

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK**



**SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM**

Nagy méretű képkollekciók megtekintése

DIPLOMADOLGOZAT

Témavezető:
Dr. Egyed Zsigmond Előd,
Egyetemi Docens

Végzős hallgató:
Molnár Krisztián

2023

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ**



UNIVERSITATEA SAPIENTIA

Vizualizarea online a colecțiilor mari de imagini

LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Egyed Zsigmond Előd,
Conferențiar universitar

Absolvent: Molnár Krisztián

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA**
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

Visualizations of large collections of images

BACHELOR THESIS

Scientific advisor: Dr. Egyed Zsigmond Előd,
Associate professor Student: Molnár Krisztián

2023

LUCRARE DE DIPLOMĂ

Coordonator științific
dr. : EGYED-ZSIGMOND Előd

Candidat: **MOLNAR Krisztián**
Anul absolvirii: 2023

a) Tema lucrării de licență: Vizualizarea online a colecțiilor mari de imagini

b) Problemele principale tratate:

Scopul cercetării este de a afișa o cantitate mare de imagini într-un browser web.

Există mai multe programe de gestionare a colecțiilor de imagini, dar puține dintre ele sunt capabile să grupaze și să afișeze simplu imagini în funcție de aspectul lor. Scopul cercetării este dezvoltarea unui astfel de sistem. Intrarea constă într-un set de imagini (foarte multe imagini) și, eventual, o bază de date care conține cuvinte cheie. Utilizatorul selectează o metodă de comparare a imaginilor și/sau introduce o cuvânt cheie, iar sistemul afișează imaginile grupate într-un browser în funcție de distanța corespunzătoare funcției de distanță. Rearanjarea și filtrarea imaginilor sunt, de asemenea, importante.

c) Desene obligatorii:

- Schema bloc a sistemului
- Diagrame de proiectare pentru aplicația software realizată.
- Capturi de ecran

d) Softuri obligatorii:

Prototip de afisare a imaginilor

e) Bibliografia recomandată:

<http://twittertoolsbook.com/10-awesome-twitter-analytics-visualization-tools/>

http://web2.cs.columbia.edu/~blei/seminar/2016_discrete_data/readings/MikolovSutskeverChenCorradoDean2013.pdf

<http://lab.softwarestudies.com/p/imageplot.html>

http://www.phash.org/docs/pubs/thesis_zauner.pdf

<http://manovich.net/>

f) Termene obligatorii de consultații: săptămânal, preponderent online

g) Locul și durata practiciei: Universitatea „Sapientia” din Cluj-Napoca,

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș,

Primit tema la data de: 17/01/2023

Termen de predare: 06/07/2023

Semnătura Director Departament

Semnătura responsabilului
programului de studiu

Semnătura coordonatorului

Semnătura candidatului

Declarație

Subsemnatul/a **MOLNAR KRISZTIAN**, absolvent(ă) al/a specializării **INFORMATICĂ**, promoția **2023** cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, **CORUNCA**
Data: **16.06.2023**

Absolvent

Semnătura... 

Kivonat

Napjainkban mind személyesen, mind kutatásban hozzáférésünk van nagy méretű képkollekciókhöz. Ezek vizualizálása hozzájárúhat egyébként látens minták felfedezéséhez.

Dolgozatom témája nagy méretű képkollekciók megjelenítése és vizualizálása kölünböző elrendezések szerint. Az általam felhasznált képkollekciók mérete meghaladja a két-százötvenezer képet, viszont az elméleti felső határ ennél jóval több.

Készítettem egy alkalmazást, amely másodpercen belül képes megjeleníteni a megnyitott kollekció képeit, és elrendezni azokat a képekből kivont metaadatok alapján.

Az alkalmazásom szerver-kliens arhitektúrára épül. A szerver a Rust[[rus23](#)] programozónyelvben készült az Axum[[axu23](#)] keretrendszerrel. A böngészőben futó kliens Typescript[[typ23](#)]-ben készült és két fő részből áll. A kívánt elrendezést egy React[[rea23](#)] keretrendszerrel készített felhasználói felületen lehet kiválasztani, és ez a kód rész felelős a kiválasztott elrendezés beállításaiért is. A képek megjelenítéséért felelős rész nem használ keretrendszeret, saját tervezésű arhitektúrát követ. A képek kirajzolásához a WebGL[[web23](#)] könyvtár került használatra.

Az alkalmazás sok fejlesztési cikluson ment keresztül; a továbbiakban bemutatom hogy ekkora képkollekciók megjelenítéséhez milyen tervezési döntések voltak szükségesek, és ezeket mérésekkel támasztom alá.

Továbbá, bemutatom az alkalmazás által előállított elrendezéseket az általam használt kollekcióra.

Kulcsszavak: képek, megjelenítés, kollekciók, elrendezések, klaszterezés

Rezumat

În prezent, avem acces atât personal, cât și în cercetare, la colecții mari de imagini. Vizualizarea acestora poate contribui la descoperirea de modele latente.

Tema lucrării mele se referă la afișarea și vizualizarea colecțiilor mari de imagini în diferite aranjamente. Dimensiunea colecțiilor de imagini pe care le-am utilizat depășește patruzece de mii de imagini, însă limita teoretică superioară o depășește cu mult.

Am creat o aplicație care poate afișa imaginile din colecția deschisă în câteva secunde și le poate aranja pe baza metadatelor extrase din acele imagini.

Aplicația mea este construită pe o arhitectură client-server. Serverul a fost realizat în limbajul de programare Rust[rus23], utilizând framework-ul Axum[axu23]. Clientul care rulează în browser a fost realizat în Typescript[typ23] și constă în două părți principale. Dispunerea dorită poate fi selectată printr-o interfață de utilizator realizată cu framework-ul React[rea23], iar acest cod este responsabil și pentru setările de dispunere selectate. Partea responsabilă de afișarea imaginilor nu utilizează un framework, ci urmează o arhitectură proiectată de mine. Pentru afișarea imaginilor, a fost utilizată biblioteca WebGL[web23].

Aplicația a trecut prin mai multe cicluri de dezvoltare; în continuare voi prezenta deciziile de proiectare necesare pentru afișarea unei astfel de colecții de imagini și le voi susține cu măsurători.

De asemenea, voi prezenta aranjamentele generate de aplicație pentru colecția pe care am utilizat-o.

Cuvinte cheie: imagini, afișare, colecții, aranjamente, clusterizare

Abstract

Currently, we have access to large image collections both personally and in research. Visualizing these collections can contribute to the discovery of latent patterns.

The topic of my thesis is the display and visualization of large image collections in various layouts. The size of the image collections I used exceeds forty thousand images, but the theoretical upper limit surpasses this by far.

I have developed an application that can display the images from the opened collection and arrange them based on extracted metadata, all within seconds.

My application is built on a client-server architecture. The server is written in the Rust[[rus23](#)] programming language using the Axum framework. The client, running in the browser, is built in TypeScript[[typ23](#)] and consists of two main parts. The desired layout can be selected through a user interface created with the React[[rea23](#)] framework, and this code is also responsible for the settings of the selected layout. The part responsible for image display does not use a framework but follows a custom-designed architecture. The WebGL[[web23](#)] library is used for rendering the images.

The application has gone through several development cycles. I will now present the design decisions necessary for displaying image collections of this size and support them with measurements.

Furthermore, I will showcase the layouts generated by the application for the collection I used.

Keywords: images, display, collections, layouts, clustering

Tartalomjegyzék

1. Bevezető	11
2. Szakirodalom	12
2.1. Google Images [goo23]	12
2.2. ImagePlot [ima23]	12
2.3. Dolgozatok	12
2.3.1. Kovács Szabolcs: Vizualizare și gestiune de colecții foarte mari de imagini (2018) [Sza18]	13
2.3.2. Bíró Tünde: Nagy képhalmazok online megjelenítése (2021) [Tü21]	13
2.4. Következtetések	14
3. Felhasznált szoftverek	15
3.1. WebGL	15
3.2. Rust	15
3.3. Axum	16
3.4. MessagePack	16
3.5. React	16
4. Rendszer specifikációja	17
4.1. Felhasználói követelmények	17
4.1.1. Általános eset diagramm	17
4.1.2. Kölleciókezelés eset diagramm	18
4.1.3. Elrendezés eset diagramm	18
4.1.4. Navigáció eset diagramm	19
4.2. Rendszer követelmények	19
4.2.1. Funkcionális követelmények	19
4.2.2. Nem-funkcionális követelmények	20
5. Tervezés	21
5.1. Arhitektúra	21
5.1.1. Kliens	22
5.1.2. Sserver	23
5.2. Kísérletek	24
5.2.1. DOM	24
5.2.2. WebGL	24
5.2.3. Textúra kezelés	25
5.2.4. Dinamikus atlasz a szerveren	25

5.2.5. Résztextúra feltöltés	25
5.2.6. Statikus atlasz	25
5.2.7. Textúrák eltárolása	26
5.2.8. Metaadatok tárolása	26
5.2.9. Elrendezések előállítása	26
5.2.10. TSNE [tsn23]	27
6. Kivitelezés	29
6.1. Felhasználói felület	29
6.1.1. Kollekciókezelés	29
6.1.2. Elrendezések kezelése	30
6.2. Elrendezések	34
6.2.1. Négyzetrács	34
6.2.2. TSNE	35
6.2.3. Időhisztogram	36
7. Mérések	38
7.1. Kollekció véglegesítése	38
7.2. Betöltési idő	38
7.3. Elrendezések előállítása	39
8. Összefoglaló	40
8.1. Továbbfejlesztési lehetőségek	40
Ábrák jegyzéke	41
Táblázatok jegyzéke	42
Irodalomjegyzék	43

1. fejezet

Bevezető

A képkészítés megjelenése óta sokat fejlődött a tehnológia fotók, és videók készítésére, tárolására és megtekintésére. Az internet jelentős része szolgálja a média tárolását, szűrését, és megtekintését.

Ennek köszönhetően mind személyes felhasználásban, mind kutatásban (például mesterséges intelligencia kutatásban) találkozunk nagy méretű képkollekcíókkal. Ezek kezelése, feldolgozása modern időkben is kihívást okoz, főleg megjelenítése és rendezése, mikor a képek száma meghaladja az ezer vagy akár tízezer képet.

Nagy méretű képkolleciók mintákat rejthetnek, amelyek felderítése személyes és tudományos előnyt is jelenthet. Minták felfedezésére számos algoritmus létezik, viszont történelmileg a legsikeresebb eszköz az emberi szem.

A bemutatásra kerülő szoftver egy rugalmas keretrendszer nagy méretű képkolleciók megtekintésére, szűrésére, és átrendezésére annak érdekében hogy a különböző elrendezésekben olyan minták legyenek szembetűnők, amelyek egyébként nem lennének egyértelműek a kollekció egyes képeit megtekintve.

Programunk lehetőséget ad több kollekció feltöltésére, ezek tartalmának metaadatainak kinyerésére, és a metaadatok alapján való szűrésre, átrendezésre, és végső soron a megjelenítésre az elkészített elrendezés alapján, és ez által betekintést nyerhetünk a kollekcióba.

Az alkalmazás elsődleges tervezési szempontja egyre nagyobb méretű kollekciók megjelenítése volt. A fejlesztés során felmerülő korai arhitekturális elköpzelések csekély ezerötözer képet voltak képesek megjeleníteni tűrhető idő alatt. A fejlesztés végeredménye-képp kapott alkalmazás viszont többszázezer képet képes megjeleníteni másodpercek alatt.

Másodlagos tervezési célnak nevezhető a gyorsaság. Amíg a megjelenítés nem ütközik problémába általa nagy kollekciók esetén, a fejlesztés során az arhitekturális döntések a betöltési, böngészési, és átrendezési sebességet kedveztek.

2. fejezet

Szakirodalom

A kép és videótartalom alkotására, szerkesztésére és publikálására használatos szoftverek és platformok száma rohamos növekedésben van. Ennek hatására másokban is felmerült a nagy képkollekciók vizualizásának ötlete. E miatt ihletet meríthettem pár hasonló elgondolásból készített szoftverből és dolgozatból.

2.1. Google Images [[goo23](#)]

A Google Images egy keresőszoftver, amely képek alapján képes általa ismert képek közül hasonlókat mutatni, ill elárulni a felfedezési helyüket. Nagy számú képet ismer, és képes feldolgozni. Mivel folytonos felfedezéssel működik, interneten elérhető képek előbb vagy utóbb bekerülnek az általa ismert képek közé.

2.2. ImagePlot [[ima23](#)]

Az ImagePlot egy nyílt forrású és ingyenes bővítménye az ImageJ képmegjelenítő és képszerkesztő szoftvernek. Nagy adathalmazok alapján képes elrendezéseke előállítani és kirajzolni, viszont a megtekintési lehetőségei limitáltak.

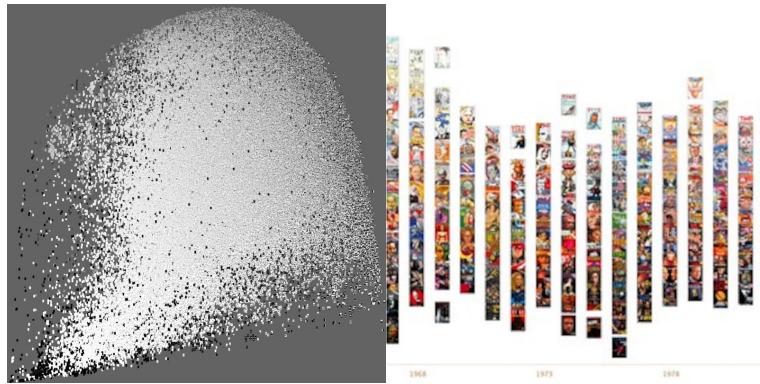
A projekt rég óta nem látott új fejlesztési erőfeszítést, és e miatt nehéz mind letölteni, minden használni, viszont a honlapja alapján képes volt akár egy millió kép elrendezésére.

Ennek ellenére leírása és a honlapon megtalálható példái alapján képes volt egy millió képet elrendezni és kirajzolni (lásd [2.1](#)).

2.3. Dolgozatok

Korábbi években többen is megpróbálták kivitelezni ezt az ötletet. mindenki saját elképzése alapján, és sajátos tervezési döntésekkel minden felhasznált tehnológiák terén, minden tervezési megközelítés terén.

Az alább felsoroltak ihletet adtak a saját megközelítéseimhez, és saját dolgozatomat az évek során felhalmozott kísérleti tudás végeredményének tekintem. A továbbiakban felsorolom és bemutatom ezeket a dolgozatokat, és különbségeiket az enyémhez képest.



2.1. ábra. Két példa az ImagePlot féle képvizualizációról

2.3.1. Kovács Szabolcs: Vizualizare și gestiune de colecții foarte mari de imagini (2018) [Sza18]

Kovács Szabolcs egy JAVA alapú megközelítést választott. Dolgozatához kötődő alkalmazás egy asztali alkalmazás volt, amely képes volt a LIRE (Lucene Image Retrieval, lásd 2.3.1) szoftver képességeivel bírni, és ezt alkalmazta az eltárolt képek metaadatainak kivonására és indexelésére.

LIRE [lir23]

A LIRE egy mostmár 2021 óta nem karbantartott vagy fejlesztett Java könyvtár, amely képes gyorsan nagy képhalmazok tulajdonságait kibányászni és indexelni. A technikát, amelyet a LIRE lehetővé tesz *Content-Based Image Retrieval*-nak hívják (Tartalom Alapú Kép-Keresés). Ez a technika képek közötti keresést tesz lehetővé képekből kibányászott metaadatok alapján.

A LIRE a gyors keresés érdekében leíró vektorokat bányász ki a képekből, és azokat használja fel a képek indexeléséhez. Majd, lehetőséget kínál a felhasználó által biztosított metaadatokat leíró vektorokká alakítani, és azok alapján keresni. Vagy, egy példakép alapján előállítani a keresés tárgyának szánt leíró vektorokat, és elvégezni a keresési folyamatot az alapján.

A LIRE több okból nem volt felhasználva az alkalmazásban. Egyrészt, a projekt 2021 óta nincs aktív fejlesztés alatt. Ez kétélyeket vet fel a könyvtár használhatóságával kapcsolatban modern projektben. Másrészt, a LIRE egy Java alapjú könyvtár. Tudtommal nem létezik könnyű/gyors módszer arra, hogy natív (gépi) kód és Java kód között kompatibilitási felületet építsünk ki. Harmadrészt, rugalmasság érdekében saját megközelítést szerettem volna kiépíteni a *tartalom alapú* szűrésre és rendezésre.

2.3.2. Bíró Tünde: Nagy képhalmazok online megjelenítése (2021) [Tü21]

Bíró Tünde egy C# alapú alkalmazást készített, amely a korábbiakkal ellentétben webes felületet használt. Ennek előnye, hogy a szoftver távolról is használhatóvá válik. Hogyha képkollekciókat például egy szerveren tároljuk, asztali alkalmazás nehezen férne

hozzá a tárolt képekhez. Ebben az esetben hogyha a szerveren futtatjuk az alkalmazás szerver-oldalát, akkor viszont távolról is tudjuk kezelní/használni az alkalmazást.

Ezek miatt, a saját alkalmazásomhoz is webes felületet választottam. Továbbá, a WebGL használatának ötlete is Bíró Tünde dolgozatából eredt.

2.4. Következtetések

Számos próbálkozás történt a nagy számú képmegjelenítés irányába, és ezek eredményei mind sajátos tulajdonságokkal rendelkező szoftverek.

A saját alkalmazásom tervezése felhasználta a korábbi alkalmazások kísérleteit, és ezek eredményét, de ugyanakkor sajátos kísérleteket is végeztem a teljesítmény maximálásának érdekében.

Az online elérhető képmegjelenítő szoftverek kontextusában a dolgozatom a nagy számú képmegjelenítést célozza, míg például a Google Images képeket legfeljebb tízesével képes megjeleníteni, közel sem éri el a százezres nagyságrendet. Az ImagePlot-al szemben, az alkalmazásom képes saját metaadatokat kinyerni, így nincs szükség harmadik szoftverre, és képes valós időben megjeleníteni az elrendezéseket, nem csak teljes képi exportálás után.

3. fejezet

Felhasznált szoftverek

Az alkalmazás számos szoftveres könyvtárt és technológiát használt fel. Ezek közül a fontosabb technológiákat az alábbiakban ismertetem.

3.1. WebGL

Az OpenGL a legidősebb, és legnépszerűbb alacsony szintű grafikai könyvtár. Általa használhatóvá válik a videókártya bármely felületen, amelynek videókártya illesztőprogramja támogatja (az összes fő felület: Windows, Linux, Mac, Android...).

A WebGL[[web23](#)] egy böngészőben elérhető könyvtár, amelyet JavaScript kódóból lehet használni. Célja, hogy webes környezetből is elérhetővé tegye az OpenGL könyvtár grafikai képességeit.

Mivel WebGL-t használva bármilyen grafikai képesség elérhető, amelyet asztali alkalmazás is képes lenne előállítani OpenGL-t használva, használatával hardveres gyorsítást kap a rajzolás. Ennek köszönhetően az alkalmazásban a vizualizációk kirajzolása gyors (1ms, GeForce GTX 1660-on).

A WebGL számos optimalizációra adott lehetőséget a hardveres rajzolás mellett. Például, WebGL-t használva futási időben nyílt lehetőség textúra atlaszok felépítésére, ezzel gyorsítva a jobb minőségű képek betöltését mikor a felhasználó ráközelít a felületen.

3.2. Rust

A Rust programozónyelv[[rus23](#)] úttörő a modern megjelenésű programozónyelvek között. A modernizációs hullám, amely sok új nyelv megjelenését ihlette a Rust esetében egy új programozási paradigmának adott teret. Ennek a paradigmának köszönhetően a Rust C++-kaliberű sebességet ér el, és mégis könnyen kezelhető, magas szintű nyelv marad, amely garantált memóriabiztonságot kölcsönöz tervezési paradigmájátó.

Azért döntöttem úgy, hogy Rust-ot fogok használni ehhez a projekthez, mert komplált nyelv, és ennek köszönhetően a lehető leggyorsabb futtatható fájlok- ra kompilál. Ugyanakkor, az elmúlt pár évben egyedi arhitektúrája, és bizonyítottan eredményes teljesítménye népszerűséggel ruházta fel, ami miatt széles körben támogatott, és rengeteg felhasználási esetre elérhető ökoszisztemájában könyvtár.

3.3. Axum

Az Axum[[axu23](#)] a Rust ökoszisztéma egy könyvtára, mely webes szerver- oldali keretrendszerként szolgál. A Rust nyelvi fejlődése során rendszeresen jelentek meg webes keretrendszerek. Eredményes teljesítményük miatt versengés alakult ki a Rust és C++-ban írt keretrendszerek között, ami táplálta a gyorsabbnál gyorsabb keretrendszerek megjelenését. Az Axum ennek az evolúciós fejlődésnek eredményeként az eddigi leggyorsabb, és legszélesebb körben támogatott keretrendszer.

A fenti okok mellett azért is esett az Axum-ra a választás, mert az elérhető keretrendszerek között is az egyik legrugalmasabb és bővíthetőbb. Mivel a fejlesztés elején nem tudhattam, hogy milyen problémával állok szemben, és megoldásához milyen eszközöket lesz szükséges használnom, a bővíthetőség igen fontos szempont volt.

3.4. MessagePack

A MessagePack[[msg23](#)] egy bináris adathordozó formátum, a JSON formátum egy alternatívája azon felhasználási esetekre, amikor a JSON túlságosan beszédesnek bizonyul. Szemben a JSON szöveges kódolásával, a MessagePack bináris kódolása előnyt jelent nagy adathalmazok szállításában, hiszen kevesebb memóriát foglal az elkészített üzenet.

Felhasználása az alkalmazásban, ahogy a fenti bekezdés is sugallja, azokra az esetekre szorult, amelyek nagy adatmennyiség közlését igényelték. Például, képadatok közelése sok kép számára JSON formátumban a kódolt képadatok bájtjainak egyes kódolásából állna számokként. Ez a szöveges tárolás az adatok bináris memóriaigényének sokszorosát jelentené.

A MessagePack, mivel célja az adathordozás (főleg hálózaton keresztül), a szerver és a kliens oldalon is használatba került.

3.5. React

A React[[rea23](#)] egy népszerű böngészőben használatos felhasználói felület keretrendszer. A korábbi megközelítésekhez képest alternatív ötletekkel teszi egyszerűvé webes felhasználói felületek elkészítését.

A React használatára a fő érv, hogy az elérhető, fejlett keretrendszerek közül a React-et ismerem leginkább, és tapasztalom szerint valóban könnyen megvalósíthatóvá tesz komplex felületek is.

4. fejezet

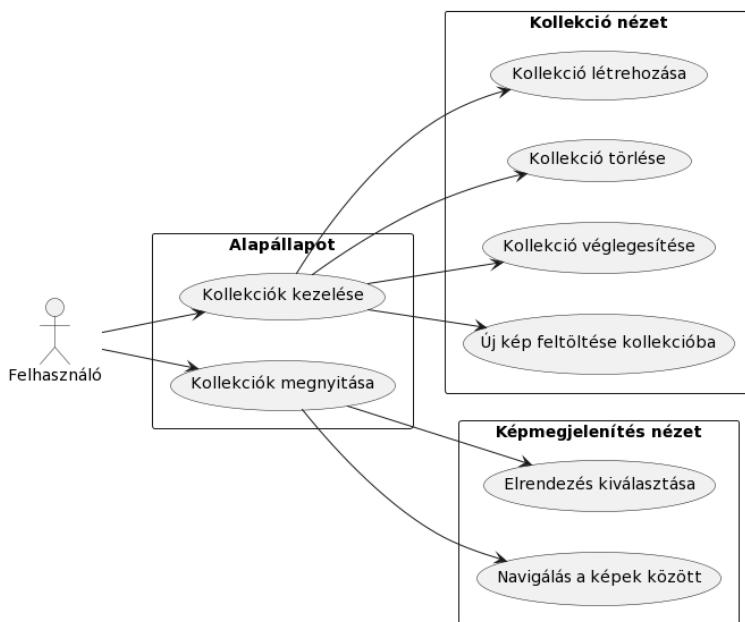
Rendszer specifikációja

4.1. Felhasználói követelmények

A rendszer bemutatását felhasználói szempontból eset-diagrammokkal fogom végezni. Ezek átláthatóvá teszik hogy különböző esetekben mire képes a felhasználó az alkalmazáson belül.

4.1.1. Általános eset diagramm

Az általános eset diagramm (lásd 4.1) felületesen vázolja fel a felhasználói eseteket, kategóriákra osztva azt. Az alábbiakban a felhasználói eseteket rész-esetekre osztom, amiből a további felhasználói eset fog leszűrődni.



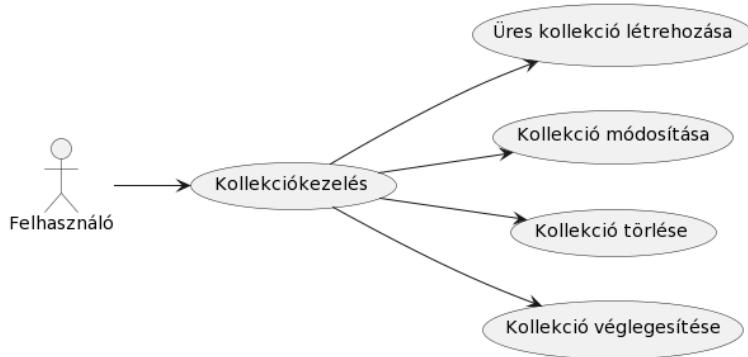
4.1. ábra. Az alkalmazás általános eset diagrammja

A 4.1 ábrán látható, hogy a felhasználó, belépve az alkalmazásba, képes kollekciókat kezelni (létrehozás, módosítás, törlés és véglegesítés), ill. kollekció megnyitás után navigálni képek között, opcionálisan kiválasztani képet, majd elrendezést kiválasztani, amely,

ahol szükséges, amely referencia képként tekint a kiválasztott képre, majd előállítja és megjeleníti az elrendezést.

4.1.2. Kollekciókezelés eset diagramm

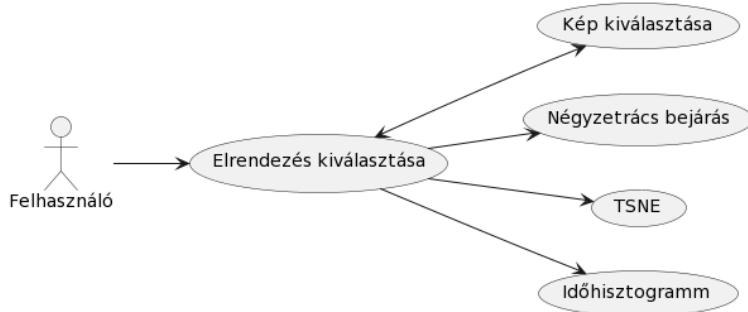
A kollekciókezelés eset diagramm (lásd 4.2) azt mutatja be, hogy a felhasználónak milyen képességei vannak a kollekciókezelésre.



4.2. ábra. Az alkalmazás kollekciókezzelés eset diagrammja

A 4.2 ábrán látszik, hogy a felhasználó képes kollekciókat létrehozni, törölni, módosítani és véglegesíteni. A módosítás ebben az esetben új kép feltöltését jelenti a kollekcióba. Új kép feltöltése által a kollekció bezárul, egyből nem tekinthető meg, hanem hamarabb véglegesíteni kell. A véglegesítés folyamata előfeldolgozási lépésekkel áll, amik nélkül nem lenne lehetséges a kollekciót megtekinteni, vagy rendezéseket lekérni róla.

4.1.3. Elrendezés eset diagramm

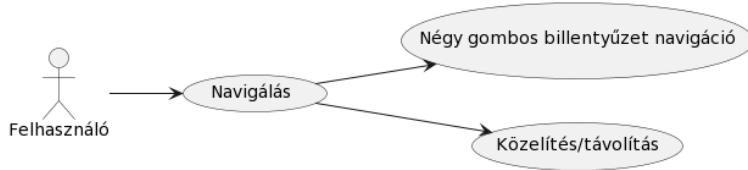


4.3. ábra. Az alkalmazás elrendezés eset diagrammja

A 4.3 ábrán látszik, hogy a felhasználó kiválaszthatja a kívánt rendezést, amelyben láttni szeretné a képeket. Esetenként ez egy minta-képet igényel, amelyhez képest lesz elrendezve a többi kép. Ezt a képek a felületen ki lehet választani.

4.1.4. Navigáció eset diagramm

A 4.4 ábra azt mutatja be, hogy a felhasználó megnyitva egy kollekciót navigálhat billentyűzettel a megjelenített képek között. Ebbe bele tartozik a rátárolás és távolítás is.



4.4. ábra. Az alkalmazás navigáció eset diagrammja

4.2. Rendszer követelmények

Ebben a fejezetben a rendszer követelményeit mutatom be, funkcionális és nem-funkcionális követelményekre lebontva.

4.2.1. Funkcionális követelmények

Az alábbiakban a rendszer funkcionális követelményeit sorolom fel. Funkcionális követelménynek sorolható minden képessége a rendszernek, amelyek valamely funkcionálisról felelősek.

Kliens

1. Kollekciókezelés
 - (a) Kollekciók létrehozása
 - (b) Kép feltöltése kollekcióba
 - (c) Kollekció véglegesítése
2. Kollekció megnyitása
3. Elrendezés kiválasztása
 - (a) Távolságfüggvény kiválasztása
 - (b) Összehasonlítási függvény kiválasztása
 - (c) Mintakép kiválasztása
 - (d) Elrendezés lekérdezése
4. Navigálás a képek között
 - (a) Kamera mozgatása billentyűzettel
 - (b) Nagyítás és kicsinyítés

Szerver

1. Feltöltött képek eltárolása eredeti és thumbnail méretekben (lásd [5.2.7](#))
2. Adatbázis nyílvántartás képekről és kollekciókról
3. Metaadatok kinyerése képekből és eltárolása adatbázisban (lásd [5.2.8](#))
4. Statikus atlasz (lásd [5.2.6](#)) elkészítése és felszolgálása
5. Elrendezések kiszámolása
 - (a) Távolságok kiszámolása képek között metaadatok alapján
 - (b) Képek sorrenbe rendezése távolságfüggvények vagy metaadatok alapján
 - (c) Elrendezések összeállítása

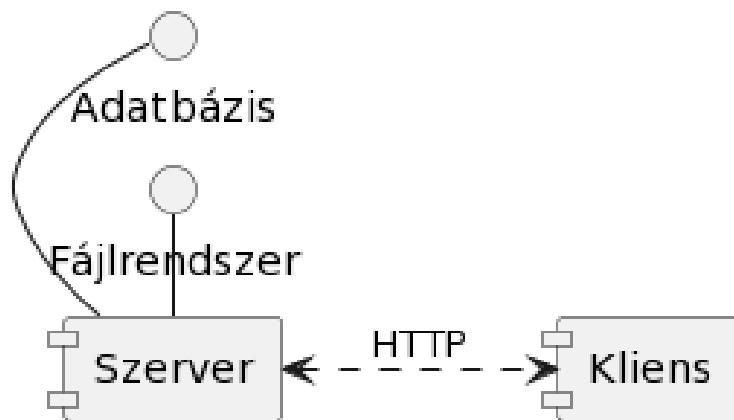
4.2.2. Nem-funkcionális követelmények

1. Git verziókövető
2. Szerver:
 - (a) Nagy mennyiségű tárhely a nagy mennyiségű kép eltárolásához
 - (b) Adatbázis hozzáférés
 - (c) Internet
3. Kliens
 - (a) Böngésző
 - (b) Asztali gép vagy laptop (billentyűzet, egér)

5. fejezet

Tervezés

5.1. Arhitektúra



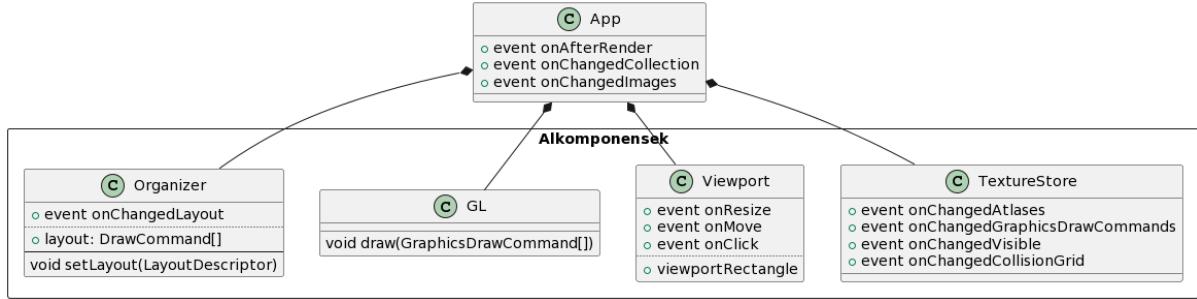
5.1. ábra. Az alkalmazás komponens diagrammja

A rendszer arhitektúráját a 5.1 ábrával mutatom be. Az alkalmazásnak két fő része van: egy szerver és egy kliens.

A szerver tárolja a képeket, és a futtatáshoz szükséges előfeldolgozási lépések eredményeit, ill. adatbázisban nyílván tartja a képeket, a kollekciókat, és a képek metaadatait.

A kliens böngészőben futó program, avagy weboldal, amely a szerverrel kommunikálva működik HTTP protokollon keresztül.

5.1.1. Kliens



5.2. ábra. A kliens osztály diagramma

A kliens oldali kód arhitektúrája osztályokra van osztva. A 5.2 ábra a kliensen használt osztályokat mutatja be.

App

Az *App* osztály a kezdeti inicializálásért felelős: létrehozza az alkomponenseket és összeköti őket, ahol szükséges eseménykezelőkkel. Az *App* felelős kollekciók megnyitása, bezárása, és a kollekciókban tárolt képek felületes metaadatainak letöltéséért. Amint ez megtörtént, induláskor, az *Organizer* értesül róla eseményen keresztül.

Organizer

Az *Organizer* azért felelős, hogy tárolja a jelenleg kiválasztott elrendezés leírását, és előállítsa vagy helyben, vagy kérésként a szervertől az elrendezést. Ennek típusa *DrawCommand*-ok listája. Egy *DrawCommand* (rajzolási parancs) a kirajzolni kívánt kép azonosítóját, és a rajzolás célját tárolja a virtuális térben (koordináták szerint ahova rajzolni szeretnénk). Amint ez előállt, értesül róla a *TextureStore*.

TextureStore

A *TextureStore* szerepe a korábban előállított elrendezés (*DrawCommand*-ok formájában) fogadása és átalakítása *GraphicsDrawCommand*-á. Egy *GraphicsDrawCommand* (grafikai rajzolási parancs) a *DrawCommand* bővítménye. A korábbiak mellett tartalmazza hogy milyen, videokártyán tárolt textúrából szeretnénk melyik részt kirajzolni. Az utóbbi rész fontos, hiszen textúra atlaszokat (5.2.3) használunk a rajzoláshoz, amelyek a videokártya szempontjából egyetlen textúra, viszont sok képet tartalmaznak.

Amint a *GraphicsDrawCommand*-ok előálltak, értesül róluk az *App*, amely eltárolja a legújabb grafikai rajzoló parancsokat, és ezeket használja fel a következő rajzoláshoz.

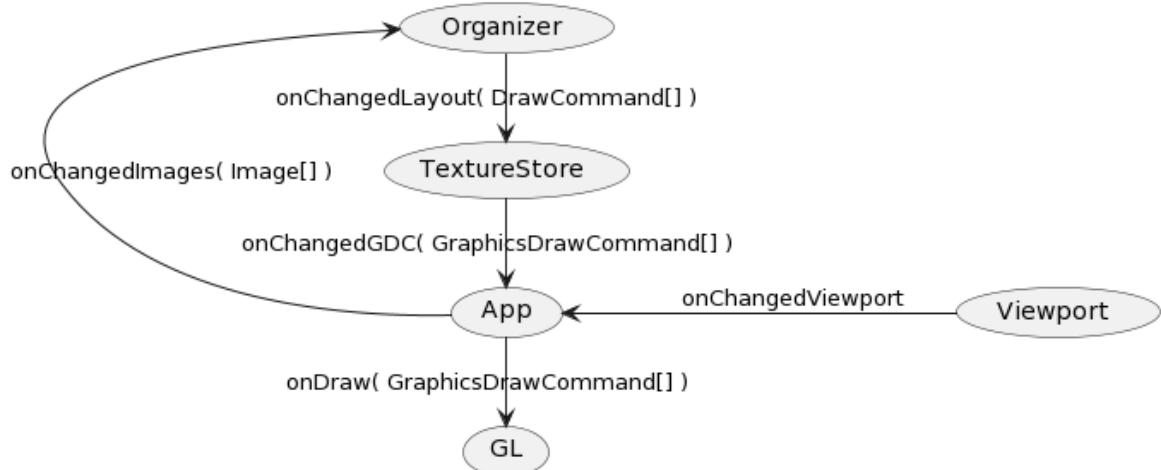
GL

A *GL* szerepe a rajzolás. Egyetlen metódusa van és nincs belső állapota vagy eseményei, így az *App* hívja meg a *draw* metódusát mikor el szeretné indítani a rajzolást.

Viewport

A *Viewport* szerepe követni a kamera jelenlegi pozíóját, ill módosítani azt annak esetén, hogy a felhasználó releváns billentyűt nyom le, vagy egérkattintást végez, vagy átméreteződik az ablak, amikor meg a kamera is át kell méreteződjön.

A korábban felsorolt interakciókat a kliens különböző komponensei között a 5.3 ábra mutatja be.



5.3. ábra. A kliens aktivitás diagramma

5.1.2. Szerver

A szerver arhitektúrája könnyen leírható, nem objektum-orientált megközelítéssel készült, így nem írható le osztály-diagrammal.

Végpontok

A szerver HTTP-n keresztül kommunikál a kliensel (lásd 5.1). E miatt a szerveren végpontok vannak meghatározva, amelyek HTTP kéréseket fogadva végzik el funkcióikat. Ezek a következőek:

1. GET `/collections`
Összes kollekció lekérése
2. POST `/collections`
Kollekció létrehozás
3. POST `/:collection_id`
Kollekcióból levő összes kép lekérése
4. POST `/:collection_id`
Kollekcióból levő összes kép lekérése

5. GET `/:id/metadata`
Kollekcióban levő képek lekérése metaadataikkal együtt
6. POST `/:id/upload`
Kép feltöltése kollekcióba
7. POST `/:id/finalize`
Kollekció véglegesítése
8. POST `/:id/bulk`
Sok kép thumbnail méretben való lekérése
9. GET `/:id/atlas`
Kollekcióhoz tartozó statikus atlasz (lásd [5.2.6](#)) lekérése
10. POST `/:id/layout`
Elrendezés lekérése

5.2. Kísérletek

Az alkalmazás végleges állapota számos arhitektúrális lehetőség kipróbálásának eredménye. Az alábbiakban felvázolom a próbálkozásokat és eredményeket.

5.2.1. DOM

Az ú.n. Document Object Model (a továbbiakban DOM) a böngészőkben használt híd-könyvtár JavaScript és a betöltött HTML oldal között. Lehetőséget ad törölni, módosítani, és létrehozni DOM csomópontokat (HTML elemeket).

Az első, és legsikertlenebb próbálkozás abból állt, hogy rajzolási lépésként elrendezésnek megfelelően *img* (kép) elemeket helyeztem el a weboldalon. A töltési idő meghaladta a türelmem ($>60\text{mp}$) már 10000 kép esetén is, majd elvárásaimnak megfelelően fel is emészttette a számítógépem összes elérhető memóriáját, és ez után a böngészőm össze is omlott.

Számomra az utóbbiak miatt egyértelművé vált, hogy ez nem egy járható út.

5.2.2. WebGL

A WebGL használata sokkal sikeresebb volt, viszont további arhitekturális döntésekre volt szükség ahhoz, hogy eljuthassak a mai teljesítményre. A WebGL, mivel hardveres gyorsítást használ, összetettebb arhitektúrát igényel. A WebGL-t kezelő szoftvernek dekódolnia kell képeket nyers pixel-bájtakra, ezeket fel kell töltenie a videókártyára, elrendezésnek megfelelően puffereket kell feltöltenie a videókártya memóriájában kirajzolási adatokkal, és ezt mind úgy tennie, hogy optimális idő alatt fusson le az alkalmazás.

Számos tervezési döntés született meg kizárálag annak érdekében, hogy az alkalmazás jól teljesítsen WebGL rajzolási felületen. E miatt az alkalmazásban a hardveres rajzolás központi részévé vált.

5.2.3. Textúra kezelés

A WebGL, az OpenGL-hez hasonlóan textúra kirajzolását csupán abban az esetben tudja megvalósítani hogyha a textúra a videókártyára van feltöltve. Továbbá, normális esetben minden textúrához külön *rajzoló hívás* szükséges. Számítógépes grafikában ismert tény, hogy a hatékony rajzolás a processzor és a videokártya közötti kommunikáció minimalizálásából áll. Ez esetünkben leginkább a *rajzoló hívásokra* érvényes.

Ennek orvosolására *textúra atlaszok*[atl23] használata tanácsolt, ahol lehetséges.

Esetünkben a textúra atlaszok használata teljes mértékben lehetséges, szóval ebbe az irányba fejlesztettem tovább, viszont itt felmerül a következő kérdés: Hol állítsuk elő az atlaszt?

5.2.4. Dinamikus atlasz a szerveren

Első és legegyszerűbb ötletként a szerver állította elő az atlaszokat minden lekérdezéskor. Mivel WebGL-ben a textúrák határolva vannak méretük szerint, ezért egyébként is sok atlaszt kell kezelnünk. Így, sok lekérdezés keretei közt tett szert a kliens az összes textúrára.

Ahogyan egyre több textúrát próbáltam átküldeni ezzel a módszerrel, problémába ütköztem. Először is a képek kicsinyített változatait küldtem át, mivel a kollekció teljes mérete több gigabájt volt, viszont a kicsinyítéseket lekérdezés-időben hajtottam végre szerver-oldalon.

Ezzel a megoldással minden lekérdezés esetén minden képet ki kellett csomagolnia a szervernek JPEG-ről nyers pixel bájtokra. Ezeket le kellett méreteznie, bele kellett vágnia az atlaszba egy adott helyen, majd az atlaszt le kellett kódolnia JPEG formátumban, amit végsősoron a kliensnek újra ki kellett csomagolnia. Ez a folyamat 57-ezer képre több mint 15 másodpercig tartott.

5.2.5. Résztextúra feltöltés

A WebGL egy képessége egy, a videókártyán létrehozott adott méretű textúrának részleges feltöltése (`texSubImage2D()` metódus). Ezt felhasználva egyes képeket tudtam feltölteni az atlasz textúrára. Ennek köszönhetően a szerver át tud küldeni egy listányi kép- adatot eredeti JPEG bájtfájlban, amit a kliens sorra feltölt egy általa készített atlaszra, és ennek köszönhetően minden képet csak egyszer kell dekódolni az egész folyamat során.

5.2.6. Statikus atlasz

Az eddigi megoldás remek teljesítményt ért el, viszont további optimalizációk rejlettek egy kollekció első betöltésére vonatkozóan egy olyan atlasz elkódolásában, amely tartalmazza a kollekció összes képét parányi méretben.

Ezt egy kollekcióhoz tartozó *statikus atlasznak* neveztem el. Ez az atlasz egy kollekció létrehozásakor jön létre, és akkor el is lesz kódolva JPEG formátumban. A kliens csupán letölti a szerverről, feltölti a videókártyára, és már ki is rajzolhatjuk a kezdeti elrendezést.

5.2.7. Textúrák eltárolása

Mivel a textúrák atlaszba foglalása futási időben történt a *Dinamikus atlasz* megközelítéssel (5.2.4), a textúrák leméretezésével együtt, lehetőség volt optimalizálásra a *thumbnail* méretű textúrák lementésével előfeldolgozási lépésként.

Ezt az egyes textúrák feltöltésekor teszi meg a szerver, textúra azonoítóhoz és mérethez kötve. Több thumbnail méretet mentünk le minden textúrához. Egy mini méret (30x30 pixel) szükséges a Statikus Atlasz (5.2.6) előállításához, de más méretekkel is lementünk hogy ráközelítés esetén, a látható textúramérettől függően, ne egyből nagy méretű képeket töltön le a kliens, hanem thumbnail méretekkel a gyorsaság érdekében.

5.2.8. Metaadatok tárolása

Több féle metaadatot nyerünk ki a képekből. Egyszer általános exif metaadatokat, de színpalettát is, amely a teljes textúra feldolgozását igényli. E miatt a metaadatok kinyerése nagy kollekció esetén hosszas folyamat lehet. A rendezések távolságfüggvényei arra számítanak, hogy a metaadatok gyorsan kézbe kerülnek, viszont hogyha ekkor próbáljuk kinyerni őket, a rendezés is lassú lesz.

E miatt tároljuk el őket adatbázisban, viszont a helyett hogy minden metaadat számára új oszlopot biztosítanánk, ami nem egy rugalmas megoldás, a Postgres adatbázis JSONB típusát használjuk, ami indexelhető, és ennek köszönhetően kötetlen objumként tudjuk lementeni az egyes képek metaadatait és ugyanakkor gyorsan tudjuk előhívni őket.

5.2.9. Elrendezések előállítása

Mivel úgy döntöttem, hogy nem fogok könyvtárat felhasználni a metaadatok kinyeréséhez vagy az elrendezések előállításához, ezért kérdéses volt, hogy ezek miként történjenek. Az elrendezések kiszámolását a lehető leggyorsabban szerettem volna végezni, és erre lehetőséget adott a Rust programozónyelv.

Történt egy kísérlet a kliens oldalon is elrendezések kiszámításához, viszont lassúnak bizonyult komplexebb algoritmusok, illetve nagyobb kollekcióméretek esetén.

Felület	Betöltési idő (ms)
Szerver	160
Kliens	250

5.1. táblázat. Felület függvényében a "négyzetrács bejárás" kiszámításának ideje

A 5.1 táblázaton láthatóak ehhez a kísérlethez fűződő eredményeim. A kísérlethez használt elrendezés a "négyzetrács bejárás" (lásd 6.2.1), amely közepes komputacionális igényekkel bír, így egyenlő esélyeket ad mind a szervernek, mind a kliensnek.

A kísérlet negyvenezer képen történt, ahol a rendezés kulcsa egy-egy szám volt (JavaScript number típus), amelyet mind a szerver, mind a kliens valós időben számol ki a képek metaadataiból, azon belül a képek színpalettájából (domináns színek halmaza), amely egy három elemű vektorok listája. Az összegző algoritmus, amely két kép színpalettája alapján számként fejezi ki a köztük levő távolságot lineáris különbséget számol az

összes szín összes színcsatornája esetén, majd ezeket összeadják. A képek között távolságokat előre kiszámolni és eltárolás után felhasználni nincs lehetőség, hiszen már negyvenezer képnél ez több mint 2GiB memóriát igényel.

A leszűrt következtetésem az, hogy szerver-oldalon kiszámítani igényesebb elrendezések gyorsabb lesz még úgy is, hogy a kiszámított elrendezést HTTP válaszként vissza kell küldeni a kliensnek, ami valamennyivel mindenképp lassítja a folyamatot.

5.2.10. TSNE [[tsn23](#)]

Az egyik központi ötlet az alkalmazás fejlesztése során annak érdekében hogy képkollekciókból minták emelkedjenek ki az *klaszterezés* volt. A klaszterezés célja egy homogén adattömb csoportosítása oly módon, hogy az adathalmazban bújkáló minták kiemelkedjenek, hangsúlyossá váljanak. Továbbá, a térbeni klaszterezés az egyes adatpontokhoz térbeli pontokat csatol, amely meghatározzák az elhelyezkedését az adathalmazban. Ez az elhelyezés jó klaszterezés esetén minden adatponthoz olyan térbeni pontot csatol, amelynek a többihez képesti távolsága tükrözi a többi adatponttal való hasonlóságát. Más szavakkal, jó klaszterezés esetén hasonló adatpontok térbeni pontjai egymáshoz közel fognak elhelyezkedni, míg egymástól távoliak térben is távol.

A TSNE (t-distributed stochastic neighbor embedding) egy klaszterező algoritmus, amely nem tökéletes (mivel sztohasztikus, véletlenszerűségre alapszik), de annál érdekesebb vizualizációkat képes előállítani.

Nagy a komputacionális igénye, viszont minőségi klaszterezést állít elő.

Viszont, mivel memóriaigénye szintén magas, az eredeti TSNE klaszterező algoritmust nem használhatom. Egy projektben, ahol többszázezer kép megjelenítése a cél, a TSNE már negyvenezer képnél közel 100 GB memóriát szeretett volna lefoglalni, amihez egy átlag számítógépnek nincs hozzáférése.

Barnes-Hut TSNE [[vdM14](#)]

A hagyományos TSNE implementáció erőforrásigényei miatt, egy párhuzamosított implementációját használtam a Barnes-Hut TSNE megközelítésnek ([\[vdM14\]](#)), amely egy θ paraméterrel határozza meg a megközelítés valósághűségét. Ezt leginkább a [7.3](#) táblázatban láthatjuk, ahol egy $\theta = 0.01$ minőségi eredményeket ad, viszont sok ideig fut, míg $\theta = 0.5$ -el az algoritmus sokkal hamarabb végez ugyanannyi kép klaszterezésével.

WASM [[was23](#)]

A WASM (WebAssembly) egy kísérleti fázisban levő tehnológia, mely lehetővé tesz egy, a natív kódhoz közeli környezetben futtatható kódot betölteni egy weboldalra. Célja kizártani a JavaScript dinamicitásának köszönhető kiszámíthatatlanságot, ezzel gyorsítva a weboldalakon futó kódot.

Egy említésremélő ötlet volt a fejlesztés során WebAssembly modulokat használni elrendezések kiszámítására, amely egy *arany középút* lett volna a kliens oldali és a szerver-oldali számítás között, mivel jó teljesítményt kölcsönözött volna a WebAssembly tehnológiától, és ugyanakkor nem lett volna szükség szerver-oldalú kódra új elrendezés bevezetéséhez.

Ez az ötlet idő híján el lett vettve, mivel a fejlesztési idő is prioritássá vált, így a már elérhető szerver-oldalú kódra került a funkció.

Ennek ellenére, a WASM ezen felhasználása egy jövőbeli kísérlet számára ígéretes ötletnek tűnik.

6. fejezet

Kivitelezés

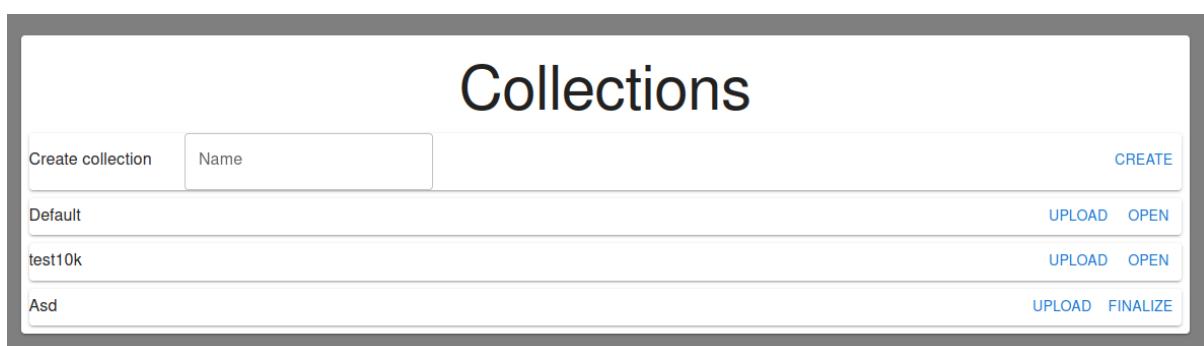
A *kivitelezés* fejezetet annak szentelem, hogy bemutassam az elkészített alkalmazást különböző szempontkból. Egyszerűen, be fogom mutatni az alkalmazás működését felhasználói nézetről (azaz, a felhasználói felületet), másrészről, be fogom mutatni az alkalmazást technikai szempontból.

6.1. Felhasználói felület

A felhasználói felület egy alkalmazás azon része, amellyel a felhasználó kizárolagos interakcióba kerül. Ez alapján, felhasználói szempontból, a felület az, ami meghatározza, mire képes a felhasználó, mire nem, és hogy az egyes funkciókat milyen könnyedséggel éri el.

E miatt fontosnak tartom bemutatni az alkalmazás felhasználói felületét mint a szoftver szerves és egyik legfontosabb része. Továbbá, ez a fejezet használati szempontból dokumentatív értékű.

6.1.1. Kölcsönöskezelés



6.1. ábra. Felhasználói felület kölcsönöskezelésre

Az alkalmazás elindításakor, a felhasználó a kölcsönöskezelési menüvel találkozik, amely kölcsönök létrehozására, meglévő kölcsönökba való új képek feltöltésére, frissen változtatott vagy létrehozott kölcsönök véglegesítésére és már véglegesített kölcsönök megnyitására szolgál.

A 6.1 ábrán látható, hogy egy tömör menürendszerből minden végrehajtható. A menü frissül bármely változtatás kivitelezésekor. Megnyitva egy kollekciót, a következő nézetben kerülünk, amely megjeleníti a kollekcióban tárolt képeket.

Továbbá, a következő nézetben hozzáférésünk van irányítani és paraméterezni a megtekintésre szánt elrendezést.

6.1.2. Elrendezések kezelése

A fő alkalmazásnézetben lehetőségünk nyílik beállítani a kívánt elrendezést, és felparaméterezni azt. A 6.2 ábrán láthatóak fő menüpontok, amelyek ezt lehetővé teszik. A menüpontok kategóriákra bontják az elrendezés beállításait, és e szerint léteznek a megjelenítésben felhasznált képek szűrésére vonatkozó beállítások, fő elrendezés-beállítások, ill. képek kiválasztására vonatkozó beállítások. Az utóbbiak annyira terjednek ki, hogy egyes elrendezések elvárnak egy referenciaképet, mint paraméter. Ezt vizuálisan tudjuk kiválasztani az egeret használva.



6.2. ábra. Felhasználói felület elrendezés beállítására

Szűrés

A szűrés beállítások arra szolgálnak, hogy a megjelenítéshez lekérdezett képeknek valamely feltételeket szabunk, amelyek teljesítésének fejében fognak megjelenni az elrendezésben.

Lehetőségünk van szűrni képek között az alapján, hogy az adott kép rendelkezik-e egy bizonyos metaadat-taggal (lásd 6.4), és az alapján, hogy összesen hány képre szeretnénk megjeleníteni az elrendezést.

Filter

Has metadata
palette

SUBMIT

6.3. ábra. Képek közötti szűrés

Filter

Has metadata
palette

palette

date_time

6.4. ábra. Szűrés metaadatok elérhetősége alapján

Elrendezés paraméterek

A fő felhasználói felületi rész arra vonatkozik, hogy milyen elrendezést választunk ki a képek megtekintésére, és milyen paramétereket választunk számára. A 6.5 a négyzetträcs bejárás elrendezés paramétereit mutatja. A 6.6 ábrán láthatóak az elérhető elrendezések.

Ennek az elrendezésnek különösen sok paramétere van. Ezek egy későbbi fejezetben kerülnek bemutatásra.

Layout

Layout
Grid Expansion

Compare Function
Signed Distance

Distance Function
Palette

Anchor
Center

Grid Distance
Pythagorean

SUBMIT

6.5. ábra. Elrendezés felparaméterezésére

Layout

Layout
Grid Expansion

Grid
Grid Expansion
Time Hist
Tsne
Center

Grid Distance
Pythagorean

SUBMIT

6.6. ábra. Elrendezés kiválasztására

Layout

Layout ─ Grid Expansion ─

Compare Function ─

Signed Distance ─

Signed Distance
Comparative Distance
Center
Grid Distance
Pythagorean

SUBMIT

Layout

Layout ─ Grid Expansion ─

Compare Function ─

Signed Distance ─

Distance Function ─

Palette ─

Palette
Palette Cos
DateTime

SUBMIT

6.7. ábra. Összehasonlítási függvény kiválasztása

6.8. ábra. Távolságfüggvény kiválasztása

A 6.7 ábrán látható az *összehasonlítási függvény* kiválasztásához társuló felhasználói felület, ill a 6.8 ábrán látható a távolság- függvény kiválasztásáról szóló felhasználói felület. Ezek kiválasztása minden metaadatuktól függő elrendezés számára fontos, hiszen ezek határozzák meg sorra annak módját, hogy képeket hogyan rendezük sorba, illetve hogyan hasonlítunk össze az adott elrendezés keretei közt.

Az ábrákon látható, hogy több opciónk van ezek között válogatni. A választék itt azért fontos, mert ahány kombináció van a paraméterek lehetséges értékei között, annyi egyedi vizualizációt képes előállítani az alkalmazás minden kollekcióra.

Szelekció

Az alkalmazás egyelőre csak egy kép kiválasztását teszi lehetővé. Mivel nincsenek arra vonatkozó funkciók hogy több kép alapján rendezzünk vagy több képek kezeljünk, ezért több kép kiválasztására nincs is szükség.

A *Szelekció* menü egyetlen funkciója jelezni a kiválasztott kép azonosítóját, hogyha van kiválasztott kép, másképp pedig jelezni ennek hiányát. A 6.9 és 6.10 ábrák sorra annak megjelenítését ábrázolják, hogy van, illetve nincsen kiválasztott kép.

Select	b533c1ca-8cbc-457d-a85d-21aeaa37ed0d
--------	--------------------------------------

6.9. ábra. Kiválasztott kép azonosítója

Select	None
--------	------

6.10. ábra. Nincs kiválasztott kép

Referenciakép kiválasztása

Layout	Layout
Grid Expansion	Grid Expansion
Compare Function	Compare Function
Comparative Distance	Comparative Distance
Selected	Selected
e2cf68d5-aa0a-42e6-9980-5feb1f042	None
Distance Function	Distance Function
Palette	Palette
Anchor	Anchor
Center	Center
Grid Distance	Grid Distance
Pythagorean	Pythagorean
SUBMIT	

6.11. ábra. Kiválasztott kép azonosítója

Layout	Layout
Grid Expansion	Grid Expansion
Compare Function	Compare Function
Comparative Distance	Comparative Distance
Selected	Selected
None	None
Distance Function	Distance Function
Palette	Palette
Anchor	Anchor
Center	Center
Grid Distance	Grid Distance
Pythagorean	Pythagorean
SUBMIT	

6.12. ábra. Nincs kiválasztott kép

Egyes elrendezések, mint a *négyzetrács bezárás* azt igényelik, hogy valamilyen eljárás alapján sorrendbe tudjuk rendezni a képeket. Ennek egyik módszere, felhasználva egy képekre meghatározott távolságfüggvényt, hogy az alapján rendezzük sorrendbe a képeket, hogy egy adott referenciaképtől melyik van kisebb távolságra.

Ehhez is szükséges felhasználói felület, hiszen a felhasználónak tudnia kell, hogy kiválasztott-e képet. A 6.11 és 6.12 ábrák azt mutatják be, hogy ebben az esetben mit mutat a felhasználói felület.

Mind funkciójában, mind működésben hasonlít ez az elem a szelekció menüben látottakra, hiszen egérgombnyomásra visszajelzést ad a kiválasztott képről, vagy éppen annak hiányáról.

6.2. Elrendezések

Ebben a fejezetben bemutatom az elérhető elrendezéseket és paramétereiket, illetve tulajdonságaikat.

6.2.1. Négyzetrács

A négyzetrács elrendezés fotók számára az egyik legtermészetesebb, hiszen esztétikus és rendezett módon mutatja be az egész képkollekción, és ugyanakkor elég rugalmas közvetíeni hasonlóságokat és különbségeket képek között csupán az elhelyezkedésük alapján.

Továbbá, a négyzetrács az egyik leggyakrabban használt megjelenítése képeknél. Például, mind a gyakran használt Google Fotók, mind a Windows Fájlok (a Windows operációs rendszer fájlbüngészője) elérhetővé teszi, gyakran alapértelmezett nézetként olyan mappákban, ahol sok a fotó.

Egyszerű négyzetrács

A korábbiak (lásd 6.2.1) szerint az alapértelmezett megjelenítés az alkalmazásomban szintén a négyzetrács egy változata, ahol az egyes képek véletlenszerű elhelyezést kapnak a rácson belül. A 6.13 és 6.14 ábrák az egyszerű négyzetrács elrendezést mutatják be alapállapot, ill. ráközelítve.



6.13. ábra. Egyszerű négyzetrács



6.14. ábra. Egyszerű négyzetrács (ráközelítve)

Négyzetrács bejárás

A *négyzetrács bejárás* nem klaszterezés, viszont ahhoz hasonló tulajdonságokat mutat. Mivel nem teljes klaszterezésről van szó, hanem négyzetrácsban való elhelyezésről, ezért szemben a TSNE-vel (lásd 5.2.10), egyrészt nem kivehetők tisztán klaszterek, hiszen négyzetráccson belül ezt nehéz arányosan kifejezni, de másrészt, a hasonló adatok egymás mellé kerülnek. Ugyanakkor, szintén a TSNE-vel szemben, a *négyzetrács bejárás* négyzetráccsal megjelenítésének hála átlátható vizualizációkat ad, mivel képek nem csúsznak egymásra, vagy jelennek meg aránytanál nagy vagy kis méretben.

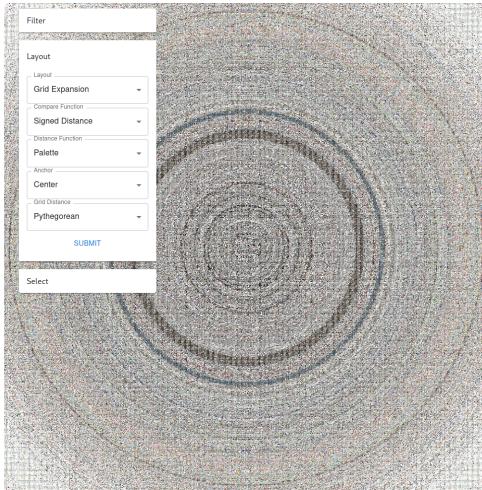
Működése abból áll, hogy sorrendbe helyezzük képeket egy paraméterként megadott távolságfüggvény alapján, és az így kapott sorrend szerint helyezzük el a képeket a négyzetrácsban úgy, hogy a négyzetrácsot egy adott origótól vett távolság szerint járjuk be.

Az így kapott elrendezés leginkább a referenciaiképre (hogyha kiválasztottunk egyet) vonatkozó hasonlóságokat és különbségeket mutatja, viszont az egymáshoz hasonló ké-

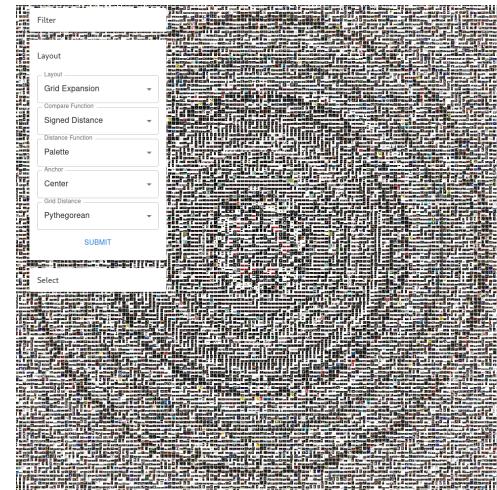
pek, mivel hasonló távolságra vannak a referenciaiképtől, hasonló távolságra lesznek az elrendezésben is az origótól.

Ennek a tulajdonságnak köszönhetően, a "klaszterezés" egy dimenzióban történik; az origótól távolabb haladva gyűrűket látunk kialakulni azokból a képekből, amelyek egymáshoz hasonlóak, avagy hasonlóan távol állnak a referenciaiképtől.

A 6.15 és 6.16 ábrákon látható, hogy valóban létrejönnek a gyűrűk, amelyekben ez esetben olyan képek vannak, amelyek domináns színeik szerint hasonlítanak.



6.15. ábra. Négyzetrács bejárás

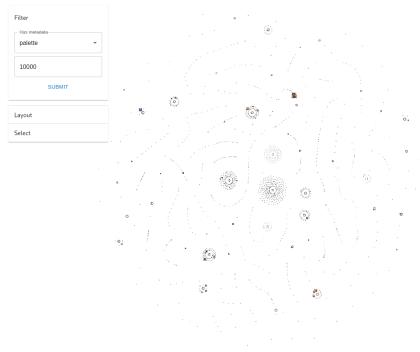


6.16. ábra. Négyzetrács bejárás (ráközelítve)

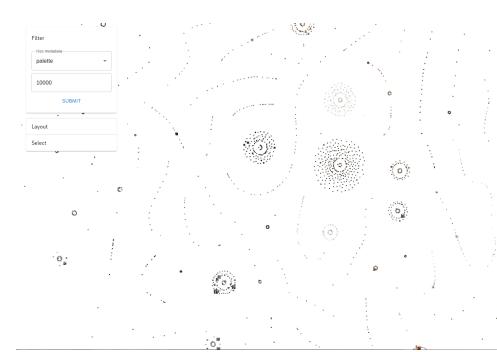
6.2.2. TSNE

A 6.17 és 6.18 ábrák egy domináns szín alapú TSNE elrendezést mutatnak be. Látható, hogy a klaszterezés valódi az egymáshoz tömörülő képek miatt. Mivel domináns szín alapján történik a klaszterezés, szabad szemmel is kivehetőek a klaszterek.

Az ábrák elsősorban arra voltak készítve hogy első látásra is kivehetőek legyenek a klaszterek, viszont kiemelik a TSNE klaszterező képességét, hiszen bármely más távolságfüggvény alapján is működtethető a TSNE.

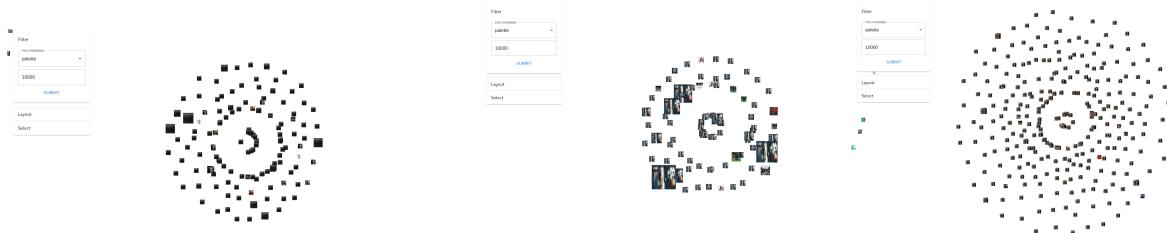


6.17. ábra. TSNE elrendezés



6.18. ábra. TSNE elrendezés (ráközelítve)

A 6.19 ábra a 6.17 ábrán látható TSNE klaszterezésen látható egyes klaszterekre közelít rá, bemutatva a csoportosítást.

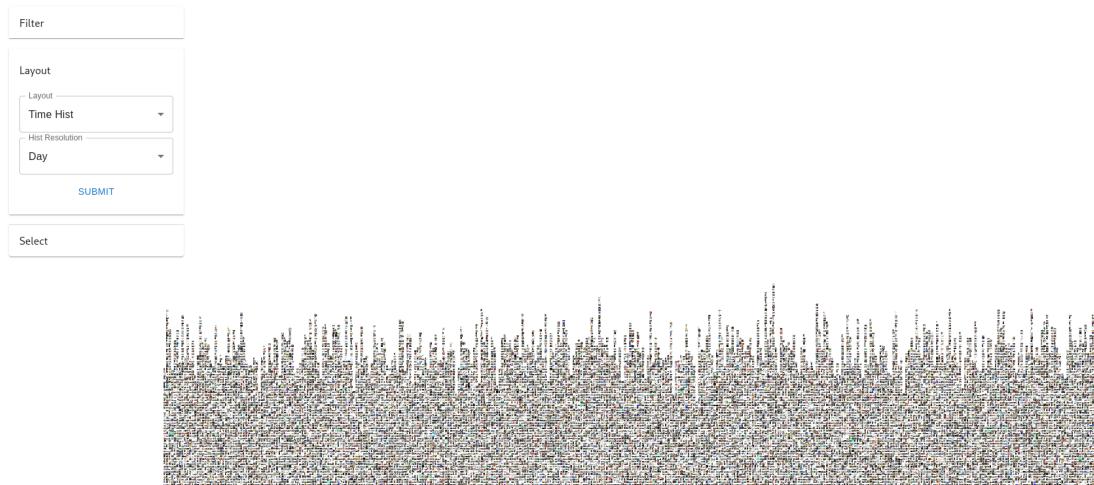


6.19. ábra. Példa klaszterek

6.2.3. Időhisztogram

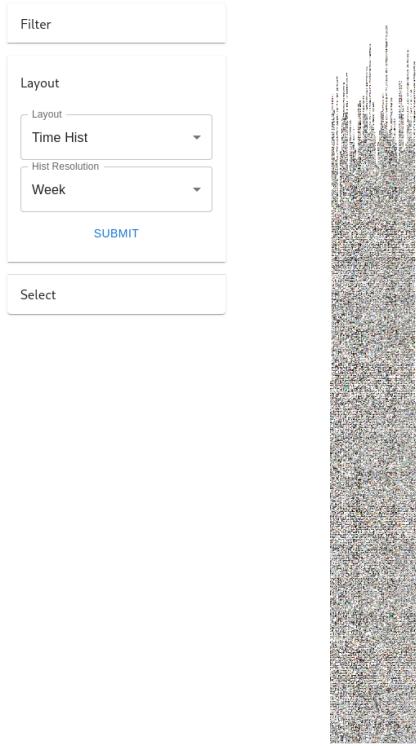
Egy alapvető tulajdonsága fotókat tároló képeknek, hogy gyakran tartalmaznak időbélyeget. Ezt felhasználva, egy elérhető rendezés az alkalmazásban az időhisztogram. Általa betekintést nyerhetünk abba, hogy az egyes képek mikor voltak készítve más képekhez képest, ill hogy hány képet tartalmaz a kollekció időbényeg szerint egy adott órába/napon/héten/hónapban/évben (a felbontás kiválasztható).

A [6.20](#) ábra egy időhisztogram elrendezést mutat be kb. negyvenezer képről (az átláthatóság érdekében) az év napjai szerint csoportosítva.



6.20. ábra. Időhisztogram az év napjai szerint (365 oszlop)

A [6.21](#) ábra egy időhisztogram elrendezést mutat be az év hetei szerint. Mint látható, kb. negyvenezer kép esetén nehezen kivehetőek az oszlopok, hiszen minden hétre, az 52 oszlopra, átlagban 770 kép jut.



6.21. ábra. Időhisztogram az év hetei szerint (52 oszlop)

7. fejezet

Mérések

7.1. Kollekció véglegesítése

Mivel a kollekció véglegesítése előfeldolgozási lépéseket végez el, annak érdekében hogy a továbbiakban gyorsan történjen az alkalmazás használata, a véglegesítés nem különösen gyors. A 7.1 táblázat a kollekció véglegesítési idejét mutatja a kollekció által tartalmazott képek számának függvényében.

Képek száma	Véglegesítés (p)
10.000	1:20
50.000	5:13
100.000	9:55
250.000	26:43

7.1. táblázat. Képek számának függvényében kollekció véglegesítési idő percben

7.2. Betöltési idő

Az utóbbiaknak (lásd 5) köszönhetően az alkalmazás betöltése elég gyors. A 7.2 táblázaton látható az egyes kollekcióméretekre a betöltési idő.

Képek száma	Betöltési idő (mp)
10.000	0.46
50.000	2.35
100.000	5.49
250.000	10.13

7.2. táblázat. Képek számának függvényében az alkalmazás betöltési ideje

7.3. Elrendezések előállítása

Az elrendezések előállítása az egyik fő folyamat az alkalmazásban, és ezért amennyire csak lehetett, optimalizálva van. Erre kivétel a TSNE (lásd 5.2.10), amely fölött nincs befolyásom, csak a θ paraméter által, ami klaszterezési minőséget áldoz fel teljesítményért cserében.

Képek száma	Időhisztogram	Négyzetrács bejárás	TSNE ($\theta = 0.01$)	TSNE ($\theta = 0.5$)
10.000	1.09mp	1.2mp	9:27.6	1:18
50.000	0.96mp	1.01mp	-	7:08
100.000	1.4mp	1.4mp	-	-
250.000	1.7mp	1.8mp	-	-

7.3. táblázat. Képek számának függvényében elrendezések előállítása

8. fejezet

Összefoglaló

A projekt forráskódja elérhető GitHub-on

1. Kliens: <https://github.com/EmmChriss/megallery-frontend>
2. Szerver: <https://github.com/EmmChriss/megallery-backend>

A projekt nagyrészt sikeresnek tekinthető. A nagy számú képmegjelenítés minden-képp teljesült, hiszen többszázezer képet meg lehet jeleníteni másodpercek alatt. Ahhoz, hogy erre képes legyen az alkalmazás, számos fejlesztési cikluson keresztül ment. E miatt kevés időm maradt kidolgozni a másodlagos célkitűzést. A sok féle elrendezés és sok féle leíró-vektor és metaadat kinyerése képekből.

Sikerült kiépíteni egy keretrendszeret, amely sebességet garantál bármely szerver- oldalon készített elrendezésnek, ill távolságfüggvénynek. Viszont, kevésnek tartom az elérhető elrendezések és távolságfüggvények számát.

8.1. Továbbfejlesztési lehetőségek

A továbbiakban az alkalmazásnak leginkább leíróvektorok (metaadatok), elrendezések és távolságfüggvények számában és kidolgozottságában van fejlődnivalója. Ebben komoly alapot adhat a LIRE (lásd 2.3.1). A korábbi részben felvázoltak alapján a LIRE több okból nem volt használva a projektben, viszont a tudástár, amit a forráskód tárol képekből való leíróvektorok kinyeréséről felhasználható az alkalmazásom továbbfejlesztéséhez. Sőt, akár egy könyvtár is készíthető belőle, amelyet mások is fel tudnak használni.

Más továbbfejlesztési lehetőség, hogy a felületet még felhasználó barátabbá tenni: például, több paramétert elérhetővé tenni a felhasználónak, vagy akár készíteni egy leíró nyelvet elrendezések számára, hogy a szerver tudathassa a kliensekkel, hogy milyen elrendezések elérhetők, azokon milyen paraméterek állíthatóak, és a paramétereknek milyen értékek adhatóak meg.

Egy további fejlesztési lehetőséget jelentene a WASM (lásd 5.2.10) felhasználása komputacionálisan intenzívebb kliens-oldali feladatokra. Ilyen lenne a textúrák kezelése, MessagePack (3.4) üzenetek kicsomagolása, vagy akár elrendezések előállítása.

Ábrák jegyzéke

2.1. Két példa az ImagePlot féle képvizualizációról	13
4.1. Az alkalmazás általános eset diagramma	17
4.2. Az alkalmazás kollekciókezzelés eset diagramma	18
4.3. Az alkalmazás elrendezés eset diagramma	18
4.4. Az alkalmazás navigáció eset diagramma	19
5.1. Az alkalmazás komponens diagramma	21
5.2. A kliens osztály diagramma	22
5.3. A kliens aktivitás diagramma	23
6.1. Felhasználói felület kollekciókezelésre	29
6.2. Felhasználói felület elrendezés beállítására	30
6.3. Képek közötti szűrés	31
6.4. Szűrés metaadatok elérhetősége alapján	31
6.5. Elrendezés felparaméterezésére	31
6.6. Elrendezés kiválasztására	31
6.7. Összehasonlítási függvény kiválasztása	32
6.8. Távolságfüggvény kiválasztása	32
6.9. Kiválasztott kép azonosítója	33
6.10. Nincs kiválasztott kép	33
6.11. Kiválasztott kép azonosítója	33
6.12. Nincs kiválasztott kép	33
6.13. Egyszerű négyzettrács	34
6.14. Egyszerű négyzettrács (ráközelítve)	34
6.15. Négyzettrács bejárás	35
6.16. Négyzettrács bejárás (ráközelítve)	35
6.17. TSNE elrendezés	35
6.18. TSNE elrendezés (ráközelítve)	35
6.19. Példa klaszterek	36
6.20. Időhisztogram az év napjai szerint (365 oszlop)	36
6.21. Időhisztogram az év hetei szerint (52 oszlop)	37

Táblázatok jegyzéke

5.1. Felület függvényében a "négyzetrács bejárás" kiszámításának ideje	26
7.1. Képek számának függvényében kollekció véglegesítési idő percben	38
7.2. Képek számának függvényében az alkalmazás betöltési ideje	38
7.3. Képek számának függvényében elrendezések előállítása	39

Irodalomjegyzék

- [atl23] https://en.wikipedia.org/wiki/Texture_atlas, 06 2023.
- [axu23] <https://github.com/tokio-rs/axum>, 06 2023.
- [goo23] <https://images.google.com/>, 06 2023.
- [ima23] <http://lab.softwarestudies.com/p/imageplot.html>, 06 2023.
- [lir23] <https://github.com/dermotte/LIRE>, 06 2023.
- [msg23] <https://msgpack.org/index.html>, 06 2023.
- [rea23] <https://react.dev/>, 06 2023.
- [rus23] <https://www.rust-lang.org/>, 06 2023.
- [Sza18] Kovács Szabolcs. Vizualizare și gestiune de colecții foarte mari de imagini. *Sapientia Erdélyi Magyar Tudományegyetem*, 2018.
- [tsn23] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding, 06 2023.
- [typ23] <https://www.typescriptlang.org/>, 06 2023.
- [Tü21] Bíró Tünde. Nagy képhalmazok online megjelenítése. *Sapientia Erdélyi Magyar Tudományegyetem*, 2021.
- [vdM14] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 2014.
- [was23] <https://webassembly.org/>, 06 2023.
- [web23] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, 06 2023.