INTRODUCTION AU BIG DATA

RAPPORT

Application web : Recommandation de livres à partir d'outils BigData

Sous la direction de :

Pr **Nassima SOUSSI**

Réalisé par :

AMIDOU Abdoul Rahmane

GUIDIBI Teddy

MONKOUN B. Aris Merlix

Niveau: IID2

2021 -2022

INTRODUTION

Le Big Data, à la suite de la mondialisation, a conduit à de nombreux changement dans les habitudes des uns et des autres. Il est alors devenu impératif de repenser aux solutions proposées auparavant.

Les livres ne sont pas en marge de cette révolution puisque depuis que les clients passent plus de temps sur leurs smartphones et autres appareils électroniques, on remarque une chute nette de l'attirance même des plus petits vers les livres.

L'idée est alors venue de passer par les moyens du moment pour atteindre les lecteurs : les bibliothèques numériques ou les ebooks proposes par une panoplie d'application.

C'est dans cette même dynamique que nous nous plaçons afin d'apporter notre pièce à l'édifice en y ajoutant la touche big data afin d'avoir accès à toujours plus de livres et beaucoup plus rapidement.

Outils utilisés & configurations à effectuer :

- Base de données : MongoDB - Le DataSet du TP1
  (A mettre dans la collection 'books200' d'une base de données nommée 'TP1')
  - Créer dans cette meme base de données deux collections 'users' et 'ratings'

- Scripts : Configuration de l'environnement du projet en Java

  - Librairie Mahout créée disponible dans /LibraryMahout
  - Librairie Mongo créée disponible dans /LibraryMongo
    - Les deux étant à insérer dans le BuildPath du projet

  - Pour configurer le de telle manière qu'il puisse compiler les classes de Mahout et Mongo suivre les étapes suivantes: (Dans l'IDE Eclipse)

    - Clic droit sur le projet
    - Run as
    - Run configurations
    - Selectionner le server Apache
    - ClassPath
    - Cliquer sur user entities
    - Add external jars
    - Ajouter toutes des jar dans library mahout et Mongo

- Technologies Web : HTML/CSS, JS(Ajax), JEE(.JSP)

DEROULEMENT

1. Première étape : Faire la recommandation sur un petit dataset

La librairie mahoot développée pour faire du machine Learning dans l'environnement Hadoop et en dehors nous a permis de tester deux approches de Mahout sur les moteurs de recommandations : celle basée sur les articles (ItemBasedRecommendation) ou sur les utilisateurs (UserBasedRecommendation) que nous avons décidé d'implémenter.

- Phase de Test sur un script :

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.apache.log4j.BasicConfigurator;
import org.apache.mahout.cf.taste.common.TasteException;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;

public class App
{
    public App() {
            super();
       }
    public static void main( String[] args ) throws IOException, TasteException
    {
       BasicConfigurator.configure();
       DataModel model = new FileDataModel(new File("dataset/lastfile.csv"));
       UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
       UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity,
       model);
       UserBasedRecommender recommender = new GenericUserBasedRecommender(model,
       neighborhood, similarity);
       List<RecommendedItem> recommendations = recommender.recommend(2,1);
       for (RecommendedItem recommendation : recommendations) {
         System.out.println(recommendation.getItemID());
       }
   }
}
```

- Echantion du contenu dataset :

```
4,00,5.0
4,01,5.0
4,02,5.0
4,03,0.0
```

- Sous le format :id_user, id_produit, ratings
- Et avoir des resultats sous la forme : [idItem]

- Création d'une librairie LibraryMahout à base de la collecte des fichiers .jar issus des MavenDepencies depuis le local

...

Les chemins d'accès à chaque fichier sont spécifiés donc il suffira de remonter l'arborescence de d'aller récupérer le fichier .jar en question. Cette dernière est disponible dans le dossier LibraryMahout.

- Etude du processus de recommendation sur Mahout :

Pour implementer cette recommandation, il faut recuperer le modele de donnees qui doit repondre au format de l'echantillon cite un peu plus haut.

Une fois recuperee, il faut definir la metrique a utiliser pour calculer la similarite entre les users.Plusieurs metriques peuvent etre utiliser et leurs descriptions se trouvent ici.

Il faut ensuite recuperer les utilisateurs proches(le neighborhood en precisant le seuil de la metrique au-dessus duquel la recommandation va etre faite et enfin generer les articles recommandes.

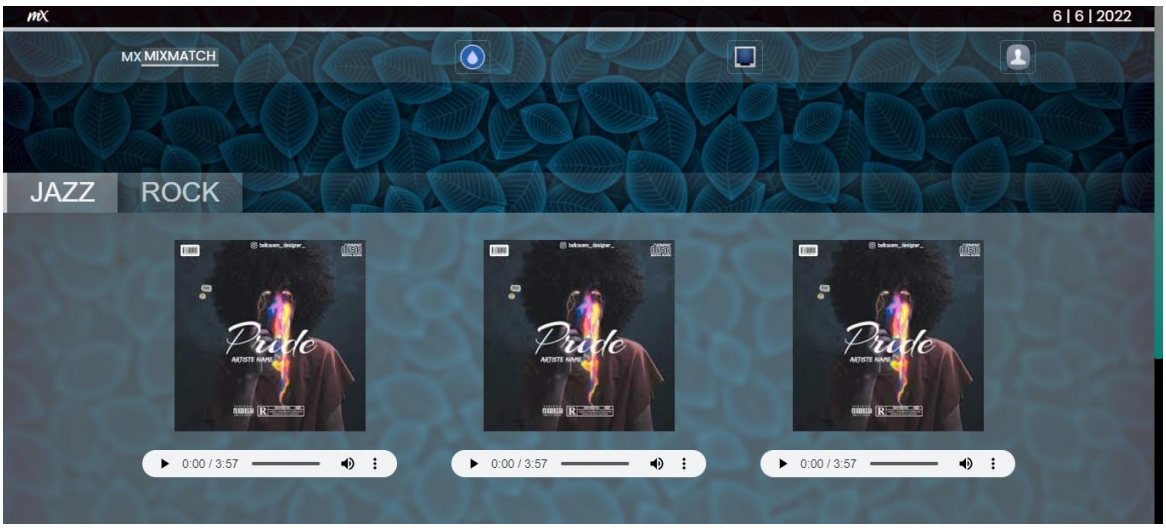
Nous avons choisi le coefficient de correlation de pearson dont les valeurs denotent du lien entre les utilisateurs tel que :

Strength of Association	Positive	Negative
Small	.1 to .3	-0.1 to -0.3
Medium	.3 to .5	-0.3 to -0.5
Large	.5 to 1.0	-0.5 to -1.0

2. Deuxième étape : Trouver une Base de données pour le Travail

× Premier Prototype : Sur des musiques :

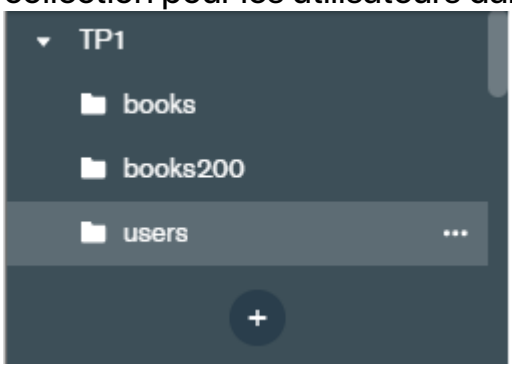
Pour ce faire, on a commencé la réalisation d'une application Web statique juste du coté Frontend :



- Par la suite on s'est heurtés à la récupération du DataSet depuis le site qui non seulement était très volumineux, mais aussi, ne nous offrait pas la garantie d'avoir les covers des albums concernant la partie visuelle su site.

× Deuxième Prototype : Sur des livres :

- Cette fois-ci on a préféré commencer par d'abord trouver le DataSet et adapter notre ébauche de site en fonction.
- Le choix a été porté sur le DataSet des livres du TP1 de BigData auquel, quelques requêtes de filtrage avaient déjà commencé à y être appliquées
- On a donc en plus de la collection contenant les livres, créé une autre collection pour les utilisateurs dans la meme base de données.



3. Prémices du second Prototype :

- Vu qu'on avait déjà Mahout de déjà implémenté en Java, on a donc décidé d'implémenter notre backend en JEE.

- Mais avant il faudra trouver un moyen de connecter MongoDB dans Java.

- Un projet Maven a donc été créé (pour étudier les opérations offertes par l'API de MongoDB) avec les dépendances suivantes pour télécharger les jars utiles.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.mongodb</groupId>
  <artifactId>mongo-java-driver</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>

      <dependency>
            <groupId>org.mongodb</groupId>
      <artifactId>mongodb-driver-sync</artifactId>
      <version>4.1.0-beta2</version>
      </dependency>

      <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
        <version>2.12.3</version>
    </dependency>

      <dependency>
            <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.7.13</version>
      </dependency>

      <dependency>
            <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.7.13</version>
      </dependency>

  </dependencies>
</project>
```

Au départ on devait juste utiliser des classes de mongodb-driver-sync|4.1.0-beta2 mais chose est que certaines classes n'ont pas le fonctionnement voulu, donc on a jugé utile d'y coupler mongo-java-driver | 2.12.3 qui aborde la problématique avec quasiment les mêmes classes, mais sous un autre angle.

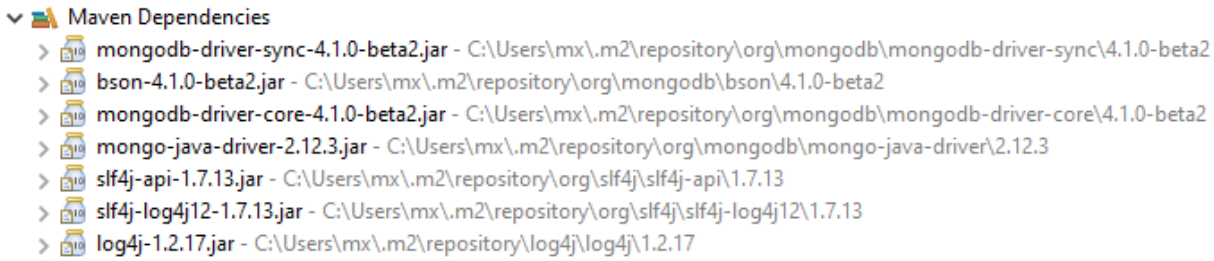
- Phase de Test sur un script :

```
import java.util.Arrays;
import java.util.List;

import org.bson.Document;

import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.DBObject;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoIterable;

public class MongoDBApp {

public static void main(String[] args) {

	//Creer une instance client Mongo
	MongoClient mongoClient =MongoClients.create();

	//Lister les DataBases
	MongoIterable<String> dbNames = null;
	try {
		dbNames = mongoClient.listDatabaseNames();
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	for (String dbName : dbNames) {
		System.out.println(dbName);
	}

	//Récupérer la base de données
	//Si elle n'existe pas, MongoDB va la créer
	MongoDatabase mydatabase = mongoClient.getDatabase("TP1");

	//Récupérer la collection
	//Si elle n'existe pas, MongoDB va le créer
	FindIterable<Document> mydatabaserecords = mydatabase.getCollection("books").find();
	MongoCursor<Document> iterator = mydatabaserecords.iterator();

	while (iterator.hasNext()) {
		Document doc = iterator.next();
		System.out.println(""+doc);
	}

	BasicDBObject doc1 = new BasicDBObject();
	Doc1.append("type", "Exception");
	Doc1.append("time", System.currentTimeMillis());

	 Document doc2 = new Document("myKey", "myValue");
	 String jsonString = doc2.toJson();
	 Document doc21 = Document.parse(jsonString);

	 MongoCollection collection = mydatabase.getCollection("books");
	 collection.insertOne(doc21);

	}
```

L'ajout de Doc1 n'a pas pu être fait car l'API offerte par MongoDB en Java ne permet que de créer des documents au format bson, donc il a été nécessaire de la convertir en json pour pouvoir l'insérer dans notre collection.

Ce qui a été fait au préalable pour le Doc2.

- Création d'une librairie LibraryMongo à base de la collecte des fichiers .jar issus des MavenDepencies depuis le local



Les chemins d'accès à chaque fichier sont spécifiés donc il suffira de remonter l'arborescence de d'aller récupérer le fichier .jar en question. Cette dernière est disponible dans le dossier LibraryMongo

La meme chose a été faite du coté de Mahout pour générer LibraryMahout.

4. Développement du second Prototype en JEE :

On a d'abord créé un DynamicWebProject Book_WebApp dans lequel on a procédé à l'insertion de notre librairie LibraryMahout.

- On a, sur la base de cette librairie, créé notre propre classe MongoDBApp pour effectuer les transactions en quelques méthodes simples.

```
package pack_sevlet;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.bson.Document;

import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.DBObject;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoIterable;

public class MongoDBApp {

	MongoClient mongoClient=null;
	MongoDatabase mydatabase=null;

	//Constructeur par défaut
	public void MongoDBApp() {
		//Creer une instance client Mongo
		this.mongoClient =MongoClients.create();
	}

	//Constructeur pour directement créer une instance de base de données
	public MongoDBApp(String nom_db) {
		//Creer une instance client Mongo
		this.mongoClient =MongoClients.create();

		//Récupérer la base de données
		this.mydatabase = mongoClient.getDatabase(nom_db);
	}

	//Méthodes pour lister les DataBases
	public void ListDatabases() {
		//Lister les DataBases
		MongoIterable<String> dbNames = null;
		try {
			dbNames = this.mongoClient.listDatabaseNames();
		} catch (Exception e) {
			e.printStackTrace();
		}

		for (String dbName : dbNames) {
			System.out.println(dbName);
		}
	}
```

```
      public void ListCollection(String nom_collection) {

        //Récupérer la collection et l'insérer dans un Iterable par find()
        FindIterable<Document> mydatabaserecords =    mydatabase.getCollection(nom_collection).find();

        //Créer un itérateur ou curseur le pour parcourir
        MongoCursor<Document> iterator = mydatabaserecords.iterator();

        //Parcours de l'itérateur pour récupérer les données
        while (iterator.hasNext()) {
            Document doc = iterator.next();
            // do something with document
            System.out.println(""+doc);
            System.out.println(""+doc.get("thumbnailUrl"));
        }
      }



      public ArrayList<String> GetFieldInCollection(String nom_collection, String nom_champ)
{

        //Récupérer la collection et l'insérer dans un Iterable par find()
        FindIterable<Document> mydatabaserecords = mydatabase.getCollection(nom_collection).find();

        //Créer un itérateur ou curseur le pour parcourir
        MongoCursor<Document> iterator = mydatabaserecords.iterator();

        //Parcours de l'itérateur pour récupérer les données
        List <String> l= new ArrayList<String>();
        while (iterator.hasNext()) {
            Document doc = iterator.next();
            String aa=doc.getString("thumbnailUrl");
            l.add(aa);
        }

        return (ArrayList<String>) l;
      }


       public void insertDoc(Document doc2, String nom_collection) {

        //Mettre le document Bson sous forme String au format Json
        String jsonString = doc2.toJson();
        //Convertir en Document Json
        Document doc21 = Document.parse(jsonString);

        //Récupérer la collection
        MongoCollection collection = mydatabase.getCollection(nom_collection);

        //Y insérer le document
        collection.insertOne(doc21);
      }
```

- On a procédé à la phase de Test de notre classe MongoDBApp dans une ClasseConsole Java TestMongo

```java
package pack_sevlet;

import java.util.ArrayList;

import org.bson.Document;

import com.mongodb.client.MongoCursor;

//import MongoDBApp;

public class TestMongo {

	public static void main(String[] args) {

		MongoDBApp M2=new MongoDBApp("TP1");

		ArrayList<String> mc= M2.GetListCollection("books200","thumbnailUrl");
		System.out.println(mc.get(0));

	}

}
```

```
log4j:WARN No appenders could be found for logger (org.mongodb.driver.cluster).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/ableson.jpg
https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/ableson.jpg
```

- On a voulu récupérer ces mêmes infos depuis une Servlet

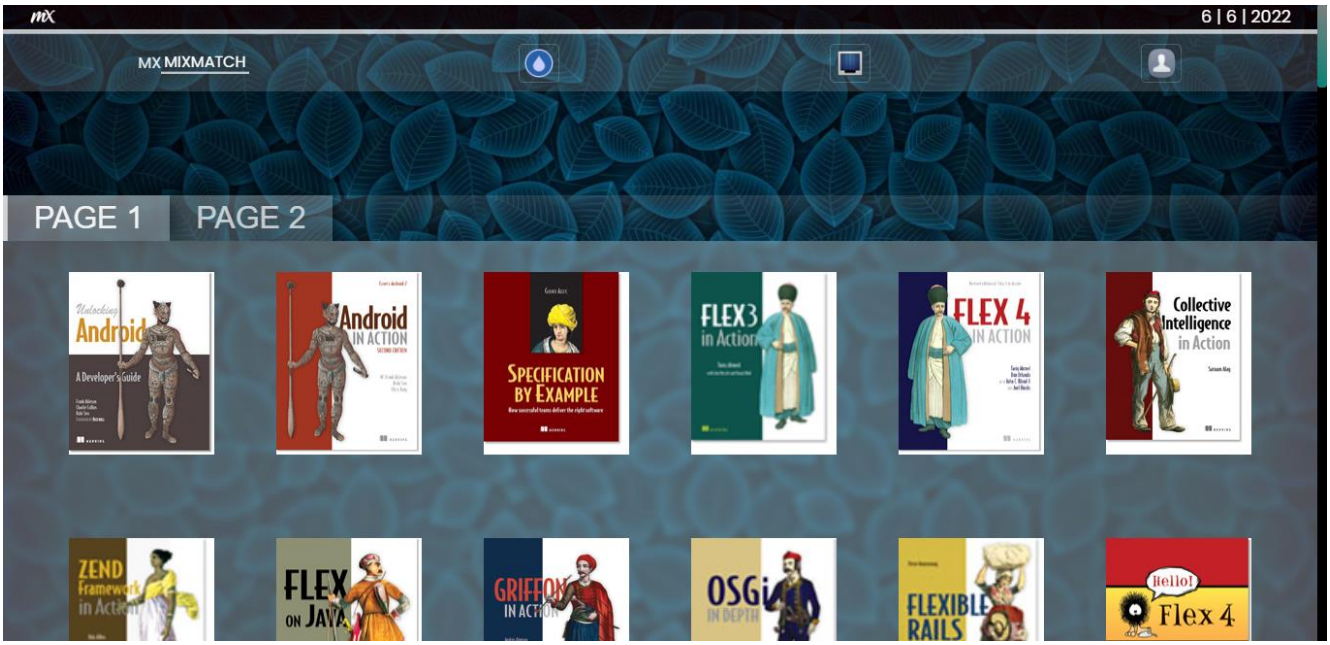
Chose qui nous a dans un premier temps été difficile, si bien qu'on ait pensé à essayer de migrer l'application sur NodeJS en lieu et place de le faire en JEE, pour les raisons suivantes :

- Mongo intégré et très pris en charge dans NodeJS
- Concernant recommandation engines on cherche un équivalent écrit en JS
- Ou à défaut faire ça en différé et actualisé la BD de façon séquentielle avec ce qui marche en java

Mais cela a été réglé en configurant l'environnement du ServerRuntime de la même façon que celui d'exécution des JavaApplications (Voir page 2)

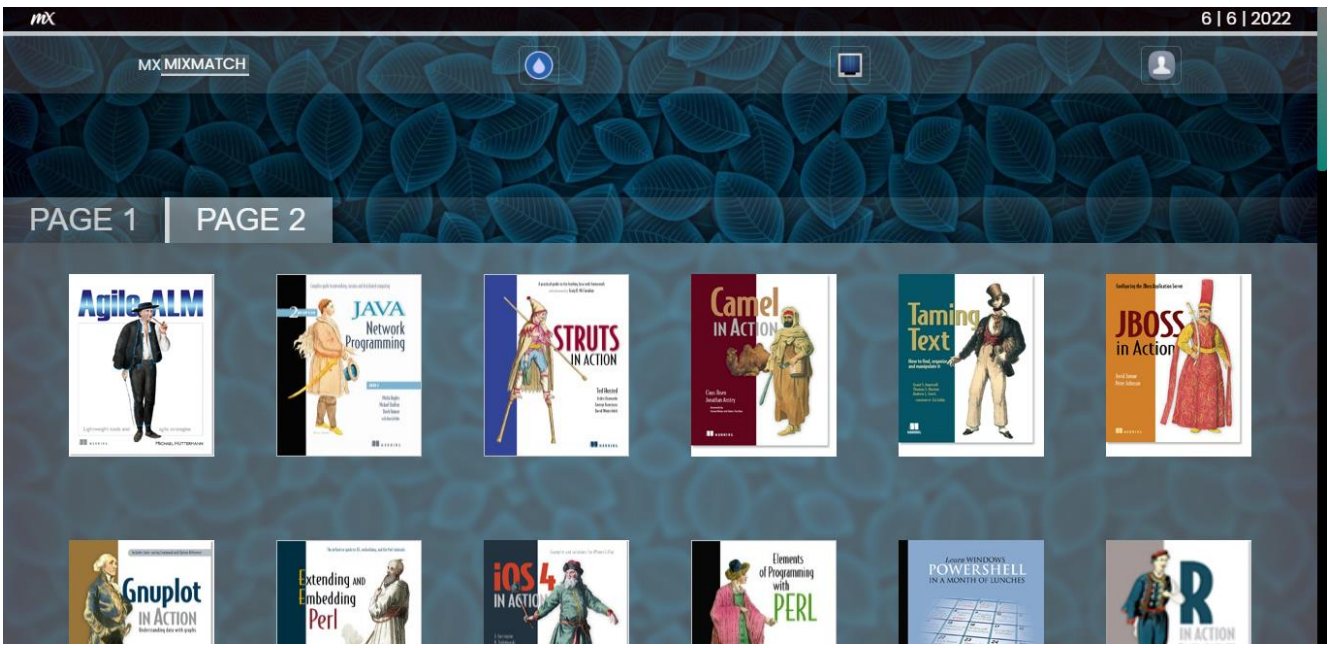
- Aperçu de la page d'accueil après récupération des 200 premières lignes de depuis MongoDB, 100 dans chacune des pages.

  Cette page est gérée par la Servlet MainSevlet



- Procédé d'affichage dans l'accueil :

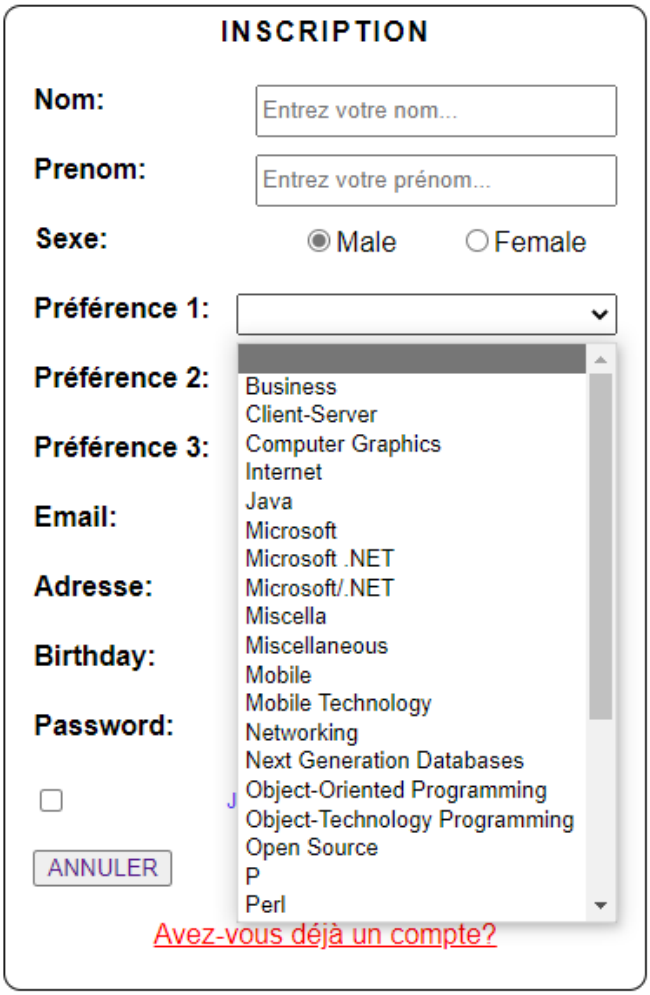
```
<%
ArrayList<String> Img_Liste=(ArrayList)request.getAttribute("Img_List");
int i;
for(i=0;i<=100;i++){
    %>
     <div class="cover">
       <center>
        <img src="<% out.print(Img_Liste.get(i));%>" style="width:150px;height:188px;"alt="">
       </center>
       <br>
     </div>
    <%}
%>
```



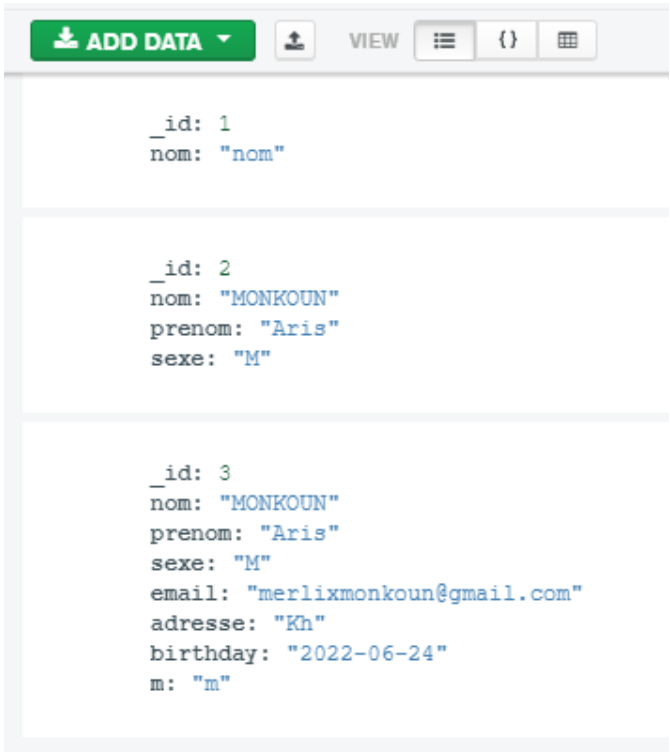
Ensuite, on aimerait obtenir les catégories et les avoir comme préférences lors de l'inscription.

Et pour cela, on a rajouté une nouvelle méthode Distincte dans notre classe MongoDBApp pour avoir les différents valeurs prises par un champ donné.

```java
public ArrayList<String> Distinct(String nom_collection, String nom_champ) {
            //Récupérer la collection
        MongoCollection<Document> collection = mydatabase.getCollection(nom_collection);

        //Création d'une liste pour y stocker les resulats finaux
        List <String> l= new ArrayList<String>();
        try {
             //Appliquer une commande distinct et recupérer les données en Iterable
            DistinctIterable<String> docs = collection.distinct(nom_champ, String.class);

            //Créer un itérateur ou curseur le pour parcourir
            MongoCursor<String> results = docs.iterator();
            while(results.hasNext()) {
             //Ajouter cette case à notre liste de résulatats
                l.add(results.next());
            }
        } catch (MongoException me) {
            System.err.println("An error occurred: " + me);
        }

        return (ArrayList<String>) l;
      }
```



Au bout de quelques essais on arrive à insérer dans la base de données NoSQL



Et ce, grâce au script suivant du doPost depuis la servlet la servlet du Login :

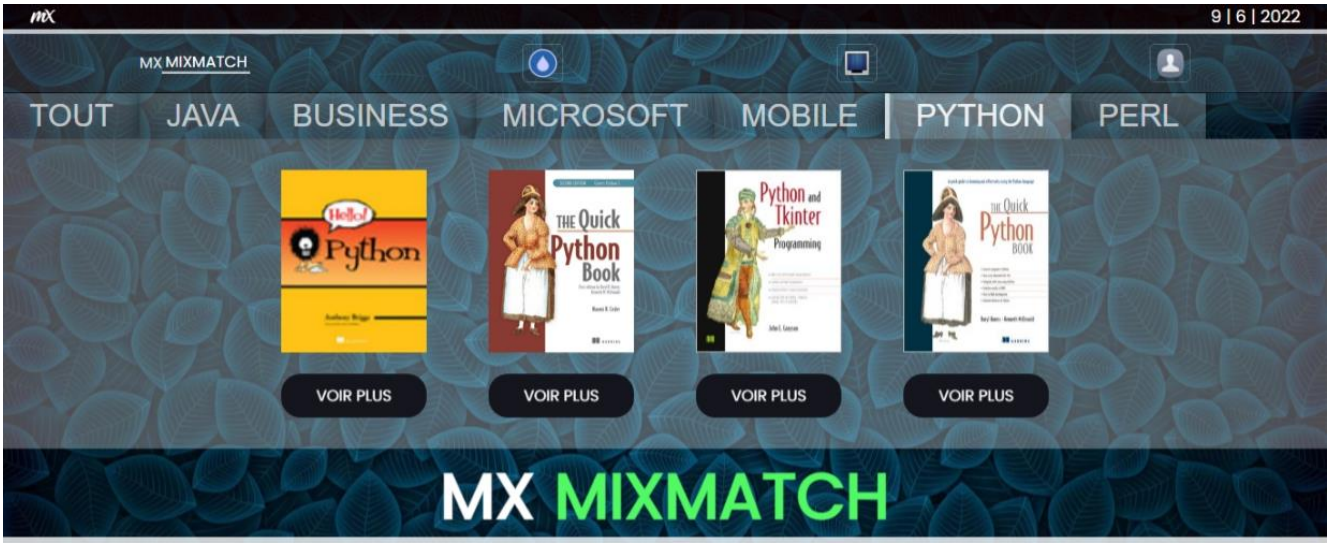
```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

            String nom=request.getParameter("nom");
            String prenom=request.getParameter("prenom");
            String sexe=request.getParameter("sexe");
            //String[] preferences=request.getParameterValues("preferences");
            String email=request.getParameter("email");
            String adresse=request.getParameter("adresse");
            String birthday=request.getParameter("birthday");
            String password=request.getParameter("password");

            MongoDBApp M2=new MongoDBApp("TP1");
            //Récupérer le dernier id-----------------------------------------------------
            ArrayList<String> ids= M2.GetListCollection("users","_id");

            Document person = new Document("_id", ids.size()+1).append("nom", nom).
            append("prenom", prenom).append("sexe", sexe).append("email",email).
            append("adresse", adresse).append("birthday", birthday).
            append(password, password);

            M2.insertDoc(person, "users");

            HttpSession maSession=request.getSession();
            maSession.setAttribute("Utilisateur",person);

            //Pour l'affichage dans la Sevlet-----------------------------------------

            ArrayList<String> Img_List= M2.GetListCollection("books200","thumbnailUrl");

            request.setAttribute("Img_List", Img_List);


     this.getServletContext().getRequestDispatcher("/index.jsp").forward(request,response);
}
```

On a rajouté une dernière méthode à MongoDBApp pour filtrer les requêtes :

```
public ArrayList<Document> FindWithFilter(String collection, String filtre) {

       //Recupérer la collection
       MongoCollection<Document> coll = mydatabase.getCollection(collection);

       //Convertir le filtre en Objet Bson
       BasicDBObject query = BasicDBObject.parse(filtre);

       //Avoir le resultat dans un Iterable
        FindIterable<Document> iterable =coll.find(query);

        //Créer l'itérateur
        MongoCursor<Document> iterator = iterable.iterator();

        //Créer une Liste pour y stocker les enregistrements sous forme documents
        ArrayList<Document> results= new ArrayList<Document>();

        while(iterator.hasNext()) {
          Document doc = iterator.next();
          results.add(doc);
         }

       return results;
       }
```

Cette méthode filter set aussi bien pour la connexion dans la Sevlet SignIn.

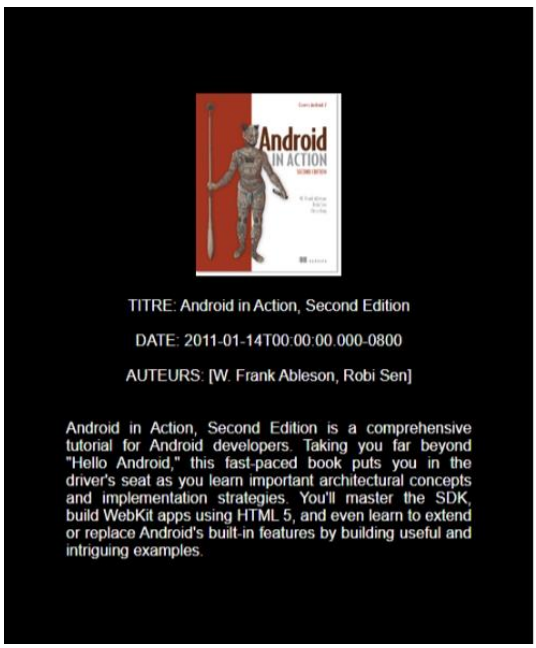
- Application du filtrage par catégorie dans le main :



- Nouveautés sur la page Index :

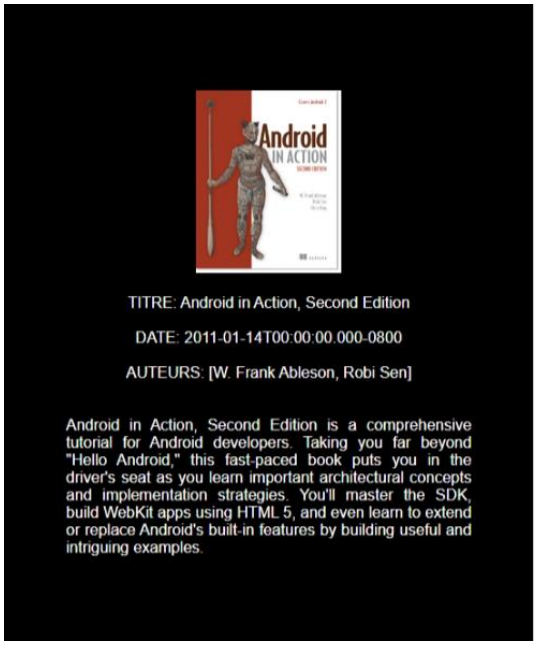
  - Création du bouton VOIR PLUS pour accéder à la page de chaque livre

    Cela a été possible par de fait de matcher _id du livre dans la BD avec id de la balise HTML contenant ce livre



    Là il s'agit de quand l'utilisateur n'est pas connecté.

S'il se connecte il pourra, il pourra alors noter le livre s'il l'a déjà lu.
La note de l'avis variant entre 1 et 5.



5. La recommandation de livres proprement dite :

L'envoi de l'avis dans une nouvelle collection ratings est gérée par la servlet Avis.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

		String note=request.getParameter("note");
		String Id_User=request.getParameter("Id_User");
		String Id_Product=request.getParameter("Id_Product");


		MongoDBApp M2=new MongoDBApp("TP1");
		ArrayList<String> ids= M2.GetFieldInCollection("ratings","_id");


		Document ins = new Document("_id", ids.size()+1).append("Id_User",
		Id_User).append("Id_Product", Id_Product).append("note", note);

		M2.insertDoc(ins, "ratings");

this.getServletContext().getRequestDispatcher("/BooksRecommender").include(request,
response);

	}
```

Pour vérifier l'ajout dans la collection ratings de la base de données MongoDB.

Avec l'entrainement ont une réunit série d'enregistrements dans la collection.

- On a créé un script pour récupérer les données depuis la collection dans un fichier ratings.txt :

```
String id="1";
MongoDBApp M2=new MongoDBApp("TP1");
String query;
query="{}" ;

ArrayList<Document> doc=M2.FindWithFilter("ratings", query);
Document actu_doc;
String a,b,c;

int i;

try {
	File file = new File("dataset/ratings.txt");
	if (!file.exists()) {
		file.createNewFile();
	}
	FileWriter fw = new FileWriter(file.getAbsoluteFile());
	BufferedWriter bw = new BufferedWriter(fw);

	for(i=0;i<doc.size();i++) {
		actu_doc=doc.get(i);
		a=(String) actu_doc.get("Id_User");
		b=(String) actu_doc.get("Id_Product");
		c=(String) actu_doc.get("note");

		bw.write(a+","+b+","+c+"\n");
	}

	bw.close();

	System.out.println("Modification terminée!");

	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

Résultat :

ratings.txt

```
1,00,1
1,01,2
1,02,5
1,03,5
1,04,5

2,00,1
2,01,2
2,05,5
2,06,4
2,02,5

3,01,2
3,02,5
3,03,4
3,04,3

4,00,5
4,01,5
4,02,5
4,03,0
```

Ainsi, à intervalles de temps réguliers, ce fichier sera actualisé dans notre projet.

On y appliquera la recommandation en ayant l'id de l'utilisateur depuis la session en cours.

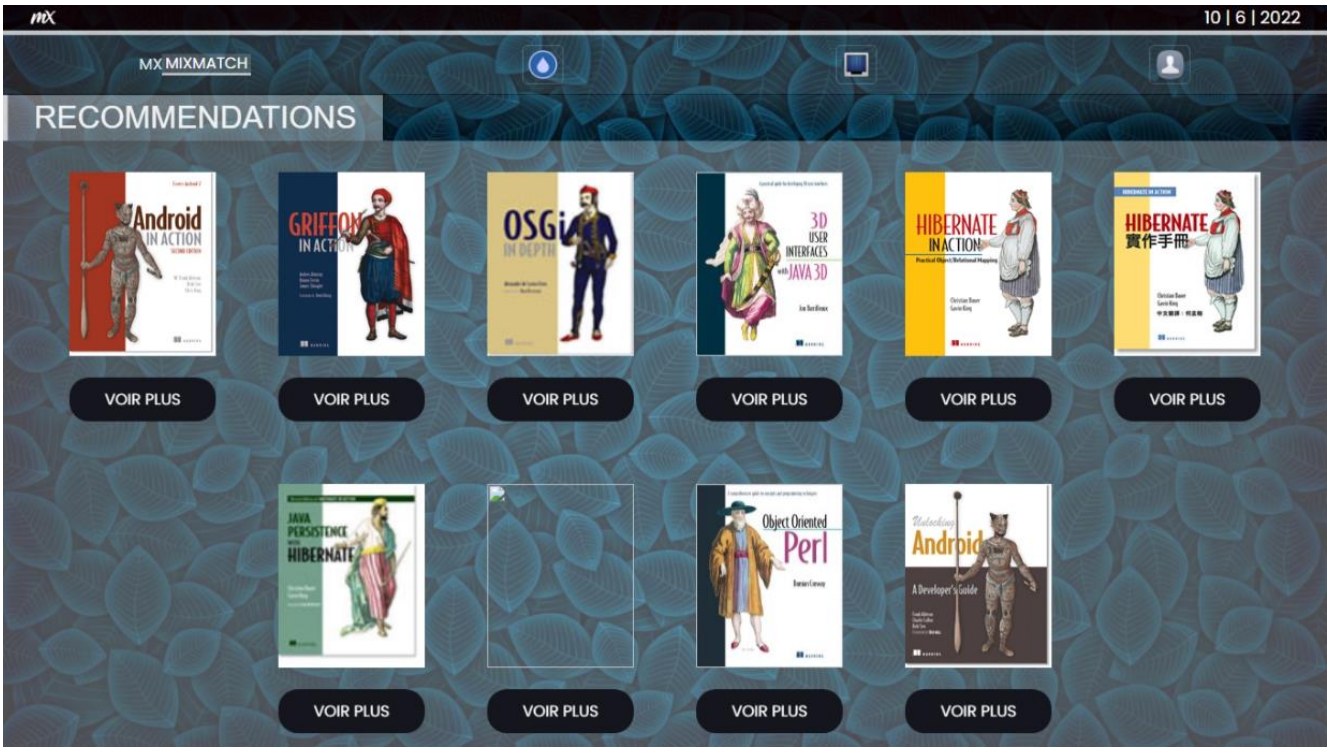
NB : Remarquons de la page PourVous a été paramétrée de sorte à ne l'avoir que quand il y a une session active.

Et pour cela, les livres présentés dans la page PourVous sont :

- A 40% issus de la recommandation Mahout
- A 30% issus de préférences 1
- A 20% issus de préférences 2
- A 10% issus de préférences 3

On veut recommander 10 livres en tout, donc on s'est servi de random() pour sélectionner respectivement 4 - 3 - 2 – 1 éléments dans chaque liste.

- Si bien que si on recharge la page des recommandations, le contenu ne se sera pas identique.



CONCLUSION

Au terme de tout ce qui précède, réaliser ce projet fut une expérience unique, dans le sens où ça nous a permis d'associer les BigData à des technologies auxquelles nous étions déjà familières.

Certes, nous nous sommes heurtés à de nombreuses difficultés, mais elles nous ont permet de cerner en profondeur les sous bassements des différents concepts qui entraient en jeu lors de la mise en œuvre du projet.