ATYPON

Capstone Project Report

Decentralized Cluster-Based NoSQL DB System

Fares Qawasmi

September 2023

TABLE OF CONTENTS

INTRODUCTION

1.1 Background

With the continuous growth of data, conventional database management systems (DBMS) often struggle to keep up, particularly when it comes to scalability, fault tolerance, and data distribution. As a response to these challenges, NoSQL databases have emerged as a viable alternative. These databases are known for their ability to scale horizontally and allow for flexible schema design, among other features. One subcategory of NoSQL databases is the cluster-based NoSQL database, designed to provide high availability and scalability by distributing data across multiple nodes.

However, traditional cluster-based NoSQL databases often rely on a central manager node, which becomes a potential bottleneck or a single point of failure. This report presents a Decentralized Cluster-Based NoSQL Database System aimed to overcome these shortcomings by eliminating the need for a central manager node.

1.2 Objective

The purpose of this project is to simulate a Decentralized Cluster-Based NoSQL DB System in Java. The system is designed to fulfill the following requirements:

- ◊ Bootstrap a cluster of nodes with initial configuration information.
- ◊ Allow for user registration and assignment of nodes in a load-balanced manner.
- ◊ Implement a document-based database that employs JSON objects.
- ◊ Implement CRUD (Create, Read, Update, Delete) functionalities for databases and JSON documents.

- ◊ Develop a system for "node affinity" to handle write queries and their subsequent distribution.
- ◊ Ensure that data, schemas, and indexes are replicated across all nodes.
- ◊ Implement an "optimistic locking" scheme to deal with race conditions during write queries.

1.3 Scope

The MMS developed covers the following functionalities:

- ◊ Creation of new medicine records with relevant information like ID, Name, and Expiration Date.
- ◊ Updating of existing medicine records.
- ◊ Deletion of medicine records after due validation.
- ◊ Retrieval of medicine records based on specific properties.

The system is containerized using Docker, which encapsulates the distributed database. The database is split across multiple nodes, with a separate bootstrap node to the initialization of the nodes. The front-end is developed using HTML, CSS, and JavaScript, with the Bootstrap framework for responsive design.

This project strictly adheres to the guidelines of decentralization, meaning that no single node acts as a manager, and no shared filesystems are used. Each node in the cluster is represented as a virtual machine (VM) through Docker containers. The system also comes with its own API for executing CRUD operations on databases and JSON documents.

Methodology

Understanding the architecture of a Decentralized Cluster-Based NoSQL Database System requires diving deep into its core design principles and algorithms. This section aims to unpack the methodology employed in developing this system, with a particular focus on how decentralization and scalability are achieved.

Design Principles

Decentralization

I. Elimination of Central Management: Traditional cluster-based NoSQL databases often incorporate a central management node, which serves as a bottleneck and a single point of failure. To avoid this, the system's design eliminates the central manager, opting for a fully decentralized model.

II. Peer-to-Peer Communication: Nodes in the system communicate directly with each other. This allows for more resilient and flexible data management, as there is no centralized node that could become a single point of failure.

III. Bootstrapping Node: Although the system is decentralized, we introduce a bootstrapping node solely for the purpose of initial configuration and user node assignment. Importantly, this node does not manage the cluster during its operation.

Scalability

◊ **Horizontal Scaling: The system is designed to scale horizontally, meaning additional nodes can be added to manage increased data loads.**

◊ **Load Balancing: Each new user is mapped to a node in a load-balanced manner to ensure that no single node becomes a bottleneck.**

◊ **Data Replication: To ensure data availability and fault-tolerance, data, schemas, and indexes are replicated across all nodes.**

System Architecture

3.1 Docker Compose

The Docker Compose configuration offers a birds-eye view of the service architecture, outlining the process of service instantiation and inter-service communication.

Purpose

The Docker Compose configuration exists to define and manage the multi-container Docker applications for our decentralized system. It details the interplay of nodes, their communication channels, and their dependencies.

Main Components

◊ Bootstrap Service: Responsible for initializing the cluster, this service is vital for disseminating configuration to other nodes.

i. Waits for node0, node1, and node2 to start using the depends_on directive.

ii. Builds from the ./bootstrapnode directory.

iii. Communicates externally via port 8080.

iv. Interacts within the network using the node-network.

◊ Node Services (node0, node1, node2): These are the database nodes, responsible for data storage and retrieval.

i. Each node is constructed from the ./database directory.
ii. Distinct naming conventions and ports facilitate node differentiation.
iii. All nodes are interconnected via the node-network.

◊ Node-network: Serves as the communication bridge, leveraging Docker's bridge driver.

i. Ensures seamless inter-service communication.
ii. Upholds the principles of decentralization and scalability.

Decentralization Elements

This configuration embodies decentralization through its structure. While the bootstrap service plays a pivotal role in initialization, it doesn’t govern node operations. Each node operates autonomously, highlighting the decentralized nature.

Inter-Node Communication

The node-network provides the backbone for inter-node dialogue. Given the Docker bridge driver's usage, nodes can easily communicate amongst themselves, bolstering the system's resilience.

Port Management

Each service specifies its communication ports, ensuring clarity in inter-service and external interactions. This distinct port assignment aids in effectively routing requests and inter-node communications.

Initialization Sequence

The depends_on directive ensures that the bootstrap service waits for all database nodes to start. This sequential start-up ensures that, by the time the bootstrapping node is operational, it has a full set of nodes to communicate with.

Data Handling

While this Docker Compose file doesn't explicitly touch on data handling, the overarching system emphasizes data availability and fault-tolerance. Each database node would typically house a replica of the system's NoSQL database, ensuring data redundancy and resilience against potential node failures.

3.2 Bootstrapping Node

The bootstrapping process is a vital component of this decentralized cluster-based system, and the code outlines the logical steps in this process. Here's a breakdown of the essential functionalities:

Purpose

The bootstrapping node's primary role is to disseminate initial configuration information and ensure the fair distribution of user requests (in this case, pharmacies) among available nodes using load balancing.

Main Components

◊ *BootstrappingController*: This component handles the registration of pharmacies (users).

    i. Checks if a pharmacy is already registered.
    ii. Generates a unique token for new pharmacies.
    iii. Assigns a node to the new pharmacy using the LoadBalancer.
    iv. Saves the pharmacy's data, including the assigned node, into a file.

◊ *ClusterInitializer*: Initializes a set of nodes available for load balancing upon system start-up. The node data is stored in a map, facilitating easy retrieval.

◊ *LoadBalancer*: Uses a straightforward round-robin algorithm to distribute incoming user requests (pharmacies) across nodes.

◊ *FileUtil*: Manages the reading and writing of user data (pharmacies) into a JSON file.

Load Balancing Mechanism

The load balancing is achieved using a round-robin approach. When a new pharmacy is registered, it's assigned to a node in a cyclical manner. This ensures that no single node is overburdened with requests, as each node gets a turn to handle a request.

Decentralization Elements

The absence of a single managing node and the use of multiple nodes initialized at start-up signify the decentralized nature of the system. The nodes act independently, and there isn't a centralized entity directing all operations.

Data Persistence

Data persistence for pharmacies is achieved using a JSON file. The system uses utility functions to read from and write to this file, ensuring that data about registered pharmacies and their respective nodes is maintained.

Redirection Mechanism

Once a pharmacy is successfully registered, the system redirects the client to their assigned node's CRUD interface (/crud.html), ensuring that subsequent operations are routed through the appropriate node.

The distributed NoSQL document-oriented database serves as the backbone for handling data within our decentralized system. Each node within the system has a copy of this database, ensuring data availability, fault tolerance, and resilience against node failures.

3.3 Database Implementation

Core Components

Database Controller

Serving as the primary interface for external interactions with the database, this controller manages a variety of responsibilities, ranging from handling CRUD operations to synchronizing cache and indexes with other nodes.

Endpoints (simplified):

- ◊ **CRUD operations such as document creation, retrieval, update, and deletion.**
- ◊ **Cache and index synchronization.**
- ◊ **Redirect updates based on node affinity.**

Document Service

The DocumentService underpins the functionality provided by the DatabaseController. It interfaces directly with storage mechanisms, caches, and indexes to ensure efficient and consistent data operations.

Key Functions:

- ◊ **Interface with persistent storage (DatabaseStorage).**
- ◊ **Handle document caching and indexing.**
- ◊ **Maintain document versioning.**

Database Storage

This class primarily focuses on data persistency. It's responsible for the CRUD operations – Create, Read, Update, and Delete – of the JSON-formatted documents, basically a derivative of the DAO design pattern.

- ◊ **Initialization: On instantiation, it checks if a storage directory exists. If not, it creates one.**
- ◊ **CRUD Operations:**
- ◊ **Get: Retrieves a document using its unique ID.**
- ◊ **Save: Persists a document on disk, saving additional metadata like affinity node and node name.**
- ◊ **Delete: Removes a document using its unique ID.**

Each document is stored as a separate JSON object named with its unique ID, ensuring direct access.

Index Service

A pivotal component for efficient querying, the *IndexService* class maintains an in-memory index for properties of documents, this section is spotlighted upon later.

This in-memory index vastly improves search performance, ensuring rapid retrieval of documents based on property-value queries.

Node Communication Service

To uphold the integrity of the decentralized system, *NodeCommunicationService* manages synchronization across nodes. It achieves this by communicating with other nodes whenever local data changes.

As their names infer:

- ◊ **synchronizeDocumentCreation**: Informs other nodes of a new document's creation.
- ◊ **synchronizeDocumentUpdate**: Communicates document updates to other nodes.
- ◊ **synchronizeDocumentDeletion**: Notifies other nodes of a document's deletion.
- ◊ **synchronizeIndex**: Ensures index updates on a property value are reflected across all nodes.
- ◊ **synchronizeIndexDeletion**: Communicates the removal of an index entry to other nodes.
- ◊ **synchronizeCache**: Used for synchronizing cache across nodes.

This service makes use of Spring's RestTemplate to communicate with other nodes. Any change on one node results in synchronized updates on other nodes, maintaining data and index integrity across the entire system.

Design Choices

i. Decentralization: the database mirrors our overarching system philosophy by decentralizing data. Each node has its local database, and operations are always synchronized across nodes to maintain consistency.

ii. Cache and Index Synchronization: Given the decentralized nature, it's imperative that caches and indexes are kept in sync across nodes. We achieve this through dedicated synchronization endpoints within the DatabaseController.

iii. Data Consistency: Concurrent updates are managed through document versioning, ensuring that data remains consistent across the network, even when simultaneous changes are made.

iv. Affinity-based Redirection: To improve efficiency, the database redirects updates to the affinity node (the node with the most up-to-date version of a document), minimizing data transfer across the network and respecting document ownership.

Utilized Data Structures

In modern computer software, selecting the right data structure is crucial for optimizing the performance and efficiency of operations. This section will examine the different data structures we used in the System and the rationale behind the choices made.

4.1 Indexing

HashMap

Usage in the System: Primarily used in the *IndexService* to create a fast lookup for property-value associations.

Rationale: Constant Time Complexity: The average case for retrieving, inserting, and deleting from a HashMap is O(1). This ensures rapid operations, especially for the indexing tasks.

Simplicity: Compared to more complex structures like trees, using HashMap is more straightforward, especially when we consider the internal implementation and operations.

Compatibility: The system heavily depends on the key-value paradigm, making HashMap's naturally compatible.

B-Tree (Attempted)

Usage in the System: Initially, it was attempted to use a B-tree for the IndexService.

Rationale & Learnings:

Balanced Tree Structure: B-trees, due to their self-balancing nature, ensure that data remains accessible in logarithmic time, making them an attractive option for indexing large amounts of data.

Efficient for Disk Storage: B-trees are particularly efficient when data exceeds memory limits and spills over to disk storage, due to the way B-trees reduce the number of disk I/O operations.

However, during the development phase, several challenges and learnings emerged:

Not Optimal for the Use Case: While B-trees are powerful, it added an unnecessary layer of complexity for this specific situation. The data and access patterns did not necessitate the use of B-trees, and the added overhead in terms of both development time and runtime operations did not provide a tangible benefit.

Hashmap Superiority: For this particular use case, it was found that the HashMap's constant-time operations offered better performance and responsiveness. The simplicity of the hashmap also meant that the system was more maintainable and easier to understand.

As a result of these challenges, the choice was made to pivot using a HashMap for the IndexService.

4.2 Database Storage

Files & Directories (File System)

Usage in the System: This is evident in the DatabaseStorage service, where each document is saved as a .json file in the STORAGE_DIR directory.

Rationale:

- ◊ **Organized Structure**: Using directories helps to keep the saved documents organized and easily retrievable.
- ◊ **Straightforward**: *Leveraging* the filesystem is a direct method of storage without requiring additional databases or third-party storage systems.

In-memory Cache (Map-based Structure)

Usage in the System: Utilized in the DocumentService class, where a cache of type Cache<String, Document> is used to store and retrieve documents.

Rationale:

- ◊ Fast Access: In-memory caches offer much faster access times than disk-based storage, helping to quickly retrieve frequently accessed documents.
- ◊ Reduced I/O Operations: By storing frequently accessed data in-memory, the system reduces the number of read/write operations to the file system or database, thereby enhancing overall system performance.

HashMap (Bootstrapping Node)

Usage in the System:

- ◊ In the ClusterInitializer class where a nodeMap of type Map<Integer, Node> is employed to map node ports to node details.
- ◊ Also seen in the LoadBalancer class where the clusterInitializer's nodes are fetched and operations are performed.

Rationale:

- ◊ **Constant Time Complexity**: Provides O(1) average complexity for search operations, making it optimal for rapid lookups.
- ◊ **Key-Value Storage**: Fits scenarios where data is associated with unique identifiers (in this case node ports).
- ◊ **Flexible**: Can be easily resized or expanded, making it useful for dynamic datasets.

ArrayList

Usage in the System: Employed in the LoadBalancer service to convert node values from the nodeMap into a list format.

Rationale:

Dynamic Size: Allows for dynamic resizing, which is beneficial when the exact size is unknown in advance.

Indexed Access: Provides constant-time positional access, O(1), which is ideal for sequential data access patterns, as seen in the round-robin load balancing logic.

Clean Code Principles

(Uncle Bob)

◊ **Meaningful Names**: Methods such as get, save, delete in *DatabaseStorage* and createDocumentEntry, retrieveDocumentEntry, updateDocumentEntry in *DocumentService* are self-explanatory. All methods are self-explanatory and align with the service's function addToIndex, removeFromIndex, getIdsByProperty, updateIndex, and removeIdFromAllIndexes.

◊ **Functions Do One Thing (SRP)**: Each function performs a specific task, such as saving or retrieving documents, thereby adhering to the Single Responsibility Principle at the method level.

◊ **Error Handling**: Error handling is consistent, using logging and exceptions. For instance, catch blocks in *DatabaseStorage* handle IOExceptions, informing the user about the failure in operations.

◊ **Separation of Concerns**: The *DatabaseController* class clearly functions as a controller, handling HTTP methods and delegating the business logic to services like *DocumentService* and *IndexService*. This separation ensures that modifications in the business logic don't necessarily affect the routing logic.

◊ **DRY (Don't Repeat Yourself):** The DatabaseStorage class has methods like get, save, and delete that each handle distinct tasks without repetition. If the logic for file storage needs to be modified, it only must be changed in one place. For instance, the logic to resolve file paths is concentrated in the getFilePath method, ensuring consistency and non-repetition.

◊ **Avoiding Side Effects**: Methods such as getDocument simply retrieve data and respond; they don't cause side effects or change states unexpectedly.

◊ **Keep It Simple, Stupid (KISS):** The NodeCommunicationService is a good example. Each of its methods like synchronizeDocumentCreation, synchronizeDocumentUpdate, etc., communicates with other nodes for a specific operation. There's no complex logic intertwined, making it easier to understand and maintain.

◊ **Function Arguments**: Most methods in classes like DatabaseStorage and DocumentService have clear arguments that hint at their use. For instance, DatabaseStorage.get(String id) makes it clear that you're getting a document based on its ID. This clarity improves the code's readability and reduces the potential for errors.

◊ **Comments Only When Necessary**: The code largely speaks for itself. There aren't excessive comments, but where they exist, they're meaningful. For instance, the comment "Cache it for 60 seconds" in the retrieveDocumentEntry method of DocumentService clarifies the purpose of the hardcoded time value.

"Effective Java" Items

(Joshua Bloch)

Item 18 Favor Composition Over Inheritance

The design principle of preferring composition over inheritance encourages more flexibility in code structuring. In the codebase:

i. Flexibility and Dynamism: By using composition, the code can dynamically inherit behavior by embedding objects, as seen where *DocumentService* composes *DatabaseStorage*, *IndexService*, and a cache.

ii. Simplified Maintenance: Changes in the superclasses won't inadvertently break the subclasses, avoiding tight coupling and unintended side-effects.

Item 73 Throw Exceptions Appropriate to the Abstraction

Throwing precise exceptions helps in conveying the exact nature of an error, aiding debugging, and user feedback.

i. Clear Communication: Custom exceptions such as DocumentNotFoundException and VersionMismatchException allow for conveying specific problems in the application. This is crucial for front-end users and developers to understand the issue without diving deep into the abstraction.

ii. Enhanced Robustness: Using meaningful exceptions means that potential issues are less likely to be ignored or misunderstood, and more likely to be appropriately handled.

Item 59 Know and use the libraries

Familiarity and consistency with libraries can significantly speed up development by reusing tried-and-tested functions.

i. Standardization: Leveraging widely accepted libraries like RestTemplate and LoggerFactory ensures that the approach aligns with community standards. It also ensures that newcomers to the code can quickly grasp its structure.

ii. Efficiency: Using these libraries helps avoid "reinventing the wheel", instead utilizing optimized solutions for common tasks, resulting in faster development cycles and performance gains.

Item 17 – Minimize mutability

Immutable objects are simpler and more reliable. They can't be changed after they're created, ensuring no unintended modifications.

i. Constants: The logger and restTemplate members in the DatabaseController class are effectively immutable. Though not marked as final, their references are never changed after instantiation.

ii. Method Variables: Method-local variables, such as those seen in createDocument (id, doc, etc.), are inherently immutable in the sense that they can't be modified outside their method scope.

Item 5 Prefer dependency injection to hardwiring resources

Dependency Injection is a technique where objects are provided with their dependencies instead of creating them internally, promoting the Inversion of Control (IoC) principle.

i. Decoupling for Modularity: The codebase uses Spring's @Autowired annotation for dependency injection, ensuring components are loosely coupled. This separation allows for easier testing and maintenance.

ii. Increased Flexibility: With dependency injection, various implementations of a service can be easily swapped without changing the client class, fostering adaptability.

Item 28 Prefer lists to arrays

Lists, with their dynamic nature and stronger type-checking, often offer more utility than arrays.

i. Endpoint Return Types: Methods like getDocumentsByIndexedProperty return a ResponseEntity<List<String>>, using lists to encapsulate return data. This aligns with the principle of preferring lists over arrays.

ii. Type Checking with Lists: Using lists ensures that only the right type of objects (e.g., strings in the above example) can be added to the list, providing type safety.

Item 68 Adhere to generally accepted naming conventions

Conforming to naming conventions ensures clarity and consistency, making the codebase easier to understand and maintain.

i. Class Naming: DatabaseController is a clear and appropriately named class. The name communicates that this class is a controller, handling database operations, following the convention of suffixing with Controller for classes responsible for managing HTTP request mappings in a Spring application.

ii. Method Naming: Method names in the class are descriptive and denote actions, such as updateCache, getDocumentsByIndexedProperty, and createDocument. This is in line with the convention that method names should be verbs or verb phrases describing the action they perform.

iii. Variable Naming:

- Dependency Variables: Names like documentService, indexService, and nodeCommunicationService are self-explanatory and denote the services they refer to.

- Local Variables: Names like id, property, value, and documentIds are aptly named, giving an immediate understanding of their purpose.

SOLID Principles

Single Responsibility Principle (SRP)

Application: Each class in the codebase embodies a unique responsibility, ensuring clean separation of concerns, some examples of this:

i. ***DatabaseStorage***: Solely focused on storage-related functionalities, ensuring that operations related to data storage are encapsulated here.

ii. ***DocumentService***: Acts as a middleman that manages document-related operations, shielding the controller from direct interactions with the underlying data structures.

iii. ***DatabaseController***: Entrusted with the task of managing HTTP requests. The beauty of this design is that it remains detached from the intricate details of the business logic, effectively delegating those duties to the appropriate services.

Open/Closed Principle

Application: The power of Spring's dependency injection is manifested in the code, promoting extensibility.

i. **DatabaseController's Flexibility**: The structure permits adding new functionalities or endpoints without disrupting the existing ecosystem. This means if the need arises, new methods can be seamlessly incorporated, or the class can be extended.

ii. **Service Expansion**: Whether it's introducing new operations or tasks related to synchronization, the services are primed for effortless extensions.

Liskov Substitution Principle

The code does not contain subclassing, ensuring that this principle doesn't get violated.

Interface Segregation Principle

Application: The code champions the essence of focused interfaces, steering clear of unwanted bloat.

i. **Focused Responsibility**: Classes such as DatabaseController are meticulously crafted to address specific responsibilities, devoid of extraneous functions.

ii. **Interface Clarity**: There's an apparent absence of cumbersome interfaces, which ensures that implementing classes are unencumbered by undesired dependencies. Such a design is a boon for future expansion or refactoring endeavors.

Dependency Inversion Principle

Application: The architecture gracefully flips the conventional dependency structure, emphasizing flexibility.

i. **Strategic Design**: Classes of higher echelons, like DocumentService, are strategically crafted to lean on abstractions, such as DatabaseStorage.

ii. **Flexible Bonding**: The DatabaseController, with its detached design, isn't firmly anchored to a particular storage modality. This visionary approach signifies that future incorporations or replacements of storage mechanisms would require nominal amendments to the existing controller and service layers.

DevOps Practices

Logging

Throughout the services (DatabaseStorage, DocumentService, NodeCommunicationService) the *DatabaseController* and even the *BootstrappingController*, we see the use of Logger which logs info, warnings, and errors.

Proper logging is a cornerstone of observability and will help in:

i. Debugging issues in development and production.
ii. Monitoring the health of an application.
iii. Understanding user and system behavior.

Configuration Injection

In classes like *DocumentService* and *NodeCommunicationService*, configurations (like node.port or nodes) are injected at runtime using the @Value annotation.

Configuration Injection is essential for**:**

i. Flexibility: Allows the application to be environment-agnostic and configurations to be changed without altering the code.
ii. Security: Sensitive information can be kept out of the codebase and injected only when necessary.

Dependency Injection

The use of @Autowired and Spring's @Component and @Service annotations show dependency injection, which facilitates:

i. Better code modularity.

ii. Easier unit testing by mocking dependencies.

Modularity and Separation of Concerns

Services like DatabaseStorage, DocumentService, NodeCommunicationService, and IndexService have been broken down to handle specific functionalities.

Modularity is essential for**:**

i. Maintainability: Easier to update or fix specific components without affecting the entire system.
ii. Scalability: Individual components can be scaled independently based on the load they experience.

Exception Handling

Raising custom exceptions (DocumentNotFoundException, VersionMismatchException) in the *DocumentService*.

Custom exceptions can help:

i. Graceful Error Recovery**:** Ensures the system doesn't crash abruptly but handles errors gracefully.
ii. User Feedback**:** Communicates more accurate information about the issue to the user or other system components.

Data Integrity and Versioning

In *DocumentService*, document versions are tracked to ensure that updates and modifications are made to the right version of the document.

Data Integrity and Versioning is key for:

i. Avoiding Conflicts: Prevents overriding of concurrent changes made by other nodes or users.
ii. Traceability: Provides a mechanism to track the history and evolution of data.

Caching

Using caching in *DocumentService* to reduce data retrieval time.

Caching is crucial for:

i. Improving performance.
ii. Reducing load on the storage system.

Synchronization in Distributed Systems

NodeCommunicationService ensures that document changes are propagated to all nodes. This highlights the importance of data synchronization in distributed systems.

Distributed consistency will:

i. Ensure all nodes have the most recent data.
ii. Prevent data loss and inconsistency.

Indexing

The *IndexService* manages indexing, crucial for:

i. Faster data retrieval.
ii. Efficient storage and organization of data.

Infrastructure Interaction

The DatabaseController class interacts with other services using the RestTemplate, showing a microservices architecture. Such an architecture aligns with DevOps, enabling independent deployment and scaling of services.

Conclusion & Future Work

9.1 Conclusion

The developed system embraces a decentralized model that redistributes authority, eliminating single points of failure, and optimizing data availability and reliability. By dividing responsibilities across bootstrapping nodes and database implementations, the system efficiently balances loads, ensuring scalability and performance. The designed architecture appropriately utilizes data persistence, in-memory indexing, and node synchronization mechanisms to bolster a robust decentralized environment.

Findings

Decentralization: Through the bootstrapping node and decentralized database design, the system successfully distributes tasks, mitigating risks associated with centralized systems.

Load Balancing: The round-robin approach incorporated ensures even distribution of tasks, preventing node overloads and fostering faster response times.

Data Integrity and Consistency: The use of document versioning, synchronization methods, and affinity-based redirection ensures data remains consistent and up to date across nodes.

Efficiency: The system benefits from the in-memory indexing technique, promoting faster data retrieval while reducing the need for exhaustive searches.

9.2 Future Work

Enhanced Load Balancing: Future implementations can explore more sophisticated load balancing techniques, perhaps incorporating machine learning algorithms to predict load and redistribute tasks dynamically.

Fault Tolerance: Delve deeper into mechanisms to detect node failures promptly and reassign tasks, ensuring minimal service interruptions.

Database Sharding: Investigate sharding techniques to further decentralize data, making it possible to distribute data fragments across nodes for even better performance and scalability. Database sharding refers to a method where data is partitioned, and these partitions (or "shards") are stored across separate database servers or clusters. The goal of sharding is to ensure that large datasets can be managed more efficiently by dividing and distributing them across multiple servers, rather than relying on a single machine.

Security Enhancements: Implement advanced encryption techniques and secure channels for node-to-node communication to bolster system security.

Logging and Monitoring: Integrate advanced logging and monitoring tools to facilitate system debugging, performance tuning, and preemptive problem detection.

Multi-level Caching: Introduce hierarchical caching techniques that utilize both in-memory and disk-based caches, striking a balance between performance and storage efficiency.

Demo Application

All the mentioned operations here can be proved by the logs provided at the end!

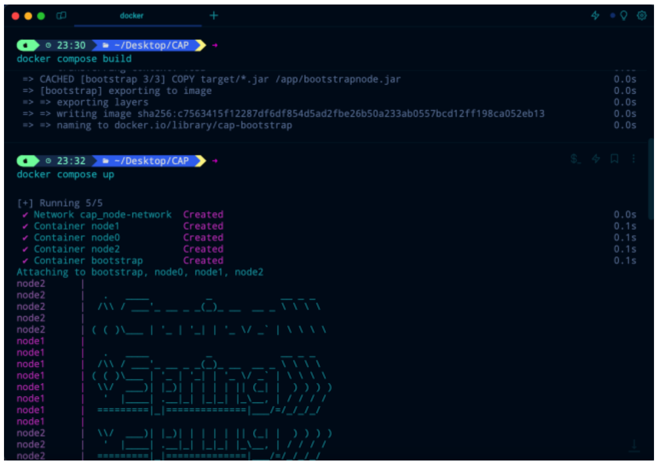
Scenario: Distributed Data Management System for Pharmaceuticals

Setting: A healthcare organization has several distributed nodes that store medical information, particularly about medicines. These nodes need to synchronize information and ensure that data is consistent, accurate, and up to date across all nodes.

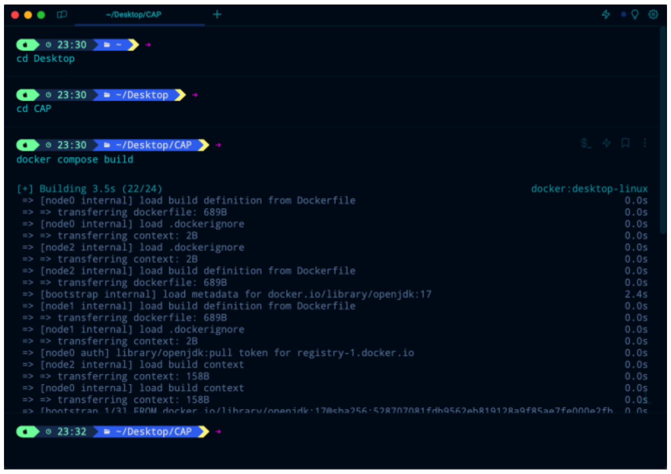
User Guide

To start up the application, we need to run docker, then navigate from the terminal to where this file is stored, for me it will be the "CAP" file, where my application and docker files are in.

Now that we are in the application file, we can build the application using "*docker compose build*." After it's done building, you can go ahead and run the command "*docker compose up*" and let docker handle the rest!



```
23:30  ~/Desktop/CAP
docker compose build
 => CACHED [bootstrap 3/3] COPY target/*.jar /app/bootstrapnode.jar                                      0.0s
 => [bootstrap] exporting to image                                                                       0.0s
 => => exporting layers                                                                                  0.0s
 => => writing image sha256:c7563415f12287df6df854d5ad2fbe26b50a233ab0557bcd12ff198ca052eb13             0.0s
 => => naming to docker.io/library/cap-bootstrap                                                         0.0s

23:32  ~/Desktop/CAP
docker compose up

[+] Running 5/5
 ✔ Network cap_node-network  Created                                                                     0.0s
 ✔ Container node1           Created                                                                     0.1s
 ✔ Container node0           Created                                                                     0.1s
 ✔ Container node2           Created                                                                     0.1s
 ✔ Container bootstrap       Created                                                                     0.1s
Attaching to bootstrap, node0, node1, node2
node2      |
node2      |   .   ____          _            __ _ _
node2      |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
node2      |
node2      | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
node1      |
node1      |   .   ____          _            __ _ _
node1      |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
node1      | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
node1      |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
node1      |   '  |____| .__|_| |_|_| |_\__, | / / / /
node1      |  =========|_|==============|___/=/_/_/_/
node1      |
node2      |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
node2      |   '  |____| .__|_| |_|_| |_\__, | / / / /
node2      |  =========|_|==============|___/=/_/_/_/
```



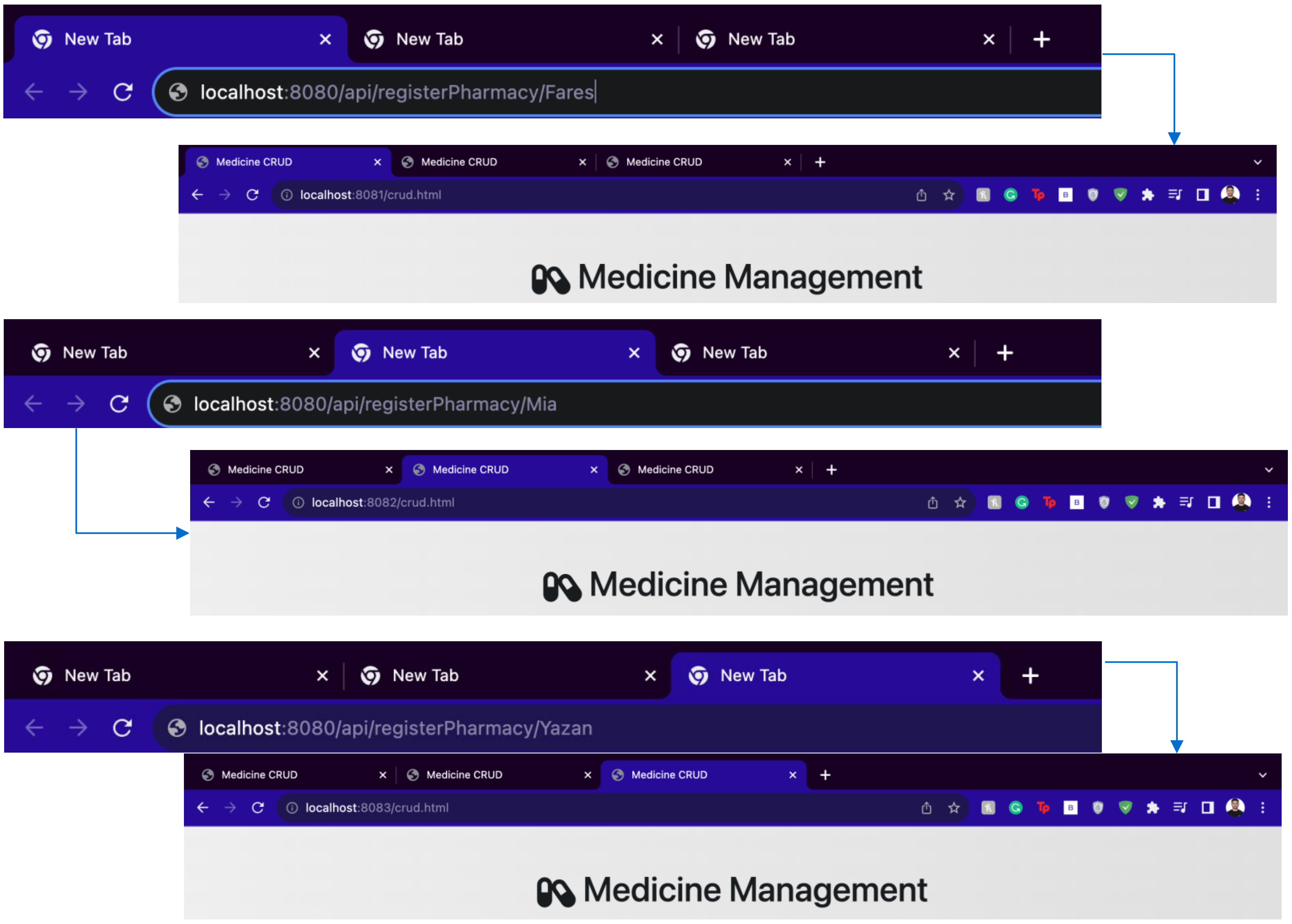
```
23:30  ~
cd Desktop

23:30  ~/Desktop
cd CAP

23:30  ~/Desktop/CAP
docker compose build

[+] Building 3.5s (22/24)                                                              docker:desktop-linux
 => [node0 internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 689B                                                                   0.0s
 => [node0 internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                        0.0s
 => [node2 internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                        0.0s
 => [node2 internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 689B                                                                   0.0s
 => [bootstrap internal] load metadata for docker.io/library/openjdk:17                                2.4s
 => [node1 internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 689B                                                                   0.0s
 => [node1 internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                        0.0s
 => [node0 auth] library/openjdk:pull token for registry-1.docker.io                                   0.0s
 => [node2 internal] load build context                                                                0.0s
 => => transferring context: 158B                                                                      0.0s
 => [node0 internal] load build context                                                                0.0s
 => => transferring context: 158B                                                                      0.0s
 => [bootstrap 1/3] FROM docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fb  0.0s

23:32  ~/Desktop/CAP
```

There you go, the bootstrapping node is running on port 8080, where you can register users using this link: localhost:8080/api/registerPharmacy/whatEverYouWantAsYourUserName

Initialization of the System

- When the system starts, each node gets its configuration from the bootstrapping node. This bootstrapping node provides the node its name and the port on which it will communicate, and most importantly, which pharmacy (user) is assigned to it.

- A pharmacy logs in, communicates with the bootstrap node, and gets assigned to a node, say node2, NODE_NAME=node2 and NODE_PORT=8083.



localhost:8080/api/registerPharmacy/Fares

localhost:8081/crud.html

Medicine Management

localhost:8080/api/registerPharmacy/Mia

localhost:8082/crud.html

Medicine Management

localhost:8080/api/registerPharmacy/Yazan

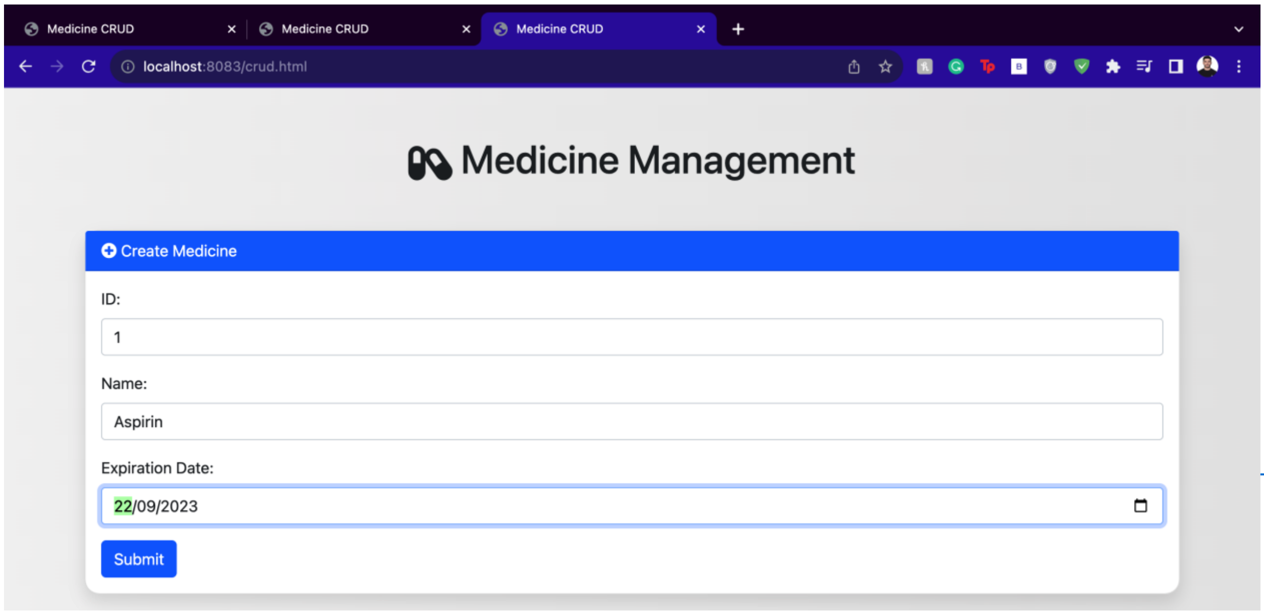
localhost:8083/crud.html

Medicine Management

Creation of a Medicine Record

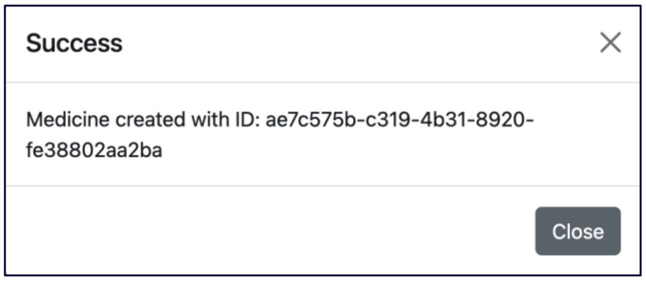
- A pharmacy wants to add a new medicine to the database.

- node2 receives the medicine information, validates the schema to ensure consistency, and then adds it to its database.



ID:
1
Name:
Aspirin
Expiration Date:
22/09/2023

- After successfully storing the information, node2 communicates with other nodes (node0 and node1) to synchronize the creation. This is crucial to ensure that if a future request comes to another node for this medicine, the node already has the information.



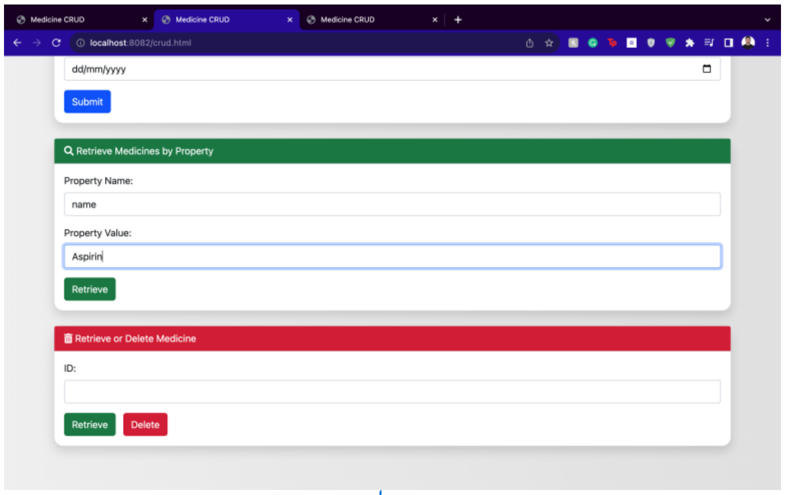
Success

Medicine created with ID: ae7c575b-c319-4b31-8920-fe38802aa2ba

Retrieving a Medicine Record

- A different pharmacy wants to retrieve information about a specific medicine.

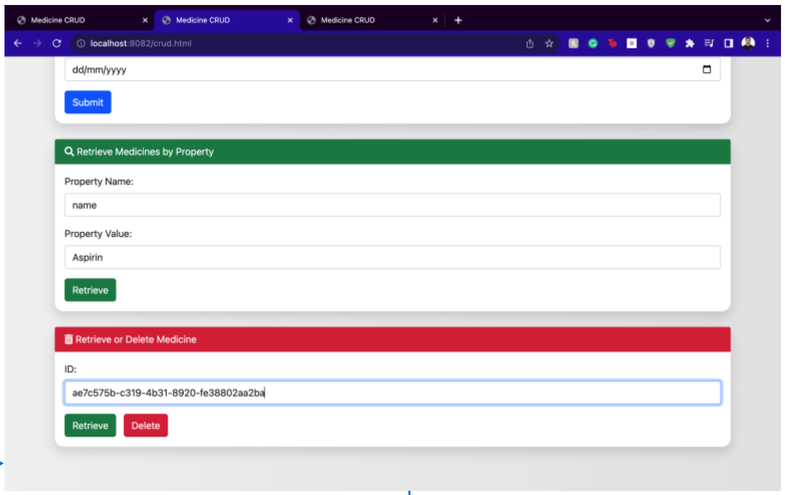
- node2 checks its database. If the information is there (because of the earlier synchronization), it returns the data to the pharmacy. For this example, we will try to retrieve the medicine from a different node to test data consistency. To do that, we need the Document ID, which is a unique ID generated when a document is created different than the medicine ID, we can retrieve that ID using the property retrieval section:



Property Name: name

Property Value: Aspirin

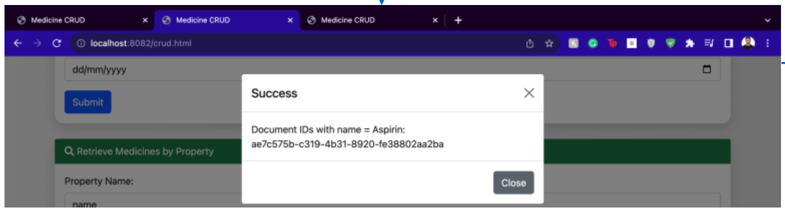
ID:



Property Name: name

Property Value: Aspirin

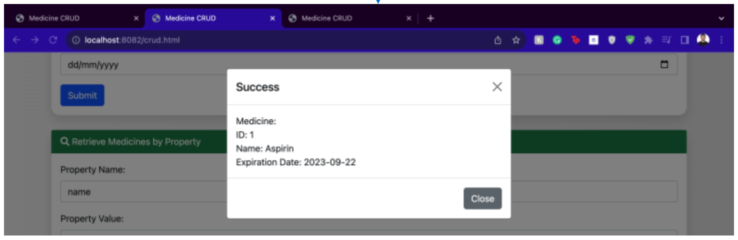
ID: ae7c575b-c319-4b31-8920-fe38802aa2ba



Property Name: name

Success

Document IDs with name = Aspirin:
ae7c575b-c319-4b31-8920-fe38802aa2ba



Property Name: name

Property Value:

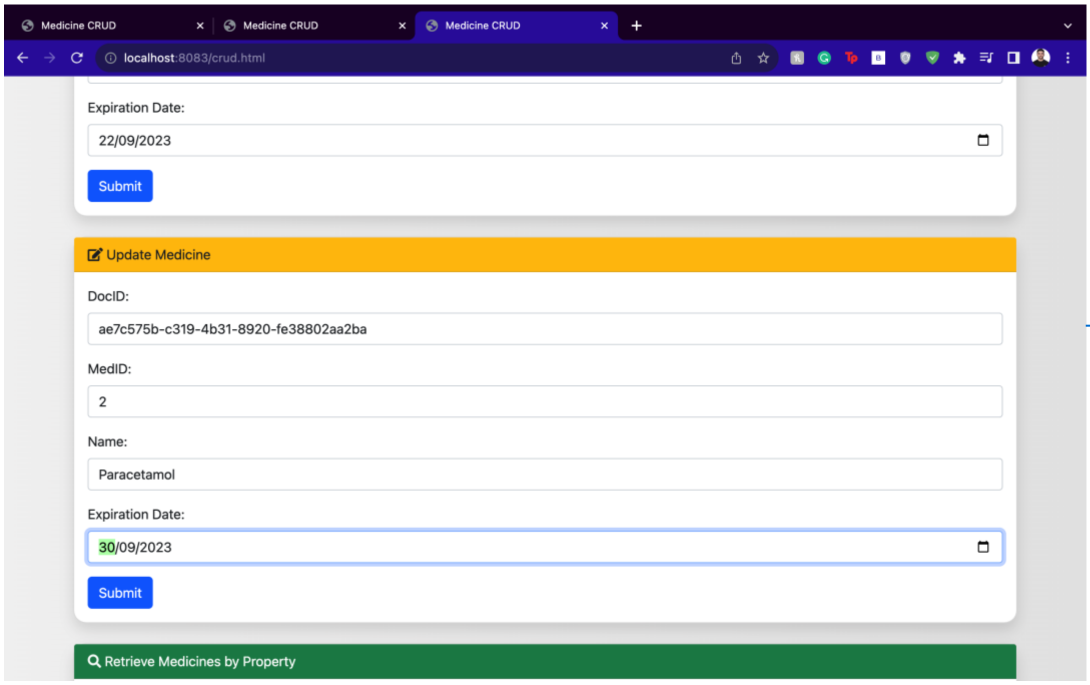
Success

Medicine:
ID: 1
Name: Aspirin
Expiration Date: 2023-09-22

Updating a Medicine Record

- A pharmacy realizes that they made an error when entering the expiration date of a medicine.

- If node2 is the affinity node for that specific medicine record (meaning it's the primary holder of that record), it directly updates its database and then communicates with other nodes to synchronize the update.



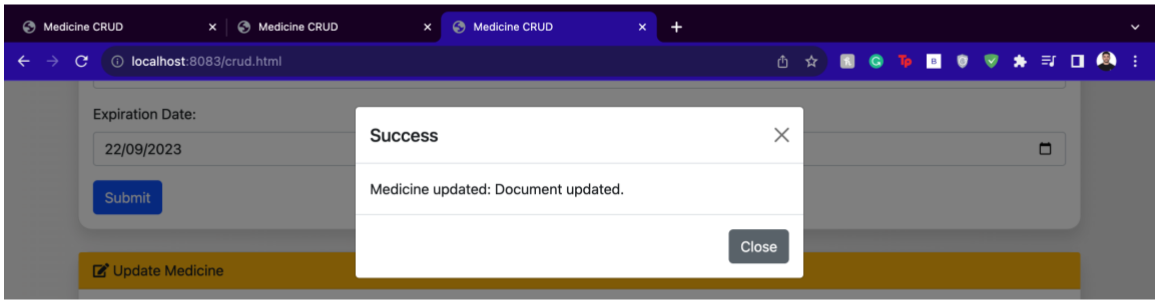
Expiration Date:
22/09/2023

DocID:
ae7c575b-c319-4b31-8920-fe38802aa2ba

MedID:
2

Name:
Paracetamol

Expiration Date:
30/09/2023



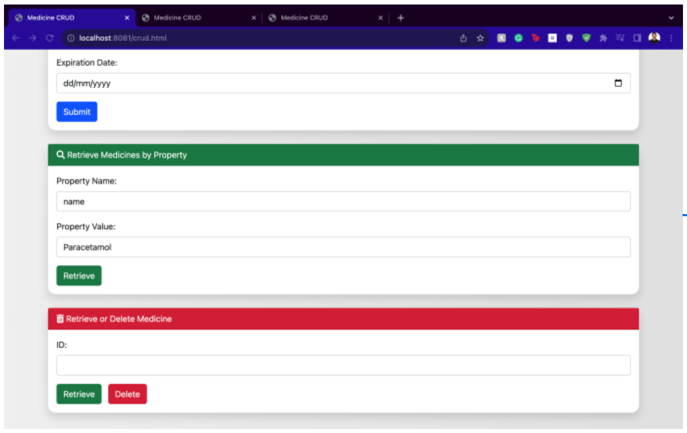
Expiration Date:
22/09/2023

Success

Medicine updated: Document updated.

- If node2 is not the affinity node, it realizes the data should be updated on the affinity node. It sends a request to the correct node (the one mentioned in the 'affinityNode' field of the document) to update the medicine record. After the affinity node updates the record, it syncs the changes with all other nodes.

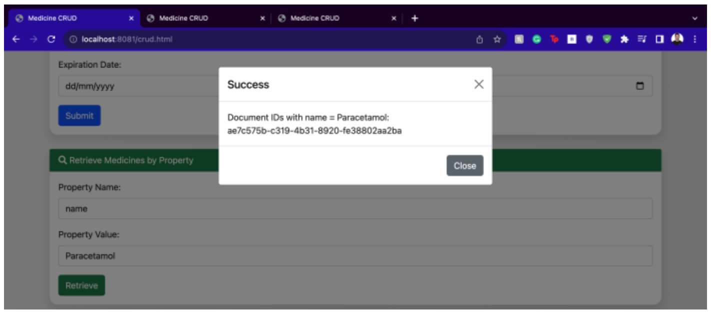
- Let's say we go to node0 on port 8081 and try to update, and to do that we need the Document ID again, this is also a chance for us to check if the update request was synchronized across the nodes.


localhost:8081/crud.html

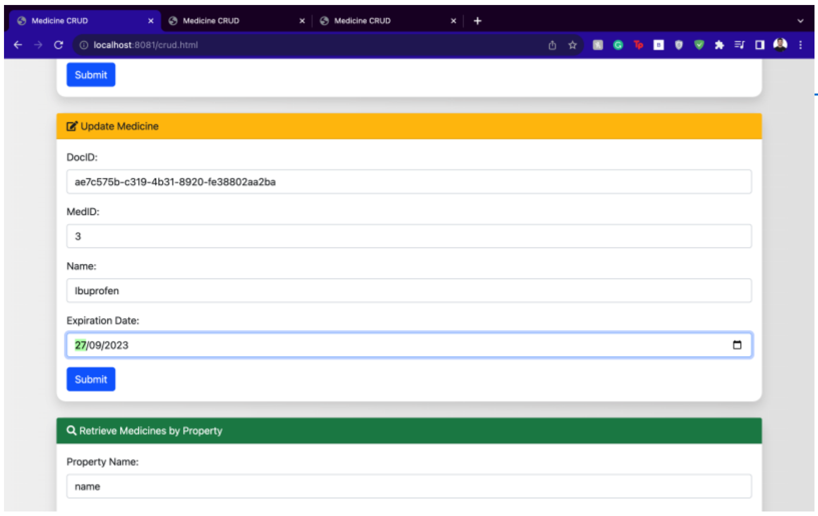
Expiration Date:
dd/mm/yyyy

Retrieve Medicines by Property
Property Name:
name
Property Value:
Paracetamol

Retrieve or Delete Medicine
ID:

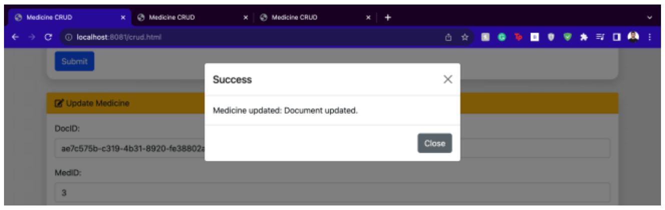

Success

Document IDs with name = Paracetamol:
ae7c575b-c319-4b31-8920-fe38802aa2ba


localhost:8081/crud.html

Update Medicine
DocID:
ae7c575b-c319-4b31-8920-fe38802aa2ba
MedID:
3
Name:
Ibuprofen
Expiration Date:
27/09/2023

Retrieve Medicines by Property
Property Name:
name


Success

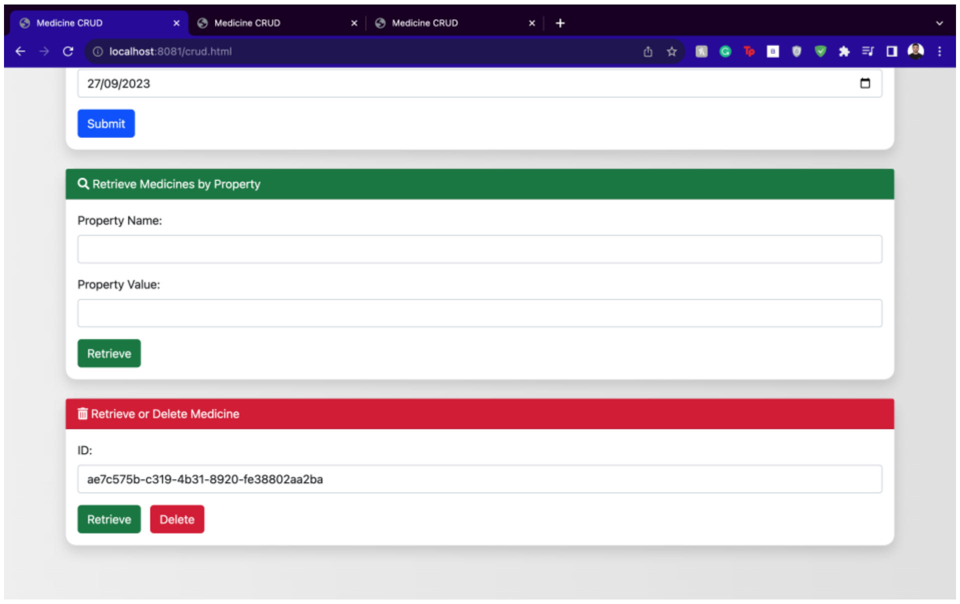
Medicine updated: Document updated.

Deleting a Medicine Record

- A pharmacy discontinues a particular medicine and wants to delete its record.

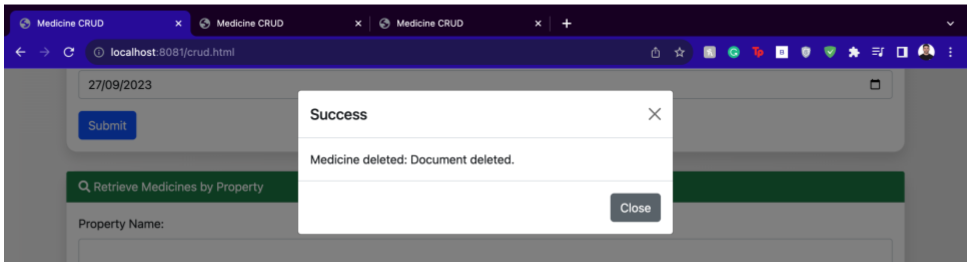
- node2, upon receiving the delete request, checks if it's the affinity node for that medicine. If it is, it deletes the medicine record from its database and then synchronizes this deletion across other nodes.

- If node2 is not the affinity node, it forwards the delete request to the appropriate node (as mentioned in the 'affinityNode' field). Once the record is deleted, the changes are again synchronized across all nodes.



27/09/2023

ae7c575b-c319-4b31-8920-fe38802aa2ba



27/09/2023

Medicine deleted: Document deleted.

Maintaining Data Integrity

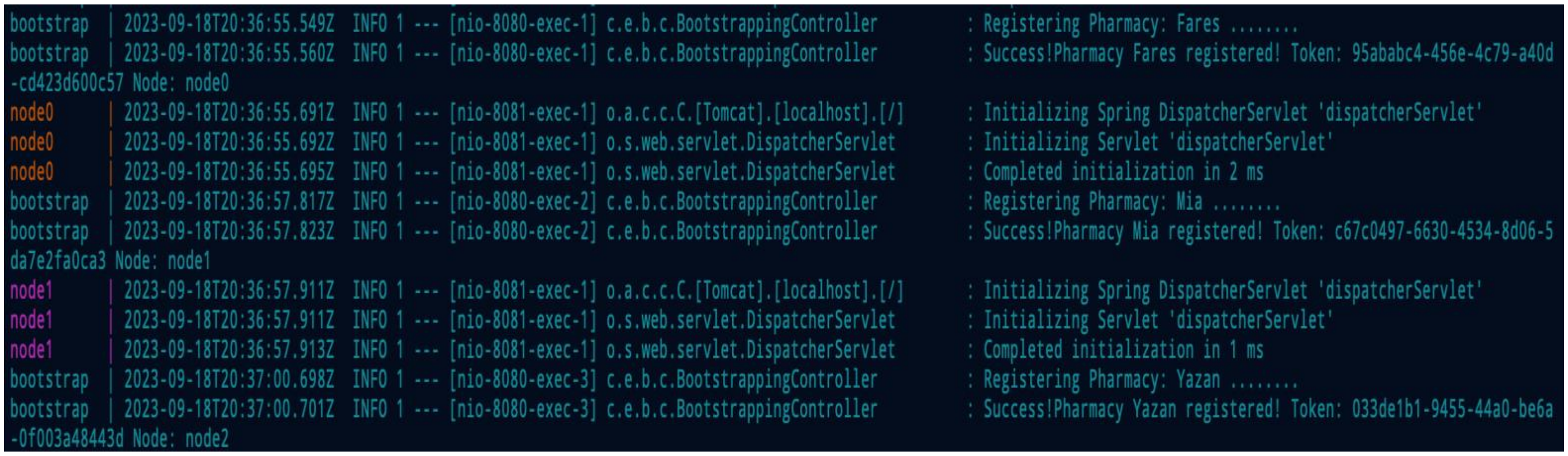
- The system is designed to address potential race conditions, particularly when multiple simultaneous updates to the same medicine occur.

- The "optimistic locking" scheme is employed to manage these conditions. When an update request is initiated, it proceeds only if the current version of the data in the node matches the version from the requesting source.

- In case of a version mismatch, the update operation is declined. The node (or requesting source) then refreshes its version of the data and attempts the update again.

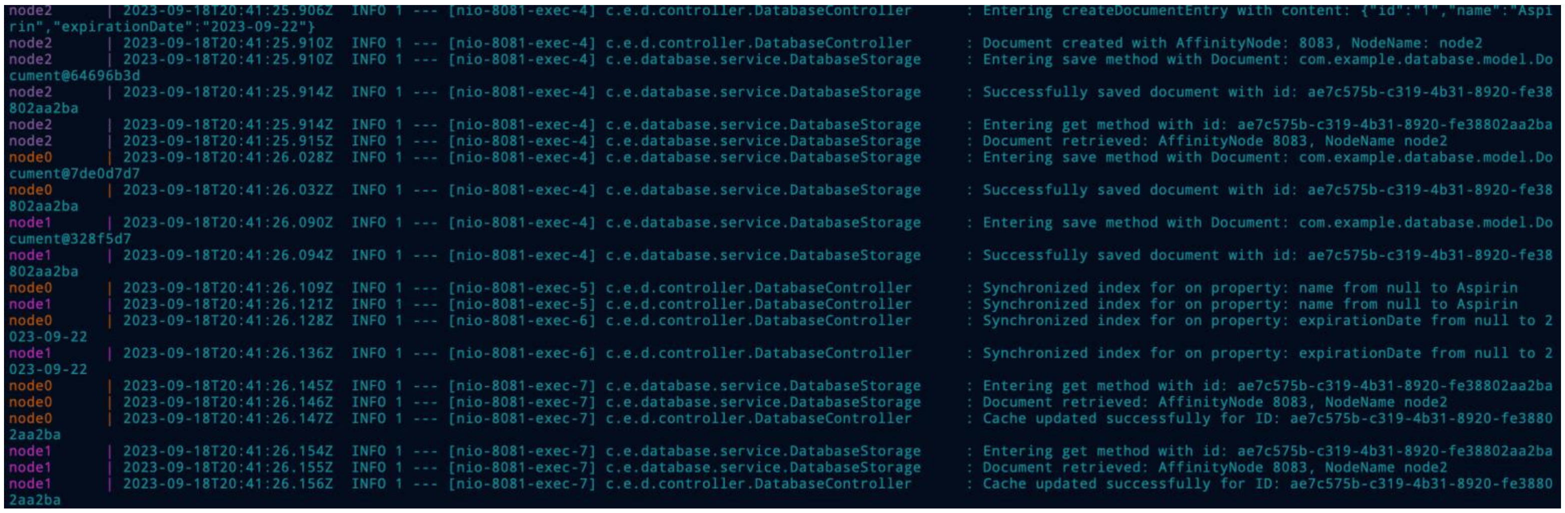
Logs

Initialization of the System:



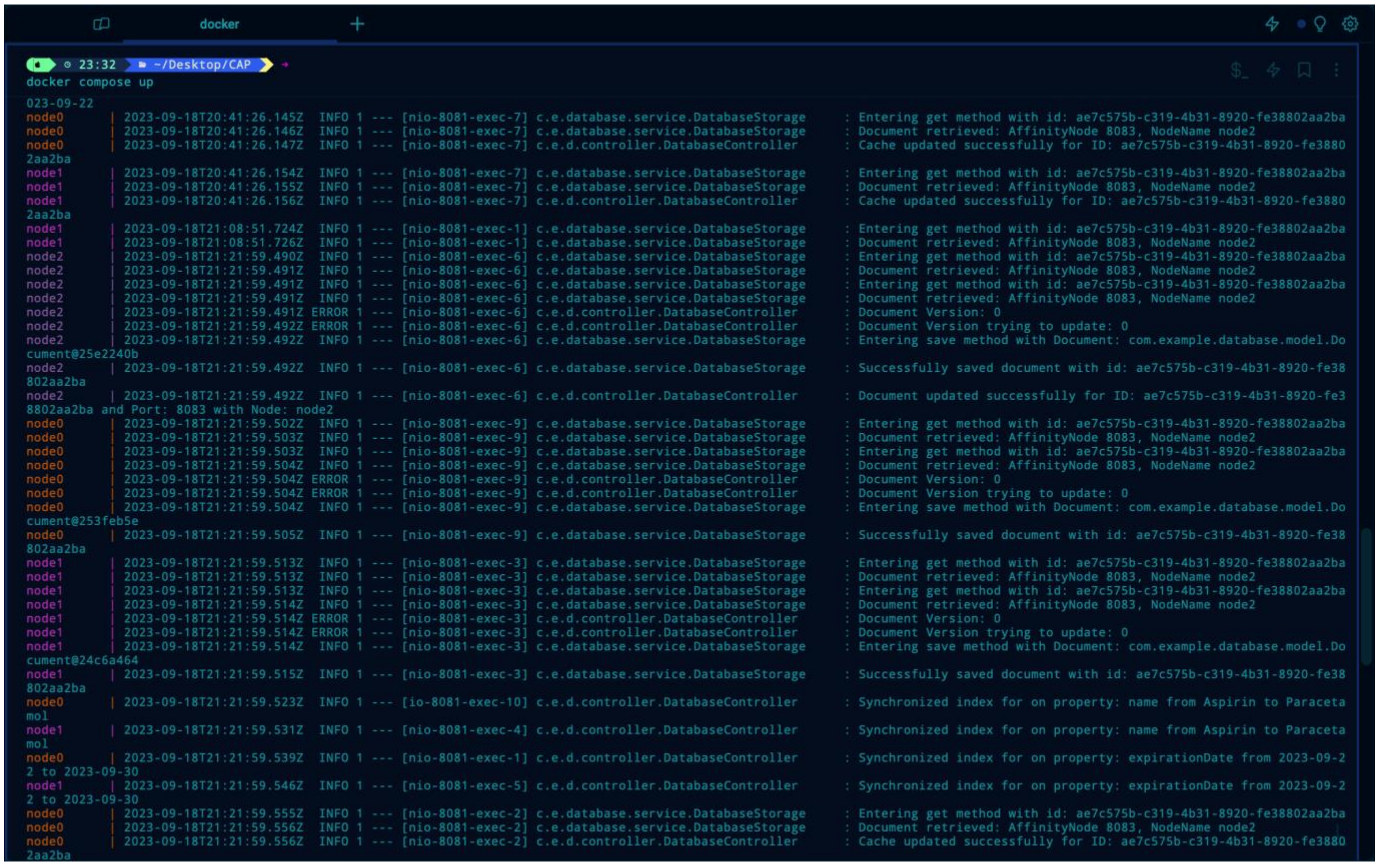
```
bootstrap  | 2023-09-18T20:36:55.549Z  INFO 1 --- [nio-8080-exec-1] c.e.b.c.BootstrappingController          : Registering Pharmacy: Fares ........
bootstrap  | 2023-09-18T20:36:55.560Z  INFO 1 --- [nio-8080-exec-1] c.e.b.c.BootstrappingController          : Success!Pharmacy Fares registered! Token: 95ababc4-456e-4c79-a40d
-cd423d600c57 Node: node0
node0      | 2023-09-18T20:36:55.691Z  INFO 1 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
node0      | 2023-09-18T20:36:55.692Z  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
node0      | 2023-09-18T20:36:55.695Z  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
bootstrap  | 2023-09-18T20:36:57.817Z  INFO 1 --- [nio-8080-exec-2] c.e.b.c.BootstrappingController          : Registering Pharmacy: Mia ........
bootstrap  | 2023-09-18T20:36:57.823Z  INFO 1 --- [nio-8080-exec-2] c.e.b.c.BootstrappingController          : Success!Pharmacy Mia registered! Token: c67c0497-6630-4534-8d06-5
da7e2fa0ca3 Node: node1
node1      | 2023-09-18T20:36:57.911Z  INFO 1 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
node1      | 2023-09-18T20:36:57.911Z  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
node1      | 2023-09-18T20:36:57.913Z  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
bootstrap  | 2023-09-18T20:37:00.698Z  INFO 1 --- [nio-8080-exec-3] c.e.b.c.BootstrappingController          : Registering Pharmacy: Yazan ........
bootstrap  | 2023-09-18T20:37:00.701Z  INFO 1 --- [nio-8080-exec-3] c.e.b.c.BootstrappingController          : Success!Pharmacy Yazan registered! Token: 033de1b1-9455-44a0-be6a
-0f003a48443d Node: node2
```

Creation of a Medicine Record:

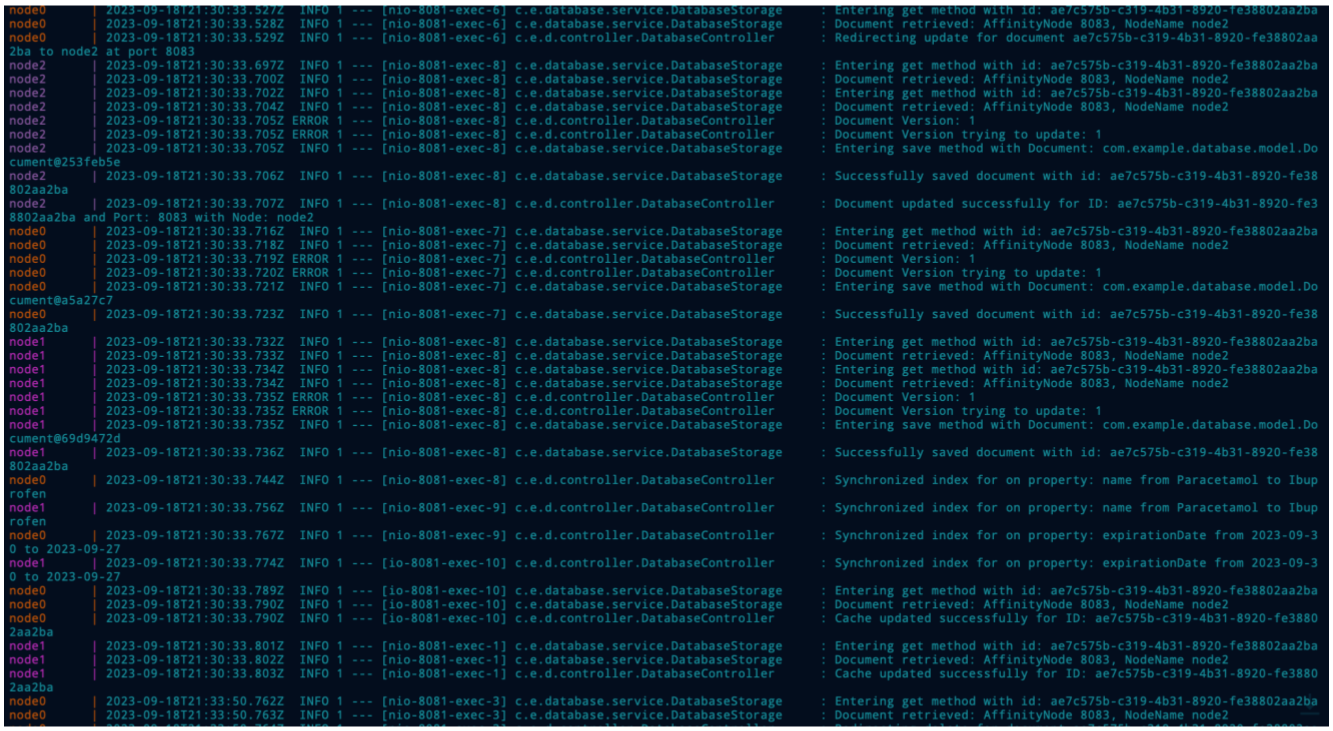


```
node2        | 2023-09-18T20:41:25.906Z  INFO 1 --- [nio-8081-exec-4] c.e.d.controller.DatabaseController      : Entering createDocumentEntry with content: {"id":"1","name":"Aspi
rin","expirationDate":"2023-09-22"}
node2        | 2023-09-18T20:41:25.910Z  INFO 1 --- [nio-8081-exec-4] c.e.d.controller.DatabaseController      : Document created with AffinityNode: 8083, NodeName: node2
node2        | 2023-09-18T20:41:25.910Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@64696b3d
node2        | 2023-09-18T20:41:25.914Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node2        | 2023-09-18T20:41:25.914Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2        | 2023-09-18T20:41:25.915Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T20:41:26.028Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@7de0d7d7
node0        | 2023-09-18T20:41:26.032Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node1        | 2023-09-18T20:41:26.090Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@328f5d7
node1        | 2023-09-18T20:41:26.094Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node0        | 2023-09-18T20:41:26.109Z  INFO 1 --- [nio-8081-exec-5] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from null to Aspirin
node1        | 2023-09-18T20:41:26.121Z  INFO 1 --- [nio-8081-exec-5] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from null to Aspirin
node0        | 2023-09-18T20:41:26.128Z  INFO 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from null to 2
023-09-22
node1        | 2023-09-18T20:41:26.136Z  INFO 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from null to 2
023-09-22
node0        | 2023-09-18T20:41:26.145Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0        | 2023-09-18T20:41:26.146Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T20:41:26.147Z  INFO 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node1        | 2023-09-18T20:41:26.154Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1        | 2023-09-18T20:41:26.155Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1        | 2023-09-18T20:41:26.156Z  INFO 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
```

Retrieval & Update of a Medicine Record:

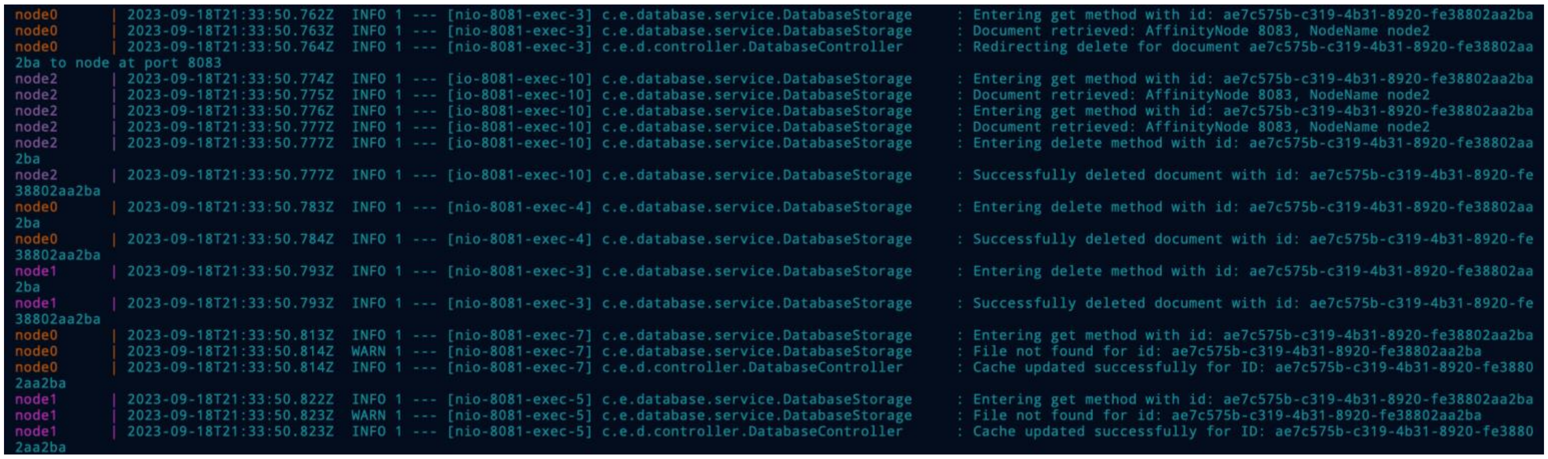


```
23:32  ~/Desktop/CAP
docker compose up
023-09-22
node0        | 2023-09-18T20:41:26.145Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0        | 2023-09-18T20:41:26.146Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T20:41:26.147Z  INFO 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node1        | 2023-09-18T20:41:26.154Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1        | 2023-09-18T20:41:26.155Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1        | 2023-09-18T20:41:26.156Z  INFO 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node1        | 2023-09-18T21:08:51.724Z  INFO 1 --- [nio-8081-exec-1] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1        | 2023-09-18T21:08:51.726Z  INFO 1 --- [nio-8081-exec-1] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2        | 2023-09-18T21:21:59.490Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2        | 2023-09-18T21:21:59.491Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2        | 2023-09-18T21:21:59.491Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2        | 2023-09-18T21:21:59.491Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2        | 2023-09-18T21:21:59.491Z ERROR 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Document Version: 0
node2        | 2023-09-18T21:21:59.492Z ERROR 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Document Version trying to update: 0
node2        | 2023-09-18T21:21:59.492Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@25e2240b
node2        | 2023-09-18T21:21:59.492Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node2        | 2023-09-18T21:21:59.492Z  INFO 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Document updated successfully for ID: ae7c575b-c319-4b31-8920-fe3
8802aa2ba and Port: 8083 with Node: node2
node0        | 2023-09-18T21:21:59.502Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0        | 2023-09-18T21:21:59.503Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T21:21:59.503Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0        | 2023-09-18T21:21:59.504Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T21:21:59.504Z ERROR 1 --- [nio-8081-exec-9] c.e.d.controller.DatabaseController      : Document Version: 0
node0        | 2023-09-18T21:21:59.504Z ERROR 1 --- [nio-8081-exec-9] c.e.d.controller.DatabaseController      : Document Version trying to update: 0
node0        | 2023-09-18T21:21:59.504Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@253feb5e
node0        | 2023-09-18T21:21:59.505Z  INFO 1 --- [nio-8081-exec-9] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node1        | 2023-09-18T21:21:59.513Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1        | 2023-09-18T21:21:59.513Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1        | 2023-09-18T21:21:59.513Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1        | 2023-09-18T21:21:59.514Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1        | 2023-09-18T21:21:59.514Z ERROR 1 --- [nio-8081-exec-3] c.e.d.controller.DatabaseController      : Document Version: 0
node1        | 2023-09-18T21:21:59.514Z ERROR 1 --- [nio-8081-exec-3] c.e.d.controller.DatabaseController      : Document Version trying to update: 0
node1        | 2023-09-18T21:21:59.514Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@24c6a464
node1        | 2023-09-18T21:21:59.515Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node0        | 2023-09-18T21:21:59.523Z  INFO 1 --- [io-8081-exec-10] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from Aspirin to Paraceta
mol
node1        | 2023-09-18T21:21:59.531Z  INFO 1 --- [nio-8081-exec-4] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from Aspirin to Paraceta
mol
node0        | 2023-09-18T21:21:59.539Z  INFO 1 --- [nio-8081-exec-1] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from 2023-09-2
2 to 2023-09-30
node1        | 2023-09-18T21:21:59.546Z  INFO 1 --- [nio-8081-exec-5] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from 2023-09-2
2 to 2023-09-30
node0        | 2023-09-18T21:21:59.555Z  INFO 1 --- [nio-8081-exec-2] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0        | 2023-09-18T21:21:59.556Z  INFO 1 --- [nio-8081-exec-2] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0        | 2023-09-18T21:21:59.556Z  INFO 1 --- [nio-8081-exec-2] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
```



```
node0      | 2023-09-18T21:30:33.527Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:30:33.528Z  INFO 1 --- [nio-8081-exec-6] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0      | 2023-09-18T21:30:33.529Z  INFO 1 --- [nio-8081-exec-6] c.e.d.controller.DatabaseController      : Redirecting update for document ae7c575b-c319-4b31-8920-fe38802aa
2ba to node2 at port 8083
node2      | 2023-09-18T21:30:33.697Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2      | 2023-09-18T21:30:33.700Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2      | 2023-09-18T21:30:33.702Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2      | 2023-09-18T21:30:33.704Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2      | 2023-09-18T21:30:33.705Z ERROR 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Document Version: 1
node2      | 2023-09-18T21:30:33.705Z ERROR 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Document Version trying to update: 1
node2      | 2023-09-18T21:30:33.705Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@253feb5e
node2      | 2023-09-18T21:30:33.706Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node2      | 2023-09-18T21:30:33.707Z  INFO 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Document updated successfully for ID: ae7c575b-c319-4b31-8920-fe3
8802aa2ba and Port: 8083 with Node: node2
node0      | 2023-09-18T21:30:33.716Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:30:33.718Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0      | 2023-09-18T21:30:33.719Z ERROR 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Document Version: 1
node0      | 2023-09-18T21:30:33.720Z ERROR 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Document Version trying to update: 1
node0      | 2023-09-18T21:30:33.721Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@a5a27c7
node0      | 2023-09-18T21:30:33.723Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node1      | 2023-09-18T21:30:33.732Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1      | 2023-09-18T21:30:33.733Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1      | 2023-09-18T21:30:33.734Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1      | 2023-09-18T21:30:33.734Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1      | 2023-09-18T21:30:33.735Z ERROR 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Document Version: 1
node1      | 2023-09-18T21:30:33.735Z ERROR 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Document Version trying to update: 1
node1      | 2023-09-18T21:30:33.735Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Entering save method with Document: com.example.database.model.Do
cument@69d9472d
node1      | 2023-09-18T21:30:33.736Z  INFO 1 --- [nio-8081-exec-8] c.e.database.service.DatabaseStorage     : Successfully saved document with id: ae7c575b-c319-4b31-8920-fe38
802aa2ba
node0      | 2023-09-18T21:30:33.744Z  INFO 1 --- [nio-8081-exec-8] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from Paracetamol to Ibup
rofen
node1      | 2023-09-18T21:30:33.756Z  INFO 1 --- [nio-8081-exec-9] c.e.d.controller.DatabaseController      : Synchronized index for on property: name from Paracetamol to Ibup
rofen
node0      | 2023-09-18T21:30:33.767Z  INFO 1 --- [nio-8081-exec-9] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from 2023-09-3
0 to 2023-09-27
node1      | 2023-09-18T21:30:33.774Z  INFO 1 --- [io-8081-exec-10] c.e.d.controller.DatabaseController      : Synchronized index for on property: expirationDate from 2023-09-3
0 to 2023-09-27
node0      | 2023-09-18T21:30:33.789Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:30:33.790Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0      | 2023-09-18T21:30:33.790Z  INFO 1 --- [io-8081-exec-10] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node1      | 2023-09-18T21:30:33.801Z  INFO 1 --- [nio-8081-exec-1] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1      | 2023-09-18T21:30:33.802Z  INFO 1 --- [nio-8081-exec-1] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node1      | 2023-09-18T21:30:33.803Z  INFO 1 --- [nio-8081-exec-1] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node0      | 2023-09-18T21:33:50.762Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:33:50.763Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
```

Deletion of a Medicine Record



```
node0      | 2023-09-18T21:33:50.762Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:33:50.763Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node0      | 2023-09-18T21:33:50.764Z  INFO 1 --- [nio-8081-exec-3] c.e.d.controller.DatabaseController      : Redirecting delete for document ae7c575b-c319-4b31-8920-fe38802aa
2ba to node at port 8083
node2      | 2023-09-18T21:33:50.774Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2      | 2023-09-18T21:33:50.775Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2      | 2023-09-18T21:33:50.776Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node2      | 2023-09-18T21:33:50.777Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Document retrieved: AffinityNode 8083, NodeName node2
node2      | 2023-09-18T21:33:50.777Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Entering delete method with id: ae7c575b-c319-4b31-8920-fe38802aa
2ba
node2      | 2023-09-18T21:33:50.777Z  INFO 1 --- [io-8081-exec-10] c.e.database.service.DatabaseStorage     : Successfully deleted document with id: ae7c575b-c319-4b31-8920-fe
38802aa2ba
node0      | 2023-09-18T21:33:50.783Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Entering delete method with id: ae7c575b-c319-4b31-8920-fe38802aa
2ba
node0      | 2023-09-18T21:33:50.784Z  INFO 1 --- [nio-8081-exec-4] c.e.database.service.DatabaseStorage     : Successfully deleted document with id: ae7c575b-c319-4b31-8920-fe
38802aa2ba
node1      | 2023-09-18T21:33:50.793Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Entering delete method with id: ae7c575b-c319-4b31-8920-fe38802aa
2ba
node1      | 2023-09-18T21:33:50.793Z  INFO 1 --- [nio-8081-exec-3] c.e.database.service.DatabaseStorage     : Successfully deleted document with id: ae7c575b-c319-4b31-8920-fe
38802aa2ba
node0      | 2023-09-18T21:33:50.813Z  INFO 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:33:50.814Z  WARN 1 --- [nio-8081-exec-7] c.e.database.service.DatabaseStorage     : File not found for id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node0      | 2023-09-18T21:33:50.814Z  INFO 1 --- [nio-8081-exec-7] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
node1      | 2023-09-18T21:33:50.822Z  INFO 1 --- [nio-8081-exec-5] c.e.database.service.DatabaseStorage     : Entering get method with id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1      | 2023-09-18T21:33:50.823Z  WARN 1 --- [nio-8081-exec-5] c.e.database.service.DatabaseStorage     : File not found for id: ae7c575b-c319-4b31-8920-fe38802aa2ba
node1      | 2023-09-18T21:33:50.823Z  INFO 1 --- [nio-8081-exec-5] c.e.d.controller.DatabaseController      : Cache updated successfully for ID: ae7c575b-c319-4b31-8920-fe3880
2aa2ba
```