**LAB 5 Assignment**

**Group Details:**

**PROG8410-SEC2 9**

Vidhya Venugopal – 8902970

**Design and Implementation of a Primary Key-Value Store using NoSQL DB:**

We are trying to build a new application for Insurance agencies and agents:

**Web Application Purpose:** The primary purpose of the web application is to assist insurance agents and agencies in creating insurance quotes for their customers. The application will have a user-friendly Graphical User Interface (GUI) that simplifies generating quotes.

**Extensive Information Handling:** Insurance quotes typically involve much information, such as customer details, policy coverage options, and pricing factors. The web application is designed to efficiently handle and manage this vast amount of data in a structured and organized manner.

**Database Management:** Given the extensive information for each insurance quote, the application requires a robust and scalable database to store and manage all the relevant details. This database will likely be optimized to handle complex queries efficiently and ensure data integrity.

**User Authentication:** Agents can create accounts and log in securely to access their personalized dashboard.

**Quote Management:** Agents can create new quotes, view existing quotes, and update or delete quotes as needed.

**Draft Quotes Feature:** The application allows users to create quotes as drafts before finalizing them. This feature enables agents to work on a quote over multiple sessions, adjusting as needed before submission.

**Draft Status Time Limit:** To prevent drafts from being left unfinished indefinitely, the application incorporates a stipulated time limit for the draft status of a quote. If a quote remains in draft status beyond the defined time frame, the system may prompt the user to complete it or take other appropriate actions.

**Efficient Data Retrieval:** As the application allows for creating complex drafts with extensive information, retrieving and displaying this data in the GUI must be efficient and fast. Proper data indexing and caching mechanisms may be employed to optimize data retrieval performance.

**Quote Updates:** During the draft stage, agents can update the quote details as required. The application should provide smooth and intuitive ways to edit, modify, and add information to the existing quote, making the user experience seamless.

**Integration with Insurance Carriers:** Once the insurance quote is finalized and ready to be submitted to the insurance carriers, the application employs APIs (Application Programming Interfaces) to facilitate seamless communication between the web application and the carrier's systems. This integration ensures that quotes can be sent quickly and accurately to the carriers for processing.

**Data Security and Compliance:** Since the application handles sensitive customer and policy information, security measures like data encryption, role-based access control, and compliance with relevant data protection regulations (e.g., GDPR, HIPAA) will be crucial aspects of the system's design.

**Scalability and Reliability:** As the application may experience varying usage levels, especially during peak hours, it must be designed to handle the load and remain highly available and responsive. Employing cloud-based infrastructure and horizontal scaling techniques can help achieve these goals.

**User Training and Support:** Given the complexity of insurance policies and the application's range of features, insurance agents and agencies should provide adequate training and user support to ensure they can utilize the system effectively.

**Document Storage:** The application allows agents to upload and store necessary documents related to quotes and policies.

**Reporting and Analytics:** Generate reports and insights to analyze sales performance and quote trends.

**Notifications:** Automated notifications for pending quotes and policy renewals.

Overall, the new web application aims to streamline the insurance quote creation process, enhance productivity for agents and agencies, and improve customer satisfaction by providing accurate and timely insurance quotes to their clients.

**a. NoSQL DB Key-Value Store:**

Based on the Insurance agencies/agent portal business requirements stated earlier, we carefully researched NoSQL DB Key-Value stores. We found MongoDB, a NoSQL primary key-valued data store, offers several benefits that align well with the needs of the insurance quote web application:

**Flexible Schema:** MongoDB's document-based data model allows for a flexible schema, meaning each quote can have its own unique set of fields without requiring a predefined schema for all quotes. This flexibility is advantageous when insurance policies vary significantly, and new fields or changes can be introduced without affecting existing data.

**Scalability:** MongoDB is designed to scale horizontally, making it a suitable choice for applications that handle large amounts of data and traffic. MongoDB's scalability capabilities become valuable as the insurance quote application is expected to store a vast amount of quote-related information and handle multiple concurrent users.

**Document-Oriented Storage:** MongoDB stores data in BSON (Binary JSON) documents, a binary representation of JSON data. This format is intuitive and well-suited for working with complex and nested data structures typically found in insurance quotes. It simplifies the process of storing and retrieving JSON-like data.

**High Performance:** MongoDB's primary key-valued data store architecture ensures fast read and writes operations for specific keys (e.g., the quote ID). Since the application needs to retrieve, whole quote details every time, this key-based access allows for efficient retrieval of specific quote documents without the need for complex joins or table scans.

**Indexing Support:** MongoDB allows the creation of various indexes, including compound indexes and geospatial indexes, which can significantly speed up query performance. With the large amount of data expected to be stored and the need for efficient data retrieval, indexing can play a vital role in optimizing database operations.

**JSON-Like Queries:** MongoDB supports rich queries using a JSON-like query language. This feature is beneficial for creating complex queries to retrieve specific subsets of quote data, enabling agents to find and manipulate the information they need quickly.

**High Availability:** MongoDB provides features like replica sets, which offer automatic failover and data redundancy. This DB ensures high availability and minimizes the risk of data loss or downtime in case of hardware failures or maintenance events.

**Integration with API Services:** MongoDB's robust driver ecosystem and extensive community support make it relatively easy to integrate with various programming languages and API services. This DB will facilitate the smooth interaction between the insurance quote web application and external services, such as insurance carriers' APIs for quote submissions.

**Aggregation Framework:** MongoDB's aggregation framework allows for advanced data processing and analytics within the database. This feature can be valuable for generating statistics, insights, and reports based on the collected quote data.

**Horizontal Scaling:** As the number of insurance agents and agencies using the application grows, the demand for the database will increase. By adding more servers to the cluster, MongoDB's horizontal scaling capabilities allow the system to handle the growing data and traffic without significantly losing performance.

Considering the requirements of managing vast amounts of quote data, flexible schema, high performance, scalability, and easy integration with APIs, MongoDB emerges as a strong candidate for the primary key-valued data store for the insurance quote web application. With MongoDB as the backend, the Agent Portal Application can offer insurance agents a reliable and responsive user experience, empowering them to provide excellent customer service.

**b. Implementation Process and Design Decisions**

This report documents the implementation process and design decisions made for the database implementation of the insurance quote web application. The primary focus was selecting an appropriate database solution that meets the application's requirements, such as handling extensive quote information, providing a user-friendly GUI, supporting draft quotes, and seamless integration with insurance carriers through APIs. After thorough research and analysis, MongoDB, a NoSQL primary key-valued data store, was chosen as the database solution for its suitability and benefits in meeting the application's needs.

**Design Decisions:**

**Database Selection - MongoDB:**

After evaluating various database options, MongoDB was chosen due to its flexibility, scalability, and efficient handling of JSON-like data. The document-based model of MongoDB allows for a flexible schema, ideal for accommodating varying insurance policies' data structures and dynamic fields. Additionally, MongoDB's ability to scale horizontally makes it suitable for handling the vast amount of quote data and providing high availability.

**Installation and Setup:**

* <https://www.mongodb.com/> - Open the URL and download the MongoDB community server’s current version.
* Choose the correct platform and package type.
* Once the download is completed, install it.
* Setup Environment Variables.
* Verify successful installation – mongo.
* Verify version – db.version()

Install MongoDB on the server or local machine and configure it as needed. Set up authentication and security to ensure data integrity.

MongoDB Compass is the official graphical user interface (GUI) tool provided by MongoDB for interacting with MongoDB databases. It is designed to simplify exploring, analyzing, and manipulating data stored in MongoDB databases, making it easier for developers, database administrators, and other users to work with MongoDB. MongoDB Compass is available for download and use as part of MongoDB's Community Edition and Enterprise Edition. It is available for Windows, macOS, and Linux operating systems.

**Key Features of MongoDB Compass:**

* Visual Database Exploration - MongoDB Compass offers an intuitive and visual interface to explore MongoDB databases.
* Data Visualization - Compass allows users to visualize the data stored in MongoDB collections using various chart types, including bar charts, line charts, and scatter plots. This feature helps users gain insights from the data and identify patterns or trends quickly.
* Query Builder - With the Query Builder feature, users can construct complex MongoDB queries using a visual drag-and-drop interface.
* Aggregation Pipeline Builder - MongoDB Compass includes an Aggregation Pipeline Builder that helps users build and visualize complex aggregation pipelines.
* Index Management - Compass allows users to create, modify, and drop indexes on MongoDB collections. Efficient indexing is crucial for optimizing query performance in MongoDB, and Compass makes index management more accessible to users.
* Document Validation - Users can define validation rules for MongoDB collections through Compass.
* Explain Plans - Compass allows users to understand query execution and identify performance bottlenecks.
* Real-time Monitoring - Compass includes a real-time monitoring feature that provides insights into the performance of MongoDB deployments. Users can monitor various metrics, such as connections, operations, and memory usage, to assess the database's health.
* Data Import and Export: Compass allows users to import data from various formats (e.g., JSON, CSV) into MongoDB collections and export data from MongoDB collections to different file formats.
* Connection Manager: Compass makes it easy to manage multiple MongoDB connections, allowing users to switch between different MongoDB instances seamlessly.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

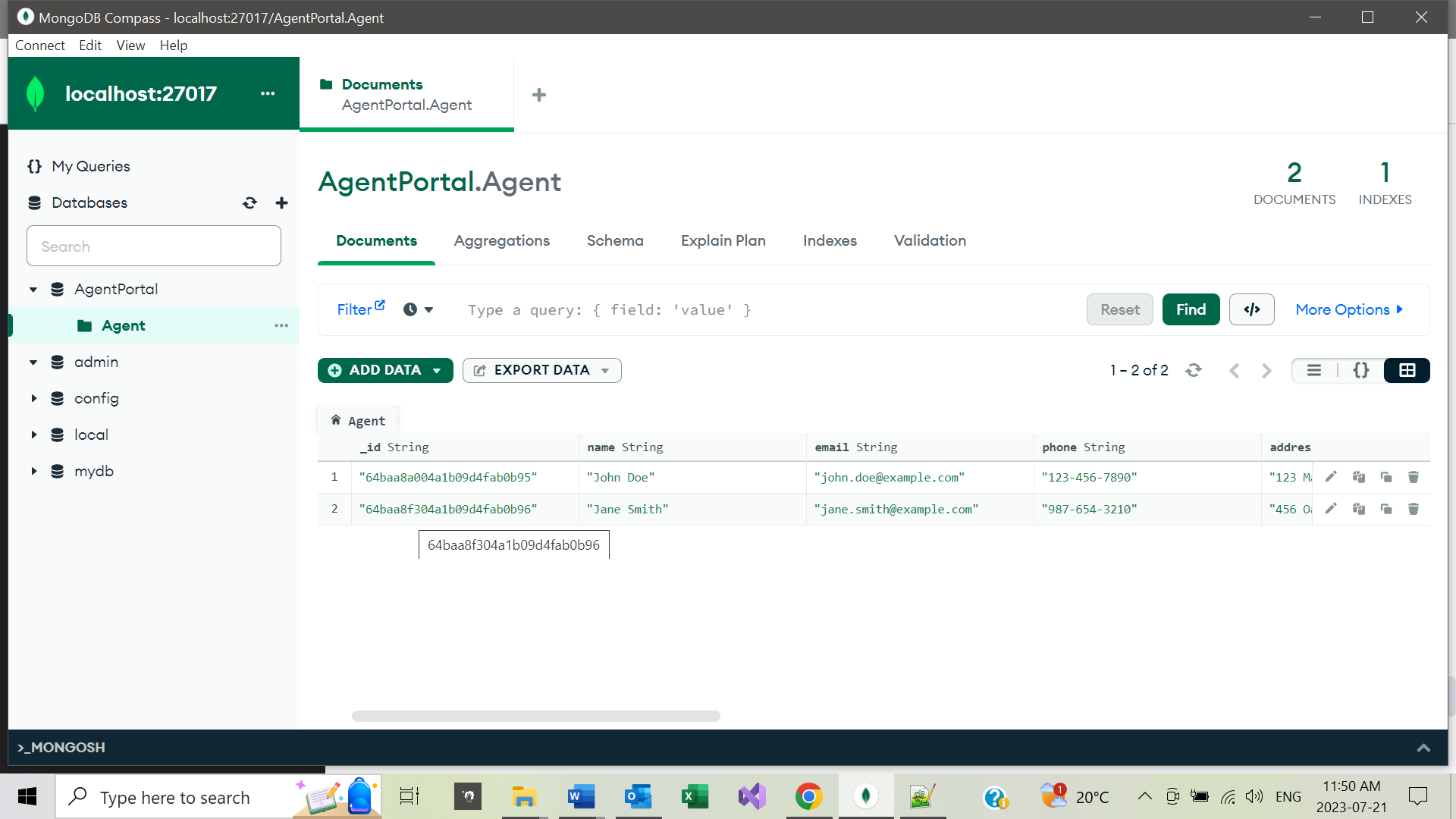
Description automatically generated

**Database and Collections:** Create a new database to store the key-value data. In this example, we will create a database called "AgentPortal" and two collections: "agent", "agency," & "QuoteDetails".

**Document Structure:** In MongoDB, documents are stored in BSON format (Binary JSON). For this key-value store, we will use the following document structure:

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Document Structure for Quotes:**

To efficiently store quote data, each insurance quote is represented as a document in MongoDB. The document structure is designed to capture essential details such as customer information, policy coverage options, pricing factors, and status (e.g., draft or submitted). Nested documents organize related data, ensuring a logical and readable representation.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Primary Key for Efficient Retrieval:**

MongoDB's primary key-valued data store architecture aligns well with the application's requirement to retrieve entire quote details efficiently. Each quote is assigned a unique primary key (quote ID), allowing for fast access and retrieval of specific quotes without complex joins or table scans.

**Indexing Strategy:**

Considering the potential volume of quote data, a thoughtful indexing strategy is adopted to optimize query performance. Compound indexes are created on fields commonly used in search and filter operations, such as customer name, policy type, and status. A unique index is also applied to the primary key (quote ID) to ensure its uniqueness and improve retrieval speed.

**Handling Draft Quotes:**

The application allows agents to create draft quotes before finalizing them. Draft quotes must be managed effectively, and a time-to-live (TTL) index is implemented to automatically expire draft quotes that remain in the system beyond a stipulated time. This DB ensures that draft quotes are completed on time and helps maintain database hygiene.

**Integration with Insurance Carriers:**

MongoDB's seamless integration with API services enables smooth communication between the application and insurance carriers' systems. When a quote is finalized, the relevant data is extracted from MongoDB and sent to the respective carriers through APIs for further processing.

**Data Security and Compliance:**

MongoDB's authentication mechanisms are utilized to control access to the database. Data encryption and role-based access control are also implemented to restrict unauthorized access to sensitive customer and policy information. To maintain and safeguard customer data using data protection regulations, such as GDPR and HIPAA.

**Conclusion:**

The implementation process for the AgentPortal database involved carefully considering the application's requirements and objectives. MongoDB was selected as the database solution due to its flexibility, scalability, and efficiency in handling JSON-like data. Design decisions were made to optimize data retrieval, handle draft quotes effectively, and ensure seamless integration with insurance carriers through APIs. By implementing MongoDB as the primary key-valued data store, the application is well-equipped to handle extensive quote information, provide a user-friendly GUI, and deliver an efficient and secure insurance quote creation process for insurance agents and agencies.

**References:**

**Citation -**[**https://www.mongodb.com/scale/nosql-database-implementation**](https://www.mongodb.com/scale/nosql-database-implementation)

**Citation -**[**https://www.mongodb.com/products/compass**](https://www.mongodb.com/products/compass)

**Citation -**[**https://www.mongodb.com/docs/manual/applications/data-models/**](https://www.mongodb.com/docs/manual/applications/data-models/)

**Citation - https://www.mongodb.com/**