MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

EASYWAY – SISTEM UTILITAR GLOBAL PENTRU PLANIFICAREA TRASEELOR RUTIERE EFICIENTE

LUCRARE DE LICENŢĂ

Absolvent: **Alexandru CÎMPANU**

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

2017

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: **Alexandru CÎMPANU**

EasyWay – Sistem utilitar global pentru planificarea traseelor rutiere eficiente

1. **Enunţul temei:** Scopul proiectului de față este proiectarea și implementarea unui sistem informatic utilitar modern menit să ajute șoferii în alegerea prealabilă a unui traseu cât mai ușor și sigur între două locații, pe baza condițiilor meteorologice și a condițiilor de desfășurare a traficului, extrase din informațiile disponibile în Twitter. Scopuri secundare urmărite de lucrare sunt efectuarea unei analize a relevanței informației extrase din Twitter și implementarea unor elemente de optimizăre pentru a îmbunătăți calitatea totală a sistemului.

2. **Conţinutul lucrării:** Pagina de prezentare, Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu și implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bibliogarfie, Anexe.

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

4. **Consultanţi**:

5. **Data emiterii temei:** 1 noiembrie 2016

6. **Data predării:** 14 iulie 2017

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

1.1. Contextul proiectului

Într-o eră în care numărul de dispozitive conectate la Internet este semnificativ mai mare decât totalitatea populației, în care informațiile din orice domeniu au atins o cotă și o importanță impresionantă, atât din perspectiva densității lor, cât și a accesibilității acestora, într-o eră a "Big Data", se pune constant problema utilizării acestor informații la potențialul lor maxim pentru a îmbunătăți calitatea activităților umane și eficientizarea timpului petrecut pentru efectuarea acestora.

În același timp, aglomerația traficului rutier este una dintre cele mai mari probleme ale lumii moderne, problemă cu care se confruntă majoritatea așesărilor urbane, de la metropole la orașe de dimensiuni mici. Aceasta poate avea un impact major asupra vieții personale, carierei, siguranței, dar și a mediului înconjurător. Din acest motiv, problema traficului, respectiv a fluidității acestuia sau a condițiilor sale de desfășurare în general, nu mai este doar o preocupare a membrilor administrației locale, ci o chestiune de interes major și pentru oamenii de rând, care caută să petreaca o perioadă de timp minimă la volanul mașinii, dar și să călătorească în permanență în siguranță.

Deși în prezent se militeaza pentru implementarea unei multitudini de soluții de eficientizare a traficului pretutindeni, aceste strategii includ, în general, planuri de dezvoltare a infrastructurii: de la crearea de drumuri ocolitoare, la mijloace de control al circulației adaptive și inteligente, sau oferirea unor servicii de car sharing sau bike sharing menite să încurajeze locuitorii să folosească mai puțin mașina. Totuși, aceste soluții nu sunt tocmai ușor de implementat, ceea ce preconizeaza un efect al acestora întarziat, traficul aglomerat fiind în continuare o problemă majoră.

Astfel, soluțiile software de avertizare asupra zonelor de congestionare a traficului și ale condițiilor de derulare, în general, ale acestuia, reprezintă o metodă de o importanță capitală în diminuarea impactului negativ al fenomenului și își propun atât să amelioreze într-o anumită măsură aglomerația, cât și să înștiințeze despre drumurile, zonele sau locațiile foarte aglomerate pentru ca utilizatorii acestora să își poată stabili o altă rută mai sigură și mai ușoară.

Din acest motiv, companii precum Waze, Google, HERE și altele, dar și cercetători din lumea academică, au devenit tot mai interesați de propunerea și dezvoltarea de astfel de sisteme software, iar caracterul deschis al informației din era "Big Data" poate, de asemenea, reprezenta un pas important în dezvoltarea de noi sisteme sau eficientizarea celor existente.

1.2. Motivația

După cum a fost menționat și în încheierea secțiunii anterioare, ne bucurăm în momentul de față de un status quo în care informația capătă un caracter din ce în ce mai deschis. Miliardele de dispozitive, mobile și nu numai, conectate la Internet, fac ca un volum impresionant de date să fie împărtășite de comunitate.

Deși există posibilitatea ca în atât de multă informație să fie uneori greu de diferențiat conținutul relevant de cel irelevant, se poate ca datele postate direct de către utilizatori la un anumit moment din timp să aibă un caracter informativ mai important

chiar și decât ale canalelor oficiale, și să poată face parte într-un mod eficient din cadrul unui sistem de informare asupra condițiilor de trafic.

Mai mult, pe lângă argumentul tehnologic și contribuția la dezvoltarea unui noi sistem software care să ajute șoferii să cunoască cât mai multe detalii despre rutele posibile înaintea luării unei decizii, motivația este și de factură personală, și anume materializarea interesului pentru sistemele distribuite, baze de date, tehnologii web și arhitecturi cloud dobândite în cadrul anilor de facultate și utilizarea cunoștințelor din aceste arii, dar și dezvoltarea de noi aptitudini vitale pentru a fi la curent cu schimbările tehnologice, în proiectarea și dezvoltarea unui sistem informatic funcțional implementat cu standarde de calitate riguroase.

1.3. Conținutul lucrării

În primul capitol, **"Introducere"**, tema lucrării de față este contextualizată și alegerea acesteia motivată, subliniindu-se aspecte ce țin de starea de fapt, probleme identificate, dar și soluții, atât actuale, cât și viitoare. De asemenea, se menționează avansul tehnologic al "Big Data" și se motivează studiul datelor cu caracter deschis în contextul problemei aglomerației traficului.

În cel de-al doilea capitol, **"Obiectivele Proiectului"**, sunt enumerate și explicate obiectivele principale și secundare urmărite de lucrarea de față pentru a aduce o schimbare pozitivă a situației prezentate în primul capitol. Mai mult, tot în această secțiune, obiectivul principal va fi segmentat în obiective specifice pentru urmărirea unor pași exacți în îndeplinirea acestuia.

Următoarea secțiune, **"Studiu bibliografic"**, constă în sintetizarea informațiilor studiate din diverse lucrări științifice care descriu concepte relevante pentru lucrarea de față. Sunt descrise abordări de geolocalizare ale tweet-urilor, metode prin care această rețea de socializare a ajutat la detectarea evenimentelor, sunt evidențiate și caracteristici ale lucrărilor care s-au preocupat cu management-ul traficului cu ajutorul informațiilor obținute din Twitter și se prezintă și structura generala a acestor informații. De asemenea, în finalul capitolului, se compară o serie de trăsături ale aplicațiilor comerciale relativ similare cu caracteristicile aplicației de față.

În capitolul patru, **"Analiză și fundamentare teoretică"**, sunt prezentate informații despre tehnologiile, serviciile cloud, tool-urile, algoritmii, framework-urile și API-urile folosite în dezvoltarea proiectului și motivarea folosirii acestora în contextul dat, subliniind punctele forte și limitările acestora în comparație cu alternativele existente. De asemenea, se realizează o privire conceptuală asupra arhitecturii generale a sistemului, vor fi definite cerințele funcționale și non-funcționale ale acestuia ce vor fi complementate de detalierea unor cazuri de utilizare.

În cel de-al cincilea capitol, **"Proiectare de detaliu și implementare"**, sunt descrise etapele de proiectare ale sistemului, este detaliată arhitectura și componentele principale ale acesteia, sunt explicați algoritmii și fluxul de date. De asemenea, se explică modalitatea de determinare a rutelor, de identificare ale punctelor cele mai importante de pe acestea, de unde se vor analiza, ulterior, informații relevante legate de condițiile meteo și de trafic, modulul de adaptare a căutării pe regiunile geografice pe care rutele le parcurg, sistemul de filtrare a rezultatelor irelevante bazat pe comunitate, dar și alte detalii de implementare.

Apoi, în capitolul șase, **“Testare și validare”**, prezintă mijloacele prin care aplicația a fost testată și rezultatele validate, menționând atât metode de testare software clasice, cât și rezultatele unui studiu de relevanță a rezultatelor efectuat pe durata a trei săptămâni. Mai mult, se prezintă simularea unui scenariu real prin ilustrarea comportamentului sistemului.

Capitolul al șaptelea, **“Manual de instalare și utilizare”**, surprinde detalii de deployment, instalare și rulare pentru a asigura, prin intermediul unui set de pași bine definiți, buna funcționare a proiectului implementat.

Ultimul capitol, **“Concluzii”**, prezintă o serie de observații desprinse din studiul și implementarea efectuate, identifică în ce măsură au fost obiectivele inițiale atinse, specifică anumite limitări ale proiectului și propune o listă de posibile dezvoltări ulterioare.

Capitolul 2. Obiectivele Proiectului

În cadrul acestui capitol va fi descris, mai întâi, obiectivul principal al proiectului implementat și obiectivele specifice în care acesta constă, și a căror implementare va duce la îndeplinirea acestuia. Apoi, vor fi descrise o serie de obiective secundare urmărite, care vor avea o contribuție semnificativă în ceea ce privește calitatea proiectului de față.

2.1. Obiectivul principal

Obiectivul principal al lucrării de față este proiectarea și implementarea unui sistem informatic sub forma unei aplicații web utilitare moderne menite să ajute șoferii vehiculelor în alegerea prealabilă a unui traseu cât mai ușor și sigur între două locații pe baza unor factori specifici: condițiile meteorologice și condițiile de desfășurare a traficului, extrase din informațiile disponibile pe una dintre cele mai populare rețele de socializare, Twitter.

2.1.1. Obiective specifice

Premergător atingerii obiectuvului principal, o serie de obiective specifice trebuie atinse în prealabil, într-o ordine specifică:

1. Geolocalizarea cu precizie sporită a locațiilor pentru care se vor determina rutele
2. Determinarea traseelor posibile între cele două locații geolocalizate și a punctelor intermediare prin care acestea trec
3. Marcarea vizuală ale acestora pe hartă într-un mod sugestiv și ușor de intuit de utilizator
4. Determinarea punctelor principale de interes de pe rutele identificate (localități importante)
5. Determinarea condițiilor meteorologice pentru punctele principale de interes identificate pe traseu și afișarea acestora într-un mod sugestiv pentru utilizatorul aplicației
6. Identificarea, filtrarea și procesarea tweet-urilor relevante despre condițiile de desfășurare a traficului și afișarea acestora într-o manieră sugestivă utilizatorului
7. Implementarea unui sistem de raportare a rezultatelor irelevante, bazat pe comunitatea utilizatorilor aplicației
8. Dezvoltarea unui sistem de supervizare a rezultatelor marcate ca fiind irelevante și a unui sistem sigur de autentificare și autorizare a utilizatorilor cu drepturi de administrare

2.1.2. Obiective generale

În implementarea obiectivelor specifice care vor duce la îndeplinirea obiectivului principal menționat anterior, se va urmări permanent alegerea abordărilor de implementare care vor spori, în ansamblu, calitatea proiectului.

Dintre aspectele considerate permanent în cadrul deciziilor de implementare, se vor evidenția măsura în care este afectată **scalabilitatea** arhitecturii per ansamblu, contribuția adusă la **utilizabilitatea** proiectului și modul în care este afectată interacțiunea cu utilizatorul, posibilitățile de **securizare** ale diferitelor componente care ar trebui restrictionate utilizatorului fără privilegii, impactul pe care decizia l-ar putea avea asupra **disponibilității** sistemului implementat și modul în care decizia contribuie la păstrarea și consolidarea unui **sistem deschis.**

2.2. Obiective secundare

2.2.1. Analiza relevanței informației extrase din Twitter

Primul obiectiv secundar în implementarea proiectului propus este efectuarea unei analize de date pentru a identifica relevanța informațiilor extrase din tweet-uri în contextul evenimentelor care pot afecta condițiile de trafic de pe anumite rute.

Acest obiectiv va putea fi realizat, în primul rând, prin analiza calității rezultatelor de căutare pentru diverse locații și, în al doilea rând, prin efectuarea unui studiu de caz pe o perioadă determinată, în care se vor studia rezultatele obținute pentru anumite locații de referință. Studiul va marca, zilnic, numărul total de rezultate obținute și ponderea datelor relevante.

2.2.2. Integrarea în cadrul unei platforme cloud

Un alt obiectiv secundar important pentru lucrarea de față este studierea diferitelor tipuri de arhitecturi cloud și migrarea în totalitate a implementării proiectului pe o platformă de acest tip.

Astfel, aplicația web dezvoltată, toate serviciile și logica de server folosită, dar și baza de date, vor utiliza stocare remote, oferită prin servicii de tip Platform as a Service (PaaS), mașini virtuale de tip Infrastructure as a Service (IaaS) și servicii cloud de tipul Software as a Service (SaaS).

Realizarea acestui obiectiv secundar va oferi proiectului numeroase avantaje non-funcționale, cum ar fi ușurința de scalare verticală și orizontală (adăugarea de noi resurse sistemului sau chiar de noi sisteme) sau creșterea disponibilității, întrucât acestea vor fi funcționale în permanență.

2.2.3. Adaptarea căutării la contextul lingvistic al punctelor de interes

Pentru ca aplicația dezvoltată să nu fie funcțională doar în anumite țări și doar pentru un număr limitat de limbi, se propune ca un alt obiectiv secundar implementarea unui sistem de internaționalizare a căutărilor, care să permită obținerea rezultatelor relevante indiferent de țara sau țările pe care traseele căutate le parcurg.

Mai mult, se dorește ca această adaptare la contextul lingvistic să fie complet automată, în așa fel încât sistemul să poată identifica individual limba țării din punctul de interes curent și a efectua o traducere a termenilor cheie, pentru a putea, ulterior, rafina și optimiza căutarea.

2.2.4. Crearea unei interfețe grafice responsive

Tendința ultimilor ani, vizibilă și în figura 2.1., ne arată creșterea semnificativă a popularității navigării pe Internet prin intermediul dispozitivelor mobile, în detrimentul desktop computerelor.

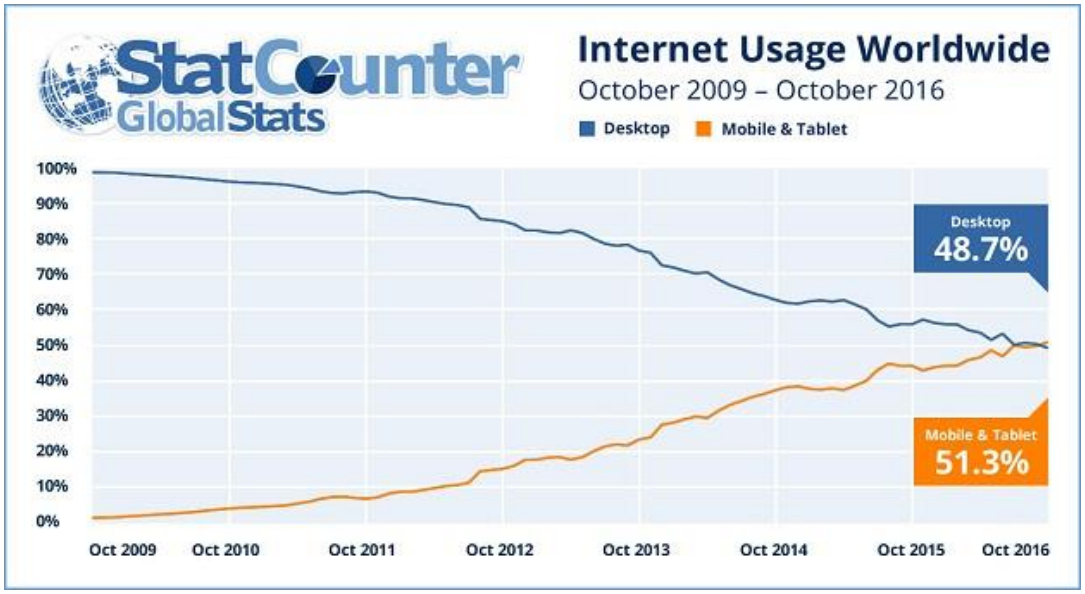

Figura 2.1 Statistică Internet Usage Mobile vs Desktop (statcounter.com)

Din această cauză, un alt obiectiv secundar important este optimizarea stilizării aplicației web pentru ecranele de dimensiuni mici (telefoane și tablete). Acest obiectiv este important atât din perspectiva utilizabilității pentru utilizatorii care doresc să acceseze sistemul de pe dispozitivele mobile, dar poate fi un prim pas decisiv în cadrul unor acțiuni de dezvoltare ulterioară care includ proiectarea de aplicații mobile native sau hibride.

Capitolul 3. Studiu Bibliografic

Capitolul curent are ca scop principal prezentarea stadiului actual al domeniului și a subdomeniilor în care este plasată tema lucrării de față.

Astfel, într-o primă fază se vor identifica aspecte importante ale lucrărilor științifice care s-au preocupat cu studiul utilizării rețelelor sociale ca sursă de informație în detecția și predicția evenimentelor și ca ansamblu de senzori sociali în cadrul sistemelor de management al traficului.

Apoi, se vor prezenta generalități ale arhitecturilor cloud, dar și tipologii și caracteristici ale acestora, iar, în finalul capitolului, se va prezenta și o privire de ansamblu asupra sistemelor similare comerciale existente, realizându-se o comparație punctuala a acestora cu proiectul implementat.

3.1. Twitter ca sursă de informație

Twitter este un serviciu online creat în martie 2006 pentru știri și socializare (rețea socială) unde utilizatorii postează și interacționează prin mesaje denumite “tweet”-uri, de o dimensiune redusă (maximum 140 caractere). Utilizatorii pot accesa platforma prin intermediul interfeței web sau a aplicațiilor mobile.

În anul 2012, mai mult de 100 de milioane de utilizatori activi postau 340 milioane de “tweet”-uri pe zi[1]. Conform datelor de pe site-ul oficial al companiei[2], la data de 30 iunie 2016, platforma găzduia 313 milioane de utilizatori activi pe lună și era accesată, prin intermediul site-ului, aplcațiilor și a site-urilor care îi afișau conținutul, de aproximativ 1 miliard de utilizatori pe lună.

Caracterul relativ deschis al rețelei de socializare Twitter a încurajat realizarea unei multidudini de studii care au folosit platforma ca sursă de date. Pe lângă numărul relativ mare de domenii academice în care a fost utilizată această platformă, există și numeroase aplicații comerciale care se bazează pe datele analizate în timp real din conținutul Twitter.

Mai mult, Twitter a devenit o sursă promițătoare de informație și datorită caracteristicilor postărilor. Platforma nu este folosită doar ca un canal de știri, ci este utilizată, mai ales, de utilizatori simpli, care își împărtășesc opiniile și experiențele de zi cu zi.

În ceea ce privește împărțirea utilizatorilor pe țări, figura 3.1. ilustrează popularitatea platformei în Statele Unite, dar prezintă un număr impresionant de utilizatori activi dintr-o multitudine de țări.

Datorită acestor factori, Twitter poate fi considerat un ansamblu foarte important de senzori sociali ce poate fi consultat și analizat într-o multitudine de domenii, pentru care poate îmbunătăți semnificativ calitatea și acuratețea informației.

Acest subcapitol are ca scop prezentarea abordărilor din lucrările de cercetare și a aplicațiilor comerciale care au folosit datele oferite de Twitter.

[1] Twitter turns six [online] - https://blog.twitter.com/official/en_us/a/2012/twitter-turns-six.html
[2] About Twitter [online] - https://about.twitter.com/company

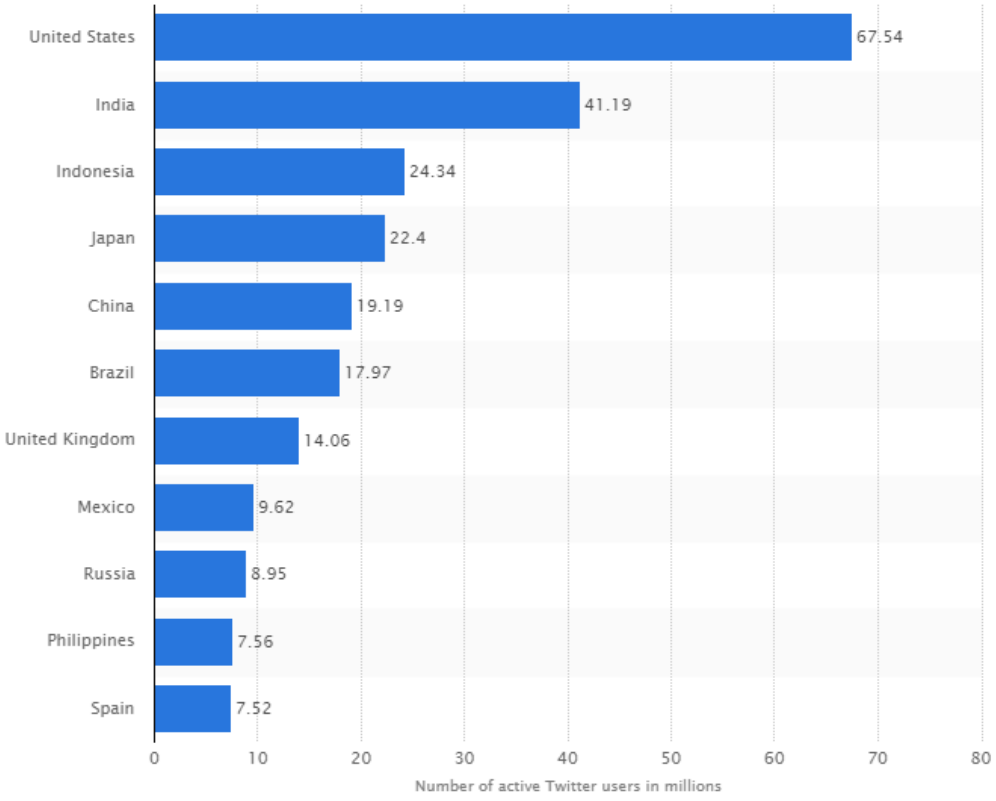

Figura 3.1 Numărul utilizatorilor activi per țară în mai 2016

3.1.1. Geolocalizarea tweet-urilor

În trecut, o provocare majoră în folosirea datelor Twitter ca sursă geografică de informație a fost georefer**e**țierea lor. După lansarea platformei de geolocalizare, conform subcapitolului "Spatial Extraction" al capitolului 3.2 din lucrarea [1], doar 1% din totalitatea tweet-urilor conțineau informații despre geolocalizarea lor.

Drept urmare, multe studii s-au preocupat cu provocarea de a geolocaliza tweet-urile. În cadrul acestora, diferiți indicatori au fost folosiți pentru a geolocaliza postări sau utilizatori. De cele mai multe ori, textul tweet-urilor a fost folosit prin aplicarea tehnicilor de procesare a limbajului natural (EN: Natural Language Processing – NLP) pe termenii din textul mesajului. O alternativă utilizată în alte studii este compararea termenilor (referiți ca tokens) din tweet-uri cu o bază de date de locații geografice, având ca avantaj simplitatea metodei [1].

Totuși, de la implementarea API-ului de geolocalizare, s-a înregistrat o creștere în ponderea tweet-urilor georeferențiate, fapt ce simplifică procesul de izolare a unei postări într-o zonă geografică specifică (o rază de câțiva km).

În scopul acestei lucrări, se va considera că tweet-urile au fost specific geolocalizate pentru a fi obiectul de studiu al condițiilor/incidentelor care ar putea influența buna desfășurare a traficului.

3.1.2. Detectarea evenimentelor folosind Twitter

Unul dintre studiile cele mai recunoscute din domeniul detecției evenimentelor în timp real utilizând date preluate din API-urile Twitter, dar și una dintre primele cercetări care a speculat potențialul informației cu caracter deschis din rețelele de socializare, este studiul realizat de Sakaki et al. [4].

Lucrarea menționată a reușit cu succes să detecteze cu o probabilitate impresionantă cutremurele din Japonia - 96% dintre cutremurele cu o intensitate seismică mai mare de 3 anunțate de Agenția Meteorologică Japoneză – și să atenționeze prin intermediul unui e-mail utilizatorii înregistrați din zonele afectate mult mai rapid decât agenția anterior menționată.

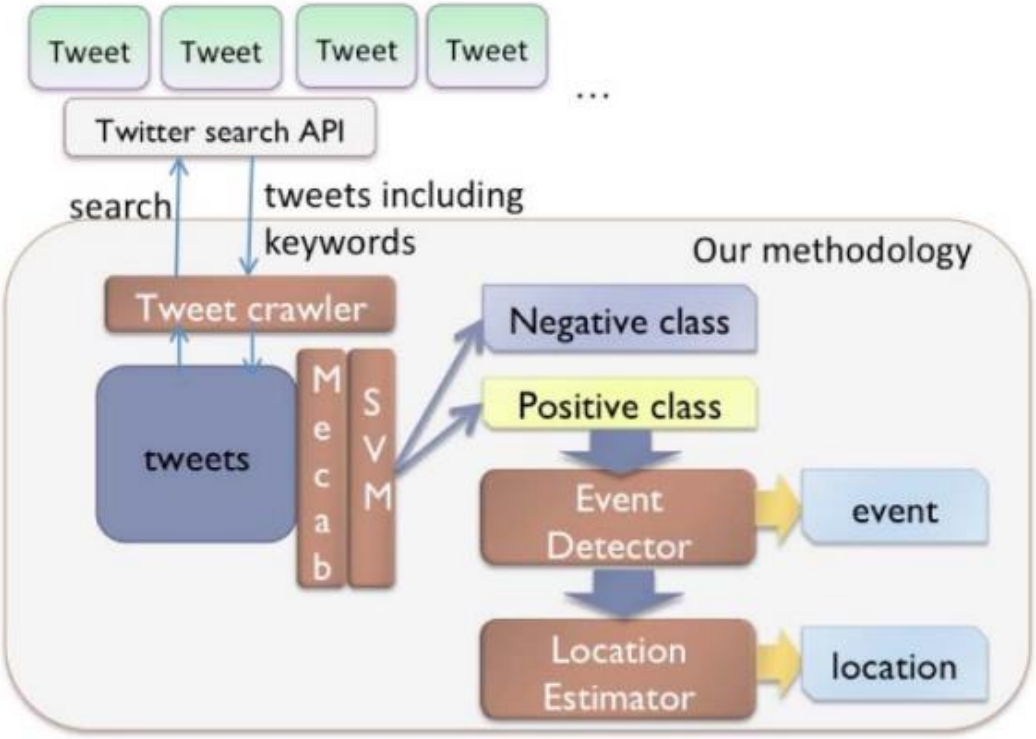

Figura 3.2 Detecția evenimentelor (Sakaki et. al., 2013)

Abordarea, descrisă în capitolul 3 al [4] și vizibilă în figura 3.2, include o analiză semantică a tweet-urilor, bazată pe identificarea anumitor cuvinte cheie (considerate, ulterior, cuvinte de interogare) și utilizează Support Vector Machine ca algoritm de machine learning pentru clasificarea rezultatelor în categorii pozitive și negative. În capitolul 4 al [4] este descrisă estimarea ulterioară a centrului evenimentului și calcularea traiectoriei probabile a acestuia bazată pe algoritmul de filtrare Kalman.

Totuși, modelul propus include restricția că o singură instanță a unui eveniment există într-un anumit timp (un singur cutremur există la un moment de timp), fapt care nu poate fi adevărat în cazul evenimentelor din trafic.

Un sistem comercial care se axează pe detecția incidentelor bazată pe medii sociale online este PublicSonar[3]. Dezvoltat, inițial, sub denumirea de "Twitcident",

[3] https://publicsonar.com/about-publicsonar/

sistemul implementat descris în articolele [5] și [6] reușește să filtreze în timp real informații din stream-uri sociale, relevante evenimentelor din lume. Utilizatorii twitcident puteau analiza un eveniment specific din prisma trend-urilor postărilor din rețelele de socializare.

În capitolul 2.1 din [6] este descris pipeline-ul datelor prin framework-ul Twitcident. Când acesta înregistrează un eveniment, sistemul pornește un nou thread pentru agregarea mesajelor din rețelele sociale. Mesajele colectate sunt ulterior procesate și filtrate utilizând un modul de semantic enrichment bazat pe Named Entity Recognition și clasificarea mesajelor.

3.1.3. Twitter ca sursă de informație în managementul traficului

Unele lucrări de cercetare discută potențialul utilizării Twitter ca sursă de informație în timp real pentru managementul traficului. Blocajele în trafic pot avea cauze multiple, cum ar fi semafoare defecte, șantiere în lucru, accidente, evenimente locale (concerte sau festivități) ș.a..

Unele dintre acestea sunt de factură previzibilă (informația poate fi sustrasă în prealabil de la surse oficiale, cum ar fi agențiile de știri), însă altele sunt imprevizibile, iar de multe ori informația transmisă de comunitate se poate propaga mai repede decât informația transmisă pe canale oficiale.

Figura 3.3 ilustrează exemple de tweet-uri găsite în urma unei căutări geolocalizate efectuate după termenii de căutare “accident” și “traffic” în Londra.

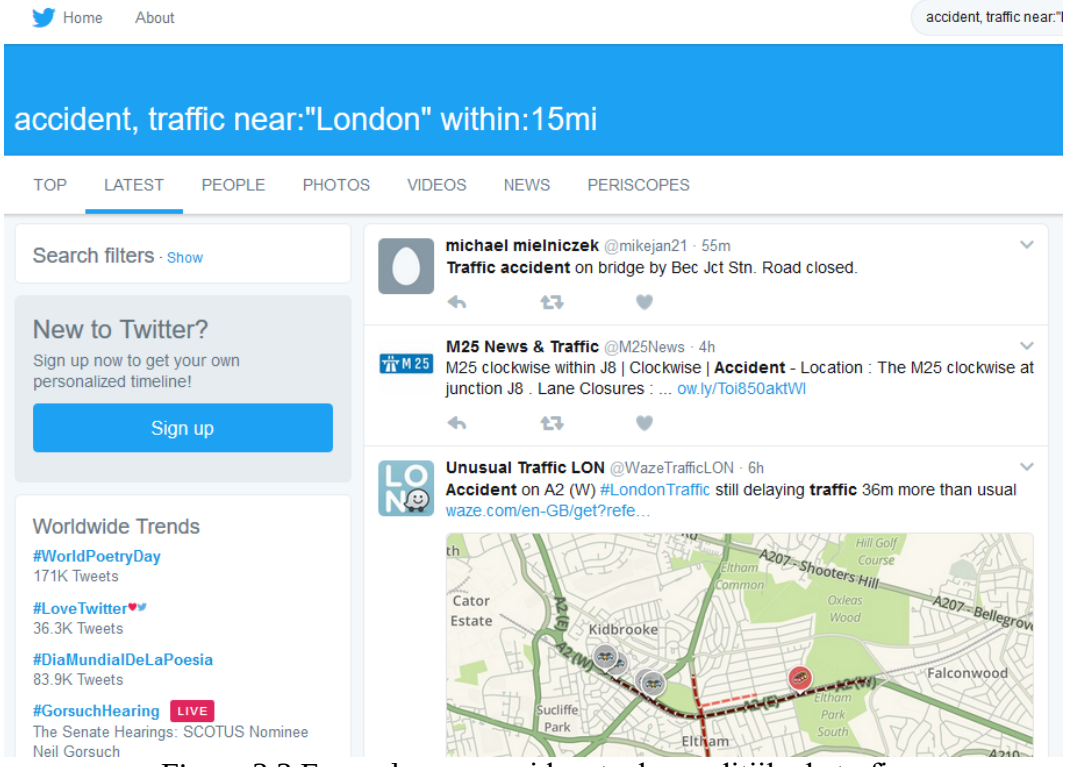

Figura 3.3 Exemple tweet-uri legate de condițiile de trafic

Lucrarea [3] reușește să propună un sistem de management al traficului particularizat pentru Olanda. Propunerea este bazată pe analiza în timp real a informațiilor obțiunte cu ajutorul Twitter Streaming API. Aria geografică pe care

sistemul funcționează fiind una redusă (nivel de țară), sistemul dorește geolocalizarea incidentelor în trafic cu acuratețe și maparea acestora la nivel de stradă. Un aspect mai puțin plăcut al acesteia este faptul că, în capitolul 4.1 al [3] se impun anumite condiții destul de restrictive despre structura tweet-ului pentru ca acestea să fie considerate utile: acestea trebuie să conțină metadate de geolocalizare la nivel de cartier, să conțină informații detaliate despre situația incidentului sau ale condițiilor de trafic.

Impunerea unor astfel de condiții ar fi nefezabilă într-un sistem care să poată funcționa la nivel global și duce la ignorarea totală a anumitor mesaje care pot avea o valoare considerabilă pentru utilizator, dar nu întrunesc toate standardele de rigurozitate specificate.

O altă lucrare cu o temă similară este lucrarea [2], în care autorul propune monitorizarea stării traficului urban folosind același Streaming API. Metodologia propusă utilizează un algoritm T-POS pentru clasificare și geo-localizare pe baza informațiilor din tweet-uri. Algoritmul este descris în secțiunea 3.4, unde se menționează rolul acestuia de a extrage substantivele din mesaje, care ulterior sunt comparate cu diferite dicționare pentru a obține locația incidentului, starea traficului și justificarea acesteia.

Lucrarea menționată este, de asemenea, de o acuratețe fină, studiul fiind concentrat pe Londra și drumurile din jurul ei.

3.2. Accesarea tweet-urilor și formatul acestora

3.2.1. Accesarea

Twitter facilitează accesul în timp real la informație (tweet-uri, utilizatori, feed-uri) prin intermediul a două API-uri diferite:

- Twitter REST API
- Twitter Streaming API

API-ul REST este folosit în general în aplicații și website-uri care realizează căutarea și filtrarea informațiilor pe baza unor criterii sau interogări. Căutările pot revendica postările publice, timeline-urile utilizatorilor, postări pe baza hashtag-urilor etc. În figura 3.4. este prezentat modelul după care utilizatorul face un request către REST API-ul Twitter, acesta generează răspunsul serverului pe baza criteriilor de căutare și îl tirmite utilizatorului.

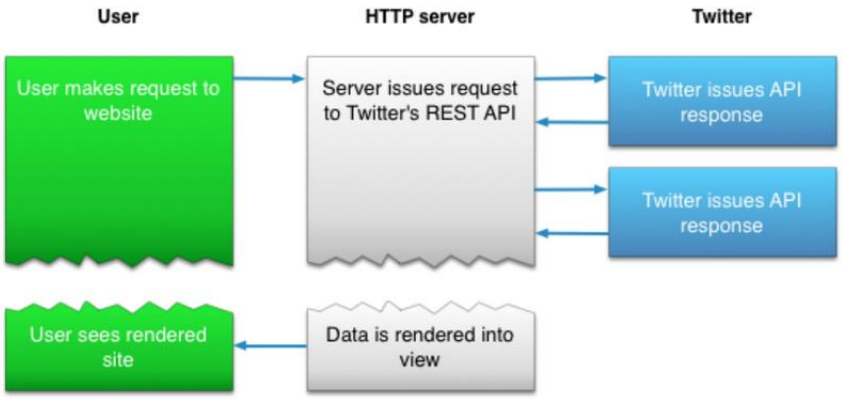

Figura 3.4 Modelul de acces prin Twitter REST API (Steur, 2014)

Streaming API-ul funcționează diferit: se creaează o conexiune continuă, menținută activă într-un proces diferit, în timp ce un alt proces se ocupă de manipularea request-urilor. Odata stabilită o conexiune, tweet-urile sunt recepționate în timp real, fapt ce îl face mai potrivit pentru data mining și analiza de date. Un aspect important de menționat este faptul ca prin intermediul Streaming API-ului se pot accesa doar o porțiune din mesajele transmise în timp real.

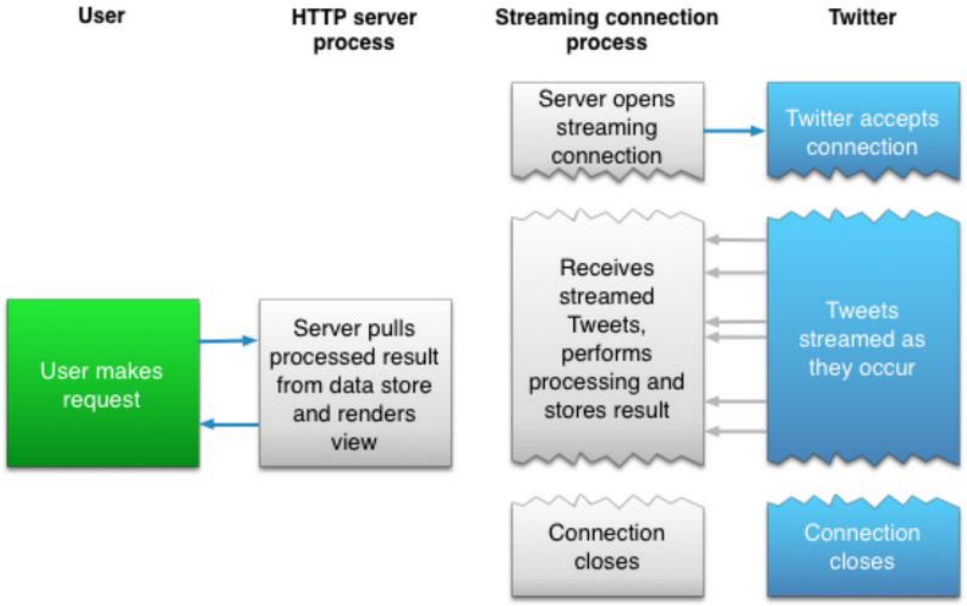

Figura 3.5 Modelul de acces prin Twitter Streaming API (Steur, 2014)

Totuși, având în vedere faptul că lucrarea de față se axează pe înglobarea datelor deja existente, și mai puțin pe analiza datelor, abordarea REST este una mai potrivită și mai eficace pentru cazul de față.

3.2.2. Formatul unui tweet

Conținutul unui tweet obținut prin intermediul API-urilor anterior menționate este unul foarte complex. Din această cauză, în continuare se va prezenta un tabel (tabelul 3.1) al proprietăților considerate relevante pentru proiectul implementat. Informațiile sunt extrase din documentația oficiala a platformei Twitter pentru developeri [7].

Tabel 3.1 Câmpuri relevante ale tweet-urilor din răspunsul API

Proprietate	Tip	Descriere
coordinates	Coordinates	Reprezintă locația geografică a tweet-ului în formatul unui obiect de tip Coordinates
created_at	String	Timpul când tweet-ul a fost creat în format UTC
entities	Entities	Entități care au fost atașate mesajului: hashtag-uri, url-uri, menționări ale altor utilizatori
id	Int64	Identificatorul tweet-ului în format de întreg pe 64 biți

id_str	String	Identificatorul tweet-ului în format String
lang	String	Când este prezent, identifică limba în care a fost scris mesajul
text	String	Conținutul mesajului în format text UTF-8
truncated	Boolean	Indică dacă mesajul din câmpul text a fost truncat
user	Users	Un obiect populat cu câmpuri legate de utilizatorul care a creat mesajul

Alte detalii importante se regăsesc în câmpul "entities", subcâmpurile acestuia conținând informații importante, cum ar fi hashtag-uri sau atașamente media. Acestea sunt descrise în cadrul tabelului 3.2.

Tabel 3.2 Câmpurile obiectului Entities din răspunsul API

Proprietate	Tip	Descriere
hashtags	Vector obiecte	Reprezintă hashtag-uri parsate din context sub forma unor indecși și a textului acestora
media	Vector obiecte	Reprezintă elementele media uploadate împreună cu conținutul mesajului. Poate conține informații despre display URL, identificatorul atașamentului, textul alternativ, dimensiuni și tip
urls	Vector obiecte	Reprezintă URL-urile incluse în textul mesajului sau în cadrul obiectului utilizator.
user_mentions	Vector obiecte	Reprezintă alți utilizatori menționați în textul mesajului.

Un exemplu complet al unui astfel de mesaj se poate observa în figura 3.6.

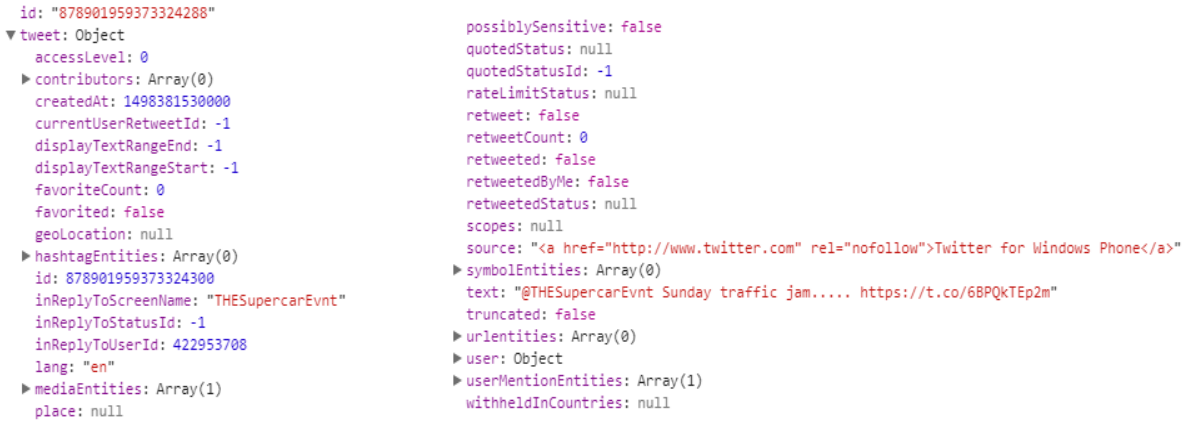

Figura 3.6 Structura unui tweet

3.3. Cloud computing

3.3.1. Concept

Un Cloud este un model computațional "la cerere" (en: on demand) compus din resurse IT (hardware și/sau software) autonome care comunică între ele. Furnizorii de servicii oferă sisteme care să satisfacă termenii predefiniți de calitatea serviciilor prin Internet ca un set de servicii ușor de folosit, scalabile și ieftine clienților interesați pe baza unei subscripții de tip abonament[8].

Principalul avantaj al arhitecturilor de tip cloud este că permit actorilor ce aleg să le folosească (de obicei companii) să se concentreze mai mult pe business-ul efectiv decât pe infrastructura computațională a companiei. În acest fel, platformele cloud reduc din costurile necesare achiziționării, întreținerii și actualizării unei infrastructuri proprii și oferă servicii de înaltă calitate, capabile să scaleze și să se adapteze la situații pe care o infrastructură obișnuită nu ar reuși într-o manieră eficientă și la fel de puțin costisitoare.

Alte avantaje ale arhitecturilor cloud se regăsesc în caracteristicile cloud computing, surprinse și în secțiunea "Characteristics" din [9]:

- Agilitate sporită pentru organizații, creștere a flexibilității utilizatorilor
- Reducerea costurilor
- Independență de dispozitive și de locație
- Menenanță mai ușoară
- Performanță monitorizată de experți IT
- Productivitate crescută datorită modelului concurent de acces al datelor
- Scalabilitate și elasticitate la cerere
- Securitate

3.3.2. Tipuri

Există, în general, trei modele computaționale de tip cloud, denumite pe baza tipului de arhitectură pe care îl oferă, după cum urmează:

1. Infrastructură ca serviciu (en: Infrastructure as a Service - IaaS)
2. Platformă ca serviciu (en: Platform as a Service – PaaS)
3. Software ca serviciu (en: Software as a Service – SaaS)

IaaS oferă spre utilizare o infrastructură completă sub forma unei mașini virtuale și alte resurse cum ar fi medii virtuale de stocare, load balancers, adrese IP, vlan-uri etc. Aceasta constituie layer-ul fundamental în modelul computațional cloud.

Exemple comune de companii care oferă astfel de servicii sunt DigitalOcean[4], Microsoft Azure[5], Amazon Web Services[6] ș.a.

PaaS oferă platforme computaționale care, în mod tipic, include sistemul de operare, mediul de execuție a limbajului de programare, baze de date, server web. Este

[4] https://www.digitalocean.com/
[5] https://azure.microsoft.com/
[6] https://aws.amazon.com/

văzut, din punctul de vedere al abstractizării, ca un nivel intermediar, deasupra IaaS, menționat anterior, dar sub SaaS, menționat în cele ce urmează.

Exemple concrete de sisteme PaaS sunt serviciile AWS Elastic Beanstalk[7], Windows Azure sau Google App Engine[8].

Într-un mediu **SaaS**, utilizatorul primește acces la serviciile de aplicație instalate pe un server. Nu sunt necesare cunoștințe de instalare, configurare sau mentenanță. De multe ori, software-ul poate fi operat direct din browser și este disponibil pentru utilizare și operare.Acest layer este cel mai înalt din punctul de vedere al abstractizării.

Exemple de astfel de servicii sunt Google Docs[9], Gmail, Microsoft Office 365, dar și alte sisteme de acest fel.

Figura 3.7.[10] ilustrează diferențele între cele trei tipuri de modele cloud și gestionarea întregului sistem.

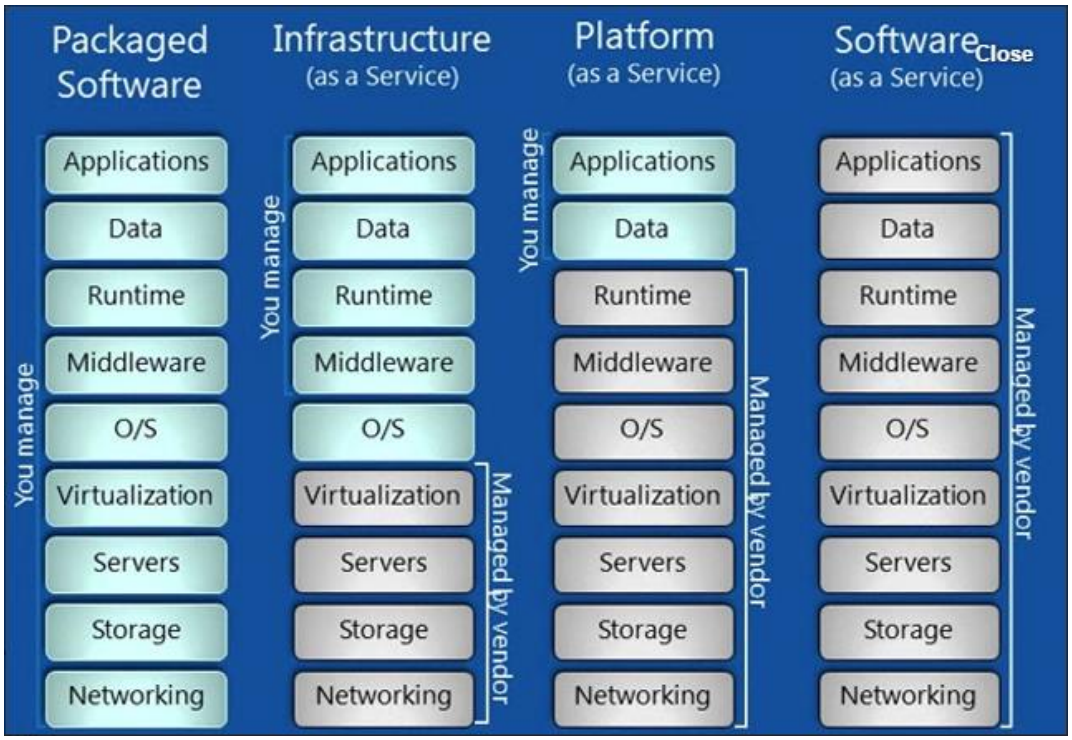

Figura 3.7 Diferențe între modele cloud computing (venturebeat.com)

3.4. Sisteme similare

În cadrul acestui subcapitol se vor descrie un număr de sisteme comerciale cu anumite caracteristici similare proiectului dezvoltat, iar apoi se va realiza o analiză comparativă a acestora cu proiectul personal.

[7] https://aws.amazon.com/elasticbeanstalk/
[8] https://cloud.google.com/appengine/
[9] https://www.google.com/docs/about/
[10] https://venturebeat.com/2011/11/14/cloud-iaas-paas-saas/

3.4.1. Descriere

Waze[11] este o aplicație mobilă disponibilă pentru Android și iOS lansată în anul 2006 sub denumirea inițială de FreeMap Israel. Este un sistem de navigare software care funcționează pe telefoane și tablete cu suport GPS și are ca prim serviciu oferirea indicațiilor de direcții între o rută sursă și o rută destinație. Deși concepută ca o aplicație de tip asistent GPS, Waze s-a facut cunoscută prin sistemul de condiții al traficului în timp real.

Utilizatorii aplicației au posibilitatea să raporteze incidente de pe traseul lor direct în sistem, făcând, astfel, vizibile condițiile de desfășurare a traficului și celorlalți utilizatori. Deși probabil cel mai mare sistem bazat pe comunitate de atenționare a condițiilor de desfășurare a traficului în timp real, Waze are anumite dezavantaje: sistemul este unul de tip închis, doar cei care sunt înregistrați și îl utilizează pot vizualiza alerte ale altor membri înscriși, poate deveni o sursă de distragere la volan și a stârnit critici în ceea ce privește posibilitatea de monitorizare a utilizatorilor. De asemenea, aplicația nu oferă informații despre condițiile meteorologice de pe traseu.

Google maps[12] este un serviciu de cartografiere online dezvoltat de Google, disponibil atât în variantă browser, cât și prin intermediul aplicațiilor mobile pentru platformele populare. Este, probabil, cel mai complex sistem de acest tip și oferă o multitudine de servicii, de la vederi panoramice ale străzilor, imagini satelitare, condiții de trafic și planificare de rute. Totuși, deși Google deține servicii de condiții meteorologice, nici în cadrul Google maps acestea nu sunt folosite pentru afișarea condițiilor meteorologice pe traseu.

O mențiune importantă despre Google maps este faptul că oferă public, prin intermediul unor API-uri bine documentate, accesul programatorilor la o arie largă dintre serviciile menționate anterior.

AccuWeather Road Trip Planner[13] este un serviciu online al platformei AccuWeather.com. Acesta se bazează pe API-ul Google maps menționat în paragraful anterior pentru geolocalizarea și identificarea rutei, asupra căreia va adăuga informații despre condițiile meteorologice de pe traseu.

Utilizatorii aplicației au avantajul că le sunt oferite aceste informații meteorologice care pot avea un impact asupra modului de desfășurare a traficului, dar dezavantajele sistemului includ absența prezentării rutelor alternative și a informațiilor despre acestea și faptul prezentarea condițiilor meteorologice este realizată pentru locații arbitrare de pe traseu în detrimentul punctelor principale de interes (spre exemplu, a orașelor pe baza importanței lor).

3.4.2. Analiză comparativă

În tabelul 3.3. este prezentată o analiză comparativă a sistemelor prezentare anterior relativ la funcționalitățile sistemului implementat. Din conținutul tabelului, se pot observa punctele cheie pe care lucrarea de față le propune și cum sunt acestea situate relativ la sistemele comerciale deja existente.

[11] https://www.waze.com/
[12] https://en.wikipedia.org/wiki/Google_Maps
[13] http://www.accuweather.com/travel-driving-directions-weather.asp

Tabel 3.3 Comparație cu sisteme similare

	Waze	Google Maps	AccuWeather RTP	Proiectul propus
Geolocalizare sursă/destinație	DA	DA	DA	DA
Geolocalizare incidente, precizie	DA, NIVEL DE STRADĂ	DA, NIVEL DE STRADĂ	-	DA, NIVEL DE LOCALITATE
Afișare rute disponibile	DA	DA	PARȚIAL, DOAR UNA	DA
Identificare puncte de interes pe baza importanței	DA	DA	NU	DA
Afișare informații condiții meteorologice rute	NU	NU	DA	DA
Informații despre trafic	DA	DA	NU	DA
Informații trafic bazate pe senzori sociali-comunitate	DA	DA	-	DA
Caracterul sursei informațiilor despre trafic	ÎNCHIS	ÎNCHIS	-	DESCHIS
Raportarea informațiilor inexacte	DA, PRIN COMENTARII	NU	-	DA
Informație adaptată la contextul lingvistic	DA	DA	NU	DA
Accesibil online	NU	DA	DA	DA
Accesibil mobile	DA, NATIV	DA, NATIV	NU, NEOPTIMIZAT	DA, PRIN BROWSER

Capitolul 4. Analiză şi Fundamentare Teoretică

În cadrul acestui capitol, se va descrie arhitectura conceptuală a sistemului implementat și modul în care aceasta va interacționa cu tehnologiile arhitecturale propuse, se vor prezenta cerințele funcționale și non-funcționale ale sistemului, se va defini o serie de cazuri de utilizare și vor fi prezentate tehnologiile utilizate și motivarea utilizării acestora în contextul dat.

4.1. Arhitectura conceptuală

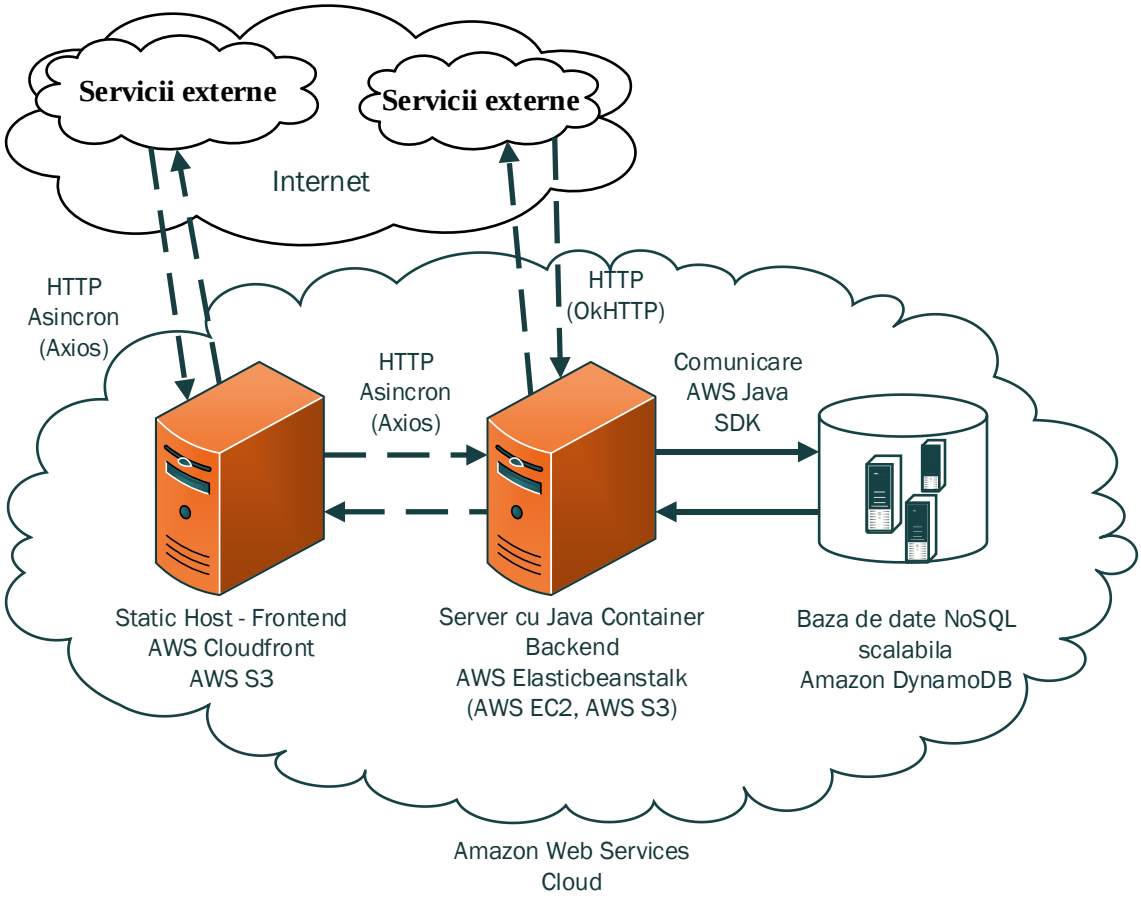

Figura 4.1 Arhitectura conceptuală a sistemului

În figura 4.1. este ilustrată arhitectura de nivel înalt a sistemului. După cum se poate observa în schemă. Sistemul este unul distribuit atât din punct de vedere logic, cât și fizic, alcătuit din trei componente principale:

1. Server Front-end
2. Server Back-end
3. Baza de date

Serverul Front-end va fi găzduit în cadrul unei arhitecturi operată de serviciile AWS Cloudfront și AWS Simple Storage Service (S3). Aici se vor regăsi fișierele statice web (fișiere de markup html, fișiere de stilizare css, scripturi client-side javascript) cu care utilizatorul va interacționa prin intermediul unui browser (desktop sau mobile). Frontend-ul va comunica în mod asincron, pe baza interacțiunii cu utilizatorul, atât cu serverul backend, cât și cu o serie de servicii externe.

Al doilea server, care găzduiește Back-end-ul aplicației, va fi creat utilizând AWS Elastic Beanstalk pentru configurarea unei instanțe de mașină virtuală AWS EC2, ce va avea instalat serverul web Apache Tomcat ca Java Container. Pentru stocarea informațiilor se utilizează AWS S3. Pentru deservirea răspunsurilor venite de la frontend, backend-ul va comunica atât cu baza de date, prin intermediul SDK-ului AWS pentru Java, pentru obținerea informațiilor persistate și actualizarea acestora, cât și cu servicii externe din Internet prin intermediul apelurilor HTTP.

Baza de date va fi găzduită de serviciile Amazon DynamoDB, sub forma unei baze de date NoSQL rapide și foarte scalabile.

4.1.1. Arhitectura conceptuală Frontend

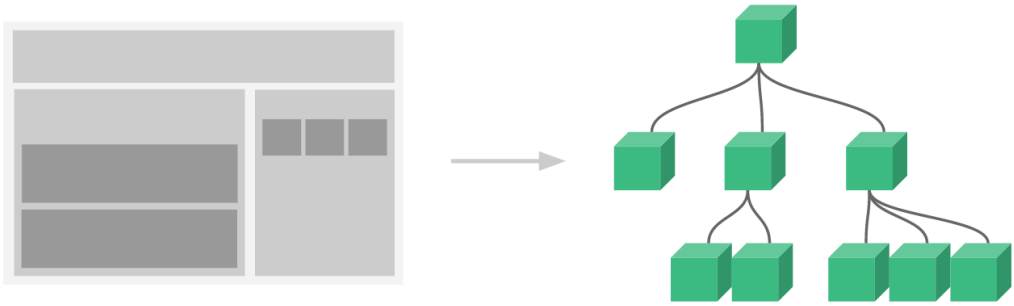

Figura 4.2 Structura conceptuală aplicație web (vuejs.org)

În figura 4.2 [10], este ilustrată schema conceptuală a modelului de abordare în dezvoltarea aplicației web. Se dorește utilizarea unei metode ce impune dezvoltarea bazată pe componente, paginile web fiind împărțite în componente mici și reutilizabile pentru a spori scalabilitatea.

Componentele implementate formează un arbore care reprezintă, în final, interfața grafică cu utilizatorul.

4.1.2. Arhitectura conceptuală Backend

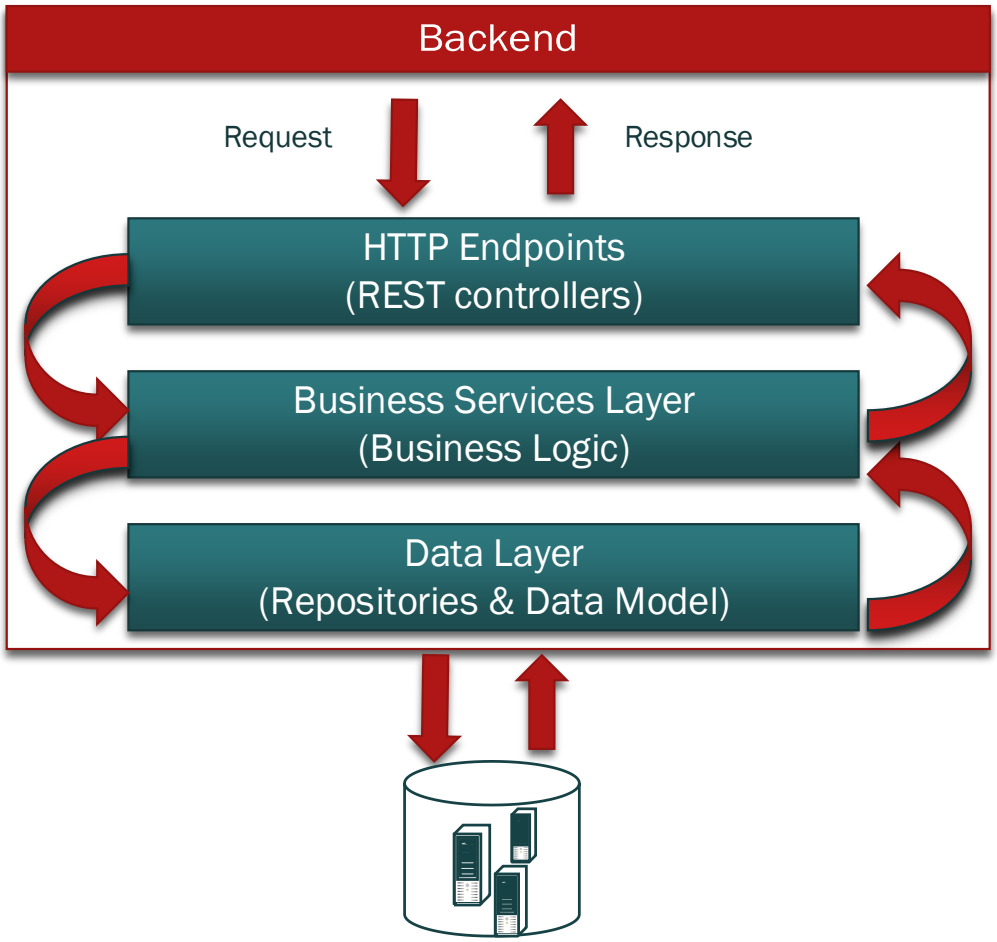

Figura 4.3 Arhitectura conceptuală Back-end

În figura 4.3 este prezentată structura abstractă a serverului principal care va găzdui backend-ului aplicației web. Arhitectura prezintă 3 niveluri (layered architecture) cu responsabilități diferite.

Cel mai înalt nivel, cel al endpoint-urilor HTTP, are ca scop interceptarea request-urilor pentru anumite adrese prin intermediul unor controllere REST și de a delega responsabilitățile necesare nivelului inferior pentru a transmite, după efectuarea operațiilor necesare, un răspuns.

În nivelul serviciilor este prezentă logica computațională necesară deservirii diverselor funcționalități ale aplicației. Acest nivel poate comunica cu servicii exterioare (din Internet), cu servicii din același nivel și poate utiliza nivelul inferior pentru a efectua operații pe baza de date.

Ultimul nivel are ca responsabilitatea maparea modelului de date pe structura bazei de date și deservirea tuturor interacțiunilor cu aceasta.

4.2. Perspectiva tehnologică

În cadrul acestei subsecțiuni se va realiza o prezentare a tehnologiilor utilizate și se va prezenta argumentul utilizării acestora în contextul implementării proiectului propus.

4.2.1. Vue.js

Vue.js[11] este un framework progresiv open-source pentru construirea interfețelor grafice prin intermediul JavaScript. Vue este un framework de aplicații web capabil să susțină aplicații single-page avansate și este construit pentru a organiza și simplifica dezvoltarea aplicațiilor web.

Framework-ul este bazat pe șablonul arhitectural MVVM (Model-view-viewmodel) și propune dezvoltarea interfețelor utilizând o abordare bazată pe componente. Este, în prezent, framework-ul UI javascript care a înregistrat cea mai mare creștere de popularitate constantă pe platforma GitHub din ultimele 12 luni[14].

Tehnologia aleasă adaugă un plus important de lizibilitate și organizare a codului, aspect foarte important în domeniul front-end, unde codul poate deveni mai ușor dezorganizat. O componentă tipică Vue conține, de obicei, trei părți: o parte de **template**, unde este descris mark-upul HTML al componentei și unde se pot injecta prin mecanisme de templating elemente dinamice, o parte de **style**, unde este definită stilizarea componentei, și partea de **script**, unde este definit comportamentul dinamic al componentei.

Figura 4.4 Exemplu de componentă Vue.js

[14] http://bestof.js.org/tags/framework/trending/last-12-months

Un alt avantaj major al Vue.js este viteza. Framework-ul învață din greșelile altor tehnologii populare, cum ar fi Angular sau React, și reușește să pună în balanță simplitatea dezvoltării aplicațiilor cu controlul pe care îl preia asupra aplicației. Astfel, în benchmark-uri[15] reușește să obțină viteze mai bune decât framework-urile menționate anterior.

4.2.2. Axios

Axios este o librărie npm pentru request-uri ajax client-side și server-side menită să înlocuiască un client HTTP standard. Va fi utilizat în implementarea proiectului pentru apelurile asincrone atât serverul aplicației, cât și o serie de servicii ce vor fi apelate direct din aplicația web.

Este o librărie foarte utilă întrucât simplifică funcționalitățile obiectului standard JavaScript XMLHttpRequest, iar printre avantajele care au condus la integrarea lui în proiect se numără:

- Suportă Promise API
- Se pot adăuga interceptori pentru request-uri și response-uri
- Transformă automat datele în format JSON
- Suport client-side pentru autentificare

4.2.3. Webpack

Webpack este utilizat în aplicațiile JavaScript moderne pentru gruparea resurselor de diferite tipuri (de exemplu, a modulelor .vue), parsarea acestora și exportarea lor într-un format standard care să includă toate modulele aplicației. Librăria realizează acest proces prin construirea unui graf al dependințelor în mod recursiv.

În implementarea proiectului, webpack este foarte util în extragerea componentelor vue, parsarea acestora (a părților **template**, **style**, **script)** și exportarea lor în fișiere standard html, css, javascript, ce pot ulterior fi încărcate pe orice fel de web server static, fără restricții.

4.2.4. Google Maps API

Google Maps API[16] este un serviciu pentru programatori oferit de Google începând din iunie 2005, care le permite să integreze hărțile Google Maps în cadrul aplicațiilor lor și să acceseze serviciile oferite pentru acestea.

Deși, relativ recent, au apărut o serie de alternative, cum ar fi HERE Maps API[17], Bing Maps Platform[18], Leaflet[19] ș.a., Google Maps API este cel mai precis serviciu de acest tip datorită investițiilor majore în platformă. Hărțile sunt în permanență actualizate și sunt oferite interfețe de utilizare pentru web, Android și iOS. Mai mult, utilizarea acestui API permite folosirea unui set larg de subservicii, dintre care se vor prezenta, în cele ce urmează, acelea ce s-au dovedit utile în dezvoltarea aplicației propuse.

[15] JS Framework Benchmark
https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts/table.html
[16] Google Maps API, https://developers.google.com/maps/
[17] HERE Maps API, https://developer.here.com/
[18] Bing Maps Platform, https://www.bingmapsportal.com/
[19] http://leafletjs.com/

4.2.4.1. Google Maps Directions API

Google Maps Directions API[13] este un subserviciu al API-ului menționat anterior. Oferă rutele disponibile între locații și stă la baza serviciului de navigare al Google Maps și va fi integrat în cadrul sistemului propus.

În cadrul aplicației web, acesta poate fi accesat prin intermediului SDK-ului JavaScript pus la dispoziție, care va realiza request-uri utilizând o serie de parametri (de exemplu origine, destinație, mod de transport etc.) și va obține informațiile necesare despre rute, sub forma unor metadate și a unui set de puncte geografice care ulterior pot fi afișate pe hartă.

Răspunsurile primite în urma request-urilor au un format relativ complex, cuprinzând o cantitate mare de informație, dar documentația oficială[13] ajută la înțelegerea completă a sistemului.

4.2.4.2. Google Maps Geocoding & Reverse Geocoding

Geocoding (codificarea geografică) este procesul de conversie a adreselor în coordonate geografice (sub formă de latitudine și longitudine), care pot fi utilizate pentru identificarea unei poziții pe hartă[14].

Reverse geocoding-ul este procesul invers, de obținere a adreselor/locațiilor inteligibile de un individ pe baza unor coordonate geografice.

API-ul prezentat va fi utilizat în cadrul sistemului pentru realizarea celor două sarcini prezentate prin intermediul unui sistem de comunicare ce utilizează protocolul HTTP pe baza SDK-ului mai sus menționat. Figura 4.5. din documentația oficială[14], prezintă formatul request-ului ce se efectuează către server pentru codificarea geografică a unei adrese. Pentru procedeul invers, cererea este de factură similară.

```
https://maps.googleapis.com/maps/api/geocode/outputFormat?parameters
```

Figura 4.5 Format request de geocodificare către Geocoding API (Google)

4.2.4.3. Google Places API Web Service

Places API[15] este un serviciu web care poate fi utilizat împreună cu (dar și independent de) Google Maps API. Acesta poate oferi o serie de funcționalități pentru listarea punctelor de interes dintr-o arie geografică, detalii despre anumite locații, un sistem de autocomplete pentru alegerea locațiilor ș.a.

În implementarea propusă, acest serviciu va juca un rol foarte important în algoritmul de identificare a punctelor de interes de pe rutele obținute prin intermediul serviciului prezentat la subsecțiunea 4.2.4.1.

Astfel, cu ajutorul acestui serviciu, putem transforma o rută dintr-o listă de dimensiuni mari de puncte geografice într-un set de dimensiuni restrânse de locații importante de pe rută (orașe, localități importante), pe care le vom putea procesa ulterior pentru obținerea de informații despre condițiile meteorologice și cele de desfășurare a traficului.

4.2.5. Twitter REST APIs

API-urile RESTful oferite de Twitter sunt fundamentul unor obiective majore enunțate în textul lucrării de față. Acestea vor fi utilizate pentru a extrage informații legate de condițiile de desfășurare a traficului din comunitatea rețelei de socializare Twitter. Prin intermediul acestor servicii, se așteaptă utilizarea mesajelor postate de utilizatori simpli, martori oculari ai evenimentelor care îngreunează circulația, și a mesajelor postate de canalele de știri prin acest mijloc de comunicare.

4.2.5.1. GET search/tweets

GET search/tweets, denumit după metoda HTTP utilizată și path-ul pentru a accesa serviciul, poate fi considerat fundamentul API-urilor REST oferite de Twitter ce vor fi utilizate în cadrul aplicației implementate.

Prin intermediul acestui subserviciu se pot revendica mesaje după formatul de tweet exemplificat în tabelele 3.1 și 3.2 din secțiunea anterioară.

Serviciul primește ca parametri, prin intermediul URL-ului, o interogare (detaliată în secțiunea 4.2.5.2), iar în mod opțional detalii despre geolocalizare (coordonate și arie), limbă, tip de rezultat (recent sau popular), opțiuni de paginare, detalii legate de intervalul de timp ș.a.

4.2.5.2. Twitter Search API

O altă tehnologie importantă ce va fi utilizată în implementarea proiectului propus este Twitter Search API, ca parte din Twitter REST API și utilizată în serviciul GET search/tweets.

Aceasta este o interfață utilă în dezvoltarea de interogări pentru a găsi mesajele relevante care corespund cerințelor de căutare din ultimele 7 zile. Un aspect important este faptul că Search API-ul are ca scop identificarea de rezultate cât mai relevante, în vreme ce Streaming API-ul[20] se axează pe aducerea câtor mai multe rezultate, fapt ce avantajează tehnologia prezentată în contextul implementării proiectului propus.

Twitter Search API are un comportament similar cu funcționalitatea de Search de pe clienții web și mobile ai rețelei de socializare. În fapt, pentru a verifica dacă interogarea a fost construită în mod corect și valid, dezvoltatorii recomandă testarea acesteia utilizând funcționalitatea precedent amintită[21].

Interogările vor fi transmise în cadrul URL-ului de request, prin intermediul parametrului “q” Figura 4.6 prezintă un exemplu simplificat de transmitere a parametrului de interogare prin URL. Într-un caz real, request-ul necesită autentificare, interogările sunt mult mai complexe, pot conține informații de geolocalizare, context lingvistic ș.a.m.d.

Figura 4.6 Transmiterea interogării prin parametru

[20] Twitter Streaming APIs Overview, https://dev.twitter.com/streaming/overview
[21] Twitter Search API, https://dev.twitter.com/rest/public/search

Un aspect important este faptul că, făcând parte dintr-un URL accesat prin protocolul HTTP, caracterele speciale ale parametrilor transmiși trebuie să fie URL Encoded[22].

Pentru înțelegerea posibilităților de dezvoltare a interogărilor și de rafinare a căutării, se prezintă tabelul 4.1., preluat din documentația Twitter pentru programatori[7], care descrie cazuri concrete de structurare a operatorilor interogării și explică rezultatul scontat al căutării.

Mai mult, din operatorii prezentați în tabel, se poate înțelege gradul ridicat de rafinare al căutării prin utilizarea diverselor combinații ale acestora și nivelul detaliu disponibil, mesajele fiind clasificate în cadrul sistemului și în funcție de sentimente.

Tabel 4.1 Operatori de interogare și semnificație (Doc. Twitter online)

Operator	Finds Tweets...
watching now	containing both “watching” and “now”. This is the default operator.
“happy hour”	containing the exact phrase “happy hour”.
love OR hate	containing either “love” or “hate” (or both).
beer -root	containing “beer” but not “root”.
#haiku	containing the hashtag “haiku”.
from:interior	sent from Twitter account “interior”.
list:NASA/astronauts-in-space-now	sent from a Twitter account in the NASA list astronauts-in-space-now
to:NASA	a Tweet authored in reply to Twitter account “NASA”.
@NASA	mentioning Twitter account “NASA”.
politics filter:safe	containing “politics” with Tweets marked as potentially sensitive removed.
puppy filter:media	containing “puppy” and an image or video.
puppy -filter:retweets	containing “puppy”, filtering out retweets
puppy filter:native_video	containing “puppy” and an uploaded video, Amplify video, Periscope, or Vine.
puppy filter:periscope	containing “puppy” and a Periscope video URL.
puppy filter:vine	containing “puppy” and a Vine.
puppy filter:images	containing “puppy” and links identified as photos, including third parties such as Instagram.
puppy filter:twimg	containing “puppy” and a pic.twitter.com link representing one or more photos.
hilarious filter:links	containing “hilarious” and linking to URL.
puppy url:amazon	containing “puppy” and a URL with the word “amazon” anywhere within it.
superhero since:2015-12-21	containing “superhero” and sent since date “2015-12-21” (year-month-day).
puppy until:2015-12-21	containing “puppy” and sent before the date “2015-12-21”.
movie -scary :)	containing “movie”, but not “scary”, and with a positive attitude.
flight :(	containing “flight” and with a negative attitude.
traffic ?	containing “traffic” and asking a question.

[22] W3Schools, URL Encoding Reference, https://www.w3schools.com/tags/ref_urlencode.asp

4.2.6. Yandex Translate API

Unul dintre obiectivele secundare ale sistemului propus, menționat în secțiunea 2.2.3 este "Adaptarea la contextul lingvistic al punctelor de interes". Pentru realizarea acestui obiectiv, aplicația va trebui să poată detecta limba utilizată în punctele de interes, iar, ulterior, să traducă automat termenii căutării dintr-o interogare după modelul prezentat anterior, în secțiunea 4.2.5.2, și să refacă interogarea.

Yandex Translate API[16] este un serviciu online disponibil gratuit care folosește platforma internă a companiei pentru traducerea termenilor și a expresiilor din mai mult de 70 limbi.

Pentru traducerea unui mesaj sau a unui termen, se va utiliza o comunicație bazată pe protocolul HTTP, care va conține în parametrii URL-ului limbile de conversie și textul mesajului ce trebuie tradus.

Avantajele serviciului constau în faptul că este disponibil gratuit și că reușește să obțină rezultate de traducere bune (traduceri corecte) pentru termeni și texte de dimensiuni scurte.

4.2.7. OpenWeatherMap API

Petru obținerea informațiilor despre condițiile meteorologice din punctele de interes, există două posibile abordări: dezvoltarea de stații meteo sau preluarea acestor informații dintr-o sursă externă sub forma unui serviciu online.

OpenWeatherMap[17] este o platformă ce oferă astfel de servicii ce poate fi integrată în aplicația dezvoltată. Prin intermediul acesteia, se pot accesa informații despre starea curentă a vremii din anumite locații, prognoze pentru 5 sau 16 zile, un istoric al datelor ș.a.

Avantajele acestui serviciu constau în gratuitatea lui, documentația bine structurată, viteza de acces și flexibilitatea formatelor de răspuns, XML sau JSON, cel din urmă fiind integrat cu ușurință în cadrul unei aplicații web care folosește JavaScript.

4.2.8. Spring Framework. Module Spring

4.2.8.1. Spring Framework

Spring[18][19] este un framework de aplicație open-source și un container IoC (Inversion of Control) bazat pe "injectarea" dependințelor(en: Dependency Injection[23]) pentru platforma Java. Principalele funcționalități pot fi folosite de orice tip de aplicație Java, dar există extensii peste platforma Java EE (Enterprise Edition) pentru a construi aplicații web.

Framework-ul conține numeroase module menite să abstractizeze și să eficientizeze anumite cerințe generice și repetitive, dintre care vor fi prezentate cele alese pentru implementarea server-ului backend.

[23] https://en.wikipedia.org/wiki/Dependency_injection

4.2.8.2. Spring Boot

Spring Boot este implementarea Spring a paradigmei de proiectare convention-over-configuration (convenție mai presus de configurație) pentru crearea unei aplicații complete, independente, pregătită pentru mediul de producție. Înglobează platforma Spring și librării conexe pentru a simplifica procesul de configurare și inițializare a proiectului.

Cu ajutorul acestui serviciu, dependințele Spring ale proiectului vor putea fi configurate cu un minim de setări. Un alt avantaj important este faptul că Spring Boot înglobează sub forma unei dependințe un server de aplicație Tomcat (sau, la alegere, Jetty) în care aplicația poate fi rulată într-un mod automatizat și ulterior și, eventual, integrată într-un serviciu cloud.

4.2.8.3. Spring Data

Spring Data este un modul al framework-ului care oferă o modalitate consistentă de acces al datelor prin intermedierea operațiilor asupra bazei de date.

Mai mult, modulul oferă funcționalități de generare a unei baze de date pe baza modelului orientat pe obiect, dar și pentru maparea unei baze de date existente într-un model orientat pe obiect, de validare a schemelor bazelor de date ș.a.

În aplicația propusă, Spring Data poate juca un rol important în cel mai de jos layer al backend-ului, vizibil în ilustrația din Figura 4.3.

Un alt avantaj major este acela că oferă un model de operare independent de tehnologia utilizată pentru baza de date, iar în cazul unei decizii ulterioare de schimbare a tehnologiei bazei de date, modificările din layer-ul de Data Access vor fi minime.

4.2.8.4. Spring Web MVC

Spring Web MVC funcționează ca un dispecer pentru request-urile web primite de aplicație. Acesta poate mapa cererile venite pe anumite adrese și poate controla asignarea lor unor metode logice Java, de cele mai multe ori prin intermediul adnotărilor @Controller și @RestController.

În cadrul dezvoltării proiectului, acest modul poate fi folosit în layer-ul superior HTTP Endpoints din Figura 4.1 prin intermedierea accesului la server pe diferitele metode ale protocolului HTTP, după cum se poate observa în Figura 4.7.

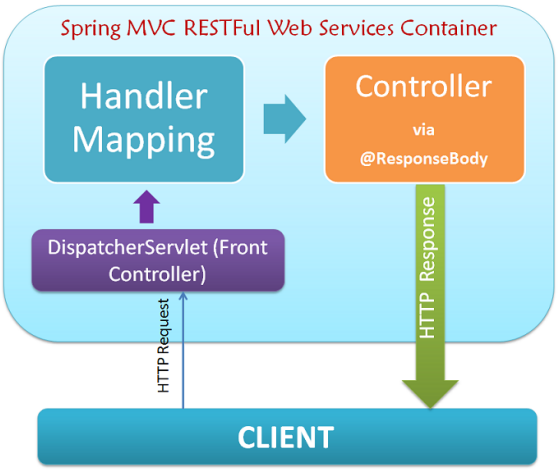

Figura 4.7 Modelul de comunicație RESTful intermediat de Spring Web MVC

4.2.8.5. Spring Security

Spring Security este un alt proiect aferent Spring care oferă funcționalități de autentificare și autorizare și poate fi integrat în suita Spring WEB MVC prezentată anterior.

Deși proiectul implementat dorește navigarea cu ușurință în cadrul aplicației, iar un utilizator normal nu ar trebui să aibă nevoie de autentificare pentru a accesa serviciul, anumite funcționalități care țin de managementul informațiilor pot compromite buna funcționare a aplicației și, drept urmare, necesită autorizarea cererilor.

Prin intermediul acestei tehnologii, cererile pot fi filtrate și procesate doar în cazul în care acestea îndeplinesc cerințele de autorizare.

Un posibil dezavantaj este acela că framework-ul este conceput pentru medii în care se utilizează o sesiune, iar comunicarea REST prin intermediul HTTP nu conține în mod implicit astfel de informații. Totuși, se oferă posibilitatea de a integra în cadrul tehnologiei alte standarde de autentificare.

4.2.9. JWT (JSON Web Tokens)

JSON Web Token este un standard deschis (RFC 2519[20]) care definește o modalitate compactă de transmitere securizată a informației între doi terți sub forma unui JSON. Informația poate fi verificată, deoarece este semnată digital prin intermediul algoritmului HMAC[21] sau a unui algoritm asimetric de criptare.

În cadrul sistemului propus, JWT poate reprezenta soluția pentru dezavantajul prezentat al Spring Security, asigurând, astfel, un sistem de autentificare eficient pe bază de tokens care funcționează după un model asemănător cu cel ilustrat în figura 4.8, preluată de pe site-ul de prezentare al tehnologiei[24].

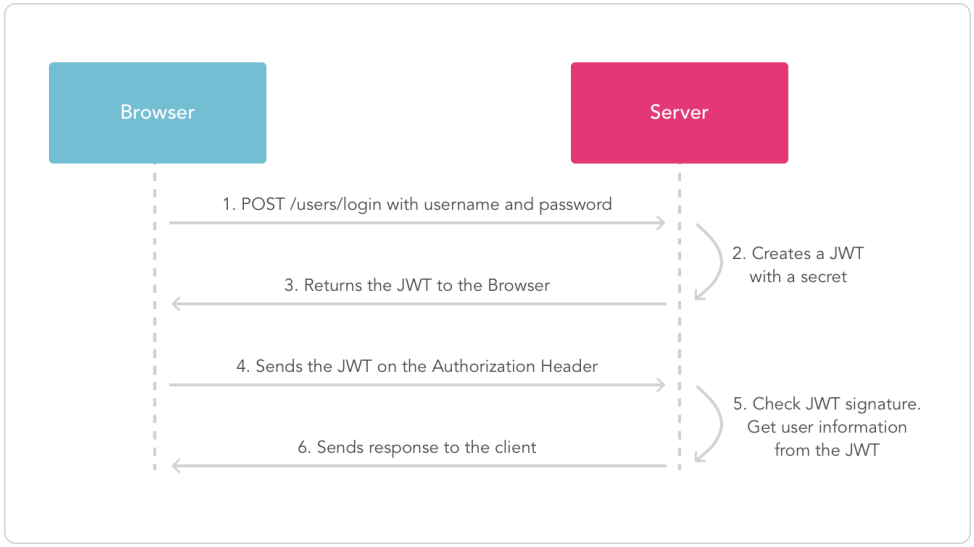

Figura 4.8 Modelul de autentificare folosind JWT (jwt.io)

[24] https://jwt.io/introduction/

4.2.10. AWS: CloudFront, S3, Elastic Beanstalk, DynamoDB

Între obiectivele secundare ale proiectului, secțiunea 2.2.2 propune integrarea aplicației dezvoltate în cadrul unei arhitecturi cloud.

Amazon Web Services este cel mai semnificativ actor de pe piață care oferă servicii de tip cloud computing[25]. AWS oferă o gamă largă de soluții pentru îndeplinirea necesităților clienților.

Din suita de produse AWS, în proiectul de față se vor utiliza AWS Cloudfront, AWS S3, AWS Elastic Beanstalk si AWS DynamoDB, descrise în cele ce urmează.

4.2.10.1. Amazon CloudFront

CloudFront este un serviciu CDN (Content Delivery Network) care accelerează transmiterea fișierelor statice (ale unui website, spre exemplu) către utilizator.

Concret, cu ajutorul tehnologiei, resursele aplicației web pot fi distribuite într-un sistem global de servere în așa fel încât acestea să poată fi cached, reducând timpul de acces la o aplicație web.

4.2.10.2. Amazon S3

Amazon S3 (**S**imple **S**torage **S**ervice) este un serviciu de stocare a datelor în cloud cu o interfață web simplă. Oferă 99.999999999% durabilitate și este scalabil până la miliarde de obiecte stocate[26].

Pentru aplicația de față, S3 va găzdui atât fișierele necesare funcționării front-end-ului, cât și deployment-ul pentru serverul de back-end.

Împreună cu CloudFront, prezentat anterior, va oferi utilizatorului acces la clientul aplicației web, într-o manieră arhitecturală exemplificată în figura 4.9.

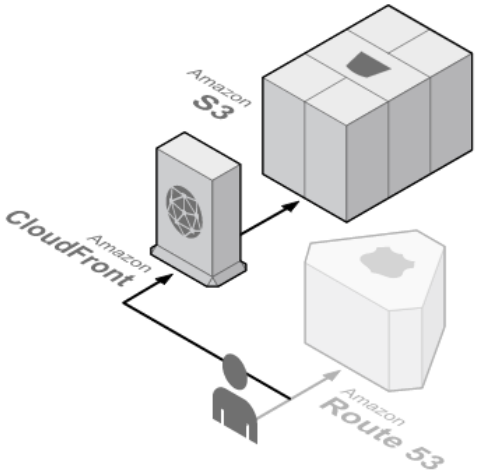

Figura 4.9 Accesarea clientului aplicației web prin CloudFront și S3 (Amazon)

[25] "AWS still owns the cloud", TechCrunch
https://techcrunch.com/2017/02/02/aws-still-owns-the-cloud/
[26] AWS S3 Introduction, https://aws.amazon.com/s3/

4.2.10.3. AWS Elastic Beanstalk

AWS Elastic Beanstalk[22] este un serviciu de automatizare și management al integrării unei infrastructuri în cloudul AWS.

În cadrul unui scenariu normal, pentru deployment-ul unui server de aplicație care să poată susține containere Java, ar fi necesară obținerea și instalarea unei mașini virtuale, instalarea și configurarea remote a tuturor programelor și mediilor necesare rulării acestuia, iar, în cele din urmă, instalarea serverului de aplicație și integrarea arhivei exportate a proiectului în acesta.

Cu ajutorul Elastic Beanstalk, clientul își poate selecta opțiunile de arhitectură și configurație dorite, iar platforma se va ocupa de pregătirea acesteia pentru mediul de producție.

Pe lângă simplitatea oferită de acesta, un alt avantaj major ar fi posibilitatea de scalare orizontală și verticală prin adăugarea de noi sisteme sau îmbunătățirea configurației sistemului curent.

În proiectul de față, tehnologia propusă poate fi utilizată pentru găzduirea distribuită a serverului de backend.

4.2.10.4. Amazon DynamoDB

DynamoDB[23] este un serviciu de baze de date NoSQL rapid și flexibil oferit de Amazon. Serviciul promite consistență și latență de o cifră în milisecunde[27]. De asemenea, oferă o interfață web pentru managementul datelor distribuite în cloud.

Viteza tehnologiei, net superioară vitezelor oferite de bazele de date relaționale, dar și flexibilitatea modelului de date și posibilitatea integrării cu Spring Data fac ca DynamoDB să fie o soluție oportună pentru persistarea datelor aplicației propuse, dar și pentru integrarea în alte sisteme web, big data, IoT sau jocuri online.

4.3. Cerințele sistemului

În cadrul subcapitolului curent se va descrie un set de cerințe funcționale și non-funcționale prin intermediul cărora se vor putea atinge obiectivele enunțate în capitolul 2.

4.3.1. Cerințe funcționale

Funcționalitățile efective ale sistemului, pot fi sumarizate de următoarea listă de cerințe funcționale:

CF1. **Afișarea unei hărți actualizate și posibilitatea navigării acesteia**. Presupune oferirea posibilității utilizatorului de a vizualiza ansamblul contextului geografic înainte și după procesarea informațiilor cerute de acesta.

CF2. **Introducerea locației sursă și a locației destinație.** Este un aspect fundamental ca utilizatorul să poată delimita zonele importante prin introducerea punctului de plecare și de sosire al traseului dorit. Astfel, algoritmii vor putea determina punctele de interes pentru a oferi informații cât mai relevante.

[27] DynamoDB Introduction, https://aws.amazon.com/dynamodb/

CF3. **Determinarea rutelor disponibile.** Sistemul trebuie să poată să găsească rutele disponibile dintre cele două locații și să le afișeze într-un mod ușor de înțeles pe harta menționată în CF1.

CF4. **Identificarea punctelor de interes de pe rute.** Un aspect deosebit de important în procesările următoare este reprezentat de identificarea acestor puncte de interes, reprezentate de cele mai multe ori de orașele importante de pe un anumit traseu. Astfel, se vor putea identifica ulterior informații geolocalizate mai relevante.

CF5. **Determinarea informațiilor despre condițiile meteorologice.** Presupune utilizarea punctelor de interes menționate în CF4, obținerea informațiilor despre condițiile meteorologice și afișarea acestora pe harta menționată în CF1.

CF6. **Determinarea informațiilor despre condițiile de desfășurare a traficului.** Este, din perspectiva utilizatorului, o completare a CF5, și implică determinarea tweet-urilor relevante condițiilor de desfășurare a traficului în punctele de interes indetificate și afișarea acestora utilizatorului aplicației.

CF6.1 **Adaptarea procesului de determinare la contextul lingvistic.** O cerință importantă pentru îmbunătățirea funcționării și pentru a asigura faptul că sistemul nu este limitat doar la o anumită regiune geografică este adaptarea căutării pentru obținerea rezultatelor relevante indiferent de țară sau de limbă.

CF6.2 **Raportarea rezultatelor irelevante.** În dezvoltarea sistemului, în ceea ce privește rezultatele obținute din rețeaua de socializare Twitter, trebuie considerat potențialul apariției rezultatelor mai puțin relevante, neactualizate sau necorespunzător geolocalizate. De aceea, aplicația trebuie să pună la dispoziție utilizatorului un sistem de raportare a rezultatelor irelevante, pentru a îmbunătăți căutările ulterioare.

Următoarele cerințe funcționale vin în completarea celor enumerate anterior, dar privesc partea de administrare a serviciului, disponibilă managerilor de informație.

CF7. **Autentificarea și autorizarea administratorilor.** Sistemul trebuie să ofere administratorilor posibilitatea de a dovedi nivelul de acces prin intermediul autentificării în sistem. Un alt aspect important este autorizarea operațiilor acestui rol doar personaleor autentificate, în prealabil, în sistem.

CF8. **Supervizarea rezultatelor marcate ca fiind irelevante.** Chiar daca sistemul dorește implementarea unei filtrări a rezultatelor bazată pe comunitate, menționată în CF6.2, un aspect care nu trebuie neglijat este posibilitatea de marcare greșită, cu sau fără rea intenție. Din acest motiv, supervizarea acestui sistem de către un operator uman este un aspect important pentru buna funcționare a sistemului.

CF9. **Supervizarea traducerilor automatizate.** În CF6.1 s-a menționat necesitatea de a traduce anumite sintagme pentru adaptarea căutării la

contextul lingvistic. Deși se dorește automatizarea acestor traduceri prin mijloacele tehnologice menționate în subcapitolul 4.2.6, trebuie luat în calcul factorul de eroare al mașinilor de traducere și supervizarea acestui proces. În cazul puțin probabil în care va fi nevoie de intervenția administratorului, acesta va putea corecta sau adapta traducerea.

4.3.2. Cerințe non-funcționale

În continuare, se vor enumera o serie de constrângeri ale serviciilor și ale funcțiilor oferite de sistem, având în vedere perspectiva aspectelor non-funcționale ale acestuia.

CNF1. **Dimensiunea.** Având în vedere faptul că se dorește integrarea sistemului în cadrul unei arhitecturi cloud, dimensiunea totală a acestuia poate fi un aspect important în determinarea costurilor. Din acest motiv, dimensiunea totală a aplicațiilor și a utilităților necesarea nu trebuie să depășească **5GB**.

CNF2. **Disponibilitate.** Un alt aspect important este asigurarea unei disponibilități ridicate a sistemului, pentru a putea fi folosit în orice moment de utilizatori. Considerând acest aspect, se dorește menținerea unui up-time de peste **90%.** Acest procent de disponibilitate poate fi atins prin alegerea unei arhitecturi de găzduire a serivciului staiblă, dar și prin tratarea erorilor aplicației pentru a asigura că indiferent de scenariu, aceasta va funcționa în continuare în mod normal.

CNF3. **Scalabilitate.** Având în vedere imposibilitatea de a anticipa cu exactitate un număr mediu de utilizatori sau volumul de lucru la care sistemul va fi expus, un aspect crucial este scalabilitatea, atât verticală, prin introducerea de noi resurse, cât și orizontală, prin distrubirea pe mai multe noduri de sistem.

CNF4. **Utilizabilitatea.** Aplicația web dezvoltată trebuie să aibă o interfață grafică cât mai intuitivă pentru utilizator și trebuie să se poată adapta la ecrane de dimensiuni mai mici. În cazul în care aplicația este într-o stare de execuție a unor procese, sistemul nu trebuie să se blocheze și este important să marcheze acest aspect grafic pentru a atenționa utilizatorul.

CNF5. **Securitatea.** Accesul la submodule ale aplicației care necesită autentificare și autorizare trebuie blocat, atât pentru aplicația web, cât si pentru serverul din spatele acesteia. Presupune securizarea accesului la anumite pagini și securizarea endpoint-urilor de pe server care pot efectua acțiuni privilegiate.

CNF6. **Extensibilitatea.** Posibilitatea dezvoltării continue a sistemului este un aspect deosebit de important, iar, din acest motiv, aplicația construită și componentele principale ale acesteia, prin arhitectura aleasă, trebuie să permită adăugarea de noi funcționalități într-un mod cât mai facil.

4.4. Cazuri de utilizare

4.4.1. Actori

Secțiunea curentă are ca scop definirea actorilor principali din sistemul implementat și descrierea unor posibile cazuri de utilizare reprezentative rolurilor acestora.

Sistemul propus are un număr redus de actori, cu posibilități relativ limitate, vizibili în tabelul 4.2 și enumerați după cum urmează:

1. Utilizator
2. Administrator aplicație
3. Administrator baza de date

Tabel 4.2 Descrierea actorilor și a responsabilităților

Nume	Descriere	Responsabilități
Utilizator	Simplu utilizator al aplicației	Navigarea paginilor aplicației nerestricționate ale aplicației, utilizarea serviciile oferite, raportarea rezultatelor irelevante
Administrator aplicație	Manager al informațiilor aplicației	Supervizarea și validarea rezultatelor raportate ca fiind irelevante, supervizarea și corectarea/adaptarea traducerilor automatizate
Administrator bază de date	Manager al bazei de date	Operațiuni de administrare ale conturilor administratorilor de aplicație

În continuare, vor fi prezentate descrierile unor cazuri semnificative de utilizare pentru actorii 1 și 2, aceștia fiind considerați reprezentativi pentru aplicația propusă.

4.4.2. Descriere

Figura 4.10 sumarizează actorii reprezentativi ai sistemului și modul în care aceștia interacționează cu cazurile de utilizare.

În subsecțiunile următoare vor fi descrise cazurile semnificative de utilizare din sistemul propus pentru actorii prezentați în figură, utilizatorul și administratorul de aplicație.

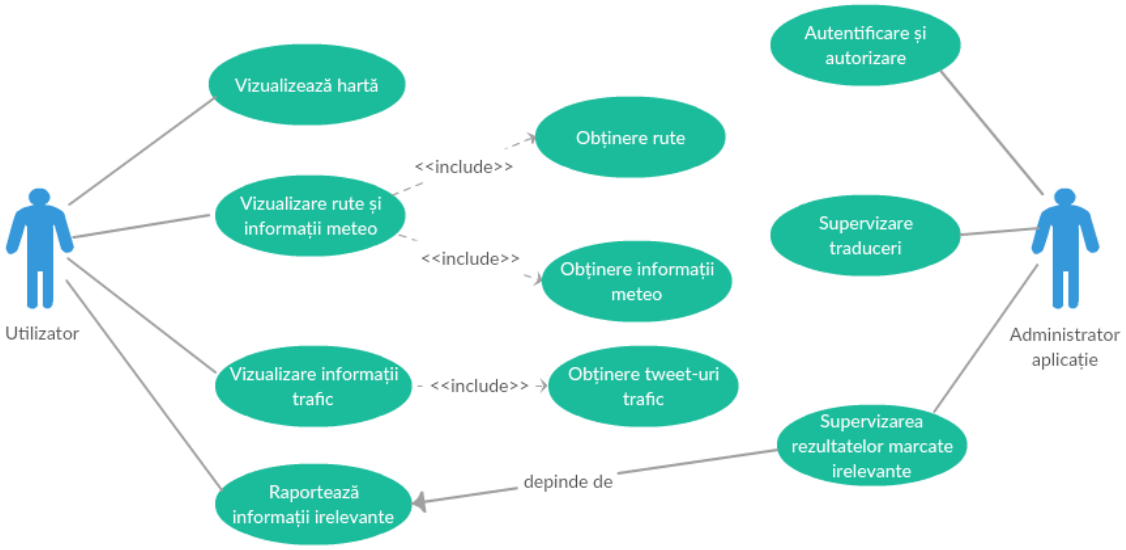

Figura 4.10 Diagramă UML a cazurilor de utilizare

4.4.2.1. Cazul de utilizare I

Numele cazului de utilizare: Vizualizare rute și informații meteorologice
Actor principal: Utilizatorul
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare

Postcondiții:

1. Rutele sunt afișate pe hartă
2. Condițiile meteorologice sunt afișate pe hartă
3. Utilizatorul poate interacționa cu informațiile

Scenariul principal (de succes):

1. Utilizatorul introduce datele despre locația sursă, despre locația destinație și declanșează încărcarea rutelor
2. Sistemul face vizibilă utilizatorului starea de încarcare a rezultatelor
3. Sistemul obține și afișează rutele disponibile între cele două locații
4. Sistemul obține și afișează informațiile despre condițiile meteorologice prin intermediul unor iconițe plasate pe hartă în punctele de interes
5. Utilizatorul poate interacționa cu iconițele plasate pentru afișarea unei descrieri mai detaliate

Scenariii alternativ:

1. Utilizatorul introduce date invalide sau inexistente
    a. Aplicația va semnala eroarea printr-un mesaj
    b. Utilizatorului i se va cere din nou introducerea locațiilor
2. Sistemul întâmpină o eroare
    a. Utilizatorul va fi înștiințat printr-un mesaj

4.4.2.2. Cazul de utilizare II

Numele cazului de utilizare: Vizualizare informații trafic
Actor principal: Utilizatorul
Precondiții:
1. Utilizatorul este pe pagina corespunzătoare
2. Utilizatorul a efectuat în prealabil cazul de utilizare I

Postcondiții:
1. Sistemul determină tweet-urile relevante condițiilor de trafic
2. Informațiile geolocalizate sunt afișate pe hartă

Scenariul principal (de succes):
1. Utilizatorul declanșează în mod deliberat determinarea tweet-urilor prin selectarea opțiunii aferente
2. Sistemul face vizibilă utilizatorului starea de încărcare a informațiilor
3. Sistemul obține și afișează informațiile despre condițiile de desfășurare a traficului și le plasează geolocalizat pe hartă sub forma unor iconițe Twitter în punctele principale de interes
4. Utilizatorul interacționează cu iconițele pentru a obține liste cu tweet-uri care conțin informații despre trafic

Scenarii alternative:
1. Sistemul întâmpină o eroare
    a. Utilizatorul va fi înștiințat printr-un mesaj
    b. Sistemul va sugera utilizatorului reîncercarea

4.4.2.3. Cazul de utilizare III

Numele cazului de utilizare: Raportează informațiile irelevante
Actor principal: Utilizatorul
Precondiții:
1. Utilizatorul este pe pagina corespunzătoare
2. Utilizatorul a efectuat în prealabil cazul de utilizare I
3. Utilizatorul a efectuat în prealabil cazul de utilizare II

Postcondiții:
1. Sistemul salvează identificatorul tweet-ului marcat ca fiind irelevant
2. Sistemul salvează numărul de raportări al identificatorului

Scenariul principal (de succes):
1. Utilizatorul deshide lista de tweet-uri interacționând cu iconița Twitter
2. Utilizatorul vizualizează și analizează lista
3. Utilizatorul marchează tweet-ul ca fiind irelevant folosind opțiunea aferentă

Scenarii alternative:
1. Sistemul întâmpină o eroare
    a. Utilizatorul va fi înștiințat printr-un mesaj
2. Tweet-ul a fost în prealabil marcat de același utilizator
    a. Sistemul va dezactiva posibilitatea de dezactivare pentru acel utilizator

4.4.2.4. Cazul de utilizare IV

Numele cazului de utilizare: Supervizare traducere
Actor principal: Administratorul de aplicație
Precondiții:

1. Administratorul este autentificat
2. Administratorul este pe pagina corespunzătoare

Postcondiții:

1. Sistemul salvează modificările administratorului asupra traducerilor
2. Sistemul actualizează schimbările în pagina curentă

Scenariul principal (de succes):

1. Administratorul selectează limba de traducere
2. Sistemul afișează lista traducerilor disponibile pentru limba selectată
3. Administratorul selectează traducerea ce trebuie actualizată
4. Sistemul afișează opțiunile de editare
5. Administratorul modifică și salvează traducerile
6. Sistemul actualizează informațiile existente

Scenarii alternative:

1. Sistemul întâmpină o eroare
    a. Administratorul va fi înștiințat printr-un mesaj
2. Nu există traduceri pentru o anumită limbă
    a. Sistemul va afișa un tabel fără intrări

4.4.2.5. Cazul de utilizare V

Numele cazului de utilizare: Supervizare rezultate marcate ca fiind irelevante
Actor principal: Administratorul de aplicație
Precondiții:

1. Administratorul este autentificat
2. Administratorul este pe pagina corespunzătoare

Postcondiții:

1. Sistemul salvează modificările administratorului asupra tweet-urilor
2. Sistemul actualizează schimbările în pagina curentă

Scenariul principal (de succes):

7. Sistemul afișează lista tweet-urilor marcate ca fiind irelevante și numărul de utilizatori care au raportat
8. Administratorul selectează tweet-ul dorit
9. Administratorul marchează tweet-ul ca fiind definitiv irelevant
10. Sistemul actualizează informațiile existente

Scenarii alternative:

3. Sistemul întâmpină o eroare
    a. Administratorul va fi înștiințat printr-un mesaj
4. Nu există raportări ale tweet-urilor
    a. Sistemul va afișa un tabel fără intrări

Capitolul 5. Proiectare de Detaliu si Implementare

În cadrul acestui capitol, se vor prezenta, în mod structurat, pașii și deciziile de proiectare considerate pe parcursul implementării aplicației propuse. În cadrul prezentărilor, se vor evidenția pattern-urile arhitecturale și de proiectare urmărite și se vor ilustra componentele care intră în arhitectura sistemului.

Aplicația finală constă în implementarea a trei subsisteme distribuite în cadrul arhitecturii Cloud menționate în capitolul anterior: aplicația web (frontend), serverul backend al aplicației web și baza de date.

5.1. Arhitectura aplicației web

5.1.1. Descriere generală. Șabloane de dezvoltare

În dezvoltea aplicației web frontend, principala tehnologie utilizată a fost Vue.js. Framework-ul javascript modern menționat a permis dezvoltarea în specificul șablonului arhitectural **MVVM** (Model-View-ViewModel). Modelul arhitectural generic în care acest aspect a fost urmărit este ilustrat în figura 5.1.

De asemenea, un alt aspect important în care tehnologia aleasă a avut un impact major este faptul că a permis utilizarea unei abordări bazate pe componente web reutilizabile, în specificul șablonului de proiectare **Composite**. Astfel, s-a reușit dezvoltarea unor componente generice care ulterior au putut fi reutilizate fără duplicarea codului existent.

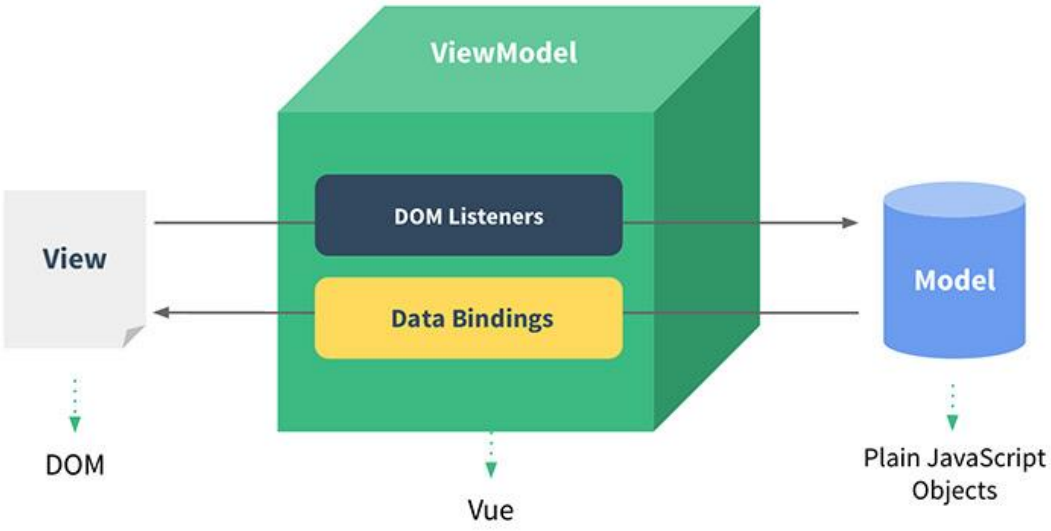

Figura 5.1 Modelul MVVM în Vue.js

5.1.2. Descrierea componentelor arhitecturii

În figura 5.2. este schițată arhitectura aplicației web dezvoltate. Se identifică componentele principale ale acesteia: componente, pagini, servicii, router-ul și punctul de intrare al aplicației. În cele ce urmează sunt explicate detaliat aceste componente și rolul lor în aplicația finală.

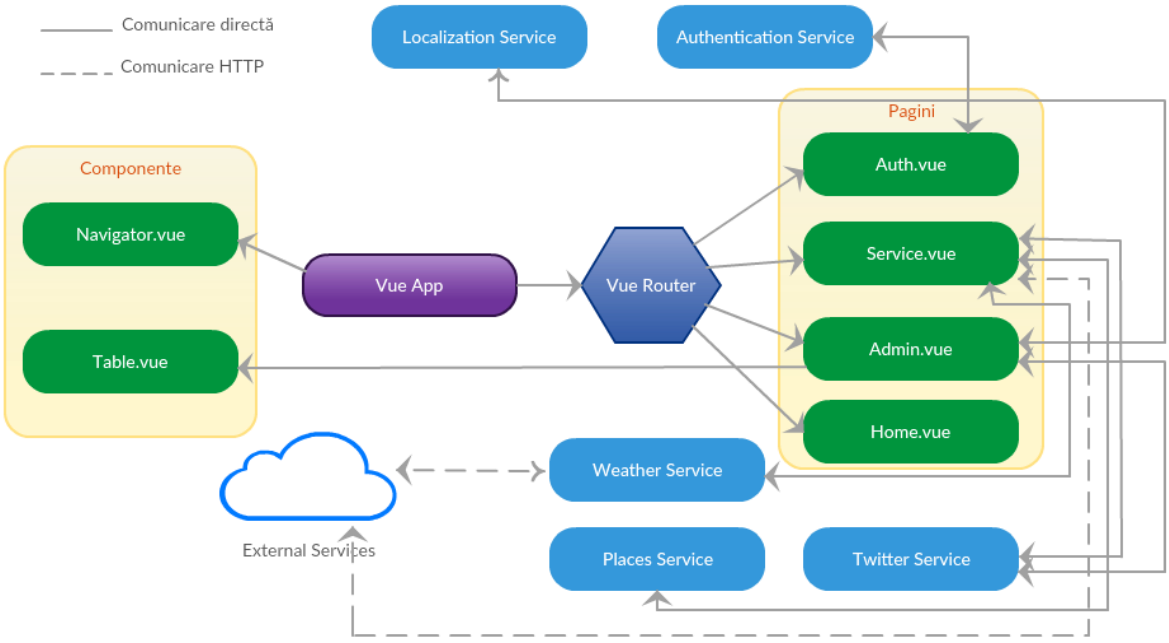

Figura 5.2 Arhitectura aplicației web

5.1.2.1. Componente

Orice instanță Vue este, în ansamblu, o componentă. Totuși, această denumire identifică, de cele mai multe ori, porțiuni de cod cu o funcționalitate desemnată care pot fi reutilizate în cadrul aplicației pentru a forma noi componente sau pagini fără a fi necesară duplicarea codului.

Un exemplu de componentă Vue a fost prezentat în Figura 4.4 din capitolul anterior. Într-o modalitate similară, componentele Navigator.vue și Table.vue definesc astfel de secțiuni de cod în care sunt implementate navigatorul vertical al aplicației, utilizat în punctul central de intrare (Vue App), vizibil din orice pagină accesată de utilizator, și un model de tabel generic, utilizat în cadrul paginii de administrare (Admin.vue).

Mai mult, pe lângă structura bine definită, partea de script a componentelor este de asemenea construită pe baza unui șablon. Aceasta conține secțiuni de proprietăți (props), date interne (data), funcții care se declanșează la modficarea datelor și ale proprietăților (watchers) și funcții ce țin de ciclul de viață al unei componente (created,

beforeMount, mounted, beforeUpdate, updated, beforeDestroy). Aceste aspecte duc la implementarea unui view model complex, dar ușor de manipulat.

Componentele pot forma, de asemenea, ierarhii complexe, ilustrând șablonul arhitectural Composite menționat anterior. Datele pot fi transmise între componente prin intermediul proprietăților, într-un mod asemănător transmiterii atributelor simple HTML, dar și prin evenimente și listeners.

În figura 5.3 sunt exemplificate ambele metode de transmitere a datelor între componente. Atributele columns și rows sunt transmise în mod direct, prin proprietăți, iar @selected definește un event listener pentru evenimentul "selected" definit în cadrul componentei CustomTable (Table.vue). Acestuia îi este atașată o funcție inline care preia informația emisă (row) și o atribuie unei date interne (selectedTranslation), definite în secțiunea de date interne (data).

```
<div>
  <custom-table :columns="translationColumns" :rows="translationRows"
    @selected="(row) => selectedTranslation = row"/>
</div>
```

Figura 5.3 Exemplu de moduri de transmitere a datelor între componente

5.1.2.2. Pagini. Vue Router.

Paginile din Vue sunt cazuri particulare de componente. Acestea respectă aceeași structură specifică de template html, stilizări css și script javascript, dar, în cadrul unei ierarhii de componente, este situată pe pozițiile superioare, însemnând că părintele direct al acesteia este însăși aplicația.

Gestiunea paginilor în funcție de URL-ul curent al utilizatorului este realizată prin intermediul Vue Router. Acesta, după cum este exemplificat și în figura 5.4, are definit un obiect cu numele, calea de acces și componenta aferentă fiecărei rute, pentru a determina ulterior în mod dinamic pagina de vizualizare care va fi afișată utilizatorului. Acesta este un element specific construcției SPA-urilor (Single Page Application), adică a aplicațiilor care au, în fapt, o singură pagină HTML și randează conținutul în mod dinamic în funcție de URL-ul aplicației.

```
10  export default new Router({
11    routes: [
12      {
13        path: '/',
14        name: 'Hello',
15        component: Hello
16      },
17      {
18        path: '/playground',
19        name: 'Playground',
20        component: Playground
21      }
22    ]
23  });
```

Figura 5.4 Exemplu definire rute utilizând VueRouter

5.1.2.3. Servicii

Serviciile din arhitectura aplicației au ca principală responsabilitate comunicarea cu API-urile serverului de back-end și ale serviciilor externe pentru a intermedia comunicarea dintre acestea și componentele care necesită informații sau procesări pe baza datelor transmise.

Un alt aspect important al serviciilor este faptul că delimitează, într-o manieră adaptată programării aplicațiilor web și utilizând un șablon de implementare de tip **arhictectură pe niveluri**, logica de business de viewmodel-urile aplicației frontend.

Pentru comunicarea serviciilor cu API-urile se utilizează client-ul HTTP axios, prezentat în secțiunea 4.2.2. Axios permite utilizarea unei comunicări asincrone și introducerea unui antet de autentificare, a cărei validitate poate fi ulterior verificată de server. Serviciile vor returna o promisiune care poate fi utilizată, apoi, de componenta care a apelat serviciul pentru a nu bloca execuția aplicației. Figura 5.5 ilustrează apelarea serviciului din cadrul unei componente și comunicarea autentificată cu serverul prin intermediul Localization Service pentru a actualiza o traducere automatizată.

Admin.vue

```
updateTranslation() {
  this.$emit('load-changed', true);
  localizationService.updateTranslation(this.selectedTranslation).then(() => {
    this.$emit('load-changed', false);
  });
},
```

LocalizationService.js

```
updateTranslation(translation) {
  return axios.put(LOCALIZATION_SERVICE_UPDATE_TRANSLATION_PATH, translation, {
    headers: {
      Authorization: sessionStorage.getItem('auth-token')
    }
  });
}
```

Figura 5.5 Comunicarea prin servicii între Front-end și Back-end

5.1.2.4. Servicii externe

Serviciile externe sunt acele API-uri utilizate în determinarea rutelor disponibile între locații și în obținerea informațiilor despre condițiile meteorologice.

Concret, reprezintă comunicarea cu serviciile aferente Google Maps API, descrise în secțiunea 4.2.4, și serviciile meteorologice oferite de Open Weather Maps, descrise în 4.2.7. Comunicarea cu acestea se va realiza, de asemenea, într-o manieră asincronă, pentru a asigura faptul că aplicația rămâne funcțională în timpul schimbului de informații.

5.1.2.5. Vue App (App.vue)

App.vue este punctul central în cadrul unei aplicații Vue.js. Practic, aceasta poate fi considerată rădăcina arborelui de componente, unde se definește fișierul HTML care va fi exportat în urma operației de build a aplicației și pe care framework-ul îl va schimba în mod adaptiv prin intermediul script-urilor și al router-ului.

Componenta conține elemente vizibile în cadrul tuturor ecranelor de vizualizare a aplicației (de exemplu, elemente de template cum ar fi Navigator.vue, ce definește navigatorul global), dar și stilizări și script-uri valabile pentru toată aplicația.

5.1.3. Funcționalitate și algoritmi

În cadrul acestei subsecțiuni va fi descris modul de funcționare al unor aspecte importante ce țin de logica aplicației frontend și a algoritmilor utilizați pentru implementarea funcționalităților necesare.

Vor fi descrise, prin urmare, metoda de determinare a rutelor prin intermediul Google Directions Service, algoritmul de determinare a punctelor de interes, obținerea informațiilor meteorologice, obținerea tweet-urilor, modul de implementare al iconițelor interactive, metoda de verificare a autentificării și autorizării pentru accesarea paginilor restricționate.

5.1.3.1. Derminarea rutelor utilizând Directions Service API

Pentru determinarea rutelor (a rutei principale și a rutelor alternative) dintre locațiile specificate de utilizator, se utilizează Google Directions Service API, disponibil în forma obiectului DirectionsService, ce poate fi inițializat la încărcarea paginii, a aplicației sau chiar la apelarea metodei.

Obiectul DirectionsService utilizează metoda route() pentru a iniția un request către serviciul amintit. Acesta poate fi configurat cu un obiect de tipul DirectionsRequest, care conține informațiile ce descriu request-ul efectuat. De asemenea, obiectului i se poate desemna o funcție callback care să fie executată cu datele despre răspuns și statusul request-ului în urmă finalizării request-ului.

Parametri obiectului DirectionsRequest utilizați în aplicație sunt:

- origin – locația sursă a navigării
- destination – locația destinație a navigării
- provideRouteAlternatives – setat true sau false în funcție de dorința de a obține sau nu rute alternative (în caz contrat, se va primi o singură rută ca răspuns)
- travelMode – mijlocul de călătorie (mașină/bicicletă).

În urma efectuării request-ului cu parametrii menționați, aplicația va recepționa prin funcția callback răspunsul prin intermediul unui obiect de tipul DirectionsResponse. Proprietățile importante ale acestui obiect sunt:

- geocoded_waypoints[] - un array care conține locația sursă și locația destinație geocodificate
- routes[] - un array care conține obiecte de tipul DirectionsRoute. În cadrul acestor obiecte există indicații de a ajunge din punctul de origine în punctul destinație, atât sub formă de pași (prin explicații și coordonate), cât si printr-o serie de coordonate geografice a punctelor intermediare.

Următorul pas important, în urma obținerii obiectelor DirectionsResponse, este afișarea lor pe hartă. Această acțiune poate fi realizată cu ajutorul obiectului DirectionsRenderer oferit de același API, în 3 pași simpli:

1. Inițializarea unui obiect DirectionsRenderer
2. Atașarea hărții prin apelul parametrizat al metodei setMap()

3. Atașarea direcțiilor DirectionsResponse prin apelul parametrizat al metodei setDirections()

În urma efectuării acestor pași, pe harta specificată va apărea ruta aferentă navigării din locația sursă în locația destinație și informațiile vor fi stocate pentru procesarea de la punctele următoare. Este important de menționat că pentru afișarea fiecarei rute trebuie creat un nou obiect DirectionsRenderer.

5.1.3.2. Determinarea punctelor de interes

Această etapă are ca scop identificarea acelor puncte geografice de pe rutele obținute care au o anumită importanță administrativă (sunt orașe sau localități cu o populație semnificativă), întrucât în aceste puncte sunt concentrate majoritatea tweet-urilor cu informații despre trafic.

O rută are, în medie, între 100-200 de puncte geografice geocodificate în cadrul obiectelor DirectionsRoute, iar aplicarea unei procesări pentru fiecare dintre aceste puncte ar fi nu doar imposibilă, dar și nejustificată. Din acest motiv, acest pas are o deosebită importanță pentru relevanța rezultatelor obținute din Twitter.

Ideea de bază a algoritmului de determinare a punctelor de interes este împărțirea segmentului total de parcurs într-un număr fix de segmente (denumit, în continuare, threshold) pentru care, ulterior, se vor obține coordonatele fiecărui punct delimitator de segmente și se va scana aria geografică de o dimensiune setată ca parametru, pentru determinarea celui mai important punct din zonă, utilizând Google Ranking Algorithm, integrat în Google Nearby Search. Rezultatele obținute se vor persista, ulterior, prin intermediul server-ului backend pentru a asigura o eficiență în ceea ce privește timpul de răspuns și traficul utilizat pentru apelarea serviciilor exterioare.

Figura 5.6 ilustrează ideea prezentată. În a), vectorul de puncte geografice este reprezentat sub forma unui segment de dreaptă. În b) este ilustrată segmentarea în 3 puncte a vectorului și scanarea ariei geografice pentru marginile segmentelor.

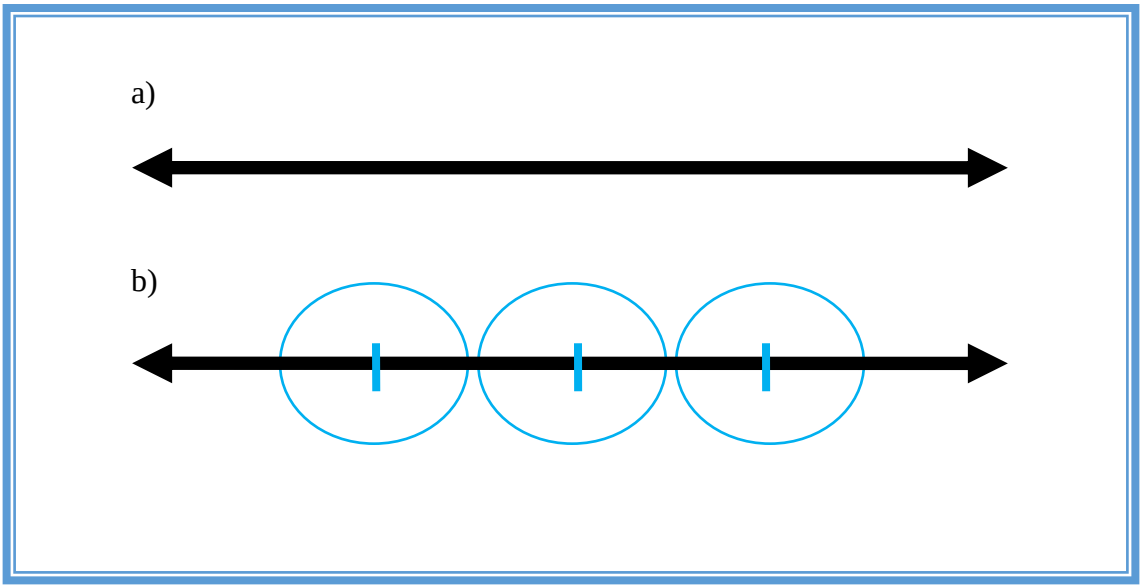

Figura 5.6 Metoda de segmentare a unui vector de coordonate pentru scanarea ariilor geografice

În figura 5.7 este prezentat codul sursă aferent algoritmului prezentat. Se pot observa pașii logici urmați de acesta: întâi are loc o segmentare în 6 puncte, apoi sunt obținute punctele geografice pentru punctele geografice obținute. Ulterior, se încearcă obținerea locațiilor importante din backend-ul aplicației, prin intermediul serviciului PlacesService, iar, în cazul în care serverul nu a returnat o locație pentru coordonatele date, se va utiliza NearbySearch pe o rază de 30km, urmărind așezări urbane de tip localitate. Google Ranking Algorithm va ordona rezultatele în funcție de popularitate, iar drept urmare primul element din lista returnată va fi cel mai important punct de interes din zona scanată. În final, se va realiza persistarea datelor obținute pentru a nu repeta aceiași pași în viitor.

```
const threshold = parseInt(response.routes[i].overview_path.length / 6, 10);
for (let j = 1; j * threshold < response.routes[i].overview_path.length; j++) {
  const lat = response.routes[i].overview_path[j * threshold].lat();
  const long = response.routes[i].overview_path[j * threshold].lng();
  const LatLng = { lat, lng: long };
  placesService.getPlace(lat, long)
  .then((resp) => {
    if (!resp.data) {
      // 'Getting place from an API'
      locService.nearbySearch({
        location: LatLng,
        radius: 30000,
        type: 'locality'
      }, (locResponse, locStatus) => {
        const city = {
          lat: locResponse[0].geometry.location.lat(),
          lng: locResponse[0].geometry.location.lng(),
          name: locResponse[0].name
        };
        this.persistLocation(LatLng, 30000, city);
        this.handleLocationResponse(city, status, i, infowindow);
      });
    } else {
      this.handleLocationResponse(resp.data, status, i, infowindow);
    }
  }).catch((error) => {
```

Figura 5.7 Implementarea JavaScript a identificării punctelor de interes

5.1.3.3. *Obținerea informațiilor meteo*

O primă funcționalitate în care este observată utilitatea determinării punctelor de interes este obținerea informațiilor meteo. Orașele descoperite sunt salvate într-un vector, pentru care se poate apela serviciul de vreme (WeatherService), ce ulterior va apela API-ul OpenWeatherMap.

Astfel, după determinarea acestora, pentru fiecare oraș descoperit, se va realiza, în speță, un request extern pentru obținerea informațiilor legate de condițiile meteorologice.

5.1.3.4. Obținerea tweet-urilor

Într-o factură similară punctului anterior, vectorul de orașe va fi parcurs și, pentru fiecare oraș identificat, se va realiza un request, prin intermediul TwitterService, către backend-ul aplicației.

După obținerea rezultatelor, acestea vor fi prelucrate și afișate pe hartă sub forma unei liste activabile la apăsarea unei iconițe plasate peste coordonatele de unde au fost revendicate tweet-urile.

Modalitatea în care s-a realizat definirea plasarea, conținutul și comportamentul iconițelor va fi explicată în secțiunea următoare.

5.1.3.5. Crearea iconițelor. Conținut și comportament

Având în vedere faptul că pentru afișarea hărții s-a utilizat Google Maps API, o soluție facilă de afișare a informațiilor sub forma unor iconițe interactive pe hartă este utilizarea Google Markers API. Soluția nu necesită utilizarea unor librării adiționale și permite utilizarea HTML pentru afișarea conținutului.

Mai mult, permite integrarea unor action listeners pentru definirea comportamentului la interacțiunea cu utilizatorul, setarea unor iconițe particularizate și animarea facilă.

În cazul de față, se dorește afișarea iconițelor meteo în funcție de starea vremii și a iconițelor Twitter, iar la interacțiunea utilizatorului cu acestea, deschiderea unei ferestre cu informații detaliate, respectiv cu o listă de tweet-uri.

Implementarea comportementului dorit se poate realiza în două etape, ilustrate și în figura 5.8: crearea Marker-ului cu detaliile specifice și asignarea unui listener pentru evenimentul ‘click’, pentru care se va deschide o fereastră cu un conținut HTML specificat.

```
createMarker(lat, lng, icon, map) {
  return new google.maps.Marker({
    position: { lat, lng },
    animation: google.maps.Animation.DROP,
    icon,
    map
  });
},
addInfoWindowListener(marker, infoWindow, stringContent, map) {
  marker.addListener('click', () => {
    infoWindow.setContent(stringContent);
    infoWindow.open(map, marker);
  });
},
```

Figura 5.8 Metode pentru crearea de Markers și determinarea comportamentului la click

5.1.3.6. Autentificare. Restricționarea paginilor

Un alt aspect important al aplicației este securizarea paginilor la care utilizatorul simplu nu ar trebui să aibă acces. Pentru implementarea acestei funcționalități, trebuie să ne asigurăm că doar un utilizator care a fost în prealabil autentificat poate accesa anumite URL-uri ale aplicației.

Sistemul de autentificare va fi discutat în detaliu în cadrul prezentării arhitecturii backend-ului, dar în secțiunea curentă se va prezenta comunicarea cu acesta, modalitatea în care este reținută sesiunea și mijloacele de restricționare ale utilizatorulor neautentificați.

Comunicarea se va realiza prin intermediul serviciului de autentificare (Authentication Service), care va transmite request-uri HTTP POST asincrone serverului backend pentru obținerea unui token de autentificare ce va fi ulterior persistat în session storage-ul browserului. În cazul introducerii unor date de utilizator invalide, se va afișa un mesaj care va indica acest fapt.

Mai mult, request-urile către endpoint-urile securizate (cele necesare pentru efectuarea sarcinilor administratorului de aplicație), token-ul va fi transmis împreună cu request-ul în câmpul de antet, iar cererea se va realiza doar dacă acesta este valid.

Pentru navigarea către pagina de administrator (Admin.vue), se va utiliza funcționalitatea Vue Router care interceptează navigarea pentru verificarea existenței token-ului, ilustrată și în exemplul din figura 5.9. În caz contrar, utilizatorul va fi redirecționat către pagina de autentificare (Auth.vue).

```
router.beforeEach((to, from, next) => {
  if (to.path === '/admin' && !sessionStorage.getItem('auth-token')) {
    next('/auth');
  } else {
    next();
  }
});
```

Figura 5.9 Interceptarea navigării prin metoda Vue Router beforeEach

5.1.4. Elemente de responsive web design și user experience (UX)

Responsive Web Design reprezintă abordarea de a construi aplicații web care să își poată adapta conținutul în funcție de dimensiunea ecranului browser-ului sau a dispozitivului prin intermediul căreia este vizualizată pagina.

Conceptul de User Experience se referă la percepțiile și atitudinea unui utilizator în timpul utilizării unui produs software.

În cadrul aplicației propuse, după cum a fost definit și în cadrul obiectivului secundar prezentat în secțiunea 2.2.4, experiența utilizatorului care utilizează sistemul prin intermediul unui dispozitiv mobil (dar nu numai) posedă o importanță semnificativă.

5.1.4.1. Grid Layout & Media Queries

Cel mai important pas în construirea unei interfețe grafice responsive este definirea unui layout adaptiv al paginii. Acest scop poate fi atins cu ușurință utilizând sistemul de împărțire a paginii într-un grid (grilă) cu rânduri și coloane, oferit de

framework-ul Bootstrap[28]. Ideea din spatele acestui sistem este de a defini elemente HTML care să ocupe un număr predefinit de coloane (dintr-un total de 12 coloane posibile) în funcție de rezoluția paginii prin asignarea unor clase specifice de forma col-X-Y, unde X reprezintă dimensiunea codificată a ecranului, iar Y numărul de coloane ocupate pentru dimensiunea X.

Posibilitățile pentru definirea ecranului (X), sunt xs (telefoane), sm (tablete), md (desktops), lg & xl (desktop-uri cu rezoluții mari). În figura 5.10 este prezentată aplicarea acestora într-un grid layout ce va poziționa harta definită pe tot ecranul – 12 coloane - pentru dispozitive mici (telefoane și tablete), și pe 66.67% din dimensiunea ecranului – 8 din 12 coloane – pentru dispozitivele mari.

```
<div class="map-container col-xs-12 col-sm-12 col-md-8 col-lg-8 col-xl-8">
  <div id="map-canvas" class="map"></div>
</div>
```

Figura 5.10 Utilizarea claselor responsive predefinite pentru poziționarea hărții

Un alt pas important urmat este definirea stilurilor particularizate pe dimensiunea ecranelor, care se poate realiza utilizând “media queries” din cadrul aceluiași framework. Acestea sunt adnotări care determină dimensiunea și tipul ecranului pentru care se va aplica o clasă de stilizare CSS.

5.1.4.2. Afișarea stării de încărcare a informației

Un aspect important pentru a spori utilizabilitatea aplicației și a îmbunătăți experiența generală a utilizatorului este de a marca grafic prin animații momentele în care aplicația este în lucru. Această metodă convinge utilizatorul că aplicația a preluat comanda lui și este în curs de procesare a acesteia.

Mai mult, implementarea acestei funcționalități previne într-un mod plăcut utilizatorul din apăsarea repetată a butoanelor sau din efectuarea simultană de multiple operațiuni costisitoare din punct de vedere computațional.

Implementarea efectivă se poate realiza dezvoltând o componentă animată prin intermediul CSS care să marcheze acțiunea de încărcare. Paginile, componentele și serviciile care utilizează procesări a căror durată depășește ordinul sutelor de milisecunde pot utiliza această componentă grafică prin intermediul evenimentelor: se vor transmite evenimente de declansare a stării de încărcare și de încheiere a acesteia.

În cazul întâmpinării unor erori, starea de încărcare va fi, de asemenea, marcată ca fiind încheiată.

[28] https://getbootstrap.com/

5.2. Arhitectura backend

5.2.1. Descriere generală. Șabloane de implementare

În cadrul dezvoltării aplicației backend, principalele tehnologii utilizate fac parte din gama de module Spring, descrisă în secțiunea 4.2.8. Modelul arhitectural a fost implementat urmărind structura conceptuală descrisă în Figura 4.3.

Drept urmare, în implementarea conceptului descris, s-au utilizat ca șabloane arhitecturale și de implementare paradigma **arhitecturii pe niveluri**, delimitând nivelurile de prezentare (HTTP endpoints) de cele de business logic și de acces la date.

De asemenea, tehnologiile alese au avut un impact favorabil în implementarea unor șabloane de implementare caracteristice acestora. Framework-ul Spring preia controlul asupra inițializării entităților definite ca Spring Beans, implementând cu succes, într-o manieră caracteristică, șabloanele **Inversion of Control** (IoC) și **Dependency Injection**.

Un alt șablon arhitectural considerat în implementarea aplicației sunt **MVC** (Model-View-Controller), aplicat într-un mediu de dezvoltare RESTful, unde în loc de ecrane de vizualizare (views) se lucrează cu răspunsuri HTTP pentru construirea dinamică de views în cadrul aplicației frontend. De asemenea, pentru desemnarea controllerelor pentru căi de acces specifice, Spring implementează în cadrul aplicației șablonul **Front Controller**.

În figura 5.11 este prezentată diagrama UML de pachete aferentă implementării conceptului descris în arhitectura conceptuală și în cadrul acestei secțiuni. În continuare, vor fi descrise componentele din această figură, atât la nivel conceptual, cât și în cadrul unei diagrame de clase.

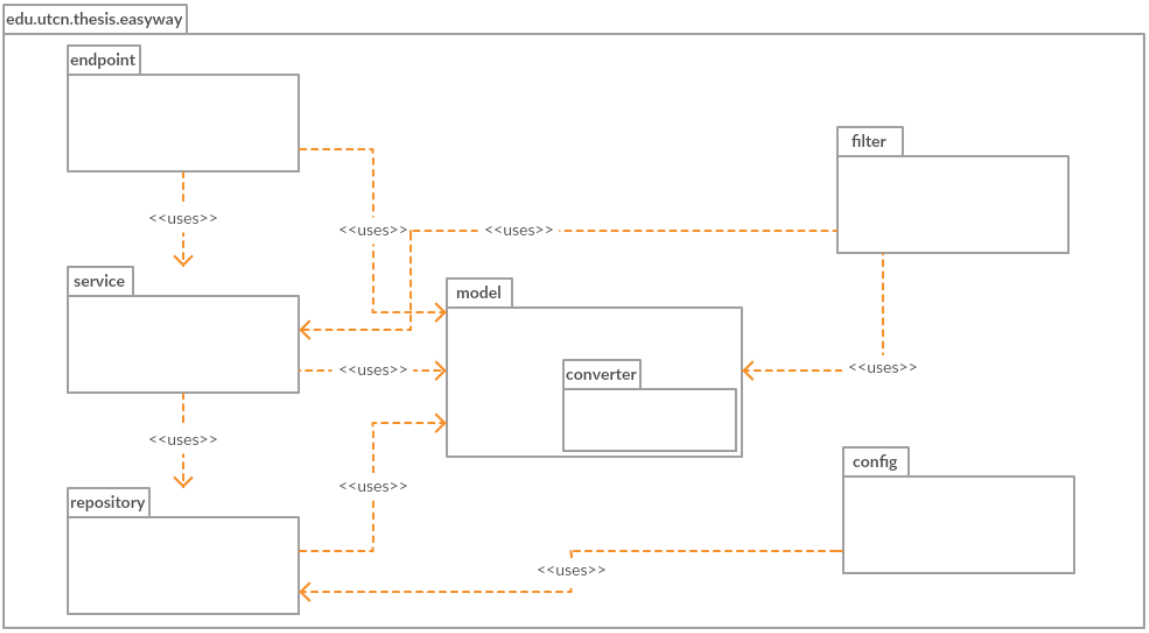

Figura 5.11 Diagrama UML de pachete a aplicației backend

5.2.2. Descrierea componentelor arhitecturii

În cadrul figurii 5.11 menționate anterior este descrisă, practic, arhitectura pe componente a sistemului și comunicarea dintre acestea.

În cadrul acesteia, se identifică următoarele componente și dependințe:

- Endpoints – comunică direct cu nivelul inferior de servicii și cu modelul aplicației
- Services – are dependințe directe la nivelul inferior, de acces la date, și la modelul aplicației
- Repositories – utilizează modelul de date al aplicației pentru a accesa baza de date
- Model – definește modelul de date al aplicației. Utilizează convertoare pentru a transforma anumite structuri de date ierarhice în structuri persistate în baza de date
- Filter – comunică cu serviciile și modelul pentru a filtra accesul din exterior
- Config – comunică cu nivelul de acces la date pentru a asigura anumite configurări necesare fucționării aplicației

În continuare, vor fi descrise în amănunt aceste componente. Descrierea claselor cuprinse de acestea (mai puțin a pachetelor Filter și Config) este ilustrată în cadrul diagramei de clase din Figura A.1 din anexa 4.

5.2.2.1. Endpoints

Pachetul de endpoints conține clase menite să expună funcționalitățile API-ului implementat în serverul de backend. Constituie nivelul de prezentare al aplicației back-end și definește căile de acces din exterior pentru aplicație.

Pentru atingerea acestor obiective, există o serie de secvențe de cod specifice, utilizabile prin intermediul adnotărilor.

La nivel de clasă, se identifică următoarele adnotări utile în definirea comportamentului dorit al unui endpoint:

1. **@RestController** – definește, la nivel de clasă, un controller RESTful pentru a semnala Dispatcher Servlet-ului existența unor mapări cale – metodă existente în cadrul acestei clase. Practic, definește metode care să fie executate la apelul metodelor HTTP de pe anumite URL-uri.
2. **CrossOrigin** – definește originile host-urilor de unde se permit request-uri. Utilizarea acestei adnotări simplifică procesul de admitere a anumitor host-uri diferite de originea serverului, care, în mod normal, s-ar fi realizat prin setarea antetelor HTTP.

La nivel de metode, adnotarea **@RequestMapping** definește modalitatea de mapare a request-ului. Concret, cu ajutorul acesteia se poate defini valoarea URL-ului pentru care metoda va fi apelată, tipul de metodă HTTP, tipul de conținut produs în răspunsul trimis ș.a.

Rolul acestui nivel de clase este de a intercepta diferitele cereri venite de la aplicația frontend și de a le delega mai departe nivelului de serviciu pentru execuție. În

urma execuției, endpoint-urile au responsabilitatea de a forma și transmite răspunsul HTTP aplicației.

În figura 5.12 este exemplificat, prin intermediul unei secțiuni de cod care ar putea intercepta procesul de actualizare a unei traduceri, atât rolul claselor din nivelul endpoints, cât și structura unei astfel de clase ce utilizează mecanismele Spring specifice explicate în cadrul paragrafelor anterioare.

```
@RestController
@CrossOrigin
public class LocalizationEndpoint {

    @Autowired
    private LocalizationService localizationService;

    @RequestMapping(value = "/admin/localization/update", method = RequestMethod.PUT)
    public Translation updateTranslation(@RequestBody Translation translation) {
        return localizationService.updateTranslation(translation);
    }

}
```

Figura 5.12 Exemplu Endpoint RESTful în Spring MVC

În această secțiune a aplicației, sunt necesare definirea de astfel de controllere pentru cererile care țin de localizare și internaționalizare, de operații pe tweet-uri și de obținerea sau persistanța locațiilor. Astfel, se vor implementa:

- LocalizationEndpoint
- TwitterEndpoint
- PlacesEndpoint

5.2.2.2. Servicii

Această parte din arhitectura sistemului este responsabilă cu logica de business a aplicației. Va prelua cererile venite din nivelul superior, cel al endpoint-urilor, și va utiliza modelul de date și nivelul de acces la date, precum și celelalte servicii disponibile, dar și servicii suplimentare și logică adițională, pentru îndeplinirea cererilor efectuate de utilizator.

Pentru definirea claselor de tip serviciu ca Spring Beans se poate utiliza adnotarea **@Service**. Adnotarea face ca repsectiva clasă să fie descoperită de scanarea Spring a bean-urilor, pentru a putea fi ulterior injectată în alte clase pentru inițializarea caracteristică Inversion of Control, dar rezumă și specificul acesteia, marcând faptul că implementarea clasei contribuie la definirea unui serviciu.

Comunicarea directă cu nivelul inferior se va realiza prin intermediul injectării dependințelor utilizând adnotarea **@Autowired**. Comunicarea cu serviciile externe se va realiza utilizând un client Java HTTP (OkHTTP).

În acest nivel se vor utiliza servicii externe de traducere automatizată (Yandex, prezentat în secțiunea 4.2.6) și servicii de geolocalizare pentru obținerea informațiilor legate de contextul lingvistic al unei locații reprezentate prin coordonate.

De asemenea, acest nivel este responsabil pentru obținerea, procesarea și filtrarea rezultatelor obținute din Twitter.

Serviciile vor fi definite în clase în funcție de categoria lor. Astfel, vom implementa LocalizationService, TwitterService și TokenAuthenticationService.

5.2.2.3. Repositories

Acest pachet conține implementarea nivelului de acces la date pentru aplicația propusă. Aici se vor defini operațiile posibile pentru citirea, crearea, actualizarea și ștergerea datelor din baza de date.

Utilizând Spring Data, prezentat în secțiunea 4.2.8.3, un modul caracteristic Amazon DynamoDB pentru Spring Data și a SDK-ului AWS, se pot defini interfețe care să implementeze în mod automat operațiile dorite pe baza de date.

Pașii necesari automatizării acestui proces cuprind:

1. Adnotarea entităților de model
2. Crearea interfețelor specifice Spring Data
3. Configurarea DynamoDB utilizând SDK-ul AWS

Primul pas presupune utilizarea unor adnotări specifice modulului Spring Data pentru DynamoDB pentru definirea numelui tabelului peste clasa implementată (@DynamoDBTable), a hash key-ului aferent fiecărei entități peste identificatorul clasei (@DynamoDBHashKey, @DynamoDBAutoGeneratedKey) și a atributelor fiecărui tabel (@DynamoDBAttribute) peste proprietățile clasei expuse în tabel.

Următorul pas presupune crearea de interfețe marcate cu adnotarea @Repository, care să extindă interfața predefinită CrudRepository<ID, T>, unde ID reprezintă tipul identificatorului utilizat, iar T tipul de clasă al entității pentru care este creat repository-ul. În urma acestor pași, entitatea va beneficia de operațiile normale CRUD implementate în mod generic de interfața extinsă. Pentru a crea noi metode de acces, se poate utiliza, de exemplu, șablonul de nume al metodei findByAttr1AndAttr2, unde Attr1 și Attr2 reprezintă denumirea proprietăților din entitate după care se dorește efectuarea operației de căutare (find). Un exemplu de astfel de interfață cu metode parametrizate este ilustrat în figura 5.13.

```
@Repository
@EnableScan
public interface CityRepository extends CrudRepository<City, String> {

    List<City> findByName(@Param("name")String name);

    List<City> findByLatAndLng(@Param("lat")Double lat, @Param("lng") Double lng);
}
```

Figura 5.13 Implementarea interfeței Repository pentru entitatea City

Ultimul pas implică crearea unei clase de configurare pentru serviciul cloud DynamoDB prin care se vor transmite datele de autentificare la serviciu și adresa asignată bazei de date create.

În cadrul aplicației, este necesară implementarea interfețelor de tip Repository pentru manipularea entităților City, NearbyResult, Translation, TweetSample și User.

5.2.2.4. Model

Această secțiune reprezintă modelul de date al aplicației. Aici va fi specificată structura entităților necesare pentru buna funcționare a sistemului și va fi definit un

subpachet de convertoare necesar pentru a transforma anumite structuri complexe pentru comunicarea acestora către și de la baza de date.

Dintre aceste entități, cele care necesită persistență cu baza de date vor fi adnotate corespunzător, după cum s-a explicat în secțiunea anterioară.

Clasele care vor fi implementate în această secțiune sunt:

- City – definește structura unui obiect de tip oraș, care conține informații despre denumirea, latitudinea, longitudinea și limba acestuia.
- NearbyResult – definește rezultatul unei scanări geolocalizate din cadrul unei arii geografice (cel mai important oraș dintr-o zonă scanată). Conține informații despre datele originale ale scanării (latitudine, longitudine, arie) și despre orașul identificat.
- Translation – descrie obiectul care persistă traducerile automatizate. Cuprinde proprietăți care identifică limba din care s-a efectuat traducerea și limba în care a fost tradus textul, mesajul original tradus și rezultatul traducerii.
- TweetSample – reprezintă o mostră a informațiilor disponibile într-un Tweet. Conține doar câmpurile necesare, cum ar fi identificatorul tweet-ului, data când acesta a fost creat, orașul geolocalizat aferent acestuia, numărul de raportări a irelevanței și dacă a fost sau nu validat de un administrator ca fiind definitiv irelevant. De asemenea, conține un atribut timeToLive, care definește durata de viață a entității, adică intervalul de timp în care acesta va fi stocat în baza de date.
- TweetWrapper – este un wrapper în jurul obiectului Twitter provenit din API care adaugă un identificator String pentru transmiterea acestuia ca răspuns către aplicația frontend.
- User – definește datele de logare ale administratorului de aplicație.

5.2.2.5. Filter

În cadrul acestui pachet sunt definite filtrele de navigare ale aplicației backend. Practic, acestea vor intercepta orice comunicare din exterior cu serverul și o vor valida sau o vor modifica pentru asigurarea unor funcționalități.

Filtrele au o deosebită importanță în implementarea funcționalității de autentificare și autorizare. În acest scop, se vor implementa două filtre. Un prim filtru de logare care va valida corectitudinea datelor de autentificare pentru path-ul de login al aplicației și va crea o intrare în tabelul de autentificare din memorie. Cel de-al doilea filtru va intercepta toate request-urile și va verifica daca path-ul requestului este unul restricționat (necesită privilegii de administrare) și, în caz afirmativ, dacă utilizatorul care a făcut request-ul deține un token de autentificare.

5.2.2.6. Config

Acest pachet găzduiește configurațiile Spring, DynamoDB și Spring Security. Prin intermediul acestor clase se va realiza conexiunea cu endpoint-ul Amazon DynamoDB, se vor specifica filtrele aferente căilor de acces și se va specifica sistemului de autentificare entitatea de utilizator pe care acesta trebuie să o valideze.

5.2.3. Funcționalitate și algoritmi

În cadrul acestei subsecțiuni va fi descris modul de funcționare al unor aspecte importante ce țin de logica aplicației backend și a algoritmilor utilizați pentru implementarea funcționalităților necesare.

În cele ce urmează, vor fi descrise implementarea subsistemului de căutare, căutare adaptată la contextul lingvistic și marcare a tweet-urilor, a funcționalității de salvare și obținere a locațiilor importante și a sistemului de autentificare.

5.2.3.1. Căutarea Tweet-urilor. Traducere. Raportare.

Subsistemul descris în secțiunea curentă are o deosebită importanță, întrucât are cel mai mare impact asupra realizării obiectivului final.

Dintr-o perspectivă bottom-up a implementării, primul pas din cadrul implementării este definirea structurii entităților necesare și a metodelor necesare de acces la date.

Fiind un subserviciu complex, acesta utilizează majoritatea entităților de model prezentate în secțiunea 5.2.2.4. În ceea ce privește persistența datelor, subsistemul necesită posibilitatea de a opera baza de date pentru entitățile de TweetSample și Translation.

Drept urmare, pentru cele două entități s-au creat interfețele repository TweetSampleRepository<TweetSample, String> și TranslationRepository<Translation, String>, în care s-au definit metode parametrizate pentru accesul la tweet-uri după identificator și pentru obținerea traducerilor în funcție de limba mesajului original, limba în care a fost tradus și textul mesajului.

În continuare, cel mai important aspect al implementării este definirea serviciilor necesare. Serviciul utilizat pentru funcționalitatea elementară este TwitterService. La inițializarea acestuia, se vor defini o serie de cuvinte cheie după care se va efectua o interogare, după sintaxa prezentată în Tabel 4.1, pentru obținerea tweet-urilor relevante. Tabelul 5.1 prezintă lista de cuvinte cheie utilizate în formarea interogării amintite. Aceasta conține cuvinte în engleză (denumite “query tokens”), ce vor putea fi ulterior traduse în formarea de interogări internaționalizate.

Tabel 5.1 Cuvinte cheie urmărite în identificarea tweet-urilor relevante

Cuvinte cheie
traffic accident
road accident
slow traffic
accident
traffic congestion
traffic jam

După inițializarea listei de query tokens, se definește metoda de căutare geolocalizată a tweet-urilor relevante, în funcție de parametri transmiși ca latitudine și lognitudine.

Un prim pas este completarea cuvintelor cheie cu traducerea acestora din limba țării în care este efectuată căutarea, dacă limba oficială a țării respective este diferită de

cea originală (engleză). Acest procedeu va fi descris ulterior, în cadrul prezentării LocalizationService, iar, pe moment, se consideră ca prin intermediul acestuia se va returna o listă de query tokens traduși.

După obținerea cuvintelor cheie, se formează interogarea necesară căutării mesaejlor Twitter. Se utilizează o tehnică simplă de înlănțuire a tokenilor în două interogări diferite (cea formată din cuvintele inițiale și cea formată din cuvintele traduse) prin intermediul operatorului SAU.

În apelarea serviciului REST al Twitter, se utilizează framework-ul Twitter4J[29], o implementare Java a unei interfețe de acces prin protocoale HTTP a API-urilor Twitter. Pentru fiecare rezultat obținut, se verifică ca acesta să nu fie marcat definitiv ca fiind irelevant și este adăugat în lista finală sortată descrescător în funcție de data la care tweet-ul a fost creat returnată utilizatorului.

Secvența de cod vizibilă în figura 5.15 ilustrează implementarea Java, utilizând Twitter4j, a metodei descrise. Metoda getFragmentedListOfQueries este o metodă privată care formează interogarea pe baza listei de query tokens după formatul menționat anterior iar. O altă metodă privată a TwitterService este checkIfIrrelevant, care verifică dacă tweet-ul curent se află marcat în baza de date ca fiind irelevant și returnează o valoare booleană corespunzătoare.

```
public List<TweetWrapper> getTweetsForLocation(Double lat, Double lng) throws TwitterException {
    List<TweetWrapper> tweets = new ArrayList<>();
    List<String> currentQueryTokens = new ArrayList<>();
    List<String> il8nTokens = localizationService.getTranslatedTokens(queryTokens, lat, lng);
    currentQueryTokens.addAll(queryTokens);
    currentQueryTokens.addAll(il8nTokens);

    List<Query> twitterQueries = getFragmentedListOfQueries(currentQueryTokens);

    for (Query query : twitterQueries) {
        query.setGeoCode(new GeoLocation(lat, lng), radius: 30, Query.KILOMETERS);
        QueryResult result = twitter.search(query);
        for (Status status : result.getTweets()) {
            TweetWrapper tw = new TweetWrapper(status, String.valueOf(status.getId()));
            if (!tweets.contains(tw) && !checkIfIrrelevant(tw.getId())) {
                tweets.add(tw);
            }
        }
    }

    tweets.sort(Collections.reverseOrder());
    return tweets;
}
```

Figura 5.14 Căutarea tweet-urilor utilizând query tokens și Twitter4J

În ceea ce privește **traducerea** automatizată, în LocalizationService s-au implementat o serie de metode necesare traducerii unui set de cuvinte și/sau expresii în funcție de coordonatele grafice ale geolocalizării.

În primul rând, pentru realizarea funcționalității menționate, se identifică orașul aferent coordonatelor specificate. Având în vedere faptul că analiza de tweet-uri este efectuată mereu ulterior obținerii punctelor de interes, mereu se va putea identifica orașul corespunzător coordonatelor metodei.

[29] http://twitter4j.org/en/

Următorul pas este crearea unei traducerea efectivă a termenilor și returnarea listei formate din aceștia. În cazul traducerii, există două posibile cazuri:

1. Traducerea se regăsește în baza de date, aceasta este preluată și transmisă mai departe
2. Traducerea nu există în baza de date, este necesară obținerea ei

În cel de-al doilea caz enumerat, serviciul va crea un request HTTP GET către API-ul Yandex Translate, prezentat în secțiunea 4.2.6. În urma obținerii rezultatelor traducerii, acestea vor fi persistate în baza de date pentru maximizarea vitezelor de acces în utilizările viitoare și pentru a oferi administratorului de aplicație posibilitatea de a le adapta și , în cazuri puțin probabile, corecta.

Pentru a identifica contextul lingvistic al coordonatelor date, aplicația utilizează funcția de Reverse Geocode a Google Maps API pentru a obține țara coordonatelor, iar, ulterior, compară rezultatul obținut cu lista Locale-urilor disponibilă în java.util pentru a obține informațiile complete de localizare.

Tot la nivelul serviciilor, în cadrul TwitterService, este implementată modalitatea de **raportare** a rezultatelor irelevante. Aceasta s-a ralizat prin efectuarea unor operații de actualizare a câmpurilor obiectului TweetSample din baza de date

Metodele implementate incrementează contorul de raportări, iar, în cazul în care acest contor devine mai mare sau egal cu un număr constant predefinit (în cazul de față 5), tweet-ul va deveni inactiv. Această acțiune reprezintă filtrarea automată a tweet-urilor în cazul în care mai mult de 5 utilizatori l-au considerat irelevant. De asemenea, administratorul de aplicație poate marca definitiv tweet-ul ca fiind irelevant, utilizând o metodă particulară.

La nivelul cel mai de sus, al endpoint-urilor, s-au definit următoarele adrese de acces în RestControllers.

În cadrul TweetsEndpoint:

- /tweets/get, metodă HTTP GET care primește prin query string coordonatele geolocalizării
- /tweets/incrementIrrelevant, metodă HTTP GET care primește un identificator și coordonate pentru a raporta (a incrementa contorul de raportări) a unui tweet
- /admin/tweets/getAllIrrelevant, metodă HTTP GET utilizată pentru afișarea tweet-urilor raportate administratorului de aplicație
- /admin/tweets/markAsIrrelevant, metodă HTTP GET parametrizată cu un identificator, utilizată de către administratorul de aplicație pentru a marca un tweet ca fiind definitiv irelevant

În cadrul LocalizationEndpoint:

- /admin/localization/getAll, metodă HTTP GET care returnează traducerile persistate în sistem administatorului aplicației
- /admin/localization/update, metodă HTTP PUT care primește în cadrul corpului mesajului HTTP obiectul de traducere ce va fi actualizat, pentru adaptarea traducerilor de către administratorul de aplicație

5.2.3.2. Revendicarea și salvarea punctelor de interes

În cadrul secțiunii 5.1.3.2 s-a prezentat metoda de identificare a punctelor de interes. Pentru accesul ulterior la aceste informații, o metodă de a eficientiza atât viteza, cât și numărul de request-uri la servicii externe este salvarea acestor puncte în cadrul serverului back-end și revendicarea lor din acesta în căutările viitoare.

Pentru implementarea acestei funcționalități, urmărind aceeași perspectivă bottom-up, s-a creat, mai întâi, entitatea NearbyResult, prezentată în subsecțiunea pachetului Model, 5.2.2.4, și repository-ul aferent acesteia, cu metoda parametrizată de căutare în funcție de latitudine, longitudine și aria de căutare (în metri).

Astfel, serviciile vor putea salva prin intermediul acestora datele obținute din frontend, definind detaliile scanării efectuate (latitudine, longitudine, arie) și obiectul de tip City obținut în urma apelării serviciilor externe. Pentru revendicarea rezultatelor, este suficientă specificarea datelor în formatul identic transmiterii acestora către API-ul extern.

În nivelul de Endpoints, s-au definit, în cadrul PlacesEndpoint, următoarele căi de acces:

- /places/get, metodă HTTP GET care primește prin query string latitudinea, longitudinea și aria zonei geografice scanate
- /places/persist, metodă HTTP POST care primește prin corpul mesajului HTTP un obiect de tipul NearbyResult

5.2.3.3. Autentificarea și autorizarea

Primul pas în implementarea sistemului este configurarea Spring Security pentru asignarea filtrelor de logare și autorizare pentru căile de acces securizate ale aplicației. Această configurare s-a realizat în clasa WebSecurityConfig, care extinde clasa spring WebSecurityConfigurerAdapter.

S-au asignat filtre path-urilor prin înlănțuirea unor comenzi specifice Spring Security pentru configurarea Http Security. Configurația setată poate fi urmărită în figura 5.16. Aceasta permite orice metodă HTTP de tipul OPTIONS către server, specifică faptul că pentru path-urile care încep cu “admin”, orice request trebuie autentificat, și adaugă un filtru de logare specific path-ului “login”. De asemenea, este asignat, în ultima linie de cod, un filtru de autentificare ce va fi descris în continuarea secțiunii.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .csrf().disable().authorizeRequests()
            .antMatchers(HttpMethod.OPTIONS).permitAll()
            .antMatchers( ...antPatterns: "/admin/**").fullyAuthenticated()
            .and()
            .addFilterBefore(corsFilter, ChannelProcessingFilter.class)
            .addFilterBefore(new JWTLoginFilter( url: "/login", authenticationManager()),
                    UsernamePasswordAuthenticationFilter.class)
            .addFilterBefore(new JWTAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
}
```

Figura 5.15 Configurația Spring Security

Pentru implementarea filtrelor și a serviciului de autentificare, s-a utilizat standardul JSON Web Tokens, descrisă în secțiunea 4.2.9, prin implementarea Java JWT.

Pasul următor este crearea unui serviciu de autentificare JWT, în cadrul clasei TokenAuthenticationService, bazat pe tokens. Serviciul JWT va avea responsabilitatea de a crea și verifica tokenii aplicației.

În cazul de față. Serviciul va crea un token pe baza username-ului și a unui timp de expirare (validitatea unui token va fi de 10 zile), care va fi semnat, ulterior, utilizând un hash criptografic generat de un algoritm HMAC cu SHA-512, pe baza unei parole secrete menținute pe serverul de backend Token-ul rezultat va conține alogirtmul utilizat, numele utilizatorului care a efectuat autentificarea, data de expirare a token-ukui și semnătura mesajului și va fi plasat în header-ul de autorizare al răspunsului sub prefixul specific “Bearer’.

Serviciul creat va fi utilizat de cele două filtre necesare: JWTLoginFilter și JWTAuthenticationFilter.

JWTLoginFilter va încerca autentificarea utilizatorului cu datele introduse și, în cazul favorabil, va adăuga autentificarea prin intermediul serviciului descris anterior, care va răspunde cu token-ul alocat.

JWTAuthenticationFilter va verifica validitatea tokenului pentru request-urile autorizate. În caz contrar, nu va permite request-ului să fie procesat mai departe.

5.3. Baza de date

Baza de date a aplicației a fost dezvoltată utilizând Amazon DynamoDB. Fiind o bază de date NoSQL bazată pe un sistem key-value, relațiile dintre tabele nu pot fi create în stilul bazelor de date relaționale. Din acest motiv, în dezvoltarea bazei de date, s-au utilizat convertoare pentru serializarea obiectelor complexe care fac parte din anumite tabele.

Tabelele bazei de date și caracteristicile acestora pot fi observate în figura 5.17. Un detaliu de menționat constă în semnificația câmpurilor de capacity (read/write). Amazon permite setarea unui număr maxim de operații pe un tabel pentru o secundă. Un astfel de raport constituie numărul de capacități pentru scriere/citire. În cazul în care este nevoie de mai multe unități decât cele atribuite, baza de date poate crește aceste capacități automat (auto-scaling), în limita numărului total de capacități incluse în planul serviciului. Un alt aspect important este acela că formatul unui tabel nu este unul rigid, fix, ci poate suferi modificări în urma operațiilor efectuate pe baza de date (adăugarea unei entități cu noi câmpuri va duce la adăugarea unui nou câmp tabelului).

Name	Status	Partition key	Total read capacity	Total write capacity
City	Active	id (String)	5	5
NearbyResult	Active	id (String)	5	5
Translation	Active	id (String)	5	5
Tweet	Active	id (String)	5	5
User	Active	id (String)	1	1

Figura 5.16 Caracteristicile tabelelor bazei de date

Tabelele bazei de date conțin perechi cheie-valoare generate în urma adnotării specifice a modelului. Astfel, în tabelul 5.2. identificăm următoarele tipuri de date cu atributele specifice și tipul aferent acestora.

Tabel 5.2 Tabelele bazei de date și atributele acestora

Tabel	Atribut	Tip
City	Id	String
	Language	String
	Lat	Double
	Lng	Double
	Name	String
NearbyResult	Id	String
	City	City
	Lat	Double
	Lng	Double
	Radius	Integer
Translation	Id	String
	From	String
	Text	String
	To	String
	TranslationText	String
Tweet	Id	String
	TweetId	String
	Content	String
	createdAt	String
	timeToLive*	String
	City	City
	markedAsIrrelevantCount	Integer
	Irrelevant	Boolean
User	Id	String
	Username	String
	Password	String

Câmpul timeToLive definește un timestamp UTC (numărul de secunde scurse de la 1 ianuarie 1970) pentru data la care intrarea respectivă din tabel va fi ștearsă în mod automat. Este utilizat în cadrul tweet-urilor raportate, pentru definirea unei perioade de 7 zile de la data creării tweet-ului în care acestea vor fi persistate. Stocarea lor după această dată ar fi inutilă, întru cât lipsa de actualitate a tweet-ului l-ar face imposibil de revendicat.

Capitolul 6. Testare şi Validare

Capitolul curent are ca principal rol definirea modalităților de testare a aplicației și argumentarea validității rezultatelor concrete obținute prin utilizarea sistemului în cadrul unui context real.

Pe lângă strategiile clasice de testare ale componentelor unei aplicații web (testare manuală prin investigarea cazurilor de testare, unit testing, testarea integrării componentelor sistemului sau stress testing), încă din definirea obiectivelor secundare, în secțiunea 2.2.1, s-a accentuat necesitatea efectuării unui studiu al relevanței rezultatelor obținute într-o perioadă de timp determinată.

Drept urmare, studiul menționat, împreună cu simularea unui scenariu real, vor constitui principalele teme ale acestui capitol, prin intermediul cărora se poate valida, într-o manieră convingătoare, buna funcționare a sistemului.

6.1. Testarea prin simularea unui scenariu real

Pornind de la premisa că în orașul Cluj-Napoca s-ar petrece un accident rutier, iar acesta ar fi observat de un martor ocular care utilizează Twitter, în cazul în care acesta ar decide să atenționeze rețeaua lui socială printr-un tweet, mesajul respectiv ar trebui să poată fi identificat de sistem și afișat în cazul în care un utilizator al aplicației studiază o rută care trece prin orașul Cluj-Napoca. Mesajul postat este ilustrat în figura 6.1.

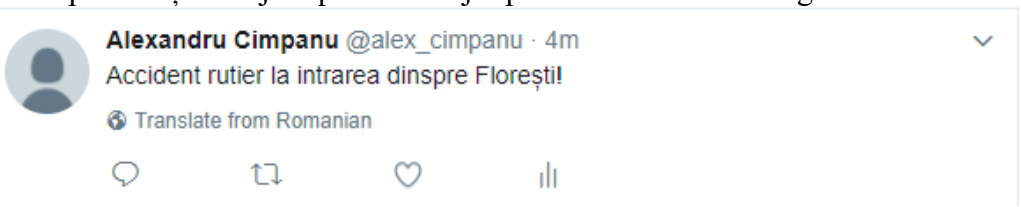

Figura 6.1 Mesajul postat pe Twitter în scenariul de test

Vom presupune că utilizatorul dorește să utilizeze aplicația pentru a identifica rutele disponibile între Oradea și Suceava. Primul pas pe care acesta îl efectuează este descoperirea rutelor și a informațiilor despre condițiile meteorologice. Obține trei trasee posibile: Oradea – Zalău – Bistrița – Suceava, Oradea – Cluj-Napoca – Bistrița – Suceava și Oradea – Cluj-Napoca – Târgu Mureș – Suceava, fapt ilustrat și de figura 6.2, extrasă din interfața aplicației.

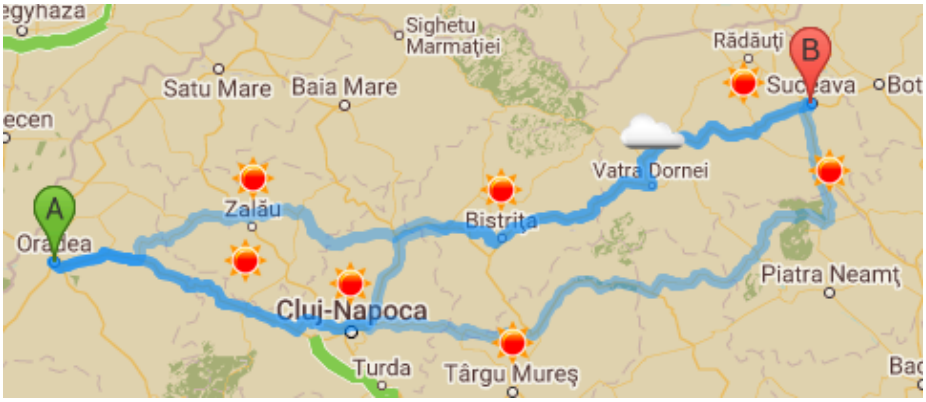

Figura 6.2 Rutele disponibile pentru scenariul de test

În următorul pas, utilizatorul va obține tweet-urile aferente condițiilor de desfășurare a traficului, iar, în Cluj-Napoca, locul accidentului fictiv descris anterior, se va regăsi, printre mesajele relevante postate în ordine cronologică, și tweet-ul martorului ocular, fapt vizibil în figura 6.3.

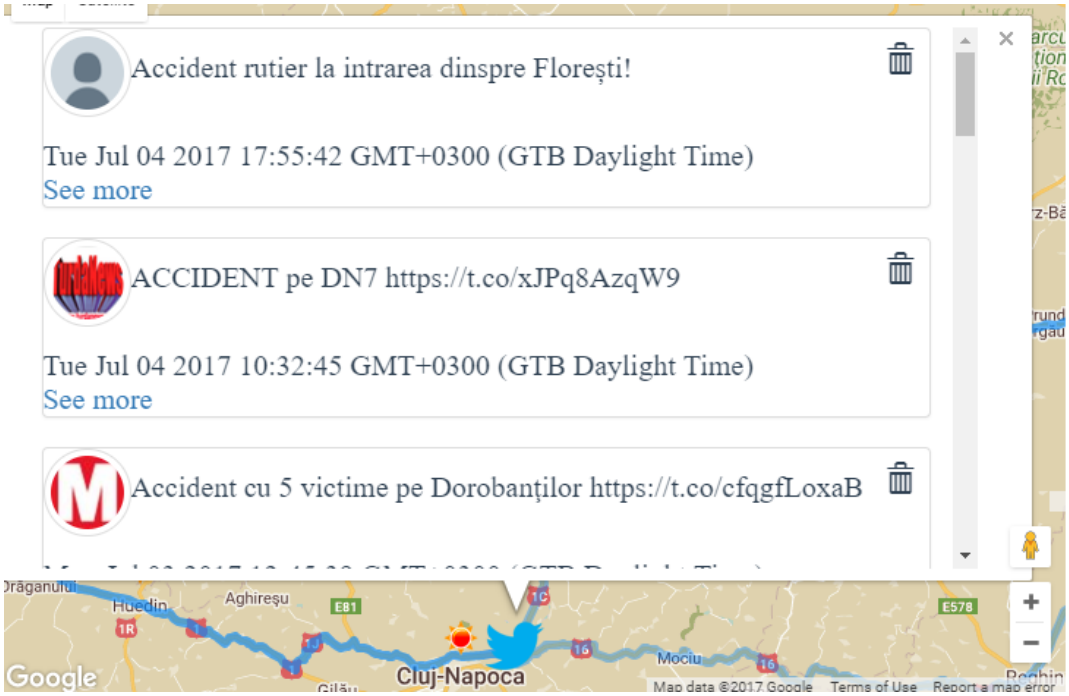

Figura 6.3 Identificarea mesajului martorului fictiv în cadrul sistemului

În concluzia acestei simulări, se demonstrează aplicarea într-un caz real ale principalelor funcționalități: identificarea condițiilor meteorologice, identificarea tweet-urilor cu informații despre condițiile de desfășurare a traficului, dar se demonstrează și relevanța informației și adaptarea la contextul lingvistic al zonei în care a fost efectuată căutarea (România), sistemul identificând rezultatele așteptate.

6.2. Validarea prin studiu de caz

Pentru atingerea obiectivului 2.2.1, dar și pentru validarea rezultatelor sistemului, un pas important a fost întocmirea unui studiu de caz, în care a fost analizată acuratețea informațiilor obținute pentru o perioadă de trei săptămâni, între 18.06.2017 și 09.07.2017, pentru două perechi sursă-destinație etalon: Dublin – Londra și Cluj-Napoca – București. Datele au fost preluate zilnic în jurul orei 20.

Primul itinerariu este reprezentativ pentru țările vorbitoare de limbă engleză în care Twitter este un mijloc popular de socializare, iar al doilea este reprezentativ pentru acele zone geografice în care engleza nu este limbă oficială, iar Twitter nu este la fel de popular.

Pentru Londra – Dublin, s-au obținut următoarele date, vizibile în graficul din figura 6.4:

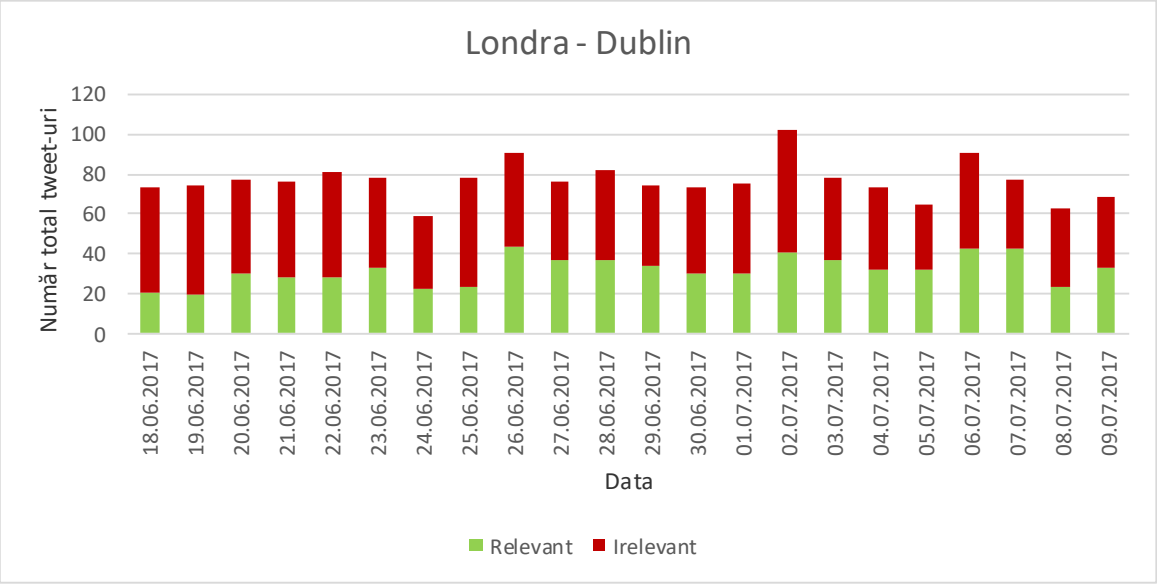

Figura 6.4 Graficul relevanței rezultatelor studiului de caz (Londra-Dublin)

Datele arată că pentru rutele Londra-Dublin, dintr-un total de 1685 tweet-uri, 701 sunt relevante, iar 984 irelevante. Așadar, aproximativ 41,6% din totalitatea mesajelor au fost relevante. Mai mult, s-au identificat, în medie, 76,59 mesaje pe zi. Dintre acestea, o medie de 31,86 mesaje au avut potențialul de a influența utilizatorul în alegerea unui traseu fără (sau cu mai puține) accidente.

În ceea ce privește al doilea itinerariu, Cluj-Napoca – București, graficul din figura 6.5 ilustrează distribuția datelor colectate.

Astfel, numărul total de tweet-uri observate a fost 533 (de trei ori mai mic raportat la graficul anterior). Dintre acestea, 317 au fost considerate relevante, iar 216 irelevante. Procentual, aproape 60% (59,47%) dintre rezultatele obținute pot fi considerate de impact. Analizând situația medie, zilnic, sistemul a identificat 24,22 tweet-uri cu potențial de descriere a condițiilor de circulație rutieră. Dintre acestea, în fiecare zi, 14,4 mesaje ar fi putut influența utilizatorul în alegerea unui traseu mai ușor, lipsit de accidente sau cu un număr redus.

O observație intersantă este faptul că, urmărind acest itinerariu, se poate deduce că în țările în care Twitter-ul este mai puțin popular, deși se înregistrează un număr semnificativ mai mic de mesaje, ponderea relevanței acestora este mult mai mare (cu aproximativ 20%). Această constatare se poate datora faptului că în astfel de situații, Twitter este folosit preponderent de agențiile de știri care doresc să disemineze informația în mediul online, și mai puțin de utilizatorii simpli, informația finală fiind de o calitate superioară.

În concluzia acestui studiu, pe lângă observațiile punctuale anterioare, se constată o relevanță medie de 41% a mesajelor din țările vorbitoare de limbă engleză în care Twitter este o rețea de socializare populară și o relevanță de aproximativ 60% în țările pentru care engleza nu este o limbă oficială și platforma nu este la fel de populară.

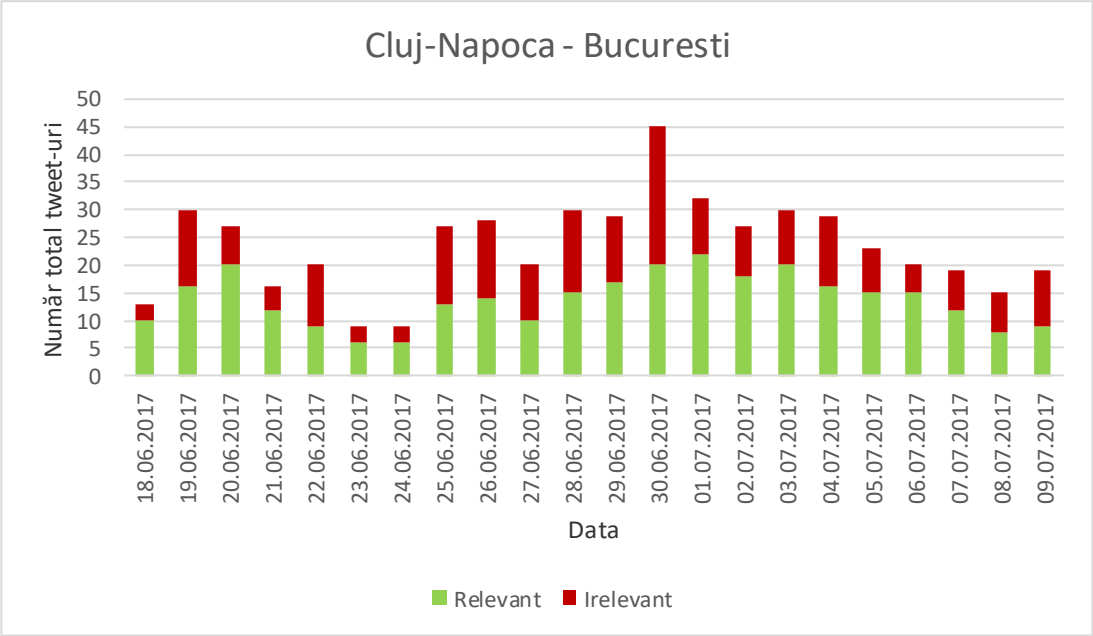

Figura 6.5 Graficul relevanței rezultatelor studiului de caz (Cluj-București)

Având în vedere faptul că sistemul implementează și un sistem de raportare a rezultatelor irelevante, intervenția umană a utilizatorilor poate duce la îmbunătățirea semnificativă a acestor proporții. De asemenea, considerând și natura problemei, este suficientă și prezența unui singur rezultat relevant pentru a influența, într-un caz real, calitatea călătoriei unei persoane.

Drept urmare, rezultatele obținute sunt satisfăcătoare, dar oferă, în același timp, posibilitatea îmbunătățirii prin strategiile de dezvoltare ulterioară, ce vor fi enumerate în cadrul concluziei acestei lucrări.

Capitolul 7. Manual de Instalare si Utilizare

Capitolul curent dorește specificarea unui set de instrucțiuni menite să ajute un utilizator nefamiliar cu aplicația implementată și tehnologiile utilizate să poată folosi sistemul propus. Se vor prezenta modalitățile de acces în cloud și local (mod de dezvoltare), tehnologiile necesare pentru rularea locală și modalitatea de deployment.

7.1. Instalare și rulare locală (mod de dezvoltare)

7.1.1. Tehnologii necesare

Un prim pas important pentru rularea, managementul dependințelor și deployment-ul ulterior al aplicației frontend este instalarea Node.js[30], care include managerul de pachete npm[31].

Pentru rularea aplicației serverului de backend este necesară instalarea ultimei versiuni de Java (JRE sau JDK) și a tool-ului Apache Maven[32].

7.1.2. Pornirea aplicației back-end

Primul pas necesar este instalarea dependințelor utilizând Maven. Prin urmare, se va accesa folderul rădăcina al proiectului back-end (directorul unde se găsește fișierul pom.xml), iar, din consola de comandă, se va rula comanda **mvn install**.

După descărcarea dependințelor, în directorul **target** al folderului rădăcină se va regăsi arhiva executabila .jar a proiectului. În cadrul acestui director, se va executa comanda **java –jar <denumirea fișierului .jar>**. În continuarea acestui pas, serverul aplicației va fi disponibil la adresa http://localhost:5000/.

7.1.3. Pornirea aplicației front-end

La fel ca în cazul aplicației back-end, primul pas necesar este instalarea dependințelor, de data aceasta prin intermediul utilitarului npm. Se va naviga în folderul rădăcină al proiectului front-end (directorul unde se găsește fișierul package.json), unde se va rula comanda **npm install**.

Următorul pas pentru pornirea unui server express de dezvoltare este rularea comenzii **npm run dev** în cadrul aceluiași director. După acest pas, serverul front-end va fi disponibil la adresa http://localhost:8081/.

7.2. Rulare și deployment în cloud

7.2.1. Accesul

Distribuirea sistemului implementat în cadrul cloudului AWS fiind efectuată cu succes, aplicația web poate fi navigată cu ușurință prin simpla accesare a link-ului ei: http://d26dmf1r2uzrns.cloudfront.net.

[30] https://nodejs.org/en/download/
[31] https://www.npmjs.com/
[32] https://maven.apache.org/

Adresa de acces a serverului backend este http://easyroute.us-west-2.elasticbeanstalk.com. Căile de acces la servicii și metodele HTTP permise sunt definite în secțiunile 5.2.2.

7.2.2. Deployment

Înaintea efectuării acțiunii de deployment în cloud, este necesară executarea unor pași premergători. În primul rând, variabilele constante care definesc căile de acces ale serviciilor backend trebuie modificate (http://localhost:5000/ -> http://easyroute.us-west-2.elasticbeanstalk.com). Acestea se regăsesc în fișierele din directorul src/services.

Următorul pas este executarea comenzii **npm run build** în directorul rădăcină al aplicației front-end pentru exportarea fișierelor statice care vor putea fi transpuse în cloud. De asemenea, primul pas al 7.1.2 trebuie repetat pentru construirea arhivei jar.

În continuare, se va accesa portalul AWS Elastic Beanstalk[33], se va selecta instanța aferentă URL-ului aplicației și se va naviga prin intermediul butonului "Upload and Deploy". În cadrul dialogului modal apărut, se poate apăsa butonul Browse pentru a încărca fișierul .jar obținut anterior.

Pentru deployment-ul aplicației web statice, se va accesa portalul AWS Quickstart Website[34], care grupează serviciile CloudFront și S3, de unde se va selecta instanța aferentă site-ului. Se va arhiva în format zip conținutul directorului **dist** al proiectului front-end, iar arhiva se va transfera prin procedura de drag & drop în cadrul căsuței de Source Code prezentă pe site, după cum se poate observa și în figura 7.1.

După efectuarea acestor pași, sistemul va fi disponibil accesului public prin adresele URL specificate în subsecțiunea anterioară. De menționat este faptul că pentru etapele de deployment este necesar accesul la contul AWS prin introducerea contului de utilizator și a parolei.

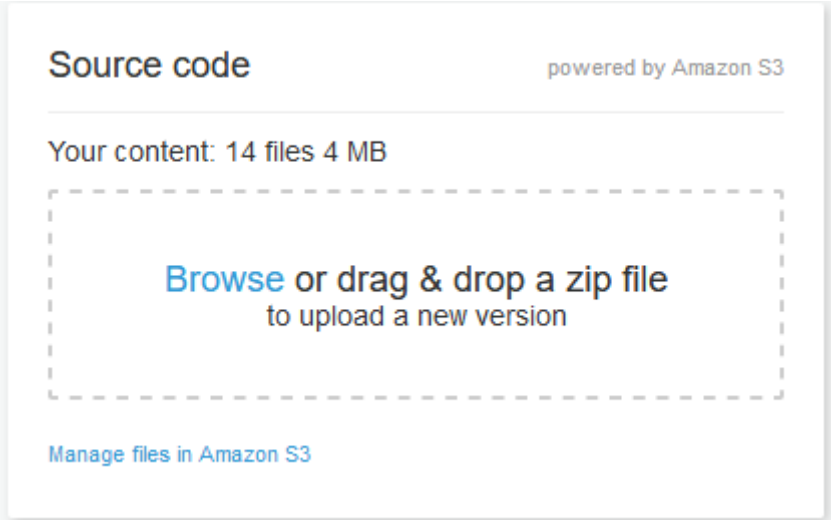

Figura 7.1 Plasarea codului sursă în Amazon S3 prin drag & drop

[33] https://us-west-2.console.aws.amazon.com/elasticbeanstalk/home
[34] https://console.aws.amazon.com/quickstart-website/home

Capitolul 8. Concluzii

În cadrul capitolului curent, se vor prezenta concluziile lucrării de față prin sistematizarea unei priviri de ansamblu asupra obiectivelor propuse și specificarea punctuală a progresului dobândit prin implementarea sistemului propus în atingerea acestora.

De asemenea, în finalul capitolului se va prezenta un set de propuneri pentru posibile strategii de dezvoltare ulterioară, astfel încât rezultatele obținute să poată fi îmbunătățite și funcționalitățile aplicației completate.

8.1. Retrospectiva obiectivelor și a rezultatelor

Obiectivul principal al lucrării de față a fost implementarea aplicației web utilitare pentru ajutarea șoferilor vehiculelor în alegerea unui traseu cât mai ușor și sigur între două locații, pe baza furnizării informațiilor despre condițiile meteorologice și cele de desfășurare a traficului (extrase din Twitter). Acesta a fost atins în totalitate prin implementarea pașilor descriși de obiectivele specifice definite în secțiunea 2.1.1.

Un prim obiectiv secundar îndeplinit a fost efectuarea analizei de relevanță a informațiilor extrase din Twitter. Atât lucrările științifice studiate, cât și lucrarea de față, prin intermediul studiului de caz prezentat în secțiunea 6.2, dovedesc un potențial colosal în exploatarea informațiilor cu caracter deschis ce se pot obține din rețeaua de socializare Twitter atât în contextul managementul traficului, cât și în alte domenii.

Integrarea în cadrul unei platforme cloud, al doilea obiectiv secundar al lucrării propuse, a fost realizată cu succes prin implementarea unui sistem distribuit scalabil, ce poate fi instalat cu ușurință în cadrul platformei AWS prin serviciile descrise.

Un alt aspect important pentru implementarea finală cu succes a sistemului informatic propus (și un alt obiectiv secundar definit) a fost adaptarea căutării la contextul lingvistic al punctelor de interes. Acest obiectiv a fost, de asemenea, îndeplinit în totalitate, utilizând mecanisme de traducere automatizată, aplicația implementată dobândind utilitate într-un context global, independent de o regiune sau o limbă anume, aspect ce s-a reflectat, ulterior, și asupra rezultatelor bune obținute în studiul de caz pentru itinerariul Cluj-Napoca – București.

Un ultim obiectiv secundar, crearea unei interfețe grafice responsive, a fost atins prin implementarea tehnicilor de grid layout și dimensiuni adaptabile descrise în secțiunea 5.1.4.

Așadar, din perspectiva obiectivelor, se poate observa îndeplinirea în totalitate a acestora, sistemul final descris implementând întregul set de funcționalități propuse inițial.

În ceea ce privește rezultatele obținute, este de recunoscut calitatea cuantificabilă a relevanței acestora, prezentată în cadrul studiului de caz din secțiunea 6.2. Mai mult, rezultatele bune sunt complementate de aspectul crucial al utilizabilității sistemului independent de regiune, iar în contextul utilizării acestuia de o comunitate, aplicația oferă, inclusiv, posibilitatea de filtrare automată a rezultatelor irelevante.

În plus, în cazul în care aplicația implementată ar fi popularizată, aceasta ar putea încuraja comunitatea online în utilizarea rețelei de socializare Twitter pentru raportarea incidentelor din traficul rutier, astfel contribuind la înștiințarea șoferilor asupra condițiilor

de desfășurare a traficului, fie prin intermediul sistemului propus, fie prin simpla publicare a acestor informații, ce pot dobândi un caracter deosebit de important în contextul general al soluțiilor software de prevenire a aglomerației rutiere.

8.2. Strategii de dezvoltare ulterioară

Deși rezultatele obținute îndeplinesc în totalitate așteptările și cerințele planificate inițial, sistemul implementat dovedește un potențial de utilitate semnificativ. Așadar, atât pentru materializarea acestui potențial, cât și pentru eficientizarea proceselor actuale, se pot stabili anumite strategii de dezvoltare ulterioară pentru îmbunătățirea generală a sistemului.

Dintre acestea, se disting următoarele posibile obiective generale de dezvoltare ulterioară:

1. Implementarea unei aplicații mobile native, prin dezvoltarea unui client pentru platformele principale software mobile (Android și iOS), sau a unei aplicații mobile hibride, pe baza clientului web deja existent. Acest obiectiv ar îmbunătăți utilizabilitatea din punctul de vedere al utilizării de pe dispozitivele mobile. De asemenea, o aplicație mobilă nativă ar putea facilita integrarea în timp real a aplicației cu sistemul de geolocalizare a dispozitivului (GPS) pentru avertizarea utilizatorului în mod automat înainte de atingerea punctelor de interes.
2. Regeolocalizarea inteligentă a rezultatelor. Există situații identificate pentru care geolocalizarea efectivă a persoanei care a scris tweet-ul nu reprezintă geolocația incidentului în cauză. În cazul utilizatorilor în tranzit, această problemă nu reprezintă un dezavantaj, întrucât aria de geolocalizare a tweet-urilor este permisivă, dar în cazul agențiilor de știri, acestea pot relata evenimente din locații diferite ale țării, sau chiar ale planetei. Posibile soluții includ utilizarea unor sisteme de geolocalizare pe baza procesării limbajului natural sau chiar complementarea modulului de raportare bazat pe comunitate prin introducerea tipurilor diferite de rapoarte și a sugestiilor venite de la utilizatori pentru acțiuni de regeolocalizare.
3. Clasificarea geolocalizării unui oraș la nivel de stradă/cartier/suburbie. Astfel, se vor putea obține informații utile pentru ocolirea punctului de interes afectat de un incident prin străzi sau zone urbane adiacente, nu prin rute alternative.
4. Efectuarea unei analize de sentimente pentru rezultatele relevante și irelevante pentru a studia impactul acestui aspect asupra relevanței conținutului mesajului în contextul actual. În cazul în care abordarea se dovedește de impact, un alt obiectiv ar fi îmbunătățirea acurateții căutării prin utilizarea tehnicilor de analiză a sentimentelor.
5. Clasificarea incidentelor descoperite pentru un itinerariu în funcție de impactul acestora asupra desfășurării circulației rutiere. Acest aspect ar putea fi implementat, de asemenea, utilizând tehnici de procesare a limbajului natural, analiză de sentimente, sau printr-un modul ce preia feedback de la comunitatea de utilizatori.

6. Sugerarea rutei recomandate considerând impactul evenimentelor întâlnite pe traseele disponibile și modul estimat în care acestea ar influența desfășurarea traficului.
7. Agregarea mai multor surse de informații cu același caracter public pentru sporirea numărului de rezultate. Utilizarea informațiilor extrase din alte rețele de socializare, din canale preconfigurate de știri, din RSS Feed-uri sau din resurse Semantic Web.

Bibliografie

[1] A. Schulz, P. Ristoski, and H. Paulheim, “I See a Car Crash: Real-Time Detection of Small Scale Incidents in Microblogs” in The Semantic Web: ESWC 2013 Satellite Events: ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013.

[2] F. A. Elsafoury, “Monitoring Urban Traffic Status Using Twitter Messages”, Master Thesis, University of Twente Faculty of Geo-Information and Earth Observation (ITC), 2013.

[3] R. Steur, “Twitter as a spatio-temporal source for incident management”, Master Thesis, University of Twente, Geographical Information Management and Applications, August 2014.

[4] T. Sakaki, M. Okazaki, and Y. Matsuo, “Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development,” IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 4, pp. 919–931, Apr. 2013.

[5] F. Abel, C. Hauff, G.-J. Houben, R. Stronkman, and K. Tao, “Semantics + Filtering + Search = Twitcident. Exploring Information in Social Web Streams,” in Proceedings of the 23rd ACM Conference on Hypertext and Social Media, Milwaukee, Wisconsin, USA, 2012, pp. 285–294.

[6] F. Abel, C. Hauff, G.-J. Houben, R. Stronkman, and K. Tao, “Twitcident: Fighting Fire with Information from Social Web Streams,” in Proceedings of the 21st International Conference on World Wide Web, Lyon, France, 2012, pp. 305–308.

[7] Twitter Developer Documentation [online]
https://dev.twitter.com/docs

[8] Q. Hassan, “Demystifying cloud computing,” The Journal of Defense Software Engineering, vol. 1, pp. 16–21, 2011.

[9] Cloud Computing, Wikipedia [online]
https://en.wikipedia.org/wiki/Cloud_computing

[10] Vue.js Guide [online]
https://vuejs.org/v2/guide/

[11] Vue.js API [online]
https://vuejs.org/v2/api/

[12] Webpack Documentation [online]
https://webpack.js.org/configuration/

[13] Google Maps Directions Service API Documentation [online]
https://developers.google.com/maps/documentation/directions/

[14] Google Maps Geocoding API Documentation [online]
https://developers.google.com/maps/documentation/geocoding/

[15] Google Places API Web Service documentation [online]
https://developers.google.com/places/web-service/intro

[16] Yandex Translate API Documentation [online]
https://tech.yandex.com/translate/doc/dg/concepts/About-docpage/

[17] Open Weather Map API Documentation [online]
https://openweathermap.org/api

[18] Spring Reference Documentation [online]
https://spring.io/docs/reference

[19] Spring Guides [online]
https://spring.io/guides

[20] JSON Web Tokens Proposed Standard
https://tools.ietf.org/html/rfc7519

[21] HMAC Algorithm [online]
https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

[22] AWS Elastic Beanstalk Documentation [online]
https://aws.amazon.com/documentation/elastic-beanstalk/

[23] Amazon DynamoDB Documentation [online]
https://aws.amazon.com/documentation/dynamodb/

Anexa 1. Lista figurilor

Anexa 2. Lista tabelelor

Anexa 3. Glosar

Cuvânt/sintagmă	Explicație
AJAX	Asynchronous JavaScript And XML. Tehnică de comunicare asincronă ce presupune utilizarea obiectului XMLHttpRequest în comunicarea cu servere.
AWS	Amazon Web Services, platformă de servicii cloud Amazon.
Benchmark	Un standard sau un punct de referință ce poate fi utilizat ca etalon în comparația unor tehnologii.
Big Data	Sintagmă utilizată pentru descrierea seturilor de date largi și complexe, a căror dimensiune face ca mijloacele tradiționale de procesare a datelor să nu fie adecvate în utilizarea lor.
HMAC	Hash-based Message Authentication Code. Tip specific de MAC ce include procedeul de criptare pe baza unei funcții criptografice de hash ce utilizează o cheie predefinită.
HTTP	Hypertext Transfer Protocol, protocol de aplicație pentru comunicare utilizat extensiv în WWW(World Wide Web).
IoC	Inversion of Control.
jar	Fișier Java Archive. Utilizat în agregarea claselor Java, metadatelor și resurselor.
JSON	JavaScript Object Notation. Format de date (specific JavaScript) flexibil și lizibil.
NoSQL	Bază de date de tip Non SQL/Non Relational/Not Only SQL, în care entitățile și relațiile sunt definite într-un mod mai flexibil.
POM	Project Object Model. Fișier xml fundamental de definire a configurației și dependințelor unei aplicații ce utilizează Maven.
REST	Representational State Transfer. Modalitate arhitecturală de comunicare interoperabilă între sisteme informatice în Internet.
SDK	Software Development Kit.
SPA	Single-Page Application.
Spring Bean	Obiect cu structură standardizată instanțiat, asamblat și gestionat de un container IoC Spring.
Template	Șablon.
URL	Uniform Resource Locator.
URL Encoding	Tehnică de codificare a anumitor caractere prin substituirea cu coduri de simboluri predefinite.
XML	Extensible Markup Language. Definește (și) un format bazat pe un set de reguli specifice pentru prezentarea datelor.

Anexa 4. Figuri

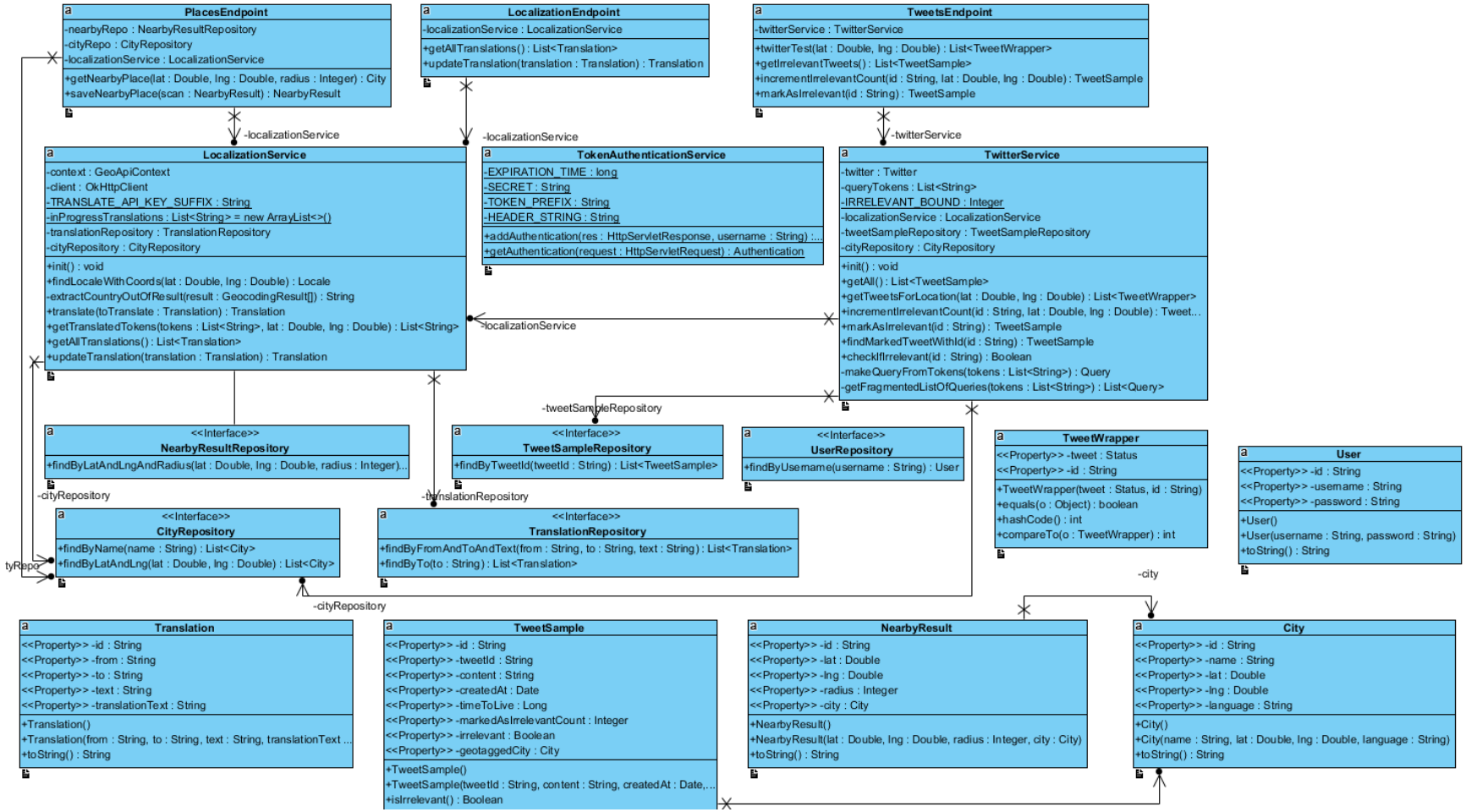

Figura A.1 Diagrama UML de clase a back-endului