INFO-H415 ADVANCED DATABASES

DOCUMENT STORES AND COUCHDB

December 18, 2017

Ziad Beyens - Hamza Nougba
Université Libre de Bruxelles

Contents

Chapter 1

Introduction

The volume of data has significantly increased with the emergence of social networks, cloud, web and smartphone application, etc. NoSQL (Not Only SQL) was an alternative to this amount of data ensuring good performance and scalability without the need of relational schema.
Unlike SQL and its inherent tabular structure, NoSQL databases are generally schema-less and has more flexible data models, which is more useful for unstructured data and these are growing more quickly than structured data. There are four common types of NoSQL databases:

- Key-Value stores:
  These are clearly the simplest and the most flexible type because each key is associated with one and only one value in a collection of data.

- Wide-column stores:
  Wide-column stores look very similar to relational databases because they also consist in columns, rows and tables but they are much more dynamic because format of the columns are not predefined but created and possibly different for each row.

- Graph stores:
  Data are represented and stored using graph structures with nodes representing data themselves and edges representing relationships. Unlike relational databases a graph model can change over time and use.

- Document stores:
  The database stores semi-structured data into documents. There exist many document

formats and the most well-known remains xml. Document-oriented databases such as mongoDB and couchDB use other formats, respectively BSON and JSON.

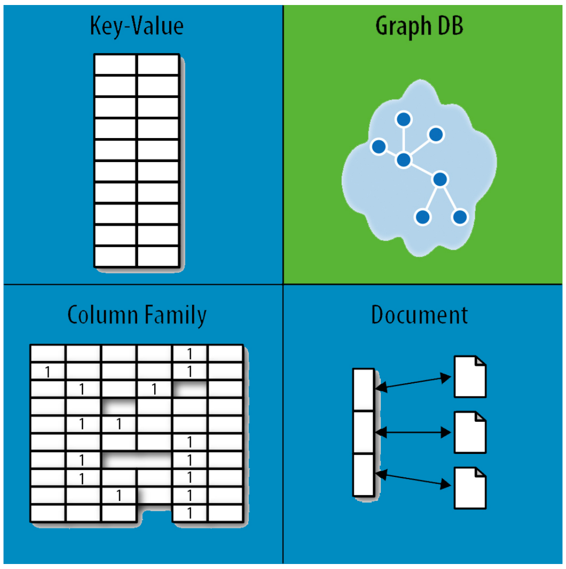


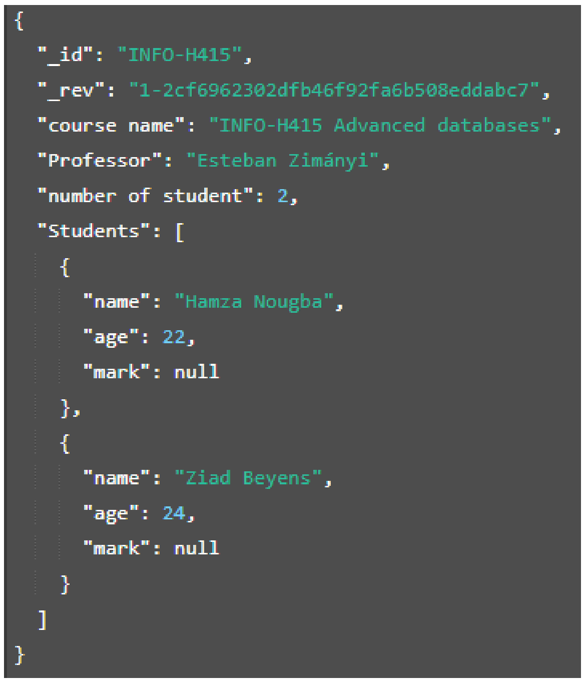
Figure 1.1: 4 types of NoSQL databases at `https://s3.amazonaws.com/dev.assets.neo4j.com/wp-content/uploads/nosql-quadrant.jpg`

1.1 JSON DOCUMENTS

JSON stands for JavaScript object Notation and is a lightweight data-interchange format where data are stored in an organized, easy-to-access way. It provides a human-readable set of data that can be accessed in a logical manner. It is based on a subset of JavaScript language (the way objects are built in JavaScript). However JSON remains language independent and can be use by other programming languages (JAVA, python, Ruby, ...).
As said above and compared to another well-known text format named XML, JSON is lightweight and consists technically in two possible structures:

- An object defined by an unordered collection of key-value pairs between two curly braces. Keys can and must be strings but values can be a strings, a number, an object, an array, a boolean or null.
- An array defined by an ordered collection of values between two brackets

A JSON document could be defined as an object with nested arrays an objects. CouchDB uses JSON format to store the data. Each document owns a distinct *_id* value and a *_rev* (revision) value. When a document is updated, the*_rev* value changes.



```
{
  "_id": "INFO-H415",
  "_rev": "1-2cf6962302dfb46f92fa6b508eddabc7",
  "course name": "INFO-H415 Advanced databases",
  "Professor": "Esteban Zimányi",
  "number of student": 2,
  "Students": [
    {
      "name": "Hamza Nougba",
      "age": 22,
      "mark": null
    },
    {
      "name": "Ziad Beyens",
      "age": 24,
      "mark": null
    }
  ]
}
```

Figure 1.2: Example of JSON document in CouchDB

1.2 REST/HTTP

RESTful services (REpresentational State Transfer) are used to communicate with CouchDB. It features an architectural client-server style, distributed systems and communication between the application and CouchDB.

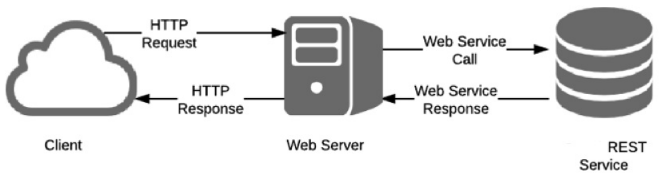


Figure 1.3: Client - Server - CouchDB representation at `https://smartdogservices.com/wp-content/uploads/2014/09/11.png`

It is stateless. It means that the servers manages the resource states and the client manages the session states. Thus, each request needs all the data to process the request.

REST is described by the Nouns, Verbs and Data formats.

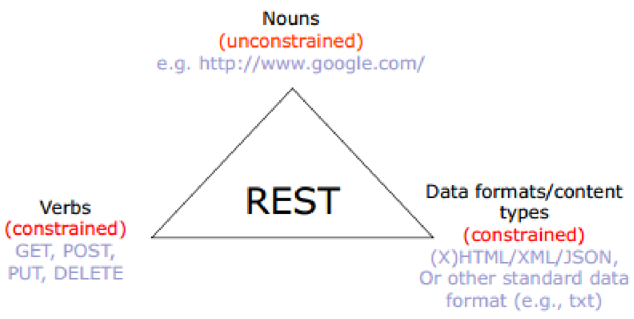
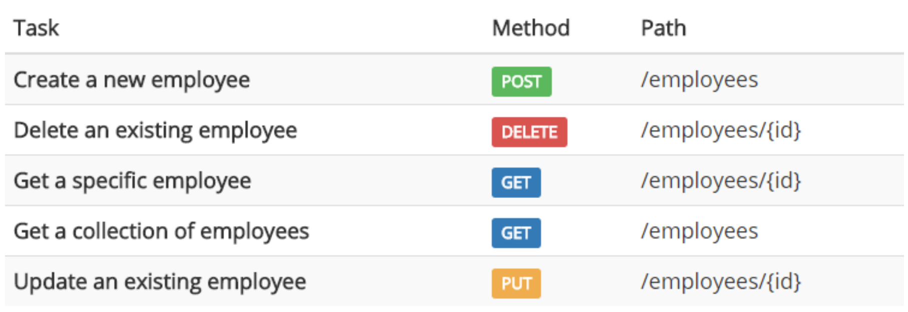

Nouns
(unconstrained)
e.g. http://www.google.com/
REST
Verbs
(constrained)
GET, POST,
PUT, DELETE
Data formats/content types
(constrained)
(X)HTML/XML/JSON,
Or other standard data format (e.g., txt)

Figure 1.4: REST triangle at `https://wso2.com/files/2_triangle_0.png`

The data formats may be HTML, XML, JSON,...
The verbs are HTTP methods to create (PUT, POST), read (GET), update (PUT), delete (DELETE) with CouchDB.



Task	Method	Path
Create a new employee	POST	/employees
Delete an existing employee	DELETE	/employees/{id}
Get a specific employee	GET	/employees/{id}
Get a collection of employees	GET	/employees
Update an existing employee	PUT	/employees/{id}

Figure 1.5: CRUD verbs at `http://www.kennethlange.com/img/restful_2/rest_http_methods.png`

The nouns are used for identification of resources. It is addressable through URIs.
Example: https://search/query?term=foo

Chapter 2

CouchDB

CouchDB is a document-oriented database that stores data in JSON format using HTTP as access protocol. JavaScript allows to query, combine and transform data in the documents. CouchDB is an open-source project created in 2005 by a developer at IBM, Damien Katz.

2.1 Basic operations

Communicating with couchDB can be done using command line with cURL utility or accessing a web interface named Fauxton (or Futon).

2.1.1 Creating a database

A database is created using the PUT method with cURL:

```
curl -X PUT http://127.0.0.1:5984/database_name
```

It will then reply:

```
{"ok": true}
```

The creation of the database can be verified using GET method:

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

the response will be the list of all databases:

```
["database_name"]
```

2.1.2 Deleting a database

Logically the DELETE method is used to delete a database:

```
curl -X DELETE http://127.0.0.1:5984/database_name
```

The verification can be done similarly to the creation of databases using GET method.

2.1.3 Creating a document

The creation of a document uses PUT method:

```
curl -X PUT http://127.0.0.1:5984/database_name/"id" -d  "content of the document"
```

_id value of the document is directly defined in the command line and -d option specifies that a document is following.

2.1.4 Updating a document

```
curl -X PUT http://127.0.0.1:5984/database_name/document_id/ -d \
"{ "key" : "value", "_rev" : "revision id" }"
```

To update a document the last *_rev* must be known and this number can be easily retrieve using GET method:

```
curl -X GET http://127.0.0.1:5984/my_database/document_id
```

2.1.5 Deleting a document

The last _rev value must be specified. :

```
curl -X DELETE http://127.0.0.1:5984/database_name/document_id?_rev
```

2.2 VIEWS

2.2.1 Concept

Views can be seen as dynamic representations of document content. They are very useful and have many purposes such as:

- querying and searching documents stored in the database.

- getting data from documents presented in a specific manner.
- creating b-tree indexes such that data in documents can be found easily. These indexes allows to represent some relationships between documents.
- performing some calculations on the data from documents.

In CouchDB, views are defined using JavaScript functions that take as single parameter a document and are representing as a map function in a map-reduce system.

2.2.2 Map function

As said above a map function takes only one parameter, a document. It results as an emission of key-value pairs using the function *emit(key, value)*. When a map function is called, the view adds rows (the number of rows depends on the number of *emit* inside the function). The views stored these data using a B-tree such that it is sorted according to the key value of each row. Let's see this example to illustrate the concept of view:

```
function(doc) {
    emit(doc.course_name, doc.number_of_student);
}
```

The map function returns a key-value with name of the course as name and the number of students as value.When this function is called, a row is added to the view. Therefore the b-tree is created with indexes representing the name of the course.

```
{
"total_rows" : 2,
"offset": 0,
"rows": [
          {"id": "INFO-H415"
           "key": "INFO-H415 Advanced databases",
           "value": 2},
          {"id": "INFO-H502",
           "key": "INFO-H502 Virtual reality",
           "value": 5}
         ]
}
```

This is the view when the map function is called two times.

2.2.3 Reduce function

As the name said , the principle goal of this function is to reduce the result of the map function. The reduce function then takes as input three parameters:

- *keys*: it represents an array of [key, doc_id]'s present in the view. According to the previous example, the keys should equal to [["INFO-H415 Advanced databases","id1"], ["INFO-H502 Virtual reality","id2]].
- *values*: an array of corresponding values present in the view. According to the previous example, it should equal to [2,5].
- *rereduce*: one particularity of the reduce function is that it can be applied recursively. CouchDB applies the reduce function to the calculated values, and will repeat the process on these resulting values. The rereduce parameter is boolean value and when it equals to *true*, the 'keys' parameter is null and the 'values' parameter is an array of intermediate values return by the reduce function itself. Rereduce property may be necessary in some cases when the number of rows in a view is very large. For example if we have 1000 elements [key, value] in the view, the reduce function is applied to $n \leq 1000$ values $\lceil \frac{1000}{n} \rceil$ times. The function is again applied to resulting values until it remains one element.

2.3 REPLICATION

In Biology, replication is a process when the DNA is synthesized producing two replicas of itself. It follows the same logic for databases in Computer Science. Database replication is the process of copying data from a database in a computer or a server to one or more databases at the same location or everywhere else. This resulting system constitutes a distributed database system.
In CouchDB, two arguments is needed for the replication: a source database and a target database. Each data in the source will be copied into the target such that it results two databases with the same content. A replication can be continuous and at each update of a document or at each deletion or creation of document the process is triggered.
Replication has many advantages:

- Horizontal scalability: increasing the number of instances of the database. Therefore, less users read the data from one database (parallel access) and as the network is less congested, the access time decreases.

- The availability is higher and there are less data movements over a network.
- It is practical for offline use. For example, a replication in a smartphone allows the user to remain offline and then synchronizes the data when it is back online.
- Optimal for reading but not for writing.

There are also some inconvenient such as having more disk space to store all these replicas and having a good concurrency control to avoid many conflicts. Hopefully, CouchDB uses a Multiversion Concurrency control.

Chapter 3

Chat Application

3.1 DESCRIPTION

An web implementation of a chat application using CouchDB and NodeJS has been done. It consists of anonymous users being able to register their name and chatting in real-time.
The user names and the messages are saved in the database. Thus, refreshing the page recover the the user names and the history of the conversation. Also, the user can choose any name available in the list.

3.2 IMPLEMENTATION

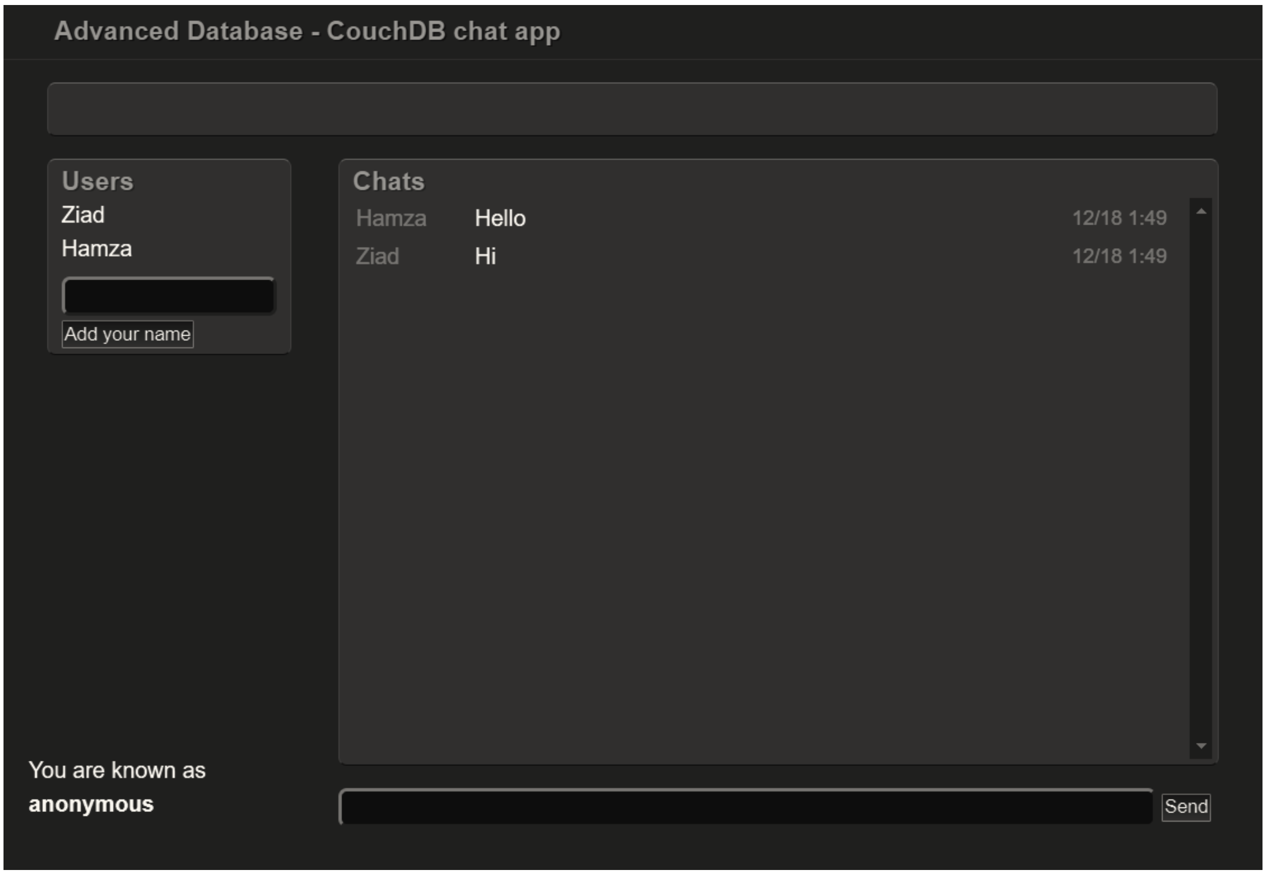
The server is set up on Node.js and a Websocket library (socket.io) is used to chat in real-time. Furthermore, a high-level CouchDB client for Node.js (cradle) is used. More precisely, Cradle is an asynchronous javascript client for CouchDB. Cradle also has an extra level of speed, and it is easier to update and delete document.
First, the database ’adb’ is created in Fauxton, then configured and setup on the server:

```
db = new(cradle.Connection)(couch_config).database('adb');
```

Then, the users views and the chats views are added to be able to query them, using the map function.

On client connection, the user list is retrieved from CouchDB and sent to the connected user. Then, the chat history is fetched from CouchDB and sent to the user.

When a client posts a message, it is saved into CouchDB and broadcasted to all the connected clients. The type (’chat’), the message content, the author name and the timestamp are saved.



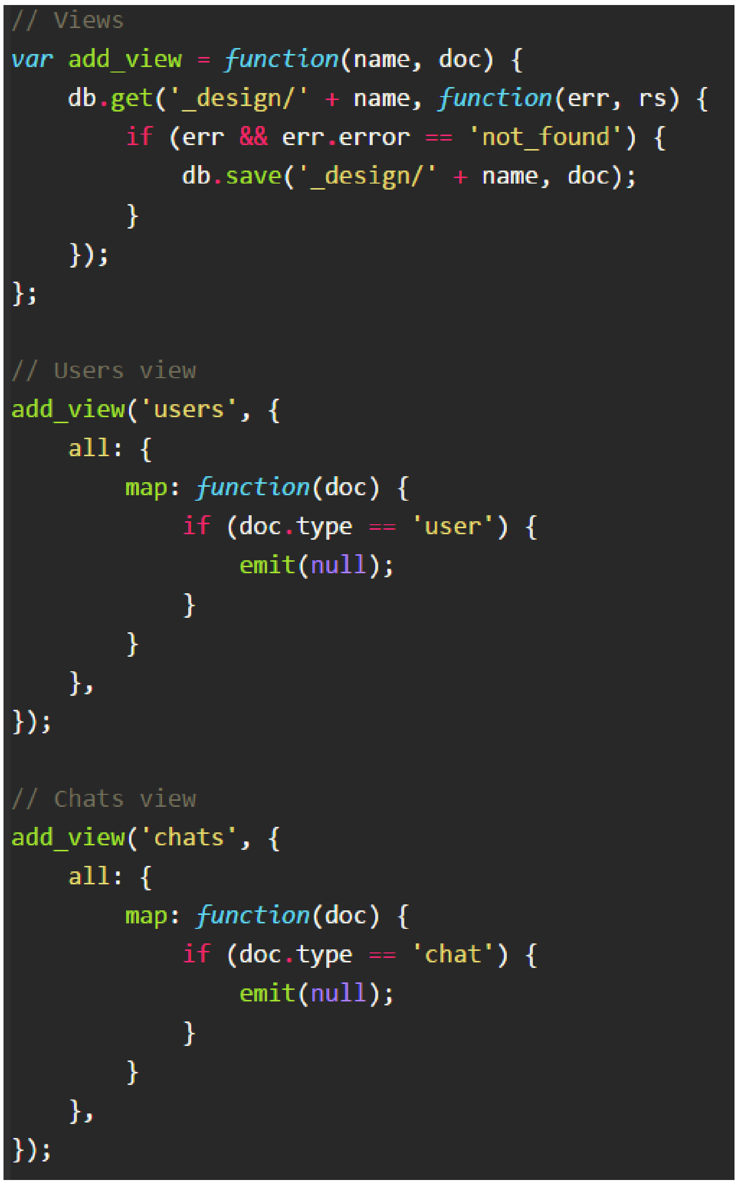
Ziad
Hamza

Hamza	Hello	12/18 1:49
Ziad	Hi	12/18 1:49

You are known as **anonymous**

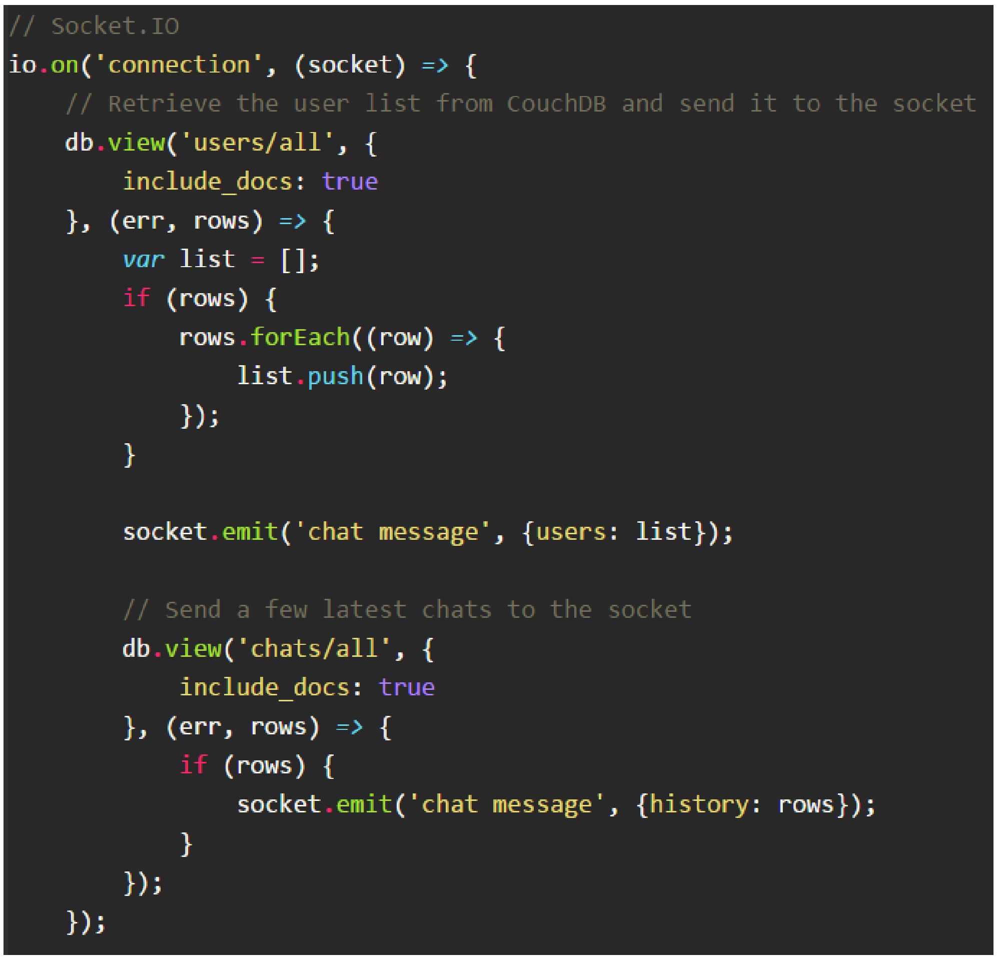
Figure 3.1: Chat application

Finally, when a client adds an user, the type ('user'), the name and the status ('online') are saved.



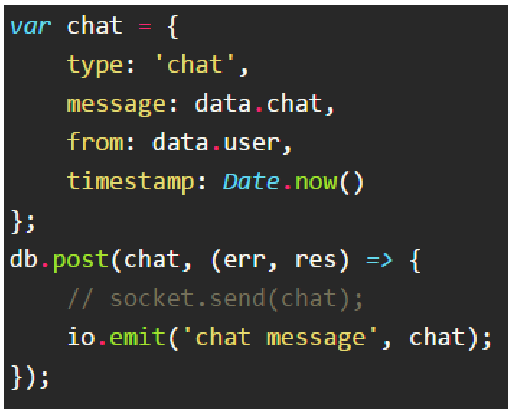
```javascript
// Views
var add_view = function(name, doc) {
    db.get('_design/' + name, function(err, rs) {
        if (err && err.error == 'not_found') {
            db.save('_design/' + name, doc);
        }
    });
};

// Users view
add_view('users', {
    all: {
        map: function(doc) {
            if (doc.type == 'user') {
                emit(null);
            }
        }
    },
});

// Chats view
add_view('chats', {
    all: {
        map: function(doc) {
            if (doc.type == 'chat') {
                emit(null);
            }
        }
    },
});
```

Figure 3.2: Adding views



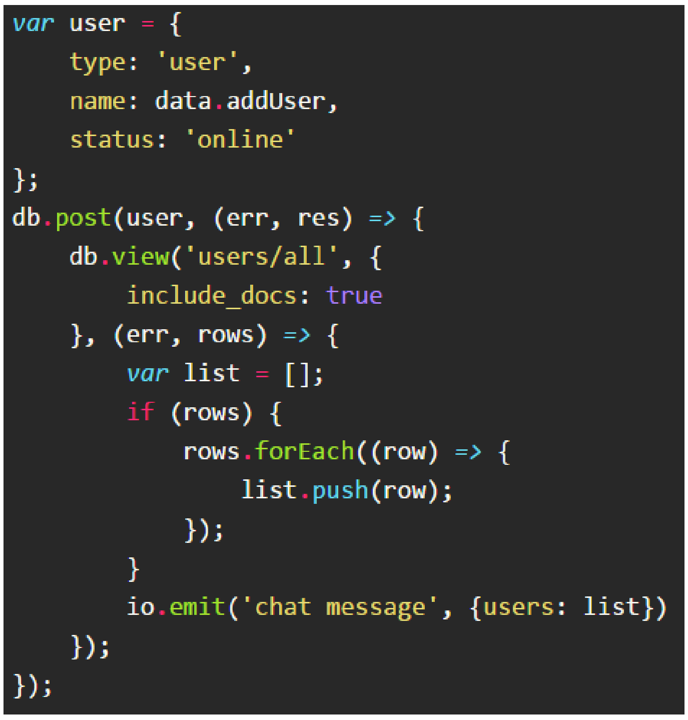
```
// Socket.IO
io.on('connection', (socket) => {
    // Retrieve the user list from CouchDB and send it to the socket
    db.view('users/all', {
        include_docs: true
    }, (err, rows) => {
        var list = [];
        if (rows) {
            rows.forEach((row) => {
                list.push(row);
            });
        }

        socket.emit('chat message', {users: list});

        // Send a few latest chats to the socket
        db.view('chats/all', {
            include_docs: true
        }, (err, rows) => {
            if (rows) {
                socket.emit('chat message', {history: rows});
            }
        });
    });
```

Figure 3.3: Fetching all the users and messages on client connection



```
var chat = {
    type: 'chat',
    message: data.chat,
    from: data.user,
    timestamp: Date.now()
};
db.post(chat, (err, res) => {
    // socket.send(chat);
    io.emit('chat message', chat);
});
```

Figure 3.4: Posting a message



```
var user = {
    type: 'user',
    name: data.addUser,
    status: 'online'
};
db.post(user, (err, res) => {
    db.view('users/all', {
        include_docs: true
    }, (err, rows) => {
        var list = [];
        if (rows) {
            rows.forEach((row) => {
                list.push(row);
            });
        }
        io.emit('chat message', {users: list})
    });
});
```

Figure 3.5: Posting a user

Bibliography

[1] Apache couchdb 2.1 documentation. `http://docs.couchdb.org/en/2.1.1/`. Accessed: 2017-12-17.

[2] Couchdb tutorial. `https://www.tutorialspoint.com/couchdb/`. Accessed: 2017-12-17.

[3] Http view api. `https://wiki.apache.org/couchdb/HTTP_view_API`. Accessed: 2017-12-17.

[4] Introducing json. `https://www.json.org/index.html`. Accessed: 2017-12-17.

[5] Introduction to rest and couchdb. `https://www.slideshare.net/partlycloudy/introduction-to-rest-couchdb`. Accessed: 2017-12-17.

[6] Nosql databases explained. `https://www.mongodb.com/nosql-explained`. Accessed: 2017-12-17.

[7] Nosql (not only sql database). `http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL`. Accessed: 2017-12-15.