

Knihovna pro určení vzájemně podobných fotografií vhodného pro produkční provoz

Bc. Dobroslav Pelc



ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Dobroslav Pelc**
Osobní číslo: **A16150**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Knihovna pro určení vzájemně podobných fotografií vhodného pro produkční provoz**
Téma anglicky: **An Image Matching Library Suitable for Production Operations**

Zásady pro vypracování:

1. Prostudujte metody pro určení vzájemně podobných fotografií.
2. Analyzujte požadavky systému pro vyhledávání podobných fotografií v reálném produkčním provozu.
3. Implementujte prototypy vybraných metod.
4. Porovnejte výsledky z pohledu časové a výpočetní náročnosti prototypů.
5. Vyberte nejvhodnější prototyp a dokončete realizaci tohoto prototypu.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ECKEL, Bruce. Thinking in Java. 4th ed. Upper Saddle River, NJ: Prentice Hall, c2006. ISBN 0131872486.
2. WALLS, Craig. Spring in action. Fourth edition. Texas: Manning, 2013. ISBN 978-161-7291-203.
3. HUNT, Charlie a Binu JOHN. Java performance. Upper Saddle River, NJ: Addison-Wesley, c2012. Java series. ISBN 0137142528.
4. PERŮTKA, K. MATLAB – Základy pro studenty automatizace a informačních technologií. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. 303 s. ISBN 80-7318-355-2.
5. DOBEŠ, M. Zpracování obrazu a algoritmy v C#. 1. vyd. Praha: BEN – technická literatura, 2008. 143 s. ISBN 978-80-7300-233-6.

Vedoucí diplomové práce:

Ing. Tomáš Dulík, Ph.D.

Ústav informatiky a umělé inteligence

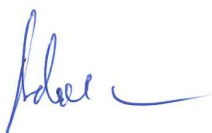
Datum zadání diplomové práce:

1. prosince 2017

Termín odevzdání diplomové práce:

16. května 2018

Ve Zlíně dne 11. prosince 2017



doc. Mgr. Milan Adámek, Ph.D.
děkan



prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

16.5.2018

..........

podpis autora

ABSTRAKT

Cílem této práce je analýza možností pro určení vzájemně podobných fotografií. Na základě analýzy, srovnávacích a zátěžových testů je vybrán nejvhodnější návrh řešení pro potřeby reálného produkčního provozu. Výsledná komponenta je realizována formou distribuované služby.

Klíčová slova: podobnost, zastupitelnost, distribuce, křížová korelace, konvoluce, Fourierova transformace, světlostní matice

ABSTRACT

The aim of this work is to analyze the possibilities for identifying similar photos. Based on analysis, benchmarking and stress testing, the most suitable solution design is chosen for real production use. The final component is implemented in the form of a distributed service.

Keywords: similarity, replaceability, distribution, cross correlation, convolution, Fourier transform, brightness matrix

Rád bych poděkoval především vedoucímu mé diplomové práce Ing. Tomášovi Dulíkovi, Ph.D. za odborné vedení, věnovaný čas a poskytnutí cenných rad a zkušeností. Dále pak mé poděkování patří společnosti STUDENT AGENCY k.s. za poskytnutí zázemí, pomoci při řešení problémů a materiálů, které umožnili vypracování této práce. Na závěr bych také rád poděkoval mé rodině, která byla vždy ochotná pomoci a při mém studiu mě vždy podporovala.

Moto

Albert Einstein: „*Pokud matematika popisuje realitu, není přesná. A pokud je přesná, nepopisuje realitu.*“

OBSAH

ÚVOD	10
I TEORETICKÁ ČÁST	10
1 KLASIFIKACE ŘEŠENÝCH OBLASTÍ	12
1.1 KONSTRUKTIVNÍ ZMĚNY FOTOGRAFIE.....	12
1.1.1 Návrh řešení	12
1.1.2 Oblast zájmu	12
1.2 DESTRUKTIVNÍ ZMĚNY FOTOGRAFIE.....	12
1.2.1 Návrh řešení	12
1.2.2 Oblast zájmu	13
1.3 KOMBINACE KONSTRUKTIVNÍCH A DESTRUKTIVNÍCH ZMĚN	13
1.3.1 Návrh řešení	13
1.3.2 Oblast zájmu	13
1.4 VÝBĚR REPREZENTATIVNÍHO VZORKU.....	13
1.4.1 Návrh řešení	13
1.4.2 Oblast zájmu	13
2 KOEFICIENT PODOBNOSTI DVOU FOTOGRAFIÍ.....	14
2.1 ZMĚNA VELIKOSTI FOTOGRAFIE	14
2.2 PŘEVOD NA SVĚTLOSTNÍ MATICI	14
2.3 PŘEVOD DO-Z VLNOVÉHO SPEKTRA	15
2.4 KŘÍŽOVÁ KORELACE	16
2.5 VÝPOČET VÝSLEDNÉHO KOEFICIENTU	16
3 KOEFICIENT ZASTUPITELNOSTI DVOU FOTOGRAFIÍ.....	18
3.1 DISKRÉTNÍ 2D KONVOLUCE	18
3.2 KONVOLUCE A VOLBA JÁDRA.....	18
II ANALYTICKÁ ČÁST	18
4 BRAINSTORMING	21
4.1 AKADEMICKÝ KRUH	21
4.2 PROFESNÍ KRUH	21
5 DIFERENČNÍ ANALÝZA (GAP ANALÝZA)	22
5.1 POPIS SOUČASNÉHO STAVU.....	22
5.2 POPIS CÍLOVÉHO STAVU	22
5.2.1 Nefunkční požadavky	22
5.3 ROZDÍLY	22
5.4 NÁVRH VARIANT K DOSAŽENÍ CÍLE	22
5.4.1 Výpočet KoP a KoZ na CPU produkčního serveru	23
5.4.2 Výpočet KoP a KoZ na CPU produkčního serveru s paralelizací na GPU.....	23

5.4.3	Výpočet koeficientů na PC farmě	23
5.5	ZHODNOCENÍ VARIANT	23
5.5.1	Výpočet KoP a KoZ na CPU produkčního serveru	24
5.5.2	Výpočet KoP a KoZ na CPU produkčního serveru s paralelizací na GPU	24
5.5.3	Výpočet koeficientů na PC farmě	24
6	BENCHMARKING	26
6.1	TESTOVACÍ PROSTŘEDÍ	26
6.1.1	Použitý HW	26
6.1.2	Použitý SW	26
6.2	VÝSLEDKY TESTOVÁNÍ	29
6.2.1	Výpočet koeficientů na CPU	29
6.2.2	Výpočet koeficientů na CPU s paralelizací na GPU	30
6.2.3	Výpočet koeficientů na PC farmě	31
III	PROJEKTOVÁ ČÁST	31
7	PŘÍPRAVA PROJEKTU	33
7.1	HW ARCHITEKTURA	33
7.1.1	Pohled na aktuální HW mapu	33
7.1.2	Pohled na plánovanou HW mapu	33
7.2	POŽADAVKY	33
7.2.1	Funkční požadavky	33
7.2.2	Nefunkční požadavky	33
7.3	PŘÍPADY POUŽITÍ	34
7.3.1	Aktéři	34
7.3.2	Případy užití	34
7.4	MODELÝ	35
7.4.1	Model tříd	35
7.4.2	Model služeb	35
7.5	SEKVENČNÍ DIAGRAM DISTRIBUOVANÉ SLUŽBY	35
8	SERVEROVÁ STRANA DISTRIBUOVANÉ SLUŽBY	37
8.1	DATABELIZACE FOTOGRAFIÍ	37
8.1.1	Výpočet poměrného počtu hran	37
8.2	FRONTA NEZPRACOVANÝCH OBRÁZKŮ	40
8.3	SERVLET PRO STAŽENÍ OBRÁZKŮ	40
8.4	SOAP API	41
8.4.1	Autentizace	41
8.4.2	Distribuce požadavků na výpočet KoP	41
9	KLIENSKÁ STRANA DISTRIBUOVANÉ SLUŽBY	46
9.1	DISTRIBUOVANÝ VÝPOČET KOP	46
ZÁVĚR		48

SEZNAM POUŽITÉ LITERATURY	51
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	55
SEZNAM OBRÁZKŮ	56
SEZNAM TABULEK	57
SEZNAM ZDROJOVÉHO KÓDU	58
SEZNAM PŘÍLOH	59

ÚVOD

Práce se zaměřuje na strojové zpracování rastrové grafiky. První kapitola se věnuje klasifikaci problémů a vybraným technikám vhodným k jejich řešení, jako je např. Fourierova transformace, křížová korelace nebo konvoluce. Další kapitola naváže analýzou potřeb všech zainteresovaných stran. Po analýze bude následovat návrh modulární architektury řešení. Zde bude kladen důraz především na aplikované techniky. V poslední kapitole bude zhodnocen výsledek práce.

Cílem této práce je analyzovat možnosti, porovnat je srovnávacím testem. Nejlepší varianta bude realizována formou prototypu distribuované služby, podrobena dalším testům a odladěna pro produkční provoz. Výsledná komponenta musí plnit všechny nefunkční požadavky a maximum funkčních požadavků.

Práce je limitována možnostmi současného produkčního prostředí. Hledá způsoby jak optimálně využít hrubou sílu produkčního prostředí, která je obecně nevhodná pro operace s rastrovou grafikou. Hlavní myšlenkou této práce je porovnat možnosti paralelizace strojového zpracování rastrové grafiky na GPU s finanční dostupností tohoto řešení pro produkční provoz.

I. TEORETICKÁ ČÁST

1 Klasifikace řešených oblastí

Pro porovnání dvou fotografií a jejich vzájemné vyhodnocení jako podobné či nikoliv bude zaveden koeficient podobnosti fotografií (dále jen KoP). Postupně budou popsány všechny silné i slabé stránky pro možnosti výpočtu KoP. Základem klasifikace je vymezení rozdílů mezi vzorovou a referenční fotografií, které definují konstruktivní a destruktivní změny.

1.1 Konstruktivní změny fotografie

Konstruktivní změny fotografie jsou takové změny, které jsou viditelné pro lidské oko, avšak nemění zásadně charakter obrázku pro strojové zpracování. Možné změny (případně jejich kombinace) jsou rozvedeny níže.

1.1.1 Návrh řešení

Pro porovnání dvou podobných fotografií, které obsahují pouze konstruktivní změny, plně postačí křížová korelace, která bude urychlena (v rámci optimalizace HW času) pomocí diskrétní Fourierovy transformace [10].

1.1.2 Oblast zájmu

Touto numerickou metodou lze detekovat většinu nejčastějších záměrných modifikací fotografií (např. automatické doostření a následné uložení jako kopie). Lze sem zahrnout jak změny vlastností vázané na původní fotografii, tak drobné změny v původním obsahu fotografie.

Změny vlastností

- Změna sytosti barev
- Změna kontrastu
- Změna jasu

Změny obsahu

- Vodotisk
- Logo
- Šum

1.2 Destruktivní změny fotografie

Destruktivní změny fotografie jsou viditelné pro lidské oko, ale pouze vjemem lidského oka je často problém tyto fotografie bezpečně prohlásit za podobné. Ještě horší situace je u strojového zpracování, pro které se výrazně mění charakter porovnávaných fotografií.

1.2.1 Návrh řešení

Výpočet Hausdorfovy vzdálenosti mezi konvexními polyedry, které reprezentující hrany v obrazu [15].

1.2.2 Oblast zájmu

- Změna komprese (rozmazaná fotografie)
- Změny rozlišení
 - Ořez (v jedné nebo obou dimenzích)
 - Deformace (v jedné nebo obou dimenzích)

1.3 Kombinace konstruktivních a destruktivních změn

Kombinace konstruktivních a destruktivních změn je vždy potřeba vyhodnotit nad konkrétním případem. Platí také, že jsou velmi obtížně řešitelné. V závislosti na míře změn lze často rozpoznat KoP stejně jako u čistě konstruktivních změn.

1.3.1 Návrh řešení

Redukce fotografie na její prahovou velikost jako příprava na křížovou korelaci viz konstruktivní změny [14].

1.3.2 Oblast zájmu

- Asymetrická změna obou stran s čímkoliv
- Změna kvality v důsledku zhoršení komprese s čímkoliv
- Logo nebo vodotisk v kombinaci s předcházejícími

1.4 Výběr reprezentativního vzorku

Pro skupinu vzájemně si podobných fotografií vybereme nejvhodnějšího kandidáta, který bude následně ostatní fotografie zastupovat. Jde o experimentální postup.

1.4.1 Návrh řešení

Bude zaveden koeficient zastupitelnosti (dále jen KoZ), který je zjednodušeně určen jako $VELIKOST * OSTROST + JAS$. Přičemž platí, že vyšší hodnota koeficientu zastupitelnosti znamená kvalitnější fotografii (nikoliv na oko hezčí fotografii). Reprezentativní vzorek bude fotografie, která bude vybrána ze skupiny podobných fotografií na základě KoP, s nejvyšším KoZ.

1.4.2 Oblast zájmu

Střední hodnota jasů je určena pomocí světlostní matice. Jedná se pouze o zohlednění, zda fotografie není příliš jasná nebo tmavá. Jde o jednoduchý algoritmus. Na výsledek nemá zásadní vliv.

Poměrný počet hran slouží jako test rozmazanosti fotografie. K realizaci se používá konvoluční matice (více v samostatné kapitole věnované této problematice) s vhodným jádrem typu horní i dolní propust (s celkovým součtem 0). Na výsledek má největší dopad.

Rozlišení fotografie je bráno jako klasická velikost fotografie v px (větší $\Leftrightarrow$ lepší).

2 Koefficient podobnosti dvou fotografií

Jde o základní ukazatel podobnosti dvou fotografií. KoP leží na intervalu $< 0, 1 >$. Přičemž hodnoty blíží se 1 symbolizují podobné fotografie. Interval podobnosti byl na základě testovacích pokusů stanoven na $< 0, 07, 1$). Hodnota 1 znamená duplicitní fotografii. Byla z intervalu vyloučena, jelikož jsou fotografie nejprve unifikovány pomocí otisku MD5 [6]. Výpočet KoP provedeme v šesti krocích, z čehož jsou čtyři kroky přípravné (optimalizační) a pouze dva kroky reálně ovlivňují výsledný KoP.

1. Změna velikosti fotografie
2. Převod na světlostní matici
3. Převod do vlnového spektra
4. Křížová korelace (Cross correlation method)
5. Převod zpět z vlnového spektra
6. Výpočet KoP z výsledné matice

Body 1–3 slouží jako přípravné a aplikují se na obě fotografie (vzorová a referenční). Do bodu 4 tedy vstupují dvě matice (pro každou fotografii jedna). Výstupem 4. bodu je již jen jedna matice, která je v bodě 6 vyhodnocena do výsledného KoP.

2.1 Změna velikosti fotografie

Jedná se o nezbytný přípravný krok, jehož cílem je sjednotit u obou fotografií počet bodů a tím také počet prvků v maticích, které vzniknou v následujícím kroku výpočtu KoP. Jako referenční velikost byla stanovena

- Šířka: 320 px,
- Výška: 240 px,

kteřé jsou nejvhodnějším kompromisem mezi relevancí výsledku a HW časem nutným k jeho zpracování.

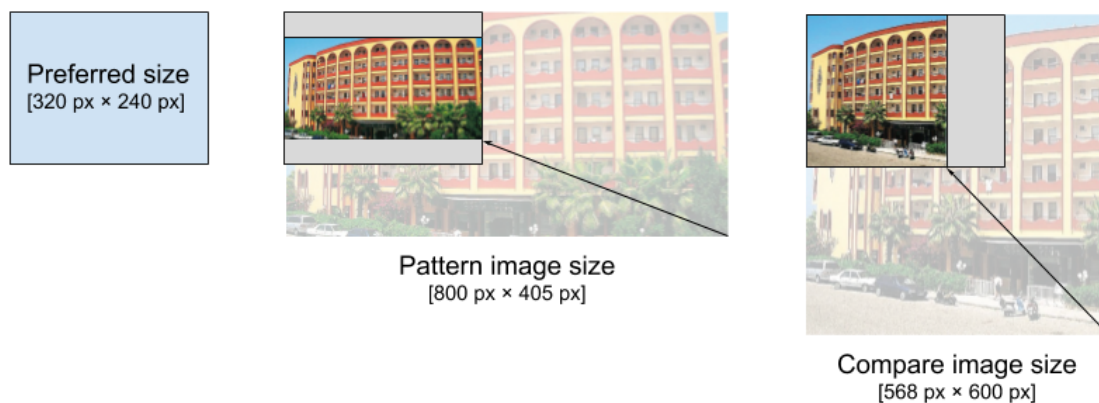
Změna velikosti vzorové fotografie na $[320 \times ?]$ nebo $[? \times 240]$ se provádí v závislosti na delší straně. Kratší strana je dopočítána podle původního poměru stran. Výsledný rozměr není doplněn nulami na plnou referenční velikost $[320 \times 240]$. Vzorová fotografie tedy určuje velikost, na kterou musíme upravit referenční fotografii. Pokud má referenční fotografie jiný poměr stran, bude doplněna nulami, aby nevznikala prázdná místa.

Takto připravené fotografie nejsou při dalším zpracování komutativní, pokud mají vzájemně jiný poměr stran. V důsledku to znamená, že se musí porovnat fotografie v obou směrech (jak vzorová vůči referenční, tak referenční vůči vzorové). Situaci ukazují příklady na Obr. 2.1 a Obr. 2.2.

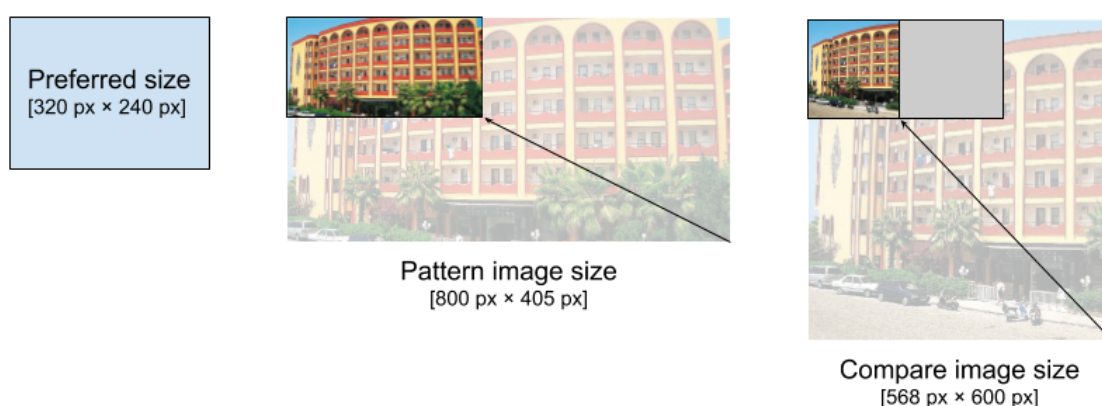
Přesto, že je tento proces HW dražší, než jeho komutativní varianta, získáme díky tomu řádově lepší relevanci výsledků (zejména pokud je jedna ze dvou fotografií velmi nekvalitní, případně má nižší nativní rozlišení, než je referenční).

2.2 Převod na světlostní matici

Světlostní matice představuje fotografii ve formátu rastrové bitmapy (někdy též šedotónový obraz; v anglicky psaných textech k nalezení pod názvem brightness-matrix). Formálně je to dvourozměrná diskrétní veličina, reprezentovaná maticí druhého řádu [7].



Obr. 2.1 Statický scaling obrázků (komutativní)



Obr. 2.2 Dynamický scaling obrázků (diskomutativní)

Každý bod původní fotografie - pixel [8] (dále jen px) je z RGB [9] hodnot převeden na hodnotu intenzity jasové funkce.

Původní px je reprezentován jako tříprvkové pole s hodnotami na intervalu $< 0, 255 >$.

- R => red,
- G => green,
- B => blue,

Výsledná hodnota intenzity jasové funkce px se spočítá jako střední hodnota z hodnot jednotlivých složek px (R, G a B). Jak napovídá interval možných hodnot, jeden px převedený na prvek světlostní matice zabírá v operační paměti 1 Byte (1 Byte = 8 bit => osmibitová barevná hloubka => 256 stupňů šedi). Na jedenu plnou referenční fotografii je tedy potřeba 75 kB.

2.3 Převod do-z vlnového spektra

Převodem do vlnového spektra dosáhneme zajímavé výkonnostní optimalizace [10]. Kdybychom tento krok z celého procesu výpočtu KoP vynechali (stejně jako následně nezbytný převod zpět z vlnového spektra), museli bychom udělat křížovou korelaci v normálním spektru. Tzn. $O(n^4)$ operací, kde n je velikost strany čtvercové matice.

Pokud ale nejprve provedeme Diskrétní Fourierovu transformaci (DFT) pro převod do vlnového spektra, až následně křížovou korelaci a nakonec Zpětnou Fourierovu trans-

formaci (IFT), se ušetří jeden řád hodnoty počtu operací. Bude potřeba $O(n^3 * \log n)$ operací. Řád hodnoty předpokládá ve výpočtu čtvercovou fotografii (reálná fotografie je obdélníková => řád hodnoty je závislý na dvou proměnných)

- Klasické spektrum: $200^4 = 1.6 * 10^9$ operací
- Vlnové spektrum: $200^3 * \log 200 = 6.1 * 10^7$ operací

2.4 Křížová korelace

Korelace je nejdůležitější krok celého výpočtu KoP. Umožní pomocí jednoduchých matematických operací rozhodnout, zda jsou světlostní matice vzorové a referenční fotografie podobné. Míra podobnosti je vyjádřena hodnotami korelačních koeficientů sestavených do jedné matice (zatím ještě ve vlnovém spektru). Čím více se výsledné hodnoty blíží 1, tím více si jsou fotografie podobné v daném bodě.

Celý proces není citlivý na konstruktivní změny. Dokáže tedy podobnost vyhodnotit bez ohledu na změnu sytosti barev, kontrastu či jasu. Stejně tak výsledek není ovlivněn přidáním vodotisku nebo šumem ve fotografii. Přidání loga do fotografie již sice sníží KoP, ale pokud není logo přes polovinu fotografie, je stále bezpečně rozpoznána jako podobná či nikoliv.

Formálně je korelace zejména statistický pojem, který označuje vzájemný lineární vztah mezi znaky nebo veličinami. Míra korelace je daná korelačním koeficientem [11].

Vzorec pro výpočet: $\rho_{A,B} = \frac{(A - \mu_A) \times (B - \mu_B)}{\sigma_A \sigma_B}$

Na první pohled se může zdát složitý, ale skrývá v sobě tři jednoduché ale důležité kroky.

1. K maticím A a B se spočítá rozptyl σ_A, σ_B a střední hodnota μ_A, μ_B .
2. Odečtením střední hodnoty od matice $A - \mu_A, B - \mu_B$ je dosaženo **invariace nastavení jasu**.
3. Dělením rozptyly $\sigma_A \sigma_B$ je zajištěna **invariace nastavení kontrastu**.

Za předpokladu, že objekty na porovnávaných fotografiích nejsou vůči sobě v prostoru posunuty, poskytuje korelační matice odpověď na otázku, zda jsou si dvě fotografie podobné. V praxi ale tento model příliš nenastává. Naopak je velmi časně, že je porovnáván např. výřez z fotografie oproti originálu, případně posunutá fotografie po horizontální či vertikální ose.

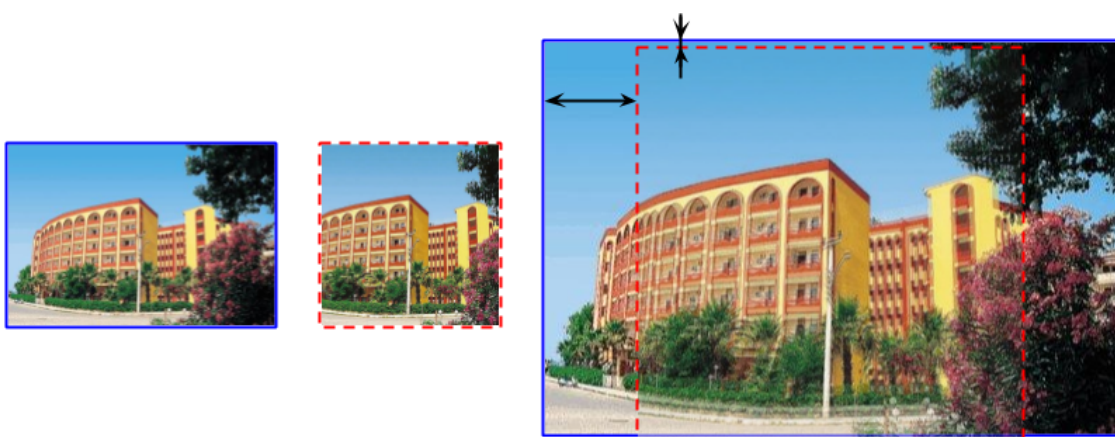
Tuto problematiku řeší křížová korelace (Cross correlation method) [12]. Princip samotné korelace je stejný, pouze se opakuje s částečným posunem tak, aby pokryla všechny možné kombinace mezi dvěma fotografiemi. Příklad je názorně vidět na původních fotografiích, jak ukazuje (Obr. 2.3) a (Obr. 2.4).

2.5 Výpočet výsledného koeficientu

Po převodu z vlnového spektra zpět vychází již matice, ze které jsou vyloučeny imaginární hodnoty a pracujeme tedy jen s reálnými čísly. Výsledný KoP pro dané fotografie je dán nejvyšším nalezeným korelačním koeficientem v matici.



Obr. 2.3 Příklad proložení dvou fotografií s použitím klasické korelace



Obr. 2.4 Příklad proložení dvou fotografií s použitím křížové korelace

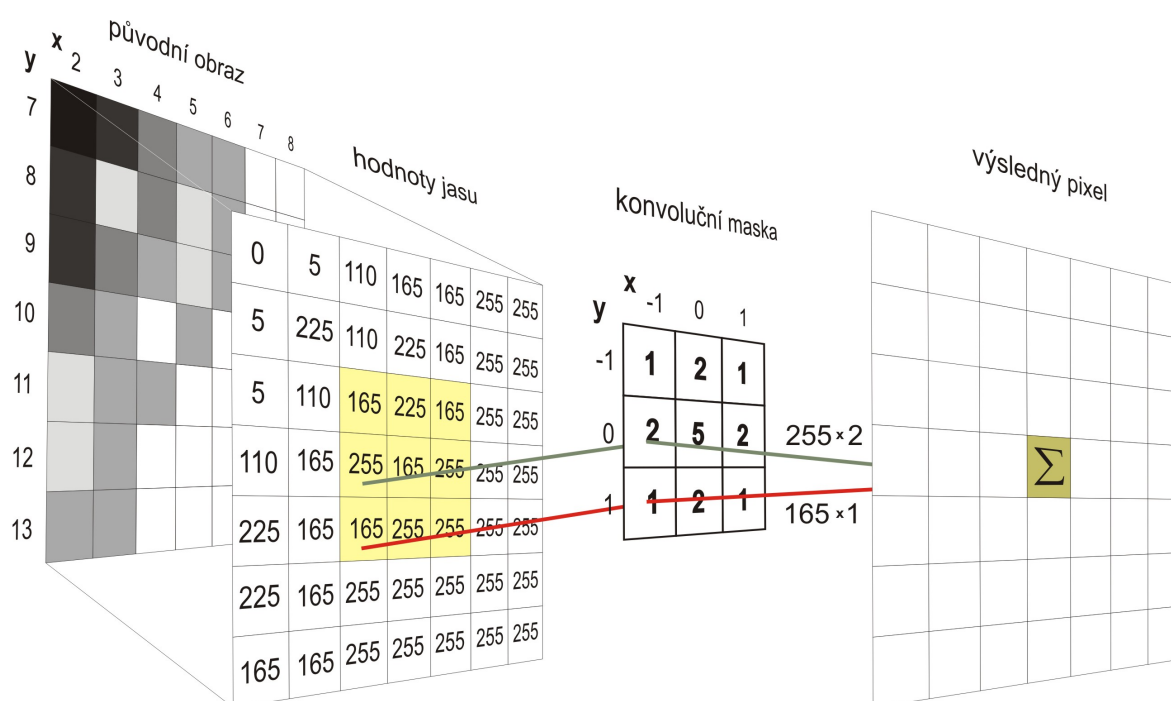
3 Koefficient zastupitelnosti dvou fotografií

Určení KoZ je experimentální proces založený na opakované aplikaci konvoluční matice s různým jádrem (konvoluční maskou). Bude jí věnována větší pozornost v projektové části. Zde budou představeny pouze techniky nutné k jejímu dosažení. Jelikož převod na světlostní matici a převod z-do vlnového spektra byly již představeny, nebudou dále znovu uvedeny.

3.1 Diskrétní 2D konvoluce

Konvoluce [19] je matematický operátor pro zpracování dvou funkcí. V algoritmech zpracovávající dvourozměrný diskretní obraz (např. v počítačové grafice) má konvoluce následující tvar: $(f * h)(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k f(x-i, y-j) \bullet h(i, j)$

Názorná ukázka konvoluce je vidět na přiloženém obrázku (Obr. 3.1).



Obr. 3.1 Příklad Diskrétní 2D konvoluce [19]

3.2 Konvoluce a volba jádra

Konvoluce ve světě počítačové grafiky je operace s obrázkem ve formátu matice pomocí jiné matice zvané „jádro“ (nebo též konvoluční maska). Jako první matice se používá obrázek určený k úpravě. Obrázek představuje dvojrozměrnou pravoúhlou souřadnicovou síť pixelů. Použité jádro závisí na požadovaném efektu.

V našem případě je požadovaný filtr na detekci hran [20]. Základní jádra pro detekci hran mají součet roven nule (Tab. 3.1 a Tab. 3.2). Nejsou vhodné na obrázky zatížené šumem. Ten je nezbytné před aplikací těchto filtrů eliminovat.

Tab. 3.1 Jádro pro detekci hran (horizontálně a vertikálně)

0	1	0
1	-4	1
0	1	0

Tab. 3.2 Jádro pro detekci hran (horizontálně, vertikálně a šikmé hrany)

1	1	1
1	-8	1
1	1	1

II. ANALYTICKÁ ČÁST

4 Brainstorming

Tato analytická metoda slouží ke sběru myšlenek, námětů a případné zevrubné konstruktivní kritice dané problematiky. V rámci této práce byla použita v akademickém a profesním kruhu za účelem identifikace základních ukazatelů pro další kroky analýzy.

4.1 Akademický kruh

Diskutovány byly zejména technické možnosti týkající se otázek kde a jak lze vůbec porovnání fotografií provádět strojově. K další analýze byly vybrány algoritmy vhodné pro strojové určení podobnosti fotek

- Porovnání pomocí histogramu
- Porovnání pomocí Hausdorff distance
- Porovnání pomocí SVM [23]

4.2 Profesní kruh

V kruhu s provozovatelem byly kladeny nejvyšší nároky na flexibilitu a propustnost celého řešení.

5 Diferenční analýza (GAP analýza)

5.1 Popis současného stavu

Je požadováno porovnání rastrových bitmap za účelem identifikace vzájemně podobných fotografií. Řešení je hledáno pro produkční provoz. Konzumentem cílového řešení je webový portál dovolena.cz, který má přibližně dva miliony fotografií. Průměrný počet přístupů k některé fotografii je přibližně 100 přístupů za sekundu. Obsahem fotografií jsou především hotely a jejich okolí. Webový portál slouží spíše jako datový konsolidátor. Nabídka portálu značně ovlivňuje cílové portfolio fotografií. Podle testovacích měření se za jeden týden obmění cca 10% fotografií z celkového množství. Jako testovací vzorek byl vybrán jeden nejmenovaný hotel a jeho 35 fotografií. Zákazník webového portálu vidí všechny fotografie v nesetříděné galerii. Některé fotografie jsou unikátní, ale většina si je velmi podobná. U některých dokonce nejsou lidským okem patrné rozdíly.

5.2 Popis cílového stavu

Konsolidované fotografie prezentované klientovi budou v maximální možné míře obsahovat unikátní fotografie. Vzájemně si podobné fotografie budou odfiltrovány a zůstane pouze jedna a to fotografie s nejvyšším KoZ. Klient nebude čekat na zpracování podobnosti obrázků. Pokud budou zpracované, klient uvidí jen unikáty. Pokud nebudou zpracované, klient uvidí vše v původním stavu. V takovém případě se poměrově zvýší priorita na výpočet podobnosti fotografií tohoto hotelu vůči ostatním ve frontě na výpočet. Cílové řešení musí být schopno operovat řádově s jednotkami milionů fotografií s týdenní fluktuací 15%.

5.2.1 Nefunkční požadavky

- Bezúdržbový systém
- Nevyžadující v průběhu času další financování
- Minimální vstupní investice
- Maximální kompatibilita s aktuálním HW
- Programovací jazyk Java [25]

5.3 Rozdíly

- Nově vznikne nástroj pro určení KoP.
- Nově vznikne nástroj pro určení KoZ.
- Fotografie jednoho hotelu budou oindexovány a vnitřně škálovány do skupin pomocí koeficientů výše.
- Dojde k navýšení celkového počtu fotografií.
- Zvýší se fluktuace fotografií (na očekávaných 15%).

5.4 Návrh variant k dosažení cíle

Pilířem celého řešení bude backendová strana cílového konzumenta. Limity a také jednotlivé možnosti pro realizaci jsou velmi omezeny nutností integrovat do současného řešení. Z těchto důvodů má serverová strana převážně podpůrný charakter v projektu

jako celku. Její význam je zejména v propojení všech jednotlivých komponent. Dojde tedy k modifikaci existujícího produkčního server-side prostředí. Nově zde poběží služba, která bude

- poskytovat zadání na určení KoP,
- poskytovat metadata nezbytná pro distribuci výpočtu,
- konzumovat výsledek distribuované operace,
- kompletovat zpracovaná data do cache vhodné pro silný organický provoz.

Naopak klientská strana je naprosto autonomní. Pro realizaci lze použít jak libovolnou platformu, tak libovolné technologie. Jediným technickým limitem je schopnost standardizovaným způsobem komunikovat se serverovou stranou pomocí SOAP api [21].

5.4.1 Výpočet KoP a KoZ na CPU produkčního serveru

Základní myšlenka je využít nejdostupnější produkční HW a na zavedeném serveru spustit novou službu. Hlavní výhodou je dostupnost produkčního HW ve vlastním datacentru cílového konzumenta. Podstatnou nevýhodou je fakt, že pro určení výše uvedených koeficientů není CPU [22] ideální platforma.

5.4.2 Výpočet KoP a KoZ na CPU produkčního serveru s paralelizací na GPU

Tento způsob předpokládá osazení produkčního serveru dedikovanou pracovní grafickou kartou a vybrané části procesu výpočtu KoP a KoZ optimalizovat pro zpracování na GPU [24].

5.4.3 Výpočet koeficientů na PC farmě

Způsob předpokládá využití HW osobních PC, kterých je v každé větší společnosti dostatek. Výpočet koeficientů tedy není nutno provádět na jednom stroji, ale lze jej provádět na každém dostupném stroji.

5.5 Zhodnocení variant

Jako nejvýhodnější varianta pro realizaci vychází PC farma. Jako záložní řešení lze využít produkční server s výpočtem na CPU. Pokud bude zvolen vhodný programovací jazyk (např. požadovaný programovací jazyk Java [25]), bude výsledné řešení přenositelné.

Zhodnocení výkonových rozdílů vychází z benchmarkingu, který byl vyhodnocen jako pomocná analýza uvedená níže. Zhodnocení prezentuje tabulka (Tab. 5.1)

Tab. 5.1 Tabulka výsledků jednotlivých variant krátkodobého testu

Varianta	Týdenní přírůstek	Všechny fotografie	Teoretická rezerva
CPU	6 dní	27 týdnů	3,7%
GPU	8 hodin	2 dny 2 hodiny	300%
PC farma (300 ks)	2,5 hodin	16 hodin	700%

Zhodnocení investičních rozdílů ukazuje tabulka (Tab. 5.2). Vychází z předpokladu, že x je investiční konstanta v Kč, vztažená na cenu jednoho produkčního serveru

bez GPU. Pro zjednodušení je stanovena na 100.000 Kč (není příliš daleko od reality).

Tab. 5.2 Tabulka výsledků jednotlivých variant krátkodobého testu

Varianta	Pořizovací cena [Kč]	Nutná investice [Kč]
CPU	x	0
GPU	$10x$	$10x$
PC farma (300 ks)	$30x$	0

5.5.1 Výpočet KoP a KoZ na CPU produkčního serveru

Výsledky banchmarkingu

- Za necelých 6 dní spočítá týdenní přírůstek.
- Za zbylou dobu z týdenního cyklu dále vypočítá přibližně 3,7% z celkového objemu sto milionu koeficientů.
- Pro plné vypočítání všech koeficientů potřebuje dalších cca 27 týdnů.
- Rezerva pro další růst (navýšení celkového počtu fotografií) je cca 18 %.

5.5.2 Výpočet KoP a KoZ na CPU produkčního serveru s paralelizací na GPU

Pro realizaci tohoto scénáře je nezbytná vysoká vstupní investice. Běžně dostupný produkční server nedisponuje GPU. Pořizovací cena takového serveru je řádově vyšší. Dále je nutné připočíst cenu vlastní dedikované grafické karty, která cenově převyšuje cenu serveru. Její výběr je velmi omezen kompatibilitou s produkčními servery. Zástupce firmy Dell byl schopen nacenit na vlastní server pouze dvě karty s možností garancí non-stop provozu a výměnou do 24 hodin.

Výsledky banchmarkingu

- Za necelých 8 hodin spočítá týdenní přírůstek.
- Za část ze zbylé doby týdenního cyklu dále vypočítá 100 % z celkového objemu sto milionu koeficientů.
- Pro plné vypočítání všech koeficientů potřebuje celkem 2 dny.
- Rezerva pro další růst (navýšení celkového počtu fotografií) je cca 300 %.

5.5.3 Výpočet koeficientů na PC farmě

Výsledek jednoho kancelářského PC je zanedbatelný a nemůže se rovnat předchozím variantám. Vezmeme-li v úvahu, že těchto strojů je k dispozici 300 kusů po dobu 16 hodin denně, posunou se výsledky na jinou úroveň.

Výsledky banchmarkingu

- Za 2,5 hodiny spočítá týdenní přírůstek (za předpokladu 300 aktivních PC).
- Za část ze zbylé doby týdenního cyklu dále vypočítá 100 % z celkového objemu sto milionu koeficientů.
- Pro plné vypočítání všech koeficientů potřebuje celkem 16 hodin, což je v tomto případě 1 den.

-
- Rezerva pro další růst (navýšení celkového počtu fotografií) je cca 700 %.

Tento krok předpokládá mimo jiné i revizi infrastruktury, především kvalitu a šířku pásma sítě, centrální správu PC apod., které v této fázi nejsou zahrnuty do hodnotících kritérií.

6 Benchmarking

Jednotlivé kandidáty pro realizaci klientské části (zavedené v diferenční analýze) podrobně vykonostním a srovnávacím testům. Základní předpoklady:

- Řádově je potřeba jednorázově odbavit 100.000.000 výpočtů KoP.
- Řádově je potřeba jednorázově odbavit 2.000.000 výpočtů KoZ.
- Na týdenní bázi následně odbavovat přírůstky řádově
 - 15.000.000 výpočtů KoP,
 - 300.000 výpočtů KoZ.

6.1 Testovací prostředí

6.1.1 Použitý HW

Simulace produkčního serveru je založena na pracovní stanici Lenovo D20, která je svoji sestavou velmi blízko produkčnímu serveru. Vzhledem k tomu, že disponuje starší generací CPU, bylo do ní osazeno i odpovídající GPU, aby bylo možné výsledky aproximovat na reálný produkční HW.

- Lenovo ThinkStation D20
- Operační systém Windows 10 pro 64 bit
- CPU Intel(R) Xeon(R) X5670 @ 2.93 GHz (2×cpu, 6×core, 12×thread)
- RAM 32 GB DDR3 (1066)
- GPU NVIDIA GeForce GTX 460
- HDD SAS 10.000 ot

Simulace klasického PC je založena na PC Dell Optiplex 780. Obecně je to velmi rozšířený kancelářský PC. Taktéž cílový konzument disponuje více než 300 ks odpovídajících kancelářských PC. Tato simulace je omezena pouze na výpočty na CPU.

- Dell Optiplex 780
- Operační systém Windows 7 pro 32 bit
- CPU Intel Core 2 Duo E7500 2,93 GHz
- RAM 4 GB DDR3
- HDD SATA 250 GB

6.1.2 Použitý SW

V úvodní fázi projektu probíhal vývoj i testování algoritmů v Matlabu [4]. Ten vnitřně používá pro zpracování DFT a IFT knihovnu FFTW [13]. Tato knihovna má dostupnou nativní implementaci v CMake [17]. Dále má pro řadu programovacích jazyků (včetně programovacího jazyku Java [16]) připravený wrapper [18].

Algoritmus v matlabu (výpočet KoP na CPU) níže (Kod. 6.1) byl použit jako základní srovnávací test pro výpočet KoP hrubou silou na CPU. Výsledný KoP odpovídá v algoritmu proměnné ccCMax.

Kod. 6.1 banchmark-cpu.m

```
1 close all; clc; clear;
2
3 method = 'nearest';
4 %method = 'bicubic';
5
```

```

6 type = 'single';
7
8 n = 35; % pocet obrazku
9
10 %obrazky
11 im_gpu = cell(1, n);
12
13 h = zeros(1, n, type);
14 w = zeros(1, n, type);
15
16 s0 = 'hotel_images\aa';
17
18 %nahravani obrazku
19 tic
20 for i = 1:n
21     if i < 10
22         s = [s0, '0', int2str(i), '.tif'];
23     else
24         s = [s0, int2str(i), '.tif'];
25     end;
26
27     im = single(imread(s)); %nahrani obrazku a prevod
28
29     [n1, n2, n3] = size(im);
30     h(i) = n1;
31     w(i) = n2;
32     im_gpu{i} = im; %nahani obrazku
33 end;
34 disp('Nahravani vseh obrazku:');
35 toc
36
37 %prevadeni obrazku na jasove matice
38 for i=1:n
39     im_gpu{i} = sum(im_gpu{i}, 3) ./ 3; %prevod na jasovou matici
40 end;
41 disp('Prevod vseh obrazku na jasovou matici:');
42 toc
43
44 log_cpu = zeros(n, n, type);
45
46 %cyklus
47 for i=1:n
48     %ft prvnio obrazku
49     ft1 = fft2(im_gpu{i});
50     ft1Norm = ft1 ./ abs(ft1);
51
52     for j=1:n
53         if i==j
54             continue;
55         end;
56
57         scale = min(h(i)/h(j), w(i)/w(j));
58         h2 = round(scale * h(j));
59         h2 = min(h2, h(i));
60         w2 = round(scale * w(j));
61         w2 = min(w2, w(i));
62
63         im2Sc = imresize(im_gpu{j}, [h2, w2], method);
64
65         %doplneni druhého obrazku nulami na jednotnou velikost
66         im2 = zeros(h(i), w(i), type);
67         im2(1:h2, 1:w2) = im2Sc;
68
69         %ft druhého obrazku
70         ft2 = fft2(im2);
71         ft2Norm = ft2 ./ abs(ft2);
72
73         %krizova korelace ve vlnovem spektru
74         ccW = ft1Norm .* conj(ft2Norm);
75
76         %prevod do obycejneho souradnicoveho systemu
77         ccC = real(ifft2(ccW));
78
79         %hledani maximalniho korelacniho koeficientu
80         ccCMax = max(ccC);
81         ccCMax = max(ccCMax);
82         log_cpu(i, j) = ccCMax;
83         disp(['obr1: ', int2str(i), ' obr2: ', int2str(j)]);
84     end;
85 end;
86 disp(['Cyklus ', int2str(n), ' obrazku:']);
87 time = toc
88
89 clearvars -except log_cpu;

```

Algoritmus v matlabu (výpočet KoP s paralelizací na GPU) níže (Kod. 6.2) vychází z původního zpracování (Kod. 6.1). Vybrané části výpočtu KoP jsou zpracovány pomocí paralelizace na GPU. Výsledný KoP opět odpovídá proměnné ccCMax. Pro paralelizaci byla využita platforma CUDA [26].

Kod. 6.2 benchmark-gpu.m

```

1 %setenv('NSIGHT_CUDA_DEBUGGER', '1')
2 close all; clear; clc;
3
4 %method = '_nearest';
5 %method = '_biquadr';
6 method = '_bicubic';
7
8 type = 'single';
9
10 n = 35;
11
12 path = 'kernels\ImgScale\ImgScale\kernel_';
13 kernel = parallel.gpu.CUDAKernel([path, type, method, '.ptx'], [path, type, method, '.cu']);
14
15 %obrazky
16 im_gpu = cell(1, n);
17
18 h = zeros(1, n, type);
19 w = zeros(1, n, type);
20
21 s0 = 'hotel_images\aa';
22
23 %nahravani obrazku na grafiku
24 tic
25 for i = 1:n
26     if i < 10
27         s = [s0, '0', int2str(i), '.tif'];
28     else
29         s = [s0, int2str(i), '.tif'];
30     end

```

```

31
32     im = single(imread(s)); %nahrani obrazku a prevod
33
34     [n1,n2,n3] = size(im);
35     h(i) = n1;
36     w(i) = n2;
37     im_gpu{i} = gpuArray(im); %nahrani obrazku na grafiku
38
39 end
40 disp('Nahravani vseh obrazku do pameti grafiky:');
41 toc
42
43 %prevadeni obrazku na jasove matice
44 for i=1:n
45     im_gpu{i} = sum(im_gpu{i}, 3) ./ 3; %prevod na jasovou matici
46 end;
47 disp('Prevod vseh obrazku na jasovou matici:');
48 toc
49
50 log = zeros(n, n, type, 'gpuArray');
51
52 %cyklus
53 for i=1:n
54     %ft prvnio obrazku
55     ft1 = fft2(im_gpu{i});
56     ft1Norm = ft1 ./ abs(ft1);
57
58     for j=1:n
59         if i==j
60             continue;
61         end;
62
63         scale = (min(h(i)/h(j), w(i)/w(j)));
64         h2 = round(scale * h(j));
65         h2 = min(h2, h(i));
66         w2 = round(scale * w(j));
67         w2 = min(w2, w(i));
68         scale_h = h(j)/h2;
69         scale_w = w(j)/w2;
70
71         %im2Sc = imresize(im_gpu{j}, scale);
72
73         im2Sc = zeros(h2, w2, type, 'gpuArray');
74         kernel.GridSize = w2;
75         kernel.ThreadBlockSize = h2;
76         im2Sc = feval(kernel, im2Sc, im_gpu{j}, int32(h(j)), int32(w(j)), scale_h, scale_w);
77
78         %dopljeni druhého obrazku nulami na jednotnou velikost
79         im2 = zeros(h(i), w(i), type, 'gpuArray');
80         im2(1:h2, 1:w2) = im2Sc;
81
82         %ft druhého obrazku
83         ft2 = fft2(im2);
84         ft2Norm = ft2 ./ abs(ft2);
85
86         %krizova korelace ve frekvencnim spektru
87         ccW = ft1Norm .* conj(ft2Norm);
88
89         %prevod do obycejneho souradnicoveho systemu
90         ccC = real(ifft2(ccW));
91
92         %hledani maximalniho korelacniho koeficientu
93         ccCMax = max(ccC);
94         ccCMax = max(ccCMax);
95         log(i,j) = ccCMax;
96         disp(['obr1: ',int2str(i),'; obr2: ',int2str(j)]);
97     end
98 end
99 disp(['Cyklus ',int2str(n),' obrazku:']);
100 log_gpu = gather(log);
101 toc
102 clearvars -except log_gpu;

```

Algoritmus v matlabu (výpočet KoZ na CPU) níže (Kod. 6.3) je svou strukturou a celkovým provedením odlišný od předchozích algoritmů (Kod. 6.1 a Kod. 6.2). Některé prvky (např. výpočet světlostní matice) zůstaly zachovány. Výpočet KoZ byl po provedení testů ve srovnávacích testech zanedbán. Především proto, že se nemusí počítat pro kombinace fotografií, ale právě jednou pro každou fotografii a to při jejím pořízení. Průměrná doba výpočtu KoZ je 620 ms.

Kod. 6.3 banchmark-koz-cpu.m

```

1 a = cell(1,15);
2 a{7} = 'image/006780_5305124.jpg';
3 a{10} = 'image/37342_cat008802_069_01.jpg';
4 a{13} = 'image/37342_cat009629_110_10.jpg';
5 a{4} = 'image/37342_010590_11837774.jpg';
6 a{11} = 'image/37342_cat008103_113_02.jpg';
7 a{1} = 'image/37342_cat009015_115_06.jpg';
8 a{9} = 'image/37342_cat011203_140_04.jpg';
9 a{6} = 'image/cat013326_136_04.jpg';
10 a{14} = 'image/cat012556_124_06.jpg';
11 a{5} = 'image/37342_cat010488_142_06.jpg';
12 a{15} = 'image/cat013842_136_04.jpg';
13 a{8} = 'image/lbc402_zimmer1_0912.jpg';
14 a{12} = 'image/37342_cat009533_052_02.jpg';
15 a{2} = 'image/37342_cat011000_055_02.jpg';
16 a{3} = 'image/37342_cat011855_060_02.jpg';
17
18 close all;
19
20 startfile = 1;
21 endfile = 15;
22
23 best = zeros(endfile - startfile+1,4);
24
25 kernel_up = [-0.0625 -0.0625 -0.0625; -0.0625 0.5 -0.0625; -0.0625 -0.0625 -0.0625];
26
27 kernel_down = [0 0.125 0; 0.125 0.5 0.125; 0 0.125 0];
28
29 for i=startfile:endfile
30     im = imread(a{i});
31     [m,n] = size(im(:,:1));
32
33     im_br = sum(im,3)/3;
34     im_ft = fft2(im_br);
35

```

```

36 kernel_up_sized = zeros(m,n);
37 kernel_up_sized(1:2,1:2) = kernel_up(2:3,2:3);
38 kernel_up_sized(m,1:2) = kernel_up(1,2:3);
39 kernel_up_sized(1:2,n) = kernel_up(2:3,1);
40 kernel_up_sized(m,n) = kernel_up(1,1);
41 kernel_up_ft = fft2(kernel_up_sized);
42
43 im_conv1_ft = im_ft.*kernel_up_ft;
44 im_conv1 = ifft2(im_conv1_ft);
45
46 kernel_down_sized = zeros(m,n);
47 kernel_down_sized(1:2,1:2) = kernel_down(2:3,2:3);
48 kernel_down_sized(m,1:2) = kernel_down(1,2:3);
49 kernel_down_sized(1:2,n) = kernel_down(2:3,1);
50 kernel_down_sized(m,n) = kernel_down(1,1);
51 kernel_down_ft = fft2(kernel_down_sized);
52
53 im_conv2_ft = im_conv1_ft.*kernel_down_ft;
54 im_conv2 = ifft2(im_conv2_ft);
55
56 im_conv1_8 = uint8(zeros(m,n,3));
57 im_conv1_8(:,:,1) = uint8(im_conv1);
58 im_conv1_8(:,:,2) = uint8(im_conv1);
59 im_conv1_8(:,:,3) = uint8(im_conv1);
60 im_conv2_8 = uint8(zeros(m,n,3));
61 im_conv2_8(:,:,1) = uint8(im_conv2);
62 im_conv2_8(:,:,2) = uint8(im_conv2);
63 im_conv2_8(:,:,3) = uint8(im_conv2);
64
65 % if i==1 || i==2 % || i==9
66 % figure;
67 % image(im_conv1_8);
68 % figure;
69 % image(im_conv2_8);
70 % end;
71
72 sharpness = sum(sum(im_conv1 > 20));
73
74 oversharpness = sum(sum(im_conv1-im_conv2 > 20));
75
76 best(i-startfile+1,1) = i;
77 best(i-startfile+1,2) = sharpness;
78 best(i-startfile+1,3) = oversharpness;
79 best(i-startfile+1,4) = sharpness / max(250, oversharpness-250);
80 end;
81
82 [sbest, sx] = sort(best(:,4), 'descend');
83 best = best(sx,:);

```

6.2 Výsledky testování

Byly realizovány tři testovací scénáře.

- Výpočet čistě na CPU s maximálním využitím produkčního HW.
- Výpočet na CPU s využitím GPU pro paralelizaci vybraných procesů.
- Výpočet čistě na CPU na PC farmě.

6.2.1 Výpočet koeficientů na CPU

Výpočet na CPU byl spuštěn v pěti vláknech. Po spuštění probíhal výpočet velmi slibně. Bohužel při delším testu se výpočet začal značně propadat. Později se ukázalo, že hlavním důvodem je odložené uvolňování RAM. Jinými slovy dokud byla další volná RAM, výpočet jel velmi rychle. S potřebou uvolnit RAM se výpočet řádově snížil.

- Krátkodobý test
 - doba: 5 minut
 - počet opakování testu: 10
 - průměrně za sekundu: 205 výpočtů obou koeficientů
- Dlouhodobý test
 - doba: 6 hodin
 - počet opakování testu: 10
 - průměrně za sekundu: 31 výpočtů obou koeficientů
- Propad o 85 %

Na obrázku níže (Obr. 6.1) je výstup z profilování výpočtu na CPU. Obsahuje poměrově zvýrazněné dlouho trvající operace vůči zbytku procesu. V rámci profilování bylo zpracováno deset tisíc iterací a doby jednotlivých částí zprůměrovány.

Celková zátěž alokovaného HW byla oproti očekávání výrazně nižší. Nepodařilo se efektivně využít CPU, které dosahovalo průměrné zátěže 30 %. Hlavní problém je



Obr. 6.1 Vizualizace procesu výpočtu koeficientů na CPU

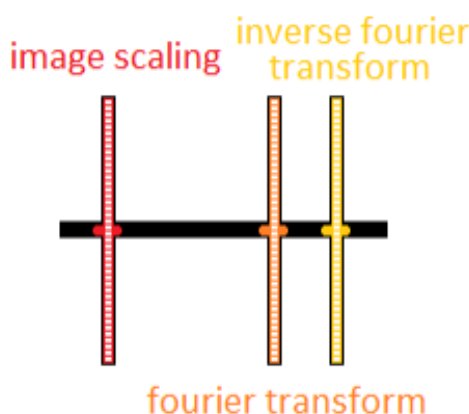
realokace operační paměti. Jednotlivé procesy pak čekají na přidělení operační paměti. Úvodní myšlenka využít hrubou sílu produkčního HW se ukázala jako nevyhovující.

6.2.2 Výpočet koeficientů na CPU s paralelizací na GPU

Výpočet byl spuštěn v jednom vlákne na CPU. Vhodné operace pro GPU jsou delegovány pro paralelní zpracování. Nejprve byl na GPU paralelizován pouze úvodní scaling obrázků, což nemělo přijatelný dopad na výsledky. Následně byly paralelizovány na GPU také prováděné transformace obrázků. Právě tento krok měl zásadní vliv na zrychlení celé operace. Je opět patrný mírný propad krátkodobého testu oproti dlouhodobému.

- Krátkodobý test
 - doba: 5 minut
 - počet opakování testu: 10
 - průměrně za sekundu: 617 výpočtů obou koeficientů
- Dlouhodobý test
 - doba: 6 hodin
 - počet opakování testu: 10
 - průměrně za sekundu: 559 výpočtů obou koeficientů
- Propad o 10 %

Na obrázku níže (Obr. 6.2) je výstup z profilování výpočtu na CPU s využitím GPU pro paralelizaci dlouhotrvajících operací na CPU. Obsahuje poměrově zvýrazněné dlouho trvající operace vůči zbytku procesu. V rámci profilování bylo zpracováno deset tisíc iterací a doby jednotlivých částí zprůměrovány.



Obr. 6.2 Vizualizace procesu výpočtu koeficientů na CPU s paralelizací na GPU

Zátěž na CPU dosahuje v průměru 5 %. Naopak GPU dosahuje zátěže cca 75 %. Otázkou zůstává, jak dlouho je schopná GPU pracovat pod tímto permanentním zatížením.

6.2.3 Výpočet koeficientů na PC farmě

Výpočet byl spuštěn v jednom pracovním vlákne s omezenou možností alokace RAM na 1 GB. Jedná se v podstatě o alternativu výpočtu obou koeficientů čistě na CPU. Tomu odpovídal i celkový průběh zpracování, který byl téměř stejný, pouze s nižší dotací vypočítaných dvojic koeficientů. Lze tedy přejít rovnou na výsledky, jelikož samotné zpracování nic nového nepřineslo.

- Krátkodobý test
 - doba: 5 minut
 - počet opakování testu: 10
 - průměrně za sekundu: 10 výpočtů obou koeficientů
- Dlouhodobý test
 - doba: 6 hodin
 - počet opakování testu: 10
 - průměrně za sekundu: 6 výpočtů obou koeficientů
- Propad o 40 %

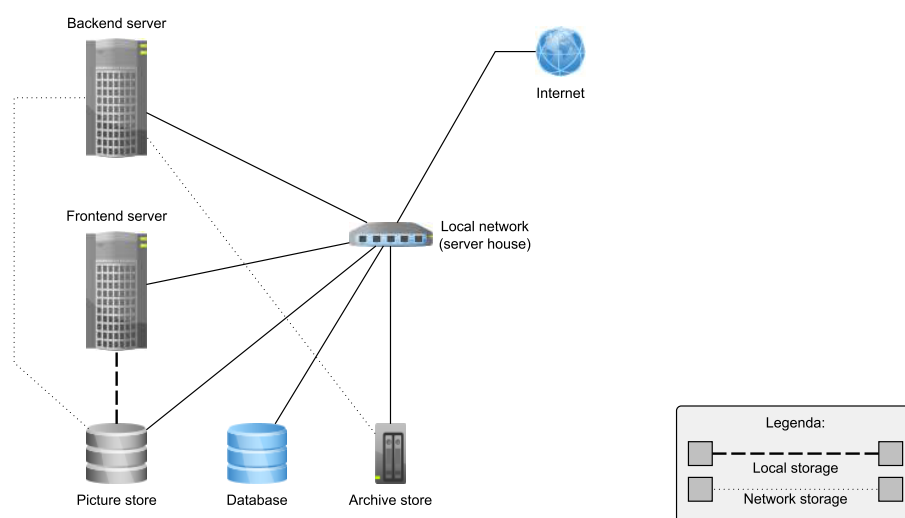
Zátěž na CPU je konstantní. Jedno jádro ze dvou jede permanentně na 100 %. Celková zátěž CPU tedy 50 %. Vzhledem k tomu, že stoj již netrpěl na realokaci paměti, je v případě PC úzkým hrdlem opravdu CPU. Ostatní sledované metriky (vytížení HDD, síťový provoz, pracovní teplota) se příliš nevychýlily z normálu.

III. PROJEKTOVÁ ČÁST

7 Příprava projektu

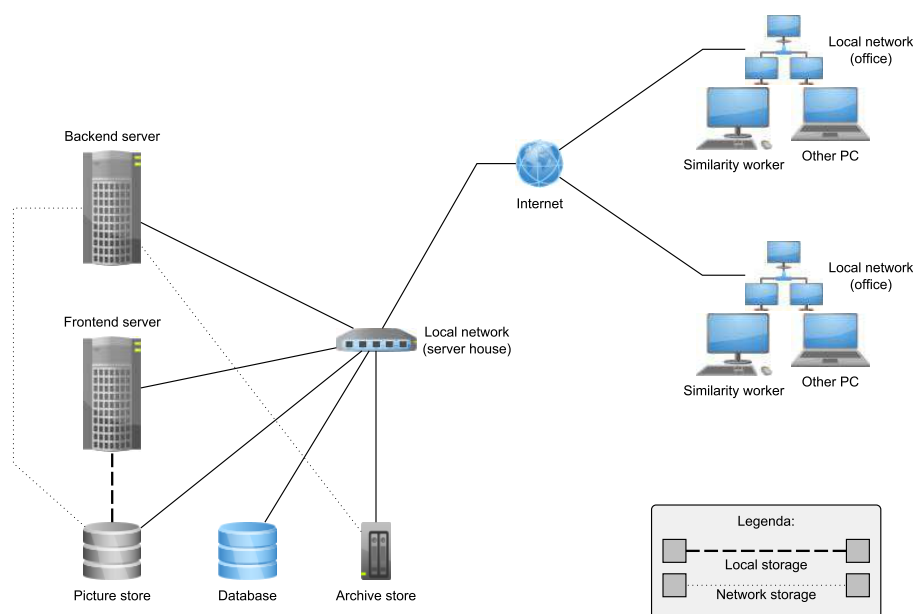
7.1 HW Architektura

7.1.1 Pohled na aktuální HW mapu



Obr. 7.1 L1 pohled aktuálního stavu

7.1.2 Pohled na plánovanou HW mapu



Obr. 7.2 L1 pohled cílového stavu

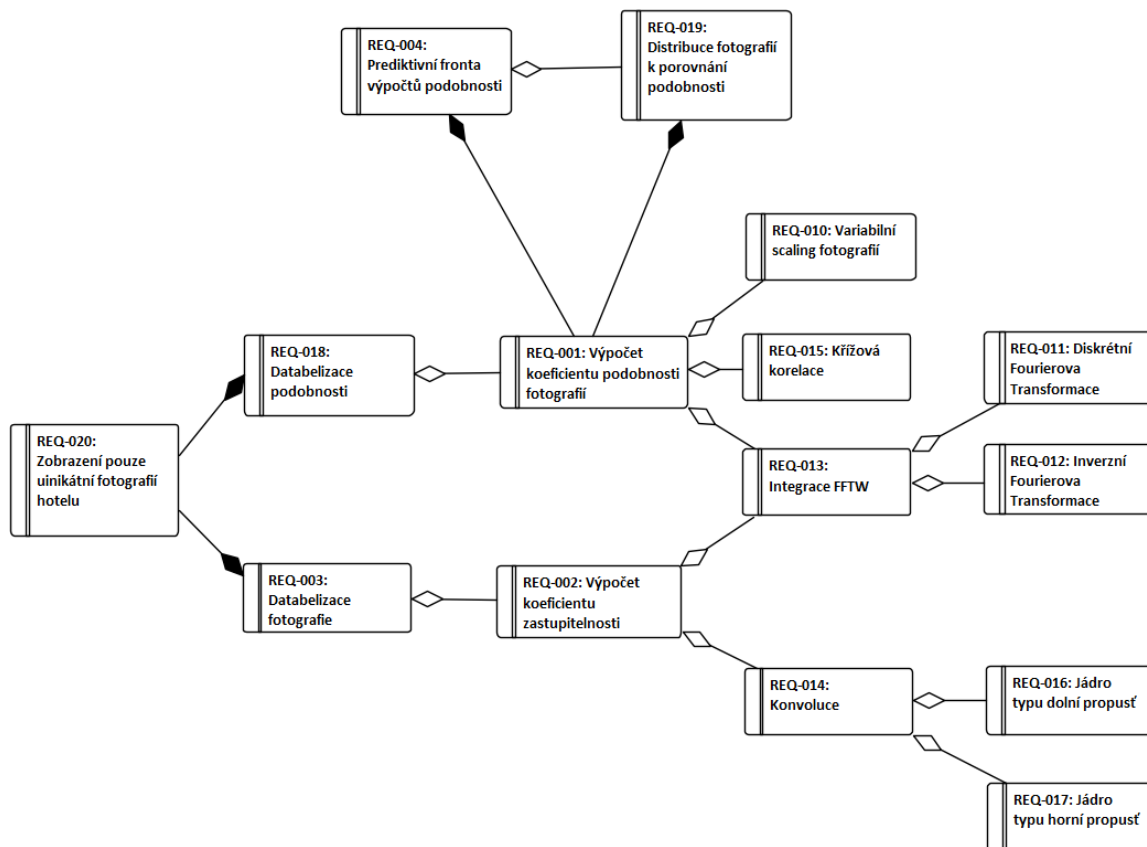
7.2 Požadavky

7.2.1 Funkční požadavky

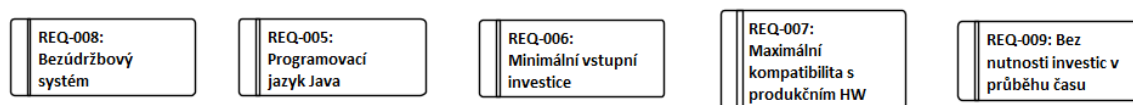
Klasifikaci funkčních požadavků systému znázorňuje obrázek (Obr. 7.3).

7.2.2 Nefunkční požadavky

Klasifikaci nefunkčních požadavků systému znázorňuje obrázek (Obr. 7.4).



Obr. 7.3 Funkční požadavky

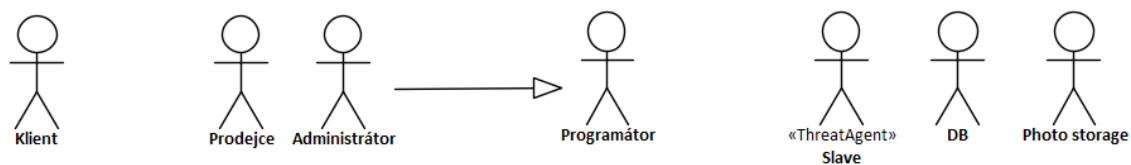


Obr. 7.4 Nefunkční požadavky

7.3 Případy použití

7.3.1 Aktéři

Klasifikace aktérů (Obr. 7.5).



Obr. 7.5 Aktéři

7.3.2 Případy užití

Klasifikaci případů použití znázorňuje obrázek (Obr. 7.6).



Obr. 7.6 Případy užití

7.4 Modely

7.4.1 Model tříd

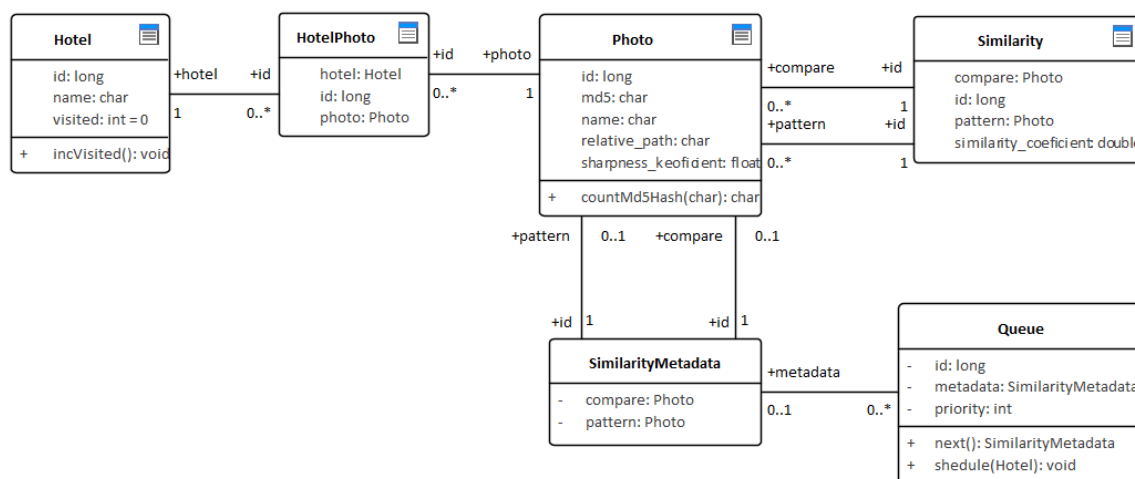
Identifikované datové modely jsou znázorněny pomocí modelu třídy na obrázku níže (Obr. 7.7)

7.4.2 Model služeb

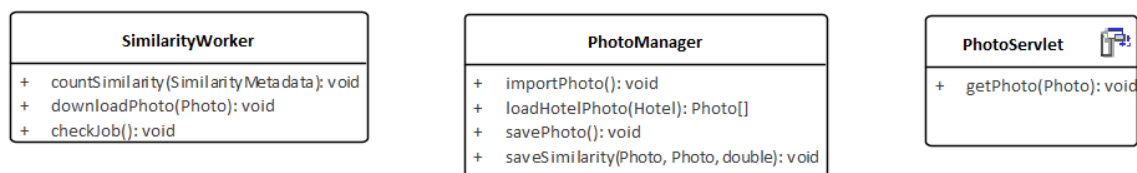
Identifikované backendové služby jsou znázorněny na obrázku níže (Obr. 7.8)

7.5 Sekvenční diagram distribuované služby

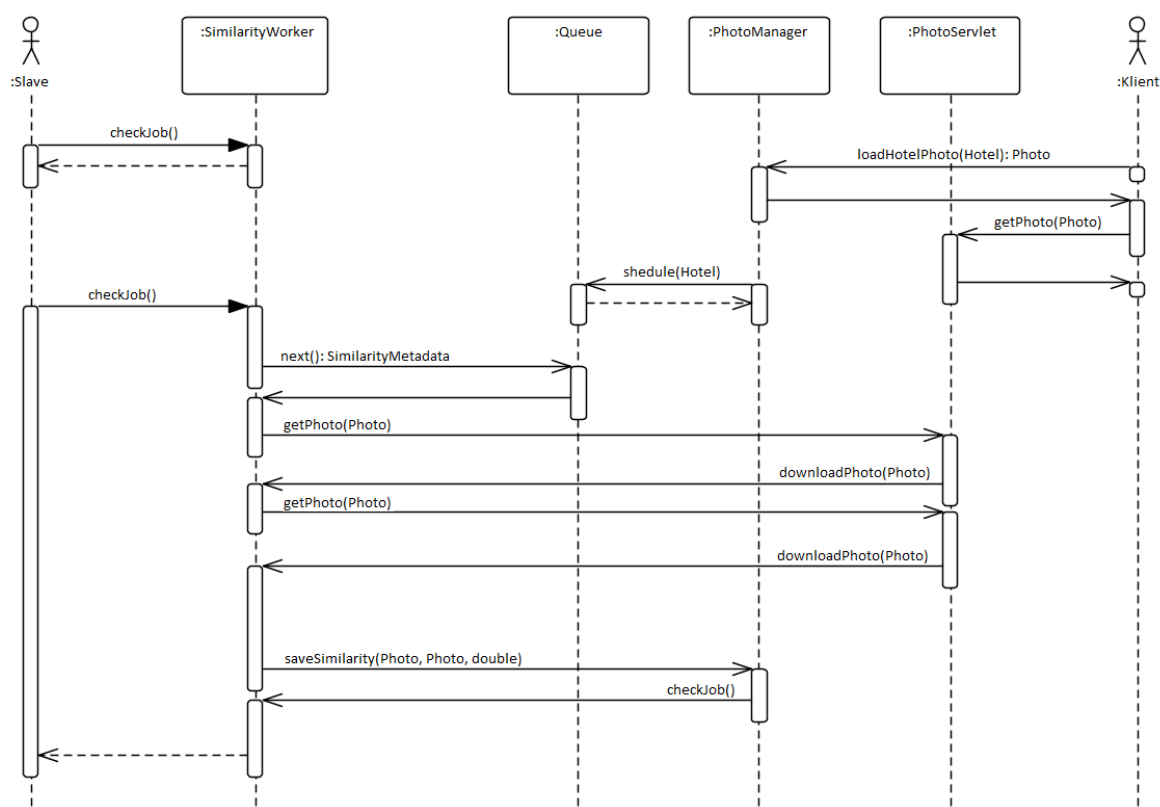
Na obrázku níže (Obr. 7.9) je znázorněn sekvenční diagram přibližující odbavení výpočtu KoP.



Obr. 7.7 Model tříd



Obr. 7.8 Model služeb



Obr. 7.9 Sekvenční diagram

8 Serverová strana distribuované služby

Současná serverová strana je webová aplikace postavená na JavaSE 7 [27] a Springu [28]. Pro komunikaci využívá SOAP API. Data ukládá do relační databáze (PostgreSQL [30]). Tato serverová strana bude modifikována. Nově zde bude

- upraven proces databelizace fotografií (výpočet KoZ),
- zkonstruována fronta importovaných fotografií čekajících na výpočet KoP,
- vystaven servlet navracející fotografie,
- vystavena webová služba pro obsluhu distribuované služby.

8.1 Databelizace fotografií

Do původního řešení byla zavedena kontrola existence totožné fotografie na základě MD5. Hodnota MD5 je ukládána do DB. Nad tímto sloupcem je aplikován unikátní index. Dále se nově při importu fotografie ukládají její vybraná metadata. Ta později lze s výhodou použít na určení KoZ přímo v SQL dotazu. Jedná se o:

- výšku a šířku v px,
- poměrný počet hran (přeostrěnost)

Výsledek po dokončení importu fotografií je patrný na obrázku níže (Obr. 8.1).

Id	Hotel	V...	Statický soubor	Cestovní kancelář	Poslední aktualizace	Hash	Šířka	Výška
28 5...	Aanari Hotel & Spa [12868]	0		Tjaereborg (DE)	2017-12-06 15:46:48	8E221E0C377E9C147CE626948D...	700	442
28 5...	Aanari Hotel & Spa [12868]	0		JT Touristik	2017-12-06 15:46:48	19D049AF8CDAEB4A0622C1EB13...	700	442

Obr. 8.1 Náhled administrace uložených fotografií

8.1.1 Výpočet poměrného počtu hran

Jedná se o matematickou operaci založenou na kovoluci, DFT a IFT. Základní torzo nosného algoritmu je vidět níže (Kod. 8.1).

Kod. 8.1 SharpnessCore.java

```

1 package cz.sa.dovolena.similarityclient.similarity;
2
3 ...
4
5 import edu.emory.mathcs.jtransforms.fft.DoubleFFT_2D;
6 import javax.imageio.ImageIO;
7
8 ...
9
10 /**
11  * Pracuje s konvolucním jádrem 3x3 typu horní propust, symetrickým, ne ideálním
12  * => propouští vysoké prostorové frekvence (nízké zahazuje)
13  *
14  * @author Dobroslav Pelc
15  */
16 public class SharpnessCounter {
17
18     ...
19
20     /**
21      * Urcí poměrný koeficient ostrosti
22      *
23      * @return koeficient ostrosti
24      */
25     public double countSharpness() {
26         log.debug("Sharpness: " + filePath);
27         File modelImageFile = new File(filePath);
28         Preconditions.checkArgument(modelImageFile.exists(), "[" + filePath + "] -> obrázek musí existovat.");
29         Preconditions.checkArgument(modelImageFile.canRead(), "[" + filePath + "] -> nad obrázkem musí být právo ke čtení.");
30         /* Konstantně skalovaný vzorový obrázek */

```

```

31 RenderedImage image = readBufferedImage(modelImageFile);
32 matrixXSize = image.getHeight();
33 matrixYSize = image.getWidth() * 2;
34 Preconditions.checkNotNull(matrixXSize >= KERNEL_SIZE && matrixYSize / 2 >= KERNEL_SIZE, "[" + filePath + "]" —> obrazek musi byt > =
    nez jadro konvolucni matice tj. " + KERNEL_SIZE + "x" + KERNEL_SIZE + " px");
35
36
37 fFTransformer = new DoubleFFT_2D(matrixXSize, matrixYSize / 2);
38 /* Obrazek preveden do matice, px = cross corel value */
39 double[][] fTMatrix = buildBrightnessMatrix(image);
40 /* Puvodni pole prebere novou hodnotu z fft transformeru */
41 fFTransformer.realForwardFull(fTMatrix);
42 /* Konvolucni jadro ve vlnovem spektru pres horni propust */
43 double[][] fTHighKernelMatrix = buildFTHighKernelMatrix();
44 /* Konvolucni jadro ve vlnovem spektru pres dolni propust */
45 double[][] fTLowKernelMatrix = buildFTLowKernelMatrix();
46 /* konvoluce ve vlnovem spektru s jadem horni propust */
47 double[][] convolutionHighMatirx = buildConvolutionMatrix(fTMatrix, fTHighKernelMatrix);
48 /*
49 * Odfiltruje sum ve vysokych frekvencich:
50 * konvoluce ve vlnovem spektru s jadem horni propust a nasledne i s
51 * jadem dolni propust
52 */
53 double[][] convolutionHighLowMatirx = buildConvolutionMatrix(convolutionHighMatirx, fTLowKernelMatrix);
54 /* zpetna (inverzni) FT */
55 /* konvoluce v puvodnim souradnicovem systemu */
56 fFTransformer.complexInverse(convolutionHighMatirx, true);
57 fFTransformer.complexInverse(convolutionHighLowMatirx, true);
58
59 sharpness = sumSharpnessPointOfBorder(convolutionHighMatirx);
60 overSharpness = sumOverSharpnessPointOfBorder(convolutionHighMatirx, convolutionHighLowMatirx);
61
62 return (double) sharpness / Math.max(OVER_SHARPNESS_ERROR_BOUND, overSharpness — OVER_SHARPNESS_ERROR_BOUND);
63
64 ...
65
66 }

```

Světlostní matice vznikne převodem fotografie (Obr. 8.2) do dvourozměrného pole (Kod. 8.2).

Kod. 8.2 BrightnessMatrix.java

```

1 /**
2  * Vytvori matici, která ma pro kazdy px obrazku vypocita prumernou svetlost
3  * a ulozi do vystupniho pole. Pokud obrazek není v pomeru x43, prazdne
4  * mista se doplni nulami.
5  *
6  * @param image vyrenderovany obrazek
7  * @return pole krizove korelace vseh px obrazku.
8  */
9 private double[][] buildBrightnessMatrix(RenderedImage image) {
10
11     Preconditions.checkNotNull(image, "Obrazek, nad kterych chcete vypocitat prumernou svetlost, nesmi byt null.");
12     final int imageWidth = image.getWidth();
13     Preconditions.checkNotNull(imageWidth <= matrixYSize / 2, "Obrazek, nad kterych chcete vypocitat prumernou svetlost, musi byt scalovany
        na x32024px.");
14     final int imageHeight = image.getHeight();
15     Preconditions.checkNotNull(imageHeight <= matrixXSize, "Obrazek, nad kterych chcete vypocitat prumernou svetlost, musi byt scalovany na
        x32024px.");
16
17     final double[][] brightnessMatrix = new double[matrixXSize][matrixYSize];
18     final RandomIter iterator = RandomIterFactory.create(image, null);
19
20     for (int x = 0; x < imageHeight; x++) {
21         for (int y = 0; y < imageWidth; y++) {
22             final double[] pixel = new double[RGB];
23             /* Pro vhodnejsi praci s matici v FFT jsme otocili vyznam indexu, z obrazku ale musim nacist porad stejne */
24             iterator.getPixel(y, x, pixel);
25
26             /* Vypocet jasu prumerneho jasu pixelu */
27             double averageBrightness = (pixel[0] + pixel[1] + pixel[2]) / 3;
28             brightnessMatrix[x][y] = averageBrightness;
29         }
30     }
31     return brightnessMatrix;
32 }

```

Matice ostrosti je matice hran, která se získá převodem světlostní matice do vlnového spektra pomocí DFT a následnou konvolucí ve vlnovém spektru (Kod. 8.3). Grafická prezentace této operace je na obrázku níže (Obr. 8.4). K docílení požadovaného stavu je nutné zvolit vhodný filtr (Tab. 8.1, (Kod. 8.3). Samotné sestavení matice ostrosti ve vlnovém spektru znázorňuje přiložený kód níže (Kod. 8.4).

Kod. 8.3 FTHighKernelMatrix.java

```

1 /* POZN: soucet vseh zapornych a kladnych koeficientu musi byt = 0. */
2 /* Rozmer konvolucniho jadra — zpravidla ctvercove. */
3 public static final int KERNEL_SIZE = 3;
4 /* Negativni koeficient konvolucniho jadra pro horni propust */
5 private static final double HIGH_KERNEL_COEF_NEGATIV = -0.0625; // -(1/16)
6 /* Kladny koeficient konvolucniho jadra pro horni propust */
7 private static final double HIGH_KERNEL_COEF_POSITIVE = 0.5;
8
9
10 private double[][] buildFTHighKernelMatrix() {
11     /* Druha matice z konvolucniho jadra */
12     double[][] kernelMatrix = new double[matrixXSize][matrixYSize];
13     /*
14     * do prvni x33 poli dame konvolucni jadro:
15     *
16     * -1/16   -1/16   -1/16
17     * -1/16    1/2    -1/16
18     * -1/16   -1/16   -1/16
19     */
20     kernelMatrix[0][0] = HIGH_KERNEL_COEF_POSITIVE;
21     kernelMatrix[0][1] = HIGH_KERNEL_COEF_NEGATIV;
22     kernelMatrix[1][0] = HIGH_KERNEL_COEF_NEGATIV;

```

```

23 kernelMatrix[1][1] = HIGH_KERNEL_COEF_NEGATIV;
24 kernelMatrix[matrixXSize - 1][0] = HIGH_KERNEL_COEF_NEGATIV;
25 kernelMatrix[matrixXSize - 1][1] = HIGH_KERNEL_COEF_NEGATIV;
26 kernelMatrix[0][(matrixYSize / 2) - 1] = HIGH_KERNEL_COEF_NEGATIV;
27 kernelMatrix[1][(matrixYSize / 2) - 1] = HIGH_KERNEL_COEF_NEGATIV;
28 kernelMatrix[matrixXSize - 1][(matrixYSize / 2) - 1] = HIGH_KERNEL_COEF_NEGATIV;
29
30 fftTransformer.realForwardFull(kernelMatrix);
31 return kernelMatrix;
32 }

```

Tab. 8.1 Jádru typu horní propust

$-1/16$	$-1/16$	$-1/16$
$-1/16$	$1/2$	$-1/16$
$-1/16$	$-1/16$	$-1/16$

Kod. 8.4 ConvolutionMatrix.java

```

1  /**
2   * Konvoluce ve vlnovém spektru.
3   *
4   * @param a matice FT z obrazku
5   * @param b matice FT konvolucního jádra
6   * @return a .* b' (vynasobi prvek po prvku a a b)
7   */
8  protected double[][] buildConvolutionMatrix(double[][] a, double[][] b) {
9      double[][] result = new double[matrixXSize][matrixYSize];
10     for (int x = 0; x < matrixXSize; x++) {
11         for (int y = 0; y < matrixYSize / 2; y++) {
12             final double realPartA = a[x][2 * y];
13             final double realPartB = b[x][2 * y];
14             final double imagPartA = a[x][2 * y + 1];
15             final double imagPartB = b[x][2 * y + 1];
16             /* realna cast vysledne matice */
17             result[x][2 * y] = realPartA * realPartB - imagPartA * imagPartB;
18             /* imaginarni cast vysledne matice */
19             result[x][2 * y + 1] = realPartA * imagPartB + imagPartA * realPartB;
20         }
21     }
22     return result;
23 }

```

Matice přeostrěnosti je matice hran za prahovou hodnotou ostrosti (přeostrěných hran). Vzniká konvolucí (Kod. 8.4) matice ostrosti s použitím jádra typu dolní propust (Tab. 8.2, Kod. 8.5). Jádro bylo sestaveno experimentálně v mnoha iteracích a jeho funkčnost byla ověřena testováním na lidech, stejně jako aplikovaná prahová hodnota míry ostrosti.

Kod. 8.5 FTLowKernelMatrix.java

```

1  /**
2   * Koefficient okolních bodů konvolucního jádra pro dolní propust */
3  private static final double LOW_KERNEL_NEIGHBOUR_COEF = 0.125; // 1/8
4  /* Středový koefficient konvolucního jádra pro dolní propust */
5  private static final double LOW_KERNEL_MIDDLE_COEF = 0.5;
6
7  private double[][] buildFTLowKernelMatrix() {
8      /**
9       * do prvních 3x3 polí dame konvolucní jádro
10      *
11      * 1/8    1/8    1/8
12      * 1/8    1/2    1/8
13      * 1/8    1/8    1/8
14      */
15      double[][] kernelMatrix = new double[matrixXSize][matrixYSize];
16
17      kernelMatrix[0][0] = LOW_KERNEL_MIDDLE_COEF;
18      kernelMatrix[0][1] = LOW_KERNEL_NEIGHBOUR_COEF;
19      kernelMatrix[1][0] = LOW_KERNEL_NEIGHBOUR_COEF;
20      kernelMatrix[matrixXSize - 1][0] = LOW_KERNEL_NEIGHBOUR_COEF;
21      kernelMatrix[0][(matrixYSize / 2) - 1] = LOW_KERNEL_NEIGHBOUR_COEF;
22
23      fftTransformer.realForwardFull(kernelMatrix);
24      return kernelMatrix;
25 }

```

Tab. 8.2 Jádru typu dolní propust

$1/8$	$1/8$	$1/8$
$1/8$	$1/2$	$1/8$
$1/8$	$1/8$	$1/8$

Poměrný počet hran je podíl sumy hran a sumy přeastřených hran. Suma hran je dána součtem jednotlivých prvků matice ostrosti po převodu zpět z vlnového spektra pomocí IFT (Kod. 8.6). Suma přeastřených hran vzniká podobně (Kod. 8.7). Jediný rozdíl je zohlednění prahového hodnoty ostrosti (respektive přeastřenosti).

Kod. 8.6 sumSharpness.java

```

1  /**
2   * Secteme body na hranach, které jsou dostatečně ostre. Za míru ostrosti
3   * považujeme ostrost obrazu, která je závislá na jeho velikosti.
4   *
5   * @param matrix matice hran
6   * @return "míra ostrosti"
7   */
8  private int sumSharpnessPointOfBorder(double[][] matrix) {
9      return sumOverSharpnessPointOfBorder(matrix, new double[matrixXSize][matrixYSize]);
10 }

```

Kod. 8.7 sumOverSharpness.java

```

1  /* Prah přeastřených hran */
2  private static final int BORDER_SHARPNESS_LIMIT = 20;
3
4  /**
5   * Četeme body na áhranch, ékter jsou čédostaten řřépeosten.
6   *
7   * @param matrixA matice hran
8   * @param matrixB matice hran bez šumu (áprojekt ídoln ípropust)
9   * @return ímra řřépeostenosti
10 */
11 private int sumOverSharpnessPointOfBorder(double[][] matrixA, double[][] matrixB) {
12     int sum = 0;
13     for (int x = 0; x < matrixXSize; x++) {
14         for (int y = 0; y < matrixYSize / 2; y++) {
15             if (matrixA[x][y * 2] - matrixB[x][y * 2] > BORDER_SHARPNESS_LIMIT) {
16                 sum++;
17             }
18         }
19     }
20     /* íMra řřépeostenosti */
21     return sum;
22 }

```

Příklad přeastřenosti fotografie prezentují obrázky níže. Z původní fotografie (Obr. 8.2) je po oddělení přeastřených hran (Obr. 8.3), ode všech hran (Obr. 8.4), stanoven jejich poměrný koeficient.

8.2 Fronta nezpracovaných obrázků

Obrázky, které čekají na výpočet KoP, lze vyčíst z DB pomocí SQL dotazu (Kod. 8.8). Priorita nezpracovaných obrázků je dána počtem zhlédnutí webové stránky hotelu. Dávka 10.000 požadavků na výpočet KoP, která je jednorázově načtena z DB, vystačí přibližně na 10 sekund (v závislosti na počtu aktivních slave). Počty shlédnutí jednotlivých hotelů se propisují do DB v reálném čase. Každá nově načtená dávka tudíž zohlední prioritu KoP.

Kod. 8.8 queue.sql

```

1  SELECT
2      pattern.photo AS pattern,
3      compare.photo AS compare,
4      h.visited AS priority
5
6  FROM hotel AS h
7  JOIN hotel_photo AS pattern ON pattern.hotel = hotel.id
8  JOIN hotel_photo AS compare ON compare.hotel = hotel.id AND pattern.photo <> compare.photo
9
10 /* Hledáme pouze fotky, u kterých není spočítána podobnost; záleží na poradí! */
11 LEFT JOIN similarity AS s ON (s.pattern = pattern.photo AND s.compare = compare.photo)
12 OR (s.pattern = compare.photo AND s.compare = pattern.photo)
13
14 WHERE s IS NULL
15 ORDER BY h.visited DESC NULL LAST
16 LIMIT 10000;

```

8.3 Servlet pro stažení obrázků

Servlet je součástí serverové strany. Je realizován pomocí spring bean [36] typu singleton [2] (Obr. 8.5). Na základě vstupní URL adresy [34] (GET dotazu [35]) určí parametr md5. Pomocí něj vyčte z DB relativní cestu na datovém, síťovém úložišti, kde se fotografie nachází (Kod. 8.9).

```
1 SELECT relative_path  
2 FROM photo  
3 WHERE md5 = $arg;
```

8.4 SOAP API

Kompletní komunikační rozhraní ve formátu WSDL [29] je k dispozici na přiloženém CD. Vybranou část WSDL znázorňuje obrázek níže (Obr. 8.6).

8.4.1 Autentizace

Aby bylo možné konzumovat vystavené webové služby, je nejprve nutné provést autentizaci. K těmto účelům je použita technika CRAM [31] (někdy také jako challenge–response).

attach je metoda (Obr. 8.7), která slouží k navázání vyhrazeného komunikačního kanálu (relace nebo také session [32]) mezi serverem a strojem pro odbavování distribuovaných požadavků (dále jen slave). Jedná se nezabezpečenou metodu (nevyžaduje autentizaci). Slouží pouze k zaregistrování *slave_id* (jedinečný identifikátor slave) na straně serveru.

challenge je metoda, kterou slave žádá server o vytvoření zabezpečeného komunikačního kanálu. Server na tento požadavek reaguje pouze v případě, že se slave prokáže platným *slave_id*. V odpovědi server vrací vygenerovanou sůl (pseudonáhodné hexadecimální číslo [33]) potřebnou pro další komunikaci (Obr. 8.8).

challengeLogin je metoda pro autentizaci slave. Server i slave provedou nezávisle na sobě výpočet hashe (s přednastaveným hashovacím algoritmem) za použití soli. Slave odešle data (Obr. 8.9) a pokud jsou ve shodě se serverovými, je relace tohoto slave nadále autentizována.

8.4.2 Distribuce požadavků na výpočet KoP

Autentizovaný slave může zažádat o metadata k výpočtu KoP. Pokud jsou ve frontě na výpočet KoP zařazeny nějaké požadavky, jsou vráceny. Jakmile je slave vyhodnotí, odesílá výsledek zpět na server.

getDataToCompare je metoda určená pro distribuci výpočtu (metadat k výpočtu) KoP (Obr. 8.10).

submitResult je metoda určená pro odeslání výsledku distribuovaného výpočtu KoP zpět na server (Obr. 8.11).



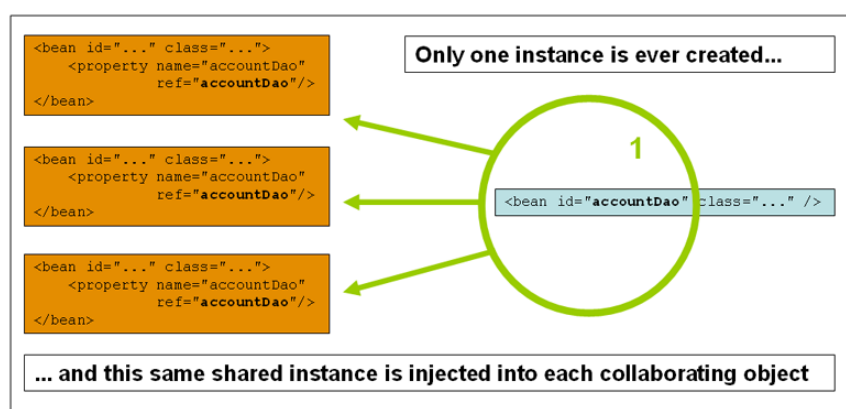
Obr. 8.2 Fotografie před úpravou



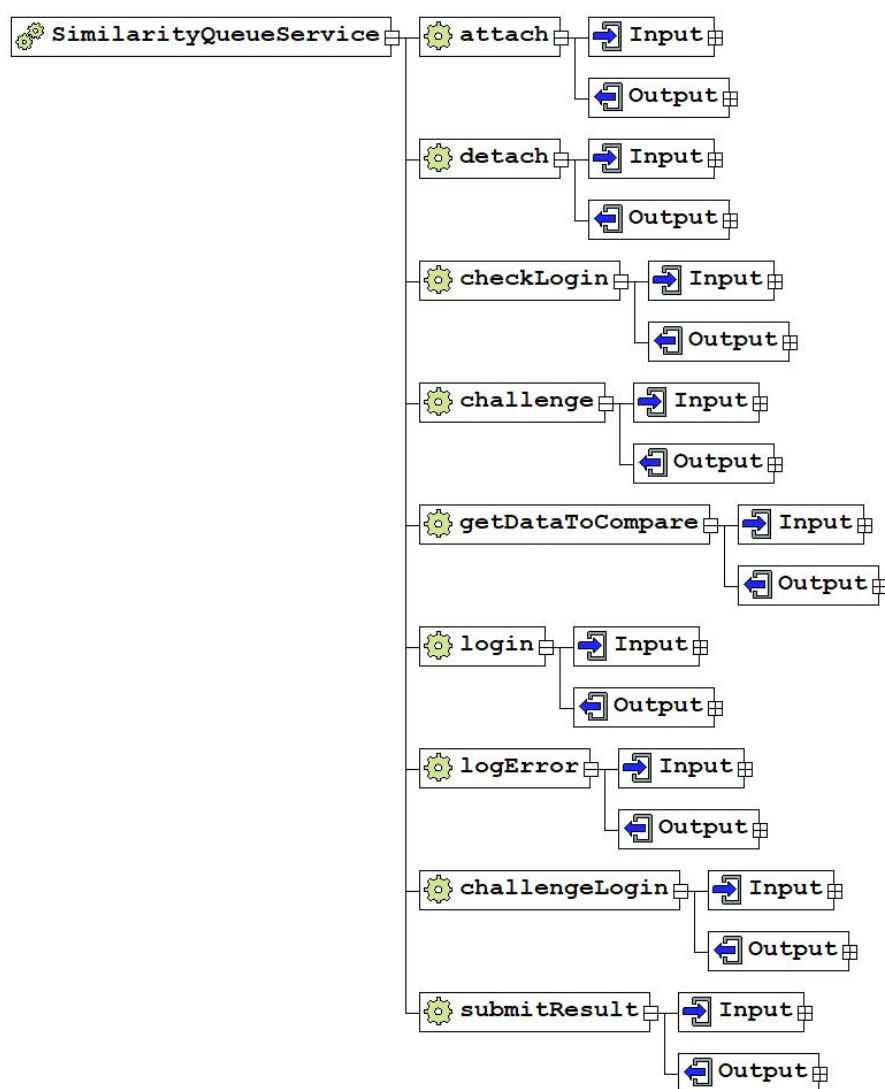
Obr. 8.3 Vizualizace matice přestřehých hran



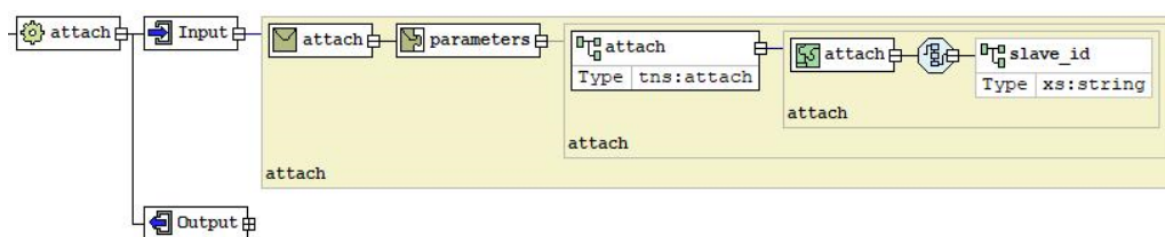
Obr. 8.4 Vizualizace matice hran



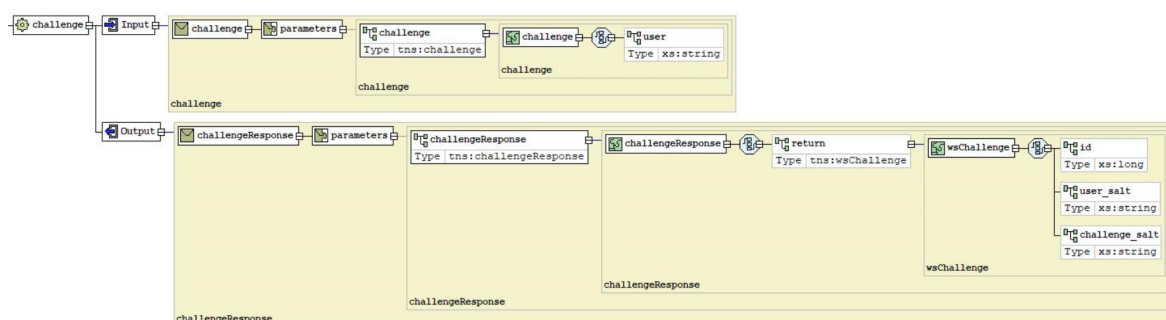
Obr. 8.5 Ukázka použití singleton [37]



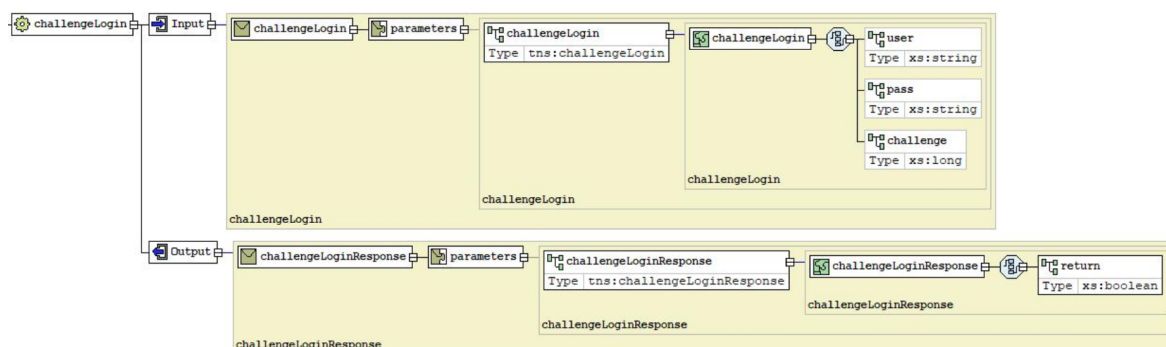
Obr. 8.6 Vizualizace komunikačních metod distribuované služby



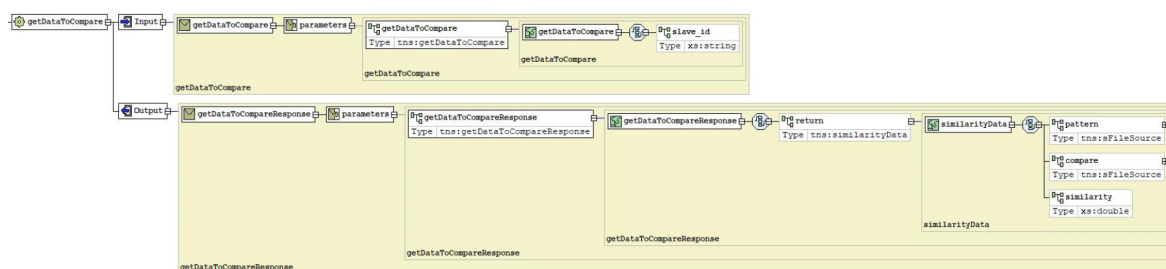
Obr. 8.7 Detail metody attach



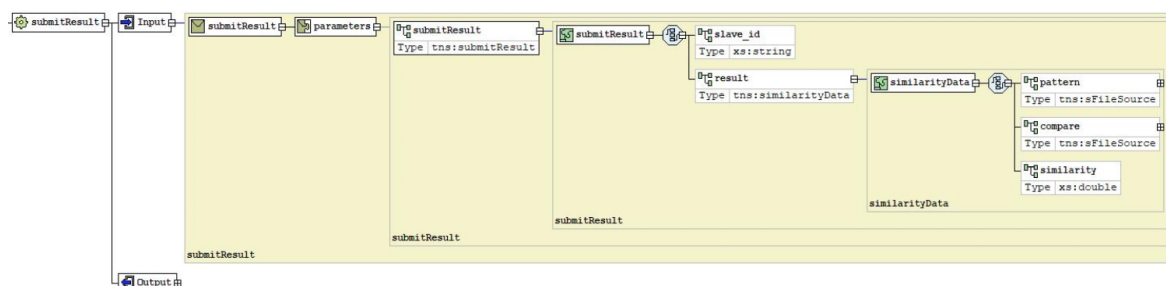
Obr. 8.8 Detail metody challenge



Obr. 8.9 Detail metody challengeLogin



Obr. 8.10 Detail metody getDataToCompare



Obr. 8.11 Detail metody submitResult

9 Klientská strana distribuované služby

Výsledná komponenta pro výpočet KoP je JAR [39] knihovna optimalizovaná pro běh na kancelářském PC. Přesto, že její spuštění je plánováno v čase, kdy je stroj nevyužívaný uživatelem, je možné pohodlně využívat PC i v době, kdy je komponenta aktivní.

Není k dispozici žádné GUI [40]. Knihovna si po startu vytvoří v domovském adresáři uživatele pracovní adresář. V něm se nachází veškerá data i konfigurace, která jsou s komponentou svázána. Je zde potřeba nastavit zejména adresu serveru cílové služby, uživatelské jméno a heslo pro komunikaci aj.

Po navázání spojení s cílovým serverem běží v nekonečné smyčce kontrola nezpracovaných požadavků na výpočet KoP a jejich případně odbavení. Pokud dojde k výpadku odezvy cílového serveru, zkouší se knihovna opakovaně připojit v pravidelných intervalech po dobu 15 minut.

9.1 Distribuovaný výpočet KoP

V případě úspěšného stažení metadat požadavku na výpočet KoP a následného fyzického stažení samotných fotografií, je zahájen výpočet KoP.

Úprava velikosti fotografie se provádí poměrným zvětšením / zmenšením na referenční velikost (Kod. 9.1) podle delší strany vzorové fotografie (při zachování poměru stran). Pokud referenční obrázek poměrem stran neodpovídá vzorovému, je doplněn nulami. Algoritmus použitý pro scaling [41] je na bázi interpolace blízkeho okolí (Nearest-neighbor interpolation [42]) a to zejména díky rychlosti provedení.

Kod. 9.1 scale.java

```
1 import javax.media.jai.JAI;
2 ...
3 /* Pozadovana sirka obrazku */
4 protected static final int BASE_SIZE_WIDTH = 320;
5 /* Pozadovana vyska obrazku */
6 protected static final int BASE_SIZE_HEIGHT = 240;
7 ...
8 /**
9  * Vratí vyrenderovaný obrázek v nových rozměrech. Kontroluje větší stranu,
10  * podle které určí poměr. NEDEFORMUJE poměr stran => Výsledný obrázek je
11  * ještě potřeba doplnit nulama, protože nemusí odpovídat přesně zadaným
12  * rozměrům v obou osách. scaledImage@param image originální obrázek
13  *
14  * @param scaleToWidth pozadovaný sirka, do které se má obrázek skalovat
15  * @param scaleToHeight pozadovaná výška, do které se má obrázek skalovat
16  *
17  * @return obrázek v nových rozměrech
18  */
19 protected RenderedImage scale(RenderedImage image, int scaleToWidth, int scaleToHeight) {
20     RenderedOp rescale = null;
21     final int originalWidth = image.getWidth();
22     final int originalHeight = image.getHeight();
23
24     float widthRatio = (float) scaleToWidth / (float) originalWidth;
25     float heightRatio = (float) scaleToHeight / (float) originalHeight;
26
27     float ratio = Math.min(widthRatio, heightRatio);
28
29     // Scales the original image
30     ParameterBlock pb = new ParameterBlock();
31     pb.addSource(image);
32     pb.add(ratio);
33     pb.add(ratio);
34     pb.add(0.0f);
35     pb.add(0.0f);
36     pb.add(new InterpolationNearest());
37
38     // Creates a new, scaled image and uses it on the DisplayJAI component
39     rescale = JAI.create("scale", pb);
40
41     return rescale;
42 }
```

Světlostní matice se tvoří stejně jako u výpočtu KoZ. Jediný rozdíl je v tom, že nyní se převádí do světlostní matice zmenšený obrázek (nikoliv originál v plné velikosti). Může tedy dojít k lehké modifikaci světlostní matice na základě zvoleného algoritmu pro scaling [41] obrázku.

Normalizace světlostní matice převedené pomocí DFT do vlnového spektra připraví matici vzorového i referenčního obrázku na křížovou korelaci (Kod. 9.2). Podělí reálnou i imaginární část absolutní hodnotou komplexního čísla.

Kod. 9.2 NormalizedFTMatrix.java

```

1  protected double[][] buildNormalizedFTMatrix(RenderedImage scaledModelImage) {
2      /* Obrazek převedeny do matice, px = cross corel value */
3      double[][] matrixImage = buildBrightnessMatrix(scaledModelImage);
4      /* Obrazek je již převedeny furierovou transformací */
5      /* Puvodní pole prebere novou hodnotu */
6      fFTTransformer.realForwardFull(matrixImage);
7      /* Normování Fourierovi transformace obrazku */
8      for (int x = 0; x < matrixXSize; x++) {
9          for (int y = 0; y < matrixYSize / 2; y++) {
10             /* bereme suče y => realne cislo */
11             final int realPartOfMatrixFieldY = 2 * y;
12             final double realPart = matrixImage[x][realPartOfMatrixFieldY];
13
14             /* bereme liche y => imaginarni cislo */
15             final int imagPartOfMatrixFieldY = 2 * y + 1;
16             final double imagPart = matrixImage[x][imagPartOfMatrixFieldY];
17
18             /* absolutni hodnota komplexního čísla */
19             double absoluteMatrixFieldValue = Math.sqrt(imagPart * imagPart + realPart * realPart);
20
21             /*
22              * Pokud vyjde nula, tak obecně doslo k tomu, že pro nějakou
23              * mocninu dvou 2^k je v nějakém radku nebo sloupci každé
24              * 2^k-tý pixel stejný
25              */
26             if (absoluteMatrixFieldValue != 0) {
27                 Preconditions.checkNotNull(absoluteMatrixFieldValue != 0, "Nulou nelze delit.");
28                 matrixImage[x][realPartOfMatrixFieldY] /= absoluteMatrixFieldValue;
29                 matrixImage[x][imagPartOfMatrixFieldY] /= absoluteMatrixFieldValue;
30             }
31         }
32     }
33     return matrixImage;
34 }

```

Křížová korelace vytvoří výslednou matici z matice vzorového obrázku a komplexně sdružené matice referenčního obrázku (Kod. 9.3). Matice se stále ještě nachází ve vlnovém spektru.

Kod. 9.3 CrossCorrelation.java

```

1  /**
2   * Křížová korelace ve vlnovém spektru.
3   *
4   * @param a normovaná matice FT vzoru
5   * @param b normovaná matice FT obrazu
6   * @return a .* b' (vynásobím prvek po prvku a a b, přičemž b je komplexně
7   *         združená)
8   */
9  protected double[][] crossCorrelation(double[][] a, double[][] b) {
10     double[][] result = new double[matrixXSize][matrixYSize];
11     for (int x = 0; x < matrixXSize; x++) {
12         for (int y = 0; y < matrixYSize / 2; y++) {
13             final double realPartA = a[x][2 * y];
14             final double realPartB = b[x][2 * y];
15             final double imagPartA = a[x][2 * y + 1];
16             final double imagPartB = b[x][2 * y + 1];
17             /* realna cast vysledne matice */
18             result[x][2 * y] = realPartA * realPartB + imagPartA * imagPartB;
19             /* imaginarni cast vysledne matice */
20             result[x][2 * y + 1] = realPartA * imagPartB - imagPartA * realPartB;
21         }
22     }
23     return result;
24 }

```

KoP udává největší číselná hodnota ve výsledné matici křížové korelace zpět po převodu z vlnového spektra pomocí IFT (Kod. 9.4).

Kod. 9.4 maxCorrelationCoefficient.java

```

1  public double maxCorrelationCoefficient(double[][] matrix) {
2      double max = 0;
3      for (int x = 0; x < matrixXSize; x++) {
4          for (int y = 0; y < matrixYSize / 2; y++) {
5              max = Math.max(
6                  Math.abs(matrix[x][y * 2]),
7                  max);
8          }
9      }
10     return max;
11 }
12

```

Zadání práce bylo splněno ve všech bodech. Výsledný produkt byl spuštěn v produkčním provozu. Možným námětem na rozšíření je podpora CMYK [38] formátu barev. Aplikované algoritmy jsou optimalizované na zpracování RGB formátu barev.

Během realizace byl identifikován možný bod rozšíření algoritmu na výpočet KoP. Jak se ukázalo, některé fotografie jsou vůči sobě pootočené v prostoru. V úvodu práce je této problematice věnován odstavec v teoretické části. Lidské oko vyhodnotí tyto fotografie jako duplicitní. Použité algoritmy tuto problematiku neřeší.

Níže jsou výsledky testovacího vzorku (35 fotografií vybraného hotelu). Na základě určeného koeficientu podobnosti KoP (Tab. 9.1) lze rozdělit fotografie na obsahově unikátní a duplicitní. Celkem bylo v testovacím vzorku nalezeno 19 obsahově unikátních fotografií.

Tab. 9.1 Výsledky porovnání KoP 35 hotelových fotografií

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	1,000	0,645	0,440	0,669	0,407	0,948	0,174	0,655	0,018	0,018	0,018	0,019	0,019	0,016	0,017	0,016	0,017	0,017	0,016	0,019	0,018	0,017	0,016	0,018	0,018	0,018	0,020	0,018	0,017	0,018	0,018	0,018	0,017	0,016	0,016
2	0,645	1,000	0,426	0,925	0,521	0,638	0,161	0,876	0,016	0,018	0,017	0,016	0,017	0,016	0,018	0,018	0,017	0,017	0,017	0,017	0,017	0,016	0,018	0,017	0,017	0,016	0,017	0,018	0,018	0,018	0,018	0,016	0,017	0,017	
3	0,440	0,426	1,000	0,448	0,520	0,432	0,140	0,421	0,018	0,018	0,020	0,018	0,017	0,016	0,017	0,018	0,018	0,016	0,018	0,018	0,018	0,017	0,017	0,017	0,017	0,019	0,016	0,017	0,018	0,018	0,018	0,018	0,016	0,017	0,017
4	0,669	0,925	0,448	1,000	0,547	0,661	0,172	0,877	0,017	0,020	0,017	0,015	0,017	0,016	0,018	0,018	0,016	0,017	0,016	0,017	0,018	0,016	0,016	0,017	0,019	0,015	0,017	0,017	0,016	0,018	0,016	0,017	0,015	0,017	0,017
5	0,407	0,521	0,520	0,547	1,000	0,401	0,151	0,522	0,018	0,019	0,021	0,016	0,019	0,016	0,017	0,017	0,018	0,018	0,018	0,018	0,017	0,017	0,017	0,017	0,016	0,019	0,018	0,018	0,017	0,017	0,017	0,020	0,016	0,017	
6	0,948	0,638	0,432	0,661	0,401	1,000	0,171	0,650	0,020	0,017	0,017	0,017	0,019	0,015	0,017	0,016	0,017	0,016	0,016	0,018	0,017	0,018	0,017	0,016	0,018	0,016	0,022	0,019	0,015	0,017	0,019	0,018	0,016	0,017	0,017
7	0,174	0,161	0,140	0,172	0,151	0,171	1,000	0,173	0,017	0,018	0,016	0,016	0,019	0,019	0,018	0,017	0,016	0,019	0,018	0,016	0,018	0,016	0,018	0,018	0,020	0,019	0,018	0,017	0,016	0,018	0,016	0,016	0,018	0,019	
8	0,655	0,876	0,421	0,877	0,522	0,650	0,173	1,000	0,016	0,019	0,018	0,015	0,016	0,016	0,016	0,017	0,017	0,018	0,016	0,018	0,017	0,016	0,018	0,017	0,018	0,016	0,018	0,018	0,018	0,017	0,017	0,015	0,015	0,018	0,018
9	0,018	0,016	0,018	0,017	0,018	0,020	0,017	0,016	1,000	0,807	0,217	0,021	0,019	0,019	0,019	0,018	0,017	0,016	0,015	0,020	0,022	0,020	0,023	0,018	0,020	0,018	0,018	0,020	0,019	0,023	0,020	0,019	0,017	0,017	0,017
10	0,018	0,018	0,018	0,020	0,019	0,017	0,018	0,019	0,807	1,000	0,232	0,022	0,017	0,019	0,019	0,018	0,018	0,016	0,019	0,022	0,018	0,020	0,020	0,019	0,017	0,017	0,019	0,022	0,022	0,018	0,018	0,017	0,017	0,017	
11	0,018	0,017	0,020	0,017	0,021	0,017	0,016	0,018	0,217	0,232	1,000	0,021	0,018	0,019	0,020	0,024	0,018	0,018	0,016	0,018	0,020	0,021	0,021	0,021	0,020	0,018	0,018	0,020	0,020	0,022	0,018	0,019	0,017	0,018	0,018
12	0,017	0,016	0,018	0,015	0,016	0,017	0,016	0,015	0,021	0,022	0,021	1,000	0,020	0,018	0,017	0,017	0,016	0,018	0,017	0,020	0,016	0,017	0,018	0,017	0,016	0,018	0,017	0,020	0,027	0,016	0,018	0,017	0,016	0,017	
13	0,019	0,017	0,017	0,019	0,019	0,019	0,019	0,016	0,019	0,019	0,018	0,020	1,000	0,023	0,572	0,174	0,016	0,016	0,017	0,017	0,019	0,021	0,019	0,019	0,020	0,016	0,018	0,028	0,018	0,019	0,020	0,017	0,018	0,017	
14	0,016	0,016	0,016	0,016	0,016	0,015	0,019	0,016	0,019	0,019	0,018	0,023	1,000	0,021	0,022	0,016	0,017	0,018	0,018	0,016	0,017	0,017	0,019	0,016	0,020	0,017	0,020	0,023	0,018	0,020	0,017	0,021	0,020		
15	0,017	0,018	0,017	0,018	0,017	0,017	0,018	0,016	0,019	0,019	0,020	0,017	0,572	0,021	1,000	0,158	0,017	0,017	0,018	0,018	0,019	0,021	0,017	0,018	0,017	0,016	0,020	0,023	0,020	0,019	0,020	0,016	0,018	0,018	
16	0,016	0,018	0,018	0,018	0,017	0,016	0,017	0,017	0,017	0,018	0,024	0,017	0,174	0,022	0,158	1,000	0,019	0,019	0,019	0,019	0,018	0,018	0,021	0,020	0,021	0,017	0,017	0,021	0,020	0,019	0,020	0,018	0,019	0,017	
17	0,017	0,017	0,018	0,016	0,018	0,017	0,016	0,017	0,017	0,018	0,018	0,017	0,016	0,016	0,017	0,019	1,000	0,160	0,154	0,017	0,017	0,016	0,018	0,017	0,017	0,017	0,018	0,018	0,017	0,017	0,017	0,017	0,016	0,016	
18	0,017	0,017	0,016	0,017	0,018	0,016	0,019	0,018	0,018	0,016	0,016	0,016	0,016	0,016	0,017	0,019	0,160	1,000	0,810	0,020	0,017	0,019	0,018	0,016	0,021	0,016	0,017	0,018	0,018	0,017	0,017	0,017	0,016	0,016	
19	0,016	0,017	0,018	0,016	0,018	0,016	0,016	0,016	0,015	0,016	0,016	0,018	0,017	0,018	0,017	0,019	0,154	0,810	1,000	0,017	0,018	0,016	0,016	0,017	0,019	0,015	0,019	0,017	0,016	0,017	0,019	0,017	0,015	0,016	
20	0,019	0,017	0,018	0,017	0,018	0,018	0,016	0,018	0,020	0,019	0,018	0,017	0,017	0,018	0,018	0,019	0,017	0,020	0,017	1,000	0,019	0,018	0,018	0,017	0,017	0,017	0,018	0,019	0,017	0,016	0,019	0,016	0,016	0,016	
21	0,018	0,017	0,017	0,018	0,017	0,017	0,018	0,017	0,022	0,022	0,020	0,020	0,019	0,016	0,018	0,018	0,017	0,017	0,018	0,019	1,000	0,102	0,017	0,018	0,019	0,017	0,016	0,019	0,018	0,019	0,018	0,017	0,017	0,018	0,018
22	0,017	0,016	0,017	0,016	0,017	0,018	0,016	0,016	0,020	0,020	0,018	0,021	0,016	0,021	0,017	0,019	0,018	0,016	0,019	0,016	0,018	0,102	1,000	0,020	0,016	0,022	0,018	0,016	0,019	0,021	0,017	0,020	0,019	0,018	0,017
23	0,016	0,018	0,017	0,016	0,017	0,017	0,016	0,018	0,023	0,020	0,021	0,017	0,019	0,017	0,021	0,021	0,018	0,018	0,016	0,018	0,017	0,020	1,000	0,524	0,019	0,016	0,016	0,019	0,020	0,020	0,018	0,018	0,021	0,020	
24	0,018	0,017	0,017	0,017	0,016	0,016	0,018	0,017	0,018	0,020	0,021	0,018	0,019	0,017	0,017	0,020	0,017	0,016	0,017	0,017	0,018	0,016	0,524	1,000	0,018	0,018	0,017	0,017	0,020	0,018	0,019	0,016	0,018	0,018	
25	0,018	0,017	0,017	0,019	0,018	0,018	0,020	0,018	0,020	0,019	0,020	0,017	0,020	0,019	0,018	0,021	0,017	0,021	0,019	0,017	0,019	0,022	0,019	0,018	1,000	0,018	0,017	0,021	0,019	0,018	0,018	0,020	0,017	0,020	
26	0,018	0,016	0,019	0,015	0,016	0,016	0,019	0,016	0,018	0,017	0,018	0,016	0,016	0,016	0,016	0,017	0,017	0,016	0,015	0,017	0,017	0,018	0,016	0,018	1,000	0,017	0,016	0,015	0,020	0,016	0,016	0,017	0,016	0,016	
27	0,020	0,017	0,016	0,017	0,019	0,022	0,018	0,018	0,018	0,017	0,018	0,018	0,018	0,020	0,016	0,017	0,017	0,017	0,019	0,018	0,016	0,016	0,016	0,017	0,017	0,017	1,000	0,017	0,018	0,022	0,017	0,016	0,015	0,018	
28	0,018	0,018	0,017	0,017	0,018	0,019	0,017	0,018	0,020	0,019	0,020	0,017	0,028	0,017	0,020	0,021	0,018	0,018	0,017	0,019	0,019	0,019	0,019	0,017	0,021	0,016	0,017	1,000	0,022	0,019	0,018	0,018	0,016	0,017	
29	0,017	0,017	0,018	0,016	0,018	0,015	0,016	0,017	0,019	0,022	0,020	0,020	0,018	0,018	0,020	0,023	0,020	0,018	0,018	0,016	0,017	0,018	0,021	0,019	0,020	0,019	0,015	0,018	0,022	1,000	0,021	0,019	0,018	0,019	0,018
30	0,018	0,018	0,018	0,018	0,017	0,017	0,016	0,017	0,023	0,022	0,022	0,027	0,019	0,023	0,020	0,019	0,017	0,017	0,016	0,019	0,019	0,017	0,020	0,018	0,018	0,020	0,022	0,019	0,021	1,000	0,019	0,017	0,018	0,017	
31	0,018	0,016	0,018	0,016	0,017	0,019	0,018	0,017	0,020	0,018	0,018	0,016	0,020	0,018	0,019	0,020	0,017	0,017	0,019	0,019	0,018	0,020	0,020	0,019	0,018	0,016	0,017	0,018	0,019	0,019	1,000	0,019	0,016	0,019	
32	0,017	0,018	0,018	0,017	0,017	0,018	0,016	0,015	0,019	0,018	0,019	0,018	0,017	0,020	0,020	0,018	0,017	0,018	0,017	0,019	0,017	0,019	0,018	0,016	0,020	0,016	0,016	0,018	0,018	0,017	0,019	1,000	0,017	0,018	
33	0,016	0,016	0,016	0,015	0,020	0,016	0,016	0,015	0,017	0,017	0,017	0,017	0,018	0,017	0,016	0,019	0,017	0,017	0,015	0,016	0,017	0,018	0,018	0,017	0,017	0,015	0,018	0,019	0,						



Obr. 9.1 Fotografie 1



Obr. 9.2 Fotografie 2



Obr. 9.3 Fotografie 3



Obr. 9.4 Fotografie 4



Obr. 9.5 Fotografie 5



Obr. 9.6 Fotografie 6



Obr. 9.7 Fotografie 7



Obr. 9.8 Fotografie 8

Tab. 9.2 Výsledky porovnání KoZ na 8 podobných fotografiích dle KoP

	Sharpens	OverSharpness	KoZ
Fotografie 1	8922	432	35,69
Fotografie 2	13228	662	32,11
Fotografie 3	1431	1	5,72
Fotografie 4	5963	115	23,85
Fotografie 5	412	0	1,65
Fotografie 6	10900	748	21,89
Fotografie 7	261	0	1,04
Fotografie 8	8586	437	34,34



Obr. 9.9 Fotografie s nejvyšším KoZ (nejkvalitnější)



Obr. 9.10 Fotografie s nejnižším KoZ (nejméně kvalitní)

SEZNAM POUŽITÉ LITERATURY

- [1] ECKEL, Bruce. *Thinking in Java*. 4th ed. Upper Saddle River, NJ: Prentice Hall, c2006. ISBN 01-318-7248-6.
- [2] WALLS, Craig. *Spring in action*. Fourth edition. Texas: Manning, 2013. ISBN 978-161-7291-203.
- [3] HUNT, Charlie a Binu JOHN. *Java performance*. Upper Saddle River, NJ: Addison-Wesley, c2012. Java series. ISBN 01-371-4252-8.
- [4] PERŮTKA, Karel. *MATLAB - Základy pro studenty automatizace a informačních technologií*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-731-8355-2.
- [5] DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#*. Praha: BEN - technická literatura, 2008. ISBN 978-80-7300-233-6.
- [6] MD5. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-08]. Dostupné z: <https://en.wikipedia.org/wiki/MD5>.
- [7] *Computer vision* [online]. [cit. 2018-05-08]. Dostupný z: http://midas.uamt.feec.vutbr.cz/ZVS/Exercise02/content_cz.php.
- [8] Pixel. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-08]. Dostupné z: <https://cs.wikipedia.org/wiki/Pixel>.
- [9] RGB. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-08]. Dostupné z: <https://cs.wikipedia.org/wiki/RGB>.
- [10] *IEEE Transactions on Computers* [online]. 1972, **C-21**(2), 179–186 [cit. 2018-05-08]. DOI: 10.1109/TC.1972.5008923. ISSN 0018-9340. Dostupné z: <http://ieeexplore.ieee.org/document/5008923/>.
- [11] Korelace. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-09]. Dostupné z: <https://cs.wikipedia.org/wiki/Korelace>.
- [12] Cross-correlation. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-09]. Dostupné z: <https://en.wikipedia.org/wiki/Cross-correlation>.
- [13] *FFTW* [online]. [cit. 2018-05-08]. Dostupný z: <http://www.fftw.org>.
- [14] The Design and Implementation of FFTW3. In: *Proceedings of the IEEE* [online]. 2005, **93**(2), s. 216–231 [cit. 2018-05-08]. DOI: 10.1109/JPROC.2004.840301. ISSN 0018-9219. Dostupné z: <http://ieeexplore.ieee.org/document/1386650>.

-
- [15] *IEEE Transactions on Image Processing* [online]. 5(8), 1266-1271 [cit. 2018-05-08]. DOI: 10.1109/83.506761. ISSN 10577149. Dostupné z: <http://ieeexplore.ieee.org/document/506761/>.
- [16] *Shared Scientific Toolbox in Java* [online]. [cit. 2018-05-08]. Dostupné z: <http://freshmeat.sourceforge.net/projects/shared>.
- [17] *CMake* [online]. [cit. 2018-05-08]. Dostupný z: <https://cmake.org>.
- [18] Wrapper. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-09]. Dostupné z: <https://en.wikipedia.org/wiki/Wrapper>.
- [19] In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: <https://cs.wikipedia.org/wiki/Konvoluce>.
- [20] Detekce hran. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: https://cs.wikipedia.org/wiki/Detekce_hran.
- [21] SOAP. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: <https://en.wikipedia.org/wiki/SOAP>.
- [22] Centrální procesorová jednotka. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: https://cs.wikipedia.org/wiki/Centr%C3%A1ln%C3%AD_procesorov%C3%A1_jednotka.
- [23] YANG, Jianchao, Kai YU, Yihong GONG a Thomas HUANG. *Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification* [online]. NEC Laboratories America, Cupertino, CA 95014, USA, 2009 [cit. 2018-05-20]. Dostupné z: <http://www.ifp.illinois.edu/~jyang29/papers/CVPR09-ScSPM.pdf>. Vědecká práce. Beckman Institute, University of Illinois at Urbana-Champaign.
- [24] Graphic processing unit. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: https://en.wikipedia.org/wiki/Graphics_processing_unit.
- [25] Java (programovací jazyk). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: [https://cs.wikipedia.org/wiki/Java_\(programovac%C3%AD_jazyk\)](https://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)).
- [26] *CUDA Toolkit Documentation* [online]. March 5, 2018 [cit. 2018-05-16]. Dostupné z: <https://docs.nvidia.com/cuda/>.

- [27] Java Platform, Standard Edition. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-16]. Dostupné z: https://en.wikipedia.org/wiki/Java_Platform,_Standard_Edition.
- [28] *Spring* [online]. 2018 [cit. 2018-05-16]. Dostupné z: <https://spring.io/>.
- [29] Web Services Description Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-16]. Dostupné z: https://en.wikipedia.org/wiki/Web_Services_Description_Language.
- [30] *POSTGRESQL: THE WORLD'S MOST ADVANCED OPEN SOURCE RELATIONAL DATABASE* [online]. The PostgreSQL Global Development Group, 2018 [cit. 2018-05-16]. Dostupné z: <https://www.postgresql.org/>.
- [31] Challenge-response. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-16]. Dostupné z: <https://cs.wikipedia.org/wiki/Challenge-response>.
- [32] Session. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-16]. Dostupné z: <https://cs.wikipedia.org/wiki/Session>.
- [33] Šůl (kryptografie). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-16]. Dostupné z: [https://cs.wikipedia.org/wiki/S%C5%AF1_\(kryptografie\)](https://cs.wikipedia.org/wiki/S%C5%AF1_(kryptografie)).
- [34] URL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-17]. Dostupné z: <https://en.wikipedia.org/wiki/URL>.
- [35] GET. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-17]. Dostupné z: <https://cs.wikipedia.org/wiki/GET>.
- [36] JavaBeans. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-17]. Dostupné z: <https://en.wikipedia.org/wiki/JavaBeans>.
- [37] *Bean scopes* [online]. [cit. 2018-05-17]. Dostupné z: <https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html>.
- [38] CMYK color model. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-18]. Dostupné z: https://en.wikipedia.org/wiki/CMYK_color_model.
- [39] JAR (file format). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-18]. Dostupné z: [https://en.wikipedia.org/wiki/JAR_\(file_format\)](https://en.wikipedia.org/wiki/JAR_(file_format)).

-
- [40] Graphical user interface. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-18]. Dostupné z: https://en.wikipedia.org/wiki/Graphical_user_interface.
- [41] Image Scaling. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-18]. Dostupné z: https://en.wikipedia.org/wiki/Image_scaling.
- [42] Nearest-neighbor interpolation. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-18]. Dostupné z: https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
CD	Compact Disc
CPU	Central Processing Unit
CRUM	Challenge-Response Authentication Mechanism
CUDA	Compute Unified Device Architecture
DFT	Discrete Fourier Transform (Diskrétní Fourierova transformace)
FFTW	Fastest Fourier Transform in the West
GPU	Graphic Processing Unit
GUI	Graphical User Interface
HW	Hardware
IFT	Inverse Fourier Transform (Zpětná Fourierova transformace)
JAR	Java ARchive
JavaSE	Java Platform, Standard Edition
KoP	Koeficient podobnosti dvou fotografií
KoZ	Koeficient zastupitelnosti vzájemně si podobných fotografií
MD5	Message-Digest algorithm
PC	Personal Computer
px	picture element (obrazový prvek)
SOAP	Simple Object Access Protocol
SVM	Support Vector Machine
SW	Software
URL	Uniform Resource Locator
WSDL	Web Services Description Language

SEZNAM OBRÁZKŮ

2.1	Statický scaling obrázků (komutativní)	15
2.2	Dynamický scaling obrázků (diskomutativní)	15
2.3	Příklad proložení dvou fotografií s použitím klasické korelace	17
2.4	Příklad proložení dvou fotografií s použitím křížové korelace	17
3.1	Příklad Diskrétní 2D konvoluce [19]	18
6.1	Vizualizace procesu výpočtu koeficientů na CPU	30
6.2	Vizualizace procesu výpočtu koeficientů na CPU s paralelizací na GPU	30
7.1	L1 pohled aktuálního stavu	33
7.2	L1 pohled cílového stavu	33
7.3	Funkční požadavky	34
7.4	Nefunkční požadavky	34
7.5	Aktéři	34
7.6	Případy užití	35
7.7	Model tříd	36
7.8	Model služeb	36
7.9	Sekvenční diagram	36
8.1	Náhled administrace uložených fotografií	37
8.2	Fotografie před úpravou	42
8.3	Vizualizace matice přestřehých hran	42
8.4	Vizualizace matice hran	43
8.5	Ukázka použití singleton [37]	43
8.6	Vizualizace komunikačních metod distribuované služby	44
8.7	Detail metody attach	44
8.8	Detail metody challenge	44
8.9	Detail metody challengeLogin	45
8.10	Detail metody getDataToCompare	45
8.11	Detail metody submitResult	45
9.1	Fotografie 1	49
9.2	Fotografie 2	49
9.3	Fotografie 3	49
9.4	Fotografie 4	49
9.5	Fotografie 5	49
9.6	Fotografie 6	49
9.7	Fotografie 7	49
9.8	Fotografie 8	49
9.9	Fotografie s nejvyšším KoZ (nejkvalitnější)	50
9.10	Fotografie s nejnižším KoZ (nejméně kvalitní)	50

SEZNAM TABULEK

3.1	Jádro pro detekci hran (horizontálně a vertikálně)	19
3.2	Jádro pro detekci hran (horizontálně, vertikálně a šikmé hrany) . . .	19
5.1	Tabulka výsledků jednotlivých variant krátkodobého testu	23
5.2	Tabulka výsledků jednotlivých variant krátkodobého testu	24
8.1	Jádro typu horní propust'	39
8.2	Jádro typu dolní propust'	39
9.1	Výsledky porovnání KoP 35 hotelových fotografií	48
9.2	Výsledky porovnání KoZ na 8 podobných fotografiích dle KoP . . .	49

SEZNAM ZDROJOVÉHO KÓDU

6.1	banchmark-cpu.m	26
6.2	banchmark-gpu.m	27
6.3	banchmark-koz-cpu.m	28
8.1	SharpnessCore.java	37
8.2	BrightnessMatrix.java	38
8.3	FTHighKernelMatrix.java	38
8.4	ConvolutionMatrix.java	39
8.5	FTLowKernelMatrix.java	39
8.6	sumSharpness.java	40
8.7	sumOverSharpness.java	40
8.8	queue.sql	40
8.9	servlet.sql	41
9.1	scale.java	46
9.2	NormalizedFTMatrix.java	47
9.3	CrossCorrelation.java	47
9.4	maxCorrelationCoefficient.java	47

SEZNAM PŘÍLOH

P I. CD s průvodními materiály

PŘÍLOHA P I. CD S PRŮVODNÍMI MATERIÁLY

Na CD naleznete

- kompletní zdrojové kódy knihovny pro výpočet KoP i KoZ
- veškerá testovací data (především fotografie)
- pomocné projekty (např. analýza v Enterprise Architect)