

Portfolio Part 1: CouchDB

Aoife Sayers

Advanced Database Programming

Due 20th October 2017

Contents

[Introduction 3](#_Toc495491520)

[1. Implement your own CouchDB database (something requiring both a desktop/mobile app or mobile game implementation) and use the REST interface to show examples of CRUD. 40% 4](#_Toc495491521)

[CouchDB Database 4](#_Toc495491522)

[RESTful Interface CRUD – READ 5](#_Toc495491523)

[RESTful Interface CRUD – CREATE 6](#_Toc495491524)

[RESTful Interface CRUD – UPDATE 7](#_Toc495491525)

[RESTful Interface CRUD – DELETE 8](#_Toc495491526)

[2. Download a driver for a programming language of your choice (Android would be nice!). Build a UIF in Java/C# etc and show the CRUD operations working through the UIF. Show a call to a mapreduce operation from within the programming language. Comment your code and show screen pictures of it working. (60%) 1](#_Toc495491527)

[Ektorp 1](#_Toc495491528)

[Java UIF CRUD - CREATE 2](#_Toc495491529)

[Java UIF CRUD - READ 4](#_Toc495491530)

# Introduction

Developed by Apache in 2005, CouchDB is a scalable, NoSQL document oriented database storing key value maps in a JSON format. CouchDB can be implemented for a range of projects across many environments locally or over clusters of servers on mobile phones to web browsers on inconsistent networks with it’s offline first data sync. CouchDB is durable with regards data storage. Unlike MongoDB and Relational databases SQL, CouchDB does not permit ad hoc querying; instead CouchDB utilizes indexed views with MapReduce to find your documents. CouchDB can be accessed via the Fauxton web interface or using RESTful tools like CURL or Google Chrome’s web application Postman.

Also CouchDB has no major learning curve for those who have previously worked with Web applications. (Warner Onstine, 2012). Document data stores such as CouchDB and MongoDB have the least learning curve when migrating from relational databases to NoSQL databases. There is a much less rigid schema as well as avoiding having to join tables with CouchDB. (Whitehouse, 2009)

 CouchDB’s main features include:

Easy replication of a database across multiple server instances using the Couch Replication Protocol

Fast indexing and retrieval

HTTP REST based API for document insertion, updates, retrieval and deletion

JSON-based document format (easily translatable across different languages)

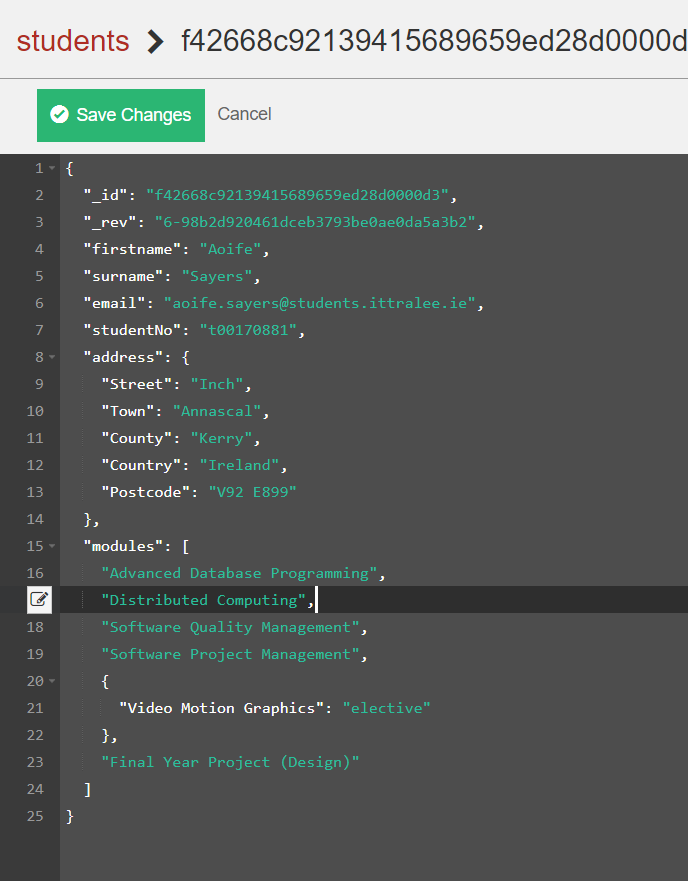
Multiple libraries for your language of your choice

MapReduce for simple, efficient, and comprehensive data retrieval. (Wasington, n.d.)

## 1. Implement your own CouchDB database (something requiring both a desktop/mobile app or mobile game implementation) and use the REST interface to show examples of CRUD. 40%

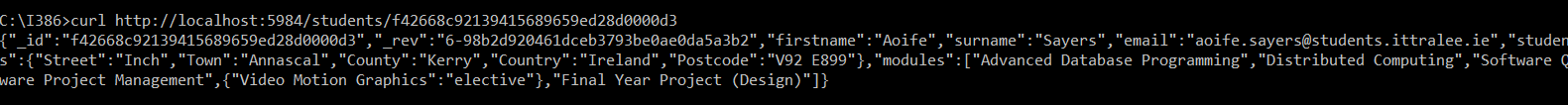
### CouchDB Database

* I created a document in the student database



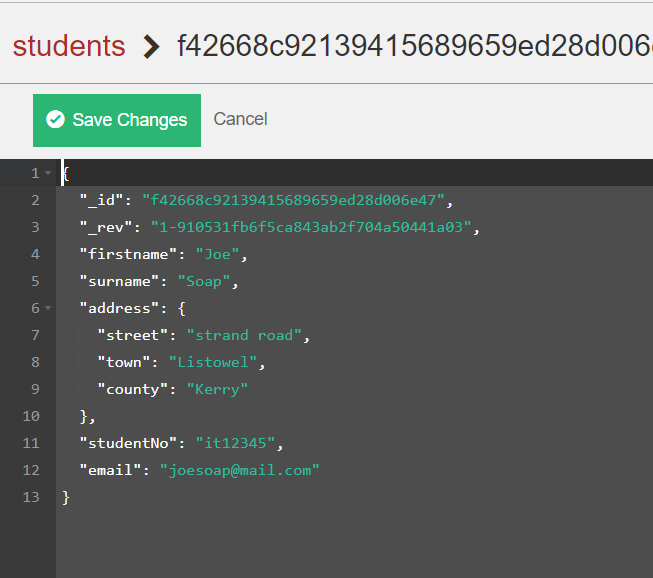
*Figure 1- Creating a document in the student database*

### RESTful Interface CRUD – READ

Issued a Curl get request to the student database using the ID as seen in the image above

*Figure 2 – Reading/ Getting from database*

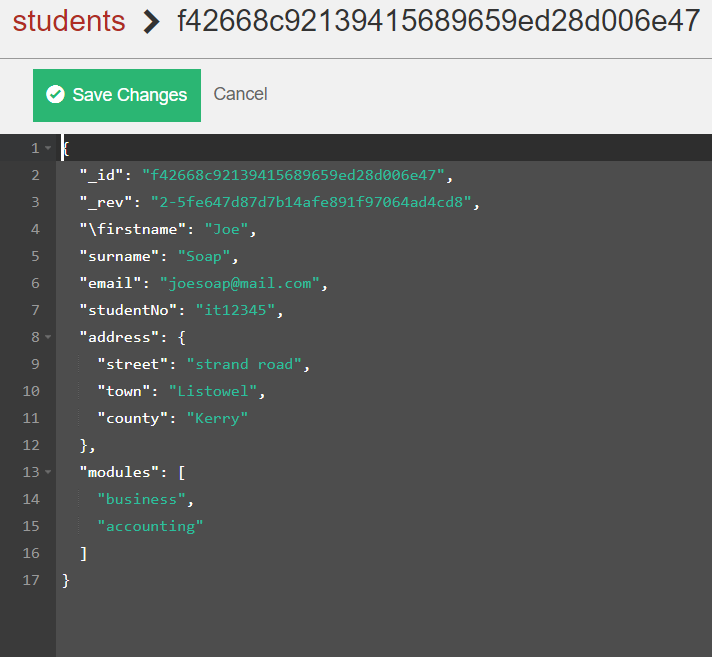
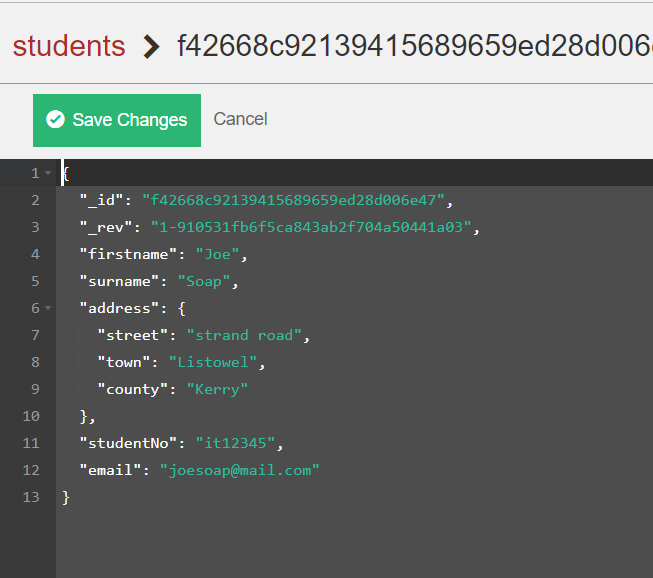
### RESTful Interface CRUD – CREATE

*Figure 3 – Creating/Inserting to the database*

Note the difference in the schema to the figure 1 database

*Figure 3.1 – Result on Fauxton interface for Creating/Inserting to the database*

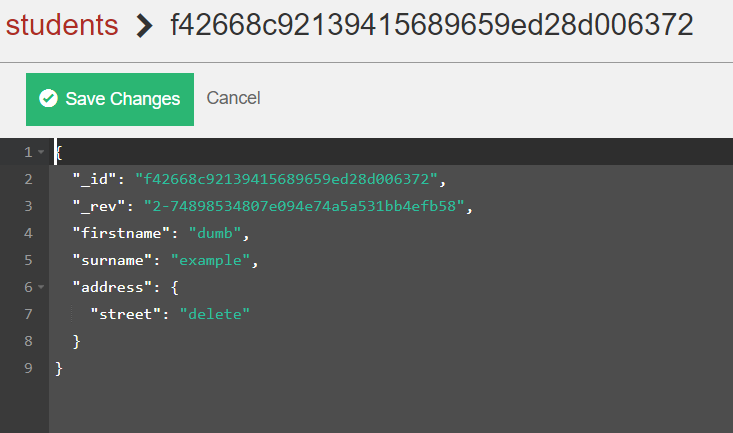
### RESTful Interface CRUD – UPDATE

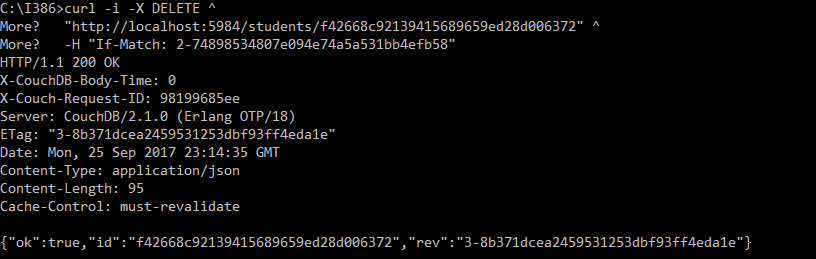
*Figure 4 - Updating documents in the database*

*Figure 4.1 – Updated document in the database Figure 4.2 – Document before it was updated*

### RESTful Interface CRUD – DELETE

*Figure 5 – Document to be deleted*



*Figure 5 – Deleting document via CURL CLI*

## 2. Download a driver for a programming language of your choice (Android would be nice!). Build a UIF in Java/C# etc and show the CRUD operations working through the UIF. Show a call to a mapreduce operation from within the programming language. Comment your code and show screen pictures of it working. (60%)

### Ektorp

I downloaded the Ektorp driver for Java. p Ektorp is a persistence API that uses CouchDB as storage engine. (helun, n.d.). Ektorp uses the Java class Jackson for mapping Objects to JSON. Ektorp gives the programmer a choice of abstraction level from full object document mapping to raw streams. . (helun, n.d.) It is also very easy to do CRUD due to it’s simple API, detailed documentation, and active growing community. Ektorp has an optional spring support module. Ektorp is set up in a Java Maven project specifying the Ektorp dependencies.

|  |  |
| --- | --- |
| Advantages of Ektorp  Object Mapping  Ektorp gives the programmer a choice of abstraction level from full object document mapping to raw streams. . (helun, n.d.)  Easy to do CRUD  Sensical methods  Good Documentation  Spring Support  Good community support  JSON annotations for mapping | Disadvantages of Ektorp  As CouchDB is web oriented, Java may not be the best choice of language for the web  Difficult to map nested objects and lists in Ektorp for JSON  Java isn’t the most popular choice for working with CouchDB. I personally thing NodeJS or PHP would have been more suitable. In hindsight, this was the first project I programmed with a NoSQL database and chose Java as it was in my comfort zone…! |

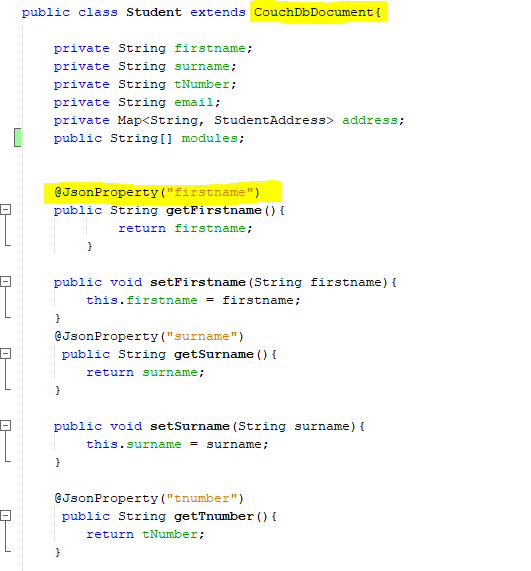
### Java UIF CRUD - CREATE

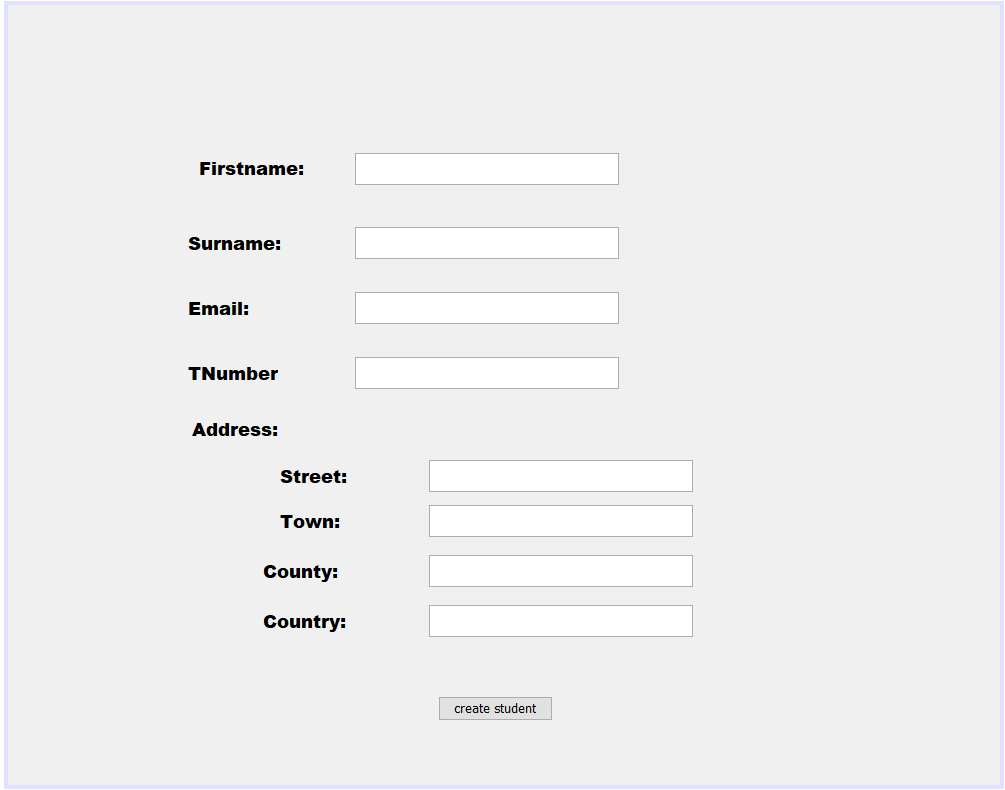
I completed the UI on Netbeans Swing JFrames

Link to full project on GitHub: <https://github.com/AoifeNicAntSaoir/EktorpJavaCouchDB>

**Student Object**

See the Jacksons Annotations for mapping to JSON *- @JsonProperty(“myProperty”)*

The StudentAddress Map attribute is for a nested JSON object e.g. {“firstname”: ”Aoife”, “surname”: “Sayers”, “tnumber”: “t00170881”, “address” : { “street”: “Strand street”, “town”: “Tralee”, “county”: “Kerry” }}



#### CREATE Code Snippet

//When the button is clicked

**private** **void** **btncreateActionPerformed**(java.awt.event.ActionEvent evt) {

HttpClient httpClient;

**try** {

//Connection details to localhost

httpClient = **new** StdHttpClient.Builder()

.url("http://localhost:5984")

.build();

//Connection to CouchDB server

CouchDbInstance dbInstance = **new** StdCouchDbInstance(httpClient);

//Connector to person database - creates database if it doesn't exist

CouchDbConnector db = dbInstance.createConnector("person", **true**);

//Creating a student object & setting the properties

Student s = **new** Student();

s.setFirstname(txtname.getText());

s.setSurname(txtsname.getText());

s.setEmail(txtemail.getText());

s.settNumber(txttnum.getText());

//For the nested Address Object

Map<String,StudentAddress> addr = **new** HashMap();

addr.put("address", **new** StudentAddress(txtStreet.getText(),txtTown.getText(), txtCounty.getText(), txtCountry.getText()));

s.setAddress(addr);

**try** {

//Creating/Inserting

db.create((s));

JOptionPane.showMessageDialog(**null**, "Student Created in the system", "Student created", JOptionPane.PLAIN\_MESSAGE);

//Clearing the UIF after insert successful

txtname.setText("");

txtsname.setText("");

txtemail.setText("");

txttnum.setText("");

txtStreet.setText("");

txtTown.setText("");

txtCounty.setText("");

txtCountry.setText("");

}

**catch**(Exception ex)

{

String exMessage = ex.getMessage();

JOptionPane.showMessageDialog(**null**, exMessage, "Error occured", JOptionPane.WARNING\_MESSAGE);

}

} **catch** (MalformedURLException ex) {

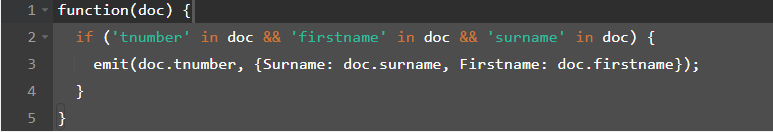
Logger.getLogger(CreateStudentGUI.class.getName()).log(Level.SEVERE, **null**, ex);

}

}

### Java UIF CRUD - READ

#### READ Code Snippet

Created a view to display the Tnumber as a key and 2 values – the firstname and surname as a value

