Twitterbot: robot de Twitter per a la generació de notícies

Pau López de Recalde i Sanfeliu

Resum– Des de sempre l'ésser humà ha tingut la necessitat de conèixer i comprendre el que succeeix al seu voltant. Primàriament es pot veure clarament com un mètode de supervivència al saber com interactua el nostre entorn i com podem nosaltres anticipar-nos a aquest. Avui en dia es valora no només els coneixements adquirits a través dels estudis, sinó el que s'anomena com 'cultura general'. Cada cop més les tecnologies guanyen terreny en els nostres móns. El que abans s'obtenia llegint premsa escrita avui ha estat substituït per premsa digital. Aquest projecte pretén crear un portal de notícies que s'actualitzi automàticament a través de notícies obtingudes de la xarxa social Twitter, evitant un manteniment de notícies manual. D'aquesta manera no seria imprescindible que hi hagués algú treballant com a redactor ni periodista. Aquest portal serà apte per a qualsevol dispositiu modern adaptant el contingut a aquest i permetent a l'usuari registrar-se i interactuar amb el sistema.

Paraules clau– Notícies, informació, diari, Twitter, automatització, portal web, *responsive*, Bootstrap, HTML, PHP, MySQL, intel·ligència artificial, llenguatge natural, MVC, Internet.

Abstract– Humans have always had the need to know and understand what is happening around them. That point can clearly be seen primarily as a method of survival in how it interacts with our environment and how we can anticipate this. Today is not only valued the knowledge acquired through education, but what is termed as 'general knowledge'. Increasingly, technologies are gaining ground in our worlds. What we previously obtained by reading newspapers today has been replaced by digital media. This project aims to create a news portal that is automatically updated via reliable sources of news (Twitter groups), avoiding manual maintenance. Thus there will be no one working as an editor or journalist. This site will be suitable for any modern device being responsive to this content and allowing the user to register and interact with the system.

Keywords– News, information, newspaper, Twitter, automatitzation, website, responsive, Bootstrap, HTML, PHP, MySQL, artificial intelligence, natural language, MVC, Internet.

1 Introducció

Actualment la necessitat d'estar informat del que passa al nostre voltant s'ha vist incrementada a causa de la crisi. Són moltes les empreses que demanen coneixements de cultura general com a filtre de processos de selecció. Per a estar informat cal una font fiable d'informació que estigui actualitzada constantment. Avui en dia el consum de premsa escrita ha baixat molt donada la despesa de diners i de temps per part del lector i de material per part de l'empresa. És per això que han sorgit els portals de notícies *online*. Aquests necessiten un manteniment i redacció constant de les notícies.

L'objectiu d'aquest article és de proposar un sistema que ens permeti agafar la informació automàticament de les notícies, facilitant així la tasca del periodista o redactor, la classificació d'aquesta informació en les categories generals que té qualsevol diari i de mostrar aquesta informació en forma de notícies a l'usuari mitjançant tecnologies que ens permetin arribar a qualsevol dispositiu d'avui en dia.

2 Objectius

L'objectiu del projecte és tenir un sistema independent i funcional que integri els mòduls necessaris per resoldre l'enunciat del projecte: tenir un portal web de notícies actualitzat automàticament mitjançant la creació d'un robot que

• E-mail de contacte: pau.lopezderecalde@e-campus.uab.cat
• Mención realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Jordi Casas-Roma (Departament d'Enginyeria de la Informació i de les Comunicacions)
• Curs 2015/16

agafi les notícies de Twitter i que classifiqui aquestes en grups estandarditzats. Podem dividir aquest enunciat en 4 grans subobjectius:

1. Desenvolupament d'un robot o recol·lector que iteri pels grups de Twitter establerts extraient la informació en forma de *tweets* i la emmagatzemi en una base de dades.

2. Elaboració d'un classificador que s'encarregui de classificar els *tweets* emmagatzemats a la base de dades.

3. Construcció d'un portal web complert que permeti visionar les notícies recol·lectades i permeti una *customització* per part de l'usuari dels grups de notícies als quals està subscrit. Aquest portal haurà d'adaptar-se a tots els dispositius i pantalles (disseny *responsive*).

4. Desenvolupament d'una base de dades que contingui tota la informació necessària, tant dels *tweets* recol·lectats històrics com dels usuaris i les relacions usuari-grup subscrites.

L'acompliment d'aquests objectius es farà assegurant en cada etapa un acabat de qualitat que respongui als estàndards de TI actuals. A l'acabar el projecte ha d'estar funcionant el portal de notícies (*front-end*) així com el motor de selecció i classificació d'informació (*back-end*). D'aquesta manera es complirà la resolució del problema inicial amb la qualitat necessària.

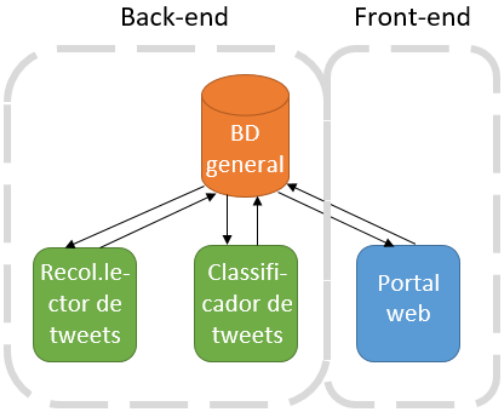

Fig. 1: Esquema dels objectius

3 Estat de l'art

Avui en dia i arrel de l'expansió de les TI han sorgit un munt de pàgines web de notícies i de premsa digital. Moltes d'aquestes pàgines són versions electròniques de premsa escrita ja existent. La majoria de consumidors s'ha passat a aquest nou paradigma d'emissió d'informació principalment pel cost reduït que els suposa, així com no haver de desplaçar-se per estar informat. És profitós també per persones amb mobilitat reduïda o que no poden sortir de casa per edat avançada, etc. El problema que tenen tots aquests serveis web és que necessiten de personal que gestioni la selecció i redacció de les notícies. Això encareix els costos de mà d'obra doncs es necessita personal qualificat en el món del periodisme i una cerca constant d'informació. Es requereixen sistemes autònoms capaços de facilitar, o fins i tot substituir, aquest dispendi de temps/diners. Un altre problema que presenten aquests portals és que l'usuari no té poder de personalització de notícies, així com la creació de portades *customitzades* al gust de cada usuari. Dit d'una altra manera, hi ha una necessitat de tenir un sistema automatitzat de recol·lecció de notícies que respongui a les necessitats de cada usuari.
Actualment existeixen alguns serveis que permeten simular un sistema de RSS o *feed* com Twitter Feed[2]. Aquests però no tenen com a objectiu principal informar mitjançant notícies periodístiques als usuaris.

4 Metodologia

Per tal de dur a terme el desenvolupament del projecte s'ha seguit un model iteratiu i incremental on a cada etapa s'han desenvolupat funcions del sistema. D'aquesta manera s'ha construït i refinat els diferents mòduls a cada iteració que ens han permès provar l'aplicació com a conjunt.

La planificació iterativa que s'ha seguit al llarg del projecte és la següent:

- Fita 1: 15/11/2015. Tenir connexió amb Twitter i mètodes de consulta de *tweets* i de *trending-tòpics* (model). També disposar d'una interfície elemental web que ens permeti veure els *tweets* que hem agafat amb el recol·lector i que hi hagi unes classificacions elementals (esports, política...). En aquesta fita ha de quedar la base de dades construïda (amb camps bàsics, no caldrà disseny complert) i enllaçada amb els diferents mòduls. No cal estil en les pàgines, només visualització de contingut.

- Fita 2: 20/12/2015. Classificador de *tweets* funcionant amb un nivell de qualitat acceptable. Pàgines web amb un estil atractiu i amb les notícies ben classificades. Base de dades refinada amb els camps necessaris i optimitzada. Models entitat-relació elaborats. Sistema treballant correctament com a conjunt i formant una agrupació funcional operativa.

- Fita 3: 24/01/2015. Refinament de l'estil de la pàgina web. AJAX aplicat i funcionant. Ús de tecnologia Javascript i jQuery. Codi web de qualitat passat per validador HTML5[3] funcionant igual que el CSS3. Classificador de *tweets* amb el mínim error possible. Base de dades i aplicació de recol·lector de *tweets* i classificador muntada externament a un servidor amb domini públic, a l'igual que el servei web operatiu (Amazon EC).

Per al desenvolupament del projecte hem dividit la nostra aplicació en 4 mòduls principals: el recol·lector de *tweets* o robot, el classificador de *tweets*, el portal web i la base de dades. Aquesta darrera és el nexe comú de les altres 3. Els mòduls no es comuniquen entre sí sinó que la informació està centralitzada a la base de dades, a excepció de la gestió del recol·lector i el classificador, que s'operen fent ús d'un *dashboard* des del portal web (comunicació via *sockets* entre Python i PHP). Aquest punt s'explicarà a l'apartat del sistema web.

Tot el desenvolupament utilitzarà el paradigma model-vista-controlador per tal d'evitar dependències i per a tenir

un projecte de qualitat amb un baix acoblament. El codi seguirà un estil de codificació acurat (altament documentat, màxim 80 caràcters per línia, etc.) i es dedicarà una part per a fer test dels diferents mòduls.

4.1 Base de dades

Comencem fent la explicació de la base de dades doncs la seva especificació ens serveix com a protocol de comunicació. Recordem que la gran majoria d'informació que es passen els mòduls del projecte és emmagatzemada a la base de dades.

La nostra base de dades consta de 4 taules:

- Category. Conté l'identificador i el nom de cada una de les categories que poden tenir els tweets.

- Tweet. Conté tota la informació de cada *tweet*. Aquesta és: identificador propi del *tweet* de Twitter (clau primària de la taula i és un *long int*), l'idioma en que està escrit (ens ho dóna Twitter), el text del *tweet* o notícia, els cops que ha estat *retweeteijat*, els cops que ha estat marcat com a preferit, el dia i la hora de creació i les coordenades (per en línies futures classificar per zones geogràfiques). Tots aquests atributs venen directament dels atributs d'un *tweet* de Twitter, no s'ha de calcular res sinó fer la petició REST des de l'API que hem escollit.

- User. Conté la informació dels usuaris del nostre portal. D'aquests guardem un id (clau primària i índex autoincremental per motor de la base de dades), el nom d'usuari o *username*, la contrasenya o *password*, l'email i un camp booleà que indica si és administrador o no. Aquest camp és vital per a múltiples comprovacions en apartats d'administració i és constantment comprovat des del sistema web. Per l'emmagatzematge de contrasenyes s'utilitza un mètode segur que s'explica a l'apartat de seguretat del sistema web més endavant.

- Twitter_user. També anomenat grups de notícies de Twitter, fa referència als grups dels quals agafem informació. En guardem el nom i un camp booleà *valid* que ens indica si ha estat moderat per part d'un administrador.

A l'annex A.1 es pot veure el diagrama entitat-relació complert de la base de dades del projecte. Aquesta base de dades està funcionant per un sistema gestor de bases de dades de MySQL. Les consultes doncs estan desenvolupades en el llenguatge SQL adaptat al gestor.

4.2 Recol·lector de tweets

4.2.1 API REST o Streaming

El primer punt a tenir en compte a l'hora de començar amb el recol·lector és saber com farem aquesta obtenció dels *tweets*. Twitter ens proporciona dos mètodes diferents d'estar connectats amb la seva xarxa social[4]: el mètode REST i el mètode Streaming. Cadascun d'ells té les seves particularitats i serà més apte en uns casos que en uns altres.

TAULA 1: COMPARATIVA API REST I STREMING

	API REST	API Streaming
Recursos	Baix	Alt (memòria, CPU i xarxa)
Complexitat	Baixa	Alta (estructures de dades)
Comunicació	Discrecional	Simultània i permanent

- REST. És una API que, en essència, utilitza per a comunicar-se missatges discrecionals. És a dir, s'envia una petició única i aquesta es resposta per part de la xarxa de Twitter. Un cop s'ha donat resposta a la pregunta es tanca la connexió i ambdós nodes extrems s'obliden de la comunicació que passa a ser històrica. Els missatges són en formats ja establerts en estàndards de comunicacions a nivell d'aplicació, tals com XML o més utilitzat el JSON[5], que agafa el format de missatges Javascript per a treballar amb objectes. L'avantatge d'aquest mètode és el poc consum de memòria i de connexions simultànies que ha de gestionar el sistema operatiu del servidor on allotgem la nostra aplicació.

- Streaming. És una API que es manté amb una connexió establerta amb Twitter a llarg termini de forma que va rebent actualitzacions constants (semblant a un navegador web que rep notificacions i missatges). D'aquesta manera té un sistema més complex de connexions a nivell d'aplicació i treballa amb conjunts molt grans de connexions HTTP establertes simultàniament. És deduïble que consumirà més recursos del nostre servidor d'aplicació final tant en memòria i CPU com en entrada/sortida de xarxa.

Pel tipus de projecte que ens ocupa la millor opció per nosaltres és la d'utilitzar la API REST[6], doncs les consultes que farem seran discrecionals on emmagatzemarem la informació al moment de rebre-la. També la complexitat és molt inferior i ens permetrà dedicar més temps als altres apartats sense ocupar-nos de funcionalitats innecessàries. Per accedir a la API de REST utilitzarem la llibreria ja existent de Python anomenada Tweepy[7]. És una llibreria senzilla i molt potent que ens permet estalviar molt de temps desenvolupant les funcionalitats necessàries. El seu lema és "*Una llibreria de Python fàcil d'utilitzar per a accedir a la API de Twitter*".

Per la realització del recol·lector o robot de *tweets* hem creat les classes Controller, Model, DBConnector i Server. Cadascuna d'elles és un mòdul independent i necessari per a l'obtenció de *tweets*. El Controller és la classe principal que s'encarrega de l'obtenció i la pujada de *tweets* a la base de dades. Per fer-ho utilitza funcions implementades a la classe Model, DBConnector i Server. A continuació hi ha una explicació de les classes:

4.2.2 Classe Controller

Classe principal del recol·lector de *tweets*. Els seus atributs permeten l'emmagatzematge d'una llista de *tweets* per a ésser pujada a la base de dades, una instància per a la connexió a la base de dades de la classe BDConnector, una instància concurrent de la classe Server i un atribut objecte de la classe Classifier per a classificar. Tenim també els

següents mètodes que ens serveixen per a realitzar les funcions principals del classificador:

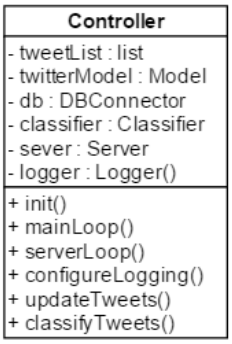

Fig. 2: Diagrama de classe del Controller

- Constructor __init()__: inicialitza els atributs del model i classificador amb instàncies de les classes Model i Classifier, així com l'atribut "db" amb una instància de la classe DBConnector. Llança el *thread* que s'executarà de manera paral·lela com a servidor del classificador i configura el sistema de *logs* de l'aplicació de recol·lecció i classificació.
- mainLoop(): *thread* principal d'execució on cada cert temps (demanat a l'iniciar el recol·lector) es crida a recol·lectar i a classificar.
- serverLoop(): *thread* que conté el mètode principal d'escolta del servidor de la classe Server.
- updatetweets(): afegeix a la base de dades els *tweets* que hem anat agafant.
- classifytweets(): crida al classificador per a realitzar la classificació.

Aquesta classe fa ús també del Threading de Python, el qual ens permet executar un servidor (classe Server explicada més endavant) que ens permet rebre peticions del servei web per dur a terme operacions des del portal.

4.2.3 Classe Model

Classe que conté els mètodes per a la connexió i obtenció de *tweets*.

Autenticació amb OAuth Aquesta classe Model té els atributs necessaris per a fer l'autenticació amb OAuth[8] (4 cadenes de text: *consumer key, consumer secret, access token, acces token secret*; un objecte auth i un objecte api). Aquest darrer atribut 'api' és el que ens permet la obtenció dels *tweets* i l'iterador sobre Twitter dins del nostre usuari creat. A continuació es pot veure l'esquema (figura 3) de missatges que comparteix la nostra aplicació amb Twitter mitjançant OAuth. En primer lloc, nosaltres demanem a Twitter, com a administradors d'una aplicació, les claus i els tokens necessaris per a desenvolupar una aplicació que estarà connectada a la xarxa social de Twitter mitjançant la plataforma per a desenvolupadors . Twitter ens demana doncs que tinguem un usuari creat per a vincular-lo amb la nostra aplicació. D'aquesta manera, la nostra aplicació

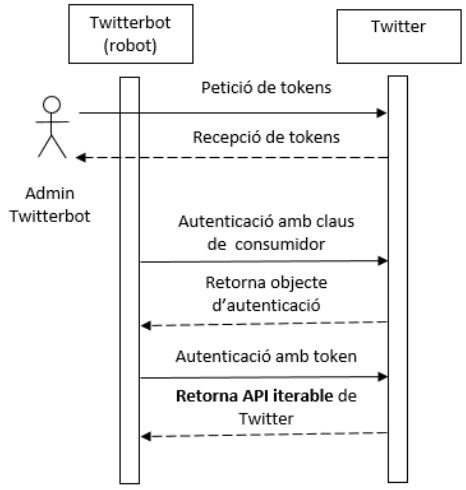

Fig. 3: Diagrama de seqüència OAuth

tindrà els mateixos permisos de consulta i escriptura de *tweets* que un usuari normal de Twitter. Un cop creat l'usuari i vinculat a la nostra aplicació Twitter ens retorna 4 claus necessaries per a poder utilitzar una aplicació estant connectats a la xarxa social:

- Consumer key. Identificador del client. És la clau de la API associada a la aplicació de Twitter.
- Consumer secret. La contrasenya del client que ens permet autenticar-nos davant del servidor de Twitter.
- Access token. És sol·licitat al client un cop s'ha autenticat. Aquest defineix els privilegis del client.
- Access token secret. És enviat junt a l'Access token i el protegeix d'usos indeguts en cas de conèixer únicament aquest.

Resumidament les dues primeres claus ens serveixen per a autenticar-nos a Twitter i les dues últimes per a autoritzar-nos a fer ús de la API que ens permet operar (es separa autenticació de autorització).

Estructura Per al tractament senzill des de fora de la classe hem creat els següents mètodes:

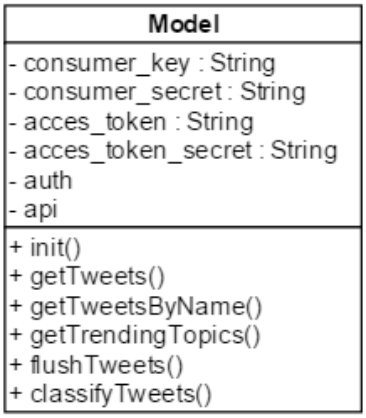

Fig. 4: Diagrama de classe del Model

- Constructor __init__(): Inicialitza les variables per al OAuth treballant amb la classe OAuthHAndler i li dóna a l'atribut 'api' el valor resultant de demanar-lo amb les dades d'autenticació.
- getTweetsByName(int name, int number): Ens retorna una llista de Python els "number" *tweets* de l'usuari 'name'. Exemple getTweetsByName('324cat', 10).
- getTrendingTopics(): Ens retorna una llista de Python amb els 10 trending topics.
- flush(): crida al mètode flushTweets() de DBConnector. Purga la taula 'tweet' de la base de dades. Aquest mètode només es pot invocar des del portal web estant logat com a administrador des del *dashboard* del recol·lector. Està pensat únicament per a buidar base de dades en cas de tests i proves.
- classify(): Crida a classificar a la instància Classifier.

Com podem veure des de la classe Controller podem invocar des de molt alt nivell operacions que utilitzen d'altres classes (disseny per mòduls).

4.2.4 DBConnector

Aquesta és la classe encarregada de treballar amb la base de dades externa que uneix els diferents mòduls. Ofereix connexió a aquesta i resolució d'operacions utilitzades pels mòduls del classificador (Controller i Model). Per al maneig de bases de dades de MySQL des de Python s'ha fet servir la classe nativa que ens ofereix el llenguatge de Python: la MySQLdb[9]. La estructura de la nostra classe DBConnector és la següent:

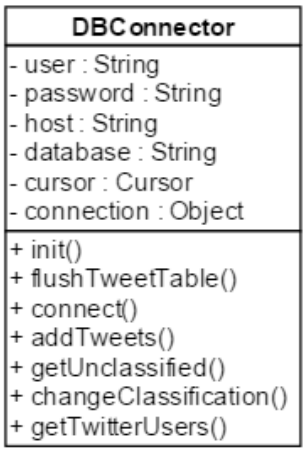

Fig. 5: Diagrama de seqüència OAuth

- Constructor __init__(): inicialitza els atributs als valors passats per paràmetre (des de Controller).
- flushTweetTable(): buida la taula de *tweets*. Mètode utilitzable des de l'interfície web.
- Connect(): estableix connexió amb la base de dades.
- addTweets(tweetList): afegeix a la base de dades els *tweets* de la llista tweetList.
- getUnclassified(): retorna una llista de ids i text amb els *tweets* no classificats (category = 0).
- changeClassification(classificationList): rep una llista de 'id, valor' i actualitza la classificació del tweet amb aquella id.
- getTwitterUsers(): ens retorna els grups dels quals llegim notícies.

4.2.5 Server

Finalment, per a tenir enllestit el mòdul del recol·lector ens fa falta que aquest sigui operat, no des d'una línia de comandes, sinó des del portal web, que serà la única entrada i sortida dels usuaris (i dels administradors del lloc web). Perquè el sistema funcioni aquest ha de poder ser activat i desactivat, així com consultar l'estat en el que es troba i els logs, des de la interfície web. És per això que és necessari que en el recol·lector hi hagi corrent constantment una classe Server que estigui a la escolta de peticions que arribin des del servidor web. Evidentment aquestes peticions i consultes només podran ser executades per usuaris administradors.
L'arquitectura del servidor és força senzilla. Primerament, i sense entrar en la classe en si, tenim el flux principal d'execució del programa en la classe Controller. Serà necessari definir un mètode que s'executi concurrentment amb el fil principal d'execució del programa.

Threading Hem utilitzat les classes natives de Python que ens permeten fer ús del *threading*, concretament la classe "Threading"[10].

```
t = threading.Thread(target = self.
serverLoop)
t.start()
```

Codi 1: Creació d'un thread en Python

La funció serverLoop() crida al mètode loop() del Server que analitzarem a continuació:

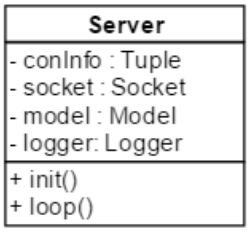

Fig. 6: Diagrama de classe del Server

Com podem veure aquesta classe té la informació de la connexió (tupla de IP/Port on estarà escoltant peticions), del *socket* de Python[11] que té la connexió oberta i un *logger* per a gestionar missatges d'aplicació. El mètode principal de la classe és el loop(), que es veu a continuació:

```
while True:
  data, addr = self.__socket.recvfrom(2)
  if data:
    if data == "01":
      self.__model.getTweets()
```

```
elif data == "02":
  self.__model.classifyTweets()
elif data == "03":
  self.__model.flushTweets()
[...]
```

Codi 2: Bucle del servidor a la escolta

Com es pot comprovar hem creat un protocol propi de comunicacions a nivell d'aplicació que tracta tots els casos possibles d'enviament (al final hi ha un *default* que s'executa en qualsevol altre cas de recepció de dades). Enviem pel *socket* 2 bytes corresponents al codi d'operació.
D'aquesta manera el Controller principal té aquesta porció de codi sent sempre executada i escoltant les peticions que vinguin del servidor web. Al sistema web tenim un fitxer doncs que també corre un *socket* de PHP[12] que és el que envia la informació. Com a mesura de seguretat a la pàgina PHP que gestiona l'enviament de missatges es valida en tot moment que la sessió d'usuari PHP sigui amb un usuari el qual tingui el camp d'administrador a la base de dades com a '1'. De totes maneres, de la seguretat i estructura del servidor web se'n parlarà més endavant.

4.3 Classificador de tweets

El classificador de *tweets* és el segon mòdul del Twitterbot. El funcionament d'aquest, a grans trets, és el d'agafar tots els *tweets* de la base de dades que no estan classificats i un per un comprovar a quina categoria s'adapta més. Seguidament agafa aquests *tweets* i els puja de nou a la base de dades alterant el valor del camp 'category'. A continuació s'expliquen detalladament els passos de l'algorisme de classificació:

4.3.1 Obtenció de tweets no classificats

Per tal d'obtenir els *tweets* que no estan classificats simplement necessitem agafar el camp 'id' i 'text' dels *tweets* que no han estat classificats, és a dir, que el camp 'category' és 0. No necessitem cap altre camp que no sigui el del text a analitzar i l'apuntador ID per a poder fer l'*update* a la base de dades un cop classificat. La consulta SQL és la següent:

```
SELECT id, text FROM tweet WHERE
category = '0';
```

Codi 3: Consulta de *tweets* no classificats

D'aquesta manera el mètode que s'encarrega d'obtenir els *tweets* no classificats retorna una llista de Python on els seus valors són tuples (id, text). Aquesta llista passa al següent pas de l'algorisme.

4.3.2 Classificació dels tweets

Un cop tenim la llista de tuples, anem agafant tupla per tupla i ens quedem amb la cadena de text del *tweet*. Per a cada categoria en el classificador tenim un conjunt de *tags* o paraules clau que formen un diccionari. Per exemple de política tenim:

```
self.__tagsPolitica = ["politica",
"congres", "president", "cup",
"pais", "parti", "artur", "sonde",
"debat"[...]]
```

Codi 4: Visió de tags del recol·lector

Per a cada *tweet* mirem de cada llista de *tags* el nombre d'aquests que apareixen. D'aquesta manera fem una classificació basada en diccionari. El resultat d'aquesta operació és un número que ens indica a quina categoria va (1 per política, 2 per internacional...). En aquest punt creem una nova llista de Python que contindrà tuples amb (id, valor), on valor és el número que identifica la classificació del *tweet*, que seran els camps únicament necessaris per a fer els *updates* a la base de dades.

4.3.3 Actualització de categoria

Finalment l'únic que ens queda fer en l'algorisme de classificació és actualitzar la base de dades. Igual que a l'inici no fa falta actualitzar tots els camps, únicament necessitem l'apuntador que és l'ID del *tweet* i el nou valor del registre 'category'. A continuació tenim la consulta SQL que executem per cada un dels camps de la llista anteriorment anomenada.

```
UPDATE tweet SET category = \%s
WHERE id = \%s;""" \%
(classification[1],
classification[0]));
```

Codi 5: Actualització de categoria

4.4 Portal web

4.4.1 Requisits del sistema web

Per a entendre el que sistema web avarca s'ha de complir el següent enunciat: es necessita un portal web que mostri la informació en forma de *tweets* als usuaris. Haurà de tenir una pàgina *home* on es mostrin les últimes notícies dels diferents apartats o categories (política, tecnologia...). També serà possible consultar cada un dels apartats diferents, de tal manera que puguem veure únicament els *tweets* d'una categoria concreta, com per exemple els de política. El portal web ha de tenir, a més, una secció anomenada "Els més"que contindrà un rànquing per a les notícies o *tweets* més *retweeteijats* o que més han donat de parlar i un rànquing per als preferits o que més han agradat. Aquests es podran veure tant per rànquing diari, com per mensual i com per vitalici (des de l'inici del portal). Per tal de mostrar o no certs grups de notícies a l'usuari, aquest tindrà un sistema de registre i *login* que permetrà als usuaris estar registrats al sistema i conèixer les seves preferències, així com introduir nous grups de notícies al recol·lector. De cada usuari se'n guardarà *username*, el correu electrònic, l'identificador únic que serà autoincremental i la contrasenya que triï. Aquesta haurà d'estar emmagatzemada de manera segura a la base de dades. Els grups de notícies hauran de ser validats per un usuari administrador a posteriori per tal de passar a formar part de manera permanent. Tal com s'ha comentat hi haurà distinció entre dos tipus d'usuari: usuari lector o estàndard i usuari administrador. L'usuari administrador pot fer el mateix que l'usuari estàndard però a més a més té accés a un menú d'administrador que li permet executar tasques avançades. Aquestes són:

- Gestió de les fonts d'informació. Permet afegir i treure grups d'informació així com moderar grups afegits per usuaris estàndards que encara no han estat validats.

- Gestió dels usuaris. Permet editar la informació emmagatzemada de cada usuari (contrassenya en cas de pèrdua i email) així com eliminar l'usuari o fer-lo administrador.

- Gestió del recol·lector. Aquest es podrà iniciar i parar, així com el classificador. Es podran llençar individualment les ordres de recol·lectar, de classificar pendents, de reclassificar tots ele *tweets* (útil per si canviem diccionaris) i de purgar o eliminar per complet la taula de *tweets*. És el *dashboard* del sistema web.

Finalment hi haurà una àrea privada per a cada usuari, tant si és administrador o usuari estàndard que li permetrà accedir a la seva portada personalitzada i a la configuració d'aquesta.

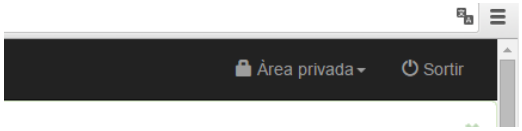

Fig. 7: Visió de menú estàndard

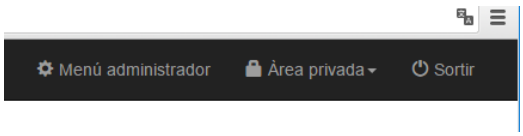

Fig. 8: Visió de menú d'administrador

Per tal de mostrar la informació als usuaris finals, així com per controlar el recol·lector i el classificador s'ha desenvolupat el portal web amb la seva interfície pròpia. Aquest portal és totalment *responsive* i s'adapta a tots els dispositius i pantalles. Per a fer-ho hem utilitzat el *framework* de desenvolupament de Bootstrap en la versió més recent, la 3.0[13].

4.4.2 Bootstrap 3.0

Ha estat el nostre 'dissenyador gràfic' del portal web. Tot i que inicialment es va fer un disseny gràfic desenvolupat totalment de manera pròpia, aquest va ser ràpidament substituït al veure que no era *responsive* i no s'adaptava ni a una mateixa pantalla canviant l'ample d'aquesta. Aquest canvi va suposar un fort contratemps tot i que al donar-se a la última iteració no s'ha anat arrossegant al llarg del desenvolupament del projecte. La part positiva era que el *back-end* en PHP s'ha pogut reaprofitar completament i que Bootstrap ens permet fer el disseny dels components, tals com botons i menús en un temps molt menor i amb uns resultats molt més professionals. A continuació es pot veure la diferència entre el disseny inicial sense Bootstrap i el disseny final.

Fig. 9: Portal sense ús de Bootstrap

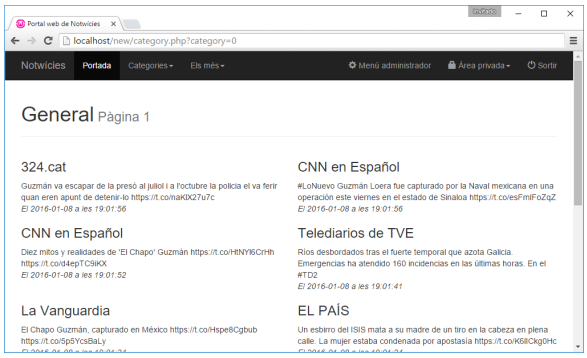

Fig. 10: Portal amb ús de Bootstrap 3.0

4.4.3 Estructura de fitxers

Per tal d'entendre les seccions que té el nostre portal web hem de conèixer l'estructura de fitxers que sosté el nostre servidor, que ens serveix per entendre com està estructurada la informació. A l'annex 3 hi ha un esquema gràfic de l'estructura de fitxers. L'arrel del servidor conté els fitxers amb les pàgines que es mostren a l'usuari i 3 directoris, que es detallen a continuació:

- bootstrap-3.3.6-dist. El directori bootstrap-3.3.6-dist és on hi ha allotjat el *framework* de disseny gràfic d'aplicacions web *responsive*. Conté documents CSS, JS (Javascript) i fonts tant de lletres com de símbols. Tots els icones i imatges gràfiques que es mostres a la interfície actual del projecte son símbols del *framework*. Això ens aporta una qualitat superior al ser adaptables a qualsevol dispositiu mantenint una interfície professional al ser únicament amb estil del Bootstrap i no dissenys externs.

- Include. Conté els fitxers que són inclosos com a parts en les pàgines principals del portal mitjançant el codi PHP:

```
<?php
    include 'include/xxx.php';
?>
```

Codi 6: Inclusió de codis en PHP

Aquestes pàgines són les de la capçalera del portal, les quals inclou el menú fixe principal superior del portal, el peu de pàgina amb la informació resumida del

portal, un script de connexió a la base de dades que s'utilitza constantment i un parell de taules usades en dos apartats del lloc web (taula de resultats dels *tweets* més votats i taula d'usuaris per a fer la cerca des del menú d'administrador).

- Operations. Conté les pàgines que en sí no mostren contingut, sinó que operen amb la base de dades i/o fan comprovacions. Inclou afegir i eliminar grups de notícies, fer el *logout* del sistema, moderar grups de notícies no validats, modificar relacions entre usuaris del portal i grups de notícies, modificar usuaris i operar amb el recol·lector des de la interfície web.

4.4.4 Qualitat de pàgines

Per tal d'assegurar una qualitat superior al nostre projecte web, aquest ha estat desenvolupat seguint les regles de sintaxi estrictes que estableix la W3C. Totes les pàgines interpretables per navegadors web del projecte han estat desenvolupades i posteriorment refinades perquè hi hagi 0 errors segons el validador oficial de HTML5, pel que ens asseguren un òptim rendiment en tots els dispositius que suportin aquest llenguatge.
Tot el codi ha estat identat correctament i amb espais en comptes de tabuladors perquè el codi es vegi de la mateixa manera independentment de l'editor de text o navegador amb la funció d'inspeccionar elements que utilitzem.

4.4.5 Seguretat web

Com a mesures de seguretat hem implementat diversos mòduls que ens ajuden contra atacs d'impersonació i per a millorar l'autenticació de l'usuari així com evitar atacs de *man-in-the-middle*.

Protecció de contrasenyes Un dels problemes que es va observar ràpidament era que quan administràvem la base de dades, si ens situàvem a sobre de la taula 'user' que conté tota la informació dels usuaris podíem veure en clar la seva contrasenya. A part d'una dubtosa legalitat, el fet de tenir aquestes contrasenyes té una responsabilitat molt gran, doncs els usuaris poden estar utilitzant les mateixes que en d'altres comptes de xarxes socials i en cas que un dia fóssim víctimes d'un atac de força major i sortissin aquestes contrasenyes seriem nosaltres els responsables legals. De totes maneres hem desenvolupat la solució per simples raons ètiques. Aquesta passa per a guardar les contrasenyes xifrades de tal manera que no estiguin en clar. Seguint l'esquema de seguretat que usa Linux en el fitxer 'shadow', s'ha aplicat la millor solució que consisteix en fer un resum de les *passwords*. D'aquesta manera ni amb un atac de força bruta podríem trobar una clau que desxifrés totes les contrasenyes. L'algorisme que hem utilitzat és el de SHA-1[14], que tot i començar a ser substituït per les variants de SHA-2, segueix sent un estàndard en quan a metodologia de resum o *hash*.
S'apliquen les funcions *hash* en dos casos, en quan ens registrem a l'aplicació web i en quan fem el *login*. Un exemple de quan fem el *login* com treballa amb aquesta informació el *back-end* de PHP és el següent:

```
\$sql = SELECT username [...]
WHERE [...] and password =
hash('sha1', \$_POST['password']);
```

Codi 7: Ús d'algorime de *hash*

Com podem veure a l'última línia mirem que el *password* que ens passa l'usuari coincideixi amb el valor del *hash* en SHA-1 d'aquest amb el valor de la base de dades. D'igual manera ho fem amb el registre d'usuaris, on inserim un registre on el *password* és el SHA-1 del que rebem de l'usuari.

password
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
069fd9de9edce3d658abee88a21387cca6eafa2a
47130e74953433924adcda4b90d9f8387a79547e
11e4b511c05a3dd8694dd82f9303a4c9454c28f7
5dd7e3ae847daabba9c2523d3e76f44e9e83b3b7

Fig. 11: Vista de les contrasenyes des de BD

Protecció de connexió La protecció anterior ens proporciona seguretat en l'emmagatzematge, però de res ens serveix si la nostra comunicació no està xifrada, doncs el càlcul del SHA-1 és fa al servidor i no a l'origen, de tal manera que està viatjant per la xarxa en clar d'un extrem a l'altre. Per protegir la comunicació només cal fer ús del protocol SSL[15] sobre HTTP, el HTTPS. Hem creat doncs un parell de certificats, un que ens serveixi com a autoritat arrel de confiança propi i un per al lloc web en concret. D'aquesta manera si canviem el nom de domini o volem fer un certificat nou no haurem de confiar en cada un d'ells sinó que ja seran de confiança pels navegadors.

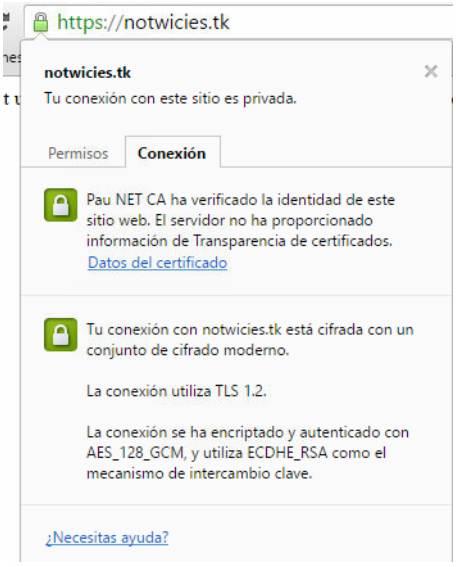

Fig. 12: Visió del certificat de confiança

Com es pot veure s'ha creat un domini gratuït anomenat "notwicies.tk"[16], necessari per a que els usuaris no hagin de conèixer l'adreça IP i per a que el certificat tingués un Common Name, camp necessari per a funcionar amb pàgines web i que requereix un nom de domini. El servidor on s'allotja és una instància de Amazon Elastic Cloud que

cubreix perfectament les necessitats del nostre portal. Podem apreciar que utilitzem un algorisme de xifrat modern i la connexió és segura perquè acceptem el certificat arrel propi autosignat de Pau NET CA.

Control d'accés a pàgines d'administració Un atac clàssic és el d'esbrinar quins fitxers del servidor porten a pàgines d'administració per tal d'accedir i modificar els paràmetres que es desitgi. Això pot ser altament destructiu si es duu a terme per usuaris que no tenen ni han de tenir el rang d'administrador.
Per a solucionar-ho s'ha afegit, a cada una de les pàgines que es requereixen privilegis d'administració un codi que mira si l'usuari que hi ha logat en aquell moment, si n'hi ha, sigui administrador. El pseudocodi que ho implementa seria el següent:

```
si(existeix sessio d'usuari) {
    si(sessio.usuari.admin == 1) {
    // si usuari logat a web és admin
        err = fals;
    } sino {
        err = cert;
    }
}
si(!(err))
    mostrar_contingut_pagina();
sino
    mostrat_error();
}
```

Codi 8: Pseudocodi de seguretat

5 Resultats

Com a resultats del projecte tenim un sistema autònom governat únicament des d'una interfície web que ens permet dur a terme els tres objectius principals: recol·lectar *tweets*, classificar-los i mostrar-los mitjançant una interfície web. A continuació s'exposen les diferents vistes de l'aplicació.

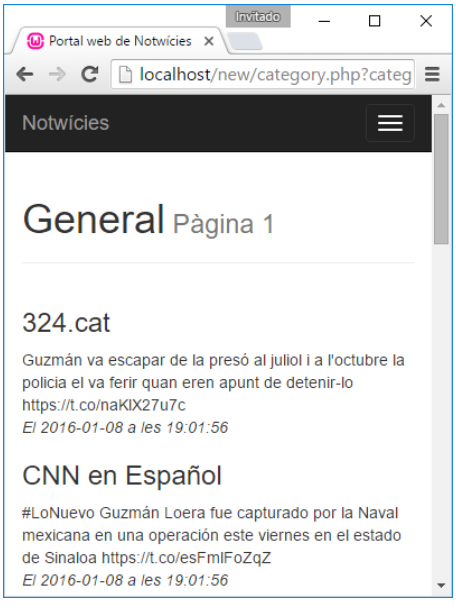

Fig. 13: Visió en dispositiu mòbil *responsive*

Podem veure a la figura 13 l'adaptació *responsive* que es fa al visualitzar-ho en un dispositiu mòbil o de baixa resolució horitzontal. Com es pot observar les notícies es mostren d'una en una i el menú principal superior es veu recollit amb un botó que genera un desplegable on es pot veure les opcions verticalment.

Podem veure també l'estil que ens aporta l'ús de Bootstrap en quan a creació de *pop-ups* desplegables dins d'una mateixa pàgina, el que ens permeten fer operacions com un *login* de manera senzilla i sense haver de desenvolupar AJAX[1] per nosaltres mateixos (figura 14).

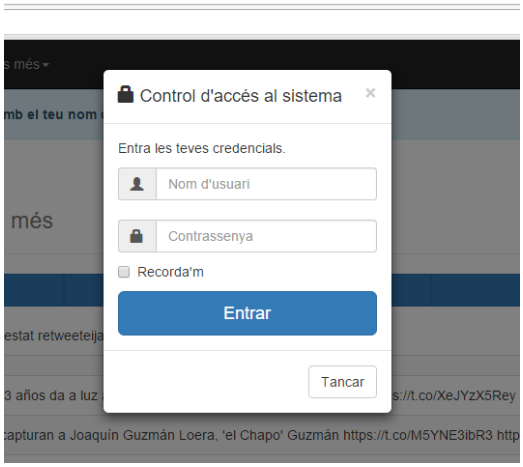

Fig. 14: Visió de desplegable login

Per tal d'evitar que l'usuari administrador hagi de tocar de codi s'ha implementat tal com s'ha comentat a l'apartat de metodologia un menú d'administrador. A la figura 15 hi ha el *dashboard* on podem seleccionar les operacions a realitzar amb el recol·lector des de la interfície.

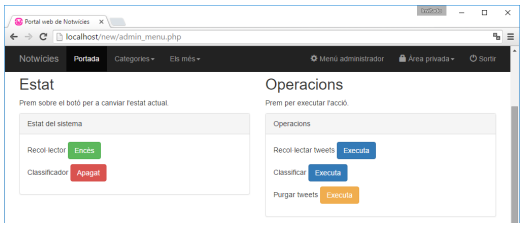

Fig. 15: Visió del *dashboard* d'administrador

6 Conclusions

Podem concloure dient que els resultats del projecte són els que s'esperaven inicialment. S'ha complet la planificació i en cada un dels terminis s'ha superat el llindar. El fet que en la primera entrega s'hagués avançat més del que estava previst ens ha permet a l'última iteració migrar tot el disseny web creat a Bootstrap 3.0.

- S'han complert els requeriments funcionals tals com recol·lectar *tweets* (100% completat), classificació de *tweets* (70-80% doncs es podria haver utilitzat un altre mètode d'intel·ligència artificial més avançat encara que no fos de la menció) i el portal web està completament funcional i desenvolupat (100% completat i ampliat) amb un paradigma de desenvolpament que

ens permet adaptar-se a tot tipus de dispositius. Tenim doncs seguint amb els objectius del treball un sistema autònom funcional que s'encarrega de mostrar notícies classificades a un portal web.

- En quan a la qualitat del programari que estava com a requeriment no funcional també s'ha complert les expectatives: el codi està altament comentat, l'estil de codificació és correcte (màxim 80 línies), alta cohesió i baix acoblament en el desenvolupament de les classes. Utilitzem un sistema de *logs* que ens ofereixen els llenguatges Python com PHP per a tenir un històric de les operacions dels usuaris i del sistema de recol·lecció per a millorar la seguretat del sistema i controlar qualsevol excepció.

A continuació s'exposen les línies futures de desenvolupament que complementen la feina feta i que no s'han pogut desenvolupar donat l'estret marge de temps del que es disposava. El compliment d'aquestes donaria un acabat amb molta professionalitat que podria donar lloc a un portal d'informació amb èxit en el món de la informació.

7 Línies futures de desenvolupament

Per tal d'ampliar el projecte i donar-li una qualitat superior hi ha un conjunt de mòduls que caldria implementar o refinar. A continuació exposo els que ampliaria per ordre:

- Personalització del disseny web. Tot i que la interfície web està dotada d'una qualitat correcta al haver estat desenvolupada al 100% amb el *framework* de Bootstrap, aquesta manca de personalització per part del projecte Twitterbot. Seria bo afegir-li elements decoratius tal com imatges o colors que el distingissin dels demés portals i donant-li personalitat a la vegada que seguís sent *responsive*.

- Millora de l'algorisme de classificació. Tal com s'ha comentat a l'apartat del classificador, hi ha d'altres mètodes que estan més a dins del que seria el camp de la intel·ligència artificial. El mètode que hem utilitzat nosaltres del diccionari està bé, però el sistema no és capaç de dur a terme un autoaprenentatge i d'acabar sent autònom en quan a la classificació.

- Gestió dels idiomes dels *tweets* i de traducció immediata. A l'àrea geogràfica on ens trobem tenim un cas clar de bilingüisme. Els *tweets* es podrien guardar a la base de dades en l'idioma que s'han extret i el camps de la base de dades de 'idioma' de la taula 'tweet' guardar quin és l'idioma. Es podria fer ús d'un dels múltiples serveis web que se'ns ofereixen gratuïts per a la traducció de *tweets*, de tal manera que un usuari establís l'idioma en que vol visualitzar tota la interfície en cas que volgués un de concret i que la informació es mostrés únicament en aquest idioma. Això seria útil tant en comunitats autònomes on no es parli en català com en països anglosaxons on no es parli ni el català ni el castellà. Aquest atribut li donaria un caràcter d'internacionalització elevat al nostre projecte.

- Sistema de valoració de classificació per part d'usuaris. Per tal d'alimentar un sistema de *Machine Learning* seria bo crear un sistema de puntuació per tal que els usuaris registrats al portal poguessin informar ràpidament si un *tweet* no ha estat ben classificat. D'aquesta manera el mòdul de classificació s'alimentaria de les valoracions dels usuaris i seria capaç d'aprendre i millorar la classificació. Mica en mica el sistema acumularia un nombre d'errors inferior fins al punt de que quasi no tingués errors. Per tal de motivar als usuaris podria haver una puntuació personal que anés augmentant cada cop que es corregeix una classificació i que els usuaris competissin per tal d'escalar el rànquing i obtenir beneficis i descomptes en diferents serveis.

- Definir més granularitat entre usuaris del portal web. Actualment només hi ha usuaris lectors i administradors. Seria convenient tenir usuaris que puguin editar notícies i grups d'informació sense arribar a tenir privilegis d'administració del sistema web complert. D'aquesta manera tindríem col·laboradors que li traurien feina a l'administrador del sistema doncs podrien validar grups de notícies i gestionar errors en aquests.

Agraïments

M'agradaria agrair al meu tutor del projecte, Jordi Casas-Roma, pels consells i guiatge al llarg del procés, a la meva família per donar-me suport durant el desenvolupament i als meus companys de sortides per entendre que no he pogut estar amb ells tant com voldria.

Referències

[1] http://www.w3schools.com/ajax/default.asp

[2] http://twitterfeed.com

[3] https://validator.w3.org

[4] https://www.quora.com/What-is-the-difference-between-Rest-API-vs-Streaming-API

[5] http://www.w3schools.com/json/json_syntax.asp

[6] https://dev.twitter.com/rest/reference/get/search/tweets

[7] http://docs.tweepy.org/en/latest/api.html

[8] https://dev.twitter.com/oauth

[9] http://mysql-python.sourceforge.net/MySQLdb.html

[10] https://docs.python.org/2/library/threading.html

[11] https://docs.python.org/2/library/socket.html

[12] http://php.net/manual/es/book.sockets.php

[13] http://www.w3schools.com/bootstrap/

[14] http://php.net/manual/es/function.sha1.php

[15] https://www.nanotutoriales.com/como-crear-un-certificado-ssl-de-firma-propia-con-openssl-y-apache-http-server

[16] http://www.dot.tk/es/index.html

[17] https://docs.python.org/2/tutorial/errors.html

[18] http://getbootstrap.com/components/

APÈNDIX

A.1 Base de dades ER

A continuació podem veure el diagrama entitat-relació de la base de dades que integra els 3 mòduls i la comunicació entre ells.

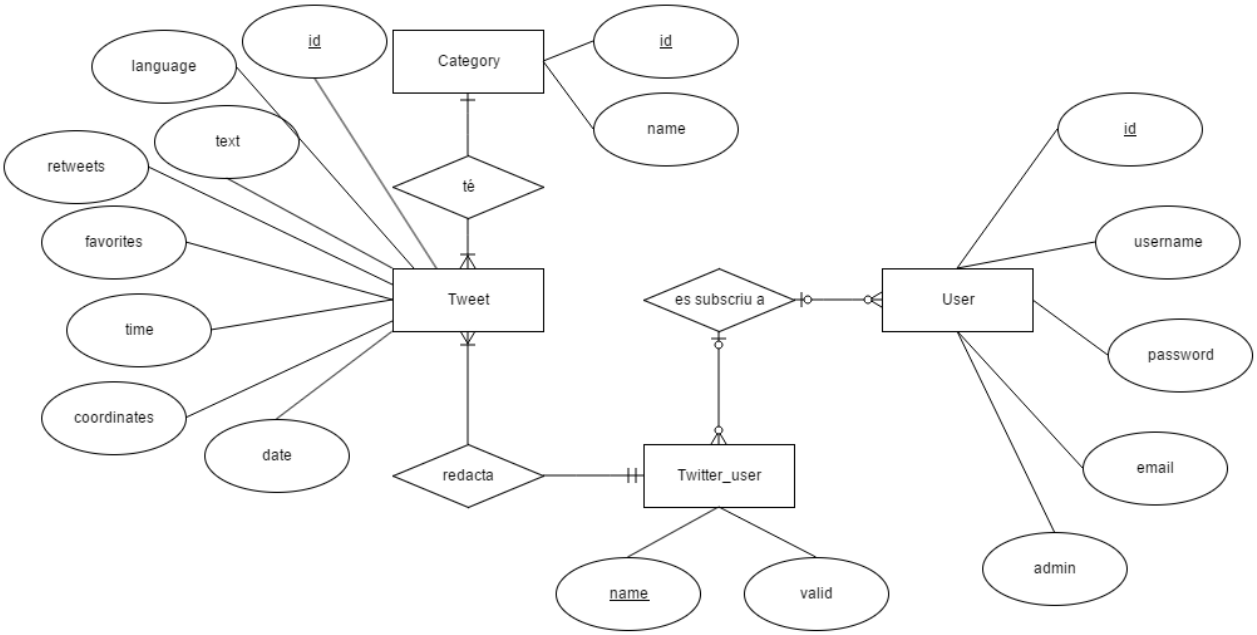

Fig. 16: Diagrama ER de la base de dades

A.2 Visió del índex del portal web

Aquí podem observar la vista en gran de la pàgina principal del nostre portal web.

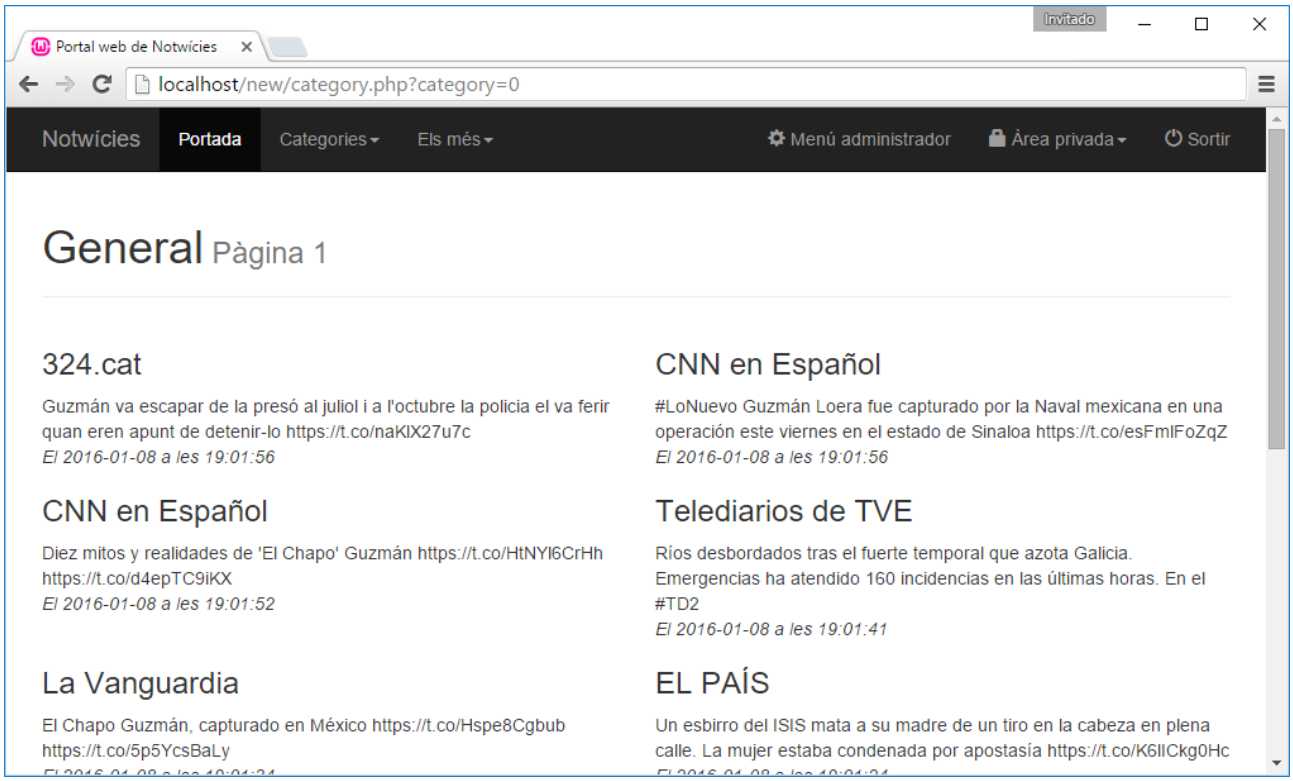

Fig. 17: Diagrama ER de la base de dades

A.3 Directori de fitxers del servidor

Aquesta és la estructura de fitxers que composa el sistema web del projecte i que està pujada al servidor.

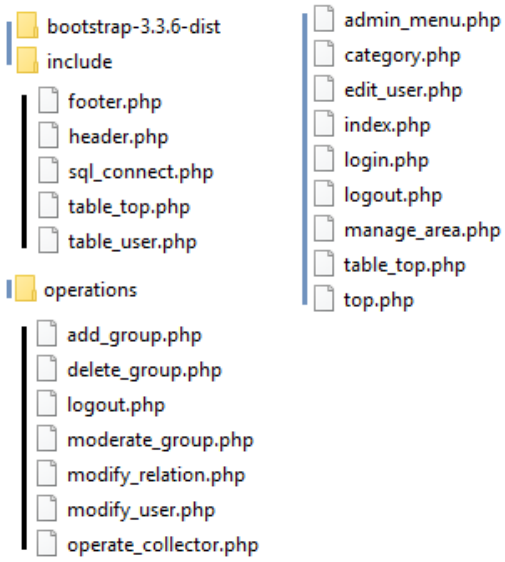

Fig. 18: Directori del servidor del portal web