Review of MongoDB as a choice of database in Online Banking Applications

Parinaaz Doctor

Abstract - **Online transactions have become an integral part of the banking sector in recent times. There are millions of transactions happening every second of every day. Wireless transactions that are fast and reliable have become the need of the hour. With people moving from wired to wireless devices as a preferable method for online transactions, availability is to be taken into consideration. While relational databases like Oracle, Microsoft SQL Server, and many more have provided reliability and security, it is yet to address availability and scalability. Many organizations have switched from relational to non-relational databases also known as NoSQL (Not Only SQL) in recent times for their ability to store and retrieve data generated by transactions across distributed databases.**

The purpose of this paper is to explain the advantages of storing ever changing data using NoSQL and the challenges developers would face with NoSQL.

Keywords: **Online Banking, NoSQL, Securities in NoSQL, Reliability of NoSQL, MongoDB**

I. INTRODUCTION

Online banking[1] is becoming increasingly popular because of the obvious flexibility it provides. A user can avoid large queues at a bank and the hassle of going to a bank and remotely carry out transactions from anywhere in the world at home, at work or on the go no matter which branch the user has an account in with just a mobile device, an internet connection and a bank or debit card. The user needs to register for the banks online service for that they must create a password and other credentials according to the banks policy. After the completion of the same, the user can avail different online banking services. Different banks offer different online services. Most banks generally offer basic services such as transfers and bill payments. Some banks also allow customers to open new accounts and apply for credit cards through online banking portals. Other functions may include ordering a cheque book, putting stop payments on cheques, or reporting a change of address.

With rapid changes in technology and the need to accommodate diverse forms of unstructured data in an application. NoSQL is a good replacement of the traditional RDBMS technology. NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. They have gained popularity with applications that focus on the social media domain. It also handles documents, emails, multimedia, and other forms of unstructured data efficiently.[2] NoSQL databases take different approaches towards storing data. NoSQL databases support multiple data models like graph, document model, wide-column, and key-value among others. It's easier to scale NoSQL databases than the traditional RDBMS because of their masterless peer-to-peer architecture. Thus, improving performance, allowing for continuous availability, zero downtime and very high read/write speeds and can adapt to high data volume and high complexity. NoSQL databases can handle structured, semi-structured and unstructured data while relational databases need a particular structure defined to store data. [3]

For NoSQL to be a choice of database for online banking applications is quite debatable. There is no correct database for an application, there is only a suitable database. While choosing a database management system, the type and the amount of data the application will handle, the type and frequency of transactions taking place, logging capabilities, security, availability, scalability, among other things is to be taken into consideration. Online banking applications involve a lot of sensitive data, high intensity transactions, and high risk and failure management capabilities.

II. COMPARISON BETWEEN RELATIONAL DATABASES AND NOSQL DATABASES

Relational database management system or an RDBMS follows the relational data model to store data. In the relational data model, data is stored in the form of relations or tables where a table may or may not have a relationship with the other tables. A table has rows and columns and each tuple in the table has a combination of unique data. It keeps the integrity of data in check with these relations. To ensure integrity there are constraints like primary key constraint, foreign key constraint, check constraint, unique constraint, etc. The check on these constraints is kept by the database management system and every time new information is entered i.e., when a new tuple is created it checks all the constraints that have been defined when specifying the structure of the relation.

On the other hand, NoSQL databases are non-tabular databases and store data differently than relational databases. There are different types of NoSQL databases based on the type of data model they use.

The most used NoSQL databases are:

1. Document Databases
Document databases are nonrelational databases that can store data as JSON(JavaScript Object Notation)-like documents. Since, the JSON data is used mostly in application code it becomes easier for developers to store and query data. A document database is flexible because there is no structured format, and each document is unique, and it can evolve over time. Hence, document databases are best suited for product information on online shopping websites, user profiles on social media and the like and content management systems which accept, store and manage dynamic content from the application users. Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents.[4]

2. Key-Value Databases
A key-value database uses the simple concept of key-value pairs to store data.[7] Like a dictionary, the key-value database is uniquely indexed by the key field. In a key-value database there is a key which uniquely identifies the location of the stored data/ value and values can be accessed using this key. Values can be numbers, strings, JSON, XML, BLOB, images, videos, etc. This type of database has no query language associated with it. Keys/values can be added using put which adds or updates data if the key already exists. Key/values can be queried using get and can be deleted using the delete command from the application. In addition, database users can only use keys and not values to work with data. [5]

3. Graph Databases
Graph databases use graph structure to store data. Data Entities are stored in the form nodes and the edges are used to store relationships between these entities. Relationships have labels and may have properties. A relationship has a start node, end node, direction and type. It can be used to describe parent-child relationships among entities, actions, etc. An entity can have multiple relationships with other entities.[6]

4. Wide Column Databases
Wide Column Databases are NoSQL databases with flexible columns. The column names, type and number of columns can vary across different rows within the same table. Unlike in RDBMS, where columns are common across all the rows in a table here a column only exists if it has a data element/value present for it in that row. There is a row key for each data element and based on the row key each individual column for that row is queried.[7]

Considering all database technologies using NoSQL and SQL, the differences between the two are
SQL uses the relational model to store data whereas NoSQL may use a variety of different models like graph, semi-structured, key-value based, document based and so on.
SQL requires a structure to be defined because it is itself a structured query language, so structure needs to be defined before storing, accessing, or modifying data. On the other hand, NoSQL has a dynamic schema which can modify itself based on the type of information coming in from the application. Also, the syntax of NoSQL varies from database to database, but the syntax of SQL remains the same irrespective of the database.
NoSQL databases are horizontally scalable which means if there is more load on one server, it can be handled by taking some load off that server by adding more servers to handle the database or by sharding(dividing a larger database into small parts) whereas SQL databases are vertically scalable i.e., load on these database servers can be handled by increasing the memory, the processor speed or the capacity of a secondary storage device.
SQL databases maintain their consistency and integrity by keeping in compliance with the ACID(Atomicity Consistency Isolation Durability) properties whereas NoSQL databases maintain consistency and availability by following the CAP

(Consistency Availability Partitioning) theorem. However, because of NoSQL databases gaining popularity and the need to give consistent solutions to database users Oracle NoSQL Database Cloud service and the latest version of MongoDB are NoSQL databases that do support ACID properties.

From the above differences it can be concluded that SQL databases are focused on maintaining consistency, security and integrity and NoSQL databases are focused on providing availability, ease of access, and flexibility.

MongoDB, DynamoDB by AWS, CosmosDB by Microsoft, Oracle NoSQL Database Cloud Service are some popular examples of NoSQL databases. Oracle 10g, Microsoft SQL Server, MariaDB are some examples of SQL databases.

With giants like Microsoft, Amazon, and Oracle, providing NoSQL databases is an example how NoSQL databases are being considered as the future of data storage solutions.

Data stored in a SQL database is normalized and then accessed using joins on the other hand data stored in NoSQL is denormalized.

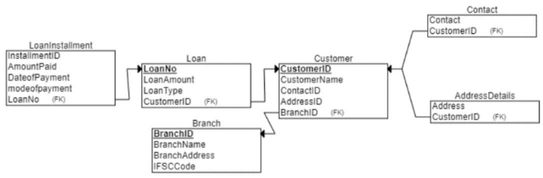

Figure 1: Relational Model for Loan Information of a Customer in a Bank

The above relational model uses the 3NF to store data.

The same information in a NoSQL database would be stored as

[{customerid: 'C001',

branchid: 'B001',

branchname: 'Andheri',

branchaddress: 'Mumbai',

customername: 'John Doe',

CustomerContact : ['9985545222','54544656'],

CustomerAddress: ['ABC, some street', 'PQR some street'], loandetails: [{loanno: '554',

loanamount:9500,

loantype: ‘PersonalLoan’,

installmentdetails:[{installmentno: 1,

dateofpayment: 2022-10-20,

amountpaid: 5000,

modeofpayment: ‘ECS’}]}]]}]

III. TRANSACTIONS & SECURITY CONSIDERATIONS IN ONLINE BANKING APPLICATIONS

Internet Banking is carrying out transactions online by using the website or the app provided by the respective bank. Online banking transactions like National Electronic Fund Transfer (NEFT), Real Time Gross Settlement (RTGS), Electronic Clearing System (ECS) and Immediate Payment Service (IMPS)[8] have become an integral part of banking. Due to the transfer of funds online, regulatory boards have laid down some security and technology standards which require all banks to have a security policy. This policy must be approved by the board of directors. Logical access control such as authentication, authorization, identification protocols have to be utilized to protect the servers on which sensitive information is stored from unauthorized access. There should be proper logging to maintain the history of transactions carried out using online banking services.

RBI states that the security and technology standards of all banks need approval by the board of directors.

Access control is to specify who will be able to access a particular resource or a particular place. It is used to regulate who is able (or not able) to view, access, or use specific resources and/or information. The two main types are physical and logical.

Physical access control refers to the selective restriction of access to a location, a task most often accomplished with a variety of security methods that control and track who is entering a location and who is leaving.

Logical access control is defined as restricting virtual access to data; it consists of identification, authentication, and authorization protocols utilized worldwide to protect hardware from unauthorized access, including password programs, smart cards, or tokens to identify and screen users and access levels.[9]

According to RBI, the database systems, application software, etc, need to have logical access control like smart card readers,

All computer accesses, including messages received, should be logged. Security violations (suspected or attempted) should be reported and follow up action taken should be kept in mind while framing future policy. Banks should acquire tools for monitoring systems and the networks against intrusions and attacks. These tools should be used regularly to avoid security breaches. The banks should review their security infrastructure and security policies regularly and optimize them in the light of their own experiences and changing technologies. They should educate their security personnel and the end-users on a continuous basis. The information security officer and the information system auditor should undertake periodic penetration tests of the system, which should include:

Attempting to guess passwords using password-cracking tools.

Search for back door traps in the programs.

Attempt to overload the system using DDoS (Distributed Denial of Service) & DoS (Denial of Service) attacks.

Check if commonly known holes in the software, especially the browser and the e-mail software exist.

The penetration testing may also be carried out by engaging outside experts or ethical hackers.

All applications of banks should have proper record keeping facilities for legal purposes. It may be necessary to keep all received and sent messages both in encrypted and decrypted form.

IV. OVERVIEW OF MONGODB

MongoDB is the most popular NoSQL database. It is an open-source database that uses the document-based data model for storing data. In MongoDB a database is acts as a container for collections and collections could be compared to tables in the RDBMS. Data is stored in the form of documents in each collection. A document is a set of key-value pairs, and they follow a dynamic schema. A dynamic schema has no structure specified at the time of creating a collection. The structure of each document in a collection can be unique and can change based on the data being stored. The documents can be thought of like JSON objects. [10]

An example of a MongoDB insert statement

```
db.loaninfo.insert({

_id: Object(785f41b2d),

customerid: 'C001',

branchid: 'B001',

branchname: 'Andheri',

branchaddress: 'Mumbai',

customername: 'John Doe',

CustomerContact : ['9985545222','54544656'],

CustomerAddress: ['ABC, some street', 'PQR some street'],

loandetails: [{loanno: '554',

loanamount:9500,

loantype: ‘PersonalLoan’,

installmentdetails:[{instalmentno: 1,

dateofpayment: 2022-10-20,

amountpaid: 5000,

modeofpayment: ‘ECS’}]}]

})
```

In the above example, loaninfo is the collection which contains information about the loans taken by a customer, the branch, the loan details, and the instalment details. The detail of each customer is stored as a document and since a customer can take multiple loandetails are stored as an array in the customer document itself.

ACID Properties in MongoDB

ACID is for Atomicity, Consistency, Isolation and Durability. The four most important properties to be maintained during transactions. Since, online banking is based on transactions it is important to look for database systems which comply with ACID properties. All SQL databases follow ACID properties to maintain integrity and consistency through the database.

A Atomicity (all or nothing)

C Consistency (transaction can never allow inconsistent data to be stored in the database)

I Isolation (transactions execute in isolation no matter how many transactions are executing concurrently)

D Durability (committed transactions persist even after system failure or loss of connectivity)[11]

BASE

B Basically Available (since NoSQL uses partitioning data will be available under all circumstances)

S Soft state (the application has to guarantee consistency of the transaction)

E Eventually Consistent (Immediate consistency isn't promised in NoSQL databases but eventually the database will achieve a consistent state)[12]

CAP Theorem

The CAP theorem was introduced by Eric Brewer in which he stated that there is a fundamental trade-off between consistency, availability, and partition tolerance. [13]

Consistency is the first requirement of the CAP theorem which specifies that at any given point in time data should be consistent.

Availability is to guarantee a response to each request sent by the client even if the data is not consistent across the system or its not the last updated data. Data should still be available on each request.

Partition-tolerance as the term suggests is tolerance to partition ie the cluster will function even if there is a break in communication between two nodes.[14]

One will have to overlook any one property in the CAP theorem i.e., either one chooses consistency and availability(CA) or consistency and partition tolerance (CP) or availability and partition tolerance (AP).[15]

Consistency and Availability guarantee consistent data that is always available, but partitioning will not be in sync. Most of RDBMS systems like SQL Server, MySQL demonstrate the CA combination of the CAP theorem.[16]

In Consistency and Partition Tolerance, the data will be distributed across nodes and all nodes will have the same data but in case of node failure data will not always be available. MongoDB is a CP system. MongoDB prioritizes consistency in every scenario.

Availability and Partition Tolerance will make sure the system is always available even if the nodes are not communicating with each other like in case of a communication failure. There is no guarantee that all nodes will have the same or consistent data. [17]

MongoDB version 4.0 and later shows support for multi-document ACID transactions and extended support for distributed multi-document transactions in version 4.2 in 2019. Multi-document ACID transactions in MongoDB will guarantee atomic updates and consistent data across distributed systems. When performing a write operation in a multi-document transaction, the updated value would be available across clusters only once the transaction has committed. [18][19]

To start a multi-document transaction, use the Mongo shell method session.startTransaction(). At a given time, there will be only one open transaction in a particular session. This is like START TRANSACTION in MySQL.

session.commitTransaction() is a mongo shell method used to commit or save the changes made to the document in the transaction. This is like COMMIT command in MySQL.

session.abortTransaction() aborts a transaction and all the data changes will be rolled back, and no changes made by the transaction will be saved. This is same as the rollback command in MySQL.

With the introduction of multi-document transactions in MongoDB it can be considered as a database of choice for banking applications.[20]

V.ISSUES WITH MONGODB

A widely used database storage option is MongoDB, but there are potential pitfalls to using MongoDB. [21]

1. Normalization

MongoDB is denormalized, here, data is stored in nested document structures and not in tables that are related using the primary and foreign keys. Even though the search is faster in MongoDB, denormalization sometimes leads to redundancy.

2. Reliability

Since the writes in MongoDB are asynchronous by default, there is no need to wait for confirmations for every insert or update operation, as even though some of the updates are unsuccessful the write operations will still complete successfully, however, this gives rise to inconsistent data across the database. In a banking application there is no room for inconsistencies, so an additional effort must be made to guarantee the "all or nothing" of a Relational Database Management System. MongoDB can be configured for synchronous writes (e.g., using appropriate write concerns) to enhance reliability, though this might impact performance.

3. Schema-less Design

In MongoDB data is stored in nested JSON objects called documents. Even though this gives flexibility with the data models, the schema-based design decisions are dependent on the application logic rather than the database. Even though this gives your application more control, the application tends to get complex at times as the data models must be specified by the application unlike Relational Database Management Systems.

VI. SECURITY FEATURES IN MONGODB

1. Encryption of Data

MongoDB allows users to encrypt data when it is stored in databases, when it is in transit and when it is being used by applications. Users can make use of Wired Tiger Storage Engine, Key Management System, and Cloud Provider Integration.[22] For encryption in use, users can make use of Client-Side Encryption, Field-Level Encryption and Queryable Encryption, allowing users to run expressive queries without decrypting the encrypted dataset. To configure encryption, users need to manage the encryption keys[23], users also need to configure the MongoDB driver to enable encryption by specifying the encryption algorithm along with the key management details, the application code will include the encryption and decryption details of specific fields, and schema validation to enforce encryption of specific fields in the collections. [24]

2. Authentication and Authorization

Authentication is a process which verifies the client's identity and authorization is the verification of authority. The first step is authentication, and the next step of security is authorization which gives you access control based on the authenticated user or the application which is supposed to access and work with data in the database.

Authentication mechanisms for MongoDB include

SCRAM-SHA-256 which is the default authentication mechanism.

X.509 certificates allow clients to authenticate themselves using digital certificates. [25]

Kerberos Authentication uses the Kerberos protocol which is the industry standard authentication protocol. It is used for large client/ server systems. It provides authentication by issuing short lived tokens as known as tickets.

OpenID Connect and OAuth 2.0 are external identity providers that enable authentication as well as authorization.[26]

Plain Authentication can be used with active directory. It is simple however less secure. [27]

Authorization and access control in MongoDB can be established using roles, user management, authentication database and Access Control Lists.[28]MongoDB implements role-based access control (RBAC) allow only users with specific roles to access specific data.[29]

3. Network Security

The best practices to ensure network security while using MongoDB are

Firewall Configuration for restricting access to MongoDB clusters. Disable Direct SSH Root Access which includes avoiding direct root access to the server.[30] Private Networks use only private IP addresses to access the clusters for internal communication. IP Access Lists allow only authorized IP addresses to connect and access the MongoDB cluster.[31]

When discussing about banks irrespective of the database being used, the network security is well taken care of by firewalls and private networks used by most banks.

4. Auditing and Logs

Maintaining logs is an efficient way of maintaining the integrity of data. MongoDB enterprise offers a wide range of auditing capabilities. It will log all the actions executed against a database as well as access granted to a database. This can provide an added advantage for using MongoDB in banking applications. MongoDB allows users to configure and filter audit trials as well as configure MongoDB logging.

The auditing system in MongoDB Enterprise supports multiple output targets for audit events: the console, syslog, JSON files, and BSON files. Enabling this functionality requires configuring an audit output destination via the --auditDestination parameter. Upon activation, the auditing system captures a range of operational data, including Data Definition Language (DDL) or schema changes, activities related to replica sets and sharded clusters, authentication and authorization processes, and standard CRUD operations. [32]

VII. CONCLUSION

This paper reviews MongoDB as a potential database of choice for online banking applications. Though the strong guarantee that traditional database systems, like relational databases offer on security, consistency and reliability, the strengths of NoSQL databases such as MongoDB lie in handling large volumes of diversely changing data along with their superior scalability and availability, present an attractive proposition for the high-transaction, always-on nature of modern online banking.

The trade-offs of NoSQL systems like the concerns around their compliance with ACID properties and robust security mechanisms made them debatable choices for such critical financial applications. However, as explored, MongoDB has made significant strides in addressing these historical limitations. The introduction of multi-document ACID transactions in versions 4.0 and later is a pivotal development, bridging a critical gap with RDBMS capabilities. In addition to that, the security features provided encompassing encryption (in transit, at rest, and in use), multi-factor authentication, role-based access control, and network security, substantially mitigate risks previously associated with NoSQL deployments in sensitive environments.

Despite these advancements, challenges such as managing denormalization, the implications of a schema-less design shifting complexity to the application layer and ensuring the rigorous data precision demanded by financial transactions remain pertinent considerations. The adage "there is no correct database for an application, there is only a suitable database" holds particularly true.

In conclusion, while relational databases continue to offer a proven track record for core banking systems, MongoDB, with its enhanced ACID compliance and comprehensive security features, is increasingly becoming a viable and compelling option for specific services within the online banking ecosystem, especially those requiring high scalability and flexibility. The decision to adopt MongoDB must be based on a careful evaluation of the specific application's requirements, risk tolerance, data characteristics, transactional integrity needs, and the organization's capacity to manage the architectural nuances of a NoSQL database. As MongoDB and other NoSQL technologies continue to mature, their role in the financial sector is likely to expand, but always contingent upon a rigorous assessment against the stringent demands of online banking.

REFERENCES

[1] Investopedia. "Online Banking." Available: https://www.investopedia.com/terms/o/online banking.asp.

[2] MongoDB. "NoSQL Explained." Available: https://www.mongodb.com/nosql-explained.

[3] DataStax. "5 Features to Look for in a NoSQL Database." DataStax Blog, [September 6, 2017]. Available: https://www.datastax.com/blog/5-features-look-nosql-database.

[4] AWS. "Document Databases Explained." Available: https://aws.amazon.com/nosql/document/.

[5] AWS. "Key-Value Databases Explained." Available: https://aws.amazon.com/nosql/key-value/.

[6] AWS. "Graph Databases Explained." Available: https://aws.amazon.com/nosql/graph/.

[7] ScyllaDB. "What is a Wide Column Database?" ScyllaDB Glossary. Available: https://www.scylladb.com/glossary/wide-column-database/.

[8] Cashless India, Ministry of Electronics & Information Technology, Government of India. "Internet Banking." Available: http://cashlessindia.gov.in/internet_banking.html.

[9] Identiv. "What's the Difference Between Logical Access Control and Physical Access Control?" Identiv Blog, [August 6, 2020]. Available: https://www.identiv.com/resources/blog/whats-the-difference-between-logical-access-control-and-physical-access-control.

[10] A. Chauhan, "A Review on Various Aspects of MongoDB Databases," International Journal of Engineering Research & Technology (IJERT), vol. 8, no. 5, May 2019. Available: https://www.ijert.org/research/a-review-on-various-aspects-of-mongodb-databases-IJERTV8IS050031.pdf.

[11] MongoDB. "ACID Transactions in MongoDB." MongoDB Basics. Available: https://www.mongodb.com/basics/acid-transactions. .

[12] P. Vanroose and K. Van Thillo, "ACID or BASE? - The case of NoSQL," White Paper/Presentation, GSE DB2 Belgium Joint User Group Meeting, IBM, Brussels, 12 June 2014.

[13] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," ACM SIGACT News, vol. 33, no. 2, pp. 51-59, June 2002.

[14] Instaclustr. "Cassandra vs. MongoDB." Instaclustr Blog, [August 03, 2021]. Available: https://www.instaclustr.com/blog/cassandra-vs-mongodb/. .
[15] B. M. Coker. "What is CAP Theorem? Brewer's CAP Theorem Explained." BMC Blogs, [October 30, 2024]. Available: https://www.bmc.com/blogs/cap-theorem/. .
[16] Mantra Labs. "MongoDB Multi-Document ACID Transactions: A Deep Dive." Mantra Labs Blog, [Oct 16, 2019]. Available: https://www.mantralabsglobal.com/blog/mongo-db-multi-document-acid/. .
[17] E. A. Brewer, "Towards robust distributed systems (Invited Talk)," in Symposium on Principles of Distributed Computing (PODC), Portland, Oregon, July 2000.
[18] DigitalOcean. "How To Use Transactions in MongoDB." DigitalOcean Community Tutorials, [November 24, 2021]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-transactions-in-mongodb. .
[19] MongoDB. "MongoDB 4.0 Release Notes: Multi-Document Transactions." MongoDB Documentation. Available: https://www.mongodb.com/docs/upcoming/release-notes/4.0/#multi-document-transactions.
[20] MySQL. "13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax." MySQL 8.0 Reference Manual. Available: https://dev.mysql.com/doc/refman/8.0/en/commit.html. .
[21] MongoDB. "Encryption." MongoDB Products: Capabilities. Available: https://www.mongodb.com/products/capabilities/security/encryption.
[22] Prisma. "MongoDB Encryption." Prisma Data Guide. Available: https://www.prisma.io/dataguide/mongodb/mongodb-encryption.
[23] GeeksforGeeks. "How to Encrypt MongoDB Data?" Available: https://www.geeksforgeeks.org/how-to-encrypt-mongodb-data/.
[24] MongoDB. "Edit Host Authentication Credentials." MongoDB Ops Manager Documentation. Available: https://www.mongodb.com/docs/ops-manager/current/tutorial/edit-host-authentication-credentials/.
[25] MongoDB. "Authentication." MongoDB v7.0 Documentation. Available: https://www.mongodb.com/docs/v7.0/core/authentication/.
[26] MongoDB. "Authenticate Using SASL and LDAP with ActiveDirectory." MongoDB v7.0 Tutorial. Available: https://www.mongodb.com/docs/v7.0/tutorial/authenticate-nativeldap-activedirectory/.
[27] MongoDB. "Authorization." MongoDB v7.0 Documentation. Available: https://www.mongodb.com/docs/v7.0/core/authorization/.
[28] Satori Cyber. "MongoDB Security: 11 MongoDB Security Features and Best Practices." Satori Cyber Blog. Available: https://satoricyber.com/mongodb-security/11-mongodb-security-features-and-best-practices/.
[29] MongoDB. "Security Checklist." MongoDB Manual. Available: https://www.mongodb.com/docs/manual/administration/security-checklist/.
[30] MongoDB. "Set Up Cluster Security." MongoDB Atlas Documentation. Available: https://www.mongodb.com/docs/atlas/setup-cluster-security/.
[31] MongoDB. "Auditing." MongoDB Manual. Available: https://www.mongodb.com/docs/manual/core/auditing/.