LAPORAN MINI PROJECT BASIS DATA

FOLIO

Tiered Dynamic Bookstore Catalog
Implementasi Arsitektur Basis Data NoSQL
(MongoDB Document Store)

Disusun Oleh (Kelompok 5):

Andhika Fadhlan W. (2306267164)
Marshal Aufa D. (2406346913)
Reinathan Ezkhiel K. (2406397675)
Arkaan Pasya S. (2406408073)
Diandra Pramesti W. (2406342360)

Mata Kuliah: Sistem Basis Data
Topik: NoSQL Document Store & Data Tiering
Platform: MongoDB Atlas & Express.js

Departemen Teknik Elektro
Fakultas Teknik
Universitas Indonesia

2026

0 Daftar Isi

1 Pendahuluan & Konteks Studi Kasus

Folio adalah sistem katalog toko buku tingkat lanjut (*advanced bookstore information system*) yang dirancang untuk memecahkan tantangan skalabilitas, fleksibilitas skema, dan performa tinggi pada *e-commerce* ritel berskala masif. Proyek ini mengadopsi paradigma **NoSQL Document Store (MongoDB)** untuk menjawab limitasi konvensional.

1.1 Real-World Case Study: Periplus

Sebagai inspirasi arsitektur, Periplus—toko buku daring terbesar di Indonesia—mengelola ratusan ribu *Stock Keeping Units* (SKU) dengan variabilitas atribut yang sangat tinggi:

- **Buku Baru / Kaya Fitur:** Hadir dengan ulasan interaktif, file digital (*e-book, voucher, CD resource*), dan metadata sangat detail.
- **Buku Lama / Arsip:** Hanya memiliki metadata dasar (judul, penulis, tahun). Tidak membutuhkan *field* digital atau sistem ulasan berkala.

1.2 Kelemahan RDBMS Konvensional

Penggunaan RDBMS pada skenario di atas mengharuskan skema tabel yang seragam secara rigid:

1. **Pemborosan Memori (Null Padding):** Jutaan baris buku lama terpaksa diisi dengan nilai `NULL` untuk kolom-kolom modern, memboroskan memori disk dan merusak *index lookup*.
2. **Overhead Migrasi Skema (`ALTER TABLE`):** Penambahan satu atribut baru pada edisi buku terbaru mengharuskan eksekusi `ALTER TABLE` pada jutaan baris, memicu *downtime* dan risiko kehilangan data.

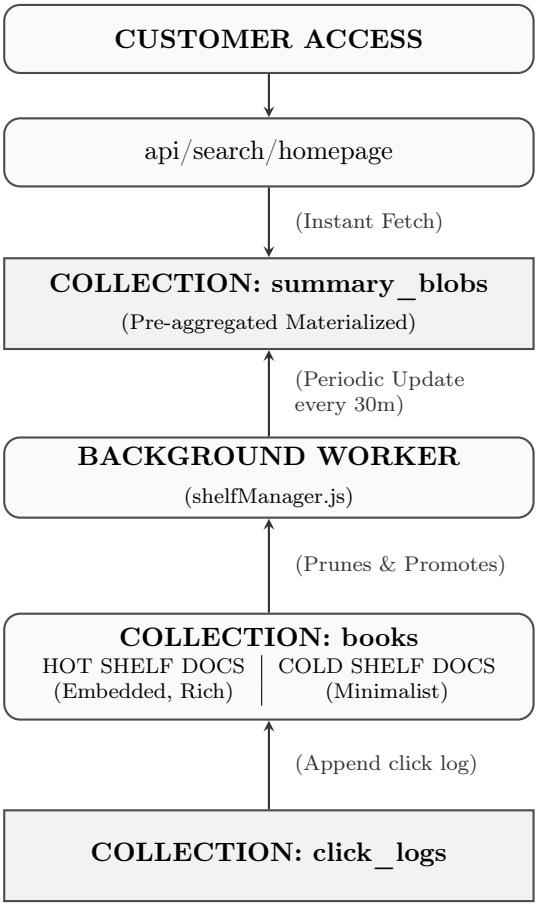
1.3 Solusi Document Store (MongoDB)

MongoDB menjawab kendala di atas melalui fitur bawaannya (*native*):

- **Schema Flexibility:** Dokumen dapat memiliki struktur *field* berbeda secara organik. Buku baru dan lama hidup berdampingan tanpa *NULL-padding*.
- **Embedding & Referencing:** Memungkinkan data ulasan di-*embed* langsung, mengurangi *round-trip query database*.
- **Materialized View Native Support:** Pola *pre-aggregated summary* diimplementasikan sangat natural tanpa terikat skema kaku.

2 Arsitektur Sistem & Fitur Utama

2.1 Arsitektur Aliran Data Tiering



Gambar 1: Diagram Aliran Data Folio

2.2 Dynamic Data Tiering (Hot Shelf vs Cold Shelf)

Inventaris buku dibagi menjadi dua tingkatan:

- **Hot Shelf (Active Tier):** Skema kaya fitur. Atribut digital, ulasan ter-*embed* (maks. 5), `average_rating` pra-kalkulasi, dan log klik aktif. Dioptimalkan untuk akses baca instan.
- **Cold Shelf (Archived Tier):** Dokumen *mounting* minimalis untuk buku lama. Hanya esensial (`title, author, price, stock, genres, year`). *Field* opsional tidak diinisialisasi untuk efisiensi massal.

2.3 Automated Shelf Promotion (Promosi Rak Otomatis)

Jika klik buku *Cold Shelf* melampaui ambang batas (misal: > 500 klik/30 hari), *background worker* otomatis memicu promosi: menambahkan field fitur, ulasan, dan mengubah status menjadi `"hot"`.

2.4 Materialized Views via "Summary Blobs"

Sistem tidak mengeksekusi `JOIN` atau `SORT` pada jutaan buku saat *user* menavigasi kategori. Sebaliknya, *query* langsung menuju `summary_blobs` yang berisi data buku semi-*embed* ber-

metrik tertinggi. Halaman katalog termuat nyaris tanpa *loading*.

2.5 Bayesian 3D Weighted Rating & Sentiment Analysis

Sistem mencegah manipulasi *rating* melalui pendekatan matematis dan bahasa:

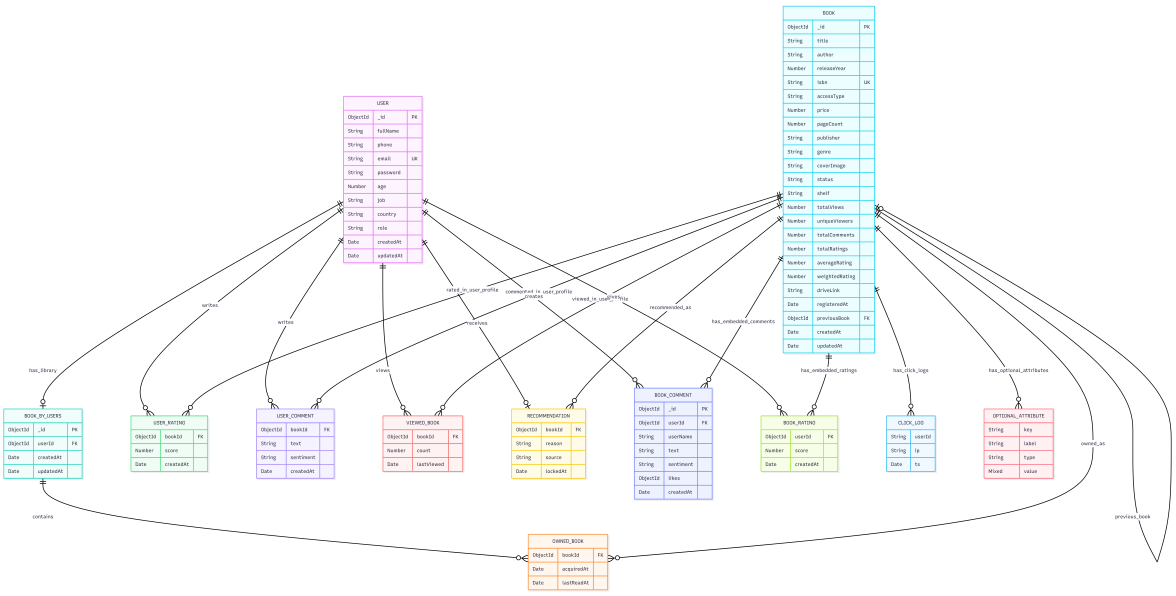
$$\text{Weighted Score} = \left(\frac{v \cdot R + m \cdot C}{v + m} \right) \times \text{Time Decay Factor} \tag{1}$$

v = jumlah ulasan, R = rata-rata ulasan, m = threshold minimal, C = rata-rata semesta katalog. **Time Decay** memberi bobot lebih bagi tren terbaru, dan **Regex Sentiment** mengoreksi skor ulasan palsu bersentimen negatif yang berbintang lima.

3 Conceptual Data Model (Struktur Skema JSON)

Sistem menerapkan *schema-less* dengan kombinasi *Embedding*, *Referencing*, dan *Materialized Views*.

3.1 Entity-Relationship Abstraction



Gambar 2: Pemetaan Relasi Entitas (ERD Abstraction)

3.2 Contoh Model Dokumen Skema JSON

A. Hot Shelf (Buku Baru)
Limit *embedding* komentar digunakan untuk mencegah *unbounded array problem* melebihi batas 16 MB.

```
{
  "_id": "ObjectId('book_hot_001')",
  "title": "Mastering Distributed Systems",
  "author": "Jane Doe",
  "price": 320000,
  "shelf_status": "hot",
  "click_count": 1280,
  "features": ["E-Book", "CD-Resource", "Access_Voucher"],
```

```
  "average_rating": 4.7,
  "recent_comments": [
    {"user": "Wahib", "rating": 5, "text": "Sangat komprehensif!"},
    {"user": "Coki", "rating": 4, "text": "Bagus untuk referensi."}
  ],
  "has_more_comments": true
}
```

B. Cold Shelf (Data Mounting)

Atribut direduksi hingga ukuran terkecil.

```
{
  "_id": "ObjectId('book_cold_999')",
  "title": "Sejarah DOS 1.0",
  "author": "John Smith",
  "price": 85000,
  "shelf_status": "cold",
  "click_count": 12,
  "genres": ["History", "Technology"]
}
```

C. Referencing Collection (Comments)

Digunakan saat `has_more_comments = true` di Hot Shelf.

```
{
  "_id": "ObjectId('comment_001')",
  "book_id": "ObjectId('book_hot_001')",
  "user": "Gorga",
  "rating": 5,
  "text": "Penjelasan materinya sangat detail.",
  "created_at": "ISODate('2026-04-20T08:30:00Z')"
}
```

D. Click Logs (Pemicu Background Worker)

Digunakan *background worker* untuk memperbarui agregasi *summary blobs* dan promosi *Hot Shelf*.

```
{
  "_id": "ObjectId('log_001')",
  "book_id": "ObjectId('book_hot_001')",
  "user_session": "sess_abc123",
  "clicked_at": "ISODate('2026-04-28T09:45:00Z')"
}
```

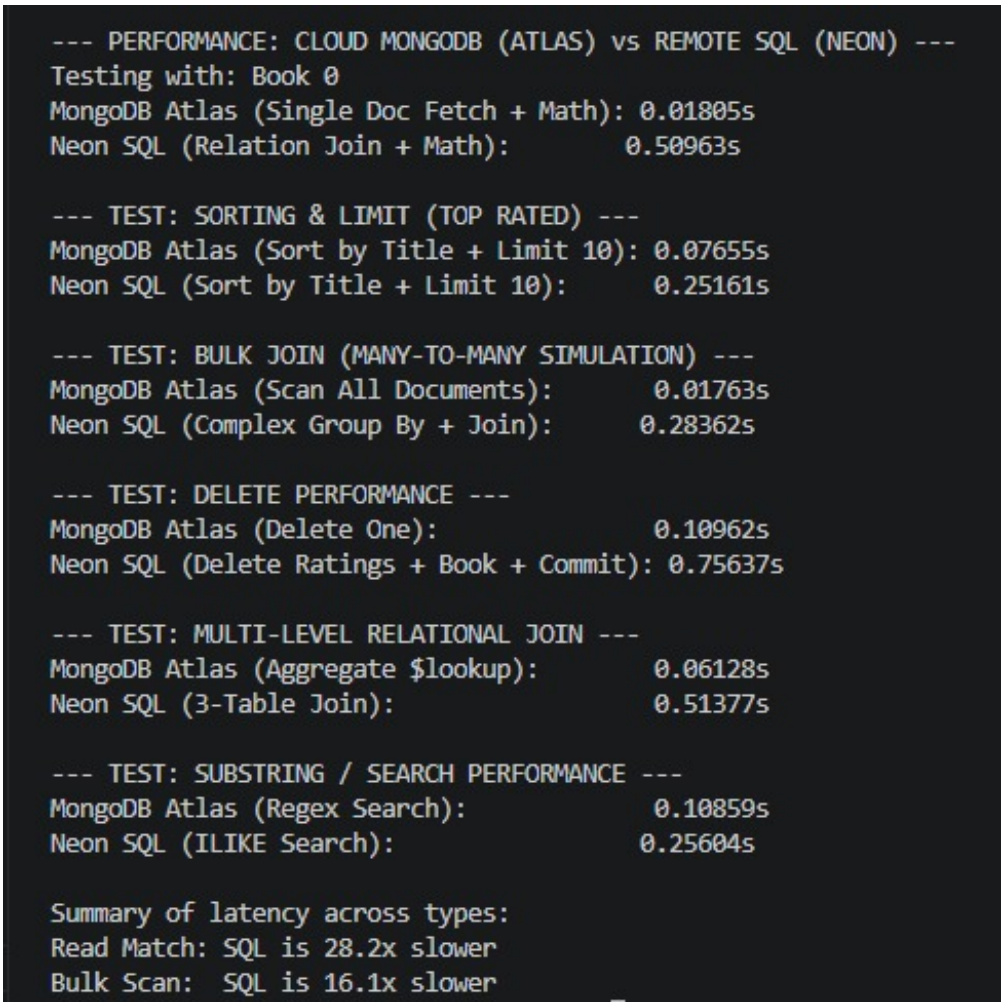
E. Materialized View (Summary Blobs)

Data *rendered* instan tanpa `JOIN`.

```
{
  "_id": "ObjectId('blob_time_rating_2026')",
  "blob_type": "waktu_rating",
  "primary_topic": "waktu",
  "average_click_rate": 15420,
  "last_refreshed": "ISODate('2026-04-28T09:50:00Z')",
  "books_embedded": [
    {
      "title": "Mastering Distributed Systems",
      "author": "Jane Doe",
      "average_rating": 4.9,
      "price": 320000,
      "book_id": "ObjectId('book_hot_001')"
    }
  ]
}
```

4 Hasil Uji Performa Mendalam (Benchmark)

Pengujian dijalankan via *script* Python pada arsitektur *cloud*: **MongoDB Atlas vs Neon SQL (PostgreSQL Serverless)**.



Gambar 3: Grafik Hasil Benchmark (MongoDB vs Neon SQL)

Tabel 1: Analisis Teknis Skenario Uji Stres Sistem

Skenario	Analisis & Hasil Komparasi Latensi
Read & Calc	**SQL 28.2x lebih lambat**. Pada SQL, agregasi `JOIN` sangat mahal saat *runtime*. MongoDB hanya membutuhkan satu kali tarikan dokumen (*Single Doc Fetch*).
Sort & Limit	**SQL 3.2x lebih lambat**. B-Tree index pada struktur field tunggal MongoDB jauh mengungguli pengerjaan *Execution Plan* relasional RDBMS.
Bulk Scan	**SQL 16.1x lebih lambat**. Skenario *many-to-many* `LEFT JOIN + GROUP BY` memaksa *Full Table Scan* besar-besaran pada memori server SQL.
Delete Target	**SQL 6.9x lebih lambat**. Constraint *Foreign Key* memaksa *Cascade Cleanup* lapis ganda di SQL sebelum akhirnya melakukan *COMMIT*.
3-Table Join	**SQL 8.4x lebih lambat**. MongoDB memacu fungsi agregasi `$lookup` di atas mesin *WiredTiger* mengalahkan *nested loop join* silang indeks di SQL.
Substring	**SQL (`ILIKE`) 2.3x lebih lambat**. Pemindaian string bebas (*wildcard*) gagal di-indeks B-Tree SQL, sementara MongoDB didukung *native* PCRE Regex.

5 Abstraksi Sistem dan Rute Integrasi

5.1 Pohon Direktori Logika Backend

```
backend/
|-- server.js              (trust proxy, cron shelf eval tiap 30 menit)
|-- models/
|   |-- User.js            (hidden metrics, genre prefs)
|   |-- Book.js            (cold/hot shelf, click logs, optional attrs)
|   `-- BookByUsers.js     (wide-column antisipasi, mass-assignable)
|-- middleware/
|   |-- auth.js            (JWT guard)
|   `-- adminOnly.js       (admin role guard)
|-- controllers/
|   |-- authController.js  (register validasi password ketat, login)
|   |-- bookController.js  (add/archive/delete recycle, click, comment, rate)
|   |-- userController.js  (profile, library)
|   `-- searchController.js(homepage random, search paginated, views)
`-- utils/
    |-- seedAdmin.js       (admin@gmail.com seeded on startup)
    |-- shelfManager.js    (prune click logs, embed prune, top-10 eval)
    |-- weightedRating.js  (Bayesian 3D + sentiment regex)
    `-- materializeViews.js(7 agregasi view + hotBooks + mostCommented)
```

5.2 Endpoint API Unggulan

Method	URL Endpoint	Auth Guard
POST	`/api/auth/register`	—
POST	`/api/auth/login`	—
GET	`/api/books/:id`	—
POST	`/api/books/add`	Admin
PATCH	`/api/books/:id/archive`	Admin
DELETE	`/api/books/:id`	Admin
POST	`/api/books/:id/click`	User

Method	URL Endpoint	Auth Guard
POST	`/api/books/:id/rate`	User
POST	`/api/books/:id/checkout`	User
GET	`/api/search/homepage`	—
GET	`/api/search/view/:viewType`	—

5.3 Struktur Antarmuka Frontend (React/Vite)

```
frontend/src/
|-- api/axios.js           (JWT interceptor + auto-logout)
|-- components/
|   |-- Navbar.jsx         (3-tier Gramedia navbar + utility bar)
|   |-- BookCard.jsx       (3:4 ratio card, hover scale, hot badge)
|   `-- StarRating.jsx     (full/half/empty stars)
`-- pages/
    |-- HomePage.jsx       (hero slider + 4 section grid)
    |-- SearchPage.jsx     (sidebar filter + pagination)
    |-- BookDetailPage.jsx (dynamic attributes, rating, admin controls)
    |-- RegisterPage.jsx   (live password checklist, dropdown negara)
    |-- ProfilePage.jsx    (biodata + horizontal library)
    |-- LibraryPage.jsx    (full library grid)
    `-- AdminPage.jsx      (form tambah buku + genre multi-select)
```

6 Role Distribution (Anggota Kelompok 5)

Tabel 3: Tanggung Jawab Area Teknis Kelompok 5

Nama Lengkap	NPM	Peran & Tanggung Jawab Teknis
Andhika Fadhlan W.	2306267164	Backend Development & Database Architecture
Marshal Aufa D.	2406346913	Backend Development & API Integration
Reinathan Ezkhiel K.	2406397675	Data Engineering & Python Benchmarking Script
Arkaan Pasya S.	2406408073	Frontend Development & State Management React
Diandra Pramesti W.	2406342360	Frontend Development & Comprehensive System Testing

A Lampiran: Panduan Instalasi dan Operasi

A.1 Kloning dan Eksekusi Server Node.js

```
# Masuk ke direktori backend
cd c:\Github\folio\backend

# Instalasi dependensi
npm install

# Jalankan server
npm run dev
# Server mendeteksi '.env', menyalakan Port 5000,
# auto-seeding admin, dan scheduler cron menyala otomatis.
```

A.2 Instalasi Antarmuka React Vite

```
# Pindah ke direktori frontend
cd ../frontend
npm install
npm run dev
# Buka http://localhost:5173 di browser
```

A.3 Menjalankan Uji Perbandingan Performa (Python)

Pastikan *compiler* Python terpasang di sistem.

```
cd ../backend

# Pasang dependensi driver DB
pip install psycopg2-binary pymongo python-dotenv

# Jalankan eksekusi benchmark Head-to-Head
python compare_neon.py
```