



Music Store – MongoDB NoSQL Project Report

Course: Advanced Databases (NoSQL)

Student Name: Abubakir Elnur

Group: BDA-2407

Instructor: Zhunissova Dinara

Database: MongoDB

Backend: Node.js, Express.js

Frontend: HTML, JavaScript, CSS

Authentication: JWT

Project Overview

This project is a web-based music store developed as a final project for the Advanced Databases (NoSQL) course. The system uses Node.js for the backend and MongoDB as the database to handle users, products, and orders. The project also includes a simple frontend to display products, manage orders, and allow users to log in and interact with the backend API.

Main Features:

- JWT-based Authentication: Users can log in to access their profiles and place orders.
- Product Catalog: Products include guitars, pianos, and accessories. Users can view product details such as price, brand, and description.
- Order Management: Users can create orders, view their order history, and interact with products.
- Admin Features: Admins can view sales statistics and manage products.
- CRUD Operations: Create, Read, Update, and Delete operations for users, products, and orders.

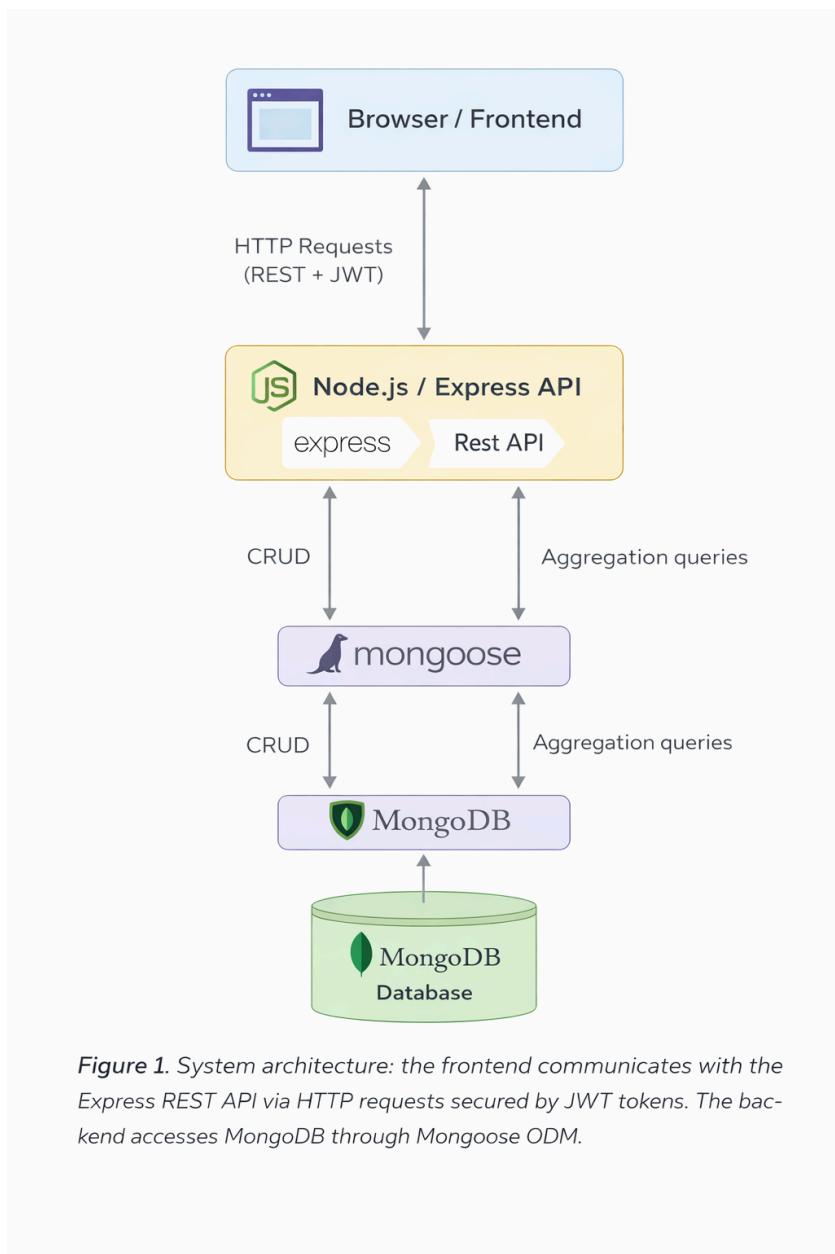
System Architecture

Components:

- Frontend: A simple interface built with HTML, CSS, and JavaScript to interact with the backend API.

- Backend: A server built using Node.js and Express that handles RESTful API requests.
- Database: The backend uses MongoDB to store users, products, and orders.
- Authentication: The system uses JWT (JSON Web Token) to authenticate and authorize users and admins.

Architecture Diagram:



Database Structure

The application uses MongoDB to store data across three main collections:

1. Users Collection

This collection stores information about users and their roles.

```
{
  "_id": ObjectId,
  "name": "John Doe",
  "email": "test1@mail.com",
```

```

"passwordHash": "hashedPassword",
"role": "user"
}

```

2. Products Collection

This collection stores information about musical instruments such as guitars, pianos, and accessories.

```

{
  "_id": ObjectId,
  "name": "Electric Guitar",
  "category": "guitar",
  "brand": "Fender",
  "price": 500,
  "stock": 10,
  "description": "An electric guitar for beginners.",
  "tags": ["electric", "beginner"],
  "ratingAvg": 4.5,
  "ratingCount": 25
}

```

3. Orders Collection

This collection stores the details of orders placed by users.

```

{
  "_id": ObjectId,
  "userId": ObjectId,
  "items": [
    {
      "productId": ObjectId,
      "nameSnapshot": "Electric Guitar",
      "priceSnapshot": 500,
      "qty": 2
    }
  ],
  "total": 1000,
  "status": "pending",
  "createdAt": "2026-02-02T10:20:30.000Z"
}

```

Collection name	Properties	Storage size	Documents	Avg. document size	Indexes	Total index size
orders	-	36.86 kB	3	222.00 B	2	73.73 kB
products	-	32.77 kB	5	233.00 B	4	131.07 kB
users	-	36.86 kB	1	210.00 B	2	40.96 kB

API Endpoints

The backend provides several RESTful API endpoints:

Authentication:

- POST /auth/login: Log in with email and password, receiving a JWT token in the response.
- POST /auth/register: Register a new user.

Products:

- GET /products: Retrieve a list of all available products.
- POST /products: Add a new product to the store (admin only).
- PATCH /products/stock: Update the stock quantity of a product (admin only).

Orders:

- POST /orders: Create a new order for the user.
- GET /orders/my: Retrieve the order history of the logged-in user.

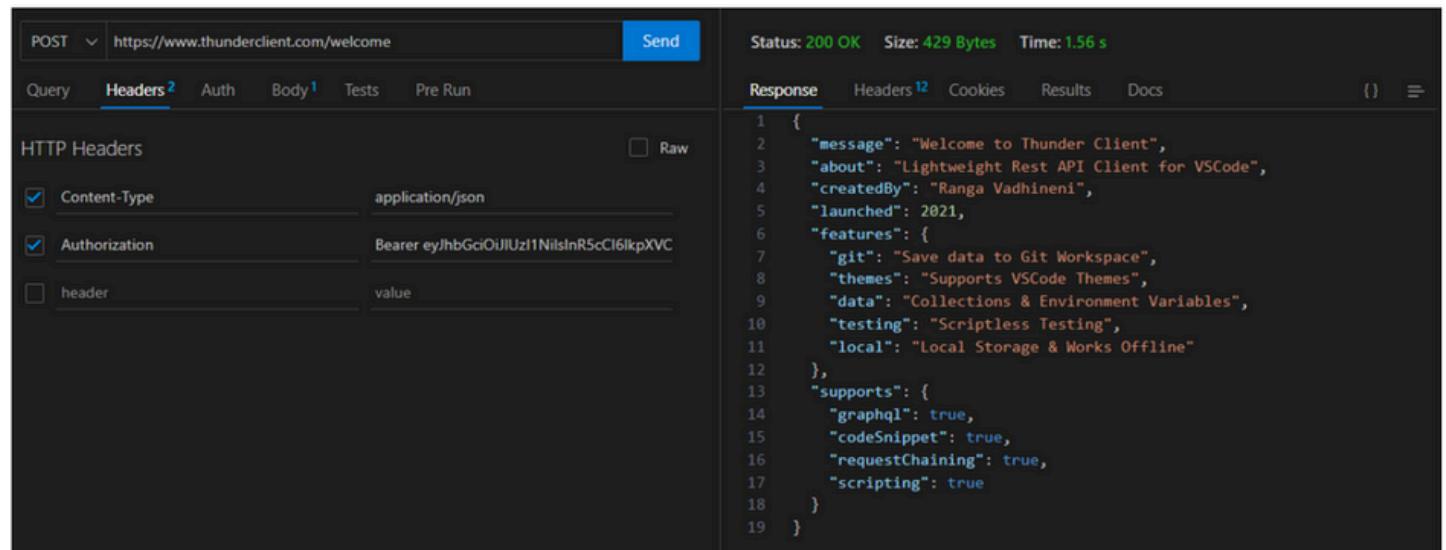
Admin:

- GET /admin/stats/sales: Get the total sales revenue and the number of orders (admin only).

CRUD Operations

CRUD functionality was implemented mainly for the Product and Order entities.

Create: add products, create orders



Status: 200 OK Size: 429 Bytes Time: 1.56 s

```
1  {
2    "message": "Welcome to Thunder Client",
3    "about": "Lightweight Rest API Client for VSCode",
4    "createdBy": "Ranga Vadhineni",
5    "launched": 2021,
6    "features": {
7      "git": "Save data to Git Workspace",
8      "themes": "Supports VSCode Themes",
9      "data": "Collections & Environment Variables",
10     "testing": "Scriptless Testing",
11     "local": "Local Storage & Works Offline"
12   },
13   "supports": {
14     "graphql": true,
15     "codeSnippet": true,
16     "requestChaining": true,
17     "scripting": true
18   }
19 }
```

Read: list products, view orders

```

Status: 200 OK  Size: 1.35 KB  Time: 4 ms
Response Headers Cookies Results Docs
1 [
2 {
3   "_id": "69827c700df2db9237ed5ea9",
4   "name": "Electric Guitar",
5   "category": "guitar",
6   "brand": "Fender",
7   "price": 1200,
8   "stock": 5,
9   "description": "Professional electric guitar",
10  "tags": [
11    "electric",
12    "pro"
13  ],
14  "ratingAvg": 4.5,
15  "ratingCount": 10,
16  "createdAt": "2026-02-03T22:53:36.293Z",
17  "updatedAt": "2026-02-03T22:53:36.293Z"
18 },
19 {
20   "_id": "69827c700df2db9237ed5eaa",
21   "name": "Dombra Pro",
22   "category": "dombra",
23   "brand": "KazMusic",
24   "price": 120,
25   "stock": 10,
26   "description": "Traditional Kazakh instrument",
27   "tags": [
28     "folk"
29   ],
30   "ratingAvg": 4.8,
31   "ratingCount": 6,
32   "createdAt": "2026-02-03T22:53:36.293Z",
33   "updatedAt": "2026-02-03T22:53:36.293Z"
34 }

```

Update: modify product fields and stock

```

Status: 200 OK  Size: 301 Bytes  Time: 10 ms
Response Headers Cookies Results Docs
1 {
2   "_id": "69827c700df2db9237ed5ea9",
3   "name": "Electric Guitar",
4   "category": "guitar",
5   "brand": "Fender",
6   "price": 550,
7   "stock": 12,
8   "description": "Professional electric guitar",
9   "tags": [
10    "electric",
11    "pro"
12  ],
13  "ratingAvg": 4.5,
14  "ratingCount": 10,
15  "createdAt": "2026-02-03T22:53:36.293Z",
16  "updatedAt": "2026-02-03T23:11:13.584Z"
17 }

```

Delete: remove products

```

Status: 200 OK  Size: 21 Bytes  Time: 4 ms
Response Headers Cookies Results Docs
1 {
2   "message": "Deleted"
3 }

```

Advanced MongoDB Operations

The project demonstrates several advanced update operators:

- \$set – update product fields
- \$inc – update stock values
- \$push – add tags
- \$pull – remove tags

These operators allow efficient partial document updates.

Data Aggregation

GET http://localhost:3000/admin/stats/sales

Headers 1

Auth Body 1 Tests Pre Run

HTTP Headers

Authorization: Bearer eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiO

header value

Status: 200 OK Size: 48 Bytes Time: 41 ms

Response

```
1 {  
2   "_id": null,  
3   "totalRevenue": 7580,  
4   "ordersCount": 8  
5 }
```

Aggregation Pipeline Example

In the project, one of the key features is the **Sales Statistics** functionality, which allows the admin to track the total revenue and number of orders. This functionality uses MongoDB's **Aggregation Framework**.

Aggregation Overview:

MongoDB's aggregation framework is used to process data and return computed results. It allows for operations like grouping, filtering, and calculating statistics.

In this specific case, I am using the \$group stage to aggregate the orders and calculate the total revenue (totalRevenue) and count the total number of orders (ordersCount).

Indexing and Query Optimization

Indexes are used in MongoDB to speed up queries and filter operations.

```
productSchema.index({ category: 1, price: 1 }); // Index for searching by category and price
```

```
productSchema.index({ name: "text", description: "text" }); // Full-text search index for products
```

By creating these indexes, the system ensures fast filtering and searching by product category and name.

Authentication and Authorization

JWT Authentication: The application uses JWT tokens to authenticate users. Upon login, a JWT token is issued, and users must send this token in the Authorization header to access protected routes.

Role-based Access Control: Admin routes are protected by checking the user's role stored in the token. Only users with the "admin" role can access endpoints like /admin/stats/sales.

API Documentation

Example Request and Response

Login Request:

POST /auth/login

The screenshot shows a Thunder Client interface. On the left, under 'Headers' tab, there are three selected headers: 'Accept' (value: */*), 'User-Agent' (value: Thunder Client (https://www.thunderclient.c...)), and 'header' (value: value). On the right, the response tab shows a successful 200 OK status with 329 bytes and 57 ms time. The response body is a JSON object:

```
1  {
2    "message": "Login ok",
3    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJ1c2VySWQiOjI2OTgxZDgzMTc0ZWNjNTIiYmVhNzhmMDkiLCJyb2x1IjoiYW
        RtaW41LCJpYXQiOjE3NzAxNTk5MzUsImV4cCI6MTc3MDE2NzEzMjQw",
4    "user": {
5      "id": "6981d83174ecc59baea78f09",
6      "name": "Test User",
7      "email": "test1@mail.com",
8      "role": "admin"
9    }
10 }
```

Products:

```

Status: 200 OK Size: 1.06 KB Time: 5 ms
Response Headers 7 Cookies Results Docs
1 [
2   {
3     "_id": "69827c700df2db9237ed5ea9",
4     "name": "Electric Guitar",
5     "category": "guitar",
6     "brand": "Fender",
7     "price": 550,
8     "stock": 12,
9     "description": "Professional electric guitar",
10    "tags": [
11      "electric",
12      "pro"
13    ],
14    "ratingAvg": 4.5,
15    "ratingCount": 10,
16    "createdAt": "2026-02-03T22:53:36.293Z",
17    "updatedAt": "2026-02-03T23:11:13.584Z"
18  },
19  {
20    "_id": "69824e2574ecc59baea78f1e",
21    "name": "Acoustic Guitar",
22    "category": "guitar",
23    "brand": "Yamaha",
24    "price": 300,
25    "stock": 5,
26    "description": "",
27    "tags": [],
28    "ratingAvg": 0,
29    "ratingCount": 0,
30    "createdAt": "2026-02-03T19:36:05.547Z",
31    "updatedAt": "2026-02-03T19:36:05.547Z",
32    "__v": 0
33  }
]

```

Orders:

```

Status: 201 Created Size: 327 Bytes Time: 10 ms
Response Headers 7 Cookies Results Docs
1 {
2   "userId": "6981d83174ecc59baea78f09",
3   "items": [
4     {
5       "productId": "69824e1d74ecc59baea78f1c",
6       "nameSnapshot": "Domra Pro",
7       "priceSnapshot": 120,
8       "qty": 1,
9       "_id": "69828c445d56f67fb9666a85"
10     }
11   ],
12   "total": 120,
13   "status": "pending",
14   "_id": "69828c445d56f67fb9666a84",
15   "createdAt": "2026-02-04T00:01:08.463Z",
16   "updatedAt": "2026-02-04T00:01:08.463Z",
17   "__v": 0
18 }

```

Orders/stats/sales:

```

Status: 200 OK Size: 48 Bytes Time: 29 ms
Response Headers 7 Cookies Results Docs
1 {
2   "_id": null,
3   "totalRevenue": 5020,
4   "ordersCount": 6
5 }

```

Contribution

Installation Steps

- Clone the repository using the following command:

```
git clone https://github.com/enxww/music-store.git
```

1. Install dependencies:

```
npm install
```

1. Create a .env file with the necessary environment variables as described in the setup section.
2. Run the application:

```
npm run dev
```

Conclusion

The Music Store project implements a fully functional e-commerce platform with user authentication, product management, order creation, and sales statistics. The system is built using Node.js, MongoDB, and follows RESTful principles. Aggregation pipelines are used to generate sales statistics, and JWT-based authentication ensures secure access to the platform.

This report outlines the project architecture, database schema, and API documentation. The use of aggregation pipelines, efficient indexing, and role-based authorization makes the system scalable and secure.