CouchDB Project

Luke Foley  
T00224345

Table of Contents

[Importing an Open Data Set 4](#_Toc180769867)

[Dataset Used 4](#_Toc180769868)

[Steps Taken 4](#_Toc180769869)

[Dataset Structure 8](#_Toc180769870)

[CRUD with Postman 8](#_Toc180769871)

[Adding a new document to the DB with POST – Shawshank Redemption: 10](#_Toc180769872)

[Reading the document that was just added with GET 12](#_Toc180769873)

[Updating the document with PUT – setting the address to the document ID, and updating some of the fields: 13](#_Toc180769874)

[Deleting a document using DELETE – deleting revision 2 of the Shawshank redemption document: 15](#_Toc180769875)

[MapReduce: 16](#_Toc180769876)

[Finding a movie (individual document) by move name: 16](#_Toc180769877)

[Find documents in a range: 17](#_Toc180769878)

[\_sum function to get total gross for the top 1000 imdb movies: 20](#_Toc180769879)

[\_count function to count number of movies: 22](#_Toc180769880)

[\_stats of ratings: 24](#_Toc180769881)

[Finding gross revenue by year and genre: 26](#_Toc180769882)

[Using Mango: 29](#_Toc180769883)

[Finding a particular document: 29](#_Toc180769884)

[Documents in Range 29](#_Toc180769885)

[Advantages/disadvantages of Mango/MapReduce 30](#_Toc180769886)

[CouchDB Clients for Python 36](#_Toc180769887)

[CouchDB-Python Client 36](#_Toc180769888)

[Functionality 36](#_Toc180769889)

[Usability 36](#_Toc180769890)

[Performance 36](#_Toc180769891)

[Support 36](#_Toc180769892)

[PyCouchDB Client 36](#_Toc180769893)

[Functionality 36](#_Toc180769894)

[Usability 37](#_Toc180769895)

[Performance 37](#_Toc180769896)

[Support 38](#_Toc180769897)

[Chosen Client 38](#_Toc180769898)

[Recommendation: 38](#_Toc180769899)

[GET Test with CouchDB-Python Client 39](#_Toc180769900)

[CouchDB Replication Protocol 40](#_Toc180769901)

[Crud App 40](#_Toc180769902)

[DBAAS 40](#_Toc180769903)

[Cloudant 40](#_Toc180769904)

# Importing an Open Data Set to CouchDB

## Dataset Used

For this project the following dataset was used: <https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows?resource=download>

This is a dataset of the top 1000 movies on IMDB.

## Steps Taken

A screenshot of a computer

Description automatically generatedFirstly, downloaded and setup CouchDB. Then from the Project Fauxton interface, created a new database called movies:

The dataset downloaded from Kaggle was in CSV format, however, CouchDB requires data in JSON format for importing into its database. In order to convert the dataset to JSON, a simple Python script utilising the pandas library was used from a Jupyter Notebook in PyCharm:

A screen shot of a computer code

Description automatically generated

The pandas library made the conversion process very easy, producing the following file:

A screen shot of a computer

Description automatically generatedA screen shot of a computer program

Description automatically generated

I then attempted to import the new JSON dataset using the bulk import feature in curl. The command resulted in the following error:

A black background with white text

Description automatically generated

This was due to the file not being formatted entirely correctly. The documents in the dataset need to be contained in a “{“docs”:[]} tag. This was appended to the file manually:

A screenshot of a computer program

Description automatically generated

Rerunning the command with the correct tag appended, the import was successful. 1000 documents were added to the database:

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

## Dataset Structure

A screenshot of a computer

Description automatically generated

# CRUD with Postman

After installing Postman, authentication details of the CouchDB database were added:

A screenshot of a computer

Description automatically generated

## Create

Adding a new movie document to the DB with POST – Shawshank Redemption:

A screenshot of a computer

Description automatically generated

Success Code 201. Success message with new ID and REV. The number of documents increased to 1001:

A screenshot of a computer

Description automatically generated

## Read

Reading the movie document that was just added, Shawshank Redemption, with GET. Adding the document ID to the address:

A screenshot of a computer

Description automatically generated

Success code 200. The movie details of Shawshank Redemption are returned.

## Update

Updating the document with PUT. Setting the address to the document ID. Changing some of the fields in the request body:

A screenshot of a computer

Description automatically generated

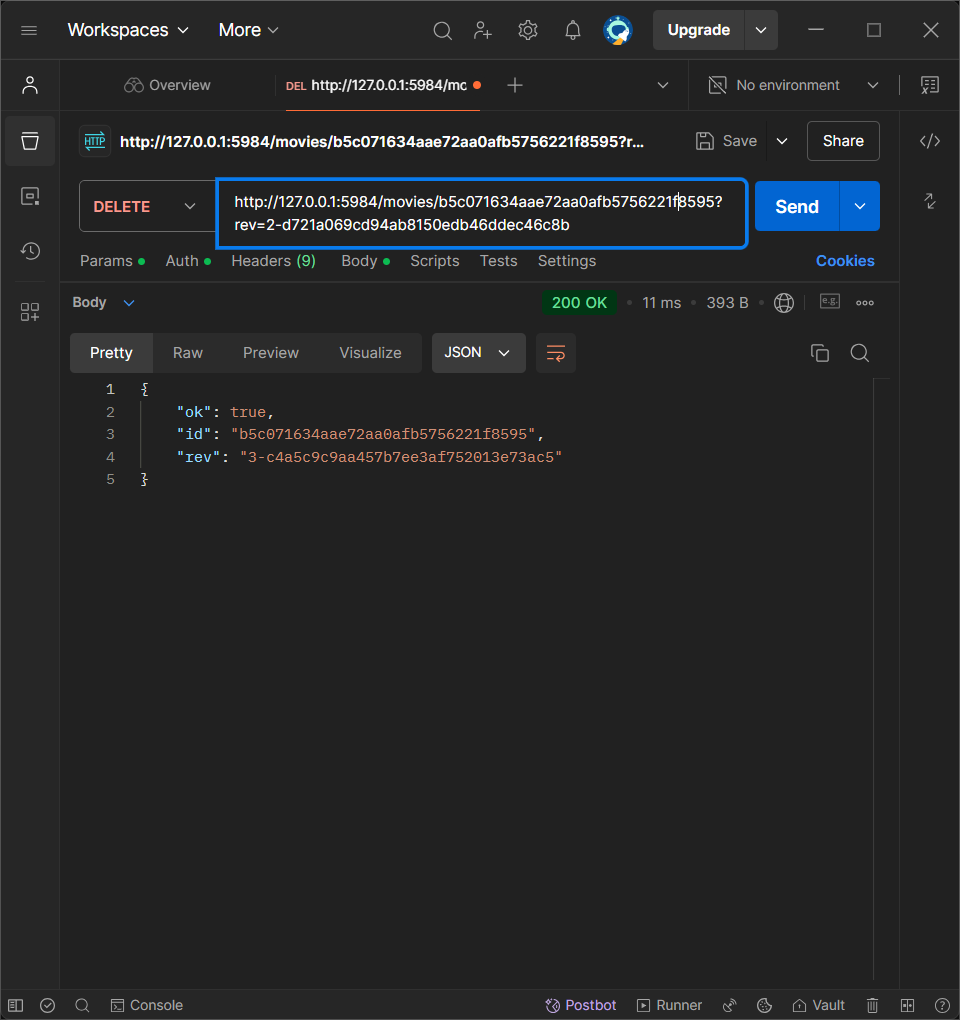
Success code 201, with a new revision for the document:

A screenshot of a computer

Description automatically generated

## Delete

Deleting the Shawshank Redemption document with a DELETE request. Address containing both the ID and the REV of the document:



Success code 200. A confirmation message is returned with a new revision.

# MapReduce:

## Finding a movie (individual document) by move name:

**A screenshot of a computer

Description automatically generated**

This function simply emits each document that matches the queried title.

Test find movie with title of Shawshank redemption:

A screenshot of a computer

Description automatically generated

## Find documents in a range:

create a query to find movies by rating:

A screenshot of a computer

Description automatically generated

Testing by finding movies in a range of ratings between 9 and 10:

A screenshot of a computer

Description automatically generated

The result shows a total of 1000 and an offset of 995, which means that the query resulted with 5 documents, or movies, with ratings between 9 and 10.

## \_sum function to get total gross for the top 1000 imdb movies:



testing in postman:

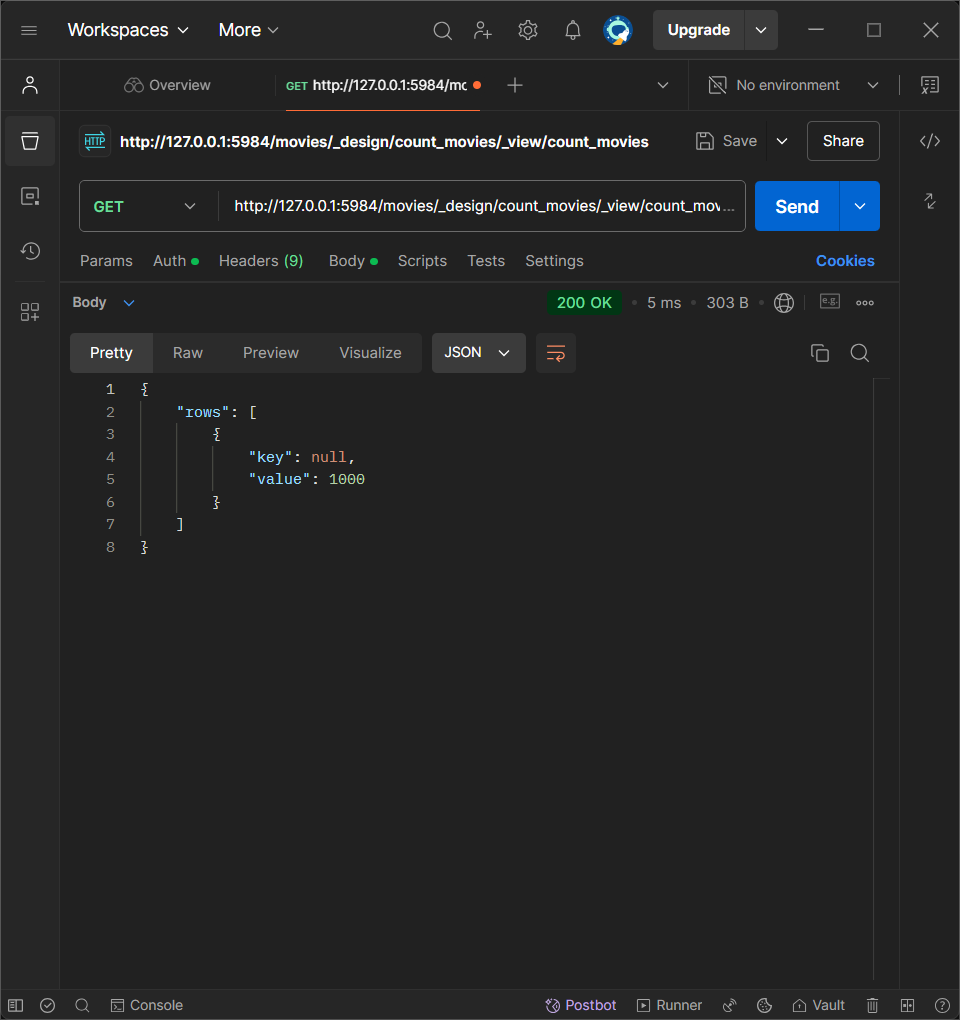


## \_count function to count number of movies:

A screenshot of a computer

Description automatically generated

testing in postman:



## \_stats of ratings:

A screenshot of a computer

Description automatically generated

testing in postman:

A screenshot of a computer

Description automatically generated

## Finding gross revenue by year and genre:

A screenshot of a computer

Description automatically generated

testing level 1 to group by year, in postman:

A screenshot of a computer

Description automatically generated

testing level 2, to group by year and genre:

A screenshot of a computer

Description automatically generated

# Using Mango:

## Finding a particular document:

A screenshot of a computer

Description automatically generated

## Documents in Range

Movies with range greater than or equal to (gte) 9 and less than or equal to (lte) 10:

A screenshot of a computer

Description automatically generated

# Advantages/disadvantages of Mango/MapReduce

When working with \*\*CouchDB\*\*, there are two primary ways to query data: \*\*MapReduce\*\* and \*\*Mango\*\*. Both have their strengths and weaknesses, depending on the complexity of the data, the type of operations needed, and the overall structure of the application.

Let’s break down the \*\*advantages\*\* and \*\*disadvantages\*\* of building queries using \*\*MapReduce\*\* and \*\*Mango\*\*, with examples to help illustrate their differences.

---

## 1. \*\*MapReduce Queries in CouchDB\*\*

### \*\*What is MapReduce?\*\*

MapReduce is a programming model used in CouchDB to process large datasets in a distributed manner. It consists of:

- \*\*Map\*\*: A function that emits key-value pairs from each document.

- \*\*Reduce\*\*: A function that aggregates or summarizes the emitted key-value pairs (optional).

### \*\*Advantages of MapReduce\*\*

#### 1. \*\*Flexible Aggregation\*\*:

- \*\*Example\*\*: If you need to perform complex aggregations, such as summing up the gross revenue of movies by year and genre, MapReduce provides powerful mechanisms to aggregate and transform data.

- \*\*Powerful Reduce Functions\*\*: Functions like \*\*\_sum\*\*, \*\*\_count\*\*, and \*\*\_stats\*\* make it easier to summarize large datasets.

- \*\*Custom Reduce Functions\*\*: You can write custom JavaScript functions to perform complex calculations (e.g., calculating average IMDB ratings per genre).

#### 2. \*\*Efficiency in Handling Large Datasets\*\*:

- \*\*Incremental Updates\*\*: MapReduce views are calculated incrementally, meaning that once a view is created, CouchDB doesn’t need to recalculate everything from scratch when new documents are added. Only new documents are processed, making querying large datasets more efficient.

- \*\*Example\*\*: If you have a large dataset of movies, CouchDB processes and stores results for each movie as it’s added. Future queries become faster since the data doesn’t need to be recalculated each time.

#### 3. \*\*Customizable Keys for Complex Queries\*\*:

- \*\*Example\*\*: You can emit composite keys like `[year, genre]` and query the dataset with different levels of grouping. This makes it possible to create highly customized views of the data.

- \*\*Group and Group Level\*\*: You can group results by partial keys using the `group` and `group\_level` parameters, which are useful for hierarchical data.

### \*\*Disadvantages of MapReduce\*\*

#### 1. \*\*Complexity\*\*:

- \*\*Learning Curve\*\*: Writing MapReduce functions requires knowledge of JavaScript and CouchDB’s query model, which can be daunting for developers not familiar with functional programming.

- \*\*Example\*\*: A simple query like finding all movies released between 2000 and 2010 requires writing a map function and understanding how to structure queries based on emitted keys.

#### 2. \*\*Lack of Flexibility for Ad-Hoc Queries\*\*:

- \*\*Predefined Views\*\*: MapReduce views must be predefined, which limits the ability to perform ad-hoc queries without creating new views.

- \*\*Example\*\*: If you need to find all movies with a runtime of more than 150 minutes, but your MapReduce view only emits year and genre, you’ll have to create a new view or modify the existing one. This isn't efficient for one-time or infrequent queries.

#### 3. \*\*Slower for Simple Queries\*\*:

- \*\*Example\*\*: If you only need to find a specific document by ID or field (like `Series\_Title`), creating a MapReduce view just for that purpose is overkill and can be slower than simply using Mango.

- \*\*View Indexing\*\*: MapReduce views are only efficient after they’ve been created and indexed. For simple queries or small datasets, the overhead of building and maintaining views might not be justified.

---

## 2. \*\*Mango Queries in CouchDB\*\*

### \*\*What is Mango?\*\*

Mango is CouchDB’s declarative query language that allows you to write queries similar to MongoDB. It is simpler and more accessible than MapReduce for querying documents based on field values.

### \*\*Advantages of Mango\*\*

#### 1. \*\*Ease of Use and Simplicity\*\*:

- \*\*Example\*\*: Finding a particular movie by title is straightforward with Mango. The query looks like:

```json

{

"selector": {

"Series\_Title": "The Dark Knight"

}

}

```

- \*\*No JavaScript Knowledge Required\*\*: Unlike MapReduce, Mango doesn’t require you to write functions. Queries are written in JSON, making it much easier for developers familiar with JSON and REST APIs.

#### 2. \*\*Perfect for Ad-Hoc Queries\*\*:

- \*\*On-the-Fly Queries\*\*: Mango is perfect for one-time or ad-hoc queries that don’t require pre-defined views. This is especially useful when you’re exploring the data or performing queries that aren’t repetitive.

- \*\*Example\*\*: If you want to search for movies released between 2000 and 2010, you can just write a query:

```json

{

"selector": {

"Released\_Year": {

"$gte": "2000",

"$lte": "2010"

}

}

}

```

- \*\*No Index Required Initially\*\*: CouchDB will execute the query without requiring predefined views (although creating an index later can improve performance).

#### 3. \*\*Flexible Query Language\*\*:

- \*\*Rich Operators\*\*: Mango supports a variety of operators like `$eq`, `$gt`, `$lt`, `$in`, `$exists`, etc., for more flexible querying.

- \*\*Example\*\*: You can query for all movies with IMDB ratings greater than 8 using:

```json

{

"selector": {

"IMDB\_Rating": {

"$gt": 8

}

}

}

```

#### 4. \*\*Better for Simple Document Retrieval\*\*:

- \*\*Example\*\*: Finding a single document by ID or a specific field is much simpler and faster in Mango than MapReduce.

- \*\*Querying Across Multiple Fields\*\*: Mango allows you to easily filter based on multiple fields, like finding all action movies released between 2000 and 2010 with an IMDB rating greater than 8.

### \*\*Disadvantages of Mango\*\*

#### 1. \*\*Less Efficient for Aggregations\*\*:

- \*\*Limited Aggregation\*\*: Mango doesn’t provide built-in support for advanced aggregations like MapReduce. If you need to calculate sums, counts, or other aggregations, you would need to use MapReduce.

- \*\*Example\*\*: Summing the total gross revenue for each genre cannot be done directly with Mango. You would need to write a MapReduce view for this.

#### 2. \*\*Performance for Large Datasets\*\*:

- \*\*Indexes Needed for Speed\*\*: Mango queries can become slow with large datasets unless indexes are created. While CouchDB can run queries without indexes, it’s not efficient, and performance can degrade with large datasets.

- \*\*Example\*\*: If you frequently query for movies released between 2000 and 2010, you would need to create an index on the `Released\_Year` field to improve query performance.

#### 3. \*\*No Built-In Support for Grouping\*\*:

- \*\*Lack of Grouping\*\*: Mango doesn’t support grouping of data like MapReduce does with `group` and `group\_level`. For example, grouping data by year and genre would require MapReduce.

- \*\*Example\*\*: If you need to calculate the average IMDB rating per genre, Mango won’t be able to handle this easily. MapReduce is required for such operations.

---

### \*\*Conclusion\*\*

| Feature | \*\*MapReduce\*\* | \*\*Mango\*\* |

|--------------------------|-------------------------------------------------------|---------------------------------------------------|

| \*\*Aggregation\*\* | Advanced aggregation via custom reduce functions | No built-in aggregation |

| \*\*Ease of Use\*\* | Requires JavaScript and CouchDB knowledge | Simple, JSON-based queries |

| \*\*Performance\*\* | Efficient for large datasets with pre-defined views | Requires indexes for large datasets |

| \*\*Ad-Hoc Queries\*\* | Not suitable for on-the-fly or exploratory queries | Excellent for on-the-fly queries |

| \*\*Flexibility\*\* | Complex composite keys and grouping (via group\_level) | Flexible for simple document retrieval |

| \*\*Use Case\*\* | Best for complex aggregations, analytics | Best for simple queries and document retrieval |

### Recommendation:

- \*\*Use MapReduce\*\* if you need to perform \*\*complex aggregations\*\*, like summing or counting over large datasets, or if you need \*\*custom logic\*\* in your queries.

- \*\*Use Mango\*\* if you need to run \*\*simple queries\*\* or \*\*ad-hoc searches\*\* based on document fields, or if you are doing exploratory analysis and don’t want to define views upfront.

Let me know if you need further clarification on any of these!

# CouchDB Clients for Python

## CouchDB-Python Client

### Functionality

CRUD Operations: Supports basic CRUD operations for documents.

MapReduce Queries: It fully supports querying CouchDB’s MapReduce views.

Mango Queries: It allows executing Mango queries via the \_find endpoint, which is a great feature for ad-hoc searches.

### Usability

Installation: The library is easily installed via pip, and the API is straightforward for connecting to the CouchDB server and accessing databases.

Usage: Connecting to the server and running queries like retrieving documents is simple and requires minimal setup. For example, retrieving all documents in a database is as simple as:

for doc\_id in db:

print(db[doc\_id])

### Performance

The client is lightweight and efficient for simple operations like fetching documents and performing queries. CouchDB’s native features, like view indexing and incremental MapReduce, help scale performance for larger datasets.

Batch Operations: It supports batch document uploads, which can improve performance when dealing with large datasets.

### Support

Documentation: The CouchDB-Python documentation is comprehensive, providing examples for all common tasks like CRUD operations, MapReduce queries, and error handling.

Community Support: While not as extensive as more widely-used libraries, there is reasonable support on GitHub and StackOverflow.

## PyCouchDB Client

### Functionality

CRUD Operations

PyCouchDB supports all basic CRUD operations: creating, reading, updating, and deleting documents.

MapReduce Queries

PyCouchDB allows you to interact with CouchDB views (which are essentially MapReduce queries). You can run queries against pre-defined views using the query() method.

Mango Queries

Unlike CouchDB-Python, PyCouchDB does not natively support Mango queries (CouchDB’s declarative query language). This could be a limitation if you need to perform ad-hoc queries on your data without pre-defining views.

Additional Features

Bulk Operations: PyCouchDB does not offer advanced support for bulk document operations, which are useful for uploading large datasets efficiently.

Replication and Advanced Features: PyCouchDB is focused on simplicity and does not natively support more advanced CouchDB features such as replication, document conflict handling, or session management.

### Usability

Installing PyCouchDB is simple and can be done using pip

PyCouchDB is designed to provide a simple interface for interacting with CouchDB’s RESTful API. Here’s an example of how to connect to a CouchDB server and retrieve documents:

import pycouchdb

server = pycouchdb.Server("http://127.0.0.1:5984/")

db = server.database('movies')

for doc in db.all():

print(doc)

Connection: Connecting to the server is straightforward.

Basic Operations: PyCouchDB offers simple methods for CRUD operations (Create, Read, Update, Delete). The API is intuitive, and operations are executed in just a few lines of code.

Rating: The client is very lightweight and easy to use. It is ideal for simple CouchDB interactions, making it accessible to data scientists or developers who prefer minimalistic tools.

### Performance

Performance Characteristics

Efficiency: PyCouchDB performs well for small to medium datasets, as it is very lightweight. For basic operations like retrieving documents or creating new ones, it’s fast and efficient.

MapReduce: PyCouchDB supports querying CouchDB views (MapReduce) using the query() method. However, it does not offer specialized support for incremental view indexing or optimizing query performance for larger datasets, as CouchDB-Python does.

Large Datasets: For large datasets, performance may depend more on how CouchDB is optimized (e.g., using CouchDB’s built-in indexing) rather than the client itself.

Rating: The client performs well for small-scale projects or one-off queries but lacks advanced performance features for large datasets, such as bulk document operations or efficient incremental querying.

### Support

Documentation

The official documentation is somewhat sparse compared to more widely used libraries like CouchDB-Python. However, basic operations are well-documented, and you can find code examples on the GitHub repository. For advanced use cases or troubleshooting, you may have to explore CouchDB’s API directly or rely on trial and error.

Community Support

Since PyCouchDB is not as widely used as CouchDB-Python, the community support is smaller. There are fewer online resources, blog posts, or tutorials available, and support is limited on forums like StackOverflow or GitHub Issues.

Rating: The documentation is sufficient for simple use cases, but it lacks depth, especially for more advanced features like querying with MapReduce. Community support is limited compared to CouchDB-Python, which could be a downside if you need help.

## Chosen Client

### Recommendation:

Based on the evaluation of the CouchDB-Python client, I recommend using this client for the project because:

It is easy to set up and use, making it suitable for data scientists.

It provides full support for CouchDB features, including MapReduce and Mango queries.

It performs well for both small and medium-sized datasets and scales well with CouchDB’s built-in optimizations.

### GET Test with CouchDB-Python Client

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# CouchDB Replication Protocol

## Crud App

A simple node.js express app was made for this section. This consists of 3 pages:

# DBAAS

## Cloudant

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated