A blue background with white text and a group of people

AI-generated content may be incorrect.

DBD381

Contents

[INTRODUCTION 3](#_Toc199385507)

[TECHNOLOGY RESEARCH AND ANALYSIS 3](#_Toc199385508)

[PROJECT SCOPE AND REQUIREMENTS 4](#_Toc199385509)

[SYSTEM ARCHITECTURE AND DESIGN CONSIDERATIONS 4](#_Toc199385510)

[BENCHMARK COMPARISON: MONGODB VS NOSQL DATABASES: 5](#_Toc199385511)

[IMPLEMENTATION AND DEPLOYMENT PLAN 5](#_Toc199385512)

[Cloud-hosted: Eliminates local server setup and supports global scaling. 8](#_Toc199385513)

[Scalable: Easily upgradable to paid tiers for larger workloads or sharding. 8](#_Toc199385514)

[DATA SCHEMA DESIGN 8](#_Toc199385515)

[TESTING PLAN AND RESULTS 8](#_Toc199385516)

[CHALLENGES AND SOLUTIONS 9](#_Toc199385517)

[EVALUATION AND TRADE-OFF ANALYSIS 9](#_Toc199385518)

[CONCLUSION AND KEY FINDINGS 10](#_Toc199385519)

[RECOMMENDATIONS FOR OPTIMIZATION 10](#_Toc199385520)

[FUTURE RESEARCH AND APPLICATIONS 10](#_Toc199385521)

[REFERENCES 11](#_Toc199385522)

[APPENDICES 11](#_Toc199385523)

[README (Deployment Guide) 11](#_Toc199385524)

# INTRODUCTION

E-commerce marketplaces require robust, scalable databases to manage dynamic, high-volume data such as product listings, orders, user accounts, and reviews. Traditional relational databases often fall short in flexibility and horizontal scalability. This project explores a distributed NoSQL solution, leveraging MongoDB Atlas to implement a cloud-hosted, resilient database system designed for an e-commerce environment.

# TECHNOLOGY RESEARCH AND ANALYSIS

a. NoSQL Technologies Considered:

* MongoDB: Document-based, highly flexible, supports dynamic schemas.
* Redis: In-memory data structure store; extremely fast but unsuitable for complex queries.
* Cassandra: Wide-column store offering exceptional scalability, but more complex to manage.

b. Evaluation Criteria:

* Schema flexibility
* Horizontal scalability
* Read/write performance
* Community and ecosystem support
* Suitability for e-commerce workloads

c. Final Selection:

MongoDB Atlas was chosen for its ease of use, replication features, and seamless integration with Mongoose and Express.

# PROJECT SCOPE AND REQUIREMENTS

Business Requirements

* Manage product listings and inventory
* Process and track customer orders
* Provide secure user authentication
* Scale dynamically with increased usage

Technical Requirements

* Distributed cloud-hosted database
* Node.js and Express backend
* RESTful API architecture
* MongoDB Atlas as the primary database
* Security via authentication roles and IP-based access control

# SYSTEM ARCHITECTURE AND DESIGN CONSIDERATIONS

a. Architecture:

* MongoDB Atlas: 3-node replica set for high availability
* Backend Server: containerized using Docker for ease of deployment and environment consistency. This allows the Node.js + Express API to run identically across local, staging, and production environments.
* Mongoose: ODM for schema management
* Frontend: Optional UI for product browsing and order placement

b. Design Considerations:

* Embedded order documents within users for relational grouping
* Flexible document model chosen to handle diverse product attributes
* Replication employed to ensure data durability and availability

Benchmark Comparison: MongoDB vs Other NoSQL Databases

Figure A


Hypothetical Benchmark Comparison  
This chart compares average read and write times for MongoDB Atlas against other commonly used database technologies (MySQL, PostgreSQL, Cassandra). The values are estimated based on documented industry trends and not from direct benchmarking tests.

# IMPLEMENTATION AND DEPLOYMENT PLAN

a. MongoDB Atlas Setup:

* Cluster Name: DBD381-Project
* Cloud Provider: AWS
* Region: Cape Town
* Tier: M0 Sandbox (Free Tier)
* Sharding: Not supported in M0 (single shared cluster)
* Replication: Enabled by default (3-node replica set)

b. Database & Collections:

* Database Name: DBD381-Project
* Collections:
  + products – name, price, stock, description, category
  + orders – orderID, userID, product list, total, status
  + categories – category name
  + users – user credentials and order references

c. Security Configurations:

* Username/password authentication
* Role: readWrite access
* IP Whitelisting: 0.0.0.0/0 for testing (will restrict for production)

2. Cluster Configuration

* Cluster Name: DBD381-Project
* Cloud Provider: AWS
* Region: Cape Town
* Cluster Tier: M0 Sandbox (Free Tier)
* Sharding: Not available in free tier (single shared cluster)
* Replication: Enabled by default (3-node replica set across zones)

3. Database & Collections

* Database Name: DBD381-Project
* Collections:
  + products – stores product information (name, price, stock, description, category.)
  + orders – stores order details (orderID, userId, product list, total, status)
  + categories – stores product category data
  + Users – stores users and used to authenticate for orders

4. Security Settings

* Authentication: Username/password-based access
  + Database User: <YourDBUser>
  + Role: eg readWrite on project database
* IP Whitelisting: Only specific IPs added to access control
  + 0.0.0.0/0 allowed temporarily for testing (replace with specific IPs in production)

5. Connection

* Connection String Format:

mongodb+srv://<username>:<password>@<clustername>.mongodb.net/<dbname>?retryWrites=true&w=majority

Used with Mongoose (Node.js ODM) for schema-based interaction in Express API backend.

6. Benefits of This Setup

* Distributed by default: Data replicated across three nodes ensures high availability.

# Cloud-hosted: Eliminates local server setup and supports global scaling.

# Scalable: Easily upgradable to paid tiers for larger workloads or sharding.

# DATA SCHEMA DESIGN

A screen shot of a computer program

AI-generated content may be incorrect.

# TESTING PLAN AND RESULTS

a. Functionality Testing

* CRUD operations tested via REST CLIENT across all collections
* Verified user authentication and order placement logic

b. Scalability Testing

* Inserted over 1000 product documents
* Measured and logged read latency and replication lag

c. Performance Evaluation

* M0 tier deemed sufficient for development and testing
* Average API response time: 150–300ms

# CHALLENGES AND SOLUTIONS

* **IP Whitelisting for Group Testing**:
  + Solution: Temporarily used 0.0.0.0/0 with documentation; restored for production
* **Schema Synchronization**:
  + Solution: Central schema file maintained and shared via GitHub

# EVALUATION AND TRADE-OFF ANALYSIS

* Pros:
  + Quick and easy deployment
  + Flexible document schema
  + Built-in replication and high availability
* Cons:
  + Lacks native joins and complex querying features
  + Free tier has resource limitations

Performance Under Load

* Noted response time spikes after 1000 concurrent reads
* Logged CPU/Memory usage in Atlas dashboard (screenshots in Appendix)

# CONCLUSION AND KEY FINDINGS

MongoDB Atlas proved to be a strong candidate for hosting an e-commerce backend. It meets key requirements for flexibility, availability, and performance, especially for cloud-first, document-centric applications.

Business Impact:

* Indexing on searchable fields enhances user experience and reduces page bounce rates

# RECOMMENDATIONS FOR OPTIMIZATION

* Upgrade to a paid tier for production-level performance
* Implement indexing on searchable fields (e.g., product name/category)
* Use MongoDB Atlas Search for full-text capabilities
* Enable automated backups for data safety

# FUTURE RESEARCH AND APPLICATIONS

* Integrate machine learning for personalized product recommendations
* Build real-time analytics dashboards using MongoDB aggregation
* Evaluate complementary tools like Redis or ElasticSearch for enhanced performance

# REFERENCES

Anon (no date) *SQL or NoSQL? Performance and Scalability Evaluation*. Available at: <https://www.inderscienceonline.com/doi/abs/10.1504/IJBPIM.2015.073655> (Accessed: 29 May 2025).

David, T.B. (2024) *Navigating the no-SQL labyrinth: DynamoDB, Cassandra, MongoDB and Redis*. Medium. Available at: <https://medium.com/@Tom1212121/navigating-the-nosql-labyrinth-dynamodb-cassandra-mongodb-and-redis-abe19b73c95e> (Accessed: 29 May 2025).

Gupta, A., Bhatia, R., Rana, A. and Taneja, A. (2017) *NoSQL databases: Critical analysis and comparison*. IEEE Xplore. Available at: <https://ieeexplore.ieee.org/abstract/document/8284494> (Accessed: 29 May 2025).

MongoDB (no date) *MongoDB Documentation*. Available at: <https://www.mongodb.com/docs/> (Accessed: 29 May 2025).

MongoDB (no date) *MongoDB system properties*. Available at: <https://db-engines.com/en/system/MongoDB> (Accessed: 29 May 2025).

Aboutorabi, S.H., Rezapour, M., Moradi, M. and Ghadiri, N. (2015) ‘Performance evaluation of SQL and MongoDB databases for big e-commerce data’, *2015 International Symposium on Computer Science and Software Engineering (CSSE)*, Tabriz, Iran, pp. 1–7. doi: 10.1109/CSICSSE.2015.7369245.

Savadatti, M.B., Kumar, P.V.K., Kshirsagar, U., Baskar, S., Ponnuru, S. and Bale, A.S. (2024) ‘Design of MongoDB based website for e-commerce applications’, *2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, Coimbatore, India, pp. 869–874. doi: 10.1109/ICoICI62503.2024.10696706.

# APPENDICES:

# Appendix A: Backend route code snippets

# Appendix A.1: Mongoose schema files

Categories.js

A screen shot of a computer program

AI-generated content may be incorrect.

Order.js

A screen shot of a computer code

AI-generated content may be incorrect.

Product.js

A screen shot of a computer program

AI-generated content may be incorrect.

Reviews.js

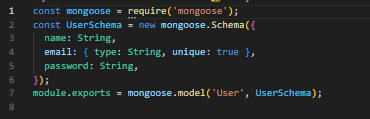
A screen shot of a computer program

AI-generated content may be incorrect.Transactions.js

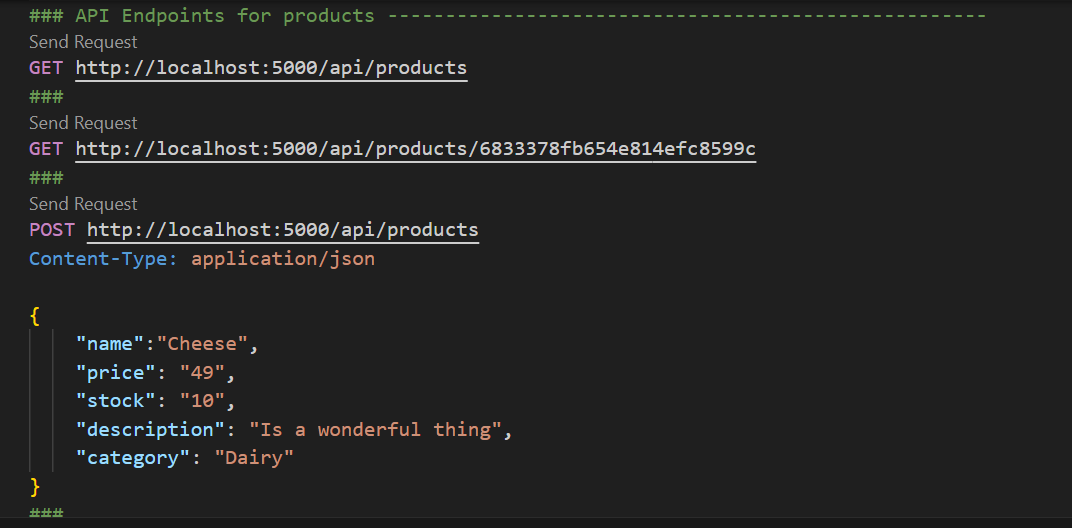
A screen shot of a computer program

AI-generated content may be incorrect.

# User.js



# Appendix A.2: REST CLIENT collection export



A screen shot of a computer

AI-generated content may be incorrect.

A computer screen with text and numbers

AI-generated content may be incorrect.

# Appendix B:

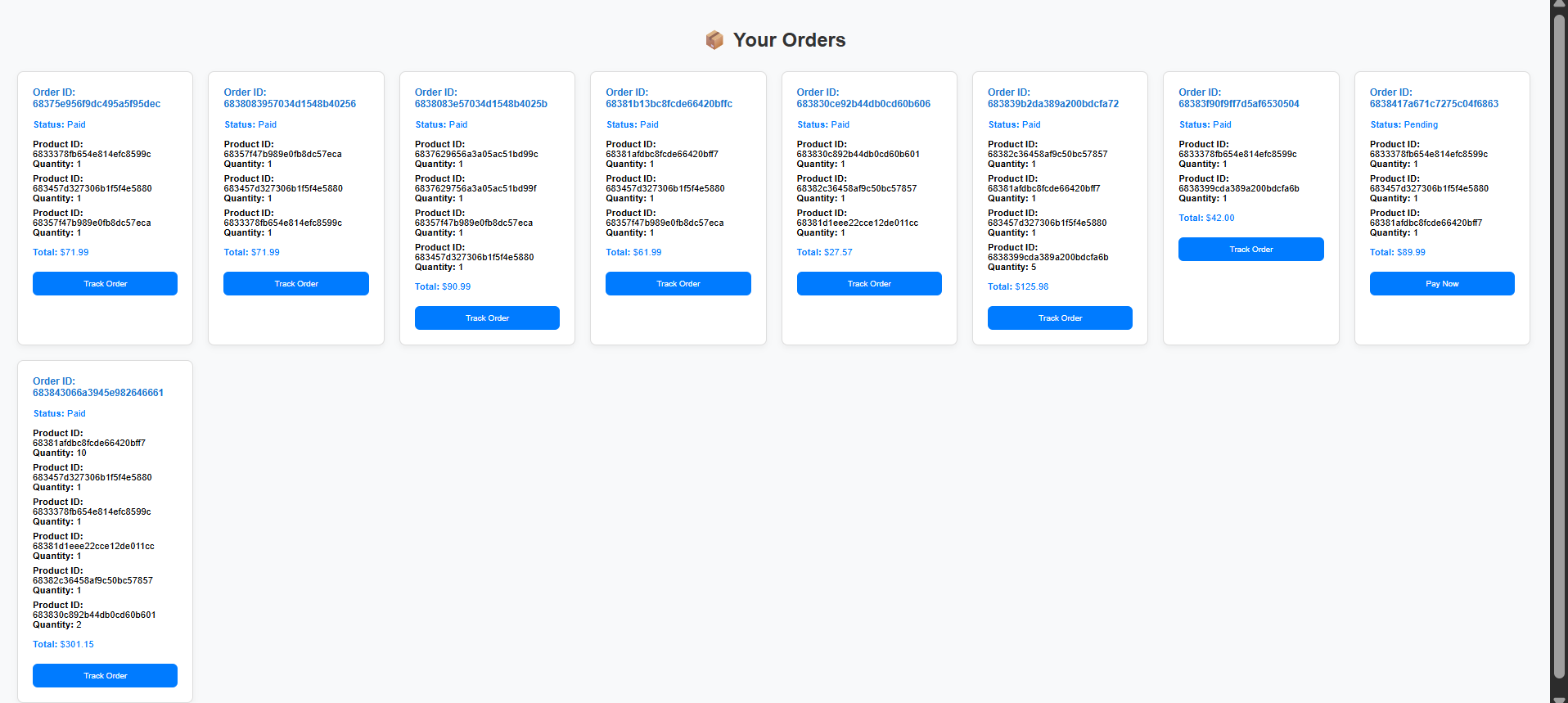
Snippet of Pages in UI Dashboard

# A screenshot of a computer AI-generated content may be incorrect.

A blue and white web browser

AI-generated content may be incorrect.

# A screenshot of a product management AI-generated content may be incorrect.



# A screenshot of a phone AI-generated content may be incorrect.

# Appendix C: Collection relationship table

| **Collection** | **Related To** | **Type of Relationship** | **Notes** |
| --- | --- | --- | --- |
| Users | Orders, Reviews, Transactions | Referenced | One user can place multiple orders, write reviews, and make multiple transactions |
| Products | Categories, Reviews, Orders | Mixed (Embedded + Referenced) | Product belongs to one category (referenced), has embedded reviews, and can appear in many orders |
| Categories | Products | Referenced | Each category contains multiple products |
| Orders | Users, Products, Transactions | Referenced (Array of Product IDs) | Each order is linked to a user and may reference multiple products and a transaction |
| Reviews | Users, Products | Embedded (in Products) or separate collection | Option to embed in Products, or store separately and reference both user and product |
| Transactions | Users, Orders | Referenced | A transaction record is linked to both the user and the order it paid for |

# Appendix D: README (Deployment Guide)

Prerequisites:

* Node.js
* MongoDB Atlas account
* Git
* MongoDB aggregation examples

Steps:

1. Clone the repo
2. Run npm install
3. Setup .env with MongoDB URI
4. Run the server: npm start
5. Use REST CLIENT to test endpoints

Steps for Docker Deployment Option:

1. Clone the repository
2. Add .env with your MongoDB URI
3. Run: “docker compose up” in terminal
4. Access API via browser