# **ONLINE BANKING APPLICATION WITH MONGODB NOSQL DATABASE**

# 

# **ACKNOWLEDGEMENT**

I would like to thank [name of your organization] for giving me the opportunity to work on this project. I would also like to thank our trainers and mentors for their guidance and support throughout the development of the Online Banking Guide.

**TABLE OF CONTENTS**

[**ONLINE BANKING APPLICATION WITH MONGODB NOSQL DATABASE 1**](#_cex8s2nqu4ah)

[**ACKNOWLEDGEMENT 2**](#_juapwss8t4lc)

[**INTRODUCTION 4**](#_55u4y81jq4vs)

[**BACKGROUND AND PROBLEM STATEMENT 4**](#_8mf2o18v5hc9)

[**OBJECTIVES OF THE PROJECT 4**](#_1ttg08bexvp)

[**SCOPE AND LIMITATIONS 5**](#_ngovvrfepi2e)

[**DATABASE DESIGN AND IMPLEMENTATION 5**](#_cdq5ffny9x3m)

[MongoDB Overview 5](#_4qn24mr89dsw)

[Data Modeling 5](#_dj56tyc9qtnx)

[Collections and Document Structure 5](#_fi05c7l8ej1e)

[NoSQL Data Types Used 6](#_xi7s2ls14pgc)

[Features and Functionality 6](#_fbjtensl3mav)

[Implementation Steps 7](#_z1hia893xkkr)

[Database Setup and Customer Data 8](#_vt69o9h3rdlj)

[**CONCLUSION** 12](#_2m120qej839q)

[**ACHIEVEMENTS AND LEARNING OUTCOMES 12**](#_pda6cb57niwn)

[**FUTURE ENHANCEMENTS 13**](#_xdse9rortbi)

[**REFERENCES 14**](#_5jvde2nvdjac)

[**APPENDIX 15**](#_28ayzxjkjn22)

[Code Snippets 15](#_uaqmtyosj805)

# 

# **INTRODUCTION**

In today’s digital age, the financial sector is undergoing a major transformation, and online banking has become an integral part of modern banking operations. The ease and accessibility offered by online banking applications has transformed consumer interaction with finance institutions. A robust and flexible technology to manage customer accounts, transactions and relationships is essential for banks to meet the evolving demands of customers This project involves the development of an Internet Banking Application using MongoDB, Providing a NoSQL database for an efficient and scalable solution for ma

# **BACKGROUND AND PROBLEM STATEMENT**

Traditional relationship databases have been widely used in the banking sector for data management. However, the limitations of these databases, such as structural complexity and scalability challenges, led to the search for alternative solutions. NoSQL databases, with their flexible schema structure and horizontal scalability, are emerging as a powerful way to manage rapidly growing datasets There is a need for online banking applications that manage customer data, transaction processing, and relationships handle effectively Using MongoDB as a NoSQL database Stimulates exploration.

# **OBJECTIVES OF THE PROJECT**

The primary objectives of this project are as follows:

***Develop an Online Banking Application***: Develop and operate a web-based application that allows customers to create accounts, perform transactions, view account history, and manage their banking transactions

***Utilise MongoDB NoSQL Database***: Explore MongoDB's ability to store and manage customer data, transactions, and relationships more efficiently.

***Demonstrate NoSQL Data Types:*** Use the various data types provided by MongoDB, including tabular (column-based), document (JSON), key-value pairs, and graph data structures

# **SCOPE AND LIMITATIONS**

A scope of this project includes building an Online Banking Application using MongoDB as the underlying NoSQL database. The application will focus on basic banking services such as customer and account management, transaction processing, account history viewing, and providing graph-based relationships. Due to time and resource constraints, some advanced features, such as real-time analysis, advanced graph querying, and multifactor authentication, may be beyond the scope of this work.

In the following sections, we will analyze the database design, functionality, features and functionality of the Online Banking Application, and the steps taken to achieve the project objectives

# **DATABASE DESIGN AND IMPLEMENTATION**

## **MongoDB Overview**

MongoDB is a popular NoSQL database that provides a flexible and flexible approach to data management. Unlike traditional relational databases, MongoDB uses a form-based model, where data is stored in flexible forms, such as JSON. This enables dynamic scheduling, making it ideally suited to scenarios with increasingly complex data structures, such as those encountered in modern applications such as online banking.

## **Data Modeling**

Data modeling involves organizing data and describing relationships to ensure efficient and meaningful retrieval of data. In our online banking application, we adopt a document-based data model, where each customer and account is represented as a document in MongoDB. This approach facilitates the retrieval and manipulation of customer data and transaction information.

## **Collections and Document Structure**

A MongoDB database contains collections of objects, similar to tables in relational databases. Each collection contains a set of letters, which are personal accounts. In our project we have collections such as "customer" and "account", where "customer" stores customer information and "account" stores account information.

Each document in a collection follows a structured format, usually similar to JSON. This option allows for nested fields, arrays, and complex structures. For example, the "accounts" collection includes nested arrays for transactions, which allow us to store historical transaction data associated with each account.

## **NoSQL Data Types Used**

In our online banking application, we use NoSQL data sets provided by MongoDB to meet different data needs and relationships. These data types include:

**Tabular (Column-Based)**

We use MongoDB BSON format to represent tabulated data. While MongoDB is not as inherently column-based as some other NoSQL databases, its flexible document structure allows us to store table-like data more efficiently.

**Document (JSON)**

The document data type is at the heart of MongoDB’s design. Every customer, account, and transaction record is stored as a JSON-like document, allowing us to model complex data structures, and easily manage the ability to query and perform additional calculations.

**Key-Value Pair**

While not a core concept of MongoDB, we can use MongoDB’s basic document structure to represent key-value pairs. For example, customer preferences or policy settings in a customer document can be stored as key-value pairs.

**Graph**

Graph-based relationships are important in representing interactions between customers, accounts, and services. Although MongoDB is not a dedicated graph database, we can model relationships between entities as graph nodes using MongoDB documents and references.

## **Features and Functionality**

**Customer Management**

Customer care is an integral part of online banking application. It involves creating, editing, and deleting customer records. Personal information about customers, including their names, email addresses, phone numbers, and addresses, is stored in the MongoDB "Customers" collection.

**Account Management**

Account management allows users to create, renew and close bank accounts. Each account is associated with a customer and stores information such as account number, account type, and account balance. Account-related data, including transactions and account history, is efficiently managed using MongoDB’s document structure.

**Transaction Processing**

Transaction processing is an important function of an online banking application. This allows customers to deposit, withdraw and transfer funds between accounts. Transactions are recorded using information such as dates, terms and amounts. MongoDB’s capabilities make it easy to efficiently track transaction history and balance changes.

**Account History and Reporting**

The application allows users to view their account history and create reports. Users can retrieve transaction records, view account balances, and get detailed reports on their financial activity. MongoDB’s query capabilities enable you to easily reconstruct historical data for complete reports.

**Graph-Based Relationships**

Graph-based relationships represent the interactions between customers, accounts, and transactions in an online banking application. MongoDB's document reference and nesting features allow us to model these relationships as graph nodes and edges. This allows for the visualization and analysis of complex customer-account relationships and ongoing interactions.

## **Implementation Steps**

***Database Setup and Customer Data***: Create "online\_banking" database and "customer" collection. Insert sample customer documents with their associated account and transaction data.

***Account Management Functions:*** Develop functions for creating new accounts, updating account details, and closing accounts. Used MongoDB's document structure to manage account-related data.

***Transaction Processing Functions:*** Implement functions to process deposits, withdrawals, and transfers. Maintain customer history and update the remaining accounts using MongoDB services.

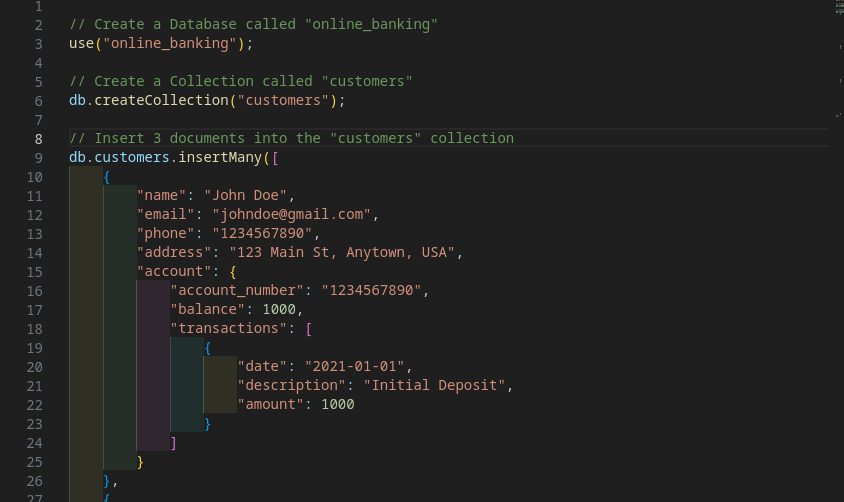
***Account History and Reporting Functions:*** Create functions to retrieve account history, generate reports, and provide account information to users.

***Graph Representation Functions***: Create functions for creating and managing graph nodes representing customers, accounts, and transactions. Use the MongoDB document reference to establish a relationship.

In the following sections, we will provide detailed insights into the actual use of the Online Banking Application, showing code snippets and explaining how to do each step.

## **Database Setup and Customer Data**

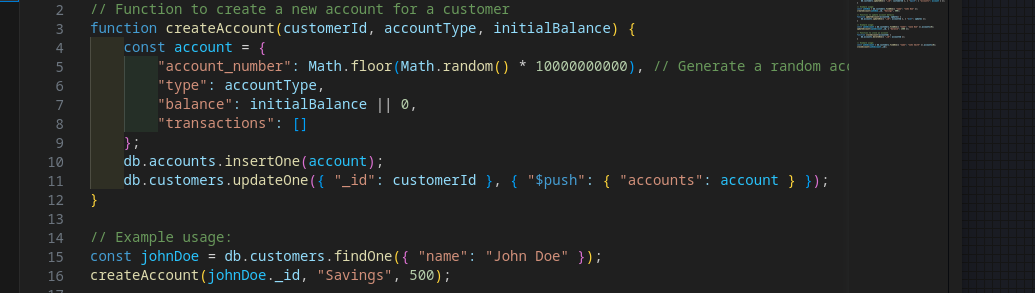
To initiate the project, I created a MongoDB database named "online\_banking" and a collection named "customers". These aggregates store customer information, including name, email, phone number, address, and associated account information. For illustrative purposes, we have added a sample of customer documents containing preliminary account transaction information to the "Customer" collection.



**Account Management Functions**

**Create Account**

The createAccount function enables you to create a new account for a customer. It’s a random account number and starts the account with a specific type and opening balance. Account information is stored in the "Account" collection, and the relationship between the customer and the account is established.



**Update Account**

The updateAccount function allows you to edit account information such as account balance and type. "Accounting" uses the MongoDB update function to update simple account information in the collection.



**Close Account**

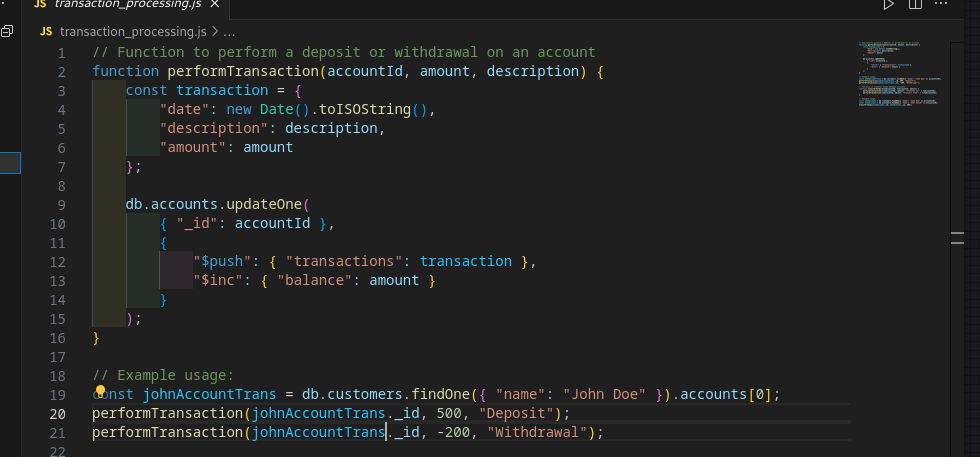
The closeAccount function makes it easy to close an account. It removes the account from the "accounting" collection and ensures that the associated customer record is updated accordingly.



**Transaction Processing Functions**

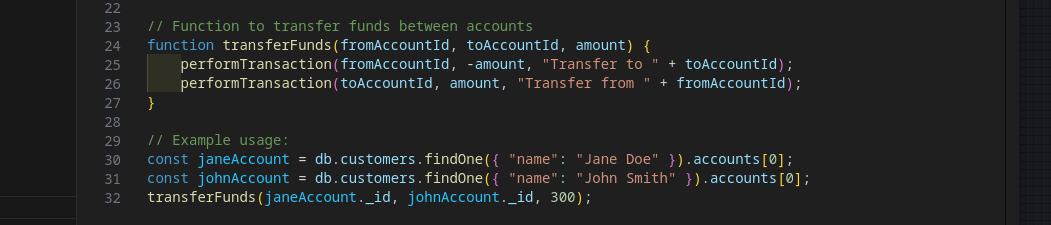
**Perform Transaction**

The performTransaction function handles deposits and withdrawals to customer accounts. Records the transaction information, such as date, description, and amount, in the associated balance sheet. Account balances are updated using MongoDB’s update operations.

****

**Transfer Funds**

The TransferFunds function allows you to transfer funds between two accounts. It uses the performTransaction function to perform withdrawals and deposits for the sender and recipient accounts, and ensures the accuracy of funds transfers.



**Account History and Reporting Functions**

**View Account History**

The viewAccountHistory function retrieves and displays the transaction history for the specified account. Query the "Accounts" collection to retrieve the transaction records associated with the account.

**Generate Account Report**

The generateAccountReport function generates a detailed report for the specified account at the given date. It retrieves transaction information and summarises accounting finance activities, enabling users to access detailed accounting reports.

**Graph Representation Functions**

**Create Graph Node**

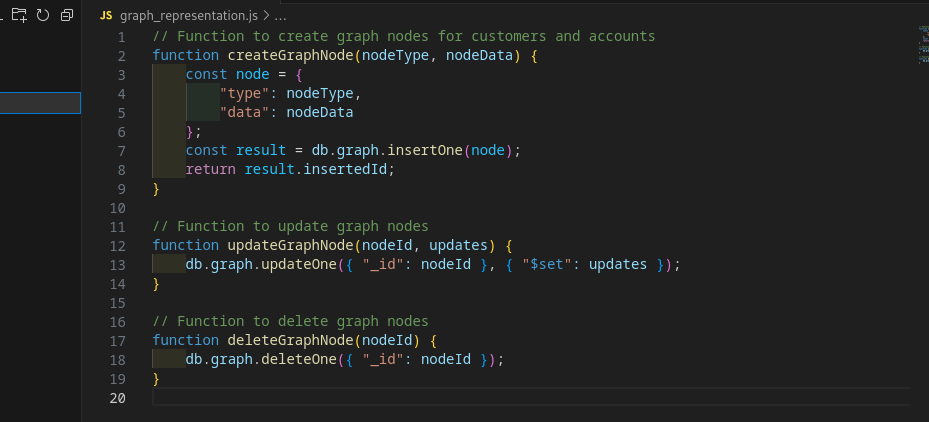
The createGraphNode function creates a graph node to represent customers, accounts, and services. It stores metadata about different nodes and data in a collection of "graphs", and establishes relationships between different entities.

**Update Graph Node**

The updateGraphNode function allows changes to graph node metadata, supporting new dynamic node properties.

**Delete Graph Node**

The deleteGraphNode function removes a graph node from the "graph" collection, effectively deleting the associated relationships.



# **CONCLUSION**

In this project, I successfully developed an Online Banking Application using MongoDB which is an underlying NoSQL database. We explored MongoDB’s flexible document-oriented model to efficiently manage customer information, account data, and transaction records. Application features including customer and account management, transaction processing, account history views, and graph-based relationships highlight MongoDB's capabilities for satisfying a variety of data needs.

Through the use of a variety of projects and workflows, we demonstrated how MongoDB’s data types and quality contribute to the development of comprehensive online banking solutions. Although our work provides a functional foundation, the application can be further developed to extend its capabilities and address other real-world banking scenarios

# **ACHIEVEMENTS AND LEARNING OUTCOMES**

Throughout the development of the Online Banking Application with MongoDB, several key achievements were realized:

**Comprehensive Feature Implementation:** Core banking functionality including customer and account management, transaction processing, account history management, and graph-based relationships have been effectively implemented.

**Proficient use of MongoDB:** Demonstrated proficiency in using MongoDB’s document-oriented approach to effectively manage complex data structures and relationships.

**Modular Codebase:** Used a modular approach to code structure, resulting in maintainable and extensible code that can be easily adapted for future development.

**Graph modeling:** Graph modeling of a document-oriented database was explored, highlighting its potential to represent complex relationships between organizations.

**Practical application of NoSQL:** Gained hands-on experience in applying NoSQL concepts in a real-world environment, focusing on the utility of NoSQL databases in dynamic and evolving environments

# **FUTURE ENHANCEMENTS**

While the current implementation is a solid foundation for online banking applications, there are many opportunities for future growth:

***Enhanced Security***: Implement strong authorization mechanisms to ensure secure user access to accounts and services.

***Real-Time Notifications:*** Integrate real-time reporting to keep users informed about account activity, transactions, and changes.

***Advanced Analytics***: Use advanced analytics features to gain insights into customer behavior, transaction patterns, and statistical trends.

***Multi-Currency Suppor***t: Extend the application to support multiple currencies and foreign currency transactions.

# 

# **REFERENCES**

Kumar, V. (2023, August 5). *Create a Database in MongoDB with ease - The Complete Guide*. Intellipaat.com. https://intellipaat.com/blog/tutorial/mongodb-tutorial/database-creation-using-mongodb/?US

Mittal, A. (2021, August 26). *All Basics of MongoDB in 10 Minutes*. Nerd for Tech. https://medium.com/nerd-for-tech/all-basics-of-mongodb-in-10-minutes-baddaf6b6625

*MongoDB Atlas Tutorial – How to Get Started*. (2021, February 4). FreeCodeCamp.org. https://www.freecodecamp.org/news/get-started-with-mongodb-atlas/

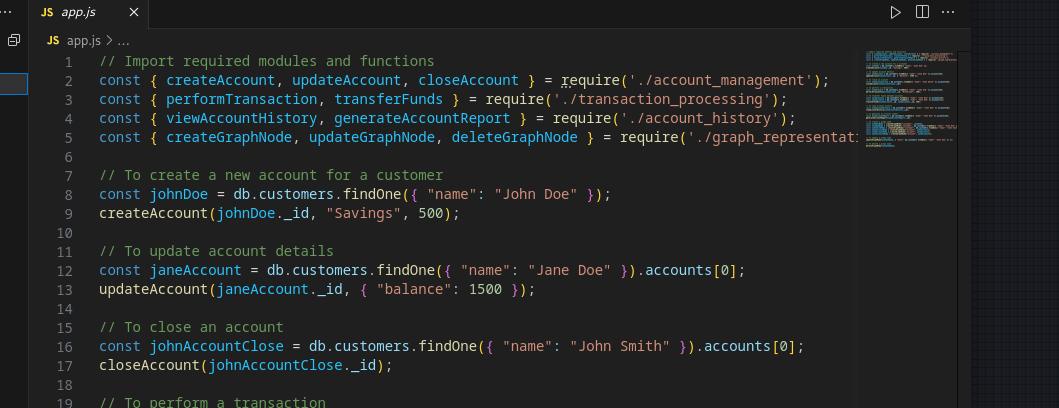
*MongoDB Tutorial*. (n.d.). Www.w3schools.com. https://www.w3schools.com/mongodb/

Saini, A. (2020, February 19). *MongoDB - Database, Collection, and Document*. GeeksforGeeks. https://www.geeksforgeeks.org/mongodb-database-collection-and-document/

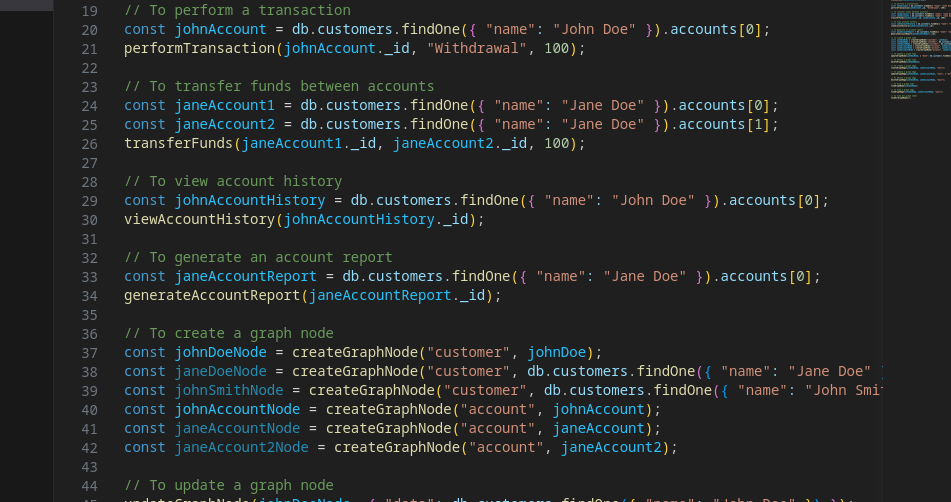
Singh, G. (2021, August 2). *MongoDB — Com*. Medium. https://faun.pub/mongodb-com-50d2f3016c2b?gi=d3b8aa5f30fa

# **APPENDIX**

### **Code Snippets**



app.js



Using The Functions