Cloud Computing (INGI2145) - Lab Session 7

Waleed Reda, Marco Canini

1. Background:

In this tutorial, you'll learn how to apply some of the web-programming know-hows that have been covered in the previous lectures. We will basically develop our own Node.js application and interact with a MongoDB instance. This application will perform rudimentary user authentication as well as database searches based on certain indexes. We'll also go through how to use AJAX to update certain pages without having to reload the entire contents. In all cases, we provide a skeleton and general guidance in order to implement the required tasks.

2. Tutorial:

Setup:

1. To download the Lab7 exercise, first git pull (or git clone if you don't have a pre-existing local repository) from the course's public repository on Github (located here: https://github.com/mcanini/INGI2145-2014).
2. **To setup the VM for these exercises**, you need to re-provision the system in order to install MongoDB. To do this, open your terminal and cd to your vagrant directory and input `vagrant up --provision`.
3. After downloading the exercise, cd to /vagrant/lab7 and run npm install in order to download all the necessary node modules required for running this exercise.
4. Next, let's populate the database with a collection of users. Simply use the command `nodejs loader.js` to load 5000 randomly generated user date into a collection called "users". Each user is defined by these fields: username ("user1" … "userN"), first_name, last_name, country (3-digit code), and group (a set with a combination of "author", "commenter" or "viewer"). For simplicity, the password is set to be a hash of the username.
5. You can then use the command `nodejs app.js` in order to run the exercise's Node.js application and use your browser to go to http://localhost:3000, which is the app's landing URL

Routes:

- The routes for our Node.js application are set-up as follows:
    - GET /
        - Goes to the login-page
    - GET /home
        - Goes to the homepage
    - POST /search
        - Performs the search operation
    - POST /validate

- Performs the login operation

I. User authentication

As you may have noticed, accessing http://localhost:3000 automatically redirects you to the login page. However, note that login operation is at this point non-functional.

Task:

Before starting to work on this, you should check out the following files:

- routes.js: This file is used by the Node.js application to maintain mappings between routes and server functionality (e.g., it discerns which view will get rendered when we access our "/home" page).
- account-manager.js: This file contains auxiliary functions that allow authenticating users.

a. Log-in Authentication and maintaining sessions:

You're asked to first modify the "account-manager.js" in order to implement the `manualLogin` function. This involves querying MongoDB for the username, and if a match is found, checking for equality of passwords. Then you should check the "/validate" route in "routes.js". This route receives two parameters: username and password. It should use the `manualLogin` function to try to authenticate the user. If successful, it stores the user information as session state in a cookie (via express's `req.session` object).

b. Redirects:

You should then use these maintained sessions that you've implemented in the previous step to redirect logged-in users to the homepage whenever they access "/", and redirect guests to the login page whenever they access "/home". These can be implemented in the "routes.js" file.

II. Indexing

After finishing the previous task, you should now be able to access the homepage only as a logged-in user. For this sub-section, we want to assess the different types of indexing that can be performed in MongoDB and on what factors are different indexes more efficient.

For the purpose of this exercise, we want you to perform certain queries on our MongoDB database using the search page. You'll then refer to the terminal in which the Node.js server is running to observe two important parameters: "nscannedObjects" and "nscanned". The former refers to the number of objects that have been scanned to produce the query results. The latter is calculated by adding the nscannedObjects plus the number of scanned indexes. These two parameters can give you an indication about how much faster has indexing made our query. For

instance, if we had to scan the entire collection in order to return a subset of documents, then indexing is more or less useless in this scenario.

If you've already started working on the second homework assignment and covered background material on MongoDB, you should be somewhat familiar with how indexing works. However, for those that didn't, we'll try to provide some insights on the different types of indexes that will be relevant for this exercise (for more information, look here: http://docs.mongodb.org/manual/indexes/).

MongoDB Index types (not a comprehensive list):

> ***Single Field Indexes***
>
> *A single field index only includes data from a single field of the documents in a collection. MongoDB supports single field indexes on fields at the top level of a document and on fields in sub-documents.*
>
> ***Compound Indexes***
>
> *A compound index includes more than one field of the documents in a collection. However, note that the ordering matters for these types of indexes. Let's take this compound index as an example: {name: 1, age: 1}. This index will first sort the table in ascending order by names and then sorts it by age. This concept is important to understand for the upcoming exercises.*
>
> ***Multikey Indexes***
>
> *A multikey index is an index on an array field, adding an index key for each value in the array. These types of indexes can be helpful if, for example, we're searching for documents containing certain keywords.*

Task:

Throughout this exercise, you'll run certain queries and report outputs. You can do this by accessing the provided "/search" page.

For the first part of the exercise and run the following queries:

- {group: "author"}
- {group: "viewer"}

Note the number of "nscannedObjects" and "nscanned" for each query. Now, go to the "routes.js" module and refer to the "POST /search" route. Try to change the indexing of MongoDB (in the "app.js" file) and re-run the same queries. Note down any differences, and reason about why they occur.

After meddling with these results, now try running the following queries:

- {first_name: “Tiffany-Elaine”, country: “SSD”, group:”author”}
- {last_name: “Barrington”, country: “ZIM”, group:”viewer”}

Again, note down the number of “nscannedObjects” and “nscanned” for each query. Try to index the collection this time using a compound index. Limit your compound index to only two fields and experiment using different permutations and observe how these parameters change. Try using three fields and compare the results further. Ask yourself why this happens in light of MongoDB’s compound index functionality. Also, what does that tell you about indexing in relation to cardinality?

Hints:

- **Open the “users.json” file and try to observe general data patterns and trends. This might give you some clues as to why different indexing methods have differing levels of effectiveness.**
- **Refer to the Node.js driver documentation: http://mongodb.github.io/node-mongodb-native/contents.html for more information on the syntax of the different MongoDB operations**

III. AJAX using JQuery

In this last exercise, we ask you to modify the login page such that it provides user error feedback (e.g. ‘wrong password’ or ‘user doesn’t exist’) using AJAX instead of having to reload the same page. As a reference, you can observe how AJAX is implemented for the “/search” page and try to replicate the same techniques for the login view. As such, you’re encouraged to read through the following files:

- “/views/home.ejs”: Walkthrough the AJAX scripting parts and make sure to read the comments.
- “/routes.js”: Observe what’s being executed for the “POST /search” route and how it differs from the “POST /validate” route.

Before implementing this, note down the differences between both approaches and reason about why they’re necessary. You can also refer to the jQuery AJAX API for more information: http://api.jquery.com/jquery.ajax/