**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI**

**Lucrare de licență**

**Analizarea vegetației din imagini satelitare**

**Absolvent**

**Viezure Bogdan-Costin**

**Coordonator științific**

**Lector Pătrașcu Andrei**

**București, iunie 2022**

**Rezumat**

Proiectul constă în mai multe componente ce funcționează împreună pentru a pune bazele unei aplicații ce ar fi folosită în domeniul agriculturii. Munca de teren poate fi ajutată de date care să ajungă la utilizatori direct, fără a mai fi nevoie de o analiză fizică a culturilor. Una dintre metode este analizarea datelor obținute din imagini satelitare.

Componentele funcționează împreună pentru a analiza și stoca date obținute din imagini satelitare open-source trecute prin filtrul de vegetație. În starea curentă, aplicația generează date geografice și statistici referitoare la nivelul de vegetație verde a parcelelor de pământ.

Conform unui studiu intitulat “Agricultura în Germania – statistici și date” publicat de  [Evgenia Koptyug](https://www.statista.com/aboutus/our-research-commitment/425/evgenia-koptyug) în Aprilie 2022, doar în Germania suprafața declarată de pamânt folosit în agricultură ajunge la 16.6 milioane de hectare. Satelitul ce pune la dispoziție imaginile acționează în Europa și Nordul Africii ceea ce înseamnă că este nevoie de o bază de date care poate stoca și folosi cantități mari de date.

Deși baza de date folosită pe parcursul dezvoltării proiectului nu se ridică la performanțele unei baze de date ce ar fi folosită în producție, faptul că poate fi integrată și folosită în proiect susține ideea că poate fi înlocuită cu orice altă bază de date, preferabil mai performantă, care folosește aceleași tehnologii.

**Abstract**

The project consists in a number of components that work together to make the back-end for an application that could be used in agriculture. Agriculture realted field work could be suported by data that would be accesible for all users without the need to go out on the field to obtain the data. One of the ways to achieve this is analising data obtained from satellite images.

The components work together to analyze and store data obtained from satellite images with a vegetation filter applied over them. In its current state, the application generates geographical data and statistics concerning the level of live green vegetation contained within plots of land.

According to a study with the title “Agriculture in Germany – statistics and data” published by Evgenia Koptyug in April 2022, only in Germnay alone the registered area of land used for agriculture amounts to 16.6 milions of hectares. The satellite supplying the images covers Europe and North Africa which means the potential amount of data requires a database that can store and use big amounts of data.

Although the database used while developing the project does not have the hardware capabilities of a real database that would be used in a production environment, the fact that it can be integrated and used in the projects supports the fact that it can be replaced with with any other database, preferably with better hardware, that uses the same tools and technologies.

**Cuprins**

[**Capitolul I** 4](#_Toc106315759)

[**I. Introducere** 4](#_Toc106315760)

[**I.1. Motivația lucrării** 4](#_Toc106315761)

[**I.2. Scopul aplicatiei** 4](#_Toc106315762)

[**I.3. Descrierea NDVI** 4](#_Toc106315763)

[**Capitolul II** 6](#_Toc106315764)

[**II. Descrierea aplicației** 6](#_Toc106315765)

[**II.1. Componenta de clasificare** 6](#_Toc106315766)

[**II.2. Baza de date** 22](#_Toc106315767)

[**II.3. Back end** 28](#_Toc106315768)

[**Capitolul III** 28](#_Toc106315769)

[**III.1. Concluzii** 28](#_Toc106315770)

[**III.2. Contribuții personale** 29](#_Toc106315771)

[**III.3. Direcții de viitor** 29](#_Toc106315772)

# **Capitolul I**

## **I. Introducere**

### **I.1. Motivația lucrării**

Motivația lucrării este faptul că domeniul agriculturii este foarte puțin susținut de aplicații accesibile pentru orice utilizator care să faciliteze deșfășurarea activităților. Există aplicații de gestionare a cheltuielilor, a datelor, a angajaților dar încă nu este pusă la dispoziție o platformă care sa susțină munca pe teren. În Romania, agricultura are o valoare de 5% din produsul intern brut si angrenează 30% din populația activă opcupată ceea ce înseamnă ca o aplicație care ar pune la dispoziție functionalități noi în acest domeniu ar putea fi de folos industriei și celor angrenați.

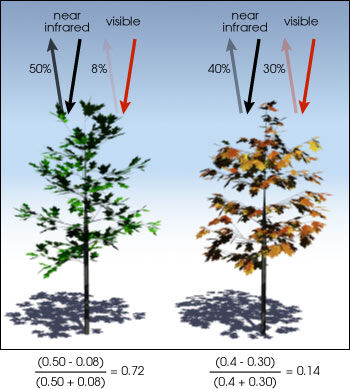
### **I.2. Scopul aplicatiei**

Scopul aplicației este de a analiza parcele de pamant pentru a facilita activiatea de teren a fermierilor. De asemenea, aplicația este concepută spre a fi folosită de orice utilizator ce dorește detalii despre vegetația dintr-o zonă anume. Pot fi fermieri independeți, proprietari ai unor întreprinderi sau chiar și persoane ce folosesc pamântul pentru scopuri personale, nu economice. Proiectul se bazază pe poze satelitare ce sunt analizate cu ajutorul componentei python de clasificare. Deoarece componenta python este relativ lentă, din cauza algoritmilor pe care ii rulează, este nevoie de stocarea datelor pentru un acces mai rapid. De exemplu, pentru o parcelă de pământ de 0.5 kilometrii pătrați, algoritmul împarte parcela în 70 de parti diferite, fiecare cu o multitudine de puncte, categorizate in funcție de indicele NDVI. Din motive de scalabilitate este nevoie de implementarea unei baze de date NOSql distribuita. Am ales să folosesc Hadoop ca tehnologie pentru baza de date. Folosind Hadoop pot fi stocate datele rezultate din analiza unei parcele din mai multe date, indiferent de dimensiune.

### **I.3. Descrierea NDVI**

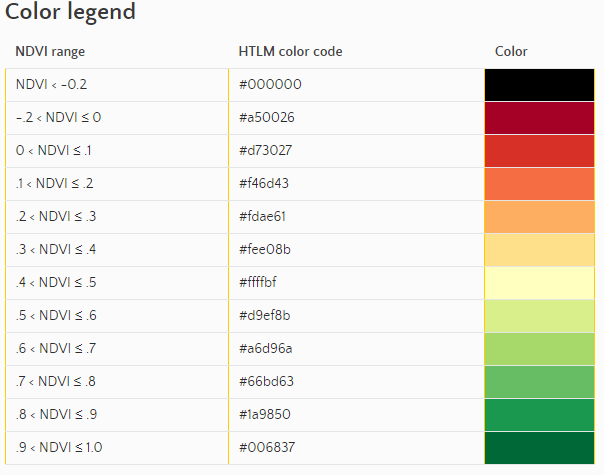
NDVI (normalized difference vegetation index) este un indicator grafic care este folosit pentru a verifica dacă zonele din imagini, de cele mai multe ori obținute de sateliți, conțin vegetație verde vie. Vegetația sănătoasă, datorită clorofilei pe care o conțin, reflectă mai multă lumină verde și lumină infra-roșie de apropiere (NIR: near-infrared) dar absorb mai multă culoare roșie și albastră. Lumina infra-roșie este o radiație electromagnetică cu lungimi de undă mai mari decăt cele ale luminii vizibile. Lumina infra-roșie poate fi împărțită în subdiviziuni una dintre ele fiind lumina indra-roșie de apropiere. Astfel, lumina infra-roșie este invizibilă pentru ochiul uman. Din această cauză oamenii văd vegetația cu clorofilă doar în culoarea verde. Totuși senzorii folosiți de sateliți precum Landsat și Sentinel-2 au capabilitatea de a vedea lumina infra-roșie reflectată de clorofilă dar și cantitatea de lumină roșie vizibilă. Cu cât este mai multă lumină infra-roșie si mai puțină lumină roșie, inseamnă că suprafața are mai multă clorofilă.

Indicatorul Ndvi este calculat folosind formula: , unde NIR este lumina infra-roșie si Red este lumina roșie. În figura I.3.1. se poate observa modul în care indicele Ndvi este calculat în funcție de lumina absorbită



(Figura I.3.1. Absorpția luminii de clorofilă, Sursă NASA)

Indicatorul poate lua valori intre –1 și 1. Valorile ce se apropie de –1 corespund corpurilor de apă. Valorile din vecinătatea și interiorul intervalului (-1, 0.1) corespund solurilor neroditoare: nisip, zapadă sau stânci, zonelor urbane, clădirilor sau drumurilor. Valorile pozitive mici, din intervalul (0.2, 0.4) reprezintă zone acoperite de arbuști și pajiști. Valorile ce se apropie de 1 corespund pădurilor din zonele temperate și tropicale. Bucățile de pământ pentru care concepută componenta de clasificare au în general valori de peste 0.1. Cu ajutorul tabelului din figura I.3.2 se poate face legătura între culoare și valoarea indicelui Ndvi.



(Figura I.3.2. Tabel Ndvi, Sursă <https://custom-scripts.sentinel-hub.com/sentinel-2/ndvi/>)

# **Capitolul II**

## **II. Descrierea aplicației**

### **II.1. Componenta de clasificare**

#### **II.1.1.Introducere**

Componenta de clasificare este un serviciu scris in python ce expune 2 endpoint-uri. Acestea returnează ca raspuns un json, respectiv un json linked list ce contine date referitoare la clasificarea parcelei în 3 clase, în funcție de indicele NDVI.

* [http://{ip}:{port}/ndvi/api/json/v1/ndvi-classification](http://127.0.0.1:5000/ndvi/api/json/v1/ndvi-classification)
* http://{ip}:{port}/ndvi/api/jsonld/v1/ndvi-classification

În exemplul din lucrare serviciul ruleaza pe localhost, pe portul 5000. Componenta este indepentă ceea ce inseamnă ca poate fi integrată în alte proiecte.

Motivele pentru care este folosit python ca limbaj sunt faptul că python este un limbaj în care tipurile sunt stabilite dinamic și existența unor biblioteci care au facilitat dezvoltarea componentei.

#### **II.1.2. Biblioteci importante folosite**

Flaskeste un micro-framework web scris în Python. Este clasificat ca microframework deoarece nu necesită instalarea altor biblioteci sau tool-uri. De asemena am folosit ca extensie peste Flask, flask\_restx, datorită suport-ului pentru Swagger. Cele 2 endpoint-uri sunt documentate cu Swagger la [http://{ip}:{port}/](http://127.0.0.1:5000/ndvi/api/json/v1/ndvi-classification) pe baza unui json scris manual ce este interpretat de Swagger.

Cele 2 metode sunt expuse cu ajutorul adnotărilor puse la dispozitie de Flask:

@name\_space.route("/api/json/v1/ndvi-classification")

class JsonApi(Resource):

@name\_space.route("/api/jsonld/v1/ndvi-classification")

class JsonldApi(Resource):

Cele 2 rute expun 2 metode de tip POST ce primesc in body-ul request-ului un JSON cu 2 atribute obligatorii:

* Primul atribut ‘polygonCoordinates’ conține coordonatele (latitudine și longitudine) punctelor care determină parcela de pământ ce va fi analizată. Un exemplu ar fi cel din Swagger: "27.199243,45.910026,27.209468,45.911885,27.209607,45.906525,27.200563,45.904793".
* Al doilea atribut ‘clientId’ este un token static folosit ca o formă simplă de autentificare. Id-ul este: "36aef768-3076-43ad-8566-4818fd82c73c"

Prin apelarea metodei run ce primește ca argumente ip-ul si portul este pornit serviciul de clasificare

flask\_app.run(host='0.0.0.0')

Portul standard este portul 5000 dar poate fi schimbat. Pentru punerea în producție o sugestie este rularea serviciului într-un container Docker care sa expună serviciul pe localhost iar mașina în care se află containerul să mapeze un port expus public către container.

**Pillow**

Pillow este o bibliotecă folosită pentru a adăuga capabilități de procesare a imaginilor interpretorului Python. Am folosit doar modulul Image al bibliotecii ce conține o clasă cu același nume care este folosită pentru a reprezenta o imagine Pillow. Am folosit funcția open pentru a deschide imaginea și pentru a efectua operații asupra imaginii. Majoritatea operațiilor efectuate asupra imaginilor sunt din biblioteca OpenCv, biblioteca pillow are funcții mai simple dar care sunt de preferat în locul celor din OpenCv în unele cazuri în care se doresțe efectuare unor operații simple.

**OpenCV-Python**

OpenCV este o bibliotecă de funcții de programare cu algoritmi de machine learning, computer vision și procesare de imagini. Biblioteca fost inițial dezvoltată de Intel. Biblioteca este independentă de platformă si gratis de folosit cu licență open-source Apache 2.

OpenCV are suport pentru mai multe limbaje precum C++, **Python** sau Java și poate fi folosită pe mai multe platforme precum Windows, Linux, Os X, Android, iOS etc. Varianta de OpenCv pentru python este numită cv2.

Procesarea imaginilor reprezintă efectuarea unor operațiuni asuprea unei imagini pentru a obține o imagine mai bună sau pentru a extrage informații folositoare. O imagine poate fi definită ca o funcție bidimensională f(x, y), unde x și y sunt coordonate de spațiu iar valoarea funcției este intensitatea culorii sau a culorilor ce definesc spațiul de culoare respectiv. Cu alte cuvinte, o imagine este o matrice, bidimensională pentru imagini alb-negru sau tridimensională pentru imaginile colorate.

#### **II.1.3.Descrierea serviciului**

Pentru a explica modul in care funcționează componenta vom analiza procesul incepând în ordine de la primirea datelor inițiale pană la generarea și returnarea json-ului de output. Singura diferență între cele două tipuri de json-uri returnate de cele 2 endpoint-uri este modul în care acestea sunt construite. Pentru a explica modul de funcționare a aplicației se va face referire la endpoint-ul pentru json simplu: [http://{ip}:{port}/ndvi/api/json/v1/ndvi-classification](http://127.0.0.1:5000/ndvi/api/json/v1/ndvi-classification).

Primul pas este verificarea token-ului. Token-ul din request trebuie sa se potrivească cu cel setat în Properties.py. Apoi, prin apelarea funcției *getBBOXFromParcelCoordinates,* ce primește ca date de intrare coordonatele parcelei din request se returnează bounding box-ul definit de aceste puncte.

Bounding Box (prescurtată si BBOX) este o zonă dreptunghiulară definită de 2 puncte (adica 2 perechi latitudine- longitudine). In cazul nostru avem nevoie de bounding box-ul ce contine toate puntele primite la input. Pentru a determina cele 2 puncte ale bounding box-ululi funcția extrage longitudinea maximă si minimă și latitudinea maximă si minimă, vezi anexa[1].

De asemenea, funcția apelează epsg4326ToEpsg3857(coordinatesBBOX)care trece coordonatele din input din sistemul epsg4326 în sistemul epsg3857. În proiect este definită si funcția inversă.

##### **II.1.3.1.Serviciile Terrascope**

Următorul pas este obținerea imaginii ce va fi analizată prin folosirea a două servicii puse la dispoziție de platforma terrascope.be.

Terrascope este o platformă dezvoltată în Belgia ce oferă acces nerestrcționat pentru toți utilizatorii printre altele la date satelitare. Satelitul Sentinel-2 obține date optice pentru Europa și Africa de Nord. Cu ajutorul datelor de la Sentinel 2 Terrascope aplică un filtru NDVI asupra imaginilor satelitare. Exemplul din figura II.1.3.1.1este o imagine decupată de pe platforma terrascope de vizionare a flitrelor de vegetație <https://viewer.terrascope.be/> cu filtrul NDVI aplicat asupra satului Cucuieți, județul Teleorman si asupra zonelor limitrofe.

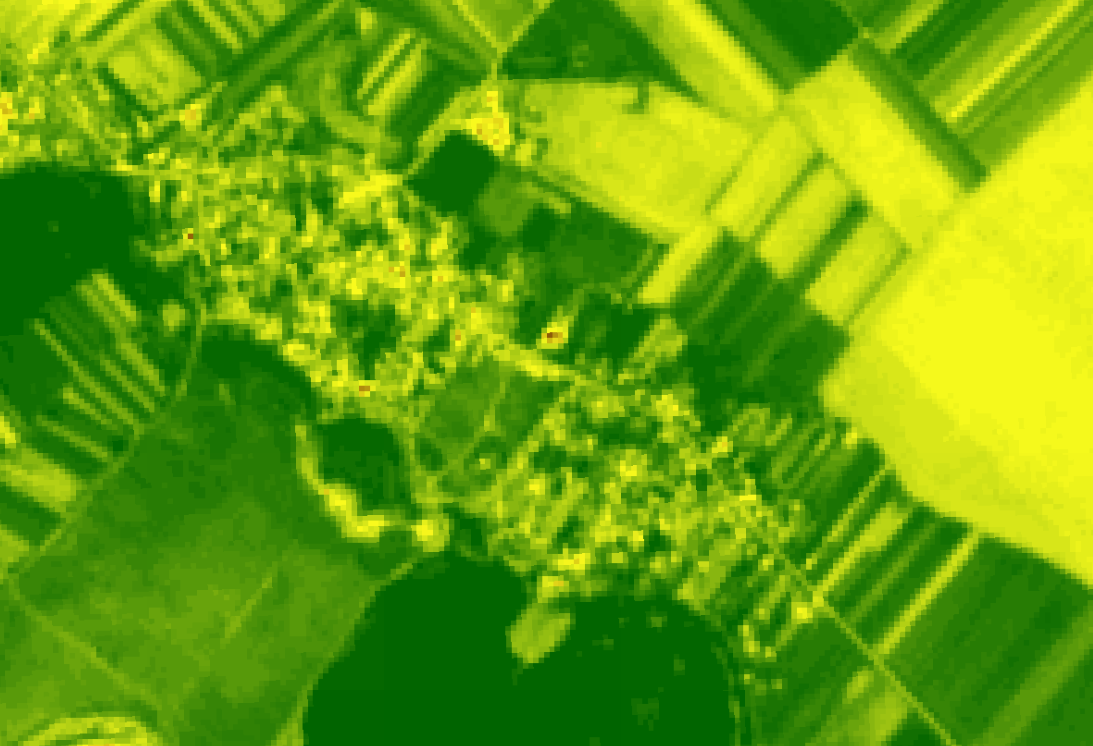


Figura II.1.3.1.1.Exemplu de imagine Ndvi

**Serviciul pentru imagini NDVI**

Unul dintre serviciile terrasope folosit este <https://services.terrascope.be/wms/v2> de tip GET. Cu ajutorul acestui endpoint se poate obține o imagine ca cea din exemplul anterior fără a mai accesa și decupa manual imaginea de pe platformă. Printre parametrii mai importanți se regăsesc:

* layers: stratul care va fi aplicat, in cazul de față: CGS\_S2\_NDVI
* format: în aplicație este folosit image.png
* srs: sistemul de coordonate folosit, EPSG:3857
* time: ziua în care a fost facută poza (terrascope stochează datele culese din mai multe zile), parametrul este calculat cu ajutorul celuilalt serviciu folosit
* bbox: bounding box-ul calculat mai sus
* height: înălțimea imaginii
* width: lățimea imaginii

Înălțimea și lățimea imaginii sunt calculate cu ajutorul funcției getDistanceFromLatLonInM de la anexa [2].

Funcția returnează distanța dintre două puncte definite în sistemul epsg4326 exprimată in metrii. Astfel pentru a obține de la serviciul terrascope o imagine cu dimensiuni proporționale cu dimensiunea reală a parcelei definită de bounding box-ul respectiv, parametrii height și width sunt dați de dimensiunile celor 2 laturi exprimate în metri. Un pixel din imagine va reprezenta aproximativ 1 de metru pătrat.

În figura urmatoare returnată de serviciul terrascope se poate observa că în unele cazuri imaginea cea mai recentă pentru suprafața dorită este neclară din cauza norilor. Vezi Figura II.1.3.1.2.

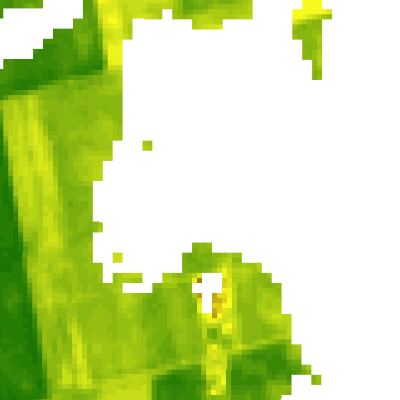


Figura II.1.3.1.2- Exemplu de imagine Ndvi acoperită de nori

**Serviciul pentru acuratețea imaginilor**

Datele ce ar rezulta din analiza unei astfel de imagini, sunt irelevante deoarece o bucata prea mare din parcelă nu este vizibilă. Pentru a evita această situație sunt folosite datele obținute din serviciul <https://services.terrascope.be/timeseries/v1.0/ts/TERRASCOPE_S2_NDVI_V2/geometry> de tip POST. Request-ul primește 2 parametrii: startDate și endDate și un body de tip json ce conține date referitoare la bucata de pamănt pentru care sunt solicitate informatiile. Serviciul retunează detalii despre imaginile pentru aria de pământ capturate în perioada definită de startDate și endDate. De exemplu pentru startDate=2022-04-31 și endDate=2022-05-31 și coordonatele [[[27.199243,45.910026],[27.209468,45.911885],[27.209607,45.906525],[27.200563,45.904793],[27.199243, 45.910026]] care corespund parcelei din Figura II.1.3.2.1, pentru fiecare zi din intervalul definit pentru care există o fotografie cu parcela este returnat numărul total de pixeli, numărul de pixeli vizibili și media indicelui NDVI. La anexa [3] se poate observa o parte din răspuns.

Pe baza acestor date se alege ziua a cărei imagine va fi analizată: se va selecta cea mai recentă dată în care raportul dintre numarul valid de pixeli si numărul total de pixeli este mai mare ca 0.8. Dacă nu se gasește o imagine satisfăcătoare in ultimele 30 de zile de la data curentă se vor adăuga 30 de zile la data de start până se gasește o imagine ce indeplinește criteriul.

Funcția postPixelCountService(polygonCoordinatesString, days)apelează serviciul terrascope pentru acuratețea imaginilor. Data selectată este returnată și folosită ca parametru în apelul pentru serviciul de imagini NDVI.

##### **II.1.3.2.Decuparea imaginii**

Următorul pas este decuparea imaginii. Serviciul terrascope returneaza o imagine dreptunghiulară ce corespunde bounding box-ului în care se încadreaza parcela. Pentru a analiza doar parcela este nevoie să decupăm imaginea transformând culoarea pixelilor ce nu fac parte din parcelă în negru. De exemplu:

Figura II.1.3.2.1- Imagine Ndvi a unui Bbox Figura II.1.3.2.2- Imagine Ndvi decupată

##### **II.1.3.3.Maparea punctelor pe imagine**

În primul rând este nevoie de a face legătura între coordonatele punctelor ce determină parcela și indicii pixelilor din imagine ale acestor puncte folosind funcția de la anexa [4].

Funcția returnează pozitiile punctelor în matricea de pixeli ce definește imaginea (o imagine este definită de o matrice de pixeli cu dimensiunile egalu cu dimensiunea imaginiil; fiecare pixel are o valoare în spațiul de culori RGB, adică 3 valori din intervalul [0, 255]). Pentru a afla indicii pixelilor punctelor sunt împărțite distanțele de la punctul din stânga jos al pozei (originPoint) cu valorile returnate de următoarea metodă:

def pixelMapValue(xmax, xorigin, ymax, yorigin, heigth, width):  
 xdif = abs(xmax - xorigin)  
 # print(xdif)  
 ydif = abs(ymax - yorigin)  
 # print(ydif)  
 pixelValueX = xdif / width  
 pixelValueY = ydif / heigth  
 values = [pixelValueX, pixelValueY]  
 return values

Este calculată distanța dintre originea (stânga jos) și punctul din dreapta sus a imaginii pe cele 2 direcții. Apoi valorile sunt impărtite la dimensiuni. Decuparea are loc in metoda cropImage de la anexa [5].

Mai întâi este decupată imaginea pe înălțime și lătime cu ajutorul metodei boundingRect() din biblioteca OpenCv. Metoda primește ca argument un array de puncte si apoi returneaza originea de unde trebuie decupată imaginea, înălțimea și lătimea pană la care trebuie decupată. În general imaginile nu ar trebui sa fie decupate pe înălțime sau lungime pentru că ele sunt deja create pe baza unui boundingbox.

În continuare, este creată o mască. Creăm o matrice de pixeli negri de dimensiunea imaginii, apoi cu ajutorul metodei drawContours() a bibliotecii OpenCv, desenăm o forma albă ce corespunde parcelei. Metoda primește ca parametrii imaginea sursă (în cazul de față masca), punctele formei (punctele identificate anterior ), parametrul care indică ce contururi trebuie desenate (-1 înseamnă ca vor fi desenate toate), culoarea conturului (255,255,255 pentru alb), parametrul ce indică daca trebuie colorat și interiorul conrului (-1 și orice altă valoare negativă indică faptul ca va fi colorat și interiorul) și tipul de linie (cv2.LINE\_AA este o linie care face trecerea între fundal si formă sa fie graduală).

Folosind metoda bitwise\_and aplicăm masca peste imagine. Metoda poate fi folosită și pentru a suprapune 2 imagini, dar pentru că este nevoie doar de funcționalitatea de aplicare a măștii primii 2 parametrii sunt imaginea ințială iar cel de-al treilea este masca. Mai departe este adăugat un fundal alb. În aplicație este folosită poza decupată cu fundal negru, cea cu fundal alb a fost creată pentru a vedea dacă afectează analiza. În figura II.1.3.2.1 se poate observa figura obținută din apelul la serviciul terrascope iar în figura II.1.3.2.2 se poate oberva rezultatul, toți pixelii ce nu aparțin parcelei sunt negrii.

##### **II.1.3.4.Separarea culorilor**

Pasul următor este separarea culorilor. Scopul este de a împărți imaginea pe 3 culori. Cele 3 culori au fost alese în funcție de indicele NDVI: zona verde(vegetație satisfăcătoare), zona galbenă(vegetație uscată) și zona maro (vegetatie foarte uscată, drumuri sau clădiri). Anexa [6] conține metoda care se ocupă de crearea și aplicarea măștilor.

Primul pas este trecerea imaginii în spațiul de culoare HSV. Imaginea care este citită este în spațiul de culoare RGB (red, green, black) dar prin folosirea metodei cv2.imread() imaginea este trecută în spațiul de culoare BGR (black, green, red). Pentru a realiza transformarea, este folosită funcția cv2.cvtColor ce primește ca parametrii imaginea citită cu ajutorul metodei din bibliotecă și tipul de transformare, în cazul de fată din spațiul BGR în spațiul HSV.

##### **II.1.3.5.Spațiul HSV**

HSV este un spațiu de culoare ce are 3 componente:

* Hue, specifică unghiul culorii pe cercul de culori RGB. Poate lua valori între 0 și 179 în OpenCV.
* Saturation, controlează cantitatea de culoare. O valoare ce se apropie de 100% va aduce cea mai pură varianta a culorii, în timp ce o valoare apropiată de 0% va aduce o culoare gri. Poate lua valori între 0 și 255 în OpenCV.
* Value controlează luminozitatea culorii. O culoare cu value aproape de 0% este de fapt negru pur în timp ce o culoare cu value aproape de 100% nu va conține negru. Poate lua valori între 0 și 255 în OpenCV.

În urma rulării algoritmului pentru mai multe valori am ajuns la următoarele intervale HSV pentru cele 3 culori: Verde (41, 25, 25)-(70, 255, 255); Galben (23, 25, 25)-(40, 255, 255); Maro: (13, 25, 25)-(22, 255, 255). Funcția cv2.inRange primește 3 parametrii: imaginea și cele 2 limite între care trebuie sa se situeze culoarea în spațiul HSV. Output-ul este este o mască binară în care pixelii care se încadrează în interval sunt albi iar pixelii care nu se încadrează sunt negri. În figurile Figura II.1.3.5.1., Figura II.1.3.5.1. și Figura II.1.3.5.1. se pot vedea măstile pentru cele 3 culori.

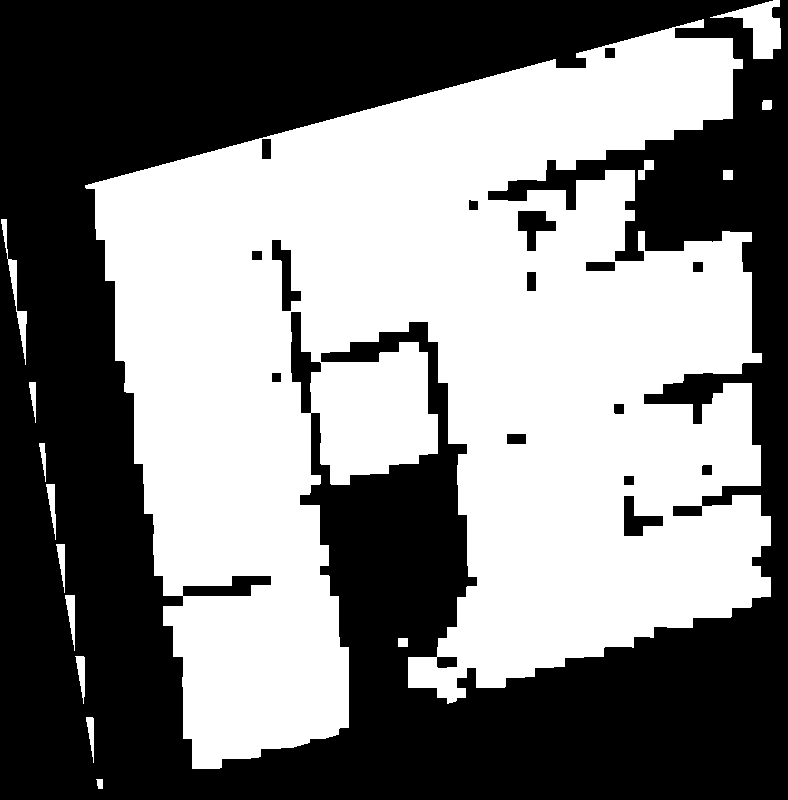
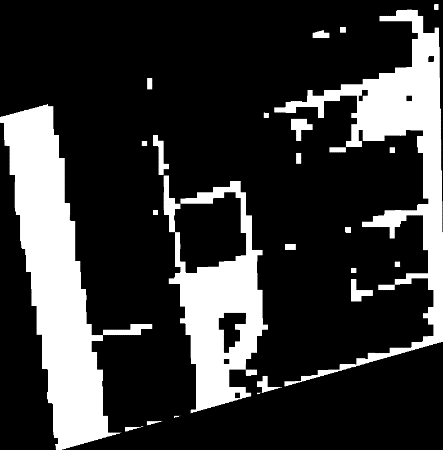
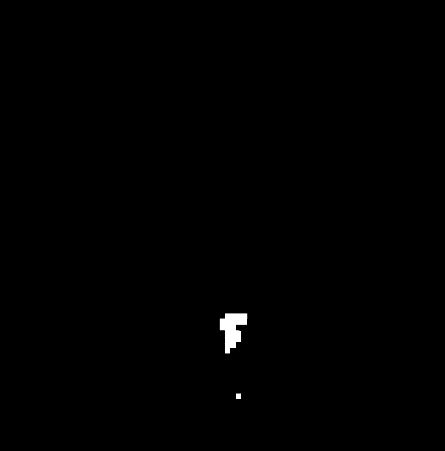
  

Figura II.1.3.5.1. Masca verde Figura II.1.3.5.2. Masca verde Figura II.1.3.5.3. Masca maro

După crearea măștilor, este creată matricea de pixeli imax ce are ca valori True si False. True pentru pixelii albi și False pentru pixelii negri. Apoi este creată o imagine nouă, color, de dimensiunile imaginii inițiale care inițializata cu toți pixelii negri. Apoi folosind matricea imask, pixelii din matricea imask care au valoarea True (sunt albi deci se află în intervalul culorii) copiază valoarea din imaginea inițială (ce corespunde imaginii din input) în imaginile cu cele 3 culori. Rezultatele aplicării măștilor se pot vedea în figurile Figura II.1.3.5.4., Figura II.1.3.5.4. și Figura II.1.3.5.4.

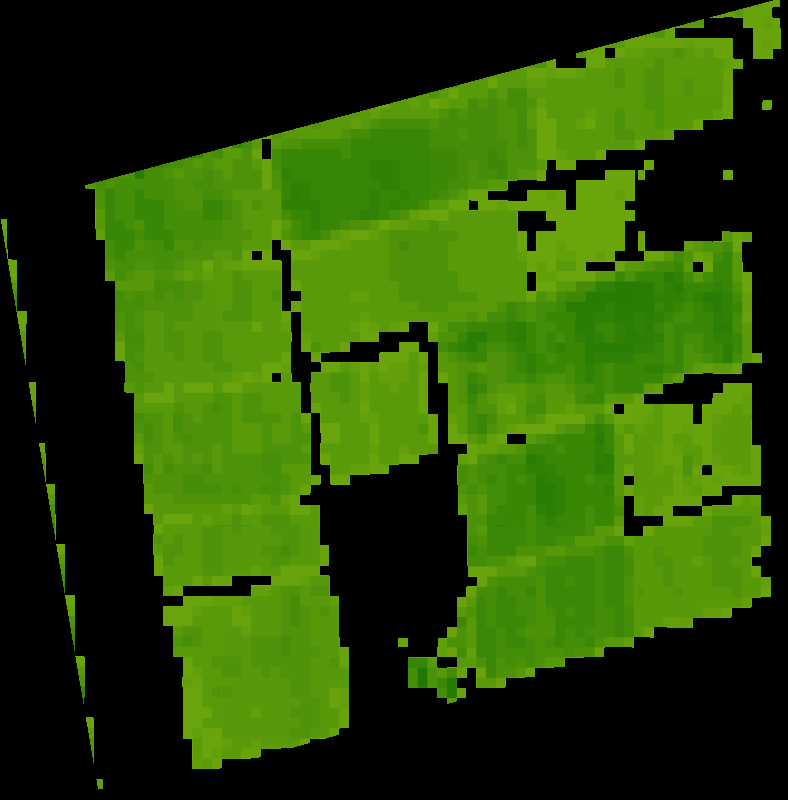
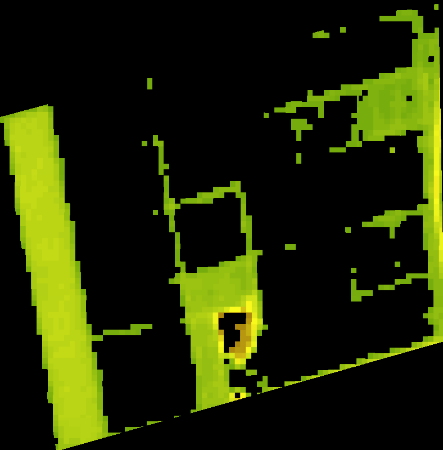
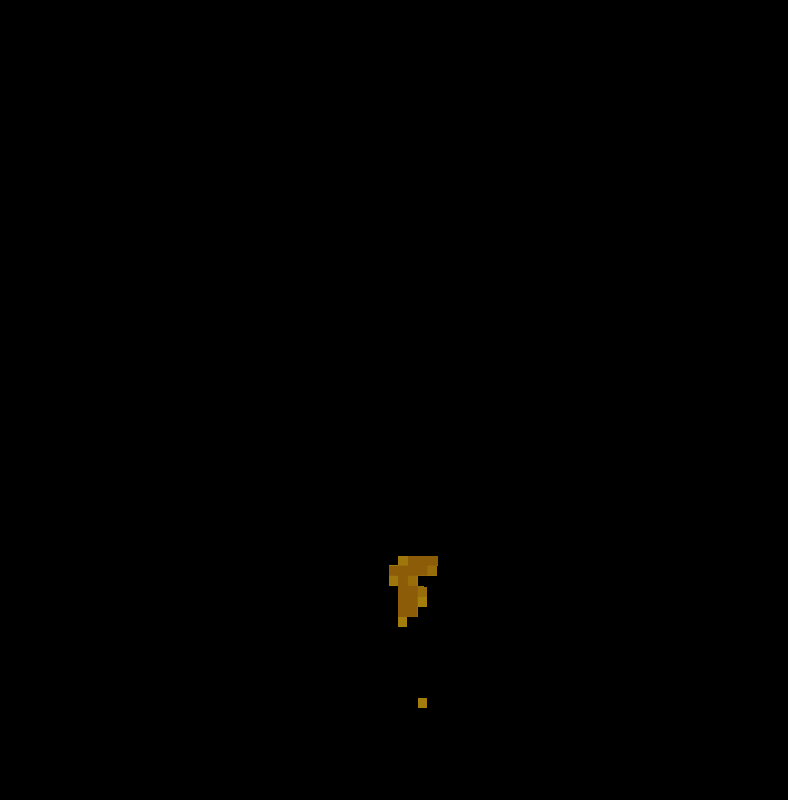
  

Figura II.1.3.5.4.Masca verde Figura II.1.3.5.5.Masca galbenă Figura II.1.3.5.6.Masca maro

##### **II.1.3.6.Determinarea poligoanelor**

Procesul de determinare al poligoanelor are loc pentru fiecare dintre cele 3 cateogrii NDVI sau pentru cele 3 culori verde galben și maro.

Pentru a produce un output cu date care să ușureze analizarea terenului nu este de ajuns doar impărtirea imaginii pe categorii de culoare. Aceste zone trebuie localizate prin coordonate reale. În exemplul de mai sus se poate oberva că zonele pot fi reprezentate de mai multe suprafețe separate de pămănt, deci este nevoie de identificarea acestor poligoane și a coordonatelor.

##### **II.1.3.7.Identificarea contururilor**

def findContours(image):  
 gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)  
 edged = cv2.Canny(gray,100,600)  
 contours, \_ = cv2.findContours(edged,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_NONE)  
 return contours

Primul pas este identificarea contururilor. Primul pas este transformarea imaginii în grayscale folosind funcția cv2.cvtColor(), cu parametrul COLOR\_BGR2GRAY ce specifică transformarea din spațiul de culoare standard din OpenCv BGR în Grayscale. Apoi este aplicată metoda de detectare a marginilor Canny. Marginea este o discontinuitate în intensitatea pixelilor. Funcția din biblioteca OpenCv primește ca input imaginea grayscale și 2 argumente: limita de jos (pixelii cu intensitatea mai mică nu pot fi margini) și limita de sus (pixelii cu intensitatea mai mare sunt considerati automat margini). În imaginile din Figura II.1.3.7.1, Figura II.1.3.7.2 și Figura II.1.3.7.3 sunt desenate contururile pentru cele 3 culori pe un fundal negru.

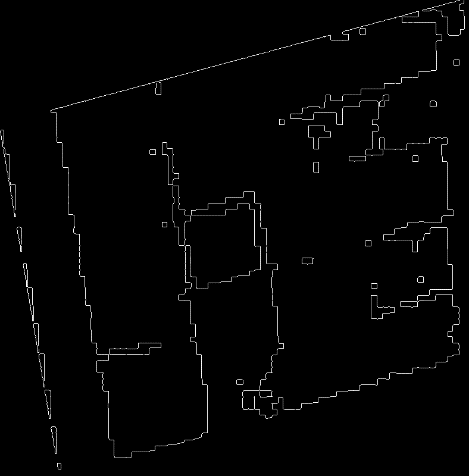
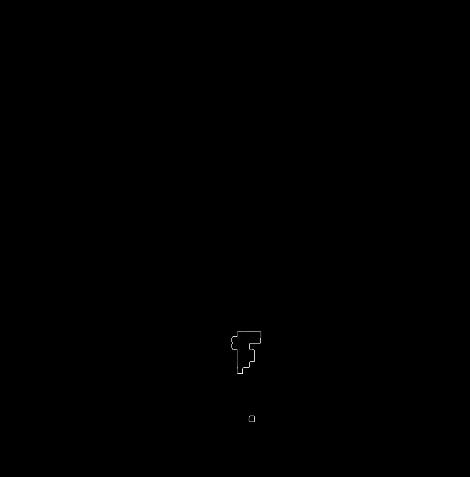
  

Figura II.1.3.7.1. Contur verde Figura II.1.3.7.2. Contur galben Figura II.1.3.7.3. Contur maro

În imaginile din output în alb-negru, pixelii pot avea doar 2 valori alb sau negru deci imaginile se pot fi caracterizate ca fiind imagini binare (imaginile binare sunt imagini în care pixelii pot avea doar 2 valori). Metoda cv.finContours() funcționează cel mai bine pentru imagini binare. Metoda de detectare este RERT\_EXTERNAL care returneaza doar contururile exterioare deoarece contururile interioare reprezintă poligoane de altă culoare sau poligoane care se află în acceași categorie dar diferă intensitatea culorii. Parametrul CHAIN\_APPROX\_NONE, reprezintă metoda de aproximare folosită. În cazul de față, pentru un output mai detaliat nu este folosită nicio metodă de aproximare. Totuși dacă output-ul ar fi testat pe teren de utilizatori există posibilitatea de a schimba parametrul în CHAIN\_APPROX\_SIMPLE pentru un output mai puțin detaliat dar mai ușor de analizat. Output-ul este o listă de contururi, un contur la randul său conține o listă de puncte. Contururile desenate peste imagini pot fi vizualizate în figurile II.1.3.7.4, II.1.3.7.5 și II.1.3.7.6.

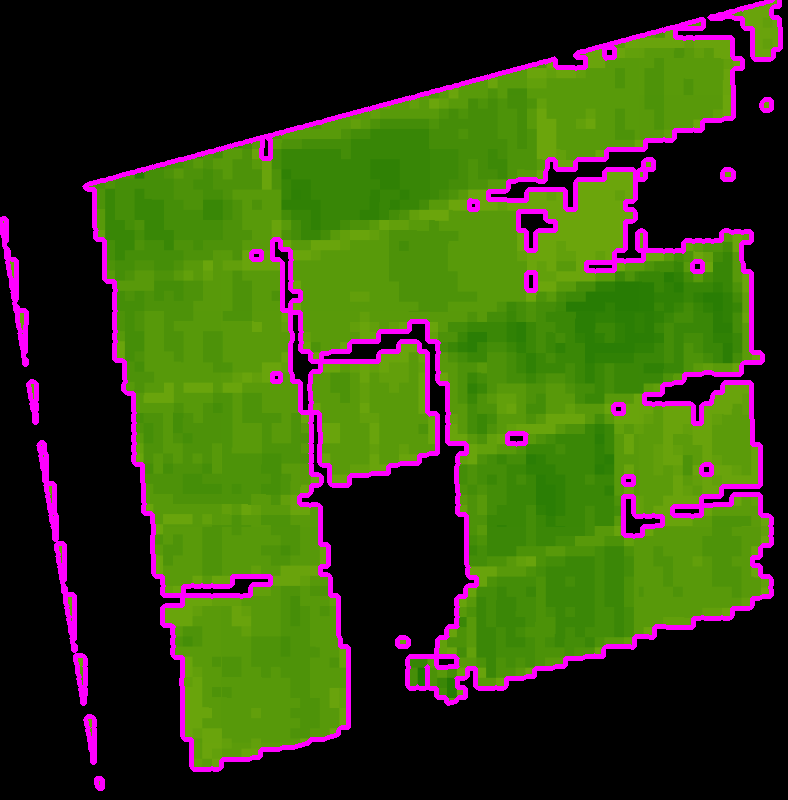
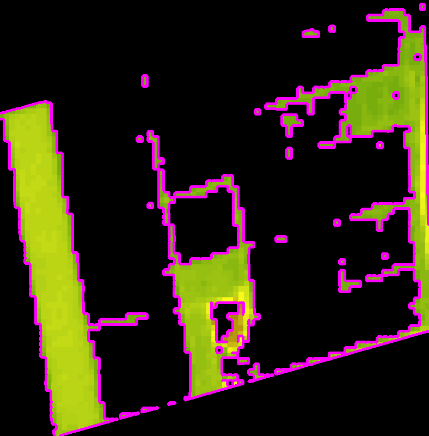
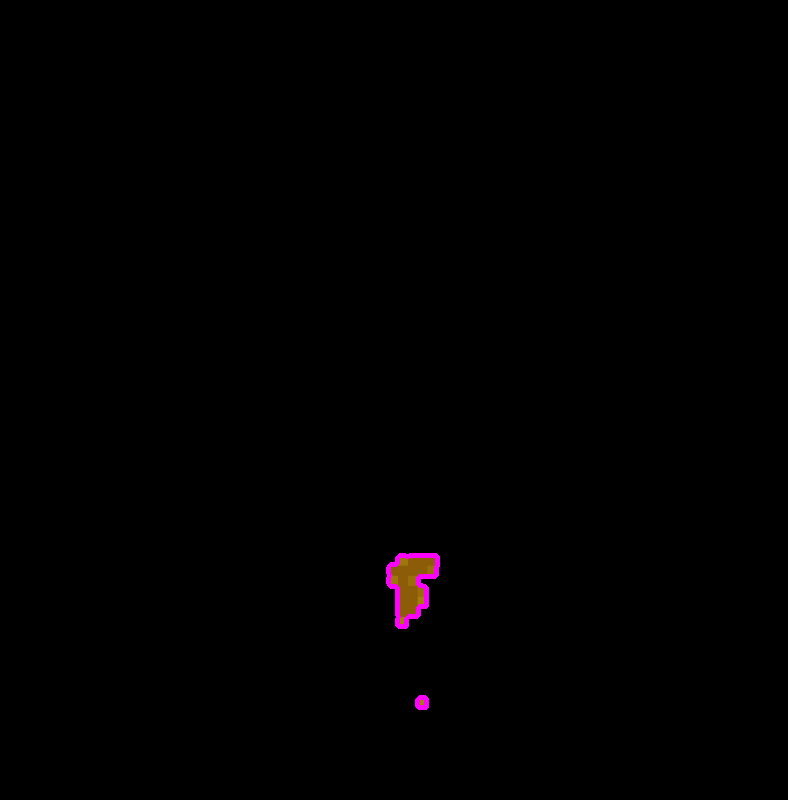
  ** **

Figura II.1.3.7.4. Contur verde 2 Figura II.1.3.7.5. Contur galben 2 Figura II.1.3.7.6. Contur maro 2

##### **II.1.3.8.Separarea în poligoane**

După cum se poate observa în imagini contururile sunt reprezentate de foarte multe puncte. Output-ul final trebuie sa contină doar punctele cele mai importante nu întreg contur. Pentru a determina colțurile imaginilor este folosită metoda cv2.CornerHarris. Colțurile unei imagini sunt regiunile în care sunt variații mari în gradientul imaginii în toate dimensiunile si direcțiile posibile. Gradientul este o schimbare direcțională în intensitatea sau culoarea unei imagini. Colțurile poligoanelor sunt extrase în metoda din anexa [7]. Metoda primește la intrare imaginea care va fi analizată.

Funcția nu returnează punctele găsite ci o matrice 2d de aceeași dimensiune ca imaginea din input cu probabilități. Fiecare poziție din matrice conține o valoare numerică ce reprezintă probabilitatea ca pixelul de pe acea pozitie sa fie un colț. Este nevoie de un threshold după care vor fi selectate punctele. În cazul de față valoarea este 0.1 din valoarea maximă a matricei din output-ul metodei Corner Harris. Fiecare punct ce trece de treshold este considerat colț și este transformat în culoare roșie în imaginea inițială din spațiul de culoare BGR, adică primește valoarea [0, 0, 255]. Apoi, pe baza acestei imagini este creată o listă de perechi care conține indicii colțurilor. Folosind doar Corner Harris este posibil ca serviciul să returneze colțurile trecănd peste detectarea contururilor. Dar pentru a repartiza colțurile pe poligoane este nevoie de ambele rezultate.

În metoda extractPolygons ce primește ca input contururile și colțurile imaginii sunt create poligoanele pentru output. Contururile conțin toate punctele ce separa poligoanele (inclusiv colțurile) grupate. Acum datorită faptului că colțurile sunt cunoscute pot fi create poligoane. Un poligon este construit pe baza unui contur din care au fost extrase doar punctele care sunt colțuri.

def extractPolygons(contours, corners):  
 polygons = []  
 for contour in contours:  
 poly = []  
 for coords in corners:  
 if coords in contour:  
 poly.append(coords)  
 polygons.append(poly)  
 return polygons

Pe baza acestor poligoane este creată o listă de obiecte de tip Poligon. Poligonul conține o listă de puncte și un cod ce corespunde unei culori.

for poly in polygonCoords:  
 if len(poly) < 3:  
 continue  
 coords = pixelsIndicesToCoordinates(poly, height, width, coordinatesBBOX)  
 p = Polygon(coords, colorCode[color.lower()])  
 polygonList.append(p)

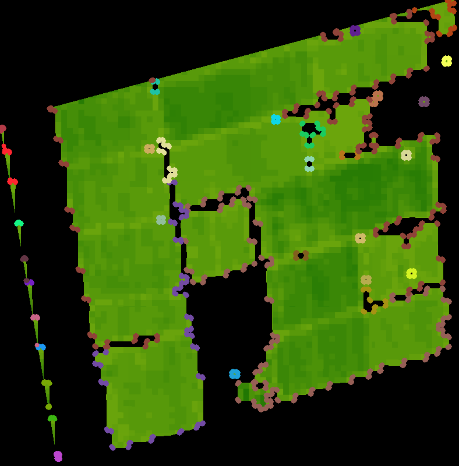
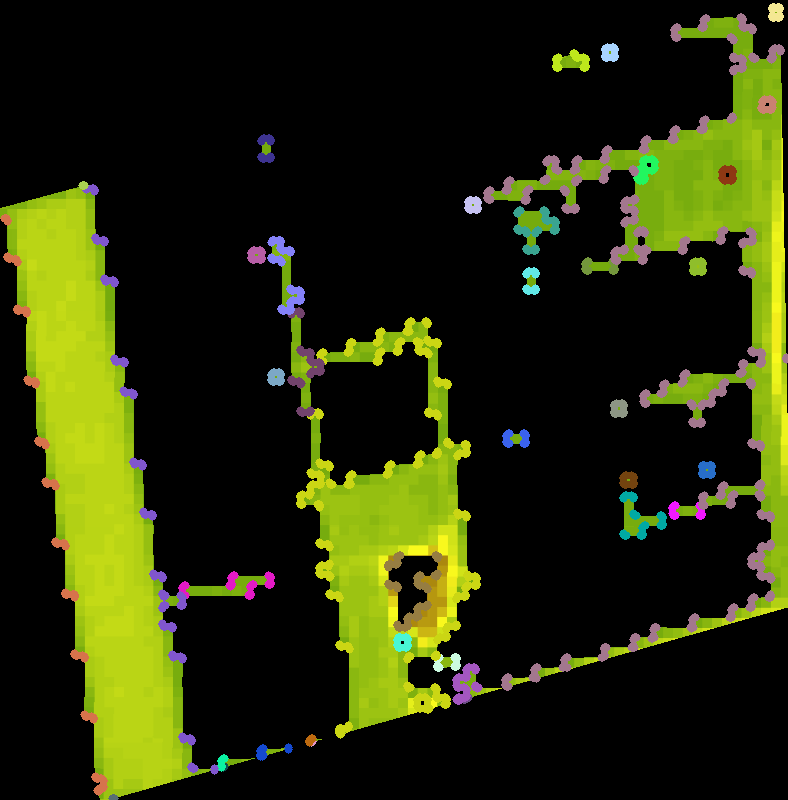
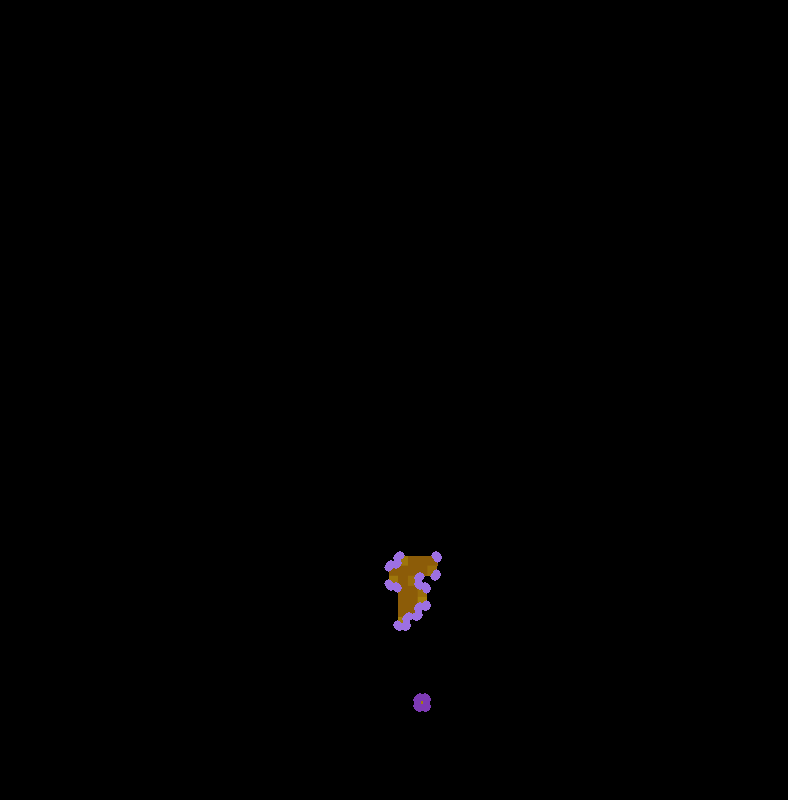
Un poligon trebuie sa conțină cel puțin 3 puncte pentru a fi considerat la output. Apoi folosind metoda pixelsIndicesToCoordinates punctele exprimate în indici sunt transformate în puncte de coordonate, este creat poligonul și adăgat la lista finală.

Din cauza modului în care sunt determinate contururile, în unele cazuri poligoanele nu sunt determinate optim. Un contur se termină la marginea imaginii sau cănd se intersectează în trasarea sa cu un punct ce îi apartine. Sunt cazuri în care punctele poligoanelor nu determină o zonele corect din cauza unor margini ale imaginii sau din cauza ateției foarte mari la detalii a metodei care găseste contururile și din cauza marginilor. O soluție are fi extinderea celor 4 laturi ale imaginilor cu pixeli negri pentru a elimna întreruperile nedorite provocate de margini. În urma analizări mai multor seturi de date am obervat că rezultatul ar putea fi imbunătățit prin eliminarea unor pixeli ce unesc mase mai mari de culoare. La întalnirea unor astfel de puncte desenarea conturului se întoace, întâlnește un punct ce îi aparține si se termină, ceea ce duce la crearea unor poligoane care nu sunt închise. Pierderea de acuratețe ar fi foarte mică și ar putea îmbunătăți calitatea output-ului. Pixelii ar fi ‘eliminați’ prin colorarea lor în culoarea negru. Exemple in figurile

II.1.3.8.1. Pixel care ar trebui eliminat II.1.3.8.2. Pixel care ar trebui eliminat

În imaginile urmatoare, ce au foarte multe variații de culoare, pot fi observate și cazuri extreme ce duc la crearea unor poligoane ce ramân deschise. Scopul este ca majoritatea poligoanelor să fie închise. Chiar și în cazuri extreme, output-ul este relevant pentru parcelă intrucât punctele tot reprezintă locuri în care este o variație a vegetației. Fiecare poligon are punctele colorate cu o culoare aleatoare pentru a putea face distincția între poligoane.

 **** 

II.1.3.8.3. Puncte verzi II.1.3.8.4. Puncte galbene II.1.3.8.5. Puncte maro

##### **II.1.3.9.Output**

Pasul final este generarea unui json folosind poligoanele obținute și statisticile. Json-ul conține un atribut ‘map’ cu datele referitoare la puncte și un atribut ‘statistics’.Atributul ‘crs’ conține date referitoare la sistemul de coordoante în care sunt reprezentate punctele. Anexa [8] conține o parte din informațiile geografice ale unui json-ului obținut pentru exemplele anterioare.

**Generarea statisticilor**

În plus față de punctele grupate în poligoane serviciul, json-ul returnat de serviciul de clasificare conține și o secțiune de statistici. În această sectiune este notat, pentru toate cele 3 culori și pentru pixelii acoperiți de nori, raportul dintre numarul de pixeli de culoare și numărul total de pixeli precum și suprafața totală exprimată în metrii pătrați. Fiecare categorie are asociat un cod și un hex de culoare :

* Codul -1 și hex-ul #FFFFFF pentru pixelii albi/acoperiți de nori
* Codul 0 și hex-ul #8A7010 pentru pixelii maro
* Codul 1 și hex-ul #8A6210 pentru pixelii galbeni
* Codul 2 și hex-ul #768A10 pentru pixelii verzi

La anexa [9] se pot observa statisticile obținute.

### **II.2. Baza de date**

#### **II.2.1.Despre Big-data**

Big-data reprezintă seturile de informații mari ce cresc exponențial. Din păcate, seturile de date big-data sunt atât de mari încât metodele de stocare tradiționale nu le pot stoca și procesa eficient. Mai mult decât volumul mare, este important și modul în care sunt utilizate datele. Datele pot fi analizate pentru a obține informații care să ajute deciziile de business pe care le iau companiile și utilizatorii. Oamenii produc aproximativ 2 chintilioane de date în fiecare zi.

Prima apariție a unui set de date de tip big-data a avut loc în 1663 când John Graunt lucra cu o cantitate copleșitoare de informații despre ciuma bubonică care se răspândea în Europa în acea perioadă. Graunt a fost prima persoană care a efectuat statistici bazate pe analiza de date.

Prima dată când lumea s-a lovit de o problema procesării big-data a fost în 1880 când biroul de recensământ din Statele Unite, a estimat ca r dura 8 ani să proceseze și să analizeze datele colectate pentru recensământ în acel an. În 1881 un angajat al biroului de recensământ pe nume Herman Hollerith a inventat mașina de tabelare Hollerith care a redus timpul de procesare.

Pe parcursul secolului XX, datele au evoluat cu o viteză neașteptată. Big-data au ajuns la baza evoluției civilației umane. În această perioadă au fost inventate mașini de stocare a datelor și de scanare a unor șabloane regăsite în mesaje. Unul dintre primele centre de date din lume a fost construit în 1965 de către guvernul Statelor Unite milioane de seturi de amprente și de documente de restituire a taxelor.

Seturile de date pot fi împărțite în 3 categorii: date structurate, date semi-structurate și date nestructurate.

#### **II.2.2.Introducere**

Din cauza timpul de răspuns foarte mare al componentei de clasificare este nevoie de stocarea output-urilor. Componenta va fi apelată pentru analiza inițială a unei parcele sau dacă se dorește efectuarea unei noi analize. Altfel, rezultatele vor fi obținute din baza de date. Cantitatea mare de date posibilie date de numărul de parcele ce pot fi analizate, nevoia de a genera analize și statistici și faptul că datele sunt structurate îcadrează setul de date ca fiind un set de date big-data structurat. Baza de date este distribuită de tip NoSql și este construită cu ajutorul framework-ului Apache Hadoop.

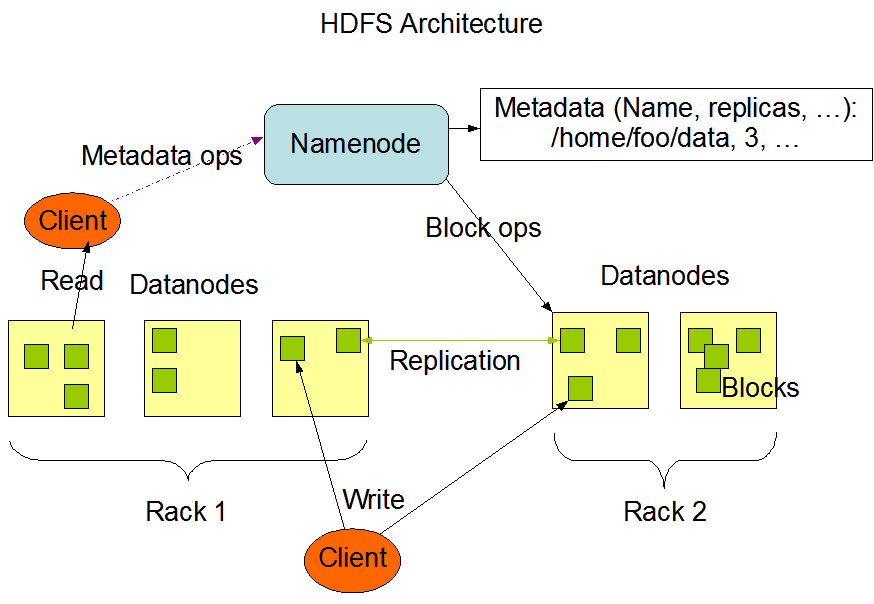
#### **II.2.3.Hadoop**

Apache Hadoop (High Availability Distributed Object Oriented Programming) este o plaformă software open-source construită cu Java ce se ocupă cu managementul, stocarea și procesarea volumelor mari de date. Hadoop are la bază un cluster de computere. Un cluster este un set de computere ce lucrează impreună ca un singur sistem. Performanța computerelor poate varia de la mașini extrem de performante până la computere normale sau chiar puțin performante. Performanța computerelor influențează direct puterea sistemului. Cluster-ul poate fi alcătuit din orice număr de computere și poate fi extins în orice moment. Componentele de bază Hadoop sunt Hdfs (Hadoop distributed file system) și modelul de programare Map Reduce.

#### **II.2.4.Hdfs**

Hdfs este un sistem de date distribuit conceput sa ruleze pe hardware standard. Hdfs, spre deosebire de majoritatea sistemelor de date, este foarte tolerant la erori și poate fi folosit pe hardware low-cost. Un sistem de date HDFS poate fi format din sute sau mii de computere fiecare stocând o parte din date. Hdfs imparte datele în blocuri și le stochează pe computerele clusterului Fiecare componenentă are posibilitatea de a eșua așa că Hdfs are un mod foarte eficient de a detecta problemele de hardware si de a le rezolva automat.

Hdfs are o arhitectură de tip master/slave. Nodul master este unic și se numește NameNode, nodurile slave se numes DataNodes. În general fiecare computer din cluster conține un nod de date dar este posibil să conțină și mai multe. Pot fi adăugate noduri de date în orice moment iar Hdfs va realoca datele pentru cât mai optim. Name node este cel care lansează toate operațiunile de lucru cu datele: citire, scriere, duplicare, deschidere, închidere etc. Orice proces solicitat de client se face prin intermediul namenode-ului. Mai departe, name node-ul împarte task-ul în sub task-uri care apoi sunt trimise către data node-uri pentru a fi rulate. Singura excepție este stocarea datelor unde clientul poate lăsa name node să distribuie datele sau le poate scrie direct în data node-uri. Din punctul de vedere al procesării, totul are loc la nivelul data node-urilor. Din puntul de vedere al stocării datelor, datele propriu-zise sunt împărțite în blocuri și stocate în data node-uri, în timp ce name node-ul conține meta-date. În figura II.2.4.1 din documentație oficială Apache Hadoop se poate vedea o reprezentare vizuală a modului în care funcționează framework-ul.

****

II.2.4.1. Schemă grafică hadoop Sursă imagine https://hadoop.apache.org/

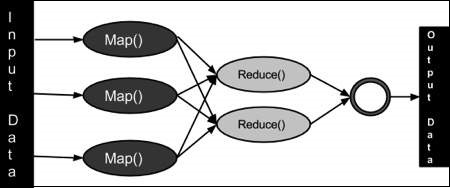
#### **II.2.5.MapReduce**

Map reduce este o tehnică de procesare pentru computarea distribuită. Algoritmul conține 2 părți map și reduce. Map primește un set de date si îl transformă în alt set de date în care fiecare element este o pereche cheie-valoare. Reduce primește la intrare output-ul funcției de map și combină perechile într-un set mai mic de perechi cheie-valoare. Cel mai mare avanaj este că procesarea datelor poate fi scalată pe mai multe noduri. Odată ce o aplicație map reduce este scrisă rularea sa pe sute, mii sau zeci de mii de computere este doar o schimbare de configurare.

MapReduce se execută in 3 pași: map, shuffle și reduce

* Map – Scopul este de a procesa datele de intrare. În general datele de intrare sunt de forma unui fișier sau director stocate in HDFS. Fisierul de intrare este transmis către mapper linie cu linie. Mapper-ul procesează datele și crează bucați mai mici de date.
* Reduce – combină pașii de shuffle și reduce. Procesează datele rezultate din mapper. Operațiile și logica aplicatiei de map reduce este scrisă la cest nivel. Noul output este stocat în HDFS.

Procesul de map reduce este recepționat la name node care trimite procesul mai departe către data nodes care vor rula procesul în paralel pe datele lor. Output-ul final este reuniunea rezultatelor de pe nodurile de date.



II.2.5.1. Schemă map-reduce, sursă https://www.tutorialspoint.com/hadoop

#### **II.2.6.Hbase**

Hbase este sistem de management al bazelor de date non relațional orientat pe coloane ce rulează peste HDFS. Hbase este o metodă toelrantă la erori de avea acces de scriere și citire aleatoare la volume mari de date. Spre deosebire de sistemele realționale de baze de date, Hbase nu are un limbaj de interogare precum SQL deoarece Hbase este scris în Java. Pentru a adăuga posibilitatea de a face interogări asupra este nevoie de a folosi Hive împreună cu Hbase.

Un sistem Hbase scalează liniar. Conține un set de tabele cu rânduri și coloane la fel ca o bază de date tradițională. Fiecare tabelă trebuie să aibă o cheie primară. Toate operațiile de acces la tabele Hbase trebuie să folosească această cheie primară.

#### **II.2.7.Ambari**

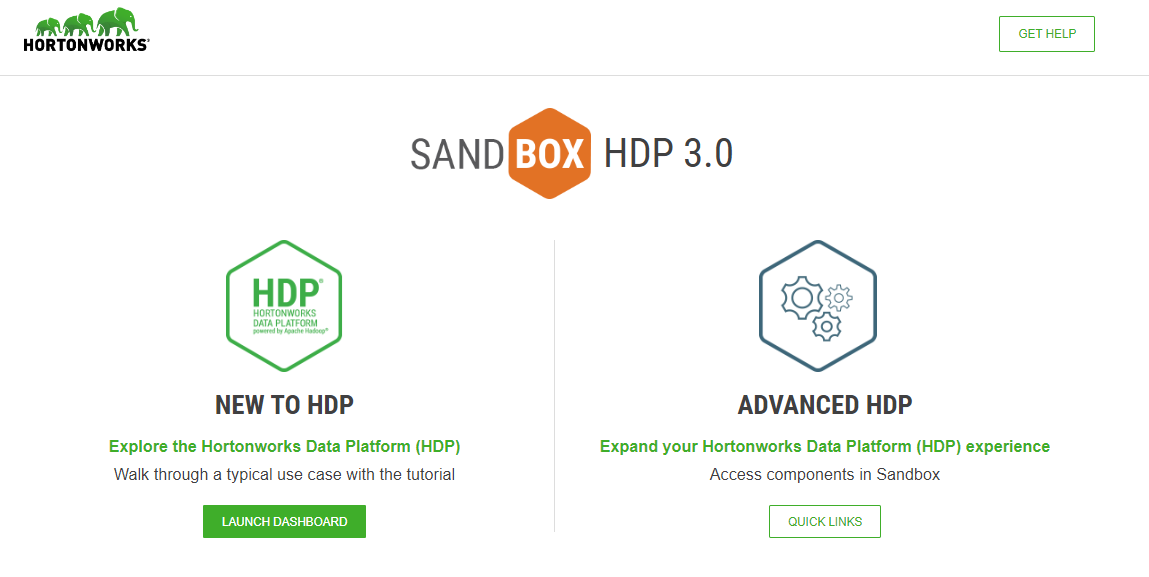
Ambari este un proiect ce are scopul de a simplifica managementul Hadoop cu ajutorul unui software ce menține si monitorizează clusterele hadoop. Ambari pune la dispoziție o interfață web și API-uri RESTful.

#### **II.2.8.Cloud Era Hortonworks Data Platform**

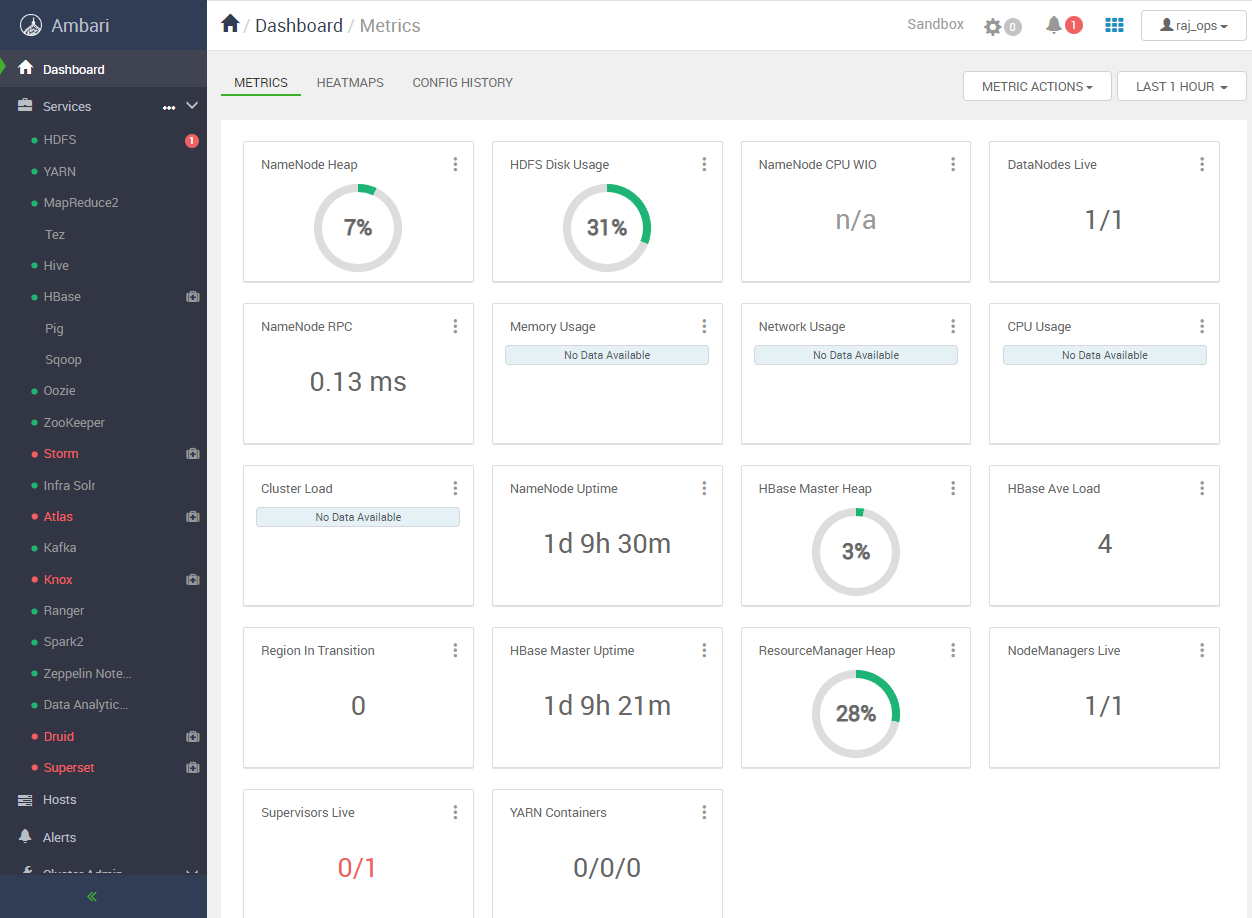
Cloud era este o companie de software ce pune la dispoziție sisteme de management al datelor ce folosesc Apche Hadoop. Unul dintre produse este Hortonworks Data Platform sau pe scurt, HDP. HDP este un framework open source pentru stocarea și procesarea seturilor de date Big Data. Aplicația folosește o instanță locală de HDP configurată pe o mașină virtuală cu un sistem de operare Linux. Varianta HDP folosită poate fi instalată direct folosind printre altele Oracle Virtua Box. Instanța HDP instalată are toate utilitățile necesare deja configurate (hadoop, hdfs, java, python, un sistem de operare Linux) dar și foarte multe framework-uri adiționale. Cele folostite în aplicație sunt: Hbase, Ambari, HadoopRest și Hive.

Accesul la platformă se poate face manual cu ajutorul interfeței Ambari, cu API-ul RESTful sau folosind api-ul Java pus la dispoziție de Hadoop.

Pentru a face referire mai ușor la mașina virtuală trebuie modificat fișierul ‘hosts’ la locația: C:\Windows\System32\Drivers\etc\hosts. Mașina virtuală folosește ip-ul 127.0.0.1 modificând fișierul se poate face referire la adresa mașinii virtuale folosind 2 nume: sandbox-hdp.hortonworks.com sau sandbox-hdf.hortonworks.com. Figura II.2.8.1 este o captură de ecran cu pagina principală a mașinii virtuale. Interfața Ambari se poate vedea în figura II.2.8.2.



II.2.8.1 Interfață HDP



II.2.8.2. Interfață Ambari

#### **II.2.9.Stocarea datelor**

Datele sunt trimise către Hdfs de pe mașina vituală prin intermediul componentei de back-end. Mașina vituală este deja setată cu mai mulți utilizatori printre care și utilizatorul ‘raj\_ops’ ce are permisiuni de management al clusterului dar și de prelucrare a datelor. Toate operațiile cu date sunt efectuate sub acest username, pentru a schimba user-ul este nevoie de o sipla schimbare a parametrilor din apelul metodelor expuse de back-end si de crearea userului în Hdfs în cazul în care nu există.

Datele neprelucrate returnate de servciul de clasificare, primite de la componenta de back-end sunt stocate în zona alocată userului menționat mai sus în directorul ‘parcels/raw’ sub forma de fișiere de tip text. Pentru a organiza datele este folosită o regulă pentru numele fișierelor. Numele este format din id-ul parcelei urmat de data în care a fost efectuat procesul de clasificare. Este important ca utilizatorii să atribuie în mod unic și să rețină numele parcelelor pe care le analizează. Datorită faptului ca o baza de date distribuită este concepută pentru a reține și analiza volume mari de date, pentru o parcelă anume vor exista mai multe fișiere din date diferite. Astfel poate fi urmarită nu doar starea actuală a vegetației dar și evoluția.

### **II.3. Back end**

#### **II.3.1.Introducere**

Pentru a asigura comunicarea între restul componentelor este implementată și o componentă de back-end. Datorită faptului că există o biblioteci pentru Hadoop scrise în Java si mai ales pentru că Hadoop este scris în Java, am ales să folosesc Java Spring. De asemenea, implementănd această compoentă este asigurată decuplarea dintre componente. Totate celelalte componente pot rula independent și pot fi integrate în alte proiecte.

#### **II.3.2.Spring**

Spring este un framework flexibil conceput pentru a construi aplicații web în Java. Spring se ocupă de majoritatea aspectelor low-level ce țin de construirea unei aplicații ceea ce permite utilizatorilor să se concentreze pe dezvoltarea logicii de business.

#### **II.3.3.Descrierea componentei**

Compoenta are o structură standard, este orgazită în controllere, servicii, modele de date și clase utilitare. Serviciul pentru Hdfs implementează metode ce asigură comunicarea cu baza de date în timp ce serviciul Ndvi implementeză metode care asigură comunicarea cu componenta de clasificare Ndvi. Metodele sunt expuse in controllere pentru a putea fi apelate printr-un Api Restful.

#### **II.3.4.Servicii**

Serviciul NDVI realizează legătura cu serviciul de clasificare. Singurul scop este de a face un apel simplu către componenta de clasificare pentru a obține datele, cu scopul de a fi afișate sau introduse în baza de date.

Serviciul Hdfs realizează legatura cu baza de date de pe mașina virtuală folosind biblioteca hadoop-client sau serviciul webHdfs ce permit efectuarea de operații specifice Hdfs, precum adaugarea fișierelor, crearea directoarelor, descărcarea fișierelor în memoria locală, ștergerea fișierelor sau a directoarelor.

#### **II.3.5.Funcționalitate**

Componenta pune la dispoziție mai multe metode ce pot fi apelate. Poate fi apelată compoenenta de clasificare pentru a obține un json. Json-ul poate fi folosit pentru a vizualiza datele sau poate fi stocat la o cale locală definită în proprietățile aplicației în interiorul unui fișier de tip text. Fișierul

# **Capitolul III**

## **III.1. Concluzii**

Componentele pot fi parte dintr-o aplicație care să aducă un mod nou de a ajuta activitățile din domeniul agriculturii. În momentul de fată cu ajutorul serviciului Ndvi există posibilitatea de a genera analiza oricărei porțiuni de pamânt din aria de acoperire a satelitului pus la dispoziție de Terrascope. De asemenea, componenta de back-end asigură comunicarea cu o baza de date distribuită hadoop care poate fi configurată din proprietăți. În lucrare, mașina virtuală folosită deși foarte convenabilă nu are performanțe care se apropie de cele ale unui cluster real. În testări s-a dovedit că există anumite limitări și erori cauzate de lipsa de resurse alocate. Totuși în ciuda limitărilor și erorilor, tot ceea ce este implementat pentru baza de date funcționează. Deci, având în vedere că singurele limitări au fost performanțele nu implementarea, o bază de date distribuită cu capabilități rezonabile ar trebui să poată rula comenzile primite de la componenta de back-end fără erori.

## **III.2. Contribuții personale**

Contribuțiile personale constau în:

* Utilizarea cât mai eficientă a algoritmilor de procesare a imaginilor pentru a obține date relevante pentru agricultură din imagini satelitare.
* Implementarea unui Api restful pentru a lega o bază de date distribuită de serviciul de clasificare al imaginilor precum și pentru a face operații asupra bazei de date.

## **III.3. Direcții de viitor**

Pentru a îmbunătăți output-ul serviciului de clasificare ar trebui implementată o metodă care să elimine pixelii ce cauzează crearea unor poligoane prea detaliate cu forme anormale. Ar putea exista mai multe opțiuni de clasificare în funcție de nivelul de detalii pe care îl solictă utilizatorul.

Aplicația trebuie testată în viitor si pe o bază de date distribuită cu performanțe reale pentru a asigura faptul că probleme care au apărut sunt cauzate exclusiv de limitările de performanță.

Pentru ajuta utilizatorii finali, să poată interpreta mai bine analiza parcelelor poate fi implementată o interfată, fie web fie desktop care să expună datele intr-un mod ușor de înteles. De asemenea, Api-ul Restful pus la dispoziție de componenta de back-end ar trebui sa poată fi folosit prin apăsarea unor butoane mult mai ușor de folosit decât apeluri curl sau folosind Postman. Punctele obținute din clasificare ar putea fi mapate pe o hartă reală interactivă.

**Bibliografie**

[1] Bert Bates, Kathy Sierra – Head First Java

[2] Bert Bates, Kathy Sierra, Elisabeth Robson – Head First Desing Patterns

[3] Craig Wallis – Spring in action

[4] Adilin Beatrice - Totul despre bazele big data: istorie, tipuri și aplicații

[5] Documentația oficială apache hadoop: <https://hadoop.apache.org/>

[6] GISGeography – Ce este Ndvi? <https://gisgeography.com/ndvi-normalized-difference-vegetation-index/>

[7] custom scripts – Informatii despre Ndvi și Sentinel-2 <https://custom-scripts.sentinel-hub.com/sentinel-2/ndvi/>

[8] Documentația oficială OpenCv - <https://opencv.org/>

[9] IBM – HDFS <https://www.ibm.com/topics/hdfs>

[10] Documentația oficială Terracope.be <https://docs.terrascope.be/#/README>

[11] Harta Ndvi Terrascope.be <https://viewer.terrascope.be/>

[12] Databriccks – Informații despre Hadoop https://databricks.com/glossary/hadoop

[13] Documentația oficială Apache Hbase https://hbase.apache.org/

**Anexe**

[1] Funcție de determinare a boungin box-ului

def getBBOXFromParcelCoordinates(coordinatesList):  
 coordinatesBBOX = []  
 xList = coordinatesList[0::2]  
 yList = coordinatesList[1::2]  
 coordinatesBBOXMinX = min(xList)  
 coordinatesBBOXMinY = min(yList)  
 coordinatesBBOXMaxX = max(xList)  
 coordinatesBBOXMaxY = max(yList)  
 coordinatesBBOX.append(coordinatesBBOXMinX)  
 coordinatesBBOX.append(coordinatesBBOXMinY)  
 coordinatesBBOX.append(coordinatesBBOXMaxX)  
 coordinatesBBOX.append(coordinatesBBOXMaxY)  
 coordinatesBBOX = epsg4326ToEpsg3857(coordinatesBBOX)  
 return coordinatesBBOX

[2] Funcție care calculează distața dintre 2 puncte geografice

def getDistanceFromLatLonInM(lat1, lon1, lat2, lon2):  
 R = 6371  
 dLat = deg2rad(lat2 - lat1)  
 dLon = deg2rad(lon2 - lon1)  
 a = math.sin(dLat / 2) \* math.sin(dLat / 2) + math.cos(deg2rad(lat1)) \  
 \* math.cos(deg2rad(lat2)) \* math.sin(dLon / 2) \* math.sin(dLon / 2)  
 c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))  
 d = R \* c  
 float(d)  
 d \*= 1000  
 return d

[3] Parte din json-ul returnat de serviciul Terrascope pentru calitatea imaginilor

"results": [

{

"result": {

"totalCount": 10214,

"validCount": 8715,

"average": 0.5399646586345382

},

"date": "2022-04-30"

},

{

"result": {

"totalCount": 10214,

"validCount": 10214,

"average": 0.6117971411787744

},

"date": "2022-05-05"

},.........

def mapPolygonPointsOnImage(bbox, polygonCoordinates, heigth, width):  
 bbox = verifyOrderOfBboxCoordinates(bbox)  
 bbox = epsg3857ToEpsg4326(bbox)  
 bboxPoint1 = [bbox[0], bbox[1]]  
 bboxPoint2 = [bbox[2], bbox[3]]  
 originPoint = [bbox[0], bbox[3]]  
 pixelPositions = []  
 values = pixelMapValue(bbox[0], bbox[2], bbox[1], bbox[3], heigth, width)  
 for i in range(0, len(polygonCoordinates), 2):  
 pixelposx = abs(polygonCoordinates[i] - originPoint[0]) / values[0]  
 pixelposy = abs(polygonCoordinates[i + 1] - originPoint[1]) /values[1]  
 pair = [pixelposx, pixelposy]  
 pixelPositions.append(pair)  
 return pixelPositions

[4] Funcția care face legătura între coordonate și indicii pixelilor

def cropImage(imagePath, pixelIndicesArray):  
 img = cv2.imread(imagePath)  
 pixelIndicesArray = roundFloatList(pixelIndicesArray)  
 pts = np.array(pixelIndicesArray)  
 polygon = cv2.boundingRect(pts)  
 x, y, w, h = polygon  
 croped = img[y: y + h, x:x + w].copy()  
 pts = pts - pts.min(axis=0)  
 mask = np.zeros(croped.shape[:2], np.uint8)  
 cv2.drawContours(mask, [pts], -1, (255, 255, 255), -1, cv2.LINE\_AA)  
 dst = cv2.bitwise\_and(croped, croped, mask=mask)  
 bg = np.ones\_like(croped, np.uint8) \* 255  
 cv2.bitwise\_not(bg, bg, mask=mask)  
 dst2 = bg + dst  
 cv2.imwrite("resources/images/croped.png", croped)  
 cv2.imwrite("resources/images/mask.png", mask)  
 cv2.imwrite("resources/images/dst.png", dst)  
 cv2.imwrite("resources/images/dst2.png", dst2)

[5] Funcția de decupare a parcelei din bounding box

[6] Funcția care crează si aplică măștile pentru cele 3 culori

def colorMask(imagePath, color):  
 try:  
 img = cv2.imread(imagePath)  
 except:  
 logging.exception("Could not read image from relative path " + imagePath)  
 hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 if color.upper() == "GREEN":  
 mask = cv2.inRange(hsv, (41, 25, 25), (70, 255, 255))  
 elif color.upper() == "YELLOW":  
 mask = cv2.inRange(hsv, (23, 25, 25), (40, 255, 255))  
 elif color.upper() == "BROWN":  
 mask = cv2.inRange(hsv, (13, 25, 25), (22, 255, 255))  
 imask = mask > 0  
 color = np.zeros\_like(img, np.uint8)  
 color[imask] = img[imask]  
 cv2.imwrite(f"resources/images/{color}.png", green)

[7] Metoda care extrage colțurile

def extractPolygonCorners(imagePath, color):  
 img = cv2.imread(imagePath)  
 hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 hsv = np.float32(hsv)  
 if color.upper() == "GREEN":  
 hsv = cv2.inRange(hsv, (41, 25, 25), (70, 255, 255))  
 elif color.upper() == "YELLOW":  
 hsv = cv2.inRange(hsv, (23, 25, 25), (40, 255, 255))  
 elif color.upper() == "BROWN":  
 hsv = cv2.inRange(hsv, (13, 25, 25), (22, 255, 255))  
 dst = cv2.cornerHarris(hsv, 5, 3, 0.04)   
 thres = 0.1 \* dst.max()  
 img[dst > thres] = [0, 0, 255]  
 cv2.imwrite('resources/images/cornerHarris.png', img)  
 corners = []  
 for i in range(len(img)):  
 for j in range(len(img[0])):  
 if (img[i][j] == [0, 0, 255]).all():  
 corners.append([j, i])  
 return corners

[8] Parte din informațiile geografice ale unui json de ieșire

"map": {  
 "crs": {  
 "name": "urn:ogc:def:crs:EPSG::4326",  
 "type": "name"  
 },  
 "features": [  
 {

"geometry": {  
 "coordinates": [  
 [  
 45.91813587640448,  
 27.200360944162437  
 ],  
 [  
 45.91813587640448,  
 27.200453010152284  
 ],  
 [  
 45.91814473033706,  
 27.200347791878173  
 ],  
 ],  
 "type": "Polygon"  
 },  
 "id": 1,  
 "properties": {  
 "classnr": 2  
 }

…..

}

[9] Statisticile parcelei.

"statistics": [  
 {  
 "code": 0,  
 "colorHex": "#8A7010",  
 "coverage": 0.007119808197484246  
 },  
 {  
 "code": 1,  
 "colorHex": "#8A6210",  
 "coverage": 0.2173989761415845  
 },  
 {  
 "code": 2,  
 "colorHex": "#768A10",  
 "coverage": 0.7754812156609313  
 },  
 {  
 "code": -1,  
 "colorHex": "#FFFFFF"  
 "coverage": 2.2056309758814254e-06  
 },

{

"area":630400

}]