MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Lifestory – Rețea de socializare pentru studenții din campusurile universitare

LUCRARE DE LICENŢĂ

Absolvent: **Marian Eugen David**

Coordonator ştiinţific: **asis. ing. Cosmina IVAN**

2016

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Marian Eugen David**

Lifestory – Rețea de socializare pentru studenții din campusurile universitare

1. **Enunţul temei:** Proiectul își propune realizarea unui sistem menit să faciliteze interacțiunea și socializarea între studenți la nivel de campusuri universitare. Sistemul își propune să ofere utilizatorilor posibilitatea de a-și crea un profil personal în care să-și adauge poze și videoclipuri. Sistemul dorește să eficientizeze modul de interacțiune între persoanele din apropiere, recomandarea de persoane realizându-se pe baza criteriilor de pasiuni și interese comune. Sistemul creat se adreseaza la nivel local utilizatorilor, interacțiunea dintre utilizatori realizându-se pe un domeniu de interes restrâns.

2. **Conţinutul lucrării:** Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretica, Proiectare și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibiliografie, Anexe.

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**: asis. ing. Cosmina IVAN

5. **Data emiterii temei:** 1 noiembrie 2015

6. **Data predării:** 30 Iunie 2016

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

În ziua de azi asistăm mai mult ca oricând la o evoluție impresionantă a tehnologiei în domeniul IT, devenind beneficiarii multor dintre aplicațiile și inovațiile aduse de companiile de profil. Ultimele noutați aduse în domeniul IT încurajează dezvoltarea de noi tehnologii open-source, reușindu-se astfel revoluționarea modului de dezvoltare a aplicaților Web.

În momentul de fața, rețele de socializare au devenit foarte populare în randul persoanelor tinere, fapt care a dus la crearea mai multor rețele de socializare pentru fiecare domeniu de activitate. O rețea de socializare trebuie să-i ofere utilizatorului posibilitatea de a interacționa cu persoane din apropiere foarte ușor și de a-și crea un spațiu virtual personal în care să-și împărtășească ideile și gândurile. Evoluția rapidă a tehnologiei impune realizarea de sisteme informatice tot mai performante și competitive.

Pentru a satisface interacțiunea utilizatorului cu sistemul, o rețea de socializare trebuie să îi ofere acestuia posibilitatea de a beneficia de toate funcționalitățile pe care le oferă și sistemele competitoare în mediul virtual, iar în plus trebuie să existe funcționalități noi pe care nu le regasim la competitori.

Pornind de la aceste premise, sistemul dezvoltat își propune să fie unul de actualitate, în care tendința tehnologiilor actuale și atenția utilizatorilor țintă sa fie captată.

1.1. Contextul proiectului

În continuarea acestui capitol va fi expusă motivarea alegerii temei pentru proiectul actual, modul prin care aceasta se adapteaza la nevoile omului dinamic, modern și dornic să afle informații noi și necesitatile pe care le îndeplineste proiectul dezvoltat.

Într-o lume în care tehnologia este prezentă peste tot in jurul nostru, în care accesul la informație se face foarte ușor se dorește realizarea unui sistem care să faciliteze utilizatorilor posibilitatea de a urmări activitatea virtuală a altor persoane pe care le are în lista de prieteni. De exemplu, în contextul unui campus univesitar existența unei rețele de socializare la nivel local este foarte folositoare, deoarece interacțiunea are loc doar între persoanele din apropiere, interacțiunea stabilindu-se pe baza pasiunilor și intereselor comune. Aplicația, pe langa partea de socializare poate fi folosită de utilizatori pentru a publica si materiale scolare.

Sistemul propus se încadrează atat în domeniul rețelelor de socializare, deoarece facilitează socializarea cu alte persoane, dar și în domeniul profesional. Sistemul se adresează tuturor actorilor care sunt implicați în interacțiunea cu acesta, de la studenți până la elevii de liceu. Procesul de socializare îi permite utilizatorului să-și creeze un profil personal, să-și adauge amintirile și pasiunile pe pagina de profil. Pe baza unitații de învățamânt adăugate de utilizator, acesta va putea interacționa în mod direct cu persoanele care se află la aceiași instituție de învătământ și au în comun una sau mai multe pasiuni. Elevii și studenții beneficiază cel mai mult de această aplicație, deoarece aceștia pot afla mai multe detalii despre persoanele cu care interacționeaza zilnic în unitățile de învățământ sau în campusurile universitare și pot publica materiale scoalare în grupurile în care au fost adaugați.

1.2. Motivația

În momentul de fața există foarte multe rețele de socializare pe piața, care îi permit utilizatorului să-și creeze un profil personal, să-și împărtășească conținutul personal, să-și adauge fotografii sau videoclipuri pe pagina de profil și să-și exprime gandurile și ideile. Cele mai mari rețele de socializare (Facebook, Twitter, MySpace si Instagram) sunt orientate spre a îndeplini alte cerințe ale utilizatorului, dar problema pe care am identificat-o la fiecare rețea de socializare este modul în care se adresează utilizatorilor și la ce nivel. Facebook este o rețea de socializare la nivel global, având aproximativ 1 miliard de utilizatori la nivel mondial, urmată de Twitter si Myspace cu aproximativ 500-600 de milioane de utilizatori activi la nivel mondial. Datorită numarului impresionant de utilizatori activi pe care îi are fiecare rețea de socializare, fiecare dintre acestea lucrează cu cantități mari de date în fiecare zi, fiecarui utilizator afisându-i-se pe pagina de homepage postările relevante ale prietenilor pe care îi are in listă. Principala problema pe care am identificat-o la fiecare rețea de socializare existentă, este nivelul la care aceasta se adresează utilizatorilor. Pe Facebook este puțin mai complicat să cunoști persoane din apropierea ta care au anumite lucruri în comun cu tine, deoarece în lista persoanelor pe care le-ai putea cunoaște sunt sugerate persoane pe baza numarului de prieteni comuni sau pe baza orasului actual în care locuiți, deci de multe ori în lista respectivă vor aparea persoane pe care nu le-ai întalnit niciodata sau nu le-ai vazut deloc, într-un mod asemanator procedeaza și site-ul de socializare Twitter si MySpace.

Principalul scop al acestei aplicații este de a crea spații virtuale pentru a permite interacțiunea între utilizatori la nivel loc, la nivel de campusuri universitare, licee sau firme. Crearea unei astfel de aplicații va oferi utilizatorilor posibilitatea de a interacționa și de a cunoaște mult mai ușor persoanele din jur. Aplicația va fi lansată initiala doar pentru studenții din campusurile din Observator pentru a urmări cerințele utilizatorilor și nevoile acestora. Aplicația dorește să ofere studenților, elevilor și persoanelor din interiorul firmelor posibilitatea de a avea un profil personal pentru a putea să-și adauge poze, videoclipuri și să-și exprime ideile si gandurile. Fiecare utilizator specifică campusul din care face parte și caminul în care locuieste, acesta fiind automat inclus in grupul destinat acelui camin, având acces la toate resursele disponibile în acel spațiu. De asemenea aplicația va putea fi accesată și de cadrele didactice ale facultaților pentru a putea publica note, cursuri și alte informatii care sunt imporatante pentru utilizatori si care trebuie să fie în atenția acestora. Se dorește crearea unei aplicații care să înglobeze toate mecanismele existe într-ul singur sistem pentru a-i oferi utilizatorului o experiența de navigare mult mai bună și posibilitatea de a găsi toate informațiile pe care le caută într-un singur loc, fara a fi nevoie să navigheze pe o mulțime de site-uri pentru a gasi o anumita informatie.

1.3. Conținutul lucrării

În următoarele paragrafe este prezentată structura lucrării, capitolele și conținutul acestora.

Capitolul 1 – Introducere – Acest capitol prezintă o scurtă descriere a contextului problemei care se dorește să se rezolve prin intermediul sistemului realizat.

Capitolul 2 – Obiectivele Proiectului – Cuprinde prezentarea și descrierea sumară a obiectivelor principale și secundare corespunzatoare proiectului implementat. Tot în acest capitol va fi prezentată și tema proiectului.

Capitolul 3 – Studiu Bibiliografic – Acest capitol descrie pe scurt conceptul de rețea de socializare și principiile care stau la baza acestora. Vor fi prezentate principalele rețele de socializare identificate si o comparație între sistemul realizat și sistemele similare cu privire la functionalitățile pe care le oferă fiecare produs.

Capitolul 4 – Studiu și Fundamentare Teoretica – În acest capitol sunt descrise pe scurt tehnologiile folosite pentru implementarea aplicației Web. Tot în acest capitol sunt prezentate cerințele funcționale, cerințele non-funcționale, dar și cele mai importante cazurile de utilizare identificate ca fiind importante în funcționarea sistemului.

Capitolul 5 – Proiectare de Detaliu și Implementare – În acest capitol este prezentată arhitectura sistemului și este descris în detaliu fiecare modul care intra in componența sistemului. Tot în acest capitol sunt prezentate diagramele de clase, diagrama de deployment a sistemului și diagrama de pachete a sistemului. Fiecare componentă importantă a sistemului este explicată in detaliu, iar la finalul acestui capitol este prezentat modelulul bazei de date folosit pentru a asigura persistența datelor și pentru a indeplini cerințele funcționale și non-functionale ale sistemului.

Capitolul 6 – Testare și Validare – În acest capitol sunt prezentate metodele folosite în testarea aplicatiei și rezultatele obtinute în urma testarii. Testarea și validarea aplicației a fost facută cu scopul de a testa și valida o parte din cerințele functionale și cerințele non-functionale ale sistemului.

Capitolul 7 – Manual de Instalare și Utilizare – Acest capitol prezintă resursele software necesare pentru instalarea sistemului. Tot în acest capitol este prezentat manualul de utilizare corespunzator aplicației.

Capitolul 8 – Concluzii – În acest capitol sunt prezentate obiectivele realizate prin implementarea sistemului și eventualele dezvoltări ulterioare ale sistemului.

Capitolul 2. Obiectivele Proiectului

În acest capitol vor fi prezentate obiectivele propuse spre a fi realizate de acest sistem. Aplicația dezvoltată operează la nivel local (campusuri universitare, licee) și este utilizată pentru stabilirea de prietenii virtuale pe baza pasiunilor și intereselor comune, întreaga aplicație fiind orientată pe nevoile și cerințele utilizatorului.

2.1. Obiective principale

Obiectivele principale ale acestui proiect sunt definirea și implementarea unui rețele de socializare care să faciliteze interacțiunea în mediul virtual mult mai usor, interacțiunea dintre utilizatori realizându-se la nivel local pe baza pasiunilor și a intereselor comune. De asemenea aplicația dorește să ofere utilizatorilor suport pentru publicarea de documente folosite în scop educativ, aceste obiective putând fi atinse prin realizarea obiectivelor secundare care urmează să fie descrise mai jos.

2.2. Obiective secundare

Primul obiectiv secundar pe care sistemul dorește să-l atingă este reprezentat de eficiența în utilizare a sistemului. Eficiența de utilizare a sistemului este masurată prin gradul de expresivitate al interfeței utilizator și prin consistența acesteia. În momentul în care un utilizator este familiarizat cu design-ul sistemului și este capabil să interacționeze cu acesta fara a primi vreun ajutor putem spune că sistemul este eficient.

Un alt obiectiv este reprezentat de faptul că se dorește realizarea unei aplicații care să ofere posibilitatea de dezvoltare ulterioară. În general un sistem nu poate fi complex de la primul *release,* tot timpul acesta trebuind să ofere posibilitatea de a fi extins foarte usor, fară un efort prea mare. O altă caracteristică importantă a sistemului o reprezintă securitatea, sistemul trebuie sa ofere utilizatorului certitudinea că navigarea este securizată și sigură.

De asemenea sistemul trebuie să le ofere utilizatorilor posibilitatea de a-și crea un cont, de a se autentifica și de a-și recupera parola în cazul in care aceasta este uitată. Obiectivul propus este ca fiecare cont creat trebuie să fie activat pentru a beneficia de functionalitățile aplicației, activarea contului realizându-se prin accesarea adresei de email și acționarea adresei furnizate de sistem pentru activare. Dacă contul creat nu va fi activat în decurs de 3 zile se va șterge.

Sistemul trebuie sa fie capabil să le ofere utilizatorilor posibilitatea de a-și crea un profil personal , de a căuta și adăuga anumite persoane în lista de prieteni, de a comunica cu persoanele din lista de prieteni folosind chat-ul și de a interacționa cu alte persoane pe baza pasiunilor și intereselor comune. Deoarece socializarea se realizează la nivel de campus studențesc, toți utilizatorii din acelasi campus reușesc să cunoască mult mai multe persoane cu pasiuni și interese comune decât ar face-o pe restul rețelelor de socializare.

Sistemul trebuie să poate fi utilizat de mai multe tipuri de utilizatori. Tipurile de utilizatori care pot folosi sistemul sunt administratorul de sistem, utilizatorul care deține un cont valid în aplicație și utilizatorul anonim care se afla la prima vizită în aplicație. Fiecare tip de utilizator trebuie să beneficieze de anumite functionalități pe care sistemul

le oferă, functionalități care sunt considerate obictive de implementare. Structurarea aplicației în functie de tipul utilizatorului implică acces diferit la resursele sistemului, iar unul din obiectivele secundare propuse este ca sistemul să aiba capacitatea de a filtra resursele disponibile în funcție de tipul utilizatorului care le solicită.

Un alt obiectiv secundar foarte important este ca utilizatorii autentificați în aplicație să aibă posibilitatea de a raporta un cont utilizator ca fiind abuziv sau fals. În această situație administratorul trebuie să aibă posibilitatea de a verifica contul raportat și să-l blocheze sau să-l șteargă dacă este cazul.

- **Managementul profilului personal**

Oferirea posibilității de management asupra profilului personal oferă o interacțiune utilizator ridicată. Gestionarea profilului personal presupune adăugarea unei fotografii de profil, adăugarea unei fotografii de copertă, adăugarea unei fotografii care să fie în centrul atenției, adăugarea unei melodii preferate și adăugarea unei postari pe pagina de profil. Într-o rețea de socializare utilizatorii pot fi identificați prin fotografia de profil pe care și-au încărcat-o, dar pentru a personaliza și mai mult conținutul paginii de profil, sistemul oferă utilizatorului posibilitatea de a se exprima și prin încărcarea unei fotografii de copertă. Deoarece într-un astfel de sistem utilizatorii încarca foarte multe fotografii și videoclipuri și reacționează la fotografiile și videoclipurile încărcate prietenilor, sistemul permite utilizatorului de a-și adăuga sau selecta o fotografie/videoclip care să fie în centrul atenției de fiecare dată când o persoană vizitează profilul acestuia.

- **Managementul fotografiilor și videoclipurilor**

Într-o lume în care rețelele de socializare gestionează milioane de fotografii pe zi venite de la milioane de utilizatori, sistemul proiectat trebuie să ofere posibilitatea utilizatorilor de a-și încărca propriile fotografii și videoclipuri în aplicație. Oferirea posibilității de management asupra conținutului încărcat îi oferă utilizatorului o satisfacție mai bună și un grad de interacțiune mai ridicat. Fiecare utilizator trebuie să aibă posibilitatea de a-și încărca o fotografie pe profilul personal, de a seta anumite drepturi de confidențialitate asupra acesteia și de a o șterge în cazul în care nu mai dorește ca aceasta să apară pe pagina de profil. În cazul în care fotografiile încarcate de utilizatori nu respectă politicile sistemului sau sunt raportate de alți utilizatori, administratorul trebuie să aibă posibilitatea de a interveni pentru a analiza situația și de a șterge fotografia dacă este necesar.

- **Managementul persoanelor din lista de prieteni**

Sistemul trebuie să ofere posibilitatea fiecarui utilizator de a folosi funcția de căutare de persoane dupa pasiunile și interesele comune. Pentru a facilita legătura dintre două persoane, sistemul oferă utilizatorului posibilitatea de a adăuga în lista de prieteni o anumită persoană. Din momentul în care între două persoane s-a realizat o relație de prietenie, fiecare persoană a relației va primi notificări referitoare la activitatea celeilalte în aplicație.

Capitolul 3. Studiu Bibliografic

În acest capitol se realizează o analiză și o evaluare a sistemelor similare existente pe piața, a domeniului de activitate al acestora și a functionalităților pe care le oferă utilizatorilor.

3.1. Conceputul de rețea de socializare

O rețea de socializare reprezintă o metoda de extindere a contactelor sociale prin efectuarea de conexiuni între indivizi. Dezvoltarea accelerată a internetului și a Web-ului a facilitat dezvoltarea unor astfel de sisteme care facilitează comunicarea între persoane aflate la departare, chiar și pe continente diferite. Concepul de separare afirmă că oricare doi oameni de pe planeta ar putea intra în contact printr-un lanț de cel mult 5 persoane intermediare. O rețea de socializare este cunoscută uneori sub numele de graf social care ajuta oamenii să se conecteze între ei pe baza anumitor criterii, lucru care nu ar fi posibil fară existența acestora [1].

În funcție de platforma de socializare, membrii au posibilitatea de a contacta oricare alt mebru al platformei folosind chat-ul. Există unele platforme în care comunicarea poate fi stabilită doar cu persoanele cu care există o legatură sau pe baza unor anumite criterii în care fiecare utilizator trebuie să-și exprime acceptul asupra inițierii comunicării.

3.2. Sisteme similare

3.2.1. Facebook

Facebook este cea mai populara rețea de socializare, cu un total de un miliard de utilizatori activi lunar care sunt responsabili de genererea a un trilion de like-uri, 200 de miliarde de fotografii și 17 miliarde de check-in. Pentru a accesa rețeaua de socializare, utilizatorul trebuie să dețină un cont valid. Pentru a facilita căutarea de persoane și integrarea cu alte sisteme, Facebook a creat Graph API. Acest API implementat de Facebook folosețte structuri de date de tip graf pentru a reprezenta relațiile dintre persoane, muzica favorită, fotografiile încarcate și locurile vizitate. Pentru a securiza API-ul Facebook folosește ca masură de securitate OAuth 2.0 .

Pentru a acoperi diferite secțiuni ale infrastructurii, Facebook folosește o varietate de tehnologii, majoritatea dintre acestea fiind create de ei, fiind open source. Arhitectura Facebook a fost creată pentru a fi stateless și distribuită, sesiunea utilizatorului nefiind dependentă de server, deci fiecare cerere poate fi servită de orice server din cluster. Layer-ul de prezentare este implementat în PHP, iar în spatele acestuia se afla un load balancer care împarte traficul spre servere. Multe din serviciile din back-end sunt realizate ca servicii, folosindu-se principiul SOA (Software Oriented Architecture), serviciile fiind implementate în limbaje precum C++, Java, Erlang si Python. Pentru optimizarea aplicației s-a ales translaterea codului din PHP in C++ prin construirea tehnologiei HipHop, dupa translatere codul urmând să fie compilat.

Pentru a îmbunătăți experiența utilizator și timpul de raspuns pentru fiecare cerere, Facebook folosește un nivel de caching pentru a reduce numarul de cereri care

ajung la bazele de date. Pentru a implementa nivelul de caching s-a folosit MemCached, o baza de date distribuită, datele fiind stocate in memoria RAM. Arhitectura Facebook se bazează pe principiul de izolare, reușind astfel să limiteze eșucurile la un numar foarte mic. Pentru a asigura persistența datelor, nivelul de date folosește MySQL si HBase. In figura, Fig 3.1 este prezentată arhitectura conceptuală a aplicației Facebook [2].

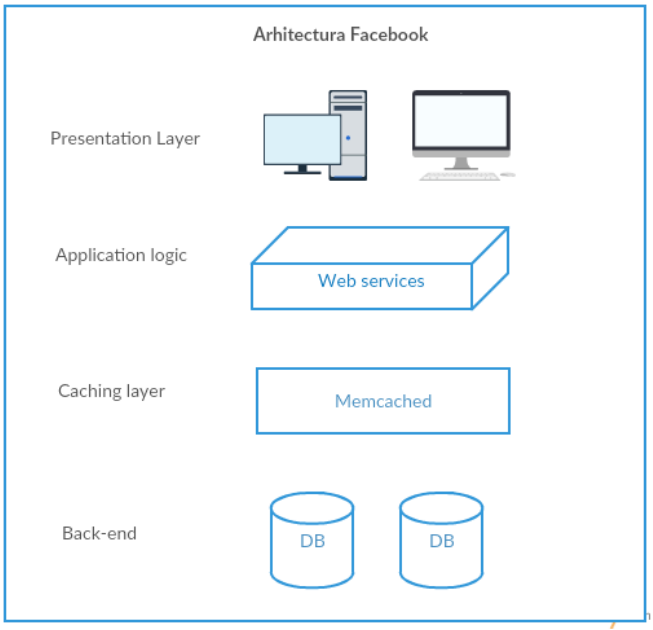

Fig. 3.1 Arhitectură Facebook

Pentru a stoca fotografiile încarcate de utilizatori, Facebook a dezvoltat un CDN numit Haystack. Mecanismul dezvoltat este foarte performant și este implementat pe baza protocolului HTTP, stocarea imaginilor realizându-se sub forma unor obiecte generice. In figura, Fig 3.2 este prezentata structura CDN-ului Haystack :

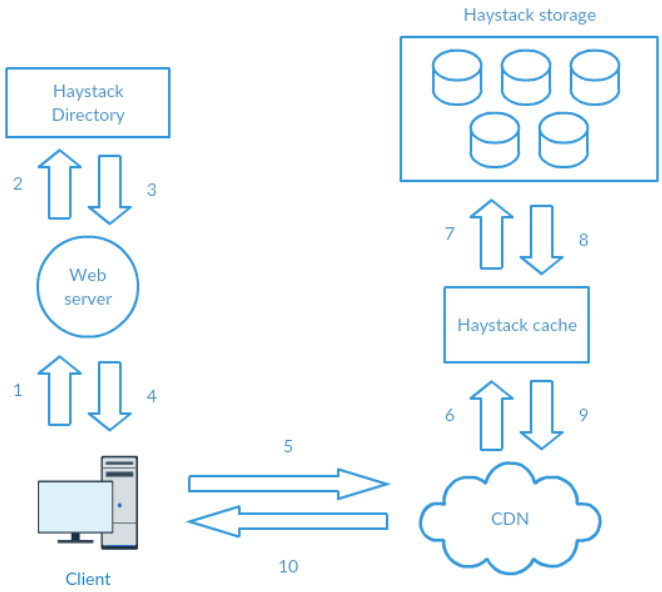

Fig. 3.2 Arhitectură Haystack

3.2.2. Twitter

La mijlocul anului 2012, Twitter a ajuns la 500 de milioane de utilizatori activi lunar, o treime din aceștia fiind din America. Cea mai folosită funcționalitate a platformei sunt tweet-urile, utilizatorul reușind prin intermediul tweet-urilor să-și exprime gandurile și locurile pe care le-a vizitat. Pentru a facilita comunicarea dintre client si server Twitter a creat un API bazat pe arhitectura REST, schimbul de mesaj între client și server realizându-se prin folosirea formatului JSON. Pentru fiecare cerere inițiată de client, API-ul returnează o colecție de date creată dupa anumite principii pentru a defini resursele disponibile pe servere și modul de acces asupra acestora. Twitter API este orientat pe protocolul HTTP și permite executarea de cereri de tip *GET*, *POST* si *DELETE* prin folosirea Streaming API, REST API si Search API.

Twitter a implementat diferite metode de securitate în funcție de API-ul folosit. De exemplu REST API suporta atât cereri care nu necesită un mecanism de autorizare, dar și cereri care sunt securizate prin folosirea mecanismului OAuth în care fiecare cerere trebuie să fie insotita de un token. Search API nu necesită mecanisme de autentificare deoarece este folosit pentru a căuta anumite resurse de pe server și poate fi folosit de orice tip de utilizator (autentificat sau anonim). Streaming API folosește ca mecanisme de autentificare OAuth și HTTP Basic care presupun ca fiecare cerere să fie însotita de un nume de utilizator și o parolă validă [2]. În figura, Fig. 3.3 este prezentată arhitectura conceptuală a sistemului Twitter.

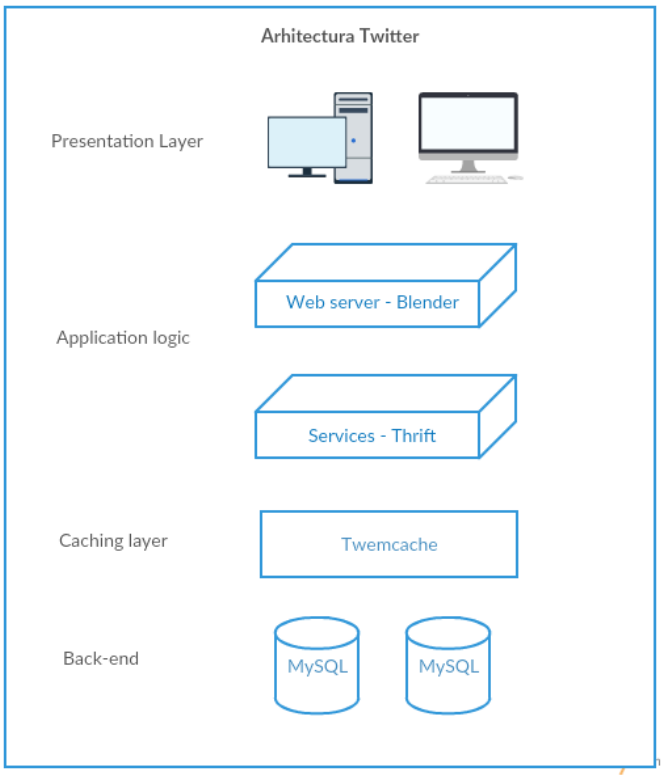

Fig. 3.3 Arhtiectură Twitter

Pentru a îmbunatăți timpul de raspuns, Twitter și-a dezvoltat propriul mecanism de caching numit Twemcache, o versiune mai îmbunătățită a Memcache permițând o gestionare și o monitorizare mai bună a serverelor de caching. Pentru persistența datelor

Twitter folosțe MySQL pentru seturi mici de date deoarece este rapid și usor de utilizat. Pentru a extinde funcționalitățile bazei de date MySQL s-au creat T-bird si T-flock peste framework-ul Gizzard, un framework folosit pentru stocarea distribuita a datelor, pentru gestiunea sharding-ului, a replicarii datelor și pentru gestiunea job-urilor care sunt planificate în background. Alte unități de stocare folosite sunt Cassandra si Hadoop, cel din urmă oferind scalabilitate, redundanță și posibilitatea de procesare a seturilor mari de date [2].

Numarul de tweet-uri a crescut considerabil în ultima perioadă, iar datorită traficului și numarului de cereri foarte mare Twitter s-a confruntat cu o problema foarte mare în privința scalabilității. Arhitectura Twitter nu era pregatită pentru a gestiona un numar atât de mare de cereri și pentru a stoca concurent atât de multe tweet-uri în baza de date. Pentru partea de front-end Twitter folosea până în anul 2011 Ruby, acesta fiind schimbat cu Blender, un limbaj care interacționează cu Apache Lucene (engine folosit pentru căutare), reușind astfel să reducă timpul de căutare ca fiind de trei ori mai puțin ca înainte [2].

3.2.3. Yammer

Creată spre sfarșitul anului 2008, Yammer este o rețea de socializare creată pentru domeniul afacerilor, în care angajații primesc continut, conversații și informații despre afaceri. Aceasta rețea de socializare se adresează unui alt domeniu de socializare, reușind să ofere suficientă siguranță pentru a capta interesul companiilor, oferind o creștere în productivitate a companiei printr-o colaborare simpla între angajați și prin posibilitatea de a luat decizii și de a gestiona noi provocari foarte ușor prin folosirea plaformei. Deoarece platforma se adresează domeniului privat, autentificarea în aplicație este posibilă doar prin existența unui cont de email valid oferit de compania la care lucrează utilizatorul.

Spre deosebire de restul rețelelor de socializare, Yammer este disponibilă într-o versiune gratuită, dar cu funcționalități limite. Pentru a beneficia de toate funcționalitățile platformei, utilizatorii trebuie să plateasca.

Securitatea platformei este asigurată prin folosirea mecanismului OAuth 2.0 pentru autentificarea utilizatorilor. Yammer a implementat Realtime API(JSON HTTP API)pentru a livra mesajele în timp real către utilizatori. API-ul a fost implementat prin folosirea protocolului Bayeux, dar la fel ca și în cazul platformei Twitter, Yammer foloseste REST API, accesul la resurse realizându-se prin apeluri JSON.

Limbajele de programare folosite pentru implementarea platformei sunt Scala, Java, Ruby, C, Javascript, Erlang si Objective-C. Exista peste 20 de servicii implementate în Java folosind Dropwizard, mecansimul fiind folosit pentru a crește performanțele JVM-ului prin arhivarea intregului ecosistem de librari și fișiere într-un singur fisier, reușindu-se astfel reducerea timpul de dezvoltare pentru un serviciu [2].

3.2.4. Foursquare

Lansat la începutul anului 2009, Foursquare este o platforma care permite utilizatorilor să salveze și să distribuie locurile pe care le-au vizitat. Aplicația este folosită de aproximativ 25 de milioane de utilizatori care înregistrează aproximativ 3 miliarde de check-in-uri zilnic. Foursquare ofera un API care permite integrarea aplicației cu alte aplicații pentru a face mai usoară navigarea utilizatorului.

Foursquare oferă doua platforme, Mechant si Venue, ambele utilizând Real-time API pentru a permite livrarea notificărilor cu privire la locația curentă către ceilalți utilizatori în timp real.

Spre deosebire de alte platforme, Foursquare a oferit o privire de ansamblu asupra arhitecturii și a limbajelor de programare folosite în implementarea aplicației. Aproximativ în totalitate front-end-ul este scris in Scala, iar API-urile sunt scrise folosind framework-ul Lift, un framework scalabil, securizat, modular și usor de întretinut pentru dezvoltarea aplicațiilor Web. Pentru persistența datelor s-a ales o baza de date NoSQL, MongoDB, iar ca mecanism de caching s-a folosit Memcached. Pentru a obține un timp de răspuns mult mai bun în cazul în care utilizatorul dorește să caute anumite vizite, utilizatori sau evenimente s-a folosit Solr și ElasticSearch împreună cu Apache Lucene [2].

3.2.5. Comparație

În tabelul 3.1 este prezentată o evaluare comparativă între tehnologiile folosite în implementarea Lifestory și tehnologiile folosite în implementarea rețelelor de socializare similare.

Caracteristici	**Lifestory**	**Facebook**	**Twitter**	**Yammer**	**Foursquare**
Limbaje de programare folosite	JavaScript	PHP, Java, Erlang, C++, Python	Ruby, Scala, Java, Javascript, C	Scala, Java, Erlang, C	Scala, Java, Javascript
Scalabilitate	Orizontala	Orizontala	Orizontala	Orizontala	Orizontala
Mecansim de caching folosit	Redis	Memcache	Twemcache	In-memory cache	Memcache
Bază de date	Cassandra / Neo4j	HBase/ MySQL	Cassandra/ MySQL	Berkley DB	MongoDB
Metodă de interogare	Nu	MapReduce	MapReduce	Nu	MapReduce
Autentificare	OAuth	OAuth	OAuth	OAuth	OAuth

Tabel 3.1 Comparație între arhitecturile sistemelor similare

Pentru a realiza o comparație între arhitectura sistemului dezvoltat și arhitectura sistemelor similare am ales ca principale caracteristici limbajele de programare în care sunt dezvoltate sistemele, tipul de scalabilitate pe care îl oferă sistemele, mecanismul de caching folosit, mecanismele utilizate pentru a asigura persistența datelor, metodele de interogare folosite asupra unui set mare de date și mecanismele folosite pentru a asigura securitatea aplicației.

În tabelul 3.2 este prezentată o evaluare comparativă între sistemul dezvoltat și sistemele similare pe baza funcționalităților oferite.

Funcționalitate	**Lifestory**	**Facebook**	**Twitter**	**Yammer**	**Foursquare**
Crearea unui profil personal	X	X	X	X	X
Administrarea profilului personal	X	X	X	X	X
Adăugarea unei fotografii/videoclip	X	X	X	X	
Administrarea continutului încarcat	X	X	X	X	X
Adăugarea unei fotografii de copertă	X	X	X		
Adăugarea unei fotografii de profil	X	X	X	X	X
Adăugarea unei fotografii în centrul atenției	X	X			
Adăugarea unei pasiuni	X	X			
Adăugarea unei amintiri	X				
Adăugarea unei persoane în lista de prieteni	X	X	X	X	X
Adăugarea unei melodii preferate pe pagina de profil	X				
Postarea unui eveniment	X	X	X	X	X
Crearea unui grup public/privat	X	X	X	X	
Căutarea de persoane folosind anumite filtre	X	X	X	X	
Comunicarea cu alte persoane folosind chat-ul	X	X			

Tabel 3.2 Comparatie între sistemului dezvoltat și sistemele similare

Capitolul 4. Analiză şi Fundamentare Teoretică

Acest capitol va prezenta arhitectura conceptuală a sistemului, cerintele functionale descrise sub forma cazurilor de utilizare, cerintele non-functionale ale sistemului si tehnologiile care au fost utilizate pentru implementarea sistemului. De asemenea acest capitol va cuprinde si detaliile considerate ca fiind necesare pentru intelegerea codului sursa, cu referire catre literatura de specialitate.

4.1. Arhitectura conceptuală a sistemului

Aplicația Lifestory este un sistem orientat pe arhitectura Client-Server, în care partea de Server a fost creată folosindu-se framework-ul Express 4.0, iar partea de Client a fost creată folosindu-se tehnologiile Web , HTML 4.0, CSS 3.0 și JavaScript. La final, aplicația va putea fi rulată pe orice dispozitiv care are instalat un browser Web.

Pentru stocarea informațiilor despre utilizatori, evenimente și mesaje, se vor utiliza doua baze de date NoSQL, Cassandra fiind orientată pe arhitectura „cheie-valoare” și Neo4J orientată pe arhitectura de grafuri.

Diagrama conceptuală a sistemului este prezentată în Figura 4.1, fiind evidențiat accesul concurent al utilizatorilor în aplicatie.

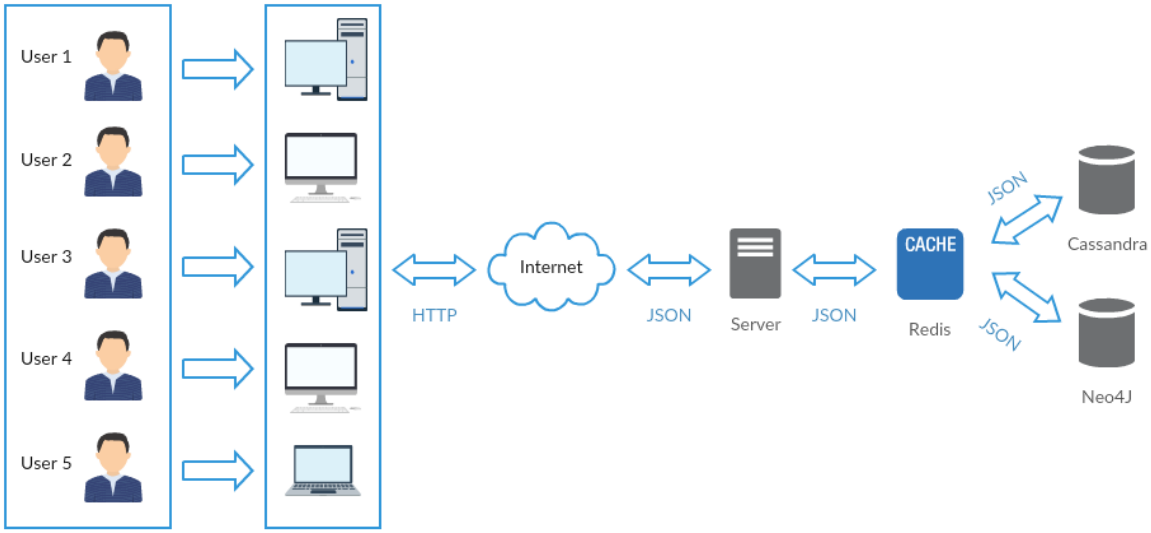

Fig. 4.1 Arhitectura conceptuală a sistemului

Pentru ca utilizatorii sa se poate conecta în aplicație trebuie să fie conectați la internet și să aibă instalat pe dispozitivul personal un browser Web (Google Chrome, Mozilla sau Internet Explorer), aplicația fiind adaptată pentru fiecare versiune de browser. Cea de-a doua cerința ce trebuie îndeplinită reprezintă existența unui email și a unei parole valide înregistrate în bazele de date ale aplicației.

Sistemul proiectat este orientat pe arhitectura Client-Server, clientul fiind reprezentat de browser-ul Web instalat pe dispozitivul clientului, iar server-ul este reprezentat de aplicația care ofera servicii clientului. La randul sau, server-ul interacționeaza cu o baza de date folosită pentru caching și două baze de date folosite

pentru persistența datelor. Implementarea arhitecturii Client-Server poate fi vizualizată în Figura 4.2.

Fig. 4.2 Arhitectura client-server

Clienții stabilesc o relație de comunicare cu server-ul pentru a putea îndeplini cerințele funcționale ale sistemului. Schimbul de mesaje între client și server se face prin respectarea modelului request-response, acesta fiind un protocol de comunicare des întalnit în proiectarea aplicațiilor Web. Pentru o comunicare cât mai eficientă, atât clientul cât și server-ul trebuie să urmeze aceste principii de comunicare pentru a se cunoaște foarte clar detaliile la care să se aștepte fiecare parte.

4.2. Cerințele aplicației

Cerințele aplicației reprezintă acțiunile la care utilizatorul se așteapta de la un sistem. Cerințele aplicației trebuie să definească capabilitățile și constrângerile la care un sistem trebuie să se conformeze. Identificarea cerințelor sistemului reprezintă cel mai important pas în dezvoltarea sistemului, deoarece implementarea și finalizarea cu succes a aplicației se bazează pe acestea.

Cerințele prezentate în acest subcapitol descriu funcționalitățile sistemului pentru a facilita crearea unui profil personal, adăugarea de activități pe profilul personal, adăugarea de fotografii, videoclipuri și criteriile de filtrare a activităților pe homepage. O parte din cerințele sistemului se regăsesc în majoritatea rețelelor de socializare, dar există și funcționalități care au fost adăugate pentru a diferenția aceasta aplicație de cele existente pe piața în acest moment.

4.2.1. Cerințe funcționale

Cerințele funcționale descriu capabilitățile și acțiunile pe care sistemul trebuie să le îndeplinească. Aceste acțiuni trebuie descrise, descrierea acestora trebuie să fie completă și trebuie să fie facută din perspectiva utilizatorului. Totodată, cerințele funcționale vor fi împărțite pe secțiuni. Fiecare secțiune descrie principalele funcționalități pe care le poate executa utilizatorul în funcție de regiunea în care se află.

1. Cerințe funcționale corespunzatoare secțiunii de autentificare/ înregistrare

Tabelul 4.1 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunea de autentificare și înregistrare.

ID	Nume	Descriere
CF-1.1	Crearea unui cont	Descrie acțiunile necesare pentru crearea unui cont.
CF-1.2	Autentificare	Descrie acțiunile necesare pentru înregistrarea în aplicație.
CF-1.3	Verificare token email	Descrie verificarea PIN-ului trimis prin email sau SMS în etapa înregistrării
CF-1.4	Recuperarea parolei	Descrie funcționalitatea pentru recuperarea parolei

Tabel 4.1 Cerințe funcționale pentru secțiunea de autentificare/înregistrare

2. Cerințe funcționale corespunzatoare utilizatorilor anonimi

Tabelul 4.2 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunile care pot fi efectuate de utilizatorii anonimi.

ID	Nume	Descriere
CF-2.1	Vizualizare parțiala a profilelor existente	Utilizatorul anonim poate vizualiza doar postarile publice de pe pagina de profil
CF-2.2	Căutarea unei persoane in aplicație	Utilizatorul anonim poate vizualiza utilizatorii care dețin un cont în aplicație
CF-2.3	Vizualizarea evenimentelor populare adăugate sub format public	Utilizatorii anonimi pot vedea conținutul încarcat sub format public
CF-2.4	Vizualizarea fotografiilor încărcate în anumite locuri de pe glob.	Utilizatorul anonim are posibilitaea de a vedea toate fotografiile încarcate pe glob.

Tabel 4.2 Cerințe funcționale pentru secțiunea destinată utilizatorilor anonimi

3. Cerințe funcționale corespunzatoare administratorului

Tabelul 4.3 prezintă cerințele funcționale ale aplicației de socializare referitoate la acțiunile care pot fi efectuate de administrator.

ID	Nume	Descriere
CF-3.1	Administrarea bazei de date	Administratorul trebuie să poată gestiona conținutul bazei de date.
CF-3.2	Blocarea unui cont utilizator raportat	Pentru un utilizator raportat de alți utilizatori utilizatorul trebuie să aibă posibilitatea de a-i bloca contul
CF-3.3	Administrarea conținutului încărcat în aplicație	Administratorul poate să gestioneze tot conținutul încărcat în aplicație

Tabel 4.3 Cerințe funcționale pentru secțiunea destinată administratorului

4. Cerințe funcționale corespunzătoare profilului personal

Tabelul 4.4 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunile ce pot fi efectuate de un utilizator pe profilul personal.

ID	Nume	Descriere
CF-4.1	Adăugarea de fotografii pe pagina personală	Utilizatorul adaugă o fotografie pe pagina personală
CF-4.2	Adăugarea unei persoane în lista prietenilor a caror activitate este urmarită	Utilizatorul selectează de pe pagina de profil un utilizator a carui activitate dorește să o urmarească mai des.
CF-4.3	Crearea unui cerc de prieteni	Utilizatorul poate crea un cerc de prieteni pe baza intereselor commune.
CF-4.4	Adăugarea unei fotografii de copertă	Utilizatorul adaugă o nouă fotografie de copertă
CF-4.5	Repoziționarea fotografiei de copertă	Utilizatorul ajustează fotografia de copertă existentă.

CF-4.6	Ștergerea fotografiei de copertă	Utilizatorul șterge fotografia de copertă existentă.
CF-4.7	Adăugarea unei fotografii de profil	Utilizatorul adaugă o nouă fotografie de profil.
CF-4.8	Repoziționarea fotografiei de profil	Utilizatorul ajustează fotografia de profil existentă.
CF-4.9	Adăugarea unei pasiuni	Utilizatorul își personalizează profilul prin adăugarea unei pasiuni
CF-4.10	Adăugarea unei amintiri	Utilizatorul crează o nouă amintire și o publică pe pagina de profil
CF-4.11	Crearea unui album	Utilizatorul crează un nou album.
CF-4.12	Selectarea unei fotografii ca fotografie de profil	Utilizatorul selectează o fotografie dintr-un album ca fotografie de profil
CF-4.13	Selectarea unei fotografii ca fotografie de copertă	Utilizatorul selectează o fotografie dintr-un album ca fotografie de copertă.

Tabel 4.4 Cerințe funcționale ale paginii de profil

5. Cerințe funcționale corespunzătoare procesului de adăugare de evenimente pe profilul personal

Tabelul 4.5 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunile ce pot fi efectuate în momentul în care se dorește postarea unui eveniment pe profilul personal sau homepage.

ID	Nume	Descriere
CF-5.1	Postarea unui eveniment pe pagina de profil sau homepage	Utilizatorul adaugă un eveniment simplu pe pagina de profil.
CF-5.2	Adăugarea de fotografii la un eveniment	Utilizatorul adaugă una sau mai multe fotografii la un eveniment ce urmează să fie adăugat pe profil.

CF-5.3	Adăugarea unui videoclip la un eveniment	Utilizatorul adaugă un videoclip la un eveniment ce urmează să fie adăugat pe profil.
CF-5.4	Adăugarea unui check-in la un eveniment	Utilizatorul adaugă locul și persoanele cu care se află la un anumit eveniment
CF-5.5	Etichetarea unei fotografii	Utilizatorul poate eticheta persoanele care sunt prezente într-o fotografie încarcată pentru un eveniment
CF-5.6	Editarea unui eveniment postat	Utilizatorul poate edita conținutul evenimentelor postate
CF-5.7	Ștergerea unui eveniment postat	Utilizatorul poate șterge evenimentele postate
CF-5.8	Adăugarea unui comentariu la un eveniment postat	Utilizatorul poate comenta la un eveniment
CF-5.9	Reacționarea la un eveniment postat	Utilizatorul poate să reacționeze la un eveniment
CF-5.10	Adăugarea unui raspuns la un comentariu	Utilizatorul poate adăuga un răspuns la un comentariu
CF-5.11	Adăugarea unei reacții la un comentariu	Utilizatorul poate reacționa la un comentariu
CF-5.12	Adăugarea unei fotografii/ videoclip la un comentariu	Utilizatorul poate adăuga o fotografie sau un videoclip într-un comentariu
CF-5.13	Distribuirea unui eveniment pe profilul personal	Utilizatorul poate distribui un eveniment pe profilul personal

Tabel 4.5 Cerințe funcționale pentru adăaugarea evenimentelor

6. Cerințe funcționale corespunzătoare mesageriei instantanee

Tabelul 4.6 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunile ce pot fi efectuate în momentul în care se dorește folosirea mesageriei instantanee.

ID	Nume	Descriere
CF-6.1	Trimiterea unui mesaj	Descrie acțiunile executate de utilizator pentru a putea comunica cu alt utilizator din lista de prieteni.
CF-6.2	Ștergerea unui mesaj	Utilizatorul poate șterge un mesaj trimis altui utilizator.
CF-6.3	Ștergerea unei conversatii	Utilizatorul poate șterge o conversație cu un alt utilizator.
CF-6.4	Trimiterea de fotografii folosind chat-ul	Utilizatorul poate trimite o fotografie folosind mesageria instantanee.
CF-6.5	Trimiterea unui videoclip folosind chat-ul	Utilizatorul poate trimite un videoclip folosind mesageria instantanee.
CF-6.6	Adăugarea unui emoticon la un mesaj	Utilizatorul poate trimite un emoticon folosind mesageria instantanee.

Tabel 4.6 Cerințe funcționale corespunzătoare mesageriei instantanee

7. Cerințe funcționale corespunzătoare secțiunii destinate grupurilor create de utilizatori

Tabelul 4.7 prezintă cerințele funcționale ale aplicației de socializare referitoare la acțiunile ce pot fi efectuate într-un grup.

ID	Nume	Descriere
CF-7.1	Crearea unui grup public/ privat	Utilizatorul poate crea un grup privat sau public.
CF-7.2	Adăugarea unui utilizator în grup	Utilizatorul(administratorul grupului) poate adăuga o persoană in grup.
CF-7.3	Ștergerea unui utilizator din grup	Utilizatorul(administratorul grupului) poate șterge o persoană din grup.

CF-7.4	Promovarea unui utilizator la titlul de administrator	Utilizatorul(administratorul grupului) poate promova un utilizator din grup în poziția de administrator.
CF-7.5	Aprobarea unui eveniment ce urmeaza a fi postat	Utilizatorul(administratorul grupului) trebuie sa își exprime acordul asupra evenimentelor ce urmează să fie adaugate in grup.
CF-7.6	Adăugarea unei postari evidențiate	Utilizatorul unui grup când adaugă un eveniment poate să-l marcheze ca fiind evidențiat.
CF-7.7	Urmărirea activităților anumitor utilizatori din grup	Utilizatorul unui grup poate selecta opțiunea de urmarire a evenimentelor adăugate de alți utilizatori.

Tabel 4.7 Cerințe funcționale destinate secțiunii grupurilor

4.2.2. Cerințe non-funcționale

Spre deosebire de cerințele funcționale ale sistemului, cerințele non-funcționale reprezintă proprietăți și constrângeri impuse asupra sistemului. Analizând orice sistem informatic observăm că, cerințele non-funcționale sunt catalogate ca factori de calitate ai sistemului. În următoarele paragrafe vom descrie și analiza care sunt principalele calități și constrângeri impuse sistemului.

Aplicația de socializare este proiectată pentru a suporta un numar foarte mare de utilizatori concurenți. Aceștia vor avea nevoie de o instruire inițiala a sistemului, dar și fară aceasta sistemul este foarte intuitiv ți usor de folosit.

Utilizabilitatea unui sistem reprezintă ușurința cu care un sistem poate fi învățat și utilizat, astfel sistemul propus își dorește ca timpul de acomodare și învățare să fie cât mai mic. Aceasta cerința non-funcțională este la fel de importantă ca orice altă cerință funcțională deoarece acest factor va conduce la creșterea numărului de utilizatori activi, în detrimentul altor aplicații de socializare. Chiar dacă utilizatori sunt instruiți la prima folosire a sistemului (crearea unui cont), utilizabilitatea nu trebuie neglijată [3].

În procesul de proiectare a interfeței utilizator trebuie să fim atenți la toate detaliile și să proiectam o interfața cât mai comună și ușor de utilizat. O aplicație care oferă o interfața utilizator simplă, intuitivă și bine organizată poate fi preferată de utilizatori chiar dacă este incompletă din punct de vedere al cerințelor funcționale , în detrimentul altor aplicații care au implementate toate cerințele funcționale, dar cu o interfața utilizator greu de interacționat. Utilizabilitatea este susținută și de eficiența pe care o au utilizatorii în momentul în care vor să îndeplinească diferite task-uri, acest factor fiind garantat prin navigarea rapida prin interfața utilizator.

Odata ce utilizatorul s-a autentificat în aplicație și bifează opțiunea *„Pastrează-mă autentificat”,* acesta nu va mai fi nevoit ca următoarea dată când dorește să folosească aplicația să completeze datele de autentificare. Aceasta funcționalitate reprezintă o

caracteristică importantă de utilizabilitate, scutind utilizatorul de la efectuarea unor pași inutili.

Disponibilitatea descrie măsura în care sistemul trebuie să funcționeze pentru utilizatori. Sistemul trebuie să fie disponibil 24 din 24 utilizatorilor, iar în cazul în care sistemul pică durata de revenire să nu depășească 30 de minute. Disponibilitatea unui sistem poate fi masurată prin siguranța care o oferă sistemul și prin tolerabilitatea la erorile care pot sa aparea. Cheia esențiala spre un sistem sigur și robust este reprezentată de etapa de validare a datelor introduse de utilizatori. Sistemul nu trebuie să aibă încredere niciodată în datele introduse de utilizatori și trebuie să efectueze filtre atât pe partea de client cât și pe partea de server. Sistemul a fost implementat iterativ, eventualele erori care ar fi putut apărea putând fi observate pentru fiecare componentă dezvoltată [3].

Mentenabilitatea se referă la ușurința cu care sunt adăugate noi modificări în sistem și la ușurința de întreținere a acestuia. Partea de server trebuie să fie structurată pe responsabilități, astfel încât să permită adăugarea de noi servicii foarte ușor fară a efectua alte modificari neesențiale în sistem [3].

Performanța unui sistem se referă la timpul maxim de răspuns la o cerere venită de la utilizator. Principala activitate pe care o execută utilizatorul este de a vedea evenimentele recente de pe pagina de homepage și de a vizualiza profilele altor utilizatori. Timpul de raspuns pentru afișarea template-ului corespunzator paginii de homepage nu trebuie să depasească 2 secunde, celelate elemente fiind încarcate în mod asincron. Aceiași constrângere de timp este impusă și pentru pagina de profil [3].

Securitatea unui sistem este caracterizată pe langă mecanismele folosite pentru protejarea datelor și împiedicarea accesului neautorizat în aplicație, de mecanisme folosite pentru restrictionarea accesului asupra resurselor importante ale sistemului. Parolele în baza de date sunt criptate, iar pentru fiecare cerere care ajunge la server se verifică daca există un utilizator autentificat și ce drepturi de acces are acesta [3].

ID	Descriere
CNF-1	Utilizabilitate – definește o interfață simplă și intuitivă
CNF-2	Disponibilitate – funcționarea 24 din 24 a sistemului. Timp de revenire mic în cazul unui eșec.
CNF-3	Mentenabilitate – ușurința de întreținere și de adăugare a unor noi funcționalitați
CNF-4	Performanța – timp de raspuns la o cerere
CNF-5	Securitate – criptare, autorizare și autentificare

Tabel 4.8 Cerințe non-funcționale ale aplicațiilor Web

4.3. Cazuri de utilizare

Cazurile de utilizare definesc o perspectiva globala asupra comportamentului și funcționalităților puse la dispoziție utilizatorilor. Cerințele funcționale ale sistemului sunt cazuri de utilizare posibile, însa nu toate cerințele funcționale trebuie tratate ca și cazuri de utilizare. Un scenariu al cazului de utilizare poate fi definit ca o secvență specifică de acțiuni și interacțiuni ce descriu actorii care utilizează sistemul pentru a îndeplini o anumită sarcină. Cazurile de utilizare corespunzatoare aplicației vor fi descrise atât sub forma unei diagrame de cazuri de utilizare cât și sub formă scrisă.

4.3.1. Actorii sistemului

Tipurile de utilizatori :

- Administrator
- Utilizator autentificat
- Utilizator anonim

4.3.2. Cazuri de utilizare

În diagramele de cazuri de utilizare reprezentate în acest sub-capitol sunt ilustrate acțiunile pe care le poate realiza administratorul, utilizatorul autentificat și utilizatorul anonim. Se observă că între cele două tipuri de utilizatori (utilizator autentificat și utilizator anonim) există o diferența majoră din punct de vedere a acțiunilor care se pot executa.

În figura, Fig 4.3, sunt prezentate cazurile de utilizare în care actorul principal este administratorul sistemului.

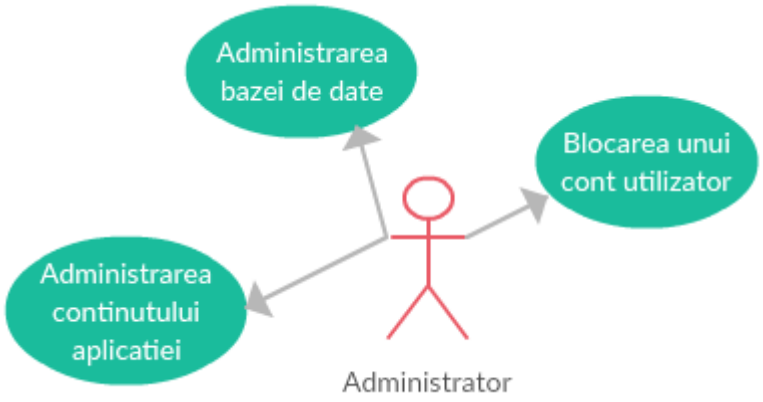

Fig. 4.3 Diagrama cazurilor de utilizare corespunzatoare administratorului

Figura următoare, Fig 4.4, prezintă cazurile de utilizare în care actorul principal este utilizatorul care deține un cont valid în aplicație.

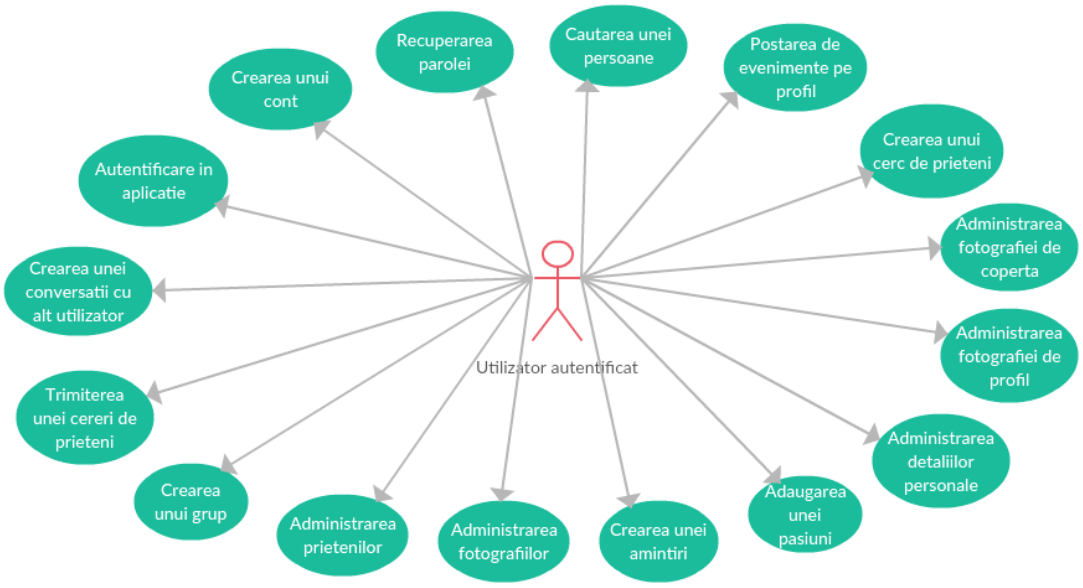

Fig. 4.4 Diagrama cazurilor de utilizare pentru utilizatorului autentificat

În figura următoare, Fig 4.5 sunt prezentate cazurile de utilizare în care actorul principal este utilizatorul anonim(nu deține un cont în aplicație).

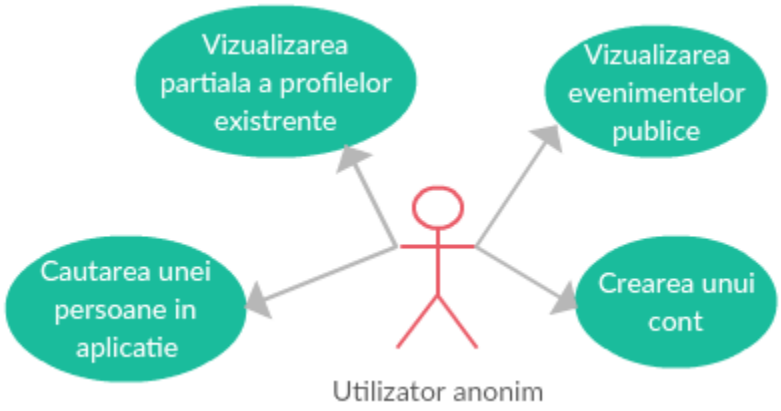

Fig. 4.5 Diagrama cazurilor de utilizare corespunzătoare utilizatorului anonim

4.3.2.1. Descrierea detaliată a cazurilor de utilizare

În continuare vor fi analizate cele mai imporatante cazuri de utilizare, pentru fiecare din acestea specificându-se precondițiile, postcondțtiile, scenariul de succes, dar și scenariul de eșec.

a) Crearea unui cont utilizator

Pentru a putea folosi funcționalitățile aplicației toți utilizatorii trebuie să dețină un cont în aplicație. Pentru a se înregistra în aplicație, utilizatorul trebuie să completeze un formular cu datele persoanele(nume, prenume, email, parola și data nașterii). În cele ce urmează se va prezenta cazul de utilizare corespunzator creării unui cont utilizator:

Actor principal: Utilizator anonim

Participanți și interese: Utilizatorul anonim își dorește să vadă mai mult conținut și să folosească funcționalitățile sistemului pentru a socializa cu alte persoane.

Precondiții: Aplicația server trebuie să fie pornită. Clientul să acceseze browser-ul și să introducă adresa URL corespunzătoare sistemului *„www.mylifestory.com”*.

Post condiții: Utilizatorul anonim va deține un cont în aplicație, deci va deveni un utilizator autentificat.

Principalul scenariu de succes:

1. Utilizatorul acceseaza secțiunea de înregistrare.
2. Utilizatorul completează formularul afișat cu datele personale.
3. Utilizatorul trimite formular completat pentru a fi valid de sistem.
4. Sistemul trimite o adresa de validare pe email-ul introdus în pasul anterior, în formular.
5. Utilizatorul accesează link-ul primit pe e-mail pentru a confirma contul creat.
6. Contul este salvat cu succes de catre sistem. În acest moment utilizatorul se poate autentifica cu succes în aplicație.

Extensii(fluxuri alternative):

4a. Sistemul detectează că nu au fost completate toate datele:
   1. Sistemul anulează cererea de înregistrarea și trimite un mesaj corespunzator(*„Trebuie completate toate datele pentru a continua înregistrarea.”).*

4b. Email-ul trimis este invalid:
   1. Sistemul anulează operațiunea de înregistrare și trimite un mesaj de eroare corespunzator(*„Email-ul este invalid.”).*

4c. Email-ul trimis există deja in sistem:

1. Sistemul nu accepta 2 conturi pe aceiași adresă de email. Sistemul anulează cererea de înregistrare ți trimite un mesaj de eroare corespunzator(*„Deja există un cont pe această adresă de email.”)*.

4d. Parola nu respectă cerințele impuse de sistem:

1. Sistemul anulează cererea de înregistrare și trimite un mesaj de eroare corespunzator*(„Parola nu este destul de puternică.”)*.

b) Postarea de evenimente pe pagina de profil/homepage

Fiecare utilizator care deține un cont în aplicație poate posta evenimente complexe pe pagina de homepage sau pe pagina de profil. Toate evenimentele postate pe homepage de un utilizator, pot fi vazute de alți utilizatori pe propriile pagini de homepage sau pe pagina de profil a utilizatorului care a postat respectivul eveniment.

Definim ca fiind un eveniment complex, un eveniment alcătuit dintr-un status, fotografii/videoclipuri sau un check-in.

Actor principal: Utilizator autentificat

Participanți si interese: Utilizatorul autentificat îți dorește să împărtășească conținut personal cu prietenii din lista sau cu toate persoanele care au un cont în aplicație.

Precondiții: Utilizatorul trebuie să fie autentificat.

Post conditii: Evenimentul creat de utilizator va putea fi observat pe profilul acestuia și pe pagina de homepage a altor persoane.

Principalul scenariu de succes:

1. Utilizatorul accesează pagina de profil sau pagina de homepage.
2. Utilizatorul selectează meniul destinat adăugarii unui eveniment.
3. Utilizatorul completează formularul afișat cu mesajul pe care vrea să-l împărtășească cu celelalte persoane.
4. Utilizatorul acționează butonul *„Postează”.*
5. Sistemul validează mesajul introdus și adaugă evenimentul în baza de date.
6. Evenimentul este afișat pe pagina personala a utilizatorului care l-a creat și pe pagina de homepage a celorlalte persoane.

Extensii(fluxuri alternative):

4a. Utilizatorul acționează butonul destinat acțiunii de încarcare a unei fotografii.

3a.1. Utilizatorul selectează fotografiile pe care dorește să le incarce.

3a.2. Fotografiile selectate sunt afișate în meniul destinat postarii unui eveniment.

4b. Utilizatorul acționează butonul destinat acțiunii de încarcare a unui videoclip.

3b.1. Utilizatorul selectează videoclipul pe care dorește să-l încarce.

3b.2. Videoclipul selectat este afișat în meniul destinat postării unui eveniment.

4c. Utilizatorul copiază în formularul destinat scrierii unui mesaj un link de pe Youtube sau alt site.

3c.1 Conținutul corespunzator link-ului adăugat este afișat în meniul destinat postarii unui eveniment.

4d. Utilizatorul acționează butonul destinat acțiunii de adăugare a unui check-in.

3d.1 Utilizatorul selectează prietenii care-l însoțesc la respectivul eveniment.

3d.2 Utilizatorul selectează locul în care se desfășoară evenimentul.

5a. Sistemul detectează un format necorespunzator al datelor introduse:

1. Sistemul anulează cererea de înregistrare a evenimentului în baza de date și trimite un mesaj de eroare corespunzător erorii.

c) Administrarea fotografiei de copertă

Actor principal: Utilizator autentificat

Participanți și interese: Utilizatorul autentificat dorește să-și administreze fotografia de copertă prin adăugarea, repoziționarea sau ștergerea acesteia.

Precondiții: Utilizatorulp se află pe pagina profilului personal.

Post condiții: Utilizatorul observă actualizarea fotografiei de copertă în urma acțiunii efectuate.

Principalul scenariu de succes:

1. Utilizatorul acționeaza butonul destinat acțiunii de administrare a fotografiei de copertas.
2. Utilizatorul alege acțiunea de a încarca o fotografie de copertă și selectează fotografia de pe propriul dispozitiv.
3. Sistemul afișează fotografia încarcata pentru a putea fi executată acțiunea de repoziționare a acesteia.
4. Utilizatorul repoziționează fotografia.
5. Utilizatorul acționează butonul *„Salveaza coperta.”*.
6. Sistemul afișează noua fotografie de copertă pe pagina de profil a utilizatorului.

Extensii(fluxuri alternative):

2a. Utilizatorul alege acțiunea de repoziționare a fotografiei de copertă.
2b. Utilizatorul alege acțiunea de ștergere a fotografiei de copertă.
3a. Dimensiunile fotografiei încarcate nu îndeplinesc cerințele referitoare la dimensiunile fotografiei. Se afișeaza un mesaj de eroare corespunzător.
5a. Utilizatorul acționează butonul *„Anulează”*.
5a.1 Sistemul afișează fotografia de copertă setată inițial.

d) Trimiterea unei cereri de prietenie

Actor principal: Utilizator autentificat

Participanți și interese: Utilizatorul autentificat dorește să adauge o persoană în lista de prieteni.

Precondiții: Utilizatorul trebuie să fie autentificat în aplicație și să se afle pe pagina de homepage.

Post condiții: Utilizatorul care inițiază acțiunea de trimitere a cererii de prietenie observă trimiterea acesteia, iar utilizatorul spre care se trmite cererea va observa prezența acesteia printr-o notificare.

Principalul scenariu de succes:

1. Utilizatorul se afla pe pagina de homepage.
2. Utilizatorul observa o persoană în lista de persoane sugerate.
3. Utilizatorul acționează butonul de „Adaugă prieten”.
4. Sistemul schimbă conținutul butonului din *„Adaugă prieten”* în *„Cerere de prietenie trimisă”*.
5. Sistemul memorează acțiunea executată de utilizator și trimite o notificare utilizatorului cu care se dorește stabilirea relației.

Extensii(fluxuri alternative):

4a. Sistemul detectează că utilizatorul spre care se trimite cererea are deja numărul maxim de utilizatori:
   1. Sistemul anulează cererea de înregistrare și trimite un mesaj corespunzător(*„Cererea de prietenie nu a putut fi trimisă.”).*

4b. Sistemul detectează că utilizatorul spre care se trimite cererea a selectat opțiunea de a nu primi cereri de prietenie :
   1. Sistemul anulează cererea de înregistrare și trimite un mesaj corespunzător(*„Cererea de prietenie nu a putut fi trimisă.”).*

4.4. Tehnologii utilizate

4.4.1. Tehnologii pe partea de Client

Clientul este reprezentat de browser-ul Web de pe dispozitivul utilizatorului. Pentru implementarea aplicației pe partea de client am folosit limbajele HTML și CSS(folosite pentru realizarea interfeței utilizator), JavaScript(folosit pentru adăugarea de conținut dinamic în pagină și pentru interacțiunea dintre utilizator și aplicația Web). Fișierele se află în directorul *„public”*, fiind structurate sub directoarele *css* si *js*. Pentru fiecare pagină a aplicației avem câte un director cu fișierele CSS și un director ce conține fișierele JavaScript corespunzătoare. Fișierele CSS și JavaScript sunt încarcate în aplicație doar în momentul în care este nevoie de acestea.

1. HTML 4.0

HyperText Markup Language prescurtat ca HTML este o formă de marcare orientată către prezentarea documentelor text pe o singură pagină, utilizând un software de randare specializat numit agent utilizator HTML, cel mai bun examplu în acest sens fiind browser-ul Web. Paginile HTML sunt formate din etichete și tag-uri și au extensia *„.html”* sau *„.htm”*. În marea lor majoritate aceste etichete sunt pereche, existând o etichetă la deschidere *„<eticheta>”* și o eticheta la închidere *„</eticheta>”*. Navigatorul parcurge aceste documente HTML și interpretează etichetele definite afisând rezultatul pe ecran. Această tehnologie a fost folosită pentru a crea paginile Web ale aplicației [4].

2. CSS 3.0

Cascadind Style Sheets prescurtat ca CSS este un standard pentru formatarea elementelor unui document HTML. Stilurile definite pentru elementele documentului se pot defini prin fișiere externe, referințe spre acestea facându-se în zona **head** a documentului HTML sau prin introducerea etichetelor *<style> </style>* în zona **head** a documentului.

În momentul actual ce mai recentă versiune a limbajului CSS este CSS3 care aduce un plus considerabil în dezvoltarea activităților de web design. Pentru a realiza design-ul fiecarei pagini am folosit ultima versiune de CSS, fiecarei pagini Web create îi corespunde un fișier CSS [4].

3. JavaScript

În aplicațiile moderne, clientul interacționează foarte mult cu server-ul, în urma fiecre interacțiuni generându-se conținut dinamic care trebuie afișat în pagina Web. Pentru a îndeplini această cerința am avut nevoie de un limbaj de programare client-side.

Este un limbaj de programare orientat obiect bazat pe conceptul prototipurilor. Javascript este folosit pentru introducerea unor funcționalități în paginile Web, codul fiind rulat de catre browser-ul Web. A fost inițial dezvoltat de **Brenden Eich** de la Netscape Communications Corporation sub numele de Mocha, apoi LiveScript si denumit la final JavaScript.

Programatorii web pot ingloba în paginile HTML script-uri pentru diverse activități cum ar fi verificarea datelor introduse de utilizatori sau crearea de meniuri și alte efecte animate. Browser-ele reține în memorie o reprezentare a unei pagini web sub forma unui arbore de obiecte și pun la dispoziție aceste obiecte script-urilor JavaScript pentru a le putea manipula. Arborele de obiecte reținut în memorie se numește Document Object Model prescurtat sub numele de DOM.

O tehnică de construire a paginilor web tot mai întalnita în ultimul timp este Asynchronous Javascript and XML prescurat sub numele de AJAX. Această tehnică presupune în executarea unei cereri HTTP în fundal, fară a reîncarca toată pagina web, ți actualizarea numai a anumitor porțiuni ale paginii prin manipularea DOM-ului paginii. Această tehnică permite construirea unor interfețe web cu timp de raspuns mic, deoarece operatia de încarcare a unei pagini HTML complete este în mare parte eliminată [5].

4.4.2. Comuncarea între Client și Server

Protocolul folosit pentru comunicarea intre Client si Server este HTTP (Hypertext Transfer Protocol). HTTP este metoda cea mai des folosita pentru accesarea informatiilor de pe internet, informatiile fiind pastrate pe servere. Protocolul HTTP este un protocol de tip text, folosit pentru a transfera documente HTML, fisiere grafice, fisiere audio, animatii, videoclipuri sau programe executabile. Modelul de referinta OSI clasifica protocolul HTTP ca fiind un protocol la nivel de aplicatie.

In prezent se folosesc doua versiuni ale protocolului, HTTP 1.0 si HTTP 1.1. Versiunea HTTP 1.0 a fost proiectata ca pentru fiecare cerere sa se creeze o noua conexiune TCP, iar in momentul in care cererea a fost finalizata si raspunsul a fost trimis clientului conexiunea va fi inchisa. Astfel daca un document HTML cuprinde 2 imagini si 3 videoclipuri se vor executa in total 6 conexiuni TCP pentru ca pagina sa fie afisata. Versiunea 1.1 aduce imbunatatiri majore asupra protocolului, deci clientul poate sa emita mai multe cereri si raspunsuri pe aceiasi conexiune TCP. Astfel pentru examplul anterior vom avea nevoie de o singura conexiune TCP in loc de 6. Pentru a realiza comunicarea dintre client si server am folosit versiunea 1.1 a protocolului HTTP. Versiunea 2.0 este in continua dezvoltare si va urma sa fie integrata de majoritatea aplicatiilor mari, deoarece ofera suport pentru multiplexare si concurenta, este dependent de stream-uri, iar dimenisunea headerelor este foarte redusa fata de versiunea 1.1 [6].

Metodele de baza oferite de protocolul HTTP sunt urmatoarele:

- **GET** : este cea mai folosita metoda de comunicare, fiind folosita atunci cand clientul doreste o resursa de pe server.
- **POST** : aceasta metoda este folosita pentru a trimtie date spre server.
- **PUT** : aceasta metoda este asematoare cu metoda POST, fiind folosita pentru a actualiza anumite date stocate pe server.
- **DELETE** : aceasta metoda este folosita pentru a sterge anumite date de pe server.

Aplicatia Server este construita pe modulul arhitectural REST(REpresentation State Transfer). Acest stil arhitectural reprezinta cea mai populara metoda de construire a unui serviciu Web. REST este un model arhitectural folosit pentru a citi, crea, actualiza si sterge date de pe server prin intermediul apelurilor HTTP. Un apel REST reprezinta o simpla cerere HTTP catre server.

Principala alternativa a modelului REST este modelul arhitectural SOAP (Simple Object Access Protocol). Acest protocol este folosit pentru a schimba informatii structurate intre client si server, folosind limbajul XML in implementarea serviciului Web [7].

SOAP	REST
Servicile Web care sunt implementate folosind SOAP implica modificari complicate pe partea de client. In momentul in care clientul este un dispozitiv mobil avem de-a face cu o problema serioasa.	Acest protocol a fost proiectat pentru comunicarea intre client si server. Fiecare cerere care se face catre server contine toate informatiile necesare pentru a putea intelege cererea. Protocolul se caracterizeaza prin simplicitate si flexibilitate in implementare.
Generarea de cod client pe baza fisierelor WSDL si XSD este destul de complexa, iar aceasta problema devine si mai complexa in momentul in care aceiasi aplicatie trebuie dezvoltata pe mai multe dispozitive mobile (Android, IOS, Windows Phone).	Cererile facute de clienti folosind acest protocol nu retin starea, deci acest protocol este foarte scalabil. Load balancer-ul, firewall-urile si proxy-urile sunt optimizate pentru traficul cu mesaje in format JSON.
Serviciile SOAP returneaza intotdeauana XML. Structura fisierelor XML este mai mare decat structura unui fisier JSON, REST oferindu-i dezvoltatorului posibilitatea de a alege in mai multe structuri de date disponibile.	Pentru afisarea de continut dinamic se folosesc apeluri Ajax. Ajax a fost adaptat sa lucreze cu servicii RESTful in format JSON. Sistemele de operare de pe dispozitivele mobile au introdus parsere pentru formatul JSON.
Informatia structurata in format XML este mai usor de citit de utilizator.	Informatia transmisa folosind standardul JSON este mai greu de citit de utilizator.

Tabel 4.9 Comparație SOAP vs REST

Integrarea nivelul de servicii cu baza de date se face prin JSON, deci in concluzie am ales sa implementez serviciile aplicatiei Web folosind modelul arhitectural REST, datorita flexibilitatii in implementare, a performantei si a optimizarilor existente si a posibilitatii de reutilizare atat in aplicatia Web cat si in aplicatiile mobile.

4.4.3. Tehnologii pe partea de Server

Aplicatia Server a fost dezvoltata in limbajul JavaScript, avand la baza framework-ul NodeJS. Pentru persitenta datelor am folosit doua baze de date, Cassandra si Neo4J. Cassandra este o baza de date de tip KV, in care datele sunt stocate secvential dupa cheia de partitionare configurata pentru fiecare familie de coloane. Am folosit aceasta baza de date pentru a stoca postarile, fotografiile, videoclipurile si mesajele utilizatorilor. In momentul in care s-a dorit stocarea relatiilor dintre utilizatori, pentru a respecta princiipile impuse de Cassandra, „query/table”, aveam o problema de consistenta a datelor in cazul in care acestea erau modificate in alte tabele. Pentru a rezolva problema de inconsistenta a datelor am ales folosirea unei baze de date orientata pe grafuri, Neo4J, pentru a reprezenta relatiile dintre utilizatori. Pentru a asigura un timp de raspuns bun si pentru a reduce numarul de cereri catre bazele de date, am folosit un layer de caching.

Pentru a imparti traficul spre servere si pentru a creste numarul de utilizatori concurenti care pot accesa aplicatia este necesara folosirea unui load balancer.

1. Framework NodeJS

NodeJS este o platforma dezvoltata pe baza V8 Engine folosita pentru dezvoltarea rapida si scalabila a aplicatiilor ce interactioneaza folosind internetul. Node JS foloseste un singur fir de executie, fiind construit pe baza de evenimente. Platforma a fost dezvoltata de Ryan Dahl in anul 2009, ajungand la versiunea 0.10.36 in prezent. NodeJS functioneaza in mod asincron, iar toate librariile care sunt puse la dispozitia dezvoltatorilor functioneaza asincron. Deoarece a fost construit sa functioneze intr-un mod asincron, NodeJS nu trebuie sa astepte dupa executia unor operatii ce necesita mai mult timp pentru procesare. In momentul in care trebuie procesata o cerere a carei executii dureaza mai mult, aceasta este pusa intr-o coada de evenimente, iar urmatoarea cererea venita este procesata. Dupa terminarea procesarii cererii anterioare, raspunsul este oferit clientului.

Node JS este folosit de multe aplicatii care lucreaza cu cantitati mari de date si sunt caracterizate prin scalabilitate. Printre acestea regasim numele unor giganti de pe piata mondiala precum: Ebay, General Electric, Microsoft, PayPal si Yahoo.

Pentru dezvoltarea rapida a aplicatilor Web NodeJS pune la dispozitie mai multe framework-uri. Pentru implementarea acestei aplicatii am ales framework-ul Express deoarece este un framework minimal si flexibil oferind posibilitatea de a dezvolta o aplicatie Web foarte rapid. Express permite setarea unui middleware pentru a raspunde cererilor HTTP venite de la clienti,deasemenea permite definirea unor tabele de rutare pentru efectuarea anumitor actiuni pe baza maparii adreselor URL. Express ofera suport si pentru randarea de pagini HTML in mod dinamic pe baza parametrilor transmisi catre template. Principalele template-uri folosite pentru randarea de continut dinamic sunt **Jade** si **EJS** [8]. Pentru crearea template-urile acestei aplicatii am folosit template engine-ul Jade. In figura urmatoare, Fig 4.6 am prezentat un exemplu care descrie structura si modul in care poate fi creat un template Jade [9] :

```
h1(id="title") Welcome to Jade
button(class="btn", data-action="bea").
  Be Awesome
```

Fig. 4.6 Structura unui template Jade

În momentul in care template-ul este randat el este convertit in format HTML, deci template-ul definit in Fig 4.6 are urmatorul continut HTML, prezentat in figura Fig 4.7 :

```
<h1 id="title">Welcome to Jade</h1>
<button data-action="bea" class="btn">Be Awesome</button>
```

Fig. 4.7 Rezultatul conversiei template-ului Jade în format HTML

Motivația de a alege framework-ul NodeJS și modulul Express 4.0 este susținută de rapiditatea și de simplitatea de dezvoltare a aplicaților Web, de modul relativ ușor de configurare și personalizare, de folosirea limbajului Javascript atât pentru implementarea front-end-ului cât și a back-endului și de capabilitatea acestuia de a gestiona un numar mare de conexiuni simulane oferind un timp de răspuns scăzut.

2. Load balancer - NGINX

Aplicatia este destinata unui numar foarte mare de utilizatori, deci vom avea mai multe servere care se vor ocupa de procesarea datelor si de oferirea de raspunsuri la cererile venite de la utilizatori. Pentru indeplinirea acestui obiectiv am folosit load balancer-ul Nginx. Load balancer-ul folosit suporta 3 algoritmi de balancing [10]:

- **Round Robin** : Acesta este algoritmul default folosit de load balancer in cazul in care nu a fost specificat alt algoritm de balancing. Fiecare server definit in sectiunea *„upstream”* primeste cereri venite de la clienti in mod secvential.
- **Least Conn** : Specifica faptul ca la aparitia unei noi conexiuni, conexiunea va fi redirectionata catre server-ul cu cele mai putine conexiuni active.
- **IP Hash** : Pentru determinarea server-ului care se ocupa de procesarea cererii este folosite adresa IP a clientului.

In figura urmatoare, Fig 4.8 este prezentat modul in care specificam algoritmul de balancing folosit si IP-urile sau adresele serverelor active si folosite de load balancer:

```
upstream myapp1 {
    least_conn;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

Fig. 4.8 Configurare load balancer

Motivația de a alege configurarea unui load balancer-ul este susținută de numărul imens de cereri care se vor face spre serverele sistemului. Pentru a distribui traficul spre servere trebuie configurat un astfel de proxy.

3. Mecanismul de caching

Pentru a minimiza numarul de interactiuni cu baza de date si pentru a indeplini cerintele nonfunctionale cu privire la timpul de raspuns si la performanta sistemului este necesara folosirea unui mecanism de caching intre aplicatia Server si baza de date. Pentru integrarea unei mecanism de caching in aplicatie am folosit Redis.

În momentul actual cele mai folosite baze de date pentru caching sunt:

- **Redis** este o bază de date NoSQL, în care datele sunt păstrate în memoria RAM a server-ului, oferind o performanță ridicată, replicare și un model de date unic. În momentul în care server-ul nu mai funcționeaza Redis oferă două forme de persistența a datelor pentru a scrie datele din memerie pe disc într-o formă compactă. Prima metoda folosită se numesș *„point-in-time”* și se execută când o anumită condiție este îndeplinită, de exemplu este atins un anumit numar de scrieri intr-o anumita perioada. Cea de-a doua metodă se numește *„append-only”*, prin acesta metodă efectuându-se o scriere pe disc de fiecare dată când sunt modificate datele in memorie. Redis permite stocarea in memorie a multor strucuri de date (liste, set-uri si hash-uri) si ofera o limitare de 512 MB pentru dimensiunile cheilor si a valorilor asociate. De asemenea Redis ofere suport si pentru replicarea datelor in cluster [11].

- **Memcache** este o baza de date open source, care ofera performanța ridicată, datele fiind stocate în memoria RAM. Acest mecanism de caching a fost creat pentru a îmbunătăți viteza aplicațiilor Web care oferă conținut dinamic prin reducerea accesului la baza de date. Este orientată pe arhitectura *„cheie-valoare”*, oferind suport pentru stocarea datelor de dimensiuni mici, de exemplu continut HTML static. Memcache limitează dimensiunea fiecare key la 250 kb și lucrează doar cu text [12].

În tabelul 4.10 este prezentata o comparatie intre principalele caracteristici si functionalitati ale mecanismelor de caching.

Nume	Tip	Optiuni de stocare	Caracteristici aditionale
Redis	In-memory, baza de date ne-relationala	String, List, Sets, Hashes, Sorted Sets	Publish/Subsribe, replicare master/slave, persistenta datelor pe disc
Memcached	In-memory, orientata pe principiul „cheie-valoare”	Maparea cheilor la valori	Server care ofera suport pentru multithread-ing pentru imbunatatirea performantei

Tabel 4.10 Redis vs Memcached

În concluzie, motivația de a alege Redis se datorează avantajelor pe care le oferă acesta în comparație cu Memcached cu privire la tipul de date și la gestiunea acestora.

4. BigPipe - randare asincrona a interfeței utilizator

Aplicația dezvoltată trebuie sa gestioneze și să afiseze în interfața utilizator o mulțime de date. Prin folosirea mecanismului clasic de randare a interfeței utilizator, aplicația nu reușește să îndeplinească cerințele nonfuncționale cu privire la performanță și timp de raspuns. Mecanismul clasic presupune faptul ca un client face o cerere catre server (accesarea profilului unui utilizator), iar server-ul furnizează conținutul întregii pagini pe raspunsul oferit clientului. Deoarece sistemul lucrează cu cantități mari de informații, folosirea acestui principiu nu este benefic și afectează vizibil performanța sistemului.

Un nou concept a fost introdus de Facebook, cu referire la randarea asincrona a interfetei utilizator. Acest concept functioneaza pe urmatorul principiu: utilizatorul face o cerere catre server(accesarea unui profil utilizator), iar sistemul raspunde cererii prin afisarea unui template simplu care nu necesita incarcarea fisierelor CSS si Javascript de dimensiuni mari. In momentul in care template-ul ce trebuie afisat este trimis spre utilizator pentru a fi afisat in interfata utilizator, raspunsul nu se incheie, ci restul informatiilor care trebuie afisate si adaugate in respectivul template sunt procesate in mod asincron si trimise pe acelasi raspuns. Prin folosirea acestui concept cream utilizatorului impresia ca sistemul are un timp de raspuns foarte bun si reusim sa indeplinim cerintele nonfunctionale impuse [13].

4.4.4. Tehnologii existente pentru implementarea bazei de date

In momentul de fata exista foarte multe persoane care acceseaza internetul si contribuie la continutul site-urilor web mai mult decat inainte. Exista aplicatii Web care permit utilizatorilor sa adauge foarte mult continut, iar din punct de vedere al stocarii datelor si a timpului de raspuns acest lucru reprezinta o problema. Deoarece aplicatia este destinata unui numar foarte mare de utilizatori care pot incarca mult continut in aplicatie avem de-a face cu un sistem big data.

Sistemele big data implica in primul rand un volum foarte mare de date care incep de la gigabytes si tind spre terabytes, chiar si mai mult. Alte caracateristici importante care trebuie mentionate intr-un sistem big data sunt disponibilitatea pe mai multe regiuni a sistemului, asigurarea unui raspuns foarte rapid si sa nu existe nici un punct de esec al sistemului. Pentru proiectarea unui sistem big data nu putem folosi o baza de date relationala deoarece acest tip de baza de date ofera o schema fixa, iar pentru extragerea mai complexa a datelor este necesara folosirea join-urilor, iar prin folosirea acestora creste timpul de raspuns. In aplicatiile moderne se prefera folosirea bazelor de date ne-relationale deoarece acestea se axeaza mai mult pe viteza si disponibilitate, consistenta nefiind atat de relevanta.

4.4.4.1. RDBMS (Relational Database Management System)

RDBMS cuprinde toate implementarile bazelor de date bazate pe modele relationale, acest concept fiind introdus de E.F. Codd in anul 1969. Intr-o baza de date relationala datele sunt reprezentate sub forma de relatii, componentele unei relatii fiind atributele (head) si tuplele stocate (body). Fiecare RDBMS suporta cel putin un limbaj de

interogare folosit pentru a extrage date din tabele, cel mai comun limbaj fiind SQL (Structured Query Language). In figura urmatoare, Fig 4.9 sunt prezentate principalele componente ale unei relatii.

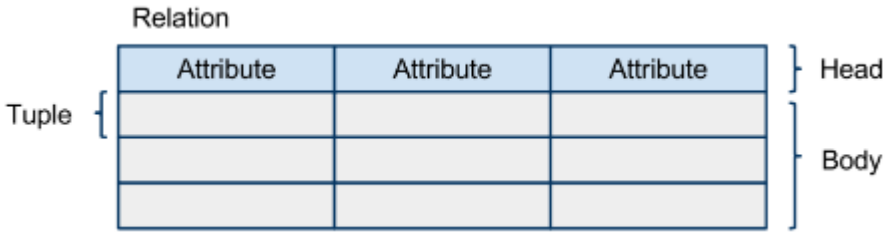

Fig. 4.9 Componentele unei relații dintr-o bază de date relațională

Fiecare rand stocat in tabela este identificat printr-o valoarea unica numita cheie primara. Deoarece putem identifica un rand intr-o tabela prin cheia sa primara, se poate stabili o relatie prin crearea unui nou tabel in care sa avem o referinta spre aceasta cheie, aceasta cheie numindu-se cheie straina.

Un concept important introdus in bazele de date relationala este normalizarea. Acest concept a fost introdus de E.F. Codd, ideea din spatele acestui concept se bazeaza pe existenta unei date intr-un singur tabel, eliminandu-se astfel anomaliile care pot aparea atunci cand inseram, actualizam sau stergem date.

Intr-un sistem ce foloseste baze de date relationale scalabilitatea este foarte importanta. Prin scalabilitate definim abilitatea unui sistem de a suporta si gestiona mai multe request-uri prin adaugarea mai multor noduri(scalabilitate orizontala) sau prind adaugarea de resurse hardware(scalabilitate verticala), fara oprirea sistemului. In momentul in care exista foarte multe request-uri si nu avem suficiente resurse hardware pentru a le procesa sau nu mai exista spatiu pe disc pentru a gestiona cantitatile mari de date existente se folosesc conceptele de replicare si de sharding, acestea nefiind foarte optime.

Prima tehnica folosita in bazele de date relationale este shardind-ul care presupune ca partitionarea datelor sa se faca pe mai multe servere. Algoritmul de partitionare se poate face in multe feluri, algoritmul potrivit putand fi gasit dupa mai multe teste de performanta [14].

In figura urmatoare, Fig. 4.10 este prezentat un serviciu de identificare a shard-urilor pe baza algoritmului de partitionare:

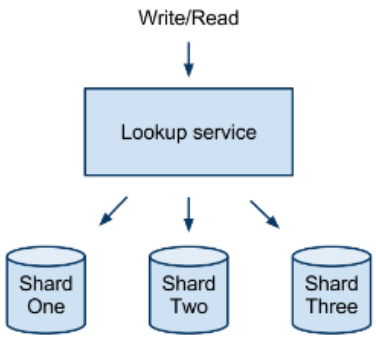

Fig. 4.10 Serviciu de identificare a shard-urilor

Cea de-a doua tehnica folosita in bazele de date relationale este replicarea datelor pe mai multe servere, crescandu-se astfel numarul de citiri care se pot efectua. O strategie de replicare standard folosita este implementata prin crearea unui cluster master si a mai multor clustere slave. Prin folosirea acestei strategii toate scrierile in baza de date sunt trimise catre master, iar toate citirile sunt trimise spre slave. Acesta strategie este prezentata in figura urmatore, Fig 4.11 :

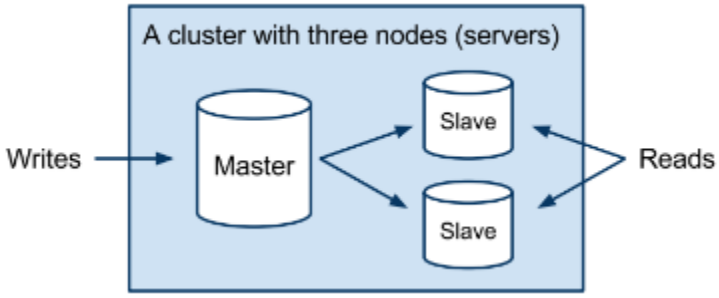

Fig. 4.11 Exemplificarea strategiei de replicare standard

4.4.4.2. Teorema CAP

Teorema CAP sau teorema lui Brewer a aparut in toamna anului 1998, fiind publica un an mai tarziu sub numele de principiul CAP. Teorema afirma faptul ca este imposibil ca un sistem distribuit sa atinga simultan toate cele 3 stari : Consistenta, Disponibilitate si Toleranta la partitionare. Teorema CAP a fost creata pentru a descrie mai precis modul in care pot fi echilibrate consistenta si alte proprietati intr-un sistem cu scalabilitate liniara. Intr-un sistem distribuit doar doua proprietati pot fi in relatie, deoarece este imposibil ca toate cele 3 stari sa fie atinse. Aceasta teorema nu se refera in mod direct la principiul de implementare a bazei de date Cassandra, ci ne ajuta sa intelegem mai bine scopul pentru care a fost creata. Bazele de date RDBMS sunt construite pe principiul CA (consistenta-disponibilitate). In figura urmatoare, Fig. 4.12 sunt prezentate cele mai comune baze de date folosite in functie de principiul la care se adapteaza [14][15].

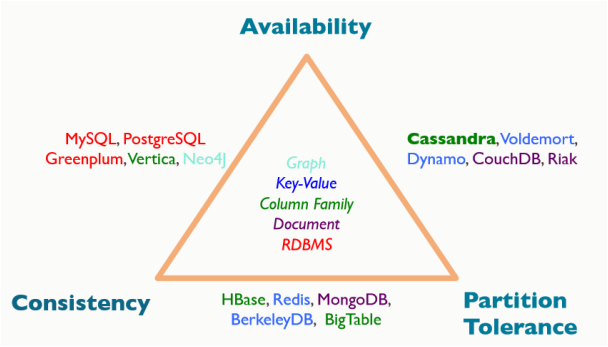

Fig. 4.12 Teorema CAP

Pentru a intelege mai bine modul de distribuire a bazelor de date in functie de criterile in care se incadreaza, mai jos este realizata o descrierea a componentelor teoremei CAP:

- **Consistență**

Conform teoremei CAP pentru ca un sistem sa fie consistent trebuie sa returneze aceleasi rezultate atunci cand sunt interogate toate nodurile. Pentru a respecta acest principiu fiecare data care este scrisa pe un nod trebuie sa fie replicata si pe celelalte noduri, altfel acest principiu nu mai este valid.

- **Disponibilitate**

Conform teoremei CAP pentru a se respecta acest principiu, sistemul trebuie sa ofera o garantie ca de fiecare data cand exista o cerere din partea unui client, sistemul poate sa ofere un raspuns la cererea venita cu privire la faptul ca respectiva cerere a avut succes sau nu. Acest principiu se bazeaza pe regula totul sau nimic, adica intreg sistemul este disponibil sau intreg sistemul este cazut.

- **Toleranță la partiționare**

Conform teoremei CAP pentru respectarea acestui principiu sistemul trebuie sa continue sa functioneze normal(sa raspunda cu date corecte si valide) in ciuda aparitiei anumitor defectiuni asupra unor parti ale sistemului. Pentru a intelege mai bine acest principiu putem considera nodurile unui sistem, aceste fiind partitionate folosindu-se doua partitii. In momentul in care nodurile nu mai pot comunica intre ele(se taie cablul de internet sau nu mai exista internet de la furnizor) si se executa interogari asupra oricarei partitii din cele doua existente trebuie sa se returneze datele corecte si valide.

4.4.4.3. NoSQL

În momentul de fața toate aplicatiile existente care ofera servicii pentru un numar foarte mare de utilizatori folosesc pentru procesare conceptul de sisteme distribuite. Un sistem distribuit este alcatuit din mai multe calculatoare (noduri) care comunica intre ele prin internet pentru a indeplini o anumita sarcina prin partajarea resurselor. Intr-un sistem distribuit se introduce conceptul de scalabilitate verticala si scalabilitate orizontala.

O baza de date NoSQL este o baza de date non-relationala, conceptul care sta la baza acestor baze de date fiind complet diferit de conceptul care sta la baza bazelor de date relationale. Conceputul de baze de date NoSQL a fost introdus pentru stocarea datelor intr-un mod distribuit, unde scalarea continutului stocat este foarte importanta. Acest tip de baze de date nu necesita o schema fixa, evita folosirea join-urilor pentru interogarea datelor stocate si se bazeaza pe principiul de scalare orizontala. A fost nevoie de crearea unui nou tip de baze de date care sa ofere suport pentru stocare distribuita a continutului, suport pentru scalarea orizontala si pentru obtinerea unei performante mult mai ridicate deoarece in momentul de fata exista o multime de aplicatii care gestioneaza in fiecare zi cantitati mari de date venite de la utilizatori.

Principalele baze de date NoSQL folosite in momentul actual sunt:

- **Google BigTable**

A fost printre primele baze de date NoSQL create de Google. BigTable este folosit de Google in mecanismul de cautare, in Google Earth si Google Finance. In termenii teoremei CAP, BigTable se bazeaza pe CA (consistenta-disponibilitate). Cassandra a impumutat unele idei din implementarea acestei baze de date, modelul de date fiind tinut in memorie si scris pe disc intr-un mod eficient.

- **Amazon Dynamo**

In timp ce Google isi dezvolta propriul sistem distribuit de stocare a datelor, Amazon a inceput sa lucreze la propriul sistem de stocare numit Amazon Dynamo. Modelul de date pe care se bazeaza cei de la Amazon Dynamo este foarte simplu. In termenii teoremei CAP aceasta baza de date se bazeaza mai mult pe toleranta la partitii si disponibilitate. Spre deosebire de BigTable care are un singur punct de esec, in Dynamo toate nodurile au aceiasi importanta (in cazul in care un nod pica sistemul va continua sa functioneze in parametrii normali). Cassandra a imprumutat acest prinicipiu de design de la Dynamo, reusind sa imbine design-ul oferit de BigTable cu design-ul oferit de Dynamo.

- **Cassandra**

Cassandra este o baza de date distribuita, fiind perfecta pentru a gestiona cantitati mari de date de pe mai multe noduri(servere). Cassandra se bazeaza pe principiul de functionare al BigTable si pe design-ul de toleranta la partitii oferit de Amazon Dynamo. Design-ul care sta la baza implementarii acestei baze de date ofera disponibilitate continua, scalabilitate liniara pe mai multe servere fara a exista un punct de esec al bazei de date. Cassandra ofera suport pentru scalabilitate(liniara/verticala), performanta bazei de date fiind imbunatatita prin adaugarea de noduri in cluster.

- **Neo4J**

Este o baza de date orientata pe structura de graf, fiind implementata in limbajul Java. O astfel de baza de date se foloseste in momentul in care exista relatii intre date. Daca folosim o baza de date relationala pentru a stoca un astfel de model de date nu vom obtine o performanta acceptabila in momentul in care dorim sa parcurgem cantitati mari de date.

Aceasta baza de date este foarte folosita in implementarea celor mai mari retele de socializare(Facebook, Twitter, Yammer si Google+) cat si in in cazul aplicatilor care afiseaza continut video(Youtube, Flickr) deoarece exista foarte multe relatii intre date.

În tabelul 4.11 sunt prezentate cateva caracteristici si avantaje ale unei baze de date orientate pe structura de graf.

Neo4J	
Caracteristici	**Avantaje**
CQL este limbajul folosit pentru interogarea bazei de date.	Relatiile dintre date sunt foarte usor de reprezentat.
Respecta principiile modului da date al unui graf.	Traversarea si interogarea bazei de date se face foarte simplu si rapid.
Suporta configurarea de indecsi si constrangeri unice.	Limbajul CQL este foarte usor de invatat
Ofera suport pentru proprietatile ACID.	Foloseste un model de date simplu si foarte puternic
Datele pot fi exportate in format JSON sau XLS.	Nu sunt necesare JOIN-urile pentru a extrage informatii complexe.

Tabel 4.11 Caracteristici si avantaje ale bazei de date Neo4J

4.4.5. Tehnologii folosite pentru implementarea bazei de date

În acest sub-capitol sunt prezentate tehnologiile folosite pentru implementarea bazei de date a sistemului si tehnicile prin care se poate ajunge la un timp de raspuns mai ridicat prin crearea si optimizarea bazei de date.

Cassandra este o baza de date NoSQL, iar in cele ce urmeaza sunt prezentate detalii despre arhitectura, structura si despre principalele aspecte de configurare care ajuta la o performanta ridicata a acesteia.

In figura, Fig 4.12 este prezentata o comparatie intre cele mai comune baze de date folosite.

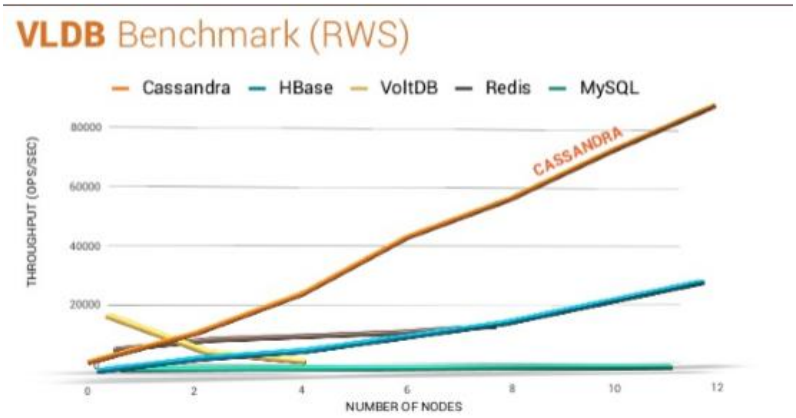

Fig. 4.13 Comparatie intre cele mai comune baze de date folosite

Dacă presupunem ca inițial avem 2 noduri in cluster, Cassandra poate suporta 100.000 de tranzacții pe secundă, dar în momentul în care adaugăm noduri în cluster observăm că numărul de tranzacții suportate crește liniar cu numarul de noduri adăugate.

Acest aspect se mai numește și scalabilitate liniara, un concept pe baza căruia Cassandra a fost construită, acesta putând fi observată în figura de mai jos, Fig 4.13:

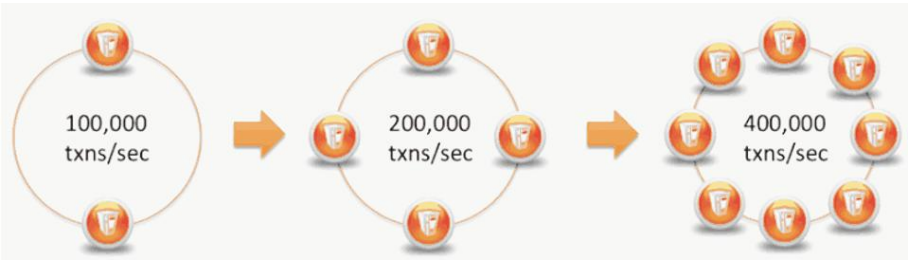

Fig. 4.14 Exemplu al scalabilitatii liniara in Cassandra

Cassandra a fost conceputa pentru a lucra cu cantitati mari de date de-a lungul mai multor noduri fara a exista vreun punct de esec. In fiecare secunda fiecare nod din cluster schimba informatii cu alte noduri existente in cluster folosind protocolul Gossip. Pentru a asigura durabilitatea datelor Cassandra foloseste un commit log pentru fiecare nod in care se scrie secvential fiecare activitate. Dupa scrierea in commit log data este indexata si scrisa intr-un Memtable, acesta comportandu-se ca o memorie cache. In momentul in care Memtabel-ul este plin datele sunt scrise pe disc intr-o structura numita SSTable, toate scrierile fiind in mod automat partitionate si replicate in intreg cluster-ul.

Pentru a efectua operatia de *„merge”* asupra acestor SSTable-uri, Cassandra foloseste un proces numit compactare care consolideaza periodic SSTabel-urile, asigurandu-se astfel ca datele irelevante sunt sterse. Cassandra este o baza de date row-oriented, iar fiecare cerere de citire sau scriere putand fi procesata de orice nod din cluster. In momentul in care un client se conecteaza la un nod din cluster pentru a-i procesa cererea acest nod va lua functia de coordonator. Acest coordonator se comporta ca un proxy intre aplicatie si nodurile care detin datele cerute. Coordonatorul stabileste care noduri din inel/cluster (depinde de modul in care a fost configurat cluster-ul) contin acele date si ar trebui sa se ocupe de cererea clientului [14].

In figura urmatoare, Fig 4.14 sunt reprezentate principalele componente care stau la baza arhitecturii bazei de date:

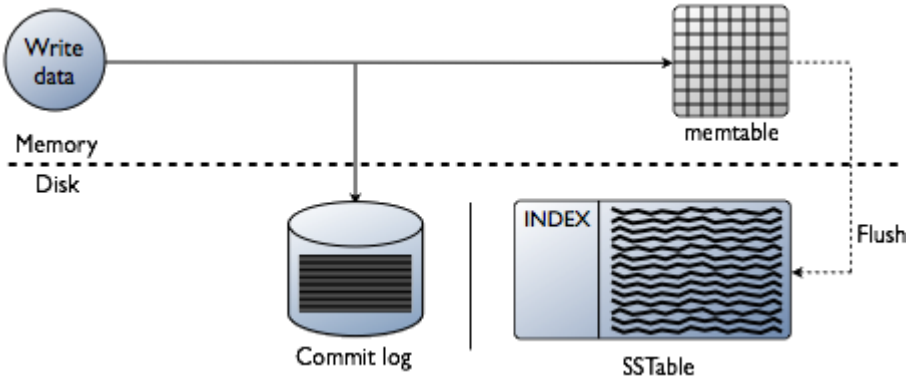

Fig. 4.15 Arhitectura Cassandra

1. Principalele elemente de configurare

Pentru a obtine o disponibiliatte si o performanta cat mai buna exista 3 criterii principale care trebuie precizate si configurate(daca este cazul):

a. **Protocolul Gossip** – este un protocol de comunicare peer-to – peer pentru a descoperi si distribui locatia si informatiile de stare despre celelalte noduri din cluster.
b. **Partitionarea** – determina modul in care sunt distribuite datele de-a lungul nodurilor intr-un cluster si in care nod va fi plasata prima copie a datelor. Fiecare rand de date este reprezentat printr-o cheie de partitionare si este distribuita de-a lungul cluster-ului prin valoarea token-ului. Strategia de partitionare folosita implicit de Cassandra este Murmur3Partitioner.
c. **Factor de replicare** – reprezinta numarul total de replici de-a lungul cluster-ului. Un factor de replicare de 1 inseamna ca exista o singura copie a fiecarui rand pe un nod. Un factor de replicare de 2 ne asigura ca exista 2 copii pentru fiecare rand situate pe 2 noduri diferite. Factorul de replicare se configureaza pentru fiecare centru de date. In general factorul de replicare trebuie sa fie mai mare de 1 pentru a respecta principiile inpuse de teorema CAP, dar nu mai mare decat numarul nodurilor din cluster.

2. Structura bazei de date

Cassandra este distribuita pe mai multe masini de lucru(servere) care functioneaza impreuna prin folosirea protocolului Gossip. Locul in care sunt grupate toate aceste unitati de procesare se numeste cluster. Fiecare nod contine o replica, deci in cazul in care apar erori asupra unui nod replica acestuia ii ia locul.

Principalele componente stocate pe un nod din cluster sunt :

1. **Keyspace :** este un container pentru datele stocate in baza de date. Principalele atribute sunt:
   - Factorul de replicare : numarul de noduri care vor primi copii ale acelorasi date
   - Strategia de replicare
   - Column family : keyspace-ul este un container pentru o lista de unul sau mai multe column family. La randul sau fiecare column family este un container pentru o colectie de randuri.
     Pentru a realiza o comparatie intre o baza de date relationala si o baza de date NoSQL, reprezentarea unei familii de coloane intr-o baza de date relationala poarta numele de tabel.

In figura, Fig 4.14 este prezentata structura unui keyspace.

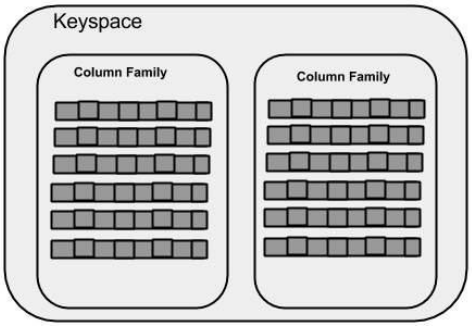

Fig. 4.16 Structura unui keyspace

2. **Column family :** reprezinta o colectie ordonata de randuri. Fiecare rand contine o colectie ordonata de coloane. Intr-o baza de date relationala schema este fixa. Dupa ce definim anumite coloane pentru o tabela in momentul inserarii datelor trebuie sa definim o valoare pentru fiecare coloana a tabelului. In Cassandra se pot adauga coloane in mod liber si nu trebuie specificata cate o valoare pentru fiecare coloana existenta in momentul inserarii datelor. In figura, Fig 4.15 este prezentata structura unei familii de coloane.

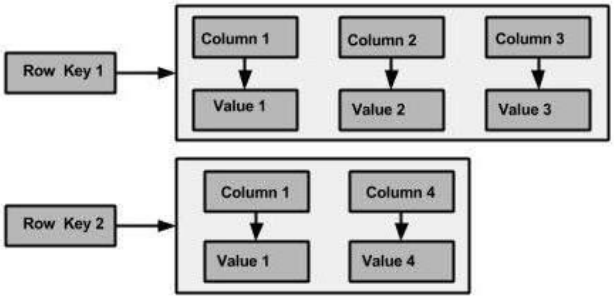

Fig. 4.17 Structura unei familii de coloane

3. **Column :** este o structura de date cu 3 valori(numele cheii, valoarea scrisa la acea coloana si un indicator de timp). In figura, Fig 4.16 este prezentata structura unei coloane.

Column		
name : byte[]	value : byte[]	clock : clock[]

Fig. 4.18 Structura unei coloane

3. Comparație între RDBMS și Cassandra

În tabelul urmator, tabelul 4.10 este prezentata o comparatie intre caracteristicile si functionalitatile pe care le ofera o baza de date relationala in raport cu o baza de date NoSQL.

RDBMS	Cassandra
Bazele de date relationale se ocupa cu date structurate	Cassandra se ocupa cu date nestructurate.
Foloseste o schema fixa.	Are o schema flexibila.
Tabelele sunt entitatile unei baze de date relationale.	Familiile de coloane sunt entitatile unui keyspace.
Fiecare rand reprezinta o inregistrare individuala.	Fiecare rand este o unitate de replicare.
Coloanele reprezinta atributele unei relatii.	In Cassandra o coloana este privita ca o unitate de stocare.
Sustine conceptul de chei straine.	Relatiile sunt reprezentate prin folosirea colectiilor.

Tabel 4.12 Comparație între RDBMS si Cassandra

În concluzie, am ales sa folosesc aceasta baza de date pentru a asigura persistenta datelor din urmatoarele motive:

- Cassandra ofera disponibilitate continua si un factor de replicare a datelor configurabil
- Cassandra se ocupa cu date nestructurate, nu exista relatii intre tabele, fiecare tabel fiind creat pentru a servi raspuns la cate o interogare.
- Scrierea datelor se face secvential folosindu-se cheia de partitionare stabilita. Prin stabilirea unei chei de partitionare ideale citirile din baza de date sunt foarte rapide.
- Cassandra ofera posibilitatea de a folosi colectii ca valori memorate intr-o coloana
- Ofera simplitate in scrierea interogarilor, deci se evita folosirea interogarilor complexe prin folosirea join-urilor.

Neo4J este o bază de date a carei arhitectura este orientata pe structura de grafuri. Pentru a asigura relatiile dintre utilizatori puteam folosi o baza de date relationala, MySQL reusind sa indeplineasca cu succes cerintele functionale, dar nu si cerintele non-functionale cu privire la performanta si timpul de raspuns. Cassandra nu reprezinta un candidat ideal pentru indeplinirea acestor cerinte deoarece criterile pentru care a fost

creata ne impiedica sa asiguram consistenta datelor in raport cu performanta. Deoarece problema pe care dorim sa o rezolvam implica relatii intre utilizatori, o baza de date orientata pe grafuri este ideala. In momentul actual, Facebook ofera un Graph API care se bazeaza pe structura de grafuri, acest API fiind folosit pentru cautarea de persoane, fotografii si evenimente. Pentru a contrui modelul de date, Neo4J foloseste noduri care contin informatii structurate si nestructurate. In figura, Fig. 4.19 este prezentat un exemplu de modelul de date realizat pentru a reprezenta relatia dintre doua persoane :

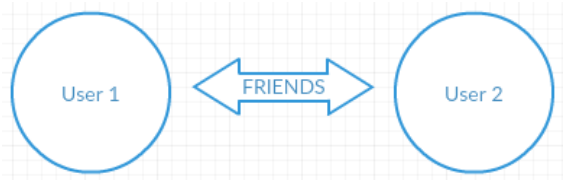

Fig. 4.19 Reprezentarea modelului de date in Neo4J

Arhitectura Neo4j este realizată pe mai multe nivele, fiecare nivel interactionand cu baza de date intr-un mod specific. In figura, Fig. 4.18 este prezentata arhitectura conceptuala a bazei de date Neo4J si principalele nivele :

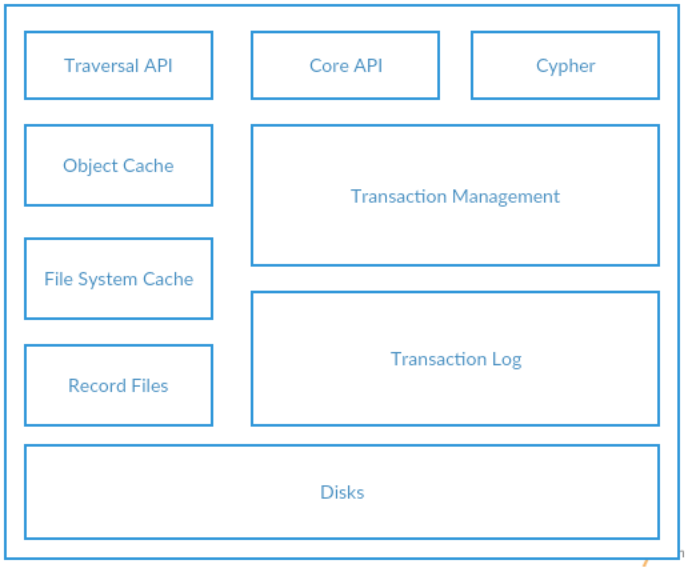

Fig. 4.20 Arhitectura conceptuala a bazei de date Neo4J

Pentru a interactiona cu alte sisteme prin formatul JSON, Neo4J ofera trei API-uri, Traversal API, Core API si Cypher. Pentru optimizarea performantei si a timpului de raspuns, Neo4J ofera si un mecanism de caching. De asemenea baza de date suport proprietatile ACID, oferind suport tranzactional.

In figura, Fig 4.19 este prezentat un exemplu al unui model de date folosit pentru modelarea relatiilor de prietenie existente intre utilizatori. Nodul marcat cu „X” este folosit ca nod de pornire in executia unei interogari, un exemple de interogare pe care o putem executa fiind „*afisarea listei de prieteni pe care o are un anumit utilizator”*. Observam ca Neo4J ne permite foarte usor sa ne stabilim modelul de date si sa executam

foarte rapid o interogare asupra acestuia. Intr-o baza de date relationala, pentru a indeplini aceasta cerinta este necesara folosirea unui JOIN intre doua tabele, rezultand astfel o performanta mai scazuta [16].

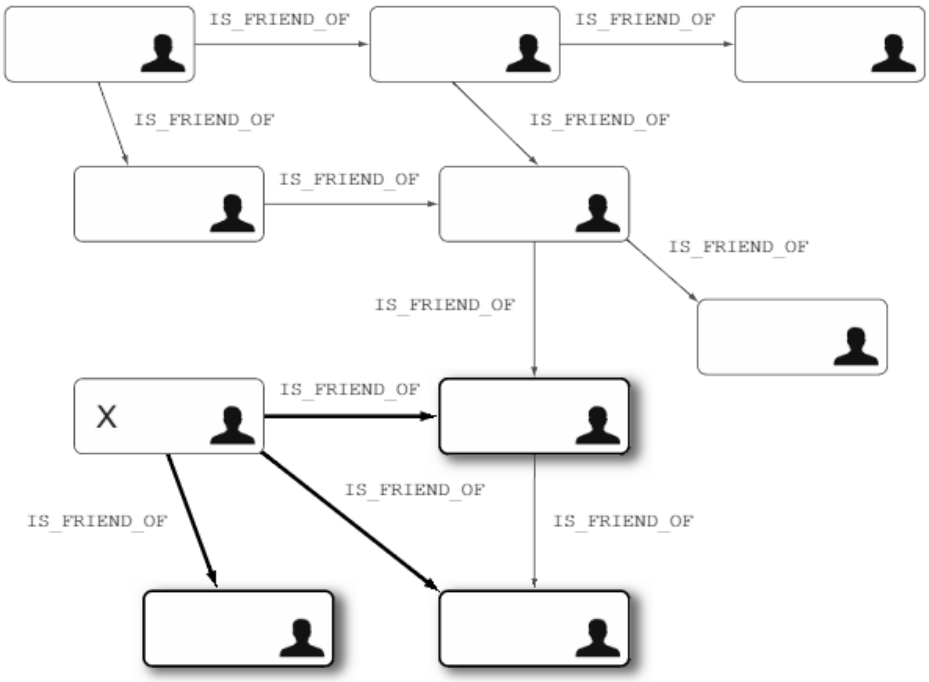

Fig. 4.21 Arhitectura conceptuală a unei baze de date de tip graf

Motivația de a alege o baza de date NoSQL s-a datorat în primul rând cerințelor non-funcționale care au fost stabilite la începutul proiectului. S-a folosit o baza de date graf deoarece modelul de date conține foarte multe relații și pentru că, consistența datelor este foarte importantă.

Capitolul 5. Proiectare de Detaliu și Implementare

Acest capitol cuprinde schema generală a sistemului, descrierea componentelor implementate la nivel de modul și diagrama de clase împreună cu o descriere a celor mai importante clase și metode folosite.

5.1. Structura generala a sistemului

Lifestory este o aplicație orientată pe arhitectura client-server, în care partea de Server este implementată în JavaScript, având la bază framework-ul Express 4.0, iar partea de Client a fost realizată prin folosirea tehnologiilor Web existente în momentul actual(HTML, CSS și JavaScript). Pentru asigurarea persistentei datelor am folosit două baze de date NoSQL(Cassandra si Neo4J).

Arhitectura aplicației Server respectă modelul arhitecturii pe trei nivele. Aplicația Client interactionează cu aplicația Server prin servicii REST. Aplicatia Client comunică cu nivelul superior al aplicației Server, iar la rândul său nivelul superior comunică nu nivelul de business, iar nivelul de business comunică cu nivelul de date.

Modelul Three-tier este un model bazat pe arhitectura client-server, în care interacțiunea cu utilizatorul, logica aplicației și accesul la date se face prin structurarea aplicației în module diferite, posibil stocate pe platforme diferite.

În această aplicație, modelul Three-tier a fost implementat prin folosirea urmatoarelor nivele: **Nivelul de prezentare**, **Nivelul logic** al aplicației și **Nivelul de date**.

1. Nivelul de prezentare

Nivelul de prezentare este nivelul superior al aplicației. Fiecare cerere care vine de la client este interceptată de acest nivel. Acest nivel se ocupă si de randarea asincronă a interfeței utilizator, comunicarea cu interfața utilizator facându-se prin mesaje în format JSON. Acest nivel interacționeaza în mod direct cu nivelul logic al aplicației pentru a obține datele necesare din baza de date conform cererințelor venite de la utilizatori.

2. Nivelul logic al aplicației

Nivelul logic al aplicației este nivelul din mijloc, care interacționeaza cu nivelul de date pentru a efectua operații de citire, adăugare, actualizare sau ștergere a datelor. Nivelul logic se ocupă și de tratarea exceptiilor care pot aparea în cazul în care baza de date nu funcționează în mod corespunzator.

În acest nivel este introdusă toata logica aplicației, de la procesarea si stocarea datelor pana la validarea datelor furnizate de nivelul superior.

3. Nivelul de date

Nivelul de date este alcătuit din serverele bazei de date și se ocupa cu stocarea sau retragerea datelor din baza de date pe baza interogarilor efectuate. Prin folosirea acestui nivel, nivelul logic al aplicației efectuează operații asupra bazelor de date existente. Acest

nivel comunica cu nivelul superior, nivelul logic al aplicației prin mesaje in format JSON în cazul în care exista date in baza de date sau prin mesaje de eroare specifice în cazul în care apar erori neprevazute sau baza de date nu funcționeaza corespunzator.

Pentru dezvolarea aplicației am ales framework-ul Express 4.0 deoarece pune la dispoziția dezvoltatorilor o mulțime de funcționalități și oferă o simplitate ridicată în dezvoltarea aplicaților Web.

În figura, Fig 5.1 este prezentată structura de nivel inalt a sistemului și principalele componente de interacțiune:

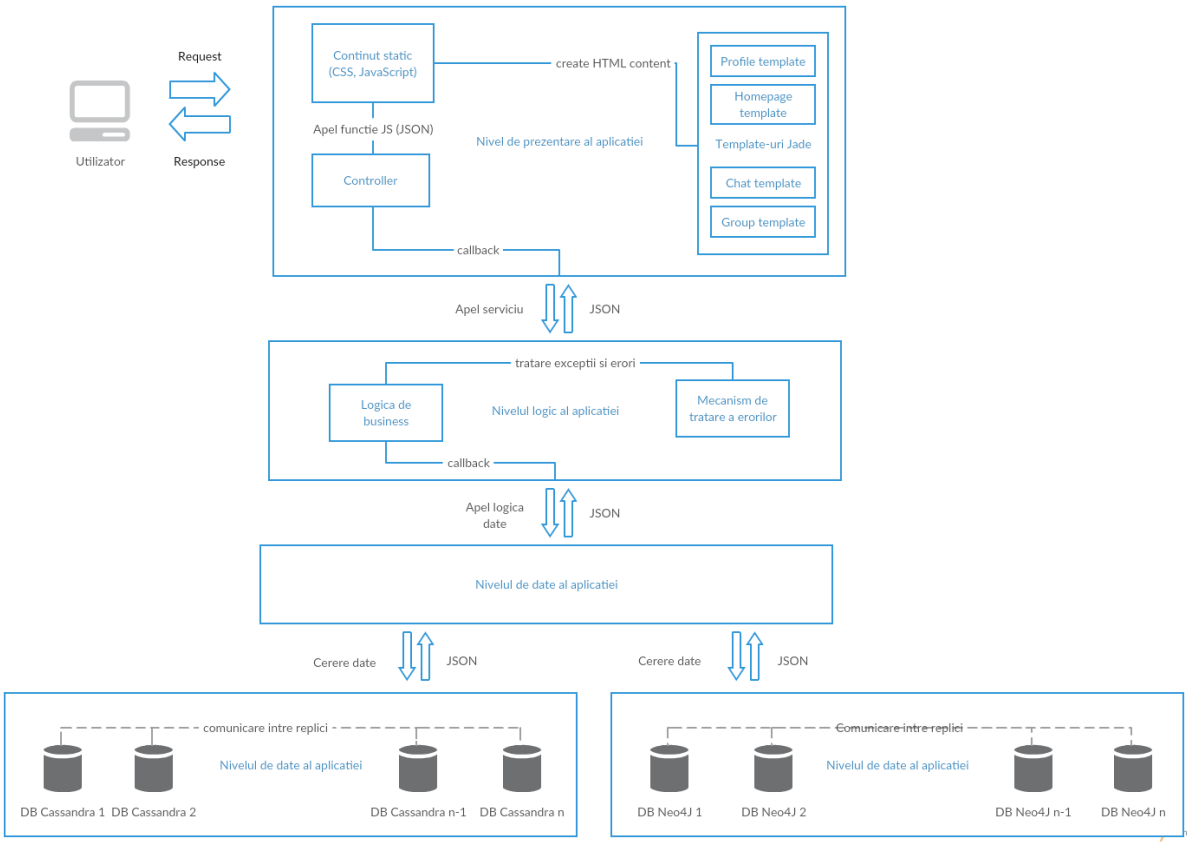

Fig. 5.1 Structura de nivel înalt a sistemului

5.2. Aplicația Server – detalii de implementare

Aplicația Server a fost dezvoltată dupa structura unei aplicații Web, structura fiind creată în mod automat de framework-ul folosit. Structura unei aplicații Web care folosește framework-ul Express 4.0 este prezentată în figura următoare, Fig. 5.1 :

```
|-- app.js
|-- config
|   |-- config.js - (your app configuration goes here)
|   |-- router.js - (tour router goes here that passes app variables in all modules)
|   |-- util.js - ( your global utility function goes here )
|   |-- //other files
|-- credential
|   |-- //credential if needed
|-- lang // this folder content all language supported files
|   |-- en.js  //
|-- lib.js // this file holds all global variables
|-- public // your public folder code
|   |-- js
|   |-- css
|   |-- image
|   |-- index.html
|   |-- // other folder
|-- modules // this folder holds all modules
|   |-- user
|   |   |-- controller.js (your user controller functinalities)
|   |   |-- index.js ( your user controller routes )
|   |   |-- model.js ( your user schema )
|   |   |-- util ( user controller utility files)
|   |       |-- authorization.js  (some files)
|   |-- // other modiles
|-- package.json // your package .json files
```

Fig. 5.2 Structura unei aplicații folosind framework-ul Express 4.0

În structura prezentată în Fig. 5.2, în directorul *„modules”* au fost adaugate următoarele module:

- Modulul pentru accesul la baza de date(*„database”)*
- Modulul pentru tratarea excepțiilor (*„exceptions”)*
- Modulul pentru rutarea adreselor URL(*„routes”)*
- Modulul corespunzator nivelului logic al aplicației *(„services”)*
- Modul corespunzator template-urilor *(„views”)*

În folder-ul *„public”* sunt adăugate toate fișierele CSS și Javascript și imaginile, acest fișier stocând tot conținutul public al aplicației, iar în folder-ul *„config”* sunt cuprinse toate configurările sistemului.

Fișierul *„package.json”* conține toate dependențele proiectului spre alte module, rolul acestuia fiind de a simplifica etapa de instalare a proiectului pe alt server.

Pentru a crea server-ul propriu-zis care ascultă cererile venite de la clienti există un fișier principal numit *„app.js”* în care sunt instanțiate toate modulele necesare pentru crearea unei aplicații Web folosind Express 4.0.

În figura următoare, Fig 5.3 se poate vizualiza modul de configurare a unui server Web care ascultă cereri pe portul 80 și modul prin care au fost incluse rutele și mapările adreselor URL:

```
var express = require('express');
var app = module.exports.app = express();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var path = require('path');
var logIn = require(path.resolve('routes/logIn/logIn'));
var register = require(path.resolve('routes/register/register'));
var user = require(path.resolve("model/user"));
var _redis = require("redis");
var redis = module.exports.redis = _redis.createClient();
var config = require(path.resolve('database/config'));
var jwt = module.exports.jwt = require('jsonwebtoken');
var profile = require(path.resolve('routes/profile/profile'));
var homepage = require(path.resolve('routes/homepage/homepage'));
var friends = require(path.resolve('routes/friends/friends'));
var wall = require(path.resolve('routes/common/wall/wall'));
var jade = require("jade");
var fs=require('fs');

app.use('/js', express.static('public/javascripts'));
app.use('/css', express.static('public/stylesheets'));
app.use('/photo', express.static('public/images/uploads'));
app.use('/logos', express.static('public/images/logos'));
app.use('/login', logIn);
app.use('/register', register);
app.use("/profile", profile);
app.use("/profile", express.static('views/profile'));
app.use("/common", express.static('views/common'));
app.use("/homepage", express.static('views/homepage'));
app.use("/homepage", homepage);
app.use("/friends", friends);
app.use("/wall", wall);

app.get('/', function (req, res) {
    res.send("Express 4.0");
});

http.listen(80, '127.0.0.1');
```

Fig. 5.3 Crerea unui server Web folosind Express 4.0

Principala motivație de a alege acest framework pentru dezvoltarea acestei aplicții Web este reprezentată de simplitatea și ușurința prin care se poate crea un server Web folosind Express 4.0. NodeJS pune la dispoziție o mulțime de module, simplificând astfel munca dezvoltatorului.

5.2.1. Nivelul de prezentare

Nivelul de prezentare conține fișierele statice (imagini, fisiere *CSS* și fisiere *JavaScript*) care asigură interacțiunea dinamica dintre utilizator și sistem. În momentul în care un utilizator face o cerere catre serverele aplicației (request *AJAX* sau accesarea unui anumit URL), serverele răspund cererii clientului prin apelarea ueni funcții definite în fisierele statice JavaScript. Funcția apelată primește ca parametru datele în format JSON și le transmite unui template Jade corespunzator pentru a produce conținutul HTML ce trebuie afișat în pagină. Fișierele care stau la baza design-ului aplicației se află de asemenea tot în acest nivel.

De asemenea acest nivel conține clasele care mapează fiecare adresă URL la anumite resurse aflate pe servere sau în bazele de date. Aceste clase poartă denumirea de rutere în Express. Pentru a putea mapa o anumită adresă URL la o anumită resursă trebuie să ne definim câte o ruta, fiecare rută putând intercepta cereri de tip *GET* , *POST*, *PUT* sau *DELETE*. În clasa principală a aplicației am definit pentru fiecare rută principală o anumita clasă care se ocupă de cererile care vin pe URL-ul respectiv sau pe URL-urile derivate. În figura , Fig 5.4 putem vedea modul în care s-a realizat maparea cererilor de pe adresa *http://127.0.0.1/homepage* la o anumită clasa :

```
var homepage = require(path.resolve('routes/homepage/route'));
app.use("/homepage", homepage);
```

Fig. 5.4 Maparea unei adrese la un router

Fisierul „routes/homepage/route.js" contine toate adresele URL derivate de la adresa *http://127.0.0.1/homepage* (de exemplu adresa URL *http:// 127.0.0.1/homepage /photos* este mapata la acest fisier.)

În figura, Fig. 5.5 se poate vedea modul in care s-a realizat maparea adresei URL *http:// 127.0.0.1/homepage /photos* :

```
var express = require('express');
var router = express.Router();
router.get('/photos', function (req, res) {
  photoService.getAllPhotos(req.logged_user.id,function(error,result){
    if(error==null)
    {
      res.send(result);
    }
    else
    {
      //exception handling
    }
  });
});
module.exports = router;
```

Fig. 5.5 Maparea unei adrese URL

În figura, Fig. 5.4 am importat modulul Express si am creat un nou Router pentru a putea intercepta cererile venite pe anumite adrese URL. Performanta din spatele framework-ului NodeJS se bazeaza pe conceptul de callback-uri, fiecare callback sugerandu-i sistemului ca urmeaza sa se execute o actiune care interactioneaza cu baza de

date sau executa operatii de I/O . In momentul in care se observa exista unui callback, acesta este pus intr-o coada de asteptare si este rulat in paralel, astfel server-ul poate sa raspunda la alte cereri. Nivelul de prezentare apeleaza nivelul de servicii, care la randul sau interactioneaza cu nivelul de date. Revenirea de la un nivel inferior spre un nivel superior se face prin folosirea callback-urilor, callback-ul fiind folosit doar atunci cand actiunea s-a executat si exista un rezultat.

Toate cele trei nivele comunică între ele prin format JSON, la finalul interacțiunii nivelul de prezentare trebuie să trimită ca răspuns anumite fișiere CSS sau JavaScript (în funcție de cerința) sau să apeleze anumite funcții JavaScript care au ca parametru principal datele obținute din nivelul de business, în format JSON. În figura, Fig. 5.6 este prezentat un exemplu al unui raspuns la o cerere utilizator de pe o anumită adresă URL:

```
router.get('/photos', function (req, res) {
  photoService.getAllPhotos(req.logged_user.id,function(error,result){
    if(error==null)
    {
      res.write("<link rel='stylesheet' href='" + config.sitePath + "/css/profile/photos.css' type='text/css'/>");
      res.write("<script type='text/javascript' src='" + config.sitePath + "/js/profile/profile.js'></script>");
      res.send('<script>renderPhotosContent('+result+')</script>');
    }
    else
    {
      res.send('<script>displayPhotoRenderErrorMessage()</script>');
    }
  });
});
```

Fig. 5.6 Exemplul unui raspuns la o cerere utilizator

În momentul în care utilizatorul executa o cerere pe adresa http://127.0.0.1 /homepage/photos, server-ul intercepteaza cererea venita si apeleaza nivelul de business pentru a putea oferi un raspuns la cererea venita. Dupa executia si obtinerea datelor, nivelul de business apeleaza callback-ul aferent nivelului de prezentare pentru a-i trimite resursele cerute. Daca nu exista erori pe tot parcursul executiei, nivelul de prezentare trimite ca raspuns doua fisiere statice si apeleaza o functie JavaScript avand ca paramtrul fisierul JSON obtinut anterior. In caz de eroare sistemul va apela de asemenea o functie JavaScript corespunzatoare erorii aparute.

Pentru a imbunatati timpul de raspuns al aplicatiei am folosit o tehnica de randare asincrona a interfetei utilizator numita BigPipe. Aceasta tehnica dezvoltata initial de Facebook functioneaza astfel: *În momentul în care utilizatorul face o cerere pentru afișarea unei anumite pagini, se afișeaza inițial un template care nu conține foarte mult cod JavaScript și CSS, iar apoi se încarcă fiecare componentă în parte într-o manieră asincronă, fiecare componentă încarcându-și propriile fișiere CSS și JavaScript*. Prin folosirea acestei tehnici timpul de raspuns a fost îmbunătățit, iar satisfacția utilizatorului a crescut considerabil.

5.2.2. Nivelul de business

Nivelul de business se ocupa de intreaga logica a aplicatiei. Fiecare componenta a aplicatiei are propriul modul in care sunt definite serviciile folosite si tipurile de exceptii care pot aparea in urma interactiunii cu nivelul de date. Fiecare cerere utilizator interceptata in nivelul de prezentare apeleaza anumite servicii pentru a putea obtine resurse de pe server. Fiecare interactiune dintre utilizatori si servere trebuie sa fie validata

in nivelul de business pentru a putea interactiona cu restul serviciilor, altfel se va afisa un mesaj de eroare corespunzator. Validarea datelor se face atat pe partea de client(folosind JavaScript) cat si pe partea de server, validarea din urma fiind foarte importanta si necesara.

Pentru interactiunea nivelului de business cu nivelul de date se folosesc de asemenea callback-urile, deoarece operatiile asupra bazelor de date sunt consumatoare de timp. Pentru a imbunatati timpul de raspuns si pentru a reduce numarul de interactiuni cu baza de date nivelul de business interactioneaza cu un mecanism de caching folosit pentru a memora rezultatul obtinut in urma unei interogari.

In figura, Fig 5.7 este prezentata o functie care apeleaza serviciul de date pentru a incarca postarile pe pagina principala. Pentru a valida datele de intrare (id-ul utilizatorului in acest caz) am definit un filtru care se ocupa de validarea datelor de intrare. De asemenea se poate observa faptul ca in cazul unei erori venite din nivelul de date, nivelul de business trimite spre nivelul de prezentare o exceptie specifica nivelului de business.

```
var config = require(path.resolve("database/config"));
var wallException = require(path.resolve('exception/wall/WallException'));
var dao = require(path.resolve('database/homepage/wallConnection'));

function getPosts(user, callback) {
  filter.validateUserID(user.id,function(result){
      dao.getPosts(user.id,function(err,result){
          var error=null;
          if(err)
          {
              error = wallException.WallException("Wall cannot be loaded.");
          }
          callback(error,result);
      });
  });
}
```

Fig. 5.7 Definirea unei funcții în nivelul de business

În momentul în care nivelul de date returneaza un raspuns valid nivelului de business, acesta apeleaza un callback pentru a trimite datele spre nivelul superior. Acest nivel nu foloseste clase de model definite de dezvoltatori deoarece interactiunea intre toate cele 3 nivele se face pe baza formatului JSON, crearea de obiecte si distrugerea lor fiind consumatoare de timp.

În figura, Fig 5.8 este prezentata diagrama de interactiune dintre nivelul de prezentare si nivelul de business.

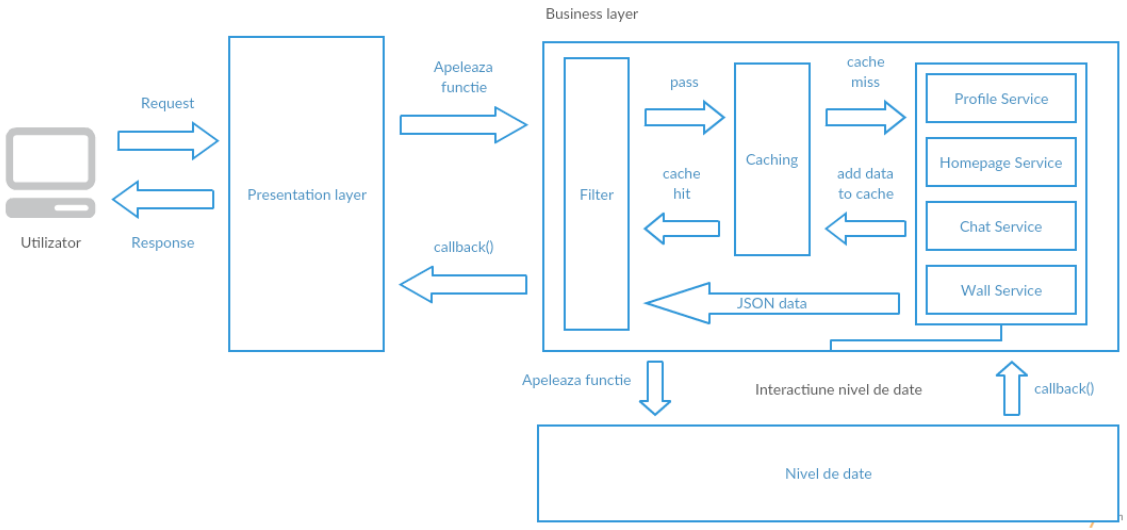

Fig. 5.8 Interacțiune între nivelul de prezentare și nivelul de business

În momentul în care utilizator face o cerere catre servere pentru anumite resurse, adresa URL pentru care se face cererea este mapata la un anumit Router in care sunt definite anumite functii pentru a obtine resursele cerute. Nivelul de business se ocupa cu definirea functiilor, filtrarea datelor de intrare si apelarea functilor corespunzatoare din nivelului de date pentru a obtine resursele dorite.

Pentru a procesa cererile de tip POST folosim modulul *„bodyparser”*, definit de framework-ul NodeJS cu scopul de a prelua continutul cererii. Pentru asigurarea validitatii datelor folosim o clasa de filtrare corespunzatoare deoarece cererile de acest tip lucreaza cu date senzitive care trebuie adaugate pe server. În figura, Fig 5.9 este prezentat modul de procesare si de validare a unei cereri de tip *POST* :

```
function addPost(loggedUser,content, callback) {
  filter.validatePostContent(content,function(validate){
    if(validate){
        dao.addPost(loggedUser,content, function (err, result) {
            var error = null;
            if (err != null) {
                error = wallException.InsertPostException("Post cannot be inserted.");
            }
            callback(error, result);
        });
    }
    else
    {
        var error=[];
        error.push({error:"Data is not valid."});
        callback(error,null);
    }
  });
}
```

Fig. 5.9 Filtrarea și procesarea unei cereri de tip POST

5.2.3. Nivelul de date

Nivelul de date interactioneaza cu doua baza de date NoSQL, Cassandra si Neo4J. Clasele corespunzatoare interactiunii directe cu baza de date se afla in directorul *database*, fiecare componenta avand definit propriul modul de interactiune. Pentru a asigura persistenta datelor si pentru a indeplini cerinte non-functionale s-a dovedit a fi necesara folosirea a doua baza de date NoSQL.

Prima baza de date folosita, Cassandra, este utilizata pentru a stoca postarile de pe profilele utilizatorilor, fotografiile de pe pagina de profil si conversatiile dintre utilizatori. Cassandra ofera o performanta de citire foarte buna deoarece datele sunt memorate secvential, ordonarea acestora realizandu-se dupa cheia de partitionare configurata. In cazul unor relatii intre anumite date, de exemplu in cazul relatiei de prietenie dintre doi sau mai multi utilizatori, Cassandra nu este cea mai buna alegere deoarece vor exista probleme de integritate a datelor.

Datorita cerintelor non-functionale impuse asupra sistemului cea mai buna solutie pentru a asigura intergritatea datelor si pentru a reprezenta relatii dintre utilizatori a fost folosirea unei baze de date de tip graf, Neo4J. Interactiunea intre nivelul de business si nivelul de date se bazeaza pe concepul de callback, dupa terminarea interactiunii directe cu baza de date se apeleaza callback-ul specific interactiunii cu nivelul superior. Nivelul de date poate arunca o multime de exceptii, exceptii datorate de multe ori functionarii defectuoase a bazei de date sau datorate modelului de date folosit.

În figura, Fig 5.10 este prezentat modul de interactiune intre modelul de business si modelul de date.

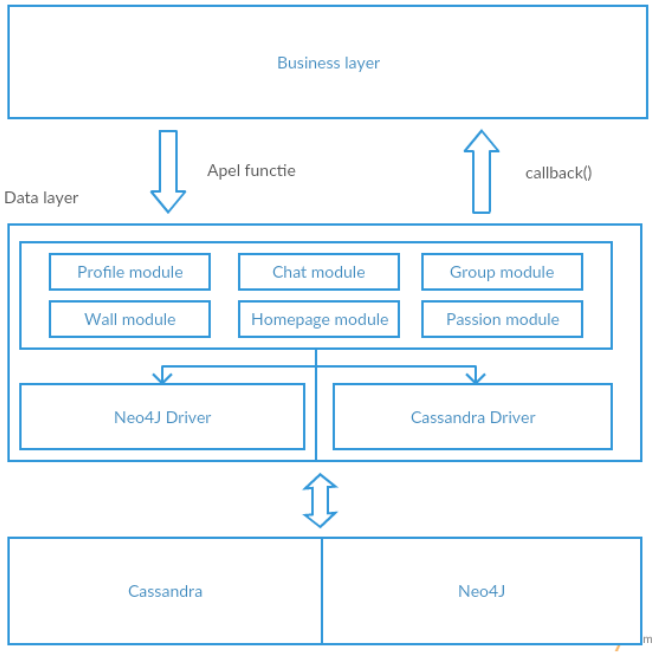

Fig. 5.10 Interacțiune între nivelul de business și nivelul de date

În figura de mai sus, Fig. 5.10 este prezentata interactiunea dintre nivelul de business si nivelul de date. In momentul in care nivelul de business efectueaza toate

validarile necesare si obtine un rezultat pozitiv, acesta interactioneaza cu nivelul de date prin apelarea unei functii specifice aflate in modulul corespunzator operatiei care se executa. In functie de tipul resurselor care se doresc de pe server, nivelul de date foloseste doua drivere pentru a se conecta la cele doua baze de date folosite, Cassandra si Neo4J. Fiecare din cele doua baze de date folosite returneaza datele in format JSON, interactiunea cu nivelul de business realizandu-se prin intermediul callback-urilor.

În figura, Fig. 5.11 este prezentata un exemplu de interactiune cu baza de date Cassandra pentru a returna informatiile de pe profilul unei persoane pe baza ID-ului acesteia. Pentru a interactiona cu baza de date am folosit CQL (Cassandra Query Language) asemanator cu SQL [17]:

```
var path = require('path');
var config = require(path.resolve("database/config"));
var cassandra = require('cassandra-driver');
var client = new cassandra.Client({contactPoints: ['' + config.address + ''], keyspace: '' + config.keyspace + ''});
var moment = require('moment');
var jsesc = require('jsesc');
var uuid = require('uuid');

var PROFILE_DETAILS = "profile_details";
var TEMPORARYPHOTOINCENTEROFATTENTION="temporaryphotoincenterofattention";

function getProfile(id, callback) {
    client.execute("SELECT * FROM profile WHERE id=" + id + "", function (err, result) {
        var row = null;
        if (!err) {
            if (result.rows.length == 1) {
                row = result.rows[0];
            }
        }
        callback(err, row);
    });
}
```

Fig. 5.11 Returnarea informațiilor de pe profilul unei persoane folosind CQL

În figura, Fig. 5.12 este prezentat un exemplu de interactiune cu baza de date Neo4J pentru a adauga o persoana in lista de persoane urmarite. Pentru a realiza interactiunea cu baza de date am folosit Cypher, un limbaj de interogare pentru bazele de date graf asemanator cu SQL [18]:

```
var neo4j = require('neo4j');
var connection = new neo4j.GraphDatabase('http://neo4j:david@localhost:7474');
var moment = require('moment');

function follow(userID, loggedUserID, callback) {
    connection.cypher({
        query: 'MATCH (u1:User{id:{id1}}),(u2:User{id:{id2}}) CREATE (u1)-[f:FOLLOW{date:{date}}]->(u2) return f',
        params: {
            id1: userID,
            id2: loggedUserID,
            date: moment(Date.now())._d
        }
    }, function (err) {
        if (err == null) {
            callback(null, true);
        }
        else {
            callback(err, false);
        }
    });
}

function unfollow(userID, loggedUserID, callback) {
    connection.cypher({
        query: 'MATCH (u1:User{id:{id1}})-[f:FOLLOW]->(u2:User{id:{id2}}) DELETE f',
        params: {
            id1: userID,
            id2: loggedUserID
        }
    }, function (err) {
        if (err == null) {
            callback(null, true);
        }
        else {
            callback(err, false);
        }
    });
}

module.exports.follow = follow;
module.exports.unfollow = unfollow;
```

Fig. 5.12 Exemplu de utilizare a limbajului Cypher

5.3. Proiectarea Bazei de Date

În acest subcapitol vor fi prezentate bazele de date alese pentru a asigura persistenta datelor cat si motivatia care sta la baza alegerii lor. In urmatoarele diagrame vor fi prezentate structurile de date folosite si relatiile dintre acestea.

Pentru a stoca informatii despre un anumit utilizator (profil personal sau postarile acestuia), Cassandra reprezinta cea mai buna alegere deoarece a fost implementata pentru a facilita accesul rapid la baza de date, acest lucru realizandu-se prin construirea tabelelor bazei de date in functie de interogarile care apar. Pentru a construi un model de date performant si pentru a beneficia de avantajele bazei de date, initial trebuie identificate interogarile care se executa asupra bazei de date si rezultatele care se asteapta.

După procesul de identificare a interogarilor se poate construi modelul bazei de date pentru fiecare interogare in parte, respectandu-se constrangerile bazei de date. Cassandra se bazeaza pe conceptul de denormalizare a datelor, principala tehnica de denormalizare folosita fiind reprezentata de *vederile materializate*. In momentul in care se alege o baza de date care suporta conceptul de denormalizare trebuie tinut cont de factorul de consistenta a bazei de date. Un design imperfect al bazei de date ar duce la o denormalizare imperfecta si ar crea o performanta mai scuzata in comparatie cu o baza de date normalizata.

In procesul de analiza si design al aplicatiei s-au identificat ca fiind cele mai utilizate interogari urmatoarele :

- Afisarea detaliilor pe pagina de profil a unui utilizator
- Afisarea postarilor de pe pagina de profil
- Afisarea prietenilor adaugati recent, a ultimelor fotografii postate si a pasiunilor si amintirilor create pe pagina de profil a unui utilizator.
- Afisarea fotografiilor si videoclipurilor incarcate de un utilizator
- Afisarea conversatiei dintre doi utilizatori
- Afisarea grupurilor in care se afla un anumit utilizator
- Afisarea continutului unui grup
- Afisarea pasiunilor adaugate de un anumit utilizator
- Afisarea amintirilor create de un utilizator

Pentru interogarile identificate in procesul de analiza si design folosirea unei baze de date relationale nu este foarte potrivita deoarece vor exista foarte multe relatii intre tabele, iar complexitatea interogarilor si a timpului de raspuns va creste proportional cu numarul de inregistrari in baza de date. Pentru a indeplini cerintele non-functionale trebuie aleasa o baza de date distribuita pentru a gestiona cantitatile mari de date ale sistemului.

O altă etapă importantă în crearea modelului bazei de date este reprezentat de alegerea cheii primare pentru fiecare tabel. Cassandra fost construită pe principiul cheii primare, datele memorâându-se secvențial pe disc și partiționarea acestora realizându-se pe baza cheii primare alese. În compoziția cheii primare mai pot intra și alte campuri dintr-o tabele, aceste campuri primind numele de *coloane de grupare*. În funcție de coloanele de grupare specificate, datele sunt memorate într-o anumita ordine (de exemplu pentru a stoca datele în ordine descrescatoare în funcție de timpul la care au fost adăugate in baza de date, putem specifica ca si coloana de grupare timpul la care au fost

adaugate datele). În urmatoarele diagrame se va prezenta structura bazei de date si motivatia pentru care s-a ales o anumita cheie primara sau in functie de tipul interogarii anumite coloana de grupare.

In figura, Fig. 5.13 este prezentata structura principalelor tabele si modul de configurare a cheilor primare, astfel incat fiecare tabela sa memoreze eficient datele :

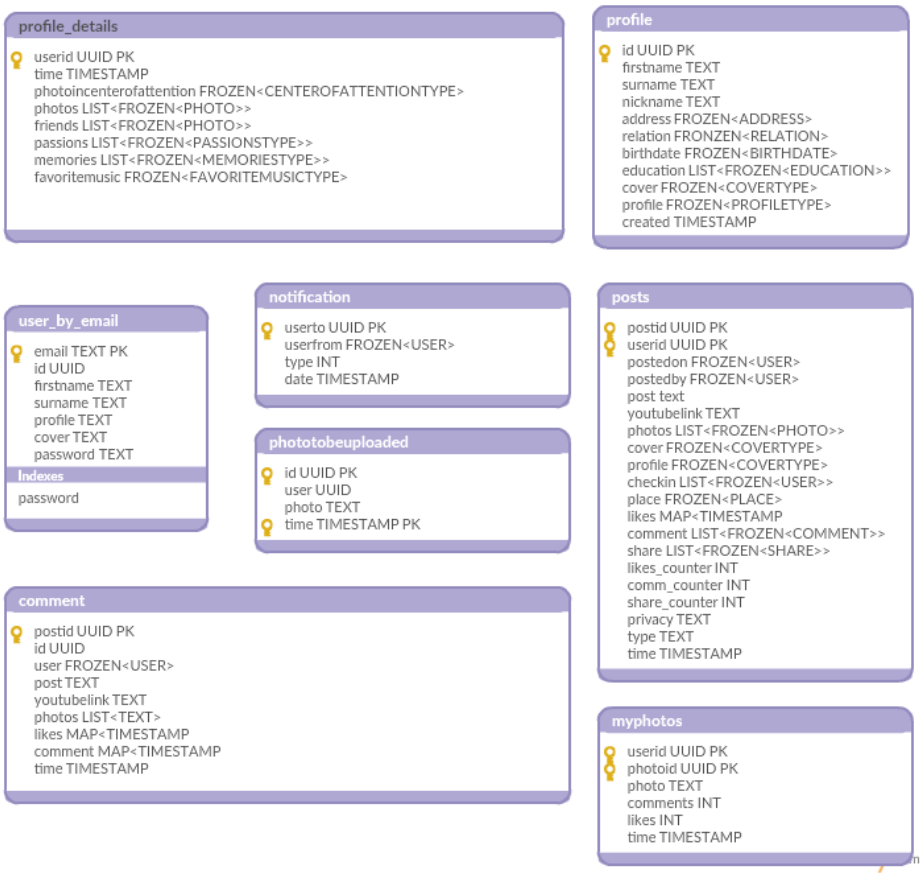

Fig. 5.13 Arhitectura bazei de date

Într-o baza de date NoSQL nu exista relatii intre tabele, continutul fiecarei tabele fiind creat pentru a satisface o anumita interogare. In figura, Fig 5.14 este prezentat un exemplu de creare a unui tabel in Cassandra:

```
create table temporary_user(
 accounttoken text,
 id uuid,
 email text,
 password text,
 firstname text,
 surname text,
 profile text,
 birthdate frozen<birthdate>,
 gender text,
 createdtime timestamp,
 PRIMARY KEY(accounttoken,createdtime)
);
```

Fig. 5.14 Crearea unui tabel in Cassandra

Tabelul exemplificat în Fig. 5.14 este folosit pentru a memora toate conturile create temporar pana in momentul in care acestea sunt activate printr-un token primit pe adresa de email specificata la inceputul crearii contului. Pentru aceasta tabela am folosit ca si cheie primara *token-ul de activare* corespunzator fiecarui cont utilizator, iar ca si camp folosit pentru grupare am folosit campul *createdtime.*

Pentru a micsora numarul de campuri definite pentru fiecare tabel, Cassandra ofera posibilitatea de a defini anumiti tipuri pentru a grupa campurile din tabele. In figura, Fig. 5.15 este prezentat modul prin care se poate defini un tip de date in Cassandra, referirea tipului definit intr-un tabel realizandu-se prin folosirea cuvantului *frozen*:

```
create type lifestory.birthdate(
 day int,
 month text,
 year int
);
```

Fig. 5.15 Definirea unui tip in Cassandra

Cassandra a fost folosita pentru a memora doar o parte din modelul de date deoarece prin reprezentarea integrala a modelului de date se crea o complexitate ridicata pentru a pastra baza de date consistenta.

Baza de date a fost folosita pentru a reprezenta urmatoarele functionalitati:

- Informatiile corespunzatoare fiecarui profil utilizator
- Postarile de pe profilul fiecarui utilizator
- Fotografiile incarcate de fiecare utilzator
- Videoclipurile incarcate de fiecare utilizator
- Pasiunile unui anumit utilizator
- Amintirile adaugate de un anumit utilizator
- Conversatiile dintre utilizatori

Pentru a reprezenta relatiile dintre utilizatori nu putem folosi conceptul de denormalizare, deoarece in momentul in care un utilizator isi schimba anumite informatii personale (fotografia de profil, fotografia de coperta sau numele) logica necesara pentru a aduce baza de date intr-o stare consistenta este foarte complexa. Primul pas necesar in identificarea bazei de date potrivite pentru cerinta noastra este reprezentat de analiza relatiilor care exista intre modelul de date.

După etapa de analiza a relatiilor din modelul de date am constatat ca o baza de date orientata pe modelul de graf se potriveste foarte bine in acest context deoarece spre deosebire de o baza de date relationala complexitatea a scazut foarte mult, iar timpul de raspuns a crescut considerabil. Pentru a reprezenta relatiile de prietenie dintre utilizatori, caminul si scoala la care studiaza fiecare utilizator, Neo4J reprezinta cea mai buna solutie.

În figura, Fig. 5.16 sunt prezentate principalele relatiile dintre utilizatori, camine si universitati.

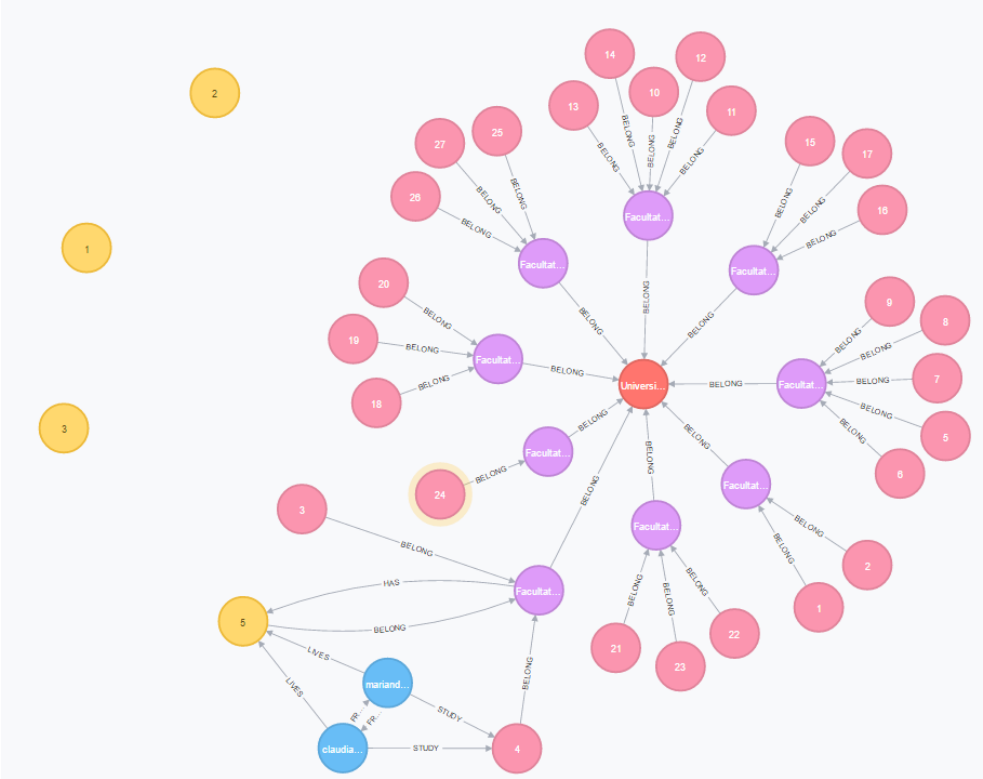

Fig. 5.16 Reprezentarea modelului de date în Neo4J

În figura de mai sus, Fig 5.16, nodul de culoare rosie reprezinta Universitatea Tehnica din Cluj Napoca, iar nodurile de culoare mov reprezinta facultatile existente din cadrul Universitatii Tehnice. Fiecare facultate este impartita in mai multe departamente, departamentele fiind reprezentate de nodurile de culoare roz. Pentru fiecare facultate se ofera un anumit numar de camine in unul sau mai multe campusuri univesitare.

Pentru a reprezenta caminele aflate in campusul Observator am folosit nodurile de culoare galbena, fiecare camin fiind identificat prin ID-ul unic asignat, iar nodurile de culoare albastra reprezinta utilizatorii care detin un cont in aplicatie.

Relatiile care se pot stabili intre nodurile descrise in paragraful de mai sus sunt urmatoarele :

- Intre facultate si universitate exista o relatie unidirectionala numita *BELONG*(o facultate apartine de o universitate)
- Intre un departament si o facultate exista o relatie unidirectionala numita *BELONG*(un departament apartine unei facultati)
- Intre caminele studentesti si o facultate exista o relatie unidirectionala *BELONG*(un camin studentesc apartine unei facultati)
- Intre un camin studentesc si un utilizator exista o relatie unidirectionala numita *LIVES*(un utilizator locuieste intr-un anumit camin)
- Intre un departament si un utilizator(student) exista o relatie unidirectionala numita *STUDY*(un utilizator este student la un anumit departament din cadrul unei facultati)

- Intre utilizatori exista o relatie bidirectionala numita *FRIEND*(utilizatorul A este prieten cu utilizatorul B si invers)

5.4. Diagrama de deployment

Diagrama de deployment prezinta componentele hardware pe care ruleaza componentele sistemului dezvoltat, dar si modul de interconectare al acestora. Componentele diagramei mai sunt numite si artefacte ale sistemului, scopul acestei diagrame fiind de a prezenta structura hardware necesara rularii sistemului.

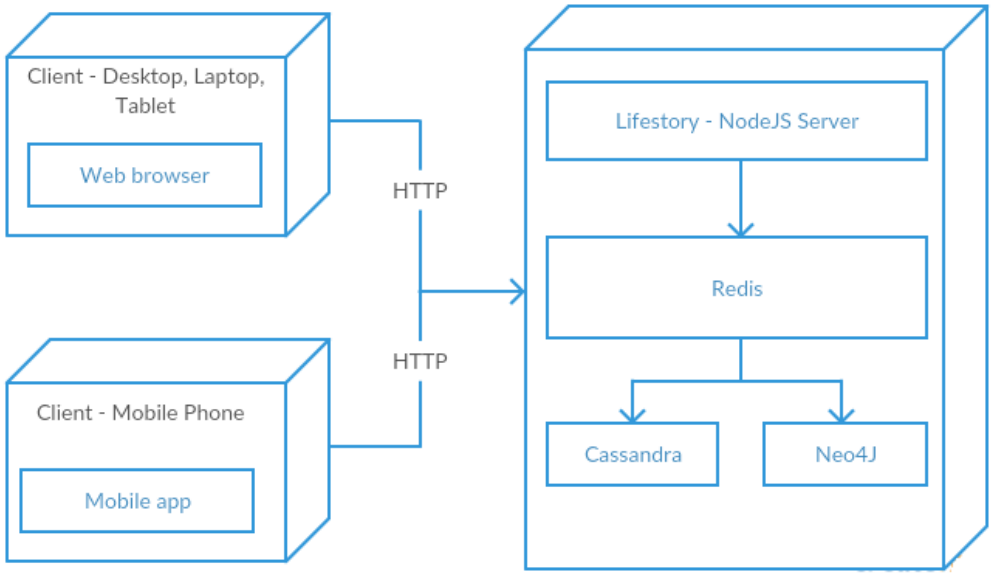

Fig. 5.17 Diagrama de deployment a sistemului

Aplicația Web poate fi accesata de pe orice dispozitiv(desktop sau mobile), insa dezvoltarea interfetei grafice a fost concentrata asupra sistemelor desktop.

Clientul comunica cu serverele aplicatiei prin intermediul browser-ului folosind protocolul HTTP. Dezvoltarile ulterioare aduse aplicatiei vor permite instalarea acesteia si pe dispozitivul mobil, protocolul de comunicare folosit si in acest caz fiind tot HTTP.

Componentele fizice ale sistemului sunt serverele pe care ruleaza aplicatia Web, serverele de caching in care sunt tinute datele pentru a asigura rapiditate si serverele de Cassandra si Neo4J, folosite pentru a asigura persistenta datelor. Cele doua componente client prezentate in diagrama reprezinta consumatorii aplicatiei. De pe orice dispozitiv desktop sa mobil, un utilizator care detine un cont valid poate sa acceseze aplicatia si sa navigheze cu succes. Aplicatia a fost dezvoltata pentru a putea rula in orice browser Web fara modificare ale interfetei utilizator.

Capitolul 6. Testare şi Validare

În acest capitol se dorește prezentarea principalelor procese de testare realizate asupra întregului sistem. Realizarea etapelor de testare nu garantează în general funcționalitatea completaă și corectă a sistemului în toate condițiile, însa poate ajuta la identificarea anumitor probleme și să ducă la găsirea de solutii și rezolvari. Testarea poate fi definită ca un proces de validare si verificare a faptului că sistemul corespunde cerințelor și constrângerilor impuse.

Procesul de dezvoltare a sistemului a fost iterativ, testarea fiecarei componente realizându-se la finalul implementării. În multe cazuri testarea s-a realizat manual, pornind de la scenarii simple de succes până la scenarii mult mai complicate. Pentru a verifica modul de raspuns al aplicației și starea în care trece acesta în caz de eroare s-au executat pentru fiecare componentă mai multe scenarii de test cu date eronate.

Indiferent de metodele de testare folosite, rezultatul testarii oferă dezvoltatorului un nivel de siguranță asupra componentelor dezvoltate, dar în acelasi timp rezultatele obținute îi oferă o idee privind rata de defectare a sistemului.

Testarea sistemului s-a realizat în trei etape, în fiecare etapa fiind testate anumite componente ale sistemului. Principalele metode de testare folosite sunt **Testarea manuală**, **Testarea unitară** și **Testarea performanței**.

6.1. Testarea manuală

Aplicația a fost testată în mod manual, testarea realizându-se pentru cele mai importante componente ale sistemului. Principalele scenarii de test au fost întocmite pe baza funcționalităților sistemului.

În continuare se vor prezenta doua seturii de scenarii de test care au fost utilizate in testarea manuală a aplicației.

TC1: Postarea unui eveniment pe pagina de profil

Descriere: Utilizatorul trebuie să aibă posibilitatea de a posta pe propria pagina de profil sau pe pagina de profil a altor utilizatori un eveniment complex. Evenimentul trebuie să conțină un status de cel puțin un caracter și opțional una sau mai multe imagini (dimensiunea maximă fiind de 10 MB) și un videoclip (dimensiunea maximă fiind de 25MB). Pentru a posta un check-in utilizatorul trebuie să adauge cel puțin o persoana și opțional locul în care se află.

Scenariu 1:

Pasul 1: Utilizatorul dorește să posteze un eveniment.

Pasul 2: Nu mai există conexiune la internet.

Pasul 3: Utilizatorul adaugă un status.

Pasul 4: Utilizatorul apasă *„Postează”*.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica ca nu exista o conexiune validă la internet.

Scenariu 2:

Pasul 1: Utilizatorul dorește să posteze un eveniment.

Pasul 2: Există o conexiune la internet.

Pasul 3: Acționează butonul *„Postează”*.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica completarea câmpului corespunzator statusului.

Scenariu 3:

Pasul 1: Utilizatorul dorește să posteze un eveniment.

Pasul 2: Acționează icon-ul corespunzător adăugării unei fotografii.

Pasul 3: Există o conexiune la internet.

Pasul 4: Acționează butonul *„Postează”*.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica ca nu se poate posta un eveniment fără un status sau fără cel puțin o fotografie.

Scenariu 4:

Pasul 1: Utilizatorul dorește să posteze un eveniment.

Pasul 2: Acționează icon-ul corespunzător adăugării unei fotografii.

Pasul 3: Există o conexiune la internet.

Pasul 4: Selectează o fotografie din calculatorul propriu de dimensiune mai mare decât dimensiunea maximă acceptată de sistem.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că dimensiunea fotografiei este mult mai mare decât dimensiunea maximă permisă.

Scenariu 5:

Pasul 1: Utilizatorul dorește să posteze un eveniment.

Pasul 2: Acționează icon-ul corespunzător adăugarii unui videoclip.

Pasul 3: Există o conexiune la internet.

Pasul 4: Selectează un videoclip din calculatorul propriu de dimensiune mai mare decât dimensiunea maximă acceptată de sistem.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că dimensiunea videoclipului selectat este mult mai mare decât dimensiunea maximă permisă.

TC2: Managementul fotografiei de copertă/profil

Descriere: Utilizatorul are posibilitatea de a-și schimba fotografia de profil sau coperta prin adaugarea unei noi fotografii sau prin alegerea unei fotografii existente. O altă acțiune pe care o poate executa utilizatorul este cea de repoziționare a unei fotografii deja existente, acțiune finalizată prin salvarea acesteia.

Scenariu 1:

Pasul 1: Utilizatorul dorește să își schimbe fotografia de copertă.

Pasul 2: Nu mai există o conexiune la internet.

Pasul 3: Utilizatorul acționează butonul din meniu *„Adaugă o fotografie”*.

Pasul 4: Utilizatorul selectează fotografia dorită din calculatorul personal.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că nu există o conexiune validă la internet.

Scenariu 2:

Pasul 1: Utilizatorul dorește să își schimbe fotografia de copertă.

Pasul 2: Există o conexiune la internet.

Pasul 3: Utilizatorul acționează butonul din meniu *„Adaugă o fotografie”*.

Pasul 4: Utilizatorul selectează fotografia dorită din calculatorul personal, dimensiunea fotografiei fiind mai mare decât dimensiunea maximă acceptată de sistem.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că dimensiunea fotografiei este mai mare decât dimensiunea maximă acceptată.

Scenariu 3:

Pasul 1: Utilizator dorește să își schimbe fotografia de copertă.

Pasul 2: Nu mai există conexiune la internet.

Pasul 3: Utilizatorul acționează butonul din meniu *„Alege din fotografiile existente”*.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că nu există o conexiune validă la internet.

Scenariu 4:

Pasul 1: Utilizatorul dorește să își schimbe fotografia de copertă.

Pasul 2: Nu mai există o conexiune la internet.

Pasul 3: Utilizatorul acționează butonul din meniu *„Repoziționează fotografia”*.

Rezultat așteptat: Afișarea unui mesaj de eroare pentru a indica că nu există o conexiune validă la internet.

Scenariu 5:

Pasul 1: Utilizator dorește să își schimbe fotografia de copertă.

Pasul 2: Există o conexiune la internet.

Pasul 3: Utilizatorul acționează butonul din meniu *„Repoziționează fotografia”*.

Pasul 4: Utilizatorul apasă tasta F5 sau butonul din browser corespunzatore acțiunii de reîncarcare a paginii.

Rezultat așteptat: Afișarea unui mesaj de informare asupra faptului că acțiune de reîncarcare a paginii vă conduce la pierderea fotografiei încarcate în pasul anterior.

6.2. Testarea unitară

Testarea reprezintă o etapă importanta în dezvoltarea aplicațiilor software deoarece ajută la îmbunătățirea calitătii. Există multe forme de testare a componenentelor sistemului, prima dintre ele fiind testarea unitară.

Pentru etapa de testare unitară a aplicației am folosit framework-ul Mocha JS, un framework simplu, flexibil și usor de folosit pentru testarea aplicatiilor construite în NodeJS. Testele create prin folosirea acestui framework pot fi scrise sub formă sincronă și sub formă asincronă.

Testele unitare sunt create pentru a testa componentele sistemului, iar în momentul în care o componentă este schimbată putem verifica dacă aceasta funcționează corect prin rularea testelor unitare. Spre deosebire de celelalte framework-uri folosite pentru testarea aplicațiilor NodeJS, ca Jasmine și QUnit, Mocha JS nu vine cu o variantă predefinită de librarie folosită pentru comparația rezultatelor. Cele mai populare librării folosite pentru comparația rezultatului așteptat și a rezultatului obținut în urma rularii testului sunt should.js, expect.js și chai. În cadrul acestui sistem am folosit Chai ca librarie folosită pentru compararea rezultatelor.

În continuare am realizat un test pentru a verifica pașii necesari autentificării în cadrul aplicației. În figura, Fig. 6.1 este prezentat un scenariu de test care acoperă componenta de autentificare a aplicației și mesajele de eroare care pot să apară în cazul în care utilizatorul introduce greșit numele de utilizator sau parola.

```
describe('LoginController', function () {
        it('initially has login inputs', function () {
      browser.get('http://localhost:9000/#/login);


      expect(element(by.id('username')).getText()).toEqual('username');
      expect(element(by.id('username')).getText()).toEqual('password');
        });
       it('clicking the sign în button changes the route of the app',
function () {
      browser.get('http://localhost:9000/#/login);
            element(by.css('[ng-model="username"]')).sendKeys('test');
            element(by.css('[ng-
            model="password"]')).sendKeys('testpassword');
            element(by.css('.btn-default')).click();
            expect(element(by.id('menu')).getText()).toEqual('Active
            widgets');
        });
       it('clicking the button does not login into app', function () {
```

Fig. 6.1 Testarea unitară a componentei de autentificare

6.3. Testarea performanței

Pentru a determina performanța aplicațiilor Web, browser-ul Google Chrome oferă o componentă de Profiling care face parte din utilitarul de dezvoltare al aplicațiilor Web. O parte din probleme care sunt greu de determinat în urma unui deployment al aplicației Web pe un server apar datorită produsului software, a sistemului de back-end sau datorită rețelei folosită pentru comunicare.

Prin folosirea profiler-ului oferite de Chrome am analizat anumite resurse folosite de aplicația Web. În figura, Fig. 6.2 este prezentată prima analiză facută cu scopul de a determina scurgerile de memorie din aplicație:

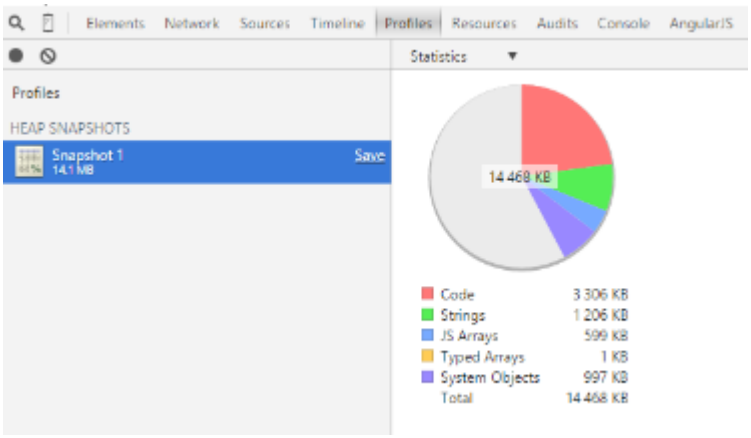

Fig. 6.2 Heap profiling

Cea de-a doua analiză a fost facută cu scopul de a demonstra timpul de încarcare a fisierelor statice JavaScript. Pentru a realiza acest lucru am folosit tool-ul de CPU Profiling, rezultatele obținute putând fi observate în figura următoare, Fig. 6.3:

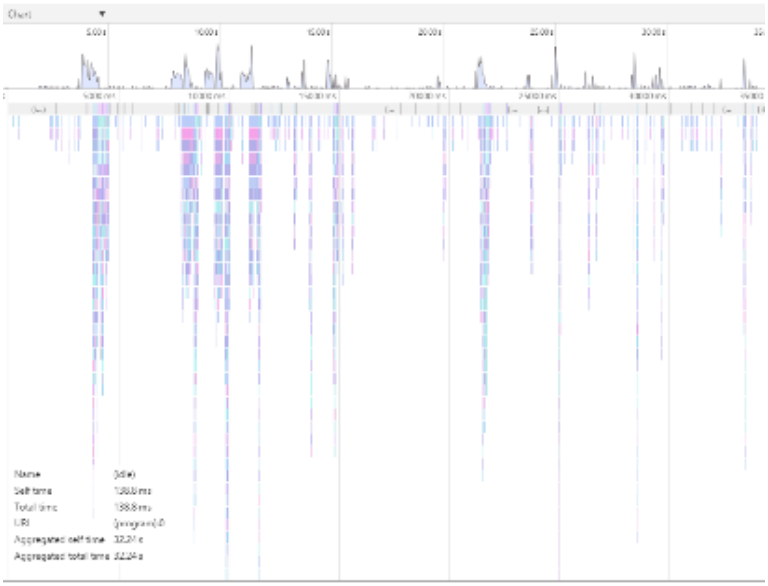

Fig. 6.3 CPU Profiling

Pentru a testa timpul de răspuns a serverelor Web la diferite tipuri de cereri ale utilizatorilor și numărul de utilizatori concurenți suportați am folosit tool-ul **Siege**. Folosind acest tool am reușit să comparăm timpul de răspuns obținut în momentul în care nu există un load balancer în fața serverelor Web cu timpul de răspuns în momentul în care serverele erau mapate la un load balancer care procesa fiecare cerere și o asigna unui server din cluster.

Capitolul 7. Manual de Instalare și Utilizare

În secţiunea de instalare trebuie să detaliaţi resursele software şi hardware necesare pentru instalarea şi rularea aplicaţiei, precum şi o descriere pas cu pas a procesului de instalare. Instalarea aplicaţiei trebuie să fie posibilă pe baza a ceea ce se scrie aici.

În acest capitol, trebuie să descrieţi cum se utilizează aplicaţia din punct de vedere al utilizatorului, fără a menţiona aspecte tehnice interne. Folosiţi capturi ale ecranului şi explicaţii pas cu pas ale interacţiunii. Folosind acest manual, o persoană ar trebui să poată utiliza produsul vostru.

Acest capitol prezinta pasii care trebuie urmati de utilizator pentru a putea realiza cu succes instalarea sistemului pe o masina locala. Tot in acest capitol vor fi descrise resursele hardware si software necesare, dar si un manual de utilizare al sistemului.

Utilizătorii aplicației nu trebuie să dispună de resurse hardware specifice, ci doar să ofere resurse browser-ului pe care îl folosesc în accesarea aplicației. Deployment-ul aplicației Web se face în rețeaua locală sau globală, dar pentru prezentarea aplicației am ales să fac deployment-ul în rețeaua locală pentru a putea testa funcționalitățile sistemului pe mai multe calculatoare interconectate în aceiați rețea.

Aplicația care rulează pe partea de server este dependentă de :

- **Framework-ul NodeJS**
- **Redis**
- **Cassandra**
- **Neo4J**

Cerințele minime ale sistemului pe care se face deployment trebuie sa fie următoarele:

- Serverul trebuie să aibă cel puțin 1024 MB memorie RAM
- Serverul trebuie să aibă spațiu pe disc de cel puțin 800 MB

7.1.1. Instalarea şi rularea aplicației

În acest subcapitol sunt descrișii pașii care trebuie urmați pentru a efectua instalarea cu succes a aplicației pe un server. Sistemul de operare care s-a folosit pentru crearea și deployment-ul aplicației este Microsoft Windows, acesta sistem de operare fiind ales datorită simplității pe care o oferă și datorită faptului că o persoană fără cunoștințe în domeniul ingineriei poate să înțeleagă toți pașii descrișii în acest scurt tutorial.

1. **Instalarea bazei de date Cassandra**

   Pentru instalarea cu succes a bazei de date trebuie să verificăm dacă platforma Java este instalată pe dispozitiv. Pentru a verifica dacă platforma Java este instalată pe dispozitivul pe care se dorește realizarea deployment-ului trebuie să deschidem *„Command Prompt-ul”* sistemului și să executăm urmatoarea comandă prezentată în figura următoare, Fig 7.1:

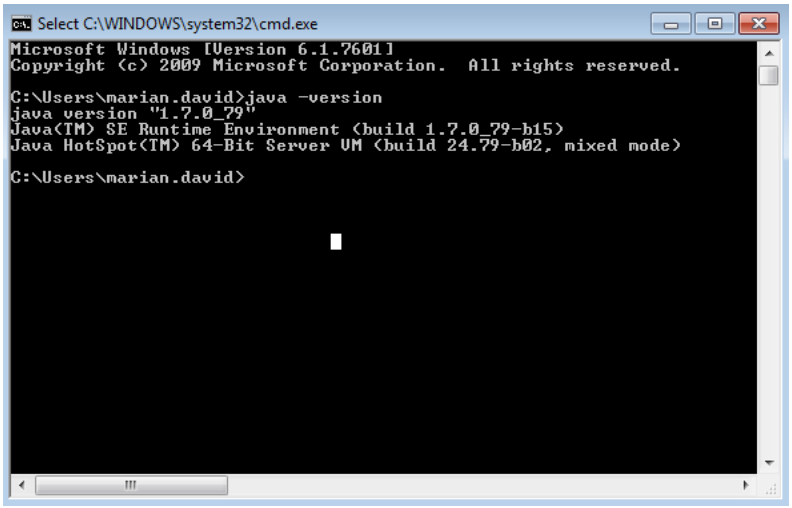

Fig. 7.1 Verificarea platformei Java instalate

În cazul în care nu există nicio versiune a platformei Java instalată, utilizatorul trebuie să-și descarce o versiune corespunzătoare sistemului de operare pe care îl folosește de la adresa http://www.oracle.com/technetwork . Dupa instalarea cu succes și configurarea *„variabilelor de mediu”* utilizatorul trebuie să-și descarce ultima versiune disponibilă a bazei de date de la adresa http://www.planetcassandra.org/cassandra/ .

Dupa ce s-a terminat instalarea bazei de date cu succes pentru a testa modul de funcționare al acesteia trebuie să pornim *„Cassandra CQL Shell”*, acest tool fiind instalat în momentul în care se instalează și baza de date. În figura următoare, Fig. 7.2 este prezentat tool-ul CQL.

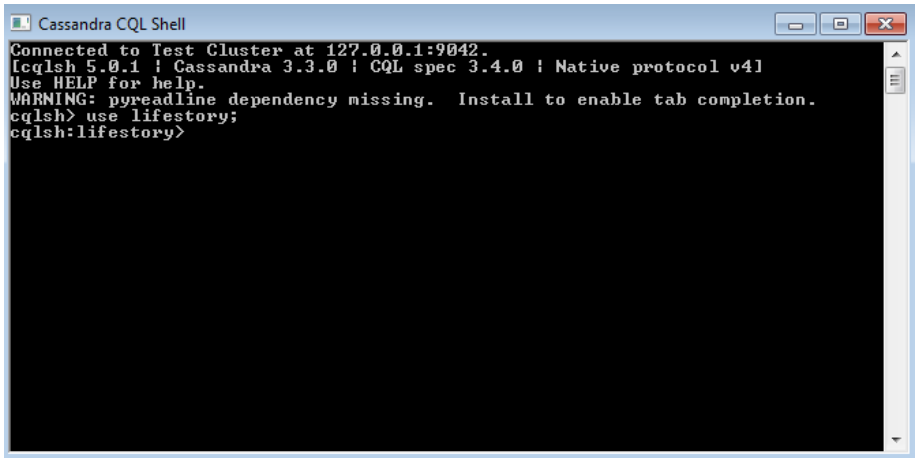

Fig. 7.2 Prezentarea tool-ului Cassandra CQL Shell

Dacă s-au urmat pașii explicați anterior, baza de date a fost configurată cu succes, putând fi folosită în continuare de aplicația server.

2. Instalarea bazei de date Neo4J

Pentru a instala baza de date Neo4J trebuie să o descarcam mai întai de la adresa http://neo4j.com/download/, alegând varianta Community Edition. După descărcarea executabilului și instalarea cu succes a acestuia, interfața grafică a bazei de date poate fi accesată la adresa http://localhost:7474. În figura urmatoare, Fig 7.3 este prezentată interfața grafica a bazei de date Neo4J.

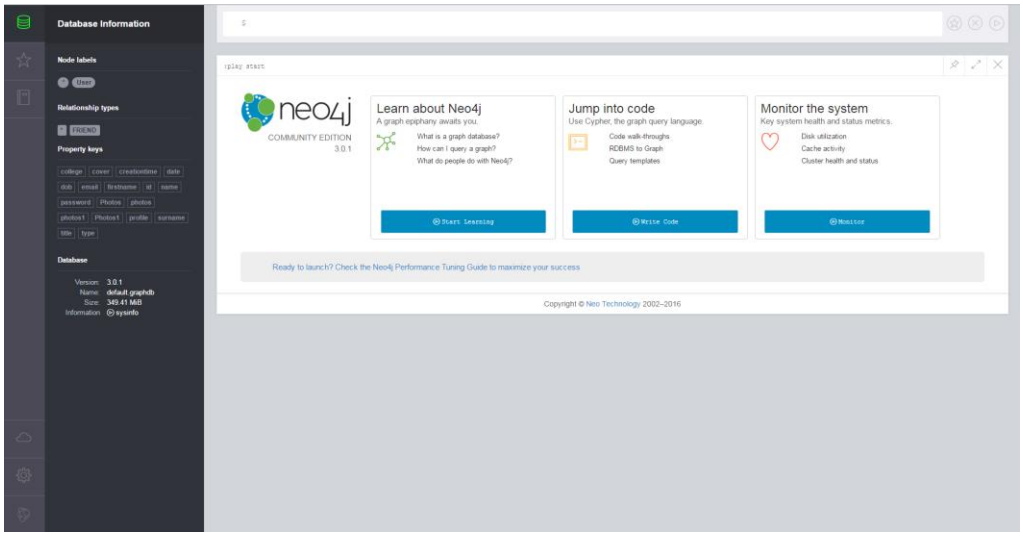

Fig. 7.3 Interfața grafică a bazei de date Neo4J

3. Instalarea mecanismului de caching - Redis

Pentru a instala Redis trebuie să-l descărcam mai întai de la adresa https://github.com/ServiceStack/redis-windows . După ce descărcarea s-a finalizat cu succes, trebuie să dezarhivăm fișierul descărcat pe disc. Următorul pas reprezintă descărcarea unui tool care facilitează folosirea mai ușoară a mecanismului folosit pentru caching. Tool-ul poate fi descărcat de la adresa http://redisdesktop.com/download .

După instalarea celor doua tool-uri navigăm pe disc la folder-ul dezarhivat anterior, iar pentru pornirea acestuia trebuie sa facem *dublu-click* pe executabilul *„redis-server.exe”.* În figura următoare, Fig. 7.3 este prezentată interfața bazei de date folosite pentru caching.

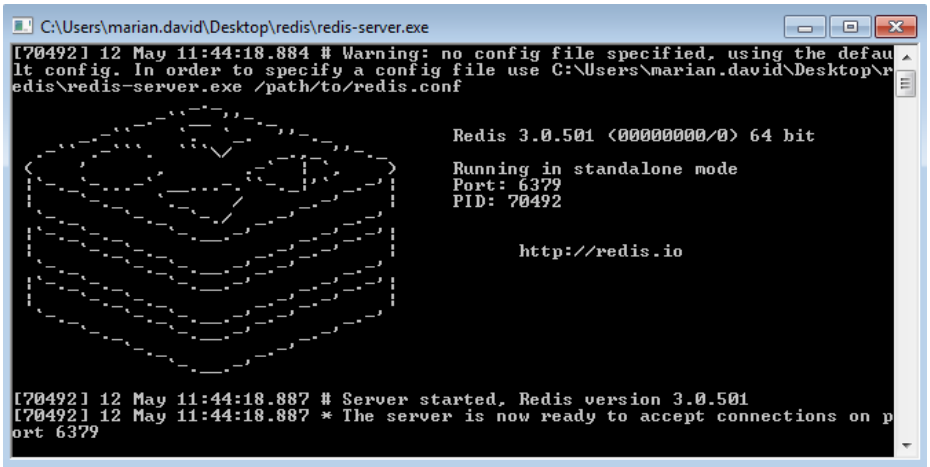

Fig. 7.4 Interfața bazei de date folosite pentru caching

Pentru a putea efectua operațiuni de inserare, ștergere și actualizare a datelor în cache trebuie să facem *dublu-click* pe executabilul *„redis-cli.exe”*, prezent în folder-ul dezarhivat anterior.

4. Instalarea framework-ului NodeJS

Pentru instalarea aplicației și pentru folosirea funcționalităților acesteia trebuie să instalam mai întâi framework-ul NodeJS. Putem instala framework-ul NodeJS de la adresa https://nodejs.org/en/download/, selectând sistemul de operare corespunzător și versiunea acestuia(32 biti sau 64 de biti). Alegerea unei alte versiuni decât cea corespunzatoare sistemului de operare va conduce la o funcționare necorespunzătoare a aplicației și la apariția mai multor erori datorate tool-urilor folosite.

Dupa instalarea framework-ului urmatorul pas ce trebuie efectuat este clonarea repository-ului de pe Git in spatiul local. Dupa executarea acestui pas, proiectul se afla pe discul dispozitivului. Pentru instalarea modulelor care asigură funcționarea sistemului, trebuie să navigam la folder-ul proiectului de pe disc și să deschidem o fereastra *„Command Prompt”*. În consolă vom folosi comanda *„npm install”*, comanda care instalează local toate modulele folosite de sistem.

La finalizarea instalării modulelor sistemul, acesta poate fi pornit prin executarea comenzii *„nodemon npm start”* în consola deschisă anterior.

În concluzie, dupa instalarea framework-ului folosit, a bazei de date folosite pentru caching și a bazei de date NoSQL care asigură persistența datelor, putem executa cu succes deployment-ul aplicației fără a apareă eventuale erori datorate tool-urilor folosite. Prin execuția comenzii „nodemon npm start” aplicația devine disponibilă utilizatorilor și poate fi accesată la adresa http://127.0.0.1.

7.1.2. Manual de utilizare

Dupa ce pașii prezentați în sub-capitolul anterior au fost îndepliniți cu succes, utilizatorul trebuie să verifice că bazele de date Cassandra si Neo4J sunt pornite și funcționează corespunzător. De asemenea este necesară și pornirea mecanismului de caching înainte de lansarea în execuție a aplicației.

Manualul de utilizare al aplicației cuprinde o parte din scenariile prezentate prin cazurile de utilizare descrise în capitolul 4.

Pentru ca autentificarea în sistem să se poată realiza cu succes, utilizatorii trebuie să introducă numărul de telefon sau adresa de email și o parolă validă, aceste date fiind introduse în sistem în momentul în care utilizatorii își crează un cont. Secțiune de autentificare și înregistrare este prezentată în figura următoare:

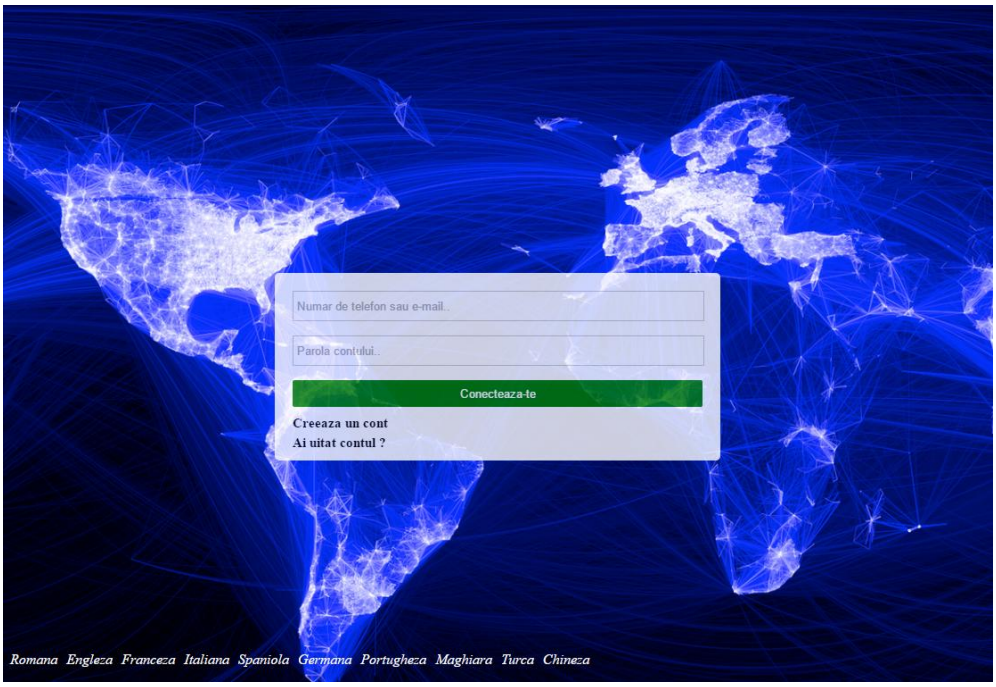

Fig 7.4 Pagina de autentificare in sistem

Pentru crearea unui cont în sistem utilizatorul trebuie să furnizeze anumite date personale, precum numele și prenumele, adresa de email sau numarul de telefon folosit în etapa de autentificare, adăugarea unei parole pentru contul utilizator și oferirea informațiilor referitoare la ziua, luna și anul nașterii. Utilizatorul are posibilitatea de a-și seta ca aceste detalii personale să poată fi vizualizate doar de un anumit grup de utilizatori, modul de configurare și de vizualizare a detaliilor personale fiind exemplificat în detaliu în pagina destinată setărilor de confidențialitate ale contului.

În figura următoare. Fig. 7.5 este prezentată pagina folosită de utilizatori pentru a-și crea un cont în aplicație:

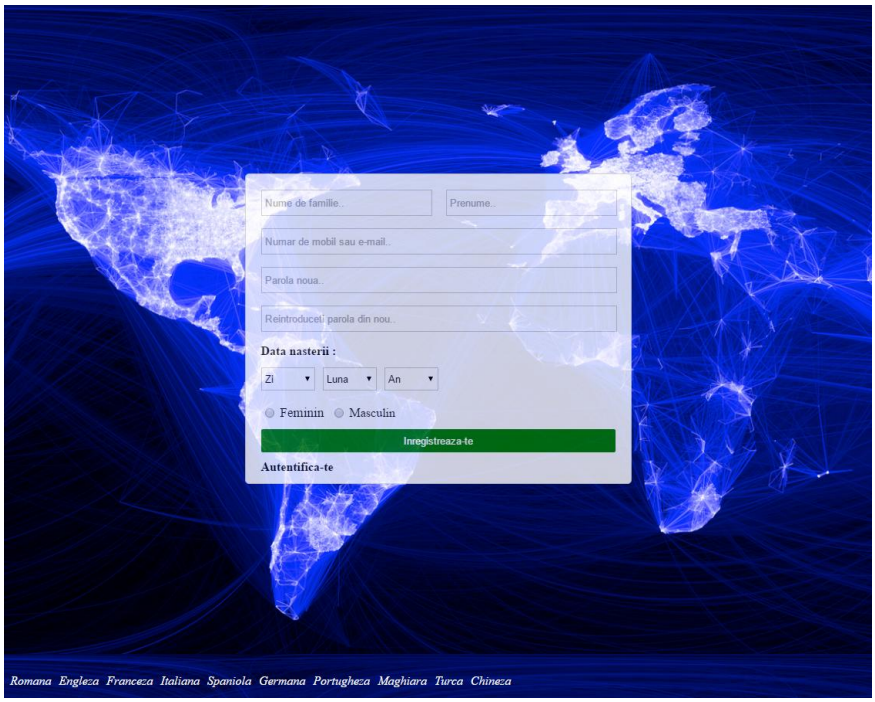

Fig 7.5 Pagina destinată creării unui cont utilizator

Dupa etapa de creare a unui cont, utilizator este redirecționat către o pagină în care sistemul îi cere acestuia să introducă un cod unic aleator primit pe adresa de email. Codul este folosit pentru a confirma contul creat, acesta fiind o masură de securitate folosită împotriva roboților care pot crea o mulțime de conturi în mod automat. Dupa etapa de confirmare a contului utilizator este redirecționat spre pagina de homepage.

In figura urmatoare. Fig 7.6 este prezentată prima pagină a rețelei de socializare:

Fig 7.6 Pagina de homepage

Pagina de homepage este împărțită în 3 părți, fiecare parte adresându-se într-un anumit mod utilizatorului. Porțiunea din stânga (meniul) cuprinde acțiunile pe care le poate executa utilizatorul pe pagina de homepage. Partea centrală, este compusă din conținutul dinamic adaugat de alți utilizatori, conținutul fiind livrat către utilizatori printr-un RealTime API. Partea din dreapta cuprinde lista cu persoanele pe care le-ai putea cunoaște și care se află în campusul tău, cea mai apreciată și vizualizată fotografie din campus și lista persoanelor care sunt autentificate în acel moment în aplicație.

Utilizatorul are posibilitatea de a-si personalize pagina de profil prin adaugarea unei fotografii de coperta, a unei fotografii de profil, a unei fotografii in centrul atentiei si a unei melodii favorite. In figura, Fig. 7.7 este prezentata pagina de profil si principalele componente configurabile:

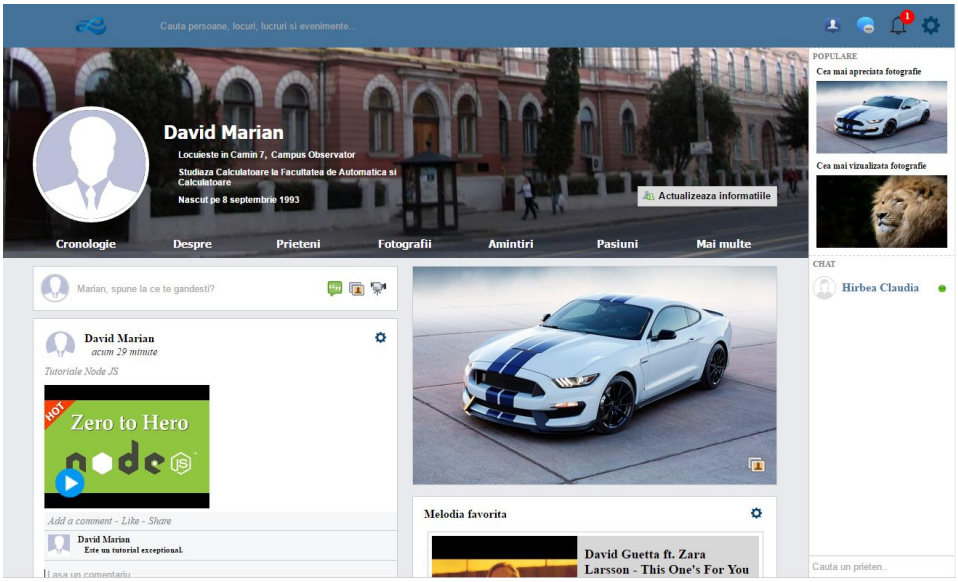

Fig 7.7 Pagina de profil

Capitolul 8. Concluzii

În acest capitol vor fi prezentate obiectele atinse de acest proiect, realizările acestuia, dar și descrierile dezvoltărilor ulterioare și îmbunătățirilor viitoare.

8.1. Realizarea obiectivelor propuse

Sistemul realizat reușește să-și atingă scopul, de a putea fi un sistem competitiv cu sistemele similare.

Obiectivele secundare propuse au fost realizate în totalitate, utilizatorii fiind clasificați în funcție de rolul pe care îl au în aplicație prin crearea de pagini pentru fiecare tip de utilizator. Dezvoltarea unui astfel de sistem a reprezentat o oportunitate de dezvoltare tehnică pentru dezvoltatorul sistemului deoarece s-au dobandit cunoștințe solide în domeniul aplicațiilor web și a „best-practice”-urilor folosite în implementarea acestora. Dezvoltatorul sistemul a înteles care sunt pașii necesarii într-un proces de dezvoltare iterativă și ce presupune fiecare pas, plus etapele de analiză și design ale sistemului care au reprezentat cea mai mare provocare pentru dezvoltator în momentul în care trebuie să ia decizii cu privire la modelul de date și la arhitectura bazei de date folosite. Proiectul creat a reușit să ajute la dobandirea de cunoștințe noi în ceea ce privește diferitele framework-uri și diferențele dintre acestea și cele mai bune moduri prin care pot fi folosite pentru a obține un sistem scalabil și performant.

Sistemul dezvoltat permite utilizatorilor să :

- Își creeze un cont personal, folosind o adresă de email validă.
- Personalizeze pagina de profil prin adăugarea unei fotografii de copertă, a unei fotografii de profil sau a unei fotografii în centrul atenției.
- Folosească funcția de mesagerie instantanee pentru a comunica cu persoanele din lista de prieteni.
- Adauge un eveniment complex atât de pe pagina de profil cât și de pe pagina de homepage.
- Creeze o amintire prin adăugarea unor fotografii, videoclipuri și persoane participante.
- Creeze un grup public sau privit și să adauge persoane pe baza de invitații.
- Adauge pe pagina de profil o melodie favorită.
- Adauge anumite melodii de pe homepage(postate de prieteni) în lista de melodii favorite.
- Adauge un comentariu sau o reacție la un eveniment de pe homepage.

Sistemul creat reușește să livreze un mediu care este ușor de înteles și de utilizat, fiecare acțiune nesolicitând mai mult de 3-4 secunde pentru a duce la îndeplinirea ei. Utilizabilitatea sistemului și lipsa nevoii de a exista un tutorial mai complex pentru a întelege funcționalitatea oferită de sistem este sustinută de design-ul creat și de modul de implementare al componentelor.

8.2. Dezvoltări ulterioare

Orice sistem informatic poate să beneficieze de dezvoltări ulterioare, prin adăugarea de noi funcționalitați sistemului sau prin îmbunătățirea funcționalităților existente. Sistemul dezvoltat nu reprezintă o excepție în acest sens, acesta permitând adăugarea de noi funcționalități într-o manieră foarte simplă.

Principalele dezvoltări ulterioare identificate sunt:

- Oferirea de aplicații pentru platformele mobile existente. În momentul de față mulți utilizatori accesează aplicațiile de pe telefoanele mobile, deci oferirea unei astfel de aplicații este necesară.
- Implementarea functionalității de live streaming pentru anumite grupuri de utilizatori. Fiecare utilizator care are dreptul de a folosi această funcționalitate poate să se înregistrare în acel moment și să distribuie videoclipul în aplicatie folosind funcția *„Watch Me”.*
- Dezvoltarea funcționalităților de pe pagina de profil pentru a permite adăugarea de fișiere în format .gif sau videoclipuri ca fotografie de copertă sau fotografie de profil.
- Dezvoltarea unei funcționalități pentru chat, astfel încat utilizatorul poate să selecteze dacă conversatia cu o altă persoană va fi înregistrată sau nu. Adăugarea unei funcționalitãăi pentru a permite convorbirile audio și video între utilizatori.
- Dezvoltarea unei funcționalități pentru etichetarea automată a fotografiilor încarcate de utilizatori. Fiecare fotografie încarcata de utilizatori a fost facută într-un anumit loc, iar pe baza acestei etichetări fotografia încarcată se va afișa pe homepage-ul altor utilizatori în momentul în care aceștia se află în acel loc sau trec prin apropierea lui.
- Dezvoltarea unui nou sistem folosit în aprecierea fotografiilor/ videoclipurilor încarcate sau a status-urilor distribuite pe profilul personal. Pentru a aprecia o fotografie nu este suficient un *„Like”*, fiecare utilizator având posibilitatea să-și exprime motivul pentru care apreciază respectiva fotografie : *„Îmi plac persoanele etichetate în fotografie”, „Îmi place descriere fotografiei”*, *„Îmi place mesajul transmis de fotografie”*, etc.
- Deployment-ul aplicației în Cloud. Principale servicii pentru deployment-ul în cloud care se poate alege sunt cele oferite de Google Cloud, deoarece pachetele de servicii sunt configurabile, iar suportul în caz de eșec este foarte bun.

Bibliografie

[1] Social Network, Disponibila la : https://en.wikipedia.org/wiki/Social_network

[2] Michael Oduor, *„Software architectures for social influence : Analysis of Facebook, Twitter, Yammer and FourSquare”,* Master's Thesis, Oulu 2013, Disponibila la : https://drive.google.com/drive/folders/0B04uRdrLhqZEaTVNR19jaV9kcDQ .

[3] **Ovidiu Pop**, *„Sisteme informatice”*

[4] **Jon Duckett**, *„HTML & CSS Design and build websites”*, Disponibila la : http://www.wufai.edu.tw/information_technology_center/datasheet/HTML%20and%20CSS%20design%20and%20build%20websites.pdf .

[5] **David Flanagan**, *„Javascript: The Definitive Guide, Sixth Edition*", JOHN WILEY & SONS, INC., 2011, Disponibila la : ftp://91.193.236.10/pub/docs/linux-support/programming/JavaScript/%5BO%60Reilly%5D%20-%20JavaScript.%20The%20Definitive%20Guide,%206th%20ed.%20-%20%5BFlanagan%5D.pdf .

[6] Hypertext Transfer Protocol, Disponibila la : https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol .

[7] **Todd Fredrich**, *„RESTful Service Best Practices”,* Pearson eCollege, 2012, Disponibila la : http://www.restapitutorial.com/media/RESTful_Best_Practices-v1_1.pdf

[8] **Mike Cantelon**, **Marc Harter**, **T.J. Holowaychuk** and **Nathan Rajlich**, *„Node.js IN ACTION ”,* Manning Publications Co., 2013, Disponibila la : http://sd.blackball.lv/library/Node.js_in_Action_(2014).pdf .

[9] **Sean Lang**, *„Web Development with Jade”*, Packt Publishing Ltd., 2014, Disponibila la : http://cdn.tsq.me/ebook/Web%20Development%20with%20Jade.pdf .

[10] **Clement Nedelcu**, *„Nginx HTTP Server”,* Packt Publishing Ltd., 2015, Disponibila la:https://drive.google.com/open?id=0B2Yd6fmyXjVnUmtER0NfWVhRVjg

[11] **Josiah L. Carlson**, *„Redis IN ACTION”,* Manning Publications Co., Disponibila la : http://htchttp.s3.amazonaws.com/books/Redis%20in%20Action.pdf .

[12] **Josef Finsel**, *„Using memcached”,* The Pragmatic Bookshelf, 2009 , Disponibila la : http://www.freeoa.net/attachments/2015/Using.memcached.2008.05.Josef.Finsel.%E6%96%87%E5%AD%97%E7%89%88.pdf

[13] BigPipe: Pipeling web pages for high performance, Disponibila la : https://www.facebook.com/notes/facebook-engineering/bigpipe-pipelining-web-pages-for-high-performance/389414033919/ .

[14] **Joel Bohman** and **Johan Hilding**, *„Cassandra”,* HTH Computer Science and Communication, Bachelor of Science Thesis, Stockholm, Suedia, 2010, Disponibila la : https://drive.google.com/drive/folders/0B04uRdrLhqZEWTF1MnFuZnc4SlU .

[15] **The CAP Theorem**, Disponibila la: https://teddyma.gitbooks.io/learncassandra/content/about/the_cap_theorem.html .

[16] **Aleksa Vukonic**, **Nicki Watt**, **Tareq Abedrabbo**, **Dominic Fox** and **Jonas Partner**, *„Neo4j IN ACTION”,* Manning Publications Co., 2015, Disponibila la : http://droppdf.com/v/DAt3d .

[17] **Cassandra CQL Tutorial**, https://cassandra.apache.org/doc/cql3/CQL.html .

[18] **Introduction to Cypher**, http://neo4j.com/developer/cypher-query- language/ .

Anexa 1 – Lista figurilor și a tabelelor din lucrare

1. Lista figurilor din lucrare

2. Lista tabelelor din lucrare

Anexa 2 – Glosar

API	Application Programming Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
NodeJS	Framework folosit pentru dezvoltarea aplicațiilor server, orientat pe o arhitectura scalabilă asincronă.
Cassandra	Baza de date distribuită creată pentru gestiunea cantităților mari de date distribuite pe mai multe noduri.
Redis	Structura de date ținută în memoria RAM, fiind folosită pentru caching.
BigPipe	Tehnica de afișare asincronă a interfeței utilizator.
Load balancer	Un server care acționeaza ca și proxy folosit pentru distribuirea traficului în mod aproximativ egal catre celelalte servere.
Proxy	Un server care acționează ca un intermediar între request-urile venite de la clienți.
Aplicație Web	O aplicație software orientată pe arhitectura client-server, in care clientul este reprezentat de browser-ul Web, iar serverul de aplicația care rulează pe server.
Browser Web	Aplicație software folosită pentru accesarea și prezentarea resurselor care circulă pe internet.
Protocol HTTP	Protocol request-response care stă la baza aplicaților distribuite.
Request	Acțiune inițiată de client pentru a obține anumite resuse aflate pe server respectând specificațiile protocolului HTTP.
Response	Acțiune inițiată de server pentru a-i oferi clientului datele cerute respectând specificațiile protocolului HTTP.
Server	Aplicație care rulează pe un server Web și furnizează servicii altor aplicații numite aplicatii client.
Sistem distribuit	Aplicație software care este executată pe mai multe calculatoare care comunică pentru îndeplinirea unui task.
Aplicatie de photo-sharing	Sistem care permite utilizatorilor să încarce fotografii și să le partajeze cu ceilalți utilizatori. Exemple de aplicatii de acest gen sunt Facebook și Instagram.
Performanța	Se exprimă în timp sau în memorie. În timp reprezintă rapiditatea cu care s-a executat o operație, iar în memorie se exprimă prin cantitatea de memorie utilizată în procesarea unui task.
Timp de raspuns	Timp total înregistrat pentru oferirea unui raspuns la o cerere venită de la client.