# **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:

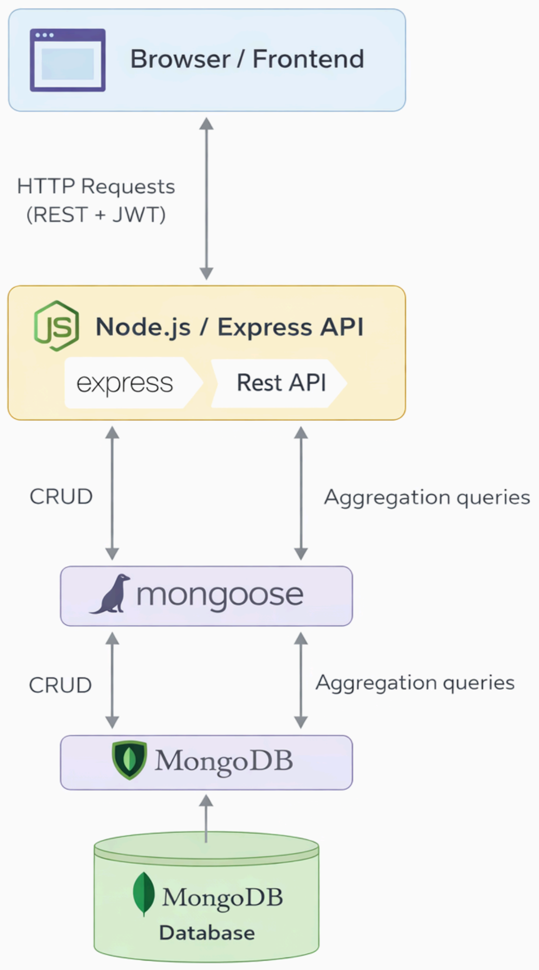


*Figure 1. System architecture: the frontend communicates with the Express REST API via HTTP requests secured by JWT tokens. The backend accesses MongoDB through Mongoose ODM.*

## 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

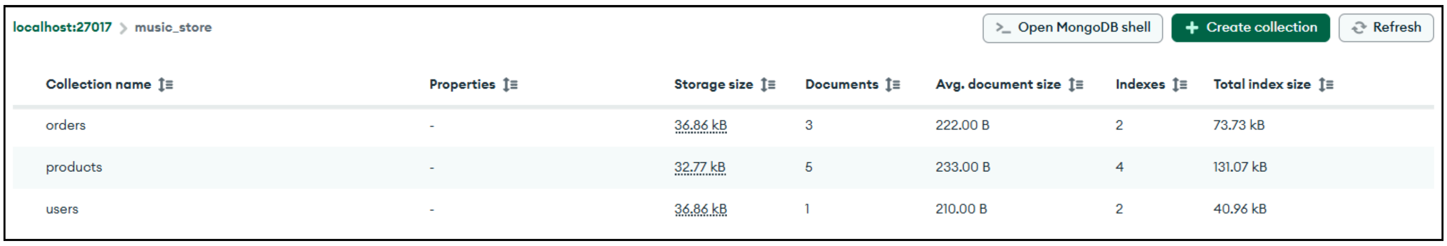
This collection stores information about musical instruments such as guitars, pianos, and accessories.

```
{
  "_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.

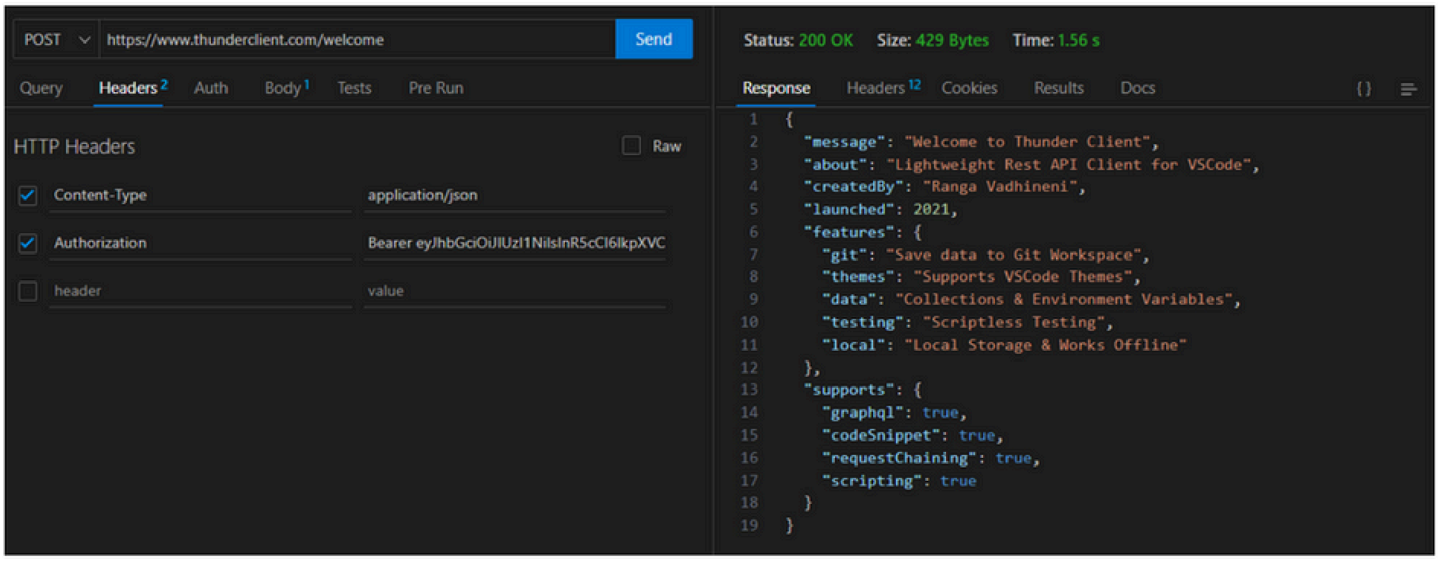
Admin:

- 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



POST https://www.thunderclient.com/welcome

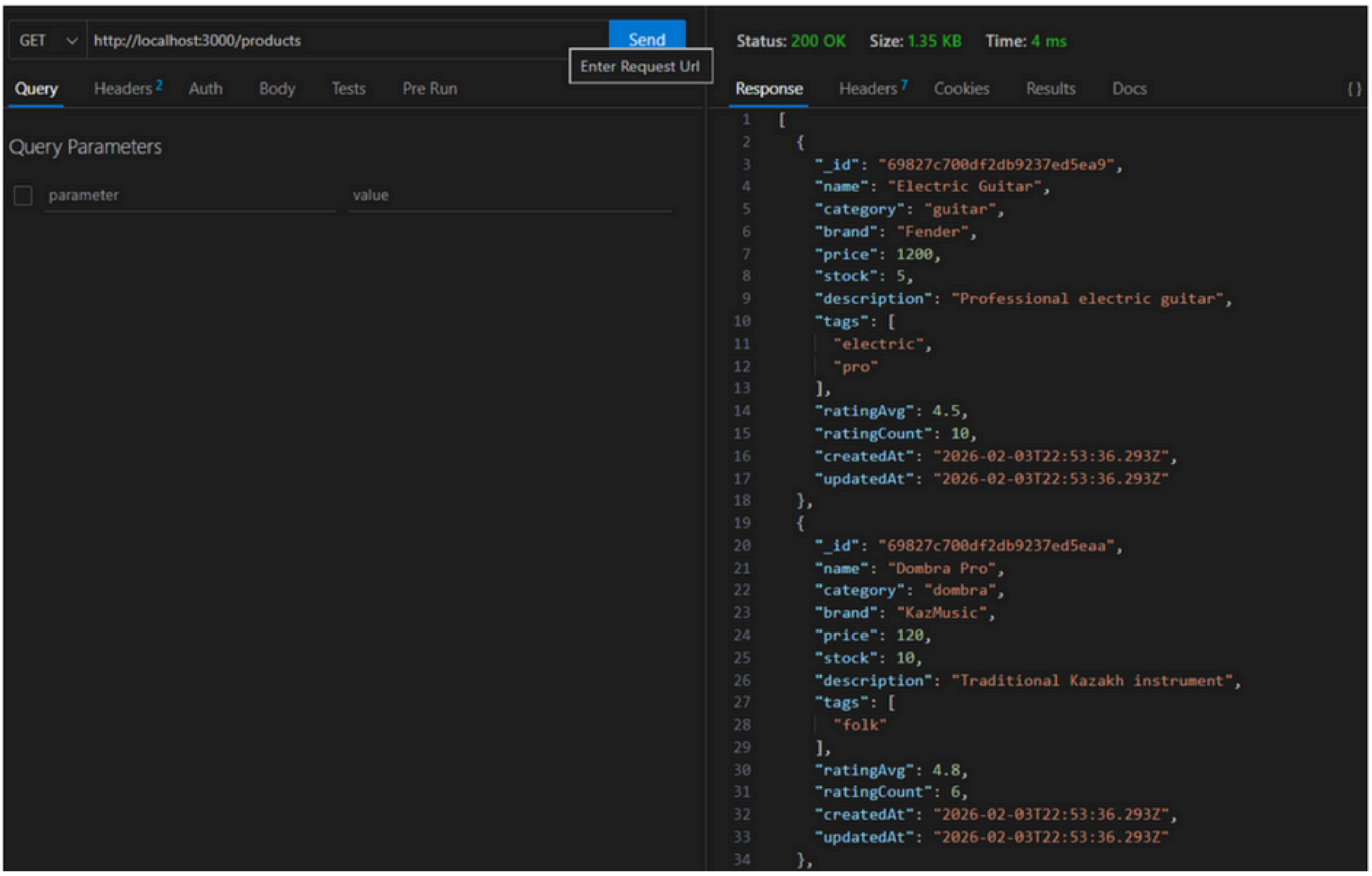
Status: 200 OK Size: 429 Bytes Time: 1.56 s

HTTP Headers

| | |
|---|---|
| ✓ Content-Type | application/json |
| ✓ Authorization | Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC |
| header | value |

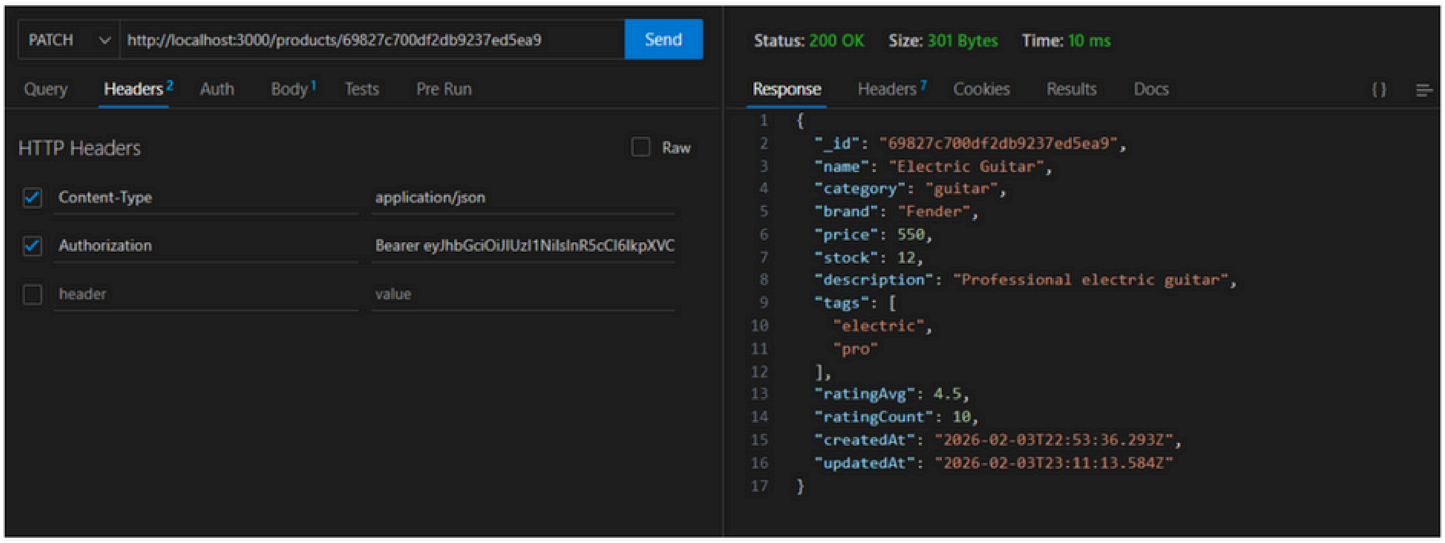
```
{
  "message": "Welcome to Thunder Client",
  "about": "Lightweight Rest API Client for VSCode",
  "createdBy": "Ranga Vadhineni",
  "launched": 2021,
  "features": {
    "git": "Save data to Git Workspace",
    "themes": "Supports VSCode Themes",
    "data": "Collections & Environment Variables",
    "testing": "Scriptless Testing",
    "local": "Local Storage & Works Offline"
  },
  "supports": {
    "graphql": true,
    "codeSnippet": true,
    "requestChaining": true,
    "scripting": true
  }
}
```

Read: list products, view orders



```
GET http://localhost:3000/products
Status: 200 OK   Size: 1.35 KB   Time: 4 ms
[
  {
    "_id": "69827c700df2db9237ed5ea9",
    "name": "Electric Guitar",
    "category": "guitar",
    "brand": "Fender",
    "price": 1200,
    "stock": 5,
    "description": "Professional electric guitar",
    "tags": [
      "electric",
      "pro"
    ],
    "ratingAvg": 4.5,
    "ratingCount": 10,
    "createdAt": "2026-02-03T22:53:36.293Z",
    "updatedAt": "2026-02-03T22:53:36.293Z"
  },
  {
    "_id": "69827c700df2db9237ed5eaa",
    "name": "Dombra Pro",
    "category": "dombra",
    "brand": "KazMusic",
    "price": 120,
    "stock": 10,
    "description": "Traditional Kazakh instrument",
    "tags": [
      "folk"
    ],
    "ratingAvg": 4.8,
    "ratingCount": 6,
    "createdAt": "2026-02-03T22:53:36.293Z",
    "updatedAt": "2026-02-03T22:53:36.293Z"
  },
```

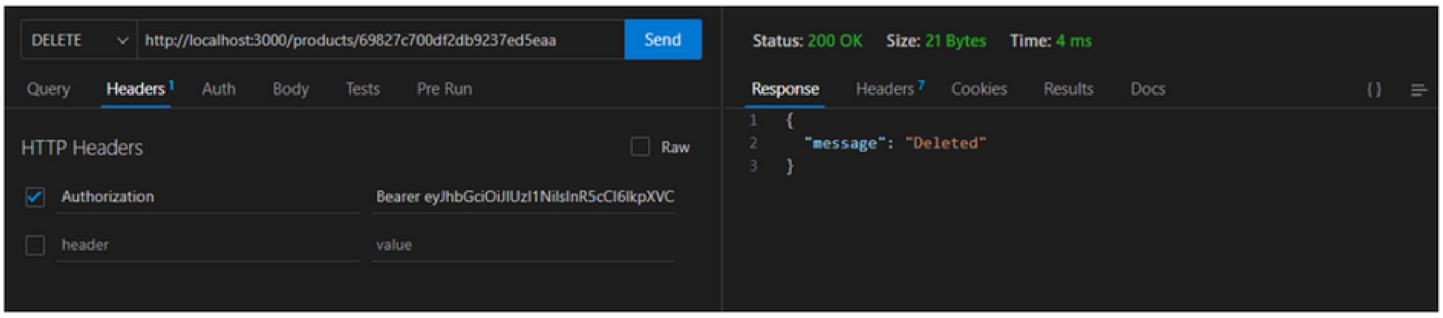
Update: modify product fields and stock



```
PATCH http://localhost:3000/products/69827c700df2db9237ed5ea9
Content-Type      application/json
Authorization     Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC

Status: 200 OK   Size: 301 Bytes   Time: 10 ms
{
  "_id": "69827c700df2db9237ed5ea9",
  "name": "Electric Guitar",
  "category": "guitar",
  "brand": "Fender",
  "price": 550,
  "stock": 12,
  "description": "Professional electric guitar",
  "tags": [
    "electric",
    "pro"
  ],
  "ratingAvg": 4.5,
  "ratingCount": 10,
  "createdAt": "2026-02-03T22:53:36.293Z",
  "updatedAt": "2026-02-03T23:11:13.584Z"
}
```

Delete: remove products



```
DELETE http://localhost:3000/products/69827c700df2db9237ed5eaa
Authorization     Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVC

Status: 200 OK   Size: 21 Bytes   Time: 4 ms
{
  "message": "Deleted"
}
```

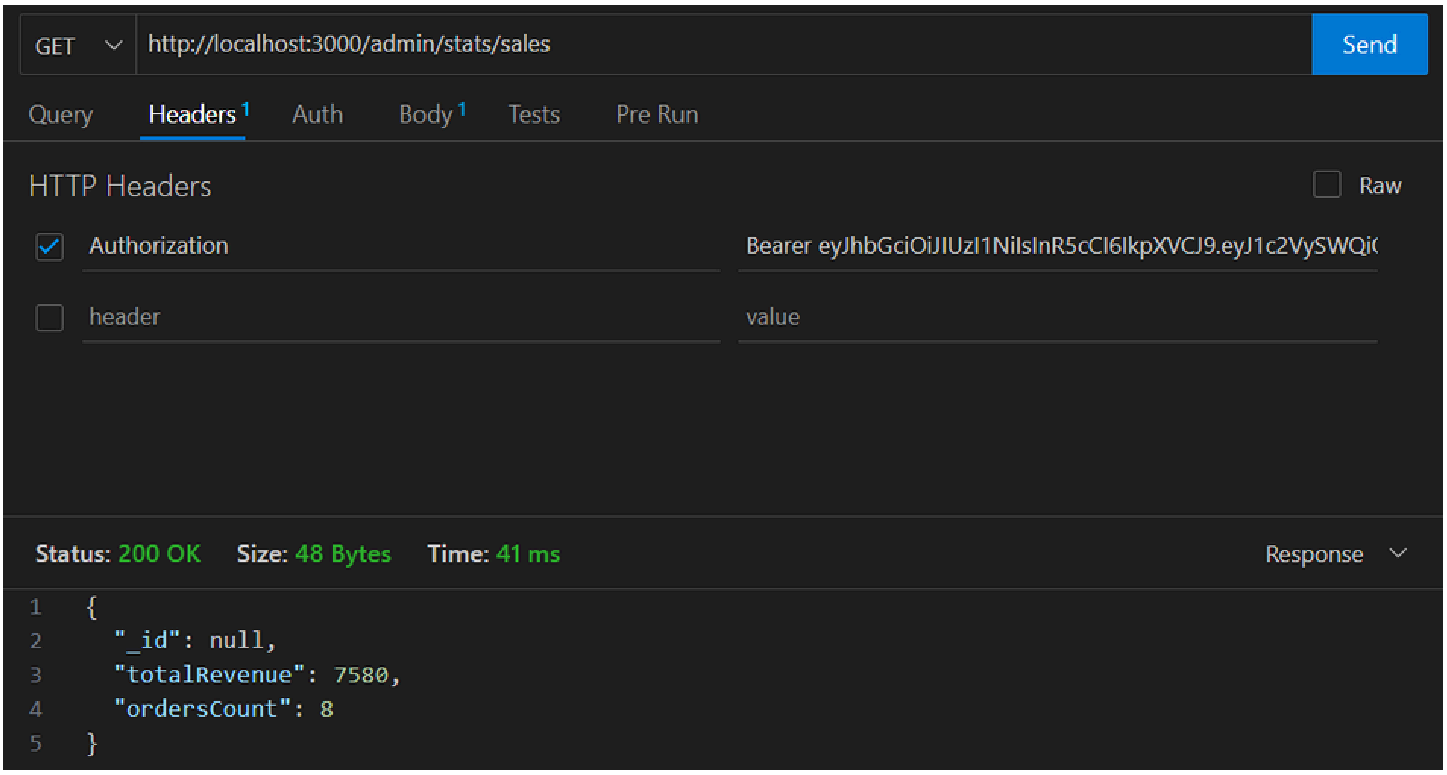
## Advanced MongoDB Operations

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
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQi(
```

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

```
{
  "_id": null,
  "totalRevenue": 7580,
  "ordersCount": 8
}
```

## 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.

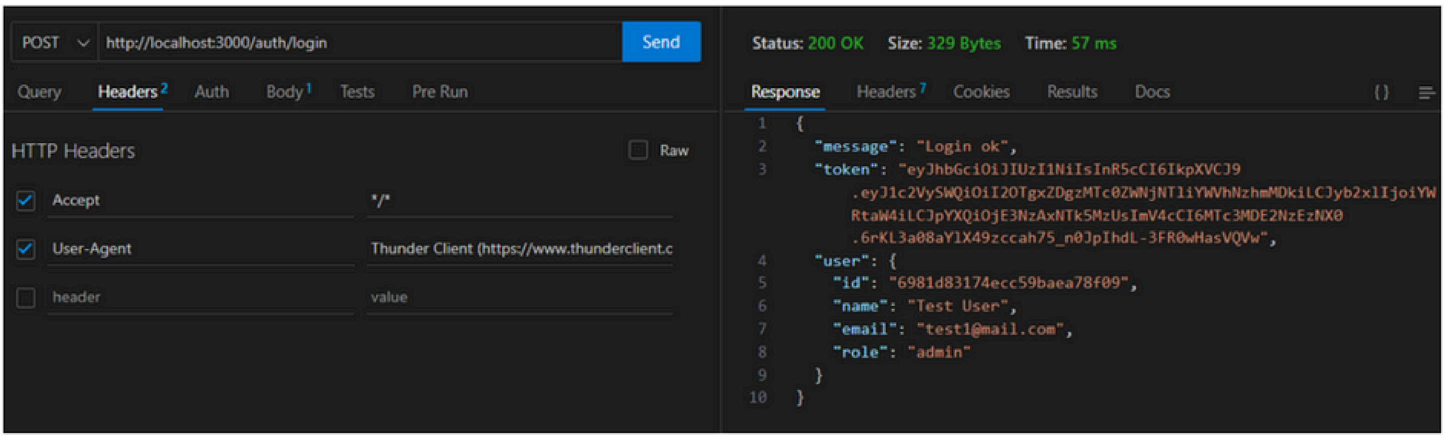
Role-based Access Control: Admin routes are protected by checking the user's role stored in the token. Only users with the "admin" role can access endpoints like /admin/stats/sales.

## API Documentation

Example Request and Response

Login Request:

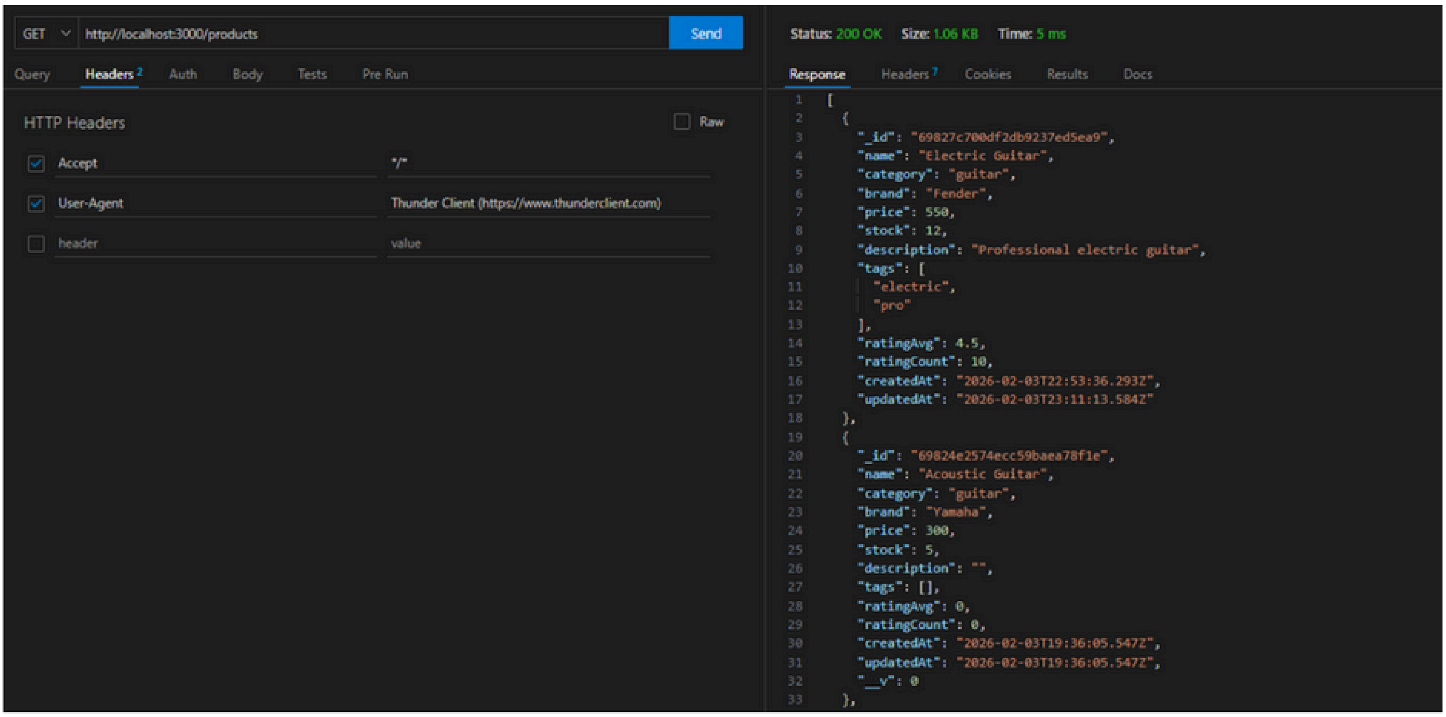
POST /auth/login



| | |
|---|---|
| ✓ Accept | */* |
| ✓ User-Agent | Thunder Client (https://www.thunderclient.c |
| header | value |

```
{
  "message": "Login ok",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
      .eyJ1c2VySWQiOiI2OTgxZDgzMTc0ZWNjNTliYWVhNzhmMDkiLCJyb2xlIjoiYW
      RtaW4iLCJpYXQiOjE3NzAxNTk5MzUsImV4cCI6MTc3MDE2NzEzNX0
      .6rKL3a08aY1X49zccah75_n0JpIhdL-3FR0wHasVQVw",
  "user": {
    "id": "6981d83174ecc59baea78f09",
    "name": "Test User",
    "email": "test1@mail.com",
    "role": "admin"
  }
}
```

<u>Products:</u>



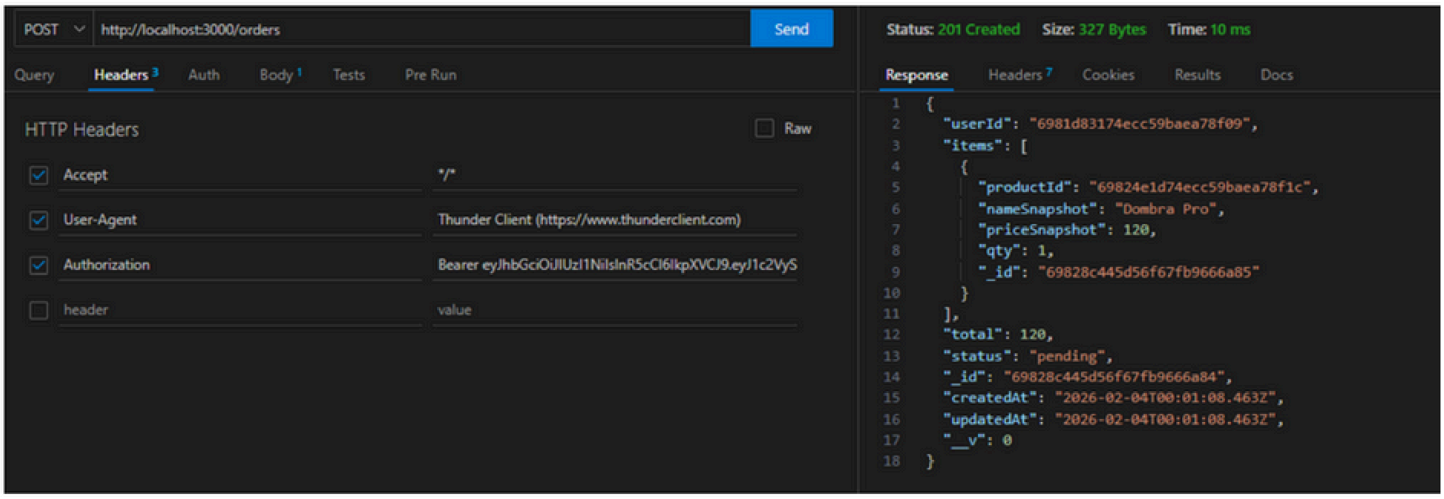
```
GET  http://localhost:3000/products                Send

HTTP Headers
☑ Accept        */*
☑ User-Agent    Thunder Client (https://www.thunderclient.com)
☐ header        value
```

```
Status: 200 OK   Size: 1.06 KB   Time: 5 ms

[
  {
    "_id": "69827c700df2db9237ed5ea9",
    "name": "Electric Guitar",
    "category": "guitar",
    "brand": "Fender",
    "price": 550,
    "stock": 12,
    "description": "Professional electric guitar",
    "tags": [
      "electric",
      "pro"
    ],
    "ratingAvg": 4.5,
    "ratingCount": 10,
    "createdAt": "2026-02-03T22:53:36.293Z",
    "updatedAt": "2026-02-03T23:11:13.584Z"
  },
  {
    "_id": "69824e2574ecc59baea78f1e",
    "name": "Acoustic Guitar",
    "category": "guitar",
    "brand": "Yamaha",
    "price": 300,
    "stock": 5,
    "description": "",
    "tags": [],
    "ratingAvg": 0,
    "ratingCount": 0,
    "createdAt": "2026-02-03T19:36:05.547Z",
    "updatedAt": "2026-02-03T19:36:05.547Z",
    "__v": 0
  },
```

<u>Orders:</u>



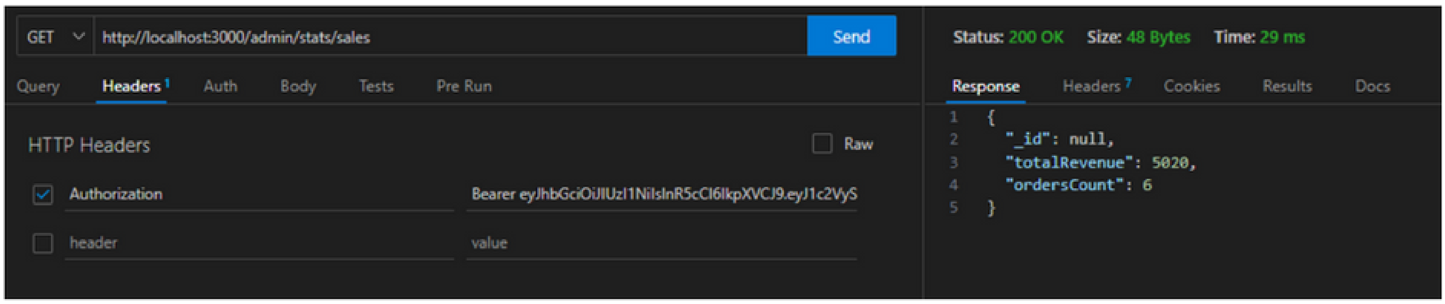
```
POST  http://localhost:3000/orders                Send

HTTP Headers
☑ Accept          */*
☑ User-Agent      Thunder Client (https://www.thunderclient.com)
☑ Authorization   Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyS
☐ header          value
```

```
Status: 201 Created   Size: 327 Bytes   Time: 10 ms

{
  "userId": "6981d83174ecc59baea78f09",
  "items": [
    {
      "productId": "69824e1d74ecc59baea78f1c",
      "nameSnapshot": "Dombra Pro",
      "priceSnapshot": 120,
      "qty": 1,
      "_id": "69828c445d56f67fb9666a85"
    }
  ],
  "total": 120,
  "status": "pending",
  "_id": "69828c445d56f67fb9666a84",
  "createdAt": "2026-02-04T00:01:08.463Z",
  "updatedAt": "2026-02-04T00:01:08.463Z",
  "__v": 0
}
```

<u>Orders/stats/sales:</u>



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

HTTP Headers
☑ Authorization   Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyS
☐ header          value
```

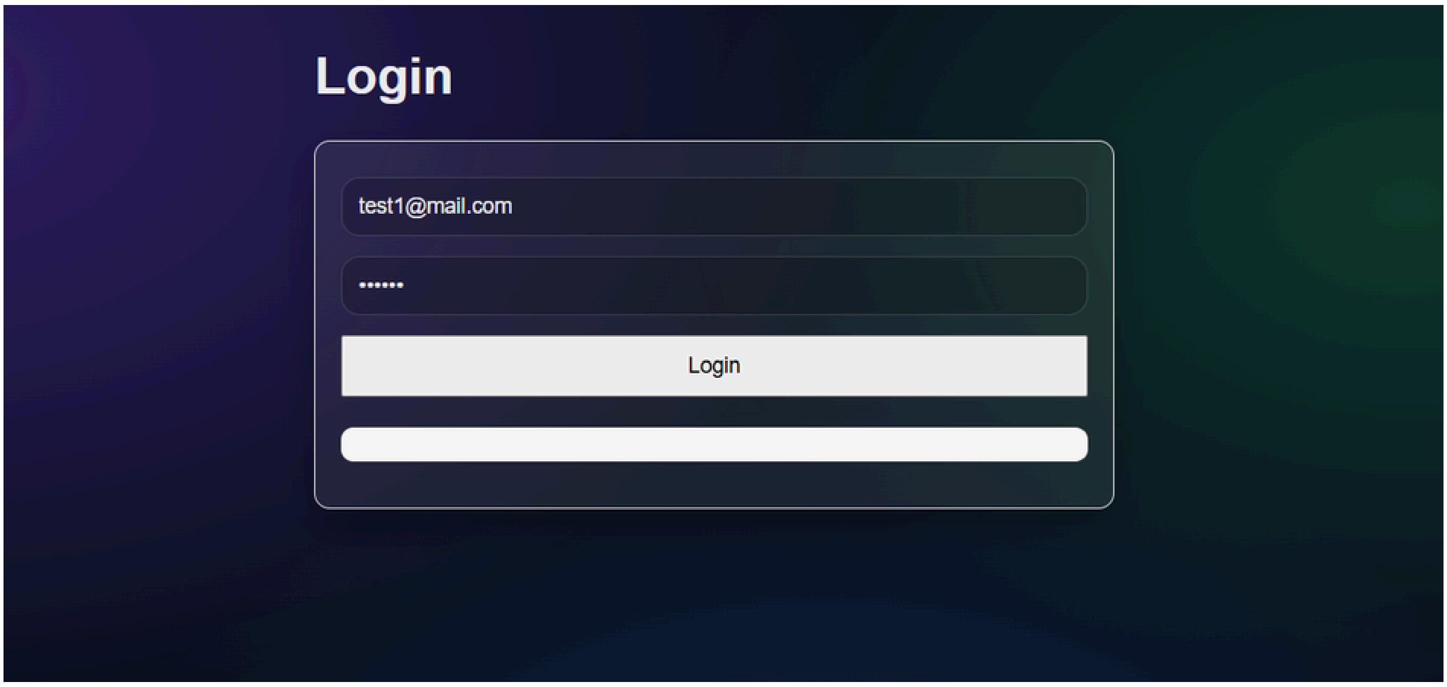
```
Status: 200 OK   Size: 48 Bytes   Time: 29 ms

{
  "_id": null,
  "totalRevenue": 5020,
  "ordersCount": 6
}
```

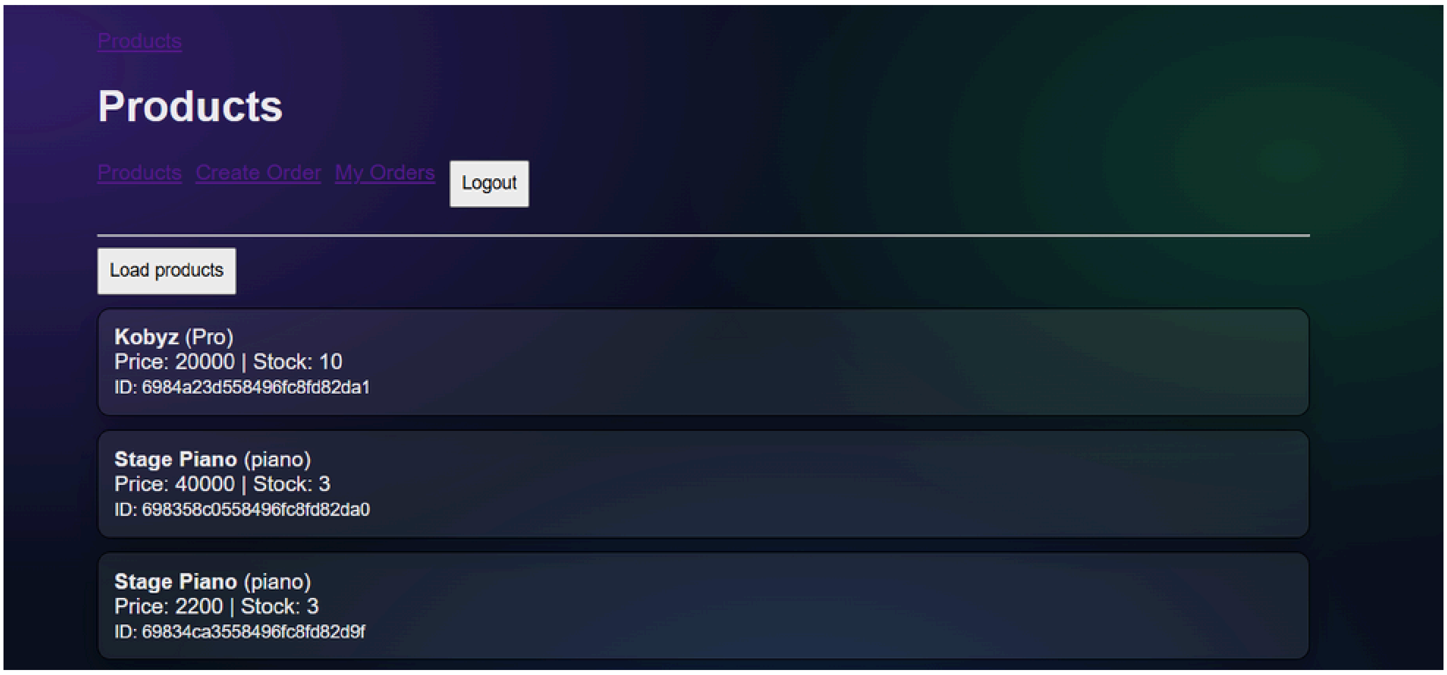
## 11. Frontend Implementation

The frontend consists of four pages:

Login page:



Products list:

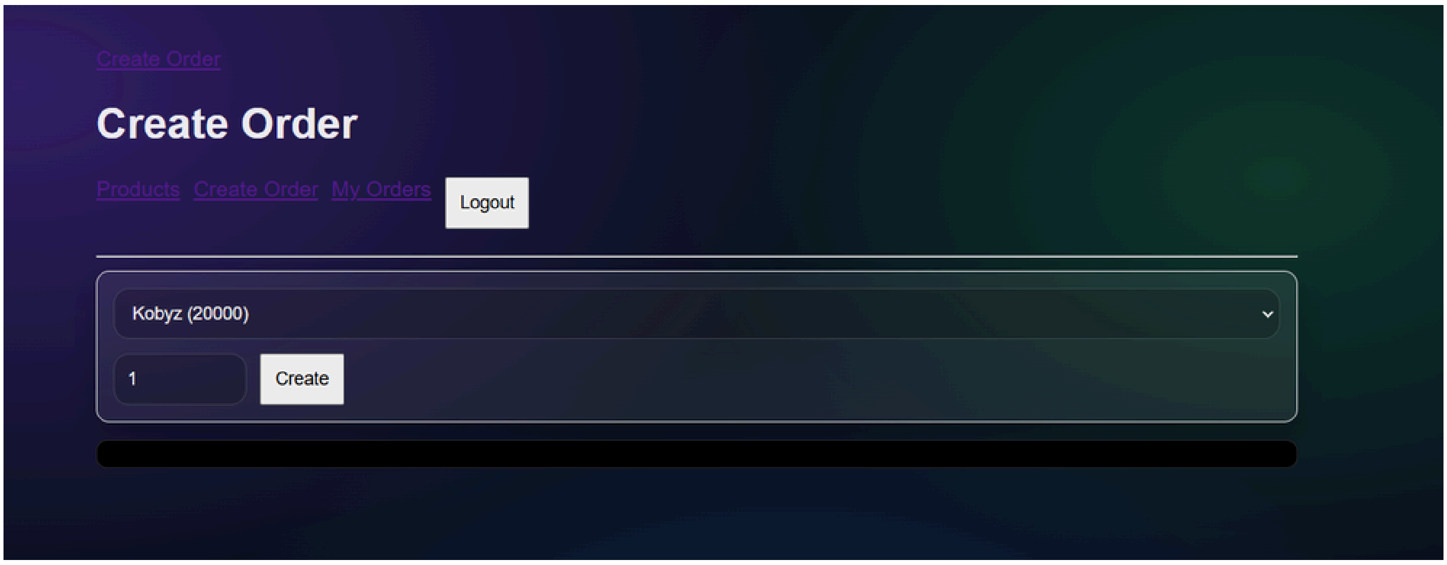


**Kobyz** (Pro)
Price: 20000 | Stock: 10
ID: 6984a23d558496fc8fd82da1

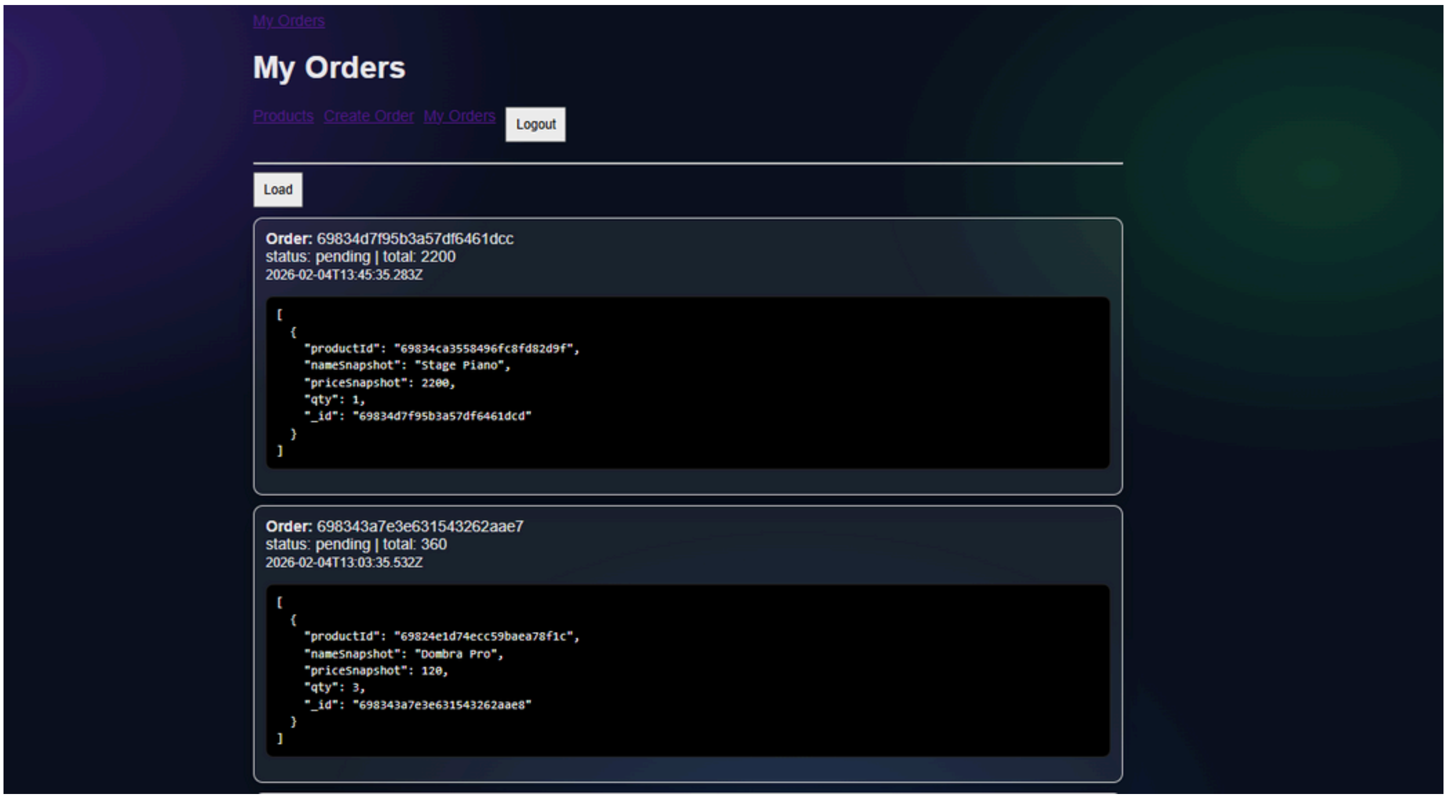
**Stage Piano** (piano)
Price: 40000 | Stock: 3
ID: 698358c0558496fc8fd82da0

**Stage Piano** (piano)
Price: 2200 | Stock: 3
ID: 69834ca3558496fc8fd82d9f

Create order:



My orders:



**Order:** 69834d7f95b3a57df6461dcc
status: pending | total: 2200
2026-02-04T13:45:35.283Z

```
[
  {
    "productId": "69834ca3558496fc8fd82d9f",
    "nameSnapshot": "Stage Piano",
    "priceSnapshot": 2200,
    "qty": 1,
    "_id": "69834d7f95b3a57df6461dcd"
  }
]
```

**Order:** 698343a7e3e631543262aae7
status: pending | total: 360
2026-02-04T13:03:35.532Z

```
[
  {
    "productId": "69824e1d74ecc59baea78f1c",
    "nameSnapshot": "Dombra Pro",
    "priceSnapshot": 120,
    "qty": 3,
    "_id": "698343a7e3e631543262aae8"
  }
]
```

Each page communicates with the backend using real HTTP requests. CSS styling was added to create a clean and usable interface.

## Contribution

Installation Steps

1. 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.