## CouchDB report (advantages for a selected app, available drivers, prototyping)

Modestas Cepulis  
 t0019237

Table of Contents

[CouchDB report (advantages for a selected app, available drivers, prototyping) 1](#_Toc55560612)

[**Chapter 1: Setting up the database** 3](#_Toc55560613)

[1.1 Introduction 3](#_Toc55560614)

[1.2 Why CouchDB would be beneficial (suitable) for the selected application (mobile gaming app) 3](#_Toc55560615)

[1.3 Setting up the variables and creating them in postman 3](#_Toc55560616)

[1.4 Retrieving all documents and specific documents with postman 4](#_Toc55560617)

[1.5 Adding documents (data) with Postman POST command. 5](#_Toc55560618)

[1.6 Updating documents (data) with Postman PUT command. 6](#_Toc55560619)

[1.7 Showing Map Reduce and Views with POSTMAN command. 6](#_Toc55560620)

[1.8 Creating views with CouchDB 7](#_Toc55560621)

[**Chapter 2: Setting up the prototype** 9](#_Toc55560622)

[2.1 Introduction 9](#_Toc55560623)

[2.2 Working on a prototype 9](#_Toc55560624)

[2.3 Setting up the basic UI 10](#_Toc55560625)

[2.4 Setting up the GET command to get all existing documents and manipulating data. 11](#_Toc55560626)

[2.5 Setting up the PUT (UPDATE) command to update existing document results and manipulate its data. 13](#_Toc55560627)

[2.6 Setting up the POST (ADD) command to add a new document/player 15](#_Toc55560628)

[2.7 Setting up the DELETE command to delete an already existing document/player 17](#_Toc55560629)

[2.8 Setting up the views/map reduce code 17](#_Toc55560630)

[**Chapter 3: Conclusion** 19](#_Toc55560631)

[**References** 19](#_Toc55560632)

# **Chapter 1: Setting up the database**

## 1.1 Introduction

For my database idea, I selected to work on a ‘mobile gaming app’. The database will have different string and int attributes that would reflect different things in the game (such as player\_name, player\_xp, etc). Every attribute would be assigned to each player. Each player might have a different value for each attribute as one player might be playing the game differently from other players.

## 1.2 Why CouchDB would be beneficial (suitable) for the selected application (mobile gaming app)

Easy to access files/documents  
While making a mobile app it is important to keep everything tidy and easily accessible, couchdb allows the user to filter documents and find relevant information quickly. “*CouchDB uses views as the primary tool for running queries and creating reports from stored document files. Views allow you to filter documents to find information relevant to a particular database process” (ibm.com)*

Efficient document storage  
While storing documents in CouchDB, for each document there is no limit to the text size or element count of each document. This is really important while developing a gaming app as the gaming app might always expand and new information may be added to the game.

Simple Structure  
According to (geeksforgeeks.org) the CouchDB has a simple structure which is quite easy implement. Because of this reason it is better to use CouchDB and spend less time on learning how to implement a CouchDB database.

Offline data storage capabilities  
CouchDb has the capability to save data without the need of internet connection. Not all of the time, users are connected to an internet service. This helps to maintain as much information or data as possible at all times while being offline. Once the connection is recovered, the data can be saved and pushed up. This would benefit the selected application such as a gaming mobile app as most of the time gaming mobile apps only save information only based on what the player has achieved (such as player level, player xp, and so on)



## 1.3 Setting up the variables and creating them in postman

First of all, we need to set up the database variables. I have chosen to go with a documentation file called ‘players’. ‘players’ would have different string attributes to reflect different things that would be used in the mobile gaming app.

Figure 1(Database variables)

Here we have created some variable names and their values in postman. We have posted these variables to the database and have received a confirmation that they have been added to the database.

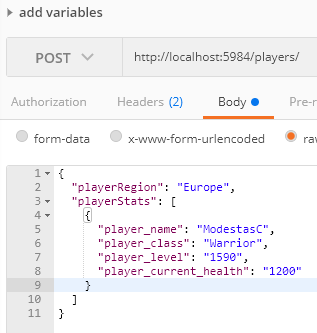


Figure 2 (Variables through postman)

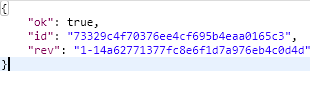


Figure 3 (Confirmation message postman)

## 1.4 Retrieving all documents and specific documents with postman

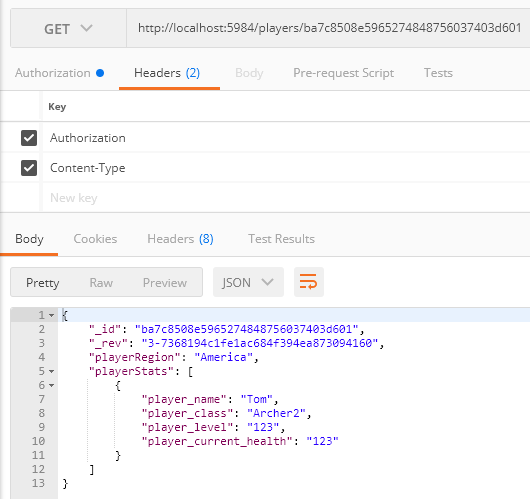
 To retrieve the ‘players’ document through postman, a command needs to be used:  
“http://localhost:5984/players/ba7c8508e5965274848756037403d601?include\_docs=true”  
The results are:

Figure 4 (results from selecting a specific document postmant)

By knowing how to retrieve results from a database, we can display them in our working prototype in the future.

## 1.5 Adding documents (data) with Postman POST command.

Postman can be used to add information to the CouchDB by using a POST command. In this case, a document with an ID of ‘*ba7c8508e5965274848756037400018f*’ will be used.  
A simple line of information is going to be pushed to the CouchDb and the data is going to be added:

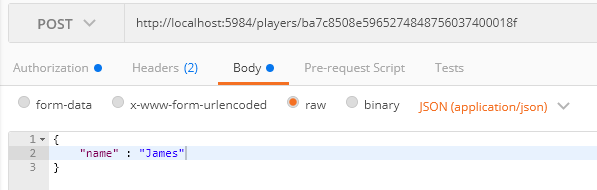


Figure 5(Postman post command)

## 1.6 Updating documents (data) with Postman PUT command.

PUT is a command that can be used in postman to update existing variables in a CouchDB document. As there is an outline or template of variables that we have from before, we can use that format to make up a PUT command:

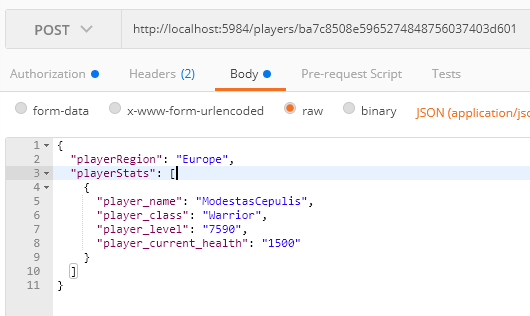


Figure 6(PUT postman command for updating a document)

## 1.7 Deleting with POSTMAN

It is quite easy to run a ‘DELETE’ command through postman. We just need to http:// address, database name and the document ID address:



Figure (POSTMAN delete command)

After this command in run, the specified document with the document id is removed/deleted from the CouchDB.

## 1.8 Showing Map Reduce and Views with POSTMAN command.

In order to show and retrieve the specific view with postman, we need to use the following command:  
“http://localhost:5984/players/\_design/allDocs/\_view/byPlayerRegion?group=true”  
“players” is the name of the database.  
“\_design” is the design section of the database.  
“\_allDocs” is the view sub-name in the database.  
“\_byPlayerRegion” is the name of the view.  
“?group=true” is a map reduce command that lets us group same values together.

After we run this command, we are prompted with these results:

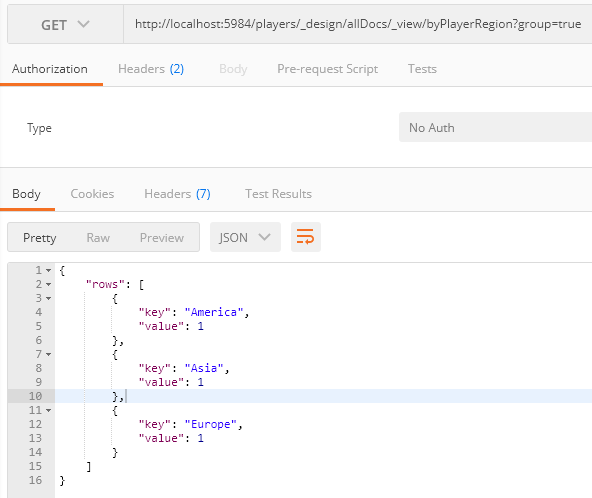


Figure (Postman view results)

## 1.8 Creating views with CouchDB

CouchDB allows to create views that work similar to lines of code. These lines of code might help to gather information or do specific tasks. In this case, we will be getting all of the players regions and count the number of players who are connected from different regions. Those regions can be different values such as ‘Europe’, ‘America’, ‘Asia’, etc.

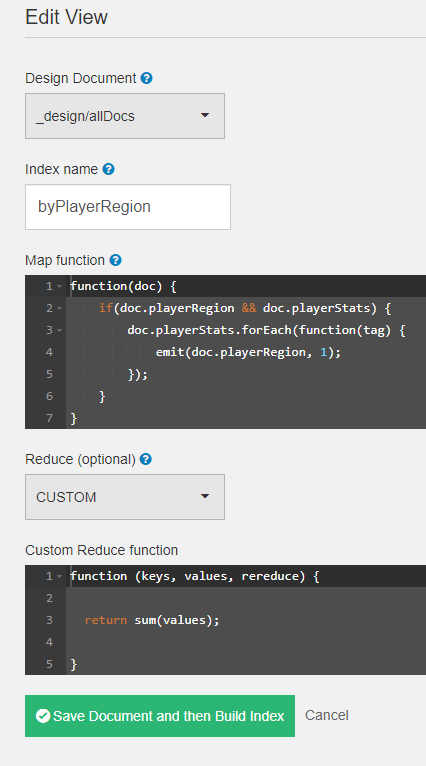
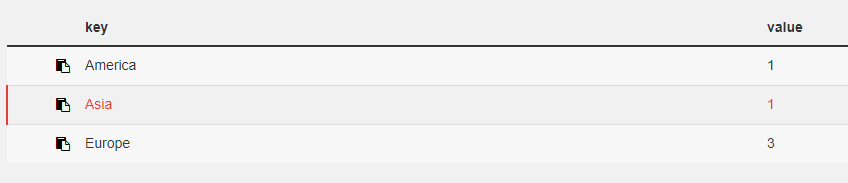


Figure 9(The creation of the view)

This view checks if a playerRegion and playerStats exists, and if it does then the function takes the value for each playerStats (the actual player information such as level, etc) and runs the function again which takes the tag variable. This helps the function to collect the required data. In this case: Count how many players are in each region based by a variable ‘playerRegion’ and get the sum of it.

After processing **map reduce** **in couch db** options on the results, we are prompted with the following:



These results show us that there is different number of players playing in different regions. (1 from America, 1 from Asia, 3 from Europe)

# **Chapter 2: Setting up the prototype**

## 2.1 Introduction

Prototyping is an important part when creating any database. It helps to plan out for the future and helps to get a better insight into the database and its components. For the prototype development, ‘couchdb’ driver has to be chosen. In order to do that, research on all of the CouchDB drivers has to be done, and one driver has to be chosen: *(All references are from the same website – cwiki.apache.org)*

LoveSeat  
A couchdb driver that works well with Generics and populating Domain objects. To fit the theme of ‘Gaming mobile app’ this driver would not be suitable to make a prototype as I do not have any prior experience with it.

Divan  
Divan is using C# library for CouchDB while using Newtonsoft.JSON and NUnit as its only external dependencies. This driver would suit me better as I have a prior knowledge of working with C#, only downside would be that I never worked with Newtonsoft.JSON and NUnit, but for now the research is going well.

MyCouch  
“*It is an asynchronous CouchDB client for .Net applications. It builds top of the asynchronous HTTP client and uses JSON.Net to provide flexible serialization behavior (cwiki.apache.org)*”. As I have a prior knowledge of working with C# and .Net applications such as visual studio, this would be the best pick for my CouchDB driver.

## 2.2 Working on a prototype

For ‘CouchDB’ driver, I will be using a driver/library called ‘MyCouch’ (*by Danielwertheim) and microsoft visual studio for the prototyping*. ‘MyCouch’ can be used for .NET services such as visual studio. This driver is going to help me to link the visual studio’s workstation to the CouchDB database. The prototype is going to be based on ‘Windows forms’ and the user is going to be able to perform such tasks as: ‘List all player details (information) based on the document ID’, ‘Add a new player’, ‘Update existing player details’, ‘Delete current selected player and ‘Show all player regions’.

## 2.3 Setting up the basic UI

In order to set up the get, post, view/map reduction and delete commands, an UI needs to be set up.

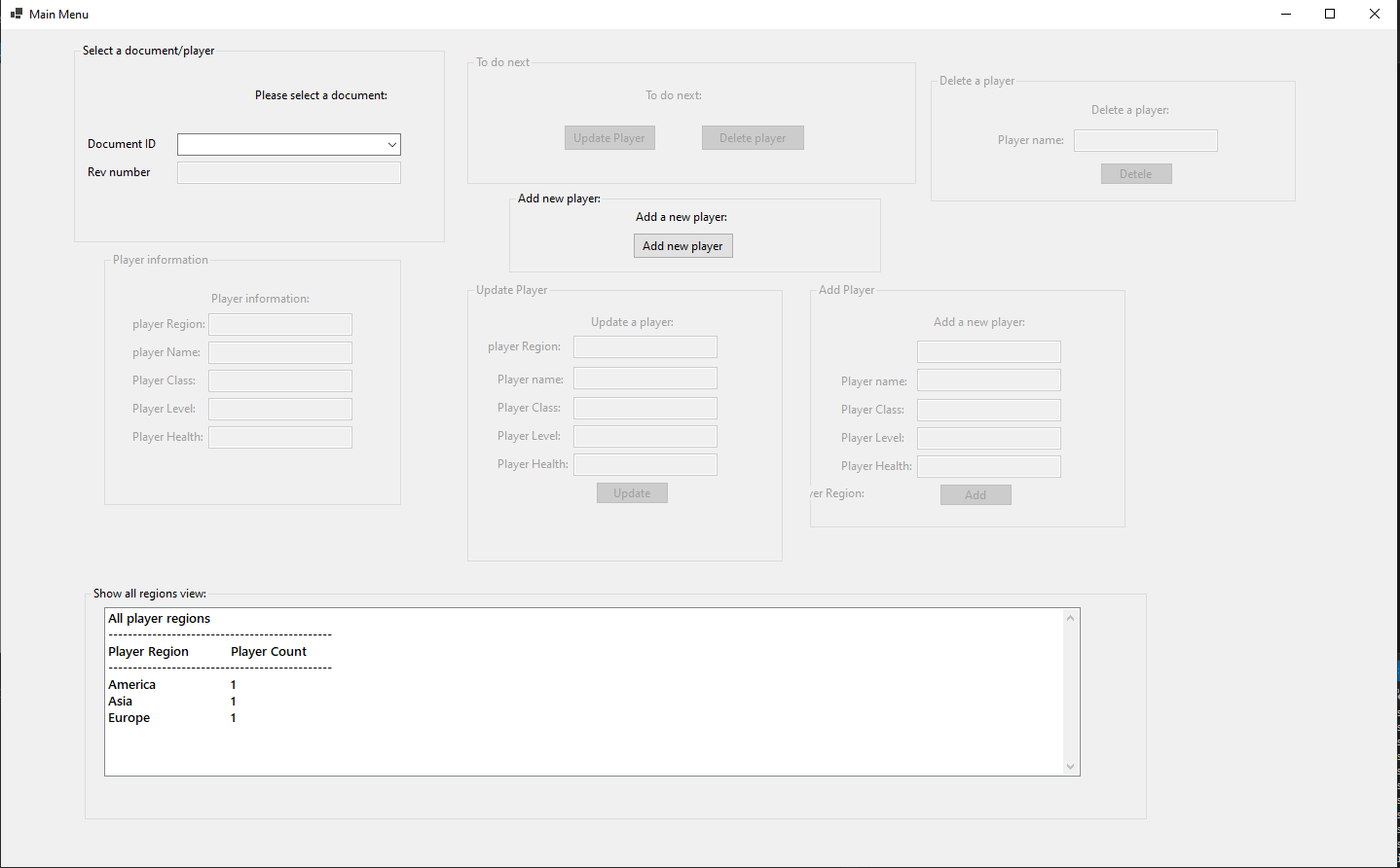


Figure 10 (Windows form UI from visual studio)

These are the main UI elements that are shown in a windows form. While having these textboxes, we can continue to work on additional steps that need to be done in order to retrieve the information from CouchDB.

## 2.4 Setting up the GET command to get all existing documents and manipulating data.

In order for us to be able to select a player (which in this case each player is going to be in each new document in the database) we need to retrieve the total amount of documents that are currently on in the database. For this we are going to be using this command:

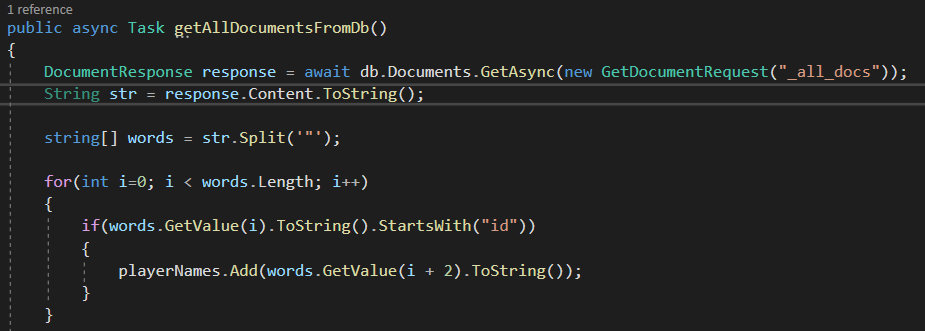


Figure 11(Code to get all documents)

This code gets the ‘DocumentResponse’ as response which connects to the database and gets a Async request with all of the data that corresponds to ‘\_all\_docs’. ‘\_all\_docs’ retrieved all of the existing documents in the database. The information comes out in a badly formatted string:

“{"total\_rows":2,"offset":0,"rows":[

{"id":"73329c4f70376ee4cf695b4eaa0165c3","key":"73329c4f70376ee4cf695b4eaa0165c3","value":{"rev":"8-5d70543dcc47535725ce25505a8be587"}},

{"id":"ba7c8508e5965274848756037400018f","key":"ba7c8508e5965274848756037400018f","value":{"rev":"5-bfb9c6ebda3432eab8e7b17361e17978"}}

]}”

As there is a lot of more information that we need in this area, we can separate this string value by separating the parenthesis (“) and adding all of the strings to a string array.

While having a string array, next step is to check which column has the “ID” value (as we know that the next value is the document ID value which is what we want). Once we get the specific number inside of the array, we can add additional 2 numbers and retrieve our document ID.

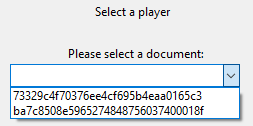


Figure 12(Selecting document ID)

By having these document Id’s, we can create another DocumentResponse which would retrieve all the player name while using the document’s id number.

To do that, we are going to be using this peace of code:

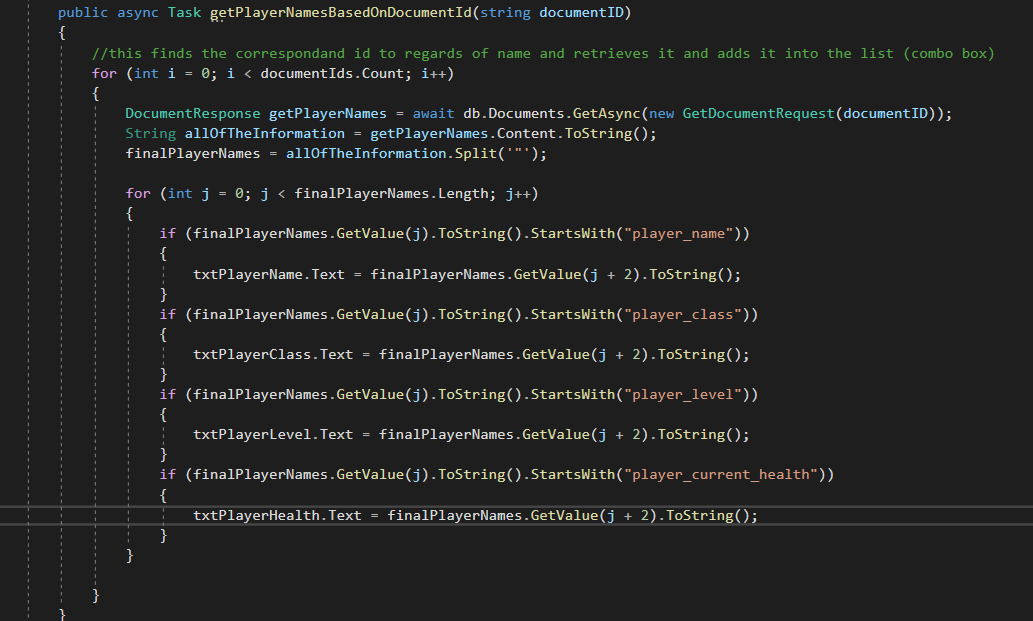


Figure 13(Code for sorting out appropriate values)

This just gets a ‘DocumentResponse’ depending on the document ID that is selected by the user, then the system retrieves all of the data from a document puts it in a String and splits it by parenthesis and adds it to a string array. This array then is checked against certain values such as ‘player\_name’, and those values are displayed on the UI. Here are 2 possible outcomes for 2 possible documents:

One: Two:

Figure 14(First document ID selection)

Figure 15(Second document ID selection)

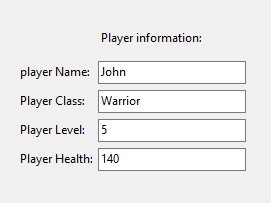
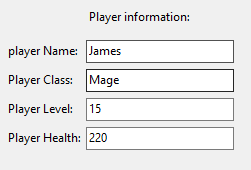


Figure 17(first document results)

Figure 16(Second document Results)

## 2.5 Setting up the PUT (UPDATE) command to update existing document results and manipulate its data.

Now we have successfully retrieved the data by using GET command, it’s time to update existing data. To update an existing user, the user has to select the document id from one of the available ones. We are going to be using this code to perform this action:



Figure (Put code for updating)

In this code a ‘DocumentHeaderResponse’ request variable was created. This variable holds the documentID (key) and revID (\_rev) value for referencing. It contains the string format of a document and the appropriate referencing to the textboxes. These textboxes are filled up by the user.



Figure (Update document selection)

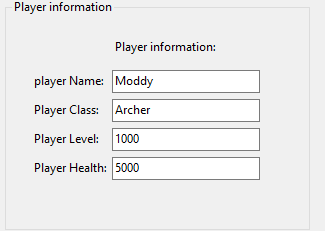


Figure (Selected Player's information)

Once the user has selected the document id, the user has to choose (press one of the buttons) from one of the following steps:

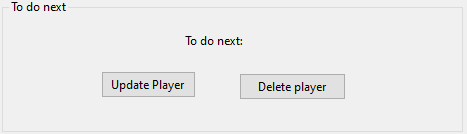


Figure (Following steps)

In this case, the user has to select ‘Add new player’. Once this button is pressed another form comes up with the following:

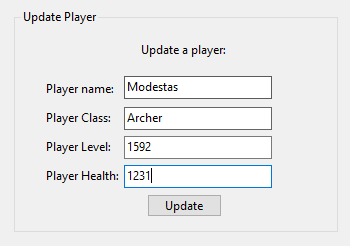


Figure (New values for update)

Once this form is active, the user needs to input new values into the boxes, these values then going to be updated to the selected player. Then the user has to press ‘Update’ button in order to update the values in the database, then shortly the user is prompted with a confirmation message.

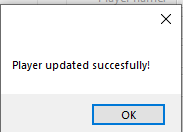


Figure (Update confirmation Message)

The values have been updated successfully and should appear appropriately in the document that was selected to update.

## 2.6 Setting up the POST (ADD) command to add a new document/player

We are going to be using this code in order to achieve the action:

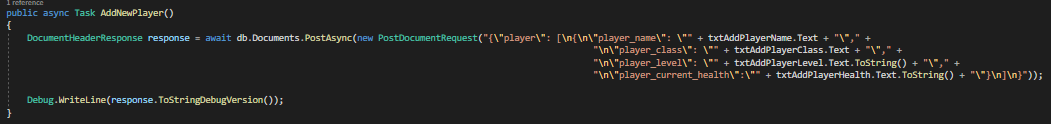


Figure (Add/post code)

This code creates a ‘DocumentHeaderResponse’ request which then takes the information that the user wants to add to the CouchDB database. The information is taken from user’s input boxes. The String is set up accordingly to the file/document that would appear in CouchDb.

In order to add a new player, the user has to press the following button:

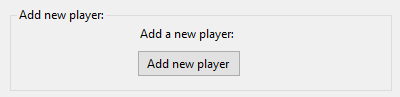


Figure (add new player button)

Once this button is pressed, another form/group becomes active. The user needs to put in new information into the text boxes in order to create a new player. (The new player document/file is going is only going to be visible once the prototype is restarted)

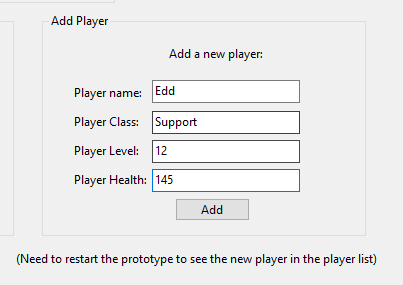


Figure (New values for adding a new player)

Once these values are put in by the user, the next step is to press the ‘add’ button to create a new document in the database and insert the values. (A confirmation message would appear shortly)

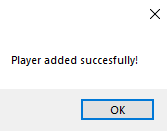


Figure (player add confirmation message)

The query was successful and the appropriate values have been added into the newly created document.

## 2.7 Setting up the DELETE command to delete an already existing document/player

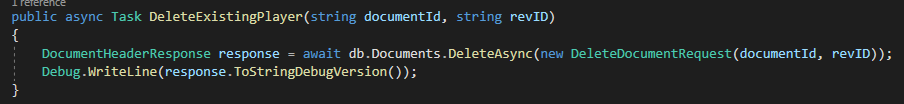
In order to delete an existing document/player we need to use this code:  


Figure (Deleting Async code)

This code creates a ‘DocumentHeaderResponse’ called response, which sends out a request to delete a document to the CouchDB database. (It deletes the document based on documentID and revID that is provided by the user choosing the document’s id)

When a user selects a player from the select menu, another form gets activated:

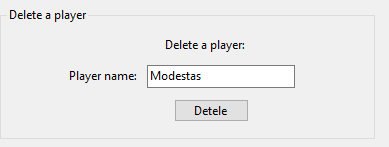


Figure (delete form)

This lets the user to delete an already existing customer. Once the user presses the Delete button, the player/document is deleted from the database (the user needs to restart the protype in order to see any results)

## 2.8 Setting up the views/map reduce code

The map reduce code should get the view from couchDB and put it in a box with a text that would count as show how many players are from each region and show the total count. To start with the map reduce view, we need to set up the UI.

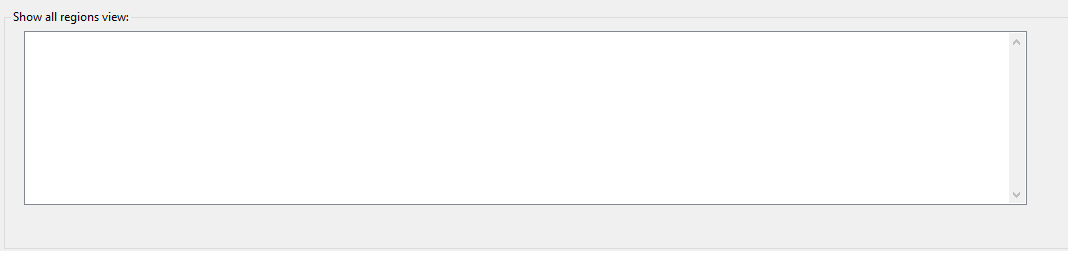


Figure (UI interface for map reduce)

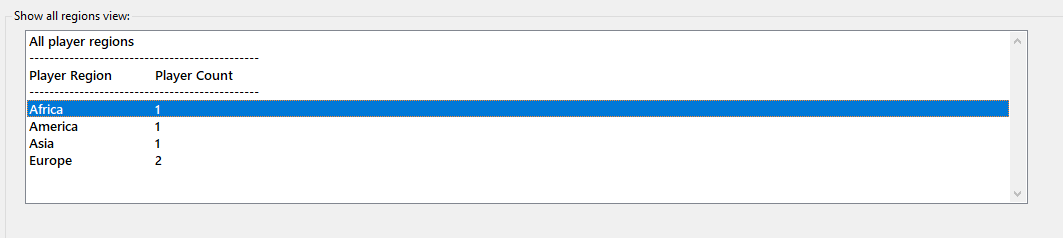
In order to get the information needed and access the view from CouchDB database, we need to use this following code:



Figure (Code for map reduce/views)

This code creates a new variable query. This variable makes a new QueryViewRequest with the design document name ‘allDocs’ and the view name ‘byPlayerRegion’. This is stored in the query variable. Then the query variable is used in ‘ViewQueryResponse’ called result. This result is then but through a loop to determine how many keys and values there is in total. Then these keys (Region, etc America) and values (The number of values, etc 1) is put into the box.

Once the code runs, the user is prompted with the following results:



The sum from the view does not allow the textbox to show region such as Europe twice. (If any of the other buttons are pressed “add”, “update”, “delete”, the textbox refreshes and shows the correct updated/added/deleted values)

# **Chapter 3: Conclusion**

CouchDB is a great database model. It allows to gather data offline and push it up to the database whenever the user is back on the internet. It has a simple and easy to understand structure, which makes the use of the database easier. CouchDB is great for apps such as a ‘mobile game app’ as there is no limit on how much data can be stored. (no text size limits). It also has a quite an easy way to access documents through having a good view system. CouchDB has many drivers, such as MyCouch. MyCouch can be used to make ‘window forms’ prototype or apps in which the user could manipulate the data in the database with appropriate UI. In the upcoming future, CouchDB will be used more and more while dealing with database processes.

# 

# **References**

Peter Given Notes, *CouchDB* [Online pdf]  
https://online.ittralee.ie/bbcswebdav/pid-331819-dt-content-rid-1786189\_1/courses/SEM1\_DBMS81001\_48064/Lecture%20CouchDB.pdf

IBM Cloud education, *apache CouchDB*

https://www.ibm.com/cloud/learn/couchdb

Geeksforgeeks.org, advantages and disadvantages of CouchDB

https://www.geeksforgeeks.org/introduction-to-apache-couchdb/

Daniel Wertheim, MyCouch github .NET addon

https://github.com/danielwertheim/mycouch

cwiki.apache.org, *different available drivers for CouchDB*

https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=97555303