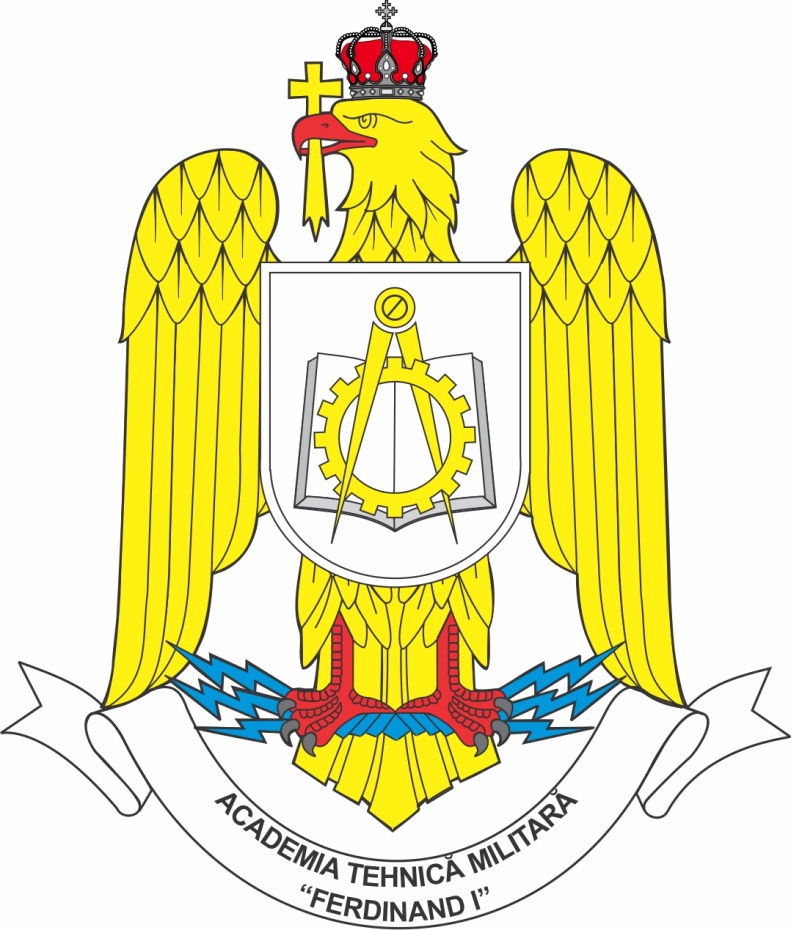
**ROMÂNIA**

**MINISTERUL APĂRĂRII NAŢIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare şi sisteme informatice pentru apărare şi securitate naţională**



**Aplicație web pentru analiza reflectorilor permanenți în imagini radar din satelit**

CONDUCĂTOR ŞTIINŢIFIC:

**Mr. conf. univ. dr. ing. Ștefan-Adrian TOMA**

ABSOLVENT:

**Std. sg. maj. PARASCA Andrei-Nicușor**

Conţine \_\_\_\_\_\_\_\_\_\_\_ file Inventariat sub nr. \_\_\_\_\_\_

Poziţia din indicator: \_\_\_\_

Termen de păstrare: \_\_\_\_\_

**BUCUREŞTI**

**2020**

**Abstract**

**Web application for persistent scatterers analysis**

**Author: Andrei Parasca**

**Coordinator: Adrian-Ștefan Toma**

Interferometria Scatterer persistentă este o tehnică puternică de teledetecție capabilă să măsoare și să monitorizeze deplasările suprafeței Pământului în timp. În mod specific, PSI este o tehnică bazată pe radar, care aparține grupului radarului de diafragmă sintetică interferometrică diferențială (SAR)

Scopul acestui document este de a prezenta modul in care se poate realiza o analiza a unor profile de deformare folosind tehnici specifice si vizualizarea acestora in cadrul unei aplicatii GIS. Prima parte a acestui document prezinta elementele de baza in cadrul SAR si modul de prelucrare a datelor. A doua parte prezinta o aplicatie web al carui scop este interfatarea informatiilor extrase in vederea analizei acestora.

# Introducere

## Motivație

În cadrul topogeodeziei, monitorizarea înseamnă compararea condițiilor sale (de exemplu, întinderea arealului, viteza de mișcare, topografia suprafeței sau umiditatea solului) în diferite perioade pentru a evalua activitatea alunecării sau deplasării suprafeței terestre. Măsurarea deplasării superficiale indusă de o mișcare de pantă reprezintă adesea cea mai eficientă metodă pentru definirea comportamentului acesteia, permițând observarea răspunsului la factorii declanșatori și pentru a evalua eficacitatea măsurilor de atenuare [2]. Recuperarea în timp a deplasărilor superficiale la sol se bazează istoric pe tehnici tradiționale, inclusiv extensometre convenționale de sârmă [3], inclinometre [4], GPS [5], nivelare [6,7] sau, mai recent, fotogrammetrie [8] și laser terestru scanare [9,10]. Aceste tehnici, în ciuda robustetei și fiabilității lor, consumă mult timp și consumă resurse, deoarece necesită mult timp și bani pentru actualizări în timp util. Imaginile cu teledetecție reprezintă un instrument puternic pentru măsurarea deplasării alunecărilor de teren, deoarece oferă o vedere sinoptică care poate fi repetată la intervale de timp diferite și poate fi disponibilă la diferite scale. În special, interferometria SAR-ului satelit (Synthetic Aperture Radar) [11,12] reprezintă un instrument de sunet pentru a evalua modificările pe suprafața Pământului. Este notoriu faptul că analiza imaginilor SAR unice nu este utilă, întrucât nu este posibilă diferențierea și separarea contribuțiilor de fază diferite legate de reflectivitatea obiectului, topografia, atmosfera și zgomotul inerente oricărui sistem de achiziție. Acoperirea largă și timpul de revizuire în scurt timp la Sentinel-1A / B și în afară de satelit duc la deformarea nivelului național de la nivelul interferometriei dispersante persistente (PSI). dimensiunea scăzută a a putea genera prin PSInSAR o față imposibilă pentru operatorii umani să-l analizeze. De exemplu, există aproximativ 9 milioane de împrăștiatori persistenți (PS) care pun în evidență o singură încălzire de îngrijire acoperită partea de est a României (3 imagini Sentinel-1 SLC). Vizualizarea tuturor profilurilor și identificarea acestora anomale (de exemplu, îngrijire cu privire la funcții de pas sau accelerații bruște, ceea ce poate arăta că ar putea indica probleme structurale la clădiri) este imposibil pentru un om.

## Scopul și obiectivele lucrării

## Structura lucrării

## Sumarul rezultatele obținute

# Interferometria cu reflectori permanenți

PSI este cea mai avansată clasă de tehnici de radar cu diafragmă sintetică interferometrică diferențială (DInSAR) bazată pe datele obținute de senzorii SAR spațiali. Tehnicile PSI pot fi utilizate, în principiu, cu date provenite de la senzori SAR terestre (Crosetto și colab., 2009) sau aerieni; cu toate acestea, senzorii SAR spațiali sunt de departe cea mai importantă sursă de date PSI. Pentru o revizuire generală a interferometriei SAR și a DInSAR, vezi Rosen și colab. (2000) și Crosetto și colab. (2005). După cum am menționat mai sus, PSI reprezintă cea mai avansată clasă de tehnici DInSAR, care a început cu așa-numita tehnică Scatterers permanente propusă de Ferretti și colab. (2000). În urma acestei abordări au fost propuse diferite tehnici în anii următori, vezi de ex. Ferretti și colab. (2001), Berardino și colab. (2002); Colesanti și colab. (2003), Mora și colab. (2003), Lanari și colab. (2004); Hooper și colab. (2004), Crosetto și colab. (2005), Pepe și colab. (2005), Crosetto și colab. (2008). Chiar dacă inițial aceste tehnici au fost numite „tehnici de scatterere permanente”, acum toate, inclusiv tehnica originală Scatterers permanente, sunt denumite „tehnici PSI”. De menționat este faptul că termenul „Scatterers permanenți” este asociat direct cu tehnica originală patentată de Ferretti et al.

Există două diferențe principale între tehnicile DInSAR și PSI: prima este numărul de imagini SAR procesate (PSI folosește serii mari de imagini SAR, de obicei mai mult de 20), iar cealaltă este implementarea procedurilor de modelare și analiză adecvate de date. care permit unuia să obțină următoarele produse cheie: (i) seria de timp a deformării; (ii) ratele medii de deplasare în perioada observată; (iii) componenta fazei atmosferice a fiecărei imagini SAR; (iv) așa-numita eroare topografică reziduală (diferența dintre adevărata înălțime a centrului fazelor de împrăștiere și înălțimea dată de DEM-ul folosit). Acest parametru este important pentru scopuri de modelare (adică pentru a separa componenta topografică reziduală de cea de deformare) și pentru scopuri de geocodare. Produsele principale ale oricărei analize PSI sunt date de harta ratelor medii de deplasare și a seriei de timp de deformare a fiecărui Scatterer persistent măsurat (PS). Achiziția repetată de imagini pe o anumită zonă se realizează de obicei cu același senzor sau senzori cu caracteristici identice ale sistemului. Gabriel și colab. (1989) a oferit prima descriere a DInSAR, care s-a bazat pe date ale benzii L Seasat. Pentru o revizuire generală a interferometriei SAR, a se vedea Bamler și Hartl (1998) și Rosen și colab. (2000).

Principiul DInSAR este prezentat pe scurt mai jos. Având în vedere o amprentă de un singur pixel pe solul P, senzorul achiziționează o primă imagine SAR dintr-o poziție de satelit M, măsurând o fază uM:

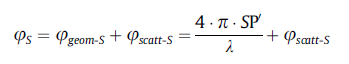


unde MP este senzorul la distanța țintă, uscatt este deplasarea de fază generată în timpul interacțiunii dintre microundele și P țintă, k este lungimea de undă a radarului, iar factorul 4p este legat de calea cu două sensuri, radar-țintă-radar. Presupunând că senzorul achiziționează o a doua imagine dintr-o poziție S a satelitului, măsurând faza uS peste aceeași amprentă de pixel P, apoi:

Tehnica SAR interferometrică (InSAR) exploatează diferența de fază uS \_ uM:



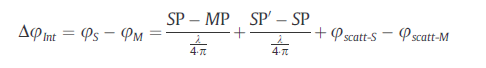
Această fază se numește fază interferometrică și este legată de diferența de distanță SP \_MP, care este fundamentală pentru generarea modelului digital de elevație (DEM), adică pentru a estima topografia scenei observate, a se vedea Bamler și Hartl (1998) și Rosen și colab. (2000). Sensibilitatea InSAR la topografie depinde de linia de bază a satelitului SM, și mai exact de proiecția SM în direcția perpendiculară pe linia SAR de vedere (LOS), numită linie de bază perpendiculară. În cazul măsurării deformării DInSAR, luând în considerare o amprentă P de un singur pixel și o primă achiziție din poziția satelitului M (vezi Fig. 1), se măsoară o fază uM, vezi Eq. (1). Apoi, presupunând că ținta se mută de la P la P0 și că, ulterior, senzorul achiziționează o a doua imagine din poziția S a satelitului, uS este estimat:



În acest caz, faza interferometrică DuInt este dată de:



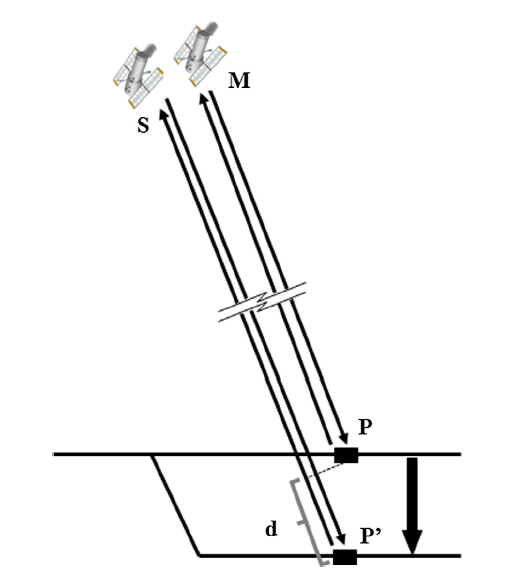
Prin adăugarea și scăderea termenului SP = k\_4\_p\_ se obține următoarea ecuație:



unde primul termen este componenta de fază topografică uTopo, care include așa-numita componentă faza elipsoidală de referință, iar al doilea termen este componenta de deplasare a fazei uDispl, legată de deplasarea LOS d prezentată în Fig. 1. Presupunând că ultimele două termeni de ech. (6) anulare, dacă este disponibil un DEM al scenei imaginate, uTopo poate fi simulat și scăzut din DuInt (aceasta este operația inversă realizată în generația InSAR DEM), obținând așa-numita fază DInSAR DuD-Int:



unde uTopo simu este componenta topografică simulată, care implicit conține o fază de pământ plat. Rețineți că erorile orbitale afectează această componentă topografică simulată, chiar dacă procesul de aplatizare nu se face în mod explicit. Eq. (7) rezumă principiul de lucru DInSAR, care permite deplasările scenei imagistice să fie obținute din două imagini SAR complexe. Acest principiu a fost exploatat pe scară largă în ultimii 25 de ani, obținând rezultate semnificative în domeniile seismologiei (Massonnet și colab., 1993; Peltzer și Rosen, 1995; Dalla Via et al., 2012), vulcanologie (Massonnet și colab., 1995; Massonnet și Sigmundsson, 2000; Antonielli și colab., 2014), glaciologie (Goldstein și colab., 1993; Rignot și colab., 1997), alunecări de teren (Carnec și colab., 1996; García-Davalillo și colab., 2014 ), dependența solului și ridicarea (Galloway et al., 1998; Amelung și colab., 1999), etc. Recenzii complete ale diferitelor aplicații DIn-SAR sunt furnizate de Massonnet și Feigl (1998) și Hanssen (2001).



Tehnica permanentă (de asemenea, persistentă) Scatterers InSAR (PSInSAR) este o actualizare a DInSAR. În scopuri analitice, această metodă folosește ținte radar coerente care pot fi distinse în mod clar în toate imaginile și nu variază în proprietățile lor (Ferretti și colab., 2001). Pe baza proprietăților permanente, ele sunt numite împrăștiate permanente. Folosind radiatori permanenți, efectele atmosferice pot fi filtrate în timp și decorelația geometrică poate fi elimiată. Dezavantajul acestei metode este pierderea continuității datelor. Datele sunt un set de puncte cu o densitate în funcție de forma și acoperirea suprafeței. Aceste ținte radar coerente sunt abundente în zonele urbane, dar sunt foarte rare în zonele vegetate și muntoase.

O minimizare statistică a acestor tulburări poate fi obținută prin utilizarea datelor radar pe o perioadă mai lungă și prin determinarea țintelor radar coerente - dispersoare permanente. Această metodă poartă numele de Radar de Interferometrică cu Scatter Permanent cu Apertură Sintetică sau PSInSAR. Fig. 5 prezintă principiile de bază ale funcționării dispersoarelor permanente InSAR. Sateliții (ERS-1 în ERS-2) care furnizează imagini care sunt sursa principală pentru PSInSAR (Permanent Scatter InSAR) au un ciclu de orbită de 35 de zile. Mișcările (Δ) în linia vizuală care sunt în acest timp mai mici de jumătate din lungimea de undă folosită (5,6 cm) pot fi înregistrate pe baza diferenței de undă (Δφ) a semnalului retras. Metodologia poate fi utilizată pentru a înregistra modificări ale suprafeței, adică subsidența sau creșterea suprafeței, cu accent pe mișcări în direcția semnalelor emise (Δ). Înregistrarea modificărilor relative ale creșterilor suprafetei este interesantă pentru numeroase zone. Subvențiile pentru terenuri pot fi un indicator al unei zone de subsol datorită exploatării miniere sau de suprafață, pompării apelor subterane, alunecării terenurilor, subsidenței datorate consolidării solului după construcție și altele similare. Rezultatele sunt cele mai aplicabile și în analiza mișcărilor tectonice ale scoarței terestre. Metoda este încă în faza de dezvoltare a aplicației, de aceea, intervalul aplicațiilor sale este de fapt nedefinit. La fel ca măsurile GPS diferențiale, toate măsurătorile PS se bazează pe măsurarea modificărilor în raport cu ținta radar de referință. Această proprietate sau condiție are impact asupra preciziei măsurătorilor InSAR. Precizia este determinată de mai mulți factori precum: numărul de imagini utilizate, densitatea dispersoarelor permanente, condițiile atmosferice în timpul reîncordărilor, distanța locației măsurate față de punctul de referință etc. prin utilizarea datelor InSAR pe o perioadă mai lungă și prin definiția coerent-urilor de difuzare - dispersoare permanente - tulburările menționate mai sus pot fi reduse la minimum. Măsurătorile modificărilor de suprafață (sau ale obiectelor observate) cu metoda PSInSAR sunt foarte precise, deoarece măsurătorile mișcărilor în direcția traversării semnalului pot atinge o precizie anuală sub un milimetru. Tabelul 2 oferă descrieri ale valorilor obișnuite ale calității măsurătorilor pentru o loțiune la mai puțin de 2 km distanță față de punctul de referință. Procedeul procesului interferometric este în ciuda bazei teoretice mai mult sau mai puțin explicite relativ complexe. Acesta poate fi împărțit aproximativ în câteva etape de bază, care sunt ilustrate schematic în pagina următoare (Fig. 4; O {tir, 2000):

* selectarea perechilor de imagini,

• co-înregistrare de imagini,

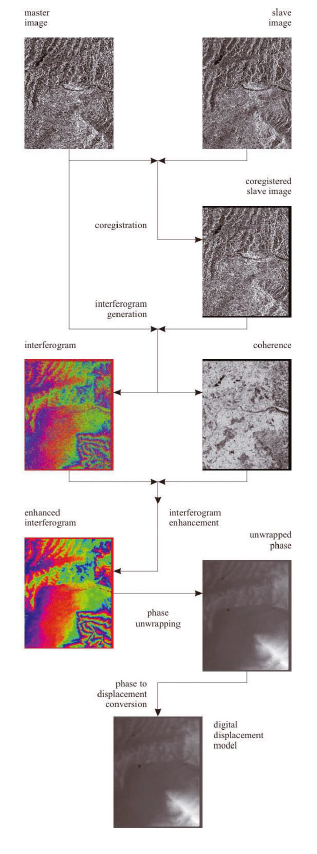
• pregătirea modelului digital extern de alegere1,

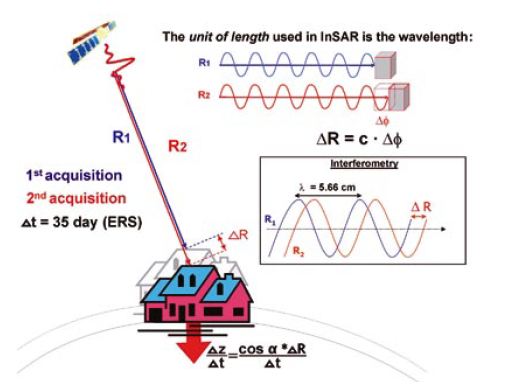
• generarea de interferograme;

• îmbunătățirea interferogramelor,

• desfacerea fazelor,

• producerea unui model digital de ridicare și mișcare și

• geocodare.  
Interferometria este foarte sensibilă la parametrii de intrare și la algoritmii folosiți. Cert este că calitatea rezultatelor este influențată de fiecare etapă individuală. Deși echipamentele hard disk din ziua de azi sunt eficiente pentru prelucrare, software-ul prezintă încă o problemă majoră, deoarece este limitat în calitatea algoritmului, în optimizarea vitezei și în cerințele de memorare.



Tehnologia logistică PSInSAR funcționează cu aceeași procedură ca InSAR „clasică”, numai că nu observă întreaga suprafață, ci doar obiecțiuni individuale numite dispersoare permanente (Ferretti și colab., 2001, 2005). Dispersoarele permanente sunt obiecte care pot fi recunoscute pe înregistrări radar satelit și sunt coerente pe o perioadă mai lungă de timp. Cu alte cuvinte, asta înseamnă că proprietățile lor practic nu se schimbă. Tehnologia PSInSAR oferă cele mai bune rezultate în zonele urbane și în zonele cu roci goale și, în general, unde este posibil să se identifice obiecte a căror reflectare nu se schimbă cu timpul. Datorită lungimii de undă a semnalului emis, această metodologie este utilă pentru mișcările care sunt mai mici decât lungimea de undă a emisiilor în perioada cuprinsă între două achiziții în direcția de deplasare a semnalului. În cazul sateliților ERS, ale căror imagini sunt cel mai frecvent utilizate, această valoare este de aproximativ 5 cm în perioada de 35 de zile. Mișcările sunt efectiv determinate relativ în ceea ce privește punctul de referință din zona observată. De regulă, acesta este un punct geodezic bine măsurat, care este estimat să nu fie supus mișcărilor majore.

Dispersoarele permanente pot fi naturale, de ex. afecțiuni de rocă, sau artificiale, zăcăminte, poduri, baraje, antene și obiecte similare (Ferretti și colab., 2001). De asemenea, pot fi utilizate împrăștiate construite în mod intenționat, cum ar fi plăcile metalice simple sau reflectoarele dreptunghiulare, construite din plăci de formare a materialelor (Fig. 8). Tehnologia PSInSAR analizează o suprafață de câteva mii de kilometri pătrați și prin căutarea împrăștierii creează un fel de „rețea geodezică naturală”, prin care se pot determina deformările de suprafață și stabilitatea anumitor obiecte.

# Metodologia detecției profilelor de deformare cu anomalii

# Aplicația PSTOOL

Aplicația descrisă in acest document reprezintă o aplicație web destinată analizei și monitorizarii topogeodezedice prin intermediul unor date obținute în urma unor operații specifice interferometriei cu reflectori permanenți.

Aplicația iși propune să fie utilizată in vederea determinarii proprietăților unor puncte, segmente sau arii geografice sau pentru a realiza anumite statistici (numerice și grafice) asupra unor suprafețe terestre utilizând date obținute prin procedee de interferometrie *specifice*. Cele din urmă pot conduce la concluzii ce pot fi folosite in alte domenii ( spre exemplu construcții, arheologie, turism, domeniul militar). În funcție de corectitudinea datele de intrare pe care utilizatorul le va oferi aplicației, acestea vor permite realizarea unei analize și studiu complet a fenomenelor de analizat.

## Definirea cerințelor

Din punct de vedere funcțional, aplicația va îndeplini următoarele cerințe:

* trebuie să pună la dispoziție o hartă pe care utilizatorul aplicației o poate utiliza si cu care acesta poate interactiona. Harta va fi o reprezentare in plan a planetei si va dispune de un nivel de zoom care sa satisfacă scările din intervalul 1: 1.000.000 (scara mica) si 1 : 20.000 (scara mare).
* Tipul hărții pe care utilizatorul va naviga trebuie să fie configurabilă și să permită utilizatorului acesteia poate fi general, sau tematic: geologic sau geografic (diurn).
* Utilizatorul aplicatiei va putea realiza operatii precum mărire, micșorare, deplasare, deplasare a hărții, rotire asupra hartii.
* Aplicația trebuie să permită utilizarea acesteia în mod autentificat, fiecărui utilizator fiindui asociat câte un rol ce îi va permite să realizeze operații în funcție de acțiunile permise. Rolurile disponibile în aplicație sunt definite în tabelul de mai jos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Roluri** | | | |
| **Denumire acțiune** | **Demo** | **Utilizator** | **Admin** | **Super Admin** |
| *Navigare în aplicație* | Limitat | Da | Da | Da |
| *Schimbare*  *sursă de puncte* | Da | Da | Da | Da |
| *Vizualizare hartă* | Da | Da | Da | Da |
| *Vizualizare profile de deformare* | Da | Da | Da | Da |
| *Vizualizare palete de culori* | Da | Da | Da | Da |
| *Vizualizare puncte inserate in aplicație* | Limitat  (un singur set) | Limitat | Da | Da |
| *Vizualizare logurile aplicației* | Nu | Nu | Nu | Da |
| *Inserare palete de culori* | Nu | Da | Da | Da |
| *Inserare seturi de puncte* | Nu | Nu | Da | Da |
| *Gestionarea utilizatorilor existenți* | Nu | Nu | Da | Da |
| *Gestionare seturi de puncte* | Nu | Nu | Da | Da |
| *Modificare date asupra unui utilizator* | Nu | Nu | Nu | Da |
| *Primire notificări* | Nu | Da | Da | Da |

* Trebuie să se pună la dispozitie un mecanism de inregistrare a unui utilizator nou. Pentru confirmarea realizării contului, acesta va fi notificat prin emailul furnizat către aplicație în momentul completării formularui pentru înregistrare. Autentificarea utilizatorului va fi persistentă, indiferent dacă acesta închide sau nu agentul pe care acesta îl folosește (browserul), singura modalitate de ieșire din aplicație reprezentând un buton sau un link creat special in acest sens vizibil in interfață.
* Aplicația trebuie să permită unui utilizator autentificat (cu rolul de administrator sau delogat cu această calitate) să incarce puncte de analiză din fisiere (in format **csv**) semantica si sintaxa datelor fiind stabilite de aplicatie sau il va stabili utilizatorul dintr-un meniu din interfata ( sau prin intermediul unui alt fisier in care este precizat semnificatia datelor incarcate). Punctele vor reprezenta coordonate geografice ( exprimate printr-un format ales de utilizator si precizat in momentul incarcarii) si descriere acestora ( identificator, pozitiile in imaginea satelitara de referinta, proiectiile, inaltimea, rata de deformare, devierea standard, estimari, alte date de interfometrie ). Stocarea punctelor și a datelor despre acestea va fi non-volatila, iar orice utilizator al aplicatie va avea acces la vizualizare acestora (indiferent daca au fost incarcate de acesta sau nu, în limita rolului pe care acesta îl are).
* Aplicația trebuie să permită unui utilizator să încarce puncte din surse si servicii externe (sursa Geoserver sau orice serviciu care implementează API-ul specificat la **https://docs.geoserver.org/latest/en/user/data/cascaded/wmts.html**) și vizualizarea acestora din cadrul acestor surse.
* Aplicația trebuie sa puna la dispozitie capacitatea de a vizualiza puncte prelucrate conform in vederea analizei acestora ( informațiile vor fi accesibile utilizatorului pentru fiecare punct din setul de puncte incarcate ).
* Vizualizarea acestora se va face pe hartă, culoarea fiecarui punct fiind aleasa dintr-o paleta fixa de culori, in functie de un criteriu ales de utilizator (default va fi inaltimea).
* Aplicatia trebuie să permită unui utilizator (dacă are rolul corespunzător) posibilitatea de a crea o paleta de culori pe care orice utilizator o va putea utiliza în vederea vizualizării punctelor in respectiva gamă de culori descrisă de paletă.
* Aplicația trebuie să permită folosirea unor palete de culori din surse externe în vederea aplicării unor culori alese de utilizator pentru datele prezentate (sursa Geoserver sau orice serviciu REST care implementează API-ul specificat la **https://docs.geoserver.org/stable/en/user/rest/** )
* Aplicația trebuie sa creeze grafice bidimensionale asociate informațiilor din punctele analizate, și să dispuna de un mecanism de vizualizare concomitenta a mai multor grafice. De asemenea, vor fi necesarea trasarea unor funcții (regresie liniară, polinomială) asupra acestor grafice
* Aplicația trebuie să dispuna de un clasificator al punctelor analizate si de o metoda de vizualizare 3D a clasificarii realizate ( a clusterelor care se formeaza ).

Din punct de vedere non-funcțional, aplicația va îndeplini urmatoarele cerințe:

* Aplicația trebuie să ofere un timp de raspuns rapid in momentul navigarii pe harta, implicit in momentul in care se deseneaza punctele analizate pe aceasta ( cel mult *500* de ms pentru fiecare actiune realizata ).
* Daca o actiune (hover, drag, zoom in/out) necesita un timp de raspuns mai mare de *500ms*, acest lucru trebuie sa fie notificat utilizatorului printr-un indicator de incarcare, cu precizarea ca experienta acestuia cu utilizarea interfetei nu trebuie sa fie ingreunata de timpii de raspuns ( nu pot sa existe mai mult de 10 incarcari cu timpi mai mari de 500 de ms intr-un interval de 30 de secunde ).
* Aplicatia trebuie sa realizeze comunicarea cu browserul utilizatorului sau cu orice tert (server de harti, server de autorizare, etc.) criptat si sa dispuna de mecanismele de securitate necesare.
* Aplicația trebuie sa aiba activitatea politica CORS
* Aplicatia trebuie sa fie portabila si compatibile cu urmatoarele browsere:
  + Microsoft Edge v.12
  + Firefox v.70
  + Google Chrome v.77
  + Safari v.13
  + Opera v.63
  + Android Browser v.76
  + Android Chrome v.78
  + Android Firefox v.68
  + iOS Safari v.13
* Compatibilitatea va ramane de asemenea cu orice versiune nouă a browserelor amintite mai sus.
* Aplicația trebuie să puna la dispozitie optiunea de schimbare a limbii ( engleza / romana )
* Indisponibilitatea aplicatiei sau a unor resurse solicitate de la aceasta trebuie sa fie notificata utilizatorului prin intermediul unor mesaje corespunzatoare
* Performantele server-ului trebuie sa scaleze odata cu resursele hardware pe care acesta le utilizeaza
* Aplicatia și sistemul de stocare/administrare al datelor trebuie sa permita capacitatea stocarii si administrarii unui numar de puncte de ordinul *sutelor de milioane*
* Aplicația trebuie să suporte cereri multiple și să gestioneze cazurile de acces concurențial la resurse, cât și funcționalitatea corectă în cazul unor astfel de scenarii
* Design-ul aplicației trebuie să **faciliteze** analiza punctelor și să nu ingreuneze experienta utilizatorului ce are ca scop principal analiza datelor prezentate in aplicație. Pentru acest lucru se va ține cont de timpii de răspuns ai serverului dar și creearea de feedback continuu asupra utilizatorului sau mesaje de eroare apropiate nivelului de ințelegere a unui utilizator care nu deține cunoștiințe tehnice in domeniul programării. Design-ul interfetei grafice trebuie sa fie solid, simplist și să respecte reguli, standarde și convenții de bază in cadrul layout-urilor, proporțiilor, dimensiunilor si dispuneri elementelor și controalelor cu care utilizatorul interacționează.

### Cazuri de utilizare

-- de adaugat cazuri de utilizare si diagrame --

## Arhitectura aplicației

Pentru dezvoltarea aplicației web, frameworkul principal folosit este ASP.NET MVC Framework la care se adaugă următoarele frameworkuri/biblioteci/sisteme, module, standarde software:

* OpenLayers (bibliotecă ce furnizează funcții pentru randarea hărții în browserul clientui, precum și adăugarea de date vectoriale sau markere pe suprafața randată. Este opensource, distribuit sub licență MIT fapt ce permite customizări și optimizări adaptate aplicației dezvoltate)
* SignalR (bibliotecă ce este folosită pentru a realiza sistemul de notificări și comunicația statefull între client si server. Ca tehnologie de bază, aceasta folosește protocolul WebSocket și API-ul standardului HTML5, însă este compatibil și cu browsere care nu dispun de aceste tehnologii prin implementarea unor algoritm de pooling sau hook-uri),
* MSSqlServer (sistem de gestiune a bazelor de date utilizat pentru administrarea metadatelor despre punctele vizualizate in aplicație, cât și a datelor despre utilizatori),
* Geoserver (reprezintă un server open-source dezvoltat în vederea implementării unor serii de servicii precum **WFS, WMS, WCS, WPS** sau **WMTS**. Acesta este folosit ca unul din serverele ce furnizează punctele afișate pe hartă),
* PostgreSQL (Folosit ca sistem de gestiune pentru administrarea detaliilor despre puncte, in condițiile in care punctele afișate pe ecran sunt furnizate de Geoserver),
* Cassandra **DBMS** (folosit ca sistem de gestiune pentru administrarea punctelor si a detaliilor despre puncte, cât și ca sursă pentru punctele afișate),
* Katana ca implementare a standardului **OWIN** (prin intermediul căruia s-au adăugat modulele auxiliare de autentificare sau notificare și care permite o extensibilitate simplă în server-side)
* Three.JS (bibliotecă de javascript open-source ce funcționează ca layer de abstractizare peste standardul **WebGL** și care permite utilizarea unor primitive pentru realizare de obiecte grafice in browser)

Pe lângă acestea, au fost implementate în particular:

* O bibliotecă javascript ce furnizează diferite module pentru interfațarea cu utilizatorul (precum generarea de tabele, popup-uri, tooltip-uri, grafice, ferestre, containere pentru elemente, color-picker și acces la API-urile aplicației web prin RPC sau SignalR). Motivația din spatele implementării acesteia este cauzată de necesitatea existenței unor funcționalități de bază fără introducerea unui overhead pe care l-ar fi introdus utilizarea unor biblioteci generale existente, precum și expunerea unor metode parțial specifice aplicației de care restul componentelor să se folosească.
* O bibliotecă ce funcționează în orice aplicație .NET Standard (> 2.0) care oferă funcționalități de bază pentru un client al API-ului REST expus de Geoserver (precum tipuri de dată specifice obiectelor utilizate în cererile HTTP, endpoint-urile serverului sau mecanisme de construcție a cererilor)

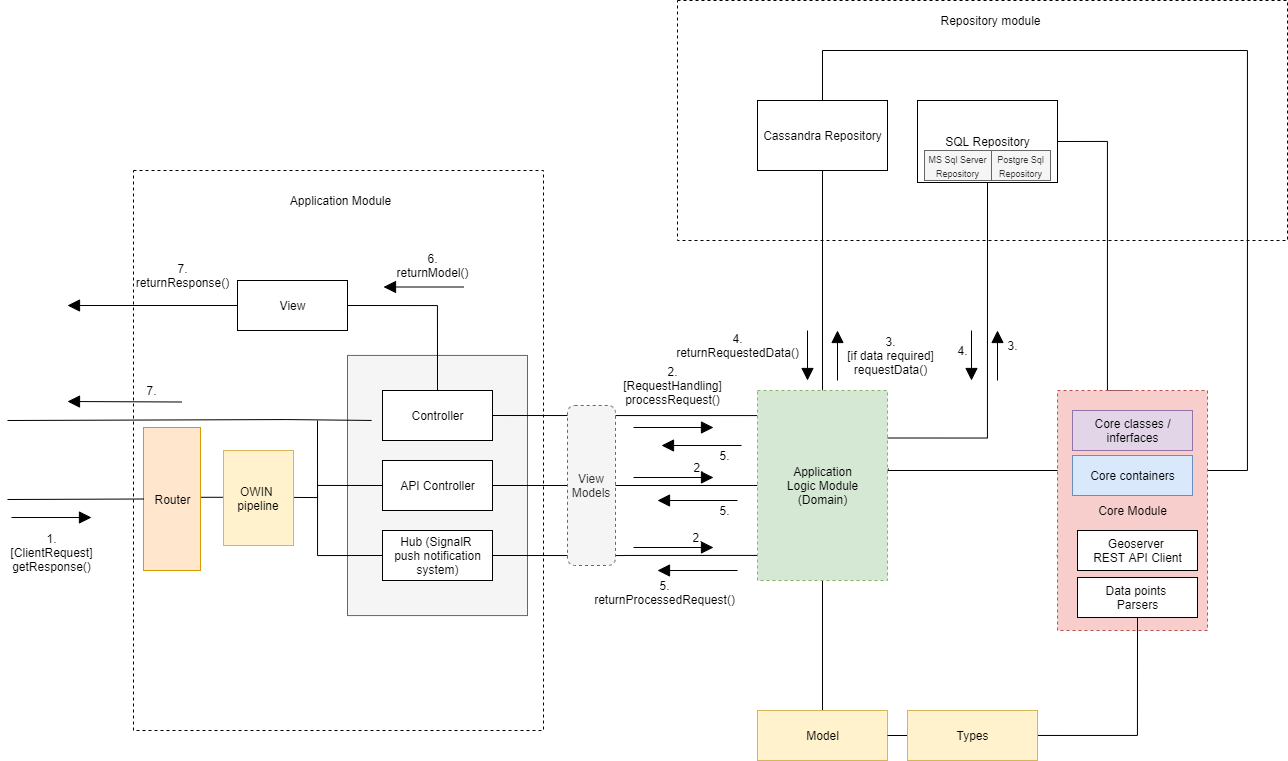
### Prezentarea API-ului bibliotecii de javascript

În tabelul de mai jos sunt prezentate principalele entități implementate de bibliotecă în vederea furnizării funcționalităților amintite anterior. Modul de utilizare a acestora se bazează pe furnizarea unor parametrii la apelul funcțiilor expuse de module sau în baza creării unor obiecte ce conțin informații despre modul în care modulele trebuie să reacționeze în momentul apelării metodelor expuse de acestea. Cel din urmă procedeu asigură extensibilitatea API-ului, precum și asigură compatibilitatea cu versiuni precedente în cazul în care noi versiuni ale acesteia sunt dezvoltate.

|  |  |  |
| --- | --- | --- |
| Denumire modul | Descriere | Dependințe |
| Cards | Asigură funcționalități in vederea generării de ferestre (frame-uri) în cadrul cărora pot fi furnizate informații.  Acest lucru este realizat prin intermediul unui manager ce gestionează ciclul de viață al acestora | jQuery (versiune minim v.3.3.1) |
| List | Asigură funcționalitate pentru implementarea unor liste generice ale căror conținut poate fi customizat. De asemenea, pune la dispozitie callback-uri pentru afișarea elementelor pentru care conținutul a fost preluat asincron prin intermediul apelării unor API-uri a unor servicii web | jQuery (versiune minimă v.3.3.1) |
| Table | Asigură funcționalitate pentru implementarea unor tabele generice (ce dețin funcționalități de bază precum paginare, adăugare, actualizare). În același timp permite afișarea de date încărcate asincron, utilizând același principiu ca modulul List. | jQuery (versiune minimă v.3.3.1) |
| Plot | Utilizat pentru crearea de grafice 2d în vederea vizualizării datelor, precum și a customizării acestora sau aplicării de funcții și operații asupra datelor afișate. | jQuery(versiune minimă 3.3.1)  d3(versiune minimă 4) |
| Popup | Asigură funcționalitate pentru gestionarea de popup-uri în cazul aplicației. Conținutul popup-ului poate fi furnizat în urma unor apeluri asincrone către API-urile unor servicii web care să le furnizeze conținutul. | jQuery (versiune minimă v.3.3.1) |
| Tooltip | Reprezintă un modul ce asigură și gestionează apariția de tooltipuri ce pot fi utilizate în vedere asigurări unor guideline-uri utilizatorului aplicației | jQuery (versiune minimă v.3.3.1) |
| Utils | Asigură funcționalitate metode și funcționalități generice (parsare, interpretare, ș.a) | jQuery (versiune minimă v.3.3.1) |

### Prezentarea structurii bibliotecii client Geoserver

### Descrierea arhitecturii server

 Designul arhitecturii serverului, așa cum este prezentat in diagrama de mai jos, a fost gândit astfel incât să fie asigurată o mentenabilitate și extensibilitate ridicată a aplicației întrucât sistemul implică coordonarea și comunicarea între o multitudine de standarde și tehnologii.

Construcția arhitecturii este realizată în baza unui model domain driven care presupune concentrarea principală a întregului sistem în jurul domeniului aplicației, fapt ce conduce la o înțelegere mai simplă pentru ceea ce înseamnă dezvoltare tehnică în baza unor explicații în limbaj specific (non-tehnic). Din acest punct de vedere, modulul Application Logic reprezintă centrul de interes în cadrul căruia sunt implementate principalele flow-uri utilizând doar abstracțiuni ale entităților, denumit generic servicii, care furnizează logica și comportamentul evenimentelor, al căror interfețe și API-uri sunt descries în modulul Core. În acelasi timp, în core sunt reținute înregistrările tuturor serviciilor utilizate în cadrul domeniului. Infrastructura de bază (modelele și tipurile de date utilizate) provin din cele două layere Model si Types, în cadrul cărora sunt construite entitățile principale care sunt folosite în vederea specificării metodelor și funcțiilor de la nivelul serviciilor.

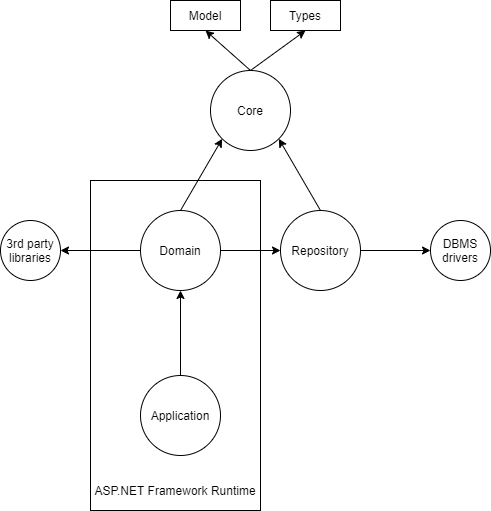
Principalul șir logic al evenimentelor constă în preluarea cererii de la client (prin intermediul unui layer denumit aplicație) și redirecționarea acestuia către domeniu în baza unor informații existente in contextul cererii. În cadrul acestui flow, după ce este realizată invocarea modulul central (domain) în cadrul căreia se realizează procesarea, sunt construite diferite obiecte model necesare construirii răspunsului; a nu se confunda cu modelele aplicației, aceste entități servesc rolul de view-model sau controler-model (în cazul unor răspunsuri ce nu necesită crearea unui view care să fie întors la client). Arhitectura modului aplicație constă intr-o organizare clasică **MVC** cu precizarea că modelul a fost înlocuit cu entitățile view-model sau controler-model anterior menționate. Trebuie precizat însă că că fiecare cerere care ajunge în cadrul layer-ului aplicație, poate fi manipulat în baza a 3 tipuri de situații:

1. Este invocat un controler în cazul clasic al cărui scop al cererii este obținerea unui view în vederea afișării acestuia pe ecran (și prezentării utilizatorului).
2. Este realizat o cerere asincronă în vederea obținerii de informații suplimentare sau alte date în vederea completării view-ului, context în care invocarea este asupra unui api-controler al cărui metode sunt invocate sub forma unui remote procedure call, la un endpoint configurat în momentul pornirii serverului.
3. Comunicarea este realizată în ambele direcții (client-server) prin intermediul unui canal de comunicație inițializat de către client. Acest canal folosește în principiu WebSockets ca protocol la nivel aplicație.

Informații suplimentare cu privire la modul de funcționare al modului aplicație precum și detalii cu privire la aceste scenarii de funcționare vor fi furnizate în secțiunile ce urmează.

#### Infrastructura și tipurile de dată principale

Înainte de a prezenta rolul și modul de funcționare al fiecărui layer din cadrul arhitecturii, este necesar explicarea dependințelor intra-module. Fiecare dintre layere sunt implementate sub forma unor biblioteci al cărui target este .NET Standard (minim 2.0); cu alte cuvinte, acestea pot fi folosite în cadrul oricărei aplicații ce funcționează într-un runtime .NET ce suportă standardul anterior amintit. Excepții de la această regulă însă sunt layerele Application si Domain care sunt dependente de runtime-ul de .NET Framework 4.7, fapt al cărui consecințe vor fi detaliate mai mult în partea de concluzii și probleme în implementare.

**** În graful orientat din figura de mai jos, sunt reprezentate dependințele layerelor. Așa cum a fost descris și în partea de incipit a acestei părți, layerul de bază, critic din punct de vedere al posibilități de modificare al API-ului, este reprezentat de core. Următorul layer ca importanță (din același punct de vedere) este layerul Repository în care sunt regăsite implementările surselor de informații bazate pe funcționalitățile necesare descrise în Core. De asemenea, există dependințe față de biblioteci 3d party care implementează drivere pentru comunicarea cu sistemele de gestiune ale bazelor de date. În cazul aplicației prezentate, sunt folosite drivere pentru comunicația cu MSSQL Server, Cassandra si PostgreSQL.

Domeniul reprezintă un layer inferior pe această ierarhie a dependințelor. Aceasta depinde de toate celelalte layere întrucât este responsabil de injectarea dependințelor aplicației, cât și de facilitarea funcționalităților către layerul aplicație, aflat cu o scară mai jos în ierarhie. În cadrul ambelor straturi, este necesar existența runtime-ului de ASP.NET întrucat la acest nivel se realizează găzduirea capacităților de server web.

Luând in considerare descrierea acestor dependințe, în continuare vor fi explicate în linii mari principalele tipuri de dată construite ca bază pentru toate layerele existente. Referințe sau trimiteri către denumirile acestora vor fi reluate de-a-lungul explicațiile realizate pentru fiecare layer din arhitectură.

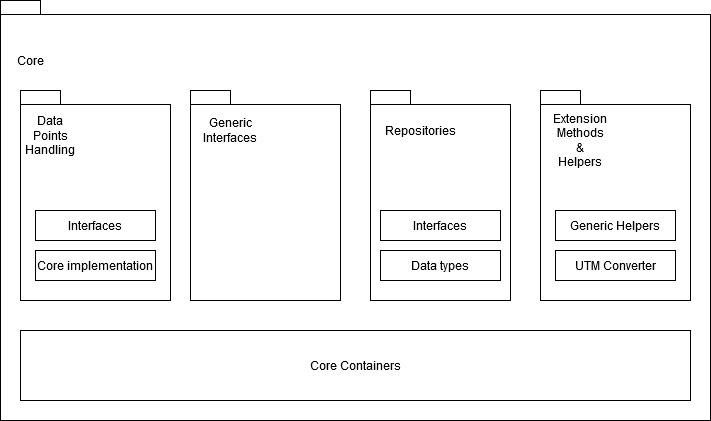
Principalele structuri sunt reprezentate de clasele

* **PointsDataSetHeader:** abstractizează o entitate ce deține metadate despre seturile de puncte geografice ce reprezintă subiectul acestei aplicații. Metadatele presupun proprietăți precum identificatorul, utilizatorul care deține setul de date, numele setului, statusul încărcării și disponabilității, sursa, validitatea, latitudinea/longitudinea/înălțimea/deformația/deviația standard maximă/minimă.
* **User**: reprezintă un utilizator și are asociate în principal câmpuri necesare autentificării și autorizării. Rolurile pe care un user le poate avea sunt cele definite în cadrul cerințelor funcționale ale aplicației.
* **ColorMap:** clasă folosită pentru a modela entitățile necesare implementării unui procedeu de colorarea al punctelor geografice în momentul randării acestora pe hartă. Pentru facilitarea exprimării în cadrul explicațiilor, această clasă va fi adesea denumită și ‘paletă de culori’ .

Denumirile și operațiunile asociate modelelor au fost alese astfel încât să se mapeze pe logica domeniului.

#### Layerul Core

Așa cum este prezentat și mai sus, acest layer reprezintă abstracția de bază a aplicației alături de cele două module asociate modelelor și tipurilor de date de bază. În cadrul acesteia, sunt declarate interfețele repository-urilor și serviciilor integrate în domeniul aplicației și extensii ale metodelor ce pot fi folosite în cadrul celorlalte layere. De asemenea, tot acest layer este responsabil pentru furnizarea container-ului **DI** în cadrul căruia au fost asociate serviciile folosite în aplicație. Inițializarea acestui container reprezintă responsabilitatea domeniului.

 O previzualizare a acestui layer este realizată prin intermediul diagramei de module de mai jos, în care sunt scoase în evidență în mod generic toate subcomponentele.

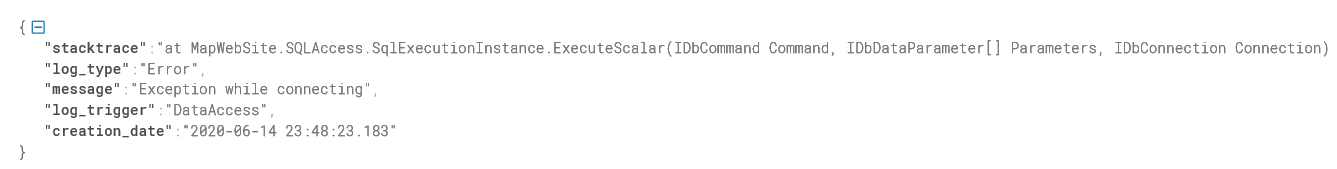
În tabelul de mai jos sunt prezentate interfețele principalelor repository-uri, interfețe ce sunt implementate de entitățile din layerul repository.

|  |  |
| --- | --- |
| IUserRepository | |
| Descrie funcționalitatea unei entități responsabile cu gestiunea și furnizarea datelor despre utilizator ( informații necesare în cadrul autentificării, autorizării sau navigării în aplicație), precum și datelor asociate acestora (precum metadate despre seturile de puncte geografice create și utilizate, sau informații despre paletele de culori pe care un utilizator le-a incărcat). Nu sunt impuse specificații sau detalii pentru modalitatea de stocare sau sistemul de gestiune folosit. | |
| Metodă/Proprietate | Descriere |
| InsertUser | Pune la dispoziție inserarea unui utilizator în repository |
| UpdateUser | Implementează actualizarea datelor despre un utilizator |
| GetUser | Metoda aceasta poate fi folosită pentru a prelua informații asociate cu un utilizator |
| GetUsersFiltered | Această metodă poate fi folosită în vederea preluării filtrate a utilizatorilor (filtrele posibile sunt nume utilizator, email, nume de familie, prenume, dar pot fi adăugate și altele ) |
| GetUsersCount | Furnizează numărul de utilizatori din repository ( poate fi furnizat și numărul în urma aplicării anumitor filtre ) |
| GetUserByEmail | Returnează un utilizator pe baza emailului |
| SetEmail | Actualizează emailul unui utilizator |
| SetEmailConfirmed | Actualizează confirmarea emailului unui utilizator |
| GetUserRoles | Metoda poate fi folosită in vederea obținerii rolurilor unui utilizator |
| CheckUser | Metoda poate fi utilizată în vederea validării credențialelor unui utilizator |
| GetUserHashedPassword | Această metodă pune la dispoziție preluarea parolei (hash concatenat cu salt) unui utilizator |
| GetUserAssociatedDatasetsCount | Această metodă oferă informații despre seturile de puncte asociate unui utilizator |
| AddPointsDatasetToUser | Această metodă poate fi folosită în vederea adăugării unui set de puncte (metadate/headerul setului de puncte) în setul de puncte vizibile unui utilizator |
| RemovePointsDatasetFromUser | Metoda poate fi utilizată în vederea ștergerii unui set de puncte din colecția unui utilizator |
| CreateUserPointsDataset | Această metodă poate fi utilizată în vederea creării unui set de puncte (metadate) ce va fi stocat pe numele unui anumit utilizator precizat ca parametru |
| RemovePointsDataset | Această metodă poate fi utilizată pentru ștergerea unui set de puncte din baza de date. Ștergerea poate fi logică sau definitivă, nu se impun restricții asupra acestui aspect |
| RaiseToGeoserverDataset | Reprezintă o extensie a metodei CreateUserPointsDataset ce poate fi utilizată pentru upgrade-ul unui set de puncte în vederea asocierii acestuia cu un set de puncte stocat intr-un provider Geoserver. |
| GetDatasetID | Preia ID-ul asociat unui set de puncte (pe baza numelui și numelui utilizatorului care a creat setul) |
| GetGeoserverDatasetID | Preia ID-ul din geoserver pentru un set de puncte dat |
| GetGeoserverDefaultColorPaletteID | Preia ID-ul unei palete de culori default cu care este asociat un set de puncte (asociat cu geoserver). Este presupusă încărcarea și existența paletei în Geoserver (ca și stil) |
| GetDatasetHeader | Obține metadatele (headerul) unui set de puncte încărcat in repository |
| GetDataSetsFiltered | Obține seturi de puncte (header) în baza unor filtre |
| GetDatasetsCount | Furnizează numărul de seturi de puncte (se pot aplica filtre) |
| GetColorMapID | Obține ID-ul unei palete de culori create de un utilizator |
| GetColorMapsFiltered | Obține date despre paletele de culori utilizate în aplicație în baza anumitor filtre |
| GetGeoserverColorMaps | Preia toate paletele de culori care sunt asociate cu stiluri din geoserver |
| GetColorMapSerialization | Preia serializarea unor palete de culori în vederea utilizării acestora pentru colorarea seturilor de puncte |
| InsertGeoserverColorMap | Pune la dispoziție inserarea unui stil asociat în Geoserver sub forma unei palete de culori. Repository-ul nu se va ocupa de validarea existenței stilului și nici nu va monitoriza actualizarea sau ștergerea acestuia in Geoserver. |
| CreateColorMap | Crează o paletă de culori asociate unui utilizator. Modul de stocare al culorilor va fi sub forma unei serializări JSON |
| GetColorMapsNames | Preia numele tuturor paletelor de culori pentru un anumit utilizator |
| UpdateDatasetStatus | Actualizează statusul unui set de date (header). Statusurile posibile sunt Uploaded (încărcat, reprezintă prima stare în care un set de date ar trebui să se afle), Generated (reprezintă starea în care un set de date este pregătit în vederea utilizării), Pending (așteptare, această stare poate semnifica că setul de date este procesat sau indisponibil temporar), UploadFail (semnifică o eroare la încărcare, dacă este cazul), GenerateFail (semnifică o eroare în momentul generării, dacă este cazul). În cazul existenței unui alt caz decat cele anterior menționate, se poate utiliza statusul None. |
| UpdateDatasetLimits | Actualizează limitele unui set de date. Poate ajuta în vederea creșterii performanței randării setului de puncte. |
| UpdateDatasetRepresentationLimits | Actualizează date precum înălțimea minimă sau maximă, deformarea minimă sau maximă din cadrul headerului unui anumit set de puncte |
| IDataPointsRepository | |
| Această interfață are ca scop descrierea funcționalități unui provider de puncte propriu-zise. Dacă în cadrul IUserRepository existau informații ce țin de header-ul unui set de puncte, un repository care va implementa metodele acestui model va avea ca scop furnizarea de puncte geografice propriu zise cât și detalii despre fiecare punct din setul de date. Nu există precizări în ceea ce privește modalitatea de stocare, însă se recomandă ca viteza de răspuns al fiecărei metode să fie cât mai mare iar implementarea să fie asincronă/paralelizată daca este posibil. | |
| InsertPointsDataset | Această metodă va pune la dispoziție un mecanism de inserare unui set de puncte. Este recomandat paralelizarea acestei metode cât și luarea în considerare a dimensiunilor mari pe care seturile de puncte inserate le pot ocupa |
| GetBasePoints | Preia doar punctele împreună cu informațiile de bază (latitudine, longitudine, înălțime, deviere standard, deformare). Această metodă este folosită în principiu în vederea randării pe ecran a punctelor. |
| GetPointDetails | Pune la dispoziție detalii despre un anumit punct geografic aflat la o anumită coordonată geografica (sau în baza identificatorului punctului) |
| GetRegions | Preia doar punctele împreună cu informațiile de bază (latitudine, longitudine, înălțime, deviere standard, deformare).  Această metodă este folosită în principiu în vederea randării pe ecran a punctelor. Spre deoserbire de GetBasePoints, preluarea se face din cadrul unei regiuni predefinte (in baza rândului, coloanei și a nivelului de zoom pentru care este realizată cererea) |
| GetRegion | Preia doar punctele împreună cu informațiile de bază (latitudine, longitudine, înălțime, deviere standard, deformare) dintr-o anumită regiune. Această metodă este similară cu GetRegions iar performanțele ar trebui să fie cel puțin la fel de rapide. |
| ILogsRepository | |
| Reprezintă de asemenea o interfață de bază în cadrul aplicației întrucat pune la dispoziție mecanisme de logare a erorilor sau informațiilor ce au loc în cadrul întregului sistem adaptate la modulele/layerele acestuia. Orice eroare sau nesincronizare a datelor ar trebui stocată utilizând un repository ce implementează metodele acestei interfețe pentru a ușura procesul de depanare și fixare. Accesul la acestea vor fi accesibile administratorului sistemului. | |
| LogInfo | Poate fi folosit în vederea stocării unui mesaj de informare (precume început sau terminarea unui eveniment sau adăugarea cu succes a unor date în cadrul repository-urilor) |
| LogWarning | Orice eveniment care nu se comportă conform algoritmul sau șirului de evenimente normal, dar care nu este critic pentru rularea sistemului ar trebui logat folosind această metodă |
| LogError | Orice eveniment ce poate genera probleme în ceea ce privește interacțiunea cu utilizatorul și răspunsurile cererilor (precum inactivitatea – încetarea unui anumit serviciu sau repository) trebuie să fie logate folosind această metoda. |

O remarcă importantă în cadrul realizări logurilor, este aceea că în cadrul fiecărui log se precizează nivelul la care a avut loc eroarea sau mesajul de informare. În acest sens, există 7 posibile taguri ce pot fi asociate fiecărui :

* Core: erorile marcat cu acest tag au loc in layerul Core
* CoreModule: acest tag desemnează faptul că o eroare a avut loc din pricina modelelor sau a unor module atașate de Core, dar care nu sunt în Core
* Controlers: prin acest tag se prezintă loguri provenite din controlerele layerului Application (mai multe detalii despre acestea în descrierea layerului Application)
* DataAccess: desemnează erori provenite din cadrul layerului Repository sau a oricărui modul/serviciu ce furnizează date
* DataServices: reprezintă erori ce sunt logate din cadrul unor servicii
* Domain: tag asociat erorilor produse în Domeniu
* Other: orice alt caz ce nu se încadrează în scenariile menționate mai sus

Un exemplu de logare ce poate fi stocat în cadrul unui repository ce se ocupă de loguri poate conține informații asemănătoare cu cele prezentate in poza de mai jos.

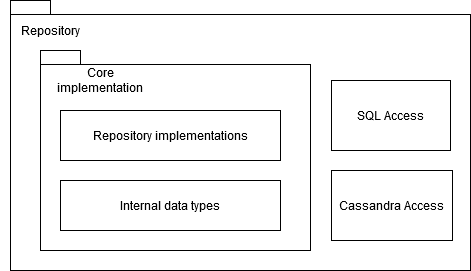


#### Layerul Repository

Când vine vorba despre cuvantul ‘repository’, acesta se referă generic la un loc central în care datele sunt stocate și păstrate. Prin Repository ar trebui să se înțeleagă clase sau componente care încapsulează logica necesară accesării surselor de date. Acestea centralizează funcționalitatea comună de acces la date, oferind o mai bună întreținere și decuplarea infrastructurii sau tehnologiei utilizate pentru accesarea bazelor de date din stratul modelului de domeniu.

Adesea, în cadrul unei astfel de componente există un ORM care să faciliteze modul în care sunt utilizate driverele ce oferă acces la sistemele de gestiune a datelor. Un ORM (object relational mapper) reprezintă un produs software ce asigură o mapare relațională de obiecte utilizând o tehnică de programare pentru convertirea datelor între sisteme de tip incompatibile folosind deobicei limbaje de programare orientate pe obiecte. Deși sunt extrem de utili și pot ușura timpii de implementare al pachetelor software, un aspect negativ al celor mai multe implementări de ORM-uri este reprezentat de timpii mari de execuție în vederea obținerii datelor, cauzate din pricina aspectului generic al acestora. Din acest motiv, în cadrul acestui layer nu au fost folosite ORM-uri pentru accesul la date, ci s-a preferat o implementare ‘bear-metal’ a query-urilor în baza unor wrappere simplificate peste driverele de C#, adaptate pentru aplicația noastră.

Trecând de acest aspect al metodologiei de implementare, layerul descris în această secțiune este folosit deoarece oferă avantajul de a permite o concentrare pe logica de persistență a datelor, mai degrabă decât pe instanțele și procedurile de acces la date, în cadrul celorlalte layere (acest lucru este evidențiat cu precădere la domeniu).

 În figura de mai jos este prezentat o privire de ansamblu a acestui layer.

Funcționalitatea acestui layer se concentrează în jurul unor wrapere peste driverele sistemelor de gestiune ale bazelor de date implementate pentru aplicațiile scrise in C#. In principiu, aceste wrappere trebuie să faciliteze execuția de interogări pentru clasele din cadrul pachetului ‘Core implementation’, iar în caz de eroare, trebuie să gestioneze logarea unor mesaje generice ce pot facilita administrarea aplicației. Interfața unui astfel de wrapper constă în principal din 3 metode:

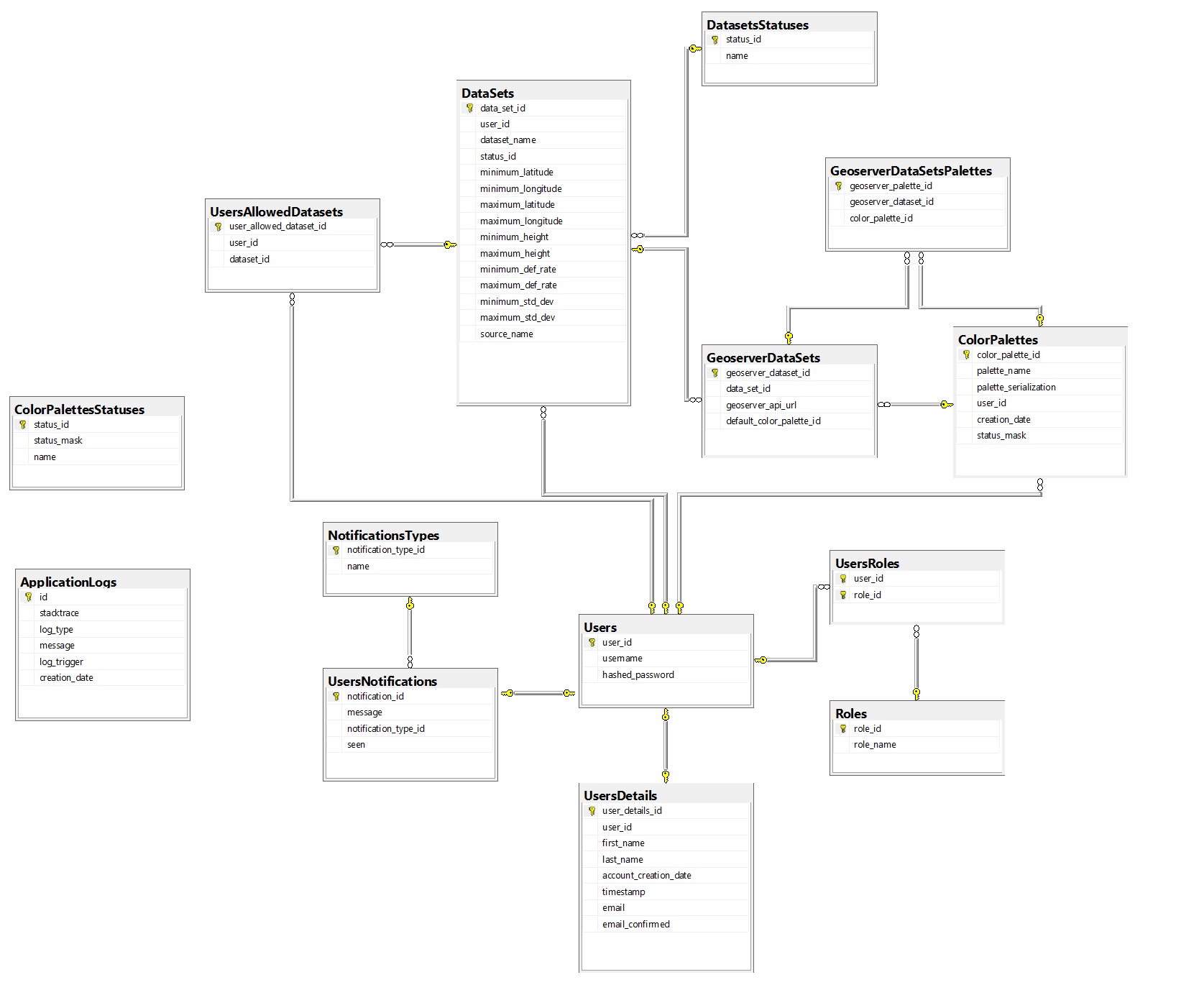
* ExecuteQuery: permite realizarea de operațiuni ce au ca scop returnarea de date din cadrul bazei de date.
* ExecuteNonQuery: permite operațiuni create în vederea realizării de operațiuni update sau delete
* ExecuteScalar: această funcție poate realiza interogări care returnează un singur rezultat

În vederea implementării interfețelor din Core, s-a ales utilizarea sistemului de gestiune MS SQLServer pentru stocarea datelor despre utilizatori (IUserRepository) și a logurilor aplicației (ILogsRepository), iar pentru managementul punctelor (IdataPointsRepository) s-au realizat două repository-uri diferite ce folosesc sistemele Cassandra respectiv PostgreSQL.

Criteriul de alegere al acestora a fost condiționată de scenariile de utilizare, respectiv modul si numărul de date ce pot fi stocate.

##### Modelul relațional

În cadrul gestionării utilizatorilor și a logurilor, un sistem relațional (cum este SQL) aduce avantajul unui model simplu, deoarece nu necesită nici o procesare complexă de structurare sau interogare. Nu implică procese de structurare sau definirea ierarhică a bazelor de date. Deoarece structura este simplă, este suficient să fie tratate cu interogări SQL simple și nu necesită să fie proiectate interogări complexe. Datele rezultate pot fi modificate pe baza valorilor din orice coloană, pe orice număr de coloane, ceea ce permite utilizatorului să recupereze fără efort datele relevante ca rezultat. Un alt punct forte al acestui mod de implementare este fiabilitate relațională dintre tabelele din baza de date care ajută la evitarea înregistrărilor imperfecte, izolate sau fără legătură. În diagrama de mai jos este reprezentat o vedere de ansamblu al tabelelor în care sunt stocate datele despre loguri și utilizatori.



Pe de altă parte, în cadrul stocării punctelor s-a decis utilizarea unei abordări ce să nu implice existența unor relații complexe între tabelele/entitățile care stochează informațiile. Procedând în această manieră, proiectarea bazei de date se concentrează în jurul obținerii unor timpi de răspuns rapizi în momentul preluării informațiilor. În acest sens, una din abordarea implementării repository-ului pentru puncte implică utilizarea unui sistem de gestiune no-sql care oferă un mecanism de stocare și recuperare a datelor care este modelat în alte mijloace decât relațiile tabulare utilizate în bazele de date relaționale, acestea fiind utilizate din ce în ce mai mult în aplicațiile web de date mari și în timp real.

##### Modelul nerelaționar

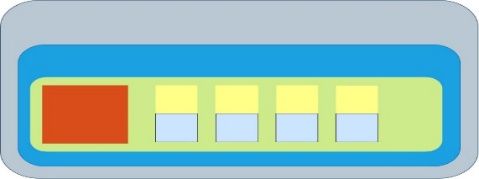
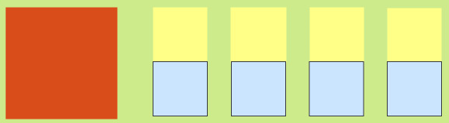
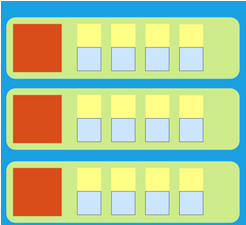
În continuare se vor detalia câțiva pași ce au fost parcurși în momentul proiectării bazei de date nonrelaționale pentru a scoate în evidență utilitatea acesteia. Înainte de a detalia modalitatea de gândire a bazei de date utilizate de repository, vor fi prezentate câteva detalii tehnice cu privire la capacitățile Cassandrei din punct de vedere de al modului de operare cu structurile de date interne.

Modelul de date Cassandra poate fi abstractizat în patru elemente de bază:

**Coloane**: O coloană este cel mai mic element al structurii de date Cassandra. Este o pereche care constă dintr-un nume, valoare și un timestamp.

**Rânduri**: Un rând grupează impreuna coloanele cu aceleasi proprietati. Fiecare rând primește o cheie care identifica unic randul.

**Familii de coloane**: Acestea sunt pentru Cassandra echivalentul unui tabel pentru sistemele de tipul SQL. Familiile cu coloane conțin rânduri și fiecare rând este gestionat printr-o cheie.

******Keyspace**: Un Keyspace este aproximativ echivalentul unei scheme sau a unei baze de date în sistemele SQL. Spațiile cheie conțin colecții de familii de coloane.

Administrarea datelor în Cassandra folosește o abordare bazată pe interogare, în care interogările specifice sunt cheia organizării datelor. Interogările sunt rezultatul selectării datelor dintr-un tabel; schema este definiția modului în care sunt aranjate datele din tabel. Proiectarea bazei de date Cassandra se bazează pe cerința de citire și scriere rapidă, astfel încât, cu cât este mai bună proiectarea schemei, cu atât datele mai rapide sunt scrise și preluate. Indexarea datelor poate duce la performanța sau degradarea interogărilor, astfel încât înțelegerea indexării este un pas important în procesul de modelare a datelor.

Spre deosebire de metodologia abordată în momentul creării unei baze de date relaționale cum ar fi MSSQL sau PostgreSQL, în cazul Cassandrei tehnica a diferit când vine vorba de abstractizarea datelor. Demersul s-a concentrat spre crearea unui model rigid, dar optimizat, care să profite de avantajele arhitecturii interne ale Cassandrei. Principali pași ai metodologiei au fost:

1. *Înțelegerea fluxului de lucru*

În cadrul unei baze de date nerelaționare, în loc să se înceapă dezvoltarea cu crearea modelului de date (a tabelelor), abordarea este orientată înspre abstractizarea fluxului de date în cadrul aplicației și a unor tipare ce se repetă în funcționalitatea aplicației. Această abordare, „design-first” presupune analiza modului în care entitățile se comportă în aplicație, precum și felul în care utilizatorul are nevoie de date. Din punct de vedere tehnic, acest lucru constă în analiza și realizarea unor diagrame de activități din care să reiasă toate acțiunile posibile ce pot fi inteprinse.

În cadrul repository-ului ce furnizează punctele geografice, de interes este fluxul de date ce constă în vizualizarea unor puncte la un moment dat într-o limită delimitată de coordonate de start si stop, încadrate într-un dreptunghi de dimensiune variabilă și obținerea de detalii la cererea utilizatorului, precum și capacitatea acestor puncte de a-și expune doar anumite proprietăți în vederea supunerii lor la anumite acțiuni (luăm aici ca exemplu actul colorării lor, care se poate realiza în funcție de înălțime sau deformare)

1. *Crearea și modelarea interogărilor*

Această etapă presupune în principiu construirea interogărilor (sau conceperea unui pseudocod pentru acestea) în baza fluxului de date stabilit la pasul 1. Această etapă se realizează prin transpunerea unui limbaj informal într-un limbaj formal care mai tarziu va fi transpuns în limbajul de interogare. Un exemplu de query ce trebuie realizat arată ca în figura de mai jos.

select

latitude,

longitude,

number,

height

from points\_by\_dataset

where

dataset\_id = :dataSetID

and latitude >= :leftLatitude

and longitude >= :leftLongitude

and latitude <= :rightLatitude

and longitude <= :rightLongitude

allow filtering

1. *Proiectarea tabelelor și determinarea cheilor primare*

Această finală etapă presupune construirea tabelelor în baza interogărilor ce au fost realizate. Acestea vor fi denormalizate, nu vor suporta operații de join, consistența lor va admite erori sub un anumit prag, aceste lucruri reprezentând un trade-off pentru performanță. Recomandarea principală este ca tabelele să se bazeze pe entități cu o referință ușoară (în cazul nostru - punctele). Cheile vor fi desemnate sau alese în funcție de clauzele interogărilor, acest lucru permițând efectuarea căutării rapide a intrărilor în baza unui hash.

O astfel de tabelă construită este reprezentată în figura de mai jos

create table **points\_by\_dataset**(

dataset\_id int,

number int,

longitude decimal,

latitude decimal,

…

observations text,

displacements list<FROZEN<points\_displacements>>,

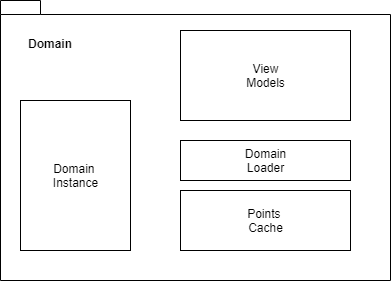
PRIMARY KEY((dataset\_id, number), longitude, latitude)

) WITH CLUSTERING ORDER BY(longitude desc, latitude desc);

#### Layerul Domain

În ingineria programării, cuvântul domeniu se referă în mod obișnuit la zona de aplicație pentru care este destinată aplicația. Din punct de vedere tehnic, un domeniu se traduce prin clase și structure de date, care prin funcționalitatea lor descriu funcționalitatea aplicației cu toate procesele, elementele, relațiile și interdependențele lor. Conceptul de DDD se bazează pe principiul că toți participanții la dezvoltarea proiectului, fie aceștia dezvoltatori sau nu, se concentrează pe domeniul de aplicabilitate al software-ului. Acest lucru conduce la faptul că atât specialiști din domeniu, cât și programatorii trebuie să aibă cunoștințe comune în ceea ce privește dezvoltarea funcționalități aplicației. În cadrul acestei aplicații, acest lucru poate fi esențial deoarece vizualizarea datelor sau fluxul acestora este complexă având în vedere numărul de sisteme implicate. Rezultatul acestei abordări este un model de domeniu care mapează toate relațiile funcționale și prezintă cunoștințele domeniului într-o formă abstractă și structurată.

Așadar, acest layer desemnează logica de bussines a aplicației și conține regulile de nivel superior pentru modul în care datele interacționează între ele pentru a crea și modifica fluxul de informații. Modul de implementare, așa cum este reprezentat în schema de mai jos se concentrează pe existența unei abstracții principale ce conține metode pentru săvârșire evenimentolor din cadrul fluxului.



Entitatea de bază din acest modul este reprezentat de DomainInstance în cadrul căruia sunt expuse preponderant metode de prelucrare sau manipulare a punctelor geografice. Logica metodelor se folosește de obiecte ce implementează interfețele din Core descrise în secțiunile anterioare, inițializate în cadrul unui Domain Loader, unic la nivelul contextului aplicației și care are ca scop principal încărcarea domeniului.

public static void Load()

{

/\*Inject the logs repository in core containers\*/

Core.CoreContainers.LogsRepository = new SQLLogsRepository();

/\*Inject the users repository in core containers\*/

Core.CoreContainers.UsersRepository = new SQLUserRepository();

/\*Inject the main datapoints repository in core containers\*/

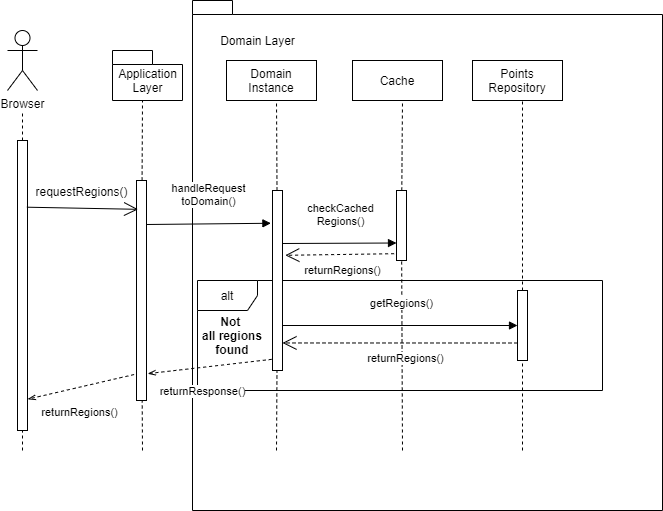
Core.CoreContainers.DataPointsRepository = CassandraDataPointsRepository.Instance;

Core.CoreContainers.DataPointsRegionSource = new PowerOfTwoRegionsSource();

Core.CoreContainers.LogsRepository.LogInfo("Domain loaded", Core.Database.Logs.LogTrigger.Domain);

}

O altă abstracție care este regăsită în diagrama acestui layer este un cache pentru punctele geografice. Acesta este utilizat ca un mecanism de accelerare al răspunsului către client, în cazul în care furnizarea punctelor (in scopul afișării) este realizată din baza de date Cassandra, sau orice alt furnizor care să nu expună posibilitatea directă de a afișa datele în interfață. Ordinea simplificată a fluxului de date în cazul unui astfel de context poate fi vizualizat în cadrul diagramei de secvență de mai jos. În acest caz, browserul este cel care inițiază comunicația prin intermediul unor mecanisme de comunicație asincrone, întrucat preluarea datelor din repository poate să dureze timpi se. Protocolul folosit pentru a realiza acest lucru este WebSockets, iar modalitatea de returnare a datelor se face prin intermediul unor callback-uri din layerul aplicație ce vor acționa la rândul lor asupra unor callback-uri existente in scripturile ce rulează pe browserul utilizatorului.



La nivelul domeniului, pseudocodul pentru această operațiune poate fi exprimat astfel:

* + Preia headerul setului de date din Repository
  + Daca headerul nu exista atunci returneaza mesaj de eroare
  + Redu dimensiunea extremitatilor (lat/long) cererii la dimensiunea extremitatilor descrise in header
  + Preia cheile regiunilor pe baza extremitatilor
  + Preia regiunile din cache
  + Daca toate regiunile au fost preluate
    - Atunci returneaza raspunsul prin callback
  + Altfel
    - Cere regiunile din repository
    - Dacă sunt erori in preluare returneaza mesaj de eroare
    - Stocheaza regiunile in Cache
    - Returneaza raspunsul prin callback

Implementarea acestui pseudocod este realizată în cadrul metodei RequestDomainPointsRegions primind ca parametrii de intrare pozițiile din stânga sus, respective dreapta jos a dreptunghiului ce descriu zona cu puncte cerută (atenție, o astfel de zonă poate conține mai multe regiuni), nivelul de zoom la care este realizată cererea, date pentru preluarea headerului setului de puncte (nume de utilizator și numele setului), o listă cu cheile regiunile deja cache-uite în client și care nu mai sunt necesare pentru a fi returnate, precum și o funcție de callback pentru întoarcerea punctelor. Pentru baza implementării cache-ului a fost folosită clasa MemoryCache (tip de dată <thread save>, valabil în toate runtime-urile de .NET Framework > 4.0 și în runtime-ul de .NET 5) cu politica de înlocuire Time aware least recently used.

Alte metode ce sunt existente la nivelul blocului de logică sunt operații clasice de CRUD ce sunt wrapper pentru metodele repository-urilor (acest lucru asigură decuplarea layerului Application de layerul Repository), dar și metode ce au ca scop ‘sincronizarea’ anumitor elemente din sistem, sau activarea anumitor servicii. Acestea sunt descrise succint în tabelul de mai jos.

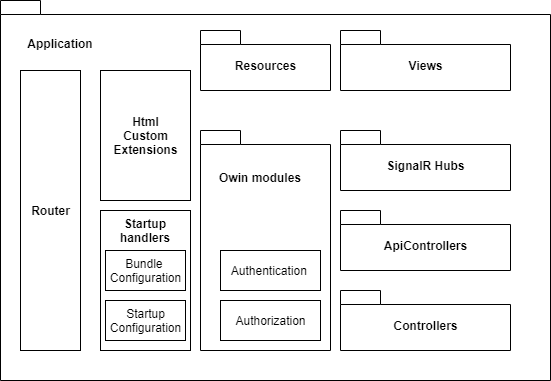
|  |  |
| --- | --- |
| Denumire/Ansamblu Funcții | Funcționalitate |
| UploadDataFiles | Permite uploadarea de fișiere ce conțin date despre punctele ce pot fi vizualizate în aplicație.  Această funcție se ocupă de logica de merge(unire) în ordinea corectă a blocurilor de date ce sunt trimise secvențial de la client pentru a putea fi preluate și parsate de către un serviciu ce inserează datele în sursa de date (în cazul versiunii aplicației prezentate în acest document, inserarea se realizează in Cassandra) |
| CreateDataset,  GetDataset,  UpdateDatasetStatus, GetUsersAssociatedDatasets,  ChangeUsersAssociatedDatasets | Acest ansamblu de operații reprezintă în principiu wrappere peste acțiunile de manipulare a seturilor de puncte sau a metadatelor despre acestea.  Operațiile nu sunt asincrone. |
| GetUser, GetUsersCount | Metode ce sunt wrappere pentru acțiunile de manipulare a datelor despre utilizator. Este important de menționat că administrarea autentificării sau autorizări utilizatorilor nu este realizată în cadrul acestor funcții sau a acestui layer. |
| GetColorPalettes  GetColorPaletteSerialization  InsertColorPalettes | Reprezintă metode ce sunt wrapper pentru acțiunile de manipulare a paletelor de culori. |
| ValidateGeoserverLayer | Conține implementarea unei secvențe de cod ce are ca scop validarea existenței unui stil creat în Geoserver pe baza unei palete create în aplicație sau crearea unui stil nou folosind un stil creat în aplicație. Această funcționalitate utilizează biblioteca client pentru API-ul REST al Geoserver ce a fost implementată și descrisă în secțiunile anterioare. |
| RequestPointDetails | Metodă ce funcționează ca modalitate de furnizare a detaliilor despre anumite puncte. Această metodă comută între sursele de date existente/înregistrate în aplicație în baza parametrilor veniți din partea cererii. |

O ultimă atribuție a acestui layer este aceea de a pune la dispoziție ‘modele de prezentare’ (view-models) pentru paginile din cadrul layerului Aplicație.   
În general, modelul de prezentare (sau vizualizare) este o abstractizare a vizualizării care expune proprietățile și comenzile publice. Altfel spus, reprezintă clase care reprezintă modelul de date utilizat într-un view specific. Din punct de vedere al arhitecturii construite în cadrul aplicației prezentate în acest document, aceste entități se află la granița dintre layerul Application si layerul Domain.

#### Layerul Application

Cel din urmă nivel in ierarhia arhitecturii prezentate, dar nu cel mai puțin important, este reprezentat de acest layer, în cadrul căruia se realizează comunicarea directă cu clientul. În cadrul conceptului de DDD serviciul de aplicații este un strat subțire care se află în strictă legătură cu modelul de domeniu și coordonează activitatea aplicației. Nu conține logică de bussiness și nu deține starea unei entități. Cu toate acestea, trebuie să se remarce faptul că acest layer nu este complet decuplat de modulele de modele sau tipuri de date, acesta reprezentând unul din motivele pentru care entitățile menționate nu fac parte izolat din niciun layer; din pricina faptului că includerea acestora în layere ar fi condus la un overhead prea mare în momentul prelucrării datelor.

În vederea implementării acestui nivel, a fost utilizat frameworkul MVC oferit de .NET Framework 4.7, platformă care pune la dispoziție instrumente și biblioteci (extensii pentru metode, clase și structuri) special pentru construirea aplicațiilor web. Cele mai importante funcționalități puse la dispoziție sunt sintaxa de ‚template’ a paginilor web, cunoscută sub numele de Razor, pentru construirea de pagini web dinamice folosind C #, biblioteci pentru ușurința utilizări specificațiilor din protocolul HTTP, un sistem de autentificare care include biblioteci, o bază de date inclusiv autentificarea cu mai mulți factori.

 O vedere de ansamblu pentru modul în care frameworkul a fost adaptat aplicației noastre, poate fi vizualizată în diagrama de mai jos.

În continuare vor fi descrise principalele module. Din perspectiva unei arhitecturi MVC clasice, componentele de bază regăsite și în acest layer sunt reprezentate de routere, controlere și view-uri.

##### Router-ul

O rută definește aspectul fiecărui URL expus de aplicația web și informațiile despre manipularea acestora. Toate rutele configurate ale unei aplicații sunt stocate într-o tabelă de rutare și vor fi utilizate de motorul de rutare al frameworkului pentru a determina clasa de gestionare adecvată sau fișierul pentru o solicitare primită. Modulul de routare pentru această aplicație este bazată pe accesul la trei tipuri de rute:

* Rute pentru controlerele de bază, acestea nu sunt prefixate și sunt de forma <numeControler>/<metodă>, responsabile în principiu pentru obținerea de pagini web (view-uri) sau de secvențe de cod html necesar randării
* Rute pentru satisfacerea unei metode de remote procedure call, al cărui șablon este rpc/<numeApiControler>/<metodă>. Utilizarea acestor metode este recomandată în momentul în care sunt necesare resurse după ce pagina a fost încărcată (spre exemplu cereri asincrone realizate prin intermediul API-ului AJAX implementat în browsere)
* O rută specială pentru inițializarea comunicației cu hub-ul din cadrul bibliotecii(frameworkului) SignalR pentru o comunicare statefull. SignalR se folosește de mai multe mecanisme de comunicație, selectând automat cel mai bun mecanism de comunicație disponibil, având în vedere capacitățile clientului și ale serverului.

##### Componentele OWIN

Conform documentației oficiale, OWIN reprezintă un standard ce definește o interfață standard între serverele web .NET și aplicațiile web. Motivul principal pentru apariția acestui standard, este decuplarea serverului și a aplicației, prin implementarea unor module simple, reutilizabile în baza standardului .NET, dar și stimularea ecosistemului open source al instrumentelor de dezvoltare web .NET. Implementarea standardului pe care am folosit-o în acest proiect este Katana, versiune ce a fost dezvoltată de Microsoft cu sprijinul comunități, sub licență open source.

Din punct de vedere al fluxului datelor, modulele OWIN funcționează ca niște entități intermediare situate la intrarea în aplicație, ce au ca scop filtrarea cererii. Acest lucru este folosti în vederea autentificării și autorizării utilizatorului în momentul în care acesta realizează o cerere HTTP.

În vederea implementării, s-a ales suprascrierea funcționalități framework-ului Identity care la bază funcționează ca un middleware OWIN inserat în pipeline-ul aplicației. Pentru suprascriere, a fost necesară crearea unor clase ce să extindă o serie de interfețe, al căror implementare e explicată în tabelul de mai jos.

|  |  |
| --- | --- |
| \*pentru furnizarea de date, a fost utilizat repository-ul de utilizatori descris in layerul Core | |
| Interfață | Descriere |
| IUserStore<User, string> | Reprezintă principala interfață necesară suprascrierii modulului . În cadrul acesteia sunt definite metodele pentru crearea, actualizarea, ștergerea și preluarea utilizatorilor. |
| IUserPasswordStore<User, string> | Definește metodele pentru stocarea și verificarea parolelor |
| IUserRoleStore<User,string> | Definește metodele ce trebuie implementate în vederea mapării unui utilizator cu un rol. |
| IUserEmailStore<User,string> | Definește metodele utilizate înspre gestiunea adreselor de e-mail ale utilizatorului. |

##### Controler-ele

Cea mai simplă definiție a acestor clase poate fi că reprezintă obiecte care nu fac nimic altceva decât să gestioneze solicitările browserului. Principalul lor scope este de a prelua datele din cerere, pasarea acestor date către layerele superioare, iar în cele din urmă returnarea unui răspuns sub diferite forme, fie obiecte serializate, fie view-uri. Pentru aplicația aceasta organizarea (sau gruparea) controlerelor formează următoarele structuri:

* Controlere de uz general: răspund pentru returnarea de pagini de uz general sau pentru secvențe de cod html care să fie folosit în cadrul unor controale (tooltipuri, message-box-uri sau tabele)
* Controlere pentru detaliile punctelor: reprezintă controlere ce returnează view-urile necesare în cadrul afișării interfeței utilizate înspre gestionarea sau filtrarea seturilor de puncta. De asemenea, acest grup conțin și API-urile folosite în vederea obținerii informațiilor filtrate sau sortate.
* Controlere pentru setări: acest grup conține controlere ce administrează paginile de setări din cadrul aplicației. Fiecare pagină de setări are asociată metode în această secțiune, ce pot fi folosite în vederea furnizării de date la server, sau pentru a realiza apeluri de la distanță în vederea unor validări sau a altor date.
* Controlere pentru pagina principală: conține controlerele ale căror metode sunt asociate paginii principale aplicației (cea în care este randată harta), dar reprezintă și un leant pentru furnizarea paginilor de setări.
* Controlere pentru autentificare: autentificarea și înregistrarea utilizatorului este gestionată prin intermediul acestor controlere care conțin metode de înregistrare, autentificare, validare a parolei , confirmare sau validare a datelor introduse de către utilizator în controale.

Tot in cadrul acestui modul au fost definite tipuri de date auxiliare, care să ușureze extensibilitatea și mentenabilitatea componentei vizuale a aplicației. Pentru exemplificare, în tabelul următor sunt reprezentate câtorva din structurile create în vederea gestionări anumitor elemente vizuale, care în cadrul furnizării unor cereri de actualizare sau aducerea unor noi funcționalități asupra aplicației, facilitează implementarea.

|  |  |
| --- | --- |
| Denumire structură | Descriere |
| \*valorile posibile sunt valabile pentru versiunea aplicației descrise în acest document | |
| Tooltip | Gestionează numele și codul tuturor tooltipurilor disponibile în aplicație. Acestea pot fi obținute prin RPC la metoda GetTooltip sau GetGifTooltip din cadrul controlerului de uz general. Valorile posibile sunt: ChoseMapTypeTooltip, ChosePointsSizeTooltip, ChoseColorCriteriumTooltip, SearchTooltip, PointsSource, PlotWindow |
| Page | Este o structură ce gestionează paginile disponibile în aplicație. Valorile disponibile sunt: About, Account, CreateColorPalette, UseGeoserverPalette, UploadPoints, UseGeoserverLayer, ManageUsers, ManageDatasets |
| PaletteColorsHue | Reprezintă o structură al cărui valori posibile reprezintă nuanțe de culori. Valorile posibile sunt: Red, Orange, Yellow, Green, Lightblue, Blue, Pink, Black |
| DefaultPalette | Reprezintă o structură de date al cărui valori reprezintă ‘palete default’ de colorare. Valorile posibile sunt: Crown, Default, Fox, Kuller |

##### View-urile

În contextul unei arhitecturi MVC, prin view se înțelege entitatea desemnată pentru descrierea modului în care arată (este randată) interfața aplicației, în baza unei sintaxe HTML sau în acest caz, a unui limbaj particular ce facilitează scrierea limbajului de marcare. Așa cum a mai fost descris, acesta primește date de la controlerul și le ambalează și le prezintă browserului pentru afișare. Prin asociere, această componentă este cea mai puțin autonomă entitate din întreaga aplicație, întrucât este strict dependent de viewmodel și de controler. Tot cu acest layer poate fi asociată și preluarea datelor, întrucât pe baza a ceea ce este trimis către browserul utilizatorului, acesta va realiza alte cereri către controlere.

Din punct de vedere structural, organizarea acestor elemente se poate împărți în:

* Layout-uri: reprezintă principală ‘gazdă’ a paginilor, iar în cadrul aplicației poate avea două forme diferite, o formă fiind folosită în contextul paginilor de înregistrare și autentificare, iar alt aspect în contextul navigării pe hartă
* Pagini: reprezintă o resursă ce este utilizată în principiu în combinație cu oricare din cele două layouturi. Utilizarea acestora presupune încărcarea în regiunea special destinată din cadrul layoutului. Exemplu de pagini sunt cele utilizate pentru setări sau autentificare.
* Secțiuni: reprezintă elemente reutilizabile la nivelul paginilor, dar care arată diferit în funcție de contextul acțiunii utilizatorului. Un astfel de exemplu îl reprezintă meniul aplicației.
* Controale: sunt elemente reutilizate în întreaga aplicație, realizate customizat (astfel de controale sunt tooltipuri, message-box-uri sau tabele). Acestea sunt acționate utilizând elementele din cadrul părții de client, ce va fi descrisă în următoarea parte a acestui document.

### Descrierea arhitecturii client

## Tehnici utilizate pentru implementarea modulelor

## Prezentarea aplicației

## Testarea aplicației

https://softwareengineering.stackexchange.com/questions/140999/application-layer-vs-domain-layer

# Rezultate

# Concluzii

# Bibliografie

* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>
* <https://land.copernicus.eu/user-corner/technical-library/egms-specification-and-implementation-plan>
* insar.ngu.no
* https://tremaps5.tre-altamira.com/treaviewer/Index.aspx
* <http://www.mapas.abe.bo/deslizamientos/mapa.html>
* <https://docs.datastax.com/en/landing_page/doc/landing_page/current.html>
* http://owin.org/

Contents

[Introducere 6](#_Toc43259033)

[Motivație 6](#_Toc43259034)

[Scopul și obiectivele lucrării 7](#_Toc43259035)

[Structura lucrării 7](#_Toc43259036)

[Sumarul rezultatele obținute 7](#_Toc43259037)

[Interferometria cu reflectori permanenți 7](#_Toc43259038)

[Metodologia detecției profilelor de deformare cu anomalii 13](#_Toc43259039)

[Aplicația PSTOOL 13](#_Toc43259040)

[Definirea cerințelor 14](#_Toc43259041)

[Cazuri de utilizare 18](#_Toc43259042)

[Arhitectura aplicației 18](#_Toc43259043)

[Prezentarea API-ului bibliotecii de javascript 19](#_Toc43259044)

[Prezentarea structurii bibliotecii client Geoserver 20](#_Toc43259045)

[Descrierea arhitecturii server 22](#_Toc43259046)

[Infrastructura și tipurile de dată principale 24](#_Toc43259047)

[Layerul Core 25](#_Toc43259048)

[Layerul Repository 31](#_Toc43259049)

[Modelul relațional 33](#_Toc43259050)

[Modelul nerelaționar 34](#_Toc43259051)

[Layerul Domain 37](#_Toc43259052)

[Layerul Application 41](#_Toc43259053)

[Router-ul 42](#_Toc43259054)

[Componentele OWIN 43](#_Toc43259055)

[Controlerele 44](#_Toc43259056)

[View-urile 46](#_Toc43259057)

[Descrierea arhitecturii client 46](#_Toc43259058)

[Tehnici utilizate pentru implementarea modulelor 46](#_Toc43259059)

[Prezentarea aplicației 46](#_Toc43259060)

[Testarea aplicației 46](#_Toc43259061)

[Rezultate 46](#_Toc43259062)

[Concluzii 46](#_Toc43259063)

[Bibliografie 46](#_Toc43259064)