Developing a board game recommender system based on Linked Data

David Castellà

14 de setembre de 2015

Agraïments

Primer de tot donar les gràcies als directors d'aquest projecte, Juan Manuel Gimeno i Roberto García, que sense la seva ajuda aquest treball no hauria estat possible.

També donar les gràcies a TOTS els membres del grup de recerca GRIHO, on al llarg d'aquests 4 anys no només he tingut grans companys de feina, sinó grans amics. Sé que enlloc podré estar tant a gust com aquí.

Agraeixo de tot cor a la Sandra de Mesas, la meva parella, no només el disseny dels logotips de les aplicacions, si no pel suport i ànims que m'ha donat, i aguantar-me en els moments de més nervis i estrès, que no es fàcil.

A la meva família, que sigui on sigui, sempre rebo el seu suport i sense ells, segurament, avui no estaria aquí.

Per últim agrair a tots aquells que em van ensenyar el *hobbie* dels jocs de taula, si no fos per vosaltres no sé quin treball hauria fet.

Índex

Índex de figures

Índex de taules

Capítol 1

Introducció

En els últims anys s'està veient com en el món de la informàtica apareixen noves tecnologies contínuament, on aquestes generen nous models de negoci. Una d'aquestes és l'explotació de dades per a fi d'implementar noves funcionalitats o bé fins comercials. Actualment grans empreses com *Google*, *Amazon*, *Ebay*, entre altres, rastregen i emmagatzemen tot el que fa l'usuari a dins dels seus sistemes, després tota aquesta informació és processada per a ser reutilizada de diferents maneres, doncs així són capaços de mostrar-nos els productes més afins a les nostres preferències o inclús mostrar la publicitat que serà del nostre interès, inclús estant fora dels seus sistemes (p.e. *Google AdSense*).

Una de les tecnologies que ens permet el tractament d'un gran volum de dades, tot navegant per diferents sets de dades, són les dades enllaçades o *Linked Data*. Mitjançant aquesta tecnologia és poden efectuar consultes, fetes amb el llenguatge *SPARQL*, per a obtenir les dades que es busquen per a un fi determinat. Un exemple de conjunt de dades seria la *DBPedia*, un clon de *Wikipedia* o *LinkedMDB*, un clon d'*IMDB*, la base de dades de pel·lícules.

Per tant, en aquest treball, s'ha volgut treballar en aquesta tecnologia, no només creant un sistema de recomanacions de jocs de taula, si no també creant un nou set de dades on hi figurin els jocs, usuaris i les valoracions que han fet els usuaris per cada un d'aquest jocs. Per això, es mostra tot el procediment que s'ha seguit per a crear tant el motor de recomanació, com l'aplicació web que utilitza aquest motor per a treure aquestes recomanacions d'una forma clara i amigable per a l'usuari.

1.1 Motivació

Des de fa ja uns anys, que un dels *hobbies* que més m'apassionen, són els jocs de taula moderns. A diferència dels jocs més comercials o els de tota la vida com l'Oca o el Parxís, que tota l'estratègia de la partida està entregada a l'atzar, aquests jocs, premien més la bona estratègia i l'adaptació als canvis, deixant l'atzar en segon pla. Aquest tipus de jocs van començar a sortir cap a l'any 1995, a partir de la publicació d'Els Colons de Catan, creat per Klaus Teuber (protesista dental alemany), que va introduïr unes mecàniques i una interacció entre jugadors que el van fer un èxit de vendes, ha estat traduït a 17 llengües diferents, que entre les quals hi ha el català. Des de llavors, s'ha creat el terme de joc de taula modern i milers de jocs, alguns amb més èxit que d'altres han sortit al mercat, creant tota una comunitat de jugadors, competicions i events de renom a nivell mundial.

El problema ve a l'hora de comprar un joc nou, doncs a diferència dels videojocs, que et cedeixen les *demos* per a que puguis jugar un temps reduït per saber si t'agradarà, en els jocs de taula no hi ha aquest avantatge. Per tant, acabes comprant jocs que els té un company i ja hi has jugat o bé tenen bona crítica, però t'arrisques a que no sigui del teu estil. Aquí és on entra en joc la pàgina *BoardGameGeek*, una gran base de dades on hi figuren tots els jocs de taula publicats fins a la data i també una grans base de dades per perfils de jugador. Per això ha sortit la idea d'aquest treball, utilitzant les dades extretes de *BoardGameGeek*, implementar un sistema de recomanacions que ajudi a l'hora de comprar un joc, assegurant que així sigui afí a les preferències de l'usuari.

1.2 Objectius

L'objectiu principal del treball és la familiarització amb eines de programació d'alt nivell, l'aprententatge de tècniques i conceptes que rodegen els paradigmes de la web semàntica i *linked data* i també la implementació d'un motor de recomanació, que entraria a dins del camp de *machine learning*.

Un segon objectiu és la creació d'un node de dades enllaçades amb dades de jocs de taula, juntament amb la publicació de la seva ontologia per a la seva modelització.

I com a tercer objectiu és el dur a terme tot el el projecte seguint directius basades

metodologies de desenvolupament àgil com per exemple TDD (*Test-Driven Development*) o BDD (*Behaviour-Driven Development*).

Mitjançant l'acompliment d'aquests objectius, s'assegura l'aprententatge de diferents camps i tecnologies que no estan dins del temari acadèmic i l'assoliment d'un producte sòlid i professional si es segueixen estàndards de l'enginyeria i qualitat del programari.

1.3 Estructura del document

Aquest document s'estructura de la següent manera:

Capítol 2: S'explica l'estat actual del camp de les dades enllaçades així com la explicació dels diferents algorismes de recomanació i com funcionen. També es contextualitza el projecte en el marc de l'aplicació **BGRec** i **prophetik**, els 2 productes creats en aquest treball.

Capítol 3: Aquí s'exposen, juntament amb una breu introducció, tots els passos del desenvolupament dels 2 productes, definint les fites i les diferents tasques que hi havia dins de cadascuna d'aquestes.

Capítol 4: S'hi mostren els comentaris sobre la utilització del sistema i el seu comportament, així com la fiabilitat i precisió de les recomanacions.

Capítol 5: Aquí s'expliquen les diferents conclusions que s'han extret del desenvolupament del sistema, així com l'aprententatge que hi ha hagut a darrera de els diferents eines, metodologies i llenguatges que s'han utilitzat al llarg del treball.També s'hi exposa el treball futur a fer per a millorar el sistema.

Apèndix A: S'expliquen les diferències que hi han hagut treballant amb les dues eines d'automatització Maven i Gradle, juntament amb una breu opinió pròpia de cadascuna d'elles.

Capítol 2

Estat de l'Art

2.1 Linked Data

És un estàndard dins de la W3C, sobre connectar sets de dades utilitzant la Web, que ens descriu una sèrie de mètodes i pràctiques per publicar dades estructurades a la Web.

Les dades enllaçades es construeixen sobre dos estàndards web, l'HTTP i les *URIs*. Les entitats s'identifiquen per una *URI*, i s'obté aquesta referenciant-la a través del protocol HTTP. Aquestes dades enllaçades, es descriuen mitjançant *RDF* (*Resource Description Framework*), que proveeix d'un model genèric per a descriure entitats o recursos.

Per tant, *Linked Data* es basa en 4 principis: Utilitza les *URIs* per identificar models, entitats o recursos. Utilitza HTTP amb les *URIs*, així els usuaris poden obtenir la informació fàcilment. Si algú consulta una *URI*, pot donar més informació d'utilitat utilitzant els estàndards, p.e. *RDF* o *SPARQL*. Inclou enllaços cap a altres *URIs*, així els usuaris poden descobrir noves coses sobre el recurs consultat.

Durant els últims anys, s'ha fet molta recerca i estudis en els camps de la publicació de dades enllaçades, cercar dades enllaçades, consultar i consumir dades enllaçades i navegar per dades enllaçades.

En aquest treball, es treballa sobre un *dataset* de dades enllaçades, bé més concretament de *Linked Open Data* (*LOD*), que és la idea de proporcionar aquestes dades de forma oberta i gratuïta al públic. El moviment de l'*Open Data* ha guanyat po-

pularitat d'ençà del llançament d'iniciatives governamentals i locals d'obrir al públic els seus *datasets*.

La comunitat de *LOD*, anima a estendre la Web publicant dades obertes utilitzant els principis del *Linked Data*. Avui en dia, el núvol de *LOD* ha crescut fins a més de 300 *datasets*, més de 31 mil milions de tripletes *RDF*, entrellaçats amb més de 504 milions d'enllaços *RDF*.

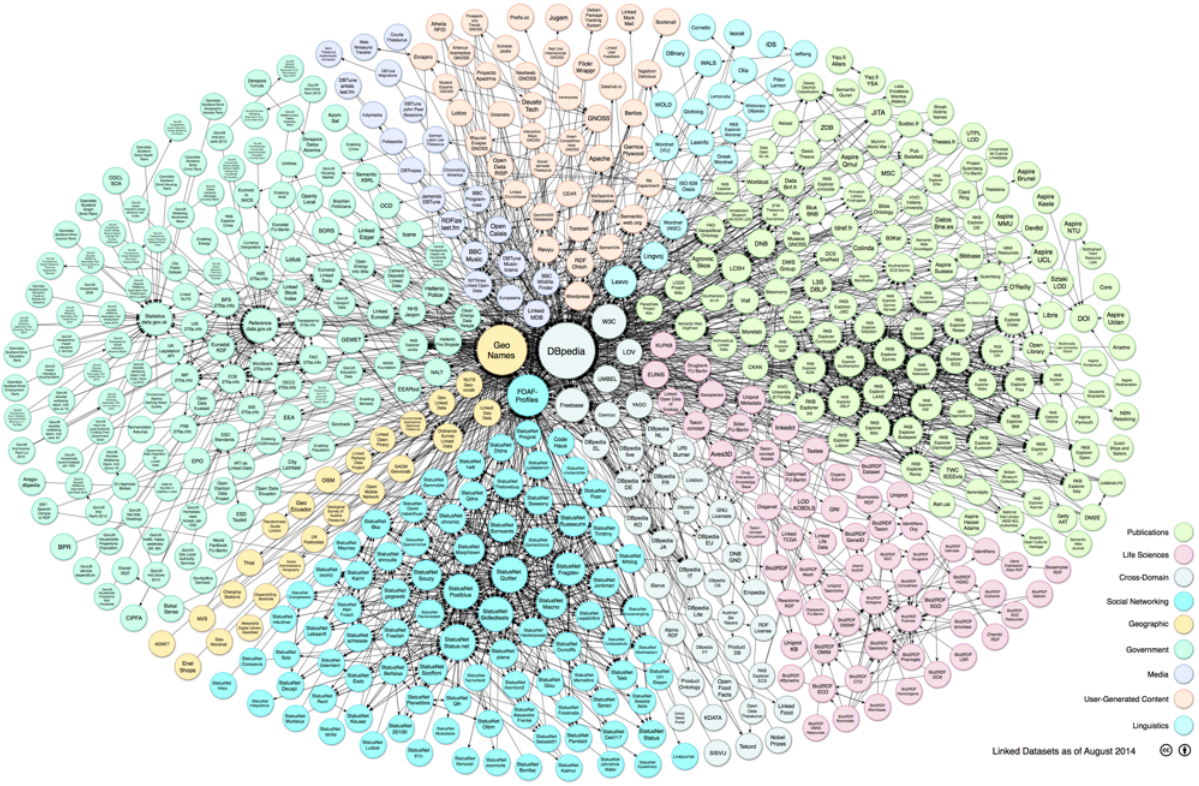

Figura 2.1: LOD Cloud a data d'agost de 2014

L'anterior imatge representa el núvol de *LOD*, on cada node representa un *dataset* publicat en *Linked Data* i on els arcs representen enllaços *RDF* existents entre recursos dels sets de dades connectats.

L'any 2010, Tim Berners-Lee va proposar una escala de 5 estrelles per animar als usuaris i organitzacions a publicar els seus *datasets* com a part del *LOD*.

1 estrella: Les dades estan disponibles en línia, sota una llicència oberta.

2 estrelles: Les dades estan disponibles en un format que un ordinador pot interpretar.

3 estrelles: Les dades estan disponibles (2), però en un format no propietari. Per exemple CSV en comptes d'un arxiu *Excel*.

4 estrelles: Les dades estan disponibles d'acord als punts anteriors, i a més a més també utilitzen estàndards oberts de la W3C per a identificar coses, així els usuaris poden enllaçar a aquestes dades.

5 estrelles: Les dades estan disponibles d'acord als punts anteriors, i a més a més, el *dataset* conté enllaços de sortida cap a altres *datasets* per tal de proporcionar context.

2.2 Sistemes de Recomanació

Al llarg dels anys molts llocs web han utilitzat i utilitzen algorismes de recomanació per tal d'incentivar als seus usuaris a comprar productes d'acord a les seves preferències o a les seves anteriors compres, o bé a recomanar altres productes mitjançant les dades que l'usuari, directament o indirectament, entrava a la pàgina.

Aquests sistemes de recomanació, generalment, comparen el perfil de l'usuari amb algunes característiques de referència dels temes, i així poder predir la puntuació que donaria un usuari a un producte o ítem que els sistema encara no ha considerat. Aquest tipus de sistemes de recomanació reben el nom de filtrat col·laboratiu, que ha estat el tipus d'algorisme que s'ha implementat en aquest projecte, en concret l'anomenat *Nearest Neighbourhood*, que busca els usuaris més propers a les preferències d'aquest mitjançant el càlcul de la correlació *Pearson*, tot i que la distància Euclidiana o Manhattan també serien vàlides.

A l'hora de crear un perfil d'usuari, es poden recol·lectar les característiques de dues maneres diferents, implícitament o explícitament

Explícitament L'usuari insereix les dades de les seves preferències voluntàriament i en una tasca que és per a tal fet. Exemple serien valorar un producte, seleccionar entre vàries opcions o crear una llista de preferències.

Implícitament L'usuari insereix les dades de les seves preferències involuntàriament, és a dir, en una tasca que no ha estat concebuda per a tal fet. Un exemple seria guardar els productes que un usuari a visitat a una tenda en línia, analitzar el nombre

de visites que rep un producte o analitzar les xarxes socials de l'usuari per a obtenir les seves preferències.

En el cas d'aquest projecte, les dades es recol·lecten de forma explícita, tot i que el procés es dóna a la pàgina *BoardGameGeek*.

2.3 BoardGameGeek (BGG)

2.3.1 Descripció

BoardgameGeek és un lloc web fundat l'any 2000 per Scott Alden i Derk Solko com a eina per a la popular afició dels jocs de taula. La seva base de dades emmagatzema revisions, articles, explicacions per als més de 74.000 jocs diferents, expansions i autors. També té una comunitat que supera els 400.000 (a data de 3 d'agost de 2011) usuaris, dels quals 100.000 són actius en el dia a dia. La pàgina també utilitza un sistema de rànquings on els usuaris valoren en una puntuació de 1 al 10 els jocs publicats, i de les puntuacions en calcula la mitjana aritmètica i bayesiana, aquesta última és necessària per acostar els jocs nous o poc coneguts a posicions superiors en el rànquing. En els últims 5 anys només 3 jocs han aconseguit fer-se amb la primera posició de la taula: *Puerto Rico*, *Agricola* i *Twilight Struggle*.

2.3.2 API en XML

BoardGameGeek proporciona un accés obert a les dades de jocs i col·leccions d'usuaris amb el format *XML*, de manera que es poden fer peticions a una *URL* concreta per aconseguir la informació desitjada. Aquest mètode serà l'utilitzat per aconseguir la informació necessària i manipular-la adequadament per a **prophetik** faci les recomanacions per al sistema **BGRec**.

Capítol 3

Desenvolupament del Sistema

En aquest capítol s'exposa tota la informació referent al desenvolupament dels sistemes, és a dir, l'ontologia *BoardGameOntology*, **BGRec Scraper**, **BGRec API**, **prophetik** i **BGRec WebApp**.

Per al desenvolupament de tots ells, s'ha utilitzat el sistema de control de version *git*, creat per Linus Torvalds l'any 2005. En un principi va ser creat per per al control del desenvolupament del *kernel* de Linux, però des de llavors els seu ús anat creixent fins a esdevenir el sistema de control de codi font més estès arreu del món. Aquest èxit també és degut al naixement de plataformes com **GitHub** o **BitBucket**, que permeten emmagatzemar i sincronitzar mitjançant git el codi font escrit al núvol. Per a aquest projecte, s'ha utilitzat la plataforma **GitHub**, per tant hi ha tot el codi disponible al següent perfil:

http://github.com/davidkaste

Pel que fa als entorns de desenvolupament utilitzats, han estat *IntelliJ IDEA 14.1.4* de l'empresa *JetBrains* i *Atom 1.0*, un editor de codi font creat pel mateix equip de **GitHub**. *IntelliJ IDEA* s'ha utilitzat per a implementar tot el codi Java, ja que les diferents utilitats i eines que té fan més fàcil el desenvolupament. *Atom* s'ha utilitzat per a codificar tota la part de *Javascript* de **BGRec WebApp**, ja que té diferents *plugins* disponibles per a la descàrrega que ajuden molt amb el desenvolupament en aquest llenguatge.

Per altra banda, la distribució d'aquest capítol es basa en diferents *milestones* o fites que s'han d'assolir per tal de tenir tota la funcionalitat del sistema implementada. Dins de cada fita, hi figuren les diferents tasques o *sprints* (en referència a

la metodologia àgil *SCRUM*, tot i que no s'hagi seguit) que s'han de realitzar per cada *milestone*, juntament amb una breu explicació dels requisits necessaris per al desenvolupament. Aquestes fites són:

M0 BoardGame Ontology Conté tota l'explicació del disseny de la ontologia

M1 BGRec Scraper Conté tota la informació sobre el desenvolupament del sistema que agafa les dades de *BoardGameGeek* i les converteix a *RDF*.

M2 prophetik S'hi explica tots els passos que s'han seguit per a desenvolupar la llibreria de recomanacions basada en *Linked Data*.

M3 BGRec API Aquí s'explica el desenvolupament de l'*API* REST que consumirà l'aplicació web i els diferent endpoints implementats.

M4 BGRec WebApp S'hi explica el desenvolupament seguit per a implementar tota la funcionalitat de la que disposaran els usuaris a l'aplicació web.

3.1 M0 BoardGame Ontology

3.1.1 Requeriments

Es necessita dissenyar una ontologia en format *OWL* per tal de modelitzar correctament els objectes de joc de taula, usuaris i *ratings* per a poder crear les recomanacions des del sistema a desenvolupar.

L'ontologia es dissenyarà a partir de la ja existent *schema.org*, la qual implementa els esquemes de dades més comuns a la web, així podem estendre els recursos ja existents com *Product*, *Person*, etc. Actualment, *schema.org*, està esponsoritzada per **Google**, **Microsoft**, **Yahoo** i **Yandex**.

3.1.2 M0-S1 Modelització de l'objecte Boardgame

A partir de la ontologia de *schema.org*, es crea una subclasse de *Product* anomenada *BoardGame* amb les propietats següents:

- **Designer**: (URI) Autor del sistema de joc
- **Artist**: (URI) Autor de l'apartat gràfic del joc
- **MaxPlayers**: (Enter) Nombre màxim de jugadors per partida
- **MinPlayers**: (Enter) Nombre mínim de jugadors per partida
- **Expansions**: (URI) Expansions que afegeixen funcionalitats al joc principal
- **Honor**: (String) Premis que ha aconseguit el joc
- **Family**: (String) Família per tipus de joc
- **totalTime**: (Enter) Temps de durada d'una partida en minuts
- **Name**: (String) Nom del joc
- **Review**: (URI) Valoracions dels usuaris
- **AggregateRating**: (Decimal) Puntuació "mitjana"dels usuaris que han valorat
- **Image**: (String (URL)) Imatge de la portada original del joc
- **Descripció**: (String) Breu descripció del joc

- **Editorial**: (URI) Referència a l'editorial que el publica

Com es pot veure, les propietats que són del tipus *URI* apunten cap a uns altres recursos que també estan contemplats a l'ontologia. Aquests recursos són els següents:

Autor (BoardGameDesigner)

- **Nom**: (String) Nom de l'autor

Dissenyador (BoardGameArtist)

- **Nom**: (String) Nom del dissenyador o artista

Expansió (BoardGameExpansion)

- **Author**: (URI) Autor del sistema de joc
- **Designer**: (URI) Autor de l'apartat gràfic del joc
- **ExpansionOf**: (URI) Referència al joc de taula base
- **MinPlayers**: (Enter) Nombre mínim de jugadors per partida
- **MaxPlayers**: (Enter) Nombre màxim de jugadors per partida
- **playingTime**: (Enter) Durada aproximada de la partida, en minuts.
- **Name**: (String) Nom de l'expansió

Jugador (BoardGamePlayer)

- **Username**: (String) Nom d'usuari del jugador
- **Owns**: (URI) Referència als jocs de taula de la seva col·lecció personal

Editorial (BoardGamePublisher)

- **Name**: (String) Nom de l'editorial de jocs de taula
- **Publishes**: (URI) Referència a jocs de taula o expansions que publica. (2x)

3.1.3 M0-S2 Reutilització de propietats de les ontologies existents

Ja que hi ha propietats que definir-les de nou crearia redundància amb propietats ja definides en altres ontologies, es decideix reutilitzar el màxim de propietats ja especificades. Les taules següents indiquen els canvis que s'han fet:

Propietat original	**Substituïda per**
bgo:designer	dc:Creator
bgo:Artist	schema:Artist
bgo:MaximumPlayers	=
bgo:MinimumPlayers	=
bgo:hasExpansion	=
bgo:Honor	=
bgo:family	=
bgo:playingTime	=
bgo:Name	dc:Title
bgo:Review	schema:Review
bgo:Image	schema:image
bgo:Description	schema:description
bgo:Editorial	schema:manufacturer

Taula 3.1: Reutilització de propietats en el recurs **BoardGame**

Classe BoardGameAuthor

Propietat original	**Substituïda per**
bgo:Name	schema:name

Taula 3.2: Reutilització de propietats en el recurs **BoardGameAuthor**

Classe BoardGameArtist

Propietat original	**Substituïda per**
bgo:Name	schema:name

Taula 3.3: Reutilització de propietats en el recurs **BoardGameArtist**

Classe BoardGameExpansion

Propietat original	Substituïda per
bgo:Designer	dc:Creator
bgo:Artist	schema:artist
bgo:MaximumPlayers	=
bgo:MinimumPlayers	=
bgo:expansionOf	=
bgo:Honor	=
bgo:playingTime	=
bgo:Name	dc:Title
bgo:Review	schema:Review
bgo:AggregateRating	schema:AggregateRating
bgo:Image	schema:image
bgo:Description	schema:description
bgo:Editorial	schema:manufacturer

Taula 3.4: Reutilització de propietats en el recurs **BoardGameExpansion**

Classe BoardGamePublisher

Propietat original	Substituïda per
bgo:name	schema:name
bgo:publishes	=

Taula 3.5: Reutilització de propietats en el recurs **BoardGamePublisher**

3.1.4 M0-S3 Canvi de l'ontologia *schema.org* per la mantinguda per *TopBraid*

TopBraid és una de les organitzacions que mantenen l'ontologia *Schema.org* de forma més activa, per tant, és de gran ajuda canviar la importació de l'ontologia original per aquesta més completa. Segons la documentació original, els principals canvis són:

- *schema:Thing* s'ha *mapejat* a *owl:Thing*
- *schema:name* s'ha *mapejat rdfs:label*

- *schema:description* s'ha *mapejat* a *rdfs:comment*
- *schema:url* ha estat eliminada
- Els *datatypes* utilitzats són els de l'estàndard *XSD*
- Els tipus *owl:Class*, *owl:ObjectProperty* i *owl:DatatypeProperty* s'utilitzen al llarg de la ontologia
- Les propietats obsoletes han estat esborrades (p.e. Formes en plural)
- *schema:Class*, *schema:Property* i *schema:additionalType* han estat esborrades
- *rdfs:labels* s'han passat a forma no-*camelcase*
- Propietats que tenen múltiples *schema:domain* inclouen la declaració *owl:unionOf*.
- Propietats que tenen múltiples *schema:range* inclouen la declaració *owl:unionOf*.
- Propietats que admeten tant nombres com cadenes han estat *mapejades* a *xsd:float*
- Propietats que admeten tant cadenes com *URLs* han estat *mapejades* a *xsd:anyURI*.

Després d'aplicar els canvis necessaris per a la correcta importació de l'ontologia, les classes definides queden de la següent forma:

Classe BoardGame

Propietat original	**Substituïda per**
bgo:designer	dc:Creator
bgo:Artist	schema:Artist
bgo:MaximumPlayers	=
bgo:MinimumPlayers	=
bgo:hasExpansion	=
bgo:Honor	=
bgo:family	=
bgo:playingTime	=
bgo:Name	dc:Title
bgo:Review	schema:Review
bgo:Image	schema:image
bgo:Description	rdfs:comment
bgo:Editorial	schema:manufacturer

Taula 3.6: Recurs **BoardGame** després d'importar *TopBraid*

Classe BoardGameAuthor

Propietat original	**Substituïda per**
bgo:Name	schema:name (creada)

Taula 3.7: Recurs **BoardGameAuthor** després d'importar *TopBraid*

Classe BoardGameArtist

Propietat original	**Substituïda per**
bgo:Name	schema:name (creada)

Taula 3.8: Recurs **BoardGameArtist** després d'importar *TopBraid*

Classe BoardGameExpansion

Propietat original	**Substituïda per**
bgo:Designer	dc:Creator
bgo:Artist	schema:artist
bgo:MaximumPlayers	=
bgo:MinimumPlayers	=
bgo:expansionOf	=
bgo:Honor	=
bgo:playingTime	=
bgo:Name	dc:Title
bgo:Review	schema:Review
bgo:Image	schema:image
bgo:Description	rdfs:comment
bgo:Editorial	schema:manufacturer

Taula 3.9: Recurs **BoardGameExpansion** després d'importar *TopBraid*

Classe BoardGamePublisher

Propietat original	Substituïda per
bgo:name	schema:name (creada)
bgo:publishes	=

Taula 3.10: Recurs **BoardGamePublisher** després d'importar *TopBraid*

3.1.5 Observacions

Per a dur a terme aquesta part del treball, s'ha hagut d'aprendre com editar i crear ontologies, així com tota una gamma de nou vocabulari, ja que la forma de crear l'estructura de dades difereix bastant del que ja es coneixia de les bases de dades relacionals. També s'ha aprés a utilitzar l'aplicació *Protégé*, un editor d'ontologies mantingut per la *Stanford Center for Biomedical Informatics Research*, centre adscrit a l'Escola de Medicina de la Universitat de *Stanford*.

La integració de l'ontologia *schema.org* mantinguda per *TopBraid* a la *BoardGame Ontology* no va ser trivial, doncs em vaig trobar amb varis errors ja que hi havia propietats i classes que havien desaparegut, que fins que no es va trobar la documentació oficial, no sabia a que eren deguts.

Gràcies a aquests errors es va poder aprendre noves propietats a les ontologies, com la equivalència entre *datasets* i de la gestió dels errors en les ontologies amb l'eina *Protégé*.

3.2 BGRec Scraper

3.2.1 Requeriments

Es necessita dissenyar una aplicació que pugui accedir a les dades dels usuaris de la *BoardGameGeek*, carregar la seva col·lecció i valoracions i extreure un graf *RDF* basat en l'ontologia descrita en l'anterior capítol.

Per tant, es necessiten dues parts. La primera accedeix a les dades dels usuaris i n'extreu la informació que es necessita i la segona s'ocupa d'agafar aquestes dades extretes i inserir-les a dins d'el graf *RDF*.

Ja que en un futur l'aplicació pot créixer i és possible que es puguin agafar dades de diferents fonts, és necessari de dissenyar l'aplicació de la forma més genèrica possible, fent ús dels coneixements adquirits en l'enginyeria de programari per tal d'evitar modificar codi ja existent.

Tant la part que extreu les dades com la part que genera el graf, han d'estar provades mitjançant tests unitaris i d'integració per tal d'assegurar el seu correcte funcionament.

3.2.2 M1-S1 Desenvolupament del BGGRetriever

Per tal de crear una classe amb la funcionalitat desitjada en els requisits, s'ha creat una interfície *Retriever* amb els següents mètodes:

- **void setDataUri(String dataUri);** Fixa la *URL* del lloc on ha d'agafar informació.
- **String getDataUri();** Obté la *URL* del lloc on està agafant informació actualment.
- **List<Match> query(String value);** Retorna els valors del nom de la propietat que se li passa per paràmetre.

Per tant, la classe que implementi aquesta interfície haurà de implementar aquests mètodes de forma obligatòria. Per tant, es deixa la porta oberta per a possibles ampliacions per a recollir dades d'altres llocs i incorporar-les al sistema.

1a iteració

En aquesta fase s'ha seguit el requeriment especificat anteriorment i s'ha implementat la classe *BGGRetriever* (*BGG* al davant del nom, ja que extreu les dades de la pàgina *BoardGameGeek.com*).

Les llibreries elegides per a la implementació són les següents:

- **Llibreria estàndard de JAVA**
- **org.joox**: Per a consultar dades de dins d'un document *XML* amb una notació semblant a *jQuery*, amb el qual hi estic familiaritzat.
- **com.sun.jersey**: Per a efectuar peticions *HTTP*, és una de les llibreries més utilitzades per a fer aquest tipus d'operacions.

A part dels mètodes parametritzats a la interfície, també s'han implementat els següents mètodes auxiliars:

- **String getBaseUri()**: Obté la URL d'on ha de capturar dades.
- **void setBaseUri(String baseUri)**: Modifica la URL d'on s'han d'obtenir les dades.
- **Document retrieve()**: Mètode encarregat de fer la petició GET a la URL i obtenir el document XML sencer amb les dades.
- **String checkBaseUri(String uri)**: Comprova i modifica la URL per assegurar el correcte funcionament de la classe.

2a iteració

S'ha de canviar el mètode query per a que retorni una List<String> així reduir una dependència de cara a futures implementacions.

3.2.3 M1-S2 Desenvolupament del BGGRDFizer

Un cop estigui funcionant el *BGGRetriever*, necessitem una classe que ens pugui extreure un graf *RDF* de les dades extretes. En el cas del *BGGRDFizer* el seu propòsit es simple; Donada la *URI* del graf, un objecte de la classe *Retriever*, el

prefix de l'espai de noms i la *URI* de l'espai de noms, agafarà tota la informació possible que li retorni el *Retriever* i en generarà el graf *RDF*.

Per tant, per implementar la funcionalitat desitjada en els requisits, s'ha creat la interfície *RDFizer* amb els següents mètodes públics:

- **Model doScrap();** Aquest sol mètode s'encarrega de consultar les dades a l'*scraper* (*BGGRetriever*) i carregar totes les relacions en un graf *RDF*, que serà el mateix que retorna com a resultat.

1a iteració

Segons els requeriments, s'ha implementat la interfície en la classe *BGGRDFizer*, encarregada de generar el graf *RDF* amb les dades dels jocs.

Les llibreries utilitzades per a la seva implementació són:

- **Llibreria estàndard de Java**
- **Apache Jena**: Framework per a fer aplicacions basades en linked data i web semàntica. És la llibreria més utilitzada en aquests contexts.
- **org.joox**: Per a consultar dades de dins d'un document XML amb una notació semblant a jQuery, amb el qual hi estic familiaritzat.

Per al seu correcte funcionament també s'han creat diverses funcions amb visibilitat privada per aconseguir una millor llegibilitat del codi i facilitar la implementació. Aquests funcions són les següents:

- **void createGraph(String URI)**: Inicialitza el graf *RDF* a la *URI* passada per paràmetre.
- **void insertTriple(Resource s, Property p, Literal o)**: Insereix una tripleta al graf
- **void insertTriple(Resource s, Property p, Resource o)**: Sobrecàrrega de l'anterior, en cas de que l'objecte no sigui un literal i faci referència a una altra instància.
- **Model getGraph()**: Obté el graf *RDF*
- **Resource makeResource(String uri)**: Crea una instància de la classe *Resource* partir de la *URI* parametritzada.

- **Property makeProperty(String namespace, String localName)**: Crea una instància de la classe *Property* a partir de l'espai de noms i el nom de la propietat desitjada.
- **Literal makeLiteral(String lit/int lit/double lit)**: Crea una instància de la classe Literal a partir de la cadena passada per paràmetre. (Sobrecarregada)
- **String getBGGUsername()**: Obté el nom d'usuari de la *BGG* a partir de la *URL* fixada al *BGGRetriever*.

2a iteració

Per a evitar confusions, s'ha reanomenat la classe, mitjançant *refactor*, a *BGGUserRDFizer*, ja que extreu les dades d'un sol usuari, i serà una altra classe la encarregada de agafar un llistat d'usuaris i muntar el graf complet.

3a iteració

Aquesta iteració ha estat efectuada després del 3r *sprint* en la ontologia, a l'utilitzar com a ontologia base la mantinguda per *TopBraid*. Per tal de conservar la coherència, s'han hagut de canviar les tripletes generades amb les noves propietats, tal com hi figura a les taules del 3r *sprint* en la ontologia.

4a iteració

Durant el desenvolupament de l'*API* **BGRec**, es va descobrir que donades les limitacions de peticions a la base de dades de *BoardGameGeek* (2 peticions per segon), la importació dels jocs era exageradament lenta, ja que havia de fer una petició per aconseguir la col·lecció de l'usuari i una petició per cada joc, i en usuaris que tenen col·leccions de més de 50 jocs la importació podria trencar la interacció amb el sistema. Per tant, s'ha decidit de eliminar la petició per aconseguir la informació del joc i treballar sobre les dades que ens proporciona la col·lecció de l'usuari.

3.3 Prophetik

Figura 3.1: Logotip de la llibreria **prophetik**

3.3.1 Requeriments

Aquesta és la part on es concentra el major esforç del projecte. Es necessita escriure un motor de recomanació 'genèric' en grafs tripartits, de manera que a partir d'un graf i indicant els recursos d'usuaris, puntuacions i productes, sigui capaç de generar una recomanació acurada. El projecte **prophetik** com a tal, ha de ser una llibreria Java, de manera que per fer-ne ús, pot ser ser importada a través d'un gestor de dependències com pot ser *Maven* o *Gradle*.

El disseny del programa ha de complir els estàndards de qualitat en el programari, per tant, el seu comportament ha d'estar testejat i aplicar els coneixements necessaris d'enginyeria del programari per a dur-lo a terme.

Aquesta part del projecte es continuarà mantenint inclús després de la seva lectura, com a projecte personal, disponible sota llicència de programari lliure (per decidir) i disponible per al públic en general a **GitHub**.

Per què Prophetik?

Com es pot veure, és un mot derivat de profecia, en anglès *prophecy*, donant a entendre que el motor de recomanació que implementa és capaç d'endevinar el futur,

donant recomanacions de productes que la gent no sap que li agraden. Per altra banda i en referència a la terminació *tik* ha estat, senzillament, per inspiració del portal semàntic que mantenen la Rosa Gil i el Roberto García anomenat *Rhizomik*.

3.3.2 M2-S1 Test Driven Development and Behaviour Driven Development

En aquesta part del projecte es vol ser meticulós a l'hora de desenvolupar, doncs és la peça central del projecte, i la part que, en teoria, serà alliberada com a projecte en un futur. En un principi es va pensar de fer el test mitjançant *Cucumber* i avaluant el comportament del recomanador, però donat que l'usuari final no té interacció amb el sistema recomanador, doncs és una llibreria de Java i la interacció la té el programador que l'utilitzi, s'ha decidit d'utilitzar *jUnit* juntament amb una metodologia basada en *TDD*.

Creació del graf de dades de proves amb l'*OpenRefine*

Primer de tot, s'han d'escriure els tests, i per no haver de provar el sistema sobre una versió final de les dades, cosa que implicaria tenir la part del **BGRec** operativa, s'ha capturat l'exemple que ve en el llibre de Toby Segaran anomenat *Programming Collective Intelligence*. En el capítol 2 del llibre explica com fer recomanacions, juntament amb un petit bloc de dades, representant diferents usuaris que han valorat diferents pel·lícules.

El bloc de dades del llibre ve donat en forma de diccionari *Python*, molt semblant a l'estructura d'un objecte *JSON*. A partir d'aquestes dades, s'ha generat un projecte a l'aplicatiu *OpenRefine*, una aplicació web (executada en local) desenvolupada, inicialment, per *Google* i mantinguda per la comunitat *open source*. Aquesta eina permet la neteja i transformació de dades, ja siguin *XML*, *JSON*, *CSV* entre d'altres. També s'ha afegit l'extensió *RDF* desenvolupada per *DERI*, un Institut de recerca en tractament de dades amb seu a Galway, Irlanda, permet el tractament de dades enllaçades. Un cop importades les dades a dins de l'aplicació, podem identificar els diferents camps i poder fer el *mapejat* amb l'ontologia *schema.org* i tenir el graf de dades inicials per començar a escriure els tests. De manera que el graf ha quedat de la següent manera:

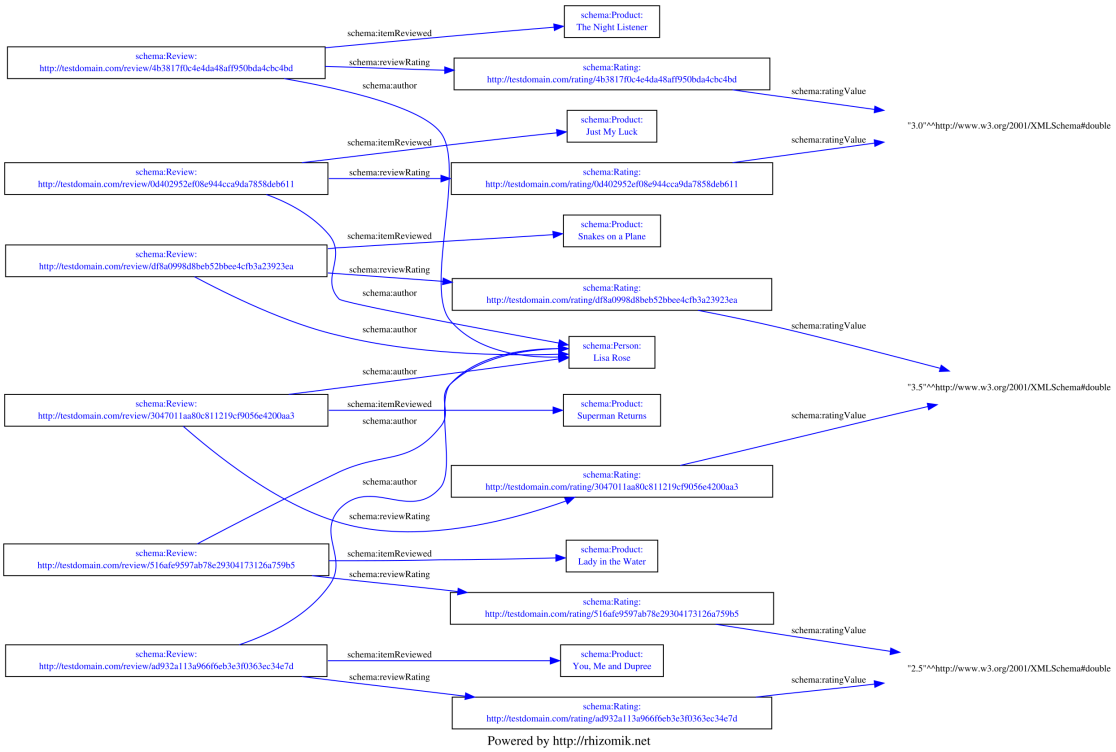

Figura 3.2: Detall de les valoracions de l'usuari Lisa Rose

En la figura 3.2, es pot veure el detall de les valoracions de diferents pel·lícules de l'usuari Lisa Rose. (Generat amb l'aplicatiu ReDeFer[1] de Rhizomik)

Creació de la bateria de tests per a desenvolupar mitjançant *TDD*

Els tests s'han desenvolupat a partir dels exemples del 2n capítol del llibre *Programming Collective Intelligence* de Toby Segaran, on parla dels algorismes de recomanació.

De forma paral·lela al *TDD* també s'ha configurat el llançament de tests automàtics a *TravisCI*, un servidor d'integració continuada, de manera que per cada *commit* enviat al repositori de **GitHub**, aquest executava tota la bateria de tests, i així es podia saber si l'últim canvi que s'havia fet al codi havia trencat la compilació.

[1] Aplicació que ens permet veure el graf en imatge a partir de les dades en *RDF*

L'altra eina externa que s'ha utilitzat, és una eina de mesura de qualitat, en aquest cas el test *coverage*, que mesura el percentatge de línies de codi que han estat executades, com a mínim, per un dels tests. El servei que ho proporciona és *Coveralls.io*, i mitjançant un *plugin* a la configuració de *Maven*, *Travis-CI* executava el llançador per a fer aquesta mesura un cop acabava l'execució de les proves.

La metodologia basada en TDD s'ha portat a terme en 3 blocs que s'expliquen a continuació:

Package UserDistance: Aquest és el bloc que mesura la distància entre usuaris, és a dir, com menys distància hi ha entre dos usuaris, més semblants són les seves preferències i viceversa. S'ha fet el càlcul amb 2 algorismes diferents, per una banda la distància euclidiana i per altra el coeficient de correlació *Pearson*, tot i que per defecte només s'utilitza aquesta última. Cal dir també, que són les mesures que figuren el el llibre que s'ha utilitzat de guia.

Ja que s'està seguint una metodologia basada en *TDD*, s'han escrit els test primer i després s'ha implementat el codi que passi els tests escrits. L'estratègia per escriure el test ve donada per l'exemple del recomanador del llibre, ja que hi figuren els resultats que haurien de donar les funcions com a sortida.

A l'hora d'implementar les classes s'ha creat una interfície anomenada *UserDistance*, que té l'especificació per a les classes que l'implementin, com *EuclideanDistance* i *PearsonCorrelation*. S'ha creat d'aquesta manera ja que si en un futur s'implementa un nou algorisme per a mesurar la distància entre usuaris, la implementació sigui el més senzilla possible.

Les classes implementades es troben sota el *package* de *Java* anomenat *userdistance*.

Abans de fer la implementació s'executen els tests per tal de veure que no hi existeixen errors de programació, i la sortida és la fallada de totes les instruccions assert.

Package Matcher: Aquest bloc, tot i no tenir una funció imprescindible per al motor de recomanació, ens facilita la identificació dels usuaris que tenen una distància menor respecte a l'usuari que vol la recomanació, per tant les seves valoracions tenen més pes a l'hora de fer la recomanació.

La implementació dels tests ve donada pels resultats exposats al llibre *Programming Collective Intelligence* de Toby Segaran, així que sabent la crida que s'havia de fer a la funció, ja tenia el retorn que s'havia de produir. En els tests s'ha acotat el nombre de decimals a comparar per la distància de cada usuari, ja que els exemples del llibre

són en llenguatge *Python* i **prophetik** està implementat en Java, i els 2 llenguatges tenen una aproximació diferent amb els nombres de coma flotant.

Abans de fer la implementació s'executen els tests per tal de veure que no hi existeixen errors de programació, i la sortida és la fallada de totes les instruccions assert.

Package Engine: Aquest és el bloc encarregat de treure les recomanacions per als usuaris, és a dir, l'algorisme principal de **prophetik**. De la mateixa manera que en els blocs anteriors, els tests s'han implementat en base als resultats del llibre.

L'algorisme, donat l'usuari del qual es vol treure la recomanació, busca les seves preferències i les compara amb la resta d'usuaris, donant més valor a les preferències dels usuaris que tenen menys distància, euclidiana o correlació *Pearson*, amb l'usuari en qüestió. De la mateixa manera que en bloc anteriors, s'han acotat la mida dels decimals per facilitar la comparació.

Abans de fer la implementació s'executen els tests per tal de veure que no hi existeixen errors de programació, i la sortida és la fallada de totes les instruccions assert.

3.3.3 M2-S2 Desenvolupament a partir dels tests

Un cop els tests estaven escrits, s'ha continuat per la implementació del codi. D'aquesta manera es pot focalitzar més en la funcionalitat que ha de desenvolupar, ja que s'implementa només el codi necessari per a passar les proves.

Package UserDistance

Aquest *package* conté 2 tests, un per l'algorisme de càlcul de la distància euclidiana i l'altre per correlació *Pearson*, ambdós per calcular la distància entre usuaris, que és la principal funció d'aquest paquet.

Distància euclidiana: La distància euclidiana s'utilitza, normalment, per fer el càlcul de la distància entre 2 punts en un pla. Si s'imagina als usuaris com a punts en un pla i la seva posició ve donada per un coeficient en base a les seves preferències o valoracions, aquest tipus de càlcul és perfectament aplicable per trobar quin usuaris són els més semblants entre ells. El pseudocodi de l'algorisme queda de la següent manera:

```
donat user1 i user2
llistatPelis = obtenirPelisIValoracions(user1)
```

```
llistatPelis2 = obtenirPelisIValoracions(user2)

per cada peli de user1:
  si user2 tambe la te:
    quadrats=quadrats+alQuadrat(user1.peli.punts-user2.peli.punts)

si no hi ha coincidencies:
  retornem zero

retornem 1/(1 + quadrats)
```

D'aquesta manera la distància entre usuaris queda expressada en un nombre de coma flotant entre 0 i 1, on l'u ens indica que dues persones són completament idèntiques i el 0 que són totalment diferents.

Un cop fet el desenvolupament s'executen els tests per comprovar que els resultats són els esperats.

Correlació *Pearson*: El coeficient de correlació *Pearson* és un índex que indica la relació lineal entre dues variables quantitatives. Si es representen els usuaris com a variables quantitatives en base a les seves preferències, es pot utilitzar aquest índex per a identificar la distància entre ells.

El pseudocodi de l'algorisme queda de la següent forma:

```
donat user1 i user2
llistatPelis = obtenirItemsIValoracions(user1)
llistatPelis2 = obtenirItemsIValoracions(user2)

per cada item de user1:
  si user2 tambe el te:
    sumaUser1 += user1.item.punts
    sumaUser2 += user2.item.punts
    sumaQuadrat1 += alQuadrat(user1.item.punts)
    sumaQuadrat2 += alQuadrat(user2.item.punts)
    sumaT += user1.item.punts * user2.item.punts

si no hi ha coincidencies:
  retornem zero

num=sumaT-((sumaUser1*sumaUser2) / n_coincidencies)
den=alQuadrat((sumaQuadrat1-alQuadrat(sumaUser1)/n_coincidencies) *
   (sumaQuadrat2-alQuadrat(sumaUser2)/n_coincidencies))

si den es igual a 0:
    retorna 0
```

```
retorna num / den
```

On *num* és la covariància (mesura de dispersió conjunta de dues variables estadístiques) i *den* és el producte de les seves desviacions estàndard. A diferència de la distància euclidiana, dóna un valor entre -1 i 1, on l'[1] indica que dues persones són completament idèntiques i el 0 que són totalment diferents.

Degut a que aquest és el càlcul més acurat dels 2, s'ha configurat per defecte del motor de recomanació. La principal diferència entre la correlació *Pearson* i la distància euclidiana és que la correlació és independent de la unitat, és a dir, més que tenir una distància entre els 2 usuaris, obtenim el valor de com canvien les variables conjuntament dividides per la seva variació de forma individual.

Un cop fet el desenvolupament s'executen els tests per comprovar que els resultats són els esperats.

Package Matcher

Com ja s'ha explicat en el punt anterior, aquest *package* no conté codi imprescindible per al normal funcionament del recomanador, tot i que la funcionalitat que té és útil per saber quin són els n usuaris més propers, és a dir, els que tenen més pes a l'hora de fer recomanacions. L'algorisme que segueix és el següent:

```
donat usuari1, llistatUsuaris, nombreResultats i metDistancia
per cada user de llistatUsuaris:
  dist = metDistancia.calc(usuari1, user)
  similaritat.afegir(dist, user)

ordenar similaritat desc

cont = 0
mentre cont < nombreResultats:
  mapaRet.afegir(similaritat[cont])
  cont++

retorna mapaRet
```

On *metDistancia* pot ser un objecte del *package UserDistance*. L'execució ens retorna una llista *hash* amb els *n* (*nombreResultats*) usuaris més propers a *usuari1*.

En el pseudocodi es perden detalls, com per exemple la ordenació de la taula *hash similaritat* pel valor de la distància, cosa que inicialment no és possible. I a més hi

ha la necessitat de treure la taula *hash* ordenada per usuari més proper. Per tant, a la vegada que s'insereix una parella de clau i valor a la taula *hash similaritat*, també s'afegeix el valor en una llista. D'aquesta manera, si que es pot ordenar l'*array* segons el valor, per a després generar una nova taula *hash* segons la distribució de valors de la llista.

Un cop fet el desenvolupament s'executen els tests per comprovar que els resultats són els esperats.

Package Engine

Aquest és el paquet que s'encarrega de fer ús d'altres classes del projecte i les dades, per a treure recomanacions a partir de les preferències de l'usuari. El funcionament no és cap altre que esbrinar quins són els usuaris amb els quals existeixen coincidències, i veure quins són els ítems que s'han valorat. A partir d'aquesta informació es pot esbrinar quins són els productes o objectes que l'usuari en qüestió no ha valorat i que segons els usuaris més propers, li podria agradar. L'algorisme de recomanació queda de la següent manera:

```
donat dades, usuari, metDist // metDist=Medtode de distancia
llistaUsers=dades.getLlistaUsers();

per casa user de llistaUsers:
  dist = metDist.calc(usuari, user)
  si dist mes gran que 0:
    per cada item de user2:
      si usuari no te item o usuari.item.punts == 0:
        totalHash.add(item, totalHash.item+(user.item.punts*dist))
        distSumH.add(item, distSumH.item + dist)

per cada item de totalHash:
  tvalor = totalHash.item
  svalor = distSumH.item
  rank = tvalor / svalor
  rankingH.add(rank, item)

ordena rankingH desc

retorna rankingH
```

De la mateixa manera que en el paquet *Matcher* s'havia d'ordenar la taula *hash*, també s'han d'ordenar els resultat de les recomanacions segons el producte més

pròxim a les preferències de l'usuari. Doncs s'ha creat una llista auxiliar per a poder ordenar els resultats de les recomanacions i així poder utilitzar la llista com a índex per a generar la sortida.

Un cop executat aquest algorisme, es tindrà una llista *hash* amb els ítems a recomanar a l'usuari, juntament amb el seu valor obtingut, ordenat de major a menor. Així la primera entrada de la llista serà el producte amb més possibilitats de ser del gust de l'usuari.

Codi complementari

A part del codi generat per a portar a terme la funcionalitat principal de **prophetik**, s'ha programat més codi per tal de suplir les necessitats paral·leles que sorgien.

Classe Queries: Aquesta classe ve a ser un repositori de consultes *SPARQL* (ja que les dades es tenen en format *RDF*) per tal d'aconseguir les dades que necessita l'algoritme. Són funcions que retornen una cadena de text (*java.lang.String*) que conté la dita consulta.

Mètode getUserProductsRating(String userResource); Donada la *URI* d'un usuari, la consulta obté la valoració de tots els productes que ha puntuat:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schema: <http://schema.org/>

SELECT ?product ?ratingValue WHERE {
  ?review schema:author <userResourceURI>.
  ?review schema:itemReviewed ?product.
  ?review schema:reviewRating ?rating.
  ?rating schema:ratingValue ?ratingValue
}
```

Mètode getAllUsers(String userResource); Donada la *URI* d'un usuari, la consulta obté tots els recursos de tipus usuari (*schema:Person*) que hi ha al graf, a excepció d'ell mateix:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX schema: <http://schema.org/>

SELECT ?user WHERE {
  ?user rdf:type schema:Person
  FILTER(str(?user) != "<userResourceURI>")
}
```

On, tant a *getAllUsers(...)* i *getUserProductsRating(...)*, *<userResourceURI>* és el paràmetre passat a la funció.

Classe Querier: Aquesta es la classe que utilitza la classe Queries. S'ocupa d'agafar la consulta que retorna, per executar-la al graf de dades que hi ha. Per cada consulta de la classe Queries, hi ha la funció equivalent a aquesta classe. D'aquesta manera es pot canviar la consulta sense haver de tocar el codi que resol contra el graf.

El resultat que retornen aquestes funcions són taules *hash* de la llibreria estàndard de Java (*java.util.HashMap*), de manera que per al posterior tractament de les dades quedi completament transparent de que utilitzem un graf de *Linked Data*, enlloc de les taules convencionals de bases de dades. De fet, l'estructura que s'ha elegit per a la llibreria permet que si en un futur es vol consultar a una base de dades *SQL*, canviant l'accés i les consultes, et pot tenir la funcionalitat completa sense haver de retocar el codi que ja ha estat escrit.

Classe Prophetik: Aquesta és la classe que permet interactuar amb el nucli de la llibreria, s'ocupa de crear les instàncies del motor de recomanació, algorismes de càlcul de distàncies i el *matcher*. A part d'això, s'ocupa també de la gestió del graf de dades, tal com la seva inicialització, com la seva ampliació amb noves dades que es passin.

3.3.4 M2-S3 Publicació de la llibreria (Repositori Maven)

Per a fer les proves en local, només calia importar la llibreria des del repositori local de *Maven*. Un cop feta la configuració amb el servidor d'integració Travis-CI, al no tenir la llibreria pujada a cap servidor, els tests de l'**API BGRec** fallaven.

Primer de tot es necessita generar el *.jar* de **prophetik**, i aquesta acció es fa mitjançant *Maven*, ja que no només ens ajuda en el tema d'injecció de dependències, si no que també ens proporciona una sèrie de tasques o *goals* per a passar tests, generar paquets etc. Per tant amb la comanda *mvn package*, ens neteja el projecte, genera les classes, passa els tests i per últim en genera el paquet *.jar* que es necessita.

Per tant, es necessitava pujar la llibreria a un repositori *Maven* remot per tal de que *Travis-CI* pogués descarregar la llibreria i no llancés l'excepció per no haver-la trobat. Al final més que pujar *prophetik* al repositori oficial d'*Apache*, es va decidir utilitzar el servei **Bintray**, per una qüestió de rapidesa i facilitat de configuració. **Bintray** és un servei d'emmagatzemament de paquets/binaris i centre de descàrrega, per a

programadors (gratuït si els projecte es *open source*), que proporciona repositoris de *Maven*, *apt*, *rpm*, etc. per a la distribució del software.

Un cop penjat el paquet *.jar* de **prophetik** (*prophetik-0.2-BETA.jar*), sota el directori *com.davidcastella*, només queda afegir la dependència a l'arxiu *build.gradle* a l'arrel del projecte **BGRec**.

Per a tal fet, hem d'afegir els repositoris a on resoldrà *Gradle* per baixar-se les dependències:

```
repositories {
    jcenter()
    maven {
        url  "http://dl.bintray.com/davidkaste/maven"
    }
}
```

A la configuració de les dependències, afegim el repositori general *jcenter()*, que es un *mirror* de l'oficial *MavenCentral* mantingut per *JFrog*, els mateixos desenvolupadors de **Bintray**. I a continuació, afegir les dades del repositori a on tenim els binaris per a la seva distribució.

Un cop configurats els repositoris ja es podrà afegir la dependència, afegint la següent línia al grup de compilació:

```
dependencies {
  testCompile group: 'junit', name: 'junit', version: '4.11'
  [...]
  compile group:'com.davidcastella',name:'prophetik',version:'0.2-
     BETA'
  [...]
}
```

3.4 BGRec API

Un cop acabat el projecte **prophetik**, es necessita una aplicació que faci ús d'ella amb el graf generat de jocs de taula. Per a tal fet, es necessita que l'aplicació tingui 3 funcionalitats bàsiques:

- Obtenir la col·lecció de jocs d'un usuari
- Obtenir una recomanació de jocs
- Obtenir quins són els usuaris més semblants

Donada la poca disponibilitat de temps i la seva senzillesa, s'ha decidit utilitzar el *micro framework* de Java, **Spark**. Inspirat en altres *frameworks* semblants com *Flask* (*Python*), *Sílex* (*PHP*) o *Sinatra* (*Ruby*), és la implementació en pur Java per a generar una aplicació web amb el mínim esforç, sense haver de fer configuracions en *XML* com en *SpringFramework* o bé anotacions com en *JAX-RS*.

Figura 3.3: Logotip del *framework* **Spark**

Aquestes 3 funcionalitats que ha de tenir l'aplicació seran 3 *endpoints* per a fer peticions *GET*, és a dir, una *API REST* (no *RESTFul*, ja que no hi ha la funcionalitat per alterar la informació del graph, i per tant les peticions *PUT*, *POST*, *DELETE* d'HTTP no s'executen). Els endpoints s'han dissenyat de la següent manera:

- **GET /recommendation username: <username>**: Obté una llista d'objectes *JSON* representant la recomanació dels jocs de taula, amb el nom, la *URI*, la puntuació de l'usuari i dues imatges, la imatge de la portada i el *thumbnail*.
- **GET /collection username: <username>**: Obté la llista de jocs que té l'usuari, en una llista d'objectes *JSON*, amb la mateixa informació.

- **GET /similarity username: <username>**: Obté una llista d'usuaris en format *JSON* que conté els usuaris més semblants en gustos a l'usuari passat per paràmetre, és a dir, de quins usuaris les valoracions tenen més pes.

3.4.1 M3-S1 Preparació per l'incorporació de Spark al projecte BGRec

En aquest punt, al projecte **BGRec** només hi ha la lògica de la transformació a *RDF* de les dades obtingudes de la pàgina *boardgamegeek.com*. Aquesta lògica s'allotja al paquet *scraper* del projecte, per tant s'ha creat un nou paquet anomenat api on s'allotjarà tota la lògica en referència al servidor.

El paquet api s'ha dividit en 2 subpaquets, server i models, on el primer allotja la classe que s'encarrega d'engegar el servidor i aplicar la funcionalitat als endpoints descrits a l'anterior apartat, i el 2n allotja els models a utilitzar, per facilitar la gestió de les llistes de jocs de taula.

Un cop feta aquesta distribució de paquets, s'ha d'incorporar les dependències necessàries per al *framework* **Spark**. Per tant, a l'arxiu de configuració de *Gradle* s'ha d'escriure la línia que ens proporcionarà les eines per utilitzar-lo. De forma paral·lela també s'han d'incloure les dependències necessàries per a poder retornar les respostes a les crides de l'*API* en format *JSON*, que en aquest cas s'ha utilitzat la llibreria *GSON* de *Google*. L'arxiu *build.gradle*, quedarà de la següent manera:

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'jetty'
apply plugin: 'idea'
sourceCompatibility = 1.8
version = '1.0'

repositories {
   jcenter()
   maven {
       url  "http://dl.bintray.com/davidkaste/maven" // Prophetik
          repo
   }
}

dependencies {
testCompile group: 'junit', name: 'junit', version: '4.11'
```

```
testCompile group: 'org.mockito', name: 'mockito-core', version:
    '2.0.3-beta'
compile group: 'org.apache.jena', name: 'jena-core', version:
    '2.13.0'
compile group: 'org.apache.jena', name: 'apache-jena-libs', version:
     '2.13.0'
compile group: 'com.sun.jersey', name: 'jersey-core', version:
    '1.18.1'
compile group: 'com.sun.jersey', name: 'jersey-client', version:
    '1.18.1'
compile group: 'org.jooq', name: 'joox', version: '1.2.0'
compile group: 'com.davidcastella', name: 'prophetik', version:
    '0.2-BETA'
compile group: 'com.sparkjava', name: 'spark-core', version: '2.2'
compile group: 'com.google.code.gson', name: 'gson', version:
    '2.3.1'
}
```

3.4.2 M3-S2 Implementació del model BoardGame

Per tal de facilitar la gestió dels jocs de taula, s'implementa un model per representar els objectes *boardgame* per evitar, així, haver de gestionar taules *hash* i llistes de forma nativa. El model no és res més que una classe amb els atributs *URI*, nom del joc, valoració, imatge i *thumbnail*, juntament amb un mètode anomenat *toJson()* que ens retornarà una taula *hash* amb el següent format:

```
{
  'uri': 'http://...',
  'name': 'Nom␣del␣joc',
  'ratingValue': 0.0,
  'image': 'http://.../....jpg',
  'thumbnail': 'http://.../..._t.jpg'
}
```

Llavors, si no es vol haver de manejar taules de *hash*, com és que en retorna una? Doncs gràcies a la llibreria *GSon*, es pot convertir aquesta estructura de dades a un objecte *JSON* de forma automàtica.

Per tal d'augmentar la llegibilitat del codi i la facilitat per crear un objecte *BoardGame*, s'ha implementat una classe *builder*, de manera que crear un objecte d'aquesta classe quedaria de la següent manera, en codi Java:

```
BoardGame bg = new BoardGameBuilder().setUri("uri␣del␣joc")
    .setName("nom␣del␣joc")
    .setRatingValue(0.0) // puntuacio
    .setThumbnailUri("uri␣del␣thumbnail")
    .setImageUri("uri␣de␣la␣imatge")
    .build();
```

Un cop implementades aquestes classes, els endpoints de l'*API* de **BGRec** poden retornar una llista d'objectes *BoardGame*, ja que la llibreria *GSon* s'encarregarà de fer la traducció a *JSON*. Per tant, ja es poden començar a escriure les rutes de l'*API*.

3.4.3 M3-S3 Implementació del servidor amb Spark

Per tal de concentrar aquesta funcionalitat, s'ha creat la classe *BGRecServer* que s'ocuparà de la gestió de les *URLs* i la lògica relacionada amb aquestes. Per a tal fet, es fa ús de la llibreria **Spark**, que mitjançant l'*import* següent proporciona tota la seva funcionalitat:

```
import static spark.Spark.*;
```

Un cop fet això, ja hi ha disponibles les funcions que es necessiten, que en aquest cas són les següents:

```
port(int portNum);
```

Aquesta ordre serà la que engega un servidor *Jetty* que ve incorporat al *framework*, que es quedarà escoltant peticions al número de port indicat com a paràmetre. En aquest cas al port 8080.

```
before(callback(request, response));
```

S'executa aquest *callback* per cada petició que rebi el servidor *Jetty*, s'ha utilitzat per afegir la capçalera *CORS*[2] a cada petició.

```
exception(Exception.class, callback(request, response));
```

Executa el *callback* quan es produeix una excepció en una petició al servidor, s'ha utilitzat per a gestionar els errors 404.

[2]Cross-Origin Resource Sharing; Capçalera per a permetre peticions des d'un host o servei diferent del servidor.

```
get("/ruta", callback(request, response));
```

S'executa el *callback* quan es produeix un petició get contra la ruta que li passem com a paràmetre. S'utilitza per a gestionar les rutes */similarity*, */recommend* i */collection*.

```
options("/ruta", callback(request, response));
```

Executa la petició *OPTIONS* per a la ruta especificada, en aquest cas per a la ruta */**, és a dir, totes, per a la inserció de la capçalera *CORS*.

Inicialització

Primer de tot es defineixen 2 atributs per a la classe, un objecte Model de la llibreria *Jena*, per representar el graf sobre el que es treballa i un objecte **prophetik** que proporciona les eines per a obtenir les recomanacions.

A continuació es defineix el mètode public *startServer()*, on les primeres accions que executa són: Creació de l'objecte de la classe *GSon* per a traduir el retorn de les crides a *JSON*. Inicialització de l'objecte *p* de la classe **prophetik**, amb el càlcul de distància d'usuaris per correlació *Pearson*.

Creació d'una llista d'usuaris çoneguts"de la pàgina *BoardGameGeek*, i obtenció de les seves col·leccions per afegir-les tant al graf propi de la classe com al graf del recomanador. Aquesta acció es fa per evitar el possible *cold start* a l'hora de fer recomanacions.

Engegada del servidor mitjançant la línia *port(8080);* i definició dels recursos de l'*API* que s'expliquen a continuació

GET /collection?username=[...]

Aquest recurs és l'encarregat de retornar tota la col·lecció de jocs de l'usuari, en format *JSON*, és a dir una llista de d'objectes *BoardGame*.

El primer que s'ha de fer és capturar el nom d'usuari dels paràmetres, mitjançant el mètode *queryMap()* de l'objecte *response* i indicar que la resposta serà del tipus *application/json* amb la codificació de caràcters *UTF-8*.

A continuació es necessita afegir al graf les dades de la col·lecció de l'usuari, per tant, mitjançant un objecte *BGGUserRDFizer*, generem el graf de la col·lecció i l'afegim

al graf ja existent, tant propi com del recomanador. Aquesta acció està delegada a una funció anomenada *extendGraph(String username);*.

Per últim es construeix la *URI* de l'usuari i es passa com a paràmetre a la funció auxiliar *getUserCollection(String userUri);* que s'encarrega de consultar al graf les dades de l'usuari i la seva col·lecció i retornar una llista d'objectes *BoardGame*. Aquesta llista es *mapeja* amb el mètode *gson::toJson* utilitzant el *ResponseTransformer* d'**Spark**.

La funció quedaria amb el següent aspecte:

```
get("/collection", (req, res) -> {
    res.type("application/json, charset=UTF-8");
    String username = req.queryMap().get("username").value();
    extendGraph(username);
    return getUserCollection("http://davidcastella.com/user/" +
       username);
}, gson::toJson);
```

GET /similarity?username=[...]

Aquest recurs s'encarrega de retornar la llista d'usuaris més semblants a l'usuari passat per paràmetre, segons l'algorisme utilitzat (distància euclidiana o correlació *Pearson*) en el recomanador.

Tal com es fa en la resta de recursos, primer s'indica a la resposta que serà en format *application/json* i en la codificació *UTF-8* i a continuació es captura el nom d'usuari dels paràmetres. La funció retorna el mateix retorn de la funció del mètode *getMostSimilarUsers(String userUri)* de l'objecte de la classe **prophetik**, que obté una llista dels usuaris més semblants que es *mapejarà* amb el mètode *gson::toJson*.

La funcionalitat s'implementa de la següent manera:

```
get("/similarity", (req, res) -> {
    res.type("application/json, charset=UTF-8");
    String username = req.queryMap().get("username").value();
    return p.getMostSimilarUsers("http://davidcastella.com/user/" +
       username, 5);
}, gson::toJson);
```

GET /recommend?username=[...]

Aquest és el recurs encarregat de retornar la llista amb la recomanació de jocs per a l'usuari passat per paràmetre, tot executant la funció *recommend(String userUri)* de l'objecte *p* de la classe **prophetik**.

Tal com es fa en la resta de recursos, primer s'indica a la resposta que serà en format *application/json* i en la codificació *UTF-8* i a continuació es captura al nom d'usuari dels paràmetres.

A continuació es captura el retorn de la funció r*ecommend(...)* del recomanador, que ens retorna una taula *hash* amb la *URI* del joc i la puntuació estimada, que en un cas d'estar buida, cosa que representaria que l'usuari no existeix o no té jocs a la col·lecció, es retorna un error 404, llançant una excepció de la classe *NotFoundException*, cosa que activarà el *callback* definit anteriorment.

Un cop s'assegura que la llista de recomanacions no és buida, s'envia com a paràmetre a la funció *getNRanking(Map<String, Double> recommendations, int number)*. Aquesta funció s'ocupa d'ordenar els jocs per la puntuació, de major a menor, i generar una llista d'*n* posicions (paràmetre *number*) amb els objectes *BoardGame* més afins a les preferències de l'usuari.

Ja que el recomanador és genèric, només ens retorna la *URI* i la valoració del producte, per tant s'han hagut d'implementar una sèrie de funcions auxiliars per aconseguir les dades que falten. Aquestes funcions són les següents:

```
String getBoardGameName(String gameUri);
String getBoardGameThumbnail(String gameUri);
String getBoardGameImage(String gameUri);
```

Aquestes funcions consulten al graf de dades per la propietat desitjada (nom, icona o imatge) mitjançant una consulta *SPARQL*. Donat que si diferents usuaris importen la col·lecció i tenen el mateix joc de taula en versions diferents, principalment l'idioma, la consulta ens retorna el diferents títols del joc. També cal dir que no es té la informació de l'idioma del títol, ja que l'*API* de la pàgina *BoardGameGeek* no la proporciona, per tant s'ha decidit de retornar la primera coincidència trobada, tant en el nom del joc com en la icona i la imatge. Aquest petit handicap provoca que a l'hora d'importar la col·lecció es trobi que hi hagin jocs que la portada i el nom no coincideixin amb la versió que realment es té, tot i que representi al mateix objecte.

La implementació del recurs per a obtenir les recomanacions queda de la següent

manera:

```
get("/recommend", (req, res) -> {
  res.type("application/json, charset=UTF-8");
  String username = req.queryMap().get("username").value();
  Map<String, Double> ret = p.recommend("http://davidcastella.com/
     user/"+username);
  if(ret.isEmpty()) throw new NotFoundException("404 error");
  return getNRanking(ret, 10);
}, gson::toJson);
```

3.4.4 M3-S4 Definició de la classe Main i proves

Un cop els sistema de l'*API BGRec* s'ha implementat, s'ha d'automatitzar l'engegada del servidor, per això es defineix una classe Main amb el mètode *public static void main(String[] args)*, que serà el punt d'entrada a l'execució del sistema. La principal funció d'aquest mètode és cridar al mètode *startServer()* de la classe *BGRecServer*.

La classe quedaria definida de la següent manera:

```
public class Main {
    public static void main(String[] args) {
        new BGRecServer().startServer();
    }
}
```

Per provar el sistema de peticions *GET*, s'ha utilitzat l'eina *Test RESTful Web Service* integrat en l'entorn de desenvolupament *IntelliJ IDEA*. En aquest punt s'ha comprovat l'exagerat temps de càrrega per a obtenir la col·lecció d'un usuari, cas degut a les diferents peticions a l'*XMLAPI*, concretament $n + 1$ peticions, on n és el nombre de jocs de taula de l'usuari, i el $+1$ per aconseguir la llista de jocs de la col·lecció. Tenint en compte que només es poden fer 2 peticions per segon, l'obtenció d'una col·lecció d'un usuari que posseeix sobre uns 120 jocs (cas habitual en els usuari de *BoardGameGeek*), es podia estar més de 60 segons.

Aquest problema ha estat el detonant per a obrir la 4a iteració en el projecte *Scraper* de **BGRec**.

3.5 BGRec WebApp

Figura 3.4: Logotip de l'aplicació web BGRec

3.5.1 Requeriments

Un cop acabada tota la lògica del *backend* de l'aplicació (**BGRec API** i **prophetik**), s'ha d'implementar tota la lògica del *frontend*, és a dir, la part on l'usuari tindrà la interacció amb tot el sistema.

Aquest *frontend* ha de ser una web senzilla d'utilitzar, on l'usuari pugui importar la seva col·lecció de la pàgina *BoardGameGeek*, visualitzar-la i obtenir la recomanació en base a aquesta. L'aplicació també ha de mostrar els usuaris més semblants al seu perfil, de manera que també es pugui navegar per la seva col·lecció i veure la recomanació que els sistema els hi faria a ells.

Després d'avaluar diferents eines i *frameworks*, i fer una mica de recerca per Internet, es descarta generar una aplicació en *Javascript* (*AngularJS* o *JQuery*) des de 0, ja que és una tasca feixuga i llarga. Per tant, es decideix utilitzar l'eina *Yeoman* juntament amb el generador d'*AngularJS*. Aquesta eina ens permet crear tot un esquelet i arbre de fitxers per a crear una aplicació web amb *GruntJS task runner*, *Karma testing suite*, *Bower* i *AngularJS*.

Figura 3.5: Logotips de les tecnologies utilitzades per a desenvolupar l'aplicació web

Yeoman

És una eina de tipus *CLI* (interfície de línia de comandes) per a generar l'esquelet i l'arbre de fitxers d'una aplicació completament funcional. A més, mitjançant el generador d'*AngularJS*, ens permet generar automàticament els controladors, serveis, directives o vistes que es necessitin. Per exemple:

```
...%$ yo angular app-name
```

Per a generar una aplicació d'AngularJS

```
...$ yo generate:controller controller-name
```

Per a generar un controlador

GruntJS

Aquesta aplicació, que també és del tipus *CLI*, és un *task-runner* per a *Javascript*, és a dir, ens ajuda a aplicar diferents tasques sobre el codi del projecte, entre les quals poden ser passar els tests, la comprovació d'errors en el codi, la concatenació dels diferents scripts de *Javascript* i la minificació d'aquests. La configuració del seu comportament ve donada per l'arxiu *Gruntfile.js*, que en aquest cas, s'ha utilitzat la configuració per defecte que proporciona el *Yeoman*. Per utilitzar-lo es poden introduir comandes com les següents:

```
...$ grunt build
```

Genera l'aplicació web amb tots els fitxers necessaris.

```
...$ grunt test
```

Executa els tests definits a l'aplicació del *framework Karma*.

```
...$ grunt serve
```

Alça un servidor web per a poder navegar per l'aplicació.

```
...$ grunt
```

Executa els tests i genera l'aplicació.

Bower

Aquesta eina ens permet gestionar les dependències del *frontend*, és a dir, tots aquells scripts que permeten que s'executi l'aplicació implementada. Tal i com les anteriors, també és una aplicació *CLI*. Les dependències estan definides en l'arxiu *bower.json*, que s'ha utilitzat l'arxiu per defecte que proporciona *Yeoman*. S'utilitza de la següent manera:

```
...$ bower install
```

Llegeix l'arxiu bower.json i instal·la les dependències que hi figuren.

```
...$ bower install dependency-name --save
```

Descarrega la dependència indicada i guardaria la referència a l'arxiu bower.json.

AngularJS

És un *framework Javascript*, creat per *Google* que ens permet crear aplicacions de tipus *single-page*[3] utilitzant el patró model-vista-controlador a la banda client, és a dir, al navegador. *AngularJS* proporciona diferents directives d'*HTML* per tal de connectar els controladors amb les vistes així com diferents eines per a la manipulació del *DOM*, juntament amb la llibreria *JQuery*.

[3]És tipus aplicació web que cap en una sola pàgina a fi de donar una experiència més fluida als usuaris

Karma

Karma ens proporciona un entorn de test a l'aplicació, però no és un *framework* de *testing* en si, si no que el que fa es l'automatització de les diferents bateries de test, que al ser *framework agnostic*, poden estar escrites en qualsevol de les diferents eines de testeig que existeixen per a *Javascript* (*QUnit*, *Mocha*, *Jasmine*, etc.). En el cas de B**GRec WebApp**, s'utilitza *Jasmine*, que és el que es configura per defecte amb *Yeoman*. La configuració de l'entorn de *Karma* resideix a l'arxiu *karma.conf.js*, on hi figuren els diferents arxius on hi ha els tests i les dependències necessàries per executar-los.

3.5.2 M4-S1 Controlador del formulari i *landing page*

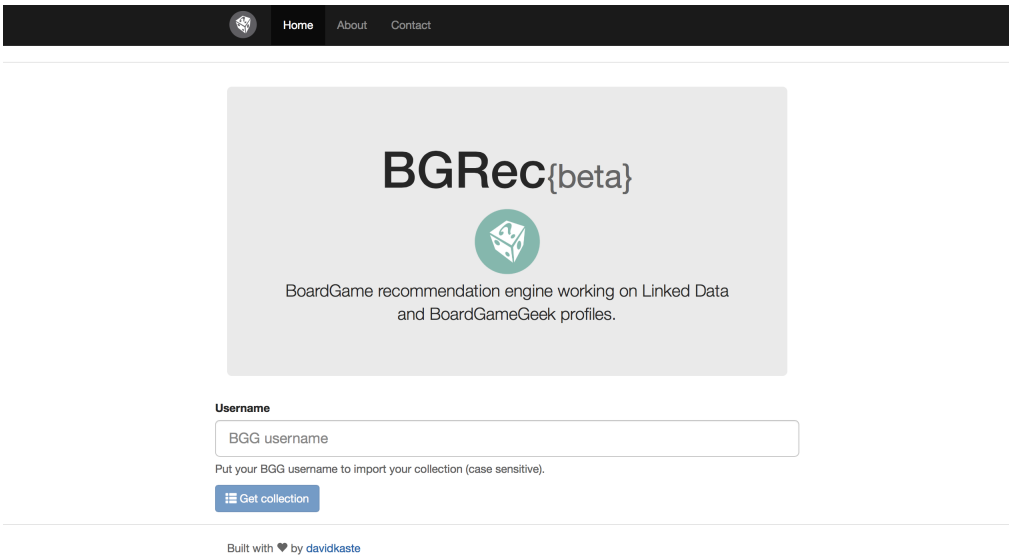

Figura 3.6: Captura de la *landing page* de **BGRec WebApp**

La pàgina principal, es crea amb *Twitter Bootstrap*, una eina que ens proporciona plantilles i dissenys predeterminats per a crear *frontends* web, de manera que es concentra l'esforç en la funcionalitat, ja que el disseny ja ve fet.

En primer lloc, es situa un menú superior per accedir a la pàgina índex, *about* i *contact*, i just a sota i en primer pla, un quadre amb el nom de l'aplicació, el logotip i una breu descripció d'aquesta.

A continuació, a sota del quadre de presentació, es situa un quadre de text on l'usuari introduirà el nom d'usuari de *BoardGameGeek* per tal d'importar la seva col·lecció de jocs de taula.

Després d'haver creat aquest disseny (Figura 3.6), s'ha de programar l'acció que es vol donar al formulari que hi ha a la part inferior de la pàgina. Per a tal fet, es crea el controlador *BggformCtrl* amb el generador de *Yeoman*. Aquest controlador conté el mètode *submit*, que serà al qual l'acció d'acceptar del formulari farà referència. Aquest mètode comprova que el nom d'usuari passat com a paràmetre no sigui una cadena de text buida, i si no ho és, captura el nom d'usuari i fa una redirecció cap a la pàgina on apareixerà la col·lecció importada de jocs de taula. La implementació d'aquest controlador queda de la següent forma:

```
angular.module('BGRec')
  .controller('BggformCtrl', ['$scope', '$location', function (
     $scope, $location) {
    $scope.submit = function(username) {
      if(username !== "") {
        $scope.username = username;
        console.log($scope.username);
        $location.path("/collection").search({username: $scope.
           username});
      }
    };
}]);
```

Un cop implementat el controlador, queda indicar a l'*HTML* quin controlador ha d'utilitzar i com ho ha de fer. Per tant, a l'arxiu *app.js* s'ha d'indicar a partir de la ruta *URL*, quin controlador i quina vista ha d'utilitzar. Com aquesta és la *landing page*, la ruta és '/', el controlador a utilitzar és *BggformCtrl* i la vista és on hem creat la pàgina que s'ha descrit, l'arxiu *main.html*. Això s'indica de la següent manera:

```
$routeProvider
  .when('/', {
    templateUrl: 'views/main.html',
    controller: 'BggformCtrl',
    controllerAs: 'bggform'
  })
```

Ara queda indicar al formulari de la vista com s'ha de comportar amb el controlador, per a això s'utilitzen diferents directives d'*AngularJS*, que es fa de la següent manera:

```
<form name="bggUserForm" ng-submit="submit(username)" class="form-
    horizontal">
  <div class="form-group␣form-group-lg">
    <label for="bgg-username">Username</label>
    <input id="bgg-username" ng-model="username" required [...] />
    <p class="help-block">Put your BGG [...]</p>
    <button class="btn␣btn-primary␣btn-primary" ng-disabled="
       bggUserForm.$invalid">
        [...]
    </button>
  </div>
</form>
```

ng-submit: Aquesta directiva indica quina mètode del controlador ha d'executar

ng-model: Aquesta directiva associada al control input, indica a quin model es desa el contingut, en aquest cas el model username on es desarà el nom d'usuari introduït.

ng-disabled: Indica al control button sota quines condicions s'ha d'activar o desactivar, ens aquest cas quan el contingut del formulari sigui vàlid, és a dir, no estigui buit.

3.5.3 M4-S2 Controlador collection i visualització

Un cop s'ha introduït el nom d'usuari de *BoardGameGeek*, s'ha de redirigir a una pàgina de visualització de la col·lecció de jocs. Per a crear aquesta funcionalitat es crea el controlador *CollectionCtrl* i la vista *collection.html* mitjançant el generador de *Yeoman*.

El controlador *CollectionCtrl* té la funcionalitat de capturar el nom passat com a paràmetre GET a la *URL* i seguidament fer una petició *HTTP* contra el recurs */collection* de l'*API* de **BGRec**. S'ha implementat de la següent manera:

```
angular.module('BGRec')
  .controller('CollectionCtrl', ['[...]', function ($http, $scope,
     $routeParams) {
      $scope.username = $routeParams.username;
      $http.get("http://[...]/collection", {params: {username:
         $scope.username}})
        .then(function(response) {
            $scope.collection = response.data;
```

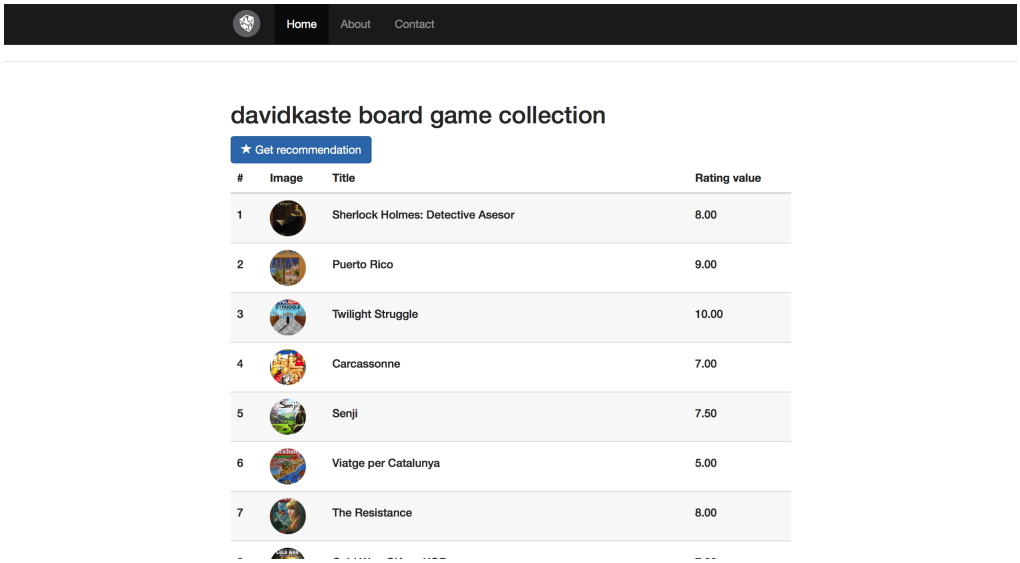

Figura 3.7: Captura de la pàgina de la col·lecció d'usuari

```
            console.log(JSON.stringify($scope.collection, null, 2));
        }).catch(function(data) {
            console.log(JSON.stringify(data));
        });
}]);
```

En la primera línia dins del controlador es veu com captura el paràmetre *username* de la *URL*, i a la següent executa la petició *GET*. L'ordre *$http.get* d'*AngularJS* retorna un *promise*, que ve a ser un valor substitutiu per al resultat de la petició mentre aquest no s'acabi de calcular, ja que no es sap quan pot tardar el servidor en tornar la resposta. El *promise* té dos mètodes, el *then* i el *catch*, on es passa una funció *callback* a cadascun d'aquests per si la petició s'executa amb èxit o falla, respectivament.

En la primera línia del *callback* passat al mètode *then*, mostra com es guarda les dades que retorna el servidor a la variable *collection* de l'*scope*, és a dir, la llista d'objectes *BoardGame* en format *JSON*. D'aquesta manera, a la vista es tindrà accés directe a aquesta variable.

Igual que en el pas anterior, s'ha d'indicar a quina vista resoldrà aquest controlador, això s'escriu a l'arxiu *app.js* de la següent manera:

```
.when('/collection', {
```

```
  templateUrl: "views/collection.html",
  controller: "CollectionCtrl",
  controllerAs: "collection"
})
```

Després d'això queda dissenyar la vista i implementar la lògica de la comunicació amb el controlador. Per a la visualització de la col·lecció de jocs, s'ha creat una taula amb les columnes imatge, nom i valoració, juntament amb un botó que portarà a l'usuari a conèixer la recomanació de jocs segons les seves preferències. El disseny final queda amb l'aparença mostrada a la figura 3.7.

Per tal d'indicar a la taula els elements de les files, s'han utilitzat diferents directives d'*AngularJS*, juntament amb l'accés al model *collection* creat al controlador.

Primer de tot s'indica a la taula, i concretament al la directiva *<tr>* (*table row*) d'*HTML*, la iteració d'*elements* de llista que haurà d'*executar*, això s'*implementa* amb la directiva *ng-repeat*, que en aquest cas s'ha indicat de la següent manera:

```
<tr ng-repeat="boardgame␣in␣collection">
  [...]
</tr>
```

ng-repeat: Aquesta directiva farà que per cada element de la llista *collection* creï un bloc *tr* amb tot el contingut indicat, ja que a dins d'aquest, es pot accedir a les diferents propietats de l'objecte *BoardGame*, que es fa de la següent manera:

```
<td>
  <h5>{{boardgame.name}}</h5>
</td>
```

Així, mitjançant els indiquem a *AngularJS* que ha d'accedir al valor de la propietat de l'objecte indicat.

Pel que fa al botó que ens portarà a obtenir la recomanació basada en les preferències de l'usuari, és senzillament un enllaç cap a la ruta /#/recommend passant el mateix nom d'usuari com a paràmetre GET.

3.5.4 M4-S3 Controlador recommendation i visualització

Un cop l'usuari prem el botó per obtenir la recomanació en base a les seves preferències, se'l dirigeix cap a una pàgina on hi ha una llista amb els 10 jocs de taula mes afins als seus gustos, ordenats de major coincidència a menor. Juntament amb

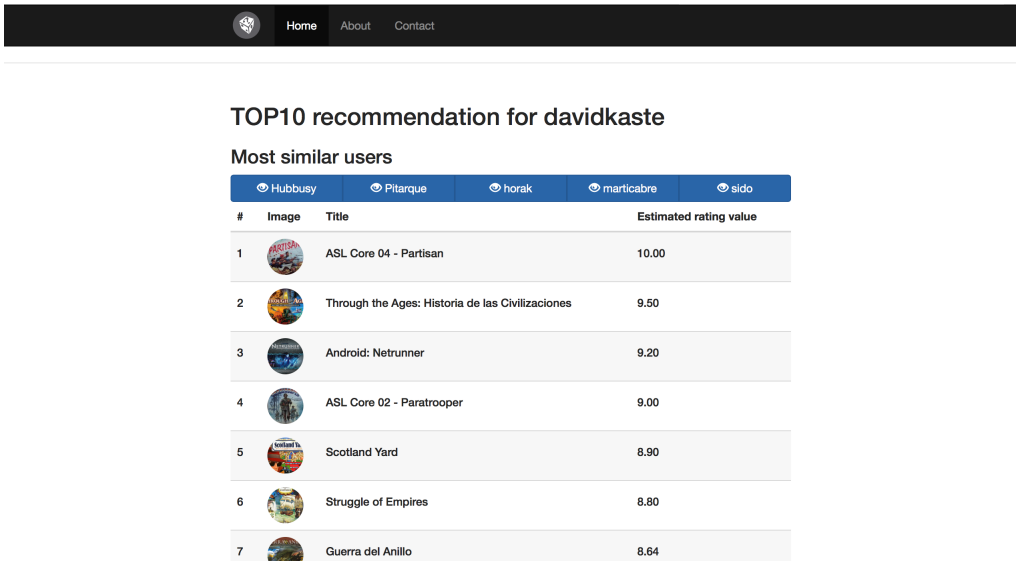

Figura 3.8: Captura de la recomanació d'un usuari

aquesta informació, també hi figuren els usuaris més semblants a l'usuari passat per paràmetre, de manera que l'usuari que obté la recomanació també pot navegar per les col·leccions i recomanacions de dits usuaris.

El primer pas per a crear aquesta funcionalitat, és crear els arxius necessaris amb el generador de *Yeoman*, en aquest cas, el controlador *recommendationCtrl* i la vista *recommendation.html*.

Acte seguit, s'implementa el controlador que, de la mateixa manera que el controlador *collection*, el primer que ha de fer és capturar el nom d'usuari dels paràmetres *GET* de la *URL*. Després d'això aquest controlador ha d'efectuar dues crides a l'**API BGRec**, una per obtenir la recomanació de l'usuari i l'altra per obtenir els usuaris més afins als seus gustos. La implementació queda de la següent manera:

```
angular.module('BGRec')
.controller('RecommendationCtrl', ['[...]', function ($http, $scope,
    $routeParams) {
  $scope.username = $routeParams.username;
  $http.get("http://[...]/recommend", {params: {username: $scope.
    username}})
    .then(function(response) {
      $scope.recommend = response.data;
      console.log(JSON.stringify($scope.recommend, null, 2));
    });
```

```
  $http.get("http://[...]/similarity", {params: {username: $scope.
      username}})
    .then(function(response) {
      $scope.sims = [];
      for(var key in response.data) {
        $scope.sims.push(key.split("/")[key.split("/").length - 1]);
      }
    });
}]);
```

Tal com es pot veure en el codi, un cop s'executi aquest controlador, hi haurà dues variables (llistes) disponibles a la vista, la llista de recomanacions de jocs *recommend* i la llista d'usuaris més afins a les preferències de l'usuari passat per paràmetre *sims* (abreviació de *similarities*). Però abans de codificar la vista s'ha d'indicar a l'arxiu *app.js* a quina vista resoldrà el controlador, per tant s'hi afegeix el corresponent bloc:

```
.when("/recommend", {
  templateUrl: "views/recommendation.html",
  controller: "RecommendationCtrl",
  controllerAs: "recommendation"
})
.otherwise({
  redirectTo: '/'
});
```

A part del corresponent bloc *when*, també s'afegeix el bloc *otherwise*, que en cas que l'usuari entri a una *URL* que no estigui *mapejada* a l'arxiu *app.js*, el redireccionarà cap a la *landing page*.

Per últim queda dissenyar la interfície on es mostrarà la recomanació per a l'usuari, que s'ha utilitzat la mateixa estructura de taula que en la col·lecció, però canviant el nom de la llista de *collection* a *recommend*, doncs no deixa de ser una llista d'objectes *BoardGame*, amb les mateixes propietats.

La particularitat d'aquesta vista és la capacitat d'iniciar una navegació cap a les col·leccions i recomanacions dels usuaris més afins. Per això s'ha creat una botonera a sobre de la taula indicant els noms d'usuaris. Aquest control s'ha creat utilitzant els següent codi:

```
<div class="btn-group␣btn-group-justified" role="group">
  <a href="#/collection?username={{user}}" ng-repeat="user␣in␣sims"
     [...]>
    <span class="glyphicon␣glyphicon-eye-open"></span> {{user}}
```

```
  </a>
</div>
```

Realment els botons no deixen de ser enllaços cap a la *URL collection* canviant l'usuari passat com a paràmetre. La directiva *ng-repeat* repeteix la directiva a (*anchor*) amb la propietat *href* canviada per a cada usuari. L'aparença de grup de botons ve donada pels estils de *Twitter Bootstrap*.

Capítol 4

Resultats

Després del seu desenvolupament, es va deixar provar el sistema a diferents contactes que tenen un perfil actiu a la pàgina *BoardGameGeek*, alguns d'ells membres de l'associació **ALEA** de Lleida, dedicada a promoure i difondre l'afició als jocs de taula moderns i jocs de rol. Després d'introduir al sistema una sèrie de noms d'usuari que els participants van facilitar, més altres noms d'usuari extrets del fòrum de la pàgina, es va arribar als 26 usuaris al graf *RDF*, tot i que la majoria d'usuaris al sistema eren d'un perfil *wargamer*, és a dir, aficionats als jocs de taula de simulació militar. Un cop feta la recol·lecció de dades, es va procedir a treure les recomanacions.

Dels 6 usuaris que van extreure la recomanació, 5 d'ells van confirmar que les 5 primeres posicions havia encertat perfectament amb les seves preferències, tot i que les 5 següents no eren del tot correctes. L'usuari que falta, té un perfil de jugador més diferent, més *eurogamer*, és a dir, aficionat als jocs on l'atzar és pràcticament nul i d'alta estratègia en gestió de recursos, per tant la seva recomanació distava molt de les seves preferències, degut al conegut problema del *cold start* i de que la majoria d'usuaris entrats al sistema són d'un perfil completament diferent. Per tenir una recomanació més fidel a les seves preferències s'haurien d'introduir més usuaris i d'un perfil mes variat.

Tots ells van coincidir que el sistema era senzill d'utilitzar, tot i que l'aparença gràfica podria ser millorable.

Aquesta prova es va dur a terme de forma informal al local de l'associació, durant la reunió setmanal per a jugar a jocs i provar diferents novetats del mercat. D'aquí que no hi hagin resultats empírics d'aquesta prova.

Capítol 5

Conclusions

5.1 Reflexió

Tot i que el treball s'havia estructurat, en un principi, per a ser desenvolupat mitjançant metodologies àgils i seguint un calendari de desenvolupament, per diferents qüestions de disponibilitat i altres factors, no ha estat així. Com a exercici d'autocrítica, si s'hagués de tornar a començar aquest projecte, segurament es seria molt més estricte i disciplinat en aquest aspecte, portant al dia la documentació i posant les fites no només del codi que s'ha d'escriure si no també la data en que hauria d'estar entregada.

5.2 Treball futur

Tot i que el sistema desenvolupat és completament funcional, la seva vida no hauria d'acabar aquí. S'ha fet aquest treball no només pensant en un projecte de final de màster, si no en la intenció de crear un producte. Tant **prophetik** com **BGRec** es continuaran actualitzant i mantenint en un futur. Les pròximes actualitzacions del sistema, en ordre cronològic, inclouran el següent:

Desplegar l'aplicació al núvol Aquest és un dels objectius d'aquest treball, crear un producte que tothom pugui utilitzar, des de gairebé qualsevol lloc del món.

Alliberar prophetik sota una llicència *open source* i crear marca La llibreria

prophetik ha de millorar cada dia, per tant s'ha d'alliberar sota una llicència de codi lliure per a que la comunitat el pugui anar millorant i els desenvolupadors els puguin utilitzar lliurement per als seus projectes. Per això també es crearà un compte de *Twitter* sota el nom de *prophetik*, per a donar a conèixer el projecte, les novetats i incentivar el seu ús.

Aplicació mòbil de BGRec D'aquesta els usuaris poden obtenir les recomanacions sigui on siguin, especialment quan s'està en una botiga especialitzada decidint quin joc s'ha de comprar.

Convertir BGRec en una aplicació autònoma més social Això significa de trencar l'enllaç cap a *BoardGameGeek* i que els usuaris puguin crear completament des de 0 una col·lecció de jocs de taula i/o ampliar la que ja tenien.

Publicació i enllaç del *dataset* de jocs de taula Un cop el sistema tingui un nombre d'usuaris significatiu, es procedirà a publicar el *dataset* al graf *LOD*, enllaçant els ítems amb altres sets de dades que facin referència al mateix producte. Segons l'escala proposada per Tim Berners-Lee, aquest conjunt de dades rebria una puntuacio de 4 estrelles, seria públic i en un format estàndard segons la W3C, però no tindria enllaços cap a altres conjunts de dades presents al LOD. Per tant, per aconseguir la puntuació de 5 estrelles s'haurien de crear aquests enllaços cap a *DBPedia* o altres. D'aquesta manera el sistema **BGRec**, podria consumir més informació sobre els jocs de taula.

Incorporar a prophetik un sistema de recomanació híbrid Això implicaria crear un sistema de filtratge basat en contingut, és a dir, enlloc de comparar la semblança dels usuaris, mirar la semblança dels productes entre si. D'aquesta manera es poden executar els dos sistemes de recomanació en paral·lel per a després combinar els resultats i així augmentar la seva efectivitat.

5.3 Conclusions

Aquesta part està escrita en primera persona, ja que són les conclusions de l'autor les que hi figuren.

El més ràpid que puc concloure d'aquest treball és l'aprenentatge de diferents eines i llenguatges que sense aquestes no hauria estat possible la finalització del treball. Entre aquestes hi figuren el llenguatge de consulta *SPARQL* i l'ecosistema de desenvolupament que proporciona *Yeoman*. Sense passar per alt al llenguatge Java 8,

que per molt que sigui un llenguatge molt utilitzat en el mon acadèmic, hi han detalls que s'han hagut d'implementar sense tenir certs coneixements que s'han hagut d'aprendre, com les funcions anònimes o els *streams*.

Pel que fa al desenvolupament, he aprés a dissenyar un producte de dalt a baix, tot un producte. Tot aquest procés m'ha fet veure que escriure el codi d'un sistema que funciona no és suficient, és a dir, és tant important el que fa el sistema com el com ho fa. Per això encara hi queda molta feina a millorar el codi que ja hi ha escrit.

Com a última conclusió, però no per això és menys important, es que fent aquest projecte m'he pogut sentir com a enginyer. Això m'ha fet provar-me a mi mateix, posar-me a test i saber del que sóc capaç i del que no. He desenvolupat un producte per a solucionar un problema i donar aquesta funcionalitat a qui la necessiti, i ho fa satisfactòriament. Aquesta és una de les màgies de l'enginyeria informàtica, que més que fer un projecte, podem crear des de zero, tenim la virtut de poder construir eines que poden canviar la forma en que concebem al món. Al final he vist que aquest projecte, que en un principi, l'he vist com a un TFM, que després de l'entrega s'oblida, al final l'he acabat veient com el naixement d'un nou projecte.

Bibliografia

[1] Toby Segaran, *Programming Collective Intelligence*, O'Reilly, 2nd edition, 2007.

[2] Bob DuCharme, *Learning SPARQL*, O'Reilly, 1st edition, 2011.

[3] JM Brunetti PhD., *Interacting with semantic web data through an automatic information architecture*, 2013.

[4] Various, *Recommendation systems*, Wikipedia.