|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** | Logo  Description automatically generated |

Иван Мршуља

**Систем за реверзну претрагу слика**

МАСТЕР РАД

- Мастер академске студије -

Нови Сад, 2023

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ МАСТЕР РАДА** | Лист: |
| 1/1 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | **Мастер академске студије** |
| --- | --- |
| Студијски програм: | **Софтверско инжењерство и информационе технологије** |
| Руководилац студијског програма: | **проф. др Мирослав Зарић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Иван Мршуља** | Број индекса: | **R2 30/2022** |
| Област: | **Електротехничко и рачунарско инжењерство** | | |
| Ментор: | **Др Драган Ивановић, редовни професор** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ МАСТЕР РАДА:**

|  |
| --- |
| **Систем за реверзну претрагу слика** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| 1. Анализирати стање у области.  2. Израдити спецификацију захтева софтверског решења.  3. Израдити спецификацију дизајна софтверског решења.  4. Имплементирати софтверско решење према израђеној спецификацији.  5. Тестирати имплементирано софтверско решење.  6. Документовати (1), (2), (3), (4) и (5). |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|  |  |
| --- | --- |
| Редни број, **РБР**: |  |
| Идентификациони број, **ИБР**: |  |
| Тип документације, **ТД**: | монографска публикација |
| Тип записа, **ТЗ**: | текстуални штампани документ |
| Врста рада, **ВР**: | мастер рад |
| Аутор, **АУ**: | Иван Мршуља |
| Ментор, **МН**: | др Драган Ивановић, редовни професор |
| Наслов рада, **НР**: | Систем за реверзну претрагу слика |
| Језик публикације, **ЈП**: | српски |
| Језик извода, **ЈИ**: | српски / енглески |
| Земља публиковања, **ЗП**: | Србија |
| Уже географско подручје, **УГП**: | Војводина |
| Година, **ГО**: | 2023 |
| Издавач, **ИЗ**: | ауторски репринт |
| Место и адреса, **МА**: | Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6 |
| Физички опис рада, **ФО**: | бр. поглавља / страница / цитата / табела / слика / графикона / прилога |
| Научна област, **НО**: | Софтверско инжењерство и информационе технологије |
| Научна дисциплина, **НД**: | Софтверско инжењерство |
| Предметна одредница /  кључне речи, **ПО**: | 3-5 кључних речи које бисте користили у претраживачу да нађете рад са овом темом |
| **УДК** |  |
| Чува се, **ЧУ**: | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад |
| Важна напомена, **ВН**: |  |
| Извод, **ИЗ**: | апстракт – један пасус који добро описује суштину рада – проблем, мотивацију, назнаку решења и резултат. |
| Датум прихватања теме, **ДП**: |  |
| Датум одбране, **ДО**: |  |
| Чланови комисије, **КО**: |  |
| председник | др Име Презиме, звање |
| члан | др Име Презиме, звање |
| ментор | др Име Презиме, звање |
| Потпис ментора | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: | monographic publication |
| Type of record, **TR**: | textual material |
| Contents code, **CC**: | master thesis |
| Author, **AU**: | Ivan Mršulja |
| Mentor, **MN**: | Dragan Ivanović, full professor, PhD |
| Title, **TI**: | Reverse Image Search System |
| Language of text, **LT**: | Serbian |
| Language of abstract, **LA**: | Serbian / English |
| Country of publication, **CP**: | Serbia |
| Locality of publication, **LP**: | Vojvodina |
| Publication year, **PY**: | 2023 |
| Publisher, **PB**: | author’s reprint |
| Publication place, **PP**: | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 |
| Physical description, **PD**: | br. poglavlja / stranica / citata / tabela / slika / grafikona / priloga |
| Scientific field, **SF**: | Software Engineering and Information Technologies |
| Scientific discipline, **SD**: | Software Engineering |
| Subject / Keywords, **S/KW**: | Ključne reči na engleskom |
| **UDC** |  |
| Holding data, **HD**: | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad |
| Note, **N**: |  |
| Abstract, **AB**: | Prevod apstrakta na engleski |
| Accepted by sci. Board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president | Ime i prezime, zvanje na eng., PhD |
| member | Ime i prezime, zvanje na eng., PhD |
| mentor | Ime i prezime, zvanje na eng., PhD |
| Mentor's signature | |

**САДРЖАЈ**

[1. UVOD 7](#_Toc94086543)

[2. PREGLED SLIČNIH SISTEMA 9](#_Toc94086544)

[3. KORIŠĆENE SOFTVERSKE TEHNOLOGIJE 9](#_Toc94086545)

[3.1 Spring 10](#_Toc94086546)

[3.2 Opis tehnologije na koji se vaš rad oslanja 10](#_Toc94086547)

[4. SPECIFIKACIJA 11](#_Toc94086548)

[4.1 Specifikacija zahteva 11](#_Toc94086549)

[4.1.1 Funkcionalni zahtevi 11](#_Toc94086550)

[4.1.2 Nefunkcionalni zahtevi 11](#_Toc94086551)

[4.2 Specifikacija sistema 13](#_Toc94086552)

[4.2.1 Model podataka 13](#_Toc94086553)

[4.2.2 Arhitektura sistema 13](#_Toc94086554)

[5. IMPLEMENTACIJA 15](#_Toc94086555)

[6. DEMONSTRACIJA 17](#_Toc94086556)

[7. ZAKLJUČAK 19](#_Toc94086557)

[8. LITERATURA 21](#_Toc94086558)

[9. BIOGRAFIJA 23](#_Toc94086559)

[KLJUČNA DOKUMENTACIJSKA INFORMACIJA 25](#_Toc94086560)

[KEY WORDS DOCUMENTATION 27](#_Toc94086561)

# УВОД

Реверзна претрага слика (*reverse image search*) је техника претраге која омогућава корисницима да пронађу сличне или идентичне слике на интернету користећи већ постојећу слику као улазни параметар. Умјесто да корисници уносе текстуалне упите, могу отпремити слику или унети *URL* слике како би пронашли друге слике са сличним визуелним карактеристикама. Алгоритми за реверзну претрагу слика анализирају кључне елементе слике, као што су боје, облици, текстуре или други визуелни детаљи, како би пронашли слике са сличним карактеристикама из база података или веб страница широм интернета. Ова техника је корисна у многим ситуацијама, као што су проналажење извора слике, идентификација објеката, проналажење сличних визуелних садржаја, откривање плагијата или проналажење више информација о одређеном објекту на слици. Реверзна претрага слика може бити од помоћи у истраживачким, креативним и безбедносним контекстима, пружајући корисницима могућност да пронађу релевантне слике на основу већ постојеће визуелне референце.

У овом раду, рјешаван је проблем имплементације једног оваквог система, гдје корисник може унијети произвољну слику као узорачки упит а систем му за исти враћа слике које најбоље одговарају на њега, поред тога, омогућено је и индексирање слика. Такође, адресиран је и проблем великог броја корисника који би потенцијално користили овај систем, те се развој водио праксама сервисно-орјентисане архитектуре гдје је сваки сервис могуће независно скалирати у произвољном обиму.

Ово софтверско рјешење се састоји из два засебна сервиса:

* сервиса за претрагу и индексирање (имплементираног у *Java Spring Boot* радном оквиру)
* сервиса за процесирање слике (имплементираног у *Python* *FastAPI* радном оквиру уз ослонац на *ОpenCV* и *NumPy* библиотеке за имплементацију и интеграцију неопходних алгоритама машинског учења који се користе приликом процесирања слике)

Поред ова два сервиса битно је напоменути и да постоји одвојени frontend сервис имплементиран у *Vue3* радном оквиру користећи *JavaScript* програмски језик. Сервис за претрагу и индексирање комуницира са сервисом за процесирање слике како би извукао својства (*features*) са слике која је задата као узорак за упит. Поред функције *information retrieval*-а и индексирања овај сервис врши и кеширање својстава слике користећи *Redis* *key-value* базу података како би убрзао вријеме претраге за више сукцесивних упита са истом сликом што представља веома битан фактор у раду овог система како би био погодан за коришћење. Индексирање као и претрага је имплементирана користећи *Elasticsearch* базу података.

Јединственост овог рјешења лежи у формату својстава која се користе приликом индексирања и претраге. Већина данашњих система се ослањају на векторске репрезентације документа како би имплементирати претрагу. Иако је ово веома ефикасан начин претраге визуелно сличних слика, велики је проблем што се експлицитно у обзир не узимају сви објекти са слике већ се читава слика репрезентује као неки *black-box* скуп својстава. Са друге стране, моје рјешење своди претрагу слика на претрагу текста, моделима машинског учења се препознају објекти, који се посматрају као кључне ријечи, помоћу којих можемо препознати све контекстуално сличне слике са узорком док се простор боја користи као опциони филтер како бисмо могли добити визуелно најсличније слике.

У поглављу 2 биће приказан преглед сличних система, са најутицајнијим рјешењима у овом проблемском домену. У поглављу 3 биће описани теоријски појмови и дефиниције неопходне за разумијевање овог рада као и коришћене софтверске технологије. Поглавље 4 посвећено је опису спецификације самог пројекта. Наредно поглавље, 5, резервисано је за опис имплементације пројекта и дубљи залазак у поједине техничке детаље. У поглављу 6, представљена је кратка демонстрација рада овог система док је у поглављу 7 представљена поставка експеримента којим су се покушале измјерити перформансе овог система. Прије самог краја, у поглављу 8 износе се и дискутују резултати овог експеримента. На крају, у поглављу 9, даје се закључак на овај рад.

# ПРЕГЛЕД СЛИЧНИХ СИСТЕМА И СТАЊА У ОБЛАСТИ

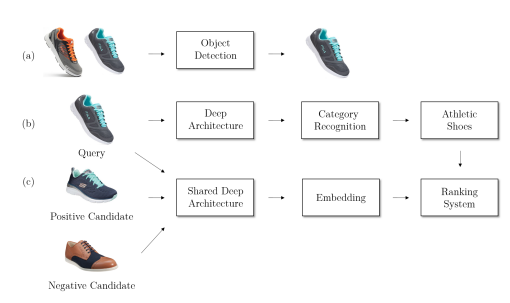
У овом поглабљу биће представљено пар актуелних система који се баве реверзном претрагом слика (*Web-Scale Responsive Visual Sear*ch и *Similar* *Looks*). Битно је напоменути да је Pinterest-ов Similar Looks био примарна инспирација приликом имплементације овог система. Критеријум за одабир релевантних система је понајвише њихова употребљивост (да ли се успјешно користе у продукцији) али и количина корисника која мора бити опслужена као и скалабилност самог система. Такође, с обѕиром да је доста комерцијалних система за претрагу слика затвореног кода, доступност информација је играла битну улогу. Како је процесирање у мојем систему само подскуп свих типова процесирања која се користе у великим системима, у пракси нема смисла поредити сам квалитет *information retrieval*-a већ су ови системи само помогли како би се добила иницијална идеја шта све систем треба да подржи и како би се најбоље могла осмислити његова архитектура.

## *Web-Scale Responsive Visual Search* (*Bing*)

Ток рада визуалне претраге је следећи: корисник upload-ује слику или је снима камером, а резултат ће бити сличне слике и предмети; корисник може изабрати да ли жели да купи (слично „*Shopping*“ секцији код *Google* претраживача) или даље истражује предмете. Овај систем обухвата три главне фазе:

1. Разумевање упита (слике): Из упитне слике извлаче се различите карактеристике које описују њен садржај, укључујући карактеристике препознавања категорије, препознавања лица, боје и откривања дупликата, енкодере дубоких неронских мрежа (*DNN*) и детекцију објеката (слика 2.1.1).
2. Претрага слика: Проналази визуално сличне слике на основу извучених карактеристика и намера корисника.
3. Обука модела: У систему се користи неколико модела дубоких неронских мрежа како би се побољшала релевантност резултата, као што су *AlexNet*, *ZFSPPNet*, *GoogleNet*, *ResNet*, а такође се користи и заједнички алгоритам *кmeans* како би се изградио инверзни индекс у нивоу-0 подударања, при чему ниво 0 означава најмању сличност са упитном сликом.

Сви модели се тренирају на обучавајућем скупу података који су прикупљени за одређене домене нпр. куповина. За надзор обуке *DNN* користе се више функција губитка, као што су *Softmax*, *Pairwise-loss* и *Triplet-loss* функције.



Слика 2.1.1 Примене *DNN* modela koji se koriste u *Bing*-овом систему

## *Similar Looks* (*Pinterest*)

Ова функционалност унутар *Pinterest* апликације омогућава корисницима да изаберу предмет унутар слике, стављајући оквир око њега, а затим враћа визуелно сличне слике/предмете. Главни циљ ове функције је помоћи корисницима да пронађу ствари које не могу именовати.

Индексирање слика се у овом систему врши по *incremental fingerprinting service* (*IFS*) систему. Овај систем функционише по принципу да свака слика има свој отисак који се састоји од свих одвојених објеката који се налазе унутар те слике (нпр. чипеле, торбе, чаше итд.). Сваки објекат има ознаку која специфицира име класе којој објекат припада.

Приликом претраге, систем користи технологију препознавања објеката да локализује и класификује предмете (објекте) на сликама. Из овако детектованих објеката се извлаче карактеристике и користе се за даљу претрагу. Другим ријечима, скраћује се задатак детекције објеката више класа у класификацију категорија, тако да уместо да се тражи подударање у свим сликама на *Pinterest*-u, прво се добављају слике које су у истој категорији. Ово је одличан приступ јер се упитна слика упоређује са сликама које имају високу вероватноћу да су сличне. Претходни корак филтрирања повећава стопу тачних позитива. Систем се заснива на концептима дубоког учења, као што су конволутивне неронске мреже (*CNN*) и детектори објеката засновани на дубоком учењу (*Single Shot Detectors, SSD*). Систем је имплементиран коришћењем радног оквира *Caffe* (*Java* технологија). Подаци система се скаладиште на *Аmazon S3*-у.

# КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ, ТЕОРИЈСКИ ПОЈМОВИ И ДЕФИНИЦИЈЕ

У овом поглављу биће описане софтверске технологије које су коришћене приликом имплементације система. Такође, осврнућу се на основне теоријске појмове и дефиниције одређених *AI* концепата како би се лакше могло разумјети функционисање као и сама имплементација сервиса за процесирање слика која ће бити представљена у неком од наредних поглавља. У одељку 3.1, 3.2 и 3.3 описани су радни оквири у којима је одрађена имплементација система, конкретно: *Spring* *Boot*, *FastAPI* и *Vue.js 3* респективно. У одељцима 3.4, 3.5, 3.6 i 3.7 представљене су помоћне технологије и алати који су помогли приликом рјешавања специфичних проблема приликом израде овог софтверског рјешења и његовог деплојмента а то су редом: *Redis*, *Elasticsearch*, *Docker* и *Docker-Compose*. На крају, у одељку 3.8 дат је кратак увод о конволуционим неуронским мрежама како би се боље разумјела архитектура *YOLOv5* модела, описана у одељку 3.9, који представља срж овог система када говоримо о проблему обраде и извлачењу својстава слике.

## *Spring* *Boot*

*Spring Boot* је моћна технологија отвореног кода која нуди једноставан и брз начин за развој *Java* *EE* апликација. Он омогућава програмерима да брзо подигну окружење и фокусирају се на развој функционалности без потребе за комплексном конфигурацијом коју традиционални *Spring* *Framework* изискује. *Spring* *Boot* интегрише *Spring* *Framework* „испод хаубе“ и доноси конвенције о конфигурацији и добрим праксама које значајно убрзавају процес развоја. Омогућава једноставно управљање зависностима, аутоматско конфигурисање и једноставно подизање микросервисних апликација. *Spring* *Boot* пружа многобројне модуле и уграђен *Tomcat* сервер за покретање апликација.

## *FastAPI*

*FastAPI* је модеран *Python* радни оквир за развој веб апликација и *API*-ја. Он се издваја по својој брзини и ефикасности, што га чини идеалним избором за развој напредних и високо перформантних апликација. Са друге стране, *Python* програмски језик је најкоришћенији језик у домену машинског учења те се овај радни оквир користи и за развој сервиса који пружају анализу ресурса коришћењем алгоритама машинског учења. *FastAPI* користи типизацију на основу анотација, што омогућава аутоматску проверу типова и генерисање документације. Поред тога, *FastAPI* подржава асинхрони развој и укључује уграђену подршку за покретање на уграђеним серверима као што су *Uvicorn* и *Hypercorn*. Разлика између *Uvicorna* и *Hypercorna* је у томе како се носи са асинхроним захтевима и руковањем конкретним *HTTP* конекцијама. Uvicorn је *WSGI* сервер базиран на *uvloop* библиотеци и користи асинхрони рад заснован на евентима (*event loop*) за обраду више захтева паралелно. Он је изузетно брз и ефикасан за обраду већег броја захтева истовремено. Са друге стране, *Hypercorn* је такође асинхрони *WSGI* сервер, али користи библиотеку *h2o* која је базирана на *h11* библиотеци за обраду *HTTP*/*1.1* и *HTTP/2* протокола. *Hypercorn* је дизајниран да буде изузетно брз и ефикасан у руковању асинхроним захтевима, посебно када се користи *HTTP/2* протокол.

## *Vue3*

*Vue.js* *3* је популаран отворени *JavaScript* радни оквир за изградњу корисничких интерфејса. Он представља еволуцију претходне верзије *Vue.js* и долази са бројним унапређењима и новим функционалностима. *Vue.js* 3 је дизајниран да буде још ефикаснији, скалабилнији и једноставнији за коришћење. Једна од кључних новина у *Vue.js 3* је *Composition API*, који омогућава писање компоненти на начин који је флексибилнији и реактивнији. Ово олакшава организацију кода, дељење логике између компоненти и управљање стањем апликације. Такође, *Vue.js 3* доноси и изузетно брз виртуелни *DOM* (*virtual DOM*) који омогућава ефикасно рендеровање компоненти и ажурирање само неопходних делова интерфејса. Ово доприноси перформансама апликације и побољшава корисничко искуство. *Vue.js 3* такође пружа бољу *TypeScript* подршку, па програмери могу лакше откривати грешке и имати бољу статичку анализу кода (битно је напоменути да у овом пројекту *TypeScript* није коришћен из разлога што је пројекат једноставан са стране корисничког интерфејса и увођење *TS*-а би непотребно компликовало технолошки стек).

## *Redis*

*Redis* је брза и флексибилна *key*-*value* база података у меморији (*in*-*memory*) која се користи за кеширање, брзи приступ подацима и рјешавање проблема складиштења података (у мањој мјери). Ова база података је дизајнирана да би била изузетно ефикасна у упитима и складиштењу података у меморији, што је чини идеалним избором за апликације са високим оптерећењем и захтевима за брзим одзивом. *Redis* пружа широк спектар функционалности, укључујући подршку за различите типове података као што су стрингови, листе, хеш мапе, торке и сортиране скупове. Такође, *Redis* има богат скуп операција које омогућавају манипулацију и приступ подацима на ефикасан начин као и богат екосистем у виду додатних библиотека и алатки које проширују његове могућности. То укључује могућности кеширања, редова (*queue*), претплате на поруке (*pub*/*sub* *communication*) и временско истицање података (*expiry*). Све у свему, *Redis* база је моћан алат за упите и складиштење података у меморији, са брзим одзивом и многобројним функционалностима које чине многе апликације бржим, скалабилнијим и робуснијим.

## *Elasticsearch*

*Elasticsearch* је високо скалирајући, дистрибуирани систем за претрагу и анализу података. Он је осмишљен да брзо и ефикасно индексира, чува и претражује велике количине структурираних и неструктурираних података. *Elasticsearch* је базиран на *Apache* *Lucene* библиотеци, која пружа моћне алгоритме за претрагу. Он такође подржава бројне напредне функционалности као што су претрага пуним текстом (*full*-*text* search), филтрирање, агрегација, геолокација, анализа текста, векторско индексирање и претрага и још много тоga. Осим претраге, *Elasticsearch* такође омогућава анализу података и визуелизацију резултата. Кроз *Kibana* интерфејс, корисници могу приступити богатим визуализацијама, извештајима и надзору над подацима. *Elasticsearch* је дизајниран да буде отпоран на кварове и пружа могућност репликације података и дистрибуције преко више чворова. Такође подржава хоризонтално скалирање, омогућавајући додавање нових чворова како би се постигла боља брзина и отпорност на висока оптерећења. Са својом флексибилношћу, скалирањем и моћним алатима за претрагу и анализу, *Elasticsearch* је постао популаран избор за разне примене као што су претрага веб страница, анализа логова и системи за праћење догађаја (*event* *tracking* *systems*).

## *Docker*

*Docker* је популарна *open*-*source* платформа која омогућава контејнеризацију апликација. Он омогућава лако покретање, паковање и дистрибуцију апликација у изолованом окружењу, познатом као контејнер. *Docker* користи контејнере за смањивање зависности и осигуравање конзистентности окружења између различитих система.

## *Docker-Compose*

*Docker-Compose* је алат која се користи за дефинисање и покретање мулти-контејнерског окружења користећи *Docker*. Омогућава дефинисање сервиса, мрежа и контејнера у конфигурационом *YAML* фајлу. Са *Docker-Compose*-ом, могуће је покренути и управљати више контејнера, мрежа и *volume*-а као са једном апликацијом. Ово олакшава развој, тестирање и деплојмент комплексних апликација са више компоненти. Укратко, *Docker* омогућава контејнеризацију апликација, док *Docker*-*Compose* омогућава дефинисање и покретање мулти-контејнерских окружења. Заједно, ове две технологије пружају моћан начин за паковање, покретање и управљање апликацијама у изолованом окружењу.

## Kонволутивна неуронска мрежа

У сфери дубоког учења, конволуционе неуронске мреже су класа дубоких неуронских мрежа (*Deep Neural Networks*, DNN) које се најчешће користе за анализу слика. Главни разлог за ово је чињеница да се коришћењем CNN умногоме смањује број параметара у односу на класичне потпуно повезане дубоке неуронске мреже. Узмимо, на примјер, слику димензија 1000 х 1000 х 3. Код обичне потпуно повезане DNN имали бисмо 3 милиона параметара на улазу. Чак иако би први следећи слој имао само 1000 неурона, димензије тежинске матрице W[1] би биле 1000 x 3 милиона. То би значило да је скоро немогуће наћи скуп података који би био довољно велик да не дође до преприлагођавања на тренинг скуп [11]. За исту димензију улазних података са 3 х 3 филтером, CNN би произвела свега 9 параметара. У наставку ће бити објашњени главни градивни блокови CNN-а.



Слика 3.8.1 Примјер архитектуре CNN

Главни градивни блокови који чине CNN (Слика 3.2.1) су:

* Конволуциони слој
* Операција испуњавања (*padding*)
* Активациона функција
* Слој сажимања
* Потпуно повезани слој
* Функција губитка (*loss function*).

Конволуциони слој се заснива на математичкој операцији конволуције. За улазну матрицу димензија N x N и филтер f x f спроводи се итеративна операција тако што се филтер поставља на почетак слике, те се сваки елемент унутар матрице који се налази „испод“ филтера множи са својим кореспондентним елементом унутар филтера. Добијени производи се тада сабирају и ово представља један елемент резултујуће матрице (*feature map*). Након тога се филтер помијера за одређени корак (*stride*), те се операција понавља све док се не дође до краја слике. Код ове операције, важно је примијетити да су димензије резултујуће матрице мање од димензија полазне матрице. Ово смањење у димензији се може изразити следећом законитошћу: за улазну матрицу димензија N x N и филтер f x f са кораком S добијамо матрицу димензија:

Ово може представљати проблем, поготово код слојева гдје је S велико, јер ће већ након неколико слојева полазна матрица бити превише мала и неупотребљива. Такође, пиксели (елементи матрице) који се налазе на крајевима ће доста мање бити узимани у обзир у односу на елементе који се налазе у средини, јер филтер мање пута прелази преко њих [12]. Као рјешење, уз операцију конволуције најчешће се примјењује и операција испуњавања (*padding*). Идеја је да се матрица прошири ивицом за онолико редова и колона колико је то потребно да резултујућа матрица буде истих димензија као почетна. Елемент који се убацује је 0, одабран тако да се не наруши валидност операције конволуције [11]. Овим се претходно дефинисана законитост мијења, те ће се за *padding* ширине P димензије резултујуће матрице износити:

Након извршене операције конволуције, на *feature* мапу се, као и у класичним DNN, додаје вектор пристрасности (*bias vector*) и примјењује се активациона функција. Популарне активационе функције су *ReLu* (*Rectified Linear Unit*), *sigmoid* и *TanH* [12].

Поред конволуционих слојева, CNN често користе и сажимајуће (*pooling*) слојеве како би смањили димензије репрезентације, убрзали рачунање и повећали робусност приликом детекције одређених особина [13]. *Pooling* слој зависи од pooling операције. Као и код конволуционог слоја, филтер димензија N x N се превлачи преко слике са кораком f, само се умјесто операције конволуције примјењује *pooling* операција. Коју *pooling* операцију ћемо изабрати зависи од архитектуре нашег модела. Данас, је најпопуларнији *max pooling* код кога као излазну вриједност узимамо највећи елемент из региона захваћеног филтером (Слика 3.8.2). Могућа интуиција иза овога је да желимо да само оне особине које су веома изражене уђу у даље разматрање, док се оне слабије изражене занемарују. Други популаран избор *pooling* операцији је *average pooling* који се историјски доста више користио, али је изгубио на популарности јер се *max pooling* показао као бржи и подједнако ефикасан у експериментима [14].



Слика 3.8.2 Разлика између различитих *pooling* алгоритама

Потпуно повезани (*fully connected*) слој представља идентичан слој као у класичној DNN. Ови слојеви се налазе на самом крају CNN. Прелаз између тродимензионалног тензора и једнодимензионалног потпуно повезаног слоја се у литератури назива и „*flattening*“. На самом крају СNN-а налази се *loss layer* који специфицира како тренинг „кажњава“ одступање између предвиђених (излазних) и реалних (лабелираних) ознака у тренинг скупу. Код проблема класификације типично се користи *Softmax loss* док се код предвиђања вјероватноће неког исхода најчешће користи *Sigmoid cross-entropy loss* [11]. На слици 3.8.1 приказан је примјер архитектуре CNN са свим горе поменутим градивним блоковима.

## *YOLOv5*

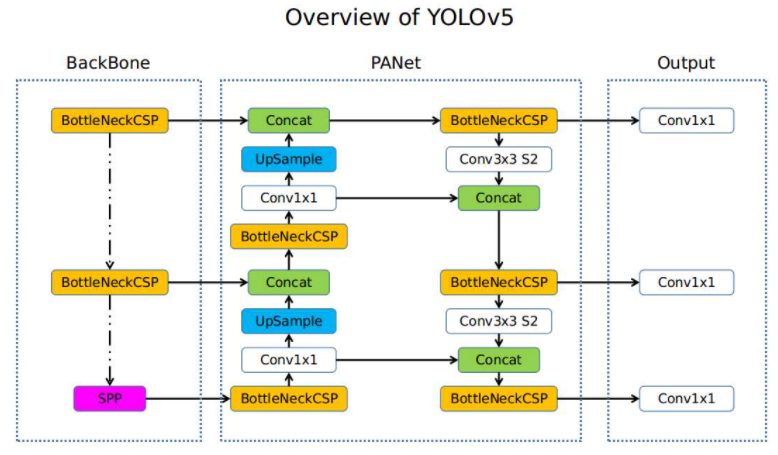
*YOLOv5* модел за детекцију објеката базиран је на *end-to-end* (*e2e*) архитектури и састоји се од три главне компоненте:

1. власничке мреже (*backbone* *network*)
2. главне мреже (*neck* *network*)
3. мреже излаза (*head* *network*)

Власничка мрежа (*backbone*) у *YOLOv5* моделу је *CSPDarknet53* мрежа, која се базира на конволуционим слојевима. Ова мрежа је модификована верзија оригиналне *Darknet53* мреже и пружа бољу репрезентацију својстава и разумевање контекста објеката. Она је одговорна за идентификацију и издвајање битних својстава из улазних слика.

Главна мрежа (*neck* *network*) је додатни слој између власничке мреже и мреже излаза. Онa има за циљ да усаврши својства из власничке мреже и подеси их тако да би лакше детектоvala објекте различитих величина и пропорција. У спецификацији *YOLOv5* модела, користи се *PANet* мрежа која је коришћена и у старијем *YOLOv4* моделу.

Мрежа излаза (*output* *network*) представља последњу компоненту модела и садржи конволуционе слојеве који изводее коначне предикције о класама и локацијама објеката. Онa преобрађује излазне *featur*-е из главне мреже у *anchor*-*box*-ove и враћа координате, сигурност и класе детектованих објеката за сваки bounding-box. У појединој литератури, овај сегмент у *pipeline*-u модела назива се и *head* *network*. На слици 3.9.1 приказана је горе описана архитектура.



Слика 3.9.1 Архитектура YOLOv5 модела

Након извршавања *forward* *pass*-a, на излаз модела се примењује неопходно постпроцесирање. Ово укључује примену non-maximum suppression алгоритма како би се уклонили преклапајући *bounding*-*box*-ови и задржали само они који задовољавају претходно дефинисан праг сигурности. Укратко гледано, *YOLOv5* архитектура користи напредне конволуционе мреже и оптимизације у виду конволутивних прескакања (*convolution skips*), *upscale* операција и сличних метода како би брзо и ефикасно детектовала објекте на сликама и у видео снимцима. Ова архитектура је постала популаран избор за брзу и прецизну детекцију објеката у различитим областима примјене.

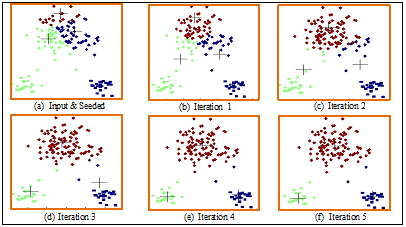
## *Kmeans*

*Кmeans* алгоритам, је алгоритам за кластеровање података. Основна идеја овог алгоритма је да групише податке у *K* кластера на основу њихових сличности. Процес почиње са случајним избором *K* тачака као почетних тачака кластера, које се називају и центроиди. Затим се улазни подаци додељују ближем центроиду на основу еуклидске удаљености или неке друге метрике сличности.

Након тога, пресмештањем центроида и доделом података новим кластерима, алгоритам итеративно конвергира ка конфигурацији у којој се минимизује укупна варијанса унутар кластера. Овај процес се наставља све док се центроиди више не промењују значајно или док се не достигне максималан број итерација.

На крају, *kmeans* алгоритам даје нам кластере који групишу податке на основу њихових сличности. Он може бити корисан у различитим областима, као што су анализа података, компјутерска визија, препознавање образаца и многим другим.

Важно је напоменути да *kmeans* алгоритам зависи од броја кластера *К*, који мора бити унапред дефинисан. Избор погодног броја кластера може бити предмет истраживања и експериментисања како би се добили најбољи резултати. На слици 3.10.1 је приказана илустрација рада овог алгоритма.



Слика 3.10.1 Илустрација рада *Kmeans* алгоритма

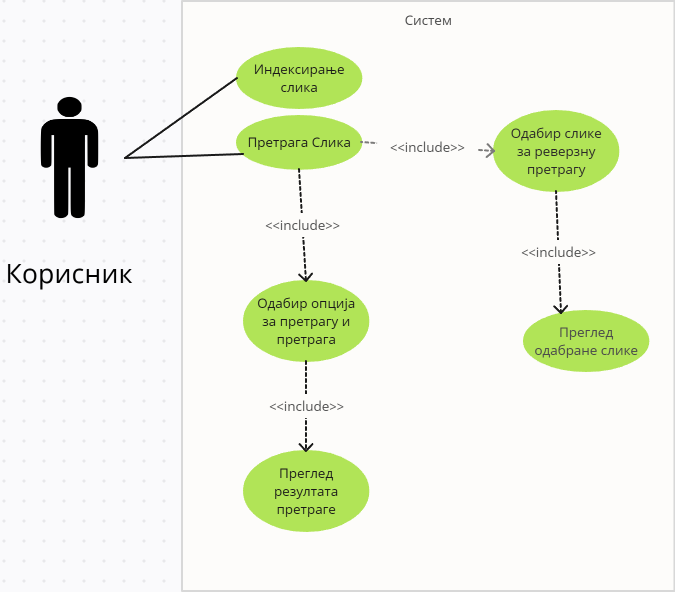
# СПЕЦИФИКАЦИЈА

## Спецификација захтјева

У овом поглављу биће представљен опис функционалних и нефункционалних захтјева које систем омогућава у одељцима 4.1.1 и 4.1.2 респективно. Одељак 4.1.3 садржи опис модела података, са класним дијаграмима за ентитете који служе за индексирање али и онима који служе за кеширање као и за трансфер података између сервиса (*Data Transfer Objects* - *DTO*). На крају, у одељку 4.1.4 биће представљена архитектура система.

### Функционални захтјеви

Функционалне захтјеве најбоље је посматрати са *UML* (*Unified Modeling Language*) дијаграма случајева коришћења (*Use Case Diagram*) који је дат на прилогу 4.1.1.1



Прилог 4.1.1.1 Дијаграм случајева коришћења система

Посебан опис сваког случаја коришћења дат је у наставку:

* Индексирање слика
  + Предуслови: Корисник је позициониран на страницу за индексирање.
  + Кораци: корисник кликом на index bar отвара засебан *upload dialog*. Селектовањем једне или више жељених слика и кликом на дугме select завршава процес одабира фајлова за индексирање и има преглед имена фајлова у *index bar*-u. Кликом на дугме за индексирање започиње се процес индексирања након чега се кориснику испусује да ли је индексирање прошло како треба. У току чекања на завршетак процеса индексирања, кориснику се приказује *loader* који га забавља и приказује му се да се нешто дешава у позадини и да апликација није запуцала.
  + Резултат ове операције је индексирање свих селектованих слика у сеарцх-енгине и могуће их је пронаћи приликом реверзне претраге.
  + Једини могући изузетак је да индексирање није прошло, у том случају слике неће бити индексиране а корисник ће бити обавијештен да је дошло до грешке приликом индексирања.
* Одабир слике за реверзну претрагу
  + Предуслови: Корисник је позициониран на страницу за претрагу.
  + Кораци: корисник кликом на *search bar* отвара засебан *upload dialog*. Селектовањем једне жељенe сликe и кликом на дугме *select* завршава процес одабира слике која служи као узорак за упит и има преглед имена фајла у *search bar*-u.
  + Резултат ове операције је селектовање слике која ће у наредним корацима бити коришћена за реверзну претрагу слика.
  + Уколико корисник изабере слику која није у .jpg или .*png* формату, ништа неће бити селектовано и корисник ће морати да понови операцију како би селектовао валидан формат фајла уколико жели да изврши претрагу.
* Преглед одабране слике
  + Предуслови: корисник је селектовао слику за реверзну претрагу.
  + Кораци: Након селектовања жељене слике, кориснику се поред имена фајла, на екрану приказује селектована слика како би био сигуран да је одабрао праву слику.
  + Резултат ове операције је приказана слика на екрану.
  + Не постоји изузетак који се може десити у овом кораку.
* Одабир опција за претрагу и претрага
  + Предуслови: корисник је селектовао слику за реверзну претрагу.
  + Кораци: Након селектовања жељене слике, корисник може селектовати опције приликом претраге (у случају овог система постоји само „*sort by color space*“) и кликом на дугме за претрагу започети процес претраге. Кориснику се приказује *loader* који нестаје чим се резултати претраге добију са сервера.
  + Резултат ове операције јесте извршена претрага слика као и примјењивање селектованих опција претраге.
  + Уколико дође до било каквих грешака приликом претраге, корисник ће о томе бити обавијештен и моћи ће да покуша поново кликом на дугме за претрагу.
* Преглед резултата претраге
  + Предуслови: Корисник је извршио операцију претраге.
  + Кораци: Након извршене претраге, кориснику се на екрану појављују (у зависности од селектованих опција) сортирани резултати претраге. Кликом на pagination bar могуће је истраживати пагиниране резултате уколико број враћених резултата не може стати на једну страницу.
  + Резултат ове операције јесте приказивање релевантних резултата претраге кориснику.
  + Уколико се приликом промјене странице деси нека грешка, корисник ће бити о томе обавијештен и мораће да понови претрагу.

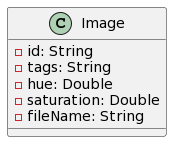
### Nefunkcionalni zahtevi

Систем за реверзну претрагу слика би требао да омогући:

* Прихватљиву брзину индексирања датотека (< 5 секунди по датотеци)
* Прихватљиву брзину претраге (<= 2 секунде да се прикаже прва страница са резултатима)
* Минималистичан и интуитиван кориснички интерфејс како се кориснику не би одвлачила пажња са приказаних резултата
* Responsive дизајн на *frontend*-u како би корисник могао да приступи апликацији са различитих уређаја
* Систем би требао да буде у великој мери отпоран на грешке јер ће крајњи корисници најчешће бити не-техничка лица и желимо да им обезбиједимо што бољи *user-experience*
* Доступност система би такође требала бити на високом нивоу с обзиром да код оваквих система, исти губе на популарности јако брзо уколико је сервис дуже времена недоступан
* Вишејезичност није нопходна, бар у првој итерацији система с обзиром да се систем не ослања на текстуалне описе да би кориснику објаснио шта да ради, већ је циљ да се интуитивним дизајном кориснику омогући да већ унапред „зна“ како се користи систем

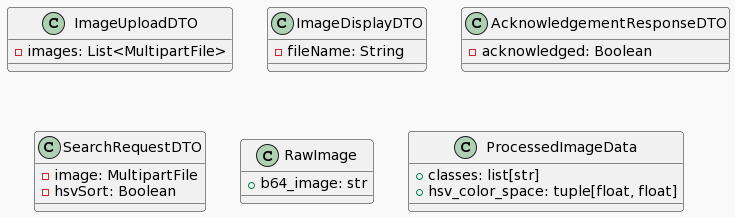
### Model podataka

* *Index* модел података представљен је на дијаграму 4.1.3.1.



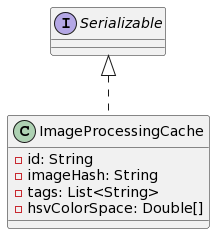
Слика 4.1.3.1 Класни дијаграм модела података за индексирање

* *DTO* модел података представњен је на дијаграму 4.1.3.2.



Слика 4.1.3.2 Класни дијаграм модела података за трансфер података између сервиса

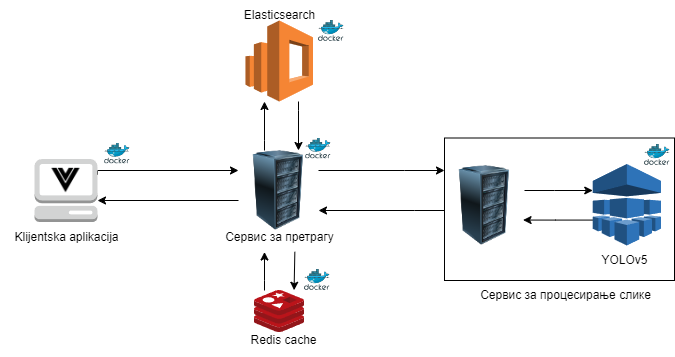
* *Cache* модел података представљен је на дијаграму 4.1.3.3.



Слика 4.1.3.3 Класни дијаграм модела података за кеширање

### Arhitektura sistema

Архитектура цијелог система приказана је дијаграмом размештаја (*deployment diagram*) и дата је у прилогу 4.1.4.1.



Прилог 4.1.4.1 Архитектура система

# ИМПЛЕМЕНТАЦИЈА

У овом поглављу представљена је имплементација система за реверзну претрагу слика. Имплемнтација је подијељена по сервисима гдје сваки одељак описује имплементацију једног сервиса. У одељку 5.1 биће представљен сервис за процесирање слика, док ће у одељцима 5.2 и 5.3 бити представљени сервис за претрагу као и *frontend* апликације респективно.

## Сервис за процесирање слика

Сервис за процесирање слика користи алгоритме и моделе машинског учења (конкретно *kmeans* и *Yolov5* како би извукао својства са задате слике). За имплементацију овог сервиса коришћен је *FastAPI* радни оквир (*uvicorn* сервер) као и библиотеке *NumPy* и *OpenCV* за интегрисање претренираног *YOLOv5* модела и имплементацију неопходних алгоритама за читање резултата из излазног слоја модела. Претренирани *YOLOv5* модел је интегрисан на начин да су са официјалног репозиторијума преузети *.onxx* фајлови за *YOLOv5n* и *YOLOv5s* моделе. *ONNX* (*Open Neural Network Exchange*) је формат датотеке који омогућава пренос и размену модела за машинско учење између различитих окружења и библиотека који између осталог подржава интеграцију са *DNN* модулом *OpenCV* библиотеке. На листингу 1 приказана је интеграција *YOLOv5s* модела у систем.

def build\_model():

global IS\_CUDA

net = cv2.dnn.readNet("./config\_files/yolov5s.onnx")

if IS\_CUDA:

print("Attempty to use CUDA")

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA\_FP16)

else:

print("Running on CPU")

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CPU)

return net

**Листинг 1 – Учитавање модела из *ONNX* фајла**

Разлог за коришћење *small* модела наспрам регуларног јесте бољи одзив који је показао приликом тестирања као и боље перформансе и ефикасност приликом извршавања на лошијем хардверу. Прецизност јесте мања него на регуларној архитектури, међутим, пракса је показала да је корисницима много боље дати неки нерелевантан резултат који ће се због своје мале релевантности свеједно рангирати при крају враћених резултата него да се не прикаже неки потенцијално релевантан резултат.

У листингу 2 је приказан handler који врши процесирање слике (на високом нивоу апстракције) и враћа резултат кориснику:

def handle\_image\_processing\_request(image: RawImage, ml\_model) -> ProcessedImageData:

class\_list = load\_classes()

image = load\_image(image.b64\_image)

input\_image = format\_yolov5(image)

outs = detect(input\_image, ml\_model)

class\_ids, \_, \_ = wrap\_detection(input\_image, outs[0])

classes = {}

for classid in class\_ids:

try:

if classes[class\_list[classid]] > 1:

classes[class\_list[classid] + "\_multiple"] = 1

classes[class\_list[classid]] += 1

except:

classes[class\_list[classid]] = 1

continue

hue, saturation = get\_dominant\_hue\_and\_saturation(image)

return ProcessedImageData(classes=list(classes.keys()), hsv\_color\_space=(hue, saturation))

**Листинг 2 – *Handler* за процесирање слике (висок ниво апстракције)**

Овдје можемо видјети да функција detect врши једноставан forward pass док се читање излазних слојева обавља у функцији wrap\_detection. Имплементација ове функције представљала је најсложенији део овог сервиса стога је нјен преглед дат у листингу 3.

def wrap\_detection(input\_image, output\_data):

global CONFIDENCE\_THRESHOLD

global SCORE\_THRESHOLD

global NMS\_THRESHOLD

class\_ids = []

confidences = []

boxes = []

rows = output\_data.shape[0]

image\_width, image\_height, \_ = input\_image.shape

x\_factor = image\_width / INPUT\_WIDTH

y\_factor = image\_height / INPUT\_HEIGHT

for r in range(rows):

row = output\_data[r]

confidence = row[4]

if confidence >= CONFIDENCE\_THRESHOLD:

classes\_scores = row[5:]

\_, \_, \_, max\_indx = cv2.minMaxLoc(classes\_scores)

class\_id = max\_indx[1]

if (classes\_scores[class\_id] > SCORE\_THRESHOLD):

confidences.append(confidence)

class\_ids.append(class\_id)

x, y, w, h = row[0].item(), row[1].item(), row[2].item(), row[3].item()

left = int((x - 0.5 \* w) \* x\_factor)

top = int((y - 0.5 \* h) \* y\_factor)

width = int(w \* x\_factor)

height = int(h \* y\_factor)

box = np.array([left, top, width, height])

boxes.append(box)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.25, NMS\_THRESHOLD)

result\_class\_ids = []

result\_confidences = []

result\_boxes = []

for i in indexes:

result\_confidences.append(confidences[i])

result\_class\_ids.append(class\_ids[i])

result\_boxes.append(boxes[i])

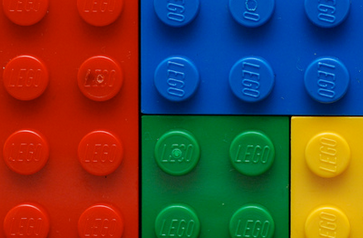
return result\_class\_ids, result\_confidences, result\_boxes

**Листинг 3 – Декодирање излазног слоја YOLOv5 модела**

Детаљно објашњење како ова функција ради дато је у наставку:

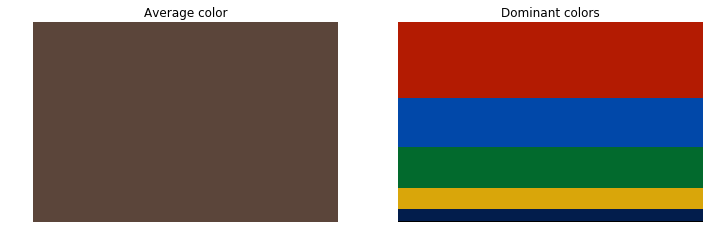
1. Креирају се празне листе class\_ids, confidences и boxes за чување идентификатора класа, конфиденцијалних оцена и ограничавајућих оквира препознатих објеката.
2. Број редова добија се из низа output\_data, који представља број препознатих објеката.
3. Издваја се ширина и висина слике input\_image.
4. Израчунавају се фактори скалирања (x\_factor и y\_factor) за конверзију координата ограничавајућих оквира (bounding boxes) са величине уноса модела на стварну величину слике.
5. Итерација се обавља за сваки ред у output\_data.
6. Проверава се да ли је конфиденцијална оцена препознатог објекта изнад CONFIDENCE\_THRESHOLD.
7. Проналази се идентификатор класе са највишом сигурношћу применом cv2.minMaxLoc на преосталим елементима реда.
8. Ако је оцена за препознату класу изнад SCORE\_THRESHOLD, сматра се да је детекција валидна.
9. Рачунају се координате ограничавајућег оквира на основу положаја и величине препознатог објекта.
10. Примењују се фактори скалирања на координате како би се прилагодле стварној величини слике.
11. Идентификатор класе, сигурност и ограничавајући оквир се додају у одговарајуће листе.
12. Примењује се non-maximum suppression (NMS) помоћу cv2.dnn.NMSBoxes да би се уклонили преклапајући гранични оквири са ниским сигурностима. Функција враћа индексе изабраних оквира.
13. На крају, враћају се крајње листе идентификатора класа, оцјена сигурности и ограничавајућих оквира.

На крају, остало је да видимо и како се рачуна доминантан простор боја на слици. Разлог за коришћење доминантног простора боја наспрам просјечног је прост: приступ са рачунањем просечне боје може довести до боје која је различита од најистакнутије визуелне боје. Да би илустровао ово понашање искористићу слику 5.1.1.



Слика 5.1.1 *Lego* коцке разних боја

На лиевом дијелу слике испод приказана је просјечна боја. Јасно се види да прорачуната просјечна боја не одговара правилно садржају боје оригиналне слике. Заправо, у оригиналној слици ниједан пиксел нема ту боју. Десни део слике приказује пет најрепрезентативнијих боја, сортираних од горе ка доље по опадајућем редоследу важности (честоћа појављивања). Ова палета јасно показује да је доминантна боја црвена, што је у складу са чињеницом да највећи регион уједначене боје на оригиналној слици одговара црвеном делу *Lego* коцкице.



Слика 5.1.2 Просјечна и доминантна боја

Функција get\_dominant\_color\_space је дата у листингу 4. Као улазни параметар, ова функција прима слику а као излаз даје торку (tuple) са HSV простором боја доминантне боје на слици.

def get\_dominant\_color\_space(image):

pixels = np.float32(image.reshape(-1, 3))

n\_colors = 3

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 200, .1)

flags = cv2.KMEANS\_RANDOM\_CENTERS

\_, labels, palette = cv2.kmeans(pixels, n\_colors, None, criteria, 10, flags)

\_, counts = np.unique(labels, return\_counts=True)

return palette[np.argmax(counts)]

**Листинг 4 – Екстракција HSV простора доминантне боје на слици**

Ова функција ради на следећи начин:

1. Претвара улазну слику у низ пиксела користећи np.float32 и реорганизује пикселе у форму (-1, 3), гдје се 3 односи на RGB вредности боја (OpenCV библиотека ради са BGR форматом).
2. Подешава параметре за извршавање K-means алгоритма, који се користи за груписање пиксела по боји.
3. Позива функцију cv2.kmeans и просљеђује низ пиксела, број жељених боја (у овом случају 3), критеријуме за завршетак и почетне центроиде.
4. Добијени резултат груписања се прима и дијели на лабеле (labels), палету боја (palette) и фреквенцију појава сваке боје (counts).
5. На крају, функција враћа боју из палете која има највећи број појава, што представља доминантну боју у слици.

## Сервис за претрагу

Сервис за претрагу служи за претрагу и индексирање слика. За имплементацију овог сервиса коришћен је *Java Spring Boot* радни оквир. Интеграција са *ES*-ом одрађена је уз помоћ *ES Јava API*-а (*Elasticsearch Java Client*). Разлог за коришћење наведеног поред *Spring Data Elasticsearch*-а је тај што сам желио да пробам нешто другачије. Сервисна метода за индексирање дата је у листингу 5.

public void indexImage(MultipartFile imageUpload) throws Exception {

var fileName = imageService.saveImage(imageUpload);

var base64Image = imageUtil.multipartImageToBase64(imageUpload);

ProcessedImageDataDTO processedImageData = new ProcessedImageDataDTO();

try {

processedImageData = imageProcessingClient.processImage(new RawImageDTO(base64Image));

} catch (Exception e) {

imageService.deleteSavedImage(fileName);

throw new ImageProcessingFailedException(

"Something went wrong when trying to process image.");

}

var image = new Image();

StringBuilder tags = new StringBuilder();

for (var tag : processedImageData.getClasses()) {

tags.append(tag).append(" ");

}

image.setFileName(fileName);

image.setTags(tags.toString());

image.setHue(processedImageData.getHsvColorSpace()[0]);

image.setSaturation(processedImageData.getHsvColorSpace()[1]);

elasticsearchClient.index(i -> i

.index("images")

.id(image.getId())

.document(image)

);

}

**Листинг 5 – Метода за индексирање слике**

Овај метод прима *MultipartFile* објекат који представља постављену слику. Процес индексирања слике састоји се од следећих корака:

1. Прво се сачува постављена слика на серверу користећи imageService.saveImage метод, који враћа име датотеке слике.
2. Затим се слика конвертује у Base64 формат помоћу imageUtil.multipartImageToBase64 методе.
3. Обрада слике се врши позивом imageProcessingClient.processImage методе, који упућује HTTP захтјев сервису за процесирање слика, гдје се шаље објекат RawImageDTO са Base64 енкодованом сликом.
4. Након успешне обраде слике, креира се нови Image објекат и пролази се кроз све класе (тагове) које су добијене са сервиса за процесирање и додају се у StringBuilder tags.
5. Постављају се подаци о слици, као што су име датотеке, тагови, hue и сатурација, користећи податке из processedImageData.
6. Индексирање слике се обавља позивом Elasticsearch клијента (elasticsearchClient.index) и прослеђивањем имена индекса, идентификатора и података о слици.

Претрага слике је приказана у листингу 6.

public Page<ImageDisplayDTO> searchForImage(SearchRequestDTO sampleImageUpload,

Pageable pageable)

throws IOException {

var base64Image = imageUtil.multipartImageToBase64(sampleImageUpload.getImage());

var imageHash = DigestUtils.md5Hex(base64Image).toUpperCase();

ProcessedImageDataDTO processedImageData = cacheService.retrieveCached(imageHash);

if (processedImageData == null) {

try {

processedImageData =

imageProcessingClient.processImage(new RawImageDTO(base64Image));

cacheService.cache(processedImageData, imageHash);

} catch (Exception e) {

throw new ImageProcessingFailedException(

"Something went wrong when trying to process image.");

}

}

var queryBuilder = buildQuery(processedImageData.getClasses());

var searchQueryBuilder = new NativeSearchQueryBuilder()

.withQuery(queryBuilder)

.withPageable(pageable);

if (sampleImageUpload.getHsvSort()) {

var script = new Script(

"Math.sqrt(Math.pow(doc.hue.value - " + processedImageData.getHsvColorSpace()[0] +

", 2) + Math.pow(doc.saturation.value - " +

processedImageData.getHsvColorSpace()[1] + ", 2))");

var sort = new ScriptSortBuilder(script, ScriptSortBuilder.ScriptSortType.NUMBER);

searchQueryBuilder.withSorts(sort);

}

var searchQuery = searchQueryBuilder.build();

var searchHits = template

.search(searchQuery, Image.class, IndexCoordinates.of("images"));

var searchHitsPaged = SearchHitSupport.searchPageFor(searchHits, searchQuery.getPageable());

var page = (Page<Image>) SearchHitSupport.unwrapSearchHits(searchHitsPaged);

return page.map(image -> new ImageDisplayDTO(image.getFileName()));

}

**Листинг 6 – Метода за претрагу слике**

Ова метода ради на сљедећи начин:

1. Прима SearchRequestDTO објекат који садржи узорачку слику и Pageable објекат који омогућава страничну претрагу.
2. Претвара постављени примерак слике у Base64 формат помоћу imageUtil.multipartImageToBase64 методе. Такође, рачуна хеш вриједност слике користећи MD5 алгоритам.
3. Проверава се да ли се обрађени подаци слике налазе у кешу помоћу cacheService.retrieveCached методе. Уколико не постоје, врши се обрада слике помоћу imageProcessingClient.processImage методе и добијени подаци се чувају у кешу помоћу cacheService.cache методе.
4. Конструише се queryBuilder на основу класа (тагова) из processedImageData.
5. Креира се searchQueryBuilder који садржи конфигурацију за претрагу. Додаје се queryBuilder и pageable у претрагу.
6. Уколико је hsvSort постављен на *true* у sampleImageUpload, додјељује се скрипта за сортирање на основу *HSV* вредности. Ова скрипта рачуна еуклидско растојање између *HSV* вредности слике и processedImageData. Примјењује се сортирање на основу ове скрипте.
7. Креира се searchQuery на основу searchQueryBuilder-а.
8. Извршава се претрага помоћу template.search методе. Добијени резултати се складиште у searchHits промјенљивој.
9. Примјењује се постранична обрада резултата помоћу SearchHitSupport.searchPageFor и добијени резултати се чувају у searchHitsPaged промјенљивој.
10. Промјенњива searchHitsPaged се *cast*-ује у Page<Image> објекат.
11. Креира се нови Page<ImageDisplayDTO> мапирањем сваке слике у ImageDisplayDTO објекат (у овом случају, само се преноси име датотеке).
12. Враћа се резултујућа страница са претраженим сликама у облику Page<ImageDisplayDTO>.

Битно је нагласити да је комуникација са сервисом за процесирање слика имплементирана помоћу *Open* *Feign Client* библиотеке, имплементација је дата у листингу 7.

@FeignClient(name = "image-processing-client", url = "http://${image-processing-service.host}:8000")

public interface ImageProcessingClient {

@PostMapping("/preprocess")

ProcessedImageDataDTO processImage(@RequestBody RawImageDTO rawImageDTO);

}

**Листинг 7 – *OpenFeign* клијент за комуникацију са сервисом за процесирање слике.**

## Клијентски део апликације (frontend)

Клијентска апликација у раду је имплементирана у *Vue 3* радном оквиру За стилизацију и компоненте интерфејса, коришћена је *Vuetify* библиотека. *Vuetify* је *Vue*-базирана библиотека компоненти која омогућава брзу и ефикасну израду лијепог и респонзивног корисничког интерфејса.

Једна од важних карактеристика *Vuetify*-а је његов grid систем. Овај систем омогућава лако позиционирање и организацију компоненти на страницама апликације и заснива се на систему колона и редова, где је могуће одредити колико колона компонента заузима и како се компоненте распоређују унутар истих. Овај *grid* систем је флексибилан и респонзиван, што значи да се компоненте адаптирају на различите екранске величине, што доприноси бољем корисничком искуству на различитим уређајима. У листингу 8 дат је примјер коришћења *Vuetify* компоненти и његовог *grid* система.

<v-row>

<v-col sm="11" cols="10">

<v-file-input

label="File input"

filled

show-size

prepend-icon="mdi-camera"

v-model="files"

@change="onFileSelected"></v-file-input>

</v-col>

<v-col sm="1" cols="2">

<v-btn

style="margin-top: 10px"

variant="tonal"

@click="searchImages()">

Search

</v-btn>

</v-col>

</v-row>

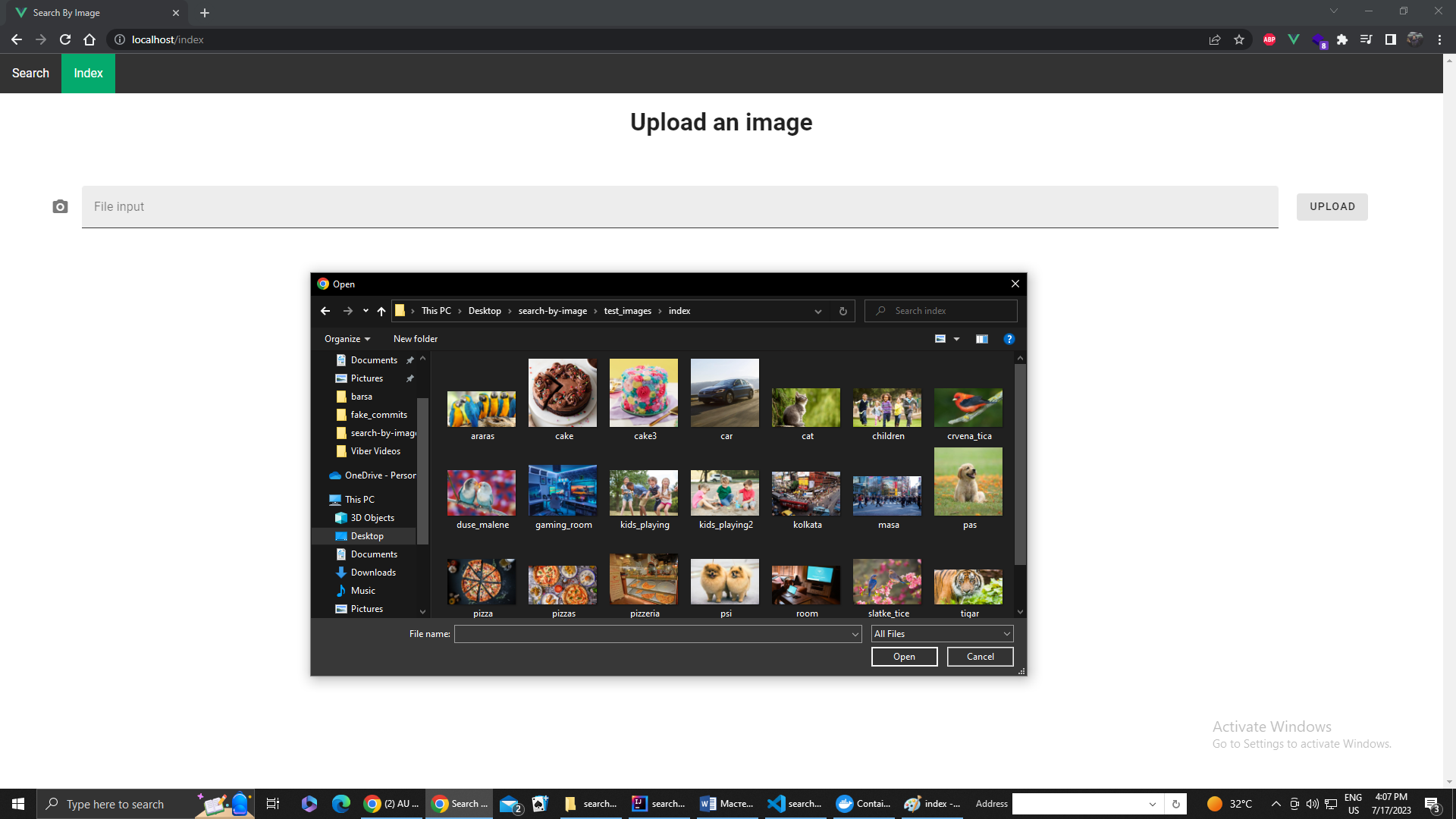
**Листинг 8 – Примјер коришћења *Vuetify* компоненти и уграђеног *grid* система**

# ДЕМОНСТРАЦИЈА

У овом поглављу биће представљена 2 сценарија коришћења система: индексирање и реверзна претрага слика.

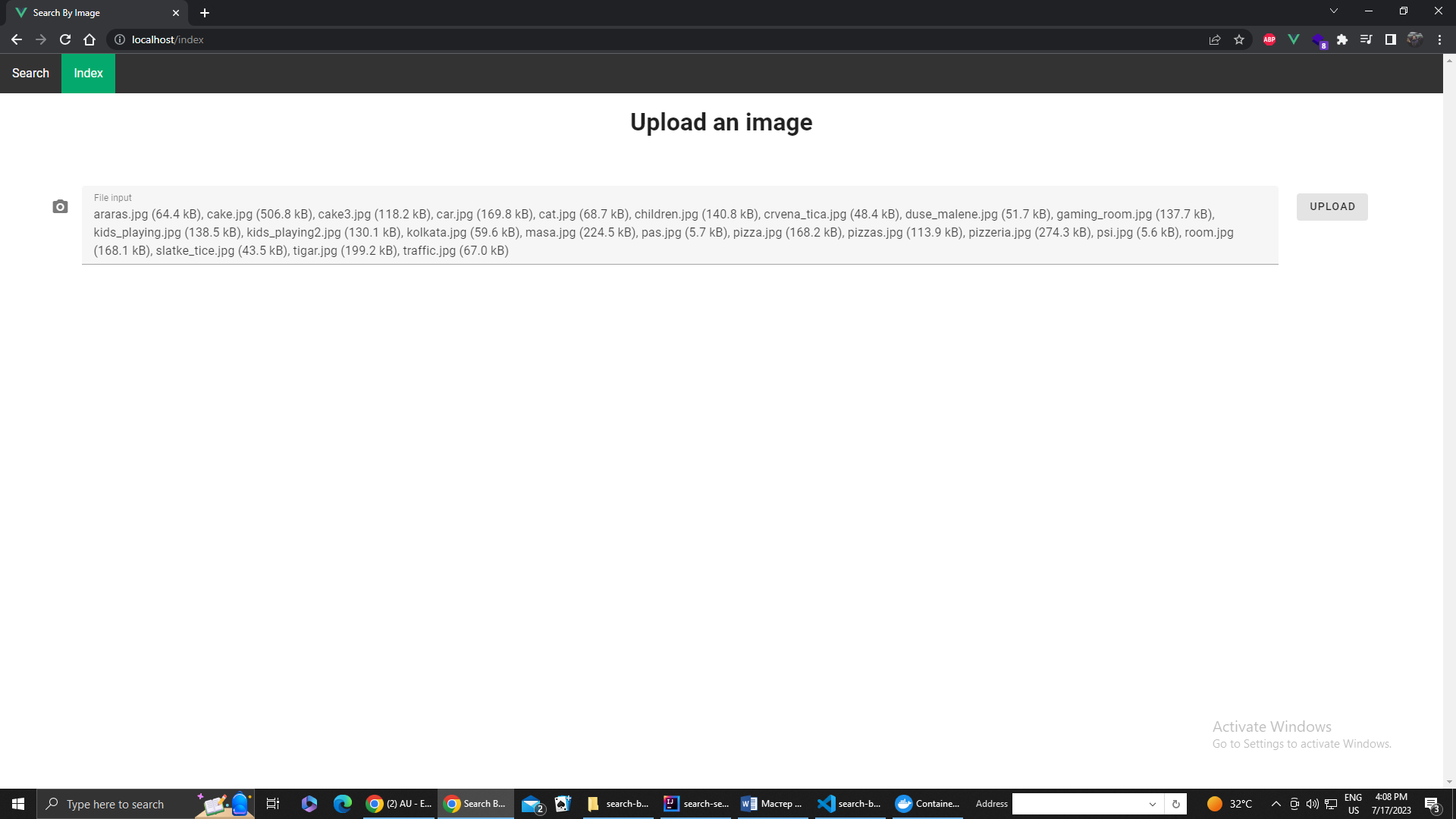
## Индексирање

Корисник почиње сценарио тако што се позиционира на страницу за индексирање и кликом на *index bar* отвори *upload dialog* (слика 6.1.1).



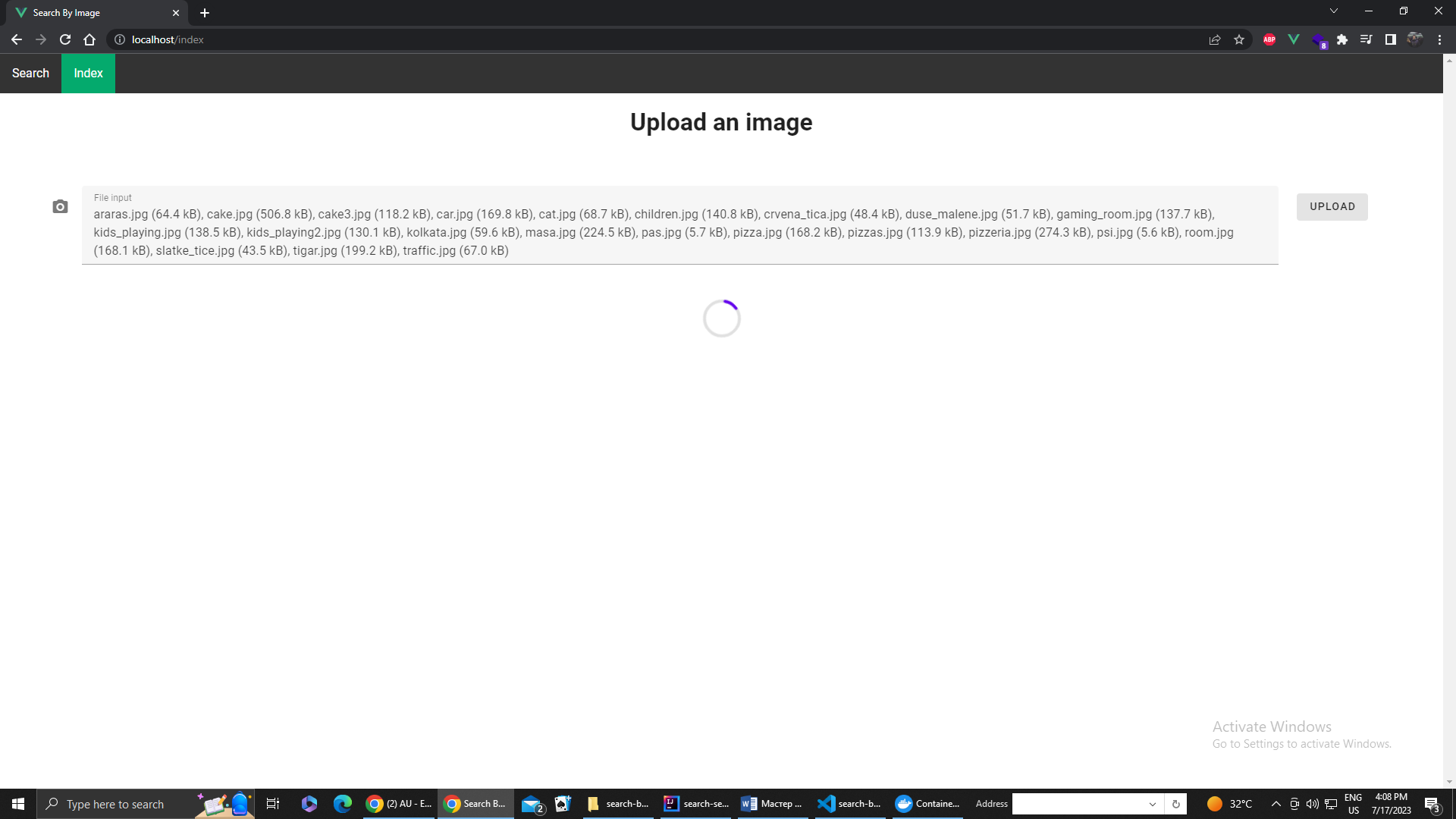
Слика 6.1.1 Селектовање слика за индексирање

Након тога, корисник селектује жељене слике и кликом на дугме „*open*“ врши селекцију слика за индексирање, имена фајлова и њихове величине су потом приказане у *index bar*-у (слика 6.1.2).

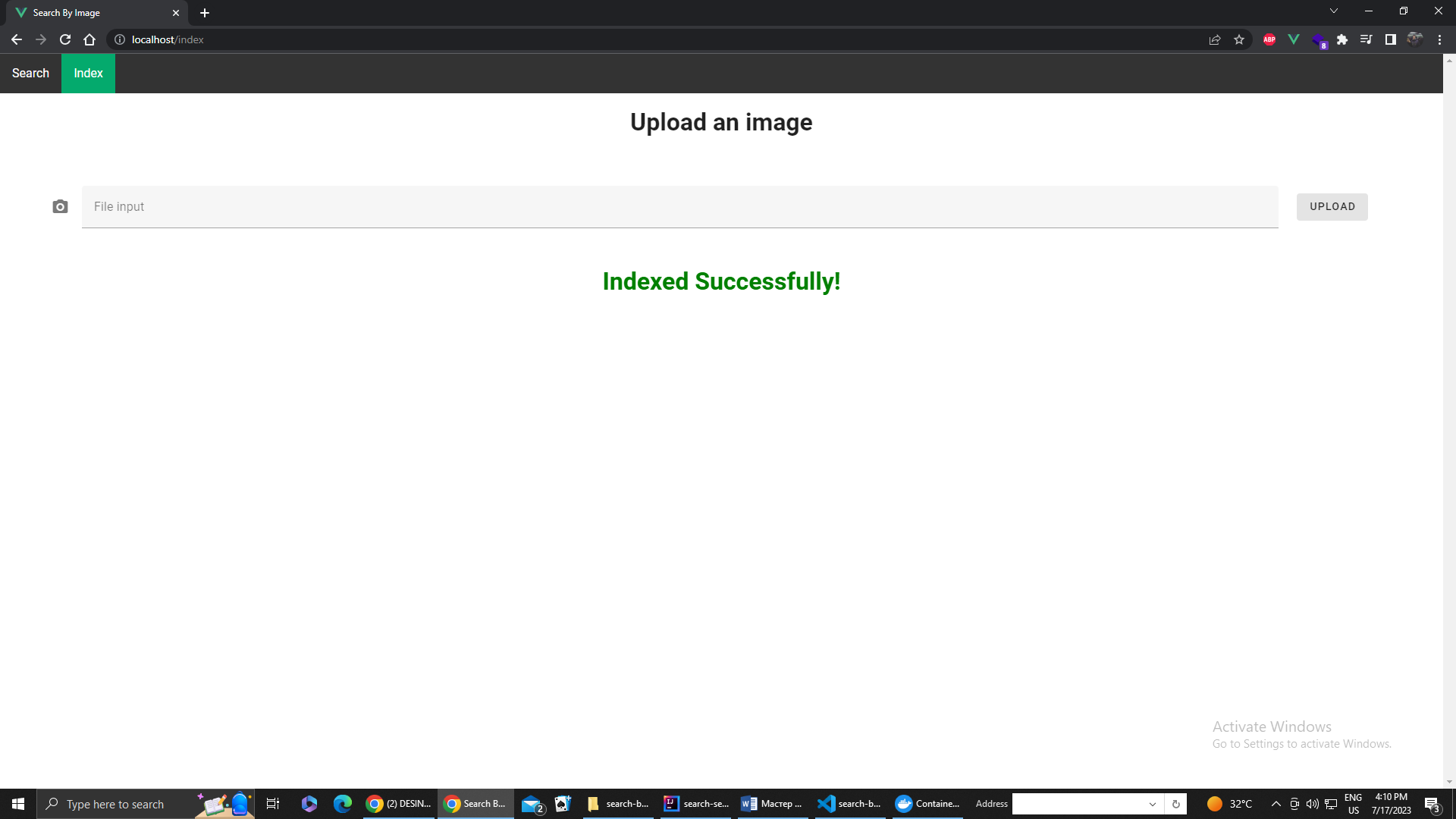


Слика 6.1.2 Селектоване слике

Кликом на дугме „*upload*“ покреће се процес индексирања, на екрану се врти *loader* све док операција није завршена за све слике (слика 6.1.3). На крају, кориснику се исписује порука о резултату операције, на слици 6.1.4 приказана је порука која обавјештава корисника да је операција индексирања успјешно завршена за све селектоване слике.



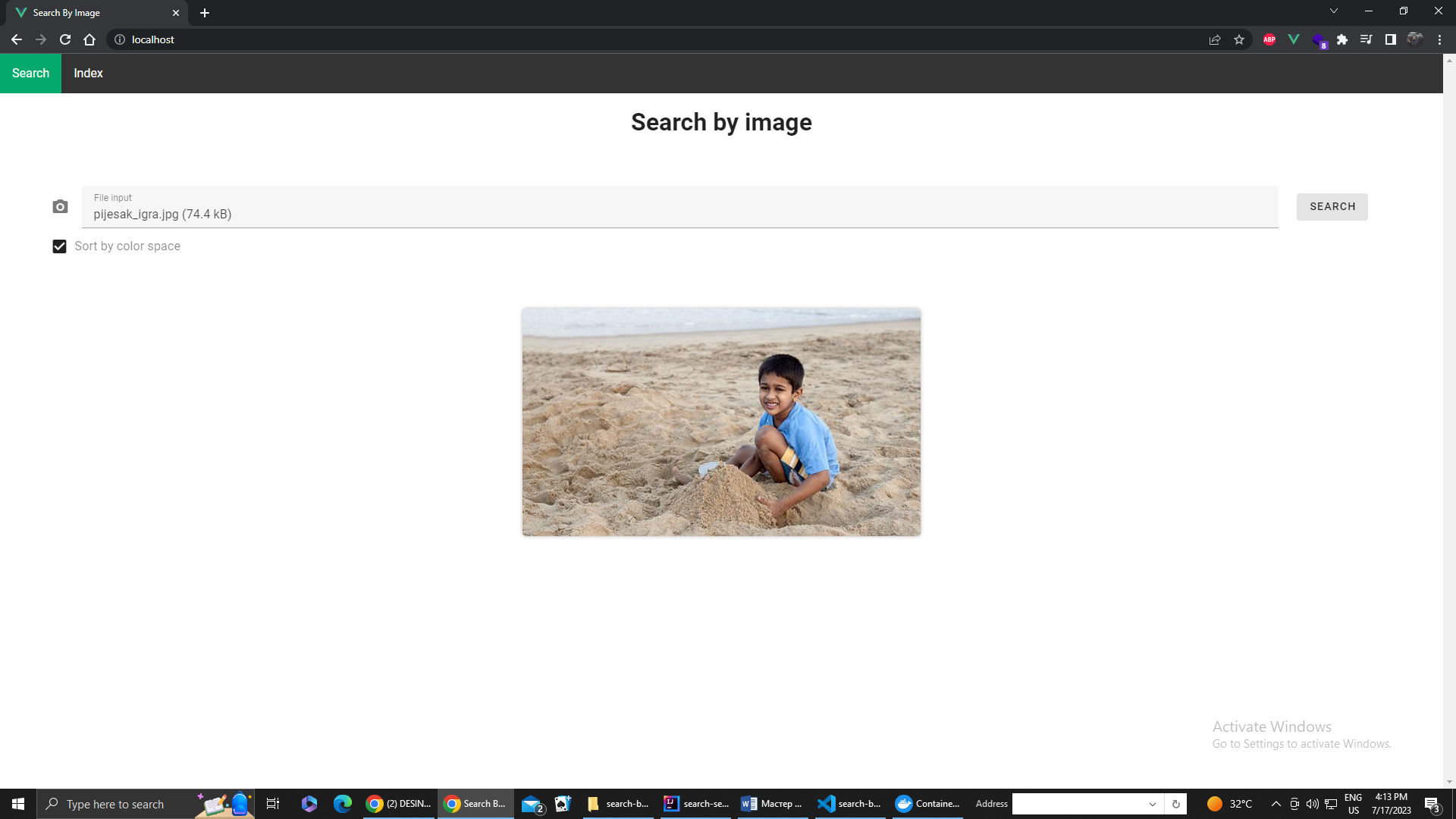
Слика 6.1.3 Изглед *loader*-а док се чека на завршетак операције индексирања селектованих слика



Слика 6.1.4 Порука о успјешно завршеној операцији индексирања

## Претрага

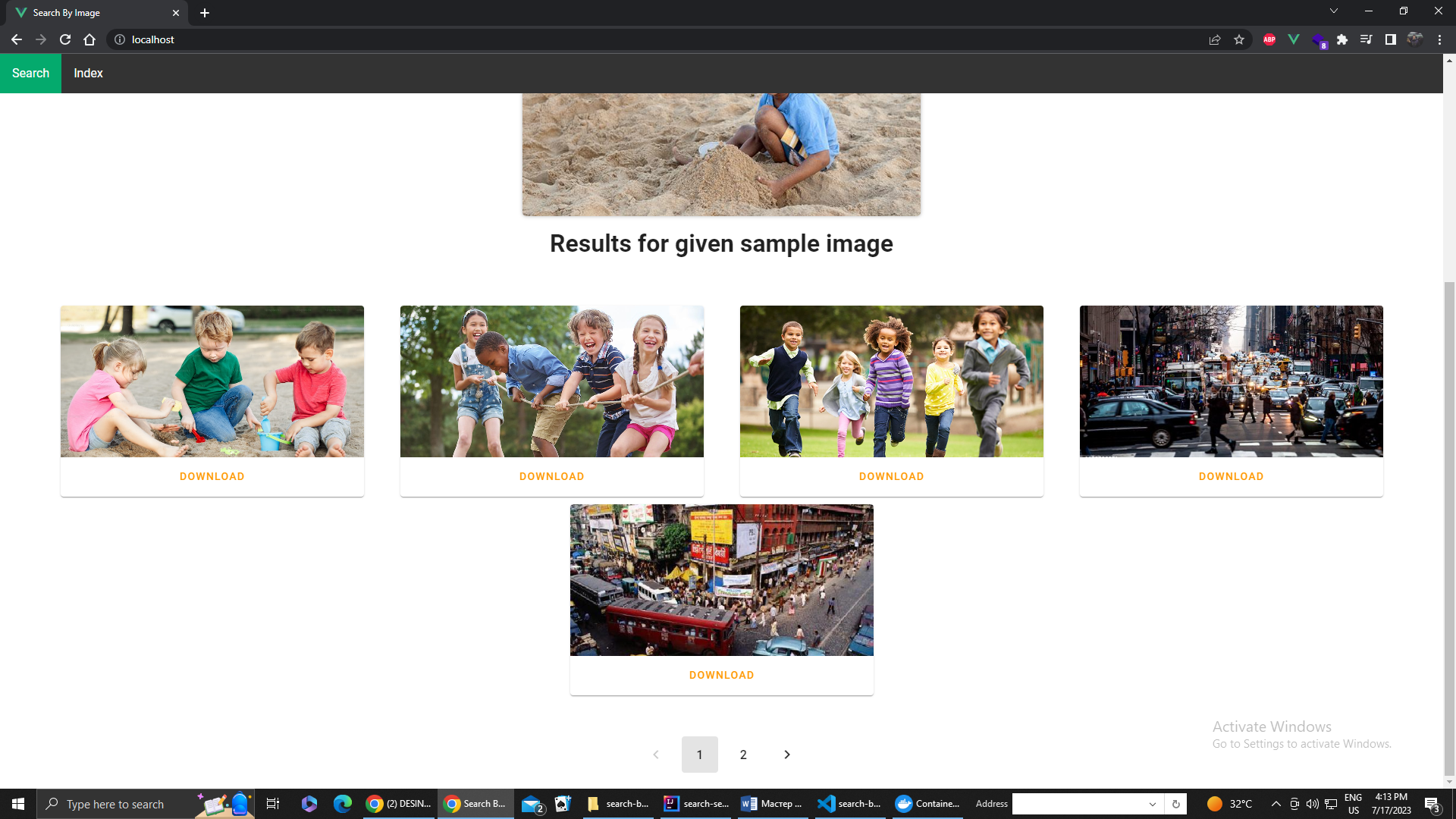
Сценарио почиње тако што се корисник позиционира на страницу за претрагу. Кликом на *search bar* отвора се *upload dialog* и процесом аналогним као у прошлом сценарију корисник селектује узорачку слику за реверзну претрагу након чега се иста приказује кориснику заједно са горе поменутим информацијама (слика 6.2.1).



Слика 6.2.1 Селектована узорачка слика и опција за сортирање уз помоћ простора боја

Следећи корак је опциони одабир сортирања резултата помоћу простора боја, у овом случају коришћења је ова опција селектована (слика 6.2.1).

Кликом на дугме „*search*“ започиње се процес претраге послије којег се кориснику приказују резултати претраге. На слици 6.2.2 је пружен приказ резултата претраге кориснику.



Слика 6.2.1 Приказ резултата претраге

# ЕКСПЕРИМЕНТ

У овом поглављу биће представљен експеримент којим је покушано да се измјере перформансе система.С обзиром да је систем у домену *information retrieval*-а најбољи начин за евалуацију истог је да се спроведе истраживање са што већим бројем корисника који могу да дају субјективну оцјену за његово функционисање (не вриједи нам да имамо перфектан резултат на аутоматизованом тесту уколико је субјективни user experience лош). У одељку 7.1 биће детаљно описан експеримент док ће у поглавњу 7.2 и 7.3 бити описани коришћени скуп података као и методе евалуације респективно.

## Опис експеримента

Експеримент је спроведен у виду анкете:

1. Сваки учесник првобитно инсталира апликацију корисћењем *docker-compose* алата
2. Након тога, сваки корисник треба да индексира индекс скуп података кроз форму за индексирање
3. На крају, за сваку од слика у упитном скупу података корисник има поље у анкети да оцијени оцјеном од 1 до 5 одзив, редослијед (прецизност) одговора, као и редослијед одговора приликом коришћења опције за сортирање помоћу простора боја

Приликом давања оцјена, корисник има на увид цијели индекс скуп података како би могао сам да донесе одлуку о броју *true positive* инстанци у датом скупу података.

## Скуп података

Тестирање се врши употребом ручно сакупљеног скупа података коришћењем *Google Images* сервиса. Сакупљено је 30 слика за индекс скуп и 10 слика за упитни скуп. Битно је напоменути да су слике изабране тако да постоји велики број слика које могу да одговоре на више од једног упита како би коначна процјена свих метрика била остављена субјективном осјећају корисника.

## Евалуација

Евалуација система се врши на веома једноставан начин. Корисницима је назначено у упитнику да по сопственом нахођењу извуку мјеру одзива и прецизности (редослијед) одговора за сваки од sample упита. Прецизност, поред тога што се рачуна за стандардно сортирање *Elasticsearch*-овим уграђеним *value* параметром, рачуна се и за случај када је селектовано сортирање помоћу простора боја. Детањније објашњење горе поменутих метрика слиједи у наставку:

Прецизност (*precision*) је мјера која представља колико тачно модел идентификује позитивне инстанце у односу на све инстанце које је модел означио као позитивне. Израчунава се као однос тачно идентификованих позитивних инстанци и укупно означених позитивних инстанци (у домену information retrieval-а на прецизност утиче и квалитет сортирања одговора тј. позитивних инстанци).

Одзив (*recall*) је мјера која представља колико тачно модел идентификује све позитивне инстанце у односу на све постојеће позитивне инстанце у скупу података. Израчунава се као однос тачно идентификованих позитивних инстанци и укупно постојећих позитивних инстанци.

# РЕЗУЛТАТИ И ДИСКУСИЈА

Резултати претходно описаног експеримента представљени су у табели 8.1. За сваку узорачку слику извучене су сљедеће статистике: просјечан одзив код свих испитаника, просјечна прецизност без коришћења додатних филтера, просјечна прецизност приликом коришћења сортирања у простору боја. У испитивању је учествовало 10 кандидата.

|  |  |  |  |
| --- | --- | --- | --- |
| Садржај слике | Одзив | Прецизност (default) | Прецизност (HSV sort) |
| ауто | 5 | 3 | 3.25 |
| торта | 4 | 3.25 | 3 |
| мачка | 4.5 | 4.25 | 4.25 |
| пас | 3 | 4.25 | 4 |
| игра у пијеску | 4.75 | 2.75 | 4.25 |
| пица | 2.75 | 2 | 4.75 |
| птице | 5 | 5 | 4.25 |
| телевизор | 3.25 | 4 | 4 |
| улица | 4.5 | 5 | 4 |
| прибор за јело | 3.75 | 4.25 | 4.25 |

Табела 8.1 Резултати експеримента

Из приложених резултата може се закључити да је одзив система задовољавајућ са просјечном оцјеном 4.05 од 5 док је прецизност оцијењена са 3.775 и 4.475 у зависности од тога да ли је резултат сортиран помоћу уграђеног *value* параметра или уз помоћ простора боја. Најлошији одзив је добијен приликом упита у којем је као узорак коришћена слика са пицом. Када је овај случај детаљније анализиран, дошло се до налазаа да је приликом процесирања слике, пица детектована као колач, стога није враћена велика количина слика на којојима се налазио релевантан објекат. Начин да се овај проблем ријеши јесте додатно дотрениравање *YOLOv5* модела над класом објекта која има највише проблема приликом класификације.

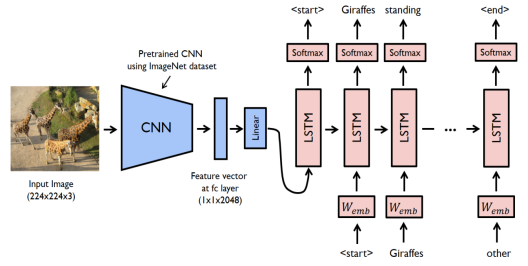
# ЗАКЉУЧАК

У овом раду видјели смо један начин имплементације система за реверзно претраживање слика. Овакав систем омогућава корисницима да уместо да се ослањају на кључне речи или описе за претрагу слика, на ефикасан начин пронађу жељене информације на основу саме слике. Ово може да има разнородну примену, од помоћи у идентификацији различитих предмета и локација, до потпуног унапређења искуства куповине преко интернета.

Систем се састоји из два одвојена сервиса: сервиса за претрагу који врши индексирање и претрагу користећи *Elasticsearch* базу података као и кеширање користећи *Redis* *in-memory* базу и сервиса за процесирање слике који користи моћне моделе машинског учења како би извукао својства из слике и преставио је у формату који је лако претражив.

За разлику од већине система који врше претрагу користећи векторски начин претраге (нпр. *Web-Scale Responsive Visual Search*), овај систем своди проблем претраге слике на проблем претраге текста гдје се својства извлаче аутоматски насупрот традиционалном приступу гдје су се уносила ручно. Овакав иновативан приступ већ дуго је заступљен у Pinterest-овом Similar Search сервису и беома лако може бити надопуњен ручним уносом описа слике, кључних ријечи итд..

Начини надоградње овог система су многобројни. С обѕиром да је имплементиран само подскуп различитих могућности процесирања слике, додавањем било каквог додатног начина процесирања могуће је постићи побољшање система. Једна ствар која највише има смисла је да се постојећи начин претраге укомбинује са векторским начином претраге тако да постојећи начин исфилтрира контекстно релевантне резултате док векторска претрага проналази визуелно најсличнију слику. Олакшавајућа околност код овог приступа је та што *Elasticsearch* подржава векторско индексирање и претрагу *out-of-the-box*. Такође, додавање модула за аутоматско генерисање описа или кључних ријечи из слике коришћењем машинског учења, могуће је побољшати квалитет извучених својстава и проширити домен примјене гдје би се поред реверзне претраге могла имплементирати и стандардна текст базирана претрага. Један од начина на који би ово могло бити уведено је коришћењем *CNN*-а за екстракцију обележја и рекурентну неуронску мрежу (попут *LSTM-*а) или трансформер за генерисање текста (слика 9.1).



Слика 9.1 Приједлог архитектуре *pipeline*-а за генерисање описа слике

Још један занимљив начин имплементације јесте интеграција овог система са неким *LLM*-ом који може генерисати опис/кључне ријечи слике (нпр. *ChatGPT4*).

# ЛИТЕРАТУРА

1. *Cider* razvojno okruženje <https://cider.readthedocs.io/en/latest/> [Datum pristupa 13.08. 2016].
2. Abraham, A., 2005. Rule‐Based expert systems. *Handbook of measuring system design*.
3. Gabriel, R.P. and Pitman, K.M., 1988. Endpaper: Technical issues of separation in function cells and value cells. *Lisp and Symbolic Computation*, *1*(1), pp.81-101.

# БИОГРАФИЈА

Иван Мршуља је рођен 31.01.2000. у Котору, гдје је стеАкао основно и средње образовање. Школске 2018/19 године се уписује на Факултет Техничких Наука на студијски програм Софтверско Инжењерство и Информационе Технологије. Положио је све испите предвиђене планом и програмом и дипломирао у септембру 2022 године са завршним радом „Праћење и препознавање геста шаке комбинацијом неуронских мрежа и традиционалних приступа“. Исте године, школске 2022/2023, уписује се на мастер академске студије, на студијски програм Софтверско Инжењерство и Информационе Технологије – Електронско Пословање. Положио је све испите предвиђене планом и програмом и стекао услов за одбрану завршног рада.