|  |
| --- |
|  |
| CouchDB Project |
| Ivan Segade Carou |

|  |
| --- |
| STUDENT Ivan Segade Carou T00219357  10-27-2023 |

Contents

[1. Introduction 2](#_Toc149296381)

[2. Dataset and CURL command 3](#_Toc149296382)

[2.1. Dataset description 3](#_Toc149296383)

[2.2. CURL command 4](#_Toc149296384)

[2.2.1. GET 4](#_Toc149296385)

[2.2.2. POST 4](#_Toc149296386)

[2.2.3. PUT 5](#_Toc149296387)

[2.2.4. DELETE 6](#_Toc149296388)

[2.3. Discussion 7](#_Toc149296389)

[3. MapReduce and Mango Query 8](#_Toc149296390)

[3.1. MapReduce 8](#_Toc149296391)

[3.1.1. Find one document 8](#_Toc149296392)

[3.1.2.. Range of documents 9](#_Toc149296393)

[3.1.3.Count 11](#_Toc149296394)

[3.1.4. Sum 12](#_Toc149296395)

[3.1.5. Stats 14](#_Toc149296396)

[3.1.6. Group Level 15](#_Toc149296397)

[Figure 3.1.6.b. Output 16](#_Toc149296398)

[Mango Query 17](#_Toc149296399)

[3.1.2. Find a document 17](#_Toc149296400)

[3.1.3. Range of documents 17](#_Toc149296401)

[3.2. Comparison 18](#_Toc149296402)

[4. Drivers for working with CouchDB 19](#_Toc149296403)

[4.1. functionality 19](#_Toc149296404)

[4.2. Support 20](#_Toc149296405)

[4.3. Maintenance/Longevity 20](#_Toc149296406)

[4.4. Scalability 21](#_Toc149296407)

[4.5. Decision 21](#_Toc149296408)

[5. Replication and offline application 22](#_Toc149296409)

[5.1. Replication 22](#_Toc149296410)

[5.2. Offline application 25](#_Toc149296411)

[5.2.1. Creating a document. 25](#_Toc149296412)

[5.2.2. Reading a document 26](#_Toc149296413)

[5.2.3. Updating a document 27](#_Toc149296414)

[5.2.4. Deleting a document 29](#_Toc149296415)

[Conclusion 31](#_Toc149296416)

[6. Working with Capella 32](#_Toc149296417)

[Appendix 1 33](#_Toc149296418)

[Appendix 2 34](#_Toc149296419)

# Introduction

How to work with CouchDB (cluster of unreliable commodity hardware) is explored. There are 5 sections that explain how this software works under the following topics:

Dataset and CURL command.

MapReduce and Mango Query

Python libraries

Replication on CouchDB and an offline application.

Working with Capella

The soccer player dataset was used on the project. provides the link to access the original dataset and how was imported on the database. A description on how to use the CURL command to run the crud operations is detailed.

The MapReduce and Mango Query and their use is described. For this purpose, there are screenshots for each of the points required on the spec file. Moreover, there is a comparison between the ways to create views on CouchDB.

The two Python libraries that can work with CouchDB. Moreover, there is an screenshot showing how one library retrieves data from the database.

How the replication works on CouchDB is described. Moreover, there is a demonstration how an offline JavaScript application works with the database along with the corresponding screenshots.

# Dataset and CURL command

## Dataset description

A CSV file was downloaded with FIFA players from several countries and different divisions. The web page provides files from several years (2015 to 2020), however for the project only one file is needed. The year chosen was the year 2020. The file name is ‘players\_20.csv’ which contains 105 fields which are listed::

sofifa\_id,player\_url,short\_name,long\_name,age,dob,height\_cm,weight\_kg,nationality,club,overall,potential,value\_eur,wage\_eur,player\_positions,preferred\_foot,international\_reputation,weak\_foot,skill\_moves,work\_rate,body\_type,real\_face,release\_clause\_eur,player\_tags,team\_position,team\_jersey\_number,loaned\_from,joined,contract\_valid\_until,nation\_position,nation\_jersey\_number,pace,shooting,passing,dribbling,defending,physic,gk\_diving,gk\_handling,gk\_kicking,gk\_reflexes,gk\_speed,gk\_positioning,player\_traits,attacking\_crossing,attacking\_finishing,attacking\_heading\_accuracy,attacking\_short\_passing,attacking\_volleys,skill\_dribbling,skill\_curve,skill\_fk\_accuracy,skill\_long\_passing,skill\_ball\_control,movement\_acceleration,movement\_sprint\_speed,movement\_agility,movement\_reactions,movement\_balance,power\_shot\_power,power\_jumping,power\_stamina,power\_strength,power\_long\_shots,mentality\_aggression,mentality\_interceptions,mentality\_positioning,mentality\_vision,mentality\_penalties,mentality\_composure,defending\_marking,defending\_

The link to access the original file is:

https://www.kaggle.com/datasets/stefanoleone992/fifa-20-complete-player-dataset

To import the dataset into CouchDB the following command was used:

cat players\_20.csv | couchimport --url http://admin:admin@127.0.0.1:5984 --db teams --delimiter ","

## CURL command

### GET

The CURL command is used to retrieve one specific document. For this purpose, the GET method sends the URL with the corresponding \_id value of that document attached to it. After running the command, the whole document is displayed on the screen.

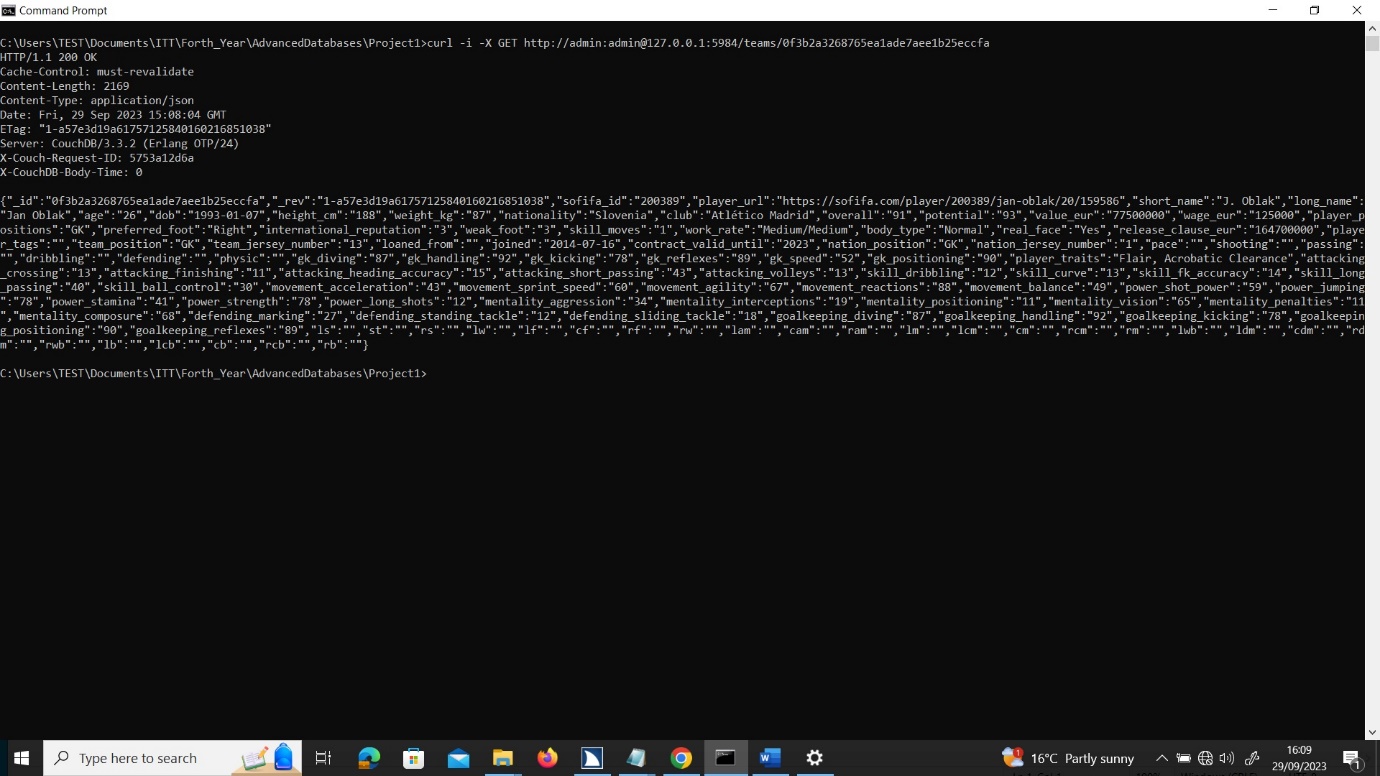


Figure 2.2.1 GET command

### POST

The CURL command is used to insert one document on the database. For this purpose, the POST method sends a JSON object with all the data of the new document on the -d parameter. The -H parameter specifies that the object sent is a JSON object. After running the command, the document is displayed along with the “OK, True” message that confirms that the process succeeded.

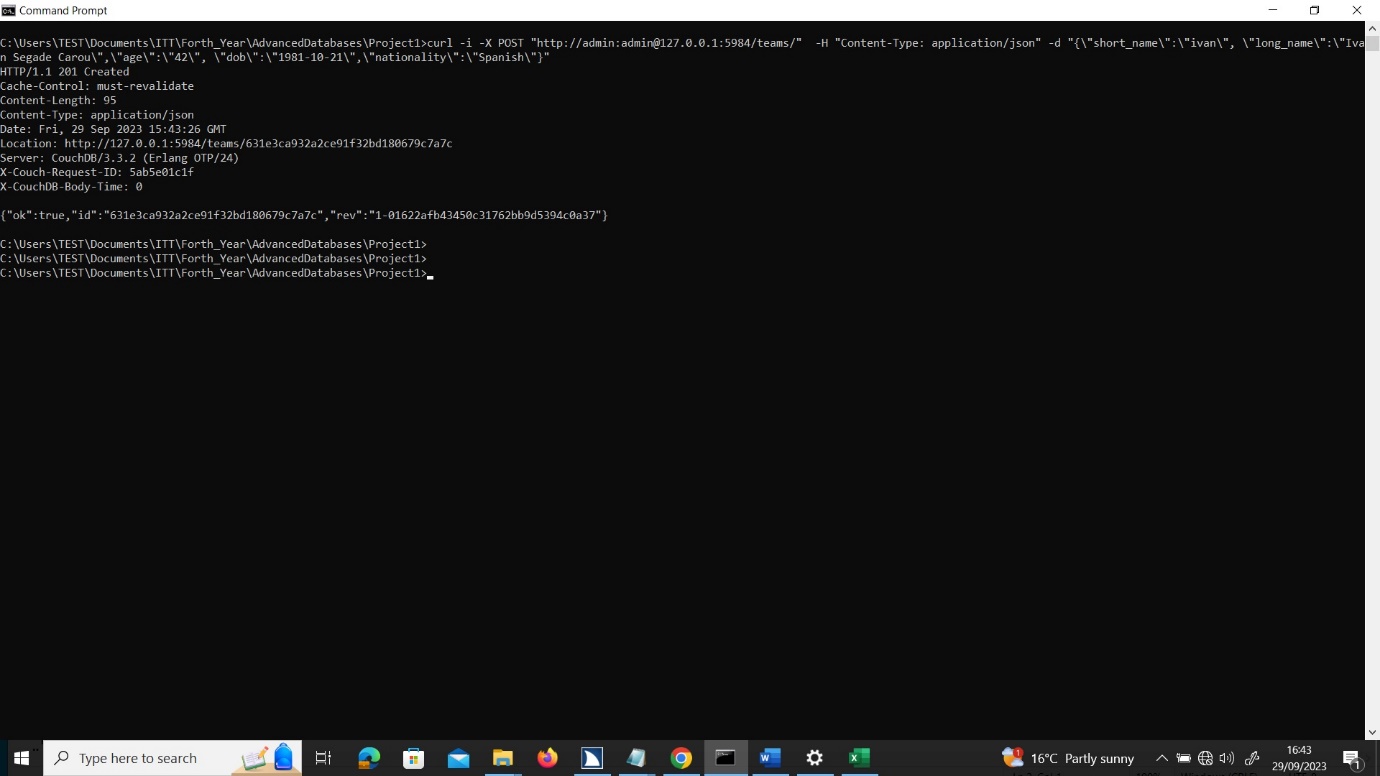


Figure 2.2.2. POST command

### PUT

The CURL command is used to update one document on the database. For this purpose, the PUT method sends a JSON object with all the new document data on the -d parameter. The -H parameter specifies that the object sent is a JSON object. After running the command, the document is displayed along with the “OK, True” message that confirms that the process succeeded.

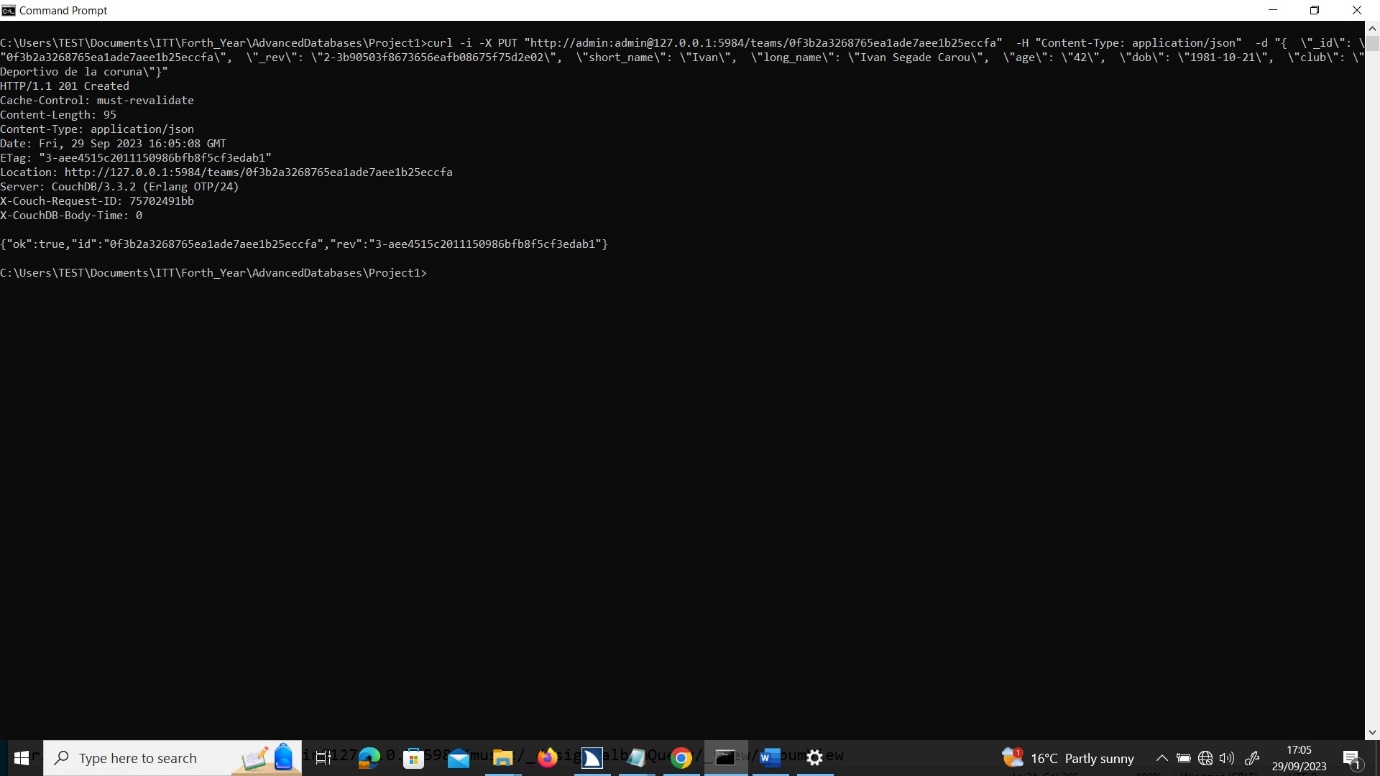


Figure 2.2.3. PUT command

### DELETE

The CURL command is used to delete one specific document. For this purpose, the DELETE method sends the URL with the corresponding \_id value of that document attached. The -H parameter specifies that the id sent has to match with the \_id value on the data base with the “If-Match” and the document id. After running the command, The “OK, True” message is displayed on the screen to confirm that the process succeeded.

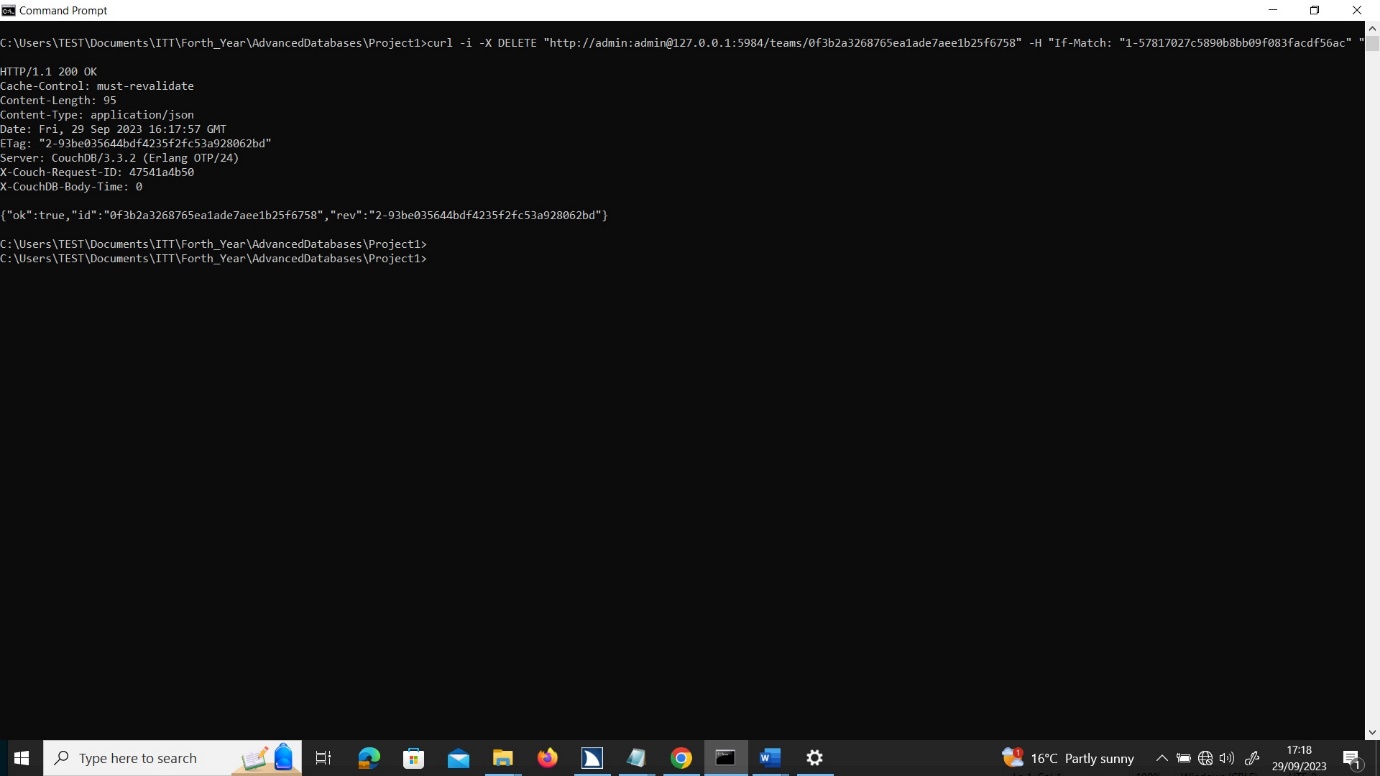


Figure 2.2.4. DELETE command

## Discussion

The use of the CURL command is outlined on the CouchDB web site (CouchDB, 2023)

The cURL command perform the crud operation through the command line. With this command, users can work with CouchDB without need of additional code. The command may also performs other useful operations, such as working with views. However, the cURL command cannot create databases as well as replications.

So the cURL command is useful for basic operations, but it does not has complete CouchDB. Therefore, tools such as Fauxton are necessary for in dept use.

# MapReduce and Mango Query

## MapReduce

### Find one document

A MapReduce function is used to find one specific document. For this purpose, the code checks that the long\_name field exists on the document, otherwise the code will crash. Then, the code emits the \_id and long\_name fields. After saving and running the function, the \_id and the long\_name fields of the specific document are displayed.

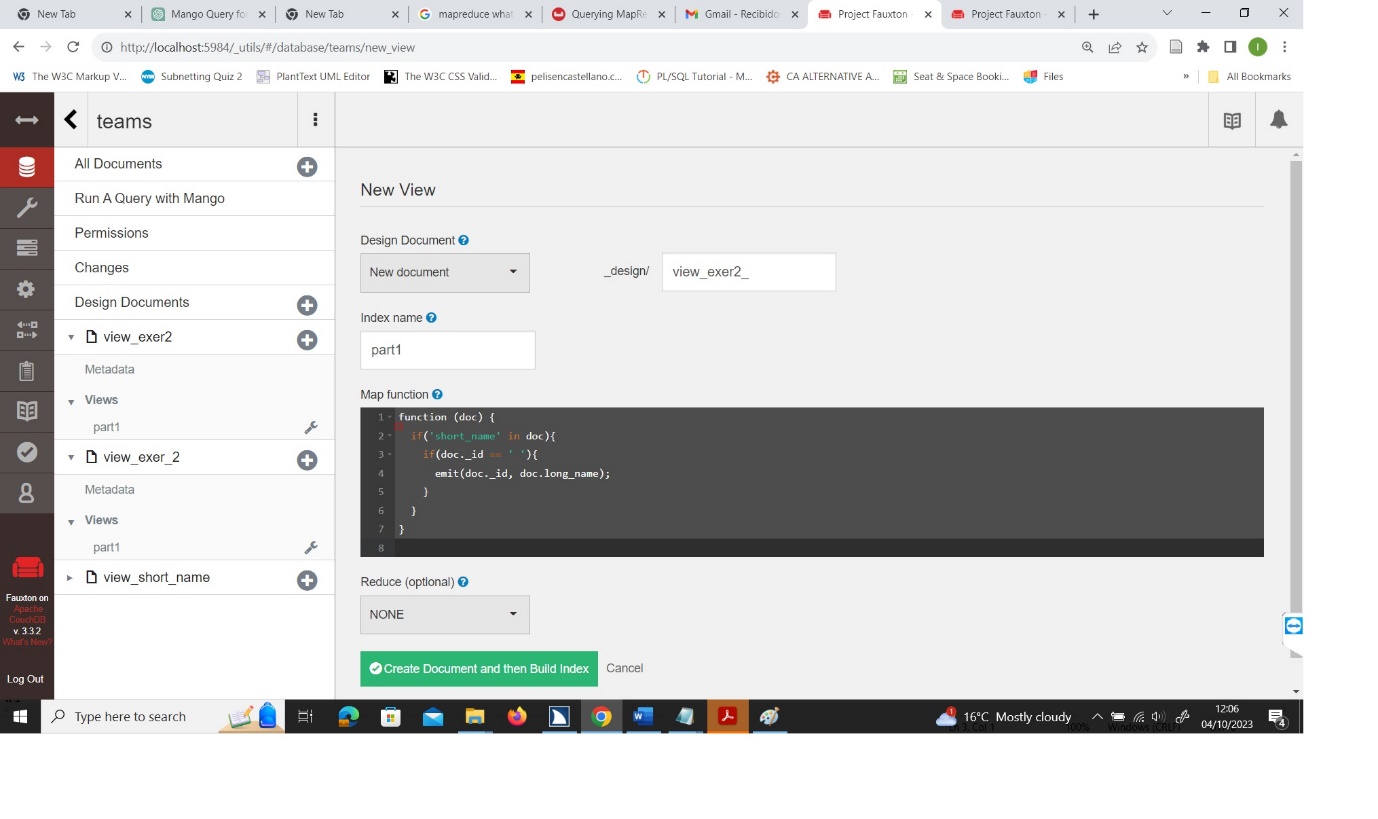


Figure 3.1.1.a. input of “find a document”

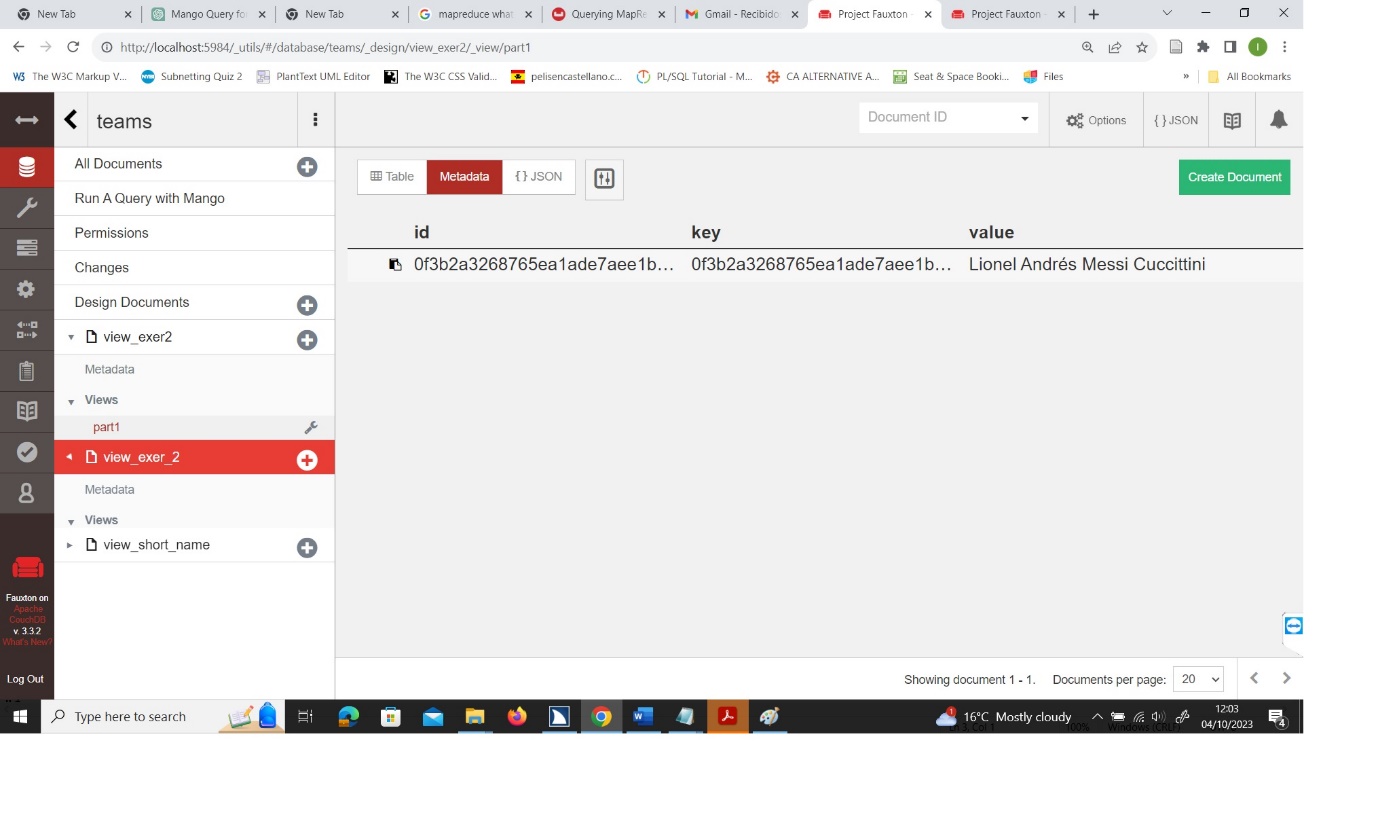


Figure 3.1.1.b output of “find a document”

### 3.1.2.. Range of documents

A MapReduce function is used to find a range of documents, where the age is >= 25 and < 30. For this purpose, the code checks if the long\_name and age fields exist on the documents for avoiding crashes. Then, the code adds the ‘if’ statement with the conditions and emits the -id, long-name and age values. After saving and running the function, the \_id and long\_name fields of all the documents that fulfil the conditions.

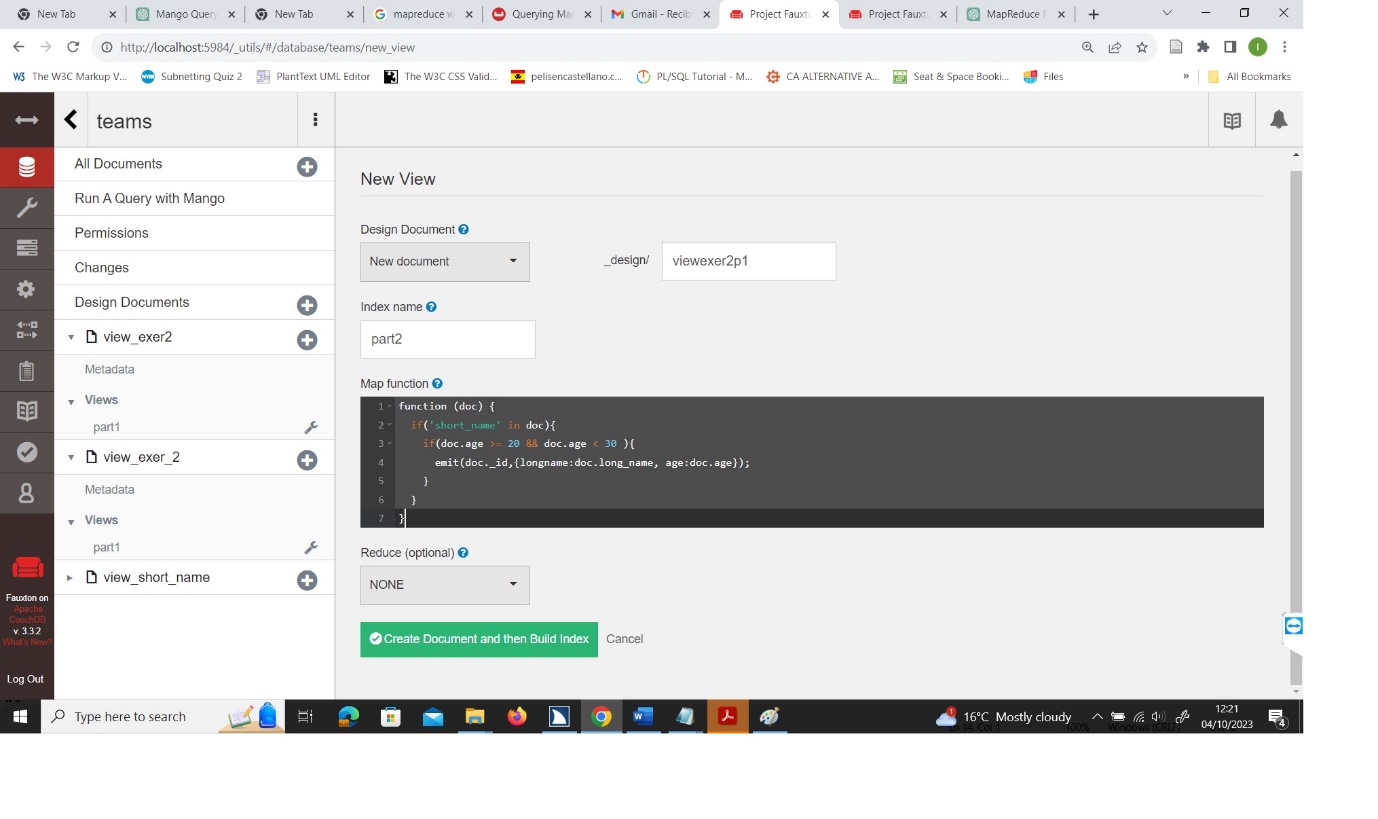


Figure 3.1.2.a. input of “range of documents”

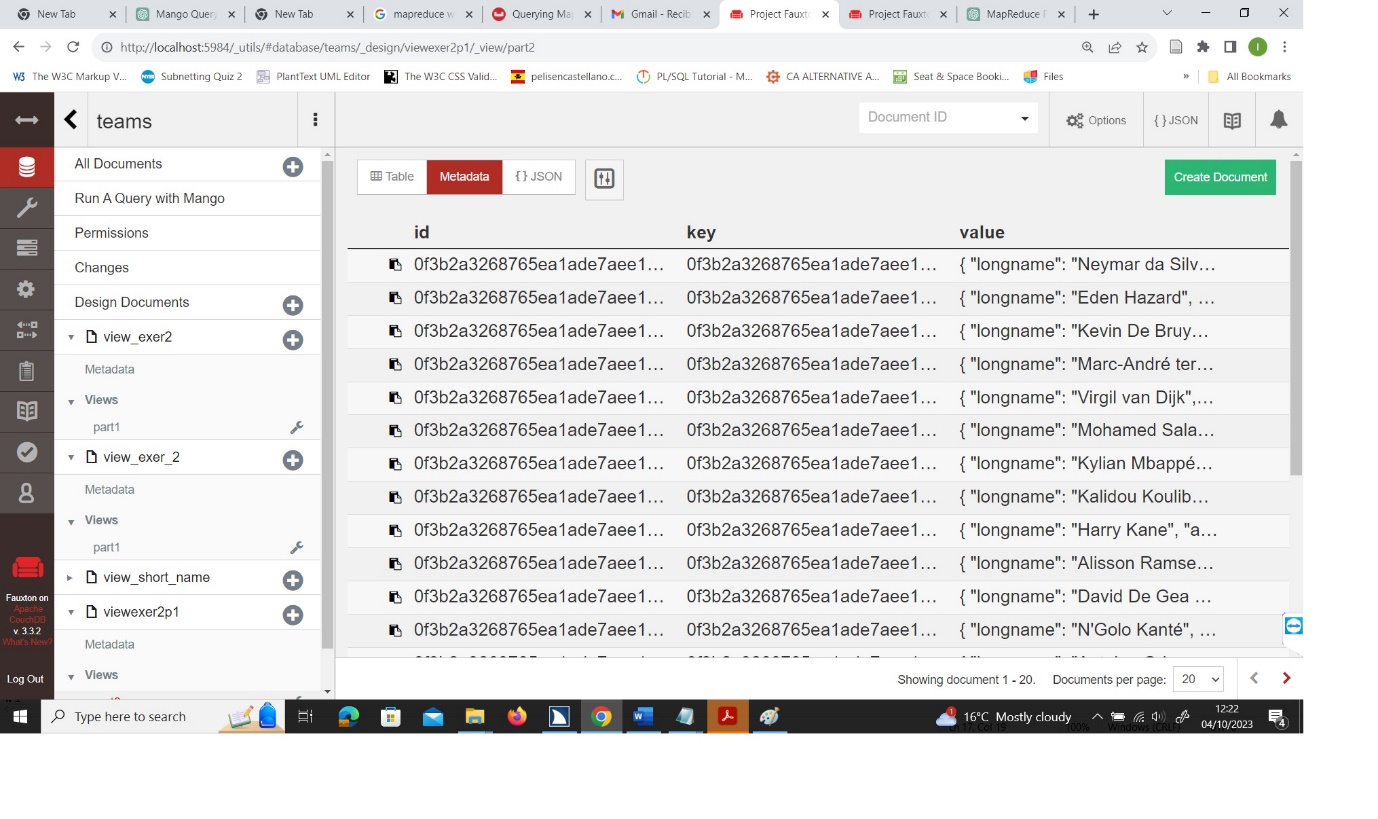


Figure 3.1.2.b. output of “range of documents”

### 3.1.3.Count

A MapReduce function is used to count the number of Spanish players on the file. For this purpose, the code adds the “if” condition (doc.nationality == ‘Spain’). Then the code emits the nationality and 1 value number for each one. After customizing the view with the \_sum option, the function is saved and run. Then the reduce option must be activated in order to sum all the 1 values up. As a result, the total number of Spanish players on the database is displayed.

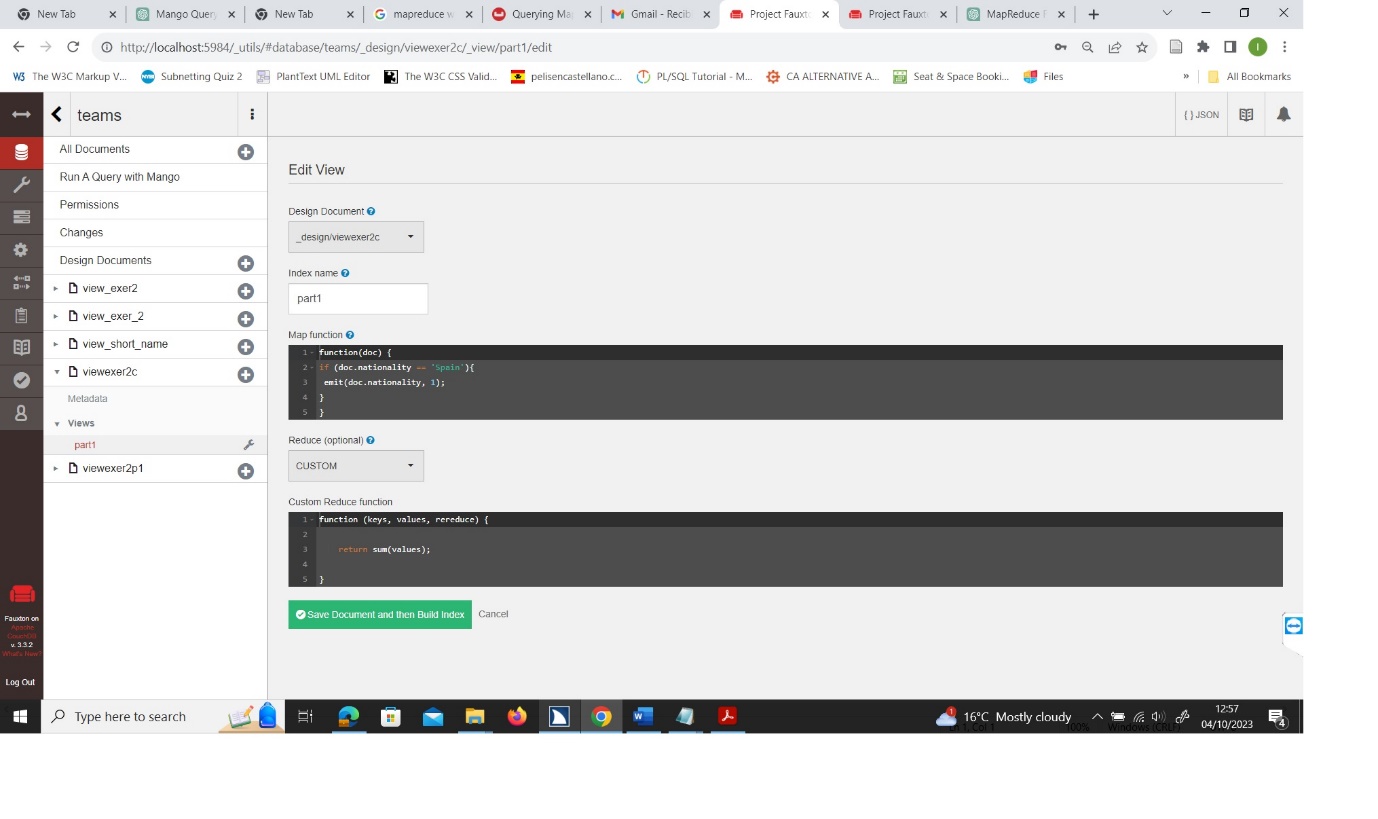


Figure 3.1.3.a. Input

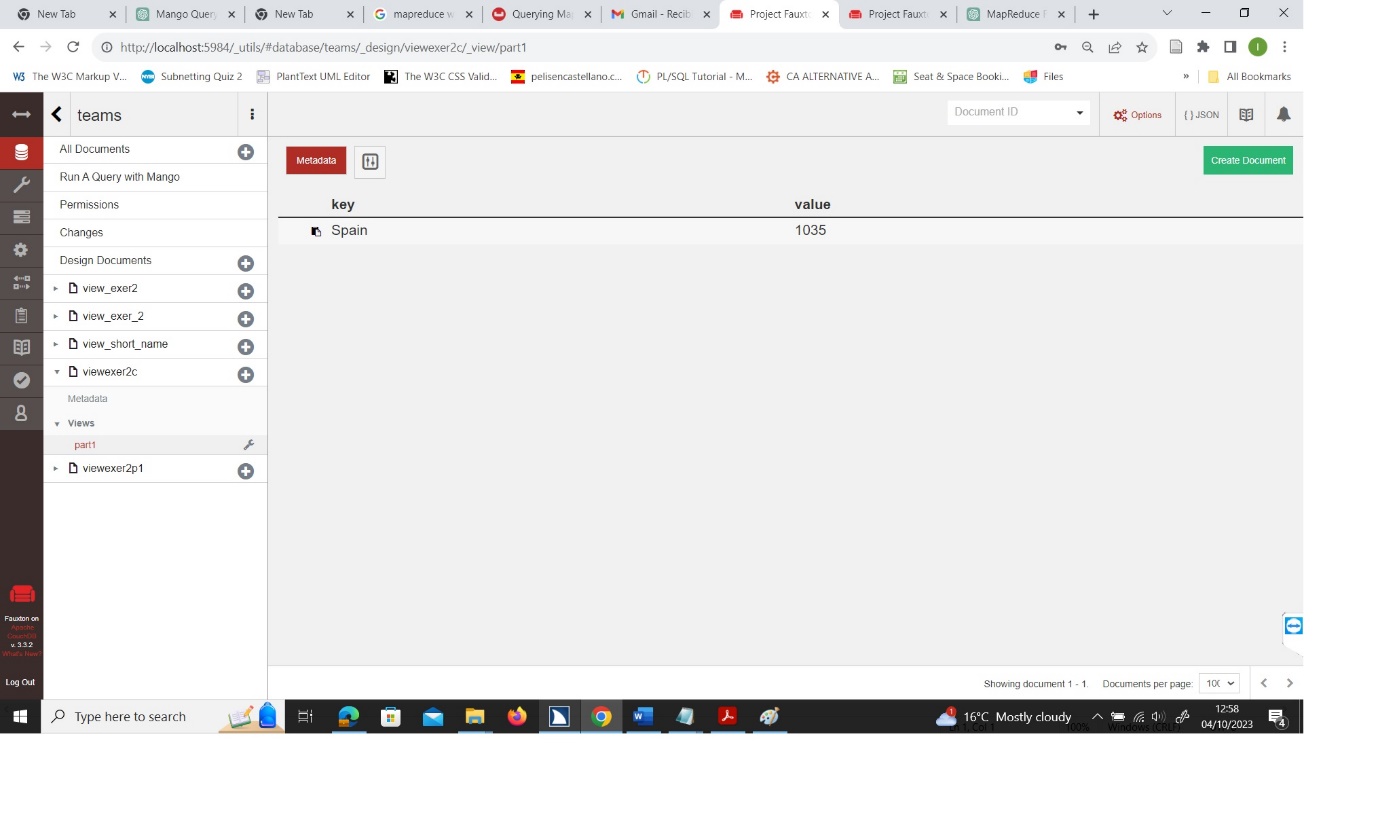


Figure 3.1.3.b. Output

### 3.1.4. Sum

A MapReduce function is used to sum the ages of all Spanish players in the Real Madrid football club. For this purpose, the function contains the two “if” conditions (doc.club =='Real Madrid' && doc.nationality == 'Spain').Then the code emits the ‘age’ field. After customizing the view with the \_sum option, the function is saved and run. Then, the reduce option must be activated in order to be displayed.

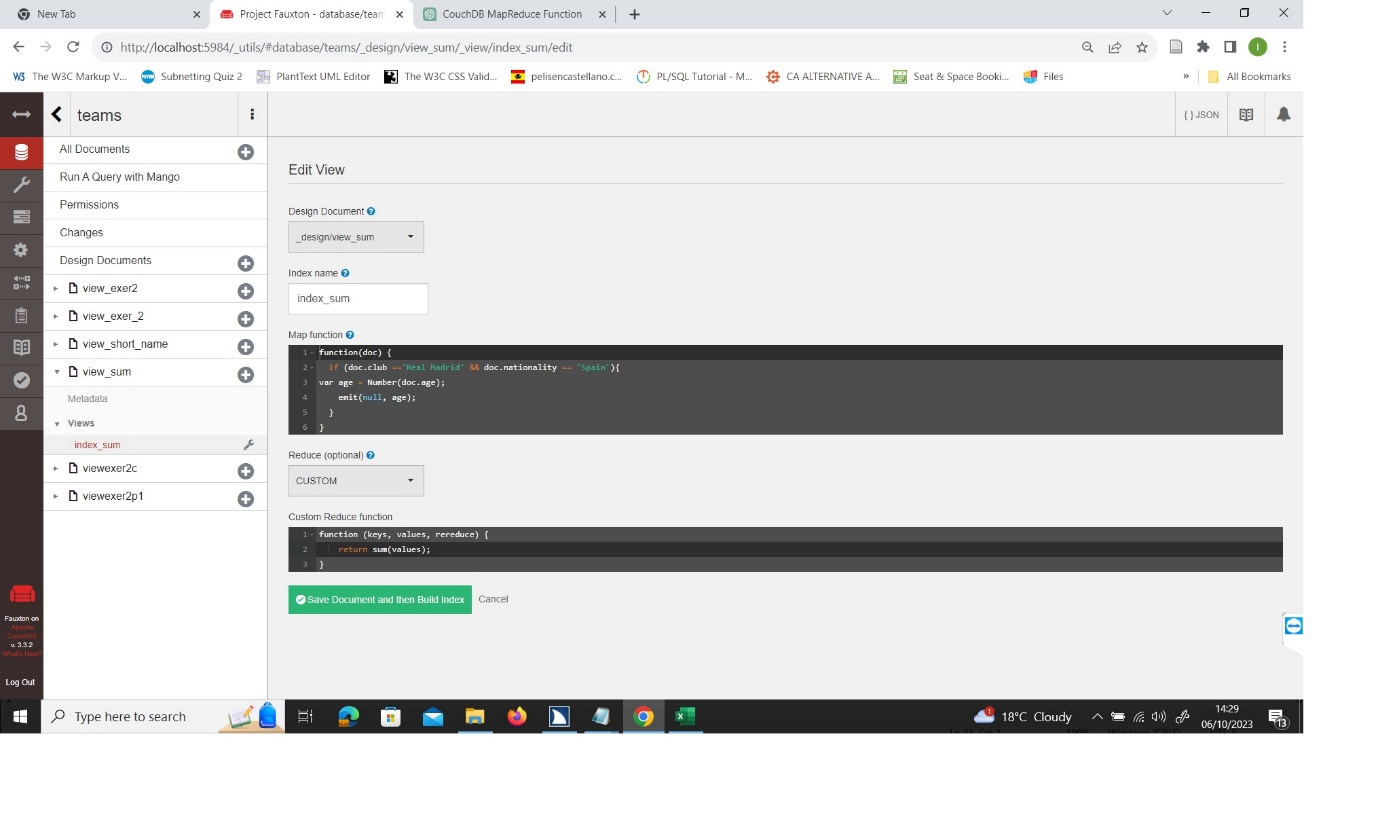
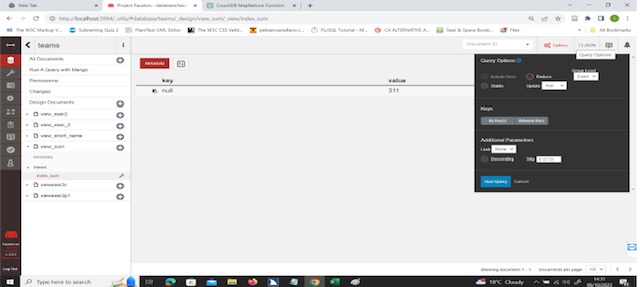


Figure 3.1.4.a. Input.

Figure 3.1.4.b. Output.

### 

### 3.1.5. Stats

A MapReduce function is used to display some stats figures related to the age of all Real Madrid players. For this purpose, the function contains the “if” condition (doc.club =='Real Madrid').Then the code emits the ‘age’ field. After customizing the view with the \_stats option, the function is saved and run. Then, the reduce option must be activated in order to be displayed.

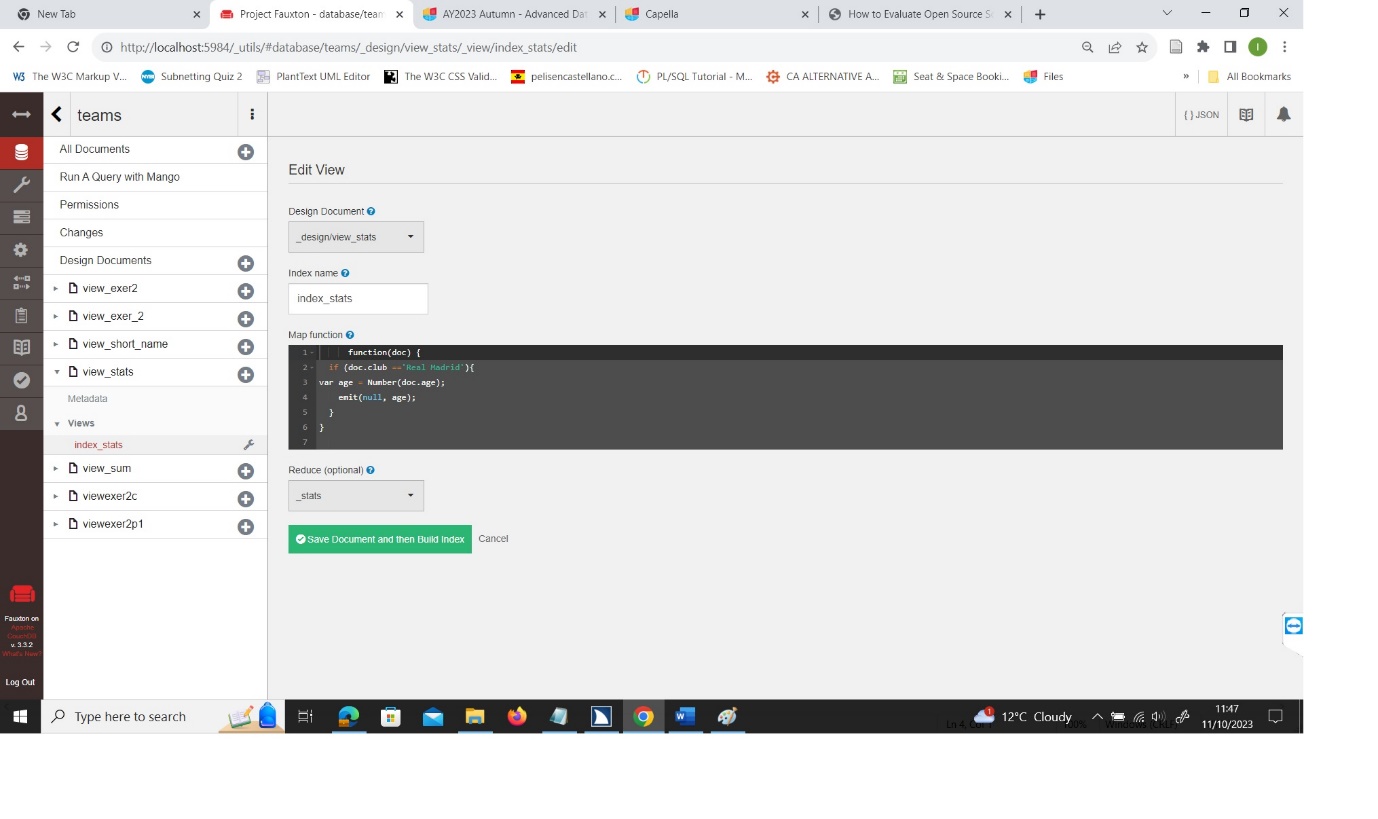


Figure 3.1.5.a. Input

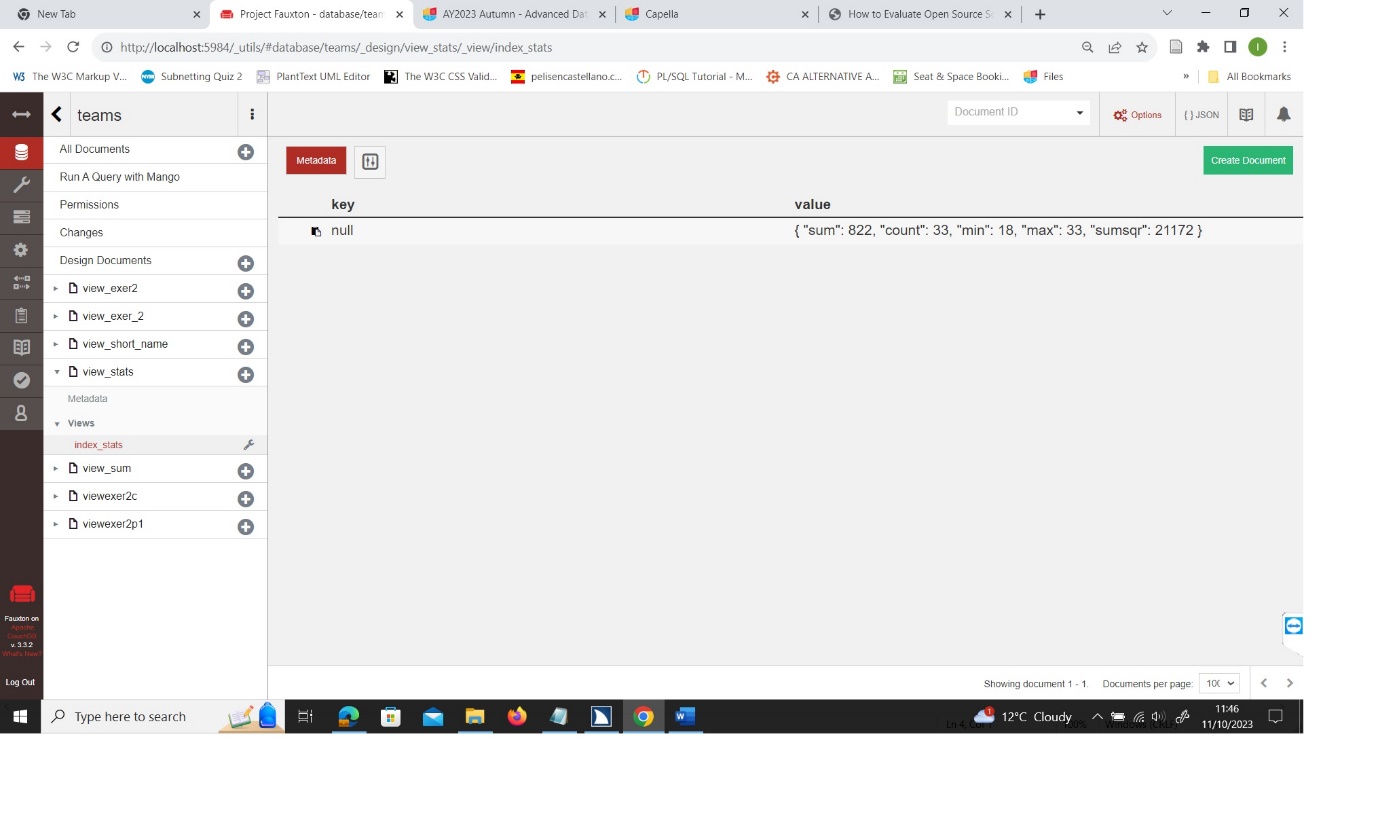


Figure 3.1.5.b. Output

### 3.1.6. Group Level

A MapReduce function is used to display grouped documents. For this purpose, the first parameter of the emit statement is set with an array of the keys to group. The second parameter is the list of fields to be displayed. In this case the documents are grouped first by club and then by nationality.

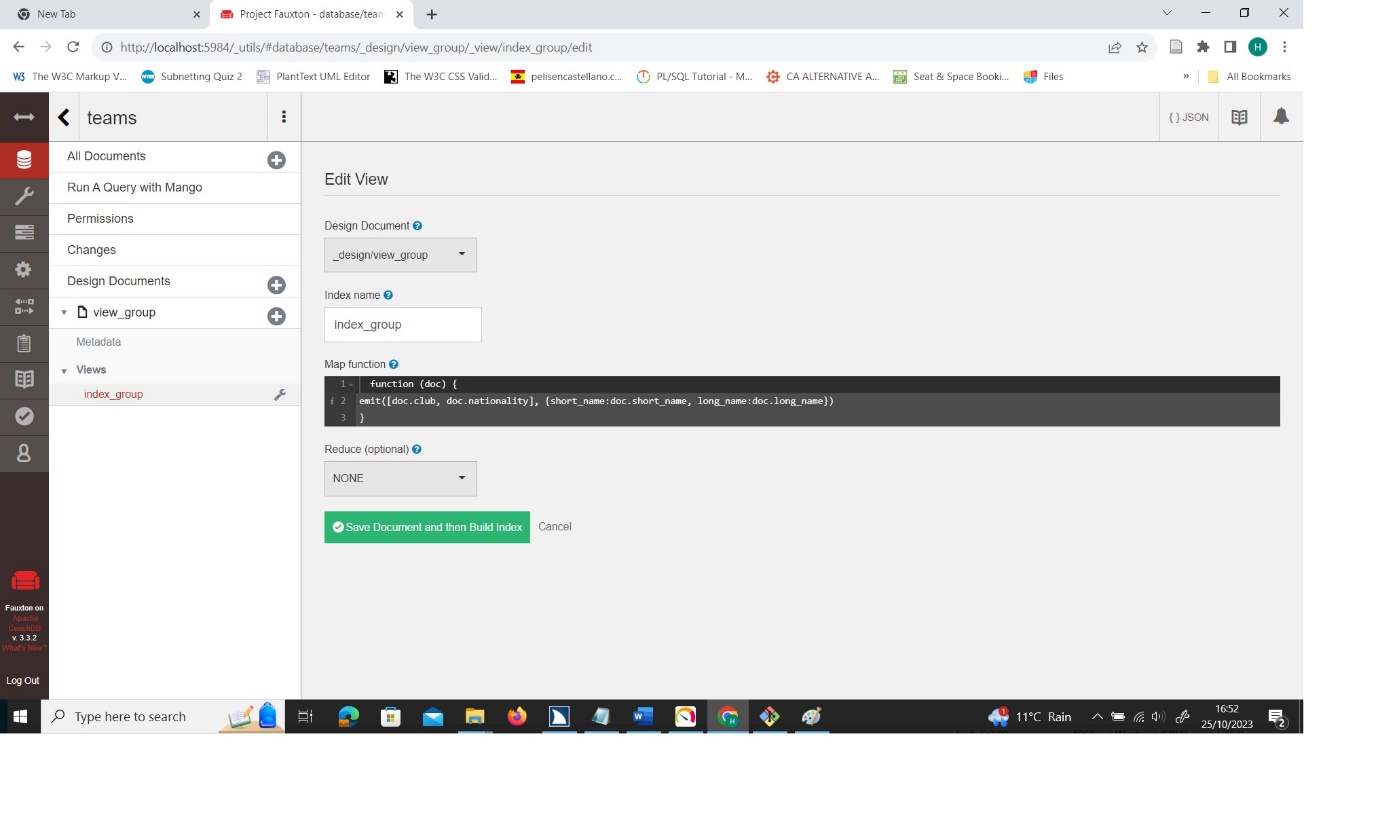
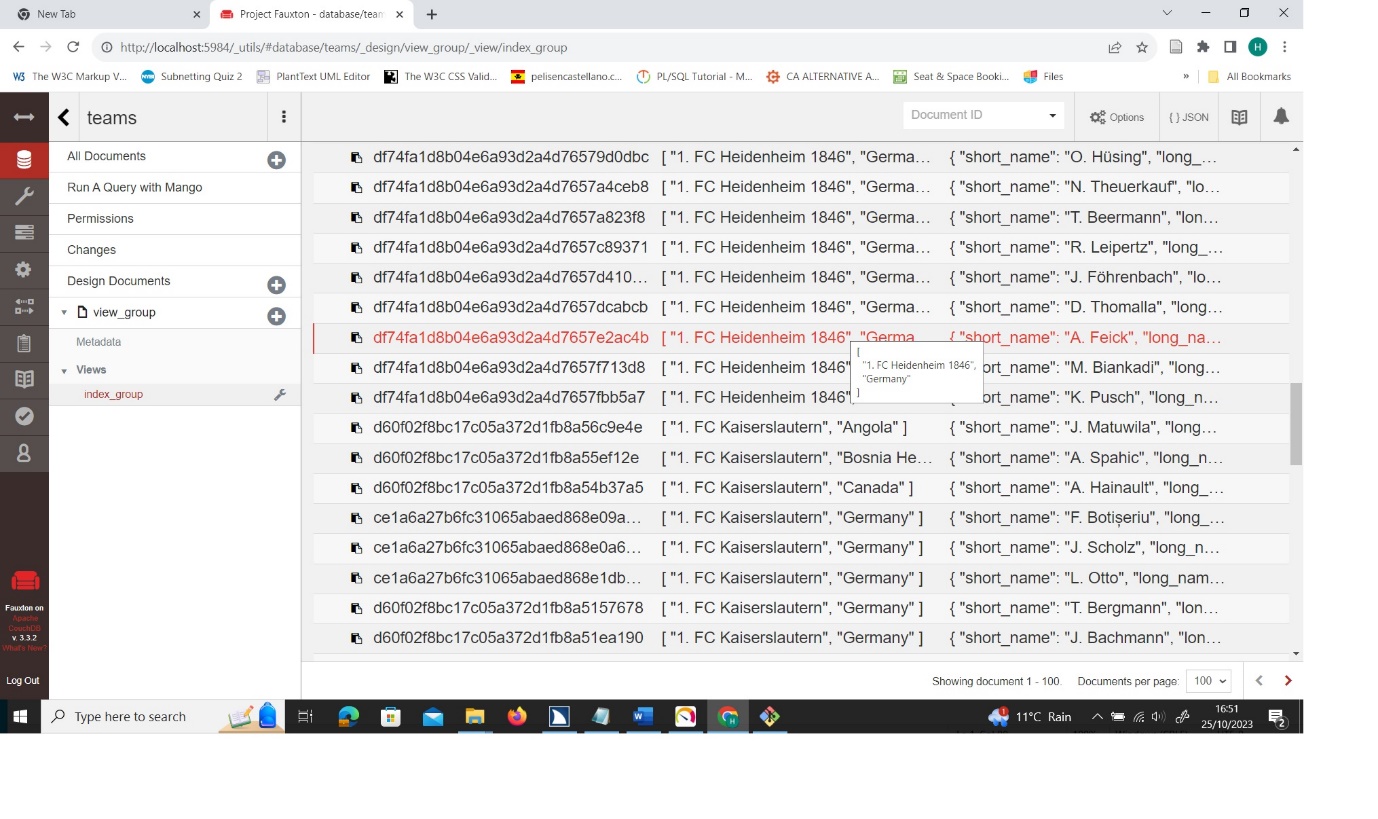


Figure 3.1.6.a. Input



## Figure 3.1.6.b. Output

## Mango Query

### Find a document

A Mango query is used to find a document. For this purpose, the “selector” key is used with the condition of the -id key must be equal to the provided id value. Also, the “fields” key is added with the \_id, short\_name and long\_name fields that will be displayed by the code. After saving and running the query, the specific fields of the player are shown.

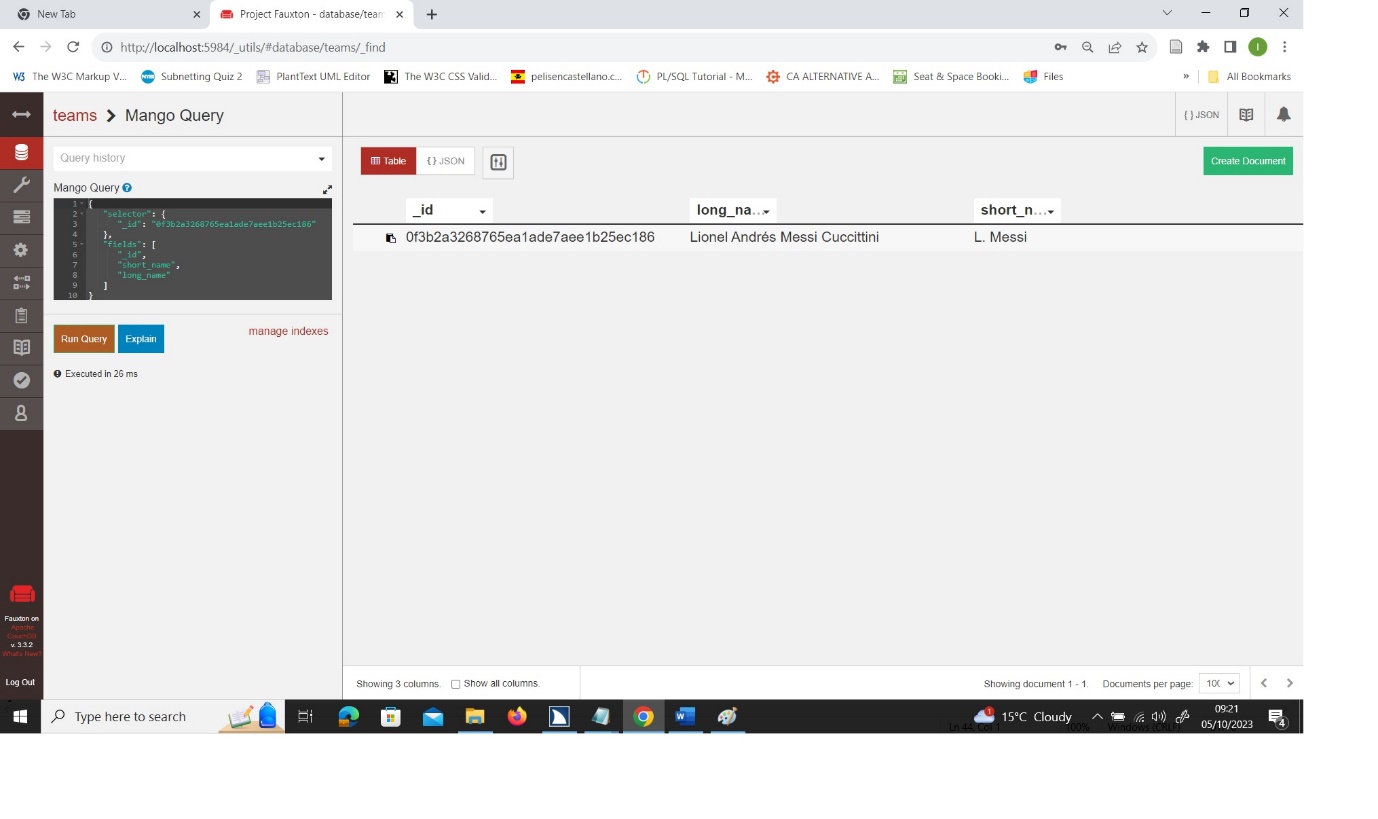


Figure 3.2.1. find a document

### Range of documents

A Mango query is used to find a range of documents, where the age is >= 25 and < to 30. For this purpose, the “selector” key is used with the conditions. To set the conditions, the $gte (greater than equal) and $lt (less than) keys are used to set the conditions. Also, the “fields” key is added with \_id, long\_name and age fields that will be displayed by the code. After saving and running the query, all the specific fields of the players that fulfil the conditions are shown.

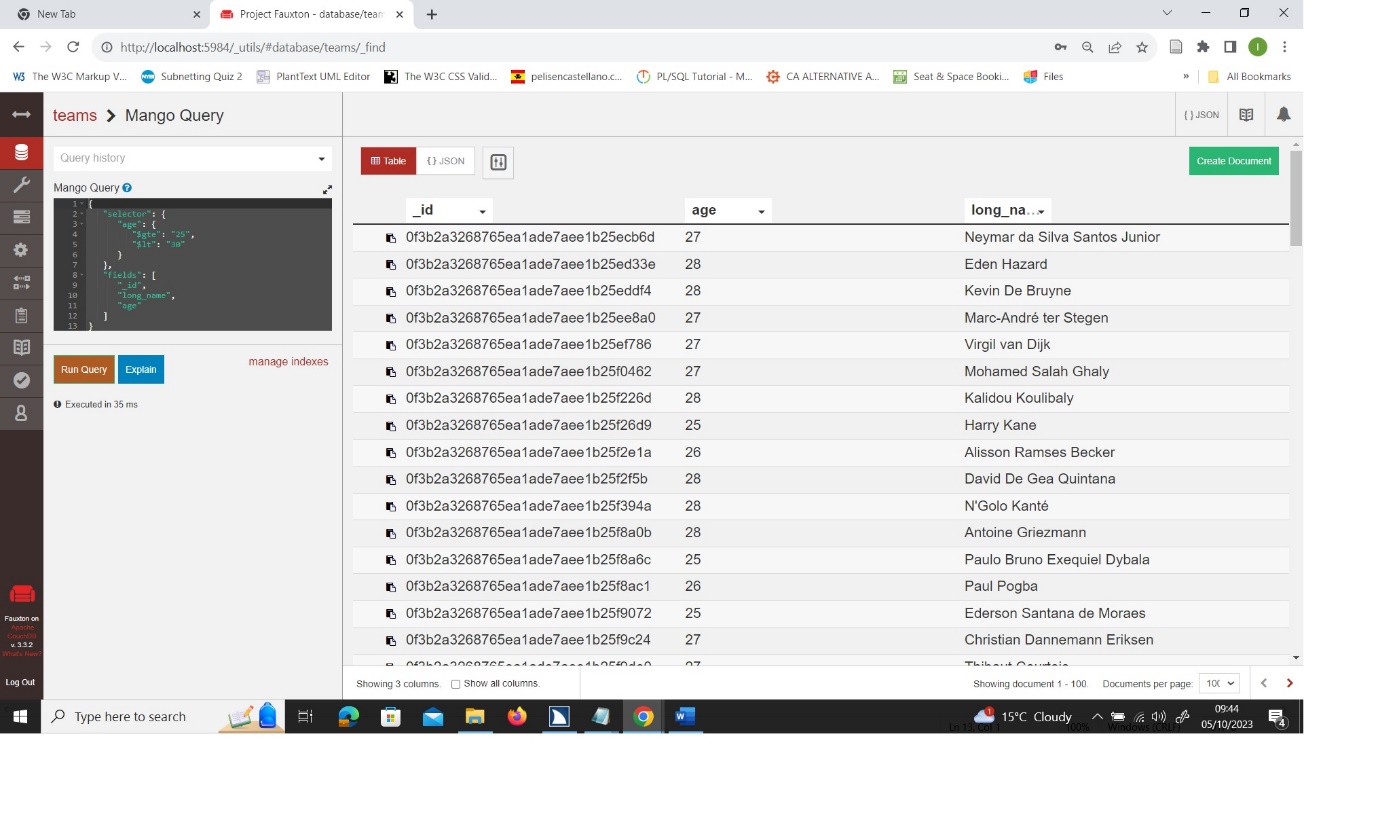


Figure 3.2.2 Range of documents

### Comparison

Mango queries (Apache CouchDB, 2023) have more simple syntax than MapReduce (Apache CouchDB, 2023), so it is easier especially for developers that come from using SQL, since it has a similar structure. Moreover, Mango query provides handy features easy to use i.e. sort (to sort the result), and limit (to output only a specific number of documents). These features are not provided by MapReduce. This language also supports ad-hoc queries, unlike MapReduce.

The problem with Mango is that as is so simple, it is not very good for very complex queries. Mango cannot handle nested JSON objects or arrays within a fields.

As MapReduce is JavaScript, we can get the most of this programming language and use all the features that JavaScript provides. With MapReduce we can create more complex views and handle

nested JSON objects or arrays, unlike Mango.

With MapReduce we must check if the document contains the fields required, otherwise, the code crashes. Also, developers must have knowledge of JavaScript to create views.

# Drivers for working with CouchDB

I chose Python because it has two drivers to work with CouchDB. The two drivers are CouchDB and (pypi, 2023) Cloudant (pypi.org, 2023). I have chosen to describe the features under the following headings: Functionality, Support, Maintenance/Longevity and Scalability.

## functionality

Both libraries provide the features needed to work with CouchDb. Users can access the server via URL with authentication. These tools assist in the most common tasks: create a database and the crud operations. All of these processes can be done in a very few lines.

The two libraries can create and run views on the databases. In order to perform a view, the MapReduce function must be stored on a variable. Then, the variable is saved on the database through the corresponding method. The result, a loop statement, must be used to go through all the documents with the result.

The CouchDB library can alter the Futon application in order to do the MapReduce functions in Python instead of JavaScript. To enable this functionality, the following script must be added to the local.ini file

[query\_servers]

python=/usr/bin/couchpy

These libraries can be installed through the command line with the pip install command.

## Support

Both libraries are held on the official Python web site. All the libraries that are on the official web site have to fulfil the standards required in order to be used in Python , such as documentation, code examples or the official repository. Therefore, by being published officially, users will find all the information to work with the libraries.

CouchDB version (1.2) is supported by Python 2.7 and upwards

Cloudant version (2.15.1) is supported by Python 3.5 and upwards.

## Maintenance/Longevity

Here it is where users will notice some differences. CouchDB had the latest release 09/Feb/2018 and Cloudant had its latest released in 10/Mar/2022.

However, on the official Cloudant web site advices to migrate to IBM Cloudant library (emlaver, 2023), and provides the necessary links for the transition. As the IBM Cloudant is still very recent, since it is on the 0.6 version. The web site warns that the library is still new so that it has some limitations.

## Scalability

Both libraries work with CouchDB. However, Cloudant also works with IBM Cloudant and has all the needed methods to connect to the IBM Cloudant server.

IBM guarantee unlimited scalability with their offering of Cloudant server.

## Decision

After reading about the two libraries, it seems that the Cloudant library is the best option for a CouchDB application. First, the CouchDB library is no longer supported by the owner. So, in case there is a problem with the library, it is clear that there will not be a solution.

The Cloudant is no longer supported either (same as CouchDB library). However, the latest update is much more recent. so that, we are guaranteed that the likelihood to have a problem is lower.

Moreover, the Cloudant library web page facilities the upgraded version link. Also, it provides the links needed for a better migration to the IBM Cloudant library. At the present time the IBM environment has a bright long term future.

The code used for working with Cloudant is given in appendix 2

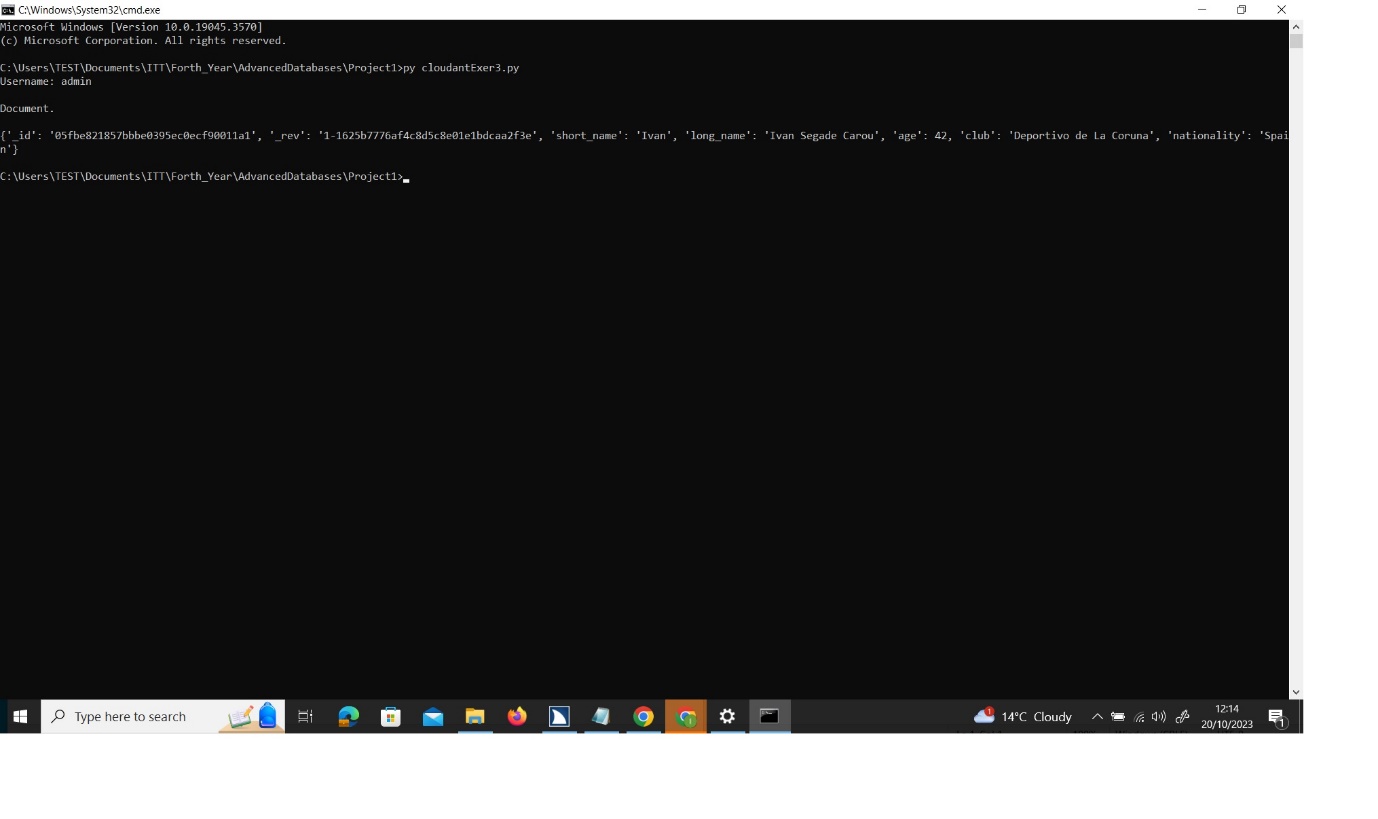


Figure 4.5.1 Output after retrieving a document from my local database

# Replication and offline application

## Replication

CouchDB has the ability to distribute the data from one node to one or more nodes. With this, CouchDB guarantees data is always available for users. Usually, there is a node master, and the rest of them just are a copy of it. In CouchDB all nodes are master in order to preserve availability of the database.

Initially, CouchDB had “Transient” replication. This way of replication meant that, when the database was restarted, the replication was deleted. Later, CouchDB incorporated the “Persistent” replication. This way keeps the documents continually.

The replication is triggered by saving a document either on the replicator or replicate endpoint. The replication compares the documents on the origin and the destination. The documents that are already reviewed on the destination are not modified. Only the document that were modified or deleted are transferred. The replication finished when it reaches the final modification. In case the settings are configured to be replicating continually, the replication is triggered for any modification. CouchDB sets break point document, to know what was the last modification in case of a crash.

The main goal of CouchDB is to have databases available in case any problem with the nodes or the network occurs. When the service is re-established, CouchDB updates all the nodes with the latest data. Therefore, the service is always running despite errors in the service. Users would never notice that the service is down, even if there is no connectivity, or any problem with a node or with network, they will be able to keep working.

To set up a replication in CouchDB, users must follow these steps:

* Access the database to be replicated.
* Create a new replication. Set the origin database and the target. In case there is partitions, users can configure this feature. Also, users can select if only one replication is needed or even continually.

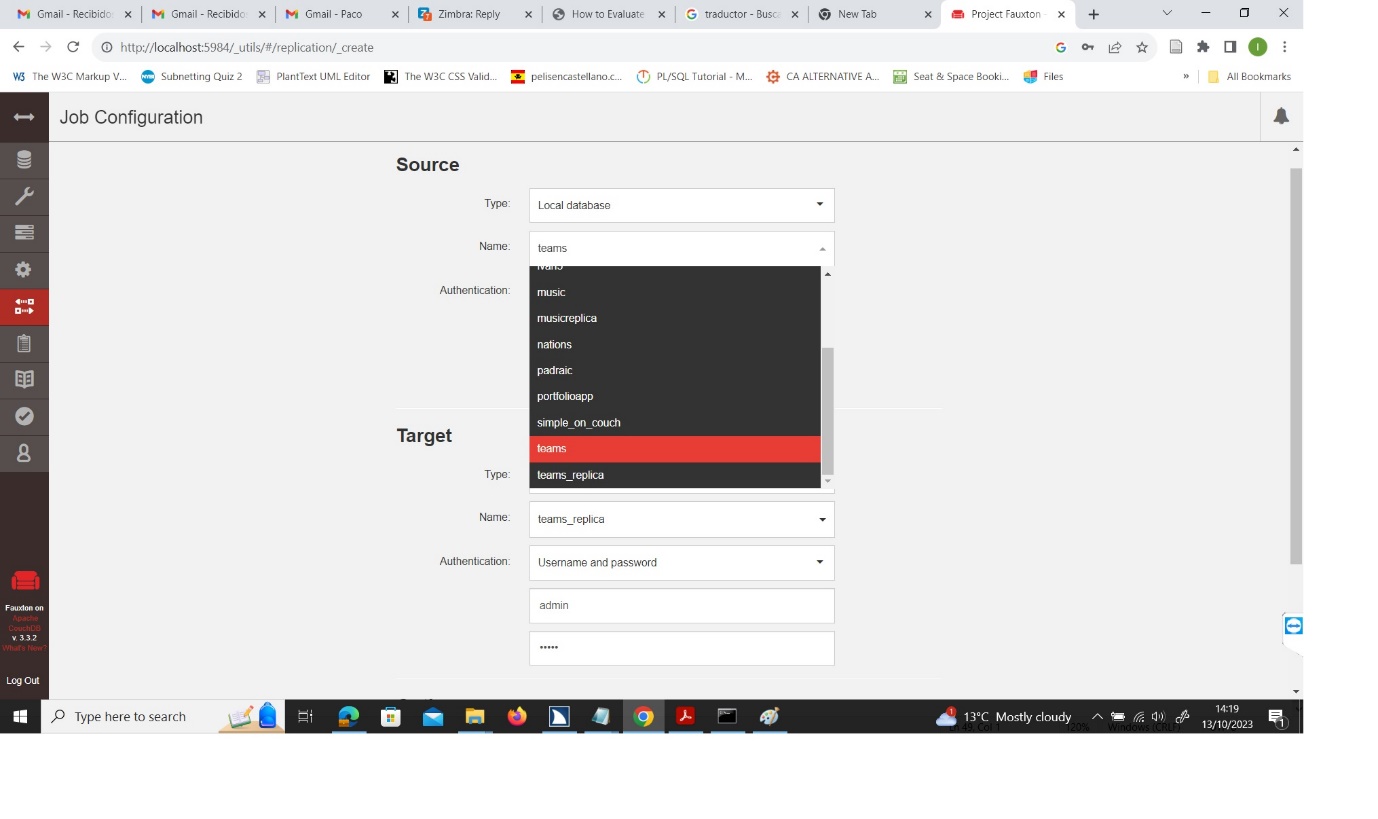


Figure 5.1. Example of setting up a replication

When the replication is finished, the new database contains all the documents of the original database, but only with the latest version. Therefore, the replica database will have a smaller size than the source.

One of the features of CouchDB is that after updating a document, it keeps the previous version of that particular document.

CouchDB manages the updating process with the \_rev (revision) field. This field contains the version number, which helps to track all the modifications that a document suffers.

## Offline application

An offline Node\_JS application is created in order to performance the four crud operations (Create, Read, Update and Delete). The code uses the “nano” library to connect to the database and process the operations. This application does not provide any interface, so that the data must be entered manually. The code used is on the appendix 2.

### Creating a document.

On the app.js file a variable is set with the values of the new document on JSON format. As the file contains the four crud methods, the methods excepting the “create” are commented out to avoid interfering with this task. The DOS output and the document on CouchDB are shown.

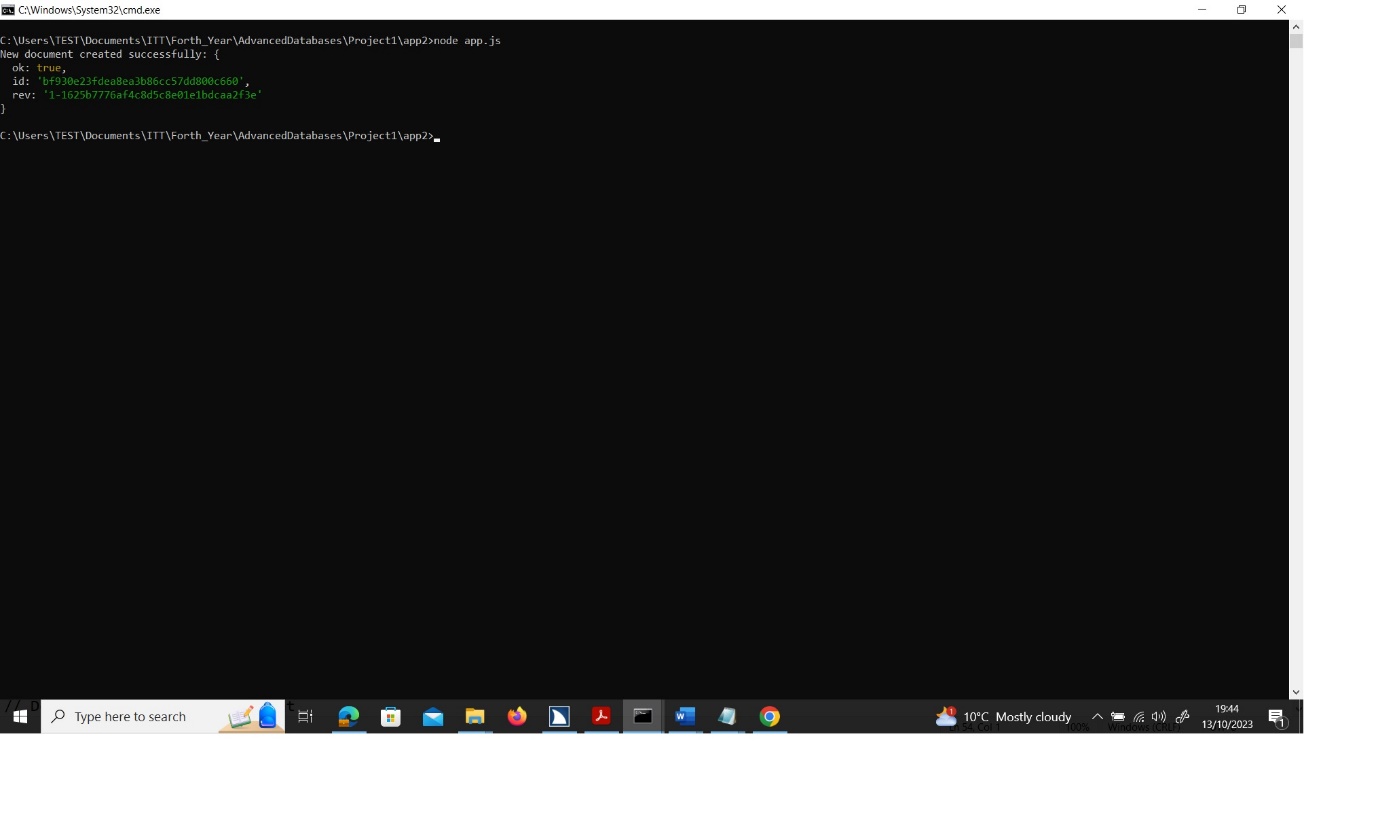


Figure 5.2.1.a. DOS output

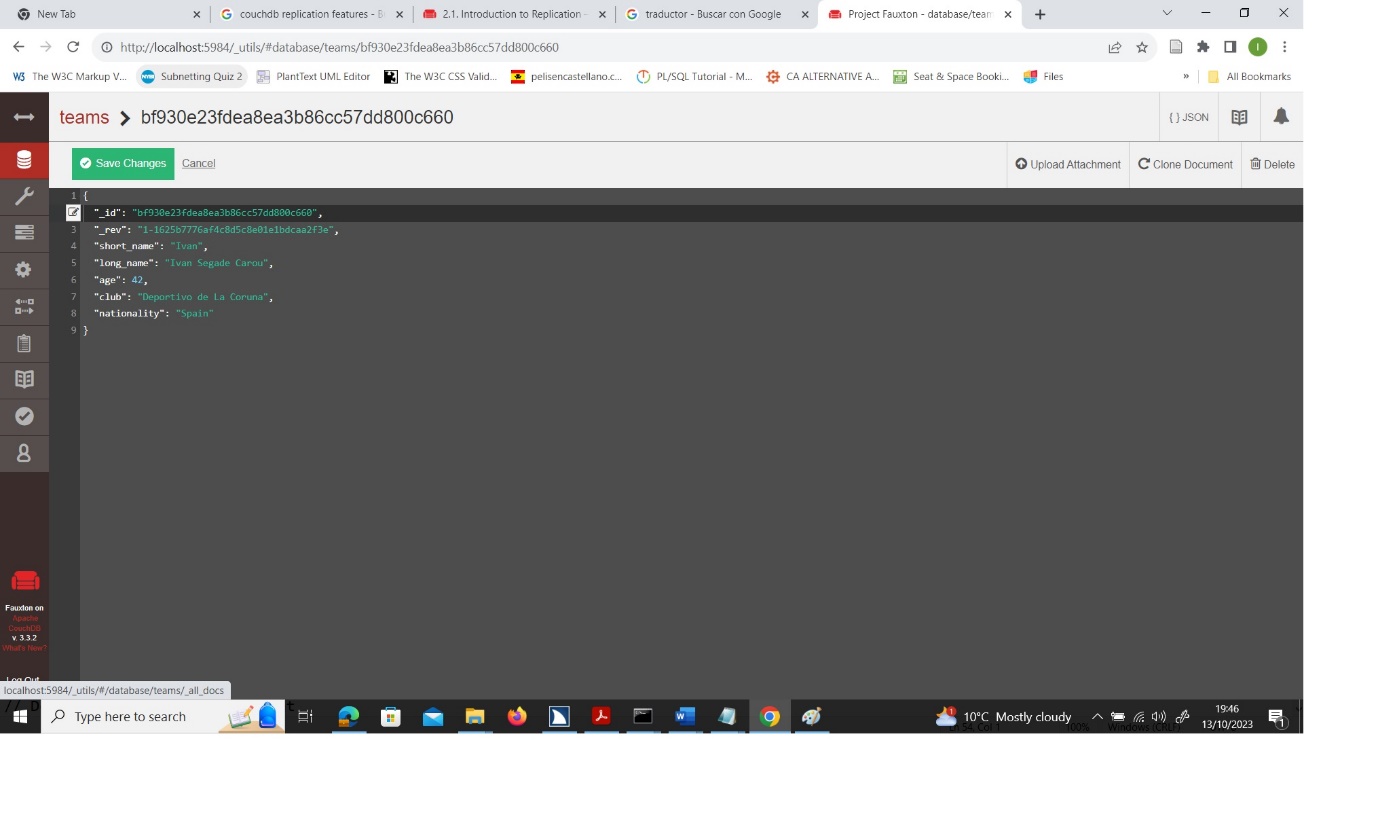


Figure 5.2.1.b. CouchDB document output

### Reading a document

On the app.js file a variable is set with the \_id value of the document to be retrieved. Now, the “read” method is uncommented out and the “created” method is commented out. The DOS output is shown.

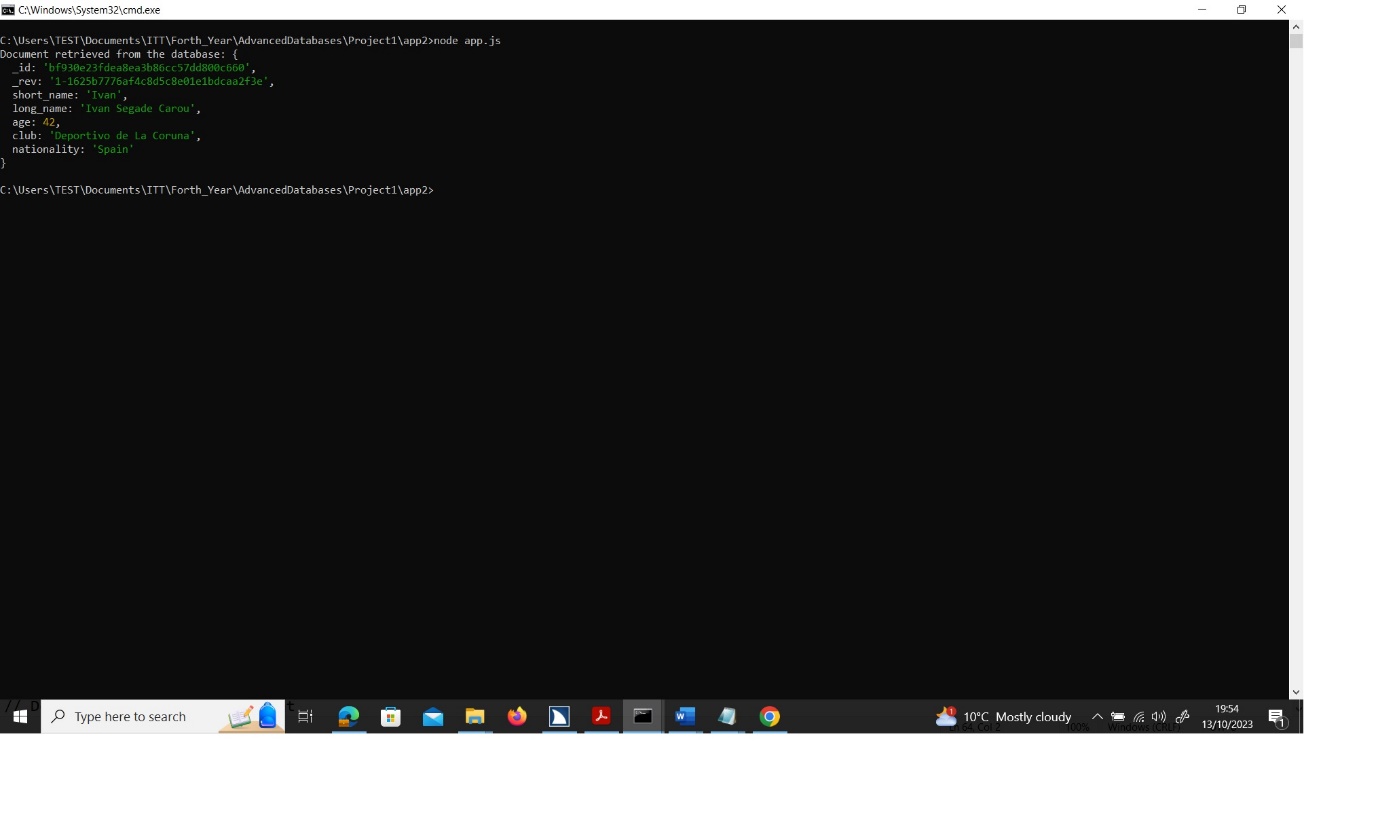


Figure 5.2.2. read DOS output

### Updating a document

On the app.js file a variable is set with the new values and another variable with \_id document to be updated. Now, only the “update” method is uncommented out. The DOS output and the document on CouchDB with the new values are shown.

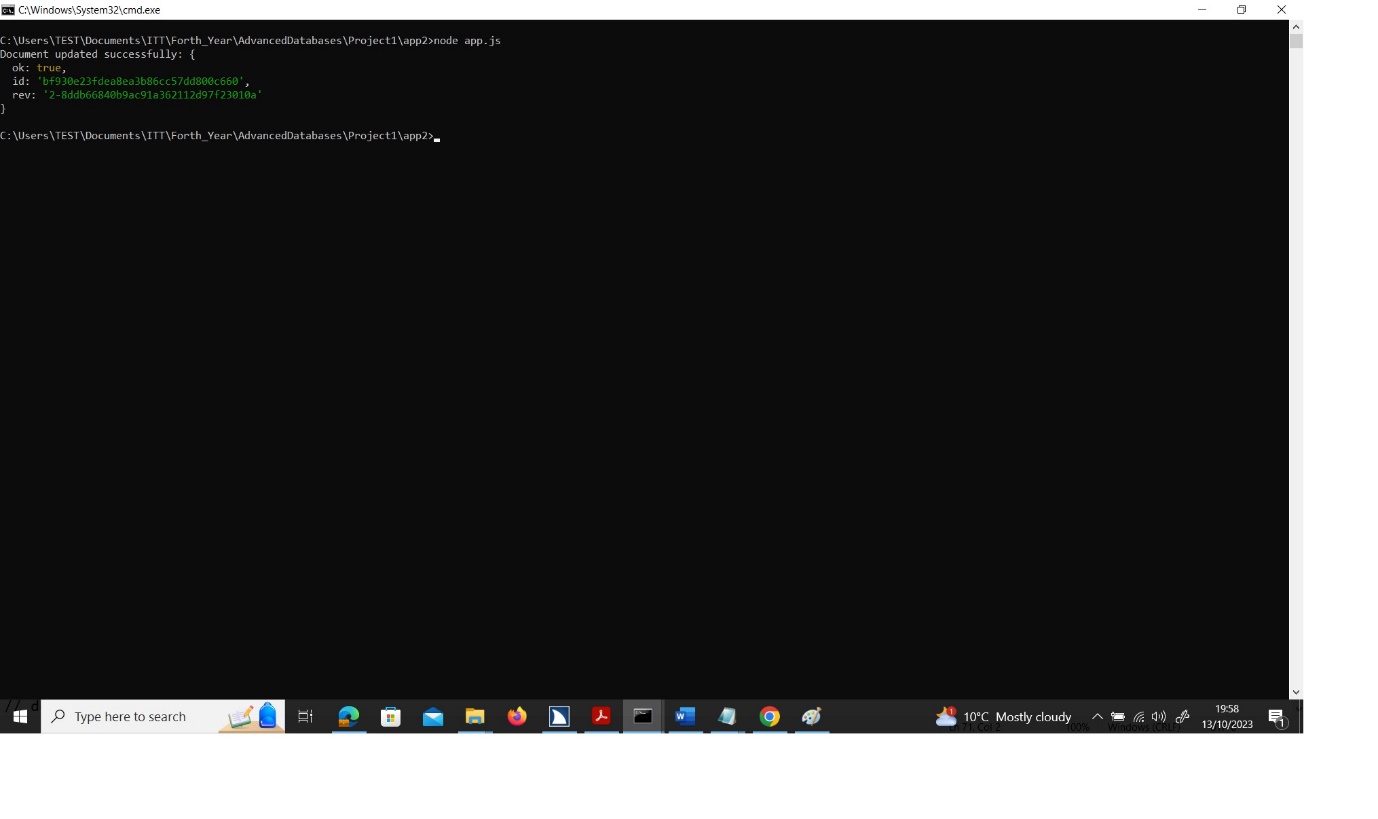


Figure 5.2.3.a. DOS output

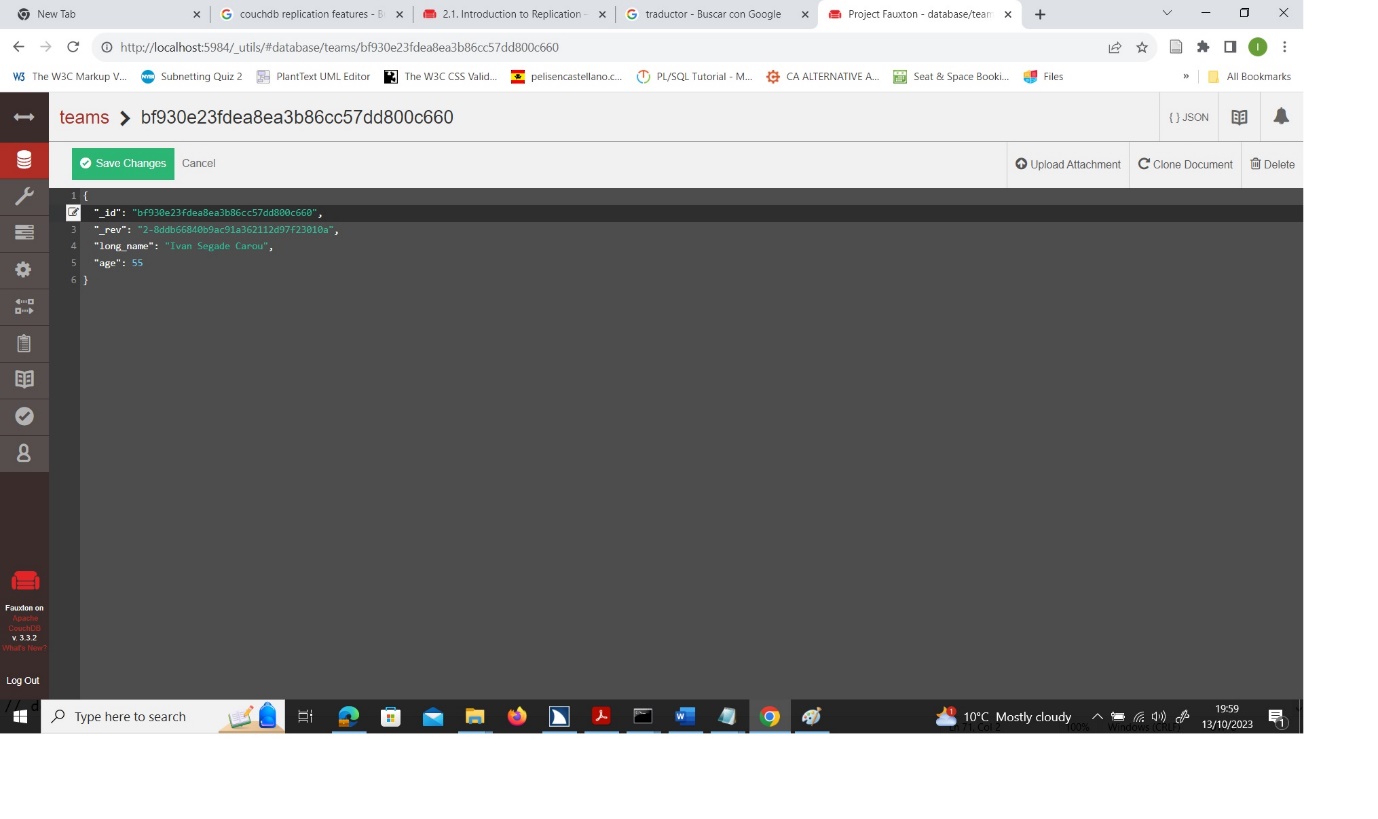


Figure 5.2.3.b. CouchDB document output

### Deleting a document

On the app.js file a variable is set with \_id document value to be deleted. Now, only the “deleteDoc” method is uncommented out. The list of documents on the database before deleting is shown. Then, the DOS output is displayed with the result of processing the “deleteDoc” method. Finally, the list of documents on the database after deleting the document is shown.



Figure 5.2.4.a. CouchDB database documents before deleting

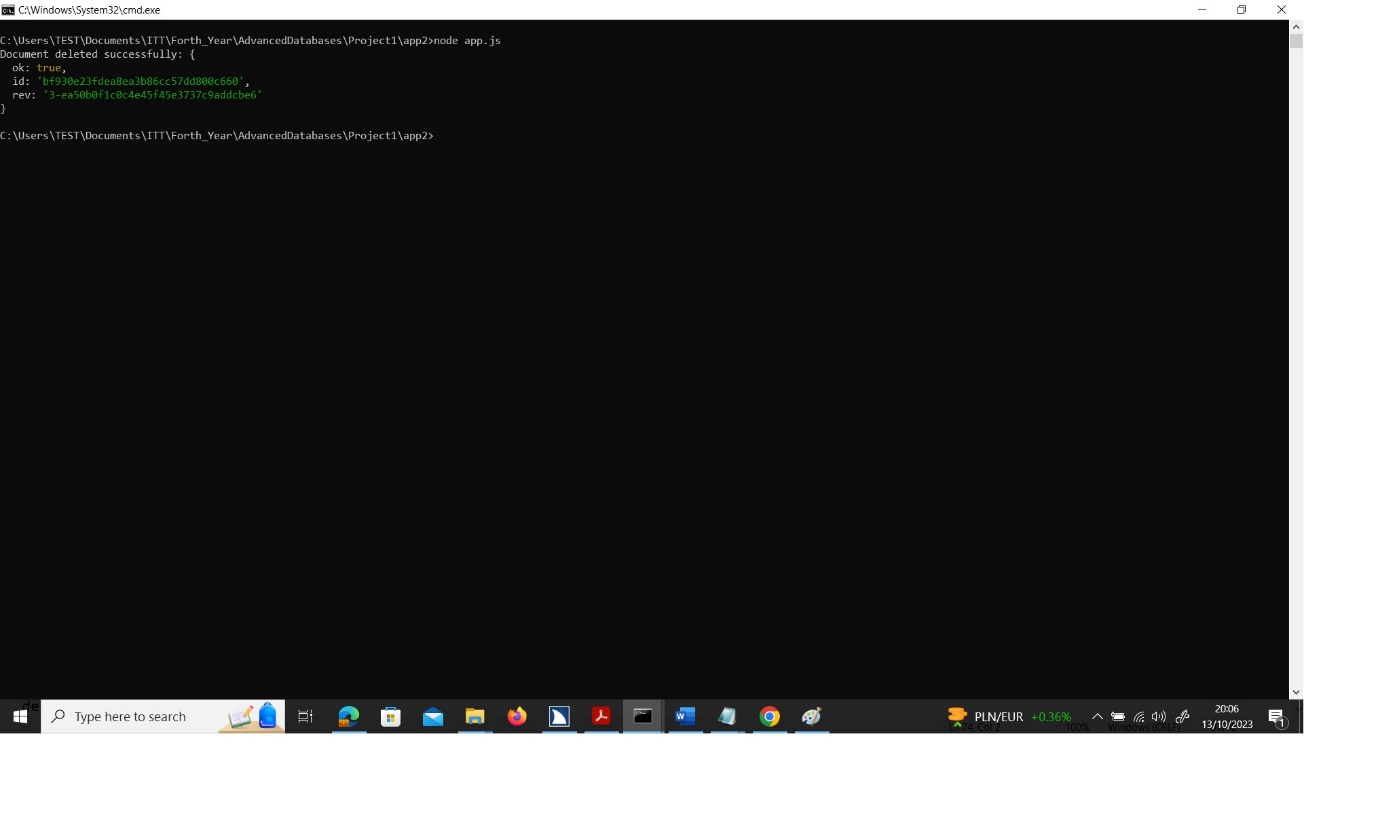


Figure 5.2.4.b. DOS output

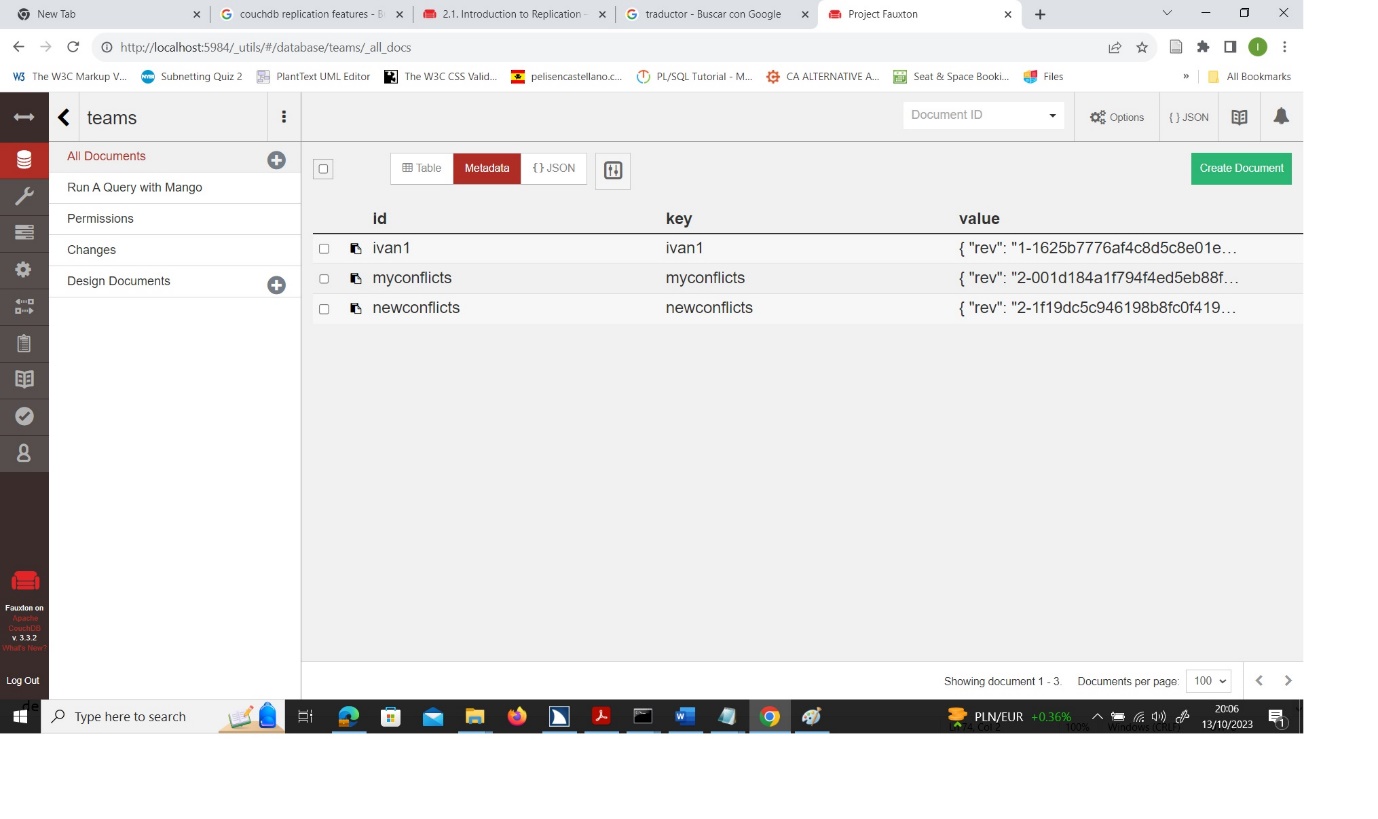


Figure 5.2.4.c. CouchDB database documents after deleting

With this code example, users can realise that with a very simple script, they can connect and perform operations with CouchDB. So, JavaScript has useful libraries in order to work with CouchDB without need of complex code.

# Conclusion

CouchDB is designed for availability rather than consistency. Therefore, in case users need an application where they need that service is operative regardless the data is not totally updated, CouchDB is a good option (i.e. games). This is possible thanks to features such as replication and partition.

As well, if the project is design for environment where, for any reason, connectivity is not guaranteed and users will need to keep working, CouchDB will cover this circumstance. For these cases, the application will be able to keep working without connection, and once the connection is available again, CouchDB will synchronise all data and try to solve possible conflicts.

Also thanks to replication, users will have the service guaranteed, although one mode fails., Since all the nodes will have the data synchronised. Therefore, if the operative node fails, another node will take over the service with the exact data.

CouchDB uses JSON objects, and this standard is widely used in the recent years. So, find compatible tools and programming languages to work with will not be a problem.

In case users need an application, where it is crucial that information is totally synchronised, this tool is not a good choice. CouchDB applications keeps working even though there is no communication with the database. Therefore, users may work with outdated data.

CouchDB is not a relational database. So, consistency is not guaranteed for projects where this feature is crucial, such as banking. Therefore, projects that work with transitional data, CouchDB may not be the right choice.

Moreover, in databases where data is constantly changing, CouchDB may not be a good option, since this tool keeps all the documents versions. So in this case, the database may become huge since the service will slow down dramatically.

Therefore, CouchDB is a good option for using no relational database, where users prioritise availability over consistency. Moreover, for situations where communication is not guaranteed. Also in the case where data tracking is required, since CouchDB keeps all document versions.

# Working with Capella

Capella was used to set up a database. Then Postman was used to send a new JSON object to the cloud by using the Post method.

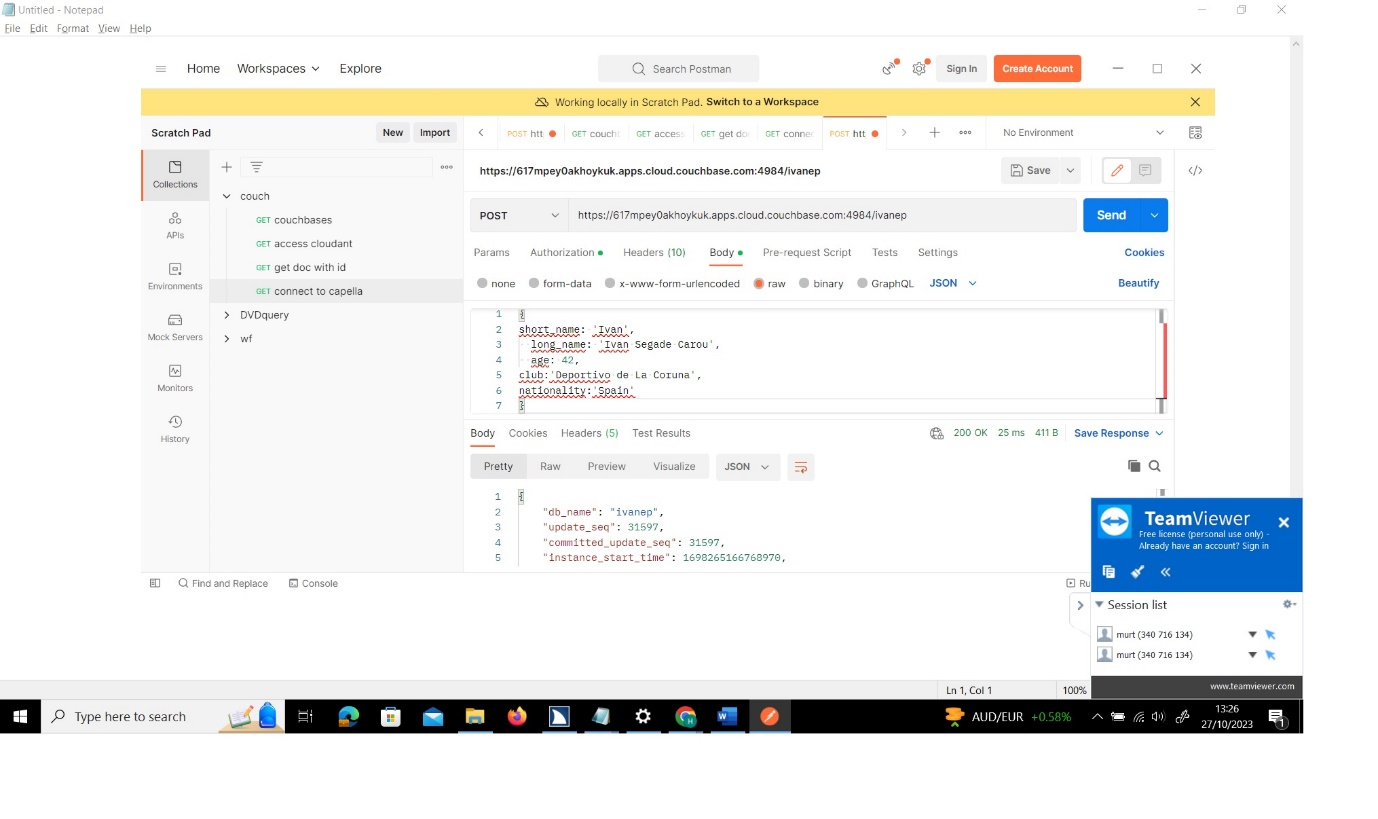


Figure 6.1. Postman result

To improve the offline application developed on the previous section, in order to work with the new database, a replication could be created. For this purpose, users must follow the steps written on the Replication section. In this case, the target database will be the Capella database.

Then, the two databases will be synchronised and users will be able to work with both databases and the data will be updated in both places.

# Appendix 1

Code of the Python script for working with CouchDB using the Cloudant library:

import couchdb

server = couchdb.Server("http://admin:admin@127.0.0.1:5984/")

db = server["teams"]

print("\nDocument.\n")

myID = "myconflicts"

doc = db[myID] # or db.get(myID)

print("\n\n",doc)

# Appendix 2

Code of the app.js file for the offline application:

const nano = require('nano')('http://admin:admin@127.0.0.1:5984');

// Connecting to the database

const db\_Name = 'teams';

const db = nano.db.use(db\_Name);

// Createing a document

async function create(doc) {

try {

const response = await db.insert(doc);

console.log('New document created successfully:', response);

} catch (error) {

console.error('Error creating the new document:', error.message);

}

}

// Reading

async function read(docId) {

try {

const document = await db.get(docId);

console.log('Document retrieved from the database:', document);

} catch (error) {

console.error('Error retrieving the existing document:', error.message);

}

}

// Updateing

async function update(docId, updatedDoc) {

try {

const document = await db.get(docId);

updatedDoc.\_id = document.\_id;

updatedDoc.\_rev = document.\_rev;

const response = await db.insert(updatedDoc);

console.log('Document updated successfully:', response);

} catch (error) {

console.error('Error updating the existing document:', error.message);

}

}

// Deleteing

async function deleteDoc(docId) {

try {

const document = await db.get(docId);

const response = await db.destroy(document.\_id, document.\_rev);

console.log('Document deleted successfully:', response);

} catch (error) {

console.error('Error deleting the existing document:', error.message);

}

}

// Running the code

// Createing a new document

const newDoc = {

short\_name: 'Ivan',

long\_name: 'Ivan Segade Carou',

age: 42,

club:'Deportivo de La Coruna',

nationality:'Spain'

};

create(newDoc);

// Reading a existing document

const docId = '05fbe821857bbbe0395ec0ecf9000842';

read(docId);

// Updateing a existing document

const updatedDoc = {

long\_name:'Ivan Segade Carou',

age: 55

};

update(docId, updatedDoc);

// Deleteing a existing document

deleteDoc(docId);