Recomanador Multimèdia

Antonio González Hidalgo
Grau en enginyeria Informàtica
Intel·ligència artificial

Consultor: **David Isern Alarcón**
Professor: **Carles Ventura Royo**
Data Inici: 2 de Març de 2017

A tota la gent que s'ha mantingut al meu costat des que vaig començar l'enginyeria fins avui dia. En especial a Sandra Martínez per la seva comprensió i ànims en moments difícils.

FITXA DEL TREBALL FINAL

Títol del treball:	*Recomanador Multimèdial*
Nom de l'autor:	*Antonio González Hidalgo*
Nom del consultor/a:	*David Isern Alarcón*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament (mm/aaaa):	*03/2017*
Titulació o programa:	*Grau en enginyeria informàtica*
Àrea del Treball Final:	*Intel·ligència artificial*
Idioma del treball:	*Català*
Paraules clau	*Intel·ligència artificial, BigData, Recomender, Mahout, Spark, AWS.*

Resum del Treball:

Amb l'elaboració d'aquest TFG volia endinsar-me en el món de la intel·ligència artificial, més concretament en el concepte de Big Data. Al mateix temps, volia obtenir un resultat amb una aplicació pràctica a la vida real.

Tot el món consumeix contingut multimèdia, ja siguin pel·lícules, música, llibres, etc. Per això, unes de les primeres opcions que se'n va passar pel cap va ser, elaborar un Recomanador de pel·lícules per poder descobrir nou material interesant.

No és un producte nou. Actualment ja existeixen moltes llibreries per la implementació de recomanador de productes. Per exemple, Amazon fa una recomanació de productes segons els productes visitats per l'usuari.

L'objectiu del TFG és elaborar una pàgina web on l'usuari obtingui les recomanacions personalitzades al seu criteri de valoració de pel·lícules. Per aconseguir-lo es farà ús de les llibreries, Mahout i Spark. A més la base de dades inicials contindrà les dades del dataset de Movilens.

La realització d'aquest TFG, m'ha permès endinsar-me en el món Big Data. Ha sigut una experiència molt satisfactòria, ja que he adquirit molts coneixements sobre les llibreries Mahout i Spark. A més de començar a conèixer serveis, tan demandats en el món laboral com és Amazon Web Service.

Em plantejo aquest treball com a punt de partida, ja que el producte resultant pot ser utilitzat com a base per elaborar un projecte més complex i afegint nous productes per recomanar: Música, Llibres, etc. (

Abstract:

With the development of the TFG I wanted to immerse myself in the world of artificial intelligence, specifically the concept of Big Data. At the same time, I wanted to get a result with a practical application in real life.

Everyone consume multimedia content, just to name a few, movies, music, books and many others. Therefore, one of the first choices that I went through the top of my mind was to produce a movie recommender to discover interesting new media and fascinating, top of the shelf content.

This is not a new concept neither a newborn idea. There are already many libraries to implement recommended products. As practical example, Amazon does give hints and tips when it comes to recommend products based on viewers discretion an likes.

The main objective of this TFG is to develop website where users get personalized recommendations for their value and score on each movie. To achieve this will be used libraries like, Mahout and Spark. In addition to the data base, it will also be feed by data coming from dataset Movilens.

The development of this TFG, allowed me to immerse myself in the Big Data world. It was a very satisfying experience because I've gained a lot of knowledge about Mahout libraries and Spark. Besides getting to know the services, as demanded in any workplace such as Amazon Web Service.

I consider this to be just the beginning, since the resulting product may get to be used as a very powerful tool to draft a more complex and complete solution to add new products and assets to recommend just to name a few, music, books, etc. (tv shows, apps, computer programs and suites)

Tabla de continguts

Llista de figures

1. Introducció

1.1. Context i justificació del Treball

Des de petit m'han agradat les pel·lícules futuristes de robots amb un comportament similar a l'humà o ordinadors dotats d'una intel·ligència capaços d'interactuar de tu a tu amb els éssers humans.

Aquesta fascinació em va fer agafar amb moltes ganes l'assignatura d'intel·ligència artificial del grau. L'assignatura em va fer veure la superfície de la matèria, que encarà, em dóna la sensació es troba en una fase molt inicial. Per això, penso que especialitzar-me en aquesta rama, em pot donar en el futur opcions a tenir un treball molt interessant.

Tot plegat va fer, que tingues clar que el treball final de grau (a partir d'ara TFG) el volgués desenvolupar sobre aquesta assignatura. El que no tenia gens clar era sobre quin tema específic treballar i desenvolupar el meu TFG.

Actualment un dels perfils més demanats a les companyies són arquitectes amb experiència al món del Big Data. Aquest era un altre tema que em cridava molt l'atenció. Aprofundir més en el concepte Big Data. Per a treballar en el camp de Big Data normalment es fan servir conjunt de dades (datasets) dels quals es vol treure un cert coneixement.

A la xarxa hi ha milers de conjunt de dades. Alguns fins i tot procedents de les administracions públiques [1]. Això dóna un munt de possibilitats o combinacions per a treballar en el camp del Big Data.

En l'actualitat el consum de material multimèdia s'ha vist incrementat en els últims anys. Les connexions d'alta velocitat a les llars, els telèfons intel·ligents i les tauletes han fet que actualment al mercat hi hagi moltes empreses que satisfan la necessitat de continguts multimèdia dels usuaris. Un exemple d'aquestes poden ser Netflix, Wualki.tv, iTunes Store, Spotify, Youtube, etc.

Tothom veu pel·lícules i això porta al fet que alguna vegada ens preguntem: Quina

pel·lícula miro? Per tal de satisfer-la, penso que estudiar el camp dels recomanadors de productes pot ser un bon punt de partida per endinsar-se en el món Big Data.

Sent un apassionat de pel·lícules i música en general, vaig pensar a elaborar una aplicació per recomanar noves pel·lícules i nous àlbums de música. A més, penso que actualment hi ha molta més demanda de contingut multimèdia que fa uns anys i això anirà en augment. Per tant, elabora un sistema Big Data per recollir, estudiar i donar recomanacions útils als usuaris pot ser un bon punt de partida per endinsar-se en el món Big Data.

Actualment les llibreries més conegudes, per implementar un recomanador de productes, amb certa facilitat i èxit, són Apache Mahout [2] i Apache Spark [3]. En aquest TFG, es duran a terme una implementació respectivament amb cada una d'aquestes llibreries. D'aquesta manera, es podrà dur a terme un estudi del rendiment obtingut amb les dues llibreries. Finalment es migrarà a Amazon Web Services [4] la implementació realitzada per testejar el recomanador en un entorn real.

1.2. Objectius del Treball

Els objectius a assolir en aquest TFG són els següents:

- Guanyar experiència i saber com implementar un sistema capaç d'explotar les llibreries actuals de Big Data.
- Adquirir coneixements en els sistemes **Hadoop**, **Mahout**, **Spark**, etc.
- Elaborar una aplicació web on els usuaris puguin aportar la seva puntuació i el sistema li recomani nous continguts. Recomanador de productes multimèdia.
- Crea les bases de dades necessàries.
- Adquirir coneixements en el desenvolupament d'aplicacions web mitjançant **Spring**.
- Desplegar l'aplicació sobre **Amazon Web Services**.
- Realitzar la documentació necessària del TFG.
- Realitzar una presentació del projecte elaborat entenedora i de fàcil comprensió.
- Obtindré un resultat final vàlid i útil per als usuaris.

1.3. Mètode seguit

L'objectiu principal del TFG en essència és elaborar un recomanador de pel·lícules. El funcionament bàsic d'aquest és el següent:

- L'aplicació té un usuari actiu. És l'usuari que actualment està fent ús de l'aplicació.
- Partint de les pel·lícules que l'usuari actiu ja té votades. L'aplicació buscarà usuaris amb uns criteris semblants. Aquests criteris s'obtenen buscant usuaris que hagi votat les mateixes pel·lícules amb una puntuació semblant a l'usuari actiu.
- De la llista de tots els usuaris semblants, s'obté una llista de totes les pel·lícules que han votat, descartant aquelles que també ha votat l'usuari actiu.
- S'ordena la llista de pel·lícules per votació per obtenir les millors pel·lícules, no visualitzades per l'usuari actiu, partint de votacions que han fet altres usuaris amb un criteri semblant.

Amb aquest guió d'execució, podem veure com existeixen diversos punts febles a tractar. Per exemple:

- On obtenim les dades per trobar usuaris amb un criteri semblant?
- Si no trobem cap usuari amb un criteri semblant?
- Si l'usuari actiu no té cap vot?
- Si només volem veure recomanacions d'un cert tipus de pel·lícules?

Al llarg d'aquesta memòria es donarà resposta a aquests i altres punts febles de la implementació del Recomanador. Per començar, a desenvolupar el recomanador, desglossarem el TFG en tres parts.

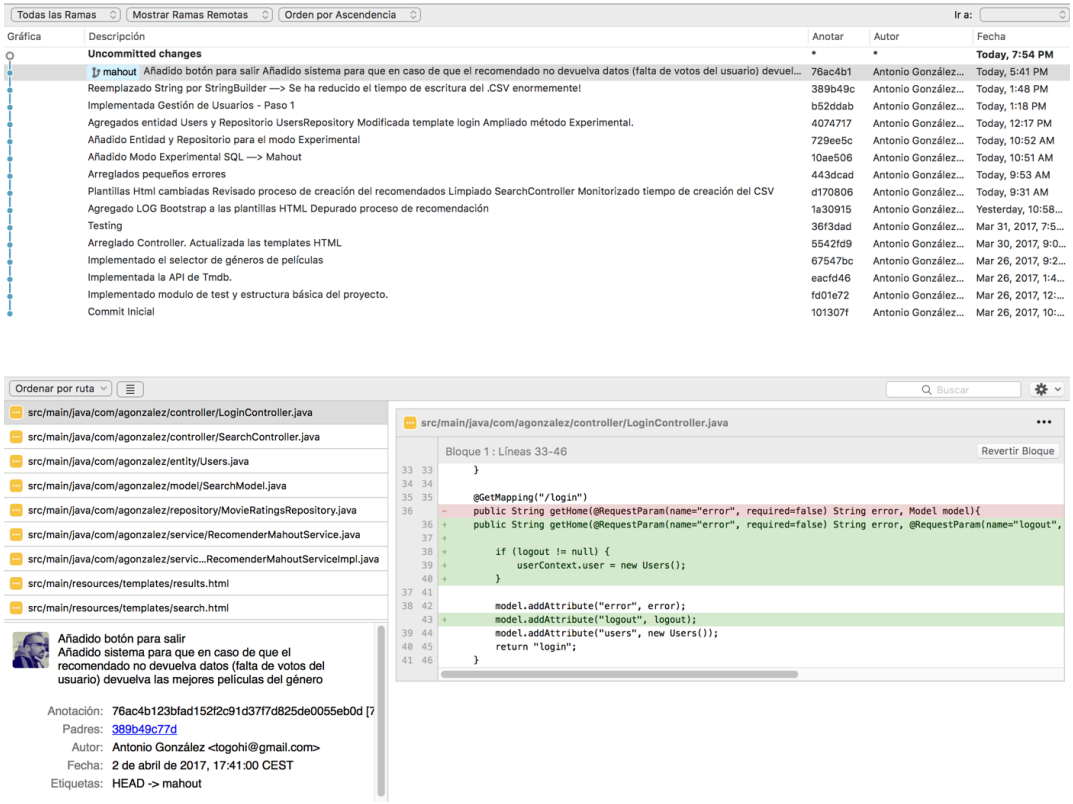
La primera es desenvoluparà tot l'entorn web per poder interactuar amb el recomanador. A més, es farà una primera implementació amb Apache Mahout del recomanador de productes multimèdia. La segona part, aprofitant l'aplicació web i el seu front-end, es durà a terme una segona implementació del recomanador multimèdia. Fent ús de la llibreria Apache Spark. Finalment es desenvoluparà una tercera part, on es migrarà l'aplicació web del recomanador multimèdia a un entorn real com Amazon Web Services.

Per elaborar aquest TFG es farà servir com a equip principal de desenvolupament un equip Apple iMac amb la següent configuració tècnica:

Il·lustració 1: Especificació tècnica del equip de desenvolupament

Una vegada definides les etapes del projecte i l'equip que es farà servir. Definirem la metodologia de treball. Tot el referent al codi font del projecte s'anirà integrant en un repositori GIT localment, per gestionar les diferents versions del codi i monitoritzant els avanços duts a terme.

També es crearà un repositori a GitHut [5]. Així el codi font el tindrem en un repositori a la xarxa com a mesura de seguretat. Idealment hauria d'estar en un repositori privat, però els plans gratuïts de GitHut no donen suport a aquest tipus de repositoris. Cada diumenge es farà un Push al repositori públic a GitHut.
Per últim la metodologia referent a aquesta pròpia memòria serà la següent: cada diumenge ha de quedar actualitzada amb les millores implementades al llarg de la setmana. D'aquesta manera s'evitarà avançar molt en el codi font i deixar la memòria per l'últim moment, amb tots els problemes que això comporta.



Il·lustració 2: Captura de pantalla de SourceTree (Git Local)

1.4. Preparació de l'entorn de treball

Abans de començar el TFG s'adjunta una llista del programari necessari i la seva versió, per començar a desenvolupar el recomanador multimèdia.

- Java 8 SDK 1.8.0_111 [6]
- Eclipse Neon 2 [7]
- Apache Maven Project 3.3.9 [8]
- SourceTree 2.5 [9]
- mySQL Server 5.7.18 [10]
- mySQL Workbench 6.3.9 [11]
- Apache Mahout [12]
- Spring [13]

1.5. Planificació del Treball

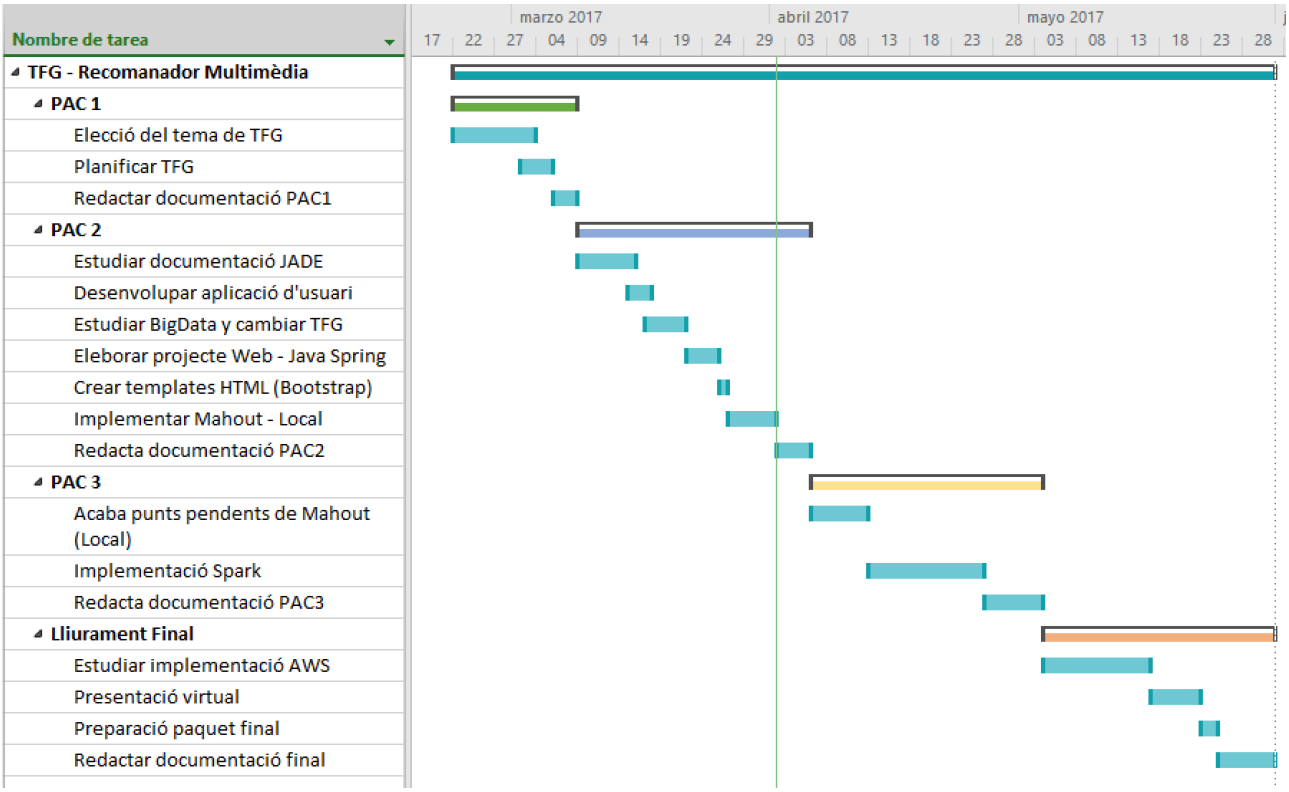
Seguint l'estratègia descrita a l'apartat 1.3. He elaborat una planificació de treball, desglossat en petites tasques. Aquesta planificació segurament anirà canviant al llarg de la vida del projecte.

S'adjunta un detall de les tasques a realitzar desglossades per les diferents PACS i un diagrama de Gantt per veure les mateixes fites de manera més visual.

NOTA: La planificació d'aquest TFG es va veure afectada pel canvi de tema que es va produir durant l'etapa del procés de la PAC2. Així queda reflectit en la planificació adjunta.

Nombre de tarea	Duración	Comienzo	Fin
TFG - Recomanador Multimèdia	**71 días**	**mié 22/02/17**	**mié 31/05/17**
PAC 1	**11 días**	**mié 22/02/17**	**mié 08/03/17**
Elecció del tema de TFG	8 días	mié 22/02/17	vie 03/03/17
Planificar TFG	3 días	jue 02/03/17	dom 05/03/17
Redactar documentació PAC1	3 días	lun 06/03/17	mié 08/03/17
PAC 2	**20 días**	**jue 09/03/17**	**mié 05/04/17**
Estudiar documentació JADE	5 días	jue 09/03/17	mié 15/03/17
Desenvolupar aplicació d'usuari	3 días	mié 15/03/17	vie 17/03/17
Estudiar BigData y cambiar TFG	3 días	vie 17/03/17	mar 21/03/17
Eleborar projecte Web - Java Spring	4 días	mié 22/03/17	sáb 25/03/17
Crear templates HTML (Bootstrap)	1 día	dom 26/03/17	dom 26/03/17
Implementar Mahout - Local	6 días	lun 27/03/17	sáb 01/04/17
Redacta documentació PAC2	4 días	dom 02/04/17	mié 05/04/17
PAC 3	**20 días**	**jue 06/04/17**	**mié 03/05/17**
Acaba punts pendents de Mahout (Local)	5 días	jue 06/04/17	mié 12/04/17
Implementació Spark	10 días	jue 13/04/17	mié 26/04/17
Redacta documentació PAC3	5 días	jue 27/04/17	mié 03/05/17
Lliurament Final	**20 días**	**jue 04/05/17**	**mié 31/05/17**
Estudiar implementació AWS	9 días	jue 04/05/17	mar 16/05/17
Presentació virtual	4 días	mié 17/05/17	lun 22/05/17
Preparació paquet final	2 días	mar 23/05/17	mié 24/05/17
Redactar documentació final	5 días	jue 25/05/17	mié 31/05/17

Il·lustració 3: Llista de tasques a elaborar



Il·lustració 4: Diagrama de Gantt

1.6. Breu sumari de productes obtinguts

Amb la finalització del segon període PAC2. S'ha obtingut un recomanador amb les següents capacitats:

- Aplicació web funcional.
    - Pendent crear nous usuaris
    - Pendent enregistrar nous vots dels usuaris.
- Recomanador implementat amb Mahout.

Amb la finalització del tercer període (PAC3). S'ha obtingut un recomanador amb les següents capacitats:

- Aplicació web
    - Implementat registre d'usuaris.
    - Implementat mecanisme per enregistrar nous vots.
    - Implementat consulta per veure els vots actuals.
    - Unificat l'idioma del front-end a l'anglès.
- Recomanador implementat amb Spark.

Amb la finalització de la quarta, i última entrega. S'ha obtingut una adaptació del producte per tal de ser desplegada a Amazon Web Service, a més d'aquesta memòria finalitzada i per completar el TFG, el vídeo de la presentació virtual.

1.7. Breu descripció dels altres capítols de la memòria

A mode introductori es fa una breu descripció dels següents capítols que constitueixen aquest TFG:

Conjunt de dades inicials.
Secció on s'explica el conjunt de dades utilitzat, que formen la base del coneixement, per poder recomanar qualsevol producte, en el cas d'aquest recomanador pel·lícules.

Apache Mahout.
Secció on s'explica la llibreria Apache Mahout, i com es pot fer servir per elaborar un recomanador.

Apache Spark.
Secció on explica la llibreria Apache Spark i com es pot fer servir per elaborar un recomanador.

Amazon Web Services
Secció on explica les adaptacions necessàries per dur el recomanador a Amazon web Service.

Punts per continuar treballant
Dona un repàs als punts que han quedat pendents de realitzar per falta de temps o recursos. Així com possibles millores que es poden implementar en el recomanador.

Conclusions.
Explicació de les meves conclusions personals després de l'elaboració d'aquest TFG.

2. Conjunt de dades inicials

Per dur a terme qualsevol implementació d'un sistema recomanador. El sistema necessitarà un volum de dades, prèvies, per poder tindre una base de coneixement. A mesura que el volum de dades sigui més gran, es podrà extreure un millor coneixement. I Per tant, poder fer millors recomanacions.

En el cas d'aquest TFG, són necessàries unes dades capaces de donar a l'aplicació la suficient informació per poder recomanar pel·lícules d'interès per a l'usuari. Com s'ha explicat a la secció 1.3. El recomanador buscarà usuaris amb criteris de votació semblants al nostre. D'aquests usuaris, mirarà les pel·lícules votades i seleccionarà les que tinguin millor nota i no hagin estat votades per nosaltres.

Per poder realitzar això, es necessita una base de dades d'usuaris, una base de dades de pel·lícules i una base de dades on es relacionin els usuaris amb les pel·lícules que ja han votat.

Entre la gran quantitat de datasets existents, vaig trobar alguns de bastants interessant. Per exemple, dataset de Last.fm [14], per si volem fer un recomanador de música. En l'àmbit de la música, també tenim el dataset de l'antic AudioScrobber [15]. Per dur a terme l'objectiu de recomanar pel·lícules, existeix una sèrie de datasets molt complets. Un d'ells és el conegut Movilens [16] de l'empresa Grouplens.

Aquest dataset, compta amb les dades necessàries per poder muntar una base de dades relacional i migrar el seu contingut. Movilens ens dóna els següents fitxers:

- Movies.csv, es tracta d'un llistat de pel·lícules amb el seu ID, el títol i una llista de generes separats pel caràcter '|'.
- Links.csv, es tracta d'un llistat on es relaciona el ID de les pel·lícules dintre del dataset de Movielens i altres serveis webs com Imdb o Tmdb.
- Ratings.csv, es tracta d'un llistat on es relaciona els ID de les pel·lícules amb el ID dels usuaris que les han votat. Així com la nota donada en un rang de l'1 al 5.

Movilens ofereix diversos formats del seu dataset, amb més o menys volum de dades. Pel desenvolupament d'aquest TFG es farà sevir la versió Lite i la completa. A continuació s'adjunta un resum del volum de dades contingut en cadascun dels formats:

Fitxer	Versió Lite	Versió Complerta
Movies.csv	9.126	40.111
Links.csv	9.126	40.111
Ratings.csv	100.005	24.404.096

Il·lustració 5: Resum del dataset MovieLens

Es farà una comparació de rendiment entre les dues versions en totes les implementacions que es duran a terme al llarg d'aquest TFG. El mode d'utilització serà el següent: Pel desenvolupament faré servir la versió Lite, per optimitzar el temps de proves i guanyar agilitat. Encara que també és dura a terme una prova final amb el dataset complet per poder elaborar la comparativa de rendiments.

3. Apache Mahout

3.1. Què és Apache Mahout?

Il·lustració 6: Logotip de Apache Mahout

Apache Mahout [2] es tracta d'un projecte per implementar de manera gratuïta sistemes d'auto aprenentatge. Aquests poden ser utilitzats de manera local o de manera distribuïda fent servir el paradigma "MapReduce". Normalment quan s'implementa de manera distribuïda es fa servir la llibreria Apache Hadoop per gestionar els equips que formen el sistema.

Apache Mahout forma part de la fundació Apache Software [17] i es distribueix sota la llicencia 2.0 de Apache [18]

3.2. Utilització de Apache Mahout

Tal com la mateixa web de Apache Mahout reflexa. Mahout pot ser-hi principalment per realitzar tres grans funcions:

- Proporciona un entorn de programació simple i extensible per desenvolupar algorismes escalables.
- Oferir una gran varietat de codi pre-dissenyat per Scala, Apache Spark, H2O, Apache Flink.
- Oferir un entorn experimental, Samsara, que és un llenguatge similar a R.

Sense perdre de vista l'objectiu d'aquest TFG, ens centrarem en el primer punt. Crear un entorn de programació simple i extensible per desenvolupar el nostre recomanador. Per arribar al nostre objectiu hem de crear un sistema que llegeixi els fitxers .CSV i recomani productes basant-se en els usuaris. Per fer això, farem servir el coeficient de correlació de Pearson [19].

Aquest coeficient serveix per determinar el grau de semblança de dues variables donades. Els seus valors oscil·len entre [-1, 1]. Els valors pròxims a 1, vol dir que existeix una relació lineal entre les variables. Totes dues creixen de manera proporcional. Si el valor és pròxim a -1, també indica una relació lineal, però en aquest cas, l'increment d'una fa decréixer de manera proporcional la segona. Per valors propers o 0, indica que no existeix una relació lineal entre les variables.

Per implementar aquest coeficient de manera senzilla amb Mahout farem servir la classe, *PearsonCorrelationSimilarity* [20] A més, com volem construir un recomanador basat en la similitud entre usuaris farem servir la interfície *UserSimilarity* [21].

El recomanador necessitarà un llindar per establir quins usuaris formen part del grup usuaris semblants o no. Per aquest TFG s'acceptaran com a usuaris semblants tots aquells que en seu coeficient de Pearson sigui més gran que 0.5. Establir un valor més gran por porta problemes, ja que potser no es troben usuaris semblants i el recomanador és incapaç de fer recomanacions. Per tant seria contraproduent.

Mahout ja implementa les classes necessàries per crear un recomanador com el que es planteja, tenim la classe *AbstractRecommender* [22] on podem veure tota mena de recomanador que es poden implementar. Per aquest TFG es farà ús de la classe *GenericUserBasedRecommender* [23]

3.3. Problemes de Rendiment.

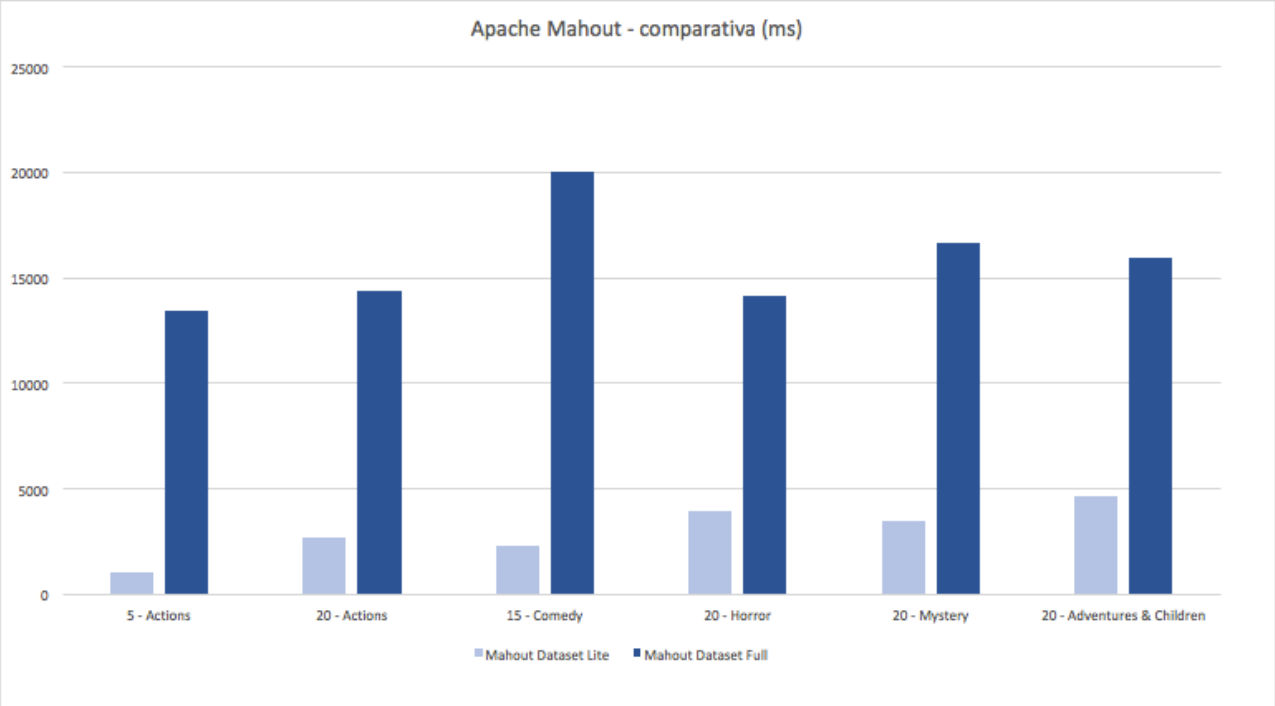
Una de les primeres mesures que s'ha tingut en compte. Per tal d'optimitzar les consultes a les taules de la base de dades, és crear-les amb uns índexs a les columnes, que es faran servir per realitzar qualsevol cerca. Així, la taula movieRatings, té un índex per cadascuna de les columnes userId i movieId.

Malauradament, encara de tenir índexs a les taules. El temps de resposta del cercador amb aquesta última implementació és bastant deficient. Superant en alguns casos els cinc minuts per presentar els resultats. Un temps totalment inacceptable en un recomanador d'avui dia. Per aquest motiu, aquests inicialitzadors del model de dades, s'ha mantingut al

codi, com 'EXPERIMENTAL' i s'ha tornat al model de l'esquema dos, que ofereix un temps de resposta bastant acceptable. Tan sols uns mil·lisegons.

3.4. Comparativa de rendiment

A continuació s'adjunta un gràfic on es pot veure, els temps en mil·lisegons, que triga a presentar els resultats el recomanador, fent ús dels dos conjunts de datasets empleats en el TFG.



Il·lustració 7: Comparativa de temps (en mil·lisegons) dels diferents datasets en la implementació Apacha Mahout

Es pot veure que el temps amb el conjunt de dades Lite és molt més ràpid que no pas el conjunt de dades complet. La mitja dels resultats són els següents:

Apache Mahout amb dataset Lite	3032 ms
Apache Mahout amb dataset full	15761 ms

Il·lustració 8: Mitjana de temps d'execució per Apache Mahout.

Observant aquests resultats, podem arribar a la conclusió que amb una configuració local es pot tindre un entorn de proves amb uns temps acceptables. Malauradament, amb un entorn real, proper al conjunt de dades complet, tindre una configuració local ofereix uns resultats bastants pobres.

4. Apache Spark

4.1. Què és Apache Spark?

Il·lustració 9: Logotip de Apache Spark

Apache Spark [24] és una llibreria per analitzar dades, especialment enfocat al processament de dades Big Data. Es podria considerar el successor de Hadoop i Mahout. Ja que una de les principals diferencies que aporta Spark és la seva rapidesa. Spark és una llibreria que manté a memòria la distribució de la feina, aquesta mena de treballar és molt més rapida que les operacions batch que realitza Hadoop amb MapReduce.

A més, Spark té integrat diferents mòduls que el converteixen en una solució millor per nous projectes de Big Data. Entre aquests mòduls es troben, MLlib per crear sistemes d'auto-aprenentatge, GraphX per càlculs amb gràfics a més d'una integració amb el llenguatge R, Spark R.

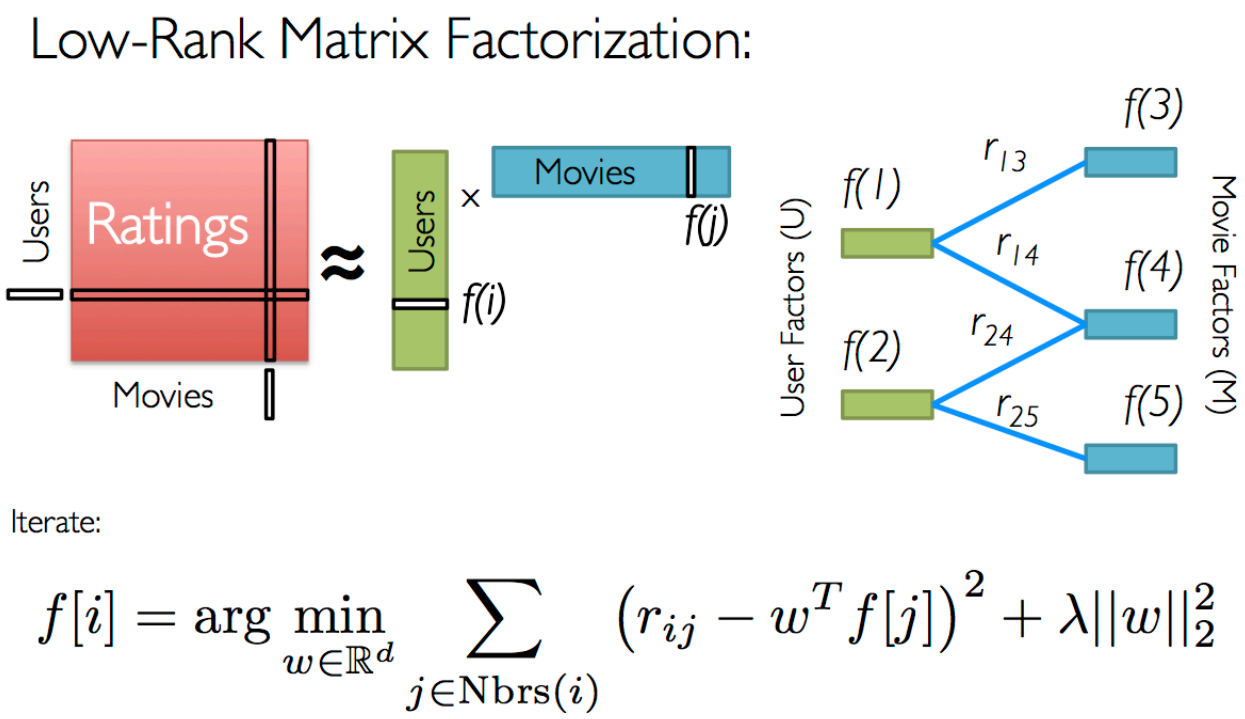
També ofereix compatibilitat amb els projectes existents, ja que pot conviure perfectament amb Apache Hadoop. Tots aquests avantatges, fan pensar que el món de les solucions Big Data, cada vegada més, s'anirà decantant per optar a fer-ne ús de Apache Spark. Per tant, adaptar el recomanador multimèdia a aquesta llibreria és una tasca primordial i molt interesant.

4.2. Utilització de Apache Spark

L'objectiu d'aquesta implementació és la mateixa que al punt anterior. Crear un recomanador de productes, més concretament de pel·lícules. Spark té una sèrie d'eines de filtratge col·laboratiu [25] i aquestes són usades per crear els sistemes recomanadors.

A diferencia de Mahout, Spark no treballa amb el coeficient de Pearson, sinó que treballa, amb un algorisme dels mínims quadrats (ALS). En el nostre cas, creant una matriu d'usuaris i pel·lícules i donat uns factors intenta trobar la relació mínima, fent ús de ALS, entre les

dues variables per predir el següent valor. La següent imatge mostrà el funcionament de l'algorisme ALS:



Il·lustració 10: Imatge extreta de https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

Per fer-ne ús d'aquest algorisme Spark té la classe *ALS* [26] per fer-ne ús s'han d'establir uns valors a les següents propietats, de no fer-ho s'agafen els valors per defecte:

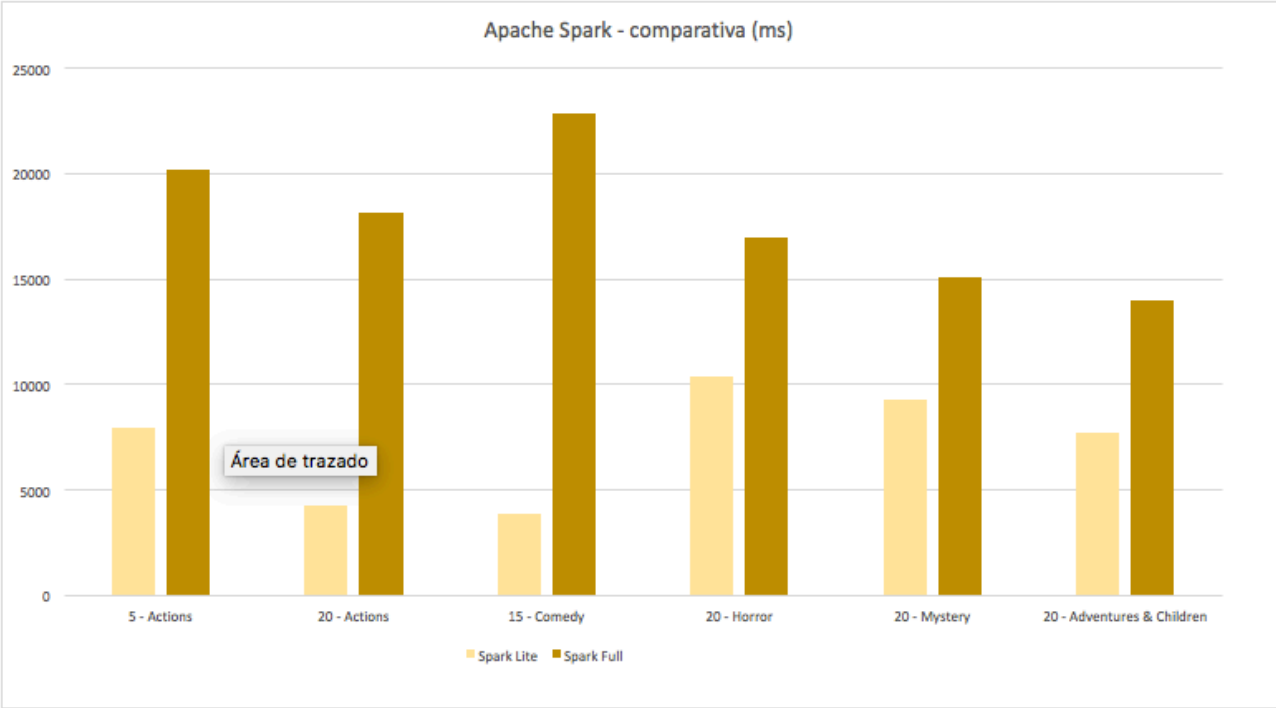
- *numBlocks,* és el nombre de blocs utilitzats per realitzat els càlculs en paral·lel. Per defecte té el valor de -1 per realitzar una autoconfiguració.
- rank, és el nombre de factors al model.
- iterations, és el nombre d'iteracions que realitzarà ALS. Normalment ALS convergeix amb valors aproximats a 20.
- lambda, especifica el paràmetre de regulació. No el farem servir en aquest TFG.
- implicitPrefs, especifica si s'ha d'usar la variant ALS de retroalimentació. No el farem servir en aquest TFG.
- alpha, és el paràmetre que s'aplica a la variant de retroalimentació. No el farem servir en aquest TFG.

Els resultats de l'execució de l'algorisme ALS és una matriu de factorització. Aquesta és representa mitjançant la classe *MatrixFactorizationModel* [27], que té un mètode,

recommendProducts, per retorna recomanacions de productes donant un userID i el nombre de recomanacions a rebre.

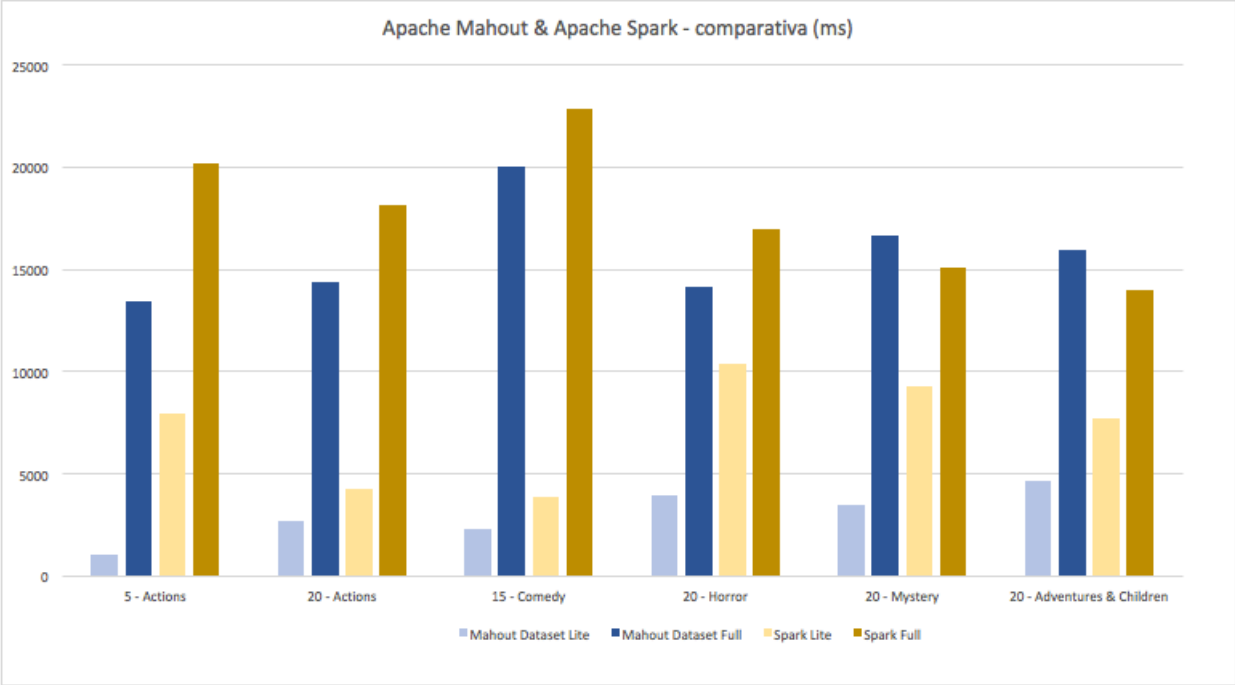
4.3. Comparativa de Rendiment

Tal com passava amb la implementació de Mahout, amb Spark, el rendiment ofert pel dataset Lite és molt millor que l'ofert amb el dataset complet de Movielens. Per tant, les conclusions d'aquesta implementació són les mateixes. Tant a Mahout com a Spark, el rendiment, en entorns local és directament proporcional a la mida del volum de dades que hagi d'analitzar. Queda pendent d'estudi analitzar el rendiment d'aquestes dues distribucions en un entorn distribuït.



Il·lustració 11: Comparativa de temps (en mil•lisegons) dels diferents datasets en la implementació Apacha Spark

En la il·lustració 27 es pot veure els gràfics de totes dues implementacions. Comparant el rendiment de Mahout i Spark, podem veure com amb la implementació de Mahout obtenim un rendiment molt millor en el dataset lite. Aquesta diferència es pot veure com es va reduint a mesura que el volum de les dades va augmentar. Això em fa arribar a la conclusió de què en entorns reals, Spark oferirà un millor rendiment.



Il·lustració 12: Comparativa de temps (en mil•lisegons) de les diferents implementacions i datasets.

5. Amazon Web Service

Amazon Web Service [28] (en endavant AWS) en l'actualitat és una solució de serveis webs que fan servir moltes empreses. Moltes empreses, molt populars, fan servir serveis propis d'Amazon per donar satisfer als usuaris o clients. Un exemple pot ser Dropbox, on la seva capacitat d'emmagatzematge i és al servei S3 [29] d'Amazon.

Per aquest motiu, cada vegada més, les empreses demanen més experiència en la gestió dels diferents recursos d'Amazon als seus futurs empleats. Si fent una cerca en Infojobs o en algun altre portal de cerca de treball, ens podem adonar que aquesta posició cada vegada més és més demandada.

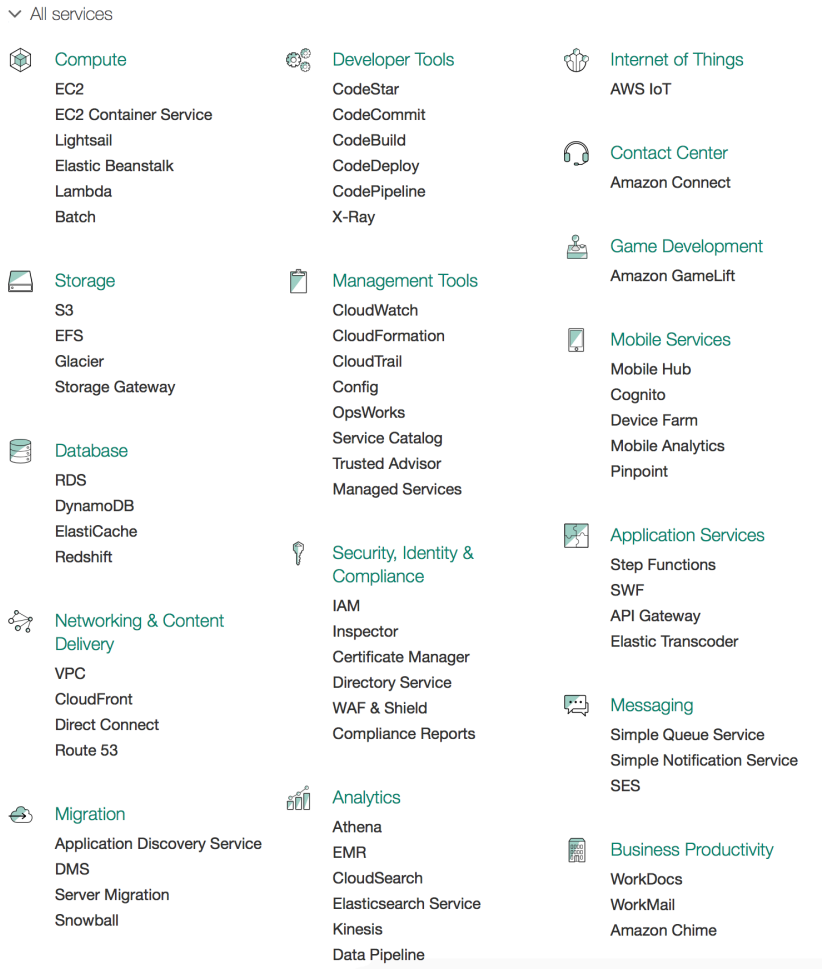
Per aquests motius, s'ha escollit fer una introducció en el món de AWS mitjançant l'adaptació d'aquest recomanador i creant una versió que sigui capaç d'executar-se als servidors d'Amazon.

De tots els serveis oferts per Amazon, els que podien ser de més utilitat per la implementació d'aquest TFG eren:

- **Amazon EC2** [30], es tracta d'un servei per crear servidors virtuals al núvol. Aquest poden ser de diferents configuracions, tant de software com de hardware. Depenent del servidor escollit el preu per hora de computació té un preu o un altre.

- **Elastik Beanstalk** [31], es tracta d'un servei per muntar servidors preconfigurats, per facilitar la feina i poder desplegar aplicacions web en Java, .Net, Php, Python, etc. La gràcia és que seleccionant una preconfiguració muntà un servidor d'Amazon EC2 amb les aplicacions ja instal·lades, per exemple un Tomcat per crear-hi un servidor web, etc.

- **RDS** [32], es traca d'un servei per poder crear bases de dades relacionals al núvol. Aquestes bases de dades poder ser de fins a sis motors diferents: Amazon Aurora, MySQL, PostgreSQL, MariaDB, Oracle i Microsoft SQL Server.



Il·lustració 13: Panel principal de AWS.

Veient les tres opcions anteriors, sembla evident que la millor opció, i menys complicada és decantar-se per:

1. Migrar la base de dades local, que s'ha fet servir a les implementacions de Mahout i Spark, cap a una base de dades al núvol d'**Amazon RDS**.

2. Crear un servidor **Elastik Beanstalk** i pujar la nostra aplicació que es troba en un paquet de tipus .WAR.

Malauradament, aquesta configuració, no va donar resultats i em va ser impossible poder fer que **Elastik Beanstalk** arranques l'aplicació WEB. A més que els temps d'accés a la base de dades eren massa lents.

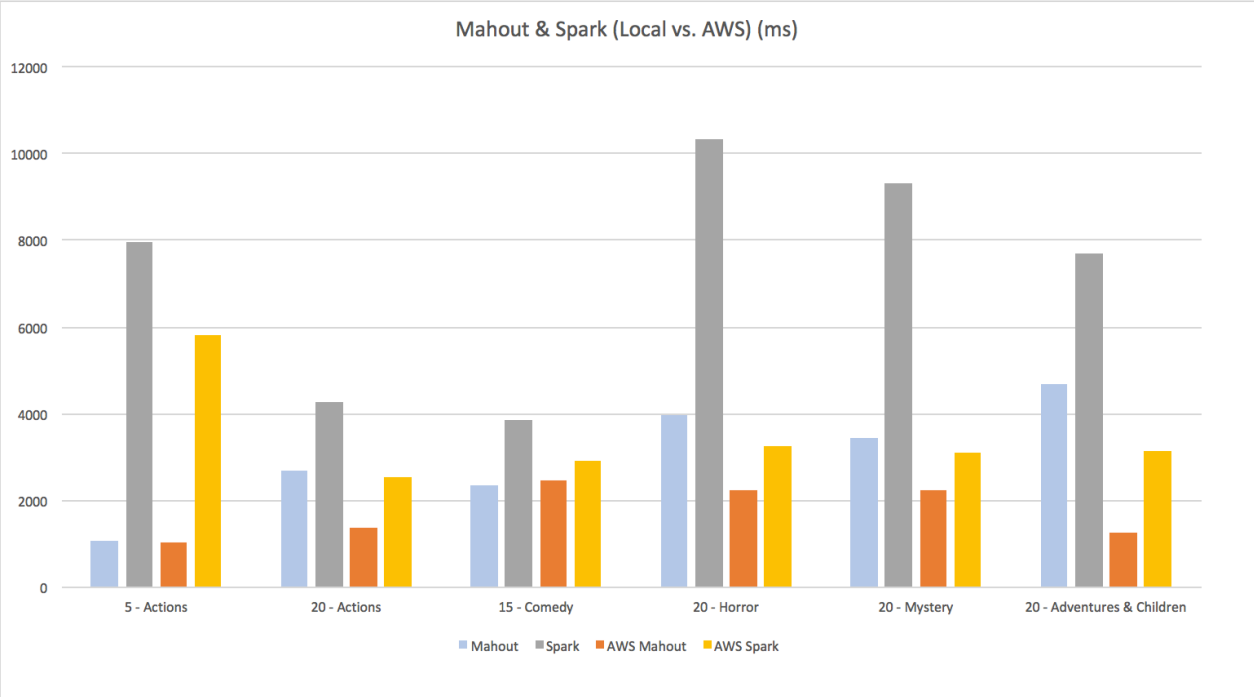
Aquest inconvenients va fer que s'implementés l'aplicació web fent ús del servei EC2, com es pot veure amb més detall a l'annex 9.5.

5.1. Comparativa de rendiments

Com també s'ha fet en les anteriors implementacions, a l'aplicació desplegada a AWS per mitjà de EC2. S'han executat un conjunt de cerques per comparar-hi el temps de resposta del recomanador en diferents escenaris de cerca.

Com es pot veure en el gràfic adjunt, el temps de resposta a AWS és molt millor que executant el recomanador en local. Es pot apreciar com la implementació de Spark és la que treu un millor rendiment dels serveis de AWS. Encarà que la implementació amb Mahout també millorà el temps de resposta no assoleix el marge obtingut per Spark.

També és cert, que encara que Spark ha millorat molt el seu temps de resposta. Mahout continuar oferint els resultats molt més ràpids que Spark. Cal dir, que per temes d'optimització d'espai a AWS, només s'ha migrat les dades corresponent al dataset lite. Queda pendent fer un estudi amb un volum de dades més proper a un entorn de producció.



Il·lustració 14: Comparativa de temps, en mil·lisegons entre execució local o Aws.

6. Conclusions

La realització d'aquest TFG m'ha deixat dos tipus de conclusions que m'agradaria compartir amb els lectors d'aquest TFG.

6.1. Conclusions personals

Durant el desenvolupament d'aquest TFG he après sobre les llibreries treballades. Això, era un dels objectius personals que em vaig imposar al començament. Trobo que encarà que no sóc un expert, he guanyat experiència amb les llibreries de Mahout i Spark. Aquest TFG també m'ha permès assolir i tenir una primera toma de contacte amb els serveis d'Amazon. També he après molt sobre el llenguatge Java. El meu punt fort és .NET i notava que en Java, em falta molt camí per recórrer, ara aquest camí és una mica més curt.

A escala de desenvolupament, on he tingut més problemes ha sigut en arrancar un programa web, ja que mai havia fet un. Això implicava un altre objectiu que no hi era materialitzat en aquesta memòria i era conèixer la llibreria de Spring. Ja que aquest Recomanador ha estat la meva primera toma de contacte amb Spring.

Mahout i Spark són dues llibreries molt populars i tenen una gran quantitat de documentació a la xarxa. Per tant fent una bona feina de recerca i llegint, sobretot llegint molt. Ha estat “fàcil” crear un recomanador de productes. Un altre dificultat, que s’ha anat treballant al llarg del semestre, ha estat adaptar aquests recomanadors genèrics de productes per convertir-los a un recomanador de pel·lícules.

En l’àmbit personal estic molt satisfet no tant sol del producte obtingut, sinó de l’experiència tant en la gestió del projecte, desenvolupament fent ús de diverses llibreries, tecnologies, capacitat de redactar documentació i capacitat en la presentació d’un treball.

6.2. Conclusions de treball

En la meva opinió s’ha obtingut un producte funcional i que compleix els principals punts establerts en la primera entrega. Al primer període vaig afitar molt bé els temps de desenvolupament de cada fase, així com les tasques a realitzar. Això juntament, amb treball continuo durant tot el semestre, ha permès assolir tots els objectius definits al començament de curs.

Hi ha hagut petites modificacions sobre el pla inicial. Sobretot modificacions que afecten el desenvolupament. Per exemple, el desplegament a Amazon Web Service el volia realitzar sobre un servei de Beanstalk i una base de dades RDD. Malauradament va ser impossible i vaig haver de pivotar sobre la maxar i adaptar el recomanador per funcionar amb EC2.

Hi ha aparegut petites desviacions com per exemple, la implementació a Amazon em va fer implementar una funció, que en cas de no ser-hi present el fitxer Settings.json, el crea amb valors predeterminants.

Afortunadament i gràcies a una bona planificació i estimació del temps, no s’han trobat problemes addicionals derivats de la manca de temps o problemes d’última hora.

6.3. Línies de treball futur

A continuació es detallen els punts en els quals es pot continuar treballant per continuar millorant el recomanador o punts pendent d'avaluar per falta de recursos o temps.

- Poder dur un conjunt de dades real a AWS per poder provar el recomanador amb un conjunt real de dades. Aquest punt no s'ha realitzat perquè la implementació de AWS s'ha realitzat fent ús d'un compte gratuït de EC2 i per tant el servidor creat es troba molt limitat tant a capacitat de càlcul com d'emmagatzematge.

- Gestió dels usuaris / seguretat. La gestió dels usuaris de la pàgina web em va donar molts problemes en temes de seguretat. Vaig llegir molta documentació de Spring Security [33], vaig veure vídeo-tutorials de la xarxa. Malauradament no vaig ser capaç de crear una implementació i notava que m'estava desviant de l'objectiu del TFG. És cert que la seguretat dels usuaris és un tema prioritari i que s'hauria de revisar en el recomanador.

- Altre punt pendent, seria desplegar el servei directament sobre un servei Beanstalk d'Amazon enlloc d'un servidor EC2.

- Ampliar els productes a recomanar. Es podria aprofitar el motor creat per adaptar-lo i que fos capaç de recomanar música, llibres, etc.

- Revisar el funcionament i consistència del servei de l'API Tmdb. En el dataset de Movielens existeix un fitxer per relacionar el Ids de les pel·lícules amb serveis exteriors. Per aquest TFG, s'utilitza aquesta relació per trobar el ID de la pel·lícula al servei Tmdb i així poder fer servir la seva API. Bé, existeixen Ids al dataset de Movielens, que han caducat i que provoquen un error al ser cridats mitjançant l'API. S'han netejat tots els que s'han trobat, però encarà poden romandre algun ID caducat.

7. Bibliografia

Context i justificació del treball

[1] http://dadesobertes.gencat.cat/es Març 2017

[2] http://mahout.apache.org/ Abril 2017

[3] http://spark.apache.org/ Abril 2017

[4] https://aws.amazon.com/es/ Abril 2017

Enfocament i mètode seguit

[5] https://github.com/ToGohi/RecomanadorMultim-dia Abril 2017

Preparació de l'entorn de treball

[6] http://www.oracle.com/technetwork/java/javase/overview/index.html Abril 2017

[7] https://eclipse.org Abril 2017

[8] https://maven.apache.org Abril 2017

[9] https://www.sourcetreeapp.com Abril 2017

[10] https://dev.mysql.com/downloads/mysql/ Abril 2017

[11] https://www.mysql.com/products/workbench/ Abril 2017

[12] http://mahout.apache.org/ Abril 2017

[13] https://spring.io Abril 2017

Conjunt de dades inicials

[14] https://labrosa.ee.columbia.edu/millionsong/lastfm Abril 2017

[15] https://datahub.io/dataset/audioscrobbler Abril 2017

[16] https://grouplens.org/datasets/movielens/ Abril 2017

Apache Mahout

[17] https://www.apache.org/ Abril 2017

[18] https://www.apache.org/licenses Abril 2017

[19] http://www.wikiwand.com/en/Pearson_correlation_coefficient Maig 2017

[20] https://archive.cloudera.com/cdh4/cdh/4/mahout-0.7-cdh4.5.0/mahout-core/org/apache/mahout/cf/taste/impl/similarity/PearsonCorrelationSimilarity.html Maig 2017

[21] https://archive.cloudera.com/cdh4/cdh/4/mahout-0.7-cdh4.5.0/mahout-core/org/apache/mahout/cf/taste/similarity/UserSimilarity.html Maig 2017

[22] http://archive-primary.cloudera.com/cdh4/cdh/4/mahout-0.7-cdh4.3.2/mahout-core/org/apache/mahout/cf/taste/impl/recommender/AbstractRecommender.html Maig 2017

[23] http://archive-primary.cloudera.com/cdh4/cdh/4/mahout-0.7-cdh4.3.2/mahout-core/org/apache/mahout/cf/taste/impl/recommender/GenericUserBasedRecommender.html Maig 2017

Apache Spark

[24] https://spark.apache.org Abril 2017

[25] https://spark.apache.org/docs/latest/ml-collaborative-filtering.html Maig 2017

[26] https://spark.apache.org/docs/1.1.0/api/java/org/apache/spark/mllib/recommendation/ALS.html Maig 2017

[27] https://spark.apache.org/docs/1.4.0/api/java/org/apache/spark/mllib/recommendation/MatrixFactorizationModel.html Maig 2017

Amazon Web Service

[28] https://aws.amazon.com/es/ Maig 2017

[29] https://aws.amazon.com/es/s3/?hp=tile&so-exp=below Maig 2017

[30] https://aws.amazon.com/es/ec2/?hp=tile&so-exp=below Maig 2017

[31] https://aws.amazon.com/es/elasticbeanstalk/?hp=tile&so-exp=below Maig 2017

[32] https://aws.amazon.com/es/rds/?hp=tile&so-exp=below Maig 2017

Conclusions

[33] http://projects.spring.io/spring-security/ Maig 2017

Annexos

Implementació Java Spring

[34] https://start.spring.io Abril 2017

[35] https://www.thymeleaf.org Abril 2017

Implementació Apache Mahout

[36] https://mahout.apache.org/users/recommender/userbased-5-minutes.html Abril 2017

[37] http://www.imdb.com Abril 2017

[38] https://www.themoviedb.org Abril 2017

Implementació Apache Spark

[39] https://spark.apache.org/docs/latest/programming-guide.html#transformations Abril 2017

[40] https://spark.apache.org/docs/latest/programming-guide.html#actions Abril 2017

Tmdb API

[41] https://developers.themoviedb.org/3/getting-started Abril 2017

[42] https://developers.themoviedb.org/3/movies/get-movie-details Abril 2017

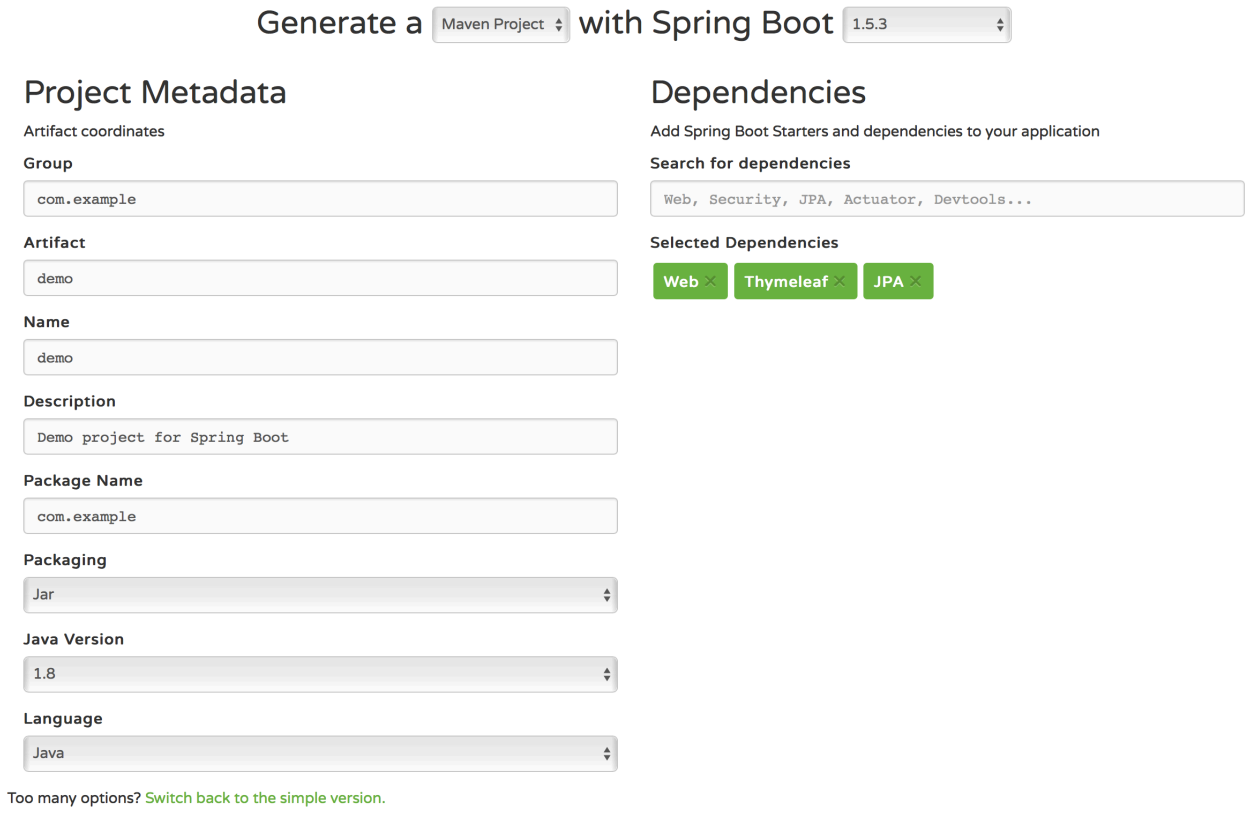
8. Annexos

8.1. Implementació Java Spring

Per implementar el front end de l'aplicació, s'ha escollit una aplicació web, fent ús de la tecnologia Java, amb l'ús del framework Spring. A més, el projecte comptarà amb les dependències de Maven, per poder gestionar l'ús de llibreries de manera més eficient.

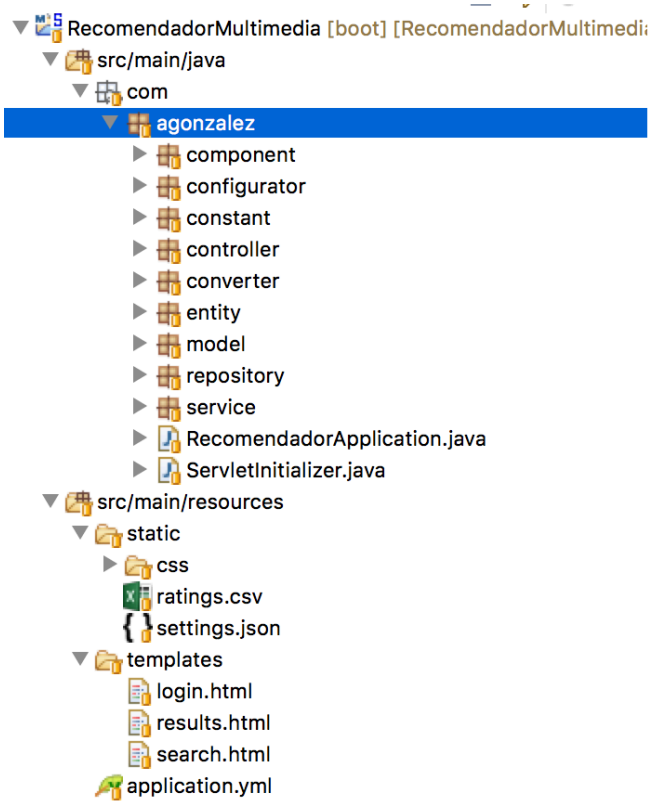
A la web [34] podem configurar un projecte base per començar el desenvolupament de l'aplicació. Per començar omplim les metadades del projecte i marquem només les següents dependències:

- Web - Web Full-stack web development with Tomcat and Spring MVC.
- Template Engines - Thymeleaf Thymeleaf templating engine, including integration with Spring.
- SQL – JPA Java Persistence API including spring-data-jpa, spring-orm and Hibernate.



Il·lustració 15: Plana principal de stat.spring.io

Després d'importar el projecte a Eclipse, es crearà una estructura de paquets per mantenir organitzat en tot moment el codi del recomanador multimèdia que es vol implementar.



Il·lustració 16: Estructura del projecte a Eclipse

Component: S'inclouran els components que faran possible l'aplicació.

Configurator: S'inclouran les classes de configuració per si es vol configurar una part de Spring.

Constant: S'inclourà una classe on es definiran les constants fetes servir al llarg del codi.

Controller: Contindrà els controladors encarregats de donar les vistes (html) als usuaris.

Converter: En cas d'haver de fer alguna conversió entre tipus o dades, aquí s'inclouran les classes necessàries.

Entity: Contindrà les entitats per controlar la base de dades. Fent servir Hibernate i JPA, aquestes entitats equivaldran a les taules de la base de dades del sistema.
Model: Els models dels objectes que contindrà l'aplicació els emmagatzemarem en aquest paquet.

Repository: Contindrà les interfícies que treballaren juntament amb les entitats per poder extreure informació de la base de dades sense escriure codi SQL.

Service: Contindrà la implementació dels serveis que oferirà l'aplicació Recomenador.

A més el projecte compte una carpeta de recursos (resources) on tindrem dues carpetes:

- **Static**, aquí s'emmagatzemaran els recursos constant de l'aplicació. Inicialment tindrem un fitxer 'settings.json' per definir els paràmetres de l'aplicació. Un fitxer buit amb el nom ratings.csv. Una carpeta css on es guardarà les fulles d'estil necessàries per a les templates.

- **Templates**, aquí es guardaran els fitxers de tipus .html. Aquests fitxers, amb l'ajut de thymeleaf, els tenim definits amb certes variables per poder escriure dades de manera dinàmica.

Fitxer settings.json

La idea principal és poder parametritzar l'aplicació d'una manera senzilla. D'aquesta manera, realitzant un petit canvi en el fitxer de configuració es pot alterar el comportament de la implementació amb el framework Mahout.

La implementació amb Mahout necessita configurar un context amb uns certs valors. Aquests valors acostumen a ser constants. En algun moment, pot ser interessar alterar un d'aquests valors, sense a veure de recórrer a modificar el codi font. Tasca que un usuari normal no realitzarà. Per tant, en la fase de configuració del recomanador, aquests valors es llegirà d'un fitxer extern anomenat 'settings.json' per obtenir el valor de similitud i valor del llindar.

```
{
    "user-similarity": "PearsonCorrelationSimilarity",
    "threshold": 0.5
}
```

Il·lustració 17: Exemple de fitxer settings.json

Plantilles HTML

L'aplicació web contindrà principalment pàgines html, correctament codificades per ser utilitzades mitjançant thymeleaf [35].

- Login.html: Per poder entrar al sistema
- Registre.html: Formulari per donar d'alta usuaris nous al sistema

- Search.html: Plana principal on s'estableixen els criteris de recomanació.
- Results.html: Plana on es presentaran els resultats obtinguts pel Recomanador Multimèdia.
- Votes.html: Plana on l'usuari pot consultar, de manera ordenada, les pel·lícules que ha valorat.

8.2. Implementació Apache Mahout

Com a l'estructura del projecte es fa ús de Maven. Per incloure les llibreries de Mahout necessàries, només s'ha d'incloure les llibreries de Mahout al fitxer pom.xml. Més detalladament les següents dependències:

```
<dependency>
     <groupId>org.apache.mahout</groupId>
     <artifactId>mahout-mr</artifactId>
     <version>0.10.0</version>
</dependency>

<dependency>
     <groupId>org.apache.mahout</groupId>
     <artifactId>mahout-core</artifactId>
     <version>0.9</version>
</dependency>

<dependency>
     <groupId>org.apache.mahout</groupId>
     <artifactId>mahout-integration</artifactId>
     <version>0.9</version>
</dependency>
```

Il·lustració 18: Dependències a Maven per incloure Mahout

Una vegada fet això netegem el projecte i instal·lem les noves llibreries amb la comanda:

```
mvn clean install
```

Il·lustració 19: Comanda Maven per netejar i instal·lar les llibreries

Amb aquests passos ja tindrem disponibles les llibreries Mahout al projecte. Per començar el desenvolupament amb l'Eclipse.

Per endinsar-se en el món de Mahout es va dur a terme una primera aproximació fent servir el manual [36] Amb aquest concepte es va elaborar el primer esquema de la implementació de Apache Mahout.

Esquema versió 1

Per implementar un simple recomanador, amb Mahout i executar-lo en local, són necessàries unes poques línies de codi:

```
DataModel model = new FileDataModel(new File("/path/to/dataset.csv"));
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood,
similarity);
List recommendations = recommender.recommend(2, 3);
for (RecommendedItem recommendation : recommendations) {
  System.out.println(recommendation);
}
```

Il·lustració 20: Example d'implementació d'un recomanador.

El primer pas és crear un objecte DataModel amb el fitxer csv que conté les dades. En el nostre cas el fitxer 'ratings.csv'. Després s'ha de definir com buscarà la similitud entre usuaris, en l'exemple ho fa amb la correlació de Pearson, però són vàlids els següents mecanismes:

- CityBlockSimilarity
- EuclideanDistanceSimilarity
- LogLikelihoodSimilarity
- PearsonCorrelationSimilarity
- SpearmanCorrelationSimilarity
- TanimotoCoefficientSimilarity
- UncenteredCosineSimilarity

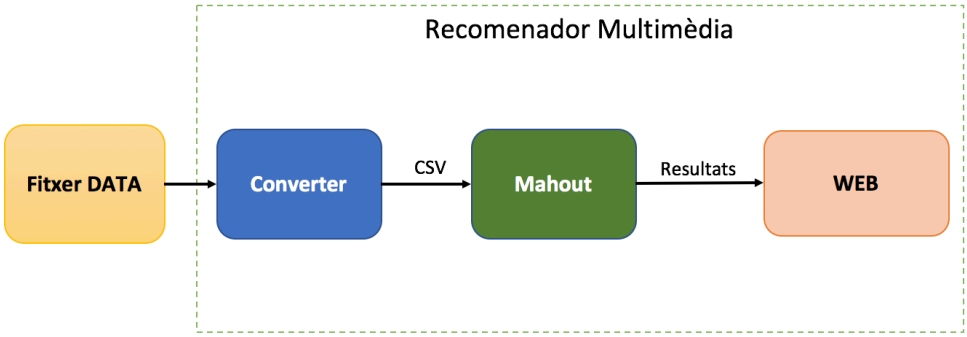
Aquestes similituds es poden ajustar amb un llindar per discernir entre els usuaris que entraran en el grup de 'veïns propers' o no. A l'exemple s'estableix un llindar de 0.1. Un llindar més proper a 1 estableix que els veïns escollits seran més semblants entre ells mateixos.

En lloc de definir un llindar. Es pot definir el nombre *N* de veïns més similars. Això es fa amb la següent crida que retornarà un objecte de tipus UserNeighborhood.

```
NearestNUserNeighborhood(int n, UserSimilarity userSimilarity, DataModel dataModel)
```

Finalment ja podem crear un recomanador genèric basat en usuaris i demanar-li que ens retorni una llista de recomanacions. Seguint l'exemple anterior, se l'hi demana 3 objectes per recomanar-li a l'usuari amb id 2.

El primer esquema que s'implementarà Mahout segueix el diagrama següent:



Il·lustració 21: Primer esquema d'implementació

Aquest esquema llegirà les dades d'un fitxer de text. El convertirà, si és necessari, a un fitxer en format csv. On les seves dades es trobaran separades pel caràcter ','. Muntarà un recomanador basat en el fitxer de configuració 'settings.json' i finalment, donarà una llista dels objectes recomanats.

Mahout retorna la llista dels productes recomanats amb un objecte propi de la llibreria. *RecommendedItem*. Per tant, també s'ha d'implementar una conversió perquè els objectes tornats tinguin un format més específic per a l'objectiu d'aquest TFG.
S'ha creat l'objecte *ItemRating*, per ara contindrà el id de la pel·lícula, el id de l'usuari i el ranting de la seva respectiva valoració.

L'entitat sobre els productes recomanats és sempre una pel·lícula. Aquestes tenen un id que és el que ens torna el recomanador. Aquest id no deixa de ser una referència interna en l'ecosistema del nostre recomanador, però no té gaire valor si volem fer-lo servir per fer alguna consulta cap a l'exterior del recomanador.

Tindre una base de dades pròpia per emmagatzemar totes les pel·lícules i totes les seves dades (caràtules, sinopsis, actors, etc.) seria molts costos en espai i en temps de manteniment. Afortunadament existeixin moltes pàgines que ofereixen aquests serveis: Imdb [37] o tmdb [38]. Justament en el dataset hi ha un fitxer 'links.csv' que relaciona el id propi amb els id d'aquestes dues webs respectivament.

Per tant, abans de presentar els resultats del recomanador al front-end es farà servir l'API de la web tmdb per recollir més informació sobre les pel·lícules. Veure l'annexa [5] per més informació.

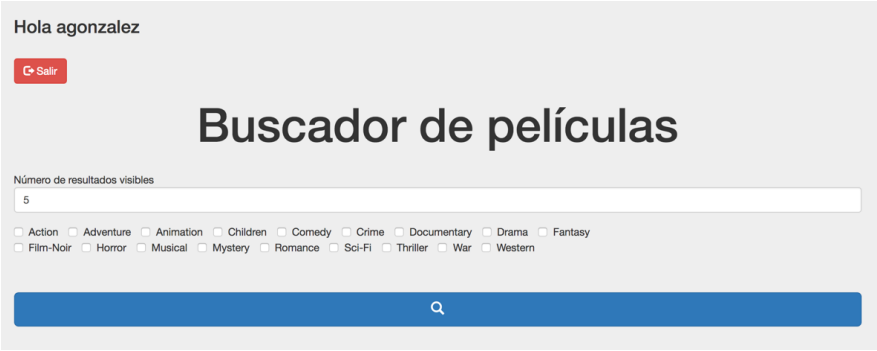
Per tal de obtindré un millor rendiment. Els fitxers 'links.csv' i 'movies.csv' es migraran a una base de dades mySQL. Respectant totes les seves dades. L'estratègia a seguir serà: una vegada obtinguts els ItemRating recomanats es tracta de fer una consulta a la base de dades per el id (intern) de la pel·lícula. Així s'obtindrà els títols, els gèneres als quals pertany i el id de la web Tmdb.

Esquema versió 2

Amb la implementació de la versió 1, s'ha obtingut un recomanador de pel·lícules, que gràcies a la base de dades relacional, s'obtenen els id de les pel·lícules recomanades a la pàgina The movie Db [38]. Ara bé, el següent pas per fer un recomanador eficaç és contestar a preguntes com: Si volem fer recomanacions d'un cert gènere de pel·lícules? O Si volem obtindré 1, 2, 5, ... N recomanacions?

Per poder donar resposta a aquestes preguntes es desenvoluparà un front-end en Spring, veure annexa [1] per més informació.

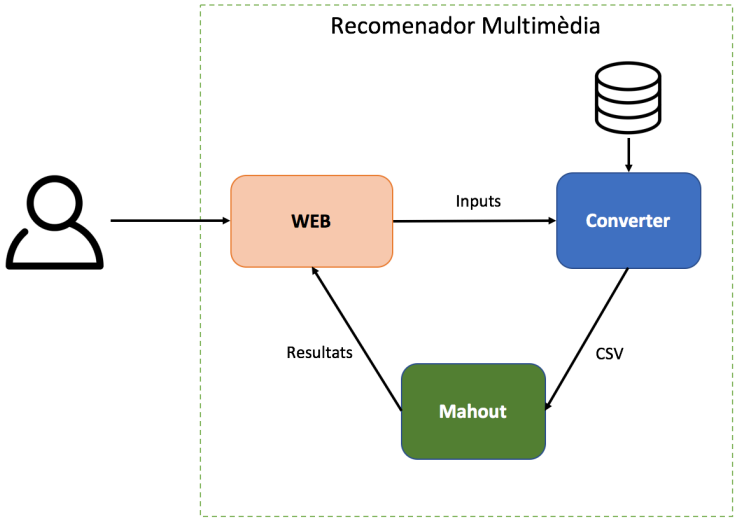
Mitjançant aquest front-end, l'usuari podrà seleccionar els gèneres que li interessen i el nombre de pel·lícules recomanades que vol obtindré. La plana principal del recomanador es basa en un formulari html, per llançar amb una petició POST la sol·licitud al back-end.



Il·lustració 22: Plana principal del cercador de pel·lícules

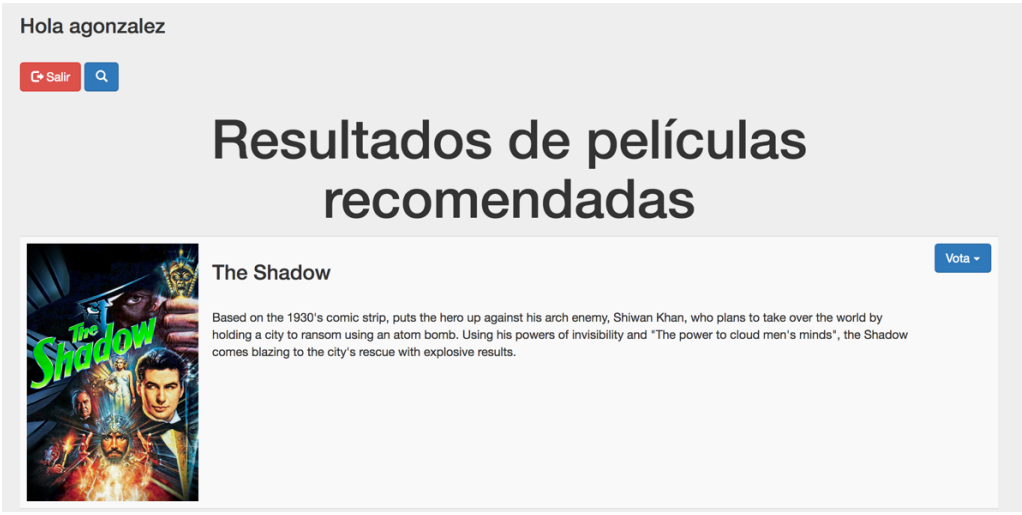
Fent ús dels repositoris de JPA, el que es fa és el següent. Quan l'usuari faci una nova cerca, la configuració d'aquesta es guardarà en un fitxer 'SearchModel'. Amb això, es farà ús de la basa de dades relacional per obtenir tots els id de les pel·lícules que pertanyin al gènere o gèneres marcats. Després, farem una segona consulta per obtenir tots els ratings

que afecten aquestes pel·lícules. Per fer això, abans també he de migrar el fitxer 'ratings.csv' a la base de dades mySQL. Finalment es generà dinàmicament un fitxer CSV, que serà l'entrada del recomanador Mahout.



Il·lustració 23: Diagrama de l'esquema 2

Un exemple dels resultats obtinguts per la temàtica d'acció:



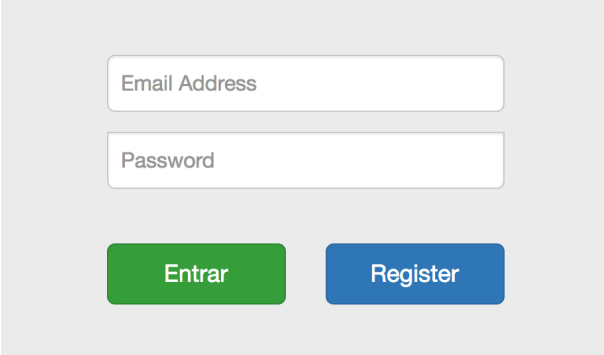
Il·lustració 24: Exemple de la plana 'results.html' per la temàtica 'Acció'

Esquema versió 3

Ara ja tenim un cercador que ofereix resultats filtrats per gènere. Aquest es poden combinar, per exemple si es marca 'Acció' i 'Aventures' només buscarà les pel·lícules que estiguin incluses en aquests dos gèneres. El comportament del checkbox funciona com

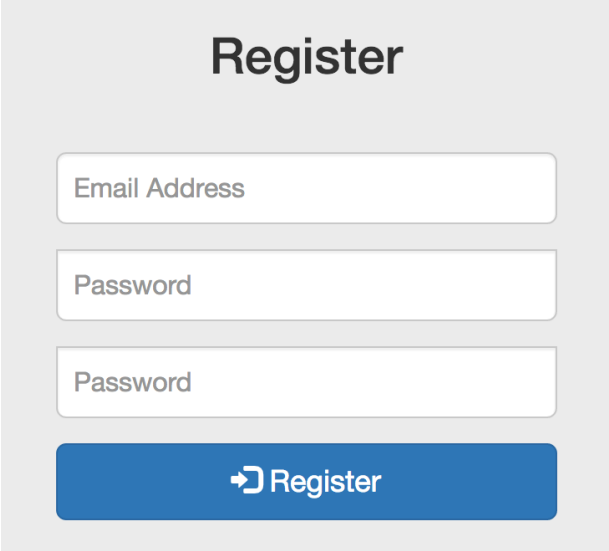
una *i* lògica. O dit d'una altra manera fa la intersecció. Si A és el grup de pel·lícules d'acció i B, el grup de pel·lícules d'aventures. El cercador tindrà en compte les pel·lícules resultants de A ∩ B.

L'altre problema que queda pendent de resoldre és que el recomanador necessitar passar-li un id d'usuari, que en teoria, hauria de ser el id del mateix usuari que estiguis fent servir el cercador. Així aquest, valoraria les seves pel·lícules i el recomanador oferiria uns resultats més personalitzats a l'usuari. Per resoldre això, es crea una nova template 'login.html' per controlar l'accés al cercador. D'aquesta manera, a obligar a l'usuari ha introduït els credencials a l'inici, es tindrà identificat al llarg de tot el circuit.



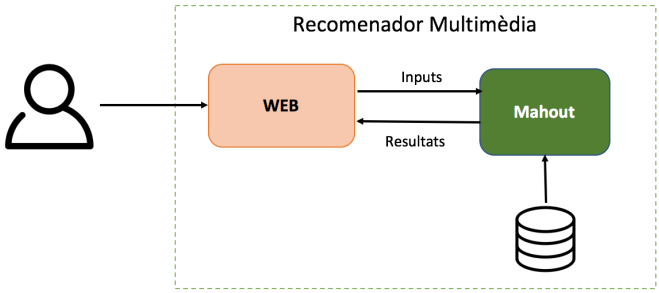
Il·lustració 25: Plana del login

Si els usuaris s'han d'identificar per entrar al recomanador, és evident que també es necessita una template per poder dur a terme el registre d'un usuari nou.



Il·lustració 26: Plana del registre

A més per aquest tercer esquema he volgut prescindir de l'escriptura del fitxer .CSV. Fent que la llibreria de Mahout interactuí directament sobre la base de dades i agafant el volum de dades dels ratings necessaris.



Il·lustració 27: Esquema sense el fitxer .CSV

Per materialitzar aquesta implementació s'ha de canviar la manera d'inicialitzar el nostre model de dades. A l'esquema de la versió 1 i 2 tenim el següent model:

```
DataModel model = new FileDataModel(new File("/path/to/dataset.csv"));
```

En aquest esquema quedaria de la següent manera:

```
MysqlDataSource dataSource = new MysqlDataSource();
dataSource.setServerName(AppConstant.DB_SERVER);
dataSource.setUser(AppConstant.DB_USER);
dataSource.setPassword(AppConstant.DB_PASS);
dataSource.setDatabaseName(AppConstant.DB_NAME);
dataSource.setPort(AppConstant.DB_PORT);


DataModel model = new MySQLJDBCDataModel(dataSource, nomTaula, nomColumnaUserID,
nomColumnaMovieID, nomColumnaRating, nomColumnaTimeStamp);
```

Una vegada implementat aquesta inicialització del model de dades, la resta dels passos són els mateixos que en els anteriors esquemes.

8.3. Implementació Apache Spark

A l'igual que es va fer amb la implementació de Apache Mahout, amb Spark, també s'ha de definir les dependències perquè el recomanador tingui accés a les llibreries de Spark. Per fer-ho s'afegeixen les següents dependències al fitxer pom.xml de Maven

```
<dependency>
	<groupId>org.apache.spark</groupId>
	<artifactId>spark-core_2.10</artifactId>
	<version>1.6.1</version>
</dependency>
<dependency>
	<groupId>org.apache.spark</groupId>
	<artifactId>spark-sql_2.10</artifactId>
	<version>1.6.1</version>
</dependency>
<dependency>
	<groupId>org.apache.spark</groupId>
	<artifactId>spark-mllib_2.10</artifactId>
	<version>1.6.1</version>
</dependency>
```

Il·lustració 28: Dependències a Maven per incloure Spark.

L'entitat principal per poder treball amb Apache Spark és el SparkContext. Aquesta classe és el context principal del motor Spark. Ella serà l'encarregada de gestionar els clústers i tot el que estigui al voltant de l'execució del nostre algorisme.

Per començar la implementació i seguint amb el patró definit amb la implementació de Mahout, es crearà un servei de recomanació de pel·lícules, *RecomenderSparkService*. Aquest servei serà l'encarregat de rebre l'input del front-end i oferint-li una llista de pel·lícules recomanades, d'igual manera que ho fa el servei de Mahout.

Per crear una instància de SparkContext i inicialitzar-la, en el nou servei farem amb la següent instrucció:

```
SparkConf conf = new
SparkConf().setAppName("RecomendadorMultimedia").setMaster("local[*]").set("spark.driver.all
owMultipleContexts", "true");
```

On podem veure:

- *setAppName("RecomanadorMultimedia"),* és una instrucció per donar-li un nom descriptiu al context.
- Set("Spark.driver.alloMultipleContexts", "true") defineix el comportament del context per permetre diversos contextos simultanis. Si no es defineix aquesta instrucció quan s'executa el context, en mode local pot donar problemes en alguns casos.
- setMaster("local[*]") defineix el comportament i la definició dels clústers que farà servir Spark durant la seva execució. Farem una aturada en aquest punt per veure amb més detall.

Clústers Spark

Tal com hem vist, és necessari definir un model amb què el context interactuarà amb els clústers d'execució de Spark. Això es fa mitjançant el mètode *setMaster* de la classe *SparkContext*.

Els valors que pot admetre *Master* són els següents:

local	Executa Spark de manera local amb un treballador.
local[K]	Executa Spark de manera local, definint K treballadors. De manera ideal aquest número no pot ser superior al nombre de cores del computador local.
local[*]	Es tracta d'una variació de l'anterior. Ja que aquesta opció executarà Spark en local amb tants treballadors com cores tingui el computador local.
spark://HOST:PORT	El context es connectarà a un clúster individual.
mesos://HOST:PORT	El context es connectarà a un sistema de clústers amb configuració MESOS.
yarn	El context es connectarà a un clúster YARN. Ho pot fer tant en mode client com en mode clúster. Aquesta és l'opció a escollir si volem que Spark treballi amb HADOOP.

Il·lustració 29: Taula amb els possibles valors de Master (SparkContext)

Com podem veure el recomanador en aquesta implementació fa ús d'un clúster local i crearà tant treballadors com cores tingui la màquina que l'executa. En el meu cas

particular, l'ordinador fet servir (veure punt 1.3) té un microprocessador amb la següent configuració:

Nombre del procesador:	Intel Core i7
Velocidad del procesador:	4 GHz
Cantidad de procesadores:	1
Cantidad total de núcleos:	4
Caché de nivel 2 (por núcleo):	256 KB
Caché de nivel 3:	8 MB

Il·lustració 30: Configuració del microprocessador usat per desenvolupar el TFG

Per tant, en el meu context farà ús de 4 treballadors.

RDD Datasets

Una vegada creat i inicialitzat el context de Spark, el següent pas és carregar el dataset que farem servir per el recomanador de pel·lícules. Aquest, serà una instància de la classe JavaRDD, a la qual és necessari especificar l'objecte que contindrà. En aquest cas, com l'objectiu és fer un recomanador, farem ús de la classe Rating del mateix Spark. Quedant d'aquesta manera:

```
JavaRDD<Rating> ratingRDD = jsc.textFile("Path to file");
```

Els objectes RDD admeten dos tipus d'accions: *transformacions* [39], serveixen per transformar un dataset en un nou conjunt de dades. *Accions* [40], serveixen per retornar un valor donat d'un dataset.

RDD Lambda funcions.

Les accions dels objectes poden ser implementades de dues vies. La primera fent ús d'interfícies definides a la mateixa classe.

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(new Function<String, Integer>() {
  public Integer call(String s) { return s.length(); }
});
```

Il·lustració 31: Exemple Spark, implementació de funció fent ús d'interfícies.

O fent ús de les funcions lambda incloses en Java 8. En aquest recomanador es faran servir aquest últim model de funcions.

Les funcions lambda tenen la propietat de ser-hi anònimes, per tant, no són necessàries definir-les en una classe especifica. La seva sintaxi bàsica és:

(paràmetres) → (cos lambda)

Un exemple bàsic podria ser:

```
  (String x) -> System.out.println(x);
```

Il·lustració 32: Exemple bàsic de funció lambda.

En el cas del recomanador, farem ús una combinació de conversió d'un dataset i una funció lambda per maqueta i convertir el dataset del fitxer de text en una col·lecció d'objectes Rating estructurat. Per fer-ho es fa ús del següent codi:

```
JavaRDD<Rating> ratingRDD = jsc.textFile(pathOut).map(s -> {
  String[] ratingArr = s.split(",");
  Integer userId = Integer.parseInt(Try.ofFailable(() -> ratingArr[0]).orElse("-1"));
  Integer movieId = Integer.parseInt(Try.ofFailable(() -> ratingArr[1]).orElse("-1"));
  Double rating = Double.parseDouble(Try.ofFailable(() -> ratingArr[2]).orElse("-1"));
  return new Rating(userId, movieId, rating);
}).cache();
```

Il·lustració 33: Funció lambda inclosa en el Recomanador Multimedia.

Aquesta funció té com a paràmetre un String d'entrada, equivalent a cada línia del fitxer de text. Per cada línia es fa un Split pel separador de columna definit en el fitxer .csv. En el cas del recomanador aquest caràcter sempre serà ','.

Seguidament és maqueta cada posició del vector en una variable que es farà servir per al constructor de la classe Rating. L'objecte Rating creat és el retornat per la funció Lambda.

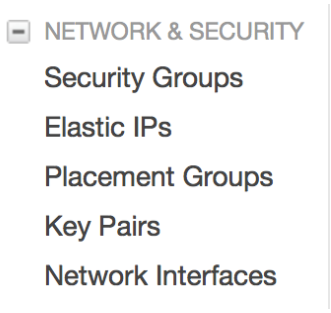
Finalment per obtindré el recomanador, farem ús de la classe MatrixFactorizationModel de la llibreria Mlib de Spark, per crear una factorització de la matriu i crear el model d'on extraurem les recomanacions. Per usuari i nombre d'elements, tal com es feia per Apache Mahout.

```
ALS als = new ALS();
MatrixFactorizationModel model = als.setRank(20).setIterations(10).run(ratingRDD);
```

8.4. Implementació Amazon Web Service EC2

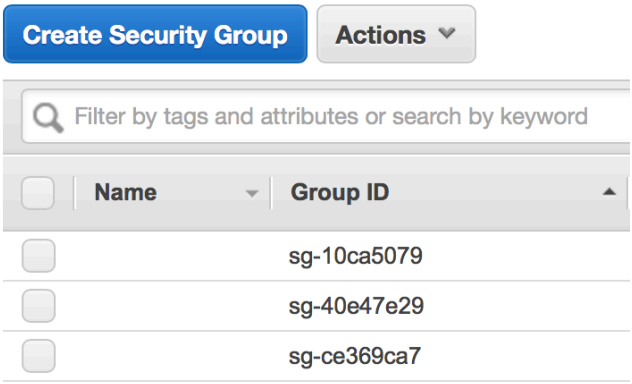
Per tal de portar el recomanador al núvol. S'ha optat per fer ús d'Amazon Web Service. El primer pas seria crear una instància d'un servidor virtual EC2. Abans de fer això aprofitarem per crear-hi un grup de seguretat.

Un grup de seguretat és com definirem com es comunicarà la nostra instància d'EC2 amb el món exterior. És equivalent al nostre encaminador de casa, on podem obrir i tancar ports, etc. Per crear el grup a la part esquerra de la finestra, fem clic sobre "Security Groups"



Il·lustració 34: Grup de opcions al panell esquerra

Fem clic sobre el botó "Create Security Group"



Il·lustració 35: Llistat dels Security Groups creats.

Ara es dóna un nom el més auto descriptiu possible i es configuren les regles d'entrada i sortida com a la següent captura:

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	8080	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
MYSQL/Aurora	TCP	3306	0.0.0.0/0

Il·lustració 36: Regles d'entrada

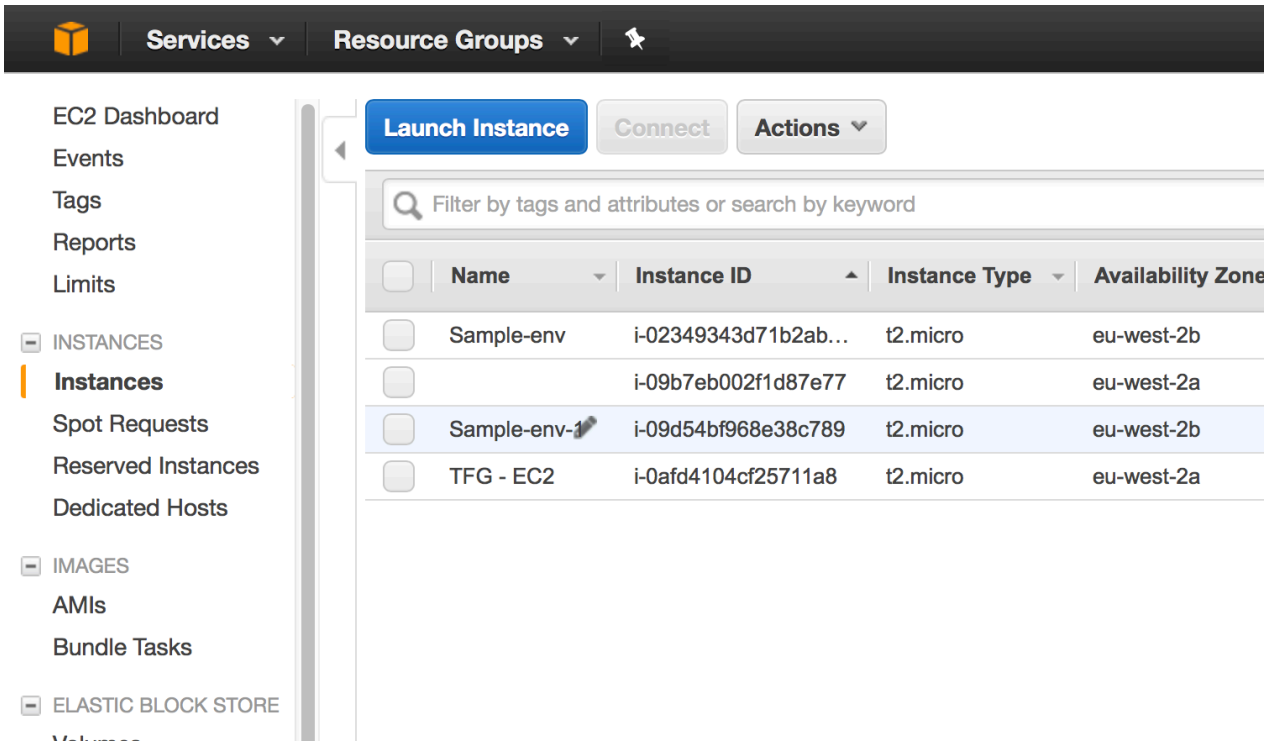
Podem veure que he configurat:

- El port 8080 que és el que farà servir Spring per arrancar el tomcat i mostrar l'aplicació a Internet.
- El port 22, per poder connectar-nos mitjançant SSH a la nostra instància EC2.
- El port 3306, per si volem connectar-nos remotament a la base de dades de la instància.

Type	Protocol	Port Range	Destination
All traffic	All	All	0.0.0.0/0

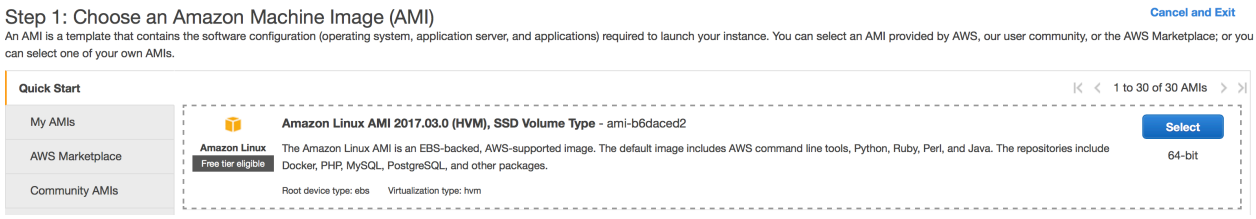
Il·lustració 37: Regles de sortida

Ara que ja tenim el grup de seguretat creat, comencem la creació de la instància EC2. Per crear una nova instància, es fa clic sobre el botó "Launch Instance".



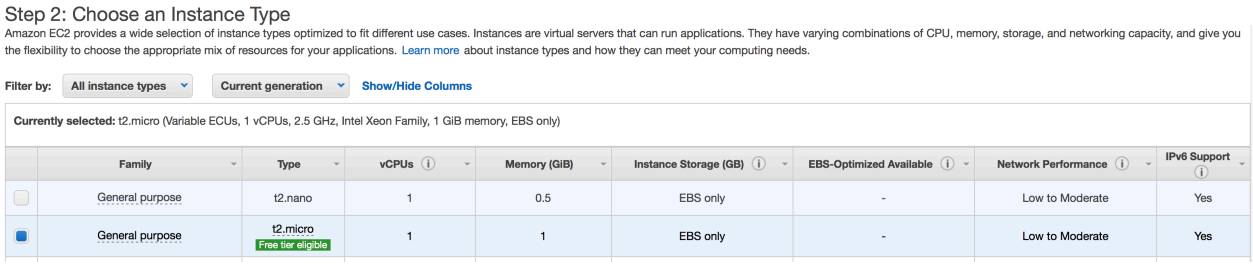
Il·lustració 38: Plana principal d'instàncies EC2

Això ens obrirà un assistent pas a pas per configurar la nova instància. El primer que ens demana és el hardware que tindrà aquesta instància. Com estem fent ús d'un compte gratuït ens fixem la màquina que té l'etiqueta "Free tier eligible" Nosaltres escollim la que ens marca per defecte.



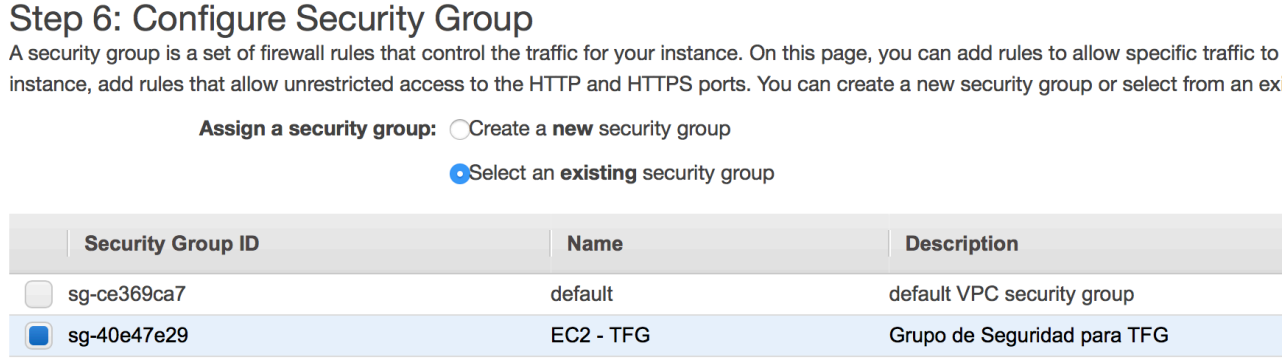
Il·lustració 39: Selecció de la imatge que tindrà la nostra instancia d'EC2

El següent pas es definir el tipus d'instància. En aquest pas, si volem fer ús del compte gratuït només podem marca la opció ressaltada.



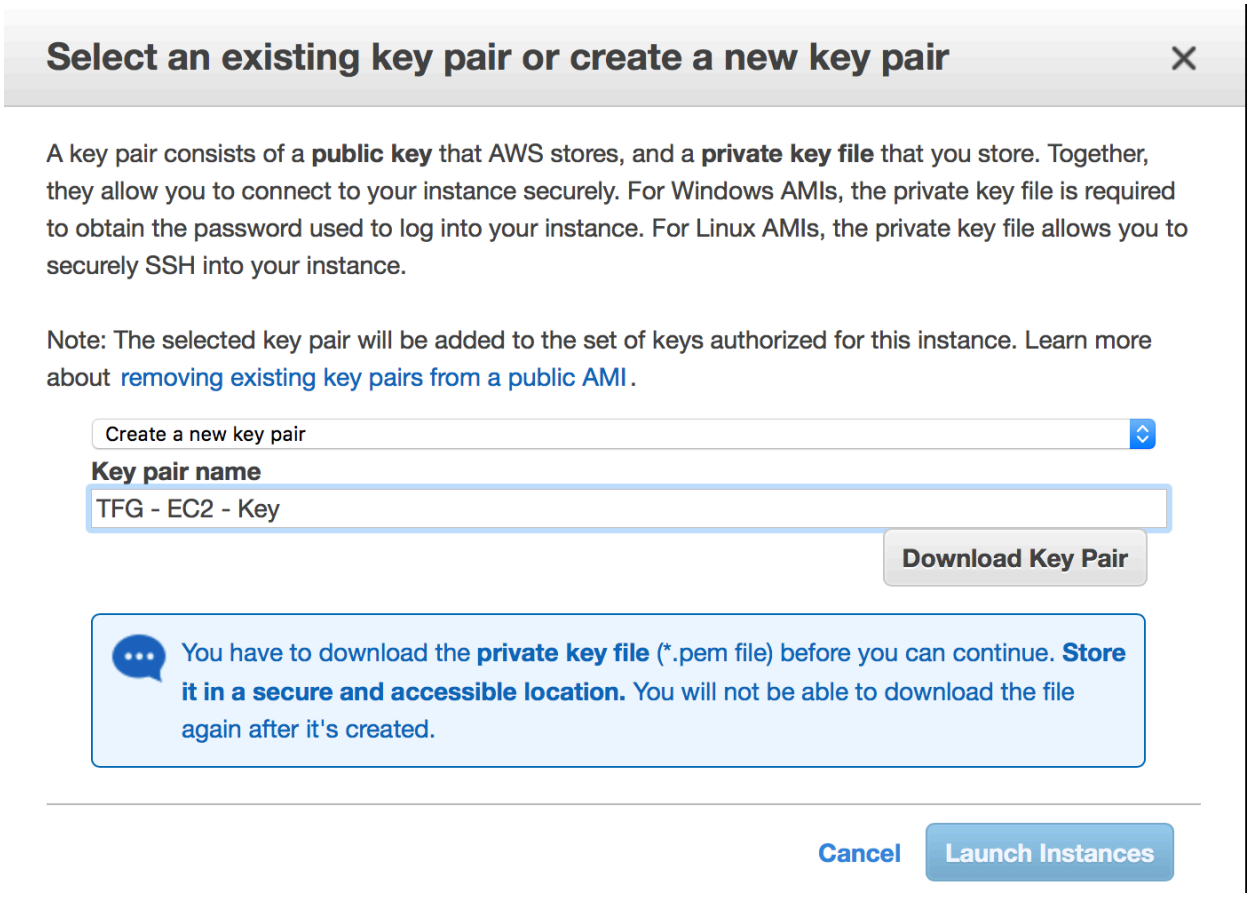
Il·lustració 40: Selecció de tipus d'instància.

Podem deixar els passos 3, 4 i 5 de manera predeterminada i saltar al pas 6. Aquí es defineix el grup de seguretat de la nova instància. Com el primer que hem fet ha estat crear el grup de seguretat, l'escollim dels grups existents.



Il·lustració 41: Selecció del grup de seguretat per la nova instància.

L'últim pas abans d'arrancar la instància és crear un parell de claus. Publica i privada. Aquestes claus ens servirà per poder connectar-nos mitjançant SSH.



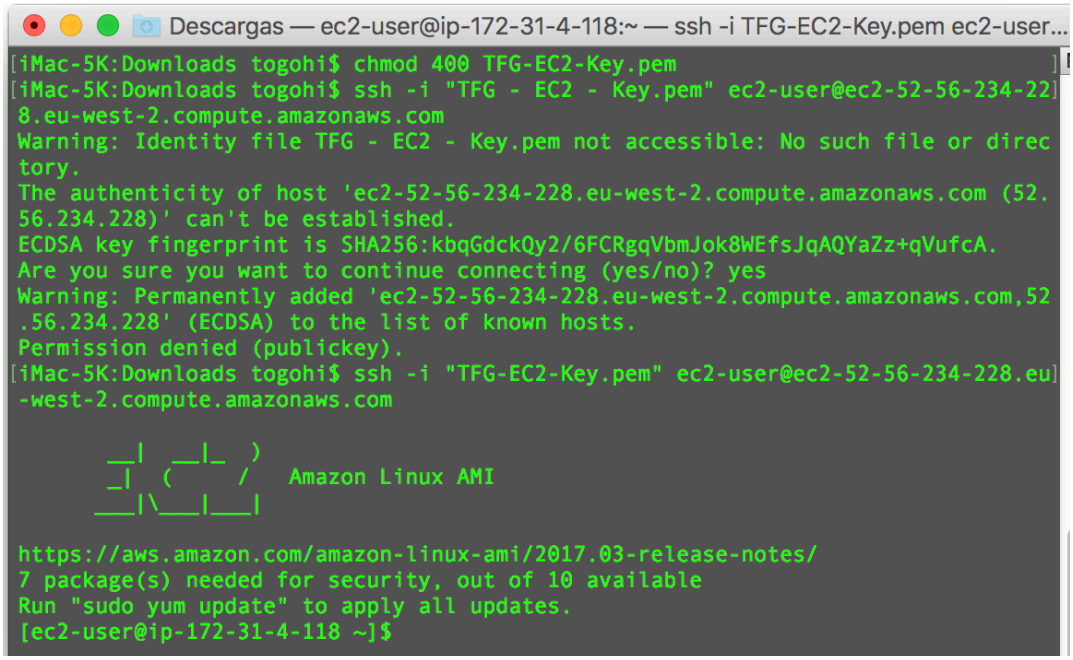
Il·lustració 42: Generació de parell de claus

Una vegada creada i hagi acabat d'arrancar la instància (Pot trigar uns minuts). En el nostre ordinador, obrim la terminal i ens situem en la carpeta on ens hem descarregat el certificat que conté la clau privada. Executem la següent comanda per actualitzar el permís del fitxer.

```
chmod 400 TFG-EC2-Key.pem
```

Ara ja estem en situació de poder connectar-nos a la nostra instància, per fer-ho executem:

```
ssh -i "TFG-EC2-Key.pem" ec2-user@ec2-52-56-234-228.eu-west-2.compute.amazonaws.com
```



Il·lustració 43: Finestra del terminal connectat a EC2.

El següent pas és instal·lar tot el programari necessari a la màquina virtual per poder fer ús del recomanador. Primer de tot, instal·lem l'última versió de Java i desinstal·lem la que té instal·lada:

```
sudo yum update
sudo yum install java-1.8.0
sudo yum remove java-1.7.0-openjdk
```

Fet això necessitem instal·lar el servidor de la base de dades. Per tant instal·lem MySql Server i s'estableix la contrasenya de l'usuari root.

```
sudo yum install mysql
sudo yum install mysql-server
mysqladmin -u root password '------'
```

Abans de continuar necessitem enviar els fitxers .csv del dataset de Movilens cap a la màquina virtual. D'aquesta manera podem fer la migració de les dades en un entorn local, ja que es farà des de la mateixa màquina virtual. Per copiar les dades executem:

```
scp -i "/Users/togohi/Downloads/TFG-EC2-Key.pem" /Users/togohi/Desktop/*.csv ec2-user@ec2-52-56-234-228.eu-west-2.compute.amazonaws.com:~/.
```

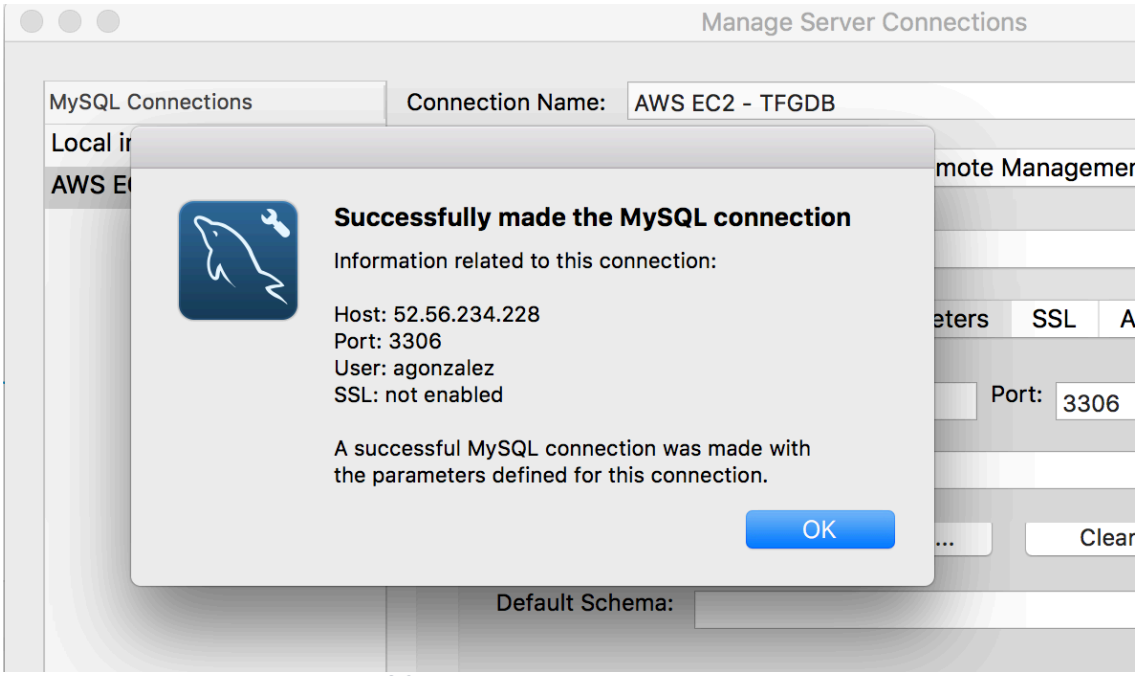
Aprofitem i també copiem l'aplicació del recomanador:

```
scp -i "/Users/togohi/Downloads/TFG-EC2-Key.pem"
/Users/togohi/Documents/workspace/RecomendadorMultimedia/target/RecomendadorMultimedia-
1.0.0.war ec2-user@ec2-52-56-234-228.eu-west-2.compute.amazonaws.com:~/.
```

Entrem dintre de MySQL i configurem un usuari. Per defecte root, no es pot fer servir en connexions remotes. També creem l'esquema 'TFG' que és el que usa el recomanador.

```
mysql -u root -p
CREATE SCHEMA TFG;
CREATE USER 'agonzalez' IDENTIFIED BY 'togohi1957';
GRANT ALL ON TFG.* TO 'agonzalez';
```

Amb aquest nou usuari podem intentar connectar-nos remotament:



Il·lustració 44: Test de connexió de SQL Workbench.

Ara podem executar l'aplicació perquè, gràcies a Hibernate, dintre de l'esquema 'TFG' creí tota l'estructura de taules. Executem el recomanador:

```
Java –jar RecomendadorMultimedia-1.0.0.war
```

Il·lustració 45: Procés d'arrancada de l'aplicació a AWS.

Per acabar ja tenim l'estructura de taules creades, ara hem de fer la importació de les dades dels fitxers .csv cap a mySQL. Per tant, tornem a entrar a MySQL i executem:

```
load data local infile '/home/ec2-user/movies.csv' into table movies
 fields terminated by ','
 enclosed by '"'
 lines terminated by '\n'
 (movieid, title, genres)
```

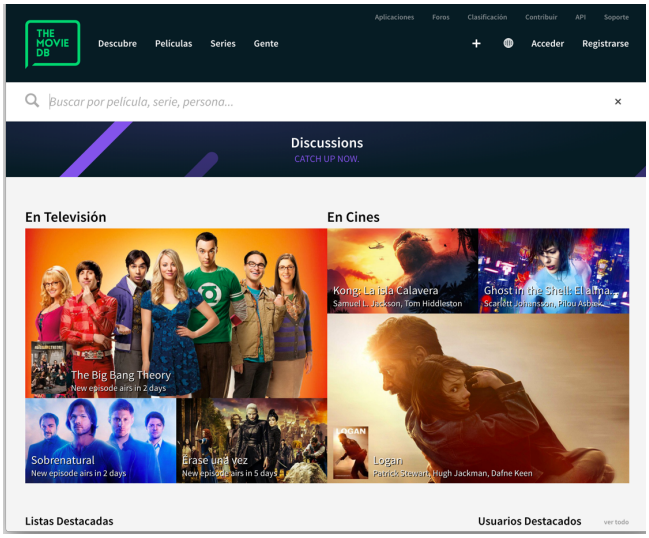
```
 load data local infile '/home/ec2-user/links.csv' into table movielinks
 fields terminated by ','
 enclosed by '"'
 lines terminated by '\n'
 (movieid, imdbid, tmdbid)
```

```
 load data local infile '/home/ec2-user/ratings.csv' into table movieratings
 fields terminated by ','
 enclosed by '"'
 lines terminated by '\n'
 (userid, movieid, rating)
```

Per acabar hem de torna a executar el recomanador i ja estarà totalment desplegat sobre Amazon Web Service.

8.5. Tmdb API

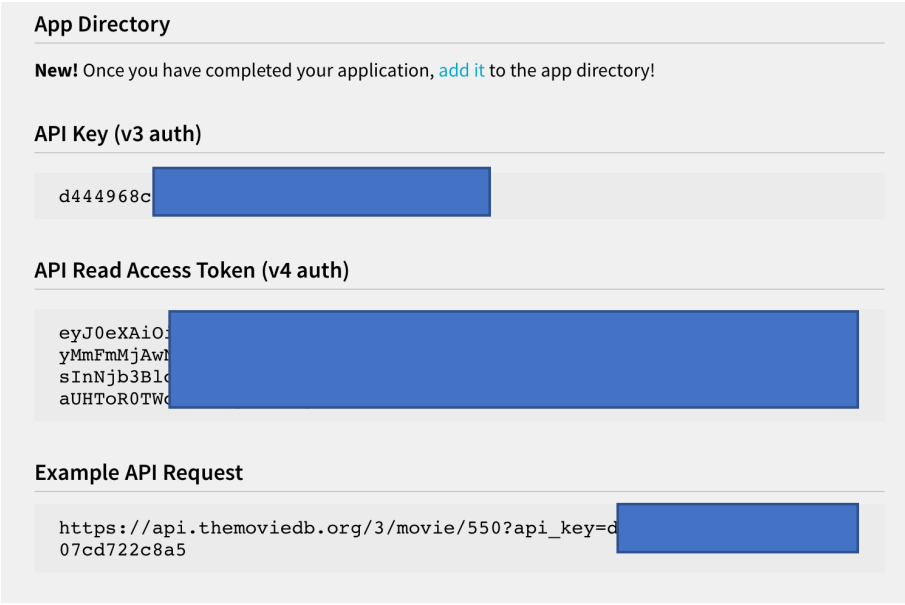
La web 'The Movie DB' [38] es tracta d'una pàgina web de recomanacions de pel·lícules i sèries de televisió. Jo la conec des de fa anys, perquè programes com Kodi (l'antic XMBC) fan servir la seva API per extreure les caràtules de les pel·lícules i els seus backdrops (una espècie de fons de pantalla).



Il·lustració 46: Plana principal de Tmdb.org

Encara que la seva API ja té una eina integrada per recomanar pel·lícules, no ho farem servir. L'objectiu d'aquest TFG és fe servir aquesta web per extreure la metadata de les pel·lícules.

Per fer servir l'API només ens enregistrem com a desenvolupadors i donar d'alta una aplicació, això ens donarà un token per poder fer les HTTP Request necessàries.



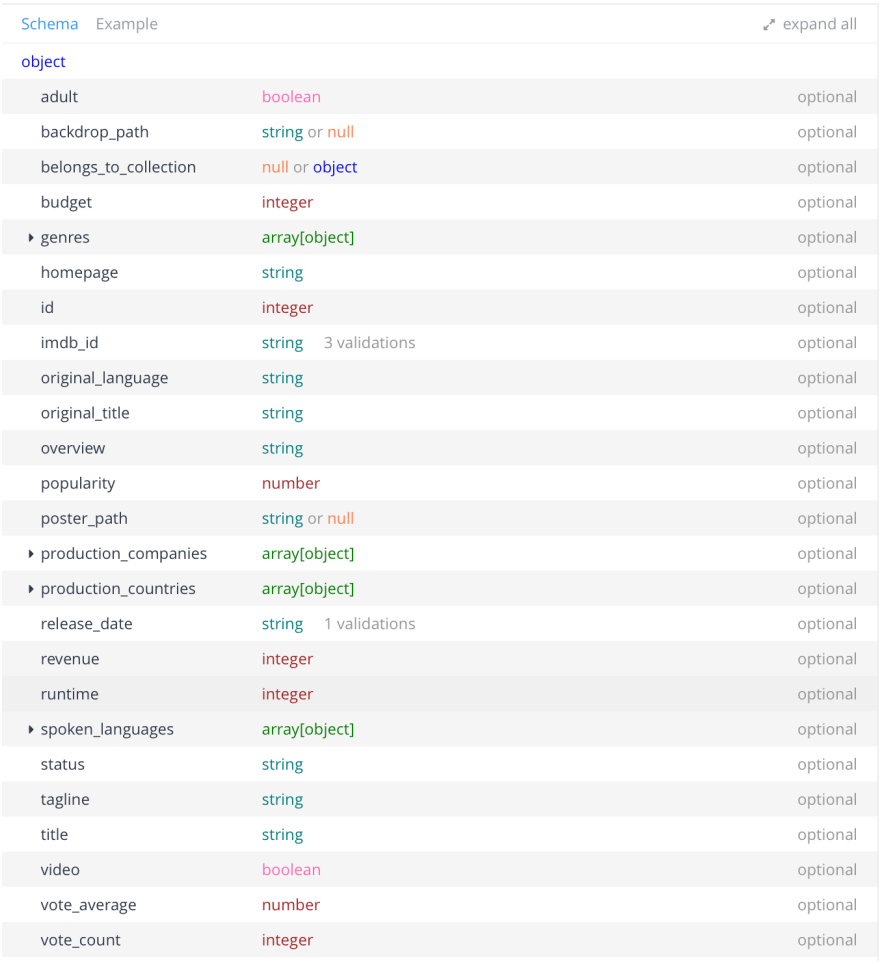
Il·lustració 47: Token i key per fer servir la API de Tmdb

Una vegada tinguem una API Key per el recomanador podem començar a fer peticions a la web. A la documentació [41] podem veure totes les crides que es poden realitzar. En el recomanador només es farà servir aquesta [42]

Il·lustració 48: Crida per obtenir la informació d'una pel·lícula

Aquesta petició ens retornarà un JSON amb moltes dades de la pel·lícula. Només recollirem la informació de les claus:

- Poster_path: Conté la ruta a la imatge de la portada.
- Overview: Conté la sinopsi de la pel·lícula.



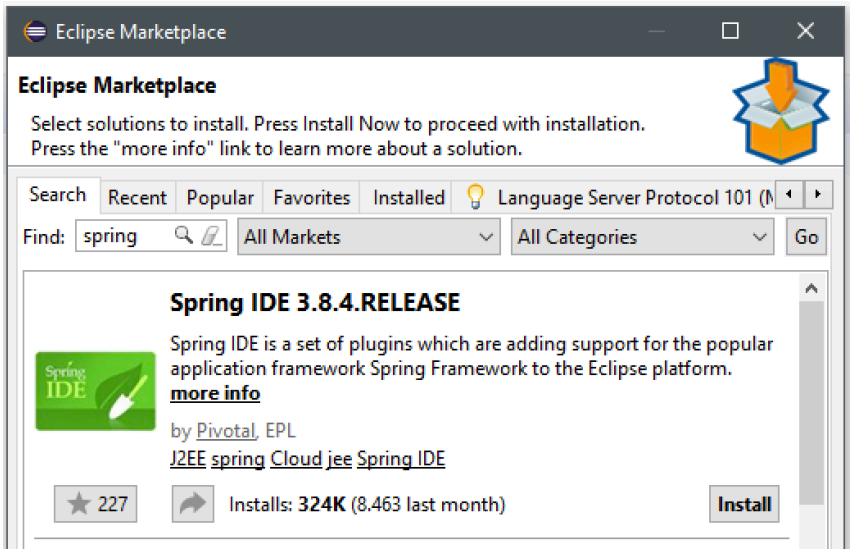
Il·lustració 49: Dades incloses en el fitxer Json

8.6. Configuració per l'execució en un entorn local

Per poder executar el recomanador en local, primer ens hem d'assegurar de:

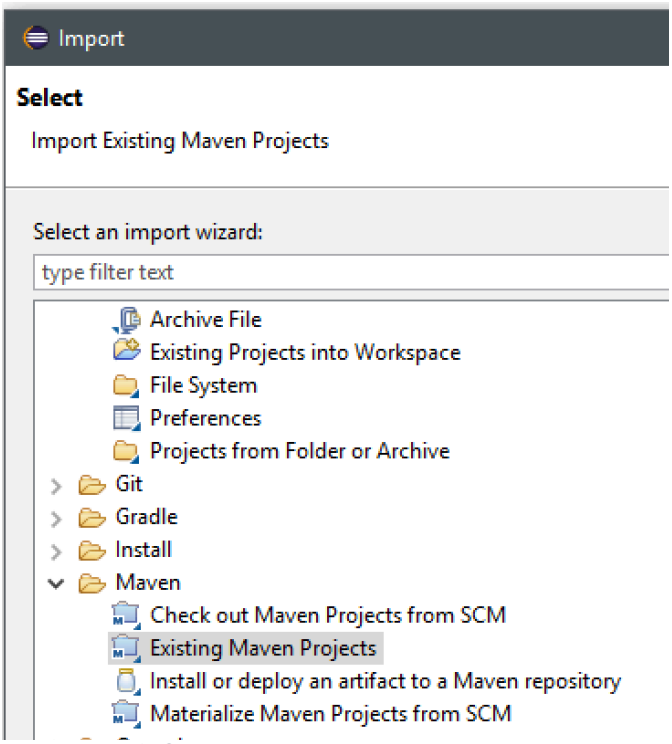
- Instal·lar mySQL Server.
- Crear un esquema a la base de dades amb el nom 'TFG'.
- Importar a la base de dades el dataset de movielens adjunts a aquesta memòria. Es preferible importar el conjunt de dades Lite.

Abans d'importar el recomanador a eclipse, ens hem d'assegurar que tenim instal·lat el plug-in de Spring. Es pot instal·lar des de el mateix Marketplace de Eclipse.



Il·lustració 50: Eclipse Marketplace - Spring

S'ha d'importar el workspace, adjunt a aquesta memòria a Eclipse i configura l'accés a la base de dades mySQL.

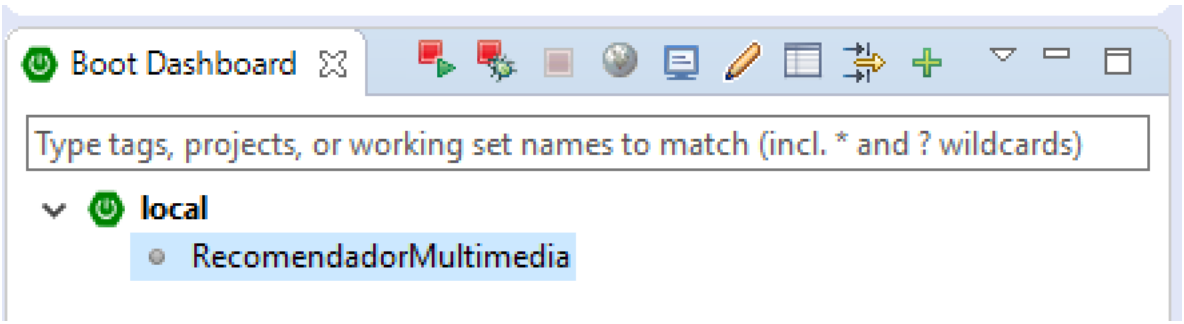


Il·lustració 51: Opció per importar el recomanador a eclipse.

Això es fa editant el fitxer 'application.yml' de la carpeta 'resources'. Més concretament les línies:

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/TFG?useSSL=false
    username: root
    password: ------
```

Una vegada configurat l'accés a la base de dades podem arrancar el recomanador mitjançant el Boot Dashboard de Spring.



Il·lustració 52: Boot Dashboard de Spring.

8.7. Configuració per l'execució sobre Amazon Web Service

Per treballar amb la versió desplegada a Amazon Web Service tant sol és necessari una consola SSH i un Navegador web.

Amb aquesta memòria s'adjunta el certificat .PEM necessari per connectar-nos a la màquina virtual d'Amazon, s'ha d'executar aquestes dues instruccions.

```
chmod 400 TFG-EC2-Key.pem
```

```
ssh -i "TFG-EC2-Key.pem" ec2-user@ec2-35-176-71-113.eu-west-2.compute.amazonaws.com
```

Una vegada dintre de la instància d'Amazon podem arrancar el recomanador:

```
java –jar RecomendadorMultimedia-1.0.0.war
```

Ara nomes farà falta, obrir un navegador i anar a la web pública del servidor virtual EC2 creat:
http://35.176.71.113:8080