**CouchDB**

**Part 1:**

**Introduction:**

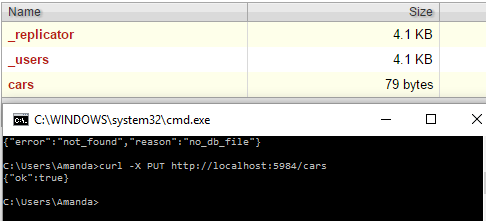
In this project, I will be using CURL commands to create, read, update and delete, through a REST interface to interact with CouchDb. I will also be creating my own GUI in Java using the Swing framework as well as the Lightcouch API to communicate with CouchDb. I want to show the uses of CouchDb along with its advantages and disadvantages.

**Why I chose my database design?**

In hindsight, I regret my choice of database design, as I did not have a proper understanding of how CouchDb worked when I started on this project. A live application would have suited this database perfectly as CouchDb is a very robust and stable database. I still feel that that I was able to show the uses of CouchDb through my database design and was still able to learn a lot about CouchDb while implementing my design.

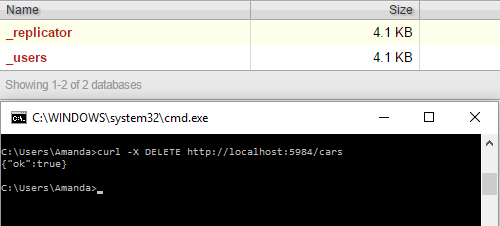
**Create Database**

curl -X PUT http://localhost:5984/cars



**Delete Database**

curl -X DELETE http://localhost:5984/cars

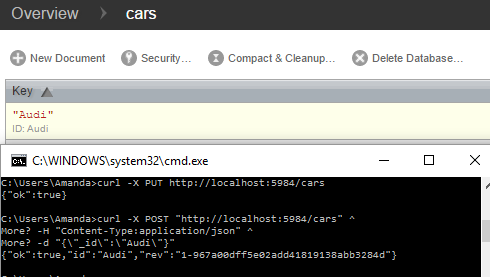


**Create Document**

curl -X POST "http://localhost:5984/cars" ^

More? -H "Content-Type:application/json" ^

More? -d "{\"\_id\":\"Audi\"}"

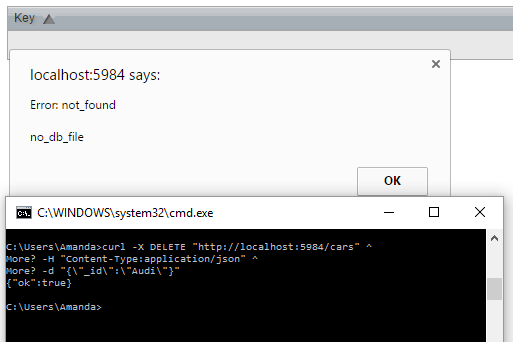


**Delete Document**

curl -X DELETE "http://localhost:5984/cars" ^

More? -H "Content-Type:application/json" ^

More? -d "{\"\_id\":\"Audi\"}"



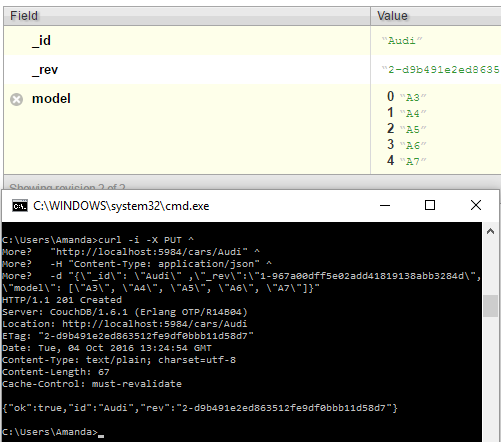
**Update Document 1**

curl -i -X PUT ^

More? "http://localhost:5984/cars/Audi" ^

More? -H "Content-Type: application/json" ^

More? -d "{\"\_id\": \"Audi\" ,\"\_rev\":\"1-967a00dff5e02add41819138abb3284d\",\"model\": [\"A3\", \"A4\", \"A5\", \"A6\", \"A7\"]}"



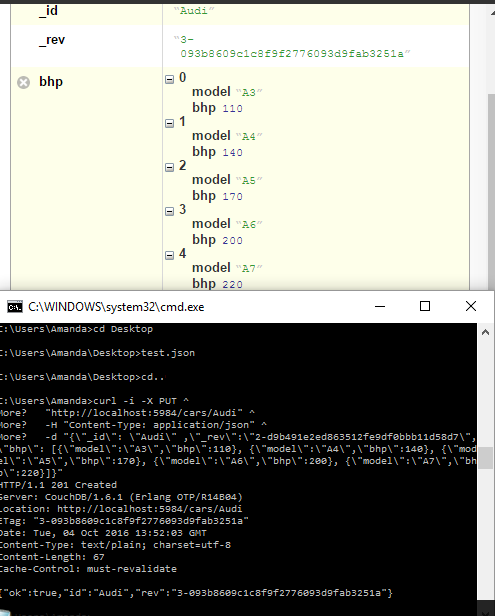
**Update Document 2**

curl -i -X PUT ^

More? "http://localhost:5984/cars/Audi" ^

More? -H "Content-Type: application/json" ^

More? -d "{\"\_id\": \"Audi\" ,\"\_rev\":\"2-d9b491e2ed863512fe9df0bbb11d58d7\",\"bhp\": [{\"model\":\"A3\",\"bhp\":110}, {\"model\":\"A4\",\"bhp\":140}, {\"model\":\"A5\",\"bhp\":170}, {\"model\":\"A6\",\"bhp\":200}, {\"model\":\"A7\",\"bhp\":220}]}"

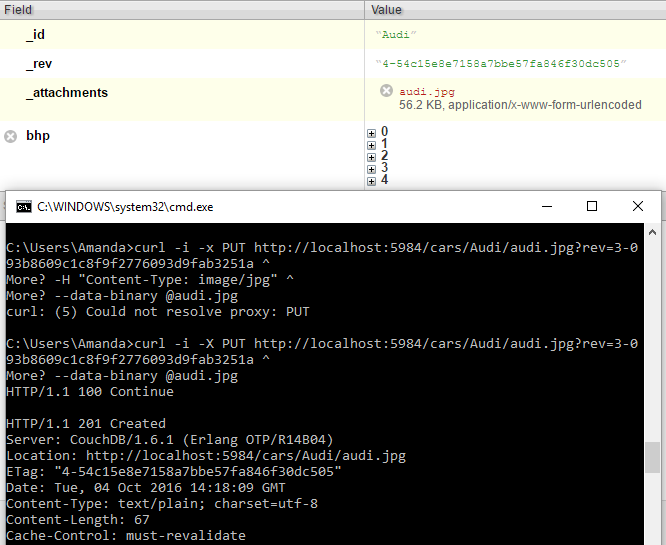


**Update Document with Image**

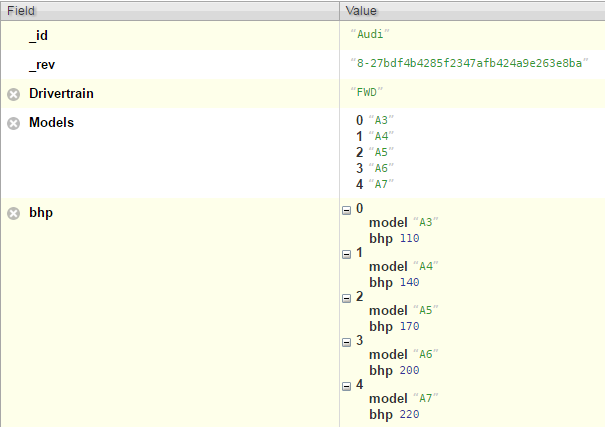
curl -i -X PUT http://localhost:5984/cars/Audi/audi.jpg?rev=3-093b8609c1c8f9f2776093d9fab3251a ^

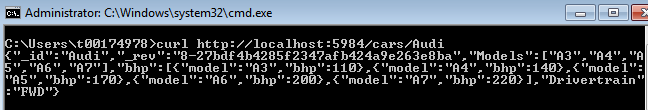
-data-binary @audi.jpg ^

-H "Content-Type: image/jpg"

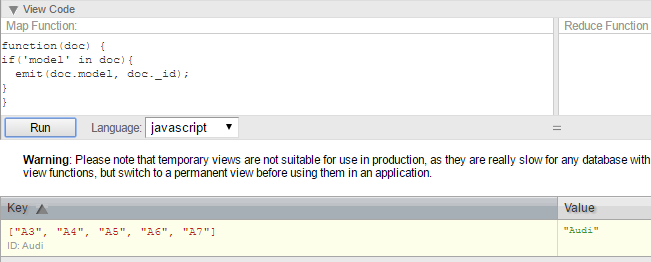


**GET Example**

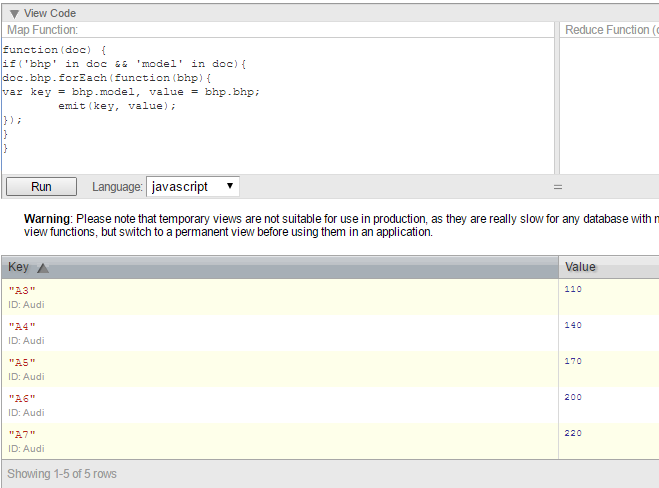




**Simple view by model**

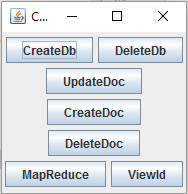


**View by model**



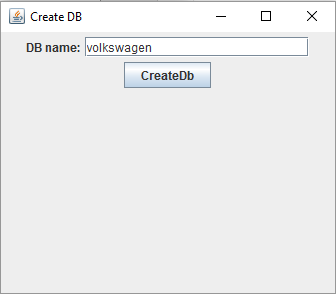
**Part 2:**

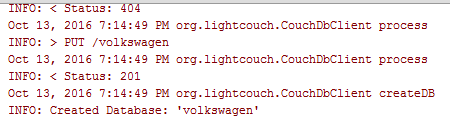
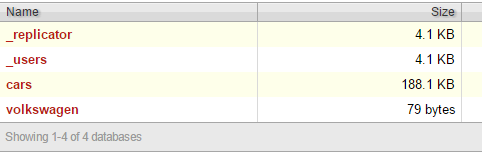
**Main GUI:**

****

Chose a funtion by clicking any of the buttons. All of the information needed by the user will be prompted by labels.

**Create Database:**

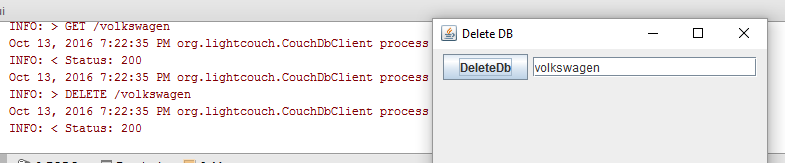


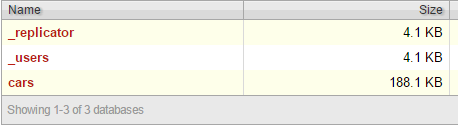


Code:

**public void** createDb(JTextField dbt) {  
 **db** = dbt.getText();  
 CouchDbClient dbClient = **new** CouchDbClient(**db**, **true**, **"http"**, **"localhost"**, 5984,**null**,**null**);  
  
 dbClient.shutdown();

**Delete Database:**

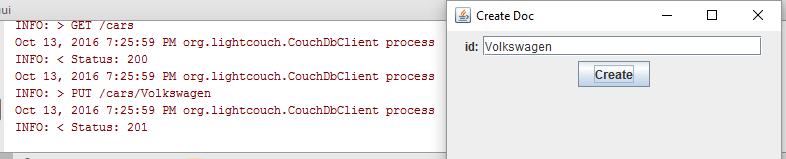


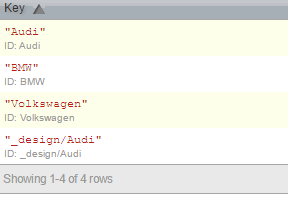


**Code:**

**public void** deleteDb(JTextField name) {  
 **db** = name.getText();  
 CouchDbClient dbClient = **new** CouchDbClient(**db**, **true**, **"http"**, **"127.0.0.1"**, 5984,**null**,**null**);  
  
 dbClient.context().deleteDB(**db**,**"delete database"**);  
  
 dbClient.shutdown();

**Create Document:**

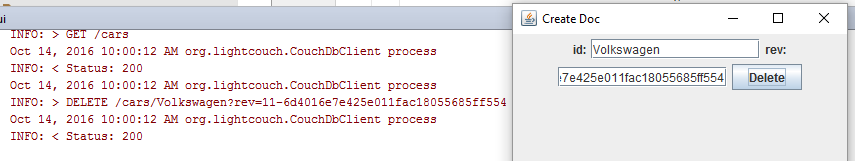


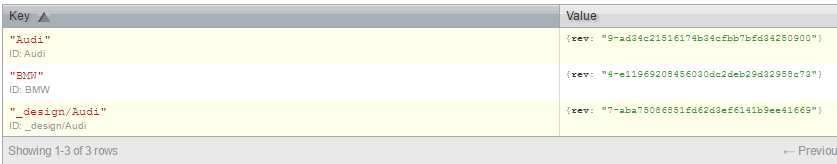


**Code:**

**public void** createDoc(JTextField idtxt){  
 **id** = idtxt.getText();  
 CouchDbClient dbClient = **new** CouchDbClient(**"cars"**, **true**, **"http"**, **"127.0.0.1"**, 5984, **null**, **null**);  
  
 String jsonstr = **new** StringBuilder().append(**"{\"\_id\":"**).append(**id**).append(**"}"**).toString();  
  
 JsonObject jsonobj = dbClient.getGson().fromJson(jsonstr, JsonObject.**class**);  
 dbClient.save(jsonobj);  
  
 dbClient.shutdown();

**Delete Document:**

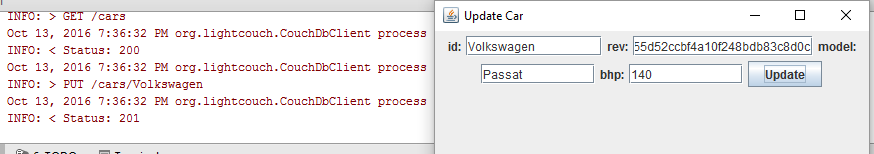


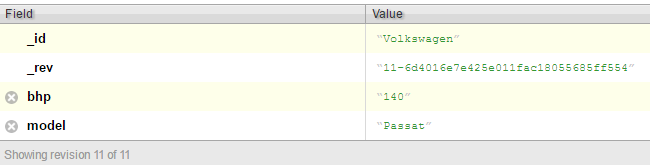


**Code:**

**public void** deleteDoc(JTextField idtxt, JTextField revtxt){  
 **id** = idtxt.getText();  
 **rev** = revtxt.getText();  
 CouchDbClient dbClient = **new** CouchDbClient(**"cars"**, **true**, **"http"**, **"127.0.0.1"**, 5984, **null**, **null**);  
  
 dbClient.remove(**id**, **rev**);  
  
 dbClient.shutdown();

**Update Document:**



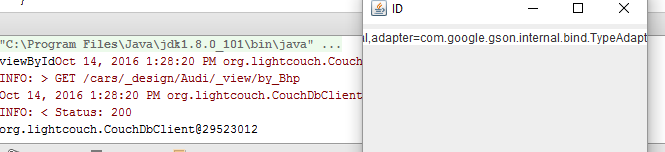


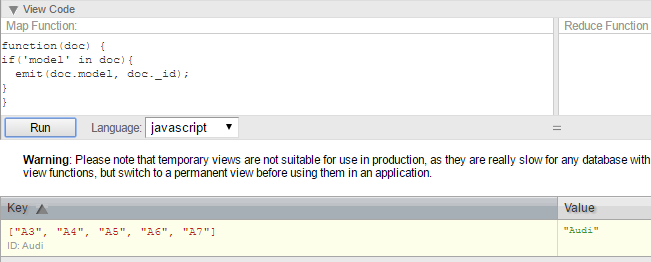
**Code:**

**public void** updateDoc(JTextField idtxt, JTextField revtxt, JTextField fieldtxt, JTextField valuetxt){  
 **id** = idtxt.getText();  
 **rev** = revtxt.getText();  
 **field** = fieldtxt.getText();  
 **value** = valuetxt.getText();  
 CouchDbClient dbClient = **new** CouchDbClient(**"cars"**, **true**, **"http"**, **"localhost"**, 5984, **null**, **null**);  
 Car car = **new** Car(**id**, **rev**, **field**, **value**);  
  
 dbClient.update(car);  
  
 dbClient.shutdown();  
}

**View by bhp:**

I had difficulty displaying the contents of my view. However in the screen shots it came be seen that the right GET commandis being sent from my GUI.



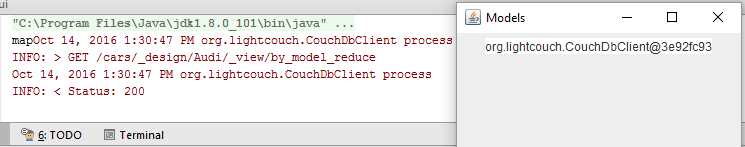


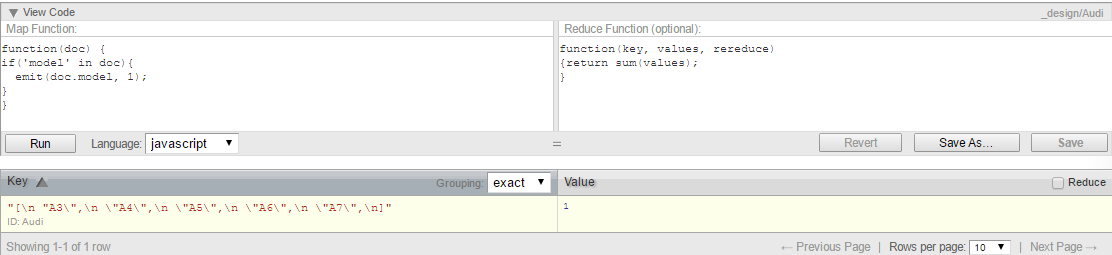
**Here is my code for the view:**

**public void** getView()  
{  
 CouchDbClient dbClient = **new** CouchDbClient(**"cars/\_design/Audi/\_view/by\_Bhp"**, **true**, **"http"**, **"localhost"**, 5984, **null**, **null**);  
 *//View view = dbClient.view("\_design/Audi/\_view/by\_Bhp")  
 // .key(2011, 10, 15) // complex key as: values, or array  
 // .reduce(false)  
 // .includeDocs(true);  
 //System.out.println(view);* System.***out***.print(dbClient);  
 **bhpView**.append(String.*valueOf*(dbClient.getGson()));

**MapReduce by model:**

I had difficulty displaying the contents of my MapReduce funtion. However in the screen shots it came be seen that the right GET commandis being sent from my GUI. This function gets all models in Audi and returns the sum of the values, which in this case is 1 as there are no duplicates.





**Here is my code for the mapReduce:**

**public void** mapReduce()  
{  
 CouchDbClient dbClient = **new** CouchDbClient(**"cars/\_design/Audi/\_view/by\_model\_reduce"**, **true**, **"http"**, **"localhost"**, 5984, **null**, **null**);  
 **map**.append(dbClient.toString());  
}

**Describe API used:**

I decided to use the Java API Lightcouch for communicating with CouchDb. Lightcouch is a quick an easy way of communicating with a CouchDb database. Using the CouchDbClient object makes CRUD commands relativly easy. And through CouchDbClient an instance can be created in your application at any time, so that it can be modified or queryed before moving on to the next command. It is important to close the object using the dbClient.shutdown();

command as leaving instances of your database open can create problems when carrying out a query on another instance.

**Conclusion:**

In conclusion I wish I had spent more time on the design of my database befor eimplementing it. I think this would have been beneficial and I would have been able to demonstrate CouchDb’s and Lightcouch’s capabilities. I did however learn a lot about CouchDb through its REST interface as well as the GUI I created in Java. CouchDb is very simple to usein the REST interface by using CURL commands. CouchDb is also very robust and never crashes which makes it perfect for web applications. There is also very little chance of losing your data as CouchDb duplicates your data over different servers in the event of one server crashing. The drawback of this being that a lot of memory can be used by unneccessarily duplicating all data.