhost:3000/orders?page=1&limit=10'
```

Cursor pagination example (concept):

```
curl -i 'http://localhost:3000/orders?limit=10&cursor=<LAST_SEEN_CURSOR>'
```

9. Common Mistakes

No deterministic sort

Without sort, ordering is arbitrary → duplicated/missing items across pages.

Deep skip with large page numbers

skip gets slower for big offsets; cursor pagination is better for infinite scroll.

Not validating page/limit

Negative or huge values can stress your DB and API.

Cursor not aligned with sort

Cursor must represent the sorted fields; otherwise pages drift.

Missing indexes

Sorting/filtering without indexes becomes slow as data grows.

10. Debugging Techniques

- Enable Mongoose debug logs to see sort/skip/limit in the query.
- Insert new orders between page fetches to observe how offset pagination shifts items.
- Confirm the 'nextCursor' is derived from the last item returned.
- Use MongoDB explain() to confirm index usage for sort/filter.

Appendix A: Pagination-Related Code Snippets

routes/orders.js

```
// OFFSET-BASED PAGINATION
// GET /orders/offset?page=1&limit=10
router.get("/offset", async (req, res) => {
  const page = parseInt(req.query.page || "1");
  const limit = parseInt(req.query.limit || "10");
```

Full Source Code

package.json

```
{
  "name": "order-pagination-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/order_pagination_demo")
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error(err));

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

app.listen(3000, () =>
  console.log("Order Pagination 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 router = express.Router();

// Seed orders for demo
router.post("/seed", async (req, res) => {
  await Order.deleteMany({});
  const orders = [];
  for (let i = 1; i <= 50; i++) {
    orders.push({
      product: "Product-" + i,
      amount: 100 + i,
      status: "CREATED"
    });
  }
  await Order.insertMany(orders);
  res.json({ message: "50 orders seeded" });
});

// OFFSET-BASED PAGINATION
// GET /orders/offset?page=1&limit=10
router.get("/offset", async (req, res) => {
  const page = parseInt(req.query.page || "1");
  const limit = parseInt(req.query.limit || "10");
  const skip = (page - 1) * limit;

  const orders = await Order.find()
    .sort({ createdAt: -1 })
    .skip(skip)
    .limit(limit);

  const total = await Order.countDocuments();

  res.json({
    paginationType: "offset",
    page,
    limit,
    total,
    data: orders
  });
});

// CURSOR-BASED PAGINATION
// GET /orders/cursor?cursor=<createdAt>&limit=10
router.get("/cursor", async (req, res) => {
  const limit = parseInt(req.query.limit || "10");
  const cursor = req.query.cursor;

  const query = cursor
    ? { createdAt: { $lt: new Date(cursor) } }
    : {};
  const orders = await Order.find(query)
    .sort({ createdAt: -1 })
    .limit(limit);

  const nextCursor =
    orders.length > 0
      ? orders[orders.length - 1].createdAt
      : null;

  res.json({
    paginationType: "cursor",
    limit,
    nextCursor,
    data: orders
  });
});

module.exports = router;
```