Advanced Database

CouchDB Project

Emmnuella G. Nyamekye

Table of Contents

[Summary 4](#_Toc212410347)

[Introduction 4](#_Toc212410348)

[CRUD on Kaggle Dataset 5](#_Toc212410349)

[Importing CSV to CouchDB 5](#_Toc212410350)

[CRUD on Database using Curl on Command Line 8](#_Toc212410351)

[MapReduce Queries 11](#_Toc212410352)

[Finding one particular document 11](#_Toc212410353)

[Find documents in a Range 13](#_Toc212410354)

[\_sum Example 14](#_Toc212410355)

[\_count Example 14](#_Toc212410356)

[\_stats Example 15](#_Toc212410357)

[Mongo Queries 17](#_Toc212410358)

[Find a particular document 17](#_Toc212410359)

[Finding documents between the year 2010 and 2011 18](#_Toc212410360)

[Advantages and disadvantages of writing NoSQL queries using CouchDB Views, Mango and MongoDBs MQL 19](#_Toc212410361)

[1. CouchDB Views (MapReduce) 19](#_Toc212410362)

[2. MongoDB Query Language (MQL) 19](#_Toc212410363)

[3. CouchDB Mango Query Language 19](#_Toc212410364)

[Python client evaluation, recommendation, and application integration 20](#_Toc212410365)

[couchdb-pyhton 20](#_Toc212410366)

[py\_couchdb 22](#_Toc212410367)

[Comparing couchdb-python to py-couchdb 23](#_Toc212410368)

[Small JavaScript application with CRUD commands 24](#_Toc212410369)

[How CouchDB replication protocol works 24](#_Toc212410370)

[index.html 25](#_Toc212410371)

[App.js code 30](#_Toc212410372)

[Create 35](#_Toc212410373)

[Read and Update 37](#_Toc212410374)

[Delete 39](#_Toc212410375)

[DBaas, Capella & Postman 40](#_Toc212410376)

[What is DBaaS 40](#_Toc212410377)

[Postman Test 45](#_Toc212410378)

[References 48](#_Toc212410379)

[Appendix 49](#_Toc212410380)

[Prompt for Importing CSV file in CouchDB 49](#_Toc212410381)

[Output for Importing CSV file in CouchDB 49](#_Toc212410382)

[Code difference 50](#_Toc212410383)

# Summary

This project shows end to end use of CouchDB as a NoSQL document store accessed from Python and a small JavaScript front end. An open dataset called “Board Games Dataset Complete Features” was downloaded from Kaggle (<https://www.kaggle.com/datasets/chik0di/board-games-dataset-complete-features>), transformed into JSON documents, and imported into a local CouchDB instance, with the following credentials; username admin and password hello123.

The work includes: data import (Python script + CouchDB API), examples of CRUD using Curl, MapReduce view implementations (find by id, range queries, and use of \_sum, \_count, \_stats), Mango queries that follow the MapReduce examples, a custom MapReduce that groups results with group / group\_level, an illustration of how Redis can be used as a caching layer, an offline first JavaScript app using PouchDB that synchronizes with CouchDB, and testing replication with a cloud DBaaS (Capella) via Postman.

The report documents methods, code snippets, results and learning points.

# Introduction

NoSQL document stores like CouchDB are generally used for flexible, schema-free data storage, offline first applications, and distributed replication. CouchDB stores JSON documents, using RESTful HTTP API, and provides MapReduce views (server-side JavaScript map/reduce functions) and Mango (a declaration JSON query language).

GitHub link: <https://github.com/EmmaNyamekye/CouchDB_Project>

# CRUD on Kaggle Dataset

The dataset downloaded from Kaggle contains detailed data of thousands of board games.

Each record represents a board game and includes the following attributes: boardgame, release\_year, min\_players, max\_players, min\_playtime, max\_playtime, minimum\_age, avg\_rating, num\_ratings, complexity, rank\_overall, owned, wishlisted, total\_plays, fans, page\_views, amazon\_price, std\_deviation, comments, monthly\_plays, previously\_owned, for\_trade, want\_trade, categories, mechanics, families, designers, artists, publishers, URL, and description.

## Importing CSV to CouchDB

To import the dataset, a Python script was written, the script reads the CSV file and uploads it to CouchDB using its REST API. The script uses the request library to interact with the database.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure Code for Importing Dataset to CouchDB

A screenshot of a computer

AI-generated content may be incorrect.

Figure boardgames database before importing Dataset

A screenshot of a computer

AI-generated content may be incorrect.

Figure boardgames database after importing Dataset

Below is an example of a single document structure after import:

A grey rectangular object with a black border

AI-generated content may be incorrect. A computer screen shot of text

AI-generated content may be incorrect. A grey background with green text

AI-generated content may be incorrect.

Figure Example of JSON Document in CouchDB

When a document is created and sent to the database, CouchDB automatically adds two mandatory fields:

* \_id: Unique identifier (like a primary key) that lets you find the document
* \_rev: Revision ID, which is initially 1-hash value

The \_rev changes every time an update or delete is done successfully on the document.

## CRUD on Database using Curl on Command Line

A screenshot of a computer program

AI-generated content may be incorrect.

Figure Creating a new JSON Document on boardgames database in CouchDB

Executing curl -X POST request to the /boardgames database, sending a JSON document for a new boardgame called “Sky Traders (by Emma)”.

Since POST was used, CouchDB automatically generated a unique document \_id.

A confirmation message with the new \_id and \_rev was returned {"ok":true,"id":"7f5e742a84ce26ebc0cc96e70b7d30d4","rev":"1-f8e2843a92895df17215a2306a1c5b40"}

“rev":"1-hash” indicates the documents first version.

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display of new Data stored in boardgames database

A screen shot of a computer

AI-generated content may be incorrect.

Figure Read data from id '7f5e742a84ce26ebc0cc96e70b7d30d4'

Executing curl -X GET request, which is the standard HTTP method for retrieving data.

The request is trying to find the document previously created \_id: 7f5e742a84ce26ebc0cc96e70b7d30d4.

Complete JSON document for boardgame “Sky Traders (by Emma)” is returned.

A computer screen with white text

AI-generated content may be incorrect.

Figure Update boardgame name from “Sky Traders by Emma” to “Sky Traders: Aerial Empires”

Executing a curl -X PUT request sends the full updated document, including the old \_rev (1-hash), to the database. The database successfully replaces the old document version, and it return, a new \_rev (starting with 2-hash), completing the document update, changing the boardgame from “Sky Traders (by Emma)” to “Sky Traders: Aerial Empires”.

A computer screen with text

AI-generated content may be incorrect.

Figure Delete JSON Document with "id":"7f5e742a84ce26ebc0cc96e70b7d30d4" and "rev":"1-f8e2843a92895df17215a2306a1c5b40"

Executed a curl -X DELETE request, along with the \_id and the latest \_rev (2-hash), permanently deleting “Sky Traders: Aerial Empires” from the database. CouchDB records this action by returning a new \_rev (starting with 3-hash).

A screenshot of a computer

AI-generated content may be incorrect.

Figure CouchDB display showing that id":"7f5e742a84ce26ebc0cc96e70b7d30d4 has been deleted

# MapReduce Queries

## Finding one particular document

A computer screen with text on it

AI-generated content may be incorrect.

Figure Creation of a new design document called \_design/find\_game, in the boardgames database, to view id document by boardgame value

The curl command PUTs a Design Document named \_design/find\_game into the boardgames database. This document contains a map function that defines a new View called by\_boardgame.

It sets up a permanent index that will allow users to quickly look up any document in the database using the value of its boardgame field.

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display of Document with boardgame = Twilight Imperium: Fourth Edition

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure Output of using \_design/find\_game design document

The command accessed the by\_boardgame View in the boardgames database and successfully retrieved the data for a document where the kye matches “Twilight Imperium: Forth Edition”. The output is a large JSON response containing all the detailed information about this specific boardgame.

## Find documents in a Range

A computer screen with numbers and letters

AI-generated content may be incorrect.

Figure Creation of a new design document, called \_design/by\_year, in the boardgames database, for viewing documents by specific year range

The curl command is performing a PUT request to create a new Design Document named \_design/by\_year in the boardgames database. This document contains a View called by\_release\_year. This view is designed to:

* Index every document that has a release\_year field
* Use the numeric value of the release\_year as the key

A screen shot of a computer

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

AI-generated content may be incorrect.

Figure Using design document to find all boardgames

released between 2010 and 2012

## \_sum Example

A computer screen with a black background

AI-generated content may be incorrect.

Figure Creation of a new design document, called ratings\_sum, for Adding Total Ratings

A MapReduce view named total\_ratings was created. The \_sum reduce function was used to calculate the grand total of all user ratings across entire database.

A computer screen with a black background

AI-generated content may be incorrect.

Figure Returning a JSON result that shows the total number of ratings accumulated from all documents

When executed, it returns the grand total of all user ratings across the database, which is 18,489,828

## \_count Example

A computer screen with text on it

AI-generated content may be incorrect.

Figure Creation of a new design document, called count\_games that counts the total number of documents

The curl command creates a new Design Document named \_design/count\_games containing a MapReduce View called total\_games. This view uses the built-in \_count reduce function, which counts every document that has a boardgame field.

A computer screen with a black screen

AI-generated content may be incorrect.

Figure Returning a JSON result that shows the total number of boardgames in Database

## \_stats Example

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure Creation of a new design document, called stats\_ratings in boardgames database, that calculate the sum, count, minimum, maximum, and sum of the num\_ratings

The built-in \_stats reducer is used within a MapReduce view to calculate a set of statistical metrics on numerical data emitted by the map function. It provides:

* Sum: the total sum of all emitted values
* Count: the number of values processed
* Min: the smallest value in the set
* Max: the largest value in the set
* Sumsqr: the sum of the squares of all values

In this case the \_stats reducer is applied to the num\_ratings field of boardgame documents, the results are grouped by the release\_year.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure The curl command successfully queries the CouchDB statistical view (sum, count, min, max, sum of ratings)

# Mongo Queries

Find a particular document A screen shot of a computer

AI-generated content may be incorrect.

Figure Connecting to a local MongoDB instance and executing a query on the boardgames Database to retrieve the detailed document for the boardgame "Twilight Imperium: Fourth Edition"

* The query searches for a document where the boardgame field matches the given title
* The result shows a full document of the matching boardgame, with \_id, boardgame, release\_year, min\_players, max\_players, avg\_rating, and more

## Finding documents between the year 2010 and 2011

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure Display of a MongoDB range query executed on the boardgames collection to retrieve the \_id and boardgame fields for all documents from Database with a release\_year between 2010 and 2011

* $gte: 2010 means greater than or equal to 2010
* $lte: 2011 means less than or equal to 2011
* { \_id: 1, boardgame: 1} specifies that only the \_id and boardgame should be displayed
* The output lists several matching documents

# Advantages and disadvantages of writing NoSQL queries using CouchDB Views, Mango and MongoDBs MQL

## CouchDB Views (MapReduce)

Advantages:

* Efficient Aggregation: Ideal for complex, cached, server-side totals and statistics (Example: The \_sum and \_starts screenshots)
* Built-in \_sum and \_count functions

Disadvantages:

* Complexity: Requires writing custom JavaScript functions for simple index tasks
* Low flexibility

## MongoDB Query Language (MQL)

Advantages:

* Highly flexible, allows filtering on any field without pre-defined indices (Example: “Twilight Imperium” query, using db.boardmahe.find())
* Easy use of range operator (Example, $gte: 2010, $lte: 2011)
* Queries are easy to write, simple JSON format

Disadvantages:

* Slow for large datasets especially if the fields do not have indices

## CouchDB Mango Query Language

Advantages:

* Uses simple JSON syntax, making it easy to write
* Provides similar queries to MongoDB
* Allows creation of secondary indexes for better performance

Disadvantages:

* Lack of built-in reducers like \_sum and \_stats
* Since no permanent index exists for the queried fields the database must build a temporary internal index, making the query slow

# Python client evaluation, recommendation, and application integration

## couchdb-pyhton

A computer screen with white text

AI-generated content may be incorrect.

Figure Connecting to CouchDB using Python and inserting a sample document into the "library" database

* Import the library and create a Server object to connect to CouchDB, allowing for default local connection of a remote server address, port, and Authentication credentials
* Creation of new database (couch.create(‘test’)) or access an existing database (couch[‘mydb’])
* Document creation of a basic JSON document (doc = {‘foo’: ‘bar’}) is created and saved using db.save(doc)
* The output confirms that CouchDB automatically assigns a unique \_id and the mandatory \_rev to the document on a successful save.

To find all documents, iterate over the database:



Figure Displaying all existing documents using in operator, deleting document and database

A computer screen with white text

AI-generated content may be incorrect.

Figure Basic CouchDB API: couchdb.client

* Connection to CouchDB server using server = Server()
* Creation of new database called ‘python-tests’
* A new JSON document ({‘type’:’Person’,’name’:’John Doe’}) is inserted into the database
* The document is retrieved using doc = db[doc\_id]
* The data is retrieved in accessed using standard Pyhton syntax (doc[‘type’]) and (doc[‘name’])
* The document is deleted from the database using its ID: del db[doc.id]
* A check (doc.id in db) confirms the document no longer exist in the database by returning False
* The entire ‘python-tests’ database is deleted from the server: del server[‘python-tests]

A computer code on a black background

AI-generated content may be incorrect.

Figure Server Commands

* Database object is created by calling the create() method on a Server object,   
  db = server.create('python-tests')
* New documents can be added using the save() method
* A specific document ID using db['JohnDoe'] = {'type': 'person', 'name': 'John Doe'}
* Documents are retrieved from the database by their ID using item access doc = db[doc\_id]
* To update an existing document, you modify the document object and then save it back to the database using item access doc[‘name’] = ‘Maria Rossi’ followed by db[doc.id] = doc
* The in operator is used to check for the existence of a documents in the database

## py\_couchdb

A computer screen with text on it

AI-generated content may be incorrect.

Figure py-couchdb quick Start

* The library is imported with import pycouchdb
* A connection to the CouchDB server is established by providing the full URL including credentials:  
  server = pycouchdb.Server("http://admin:hello123@localhost:5984/")
* A database is created using a direct method called: db = server.create("foo")
* An existing database is accessed using: db = server.database("foo")
* A database is deleted using: server.delete("foo2")

A screen shot of a computer

AI-generated content may be incorrect.

Figure Check document existence

* To check for document existence, the in operator is used.  
  "b3a3b0063139476ea6e7dc71f0c27544" not in db  
  False -> meaning the document is present in the database
* A document is deleted using: db.delete("b3a3b0063139476ea6e7dc71f0c27544")
* The next check confirms the deletion:  
  "b3a3b0063139476ea6e7dc71f0c27544" not in db  
  True -> meaning the document is no longer present in the database

## Comparing couchdb-python to py-couchdb

|  |  |  |
| --- | --- | --- |
| Feature | couchdb-python | py-couchdb |
| Support | This is the official Apache CouchDB client and is maintained by code project contributors. | This is a community project. Long term support and maintenance may be less consistent. |
| Useability | Uses familiar Python idioms, making the code clean and concise. | Uses clear method, which can be easy for beginners. |
| Security | Supports secure HTTPS connections with credentials directly in the URL. | Supports secure HTTPS connections with credentials directly in the URL. |

# Small JavaScript application with CRUD commands

## How CouchDB replication protocol works

CouchDB replication protocol is an incremental HTTP-based synchronization protocol that involves two databases (a source and a destination).

The goal of replication is for all data in the source database to be present in the destination database at the end of the operation, as well as for all deleted data from the source database to be removed from the destination database. The protocol supports one-way (push/pull) and continuous replication.

CouchDB supports two kinds of replication:

* Transient replication is temporary; it’s triggered manually and disappears once the server restarts
* Persistent replication is stored inside a special \_replicator database. This type of replication automatically restarts after a server reboot.

When replication runs, CouchDB compares the source and destination database to find what is different between the two. It reads the changes feed from the source and sends only the missing or updates revisions to the destination in batches.

Replication is also the mechanism that allows CouchDB to synchronize data to client-side database.

Tools like PouchDB implement the same replication algorithm in JavaScript, making it possible for web apps to work offline first.

## index.html

<!doctype html>

<html>

<head>

    <meta charset="utf-8" />

    <title>Boardgame Collection</title>

<meta name="viewport" content="width=device-width,initial-scale=1" />

<style>

 body{

   font-family:'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;

   background-color: #FFFDD0;

   padding: 20px;

 }

 h1{

   color: #964B00;

   text-align: center;

 }

 .status{

   text-align: center;

   font-weight: bold;

   margin-bottom: 1rem;

 }

 .controls{

   width: 90%;

   margin: auto;

   display: flex;

   flex-wrap: wrap;

   gap: 0.5rem;

   margin-bottom: 1rem;

 }

 .controls input,

 .controls select,

 .controls textarea,

 .controls button {

   flex: 1 1 48%;

   min-width: 150px;

   border: 1px solid #964B00;

   border-radius: 10px;

   padding: 0.4rem 10px;

   box-sizing: border-box;

 }

 .controls textarea {

   flex-basis: 100%;

   height: 20vh;

 }

 .controls button {

   flex-basis: 48%;

   background-color: #964B00;

   color: #FFFDD0;

   cursor: pointer;

 }

 .controls button:hover {

   background-color: #FFFDD0;

   color: #964B00;

 }

 ul#boardgames{

   list-style:none;

   padding:0;

   width: 90%;

   margin: auto;

 }

 ul#boardgames li{

   border: 1px solid #964B00;

   border-radius: 10px;

   padding: 10px;

   margin-bottom: 10px;

   background-color: #FFF8E1;

 }

 ul#boardgames li button{

   margin-top: 5px;

   background-color: crimson;

   color: white;

   border: none;

   padding: 4px 8px;

   border-radius: 6px;

   cursor: pointer;

 }

 ul#boardgames li button:hover{

   background-color: #ff6666;

 }

 table {

  width: 90%;

  margin: 2rem auto;

  border-collapse: collapse;

  background-color: #FFF8E1;

  border: 1px solid #964B00;

  border-radius: 10px;

  overflow: hidden;

 }

 th, td {

  border: 1px solid #964B00;

  padding: 8px 10px;

  text-align: center;

 }

 th {

  background-color: #964B00;

  color: #FFFDD0;

 }

 td button {

  background-color: crimson;

  color: white;

  border: none;

  border-radius: 6px;

  padding: 4px 8px;

  cursor: pointer;

 }

 td button:hover {

  background-color: #ff6666;

 }

</style>

</head>

<body>

<h1>Boardgame Collection</h1>

<div class="status" id="syncStatus"></div>

<div class="controls">

 <input id="title" placeholder="Boardgame Name"/>

 <input id="year" placeholder="Release Year"/>

 <select id="min\_players">

   <option value="">Min Players</option>

   <option value="1">1</option>

   <option value="2">2</option>

   <option value="3">3</option>

   <option value="4">4</option>

   <option value="5">5</option>

   <option value="6">6</option>

 </select>

 <select id="max\_players">

   <option value="">Max Players</option>

   <option value="1">1</option>

   <option value="2">2</option>

   <option value="3">3</option>

   <option value="4">4</option>

   <option value="5">5</option>

   <option value="6">6</option>

 </select>

 <select id="min\_playtime">

   <option value="">Min Playtime</option>

   <option value="15">15 min</option>

   <option value="30">30 min</option>

   <option value="45">45 min</option>

   <option value="60">60 min</option>

   <option value="90">90 min</option>

 </select>

 <select id="max\_playtime">

   <option value="">Max Playtime</option>

   <option value="30">30 min</option>

   <option value="60">60 min</option>

   <option value="90">90 min</option>

   <option value="120">120 min</option>

   <option value="180">180 min</option>

 </select>

 <select id="minimum\_age">

   <option value="">Min Age</option>

   <option value="6">6</option>

   <option value="8">8</option>

   <option value="10">10</option>

   <option value="12">12</option>

   <option value="14">14</option>

   <option value="16">16</option>

 </select>

 <textarea id="description" placeholder="Description"></textarea>

 <button id="addBtn">Add Boardgame</button>

</div>

<table id="boardgameTable">

 <thead>

  <tr>

   <th>Title</th>

   <th>Year</th>

   <th>Min Players</th>

   <th>Max Players</th>

   <th>Min Playtime</th>

   <th>Max Playtime</th>

   <th>Min Age</th>

   <th>Actions</th>

  </tr>

 </thead>

 <tbody id="boardgameTableBody">

    </tbody>

</table>

<script src="https://unpkg.com/pouchdb@7.3.0/dist/pouchdb.min.js"></script>

<script src="app.js"></script>

</body>

</html>

## App.js code

const db = new PouchDB('boardgames');

const remoteDB = new PouchDB('http://localhost:5984/boardgames', {

  auth: {

    username: 'admin',

    password: 'hello123'

  }

});

const addBtn = document.getElementById('addBtn');

const statusDiv = document.getElementById('syncStatus');

const tableBody = document.getElementById('boardgameTableBody');

let editingId = null;

let editingRev = null;

async function displayBoardgames() {

  const result = await db.allDocs({ include\_docs: true });

  const boardgames = result.rows

    .map(r => r.doc)

    .sort((a, b) => (a.created\_at || a.\_id).localeCompare(b.created\_at || b.\_id))

    .slice(0, 10);

  tableBody.innerHTML = '';

  boardgames.forEach(doc => {

    const row = document.createElement('tr');

    row.innerHTML = `

      <td>${doc.boardgame || ''}</td>

      <td>${doc.release\_year || ''}</td>

      <td>${doc.min\_players || ''}</td>

      <td>${doc.max\_players || ''}</td>

      <td>${doc.min\_playtime || ''}</td>

      <td>${doc.max\_playtime || ''}</td>

      <td>${doc.minimum\_age || ''}</td>

      <td>

        <button onclick="deleteBoardgame('${doc.\_id}', '${doc.\_rev}')">Delete</button>

        <button onclick="syncSingleBoardgame('${doc.\_id}')">Sync</button>

        <button onclick="editBoardgame('${doc.\_id}')">Update</button>

      </td>

    `;

    tableBody.appendChild(row);

  });

}

addBtn.addEventListener('click', async () => {

  const boardgame = document.getElementById('title').value.trim();

  const yearRaw = document.getElementById('year').value.trim();

  const minPlayersRaw = document.getElementById('min\_players').value;

  const maxPlayersRaw = document.getElementById('max\_players').value;

  const minPlaytimeRaw = document.getElementById('min\_playtime').value;

  const maxPlaytimeRaw = document.getElementById('max\_playtime').value;

  const minAgeRaw = document.getElementById('minimum\_age').value;

  const description = document.getElementById('description').value.trim();

  const year = parseInt(yearRaw);

  const min\_players = parseInt(minPlayersRaw);

  const max\_players = parseInt(maxPlayersRaw);

  const min\_playtime = parseInt(minPlaytimeRaw);

  const max\_playtime = parseInt(maxPlaytimeRaw);

  const minimum\_age = parseInt(minAgeRaw);

  const currentYear = new Date().getFullYear();

  if (!boardgame || !yearRaw || !minPlayersRaw || !maxPlayersRaw || !minPlaytimeRaw ||

      !maxPlaytimeRaw || !minAgeRaw || !description) {

    statusDiv.textContent = "Please enter every value.";

    return;

  }

  if (isNaN(year) || year < 1980 || year > currentYear) {

    statusDiv.textContent = `Year must be between 1980 and ${currentYear}.`;

    return;

  }

  if (isNaN(min\_players) || isNaN(max\_players) || min\_players >= max\_players) {

    statusDiv.textContent = "Min players must be less than max players.";

    return;

  }

  if (isNaN(min\_playtime) || isNaN(max\_playtime) || min\_playtime >= max\_playtime) {

    statusDiv.textContent = "Min playtime must be less than max playtime.";

    return;

  }

  if (isNaN(minimum\_age)) {

    statusDiv.textContent = "Minimum age must be a number.";

    return;

  }

  try {

    const doc = {

      boardgame,

      release\_year: year,

      min\_players,

      max\_players,

      min\_playtime,

      max\_playtime,

      minimum\_age,

      description,

      created\_at: new Date().toISOString()

    };

    if (editingId) {

      doc.\_id = editingId;

      doc.\_rev = editingRev;

      await db.put(doc);

      await remoteDB.put(doc);

      statusDiv.textContent = `"${boardgame}" updated successfully!`;

    } else {

      doc.\_id = new Date().toISOString();

      await db.put(doc);

      statusDiv.textContent = `"${boardgame}" added successfully!`;

    }

    resetForm();

    displayBoardgames();

  } catch (err) {

    console.error("Save error:", err);

    statusDiv.textContent = "Error saving boardgame.";

  }

});

function resetForm() {

  document.getElementById('title').value = '';

  document.getElementById('year').value = '';

  document.getElementById('description').value = '';

  document.getElementById('min\_players').value = '';

  document.getElementById('max\_players').value = '';

  document.getElementById('min\_playtime').value = '';

  document.getElementById('max\_playtime').value = '';

  document.getElementById('minimum\_age').value = '';

  editingId = null;

  editingRev = null;

  addBtn.textContent = "Add Boardgame";

}

async function editBoardgame(id) {

  try {

    const doc = await db.get(id);

    document.getElementById('title').value = doc.boardgame || '';

    document.getElementById('year').value = doc.release\_year || '';

    document.getElementById('description').value = doc.description || '';

    document.getElementById('min\_players').value = doc.min\_players || '';

    document.getElementById('max\_players').value = doc.max\_players || '';

    document.getElementById('min\_playtime').value = doc.min\_playtime || '';

    document.getElementById('max\_playtime').value = doc.max\_playtime || '';

    document.getElementById('minimum\_age').value = doc.minimum\_age || '';

    editingId = doc.\_id;

    editingRev = doc.\_rev;

    addBtn.textContent = "Update Boardgame";

    statusDiv.textContent = `Editing "${doc.boardgame}"...`;

  } catch (err) {

    console.error("Edit error:", err);

    statusDiv.textContent = "Error loading boardgame for editing.";

  }

}

async function deleteBoardgame(id, rev) {

  try {

    await db.remove(id, rev);

    statusDiv.textContent = "Boardgame deleted.";

    displayBoardgames();

  } catch (err) {

    console.error("Delete error:", err);

    statusDiv.textContent = "Error deleting boardgame.";

  }

}

window.deleteBoardgame = deleteBoardgame;

window.syncSingleBoardgame = async function(id) {

  try {

    const doc = await db.get(id);

    await remoteDB.put(doc);

    statusDiv.textContent = `Synced "${doc.boardgame}" to remote DB.`;

  } catch (err) {

    console.error("Sync error:", err);

    statusDiv.textContent = "Sync error.";

  }

};

window.editBoardgame = editBoardgame;

displayBoardgames();

db.sync(remoteDB, { live: true, retry: true })

  .on('change', info => {

    console.log("Live sync change:", info);

    displayBoardgames();

  })

  .on('error', err => {

    console.error("Live sync error:", err);

  });

A screenshot of a board game

AI-generated content may be incorrect.

Figure Initial view of the web application

* The top part of the web application contains a form that allows users to add a new boardgame
* The bottom part of the web application displays a table where users can view the most recent boardgames in the Database, as well as Delete or Update them. In cases where synchronization fails the Sync button can be clicked to synchronize manually.

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display in CouchDB where most recent boardgames match the ones on the web application

## Create

A screenshot of a board game

AI-generated content may be incorrect.

Figure Add new boardgame called "Poetry for Neanderthals"

A screenshot of a board game

AI-generated content may be incorrect.

Figure New view of the web application where the new boardgame "Poetry for Neanderthals" is now the most recent boardgame

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display in CouchDB where most recent boardgame is now "Poetry for Neanderthals"

## Read and Update

A screenshot of a game

AI-generated content may be incorrect.

Figure Retrieve "Poetry for Neanderthals" data for update

A screenshot of a computer

AI-generated content may be incorrect.

Figure View of the new boardgame in CouchDB

A screenshot of a board game

AI-generated content may be incorrect.

Figure Success message for updated boardgame

A screenshot of a computer

AI-generated content may be incorrect.

Figure View of the updated boardgame in CouchDB

## Delete

A screenshot of a board game

AI-generated content may be incorrect.

Figure Success message for deleted boardgame

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display in CouchDB where "Poetry for Neanderthals" is no longer present after deletion

# DBaas, Capella & Postman

## What is DBaaS

DBaaS (Database-as-a-Service) provides a fully managed cloud database that can be accessed online without having to worry about installation, scalability, backup or maintenance.

A screenshot of a computer

AI-generated content may be incorrect.

Figure Creating new Cluster called boardgames\_cluster

A screenshot of a computer

AI-generated content may be incorrect.

Figure Cluster boardgames\_cluster Dashboard

A screenshot of a computer

AI-generated content may be incorrect.

Figure Load CSV file into boardgames\_cluster

A screenshot of a computer

AI-generated content may be incorrect.

Figure After Loading boardgame CSV file

A screenshot of a computer

AI-generated content may be incorrect.

Figure Display of the document in Capella, to note that the ID is different from the one in CouchDB

A screenshot of a computer

AI-generated content may be incorrect.

Figure Creation of Bucket called project\_couchdb

A screenshot of a computer

AI-generated content may be incorrect.

Figure Create Cluster Access Name and Password & define bucket-Lavel Access to \_default, Read/Write

A screenshot of a computer

AI-generated content may be incorrect.

Figure Add Allowed IP Addresses

A screenshot of a computer

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

AI-generated content may be incorrect.

Figure Setup credentials to authenticate Data API requests

## Postman Test

A screenshot of a computer

AI-generated content may be incorrect.

Figure Successful GET request

The following URL was used for this GET request: <https://8twzu4twupom-etl.data.cloud.couchbase.com/v1/callerIdentity>

A screenshot of a computer

AI-generated content may be incorrect.

Figure GET document with Is = 00d1bb24-59a3-4c2d-804c-bf4972bc935d

The following URL was used for this GET request: <https://8twzu4twupom-etl.data.cloud.couchbase.com/v1/buckets/project_couchdb/scopes/_default/collections/_default/documents/00d1bb24-59a3-4c2d-804c-bf4972bc935d>

A screenshot of a computer

AI-generated content may be incorrect.

Figure Successful POST request for adding a new boardgame to the database

The following URL was used for the POST request: <https://8twzu4twupom-etl.data.cloud.couchbase.com/v1/buckets/project_couchdb/scopes/_default/collections/_default/documents/special-emmaid-abcdefgh12345>

A screenshot of a computer

AI-generated content may be incorrect.

Figure New document has been added correctly

# References

*Couchbase Data API (1.0.0)*. (2025, October 23). Retrieved from Couchbase Documentation: https://docs.couchbase.com/cloud/data-api-reference/index.html

*CouchDB Documentation Release 1.3.0*. (2025, October 23). Retrieved from CouchDB Documentation Release 1.3.0: https://app.readthedocs.org/projects/couchdb-13/downloads/pdf/latest/

*Introduction to Replication*. (2025, October 25). Retrieved from couchdb: https://docs.couchdb.org/en/stable/replication/intro.html

Ochtman, D. (2025, October 23). *couchdb-python Documentation - Release 1.0*. Retrieved from couchdb-python Documentation - Release 1.0: https://app.readthedocs.org/projects/couchdb-python/downloads/pdf/latest/

*py-couchdb Release v1.14*. (2025, October 23). Retrieved from py-couchdb Release v1.14: https://pycouchdb.readthedocs.io/en/latest/index.html

Wheeler, D. A. (2025, October 23). *How to Evaluate Open Source Software / Free Software (OSS/FS) Programs*. Retrieved from dwheeler.com: https://dwheeler.com/oss\_fs\_eval.html

# Appendix

## Acknowledging use of AI

I acknowledge the use of ChatGPT (ChatGPT-5, <https://chatgpt.com/> ) to generate Python script that reads data from a CSV file and uploads it to a CouchDB database.

### Prompt for Importing CSV file in CouchDB

Write a Python script that reads data from a CSV file and uploads it to a CouchDB database using the REST API and the requests library. The script should create the database if it doesn’t exist and then use the \_bulk\_docs endpoint to insert all rows.

### Output for Importing CSV file in CouchDB

import csv

import json

import requests

# CouchDB connection info

couchdb\_url = "http://admin:password@localhost:5984"

db\_name = "my\_database"

# Step 1: Create the database (if it doesn’t exist)

response = requests.put(f"{couchdb\_url}/{db\_name}")

if response.status\_code in (200, 201):

print(f"Database '{db\_name}' created or already exists.")

else:

print(f"Database check/creation response: {response.status\_code} - {response.text}")

# Step 2: Read data from CSV file

csv\_path = r"C:\path\to\your\data.csv" # <-- change this to your file path

with open(csv\_path, newline='', encoding='utf-8') as csvfile:

reader = csv.DictReader(csvfile)

docs = list(reader)

# Step 3: Prepare bulk upload payload

bulk\_payload = {"docs": docs}

# Step 4: Upload data to CouchDB using \_bulk\_docs

resp = requests.post(

f"{couchdb\_url}/{db\_name}/\_bulk\_docs",

headers={"Content-Type": "application/json"},

data=json.dumps(bulk\_payload)

)

# Step 5: Print server response

if resp.status\_code == 201:

print("Documents uploaded successfully.")

else:

print(f"Upload failed: {resp.status\_code} - {resp.text}")

print("Response JSON:", resp.json())

#### Code difference

* Deleted error handling
* Change file path
* Changed names to the variables
* Print only JSON output