Decentralized Cluster-Based NoSQL DB System

Atypon Capstone Project

Malek Al-Sadi

GitHub : Malek-Alsadi (malek alsadi) (github.com)

Email : mailto:malek.alsadii@gmail.com

Table of Contents

Over view of the system

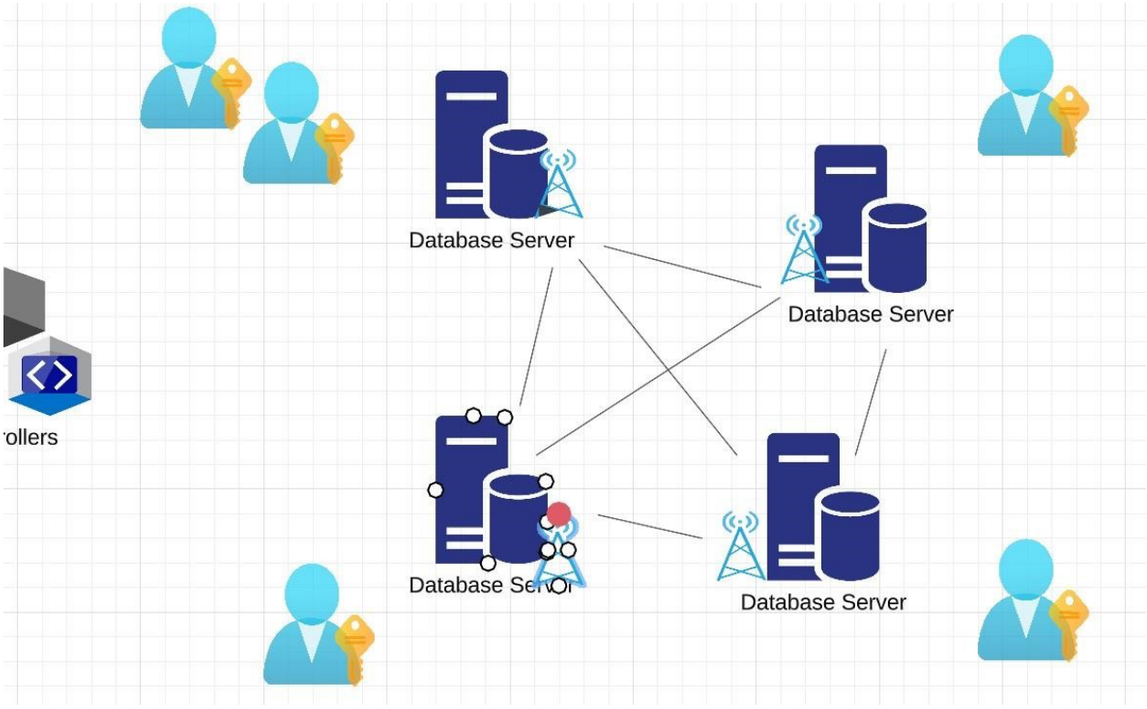

Figure-1 : Over view of the system

A decentralized cluster-based NoSQL database system is a distributed database that operates using a peer-to-peer network architecture. Unlike traditional centralized database systems, this type of database allows data to be distributed across multiple nodes, meaning that if one node fails, the other nodes can still access the data. This ensures high availability and fault tolerance, which is important for applications that require continuous uptime.

Database Structure

NoSQL DB System

I implement Document based database that stores data as JSON record in a JSON Array as required each array with it is schema represents Collection, and Database is the directory that includes many Collection.

In my design I decided to allow user to create database then a directory with the given name will be created and with a directory called “Schema” inside it for the schemas, once the user ask to create a collection and gives the desired schema a JSON file with array will be created in the directory

and schema will be stored in the schema directory inside db directory.

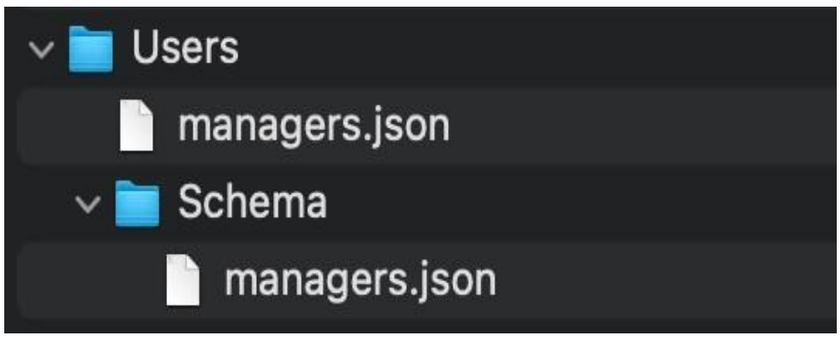

Figure-2: Database Structure

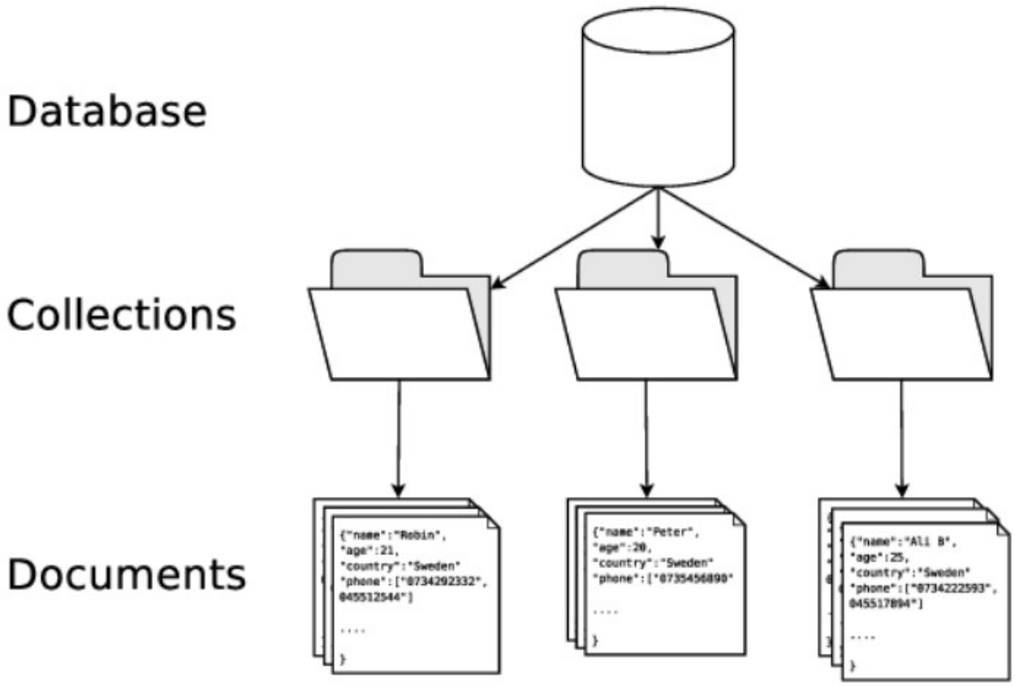

Figure-3: Database structure

JSON based

JSON is a popular data format that is widely used in web applications and APIs. using JSON in my database enables me to easily integrate it with other parts of my application making it easier to read, write and manipulate.

It’s also easy to parse and serialize, making it well-suited for storing, retrieving, and sharing data between nodes.

Fixability, is one differences between document based or nonrelational databases and the relation ones. JSON can be very fixable and allows the user to store any kind of data and can be limited or fixed using schema, which is JSON file that specified rules for JSON objects going to be accepted.

In my DB it's left to user to set the set the fixability according to schema he gives, the Only rule he has to follow that every record must include a primary key called "Id" that should be unique over the other record in the Collection.

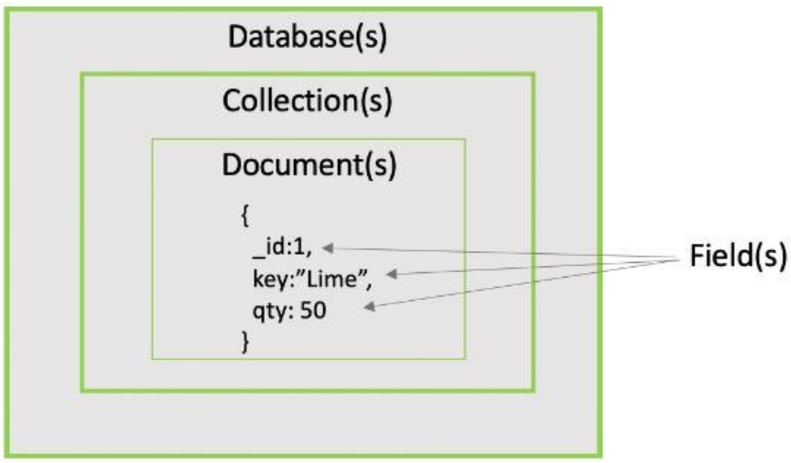

Figure-4: JSON document database

Database Implementation

Dependencies

I build the project as Spring boot maven project , so the using dependencies and reading about them took a while, the most important dependencies I've used beside the Spring standards.

- **Thymeleaf**

one of the most used dependencies, used for web development, providing a way to easily create dynamic HTML pages.

- **Jackson**

Jackson data plinding and annotation are one of the most used libraries in this project, almost in every method or process there is a JSON need to be read written or even modify.

JSON files

Reading or writing on JSON file is done throw ObjectMapper and stored in ObjectNode or JsonNode objects which has the methods "get", "has" or "set" that are used for JSON properties.

```
for(int X : list){

    ObjectNode obj = dbHandler.objectAt(Database,Collection,X);
    String id = obj.get("Id").asText();
    deleteById(Database,Collection,id);
}
```

Figure-5: JSON handling

In this snippet I got list of integers represents the index of JSON that contains some property with certain value, ex: every JSON has property “name” with value “malek” ,using get(“Id”) on each one because It’s the only role for schema that there is an Id unique property, so it is the property used to delete , update and so on.

In other words, like there is name for each person whenever calling it’s known that’s him, there is Id for each record when calling it, it’s known which record is meant.

Once the Collection is created there is usually a schema for it, before adding any record to collection or JSON file it must be validated.

Validation is not necessarily according to strict role it could be some pattern or just sure that some property exists or so on … , it can also be strict and control the number and type of data stored.

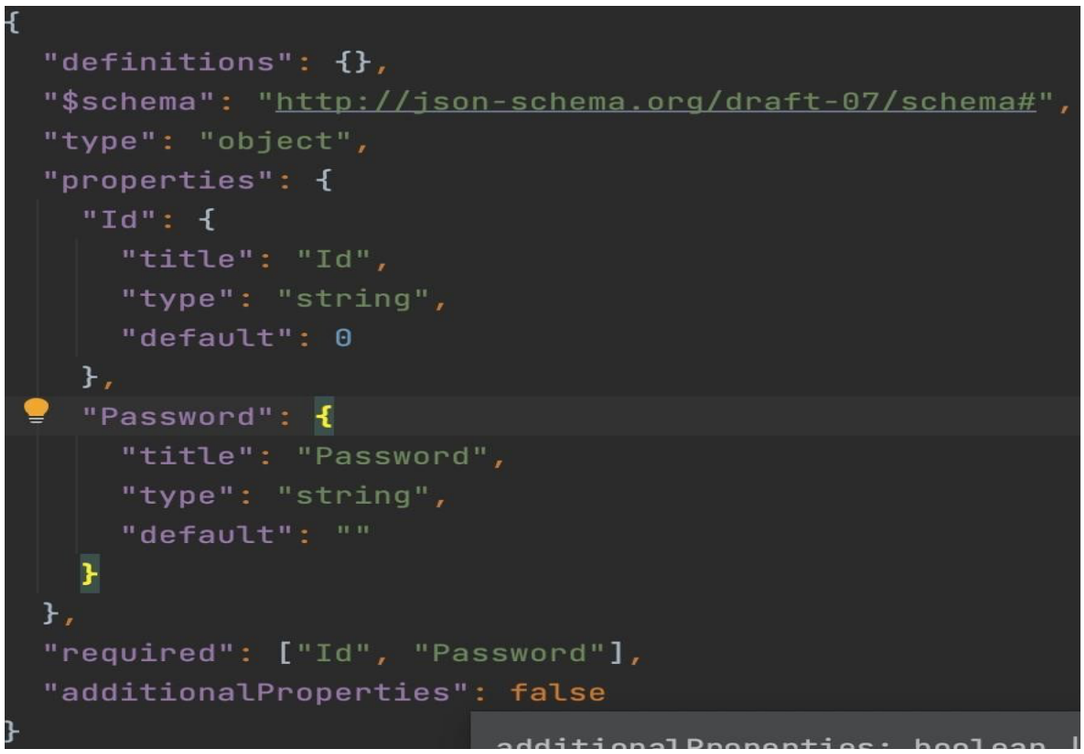

Figure-6: JSON Schema

This schema example of strict one's that required all listed properties and forbidden any additional properties.

Validating is done using object of type "jsonSchemaFactor" which needs the dependency json-schema-validator, calling method called validate that return validations messages and stored in a set, if the set is empty after validation , mean that the JSON object validates successfully.

The actually adding process done by receiving JSON object as a string from the user in the body of the http post request with the name of database and collection, it will be validated, converted to JsonNode using ObjectMapper and add it to ArrayNode called "myArray" in JSON file.

Get the ArrayNode by reading the JSON file as ObjectNode using

ObjectMapper and get the property called “myArray” -which is the name I gave for the array- using get method and store it in an ArrayNode to be able to use its methods, in our case add.

```
ObjectNode root = (ObjectNode) objectMapper.readTree(new File(jsonPath));
ArrayNode myArray = (ArrayNode) root.get("myArray");
if (myArray == null) {
    myArray = objectMapper.createArrayNode();
    root.set("myArray", myArray);
}

JsonNode jsonNode = objectMapper.readTree(Json);
myArray.add(jsonNode);
```

Figure-7: ArrayNode

Indexing

Search for certain JSON object by iterating over JsonNodes one by one in the ArrayNode will surely get the needed object, but let assume we have hundred or thousands of records in a ArrayNode and I’m searching for certain JSON, for each search I’m going to iterate over all the ArrayNode which costs a lot that why Indexing.

My indexing system contains of three indexing layers property indexing, Collection indexing and Database indexing.

Database Indexing

Database indexing based on ConcurrentHashMap with database name String as a key and Collection Indexing as value, contains all methods to add, get and remove from all subtypes of Indexing.

```
HashMap<String , CollectionIndex> map = new HashMap<>();

1 usage  malek
public void initIndex(String Database){
    CollectionIndex index = new CollectionIndex();
    map.put(Database,index);
}
1 usage  malek
public boolean addCollection(String Database, String Collection, String []Properties){
    if(!map.containsKey(Database))
        return false;
    return map.get(Database).initIndexForCollection(Collection,Properties);
}
```

Figure-8: Database indexing

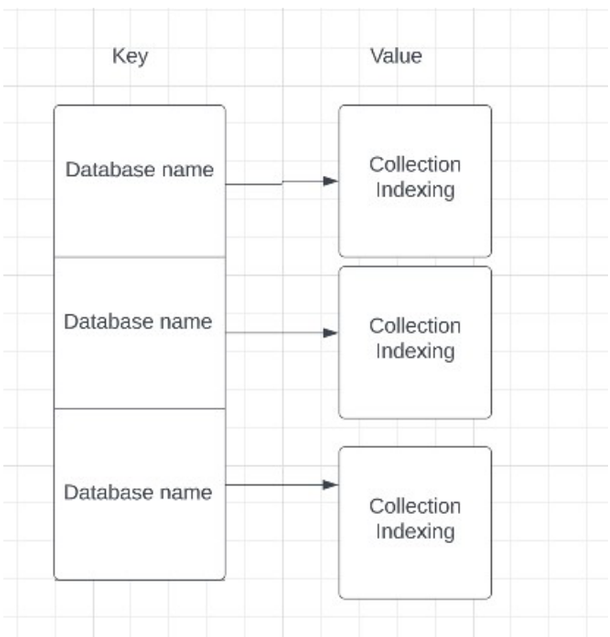

Figure-9: Database indexing

Collection Indexing

Collection indexing is also based on HashMap with Collection name String as a key and Property indexing as value which prevent any redundancy in case of collection name in one database.

```
HashMap<String , PropertyIndex> map = new HashMap<>();

1 usage   malek
public boolean initIndexForCollection(String Collection, String []Properties){
    if(map.containsKey(Collection))
        return false;

    PropertyIndex index = new PropertyIndex(Properties);
    map.put(Collection, index);
    return true;
}
```

Figure-10: Collection indexing

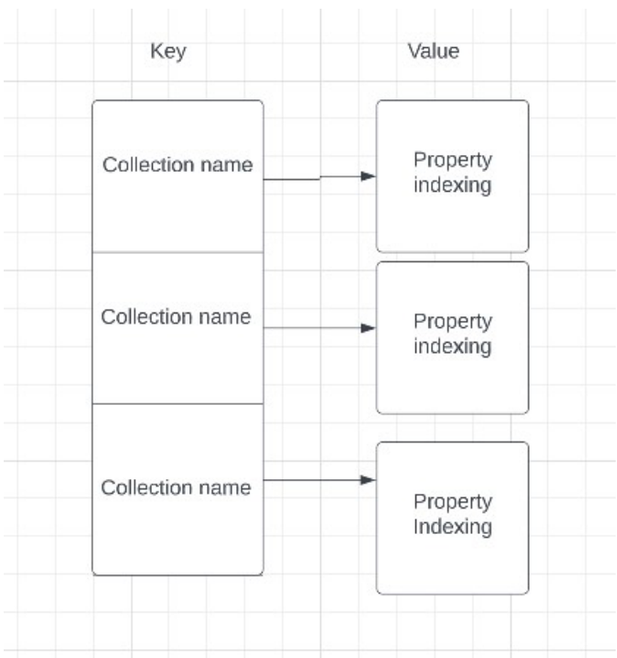

Figure-11: Collection indexing

Property Indexing

Property indexing contains a HashMap with the property name as key, like “name”, “age” … and another HashMap as value.

The second hash map contains the value it’s self as a key such as “Malek”, “Yusef”,…etc. and a list of integers representing the index of JsonNode in the ArrayNode that has that value for the that property as a value.

That makes reaching JsonNodes way faster and easier, but of course time complexity can be improved on cost of space complexity.

```
private HashMap<String , HashMap<String,List<Integer>> > map = new HashMap<>();

1 usage   malek
public PropertyIndex(String []Properties){
    for(String property: Properties)
        map.put(property,new HashMap<>());
}
```

Figure-12: Property indexing

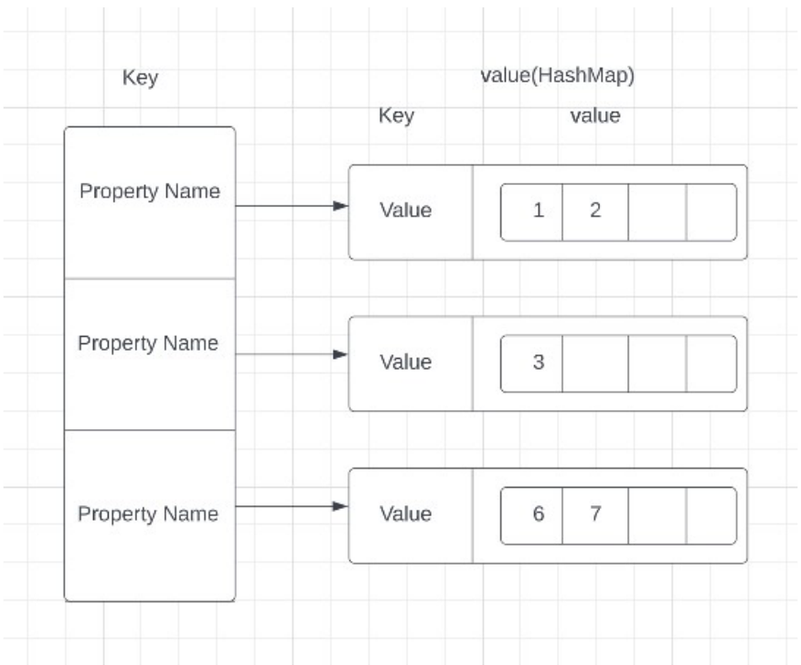

Figure-13: Property queue

Cluster

General view

In general, a cluster is a group of objects or individuals that are similar in some way and often share a common characteristic or purpose. In computer science and data analysis, a cluster specifically refers to a group of data points or objects that are similar to each other based on certain criteria or features.

Our cluster is a group of database nodes each shares the same data but every node has its own copy, and that due to the fact that every user has access to only one the database nodes to reduces load on each node by performing load balance, but they all need to have the same data.

Broadcast

Once request is received asking for any modification process, it will be send to a class called broadcast that is responsible for sending request to all nodes performing that process to their databases. Which means adding new layer before the actual request performing layer, I call it API layer.

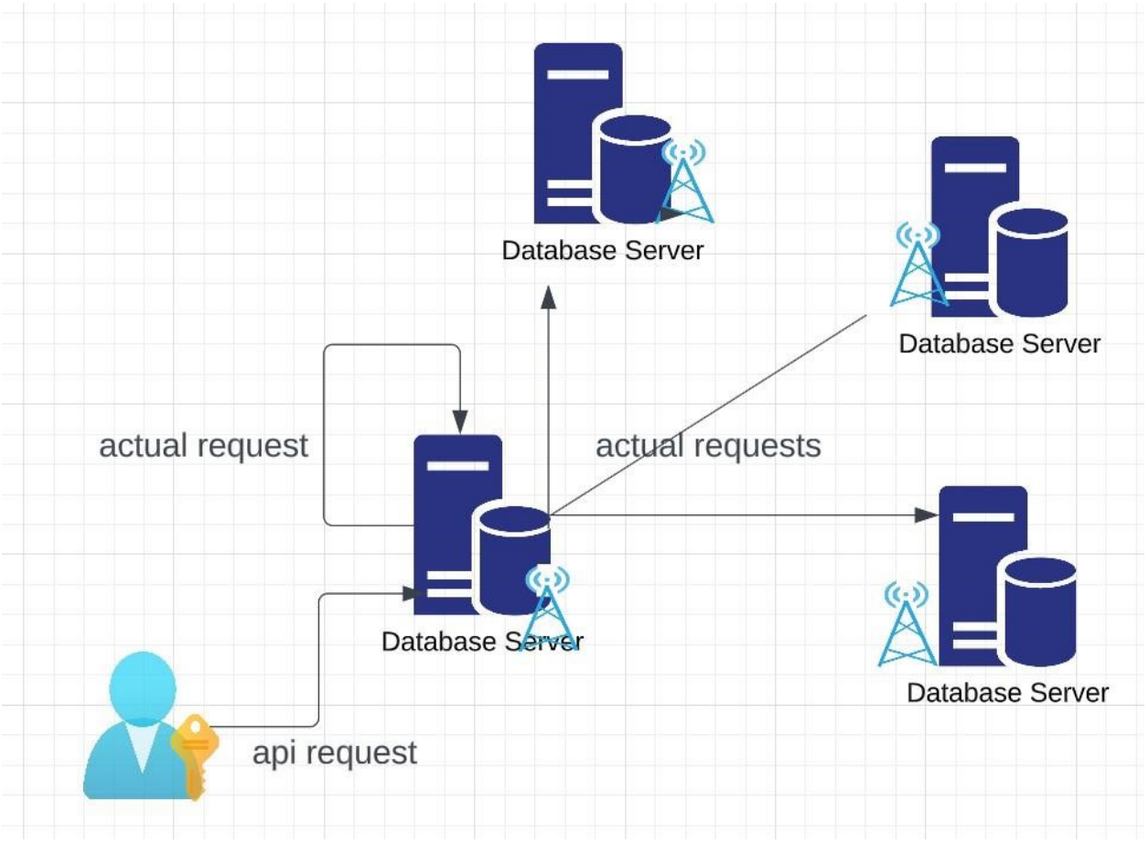

Figure-14: Broadcast

User sends Http Request throw the web app with a URL start with API/… and that's the API layer, that request will be send throw broadcast class to the request Performing layer with URL looks almost like the API request but without API/… end point.

Http Requests

Sending the request done by creating the URL and use it to create HttpURLConnection object in order to open the connection to specified URL, and set the method of the request such as GET, POST, PUT, etc..

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod(method);
```

Figure-15: Connection

OutputStreamWriter is used to write payload that will be sent with the body of the request in case of Post and Put requests using the connection's OutputStream.

Then a BufferReader with gitInputStream() parameter could be used to read the response of the connection's input stream, and here is the actual request sending.

And the response could be received using StringBuilder that reads the response line by line.

Another way of connection is by using the method getResponseCode that returns a response status code, here is where the request sent, by which I can checked weather the connection success or not , if connection success the status code should be 200 otherwise it's failed and every status has it's
own meaning.

```java
// Send HTTP request and receive response
int responseCode = conn.getResponseCode();
JsonNode jsonNode = null;
if (responseCode == HttpURLConnection.HTTP_OK) {
    // Parse JSON response
    ObjectMapper objectMapper = new ObjectMapper();
    jsonNode = objectMapper.readTree(conn.getInputStream());
    System.out.println(jsonNode);
} else {
    System.out.println("HTTP error code: " + responseCode);
}
```

Figure-16: HTTP Request

If the response was JSON node it can be read using ObjectMapper and saved in a JsonNode to be used as wanted.

Affinity

When updating some data there should be an affinity node which is the node responsible for updating, and since the project must be affinity balanced, I stored a number representing the affinity node each time I performed an update that number increases and modulated in range of the number of nodes.

Once an update request received the node checks the affinity number and sends the request to affinity node on URL with api end point, in his turn to broadcast the request to all nodes.

This way it will guarantee that updating process will be done each time by a node to guarantee affinity balance.

Edge cases

In some case user1 with worker1 modifies the database by add, deleting or updating a record in same time user2 with worker2 modifies the same record, that would cause a problem.

To solve this conflict, before sending broadcast command I send check request using broadcast class to all nodes, depends on the process needs, for example: check if exists before deleting or checking schema validating before adding record. that will ensure that once the modification is done in one it will be done in all of them.

Updating checks the Id if exists and also check if data is up to date, Once the request is received I will get the current value of the property needed to edit and broadcast it to all nodes to ensure that all nodes has the same current data, if they were same the request will be send to affinity node to be broadcasted from there.

Queries command doesn’t need this handling because its access exist data

without changing any values.

Figure-17: Check controller

Multi-threading

Multi-threading is a programming concept where multiple threads of execution run concurrently. This can lead to improved performance and efficiency in certain types of applications, such as when there is multiple users using same node.

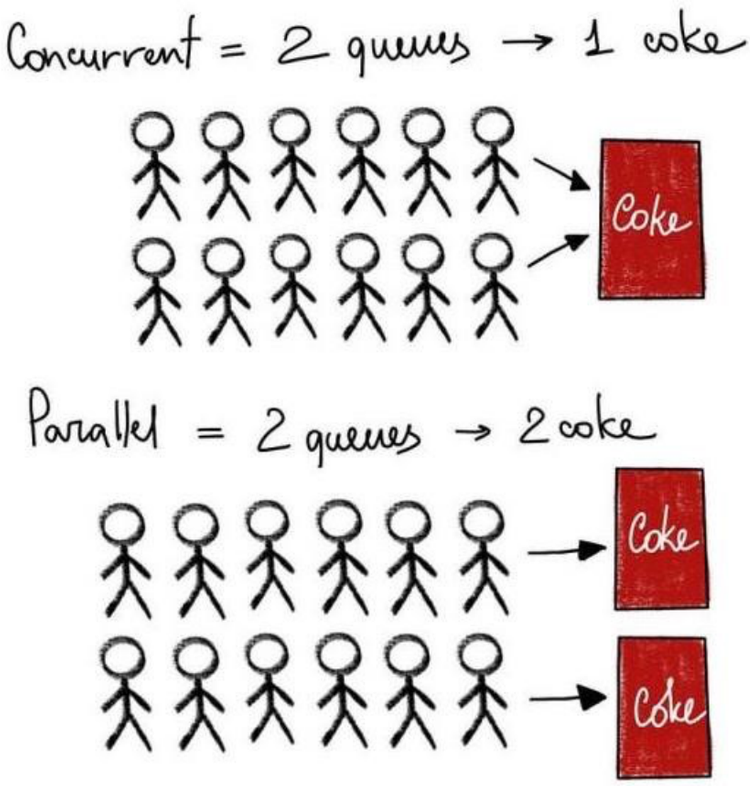

Figure-18: Multithreading

Asynchronous is a programming concept **allows multiple things to happen at the same time**. When your program calls a long-running function, **it doesn't block the execution flow**, and your program continues to run. When the function finishes, the program knows and gets access to the result

A good way to understand this:

Multithreading is about workers and Asynchronous is about tasks.

Spring multithreading

Spring is a framework to develop big enterprise applications which include servlets as well. And Servlet starts a separate thread for each incoming request.

Asynchronous

a single process thread runs all the time and it has the ability to switch from one function to another.

To configure @Async in a Spring application, several steps are required. First the application must have a task executor been defined that will be used to manage the threads that execute the asynchronous methods. This can be done by creating a ThreadPoolTaskExecutor bean in the application context and configuring it with the desired thead pool size and other properties.

If task executor is not explicitly defined, spring will use default on called

SimpleAsyncTaskExecutor that creates new thread for each task, without pooling or reuse of threads.

```java
@Bean
public Executor asyncExecutor() {
    ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
    executor.setCorePoolSize(10);
    executor.setMaxPoolSize(50);
    executor.setQueueCapacity(1000);
    executor.initialize();
    return executor;
}
```

Figure-19: Configration bean

Core Pool Task is the number of threads will be made once the application is executed, and more threads will be made as needed up to 50 threads.

Queue capacity is the tasks that are given over the max Pool size.

Next, the @EnableAsync annotation must be added to a configuration class, which enables Spring's asynchronous method execution capacities. This annotation should be placed on a configuration class that is annotated with @Configuration.

Finally, the @Async annotation can be added to any method in the application that should be executed asynchronously. When a method is annotated with @Async, Spring will execute it on a separate thread managed by the task executor bean, and the caller will immediately return without waiting for the method to complete.

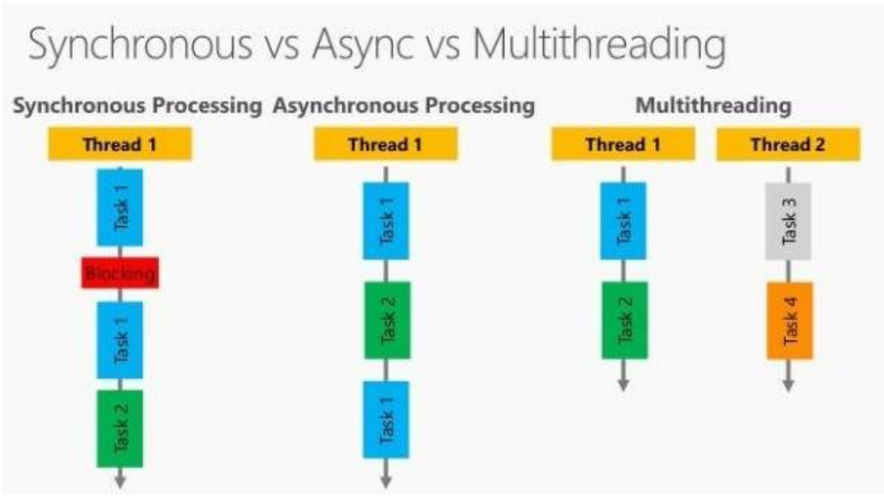

Figure-20: Sync, Async & multithreading

ConcurrencyHashmap

Also in Indexing. I've used ConcurrencyHashmap instead of Hashmap to cope with asynchronous and multithreading

```
21 usages
ConcurrentHashMap<String , CollectionIndex> map = new ConcurrentHashMap<>();
```

Figure-21: ConcurrencyHashmap

Race Conditions

Situation when data is corrupted due to concurrent threads execution.

In our case, any modification process like adding deleting or updating will cause a race condition. For example if threadA was deleting Collection c1 from Database db1, meanwhile

threadB was adding record1 to c1, both threadA and threadB had checked if the database is exist and trying to perform the process, threadA will delete the database and threadB will try to add to that database which will cause an exception.

I solved this dilemma by synchronized the block that contains the check and updating process with costume key which could by one of three levels.

1. Database_Key
2. Collection_Key
3. Record_Key

Database_key contains the database name and will be used as super class for Collection_key, Collection key contains collection name and will be used as super class for Record_key.

```
Database_Key key = new Database_Key(database);
synchronized (key) {
    File dbDir = new File(getPath(database, schema: false));
    if (dbDir.exists() && dbDir.isDirectory()) {
        return new FeedBack( message: "Database already exists.", statusCode: 300);
    }

    if (!dbDir.mkdirs()) {
        return new FeedBack( message: "Error creating database.", statusCode: 400);
    } else {
        File schemasDir = new File(getPath(database, schema: true));
        schemasDir.mkdirs();
        return new FeedBack( message: "Database created successfully.", statusCode: 201);
    }
}
```

Figure-22: Synchronize block

Synchronized block will used an object as lock and that objects will be stored in a HashMap with one of these keys as a key ,if subclass is used all its super classes will be checked, for example if some block used Database_key with database called db1 and another block uses Record_key with value(Id = 1 , collection = “c1” and database = “db1”), the equals method will compare the id then checks the collection and lastly will checks the database which will be found to be used.

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Database_lock)) return false;
    if (!(o instanceof Record_lock)) {
        return super.equals(o);
    }

    Record_lock that = (Record_lock) o;
    return Objects.equals(getId(), that.getId())
            && getCollection().equals(that.getCollection())
            && getDatabase().equals(that.getDatabase());
}
```

Figure-23: Record_Key equals

Bootstrap

Bootstrap node is the node responsible for assigning users to nodes and the users are not going connect with it after signing up.

When user signs up and entered his credentials, they will be saved In bootstrap it's self to avoid repetition and will be sent to one of the node to enable the user to connect with it, and returns the Id encrypted as a AES token and a URL for the assigned node.

If a user tried to sign up using used id it will be refused, but as mentioned before searching over all JsonNodes cost very high time complexity, and that why I used LFU caching in bootstrap.

Another functionality of bootstrap is to set up the affinity to all node to 0.

Docker

Figure-24: Docker

After creating DB node with all functionalities creating the multiple db nodes will be by creating multiple docker containers.

Docker container is a standalone, lightweight and executable package that include everything needed to run an application, include the code, system tools, libraries and dependencies.

Each Docker container runs in its own isolated environment, with its own filesystem, network, and resources. Containers are built from Docker images, which are a set of instructions for creating a container, including the operating system, application code, and configuration files.

Building images

Creating Docker file in the project needed to be containerized contains:

Image that will be used to build your image, in my case its openjdk:19. working directory that will be made inside the container and everything will be stored or done inside it.

Copy command to make a copy of the jar file of this project in working directory.

CMD is the last instruction in docker file, it used to define the default command that will be executed when a container is started, in our case it (“java”, “-jar”, “jarfile name”).

```
FROM openjdk:19
WORKDIR /app
COPY target/worker-0.0.1-SNAPSHOT.jar /app
CMD ["java", "-jar", "worker-0.0.1-SNAPSHOT.jar"]
```

Figure-25: Docker file

Docker compose

Figure-26: docker composed

It’s a tool that provides the developers a simple way to create and run multiple containers and managing the configuration of them.

Number of nodes doesn’t have to be fixed but, in my project, I created 4 docker images of type worker called worker1, worker2, etc… and run containers from them.

Only One Node of bootstrap need to be created because there not much to do and it’s depended on workers, and a docker network that all the previous containers will be connected throw it.

Running the docker files or the project in general will be by creating executing the command java -jar (JarName) in terminal.

All node in the cluster is connected at port 8090 form inside the docker, and that why sending request throw broadcast will be using the name of the container as a host instead of localhost.

But the user will still connect to them using the public port specified in the docker compose file.

```
version: '3'
services:
  worker1:
    build: ./worker
    restart: always
    container_name: worker1
    ports:
      - "8090:8090"
    networks:
      - my-network

  worker2:
    build: ./worker
    restart: always
    container_name: worker2
    ports:
      - "8091:8090"
    networks:
      - my-network

  worker3:
```

Figure-27: Docker composed

The port on the left is the host machine port used by the users and right sided one is the container port used in http requests between containers.

Demo

A bank system spring boot application that is connected to the created database and using it to store Employee and Costumers information.

Contains of 5 controllers:

- Get Update Controller
- Delete Controller
- Add Controller
- Api Controller
- Auth Controller

AuthController is the one responsible for display authentication page and information and validating user credentials.

Using AuthService that do comfimation operation using the data brough by AuthDAO that connects to the database and brings the authentication info needed.

ApiController is the one responsible for displaying html forms and transition between them according the received get request.

AddController is the one responsible for receiving post requests for creating database and collection or record for one of the created collections Using AddingService which uses the DAO to connect to the database.

DeletingController is the one responsible for receiving Delete requests and deleting exist database or record throw DeletingService that connects to
database throw DAO.

GetController is the one responsible for receving get or put requests and displays the stored data after getting them throw GetService that get them from the database using DAO.

Or sends an update request to database throw DAO and review the updated data after update.

Objects

1-Person: contains two strings Id and name.

2-Costumer: extends Person and implements double balance attribute.

3-Employee: extends Person and implements string Role and double salary attributes.

HttpSession

When a user opens a website a new session will be made and will be used to store need information like credentials that must be send with each http request send to database.

Why session not cookies?

Its due to the main difference between session and cookie that is the session information will be stored in the server side which keeps it save from any unauthorized access while cookies are saved on client side which can be access easer from unauthorized users.

How does it work?

Each session created will have unique session Id for each server opened or use.

Caching System

When a user sign up the token and URL or his worker will be saved in an ArrayNode to be used to get the url when signing in later.

Once the user sign in the session saved him token and password but to find the URL it need to iterate over all Json node stored to find the token that matches the one saved in session, doing this process for every

request is too expensive, and that why once the user signed in the search process will done but with adding cache to it, to make the most accessed data easier and faster to get.

Cache

I have mentioned that the bootstrap used caching for user credentials and demo used caching for URL and tokens.

So, what is Cache?

Cache refers to a small amount of high-speed memory that is used to temporarily store frequently accessed data or instructions in a computer system. The purpose of cache is to reduce the average time it takes to access data from the main memory, which is slower and larger than the cache.

What type did I used?

LFU cache which means Least frequently Used and it's a type of cache that prioritizes the retention of the most frequently accessed items.

In bootstrap cache was storing Boolean value based on the Id and type of user (user, manager) , if the it was found in the cache that means it has been used before so the bootstrap will refuse it.

In demo application cache was storing URL given for token that is used as

a key for that cache.

Security

When first the user sends his Id and Password to Bootstrap node it will be send in the header using basic authentication.

Basic authentication

Basic authentication merges the Id and Password with a : between them and encode the given id by Base64. To be decoded by the bootstrap sent to selected node encrypted as AES token and send
back to the user

The Id would not be used as it's to keep user information secret.

AES encryption

It's a widely used encryption algorithm that provides strong security for digital data. It is asymmetric key encryption algorithm, which means that the same key is used for both encryption and decryption.

For each send request an authentication need to be verified by decoding the received token splitting the merged string between token and password and decrypting the given token to get the id, and check if it's existing in the database and if it's password matches the given password.

Data Structures and algorithms

Hash map was used in indexing in the worker node which used O(1) time complexity to get or add data instead of searching along the ArrayNode.

Hash map was used in bootstrap and demo nodes which get and delete data in O(1) time complicity instead of searching along the ArrayNode.

Design patterns

- **Strategy method design pattern**: used in the worker node in the token authentication. An interface was implemented with generate and verify method that is implemented in a TockenAuth class the generate token using AES encryption class on the overrided generate method and verify token using AES decrypting.
- **Singleton design pattern:**
  1. used in the cache to the same instance of cache class all over the project.
  2. used in indexing in the worker project to be used all over the project.
- **Template method design pattern**: used for HttpRequest whenever it's used it's always has the same structure for get and delete requests and other structure for post and put request so I have created an abstract class called RequestSender with abstract methods getRequest and PostRequest and a

SendHttpRequest method that uses the abstract class to send request depends on the type request that will implement them.

- **DAO:** database access object is a structure pattern that allow us to isolate the application or server layer from the persistence layer, in my project it is the layer responsible for accessing file in the database or sending requests and connecting to cluster in demo.

Solid Principles

- **(SRP) Single Responsibility principle**: has been used all over the project such as setAffinity, Bootstraping endcryption, building schema in demo and validating it in workers.
- **(OCP) Open Close Principle**: most of the classes has been implemented in a way that makes it opened to be extended in the future but without being modified.
- **(LSP) Liskov Substitution**: the tokenAuth class which implements a generate and verify method from iAuth interface is good practice on this principle. which refers to using be able to use the subclass in place of the super class and the project will stay working without problems.
- **(ISP) Interface Segregation Principle**: is that the to use a method don't have to implement all the methods in the interface. I applied this principle in bootstrapping by creating

interface only for encrypting and in demo application by creating interface for schema generating.