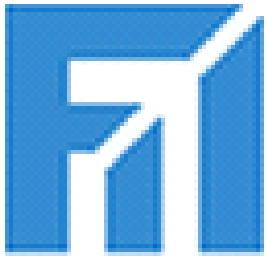
UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI
**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

# API customizabil pentru recunoaşterea de obiecte în imagini

Propusă de

***Bogdan-Cristian Luncaşu***

**Sesiunea**: *iulie, 2017*

Coordonator științific

**Lect. Dr. Anca Ignat**

**UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI**
**FACULTATEA DE INFORMATICĂ**

# API customizabil pentru recunoaşterea de obiecte în imagini

***Bogdan-Crisitan Luncaşu***

**Sesiunea**: *iulie, 2017*

Coordonator științific

**Lect. Dr. Anca Ignat**

DECLARAȚIE PRIVIND ORIGINALITATEA ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „API customizabil pentru recunoaşterea de obiecte în imagini” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași,

Absolvent

*Bogdan-Cristian Luncaşu*

_____

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de accord ca Lucrarea de licenţă cu titlul „API customizabil pentru recunoaşterea de obiecte în imagini” codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de accord ca Facultatea de Informatică de la Universitea Alexandru Ioan Cuza Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent

*Bogdan-Cristian Luncaşu*

# Cuprins

# Introducere şi motivaţie

Procesarea imaginilor este un subiect destul de recent în domeniul IT, aceasta fiind folosită în vaste domenii precum:

- Medicină:
  - Scanare folosind raze X,Gamma
  - Scanare UV
- Robotică:

  Una dintre cele mai mari provocari actuale e reprezentată de cresterea capacitaților de detectare şi identificare a obiectelor de către un robot.
- Armată:
  - Maşini autonome – necesitatea detectării obstacolelor
  - Detectarea obiectelor de la distanţe mari

Dincolo de aplicațiile în domeniile menționate anterior, în ultimii ani această ramura s-a dezvoltat extrem de mult astfel încât a ajuns sa aibă un rol destul de important în crearea conceptului de IOT(Internet of Things) ce reprezintă o reţea de dispozitive interconectate.

Subiectul acestei lucrari îl reprezintă un API[1] generic de recunoaştere a obiectelor. Motivul realizarii acestuia este necesitatea de a recunoaşte anumite obiecte care nu sunt foarte uzuale, astfel încat utilizatorii au posibilitatea de a îşi customiza API[1]-ul folosind doar obiecte menţionate de aceştia, spre deosebire de API[1]-ul oferit de Google cloud vision unde etichetarea este facută implicit de către Google. O altă aplicaţie asemănătoare este Clarifai, aceasta având şi funcţionalitatea de a crea modele customizate pe lângă ceea ce ofera google cloud vision, dar nu face şi localizarea obiectelor în imagine.

Ca client demonstrativ pentru acest API[1] am ales sa realizez o aplicaţie „Treasure Hunt” în timp real pe platforma Android.

1 API – Application Programming Interface este constituit dintr-un set de functii care sunt prezentate ca niste servicii.

# Contribuţii

Subiectul acestei lucrări a fost ales în urma discuției cu doamna profesor îndrumător, Anca Ignat. Comunicarea cu profesorul îndrumator s-a bazat prin întâlniri față în față, astfel încât să poată fi la curent cu evoluția lucrării, dar şi pentru a analiza anumite impedimente apărute în evoluţia acesteia.

Primul pas în dezvoltarea aplicaţiei a fost partea de documentare în ceea ce priveşte modul de funcţionare al librariei OpenCV în limbajul de programare Python dar şi căutarea de metode prin care anumiţi algoritmi gasiţi în acea librărie mi-ar putea fi de folos.

Inițial am pornit cu o idee simplistă, aceea de a detecta obiecte dintr-un anumit domeniu, astfel încât am început cu detectarea de obiecte din sport. Fiind necesară o bază de date foarte mare de imagini astfel incât detectarea sa aibă un oarecare succes am creat un crawler[2] web cu ajutorul căruia să facilitez preluarea seturilor de imagini de pe diferite site-uri, dar se putea observa prezenta anumitor imagini pe care le consideram zgomote.

În final am venit cu ideea de a face userul sa îşi creeze propriul set de obiecte, astfel încât acesta sa îşi poate customiza API-ul.

In concluzie, aplicația vine cu o idee nouă de imbinare a detecţiei obiectelor cu recunoaşterea acestora în imagine, şi anume aceea de a-ți customiza obiectele necesare pentru detectare, astfel încât va fi optimizat şi timpul de detectare deoarece căutarea se va face pe seturi relativ mici de date.

2 crawler – modalitate de indexare a paginilor web

# Tehnologii folosite

## Python

Este un limbaj de programare interpretat, astfel încât instructiunile sunt executate fară necesitatea compilării programului.

Acest limbaj se remarcă prin lizibilitatea codului - identare prin tab-uri pentru a delimita blocurile de cod, posibilitatea de a scrie programe complexe intr-un numar relativ mic de linii de cod şi suport pentru diferite paradigme de programare precum:

- Programare orientata pe obiect
- Programare funcţională
- Programare procedurală

Am folosit python pentru crearea API-ului deoarece ofera mod facil de a lucra cu fişiere şi cu procesarea imaginilor, dar şi prin faptul ca scurteaza timpul de dezvoltare al aplicațiilor compartiv cu alte limbaje precum Java/C#/C/C++.

## OpenCV

Este o librarie open-source creată de către Intel în anul 1999 având ca obiectiv optimizarea aplicaţiilor ce consuma multe resurse CPU cum ar fi aplicaţiile de procesare a imaginilor. Această librarie oferă o multitudine de algoritmi implementaţi folosiţi în arii precum realitatea augmentată, recunoaștere facială, detectare de obiecte, robotică, etc.

Libraria a fost folosită pentru a extrage descriptorii din obiecte şi a face potrivirea cu cei din imaginile pe care cerem detectarea acestora, dar şi pe partea verificare dacă imaginea face parte dintr-un anumit domeniu. Pentru găsirea descriptorilor s-a folosit algoritmul ORB(Oriented FAST and Rotated BRIEF) acesta fiind o alternativă eficientă pentru SIFT sau SURF, iar pentru clasificarea imaginilor am folosit histograma de culori şi algoritmul kNN pentruu n = 1.

# Flask Web Framework

Este un framework web bazat pe Werkzeug[3] și Jinja2[4] . L-am utilizat în crearea serviciilor oferite. API-ul este un API rest, iar resursele sunt aduse developerilor în format JSON.

# SQLAlchemy

Este un ORM pentru limbajul de programare Python. Acesta face legătura intre clasele din python şi tabelele dîntr-o bază de date, oferind astfel o mai mare siguranţă (Rezistența la atacuri de tip SQLInjection), dar în acelaşi timp poate fi văzut ca un layer intre baza de date şi business logic(logica aplicaţiei). în cadrul lucrării a fost folosit impreuna cu o bază de date Postgresql oferita de către cei de la Heroku.

# ASP.NET

A fost creat de Microsoft pentru a facilita crearea de aplicaţii şi servicii web. Spre deosebire de Python, .Net rulează cod compilat crescând astfel performanţele aplicaţiei.

Am folosit ASP.NET pe partea de client astfel încât în fiecare controller se fac apeluri http către api-ul creat în python. Pe partea de modele am creat DTO[5]-uri pentru a le folosi în comunicarea dintre serverul .net şi API.

# JQuery

Este o biblioteca scrisa în Javascript ce faciliteaza procese precum crearea de apeluri asincrone, traversarea arborelui DOM[6] în HTML și crearea de animatii.

3 Werkzeug – WSGI utility
4 Jinja 2 – limbaj de templatizare
5 DTO – (data transfer object) – obiecte folosite in transportul dintre 2 aplicatii ce ruleaza pe servere diferite
6 DOM – Document Object Model – creaza o structura arborescenta din documente xml

# HTML&CSS

HTML[7] este un limbaj de marcare utilizat în prezentarea informaţiilor intr-un navigator/browser. Pentru stilizarea paginilor HTML am folosit CSS[8]

# SailsJS

Este un framework web scris în javascript ce se bazează pe functionalităţile din NodeJS.

Folosește implicit un ORM denumit Waterline,

Oferă o arhitectura bazată pe MVC[9], iar motivul alegerii acestuia îl constituie ușurința integrării websocketilor pentru crearea unei aplicații în timp real.

# Java

Este un limbaj de programare de nivel înalt orientat pe obiect, atât un limbaj compilat dar și interpretat. Este compilat în bytecode și poate fi rulat pe orice mașina virtuala Java(JVM[10]), prin această particularitate limbajul poate fi rulat pe orice sistem de operare ce are instalat un JVM.

Am folosit Java în scrierea aplicației Android deoarece Android Studio oferă suport pentru scrierea aplicațiilor în acest limbaj.

7 HTML – HyperText Markup Language
8 CSS – Cascading Style Sheets
9 MVC – Model View Controller – design pattern
10 JVM – Java Virtual Machine

# Android Studio

Este o platformă de dezvoltare pentru aplicațiile Android. Oferă numeroase ustensile pentru dezvoltarea aplicațiilor pe orice dispozitiv cu sistem de operare Android.

Am folosit această platforma deoarece oferă suport pentru debug al aplicației și suport pentru rularea aplicaţiei pe mașini virtuale ce emuleaza caracteristicile unui dispozitiv Android, astfel încat îti poți testa aplicaţia pe mai multe tipuri de dispozitive.

# Cloud

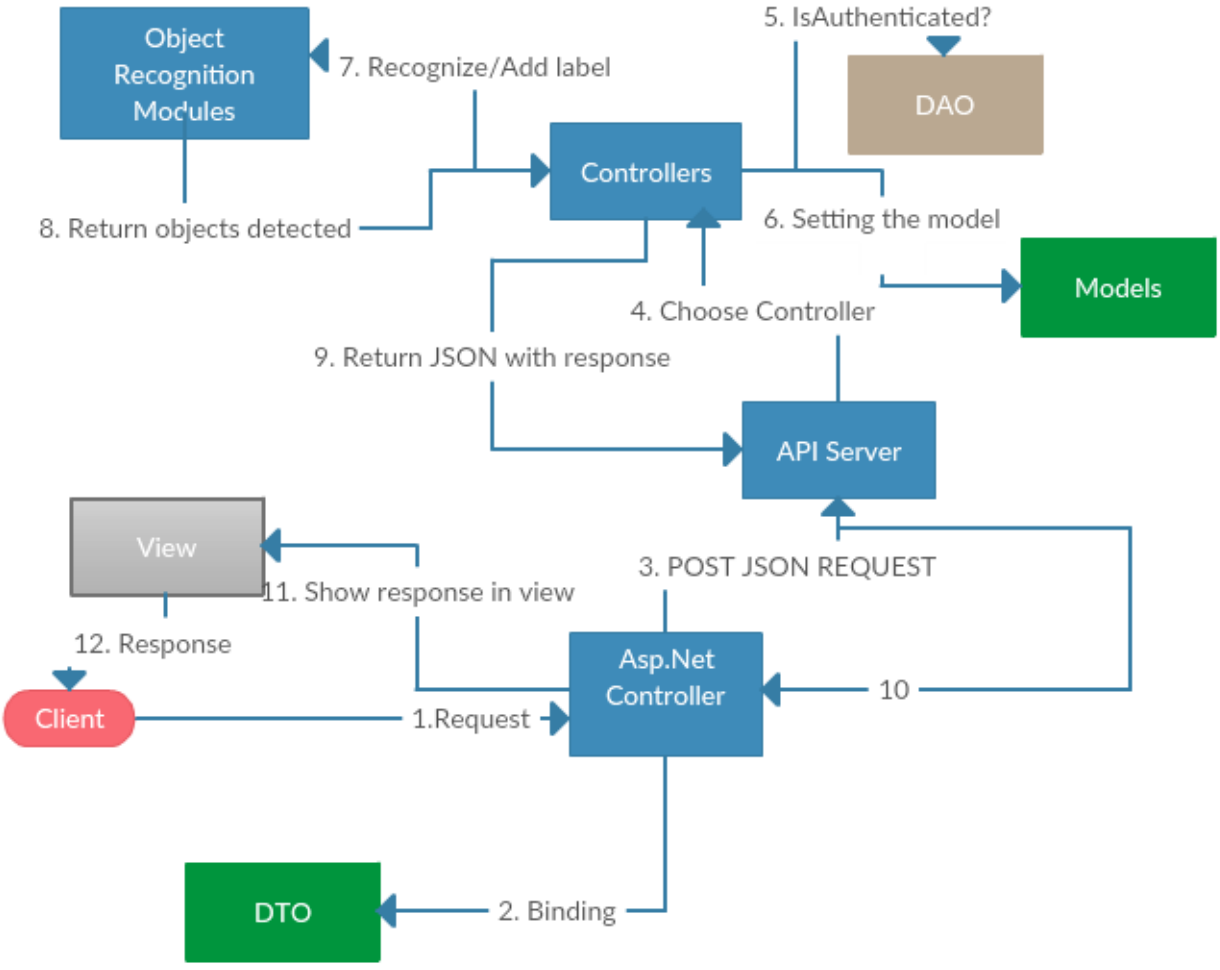
Pentru testarea fluxului de date în cadrul intregii aplicaţii am folosit serviciile oferite de către Heroku şi Openshift. Aceste servicii au fost necesare deoarece folosind un singur PC nu se putea simula un flux real fiind necesara rularea a mai multor servere:

- Flask pentru API a fost gazduit pe heroku.
- Aplicaţia client scrisa în .net a fost gazduită pe openshift deoarece heroku nu oferă suport pentru .net core.
- Serverul scris în sails.js a fost gazduit pe heroku.
- Baza de date Postgresql gazduita în cloudul celor de la Amazon pe platforma EC2 (Elastic Compute Cloud) a fost de asemenea creata prin intermediul celor de la heroku.

# Arhitectura Aplicației

Aplicația este destinată developerilor, deoarece ideea proiectului este de a-ți customiza obiectele detectate într-o imagine, astfel, mergând pe paradigma server-client, developerii pot interacționa cu api-ul folosind aplicația client existentă.

Aceasta aplicație vine şi cu un mic demo prin care developerii pot să își construiască o parere despre eficacitatea api-ului.




1. Arhitectura aplicației

# API

Partea de server este concepută din 2 mari componente, componenta web şi serviciile ce constituie logica aplicaţiei.

Componenta web implementeaza şablonul Model View Controller conferind aplicaţiei suport pentru a avea un cuplaj mic şi o coheziune mare. Controllerele aplicaţiei sunt cele care orchestrează datele primite de serviciul web şi datele trimise către client. în cadrul aplicaţiei avem mai multe controllere :

- UserController: orchestreaza datele ce tin de crearea unui user şi afisarea acestuia.
- AuthorizationController: se ocupă de partea de securitate, astfel încât pentru a avea acces la aplicaţie un client e nevoit sa se logheze folosind endpointul '/auth'
- DeveloperController: se ocupă de generarea cheilor necesare în customizarea api-ului
- ExpandApiController: se ocupă de crearea noilor obiecte
- ObjectRecogniserController: se ocupă cu detectarea obiectelor în imagini
- ImageProcessingController: se ocupă de recunoaşterea de imagini

Fiecare controller foloseşte servicii şi modele pentru a-şi indeplini taskul. Modelele sunt mapate la baza de date prin intermediul orm-ului SQLAlchemy facilitând astfel lucrul cu baza de date.

In cadrul aplicaţiei a fost iniţial folosită o baza de date SQLLite, iar faptul ca folosim un ORM ne confera siguranta ca putem inlocui oricand SQLLite cu o alta baza de date fie ea relationala sau nu, astfel am inlocuit baza de date locală(fişier sqllite) cu una remote Postgresql oferită de Amazon Cloud.

Componenta ce tine de serviciile aplicaţiei este constituita din mai multe module:

- ImageCrawler: ofera posibilitatea de a descărca imagini de pe un anumit website şi de a le folosit în cadrul aplicaţiei pentru detectarea de obiecte sau recunoaştere de imagini.
- ObjectCreator: creaza un folder pentru fiecare obiect al unui user avand structura numelui '%username%object/%label%' şi dezarhiveaza conţinutul arhivei primite de la ExpandApiController

- KnnClassifier: folosit în recunoaşterea de imagini pentru a clasifica o anumita imagine, acesta creaza un fişier cu extensia .dat în care sunt serializate datele folosite la antrenare astfel încât procesul este mult mai rapid decat daca am verifica descriptorii imaginii de test direct cu cei ai imaginilor de antrenament fiind necesara astfel extragerea trasaturilor la fiecare cerere.
- ObjectRecogniserUsingFeatures: se ocupă cu detectarea obiectelor în imagini folosind libraria openCV pentru a obţine descriptorii ORB ai imaginii şi BruteForceMatcher pentru a calcula o norma intre un anumit obiect dîntr-o imagine de test şi obiectele din imaginile de antrenament.

# .Net Client

Aplicaţia client vine cu un mic demo în care un utilizator poate să testeze eficacitatea aplicaţiei făcând recunoaştere de obiecte în imagini din fotbal dar şi clasificarea unei imagini, astfel încât recunoaşterea are loc doar daca imaginea aparţine categoriei SPORT.

Aplicaţia mai ofera şi suport pentru folosirea api-ului astfel încât developerii îl pot customiza folosind o interfaţă grafica, facilitând astfel paşii de adăugare a unui nou obiect. De asemenea, aceştia işi pot genera şi un developer key pentru a-şi putea utiliza api-ul customizat.

Respectand acelaşi design pattern precum cel folosit în crearea api-ului, aplicaţia client devine astfel uşor de modificat în cazul adăugării de noi funcţionalităţi.

# SailsJS Demo API

Acest API a fost creat pentru a testa API-ul customizat şi a observa eficienţa acestuia în detectarea obiectelor. Aplicaţia demo se presupune a fi creată de un ‘oarecare’ developer, iar tema aplicaţiei o reprezintă jocul Treasure Hunt.

Treasure Hunt este un joc în care unul sau mai multi participanţi caută o anumită comoară, aceasta putând fi : un obiect, o locaţie, etc. în cadrul aplicaţiei demo întâlnim o versiune particularizată a acestui joc astfel:

- Jocul e bazat pe challenge-uri.
- Fiecare challenge are un anumit numar de task-uri.

- Un task nu poate fi terminat de mai mulţi participanţi.
- Terminarea taskului se face în urma efactuarii unei fotografii cu comoara gasită.
- În completarea unui task este posibilă necesitatea găsirii mai multor obiecte dar doar a unei singure locaţii.
- În cazul în care există o locaţie şi cel puţin un obiect, fotografia trebuie facuta astfel încât sa cuprindă ambele elemente.

Alegerea acestui framework a fost facută pentru a facilita integrarea websocketilor dar şi pentru avantajele oferite de arhitectura acestuia(MVC) şi de orm-ul implicit, WaterLine.

# Android Client

Acest client foloseşte funcţionalităţile create în cadrul aplicaţiei demo SailsJs pentru a facilita rezolvarea taskurilor în cadrul jocului, astfel, cu ajutorul aplicaţiei, participantii sunt la un ‘touch’ distanţă de completarea unui task. Arhitectura acestei aplicaţii este constituită din 5 componente:

- Activităţi – prezintă informaţiile în mod grafic cu ajutorul layout-urilor.
- Servicii – apelează api-ul în urma anumitor cereri alea utilizatorului, proceseaza informaţiile primite şi le afişeaza în cadrul activitatii.
- Adaptoare – ajută la prezentarea grafică a unor liste de elemente.
- Modele – folosite pentru a transforma obiectele json în obiecte Java, facilitând astfel accesul la date.
- Sockets – folosiţi pentru primirea notificărilor de la server

# Detalii de implementare

## API

- runserver.py – Seteaza hostul şi portul la care serverul poate fi accesat şi porneşte serverul.

```
if __name__ == '__main__':
    HOST = '0.0.0.0'
    try:
        PORT = int(environ.get('PORT', 5000))
    except ValueError:
        PORT = 5555
    app.run(HOST, PORT)
```

- database.py – Modul ce se ocupă de configurarea conexiunii la baza de date şi de crearea tabelelor pe baza modelelor. în crearea aplicaţiei am folosit SQLLite iniţial iar după am înlocuit cu PostgreSQL, astfel conexiunea se poate face uşor cu orice baza de date atât timp cât exista in python adaptor pentru acea bază de date.

```
engine = create_engine('sqlite:///tmp/recoginserapi.db',
convert_unicode=True)
```

- models.py – Modul ce se ocupă de descrierea modelelor ce vor fi mapate la baza de date. Fiecare model extinde declarative_base din sqlalchemy.ext.declarative, astfel orm-ul va sti ca acesta este un model şi ca necesita mapat la baza de date. Modelele pot specifica tabela la care sunt mapate prin proprietatea __tablename__ . Fiecare

proprietatea ce reprezintă o anumita coloana din baza de date este scrisa sub forma proprietate = Column(tipul proprietatii, alte aspecte precum primary key,unique). Relatiile sunt descrise cu ajutorul metodei relationship astfel încât se mentioneaza ca parametru modelul cu care se creaza relatia.

- controllere – fiecare controller îşi seteaza endpointurile la care va fi solicitat şi metodele HTTP aferente.

```
@app.route('/[endpoint]',methods=['HTTPMETHOD'])
```

Este activat Cross-origin resource sharing(CORS) pentru a permite ca resursele sa poata fi accesate de pe alte domenii exceptand cel unde sunt hostate resursele iar controllere sunt adnotate cu `@cross_origin()`

Fiecare controller afişeaza resursele cerute în format JSON folosind metoda jsonify din flask şi codul de return.

```
return jsonify(object),XXX - return code
```

Query-urile se fac folosind orm-ul SQLAlchemy sub forma Obiect.query.filter(Predicate).

Pentru a genera o cheie pentru developer se folosesc caractere ASCII random, concatenate printr-un for de la 0 la 128 doar daca numarul de chei deja generate nu depaseste cifra 4:

```
devkey =''.join(random.SystemRandom().choice(string.ascii_uppercase + string.digits + string.ascii_lowercase) for _ în range(128))
```

- imagecrawler.py – Primeşte ca parametru în constructor URL-urile de unde vor fi descărcate imaginile. Menţine o listă a tuturor URL-urilor accesate inclusiv a celor aflate în paginile html pentru a nu accesa aceleaşi resurse de mai multe ori.

  Metoda crawling primeşte ca parametri numarul de imagini targetat, astfel încât crawlerul să nu fie lăsat sa descarce un număr imens de imagini şi eticheta imaginilor din URL-uri.

  ```
  def crawling(self,targetimages=10,label="notsport")
  ```

  Fiecare url este accesat cu ajutorul obiectului request din modulul urllib2, iar codul sursa este parsat cu ajutorul clasei BeautifulSoup

```
html=request.urlopen(self.urls[0]).read()
soup=BeautifulSoup(html)
```

Folosind metoda findAll se cauta toate tagurile img şi se obtin url-urile imaginilor. Aceste url-uri sunt trimise ca parametru către metoda create_opencv_image_from_url care ia streamul primit de la url-ul imaginii şi îl transforma în array numpy. Biblioteca cv2 are o metoda imdecode care transforma un array numpy într-o imagine cv2 iar apoi aceasta este salvată în folderul dedicat etichetei primite ca parametru de crawler.

- knnClassifier.py – Se ocupă cu clasificarea imaginilor folosind histograma de culori. Histograma se obţine cu ajutorul metodei clacHist din OpenCV. Primul parametru îl reprezintă imaginea pentru care se calculeaza histograma, al doilea îl reprezintă canalele de culori RGB, None specifica ca nu se aplica nici o masca peste imagine. Cel de-al patrulea parametru specifica dimensiunea histogramei astfel încât intervalul [0,256] este adus la un interval [0,16] astfel încât un pixel va aparţine intervalului [0,16] pentru un anumit canal în functie de valoarea lui pe intervalul [0,256] impartita la 16. Ultimul parametru îl reprezintă intevalul în care un pixel poate lua o valoarea intr-un anumit canal.

```
cv2.calcHist([image], [0, 1, 2], None, [16, 16, 16],
                              [0, 256, 0, 256, 0, 256])
```

Se mentinte un vector de tuple (histograma,etichetă) ce este serializat la sfarsitul antrenarii astfel încât testarea se poate face doar incarcand datele din fişierul serializat şi calculand distanta euclidiana dintre histograma imaginii de test şi histogramele imaginilor de antrenament.

```
def euclidian_distance(self,v1,v2):
    d=0
    for i,j în zip(v1,v2):
        d+=math.pow(i-j,2)
    return math.sqrt(d)
```

- createobject.py – conţine clasa CreateObject care primeşte în constructor arhiva de imagini cu obiectul creat, usernameul utilizatorului ce solicită crearea obiectului, şi eticheta acestuia. Arhiva primită este codificată base64, astfel este necesara o

decodare a acesteia folosind modulul base64. Dupa decodificare sunt extrase imaginile din arhiva şi puse în folderul creat pentru obiect.

- usingfeatures.py – conţine class ObjectRecognize ce primeşte ca parametri în constructor usernameul utilizatorului ce solicita recunoaşterea de obiecte, acesta fiind setat default cu caracterul empty astfel încât la apelarea api-ului de către un utilizator nelogat acesta va primi lista de obiecte detectate în imagine din setul de etichete implicit şi lista de obiecte a utilizatorului – implicit daca nu este logat şi lista din baza de date daca este logat. Imaginea este converită la o imagine ce foloseşte un singur canal pentru culori, adica o imagine alb negru pentru a eficientiza gasirea descriptorilor. Se genereaza descriptorii imaginii de test:

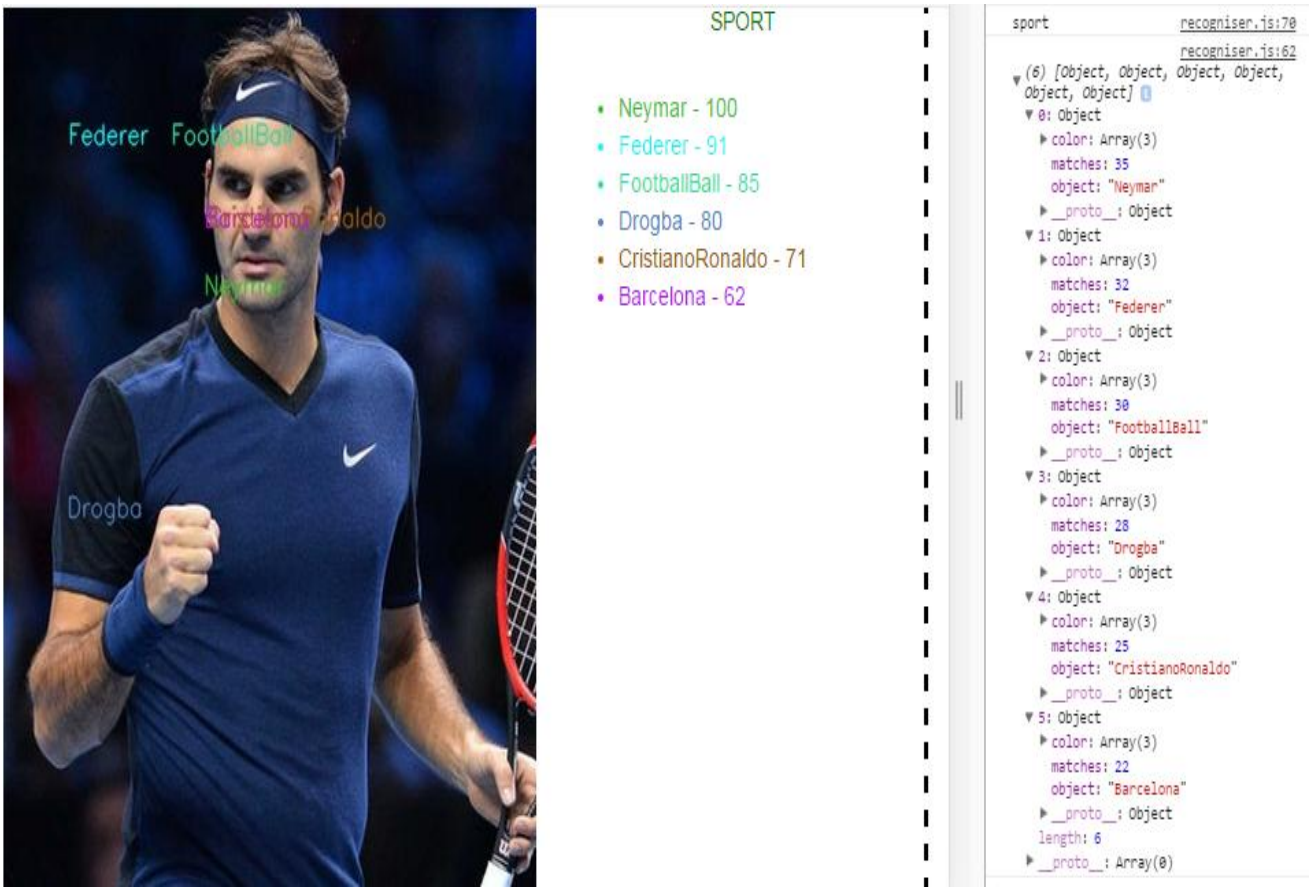
```
    def gen_orb_features(self,gray_img):
        orb = cv2.ORB_create()
        kp = orb.detect(gray_img,None)
        return orb.compute(gray_img, kp)
```

Se parcurge astfel lista de obiecte primită în constructor, iar pentru fiecare obiect se parcurge lista de imagini din folderul aferent şi se verifica existenta descriptorilor fiecarei imagini în vectorul iniţializat cu lista de obiecte de forma {“imagepath”:descriptors}. Daca descriptorul nu exista – asta însemnând că un obiect nou a fost adaugat sau că au fost adaugate noi imagini unui obiect deja existent – atunci se calculează descriptorii imaginii şi se adaugă în lista de descriptori. Dupa ce se asigură existenta descriptorilor se face match intre descriptorii imaginii de test şi descriptorii imaginii actuale. Se foloseşte BFMatch ce implementeaza algoritmul Knn iar norma este setata ca fiind norma L2 (euclidiana)

```
matches =
self.bf.match(descriptors_image_train,descriptors_image_test)
```

Pentru fiecare obiect, se obţine numarul maxim de matchuri intre imaginea de test şi o imagine din folderul aferent obiectului. Existenta obiectului în imagine este data de un treshold(prag) pe care numarul de matchuri trebuie sa îl depaseasca. în cazul în care exista cel putin un obiect detectat în imagine, este aproximata pozitia acestuia folosind coordonatele descriptorilor ce fac match astfel încât se ia cel mai de sus stanga descriptor şi incepand cu acea coordonata este scrisa eticheta obiectului folosind o culoare random.

# .Net Client




Dupa cum se poate observa se pot detecta obiecte care nu sunt în imagini datorita trecerii pragului de potriviri ale descriptorilor, dar obiectul tinta va fi gasit de cele mai multe ori printre primele obiecte detectate în imagine.

- Settings.cs – Sunt mentinute atribute de configurare precum URL-ul api-ului.
- Startup.cs – Se ocupă cu configurarea obiectelor ce vor raspunde la cererile primite de aplicaţie. în constructor sunt setate sursele de configurare. Clasa Startup necesita implementarea metodei Configure, aceasta specificand cum sunt tratate cererile HTTP. Este folosit design patternul Dependency Injection astfel încât obiectele sunt trimise ca parametru în functia de configurare. IApplicationBuilder configureaza

middleware-urile ce vor procesa cererile primite. Pentru mecanismul de autentificare este folosit middleware-ul CookieAuthentication. Schema de autentificare este necesara în cazul în care dorim sa folosim mai multe middleware-uri de tip CookieAutentication. Este setat sablonul rutelor din aplicaţie sub forma controller/actiune/id?(id poate fi null).

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env, ILoggerFactory loggerFactory)
        {
        loggerFactory.
            AddConsole(Configuration.GetSection("Logging"));

            loggerFactory.AddDebug();
            CookieAuthenticationOptions options =
                    new CookieAuthenticationOptions();

            options.AuthenticationScheme = "Cookies";
            options.LoginPath = new PathString("/UserAccount/Login");
            options.CookieName = "UserCookie";
            options.AutomaticAuthenticate = true;

            app.UseCookieAuthentication(options);

            app.UseApplicationInsightsRequestTelemetry();

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseBrowserLink();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }
            app.UseApplicationInsightsExceptionTelemetry();

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template:
"{controller=Home}/{action=Index}/{id?}");
            });
        }
```

Metoda ConfigureServices este optionala, insa în cazul în care este implementata, aceasta este apelata înaintea metodei Configuration iar rolul ei este de a lega funcţionalităţi ce necesita o configurare substantiala precum EntityFramework sau MVC.

- project.json – Conţine lista de dependinte necesara aplicaţiei ce este folosită de NuGet(manager de pachete pentru dezvoltarea aplicatiilor pe platforma Microsoft), proprietati legate e metadata: versiune, autori, se mai pot specifica optiuni de build sau/si publish.

```
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.0.1",
      "type": "platform"
    },
    ...
  },
  "buildOptions": {
    "emitEntryPoint": true,
    "preserveCompilationContext": true
  },
  "publishOptions": {
    "include": [
      "wwwroot",
      "**/*.cshtml",
      "appsettings.json",
      "web.config"
    ]
  },
  ...
```

- *Controller.cs – în Constructorul unui controller este iniţializat un obiect de tipul HttpClient cu adresa de baza din clasa sealed Settings (Settings.URL), iar headerul Content-Type este setat ca „application/json”. Acest obiect este folosit pentru a face cereri către API. Acţiunile ce apelează api-ul sunt metode asincrone astfel încât serverul sa nu fie blocat în cazul în care API-ul raspunde mai greu deoarece sunt necesare operatii consumatoare de timp sau din pricina latentei în retea. Pentru logarea în aplicaţie endpointul este UserAccount/Login. Aici sunt primite datele de

logare(username,parola) şi trimise către API în format JSON. Daca logarea are loc cu succes atunci se foloseşte mecanismul bazat pe claims(revendicari) pentru a autoriza userul, iar autentificarea se face folosind CookieAuthentication middleware.

```
 [ActionName("Login")]
 [HttpPost]
 public async Task<ActionResult> Login_Post(UserModel model)
 {
  HttpResponseMessage responseMessage;
     var content = new StringContent( JsonConvert.
         SerializeObject(model), Encoding.UTF8, "application/json");
     using (responseMessage =
               await client.PostAsync(Settings.URL + "auth", content))
     {
          if (responseMessage.IsSuccessStatusCode)
          {
             var responseData = responseMessage.Content
                                  .ReadAsStringAsync().Result;
             Dictionary<string, Dictionary<string,string>> values =
                   JsonConvert.DeserializeObject<Dictionary<string,
                   Dictionary<string, string>>>(responseData);
              if (values.ContainsKey("message"))
              {
                        ViewBag.Error = values["message"];
                        return View(new UserModel());
                  }
                  else
                  {
                     Dictionary<string,string> user=values["user"];
                      var identity = new ClaimsIdentity(new[] {
                      new Claim(ClaimTypes.Name, model.username),
                      new Claim("Token", user["token"])
                          }, "ApplicationCookie");
                     var principal = new ClaimsPrincipal(identity);

                      await
HttpContext.Authentication.SignInAsync("Cookies", principal,
                           new AuthenticationProperties
                           {
                               ExpiresUtc =
DateTime.UtcNow.AddMinutes(30),
                               IsPersistent = false,
                               AllowRefresh = false
                           });

                      return Redirect("/");
                  }
```

```
                }
            }

            ViewBag.Error = "Something went wrong";
            return View(new UserModel());
        }
```

Dupa ce utilizatorul s-a logat cu succes acesta va folosi tokenul primit de la API în urma logarii pentru a putea accesa resurse securizate. Pentru usurinta crearii query-urilor într-o obiect de tipul IEnumerable se folosesc expresii lambda cu ajutorul librariei LINQ.

```
        public string GetToken()
        {
            var identity = (ClaimsIdentity)User.Identity;
            return identity.Claims.AsEnumerable().Where(c =>
                c.Type.Equals("Token")).FirstOrDefault().Value;
        }
```

- recogniser.js – Detectia şi recunoaşterea obiectelor în imagini se face folosind apeluri asincrone AJAX cu ajutorul librariei jQuery. Imaginea este preluata din inputul cu type=’file’ din html prin id-ul acestuia:

  ```
  var image = $("#imagefile");
  ```

  si transformata intr-un buffer de octeti. Fiecare octet avand 256 biti, astfel, folosind metoda String.fromCharCode.apply() este transformat bufferul de octeti intr-un string format din caractere ASCII. Dupa obţinerea stringului, acesta este codificat în Base64 şi trimis către API folosind AJAX, iar în functia de succes se ataseaza imaginea primită de la API în html impreuna cu obiectele detectate în acea imagine.

- upload.js – Procedeul este asemanator cu cel descris în recogniser.js, diferenta fiind facuta de endpointul la care sunt trimise datele şi de structura jsonului trimis.

## SailsJS Demo API

- package.json – sunt configurate proprietati legate de metadate şi dependintele necesare proiectului.

- connections.js – sunt configurate conexiunile la diferite baze de date sub forma nume_conexiune: obiect de configurare.

```
  herokuPostgresqlServer: {
    adapter: 'sails-postgresql',
    host: 'ec2-79-125-13-42.eu-west-1.compute.amazonaws.com',
    ssl: 'require',
    user: '***********',
    password: '***********',
    database: '***********'
    }
```

- models.js – sunt configurate conexiunea la baza de date:

  `connection: 'herokuPostgresqlServer'`

  dar şi modul în care se vor face migrarile la baza de date:

  `migrate: 'alter'`

- routes.js – sunt configurate endpointurile la care se pot face requesturi în cadrul aplicaţiei şi sunt scrise sub forma `HTTPMETHOD /[controller]/[:id]/[action]` : `Controller.metoda_apelata`

```
  'POST /challenge/:challengeId/task' : 'TaskController.create',
  'GET /challenge/:challengeId/task' : 'TaskController.find',
  'POST /challenge' : 'ChallengeController.create',
  'POST /user' : 'UserController.create',
  'GET /user' : 'UserController.find',
  'GET /user/:id' : 'UserController.findOne',
  'POST /auth' : 'AuthController.index',
  'GET /challenge/:challengeId/task/:id' : 'TaskController.findOne',
  'POST /challenge/:challengeId/task/:taskId/solve' :
            'TaskController.solve',
  'POST /challenge/:challengeId/accept' :
            'UserController.acceptChallenge',
  'GET /challenge' : 'ChallengeController.find',
  'GET /challenge/subscribe' :
           'ChallengeController.subscribeChallenge',
  'GET /challenge/subscribeTasks' : 'TaskController.subscribeTask',
  'PUT /challenge/:id' : 'ChallengeController.update'
```

- policies.js – sunt setate politicile de securitate, astfel încât anumite endpoint-uri pot fi accesate doar daca se indeplinesc anumite criterii. în cadrul aplicaţiei, deoarece numarul de endpoint-uri securizate este mai mare decat a celor nesecurizate se seteaza

initial toate endpointurile cu politica isAuthorized iar dupa se suprascriu setarile pentru acţiunile ce nu necesita aplicarea acestei politici.

```
  '*': ['isAuthorized'],
  'UserController': {
    'create' : true,
        'findOne' :true
  },

  'AuthController': {
    '*': true
  }
```

- isAuthorized.js – verifica daca tokenul primit este valid. Trimiterea tokenului se face prin headerul Authorization, formatul fiind Authorization=Bearer token. Un request aplicat unui endpoint trece prin aceasta politica doar daca aceasta este aplicata endpointului, astfel  se verifica existenta headerului de authorization şi se face validarea tokenului iar daca totul este valid atunci se trimite requestrul către controllerul apelat. în cazul în care requestul este facut de socket atunci tokenul este trimis prin http query string.

```
  if(req.isSocket){
    token=req.query.token;
  }else if (req.headers && req.headers.authorization) {
    var parts = req.headers.authorization.split(' ');
    if (parts.length == 2) {
      var scheme = parts[0],
        credentials = parts[1];
      if (/^Bearer$/i.test(scheme)) {
        token = credentials;
      }
    } else {
      return res.json(401, {err: 'Format is Authorization: Bearer
[token]'});
    }
  }
  else {
    return res.json(401, {err: 'No Authorization header was found'});
  }

  jwToken.verify(token, function (err, token) {
    if (err) return res.json(401, {err: 'Invalid Token!'});
    req.token = token;
    next();
  });
```

- jwToken.js – modul ce se ocupă de generarea şi validarea tokenilor. Un JSON Web Token este un obiect json folosit pentru a reprezenta un set de informaţii folosite în comunicare de către 2 endpointuri. Un jwToken este format din antet, payload şi semnatură şi are forma header.payload.semnatură Antetul ofera informaţii despre cum este creata semnatura. Implicit este folosit algoritmul HMAC-SHA256.

  HMAC(cheie_secreta,mesaj) = SHA256((cheie_derivata xor padare_externa) || (cheie_derivata xor padare_interna) || mesaj)

  O cheie derivată este formată din padarea la dreapta în cazul în care lungimea cheii secrete este mai mică decat cea a lungimii blocului, în cazul actual lungimea este de 256 biti, iar daca lungimea cheii secrete depaşeşte lungimea blocului se aplica un hash peste cheia secreta. Padarea externa este o constata de forma 0x5c * lungimea blocului iar cea interna tot o constanta de forma 0x36 * lungimea blocului. Dupa ce este generata semnatura, se concatenează antentul codificat în baza 64 cu mesajul codificat tot în baza 64 – mesajul în cazul de fata este id-ul userului – şi hashul criptografic obtinut prin aplicarea algoritmului HMAC-SHA256 folosind cheia secreta şi payloadul(cheia userului). Astfel se asigura autentificarea şi integritatea datelor.

```
var
  jwt = require('jsonwebtoken'),
  tokenSecret = "cheia secreta";
module.exports.issue = function(payload) {
  return jwt.sign(
    payload,
    tokenSecret,
    {
      expiresIn : 2*60*60
    }
  );
};
```

Pentru a face aplicaţia sa fie real time am folosit mecanismul de PublishSubscribe prin care un socket se poate subscrie la o resursa şi va fi notificat în cazul în care resursa este modificată/ştearsă sau explicit prin apelul unor metode de tipul message(id,json)/publishUpdate/etc. astfel încât id-ul sa specifice camera la care socketii s-au abonat. Pentru a putea primi notificări referitor la crearea unei resurse se foloseşte metoda watch

asupra unei resurse sub forma Resursa.watch(request) .

```
    if (!req.isSocket) {
      return res.badRequest('Only a client socket can subscribe to
challenges.');
    }

    Challenge.find().exec(function (err, challenges) {
        for(var index în challenges){
            Challenge.subscribe(req, "__challenge"+challenges[index]["id"]);
        }
    });
    return res.ok();
```

Pentru a apela API-ul customizat se foloseşte biblioteca request astfel încât se trimit datele : cheia de developer şi imaginea primită de la clientul de Android codificată în baza 64, şi verificand obiectele primite de la api cu etichetele din baza de date ce sunt necesare pentru taskul curent, utilizatorul va fi informat daca taskul a fost rezolvat cu succes sau nu, iar socketii abonati la acel task vor primi o notificare în clientul Android prin care este specificat ca taskul X a fost rezolvat.

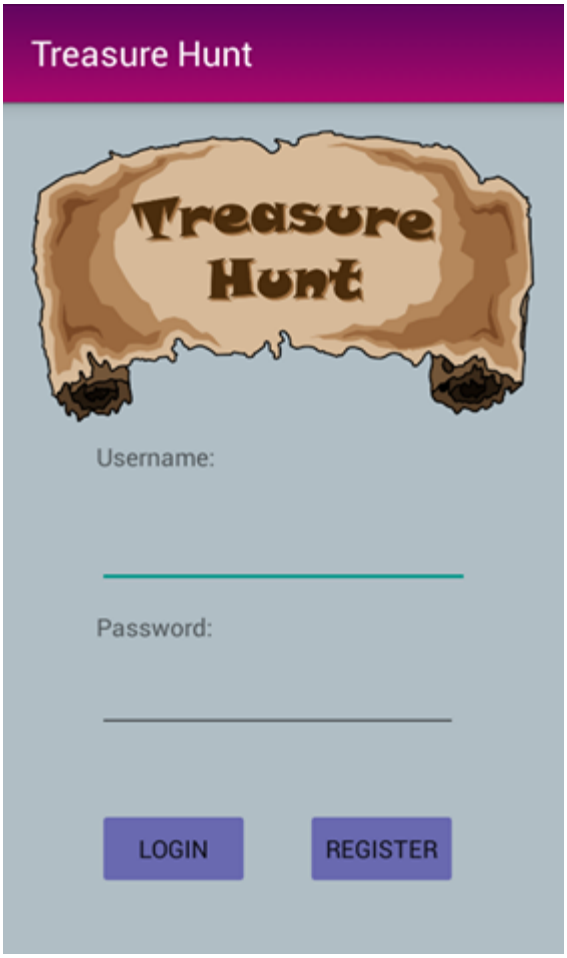
# Android

Pentru a da un exemplu de aplicabilitate a API-ului am ales crearea unui joc TreasureHunt, iar platforma Android mi s-a parut cea mai suitabilă alegere pentru crearea acestuia deoarece este necesara deplasarea utilizatorului în diferite locaţii pentru a găsi anumite `comori` şi a le fotografia.

- AndroidManifest.xml – sunt setate permisiunile aplicaţiei, metadata dar şi serviciile/activitaţile oferite de aceasta. Pentru a putea rula aplicaţia sunt necesare acceptarea a trei permisiuni, acestea fiind permisiunea de acces la internet, de folosire a camerei foto şi de accesare a memoriei externe.

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-feature android:name="android.hardware.camera"
        android:required="true" />
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- MainActivity.java – sunt tratate acţiunile de Login şi Register, astfel se obţine fiecare buton pe baza id-ului din xml şi se setează un ClickListener. în urma apăsarii butonului de login se iau datele din input şi se trimit către un AsyncTask astfel încât logarea să se facă asincron, fiind creat un nou proces ce se va ocupă de acest task. Pentru register mai există un pas pana la apelarea unui AsyncTask şi anume cel de introducere a unei adrese de email, astfel se creeaza un DialogBox în care utilizatorul îşi va putea specifica adresa de email cu care vrea să se înregistreze.





```
login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String username = ((EditText)
findViewById(R.id.username)).getText().toString();
```

```
                String password = ((EditText)
findViewById(R.id.password)).getText().toString();
                new LoginTask(APIURL, username, password,
MainActivity.this).execute();
            }});
```

APIURL este un string ce conţine adresa către serverul sails.js iar prin trimiterea ca parametru în constructorul LoginTask-ului se folosesc principiile design patternului Dependency Injection, astfel url-ul poate fi modificat intr-un singur loc în cazul în care serverul este mutat pe alt domeniu.

De asemenea în manifest este setată şi tema aplicaţiei `android:theme="@style/AppTheme"` , aceasta fiind regasită în fişierul styles.xml din folderul res/values. Acest style extinde styleul Theme.AppCompat.Light.DarkActionBar şi suprascrie actionBarStyle folosind un gradient.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:angle="90"
        android:endColor="#61045F"
        android:startColor="#AA076B" />
</shape>
```

- LoginTask.java – clasa LoginTask extinde clasa generica AsyncTask şi suprascrie cele 3 metode:
    - onPreExecute: această metodă este executată în threadul de UI şi face anumite acţiuni necesare înainte de pornirea taskului în background. în cazul aplicaţiei actuale este pornit un ProgressDialog astfel încât utilizatorul sa nu trimita mai multe requesturi de login de cât este nevoie şi pentru ca acesta să poată observa că operaţia de login e în curs de desfaşurare.
    - doInBackground – primeşte ca parametru date necesare pentru rezolvarea taskului, în cazul actual tipul generic este Void deoarece nu sunt necesare date pentru indeplinirea taskului şi executa taskul în background folosind un nou fir de executie. Pentru trimiterea datelor în retea se foloseşte HttpClient şi metoda post implementata de aceasta. Metoda post are ca ultim parametru un obiect de tip JsonHttpResponseHandler, acesta funcţionând ca un callback în urma

requestului. în cazul în care loginul are loc cu succes este extras tokenul din datele primite de la server şi trimis mai departe în onPostExecute.

- onPostExecute – primeşte tokenul şi îl salveaza în Shared Preferences, astfel încât acesta să poată fi extra din orice activitate a aplicaţiei. Dupa ce tokenul este inserat în Shared Preferences este pornit serviciul de notificare care ţine legătura cu serverul prin socketi şi este notificat atunci când apar schimbări pe server cum ar fi terminarea unui anumit task dintr-un challenge sau acceptarea unui challenge de către un anumit user. Dupa pornirea serviciului, se deschide o noua activitate în care sunt afisate challenge-urile existente. Deschiderea unei noi activitati se face folosind Intenturi şi metoda startActivity(intent) din activitatea curenta. Un intent este modul prin care datele pot fi partajate intre activitati.

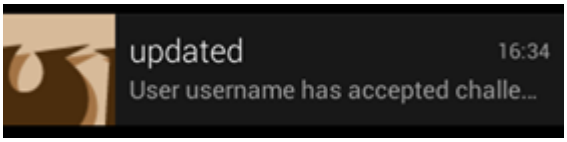

- ChallengeNotification.java – extinde clasa Service astfel încât notificarile sa poata fi primite în background. în constructor se creaza socketul şi se conecteaza la server:

```
IO.Options options=new IO.Options();
```

```
socket = IO.socket(APIURL + "?__sails_io_sdk_version=0.11.0",
options);
```

In metoda onStartCommand se face abonarea la resurse prin trimiterea unui request la endpointul prin care un utilizator se poate abona folosind metoda emit din socket io iar receptarea datelor primite se face prin metoda on având ca parametri numele eventului şi un event listener pentru a accesa datele şi a afişa notificarile.

```
JSONObject requestChallenge = new JSONObject();
try {
    requestChallenge.put("url", "/challenge/subscribe");
} catch (JSONException e) {
    e.printStackTrace();
}
socket.emit("get", requestChallenge);
socket.on("challenge", listener);
private Emitter.Listener listener = new Emitter.Listener() {
        @Override
        public void call(Object... args) {
            try {
                JSONObject data = new
JSONObject(String.valueOf(args[0])).getJSONObject("data");
                String verb = new
JSONObject(String.valueOf(args[0])).getString("verb");
                String message = data.getString("message");
                NotificationManager notificationManager =
                        (NotificationManager)
getSystemService(Service.NOTIFICATION_SERVICE);
                NotificationCompat.Builder notification =
               new
NotificationCompat.Builder(ChallengeNotification.this)
                                    .setSmallIcon(R.drawable.logo)
                                    .setContentTitle(verb)
                                    .setContentText(message);


notificationManager.notify((int)(Math.random()*10000),
notification.build());
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    };
```




- ChallengeActivity.java – Activitatea de Challenge conţine 2 listview-uri prin care sunt delimitate challenge-urile acceptate de user şi cele neacceptate încă. Pentru aducerea datelor de pe server se foloseşte un serviciu generic RetrieveDataTask<T> prin care se specifica url-ul de unde vor fi luate datele şi modelul ce va încapsula acele date.

```
RequestParams rp = new RequestParams();
client.addHeader("Authorization", "Bearer " + token);
final List<JSONObject> myobjects = new LinkedList<>();
JsonHttpResponseHandler responseHandler = new
JsonHttpResponseHandler() {
            @Override
            public void onSuccess(int statusCode, Header[] headers,
JSONArray response) {
                if (statusCode == 200) {
                    try {
                        for (int i = 0; i < response.length(); i++) {
                            myobjects.add(response.getJSONObject(i));
                        }
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }

                }
            }
        };
 client.get(apiurl, responseHandler);
 Gson gson = new Gson();

 JsonParser parser = new JsonParser();
 List<T> listOfObjects = new LinkedList<>();
 for (JSONObject obj : myobjects) {
        JsonElement mJson = parser.parse(obj.toString());
        listOfObjects.add(gson.fromJson(mJson, type));
 }
 return listOfObjects;
```

Obiectul gson transforma un obiect de tip json intr-un obiect java, iar parametrul type este setat în constructor ca fiind de tip Class<T> astfel specificand tipul parametrului generic T.

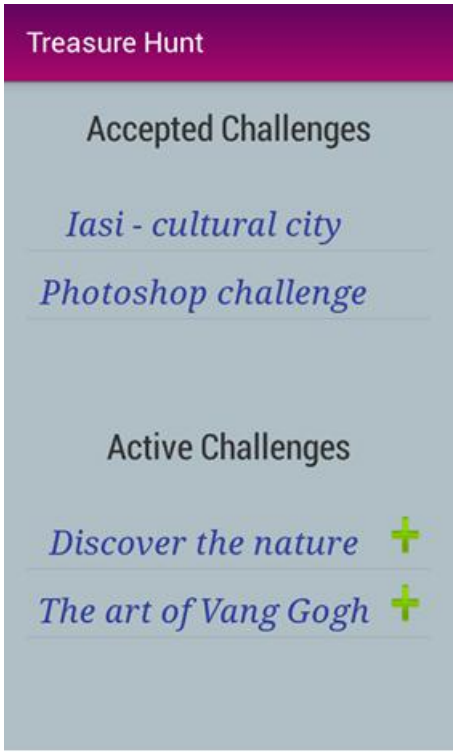
In cadrul metodei onPostExecute se verifica tipul modelului astfel încât sa se faca referinta la obiectele din layout prin intermediul id-ului acestora. Pentru a afisa lista din java în listview se foloseşte un adaptor.

Pentru a avea acces la informaţii legate de challenge acesta trebuie mai intai acceptat, iar apoi din lista challengeurilor acceptate se poate face click pe cel al caror informaţii vrem sa le vedem.

```
if (context instanceof ChallengeActivity &&
                type.equals(ChallengeModel.class)) {
            String username = (shared.getString("username", ""));
            List<ChallengeModel> challenges = (List<ChallengeModel>)
t;
            List<ChallengeModel> accepted = new LinkedList<>();
            List<ChallengeModel> active = new LinkedList<>();

            for (ChallengeModel cm : challenges) {
                if (cm.isAccepted())
                    accepted.add(cm);
                else
                    active.add(cm);
            }

        ListView activeChallengesView = ((ListView)
context.findViewById(R.id.active_challenges_list));
            activeChallengesView.setAdapter(new
ChallengesListAdapter(context.getLayoutInflater(), active, apiurl +,
context));

            final ListView acceptedChallengesView = ((ListView)
context.findViewById(R.id.challenges_list));
            acceptedChallengesView.setAdapter(new
ChallengesListAdapter(context.getLayoutInflater(), accepted, apiurl,
context));

            acceptedChallengesView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {
                    Intent intent = new Intent(context,
TaskActivity.class);
                    intent.putExtra("id",((ChallengeModel)
acceptedChallengesView.getAdapter().getItem(position)).getId());
                    intent.putExtra("description",((ChallengeModel)
acceptedChallengesView.getAdapter().getItem(position)).getDescription
());
```

```
                    intent.putExtra("title",((ChallengeModel)
acceptedChallengesView.getAdapter().getItem(position)).getTitle());
                    context.startActivity(intent);
                }
            });
        }
```

La apăsarea unuia dintre challengeurile acceptate se va deschide o activitate nouă în care vor fi prezentate taskurile ce necesită a fi completate în cadrul challengeului.





La fel ca în cazul challengeurilor, se foloseşte un asynctask şi httpclient pentru primirea datelor iar lista primită va fi trimisă la adaptorul care face legătura intre lista din java şi listview.

Lista de taskuri are urmatorul format al jsonului:

```
[
    {
        "title": "Almost there",
        "description": "Gaseste cel mai important obiectiv cultural
din Iasi.",
        "active": true,
        "challenge": 2,
        "user": 0,
```

```
        "id": 4,
        "createdAt": null,
        "updatedAt": "2017-06-25T17:52:55.000Z"
    }
]
```

astfel proprietatea active sugerează faptul că taskul nu a fost încă terminat.

In cazul în care proprietatea active este false atunci taskul nu va mai putea fi accesat din aplicaţie deoarece este deja terminat de către un user, iar conform regulilor jocului taskul nu poate fi terminat decat de o singura persoană.

- SolveTaskActivity.java – Afişeaza titlul şi descrierea taskului curent, iar pentru a termina taskul este necesara trimiterea către server a unei imagini, astfel, se creaza un nou intent cu numele acţiunii: android.provider.MediaStore.ACTION_IMAGE_CAPTURE şi se foloseşte metoda startActivityForResult(intent,REQUESTID) astfel încât la terminarea activitatii de obţinere a imaginii aceasta va fi trimisa către activitatea curenta şi folosită în onActivityResult – metoda suprascrisa – se verifica REQUESTID sa fie la fel cu resultCode trimis din activitatea photo, dupa se transforma bitmapul în ByteArrayOutputStream prin compresarea acestuia şi transformarea acestuia intr-un string în base64 şi trimiterea datelor către un AsyncTask pentru a putea fi trimise către server.

```
        if (requestCode == CAMERA_REQUEST && resultCode ==
Activity.RESULT_OK) {
            Bitmap photo = (Bitmap) data.getExtras().get("data");
            takenPhoto.setImageBitmap(photo);
            ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
            photo.compress(Bitmap.CompressFormat.PNG, 100,
byteArrayOutputStream);
            String encodedPhoto =
                    Base64.encodeToString(byteArrayOutputStream
.toByteArray(), Base64.DEFAULT);
            int taskId=getIntent().getIntExtra("id",0);
            int challengeId=getIntent().getIntExtra("challengeId",0);
            String
url=APIURL+"challenge/"+challengeId+"/task/"+taskId+"/solve";
            new SolveTask(url,encodedPhoto,this).execute();
```

```
}
```



Dacă taskul este terminat cu succes atunci fiecare utilizator care a acceptat acel challenge va primi o notificare prin care să fie anunțat ca acel task a fost terminat.

# Algoritmi utilizaţi

## ORB

În cazul in care imaginile au aceeaşi dimensiune si orientare se poate folosi un algoritm de corner detection, dar e aproape imposibil de utilizat in cadrul aplicaţiei deoarece un client nu poate şti cum sunt orientate obiectele tale in imagini şi nici dimensiunea acestora, iar presupunând ca acestea sunt descrise in documentaţie tot ar fi nişte constrangeri destul de mari deoarece ar trebui să le şi scaleze. Astfel apare un alt algoritm care poate fi folosit pentru imagini ce au scalări diferite, rotaţii diferite, perspectivă diferită si intensitatea luminii diferită denumit SIFT(scale invariant feature transform). Dezavantajul major al SIFT-ului  îl reprezintă necesitatea unei puteri de calcul foarte mare dar şi sensibilitatea la zgomote. Această sensibilitate a SIFT-ului se datorează derivatei a 2-a folosite in calculul LoG(Laplacian of Gaussian), dupa ce imaginea a fost blurata folosind formula gausiana.

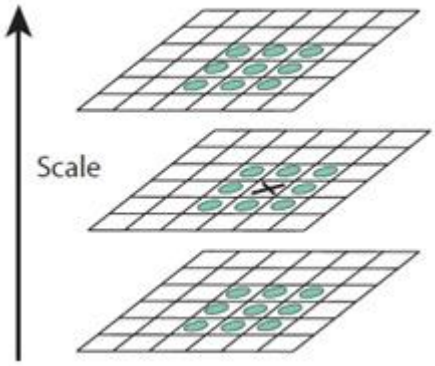
$$L(x,z,s) = (2\pi s^2)^{-1} e^{-(x\hat{}2+y\hat{}2)/2s\hat{}2} I(x,y)$$

L – imaginea blurata

I – imaginea peste care se aplica blur

Coeficientul lui I – formula gausiana aplicata peste x,y,s unde x,y coordonatele unui pixel iar s este intensitatea blurului aplicat.

Pentru a genera o aproximare a imaginilor LoG se face diferenţa(Difference of Gaussian) între imagini de blur apropiat aflate pe aceeasi octavă. O octavă se obţine injumatăţind imaginea originala din o octavă precedentă şi aplicând efectul de blur de mai multe ori(este recomandat de 5 ori iar numarul de octave recomandat este 4). Următorul pas îl constituie detectarea punctului de maximum/minimum in imaginile LoG. Astfel, se itereaza prin fiecare pixel al unei imagini se marchează pixelul curent si se calculează daca pixelul curent este mai mare decât pixelii vecini. Pixelii vecini nu sunt doar din imaginea LoG actuala dar şi din imaginea de deasupra(calculată pentru un blur mai mic) dar şi cea de sub imaginea actuală(calculată pentru un blur mai mare), exceptând cazul in care imaginea este prima sau ultima in LoG. Dacă punctul este de minimum sau maximum atunci acesta este marcat ca punct cheie.




Totuşi, în urma acestui pas sunt generaţi un numar mare de puncte cheie, multe dintre ele fiind nefolositoare(e.g pixeli cu contrast foarte scăzut), astfel se stabileşte un prag pentru care în cazul in care este punct de minimum si este sub acel prag atunci pixelul curent nu este punct cheie.

ORB foloseşte algoritmul FAST care funcţionează în modul urmator:

- se ia un pixel p din imagine care are o anumită intensitate I şi se selectează un prag t.
- pixelul p aparţine unui colţ daca există n puncte legate de punctul p dintr-un set de 16 pixeli vecini într-un cerc, astfel încât intensitatea celor n puncte este mai mică decât I+t sau mai mare decât I-t

pentru a extrage punctele cheie, iar pentru a extrage descriptorii foloseşte BRIEF (Binary Robust Independent Elementary Features). BRIEF extrage descriptorii sub forma unui bitstring astfel potrivirea imaginilor se poate face rapid folosind distanta HAMMING.

$$D_H = \sum_{i=1}^{k} \left| x_i - y_i \right|$$

BitStringul este construit pe baza unui set de teste binare pe o partiune de imagine. Un test r este definit comparand doi pixeli astfel dacă p(x) – intensitatea pixelului x – este mai mic decât p(y) atunci rezultatul testului este 1, în caz contrar rezultatul este 0 :

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases},$$

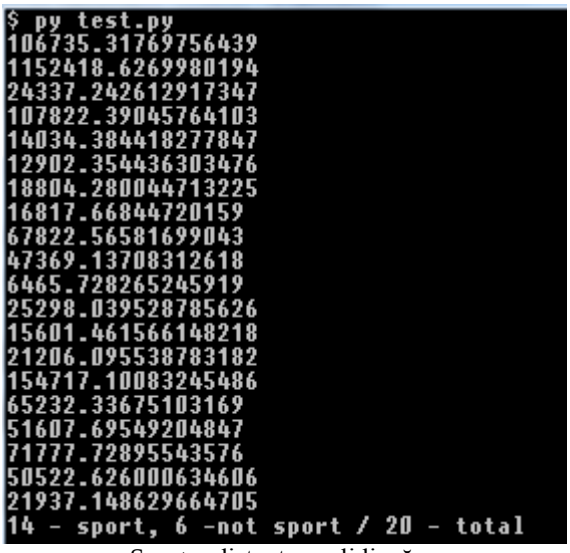
Descriptorul este scris ca un vector de n teste binare astfel:

$$f_n(\mathbf{p}) := \sum_{1 \le i \le n} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

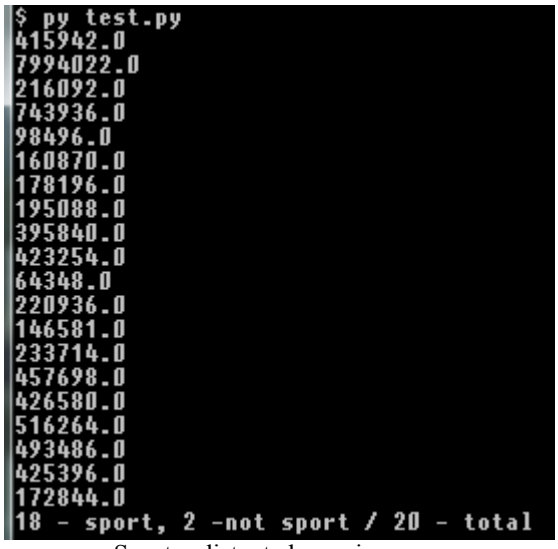
# Statistici clasificare imagini şi recunoaştere obiecte

## Clasificare

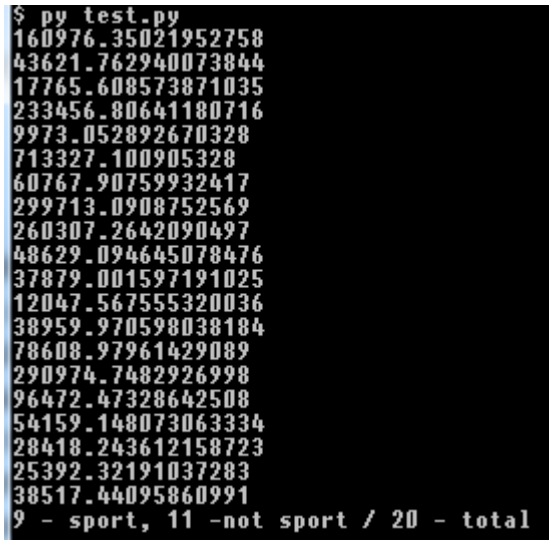
Datele de antrenament constau din 593 de imagini din sport si 694 de imagini ce nu fac parte din sport. Rulând scriptul pe un set de 20 de imagini din sport rata de clasificare corectă a fost de 70% utilizand 1NN pe histograma de culori şi folosind norma euclidiană pentru a calcula distanţa între puncte şi de 90% folosind norma hemming. Culoarea ce determină în mare măsură categoria este verde datorită faptului ca imaginile de antrenament sunt foarte mult bazate pe sporturi precum fotbalul european/american şi tenis de câmp. Pe un alt set tot de 20 de imagini dar ce nu fac parte din categoria sport procentul de apartenenţa la categoria sport a fost de 45% folosind norma euclidiana şi 55% folosind norma hemming, ceea ce era de aşteptat deoarece am încercat sa testez multe imagini care conţineau culoarea verde.



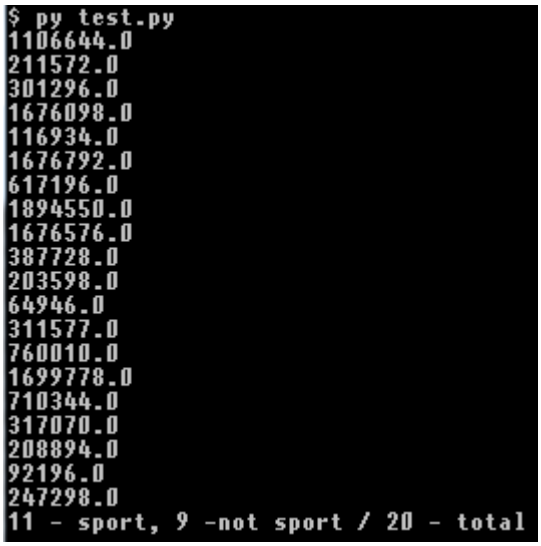
Sport – distanţa euclidiană



Sport – distanţa hemming




NotSport – distanţa euclidiană




NotSport – distanţa hemming

Dupa schimbarea numarului de intervale de la 16 la 8 se poate observa faptul ca viteza de executie a crescut considerabil deoarece vom avea un vector de 512 elemente spre deosebire de cazul în care erau 16 intervale şi 4096 elemente. Diferenţa la nivel de statistici nu este foarte mare, pentru sport şi distanţa euclidiană procentajul este acelasi, în schimb, pentru distanţa hemming procentajul scade de la 90 la 70 de procente. Pentru imaginile ce nu sunt din sport, procentajul de apartenenţa la clasa sport a scazut în cazul folosirii distanţei euclidiene de la 45 la 35 de procente iar pentru distanţa hemming de la 55 la 40.