Pagination in Express.js — Step by Step Guide

A practical, project-oriented guide with LAB25_PAGINATION example.

1) What is Pagination?

Pagination is the process of splitting a large dataset into smaller, numbered pages instead of sending everything at once. Websites like YouTube, Amazon, and Instagram use pagination to display a limited number of results per page.

2) Why use Pagination?

- Improves performance and response time
- Reduces bandwidth usage
- Provides better user experience
- Prevents crashing or freezing on large data loads

3) Core Pagination Math

```
page  = Number(req.query.page)  || 1
limit = Number(req.query.limit) || 10
skip  = (page - 1) * limit
```

4) Project Scaffold (LAB25_PAGINATION)

```
LAB25_PAGINATION/
■■ server.js        # Express backend
■■ data.json        # Sample dataset
■■ index.html       # Frontend (fetch + script)
```

5) Sample Data (data.json)

```
[
  { "id": 1, "title": "Learn JavaScript" },
  { "id": 2, "title": "Master CSS" },
  { "id": 3, "title": "React Crash Course" },
  { "id": 4, "title": "Node.js API Tutorial" },
  { "id": 5, "title": "MongoDB Basics" },
  { "id": 6, "title": "Express Pagination" }
]
```

6) Express Backend (server.js)

```
const express = require("express");
const fs = require("fs");
const path = require("path");

const app = express();
const PORT = 3000;

// Serve frontend files
app.use(express.static(__dirname));

// Load data
const data = JSON.parse(fs.readFileSync(path.join(__dirname, "data.json")));

// Pagination API
app.get("/api/items", (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 5;

  const start = (page - 1) * limit;
  const end = start + limit;

  const items = data.slice(start, end);
```

```
  res.json({
    page,
    totalPages: Math.ceil(data.length / limit),
    totalItems: data.length,
    items
  });
});

app.listen(PORT, () => console.log(`Server running at http://localhost:${PORT}`));
```

7) Frontend (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>LAB25_PAGINATION</title>
  <style>
    body { font-family: Arial; padding: 20px; }
    .item { padding: 8px; border-bottom: 1px solid #ddd; }
    .pagination { margin-top: 12px; display: flex; gap: 6px; }
    .pagination button { padding: 6px 10px; }
    .pagination button.active { background: #333; color: #fff; }
  </style>
</head>
<body>
  <h1>LAB25_PAGINATION</h1>
  <div id="items"></div>
  <div id="pagination" class="pagination"></div>

  <script>
    const itemsContainer = document.getElementById("items");
    const paginationContainer = document.getElementById("pagination");

    async function fetchData(page = 1) {
      const res = await fetch(`/api/items?page=${page}&limit=4`);
      const data = await res.json();
      renderItems(data.items);
      renderPagination(data.page, data.totalPages);
    }

    function renderItems(items) {
      itemsContainer.innerHTML = "";
      items.forEach(item => {
        const div = document.createElement("div");
        div.className = "item";
        div.textContent = item.id + ". " + item.title;
        itemsContainer.appendChild(div);
      });
    }

    function renderPagination(current, totalPages) {
      paginationContainer.innerHTML = "";
      for (let i = 1; i <= totalPages; i++) {
        const btn = document.createElement("button");
        btn.textContent = i;
        if (i === current) btn.classList.add("active");
        btn.onclick = () => fetchData(i);
        paginationContainer.appendChild(btn);
      }
    }

    fetchData(1);
  </script>
</body>
</html>
```

8) Testing

1. Install dependencies: `npm init -y && npm install express`
2. Run: `node server.js`
3. Open: http://localhost:3000
4. API Example: http://localhost:3000/api/items?page=2&limit;=4

9) Advanced Example: MongoDB/Mongoose Pagination

```
// Assuming a User model with Mongoose
app.get("/api/users", async (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 10;
  const skip = (page - 1) * limit;

  const [items, totalItems] = await Promise.all([
    User.find().skip(skip).limit(limit),
    User.countDocuments()
  ]);

  res.json({
    page,
    totalPages: Math.ceil(totalItems / limit),
    totalItems,
    items
  });
});
```

10) Best Practices

• Always return totalItems and totalPages.
• Validate page and limit query params.
• Disable Prev/Next buttons when needed.
• Use indexes in DB for performance.
• Allow server-side filters and sorting along with pagination.