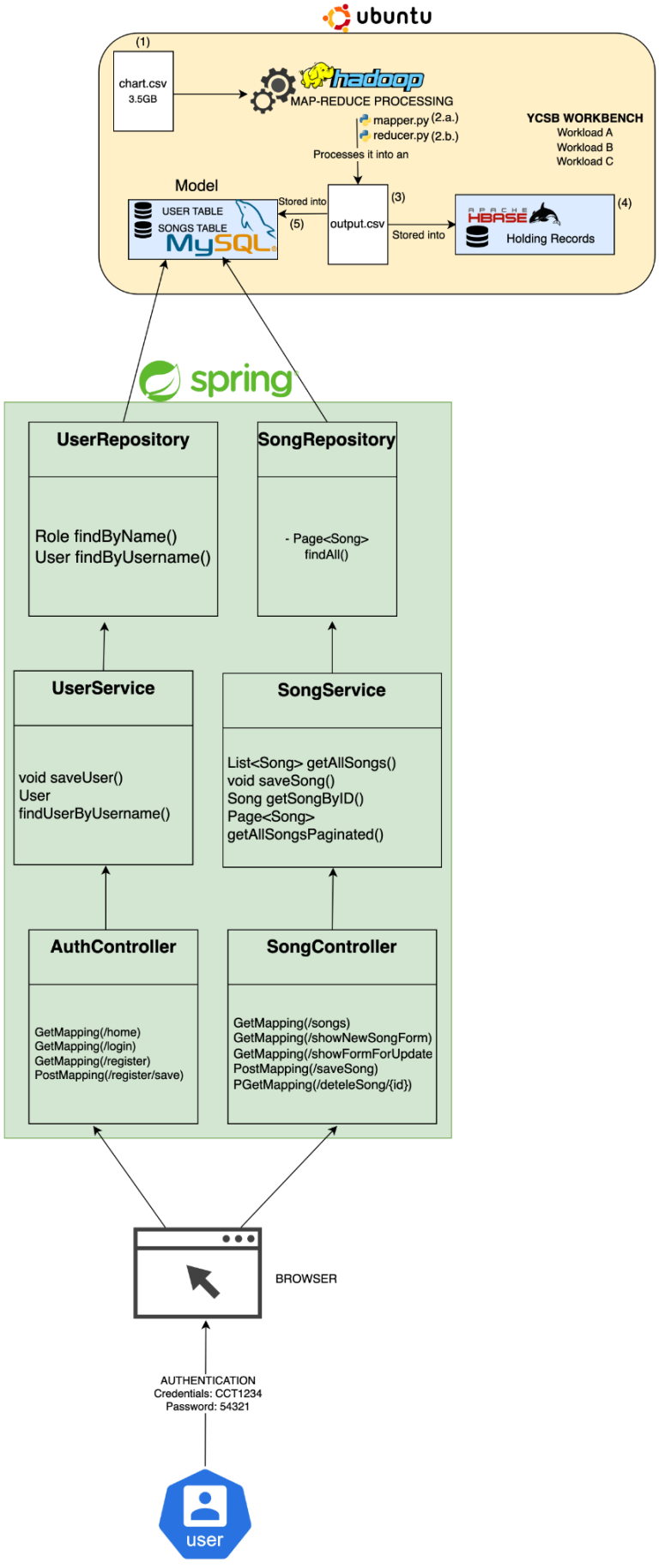
CCT College Dublin

Assessment Cover Page

Module Title:	Networking
Assessment Title:	Data Storage Solutions (5 ECTS) Commercial Solutions Design (10 ECTS)
Lecturer Name:	Muhammad Iqbal, David McQuaid
Student Full Name:	Ignacio Alarcon Bernardo Gandara
Student Number:	2021357 2021283
Assessment Due Date:	24th May
Date of Submission:	24th May

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.


ubuntu
(1)
chart.csv
3.5GB
hadoop
MAP-REDUCE PROCESSING
mapper.py (2.a.)
reducer.py (2.b.)
Processes it into an
YCSB WORKBENCH
Workload A
Workload B
Workload C
Model
USER TABLE
SONGS TABLE
MySQL
Stored into
(5)
output.csv
(3)
Stored into
HBASE
Holding Records
(4)
spring
UserRepository
Role findByName()
User findByUsername()
SongRepository
- Page<Song>
findAll()
UserService
void saveUser()
User
findUserByUsername()
SongService
List<Song> getAllSongs()
void saveSong()
Song getSongByID()
Page<Song>
getAllSongsPaginated()
AuthController
GetMapping(/home)
GetMapping(/login)
GetMapping(/register)
PostMapping(/register/save)
SongController
GetMapping(/songs)
GetMapping(/showNewSongForm)
GetMapping(/showFormForUpdate
PostMapping(/saveSong)
PGetMapping(/deteleSong/{id})
BROWSER
AUTHENTICATION
Credentials: CCT1234
Password: 54321
user

INTRODUCTION

When selecting an appropriate dataset, we prioritized two key aspects: simplicity and size. We wanted a dataset that was easy to understand to avoid complications during processing and to ensure the output was engaging. Additionally, the dataset needed to be at least 1GB in size, as per the assignment requirements. Finding a dataset that met both criteria was more challenging than anticipated. Simple datasets were typically small, and larger datasets often revolved around less engaging topics like loan approval rates.

After extensive research, we discovered a dataset on Kaggle that suited our needs perfectly. It comprised the "Top 200" and "Viral 50" charts published globally by Spotify, covering all entries since January 1, 2017. This dataset totaled 3.48GB in CSV format, striking the right balance between a fun topic and a substantial file size.

The dataset presented several challenges: it included multiple columns that were not relevant to our project (such as rank, date, URL, region, chart, trend, and streams). Additionally, due to the global nature of the data, many song titles and artist names contained characters beyond the ASCII range, posing processing difficulties. Furthermore, the frequent updates to the charts (every 2-3 days) resulted in many duplicate entries.

Given the dataset's size, we opted for the Hadoop Map-Reduce framework, which is well-suited for processing large volumes of data in parallel across large clusters of nodes (hadoop.apache.org, 2022).

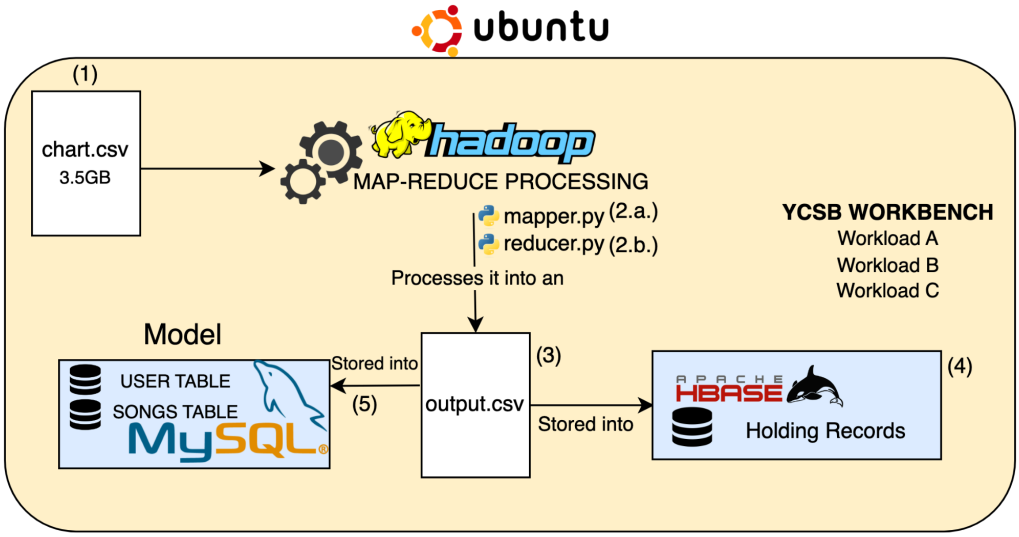
Our goal was to store a list of songs (including titles and artists) that appeared in the TOP 200 between 2017 and 2021. The `.mapper.py` and `.reducer.py` scripts were designed to filter out unnecessary columns, remove entries with special characters, and eliminate duplicates.

VIDEO DATA STORAGE DEMO:
https://drive.google.com/file/d/1AtJArydlGdidyAUbmnpF4fPYZreUuUQi/view?usp=share_link

VIDEO SPRING BOOT APPLICATION - BERNARDO GANDARA:
https://drive.google.com/file/d/1DyCfiOp9Ia67G243-jPCq4dHcLbCCaAZ/view?usp=sharing

VIDEO SPRING BOOT APPLICATION - IGNACIO ALARCON:
https://drive.google.com/file/d/1LL7pRvAA3VUKCajMLD8ymhVU1E6eLZQO/view?usp=sharing

DATA STORAGE ARCHITECTURE AND PROCESSING



1. Input CSV file (Chart.csv)

This is the original csv data set. As mentioned previously, the file has a size of 3.48GB. The columns included are *title, rank, date, artist, url, region, chart, trend* and *streams*. Besides, it has 25.960.383 records/rows. The dataset contains the "Top 200" and "Viral 50" charts published by Spotify. (www.kaggle.com, n.d.)

2. Hadoop Map-Reduce Processing

As mentioned previously, we had a lot to take into consideration at the moment of processing this file. There was a lot of data that we wanted to remove to keep the focus on the title and artist of individual songs (regardless of repetition) that fall into the boundaries of ASCII values.

The command we use to execute Map-Reduce was:

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -mapper ./**mapper.py** -reducer ./**reducer.py** -input /spotifyData/**charts.csv** -output /**output.csv**

a. mapper.py

The purpose of this script, is design to read a list of song titles and their respectives artist form our DataSet chosen, first it checks if the song title and artist names contain only letters and spaces, for the scope of the project, numbers and special characters are being excluded, if they pass the constraints it prints them out.

So in a brief explanation of what it does the steps are:

Reading the List: The script reads the list of songs and artists line by line. Think of this list as a spreadsheet where each line is a row, and the columns include details like the song title and the artist's name.
Skipping the Header: Usually, the first line of such a list contains the column names like "Title", "Artist", etc. The script skips this first line since it's not an actual song or artist.
Checking Each Entry: For each song in the list:

1. It looks at the song title.
2. It looks at the artist's name.
3. It checks if both the title and the artist's name contain only letters and spaces. If there's anything else (like numbers or symbols), it ignores that entry.

Printing Valid Entries: If both the title and the artist's name are valid (only letters and spaces), it prints them out in a specific format.

It's extremely useful since it helps filter clean data and persists in consistency, ensuring that only songs with valid titles and artist names are considered. This is useful to maintain the structure of our data set as desired. By ensuring that only "proper" names are printed, it helps in maintaining a clean dataset, which will be crucial for afterwards feeding data into the database.

```
 GNU nano 6.2                                    mapper.py
#!/usr/bin/env python3
import sys
import csv
import re

# Regular expression to match only words containing letters (ignores non-letter characters)
pattern = re.compile("^[A-Za-z\s]+$")

# Reads input line by line
reader = csv.reader(sys.stdin)
next(reader)  # Skip the header

for row in reader:
    title = row[0]  # The 1st column is title
    artist = row[3]  # The 4th column is artist

    # Apply the pattern to both artist and title
    if pattern.match(title) and pattern.match(artist):
        print(f'{title},{artist}')
```

b. reducer.py

The reducer script reads a list of songs and artists, making sure that each song title is unique, since this chart of the top 200 Songs over a few years, there are songs that have been on the top for a long time therefore we only want to count them as 1 over the years. But artists can have multiple hits of course.
The steps are:

Prepare for Tracking: The script sets up an empty collection to remember song titles it has already seen.
Print the Header: It starts by printing a header line that labels the columns as "Title" and "Artist".
Read and Process Each Entry: For each song and artist entry:

1. It cleans up the line and splits it into the song title and artist name.
2. It checks if there are exactly two parts (a title and an artist).
3. If the song title hasn't been seen before:
    a. It remembers this title.
    b. It prints the song title and artist name.
4. If there's a problem with the line (not splitting into exactly two parts), it logs an error message.

```
  GNU nano 6.2                                    reducer.py
#!/usr/bin/env python3
import sys

# Use a set to track seen song titles
seen_titles = set()

# Output the CSV header
print('Title,Artist')

for line in sys.stdin:
    # Strip and split the line by commas
    split_line = line.strip().split(',', 1)

    if len(split_line) == 2:
        title, artist = split_line
        if title not in seen_titles:
            seen_titles.add(title)
            # Output title first, then artist
            print(f'{title},{artist}')
    else:
         print("Error")
```

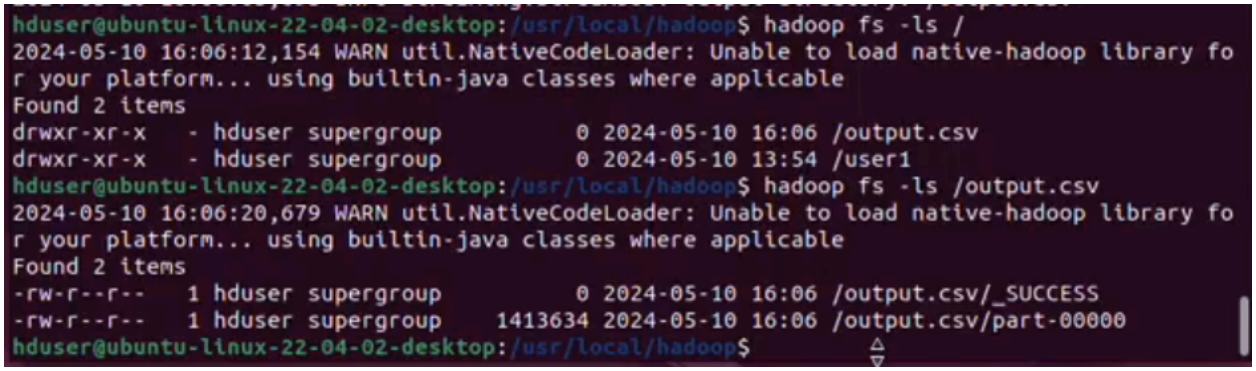
Ensuring Unique Titles and reducing to clean organized data is the result of our reducer.py

Important to highlight that after this commands is need it to given them permission with the following command:

```
$ chmod +x mapper.py reducer.py
```

3. **Output.csv**

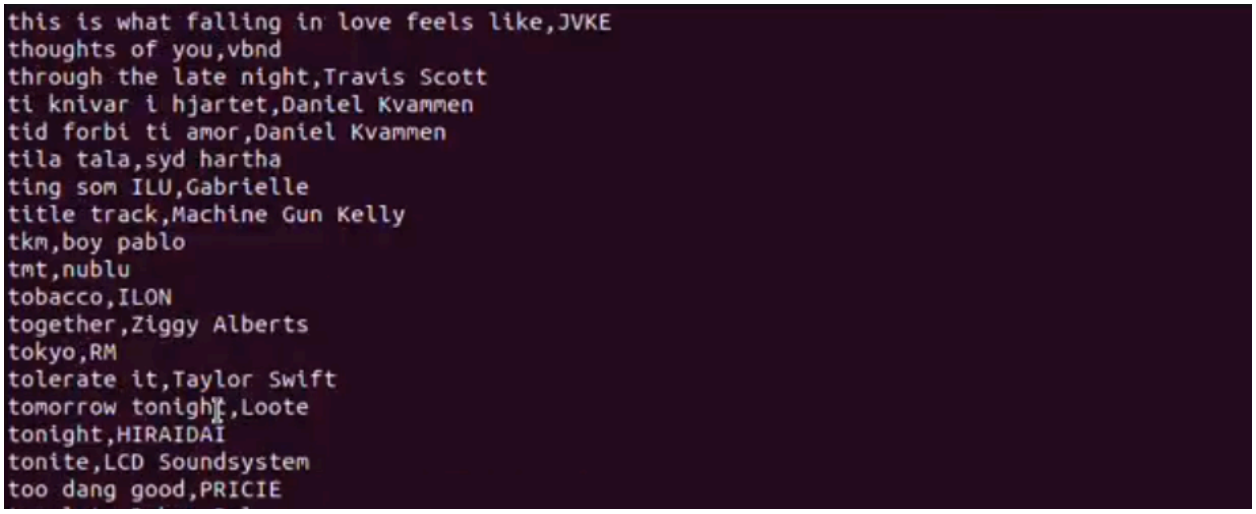
After successfully running the Map-Reduce processing framework with their corresponding mapper.py and reducer.py files, we can find an output.csv stored in the **hadoop file system:**



And if we check the content of the output.csv file using the command:

```
hadoop fs -cat /output.csv/part-00000
```

We can find the desired pieces of information stored:



4. **Storing the output.csv information into HBase database.**

So, this far we have the output.csv and an empty HBase table created after the commands:

```
Create 'songs', 'cf'
```

To import and store the output.csv we used the command:

```
bin/hbase \
  org.apache.hadoop.hbase.mapreduce.ImportTsv \
  -Dimporttsv.separator=',' \
  -Dimporttsv.columns=HBASE_ROW_KEY,cf:artist \
  songs hdfs:///output.csv/part-00000
```

```
56893 row(s)
Took 6.7742 seconds
hbase:002:0>
```

SUCCESS! Almost 57 thousand songs have been stored in the HBase table.

If we want to **retrieve all the songs from an specific artist** we can use the command:

```
scan 'songs', {FILTER => "SingleColumnValueFilter('cf', 'artist', =,
'substring:Billie Eilish')"}
```

This command scans the songs table and applies a filter to show only the rows where the cf: artist column has the value “Ed Sheeran”.

5. **Storing the output.csv data in a MySQL database**

In order to store the output.csv data into the Music database, more specifically, in the **songs table** we first copy the file into a more easily accessible directory using the command:

```
Hadoop fs -get /output.csv/part-00000 /Desktop/hadoop/output.csv
```

Then, we run the **mySQL_Loader.py** file responsible for parsing the data into the **songs** table using the command:

```
$ python3 mySQL_loader.py
```

```
hduser@dssvm:~$ cd Desktop/hadoop
hduser@dssvm:~/Desktop/hadoop$ ls
charts.csv  mapper.py  mySQL_loader.py  output.csv  reducer.py
hduser@dssvm:~/Desktop/hadoop$ python3 mySQL_loader.py
Successfully inserted 500 records into the MySQL database.
hduser@dssvm:~/Desktop/hadoop$
```

After successfully inserting the records, we can double check we actually load them into the DataBase

```
Database changed
mysql> SELECT * FROM songs;
+-----+-------------------------------------------------------+----------------------
--+
| id  | title                                                 | artist
  |
+-----+-------------------------------------------------------+----------------------
--+
|   1 | Something About You                                   | Elderbrook
  |
|   2 | A A A                                                 | PLK
  |
|   3 | A Bacalhau                                            | Ana Bacalhau
  |
```

But how did this actually happen? What was inside the mySQL_loader.py let’s dive into the script and how it helped to achieve this task

```
 GNU nano 6.2                                    mySQL_loader.py
#!/usr/bin/env python3

import mysql.connector
import csv
import re

# MySQL connection
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'password',
    'database': 'Music'
}

# Connect to the MySQL database
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()

# Prepare SQL insert statement
insert_query = "INSERT INTO songs (title, artist) VALUES (%s, %s)"

# Open the CSV file
with open('/home/hduser/Desktop/hadoop/output.csv', 'r', encoding='ISO-8859-1') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)  # Skip the header
    record_count = 0
    for row in reader:
        if len(row) == 2:
            # Used regex to replace any sequence of whitespace characters with a single space
            title = re.sub(r'\s+', ' ', row[0]).strip()
            artist = re.sub(r'\s+', ' ', row[1]).strip()
            cursor.execute(insert_query, (title, artist))

            record_count += 1
            if record_count >= 500:
                break

conn.commit()

# Close the database connection
cursor.close()
conn.close()

print(f"Successfully inserted {record_count} records into the MySQL database.")
```

Having some knowledge in python always comes useful, and in this scenario I was able to explore how to parse data and load it to the MySql created. The Script reads a list of songs and artists from output.csv and adds these songs to the Music schema that was manually created, only up to 500 songs to make things "easier" for the commercial solutions part. (RealPython, 2022)

The process of steps are:

Set Up Database Connection: The script prepares to connect to a MySQL database by setting up the necessary connection details (like the database location, username, password, and database name).

Connect to Database: It establishes a connection to the database and prepares to execute commands.

Prepare to Add Songs: It gets ready to insert new songs into the database with a specific SQL command.

Open the Song List: The script opens a CSV file that contains the list of songs and their corresponding artists.

Process Each Song: For each song in the list:

1. It reads the song title and artist name.
2. It cleans up any extra spaces in the title and artist name.
3. It adds the song to the database.
4. It keeps track of how many songs have been added and stops after 500 songs.

Save Changes and Close.

6. YCSB WORK BENCH

After installing YCSB, we can proceed to run workbench procedures in our desired database, for this exercise we run it on Hbase since all our songs were stored there
In order to run it on Hbase on our desired table, it is important to specify the table, column and number of records. (Cooper, 2014) Let's see the command:

"loader"

We can appreciate that we are loading workload a, b

```
hduser@dssvm:~/ycsb-0.17.0$
python2 ./bin/ycsb load hbase12 -P workloads/workloada -p table=songs -p columnfamily=cf -p recordcount=
10000
python2 ./bin/ycsb load hbase12 -P workloads/workloadb -p table=songs -p columnfamily=cf -p recordcount=
10000
python2 ./bin/ycsb load hbase12 -P workloads/workloadc -p table=songs -p columnfamily=cf -p recordcount=
10000
/usr/bin/java -cp /home/hduser/ycsb-0.17.0/hbase12-binding/conf:/home/hduser/ycsb-0.17.0/conf:/home/hdus
er/ycsb-0.17.0/lib/HdrHistogram-2.1.4.jar:/home/hduser/ycsb-0.17.0/lib/core-0.17.0.jar:/home/hduser/ycsb
-0.17.0/lib/htrace-core4-4.1.0-incubating.jar:/home/hduser/ycsb-0.17.0/lib/mysql-connector-java-8.0.32.j
ar:/home/hduser/ycsb-0.17.0/lib/jackson-mapper-asl-1.9.4.jar:/home/hduser/ycsb-0.17.0/lib/jackson-core-a
sl-1.9.4.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/hbase-shaded-client-1.2.5.jar:/home/hduser/ycs
b-0.17.0/hbase12-binding/lib/slf4j-log4j12-1.6.1.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/slf4j-
api-1.7.25.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/htrace-core-3.1.0-incubating.jar:/home/hduse
r/ycsb-0.17.0/hbase12-binding/lib/log4j-1.2.17.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/commons-
logging-1.2.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/findbugs-annotations-1.3.9-1.jar:/home/hdus
er/ycsb-0.17.0/hbase12-binding/lib/hbase10-binding-0.17.0.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/l
ib/hbase12-binding-0.17.0.jar site.ycsb.Client -db site.ycsb.db.hbase12.HBaseClient12 -P workloads/workl
oada -p table=songs -p columnfamily=cf -p recordcount=10000 -load
Command line: -db site.ycsb.db.hbase12.HBaseClient12 -P workloads/workloada -p table=songs -p columnfami
ly=cf -p recordcount=10000 -load
YCSB Client 0.17.0

Loading workload...
log4j:WARN No appenders could be found for logger (org.apache.htrace.core.Tracer).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Starting test.
```

and c. Which they will give different results and data to analyze, once it loads each workload without any issue, we can run the counterpart which is "run"

"runner"

```
hduser@dssvm:~/ycsb-0.17.0$ python2 ./bin/ycsb run hbase12 -P workloads/workloada -p table=songs -p colu
mnfamily=cf -p operationcount=10000
python2 ./bin/ycsb run hbase12 -P workloads/workloadb -p table=songs -p columnfamily=cf -p operationcoun
t=10000
python2 ./bin/ycsb run hbase12 -P workloads/workloadc -p table=songs -p columnfamily=cf -p operationcoun
t=10000
/usr/bin/java -cp /home/hduser/ycsb-0.17.0/hbase12-binding/conf:/home/hduser/ycsb-0.17.0/conf:/home/hdus
er/ycsb-0.17.0/lib/HdrHistogram-2.1.4.jar:/home/hduser/ycsb-0.17.0/lib/core-0.17.0.jar:/home/hduser/ycsb
-0.17.0/lib/htrace-core4-4.1.0-incubating.jar:/home/hduser/ycsb-0.17.0/lib/mysql-connector-java-8.0.32.j
ar:/home/hduser/ycsb-0.17.0/lib/jackson-mapper-asl-1.9.4.jar:/home/hduser/ycsb-0.17.0/lib/jackson-core-a
sl-1.9.4.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/hbase-shaded-client-1.2.5.jar:/home/hduser/ycs
b-0.17.0/hbase12-binding/lib/slf4j-log4j12-1.6.1.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/slf4j-
api-1.7.25.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/htrace-core-3.1.0-incubating.jar:/home/hduse
r/ycsb-0.17.0/hbase12-binding/lib/log4j-1.2.17.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/commons-
logging-1.2.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/lib/findbugs-annotations-1.3.9-1.jar:/home/hdus
er/ycsb-0.17.0/hbase12-binding/lib/hbase10-binding-0.17.0.jar:/home/hduser/ycsb-0.17.0/hbase12-binding/l
ib/hbase12-binding-0.17.0.jar site.ycsb.Client -db site.ycsb.db.hbase12.HBaseClient12 -P workloads/workl
oada -p table=songs -p columnfamily=cf -p operationcount=10000 -t
Command line: -db site.ycsb.db.hbase12.HBaseClient12 -P workloads/workloada -p table=songs -p columnfami
ly=cf -p operationcount=10000 -t
YCSB Client 0.17.0

Loading workload...
```

Once the run command finishes, the data starts coming up. You will be able to see different categories and different results, however it is hard to understand the output of all the run commands, to analyze correctly I separate each workbench result and insert them in a table:

Metric	Workload A	Workload B	Workload C
Throughput (ops/sec)	2371.35	2707.83	2934.27
Total GCs (Scavenge Count)	8	10	8
Total GC Time (ms)	27	33	25
Total GC Time (%)	0.0064	0.0089	0.0073
READ Operations	5039	9508	10000
READ Avg Latency (us)	270.24	263.18	291.15
READ Max Latency (us)	14087	18767	14799
READ 95th Percentile (us)	446	400	412
READ 99th Percentile (us)	1093	896	1270
UPDATE Operations	4961	492	0
UPDATE Avg Latency (us)	449.54	1112.41	N/A
UPDATE Max Latency (us)	39519	25567	N/A
UPDATE 95th Percentile (us)	829	2933	N/A

UPDATE 99th Percentile (us)	**3783**	**7495**	**N/A**

Analyzing Performance Variability Across Workloads A, B, and C in HBase

When benchmarking HBase with different YCSB workloads such as A, B, and C, it's amazing to observe how the same database can exhibit varying performance characteristics under different operational scenarios. The variability in performance can be attributed to several factors *(on, 2020)*:

1. Operation Mix

Each workload has a distinct mix of read and write operations, which influences the overall performance:

Workload A features a balanced mix of reads and updates (50% each). The updates add overhead due to the necessity for data durability and consistency, demonstrating moderate throughput and higher latencies for updates due to operations like log syncing and data replication.
Workload B is predominantly read-based but includes a smaller proportion of updates (95% read, 5% update). These write operations increase latency and slightly reduce throughput compared to a purely read-based workload.
Workload C, consisting solely of read operations, shows the highest throughput and consistent performance since it doesn't incur the overhead associated with write operations.

2. Data Caching and Access Patterns

HBase optimizes read access through caching mechanisms such as BlockCache and Bloom filters, which can significantly enhance performance if data is pre-loaded into memory:

Workload C benefits the most from caching mechanisms due to its exclusive focus on read operations. This consistent access pattern allows HBase to efficiently cache data, reducing disk I/O and improving read performance.
Workloads A and B, with their mixed access patterns, face periodic invalidation of cache entries triggered by write operations, which can prevent efficient caching and lead to variable performance.

3. Resource Contention and Concurrency

The inclusion of write operations introduces more lock contention and concurrency control mechanisms, which can impact performance:

Workload A and B incorporate updates that necessitate locks on data files and may involve compaction processes that can block or slow down read operations. This adds complexity and can lead to increased latency and decreased throughput.

4. System Configuration and Tuning

The default configurations of HBase may favor certain types of workloads over others, depending on system tuning, such as memstore sizes, file rolling sizes, and the configuration of the write-ahead log (WAL):

Workloads with higher write frequencies, like Workload A, might be more affected by these parameters than Workload C, which could operate more efficiently under default settings due to its read-only nature.

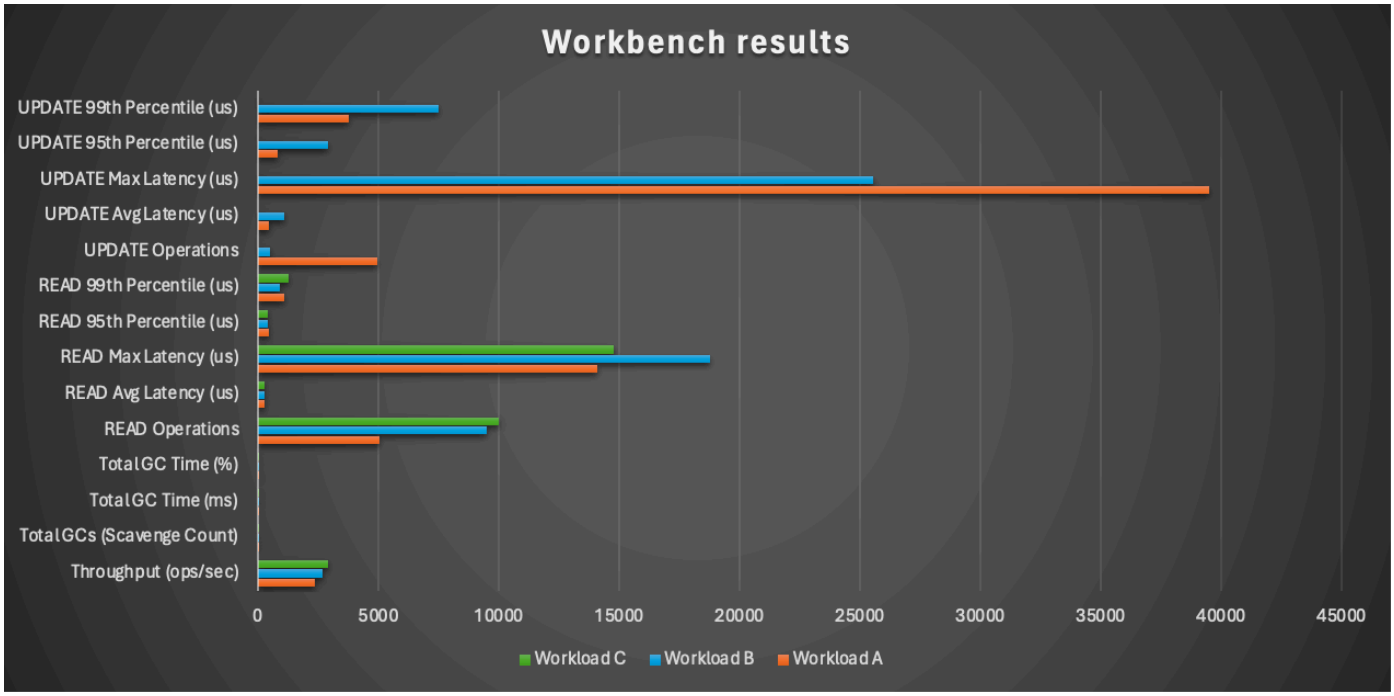
5. Garbage Collection (GC) Impact

Java's Garbage Collection impacts performance, with workloads that have frequent updates generating more temporary objects, thereby inducing more frequent GC events:

Workload C shows minimal GC impact, indicating that read operations are less taxing on memory management systems. In contrast, Workloads A and B, which include write operations, face more substantial GC overhead due to the disposal of more discarded data objects.

Conclusion

Understanding these factors is crucial for database administrators and system architects, allowing them to tailor HBase configurations to the specific needs of their applications—whether prioritizing read speed, write durability, or a balance of both. Recognizing the impact of different operation types and their interactions with database internals helps in designing more efficient data models and access patterns, ensuring that HBase deployments are robust and performant under varied operational demands.

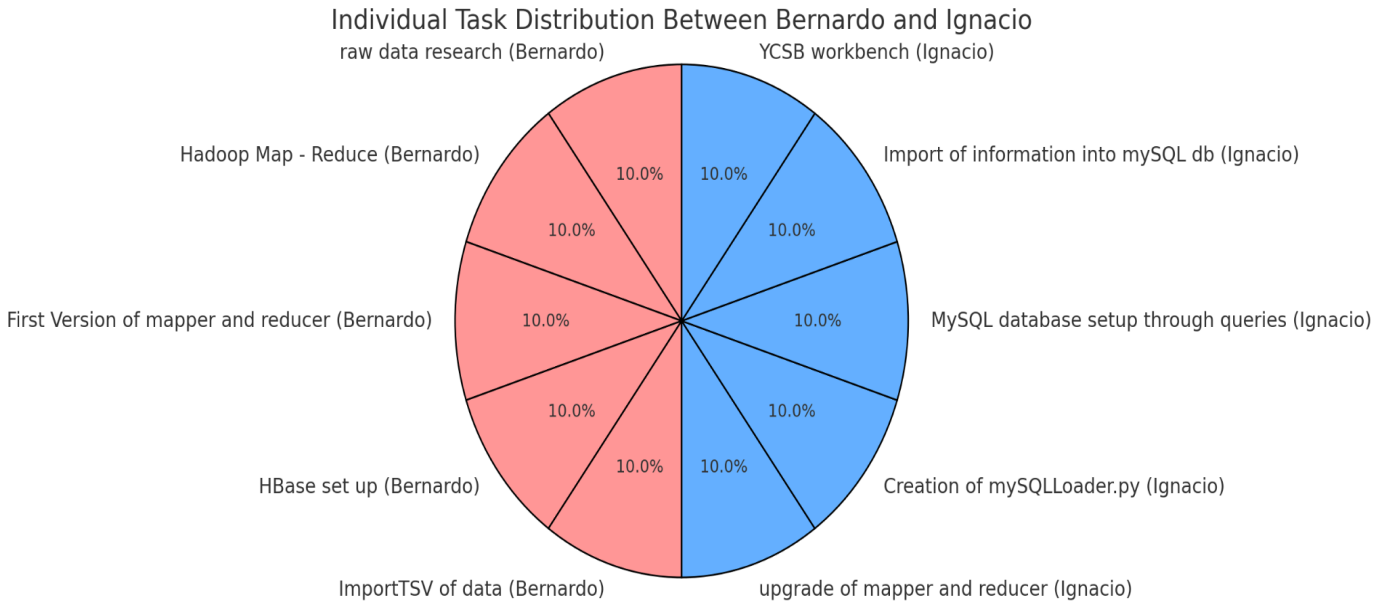


WORK DISTRIBUTION

Even though we were aligned during the whole preparation and research of each part of the assignment, at the moment of implementing the key-functionalities we did distribute the work.

Bernardo Gandara was responsible for searching the raw csv file (charts.csv), implementing the Hadoop Map-Reduce process including the first versions of the mapper.py and reducer.py that were later upgraded by Ignacio. Bernardo also did research and implement the HBase database and 'songs' table, including the ways to import the output.csv file into the database.

As mentioned previously, Ignacio was responsible for upgrading the mapper.py, reducer.py and creating the mySQLLoader.py files with the goal of making these suitable for our requirements. This includes research and implementation of migrating the output.csv into the mySQL database (created with queries using mySQL command line). Ignacio's work includes the implementation of YCSB workbench and the research regarding the performance of its different workloads.



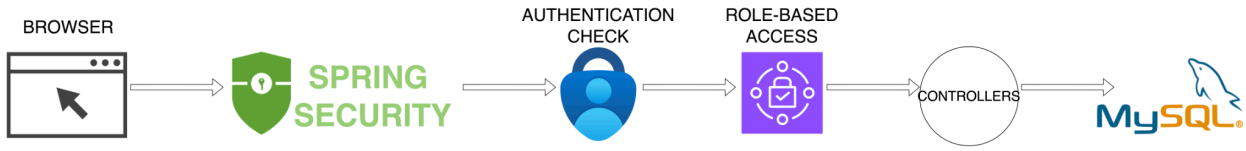
SPRING BOOT APPLICATION - Bernardo Gandara

VIDEO DEMO:
https://drive.google.com/file/d/1DyCfiOp9Ia67G243-jPCq4dHcLbCCaAZ/view?usp=sharing

Database setup: download the MusicDB_dump.sql file. Two ways to set it up:

1. **MySQLWorkbench**: Open the Music_dump.sql file in MySQLWorkbench and run it.
2. **Command Line:**
   - **a.** Get the mySQL server running using: `mysql -u [username] -p.`
   - **b.** Create a new schema using the query: "`CREATE DATABASE Music;`"
   - c. Exit the server and run "`mysql -u [username] -p Music < MusicDB_dump.sql`"

I find it very important to, before diving into the flow of the application, explain the functionalities and the logic behind the **SpringSecurity** class. As the security configuration handles requests and directs users based on their roles and authentication status.

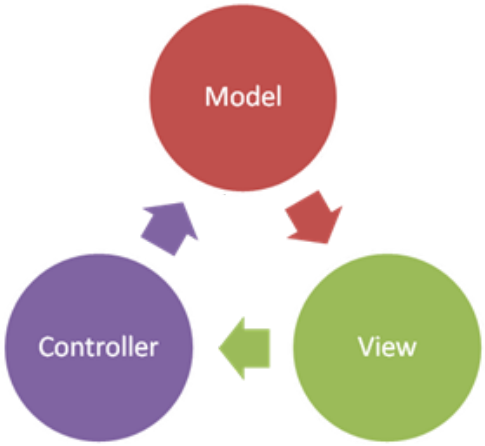


The `HttpSecurity` configuration, based on authentication and roles, defines which endpoints are accessible to the user. If a user requests to access a protected resource, such as "/songs", **Spring Security** checks if it is **authenticated**. If not, it is redirected to the login page. However, if the correct credentials are provided by the user, this is authenticated and granted access based on its role.
Now that the user is authenticated, it is routed to the appropriate controller, in the case of my application, to SongController for song-related actions (CRUD operations).
If you check my SpringSecurity class, you can see that the endpoints that are publicly accessible are '/register', '/index', '/' and '/home'. The protected endpoint is '/songs', so it requires the user to have the 'ROLE_USER'.
So, wrapping up the workflow, once the user makes a request through the browser, SpringSecurity intercepts that request and, if not authenticated, it will redirect it to '/login'. On the other hand, if it is authenticated, it checks the user role and directs accordingly (In this case there is only one single role so it will route all authenticated users to the same endpoint).

Continuing with the rationale of my application, we can establish that I have followed an **MVC architecture.** This way I can separate and classify the classes and interfaces in the following way:

MODEL

- User.java
- Role.java
- Song.java
- UserDto.java → Data Transfer Object
- RoleRepository.java → repository interface.
- SongRepository.java



- User Repository.java

CONTROLLERS

- AuthController.java → handles login and registration requests.
- SongController.java → handles song-related requests.

VIEW

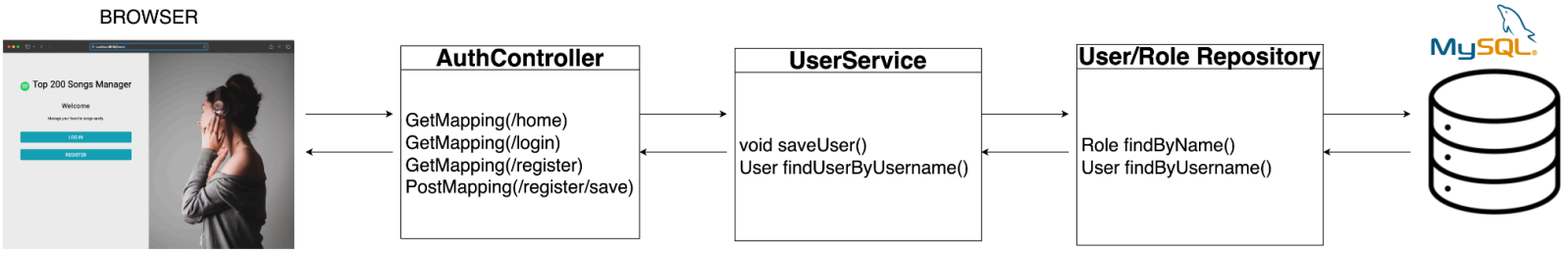
- Home.html
- Login.html
- New_song.html
- Register.html
- Songs.html
- update_song.htm

AUTHENTICATION AND REGISTRATION FUNCTIONALITY

When the user performs a request to the '/home', '/index', or '/' endpoints, the AuthController will forward him to the /home page. There the user can register (/register) or login using an existing account (/login).

The **UserDto** acts as a **Data Transfer Object** encapsulating the user registration details (username and password) to simplify the transfer of data between the client and the server.

CustomUserDetails loads the user-specific data during the authentication process. It ensures the user's credentials are retrieved from the database accurately.

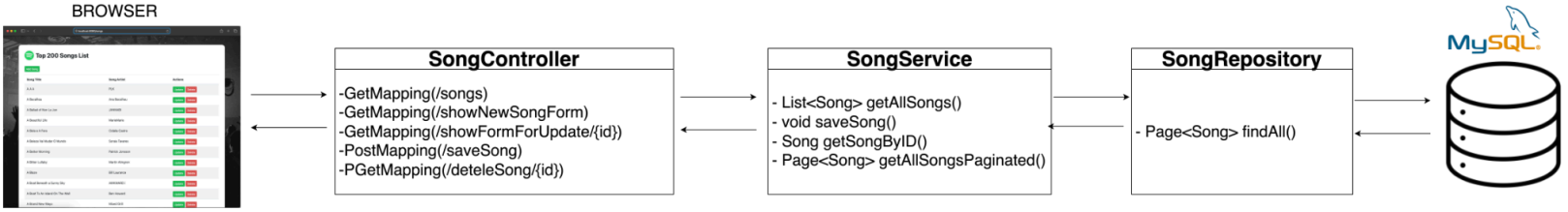


SONGS CRUD OPERATIONS FUNCTIONALITY

Once the user is authenticated and authorized, it will be forwarded to the `/songs` endpoint. The `SongController` is responsible for handling requests related to CRUD functionalities for the `Song` entity. These endpoints include:

- `/songs`: Retrieves and displays a list of all songs (Read).
- `/showNewSongForm`: Displays a form to create a new song (Create).
- `/showFormForUpdate/{id}`: Displays a form to update an existing song identified by `{id}` (Update).
- `/deleteSong/{id}`: Deletes the song identified by `{id}` (Delete).

The `SongService` layer manages the business logic, interacting with the `SongRepository` to perform database operations. This ensures a separation of concerns and promotes a clean architecture.



Upcoming features

During the following weeks I will try to add certain functionalities with the objective of making the application more user-friendly. The ones prioritized are adding **OAuth authentication** (through GitHub and Google) to make an easier registering process, and adding a **search bar** to search songs by artist or by title. The challenge will be implementing these features with the use of microservices.

We have really enjoyed the classes this semester and we hope this assignment reflects that joy. Thank you very much.

SPRING BOOT APPLICATION - Ignacio

Demo:
https://drive.google.com/file/d/1LL7pRvAA3VUKCajMLD8ymhVU1E6eLZQO/view?usp=sharing

This application starts by having great lectures and resources that impulse students to go further, or at least in my case it did. I tried making an effort and hopefully can be noticed.
The application is organized into several key packages, each serving a distinct role in the application's functionality:

- **Model**: Contains the entity classes (**User**, **Song**, **Role**) that represent the data model.
- **Repository**: Interfaces for data access operations, extending **JpaRepository** for CRUD functionalities on the models.
- **Service**: Contains business logic and service layer functionalities including transactional data handling.
- **Controller**: Handles incoming web requests and interacts with services to process business operations and return responses.
- **Security**: Configures security settings for user authentication and authorization.
- **DTO**: Data Transfer Objects used to encapsulate data and send it from the server to clients.
- **Config**: Includes configurations for the entire application, such as security configurations and any other Spring configuration.
- **Resources**: Static and template resources used in the application. Static resources include images, while templates contain Thymeleaf HTML templates for the UI.

Key Functionalities

- **User Registration and Authentication**: Users can register and log into the system. The authentication is handled through Spring Security with encrypted passwords.
- **Song Management**: Users can add, update, delete, and view songs. This includes a detailed view showing all songs paginated.
- **Search Functionality**: Users can search for songs by title or artist, which dynamically filters songs based on the input. *(Spring, 2023)*

Technical Challenges and Solutions

- **Pagination**: Implementing pagination was a significant challenge due to the need to handle large volumes of song data efficiently. The solution was integrated into the **SongRepository** using Spring Data's **Pageable** interface to fetch data in chunks, thereby enhancing performance and user experience. *(Mike. C, 2023)*
- **Dynamic Search**: Integrating a search functionality that works with pagination presented a challenge, particularly in maintaining state across different pages. This was achieved using Spring Data JPA's **@Query** annotation to create flexible queries that could filter songs based on user input without reloading the entire database. *(Spring, 2023)*
- **Styling Consistency**: Ensuring a consistent style and responsive design across all pages was crucial for providing a seamless user experience. This was managed using Bootstrap and custom CSS that matched the vibes of Spotify's branding.

Reference list

hadoop.apache.org. (2022). *MapReduce Tutorial*. [online] Available at: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html [Accessed 10 May 2024].

www.kaggle.com. (n.d.). *Spotify Charts*. [online] Available at: https://www.kaggle.com/datasets/dhruvildave/spotify-charts.

Cooper, B. (2014). *YCSB/asynchbase/README.md at master · brianfrankcooper/YCSB*. [online] GitHub. Available at: https://github.com/brianfrankcooper/YCSB/blob/master/asynchbase/README.md [Accessed 24 May 2024].

on, H.S. (2020). *HBase Performance testing using YCSB*. [online] Cloudera Blog. Available at: https://blog.cloudera.com/hbase-performance-testing-using-ycsb/.

Python, R. (2022). *Python and MySQL Database: A Practical Introduction – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-mysql/.

Code, M. (2023). *Pagination with Spring Data JPA|Spring Boot*. [online] Medium. Available at: https://medium.com/@mikecode/pagination-with-spring-data-jpa-spring-boot-b89c48ce723d [Accessed 24 May 2024].

Spring (2023). *Query by Example :: Spring Data Redis*. [online] docs.spring.io. Available at: https://docs.spring.io/spring-data/redis/reference/redis/redis-repositories/query-by-example.html [Accessed 24 May 2024].