**A logo for a company

Description automatically generated**

**CouchDB Project 2025**

Stefan Lajtar

T00225936

Computing with Games Development

Munster Technological University,

Kerry Campus

Subject: Advanced Databases

Lecturer: Peter Given

# Abstract

This project explores the implementation and integration of CouchDB and its related technologies throughout the series of practical tasks. It demonstrates how NOSQL databases can be used to analyse data using CRUD, MapReduce Functions and Mango Queries. Project also compares different CouchDB libraries and their use in C#, while Redis is introduced as a caching layer. Replication protocols are examined throughout the PouchDB. The final task investigates cloud integration using IBM Cloudant as a Database as a Service and analyses synchronisation between local and cloud databases. The aim is to provide understanding of offline-first architecture and synchronisation strategies that are useful for both cloud and local environment.

Contents

[2.0 Dataset 4](#_Toc212445609)

[2.1 Importing dataset 4](#_Toc212445610)

[2.2 CRUD Operations 5](#_Toc212445611)

[2.2.1 Create 5](#_Toc212445612)

[2.2.2 Read 6](#_Toc212445613)

[2.2.3 Update 7](#_Toc212445614)

[2.2.4 Delete 8](#_Toc212445615)

[2.3 Summary 8](#_Toc212445616)

[3.0 Map Reduce 8](#_Toc212445617)

[3.1 Find a particular document 8](#_Toc212445618)

[3.2 Find documents in range 11](#_Toc212445619)

[3.3 Three Reduce Functions 13](#_Toc212445620)

[3.3.1 SUM Function 13](#_Toc212445621)

[3.3.2 COUNT Function 14](#_Toc212445622)

[3.3.3 STATS Function 15](#_Toc212445623)

[3.3.4 CUSTOM Function 17](#_Toc212445624)

[3.4 Mango Queries 19](#_Toc212445625)

[3.4.1 Find a particular document 19](#_Toc212445626)

[3.4.2 Document in a range 20](#_Toc212445627)

[4.0 No SQL Queries 21](#_Toc212445628)

[4.1 CouchDB Views 21](#_Toc212445629)

[4.2 CouchDB Mango 21](#_Toc212445630)

[4.3 MongoDB Query Language 21](#_Toc212445631)

[5.0 Evaluation and Redis 22](#_Toc212445632)

[5.1 CouchDB Clients 22](#_Toc212445633)

[5.1.1 Introduction 22](#_Toc212445634)

[5.1.2 CouchDB.Net 22](#_Toc212445635)

[5.1.3 MyCouch 23](#_Toc212445636)

[5.1.4 Conclusion 23](#_Toc212445637)

[5.2 CouchDB.NET in Code 24](#_Toc212445638)

[5.3 Introducing Redis 26](#_Toc212445639)

[6.0 PouchDB 28](#_Toc212445640)

[6.1 Replication Protocol 28](#_Toc212445641)

[6.2 PouchDB with JavaScript 29](#_Toc212445642)

[6.2.1 PouchDB Create 29](#_Toc212445643)

[6.2.2 PouchDB Read 31](#_Toc212445644)

[6.2.3 PouchDB Update 31](#_Toc212445645)

[6.2.4 PouchDB Deletion 32](#_Toc212445646)

[7.0 IBM Cloudant 34](#_Toc212445647)

[7.1 Creating and adding to database 34](#_Toc212445648)

[7.2 Synchronisation between Cloudant and CouchDB 36](#_Toc212445649)

[References 37](#_Toc212445650)

# Introduction

The main purpose of this paper is to analyse and present CouchDB and related database technologies as a part of the Advanced Databases Module. The main goal is to develop an understanding of how NOSQL databases operate, how can they be modified using CRUD and how distributed replication works. Project starts by converting and importing dataset into the CouchDB, which is then analysed using MapReduce and Mango Queries. It then moves into evaluating .NET client libraries that are supported by CouchDB, Redis application as a caching layer and exploring how replication protocols are working. The later stage involves implementing an offline-first JavaScript application using PouchDB and extending it into the cloud environment using IBM Cloudant. Together these tasks are representing full end to end database systems. Link to the project is following: https://github.com/StefanL02/CouchDB

# 2.0 Dataset

## 2.1 Importing dataset

The data used in this project is [Disney Movies Dataset](https://www.kaggle.com/datasets/prateekmaj21/disney-movies?resource=download). Dataset was downloaded as CSV file, to use it had to be converted to .json. Website used for conversion is called [csvjson](https://csvjson.com/csv2json). Each document represents one of the Disney movies, dataset includes total of 6 fields such as movie\_title, release\_date, genre, mpaa\_rating, total\_gross, inflation\_adjusted\_gross.

|  |  |  |
| --- | --- | --- |
| Field name | Description | Example |
| movie\_title | The official title of the movie | Cars |
| release\_date | The date when movie was released expressed in DD/MM/YYYY | 09/06/2006 |
| genre | Type of the movie | Comedy |
| mpaa\_rating | The age classification | G |
| total\_gross | Total box-office revenue expressed in US dollars | 244082982 |
| inflation\_adjusted\_gross | Total gross adjusted for inflation to reflect modern value in order to compare old and new movies | 314140384 |

Table 1 Dataset fields explanation

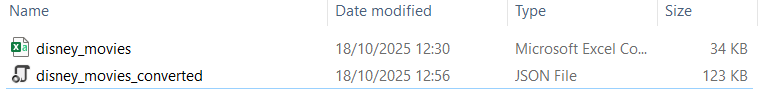


Figure 1-Original and converted dataset

## 2.2 CRUD Operations

### 2.2.1 Create

First step was to check whether CouchDB is running by contacting the “5984” port. Port is listening which means that CouchDB is running.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2-CouchDB running check

Next step was to create an empty database on Fauxton and call it movies:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3-New database

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4-New empty database on Fauxton

Step after that was to upload the dataset to the database, instead of copying and pasting the content of .json file, command bulk\_docs is used with POST request. Data binary was used instead of simple -d as it tends to crush the uploading and throw and error.

A close up of a screen

AI-generated content may be incorrect.

Figure 5-Uploading documents using POST

Now we can see that documents were successfully uploaded using Fauxton.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6-Database Content-Fauxton view

### 2.2.2 Read

By using a GET command we can retrieve all the documents/movies from the database.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 7-Database content-GET command retrieves all documents

Using a GET command to retrieve one movie

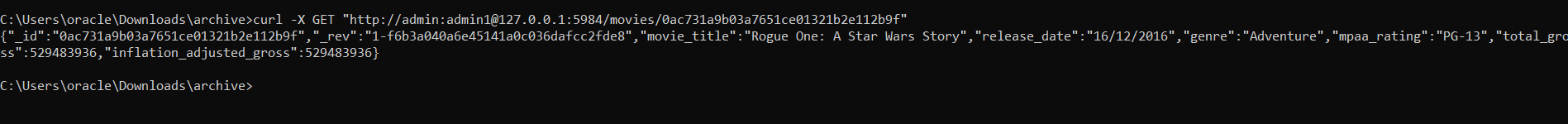


Figure 8-GET command retrieves single document

### 2.2.3 Update

Before I update any movie from this document, I will add a new movie which I will update and delete in next step so that I can keep original content of the database for next task.

Using a POST request to check if movie already exists before adding a new one

A computer code on a black background

AI-generated content may be incorrect.

Figure 9-POST request

Unfortunately, the movie: “The Jungle Book” already exists so another movie will have to be added

Movie Toy Story 4 was not present in the database

A screen shot of a computer

AI-generated content may be incorrect.

Figure 10-Checking for presence

Using POST request to add a new movie

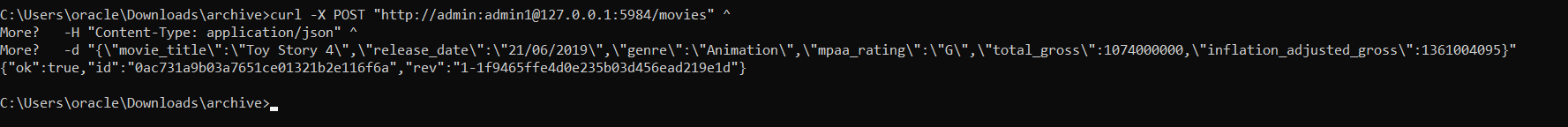


Figure 11-Adding a movie

Verifying that it’s there:

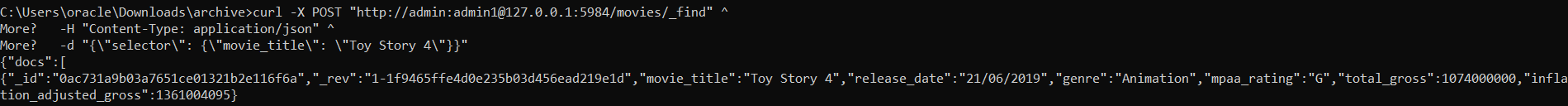


Figure 12-Verification of the movie

Updating newly added movie “Toy Story 4” by using PUT request and providing its current \_rev. Updating the movie by adding a note “Updated rating”.

A black background with many small colored lines

AI-generated content may be incorrect.

Figure 13-Updating using PUT request

### 2.2.4 Delete

The same document has been deleted by using DELETE option and providing its latest revision id. After search for the same movie error is displayed as the movie is deleted from the database

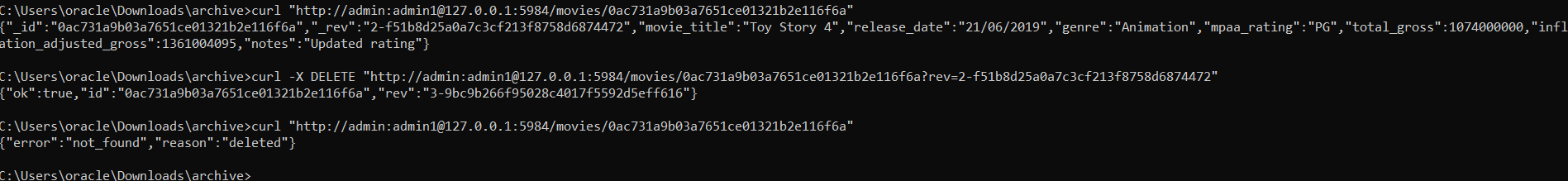


Figure 14-Deletion of the document

## 2.3 Summary

This task represents CRUD with its four components. POST and GET commands were used to create and retrieve documents. Toy Story 4 was updated using PUT request. Movie Toy Story 4 was not a part of the original dataset; it has been added later. Finally, same movie was removed using DELETE method, confirmation message is displayed, and document was not present in the database anymore.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Method** | **Example** | **Description** |
| **Create** | **POST** | Toy Story 4 | Inserted new record into movies |
| **Read** | **GET** | /movies/\_all\_docs?  include\_docs=true | Retrieved dataset and individual documents |
| **Update** | **PUT** | Toy Story 4 | Modified field using current \_rev |
| **Delete** | **DELETE** | Toy Story 4 | Modified field using latest \_rev |

Table -CRUD operations

# 3.0 Map Reduce

## 3.1 Find a particular document

In this example Map Function emits each movie title as a key. When used with a specific key it will return the matching row if title exists in the database.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Function to search for a specific document

Testing previous function using CURL to find a particular movie in the database. Movie Cars exists in the database and is retrieved.

A black screen with white text

AI-generated content may be incorrect.

Figure -Using CURL to find a particular document

In the other case we can use Fauxton to search for the movie title in the database.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Searching for movie in Fauxton

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Retrieving a movie in Fauxton

## 3.2 Find documents in range

This map function extracts the year from the release date by splitting the date string and taking only the last part of it. After that it emits year as key and the movie title and genre as the values.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Range Function

Using CURL to search for movies in range 1998-2000

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure -CURL for Range function

Fauxton setting to search for the same range

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Fauxton settings for Range search

Range results displayed in Fauxton

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Fauxton results for Range search

## 3.3 Three Reduce Functions

### 3.3.1 SUM Function

The SUM Function automatically adds together all numeric values that share same key. In this case map function emits genre as the key and it’s total gross revenue as the value.

A black rectangular object with white text

AI-generated content may be incorrect.

Figure - Function sum gross by genre

Showing total gross via CURL

A black screen with a black background

AI-generated content may be incorrect.

Figure -Using SUM Function

Using Fauxton to display SUM Function per movie genre.

A screenshot of a white screen

AI-generated content may be incorrect.

Figure -Using SUM Function in Fauxton

### 3.3.2 COUNT Function

In the COUNT Function if document has a genre field it emits genre as a key and adds a number 1 as a value. The reduce function COUNT adds all 1’s per genre and displays a result.

A long rectangular object with a black stripe

AI-generated content may be incorrect.

Figure - Count Function

Using CURL to test a COUNT Function

A black screen with a black background

AI-generated content may be incorrect.

Figure -Curl for Count Function

Displaying COUNT Function in Fauxton

A screenshot of a phone

AI-generated content may be incorrect.

Figure -Count Function in Fauxton

### 3.3.3 STATS Function

In STATS Function string is separated by split and only last element is taken which is a year. Function emits a composite key of genre and year, while movie’s total gross is the value. The Stats function automatically calculates statistical information for each group such as sum, count, min, max, sumsqr.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -STATS Function

Using CURL to test STATS Function. Level 2 includes genre and year

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure -Level 2 of STATS Function

Using CURL to test STATS Function. Level 1 includes only gender.

A black background with many small colored dots

AI-generated content may be incorrect.

Figure Level 1 of STATS Function

Using FAUXTON to test STATS Function

A screenshot of a computer

AI-generated content may be incorrect.

Figure FAUXTON STATS Function

### 3.3.4 CUSTOM Function

In CUSTOM Function string is separated by split and only last element is taken which is a year. Function emits a composite key of genre and year, while movie’s total gross is the value. Mapping part is same as for STATS Function, but in this case additional code is written to resolve the function. In the reduce phase we add all sums and record the total count. Second re-reduce phase gathers all information from other nodes and merges them together.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure -CUSTOM Function

A blurry image of a computer screen

AI-generated content may be incorrect.

Figure - Level 1 Custom view

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure - Level 2 Custom view

Level 1 Custom view returns averages per genre while Level 2 Custom view returns averages per genre and year.

## 3.4 Mango Queries

### 3.4.1 Find a particular document

Mango Query uses a selector to match the word entered. In this case it’s a movie title, The Jungle Book. In this case Mango will search for the documents that are matching movie\_title entered. In this case few documents are retrieved because they share same name but are different entities.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Mango Query for document search

After I introduced indexes into the Mango, it would look for an index field of the document and retrieve it.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Mango Query using indexes

### 3.4.2 Document in a range

Once again selector is used to search throughout the database. Search range is defined for the values greater or less than a total gross entered. After that four fields are displayed.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Magno Query document range

If we introduce Indexes, we can avoid full scan of the database and use sort to sort retrieved data.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Mango Query document range with indexes

# 4.0 No SQL Queries

## 4.1 CouchDB Views

CouchDB Views are implemented as JavaScript code containing Map Function which emits key value pairs for every document while Reduce Function summarise those values. This MapReduce Function makes CouchDB extremely flexible as it can be embedded directly into CouchDB to calculate sums, averages and custom statistics which is one of the benefits. In my examples I used built-in function such as sum, count, stats, but I also created my own to find a movie by title and to retrieve movies in the year range. It is very powerful for analytics and is fully customizable because of JavaScript. CouchDB Views can handle large datasets efficiently after first computations which is a huge advantage. However, some of the advantages can also be classified as disadvantages as MapReduce functions are hard to design and debug. Anyone who has no, or little knowledge of JavaScript would struggle to write a function. Every time code is changed, view must be rebuilt which can be slow if used in large database. CouchDB Views are not easy to use but they have a great analytical strength if used correctly.

## 4.2 CouchDB Mango

Mango Query Language is easier to use, and it also reminds of SQL style of writing. Ease of use and readability are key advantages. It requires no previous knowledge and is run in Fauxton. I used a command

“{“selector": { "total\_gross": { "$gte": 50000000, "$lt": 200000000 } } }” to retrieve all movies which total gross was between 50 and 200 million. At first query ran throughout the database as I had no indexes, but after introducing the indexes query ran faster. Also, in Mange we can chose what fields to show. I can show only time or genre or release date, or I can simply combine it the way I want. Because Mango is simple it can’t perform statistical operations like Views can. Also, indexes are playing main role in large databases, for quick execution they are necessary. Best use of Mango is for exploration of database as its analytical processing is limited.

## 4.3 MongoDB Query Language

One of the advantages of MongoDB is that it does not require predefined design document like CouchDB. In the case of MongoDB queries can be written and executed dynamically. Mongo is very successful in filtering as it contains advanced operators such as match and group. Mongo is not used in this project but, code can still be written to execute a query e.g.

db.movies.aggregate([

{ $match: { genre: "Action" } },

{ $group: { \_id: "$genre", totalGross: { $sum: "$total\_gross" } } }

])

This code would compute the total gross for the Action movies. It executes it in real time meaning that is fully recalculated every time it runs. This makes Mongo less flexible for repeated queries. There is a problem of CPU and memory consumption as repeating of complex aggregations can lead to lack of resources. The aggregation pipeline syntax is complex and specific for MongoDB which means that things written in MQL can’t be directly imported to CouchDB. Overall MQL can be seen as an combination of CouchDB’s Mango and Views but, it requires more knowledge than Mango and more computing power than Views.

# 5.0 Evaluation and Redis

## 5.1 CouchDB Clients

### 5.1.1 Introduction

One of my favourite programming languages is C# and I have explored CouchDB Clients for this language. Two clients that I have found are CouchDB.Net by Matteo Bortolazzo and MyCouch by Daniel Wertheim. Both libraries will allow developers to manage CouchDB databases. Information are mostly from GitHub repositories and developer’s forums as other sources weren’t available. I will evaluate them using Wheeler’s IRCA model under the following headings: Functionality, Support and Maintenance

### 5.1.2 CouchDB.Net

#### 5.1.2.1 Functionality

When we are talking about CouchDB.NET Functionality there is a few important things to mention. It fully supports CRUD through methods like ToListAsync() and AddAsync(). It also supports Mango queries, views and design documents directly through its API. POCO mapping is one of the advantages of CouchDB.NET as developers can define classes and automatically serialize them to CouchDB JSON documents. It has an integration support designated for ASP.NET Core.

#### 5.1.2.2 Support

Commits are frequent to GitHub repository while latest version is updated for .NET 6+. It is all well documented as README files include usage examples and sample queries. The author himself Matteo Bortolazzo has a blog where he explains setup and examples all step by step. There is also user engagement as issues and features requests on GitHub are answered within few weeks. For an open-source project it has a high-quality support.

#### 5.1.2.3 Maintenance/Longevity

CouchDB.Net is actively maintained as it has regular updates since 2019. It is compatible with .NET 6, .NET 7 and .NET 8 is planned, showing commitment and ensuring long term support. It also supports backward compatibility with older .NET versions.

### 5.1.3 MyCouch

#### 5.1.3.1 Functionality

One of the earliest C# clients is MyCouch. It focuses on simplicity which includes basic CRUD operations using HTTP calls such as Document.GetAsync. It is based on older Async models, but it still has Asynchronous support. However, it does not support Mango Queries, there is no POCO mapping which means that serialisation and deserialization is not automatic. There is no direct ASP.NET integration which requires more code for setup. MyCouch would be great for studying but there is no practical use in large projects

#### 5.1.3.2 Support

The GitHub repository is public but archived by owner. Documentation exists and is present on developer’s blog. There are no new updates on blog and all documents are from prior date. Community response is slow and almost non-existent

#### 5.1.3.3 Maintenance/Longevity

Last update was performed in early 2023 with only few commits after that. There is still no support for .NET 6+ and there is no schedule or announcement for future development, so future compatibility is uncertain. The library is stable which means it can be used in older versions of CouchDB. With current low maintenance picking MyCouch for future projects would be risky

### 5.1.4 Conclusion

After analysing both libraries, I can conclude with ease that CouchDB.Net has an advantage. CouchDB.Net is better in every aspect analysed making it better for current and future use. It is also actively being developed and improved which is not a current case for MyCouch. MyCouch can still be used in legacy projects as it is still stable, but for my demonstration I will CouchDB.NET.

## 5.2 CouchDB.NET in Code

Before testing a GET Function in C# I had to install CouchDB.NET

A white background with black and white clouds

AI-generated content may be incorrect.

Figure - CouchDB.NET

The following code is used to retrieve documents from database movies.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure -Code for GET part 1

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Code for GET part 2

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Retrieved document after executed code

The first step was to install CouchDB.Net. After that I imported required libraries for CouchDB connectivity and JSON serialization. Movie class is declaring getters and setters for the fields in the database. There is no ID or rev as part of the Movie class because they are resolved automatically throughout the inheritance from CouchDocument. JSON Property must match the field name in the database otherwise it won’t retrieve it as it does not match. Next step is to connect to CouchDB which is done by providing a link with username and password. Accessing movies database is next step where we retrieve all documents asynchronously using ToListAsync(). Foreach loop is used to print first 25 movies and display their ID, movie title and release date. At the end there is a catch in case that program doesn’t work so feedback can be provided.

## 5.3 Introducing Redis

A screen shot of a computer program

AI-generated content may be incorrect.

Figure -Inserting Redis part 1

A screenshot of a computer program

AI-generated content may be incorrect.

Figure -Inserting Redis part 2

A screenshot of a computer code

AI-generated content may be incorrect.

Figure -Inserting Redis part 3

This code here implements Redis. Redis was introduced as a caching layer to improve performance. Once again, all necessary libraries were imported. Class Movie is unchanged each JsonProperty must match corresponding field name in CouchDB while ID and rev are automatically resolved. The program first establishes connection with database where URL, username and password are. Before accessing CouchDB, program checks Redis using a unique cache key which is named movies:all:v1 in this case. If cached data exists in the Redis, it is deserialized and returned directly from memory. If data does not exist in the cache, the data is gathered from CouchDB using ToListAsync(). It is then serialized to JSON and stored in Redis cache with time to live three minutes which was declared earlier. This method is called cache-aside and it reduces query time. A try/catch is included so we can see a error in case that program crashes.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Running new code with Redis

A screenshot of a chat

AI-generated content may be incorrect.

Figure -Online Redis Database

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Cache file with TTL

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Cache and expiring TTL

While cache is existing, data will be retrieved directly from it. Redis acts as a temporary storage and due to its high speed, it improves performance and efficiency of the application by retrieving data from cache rather than from CouchDB.

# 6.0 PouchDB

## 6.1 Replication Protocol

Replication starts when change is detected. Firstly, sources were compared with targeted database via changes feed. That helps to identify which document has been changed (updated or created). After that revision check is performed where replicator checks which revisions of changed document the targeted database lacks by using revs\_diff endpoint. All documents that are missing and their revisions are extracted from the source and sent to the target. Sequence numbers are stored to track replication state and resume from last sync. It also helps to resume replication if crash happens. There is two way and one-way replications. One way are called push and they can go from source to target or other way around, while two ways are called master to master where two replications are going in opposite directions which can cause a conflict.

## 6.2 PouchDB with JavaScript

To test my PouchDB I will have to be off the internet when I am entering the data, so I am providing the screenshot of documents in the database while disconnected.

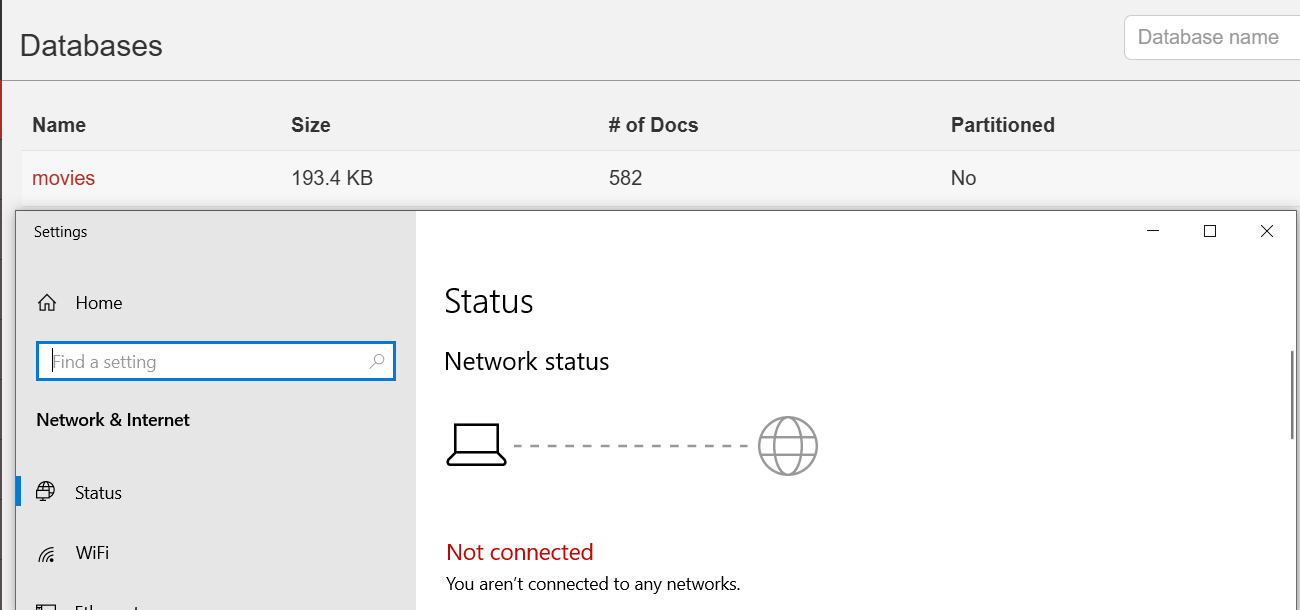


Figure -Disconnecting from the network

### 6.2.1 PouchDB Create

The main purpose of this activity is to add a movie to PouchDB so it can be added to the movies database when connection is restored. We run our JavaScript file PouchDB.html and we go completely offline.

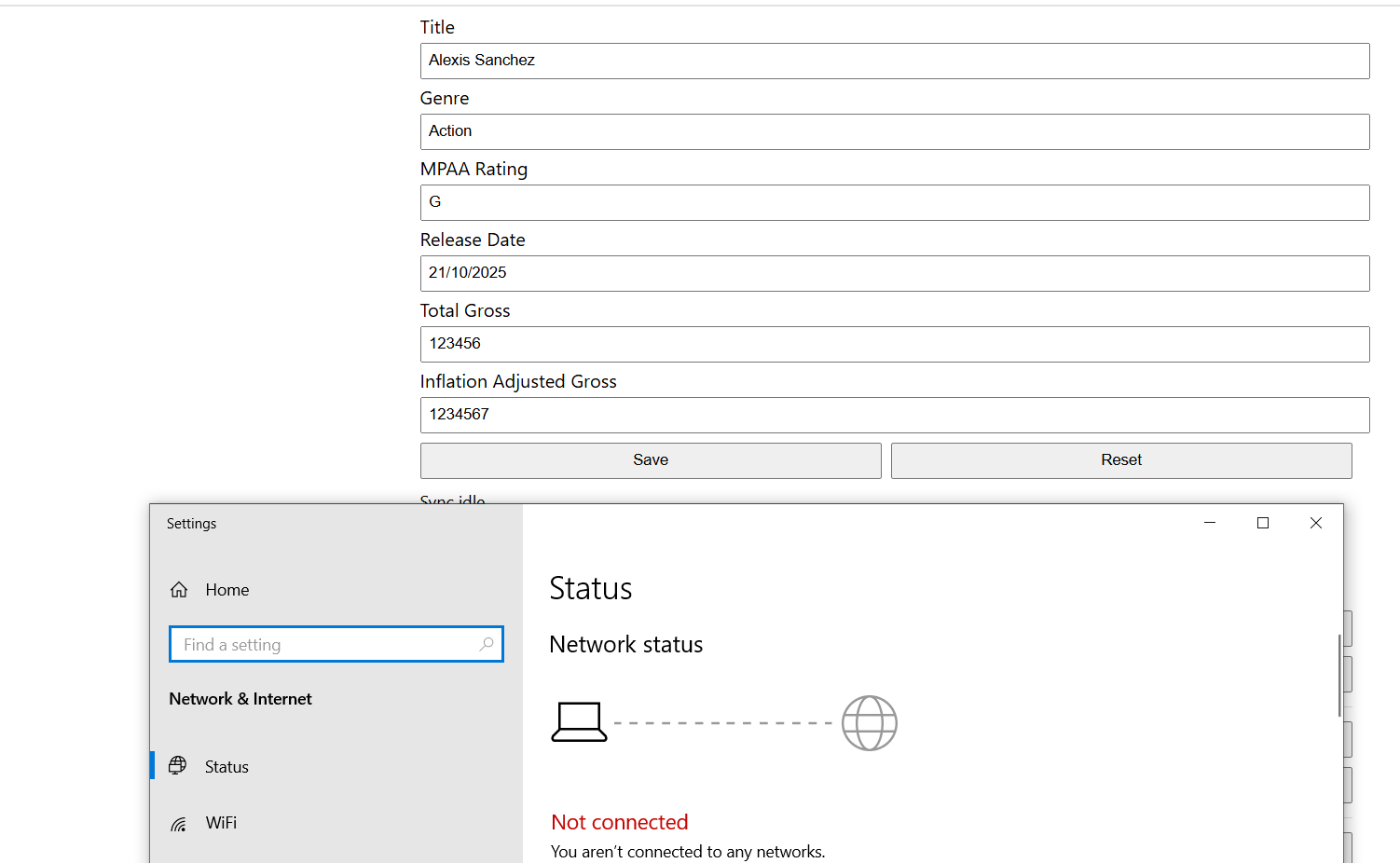


Figure -Creating a new movie

After connection has been restored we can see that a number of documents increased by one which is exactly what we wanted to see as we added a new document while being offline.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Database after restored connection

We can also search for the movie and retrieve it’s data. We can see that it contains all the details inserted earlier.

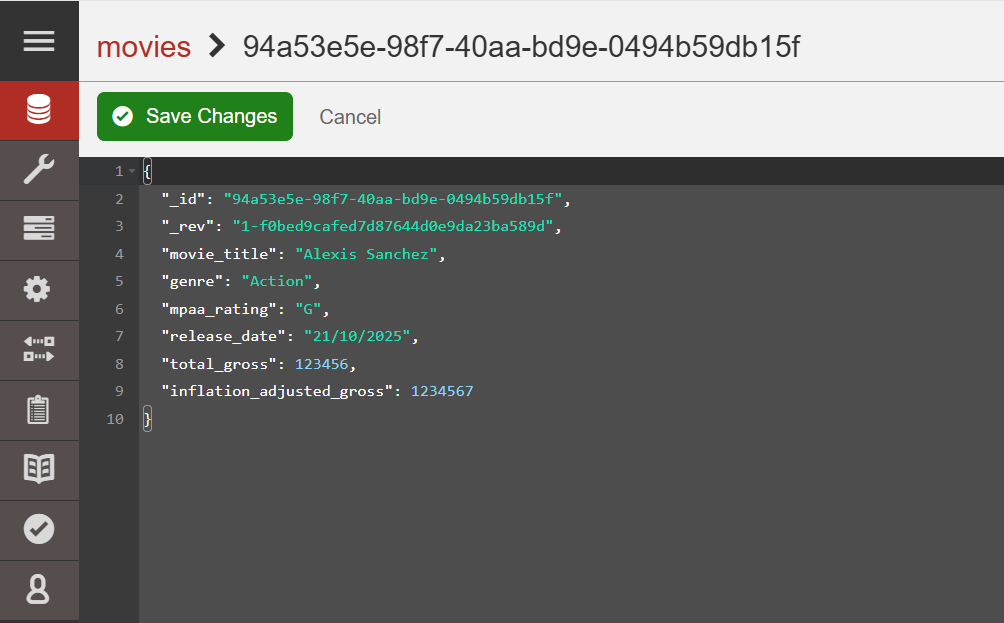


Figure -New movie in database

### 6.2.2 PouchDB Read

The following code has been used to read the documents and display them on screen.

A computer screen shot of a program

AI-generated content may be incorrect.

Figure - Read operation

The above code represents READ function which is implemented through the render() function which retrieves all movie documents using “localDB.allDocs({include\_docs:true}); and displays them on the webpage. It iterates throughout all rows and extract all fields and puts them in readable format.

### 6.2.3 PouchDB Update

Figure 55 shows the details of movie before it’s update.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure - Details before updating

Network is marked as offline and new total gross is entered along with inflation adjust gross and MPAA Rating.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - PouchDB Update

New details are matching those ones entered in the application. We can see that total gross is 1 while inflation adjusted gross is 22. MPAA Rating is marked as “G” same as in application.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Document after performing an update

### 6.2.4 PouchDB Deletion

Number of documents before any deletion is performed

A screenshot of a computer

AI-generated content may be incorrect.

Figure Number of documents

The following three movies has been deleted from DB while being disconnected from the network

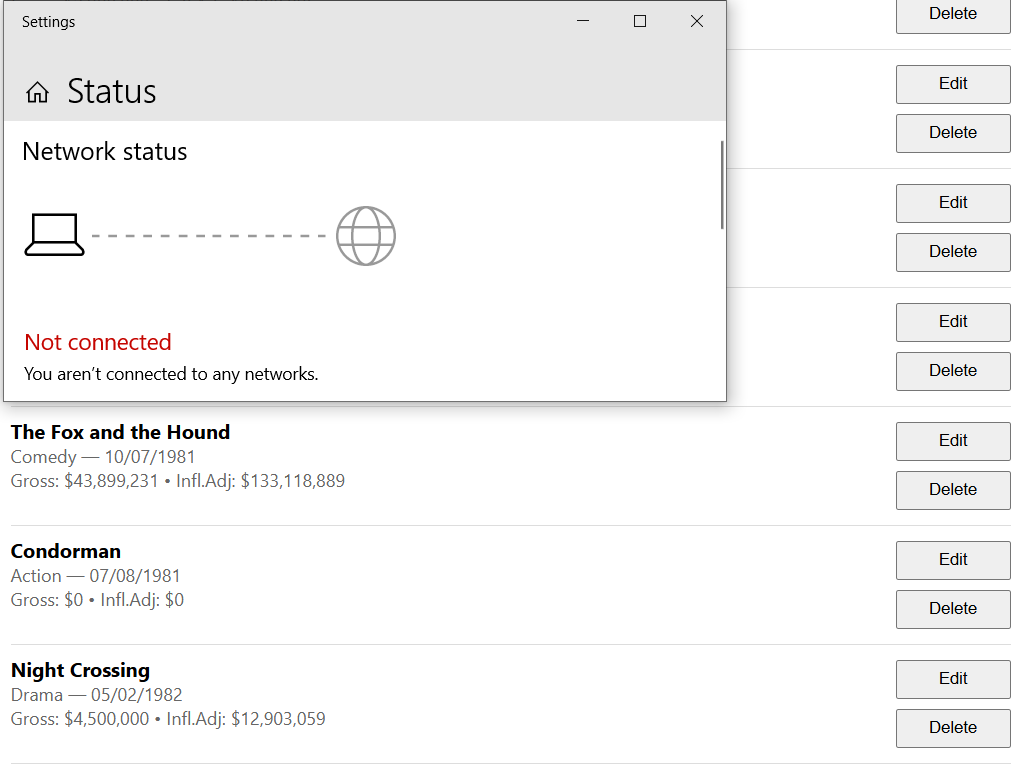


Figure -Movies ready to be deleted

After reconnecting back to the network, we can see that 3 documents are missing from the database.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Database after performing Deletion

After search for one of the deleted movies, Mango Query could not find movie with that name so as a return No Documents Found has been returned.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Mango Query to look for movie

# 7.0 IBM Cloudant

## 7.1 Creating and adding to database

For this task IBM Cloudant is used to demonstrate how Database as a service can be added to the application. A Cloudant instance was created throughout the IBM Cloud dashboard and access credentials were generated under Service Credentials. Using the provided API key, authentication was performed with a Bearer token in the command line via curl. The command PUT created a new database named movies, and the JSON response {"ok":true} confirmed successful creation. A following request to GET /\_all\_dbs returned ["movies"], verifying that the database was present and reachable.

A screen shot of a computer

AI-generated content may be incorrect.

Figure -Creating a database called movies

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure -Checking is database reachable



Figure -Cloudant database

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure -Adding a movie to the Cloudant



Figure New movies appear in DB

Unfortunately, I was able to use Postman to demonstrate any operations as my authentication was getting rejected every single time and I could not find a reason why. All information’s were entered correctly as required. Even the Postman’s bot could not resolve the issue, so I used the CURL command to POST to the DB which was successful and movie with name The Lion King has been added.

## 7.2 Synchronisation between Cloudant and CouchDB

All CRUD actions are happening locally, where PouchDB runs two live replication streams. One to the CouchDB and the other to the Cloudant database. All local changes are pushed up, and any remote change is pulled down. They would use same Couch DB replication protocol. It uses changes feed to find a modified documents. \_Revs\_diff checks which revisions are missing locally. Checkpoints are established on both sides so each stream can resume exactly where it left. Because two independent versions are involved, the same document could be edited in different places before replication takes place. In summary we write into the PouchDB and then PouchDB constantly pushes and pulls to both CouchDB and Cloudant.

# 8.0 Conclusion

Throughout the course of this project complete understanding of CouchDB and it’s ecosystem has been achieved. CRUD operations showed how documents can be easily managed in the NOSQL environment. MapReduce Functions showed how powerful they can be for data analysis, while Mango Queries showed that filtering can be easy and fast. Evaluation of two C# clients revealed that CouchDB.NET is most practical and maintainable solution for modern applications. Redis introduced caching for improved performance, while PouchDB provided offline-first approach making sure that data is available even without connection. Overall, this project represents a structured way of learning where every aspect of CouchDB was covered.

# References

Academy, F., 2016. *PouchDB Tutorial - Offline First Apps with PouchDB,* s.l.: s.n.

Anon., 2025. *stackexchange.* [Online]   
Available at: https://stackexchange.github.io/StackExchange.Redis/  
[Accessed 26 October 2025].

Bortolazzo, M., 2018. *GitHub.* [Online]   
Available at: https://github.com/matteobortolazzo/couchdb-net  
[Accessed 25 October 2025].

Bortolazzo, M., 2021. *Dev.To.* [Online]   
Available at: https://dev.to/matteobortolazzo/couchdb-and-c-efcore-like-queries-2ma4  
[Accessed 25 October 2025].

CouchDB, 2025. *Docs.CouchDB.* [Online]   
Available at: https://docs.couchdb.org/en/stable/replication/intro.html?utm  
[Accessed 25 10 2025].

CouchDB, 2025. *Docs.CouchDB.* [Online]   
Available at: https://docs.couchdb.org/en/stable/replication/protocol.html?  
[Accessed 26 October 2025].

CouchDB, 2025. *Docs.CouchDB.* [Online]   
Available at: https://docs.couchdb.org/en/stable/replication/conflicts.html  
[Accessed 26 October 2025].

Given, P., 2025. *MTU Canvas.* [Online]   
Available at: https://mtukerry.instructure.com/courses/6051/files/886832?module\_item\_id=283130  
[Accessed 25 10 2025].

Given, P., 2025. *MTU Canvas.* [Online]   
Available at: https://mtukerry.instructure.com/courses/6051/files/886838?module\_item\_id=283131  
[Accessed 10 25 2025].

Given, P., 2025. *MTU Canvas.* [Online]   
Available at: https://mtukerry.instructure.com/courses/6051/files/876497?module\_item\_id=274912  
[Accessed 25 10 2025].

Given, P., 2025. *MTU Canvas.* [Online]   
Available at: https://mtukerry.instructure.com/courses/6051/files/876413?module\_item\_id=274914  
[Accessed 26 10 2025].

Microsoft, 2025. *Learn.Microsoft.* [Online]   
Available at: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios  
[Accessed 26 October 2025].

Mozzila, 2025. *developer.mozilla.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript  
[Accessed 26 October 2025].

Mozzila, 2025. *developer.mozzila.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Learn\_web\_development/Extensions/Async\_JS/Promises  
[Accessed 26 October 2025].

Nemes, A., 2017. *Dotnetcodr.* [Online]   
Available at: https://dotnetcodr.com/2017/05/15/introduction-to-couchdb-with-net-part-1-foundations-and-setup  
[Accessed 26 10 2025].

Postman, 2025. *Postman.* [Online]   
Available at: https://learning.postman.com/docs/introduction/overview/  
[Accessed 26 October 2025].

PouchDB, 2025. *PouchDB.* [Online]   
Available at: https://pouchdb.com/api.html  
[Accessed 26 October 2025].

PouchDB, 2025. *PouchDB.* [Online]   
Available at: https://pouchdb.com/getting-started.html  
[Accessed 26 October 2025].

PouchDB, 2025. *PouchDB.* [Online]   
Available at: https://pouchdb.com/guides/replication.html  
[Accessed October 26 2025].

Redis, NA. *Redis.* [Online]   
Available at: https://redis.io/docs/latest/operate/rc/rc-quickstart/  
[Accessed 24 October 2025].

Web3, 2025. *Web3.* [Online]   
Available at: https://www.w3schools.com/js/js\_htmldom.asp  
[Accessed 26 October 2025].

Wertheim, D., 2015. *GitHub.* [Online]   
Available at: https://github.com/danielwertheim/mycouch  
[Accessed 25 October 25].

Wheeler, D. A., 2011. *dwheeler.com.* [Online]   
Available at: https://dwheeler.com/oss\_fs\_eval.html  
[Accessed 26 Octoboer 2025].

# Appendix A: Acknowledgement of AI use

I acknowledge the use of AI while doing and preparing this report. AI was mainly used for suggestions/research help and debugging some certain issues which are mostly connected to Postman and IBM Cloudant.

**AI TOOL USED:**

ChatGPT-5.0

Publisher: OpenAI

URL: <https://chatgpt.com/>

Version: 5.0

Dates of use: 14/10/2025 – 26/10/2025

**Purpose of AI**

I had to clarify and evaluate some points, but I had no sources, and I could not find anything to help me. I asked AI for help if it can find some articles/blogs on the required topic.

1. **Finding sources to do a task**

Prompt (14/10/2025): “I can't find any information on MyCouch and CouchDB.NET. Can you provide me with some website, blogs or anything that is available or useful as I want to compare them.”

AI response: “Here are some good **resources** where you can learn about and compare the two .NET clients for Apache CouchDB:

**1. MyCouch**

* GitHub repository: “MyCouch is the asynchronous CouchDB client for .NET – builds on top of the asynchronous HTTP client and uses JSON.Net…” [GitHub+2GitHub+2](https://github.com/danielwertheim/mycouch?utm_source=chatgpt.com)
* Blog post by the author (Daniel Wertheim) introducing MyCouch: *MyCouch – the async CouchDb client for .Net*. [danielwertheim](https://danielwertheim.se/mycouch-the-async-couchdb-client-for-net/?utm_source=chatgpt.com)
* A getting-started blog: *Get up and running with CouchDb and C# using MyCouch on Windows*. [danielwertheim+1](https://danielwertheim.se/get-up-and-running-with-couchdb-and-c-using-mycouch-on-windows/?utm_source=chatgpt.com)
* Q&A on StackOverflow regarding configuration/usage of MyCouch. [Stack Overflow](https://stackoverflow.com/questions/24796300/mycouch-client-configuration-and-usage?utm_source=chatgpt.com)
* Medium blog: *A modern approach to working with your data in a .Net Web Api with CouchDB* (uses MyCouch) [Medium+1](https://medium.com/%40josemarmolejos/a-modern-approach-to-working-with-your-data-in-a-net-web-api-with-couchdb-e2228c336a68?utm_source=chatgpt.com)
* NuGet package for a related extension: MyCouch.CloudantIAM to support IBM Cloudant IAM. [NuGet](https://www.nuget.org/packages/MyCouch.CloudantIAM?utm_source=chatgpt.com)

**Key points about MyCouch**:

* Focuses on async operations (uses HttpClient, asynchronous programming) [danielwertheim+1](https://danielwertheim.se/mycouch-the-async-couchdb-client-for-net/?utm_source=chatgpt.com)
* Allows working with raw JSON or POCOs (entities) [danielwertheim+1](https://danielwertheim.se/get-up-and-running-with-couchdb-and-c-using-mycouch-on-windows/?utm_source=chatgpt.com)
* Lightweight rather than trying to mirror full ORM semantics.
* Active community content (blogs, examples) but note: the GitHub repo has been archived (read-only) as of Sept 17 2023. [GitHub](https://github.com/danielwertheim/mycouch?utm_source=chatgpt.com)

**2. CouchDB.NET**

* NuGet package: CouchDB.NET 3.6.1 (targets .NET Standard 2.1) [NuGet+1](https://www.nuget.org/packages/CouchDB.NET?utm_source=chatgpt.com)
* GitHub repository: “EF Core-like CouchDB experience for .NET!” by Matteo Bortolazzo (couchdb-net) [GitHub+1](https://github.com/matteobortolazzo/couchdb-net?utm_source=chatgpt.com)
* Blog post: *CouchDB and C#. EFCore-like queries!* showing how CouchDB.NET uses LINQ-style queries. [DEV Community](https://dev.to/matteobortolazzo/couchdb-and-c-efcore-like-queries-2ma4?utm_source=chatgpt.com)

**Key points about CouchDB.NET**:

* Provides LINQ-style query interface (similar feel to Entity Framework) [GitHub+1](https://github.com/matteobortolazzo/couchdb-net?utm_source=chatgpt.com)
* Supports modern .NET frameworks (.NET 5/6+), .NET Standard 2.1 [NuGet](https://www.nuget.org/packages/CouchDB.NET?utm_source=chatgpt.com)
* More “high-level abstraction” compared to a thin wrapper — gives ORM-like experience for CouchDB.

If you like, I can dig up **some user experiences** (for example from StackOverflow, GitHub issues, Reddit, etc.) specifically comparing these two libraries in real world usage. Would you like me to fetch those?”

1. **Trying to fix Postman issue**

Prompt (26/10/2025) “Why do I get error 401 when I try either GET or POST, is there something missing”

A screenshot of a computer code

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

AI response:” 401 from Cloudant almost always means “bad/insufficient credentials.”  
From your screenshots I can see the issue:

* In Postman you’re using **Basic Auth** with:
  + **username:** apikey
  + **password:** YRpgVGfXs721p5wH2bcJuFgs-DGbY
* In your **Service Credentials** JSON the actual **apikey** value is different (it starts with 21PyQf...).

So the password you’re sending doesn’t match the service credential apikey → Cloudant returns 401.

**Fix (quickest)**

1. In Postman keep **Auth Type = Basic Auth**.
2. Set:
   * **Username:** apikey
   * **Password:** **the exact value of** apikey from your Service Credentials JSON (copy-paste carefully; no spaces).
3. Call:
   * GET https://<your-account>.cloudantnosqldb.appdomain.cloud/\_all\_dbs

That’s all that’s needed. (You don’t use the “username” from the JSON, and you don’t need any extra headers.)

**Common gotchas to double-check**

* URL must match your account domain and region exactly (the one in the JSON url field).
* Remove any old/cached **Authorization** headers so Postman doesn’t send two.
* Don’t mix up apikey with iam\_apikey or some other key you generated elsewhere—use the one in the credentials you’re targeting.
* Your credentials show role **Manager**, which is enough for GET/POST; if you rotate keys, regenerate and use the new apikey.

**Alternative (Bearer token) – if you prefer IAM tokens**

1. Get a token:
   * POST https://iam.cloud.ibm.com/identity/token
   * Body (x-www-form-urlencoded):
     + grant\_type=urn:ibm:params:oauth:grant-type:apikey
     + apikey=<your service apikey>
2. Use it:
   * Add header Authorization: Bearer <token>
   * Call your Cloudant endpoint.

Try with the correct apikey from your JSON (the one starting 21PyQf...) and the 401 should disappear.”

My response: “I tried everyting from the list but I a still not able to communicate and same error si coming up”

AI response: “Gotcha — if you still get **401** after switching to username=apikey and pasting the exact apikey value, let’s isolate the problem with a known-good, minimal test and a couple of things that commonly trip Cloudant up.

**1) Minimal, working test (Basic Auth) — no Postman involved**

Open a terminal and run this **exact** curl (replace the placeholders):

# replace <> with your real values

curl -v -u "apikey:<YOUR\_SERVICE\_CREDENTIALS\_apikey\_VALUE>" \

"https://<YOUR\_INSTANCE\_HOST>.cloudantnosqldb.appdomain.cloud/\_all\_dbs"

* <YOUR\_INSTANCE\_HOST> is the host from your credentials JSON (the long GUID-like one ending in -bluemix), e.g.  
  3403502d-dcaf-4333-9ce4-31dbd136db8f-bluemix
* The **password** is the **exact** apikey field from the Service Credentials JSON (in your screenshot it begins with 21PyQf...).  
  Do **not** use the username field from the JSON and do **not** use iam\_apikey\_id.

**Expected:** HTTP/200 with a JSON array of databases.  
If this works, your credentials are fine and the issue is Postman config (see step 3).

**2) Minimal, working test (Bearer/IAM token)**

If Basic still 401s, try IAM directly:

# 2a) Get an IAM access token

curl -X POST "https://iam.cloud.ibm.com/identity/token" \

-H "Content-Type: application/x-www-form-urlencoded" \

--data "grant\_type=urn:ibm:params:oauth:grant-type:apikey&apikey=<YOUR\_SERVICE\_CREDENTIALS\_apikey\_VALUE>" \

| jq -r .access\_token

Take the token returned and call Cloudant:

# 2b) Use the token

curl -v "https://<YOUR\_INSTANCE\_HOST>.cloudantnosqldb.appdomain.cloud/\_all\_dbs" \

-H "Authorization: Bearer <PASTE\_TOKEN\_HERE>"

**Expected:** HTTP/200.  
If IAM works but Basic doesn’t, something is off in how the Basic header is being built in your client.

**3) If curl works but Postman doesn’t**

* In Postman, **Authorization → Basic Auth**
  + **Username:** apikey
  + **Password:** <your apikey value (the one starting 21PyQf...)>
* Remove any **Authorization** header you might have manually added in **Headers**. Let Postman generate only one.
* Ensure the **URL** is **exactly** the one from the credentials (https://<host>.cloudantnosqldb.appdomain.cloud/\_all\_dbs) and **HTTPS** (not HTTP).
* Disable any Postman **Proxy** or **Bear SSL cert** fiddling for this request.
* No extra apikey: header; Cloudant doesn’t use that header.

**4) Things that often cause 401 with Cloudant**

* **Using the wrong key**: The string you pasted in Postman earlier (YRpgVG...) doesn’t match the apikey value in the credentials (21PyQf...). You must use the **exact** apikey field from the JSON.
* **Whitespace/hidden chars**: When copying the key, ensure there’s no trailing space or newline.
* **Wrong hostname/region**: Use the **host**/**url** from the same credentials block you took the key from. Mixing regions or instances yields 401.
* **Service endpoints**: If the instance is set to **private-only endpoints**, calls from your laptop will fail. In IBM Cloud → your Cloudant instance → **Endpoints**, make sure **Public** is enabled.
* **Rotated/disabled key**: If you regenerated credentials, old keys 401. Create **new service credentials** and use that apikey.
* **Clock skew with IAM** (rare): Only affects Bearer, not Basic. If using tokens, ensure system time is roughly correct.

If you run the two curl tests above:

* **Both 401** → the apikey/host pairing is wrong, the key is disabled, or the instance is private-only.
* **Bearer works, Basic 401** → client is building Basic auth wrong; stick to Bearer or fix Postman.
* **Both 200** → Postman config is the issue (duplicate Authorization header is the most common culprit).

If you can, paste the **exact JSON error body** Cloudant returns with the 401 (it’s usually {"error":"unauthorized","reason":"..."}"). The “reason” string will pinpoint which of the above is biting.”

This suggestion lead me to use CURL as I could not fix Postman even with the help of AI.