RCruncher este o aplicatie web ce se ocupa cu colectarea si analiza datelor de pe reddit. Functionalitatea aplicatiei este impartita in doua arii: colectarea si analiza datelor pentru utilizatorii de reddit si colectarea si analiza datelor pentru asa numitele subreddits.

In ceea ce priveste utilizatorii, functionalitatea consta in a le oferi subreddits in care acestia ar putea fi interesati bazat pe activitatea anterioara in cadrul siteului si de a le indica utilizatorii cu interese asemanatoare. In ceea ce priveste postarile propriu-zise de pe reddit, postarea este analizata din punct de vedere semantic si din punct de vedere a analizei sentimentelor.

Din punct de vedere al originalitatii, reddit implementeaza unele dintre functionalitatile pe care le propune RCruncher dar nu ofera niciun fel de aplicatie externa spre acest scop. Ca aplicatii externe oferite de redidt, ce nu folosesc protocolul OAuth, majoritatea sunt wrappers peste API-ul expus de reddit.

Pentru a putea furniza solutii la problema propusa, aceea de a oferi recomandari si de a analiza anumite posture, am construit o aplicatie web ce consta in server-client ce serveste acest scop. Pentru inceput, se foloseste API-ul expus de catre reddit pentru a colecta datele ce tin de utilizatori, spre exemplu: posturile create si in ce subreddits si comentariile lasate si la ce posturi. Aceste date sunt mai tarziu salvate intr-o baza de date. Pe datelor colectate, cu ajutorul unei retele neuronale, se ofera predictiile, recomandariile si utilizatorii inruditi. Pentru cea de a doua parte, se foloseste un crawler si API-ul extern pentru a obtine continutul propriu zis al postarilor dupa care datele sunt analizate de catre API-ul de limbaj natural al celor de la google, datele fiind salvate anterior in baza de date anterior mentionate.

Ca si capitole ale acestei lucrari vor fi abordare urmatoarele:

* Serverul si arhitectura acestuia.
* Construirea si antrenarea retele neuronale.
* Exemplificarea folosirii API-ului celor de la Reddit si a crawler-ului.
* Exemplificarea si argumentarea folosirii API-ului de la google.
* Baza de date.
* Entitatile folosite.
* Conexiunea cu baza de date.
* Clientul si arhitectura acestuia.
* Conexiunea dintre server si client.
* Serviciile folosite in dezvoltarea aplicatiei
* Lipsuri si posibile imbunatatirii ale aplicatiei.

Serverul si arhitectura acestuia

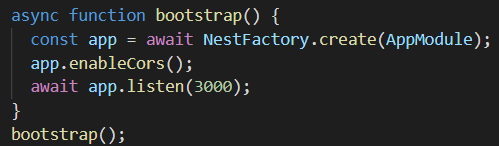
Pe partea de server am optat sa folosesc node.js si typescript in detrimentul PHP-ului.

Printre motive se enumera:

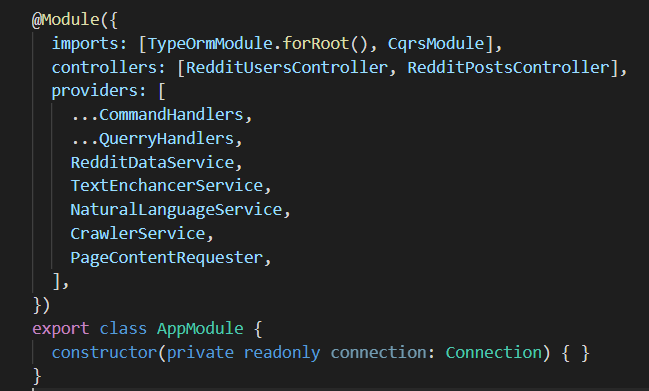
* Folosirea aceluias limbaj de programare la niveul clientului si al serverului.
* Moulele incarcate de catre node sunt descarcate si dupa initializate, ulterior fiind disponibile constant.
* Se lucreaza mai usor cu fisiere de dimensiuni mari.
* Caracterul strongly-typed al typescriptului fata de javascript
* Numeroasele module oferite de catre Node.js
* Debugging in real time

Serverul este construit cu ajutorul lui NestJS, un framework specializat pe constructia de

servere, care poate fi descris cel mai bine ca un wrapper peste Express. Hostarea acestuia are loc pe portul 3000 si reprezinta punctul de start al aplicatiei.



Aplicatia contine un singur modul in care se intializeaza dependentele acesteia, de exemplu: modulul de CQRS, conexiunea cu baza de date, controllerele si serviciile de care depinde modulul.



API-ul expus de catre server este unul de tip REST. Acesta expune doua controlere:

* Pentru userii de reddit



* Pentru posturile de pe reddit



Majoritatea metodelor expuse definite in controller-ul pentru utilizatori iar amandoua controller-ele contin metode doar de tipul POST si GET.

Pentru controller-ul ce corespunde utilizatorilor urmatoarele metode sunt expuse:

* Create, de tipul POST care primeste numele unui utilizator de reddit, creeaza entitatea corespunzatoare si o salveaza in baza de date.
* GetUserData, de tipul GET, care returneaza din baza de date un utilizator si toate datele legate de acesta.
* PredictUser, de tipul GET, care indica pozitia utilizatorului in cadul retelei neuronale.
* GetUserTopics, de tipul GET, care returneaza topicurile unui user bazate pe comentariile lasate de acestea in scopul construirii unui nor de cuvinte.
* GetTrainingSet, de tipul GET, care returneaza pozitiile tuturor utilizatorilor in cadrul retelei neuronale, utilizatori ce au facut parte din setul de initial de antrenament.
* RefreshComments, RefreshSubmitted, RefreshTopics, de tipul GET, care ii comunica serverului ca pentru un anumit utilizator se doreste innoirea respectivelor date.

Pentru controller-ul ce corespunde postarilor urmatoarele metode sunt expuse:

* CreatePost, de tipul POST, ce primeste URL unui post de pe reddit, il trimite catre analiza si ulterior catre salvarea lui in baza de date impreuna cu analiza corespunzatoare.
* GetPostWithData, de tipul GET, ce primeste URL unui post de pe reddit si returneaza date despre acesta impreuna cu analiza corespunzatoare a postarii.

Fiecare serviciu de la nivelul serverului, mai putin reteaua neuronala, este de tip Injectable, ceea ce inseamna ca, similar unui singleton, fiecare serviciu Injectabil v-a exista intr-o singura instantiere pe parcursul unei sesiuni de viata a serverului.

Toate functiile asincrone de pe server, cum ar fi aducerea datelor de pe reddit sau returnarea unor date la nivelul API-ului, folosesc tipul de date Observable din libraria Rx.js, ce reprezinta o inbunatatire de la Promise, sau patternul async, await ce sunt deseori folosite in aplicatii bazate pe node.js atunci cand este nevoie de tratarea asincronismului.

Construirea si antrenarea retelei neuronale.

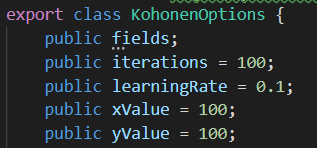
In ceea ce priveste reteaua neuronala am optat pentru una de tip SOM(self-organizing map)/ Kohonen din mai multe motive:

* Reprezinta un tip de retea neuronala cu invatare nesupervizata.
* Este disponibila alegerea unei functii de distanta customizata.
* Tipul de invatare oferit de retea este : invatare competitive in opozitie cu invatarea bazate pe corectare de erori. Am optat pentru aceasta varianta deoarece cu greu se poate vorbi de erori in cazul de recomandari sau predictii.
* Este indicata in cazul vrem sa vizualizam predictii pentru date foarte mari. Pentru o retea neuronala construita din 100 de utilizatori si 633 de sub topics reteaua neuronala, salvata sub forma unui fisier de tipul .JSON ajunge la o marime de aprox. 85 MB, in conditiile in care, in martie 2019, reddit a inregistrat 542 de milioane de utilizatorii.

Chiar daca este privita ca un serviciu, am ales sa nu ii ofer retelei neuronale caracterul de clasa

Injectabila, datorita timpului lung de incarcare si antrenare a acesteia, ci mai degraba sa o instantiez la crearea controllerului pentru utilizatori de care este strans dependenta. Astfel reteaua va fi incarcata la pornirea serverului si nu cand v-a fi nevoie de ea propriu-zis.

Setarile dupa care este construita reteaua sunt urmatoarele, urmand sa fie explicate in urmatoarele cateva paragrafe :



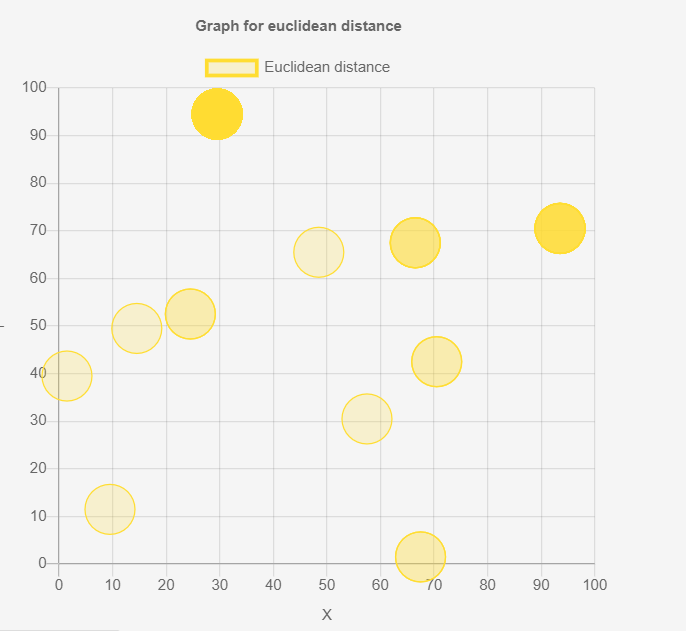
Astfel, reteaua neuronala consta in 100\*100 de celule. Iteratiile sunt de asemenea 100, ceea ce inseamna ca se va itera de 100 \* Lungimea datelor de antrenament in faza de antrenare a retelei. Coeficentul de invatare este 0.1, acesta fiind transmis mai departe catre algoritmul ce se ocupa cu invatarea.

Astfel, pentru a putea antrena reteaua sau pentru a putea produce o recomandare/prezicere este nevoie de asesarea datelor in urmatorul format: pentru fiecare utilizator este creat un json cu urmatorul tip de inregistrari : numeSubreddit : [0, PMAX], unde PMAX reprezinta numarul maxim de postari facute de orice utilizator in acel subreddit. Astfel pentru fiecare utilizator for fi parcuse toate subredditurile si create inregistrarile necesare. Daca un utilizator nu a postat niciodata intr-un subreddit, activitatea acestuia in subredditul corespunzator este considerate nula.

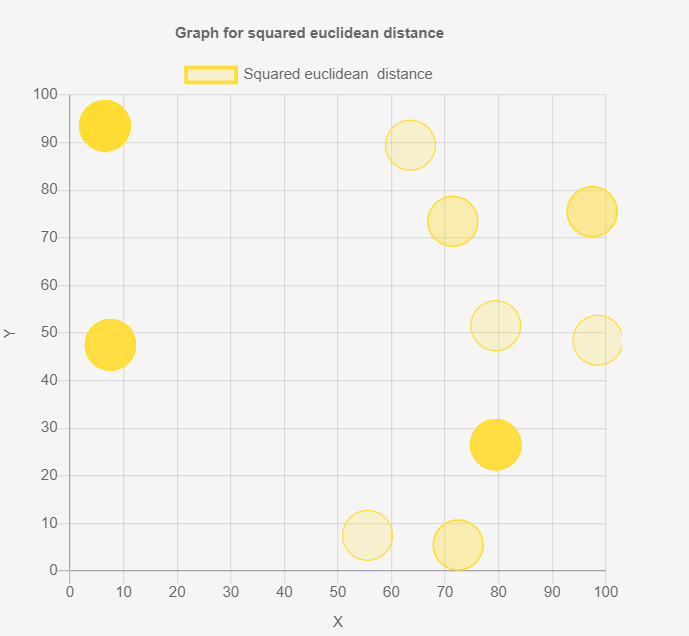


In continuare vom aborda functia de distanta dintre utilizatori si procesul alegerii acesteia.

In urmatoarele grafice este evidentiata distrbutia a 100 de utilizatori cu 633 de subreddituri, singura schimbare fiind functia de distanta.



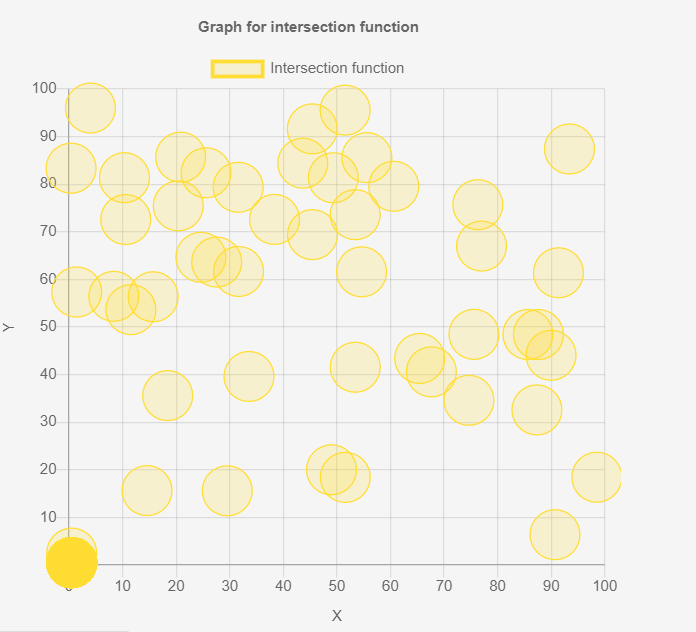
In cazul distantei euclidiene putem observa o dispersie inadecvata datelor, formandu-se doar doua clustere fixe, unul dintre ele fiind probabil pentru utilizatorii pentru care nu se inregistreaza niciun fel de postare in cadrul redditului.



In cazul radacinii patrate a distantei euclidiene, se poate constata aparitia unui al 3 cluster, tot fix, dispersia datelor in grafic fiind inca inadecvata.

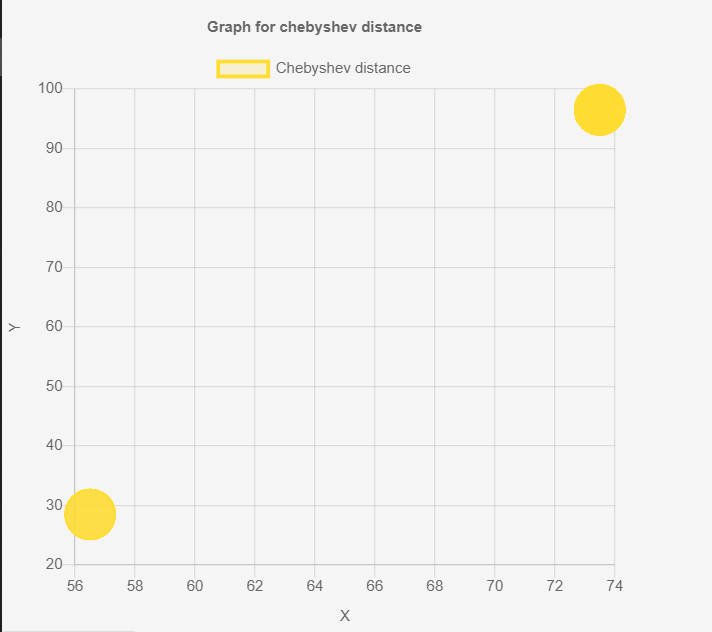
In ciuda faptului ca radacina patrata a distantei euclidiene reprezinta distanta de baza ce se foloseste in cadrul multor algoritmi de invatare automata, aici se poate observa cu usurinta gradul ridicat de inadcvare a acesteia din constructia datelor pentru fiecare utilizator.

In continuare am apelat tot la o functie de baza, si anume, intersectia intre doua inregistrari.

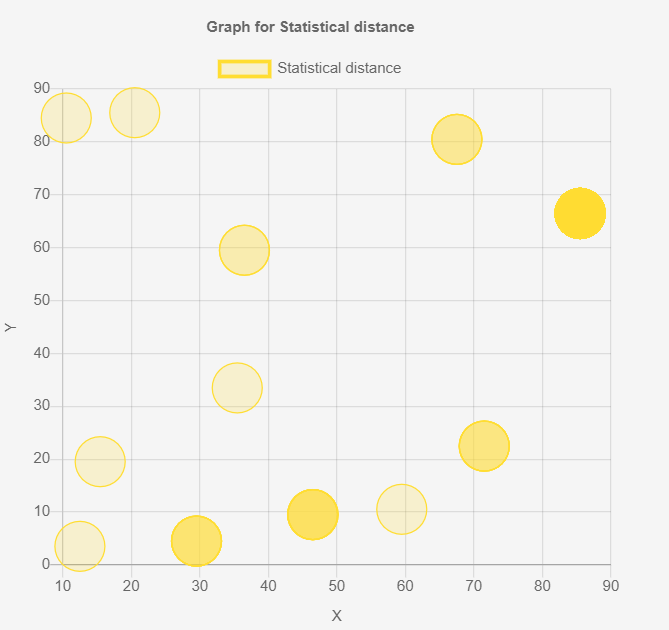


In cazul intersectiei se observa o dispersie mai buna a datelor, un singur cluster fix, si altele 2-3 clustere mai dispersate.

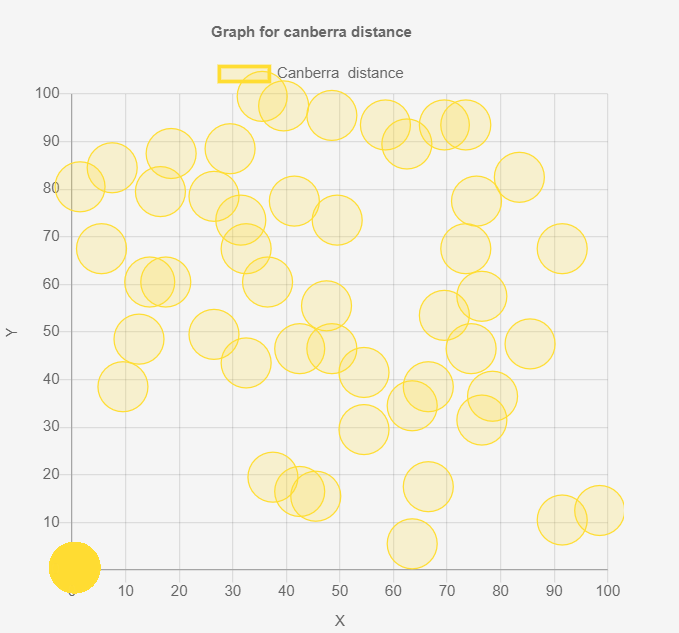
Cazul distantei Chebyshev este unul nefavorabil formandu-se doar doua clustere fixe la extremitati, din pricina modului in care se calculeaza distanta.



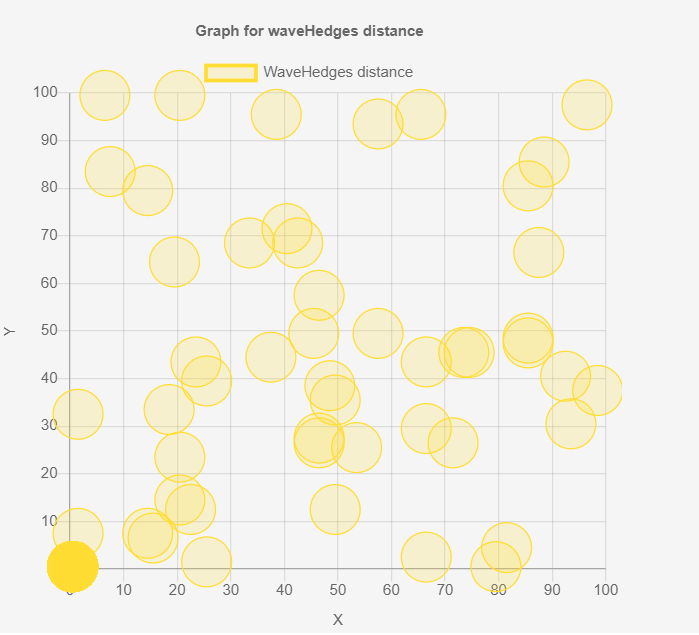
Urmatoarea distanta folosita a fost distanta statistica, rezultatele fiind destul de satfisfacatoare.

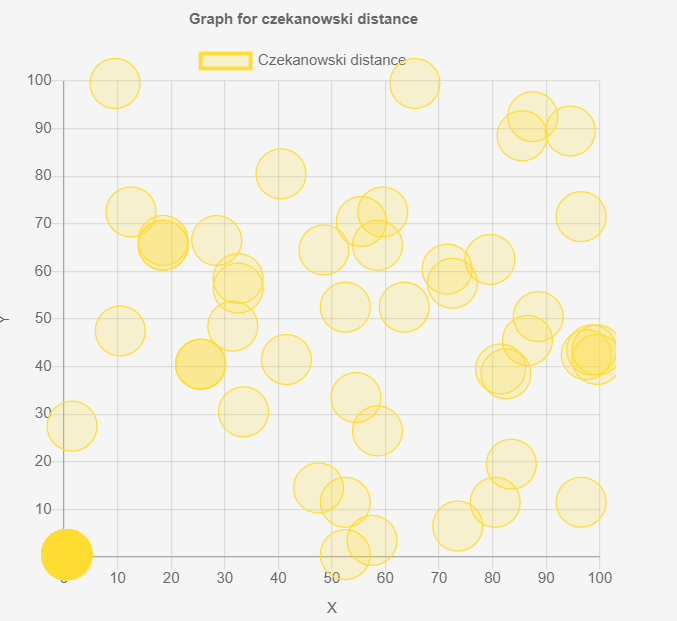


In continuare am folosit distanta Canberra insa cu mici sane de success deoarece aceasta este bazata pe distantele manhattan, ne mai fiind folosita semnificativ in invatare automata.



In cele din urma am ajuns la distantele de tipul Wave Hedges si Czekanowski care apartin familei de distante ce tin de intersectie, distante ce par cele mai potrivite pentru datele cu care se lucreaza.

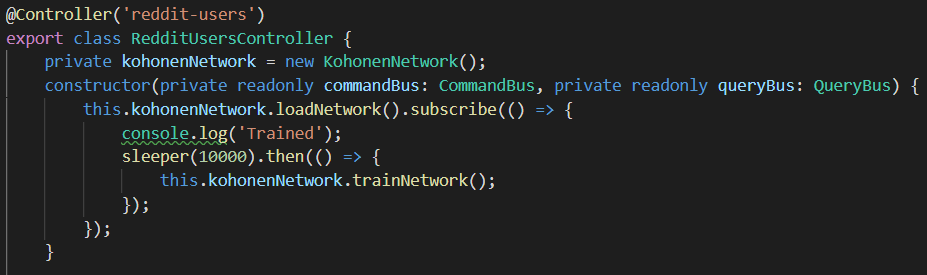




In finala distanta alesa pentru reteaua neuronala a fost cea Czeakanowski, din pricina distributiei datelor si buna clusterizare a acestora.

Astfel a fost creata reteaua Kohonen, cu ajutorul careia se vor oferi recomandari si preziceri.

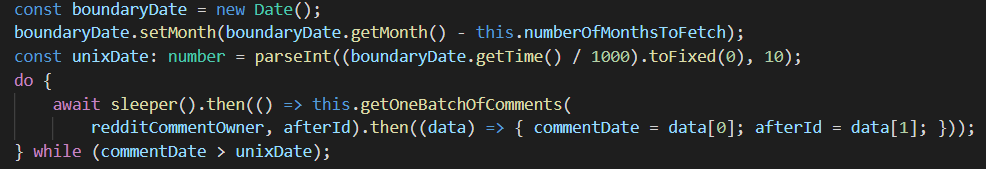
Reteaua, optiunile pentru aceasta si datele din setul de antrenament au fost folosite o singura data, dupa care reteaua a fost salvata la nivelul serverlui sub format json, urmand ca la fiecare pornire a acestuia, reteaua sa fie reincarcata din fisier si nu inceperea procesului de la capat.



Exemplificarea folosirii API-ului celor de la Reddit si a crawler-ului.

Avand in vedere ca colectarea datelor reprezinta un pas de baza din aplicatia propusa, in continuare voi exemplifica modul in care sunt obtinute datele si API-ul corespunzator al siteului reddit.

Deoarece reddit considera ca datele precum comentariile sau posturile create ar trebui sa fie publice, nu a fost nevoie de urmarea protocolului OAuth pentru a obtine aceste date. Pentru utilizatori a fost folosit numai API-ul folosit de catre reddit. Datorita volumului de date existent pe platforma, am cosiderat ca cele mai semnificative date se afla in ultimele 6 luni de activitate a utilizatorului. Astfel, cand un utilizator este creat, cele mai vechi comentarii si postari dateaza cu maxim 6 luni in urma momentului actual.

Aceste date nu sunt singurele ce pot fi colectate de catre aplicatie, fiind posibil ca, ulterior, dupa adaugarea unui user, comentariile sau postarile acestuia sa fie actualizate pana la momentul current.