VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

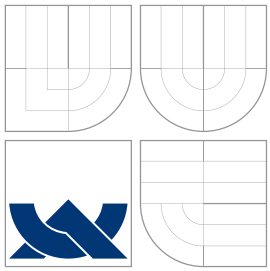
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÁ SLUŽBA - OBRAZOVÉ MOZAIKY

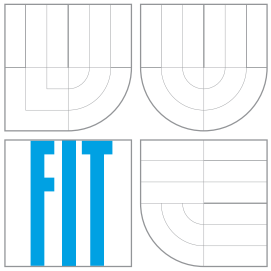
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE VOJTĚCH BEIL
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÁ SLUŽBA - OBRAZOVÉ MOZAIKY

WEB SERVICE - IMAGE MOSAICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH BEIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MIROSLAV ŠVUB

BRNO 2009

Abstrakt

Tato práce se zabývá vytvořením obrázkové mozaiky – tedy obrázku, který je složen z dalších tak, aby se z dálky jevil jako obrázek jiný. Dále se práce zabývá možností přenesení takovéto programu jako webovou službu.

Klíčová slova

mozaika, obrazová mozaika, JPEG, PNG, datová struktura, knihovna, dynamicky linkovaná knihovna, DLL, C#, .NET, ASP.NET, služba, webová služba, monitorování služby, databáze, relační databáze, SQL

Abstract

This work describes creating mosaic images – images that from distace look like another image. It also explores posibility transfering that program as web service.

Keywords

mosaic, images mosaics, JPEG, PNG, data structure, library, dynamic-link library, DLL, C#, .NET, ASP.NET, service, web service, service monitoring, database, relational database, SQL

Citace

Vojtěch Beil: Webová služba - obrazové mozaiky, bakalářská práce, Brno, FIT VUT v Brně, 2009

Webová služba - obrazové mozaiky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Miroslava Švuba Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Beil
21. ledna 2009

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvořit aplikaci, která ze zadaného vstupního obrázku vytvoří tzv. obrázkovou mozaiku, což je obraz složený z mnoha malých obrázků, přičemž z dostatečné vzdálenosti je její jako celek.

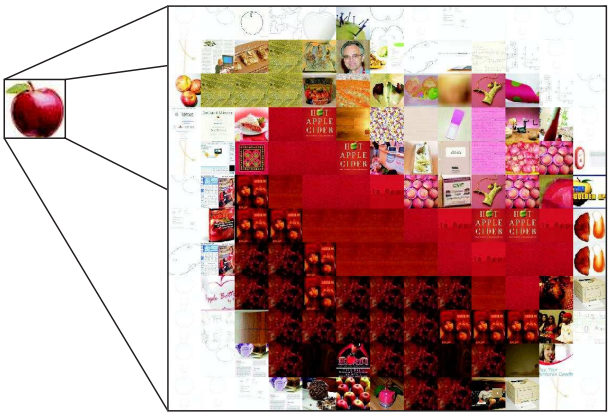
Kapitola Teorie pojednává a shrnuje použité postupy, technologie a algoritmy.

Další kapitola se zabývá návrhem aplikace. Obsahuje návrh aplikace a poskytuje základní pohled na jednotlivé části systému. Navrhuje ideální použití konkrétních algoritmů a technologií. Obsahuje též popis, návrh a analýzu webové služby. Zabývá se komunikací jednotlivých částí včetně vytváření knihoven, které mohou být využity zvlášť a které jsou pak dále používány dále v projektu. Zabývá se též přenosem dat mezi základními stavebními kameny aplikace.

Implementační kapitola se zabývá konkrétně použitými postupy, algoritmy, použitím konkrétních datových struktur. Také obsahuje výčet použitých technologií, programů a vývojových nástrojů a jejich verzí.

Pátá kapitola nadepsaná „Výsledky“ ukazuje některé konkrétní výsledky práce. Obsahuje hlavně ukázky obrázků a webového rozhraní.

Poslední závěrečná kapitola shrnuje výsledky bakalářské práce. Velká část kapitoly je věnována zamyšlením nad dalšími možnými rozšířeními. Samotný závěr shrnuje vlastní postup při vypracovávání, nastiňuje některá technologická rozhodnutí a zmiňuje nejzajímavější části práce.



Obrázek 1.1: Příklad výstupu bakalářské práce

Kapitola 2

Teorie

2.1 Co je to obraz

Obraz je grafické vyjádření, které se podobá nějakému předmětu, fyzickému objektu či osobě.

Dvourozměrné vyjádření může být fotografie, obrazovka, plátno v kině. Trojrozměrné vyjádření je např. socha, architektonické dílo.

Obraz (dvourozměrný) je většinou zachytáván pomocí optického zařízení (fotoaparát, dalekohled). Obraz může být natisknut, namalován či nakreslen.

2.2 Manipulace obrazu

Při manipilaci obrazu závisí na jeho vyjádření. Obraz (dvourozměrný) může být definován vektory (vektorová grafika) nebo jednotlivými pixely (rastrová grafika).

Ve vektorové grafice se uchovává pouze obraz uložených objektů, zatím co u vektorové grafiky lze vlastnosti uložených objektů měnit. Výhodou vektorové grafiky je zobrazení objektů stále hladké, zvětšení rastrové grafiky je diskrétní (zubaté), chybějící informaci o obrazu není odkud čerpat.

Obrazem můžeme manipulovat několika základními způsoby:

- Zmenšení, zvětšení
- Ořez obrazu
- Rotace obrazu, otočení podle osy
- Zkosení, jiná transformace
- Rozmazání

Dále pak s obrazem můžeme provádět další pokročilé operace, např. hledání hran, OCR (Optical character recognition - rozpoznávání textů), změna barev a další.

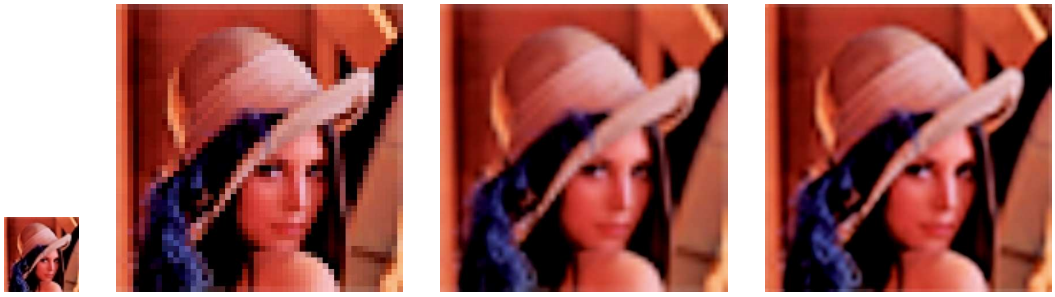
2.2.1 Zmenšení, zvětšení

Při změně velikosti obrazu máme k dispozici několik metod, např (zdroj [12]):

- interpolace nejbližším sousedem
- bilineární interpolace
- bikubická intepolace

Bilineární interpolace využívá lineární interpolaci ve dvou osách. Pro výpočet jednoho bodu ve výsledném obrázku je potřeba znát barvy čtyř okolních pixelů.

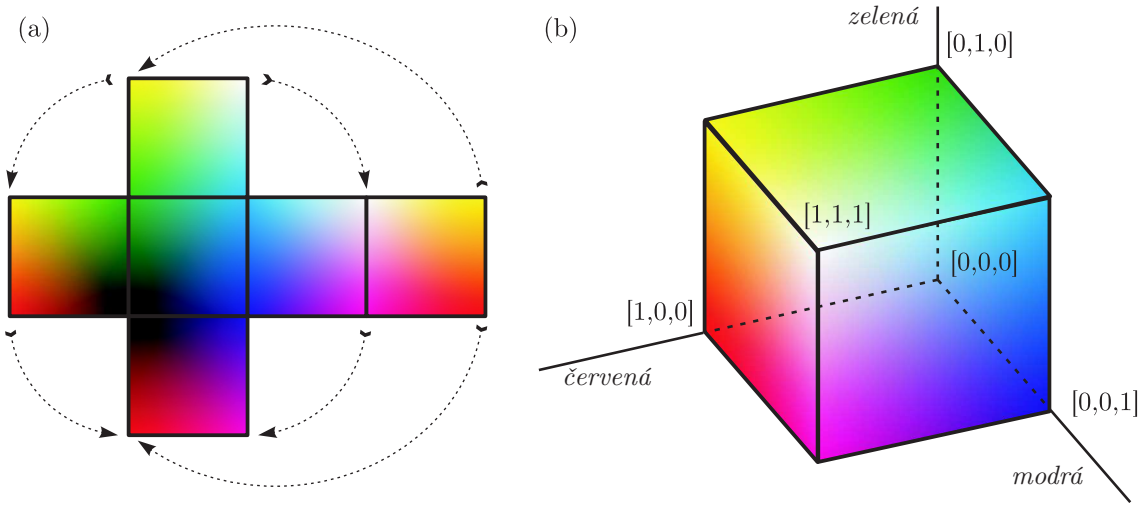
Bikubická interpolace využívá polynomiální interpolaci třetího stupně. Výpočet se provádí pomocí kubického polynomu. Pro dvojrozměrný rast je pro spočítání potřeba znát šestnáct pixelů.



Originál | Nejbližší soused | Bilineární interpolace | Bikubická interpolace

Obrázek 2.1: Porovnání interpolačních metod

2.3 Barevný RGB model



Obrázek 2.2: Plášť a model RGB krychle

barva	R	G	B	vlnová délka (nm)
červená	100 %	0 %	0 %	650 - 760
žlutá	100 %	100 %	0 %	550 - 590
zelená	0 %	100 %	0 %	490 - 760
azurová	0 %	100 %	100 %	455 - 490
modrá	0 %	0 %	100 %	430 - 455
fialová	100 %	0 %	100 %	360 - 430
bílá	100 %	100 %	100 %	
černá	0 %	0 %	0 %	

Tabulka 2.1: Barvy, intenzity složek a vlnová délka ([13])

Barevný RGB (*Red* – červená, *Green* – zelená, *Blue* – modrá) model je aditivní způsob míchání barev. Jde o míchání vyzařovaného světla. Nepotřebuje vnější světelný zdroj a zobrazuje tedy i ve tmě.

Aditivní způsob míchání barev je způsob, ve kterém se jednotlivé složky sčítají a vytváří světlo vyšší intenzity.

Na obrázku 2.2 je zobrazen plášť RGB krychle (a). Plášť, kde každá prostorová osa představuje intenzitu jedné barevné složky v bodě $[0, 0, 0]$ je barva černá, v bodě $[1, 1, 1]$ bílá. Promítnutí krychle do prostoru (b) zobrazuje model krychle, kde bod $[0, 0, 0]$ je neviditelný. Na tomto obrázku jsou naznačeny osy.

2.4 Podobnost dvou obrazů

2.4.1 Možnosti porovnání

Porovnání dvou obrázků, zda se jedná o totožné či podobné, může být obtížné s přihlédnutím k JPEG kompresi, změně velikosti, možnosti přidání nebo odstranění okrajů či přidání textu či loga k okrajům obrázků (zdroj [11]).

Techniky možné k použití k nalezení podobných obrázků:

- Hledání podle barvy: při hledání je vhodné nalézt a odstranit zbytečné okraje, snížit počet barev v obrázku, použití histogramu k identifikování několika barev či barevných významných regionů, podle kterých se bude porovnávat podobnost dvou obrazů.
- Podle barevnosti částí: porovná se průměrná barva několika (nebo všech) částí obrázku.
- Nalezení hran: Obrázky se zmenší na stejnou velikost, provede se detekce hran a naleznou se křivek. Poté se porovnají křivky podle blízkosti a podobnosti (např. podle polohy, směru, zakřivení, úhlu).
- Provést rychlou Fourierovu transformaci (na části obrázků) a porovnat podobnost (několika prvních) koeficientů.

2.4.2 Průměrná barva

Pro určení průměrné barvy obrazu je použit vzorec:

Pro obraz o rozměrech $x \times y$ s body (R_{ij}, G_{ij}, B_{ij}), kde R_{ij} je intenzita červené složky (G_{ij}, B_{ij} zelené a modré) je použit vzorec:

$$(\delta_R, \delta_G, \delta_B) = \frac{1}{x \cdot y} \left(\sum_{i=1}^{x} \sum_{j=1}^{y} R_{ij}, \sum_{i=1}^{x} \sum_{j=1}^{y} G_{ij}, \sum_{i=1}^{x} \sum_{j=1}^{y} B_{ij} \right) \tag{2.1}$$

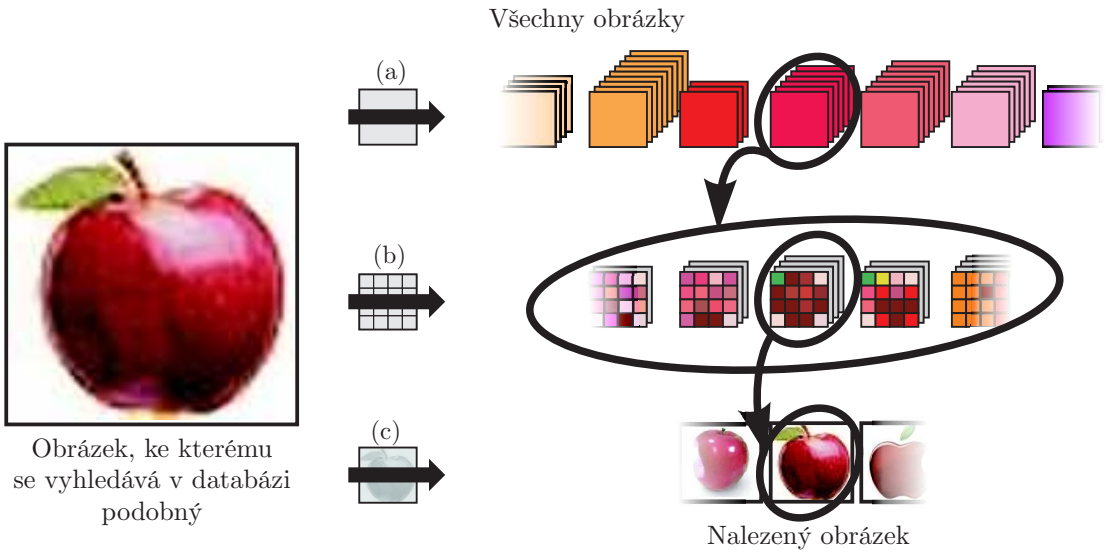
2.4.3 Nalezení vhodných kandidátů

Podobnost dvou obrázků s průměrnými barvami $\alpha = (\alpha_R, \alpha_G, \alpha_B)$ a $\beta = (\beta_R, \beta_G, \beta_B)$ je dána rozdílem jejich průměrných barev:

$$p = (\alpha_R - \beta_R)^2 + (\alpha_G - \beta_G)^2 + (\alpha_B - \beta_B)^2 \tag{2.2}$$

Úkolem algoritmu je nalézt pro vstupní čtverec α takový β, aby bylo toto číslo minimální.

2.5 Algoritmus



Obrázek 2.3: Základní princip funkce algoritmu

Pro vytvoření mozaiky je důležité, aby z větší vzdálenosti se podobala co nejblíže původní předloze. Výsledný obrázek by měl co nejblíže zachovat původní rozložení, původní barevnost a výběr mozaikových vložených obrázků by se měl co nejvíce podobat předloze.

Základem algoritmu je předpoklad, že vstupní obrázek se rozdělí do pravidelné (čtvercové) mřížky o určitých rozměrech. K těmto vstupním čtvercům se pak přistupuje jednotlivě. Pro daný čtverec se v databázi snaží nalézt nejpodobnější obrázek, který by se namísto vstupního čtverce vložil.

Nalezený obrázek nemusí být vhodných rozměrů a nemusí mít vhodný poměr stran. Vhodný rozměr (velikost) lze odstranit zmenšením nebo zvětšením, nesprávný poměr stran

oříznutím (či změnou rozměrů bez zachování poměru stran). Další možností ještě může být pokračování v hledání a nalezení obrázku, které také splňuje velikostní kritéria.

Nejpřesnější porovnání dvou obrazů (vstupního čtverce a hledaného obrázku) je pixel po pixelu, tato operace je ovšem velice náročná nehledě na požadavky na vstupní zařízení a paměť.

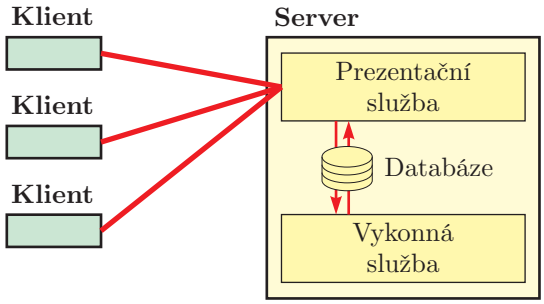
Protože pro základní nahrazení je dostačující nalézt průměrnou barvu (vzorec 2.1) vstupního čtverce a nalézt nahrazující obrázek s průměrnou barvou tak, aby rozdíl (vzorec 2.2) byl minimální.

Pro zachování hran a detailů (protože rozložení barev, hran a dalších informací na obrázcích může být rozdílné a tudíž nalezený obrázek nemusí zachovávat věrně předlohu) může být vstupní čtverec rozdělen do dalších čtverců a pro jednotlivé čtverce v předloze a v hledaném obrázku znovu provedeno hledání.

Obrázek 2.3 zobrazuje vlastní fungování algoritmu. (a) Nejdříve je nalezen seznam obrázků s podobnou průměrnou barvou. Tyto obrázky jsou pak rozděleny pravidelnou mřížkou do menších čtverců (b) a podle předlohy dochází k porovnání. Tímto krokem je opět nalezen seznam kandidátních obrázků. Všechny tyto informace lze pro urychlení přednačítat předem z databáze. V posledním bodě (c) je nalezen omezený počet kandidátů, které se již podobají předloze velmi blízce, pro maximální přesnost nyní dochází k vlastnímu načtení kandidátních obrázků a porovnání pixel po pixelu.

Nalezený obrázek je pak vložen do výsledného obrázku na odpovídající místo.

2.6 Webová služba



Obrázek 2.4: Rozdělení aplikace na aplikaci typu klient/server

Služba World Wide Web nabízí možnost prezentovat informace, databáze, výsledky výpočtů ovlivněných uživatelem nebo čímkoliv jiným.

Existuje mnoho druhů webových služeb od těch nejjednodušších až po komplexní. Příkladem webové služby může být zobrazení náhodného čísla či zobrazení statistik získaných od mnoha miliónů zkoumaných objektů.

Pro používání webových služeb je potřeba internetový prohlížeč, který je již dnes ale standardem pro naprostou většinu uživatelkých počítačů. Internetové prohlížeče podporující různých stupeň webových technologií disponuje nyní stále více mobilních zařízení (telefonů, PDA, ...)

Webová služba je služba typu klient-server (blíže popsáno např. [3]), kdy uživatel posílá na server požadavky, server požadavek zpracuje a pošle klientovi odpověď.

Komunikačním protokolem může být například HTTP (viz. [5]).

2.6.1 Rozdělení webové služby

Některé webové služby mohou poskytovat výsledky uživatelských dotazů v reálném čase jiné požadavky mohou být natolik časově náročné (nebo výsledek může být závislý na vnejších neovlivnitelných okolnostech) tak, že pouze zprostředkovávají zobrazení informací získaných z např. externího zdroje.

Ve své bakalářské práci jsem zvolil použití webové služby, která je rozdělena na dvě části – prezentační a výkonnou:

1. Prezentační – prezentuje výsledky činnosti výkonné části, zobrazuje seznam požadavků a jejich stavů, umožňuje přidávat nové úlohy.

2. Výkonná – zpracovává frontu požadavků, kvůli zatížení serveru, může zpracovávat služba pouze jeden požadavek.

2.6.1.1 Uživatelské rozhranní

K přístupu k webovým službám je používán internetový prohlížeč, který zajišťuje základní uživatelské rozhraní.

Prezentačním prostředkem je jazyk HTML. HTML (HyperText Markup Langage) je značkovací jazyk používaný pro web. Popisuje strukturu textových informací v dokumentu – označením úseků textu jako napis, odkaz nebo jako odstavec.

2.6.1.2 Služba

Služba je v prostředí Windows ekvivalentní významu démona z prostředí Unix. V multitaskovém operačním systému je program (proces), který běží na pozadí a provádí činnost.

Služba je většinou nastartována při startu počítače. Služba může vykonávat výpočet na pozadí, může zpřístupňovat zařízení, může čekat na vnější podmět a na ten reagovat.

Jako služba může být implementován např. internetový server, který naslouchá na socketu a v případě požadavku jej zpracuje a pošle odpověď.

2.6.1.3 Komunikace

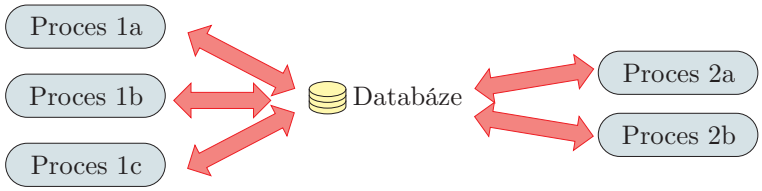
Komunikace mezi uživatelským rozhranním a službou je otázka meziprocesové komunikace. Uživatelské rozhranní i služba běžící na pozadí jsou aplikace (procesy), které si potřebují vyměňovat data.

Meziprocesová komunikace je sada technik pro výměnu dat mezi dvěma nebo více procesech.

Techniky můžeme rozdělit na metody:

- zasílání zpráv
- synchronizace
- sdílená paměť
- volání vzdálených procedur

Komunikace mezi procesy může být též provedena pomocí sdílené databáze (paměti), ke které mají oba procesy přístup. Komunikace pak probíhá tím, že první proces provede úpravu (zápis, aktualizaci, smazání) databáze (úprava jednoho či více řádků nebo celé



Obrázek 2.5: Meziprocesová komunikace pomocí (relační) databáze

tabulky), druhý proces pak v daných intervalech (na vyžádání uživatele, po určité časové době) sleduje, zda neproběhla změna v tabulce.

Výhodami tohoto řešení je pak jednoduchost implementace (mnoho informačních systémů používá jako zdroj dat databázi), komunikace mezi procesy je možná i v případě, kdy jeden proces není spuštěn.

Nevýhodou je potřeba častého dotazování na databázi, zda nedošlo k nějaké změně.

2.7 Technologie

2.7.1 C#

C# je vysoce úrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft společně s platformou .NET Framework.

C# nabízí automatický garbage collector. Garbage collector je část běhového prostředí, která má za úkol automaticky určit, která část paměti není už používaná, a uvolnit ji pro další znovupoužití.

Jazyk má objektově orientovanou syntaxi založenou na jazyce C++, ale je též velmi ovlivněn jazykem Java.

První verze jazyka C# 1.0 byla vydaná v roce 2002 společně s .NET frameworkem 1.0.

Další verze 2.0 byla vydána v roce 2005. Společně s .NET frameworkem 2.0 přináší nové vlastnosti jako částečné a statické třídy, iterátory, anonymní metody, nullovatelné typy a operátor koalescence (operátor `??` – dva otazníky).

Aktuálně poslední verze 3.0 vyšla v roce 2007 společně s Frameworkem 3.5. Tato nová verze přináší LINQ, pomocí čehož se lze ptát na objekty reprezentující databáze. Dále tato nová verze přináší jednodušší lambda výrazy, anonymní třídy a další nové vlastnosti. Tyto změny nevyžadují změnu podkladového CIL, takže s požadovanými knihovnami mohou běžet i na předchozím frameworku 2.0.

2.7.1.1 Třídy

V objektově orientovaném programování je třída základem vytváření objektů. Třída definuje vlastnosti a metody. Vlastnosti mohou odlišovat jednotlivé objekty.

Třída je jakýsi balíček metadat obsahující pravidla, podle kterých jsou tvořeny objekty a jak se objekty chovají. Objekty jsou nazývány instancemi tříd.

Jazyk C# nabízí jednoduchou dědičnost.

2.7.1.2 Atributy

```
[FormatterTable("Keyword")]
public class Keyword : Formatter
{
     [FormatterAttribute("Name",
          FormatterAttribute.FormatterSqlType.SQL_STRING)]
     public string Name;

     [FormatterAttribute("Description",
          FormatterAttribute.FormatterSqlType.SQL_STRING)]
     public string Description;


     public Keyword()
     {
     }

}
```

V jazyce C# je možné určitým entitám přiřadit „značky“ (atributy) ke specifikování dodatečných informací. Tyto informace mohou být pak za běhu získány pomocí tzv. *reflexion*.

Atributy jsou systémové i uživatelské. Programátor může definovat vlastní a ty používat.

Ukázka zápisu atributů je k dispozici v úseku zdrojového kódu uvedeného výše.

Atributy mohou být použity k různým účelům. Jednou z možností je propojení kódu C# s dynamicky linkovanou knihovnou DLL (pomocí atributu `DllImport`). Pomocí atributů `MarshalAs` lze též specifikovat, jakým způsobem bude s parametry funkcí nakládáno při volání funkcí z externích DLL knihoven.

2.7.2 .NET

Microsoft .NET Framework je soubor technologií v softwarových produktech, který je dostupný pro web, Windows i Pocket PC.

Framework pokrývá mnoho potřeb pro programování např. uživatelské rozhranní, přístup k datům, připojování k databázím, kryptografie, vývoj pro web, numerické algoritmy a síťovou komunikaci.

Programy napsané pomocí .NET frameworku jsou spouštěny v *Common Language Runtime* (CLR) (zdroj citemsdn:clr), na kterém běží zvláštní druh bytekódu.

Zdrojový kód může být psán v C#, Visual Basicu .NET či dalších a během kompilace je zkompilován do CIL kódu (*Common Intermediate Language*). Při běhu programu je CIL kódu převeden do nativního kódu operačního systému. Pro urychlení může být pro urychlení zkompilován do nativního kódu před samotným během ve zvláštním kroku.

Tato technika umožňuje programátorům ignorovat specifika procesoru, na kterém program běží.

Common Language Runtime obsahuje též:

- správu paměti,
- správu vláken,
- správu výjimek
- garbage collection
- zabezpečení

2.7.3 ASP.NET

ASP.NET je součást .NET frameworku pro vývoj webových aplikací a služeb.

ASP.NET je nástupcem technologie *Active Server Pages* (ASP).

Programátor při vytváření ASP.NET stránek může využívat kterýkoliv jazyk podporující CLR (viz. výše). Výhodou tohoto řešení je rychlost, protože aplikace založené na ASP.NET jsou překompilovány do DLL souborů (naproti skriptovacím jazykům, které při každém přístupu znovu parsovány).

ASP.NET umožňuje oddělení kódu od vzhledu – pro kód i algoritmickou část webu (např. práci s databází, odchycení událostí) existují oddělené zdrojové soubory.

Stránky jsou skládány z ovládacích prvků podobných ovládacím prvkům ve Windows.

Těmto prvkům lze nastavovat určité vlastnosti či zachytávat události. Tyto webové ovládací prvky samy generují HTML, které se pak vkládá do výsledného HTML kódu.

Událostmi řízené programování vyžaduje zachování stavu, jenže webový protokol je sám o sobě bezstavový, proto se tento problém řeší pomocí HTML (a JavaScriptu) pomocí dvou základních technik (zdroj [9]):

- ViewState – informace se uchovávají mezi požadavky ve skrytém formulářovém poli. Nevýhodou tohoto řešení je, že se mezi serverem a klientem může přenášet velké množství dat.
- SessionState – mezi serverem a klientem se posílá jedinečný identifikátor (pomocí skrytého formulářového pole, součástí URL nebo jako cookie). Toto řešení zvyšuje nároky na výkon serveru.

2.7.4 Relační databáze, SQL

Relační databáze je databáze založená relačním modelu (zdroj [14], [15]). Databáze je zjednodušeně určité úložiště, kde se ukládají data. Databáze poskytuje rychlejší přístup k datům než soubory. Databáze též umožňuje paralelní přístup k datům od více uživatelů najednou.

Základem databáze je tabulka obsahující data. Tabulka se skládá ze sloupců a řádků.

Každý sloupec v tabulce má jedinečný název a určitý datový typ. Data se ukládají po řádcích (záznamech).

Pro urychlení přístupu k datům můžeme nad sloupcem (nebo více sloupci) definovat klíč, který urychlí přístup k záznamům podle zvolených sloupců.

Záznamy mezi tabulkami jsou reprezentovány nevlastními klíči, které napomáhají k udržení integrity databáze.

S relačními databázemi se manipuluje pomocí jazyku SQL (Structured Query Language).

Kapitola 3

Návrh

Na obrázku 3.1 je naznačena základní struktura projektu, dědičnosti, zdrojů i toku dat.

Návrh se řídí myšlenkou, že nejnižší a výpočetně nejnáročnější části programu jsou co možná nejvíce optimalizovány, ideálně psané v programovacím jazyce s minimálními požadavky na režie (např. kontroly meze polí, kontroly přístupu do paměti). Další vrstvy jsou pak psány pomocí vyššího programovacího jazyka, který poskytuje mnoho doplňkových služeb a pomocí kterého je jednoduché a hlavně rychlé naprogramovat uživatelské rozhraní.

Vstupem i *výstupem* práce programu je vždy obrázek.

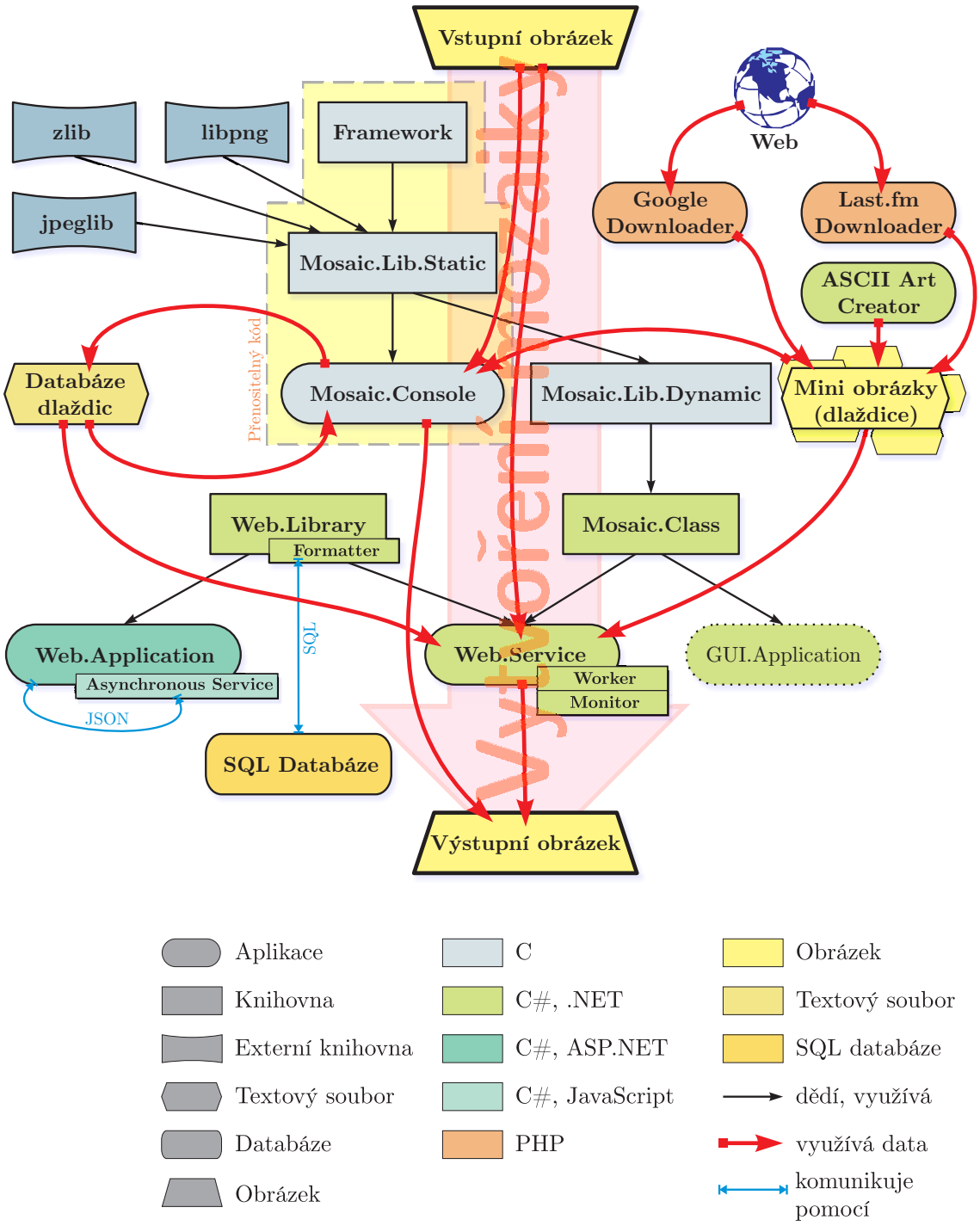
Výsledkem této bakalářské práce jsou knihovny:

- Framework – knihovna datových struktur, optimalizovaná (minimální implementace) pro bakalářskou práci
- Mosaic.Lib.Static – knihovna pro vlastní vytvoření mozaiky a správu databází
- Mosaic.Lib.Dynamic – *.dll* knihovna, která slouží pro propojení s C# kódem, správa databází je tu odstraněna, přídány jsou možnosti pro spolupráci s C# pro přímé vytváření obrázku (takto lze doimplementovat podpora dalších formátů, které zvládne .NET framework)
- Mosaic.Class – C# třída pro práci s mozaikou, toto třída je univerzální a měla by být využitelná pro jakýkoliv projekt (nejen takový, který je vázán na webovou službu či GUI)
- Formatter – třída pro propojení objektů a databáze (s minimálními měnami lze tuto třídu používat např. i pro ODBC a tedy i [skoro] libovolnou databázi)
- Web.Library – knihovna pro spolupráci mezi webovou stránkou (Web.Application) a programem na pozadí (Web.Service)

A dále pak tyto programy:

- Mosaic.Console – program pro vytváření mozaikových obrázků a správu databází, tento poslední program je psán přenositelně, následující části jsou již kvůli použitým technologiím vázány je určitou platformu
- Web.Application – informační systém, který prezentuje práci služby běžící na pozadí, pomocí této webového informačního systému lze požadavky zadávat a zobrazovat

- Web.Service – program, který je navržen tak, aby běžel na pozadí a zpracovával požadavky zadané přes webovou službu
- Google Donwloader, Last.fm Downloader, ASCII Art Creator – programy pro získávání databáze obrázků



Obrázek 3.1: Návrh struktury aplikace, knihoven, zdrojů a toku dat

Pro vytvoření obrázkové mozaiky jsou vždy potřeba tři věci:

- vstupní obrázek – obrázek, který je použit jako předloha vytvořené mozaiky
- databáze obrázků – předzpracovaná databáze s informacemi o obrázcích
- obrázky – z obrázků se skládá mozaika, obrázky nemusí být předepsané velikosti, při zpracování se automaticky jejich velikost opraví podle potřeby

3.1 Framework

Protože některé funkce a datové struktury se používají často a na mnoha místech, rozhodnul jsem se nejdříve vytvořit knihovnu těchto funkcí a zcela ji osamostatnit od vlastní implementace.

Výhodou tohoto přístupu je znovupoužitelnost v dalších programech i projektech.

Alternativou použití této knihovny by bylo např. použití STL knihovny C++.

Většina datových struktur je optimalizována pro několik základních operací, většinou není potřeba univerzálnost (například u seznam je implementován tak, aby bylo rychlé přidávání na konec).

Tyta knihovna je napsaná pomocí ISO C99.

Knihovna navíc od dále popsaných datových struktur podporuje též práci s libovolně dlouhými řetězci, alokaci velkého množství malé paměti a neomezeně velké pole.

3.1.1 Seznam

Třída jednosměrného lineárního seznamu.

Speciálně upravený linerání seznam. Tento zvláštní případ je optimalizován pro velmi časté přidávání prvku na konec seznamu.

Výsledkem je pak přidání na konec seznamu se složitostí O(1).

3.1.2 Hash tabulka

Hashovací tabulka je datová struktura, která sdružuje klíče (hashe) a hodnoty. Hodnota klíče je vypočítána podle pravidla (hashovací funkce) tak, aby:

- byl klíč co nejjednoznačněji určen,
- pravděpodobnost stejného klíče více (různým) položkám byla co nejnižší,
- rozptyl hodnot klíčů pro dvě blízké položky byl co nejvyšší

Hashovací tabulka je určena pro rychlé vyhledávání v poli. Pomocí hashovací funkce přiřazujeme hodnotě klíče ukazatel do datové struktury (pole).

Hashovací tabulka, kde klíčem je řetězec. Velikost je volena jako prvočíslo (kvůli rovnoměrnému využití datové struktury).

Složitost hledání v této datové struktuře je O(n).

3.1.3 Seřazené pole

Seřazené pole je datová struktura při manipulaci s obsahem struktury (přidávání, odstraňování položek) zachovává seřazenost pole.

Vyhledávání v této struktuře je častěji rychlejší než vyhledávání v neseřazeném poli.

3.2 Knihovny

Knihovna je soubor (kolekce) funkcí (metod, tříd) používaných pří vývoje software. Hlavním významem možnost modulárního přístupu k programování, kdy je jedna knihovna znovu využitelná ve více projektech.

Knihovny poskytují aplikační programátorské rozhraní (API).

Knihovny můžeme rozlišovat podle způsobu propojení s programem:

- statická knihovna
- dynamická knihovna

Statické knihovny jsou s programem spojovány během sestavování (linkování) programu. Do cílového programu se uloží všechen kód a data odkazovaný z programu a všechna data odkazovaná v použitém kódu knihovny.

3.2.1 Dynamicky linkované knihovny

Dynamicky linkované knihovny (v prostředí Microsoft Windows soubory s příponou *.dll*) jsou programové moduly obsahující kód, data a zdroje (zdroj [7]), jenž mohou být využity dalším modulem (aplikací nebo další dynamickou knihovnou).

Při sestavování programu jsou do programu pouze umístěny odkazy na symboly definované v dynamické knihovně, pro vlastní chod programu je potřeba pak i dynamická knihovna nainstalovaná ve stejném adresáři jako program nebo v systému (a samozřejmě další knihovny, které knihovna může využívat). Dynamická knihovna lze načíst do programu i až během běhu programu. Dynamické knihovny jsou závislé na operačním systému. Typická přípona dynamických knihoven v prostředí Windows je *.dll*, Unixových systémech pak *.so*.

Výhodou použití DLL knihovny je možnost měnit funkčnost programů bez nutnosti měnit hlavní program. Operační systém může optimalizovat využití paměti tím, že pokud využívá dynamickou knihovnu více procesů najednou, může být knihovna v paměti sdílená.

3.2.2 Externí knihovny

3.2.2.1 `libjpeg`

JPEG je často používaná metoda komprese obrázků a fotografií. Metoda je používaná ve více obrázkových formátech (například JPEG/Exif nebo JPEG/JFIF), tyto formáty nejsou většinou rozlišovány a jsou označovány jednoduše jako JPEG.

JPEG je standardizován jako norma ISO 10918-1. Standard obsahuje kompresi obrázku do datového proudu i formát uložení tohoto proudu do souboru.

Implementaci pro práci a manipulaci s obrázky ve formátu JPEG vytvořila skupina Independent JPEG Group [2].

3.2.2.2 `libpng`, `zlib`

PNG (Portable Network Graphics) je rastrový formát souboru pro uložení obrázku. Tento formát byl vytvořen jako náhrada GIF[1] (zdroj [16]).

Oproti formátu GIF přináší PNG tyto výhody:

- alfa kanál – průhlednost
- gama korekce – multiplatformní úprava světlosti obrázku
- podpora prokládání – obrázek se může zobrazovat postupně (při načítání je obrázek stále detailnější)

Knihovna `libpng` využívá bezztrátovou kompresi (pomoci knihovny `zlib`).

3.3 Vyhledávací služby

Vyhledávací služby na Internetu pravidelně prochází WWW stránky a indexují obsah. Vyhledávací služby jsou zaměřeny hlavně na indexování textu, ale většinou indexují i další obsah. Například jsou to i obrázky.

Mezi takovéto vyhledávací služby patří například:

- Google Images – `http://images.google.com/`
- Yahoo! Image Search – `http://images.search.yahoo.com/`
- Live Search – `http://www.live.com/?scope=images`

Indexování probíhá většinou podle alternativního popisu obrázku (parametr `alt`), URL obrázku, textu odkazu na obrázek, textu na stránce, kde je obrázek zobrazen. Mohou být použity i pokročilejší techniky rozpoznávání obsahu obrázků a fotografií.

Více o získávání obrázků z webových služeb lze nalézt v kapitole 4.8.

3.4 Knihovna pro vytvoření mozaiky

Kvůli možnosti jednoduchých úprav, použitelnosti, rozšiřitelnosti a možnosti znovuvyužití kódu je algoritmus a celá knihovna pro vytváření mozaiky oddělena od zbytku kódu.

Tento postup umožňuje využít knihovnu pro program pro příkazovou řádku nebo pro vytvoření DLL knihovny, která je pak zase používána dále.

Kvůli důslednému oddělování jednotlivých logických modulů během programování se tento postup velmi osvědčil.

Pokud by například někdo nyní chtěl vzít knihovnu a vytvořit GUI, stačí vzít knihovnu a využívat pouze jejich služeb. Knihovna byla napsána tak, aby žádné další zásahy do vlastního kódu nebylo potřeba provádět.

Více o rozšířeních lze nalézt v kapitole 6.1 Možná rozšíření.

[1] GIF - Graphics Interchange Format

3.5 Optimalizace rychlosti

Prapůvodní implementace programu obsahovala velmi málo optimalizací. V této implementaci byl na optimalizaci brán velký zřetel, výběr datových struktur i výběr algoritmů byl podřízen paměťové a rychlostní optimalizaci.

Protože v prvním stupni funkce algoritmu není třeba znát vlastní obsah obrázku, stačí si pamatovat pouze několik základních hodnot (průměrná barva, průměrná barva submatic), stačí tyto informace načíst z předzpracované databáze. Tímto se velmi redukuje náročnost na paměť, ale i k požadovaným informacím.

3.5.1 Trojrozměrný hash

Asi největší optimalizací, která se velmi pozitivně projevila právě na rychlosti algoritmu je použití uložení pole seznamů obrázků, které mají zhruba podobnou barevnost.

Představit tento postup si lze jako rozkrájení trojrozměrné RGB krychle (obrázek 2.2), kde každá krychle obsahuje pouze barevně podobné obrázky.

Pokud by náhodou nebyl nalezený kandidát, stačí hledat pouze v blízkém okolí.

Tento postup je také ukázal jako lepší protože algoritmus je víceúrovňový (obrázek 2.3) a pro přestup z jedné úrovně do druhé je potřeba nalézt více kandidátů.

Hledání pouze podle průměrné barvy má tu nevýhodu, že pokud se hledá nejlepší shoda, může se nalézt kandidát, který je sice podle průměrné barvy bližší, ale například podle hran by se jednalo o naprosto nevyhovující obrázek.

3.5.2 Použití seřazeného pole s omezenou délkou

V posledním kroku hledání vhodného obrázku do mozaiky se používá porovnávání pixel po pixelu. (Tato operace je také velmi náročná, proto se porovnává pouze několik málo obrázků). Pro selekci z předchozího kroku se používá seřazené pole s omezenou délkou, která automaticky řadí nejlepší kandidáty na začátek fronty a horší automaticky vyřazuje.

Odpadají tak problémy s případnou realokací (pokud by bylo použito pole), či přílišné paměťové nároky (pokud by byl použit seznam).

3.6 Optimalizace využití paměti

Práce s pamětí je při běhu tohoto programu zásadní, protože je navržen tak, aby pracoval jako služba na pozadí běhu počítače.

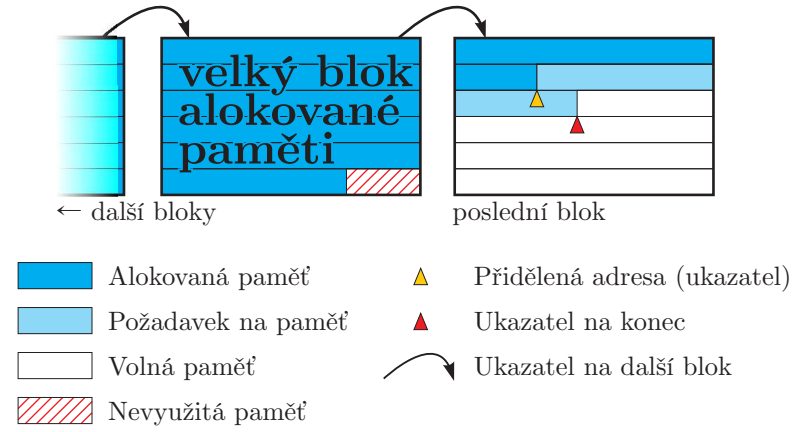
Zvláštní péče tedy byla věnována paměti – tedy její alokaci a dealokaci a pokud možno minimalizaci paměťových nároků, protože program je navržen tak, aby pracoval s desítkami tisíc obrázků v databázi.

3.6.1 Drobení alokované paměti

Protože během výpočtu je potřeba přistupovat a číst z mnoha souborů, o kterých není před spuštěním nic známo, a načítání všech je zase nemožné z nedostatku paměti, byla navrhnuta speciální struktura a postup pro alokaci paměti.

Využívá se toho, že alokovat paměť je potřeba velice často, zatímco uvolnění paměti se provádí až po skončení algoritmu.

Myšlenka je taková, že se nejdříve alokuje velký paměťový prostor, ze kterého se postupně podle požadavků přidělují paměťové bloky.



Obrázek 3.2: Názorný nakres fungování třídy allocator

Schematické znároznení třídy `allocator` je na obrázku 3.2.
Výhody:

- Operační systémy většinou přidělují paměť v blocích. Tento přístup dovoluje alokovat i po bytech.
- Paměť je lépe využita při alokaci velkého množství malých úseků paměti, nedochází k vytváření částí paměti, které zůstávají nevyužity.
- Dealokace paměti probíhá v jednom kroku není potřeba dealokovat všechny přidělené bloky jednotlivě. Snižuje se možnost memory leaků[2].

Nevýhody:

- Na konci alokovaného velkého bloku může vzniknout nepotřebná paměť. Alokace paměti je ale navržena tak, aby docházelo k potřebě relativně malých částí paměti. Pokud je alokovaný blok relativně velký, tato ztráta by měla být v porovnání s celým blokem zanedbatelná.
- Při potřebě alokovat blok větší než je celý alokovaný blok.
- Nelze dealokovat jednotlivý přidělený blok nebo ho zvětšit.

3.6.2 Využití hash tabulky

Každý záznam, který představuje obrázek má kromě barevných charakteristik přiděleno taktéž klíčové slovo (může se používat, například když chceme vytvořit obrázek auto z mnoha dalších aut).

Protože se klíčová slova častokrát opakují, je každé klíčové slovo v paměti uloženo pouze jednou. V pracovní paměti je pak pouze umístěn odkaz na toto klíčové slovo, ne samotná alokovaná paměť.

Při načítání databáze se v hashtabulce nalezne klíčové slovo a do samotného záznamu v paměti se uloží pouze odkaz.

[2]memory leak – ztráta paměti, která byla alokována a na kterou byl ztracen odkaz

3.7 C# třída

Propojení jazyků C# a C je provedeno přes dynamicky linkovanou knihovnu DLL (viz. 3.2.1).

Tato knihovna sama o sobě nepřináší žádnou další funkčnost stará se pouze o volání funkcí z nižšího programovacího jazyka.

Při vytvoření této třídy vzniká znovupoužitelná knihovna (třída), kterou lze použít v dalších programech.

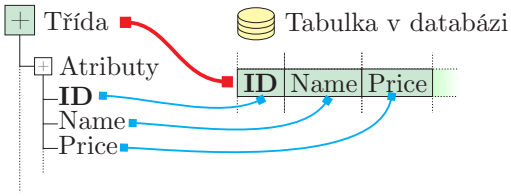
Například by bylo možné vytvořit jednoduché grafické uživatelské rozhranní, program pro příkazový řádek či službu běžící na pozadí. Z těchto možností je v této bakalářské práci implementována služba, která spolupracuje s jednoduchým webovým uživatelským rozhraním.

Další výhodou je možnost přidat další funkčnost na úrovni .NET. Příkladem může být podpora dalších grafických formátů (např. GIF) ne lepší správa vstupních databází obrázků.

3.8 Formatter

Pro práci s databází existuje několik možností. Je možné spojit se s databází, vložit SQL dotaz a zpracovat vrácenou odpověď.

Tato třída se snaží o trochu jiný přístup k databázi – snaží se přímo atributy třídy přiřadit jednotlivým sloupcům tabulky. Jedna instance objektu pak zastupuje jeden řádek tabulky.



Obrázek 3.3: Třída `formatter` – zobrazení tabulky relační databáze na objekt v C#

3.8.1 Návrh

Návrh třídy Formatter je svázán s možností jazyka C# používat a přiřazovat jednotlivým členům tříd atributy (viz. 2.7.1.2). Třída se snaží před uživatelem skrýt nutnost používat dotazovací jazyk SQL databáze.

Třída musí mít specifikováno, ke které tabulce se vztahuje.

Atributy členů třídy specifikují propojení se sloupcem v databázi a typ, který se použije pro formátování z či do SQL dotazů.

Další výhodou je dynamické generování SQL dotazů až při potřebě, aplikační programátor jednoduše například specifikuje, že potřebuje načíst záznam s konkrétním identifikačním číslem a formatter vygeneruje SQL příkaz, provede jej a výsledek dotazu zpracuje a data přiřadí přímo datům instanci objektu.

Formatter poskytuje základní služby nad objekty (záznamy):

- *Načtení* – je nalezen řádek v databázi a jeho hodnoty jsou přiřazeny instanci třídy.
- *Načtení seznamu* – podle SQL dotazu je vytvořen nový seznam a obsahuje všechny objekty, jenž zastupují záznamy v tabulce databáze
- *Vytvoření* – objekt je vytvořen, jeho hodnoty jsou inicializovány pomocí konstruktoru. Při pokusu o uložení je vygenerován SQL dotaz `INSERT`.
- *Aktualizovat* – objekt, který již byl z databáze načten, je v databázi taktéž aktualizován. Při uložení (aktualizaci) je vygenerován `UPDATE` dotaz.

Formatter poskytuje tuto základní funkčnost:

- Skrývá většinu potřeby používat SQL dotazy.
- Zjednodušuje případnou nutnost přidat nové prvky nebo některé odebírat. SQL dotazy jsou generovány podle atributů v třídě. Odebrání přidání automaticky změní dotazy.
- Zrušení nutnosti rozlišovat mezi tím, zda se jedná o vytvoření nebo aktualizaci záznamu. Formatter zvolí operaci podle typu vytvoření.

3.8.2 Konstrukce objektu

Každá, která bude takto používána musí být předkem třídy Formatter. Tímto zdědí metody pro načtení a uložení (vložení a aktualizaci) a také několik doprovodných metod (např. zjištění tabulky svázané s třídou).

Děděním od této třídy vzniká nutnost mít v každé tabulce mít sloupec ID.

Sloupec ID je typu GUID (viz. [4] nebo [6]), což je unikátní identifikátor, který by měl být celosvětově jedinečný.

3.8.3 Načtení, uložení

Existují dva druhy *čtení* dat z databáze. První možnost je naplnit daty již existující (nebo nově vytvořený) jeden objekt, další možností je načtení více položek (SQL dotaz vrací více záznamů). Protože obecně může být takových položek nekonečně, je vhodné použít pro uložení více záznamů lineární seznam.

Uložení položky může probíhat jako vložení nové položky nebo aktualizace již existující. Formatter si stav objektu vnitřně pamatuje a SQL dotaz generuje podle potřeby. Aplikační programátor nemusí mezi těmito dvěma případy rozlišovat, pro něj existuje pouze metoda pro uložení.

3.8.4 Možná rozšíření, použití

Hlavní výhodou je možnost jednoduché manipulace s daty v tabulce relační databáze.

Protože každý objekt musí být svázán s připojením k databázi, musí být toto spojení vytvořeni při vytvoření objektu. Je více možností:

- Specifikovat spojení při každém vytváření zvlášť, například předání v konstruktoru objektu.
- Provázat vytváření objektu s konkrétní konfigurační třídou, který může existovat pouze v jedné instanci – tzv. *singleton*. Tato třída by zajišťovala otevření komunikace s SQL serverem a při vytváření objektu by nebylo nutné toto spojení explicitně zadávat.
- Kombinace výše uvedených možností, implementovat obě varianty.

Další velkou výhodou je jednoduchost a rychlost používání. Nevýhodou to, že složitější dotazy na databázi se musí stále provádět pomocí původního programového rozhranní.

Možností rozšíření by bylo možnost načítání přímo celých dalších entit, pokud objekt (tabulka) obsahuje referenci na další objekt (tabulku). Např. tabulka *Osoba* obsahuje vazbu do takulky *Adresa*. Možností by bylo upravit generátor SQL kódu tak, aby všechna data byla přečtena jedním SQL dotazem za použití `JOIN`.

3.9 Webová služba

Webová služba je nadstavba nad C# knihovnou, která usnadňuje využívání služeb knihovny pomocí uživatelského rozhraní.

Webová služba poskytuje rozhranní pro komunikaci uživatele a služby zajišťující vytváření mazaiky.

Protože vytvoření obrázku je časově a výpočetně náročný úkol, který by mohl trvat do pozdější doby, než by došlo k vypršení spojení mezi serverem a klientem (timeout), byla webová služba rozdělena na dvě hlavní části - prezenční (uživatelské rozhranní) a výkonnou (službu pracující na pozadí).

Prezenční část je informační systém pro správu, zadávání a zobrazování požadavků. Pomocí jednoduchého grafického rozhranní zpřístupněného přes webový prohlížeč může uživatel uživatel procházet požadavky a vlastní přidávat.

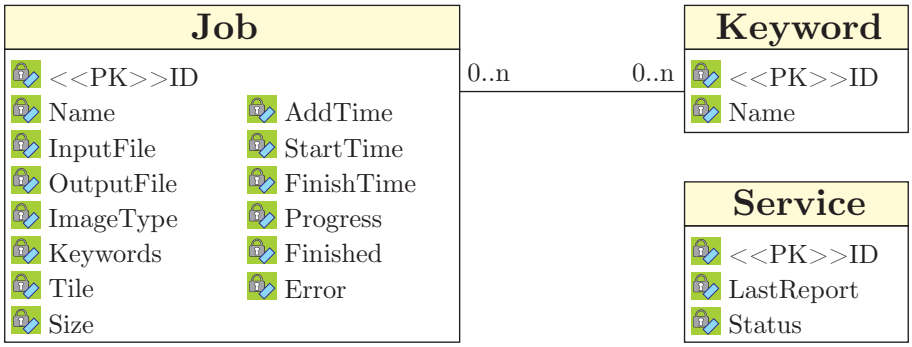
Požadavek má vlastnosti, které jsou nutné pro zpracování požadavku. Například vstupní soubor, který byl na server uživatelem nahrán, čas přidání, stav provádění, atd.

Při přidávání požadavku může uživatel specifikovat databázi (viz. 4.8 a 3.3), velikost vstupního a výstupního čtverce a typ výstupního obrázku (PNG nebo JPEG).

Při přidání požadavku je požadavek zařazen do fronty, kde čeká, až bude službou pracující na pozadí vybrán a zpracován.

Služba na pozadí se dotazuje na dosud nezpracovaný požadavek a v případě, že takový nalezne, označí jej stavem zpracováváno a spustí vytváření mozaiky.

3.9.1 ER diagram



Obrázek 3.4: ER diagram webové služby

3.9.1.1 Job

Úloha (ať již zpracovaná nebo zatím ve frontě) je ústřední entitou informačního systému.

- Name – Pojmenování požadavku (popis toho co je na obrázku)
- InputFile – vstupní soubor na serveru
- OutputFile – výstupní soubor na serveru
- ImageType – typ výstupního obrázku (může být JPEG nebo PNG)
- Keywords – klíčová slova z nichž bude složena databáze, ze které se bude generovat mozaika
- Tile – velikost vstupního čtverce
- Size – velikost výstupního čtverce
- AddTime, StartTime, FinishTIme – časy vložení, počátku a dokončení zpracování
- Progress – Procentuální vyjádření dokončení úlohy
- Finished – příznak, zda je úloha dokončena
- Error – Příznak, zda při zpracování požadavku nedošlo k chybě

3.9.1.2 Keyword

Keyword – klíčové slovo (název databáze). V tabulce jsou uložena všechna dostupná klíčová slova. Tato klíčová slova byla použita jako dotaz na Google (viz. 4.8) nebo jinak vystihují skupinu (databázi) obrázků.

3.9.1.3 Service

Monitorování služby (viz. 4.7.1) se provádí pomocí zvláštního vlákna, které při spuštění služby zapisuje do této tabulky cyklicky datum a čas aktualizace.

Teoreticky by jedna tabulka mohla být využita na monitorování více služeb, v tomto projektu je ovšem pouze jedna služba, takže v tabulce je využit pouze jeden řádek.

Monitorovaná služba je identifikování pomocí jedinečného GUID (viz. [4] nebo [6]). Toto GUID se musí náhodně zvolit a specifikovat v konfiguraci aplikace.

Do sloupce LastReport se ukládá datum a čas posledního ohlášení služby a do Statusu se ukládá aktuální stav služby (služba čeká na požadavky nebo služba zpracovává požadavek).

Pokud je poslední ohlášení služby starší než vhodně zvolený čas, je služba považována jako nefunkční. Tento stav lze pak v webové části zobrazit.

3.9.2 Knihovna pro komunikaci

Webové rozhranní i služba pro provedení požadavků na pozadí používá pro komunikaci databázi a tudíž mnoho částí projektu je společných. Právě tyto společné třídy a metody jsou umístěny v další knihovně.

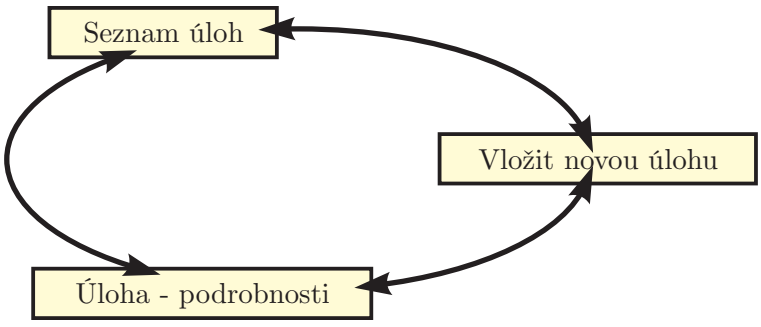
Těmito společnými částmi je například navázání spojení s databází, přístup k obsahu databáze, přístup k monitorovací službě, třídy (entity).

3.9.3 Uživatelské rozhranní

Webové rozhranní je uživatelsky přívětivá možnost zadávání a prohlížení požadavků.

Použití webového rozhranní je možné odstínit uživatele od použitého operačního systému. Klient není limitován operačním systémem ani žádnými dalšími omezeními plynoucí z použití konkrétního hardware, operačního systému ani dalších programů. Jedinými požadavky je mít přistup k webové službě (přes síť Internet), webový prohlížeč a schopnost zobrazovat obrázky. Pro kompletní funkčnost musí jeho webový prohlížeč umět posílat soubory.

Ukázka webového rozhraní je na obrázku 4.2.



Obrázek 3.5: Struktura stránek a přechodů webové služby

Uživatelské prostředí dovoluje uživateli zadávat požadavky na zpracování.

Zadání požadavku se provádí zvolením souboru s obrázkem a zvolením doprovodných parametrů.

Jako další parametry může uživatel zvolit:

- Název – název požadavku (uživatelské označení)
- Seznam klíčových slov – určuje tématiku a rozsáhlost použité databáze.
- Formát výstupního obrázku – lze zvolit formát výstupního obrázku (viz. 4.2)
- Vstupní a výstupní velikost čtverce – algoritmus dokáže pracovat s různými vstupními velikostmi taktéž dokáže upravit vstupní obrázek na požadovanou velikost (změna velikosti obrázku)

Kapitola 4

Implementace

4.1 Barevný model, barevná hloubka

Program interně pracuje v pouze jedné barevné hloubce využívající RGB model. Program pro ukládání do interní paměti využívá na jeden pixel celkem 3 byty, pracuje tedy s barevnou hloubkou 24 bitů.

U souborů JPEG předpokládá právě tento formát. PNG obrázky automaticky pomoci vnitřních filtrů knihovny převádí na 24 bitů, případnou průhledností složku (alfa kanál) odstraňuje.

4.2 Optimalizace ukládání obrázku

Knihovny pro práci s PNG (viz. 3.2.2.2) a JPEG (viz. 3.2.2.1) dovolují vytvářet obraz postupně po jednotlivých řádcích. V rámci optimalizace využití paměti tento program využívá této možnosti a zapisuje obrázky postupně.

Při zpracování požadavku je paměť přidělena pouze pro jeden výsledný obrazový blok.

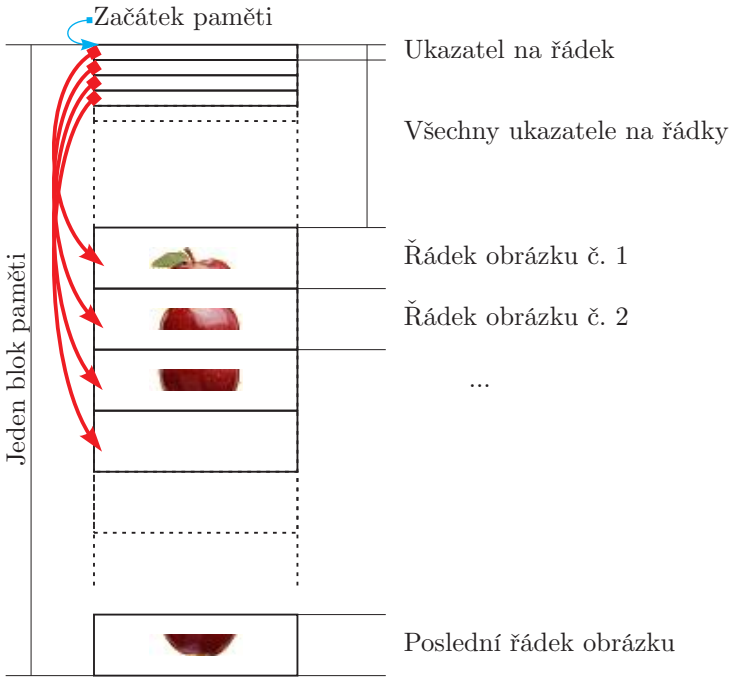
Výhodou je nízká paměťová náročnost – celý výsledný obrázek, který je často navíc obrovský, nemusí být uchováván v paměti, v některých krajích případech by mohlo docházet až ke swapování a celkovému zpomalení systému.

Program dokáže vytvořit tak velký obrázek, který nemusí být ani klient schopen zobrazit (prohlížeč jej odmítne jako příliš velký).

4.3 Optimalizace alokace paměti obrázku

Rastrový obrázek lze brát jako dvourozměrnou matici, a proto také intuitivně je jednodušší implementovat vnitřní uložení jako pole polí (jako matici). Tento přístup má ovšem tu nevýhodu, že při alokaci obrázku dochází k mnohonásobné alokaci, dochází tedy k tříštění paměti a protože operační systém většinou přiděluje paměť po pevně velkých blocích, může docházek k alokaci paměti, která nakonec nebude využita.

Alokací přesně potřebné velikosti pole může sice také dojít k nevyužití paměti, ovšem řádově menší. Názorné provedení alokace je zobrazeno na obrázku 4.1.



Obrázek 4.1: Schéma rozložení paměti pro interní uložení obrázku

4.4 Obrázky a vyrovnávací paměť

Získání informací o obrázku je časově náročná činnost. Pro získání informací, které jsou potřeba k výpočtu je potřeba obrázek načíst, projít pixel po pixelu a spočítat charakteristiky a vlastnosti obsahu obrázku.

Při velkém množství obrázků (desítky tisíc) může být paměťová náročnost obrovská a pro dnešní systémy nesplnitelná.

Proto před vlastním výpočtem je potřeba indexovat (předzpracovat) obrázky. Pro každý načtený obrázek se zjišťuje průměrná barva, pak se rozdělí na matici 4×4 a pro každou matici se opět spočítá průměrná barva.

Tato databáze se pak uloží do textového souboru společně s cestou k souboru a klíčovým slovem. Taktováto databáze je pak připravena pro použití vytváření mozaiky.

Nutnost použití vyrovnávací paměti vychází z použitého algoritmu (popis ve zvláštní kapitole 2.5).

4.5 Použité vývojové nástroje

4.5.1 Visual Studio 2005

Microsoft Visual Studio je integrované vývojové prostředí pro vývoj aplikací. Může být využito pro vývoj aplikací pro konzoli, s grafickým uživatelským prostředím, webových stránek, webových služeb.

Je podporován vývoj pomocí nativních jazyků, ale i pomocí tzv. *managed code*. Editor podporuje *IntelliSense*, které automaticky nabízí automatické dokončování kódu.

V době psaní programu existovala sice již novější verze .NET Frameworku (verze 3.5) i Visual Studia (verze 2008), ovšem tato starší verze je naprosto dostačující.

Technologie, nástroj	Verze, norma
C	ISO C99
jpeglib	6b (Win)
libpng	1.2.28
C#	2.0
.NET Framework	2.0
ASP.NET	2.0
Microsoft SQL Server	9.0
Microsoft IIS Server	7.0
PHP	5.2.5
XHTML	1.0
CSS	1.0
Javascript	1.5
Last.fm API	2.0
Subversion	1.5.2
Apache	2.2.9

Tabulka 4.1: Tabulka konkrétně použitých technologií, nástrojů a programovacích jazyků

4.5.2 SQL Server Management Studio

Microsoft SQL Server Management Studio je nástroj distribuovaný společně s Microsoft SQL Server pro konfiguraci a správu všech částí SQL Serveru.

Obsahuje editor skriptů ale i grafické uživatelské nástroje pro správu.

Pomocí tohoto nástroje lze vytvářet, měnit a prohlížet databáze i tabulky (ale i mnoho dalšího).

4.6 Použité technologie

Vlastní hlavní programová část je napsána v C, ale nadstavba je pro rychlejší vývoj napsána ve vyšším programovacím jazyce. Kvůli tomuto postupu při programování muselo být využito plno technologií (tabulka 4.1).

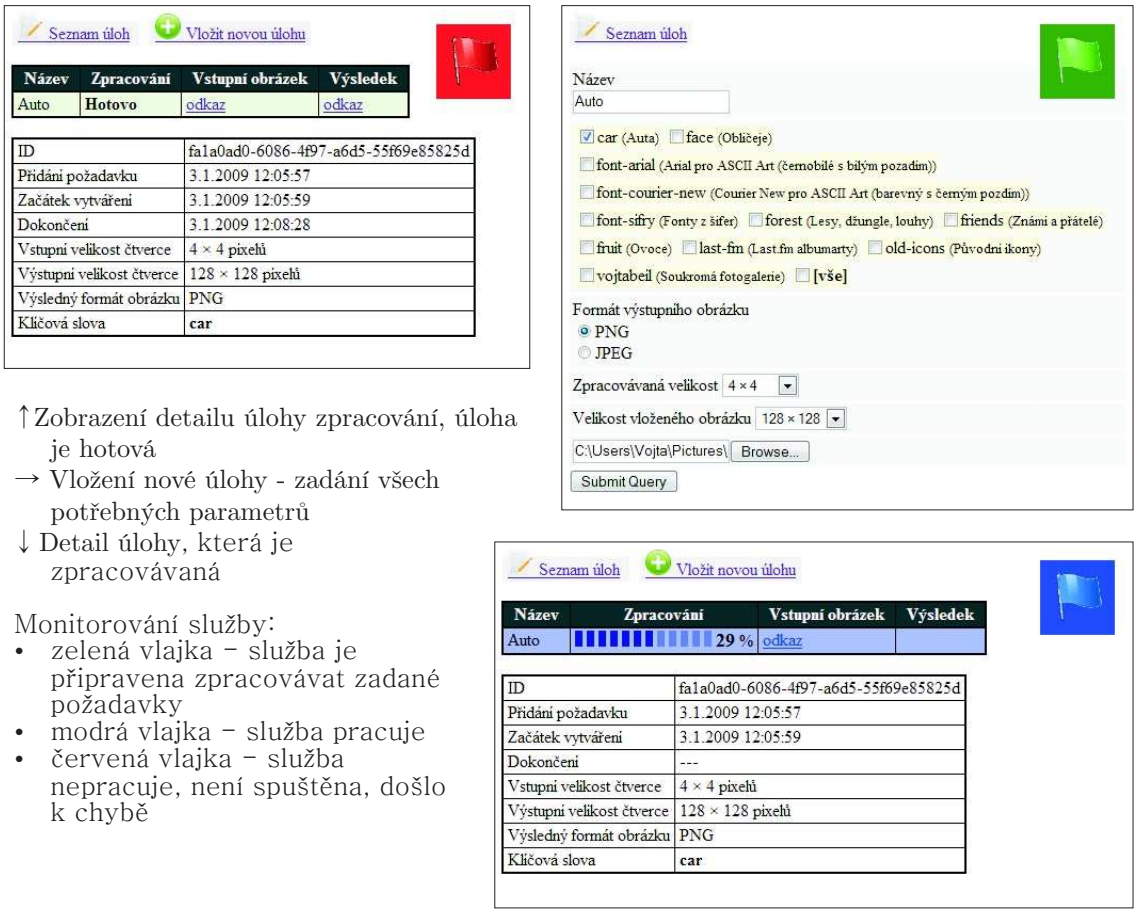
Jejich propojení není natolik složité, aby to činilo neuskutečnitelný problém. Všechny použité technologie jsou otevřené a spolupracují spolu dobře, v průběhu implementace nedocházelo ke konfliktům či vzájemné nekompatibilitě.

Během implementace bylo (pro zálohu, možnost vracet staré verze a k dokumentačním účelům) využíváno Subversion. I když je tento nástroj určen hlavně pro práci více programátorů, jeho využití se velmi osvědčilo. Používání Subversion umožňuje zobrazovat staré verze, vracet se k původním, při náhodné ztrátě souboru, adresáře či části kódu je k dispozici záloha.

4.7 Webová služba

Webová služba je implementována v ASP.NET bez použití *Web Server Controls*. Webová služba je napsána pomocí XHTML, CSS (kaskádové styly) a Javascriptu (Javascript slouží pouze k doplňkovým funkcím, pro vlastní běh není nutný).

Webová služba je napsána minimalisticky tak, aby fungovala ve většině prohlížečů. K zobrazení stránek není potřeba podpora Javascriptu ani CSS. Stránky tak lze prohlížet



↑Zobrazení detailu úlohy zpracování, úloha je hotová
→ Vložení nové úlohy - zadání všech potřebných parametrů
↓ Detail úlohy, která je zpracovávaná

Monitorování služby:
- zelená vlajka – služba je připravena zpracovávat zadané požadavky
- modrá vlajka – služba pracuje
- červená vlajka – služba nepracuje, není spuštěna, došlo k chybě

Obrázek 4.2: Webová služba v akci

a používat i na zařízených, které nemají „velký“ internetový prohlížeč. Takovýmito zařízeními jsou například telefony, PDA, komunikátory.

4.7.1 Monitorování služby

Informace o stavu provádění požadavku se ukládá do databáze.

Fungování pracuje na předpokladu, že pokud běží program na zpracování požadavků, je služba dostupná.

Služba pro zpracování běží ve dvou oddělených vláknech. Jedno se stará a vlastní zpracovávání, druhé periodicky zapisuje stav druhého vlákna s časovým údajem do databáze.

V případě ukončení služby, skončí také druhé vlákno, k žádnému zápisu již nedojde a klientský program podle doby posledního zápisu zjistí, že služba není funkční.

4.8 Stahování obrázků

Pro práci algoritmu je potřeba v ideálním případě několik tisíc různých obrázků v databázi, ze kterých se vytváří výsledná mozaika.

Pro názorné použití programu je idální použít obrázky aut při vytváření obrázku auta.

Protože ruční zařazování obrázků by bylo časově velice náročné, vznikl nápad vytvořit

jednoduchý program (skript), který bude automaticky z indexovacích služeb (viz. 3.3) stahovat obrázky a automaticky je zařazovat (přiřazovat tagy) do skupin.

Google nabízí možnost vyhledávat obrázky podle vloženého klíčového slova (textu). Tuto službu nabízí ale i další vyhledávací (indexovací) internetové služby.

Experimentováním s více službami, jsem zjistil, že většina služeb omezuje zobrazení (nalezení) asi na tisíc obrázků. Takto velká databáze je podle mnou provedených experimentů k rozumnému výsledku nedostatečná. Naštěstí některé služby dávají možnost vyhledat obrázky různých velikostí, čímž lze získat více více obrázků z jednoho dotazu.

Nakonec pro stahování obrázků (a vytváření databáze) vybral jako zdroj Google a jeho službu Image Search.

Samotný skript je naprogramován pomocí PHP. (Využívá se toho, že PHP může fungovat jako interpret.) Na požadovaný dotaz skript nalezne pomocí služby obrázky a jejich miniatury uloží do jednoho adresáře, který je pak připraven pro indexování.

K bakalářské práci je též přiložen skript, který dokáže pomocí Last.fm API nalézt pro konkrétního uživatele jeho nejposlouchanější skupiny a stáhnout k nim obrázky alb. Tyto stažené obrázky pak lze použít pro vytvoření databáze.

Kapitola 5

Výsledky

Celý plánovaný systém se podařilo zdárně implementovat.

Webová část i služba zpracovávající požadavky je funkční a stabilní.

Výsledné obrázky jsou si věrně podobné. Při vzdálenějším pohledu na obrázek, je původní předloha dobře rozpoznatelná.

Barevnost vybraných obrázků odpovídá předloze. Vybraný obrázek při pohledu z dálky sedne do výsledného obrázku.

Hrany se algoritmus snaží zachovávat. Při přiblížení je vidět, že pokud někudy vedl ostrý, barevně výrazný přechod, algoritmus se snaží nalézt obrázek s velmi podobným přechodem.

Výkon programu je v rámci možností rychlý – vytváření průměrné mozaiky trvá v řádu desítek sekund maximálně několik minut. Tímto se potvrdil původní předpoklad, že mozaiku není možné vytvářet za běhu, ale je nutné rozdělit zadávání a vytváření do oddělených částí.

Program zvládá databázi několika desítek tisíc obrázků v databázi. Programu nečiní problém vytvořit opravdu velkou mozaiku (např. větší než 7000×5000 pixelů). U větších obrázků vzniká spíše problém s jejich následným zobrazením.

Na druhou stranu algoritmus sice relativně dobře zvládá hrany, ale jemné postupné přechody ve většině případů působí problémy a ve výsledném obrázku se takovýto přechod zvýrazní.

Výkon je sice relativně dostatečný pro testované databáze o obrázky, návrh algoritmů pro výběr mozaiky způsobuje lineární náročnost vytváření mozaiky. Při použití o řád větší databáze (miliony obrázků) by zpomalení bylo citelné.

V programu existuje několik konstantních parametrů, které ovlivňují výslednou mozaiku, užitetel nemá možnost jejich hodnotu měnit. Tyto hodnoty jsou experimenty nastaveny kompromisně tak, aby výpočet nebyl příliš dlouhý a aby byl dostatečně kvalitní.

Výběr obrázku, který bude použit do mozaiky, se skládá ze tří úrovní. Mezi každou úrovní je definován počet obrázků, které jsou zařazeny i do další úrovně. Zvětšením těchto hodnot by mohlo k velkému nárůstu náročnosti.

Další konstantou je penalizace za použití obrázku. Snahou samozřejmě je, aby se obrázky vedle sebe co nejméně opakovaly, proto je při použití obrázku připočtena penalizace, která mu zhorší výběr v následujících krocích. Může se ale stát, že obrázek je nejlepší kandidát a vložení jiného by pokazilo celkový dojem z mozaiky. Z tohoto důvodu je penalizace nastavena přes mnoho pokusů pevně.

Asi největší slabinou při vytváření mozaiky je databáze obrázků. Ideální vytváření databáze je ručně nebo z obrázků, kde je zaručeno, že obrázek opravdu obsahuje požadovanou tématiku. Např. při hledání aut by bylo ideální, aby na každém obrázku bylo auto, ovšem vyhledávací služby nabídnou obrázky vzhledově ne zcela odpovídající této představě.

5.1 Popis programu pro příkazovou řádku

Program pro příkazovou řádku je nejblíže implementaci vlastního algoritmu a tudíž poskytuje nejvíce možností práce s mozaikou i databázemi obrázků.

Nápověda ke všem parametrům programu se získává příkazem:

```
Mosaic.Console.exe --help
```

Vytvoření souboru *auta.db* z adresáře *auta* a nastavení klíčového slova na *car*:

```
Mosaic.Console.exe --keyword car --lookup auta --load_files
    --save_db auta.db
```

Vytvoření mozaiky z obrázku *male_auto.png* s databází ze souboru *auta.db* jako *JPEG* do souboru *velke_auto.jpg* se vstupní velikostí 16×16 pixelů a výstupní velikostí vložených obrázků 32×32 pixelů:

```
Mosaic.Console.exe --input male_auto.png --output velke_auto.jpg
    --jpeg --load_db auta.db --tile 16 --size 32
```

5.2 Popis webové služby

5.3 Ukázky výstupů

Ke shodnocení výsledku bakalářské práce přikládám dvě výsledné mozaiky (obrázky 5.1 a 5.2).

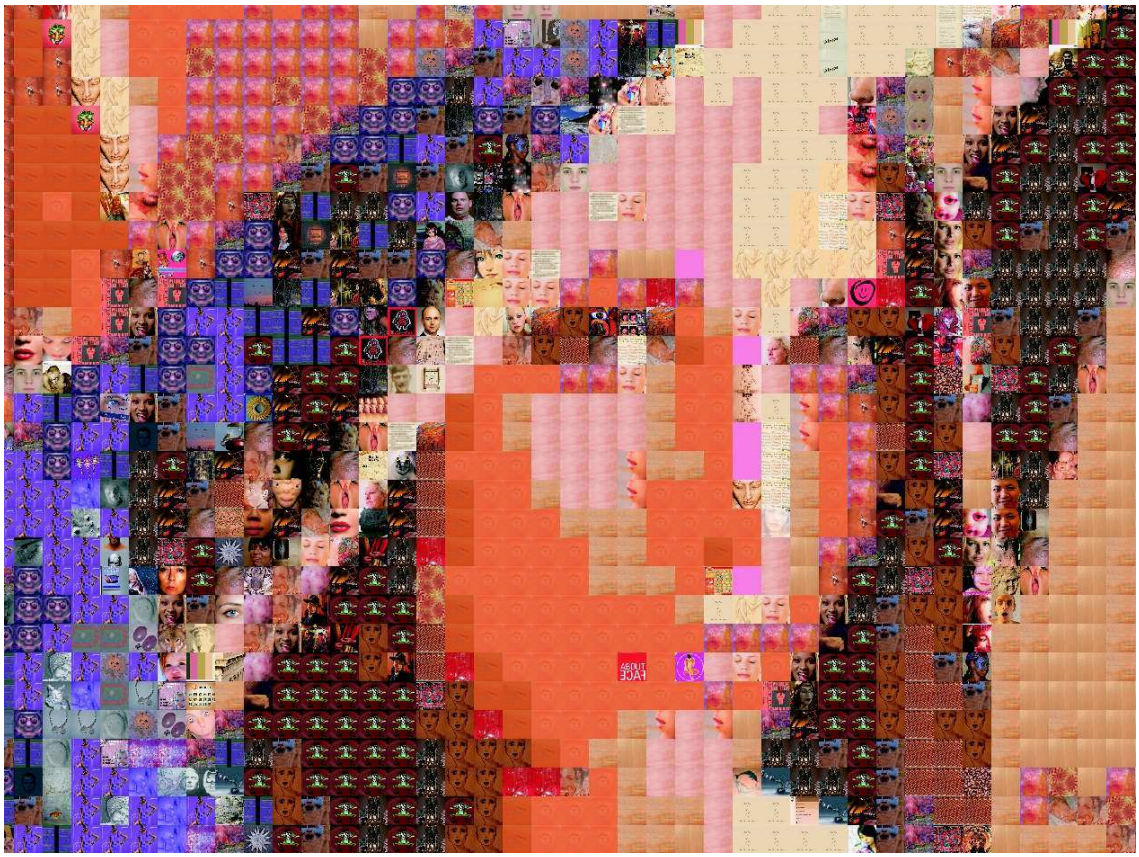
5.4 ASCII art

Jedna z možných využití této bakalářské práce je možnost používat ji k tvoření ASCII artu (skládání písmen a znaků tak, aby z větší dálky působil výsledek jako obrázek).

Pokud se jako vstupní databáze použijí obrázky tvořené písmeny je pomocí programu možné vygenerovat ASCII art. S vhodnou databází je dokonce možné vygenerovat barevnou verzi.

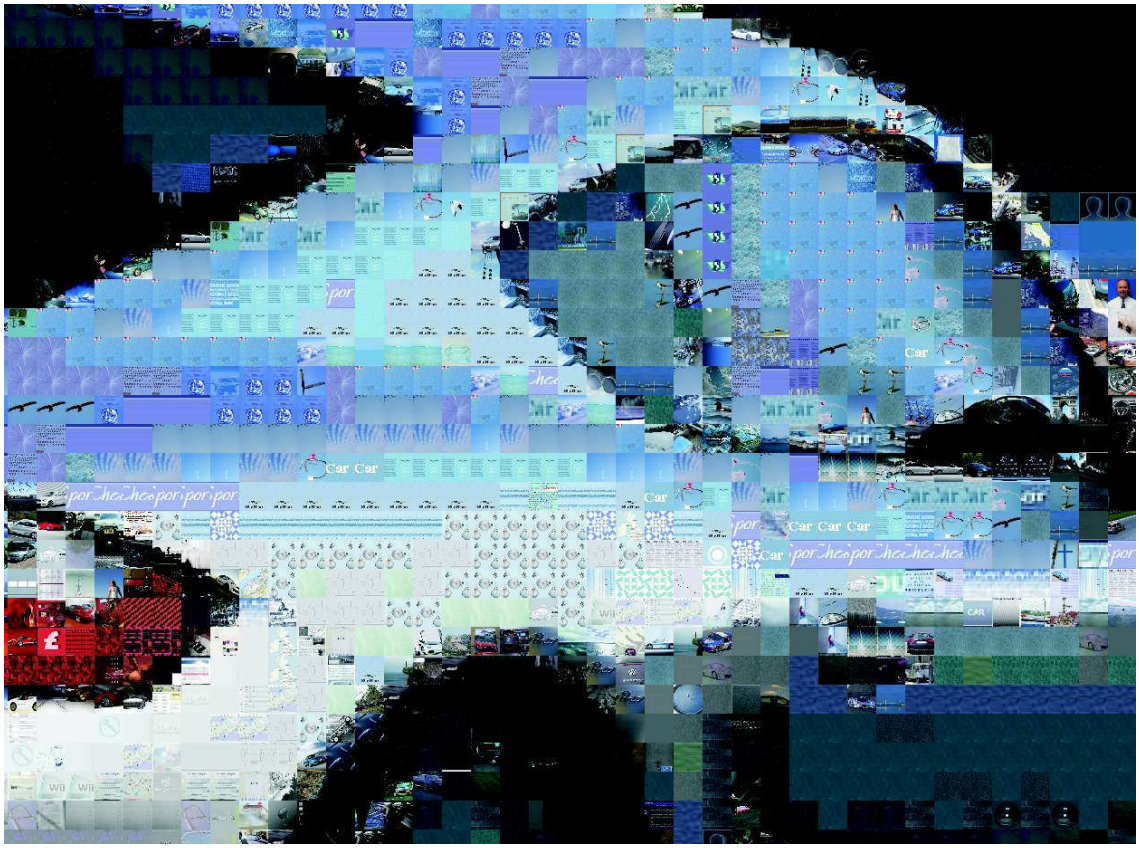
Ukázka za použití černobílé databáze je na obrázku 5.3.

↑ Vstupní obrázek
→ Výstupní obrázek vstupní dlaždice 8×8 pixelů, výstup 32×32, použity obrázky nalezené pomocí Google pro dotaz “face”
↓ Tentýž výstup s detailem tváře

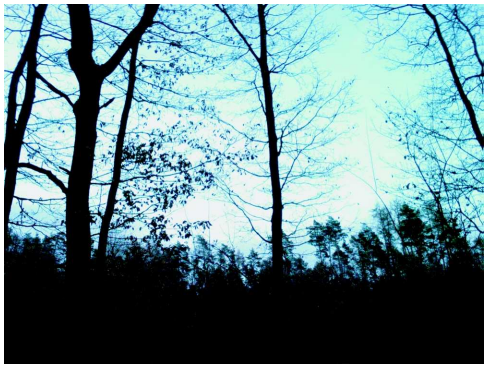


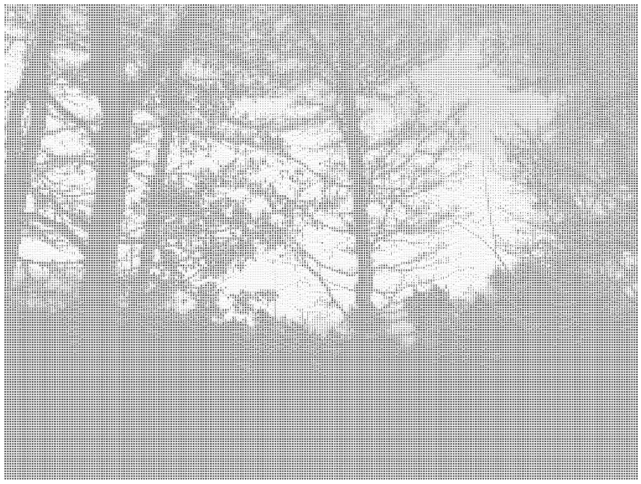
Obrázek 5.1: Příklad vstupu a výstupu

↑ Vstupní obrázek - Porshe Cayman
→ Výstupní obrázek - vstupní dlaždice 8×8 pixelů, výstup 32×32, použity obrázky nalezené pomocí Google pro dotazů “car”, “black car”, “blue car”, “red car”, “bmw car”, “porshe car”, atd. Celkem použita databáze o velikosti asi 120 tisíc obrázků.
↓ Detail auta, za povšimnutí stojí použití obrázků pro ostré přechody.
Zdroj: Porshe Cars North America: http://www.porsche.com/usa/models/cayman/cayman-s/

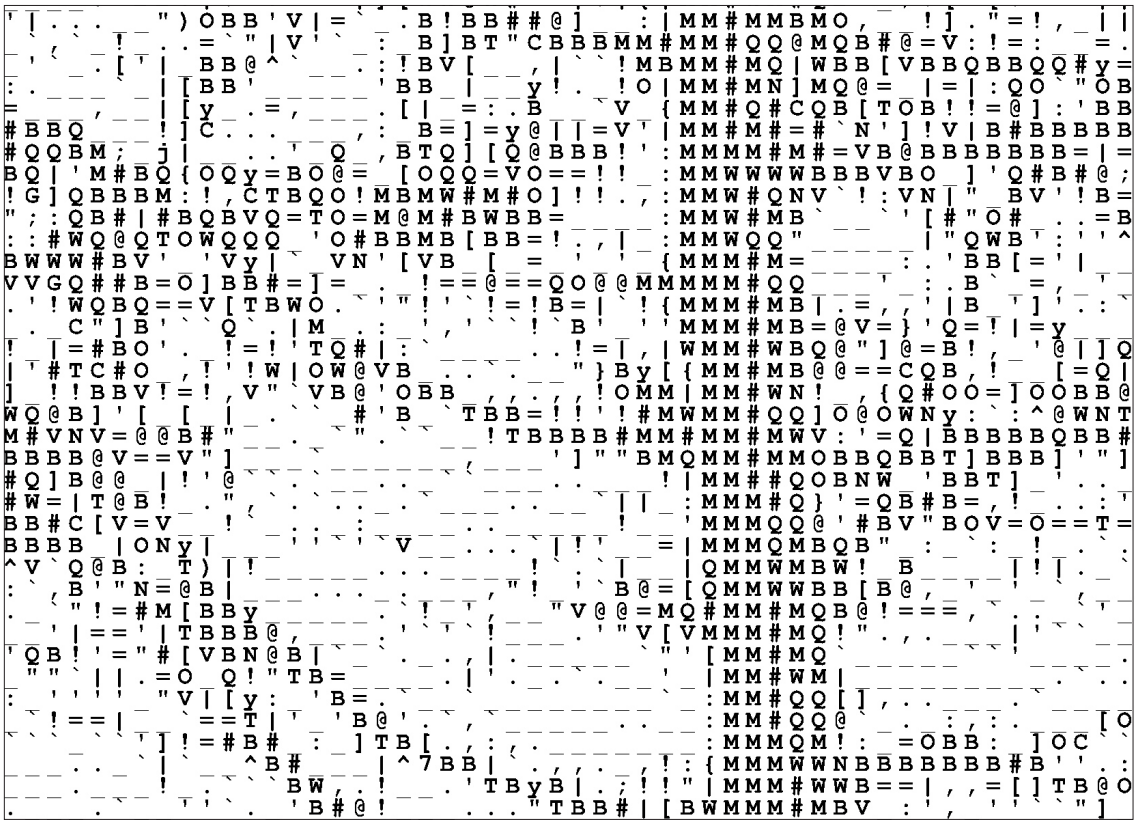


Obrázek 5.2: Porshe Cayman





↑ Vstupní obrázek - Les v Soběšicích
→ Výstupní obrázek - ukázka ASCII artu
↓ Detail části obrázku lesa - obrázek zobrazuje část kmene a využití velké části znaků



Obrázek 5.3: Další z možností využití – ASCII art

Kapitola 6

Závěr

6.1 Možná rozšíření

Na závěr uvedu několik dalších možností na další vylepšení a rozšíření. Tyto nápady vznikly při implementaci a jejich začlenění by bylo náročné a vyžadovalo by mnoho změn do již zpracovaných postupů. Další možná rozšíření jdou mimo rozsah a zadání této bakalářské práce.

6.1.1 Podpora dalších grafických formátů

Aktuální implementace podporuje čtení a zápis pouze do dvou různých formátů – JPEG a PNG. Tyto dva formáty byly vybrány, protože většina obrázků je dostupných právě v těchto formátech, dovolují postupný zápis do souboru (v paměti nemusí být uchován celý obrázek), existuje volně šiřitelná implementace a použití v projektech je jednoduché.

Dalším rozšířením by mohla být podpora čtení i zápisu dalších rastrových obrázků – např. GIF, BMP nebo TIFF.

6.1.2 Použití velkých fotografií a jejich částí

Jako rozšíření by také mohlo být doprogramováno načtení velké fotografie a její rozřeznání na menší. Problémem je výběr obrázků s dostatečně výraznými jednotlivými částmi.

Částečně v aplikaci podpora pro rozřezávání existuje, její použití a zdokumentování naráží na problém neslučitelnosti s aktuálním rozvržením databáze a s vnitřními pochody, které se starají o cachování obrázků.

Toto rozšíření se spíše ukázalo jako vhodný test pro správnou funkčnost algoritmu, kdy byl proveden pokus o sestavení obrázku z jeho vlastních částí. Počtem úspěšných umístěním části na původní místo lze měřit správnost a přesnost algoritmu.

6.1.3 Sdílení paměti

Načtení databáze do paměti z disku je relativně výkonově náročná operace, navíc v aktuální implementaci je tato struktura vytvářena pro každé zpracování vytvářena zvlášť.

Toto implementační rozhodnutí bylo zvoleno kvůli tomu, že je možné používat různé databáze obrázků při vytváření. Navíc by mohl vzniknout problém při cachování (aktuálně to znamená načtení a změna velikosti na požadované rozměry – tyto hodnoty se mohou mezi požadavky měnit, musel by být zvolen jiný přístup při implementaci).

Návrh zlepšení je takový, že tato databáze v paměti by mohla být stabilní, neměla by se vytvářet znovu při každém požadavku, a pokud by program běžel jako služba, mohla by tato paměť být využívána opakovaně.

6.1.4 Hash obrázků

Aktuální implementace je sice rychlá, ale vyhledávání podobných obrázků by při nárustech databáze mohlo vést k problémům s rychlostí. K vyřešení tohoto problému by mohlo být využití obrázkového hashe (viz. zdroj [17]).

Hash obrázků by měl mít takové vlastnosti, že vzhledově podobné obrázky by měly mít taktéž podobné hashe. A naopak rozdílné obrázky by hash měly mít co nejvíce rozdílné.

Při hledání podobných obrázků by se mohlo hledat podle hashů a při použití správných algoritmů, by mohlo dojít k urychlení hledání.

6.1.5 Rozestavění obrázků

(a) (b)

Obrázek 6.1: Možnost použít i jiné rozestavění dlaždic

Aktuální implementace počítá se čtvercovým rozestavěním mozaiky. Jako rozšíření do budoucna by výslená mozaika mohla být skládána z jiných tvarů, které lze uspořádat do pravidelné mřížky. Na obrázku 6.1 je provedeno visuální porovnání rozmístění stejných barev do pravidelné čtvercové a šestiúhelníkové mřížky.

6.1.6 Grafické uživatelské rozhraní

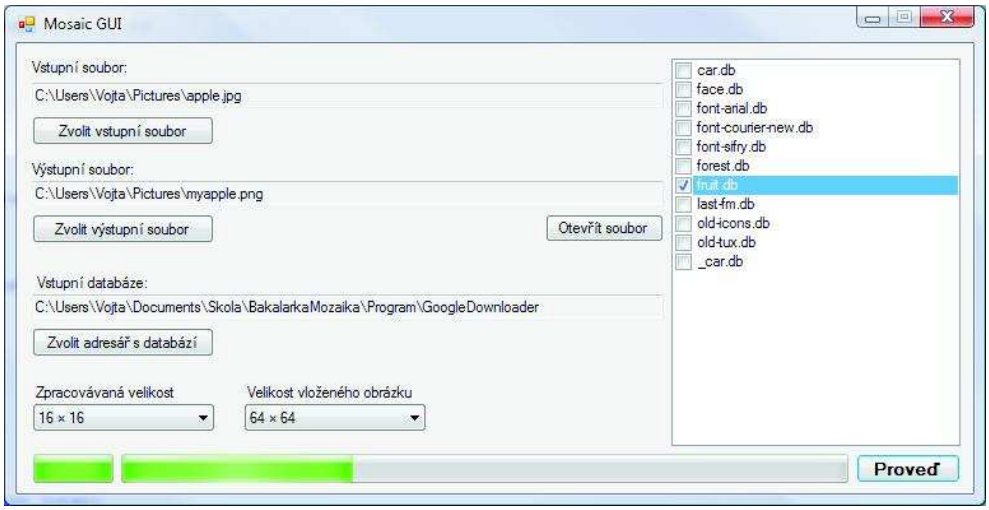
Možným rozšířením by mohlo být samostatné uživatelské rozhraní nezávislé na SQL databázi. Možný vzhled je uveden na obrázku 6.2.

Vytvoření uživatelského rozhraní je díky modularitě projektu možné vystavět na *Mosaic.Lib.Static* (v případě použití C/C++ nebo i jiného jazyka, který je možný propojit s moduly psanými v jazyce C) nebo *Mosaic.Class* pro .NET platformu (architektura Win32, v případě portování *Mosaic.Lib.Dynamic* i další).

6.1.7 Zobrazení velkých obrázků online

Pro webovou službu je možné použít prohlížeč velkých obrázků. Uživatel pak může plynule procházet obrázkem, zvětšovat a zmenšovat a přitom jsou k němu ze serveru přenášena pouze data minimálně nutná pro zobrazení.

Takovýmto projektem je například IIPImage `http://iipimage.sourceforge.net/`



Obrázek 6.2: Možný vzhled uživatelského rozhranní

6.1.8 Rozšíření webové služby

Aktuální implementace webové služby nepřináší žádný mechanismus zabezpečení či uživatelských účtů. Při implementaci by bylo možné přidat možnost zvolení veřejné nebo neveřejné mozaiky nebo možnost zasílání informace o zpracování mozaiky na email.

V případě opravdového nasazení by bylo potřeba vytvořit zabezpečenou administraci, kde by bylo možné požadavky upravovat a mazat. Taktéž by mělo být možné spravovat běh samotné služby – možnost pozastavit, resetovat a znovu spustit.

6.2 Závěr

Původní návrh se podařilo úspěšně implementovat.

Před vlastním zadáním práce jsem měl k dispozici základní pro osobní potřeby napsanou implementaci algoritmu, takže jsem měl z čeho vycházet a plno zkušeností jsem již získal dříve.

Mohl jsem se tedy soustředit na rozšiřování a zlepšování.

Opravdu velkou výzvou byly všechny použité optimalizace, původní implementace byla napsána narychlo a nikdy se nepočítalo s tím, že by mohla běžet jako služba (tedy s minimální paměťovou náročností a bez memory leaků). Použití většiny algoritmů bylo znovu zváženo a většinou byly algoritmy nahrazeny jinými nebo přepsány.

Dalším velkým úkolem bylo tento původně v jazyce C psaný program zprovoznit tak, aby fungoval a spolupracoval s kódem pro .NET framework. K tomuto byla použita jako mezistupeň *.dll* knihovna.

Následné spojení s databází a zapracování jako jednoduchý informační systém bylo již díky použitým technologiím relativně jednoduché. Původně jsem zvažoval místo použití Microsoft Internet Information Services (IIS) možnost spolupráce s Apache serverem a naprogramování webové služby v PHP. Nakonec jsem se rozhodnul pro tuto variantu hlavně kvůli osobnímu zájmu o .NET platformu.

Další velmi zajímavou částí je Formatter (kapitola 3.8), který velmi zjednodušil možnost práce s databází. Možná k času vývoje a testování se jedná o zbytečně složitý nástroj. Jeho znovuvyužitelnost je ovšem z mého pohledu velká a zjednodušuje k základním operacím nad relační databází.

V závěru jsem se pokusil ještě naznačit možná další vylepšení a cesty, kudy by další vývoj mohl pokračovat.

Vytváření mozaiky je velice zajímavou činností a mnoho lidí obrázkové mozaiky fascinují.

Literatura

[1] Troelsen, A.: *C# a .NET 2.0 profesionálně*. 1. vydání, Brno: ZONER Press, 2006, 1198 s, ISBN 80-86815-42-0

[2] IJG.ORG: Independent JPEG Group. [online], [cit. 2008-11-27], `http://www.ijg.org/`

[3] *The Client Server Architecture*. [online], [cit. 2009-01-10], `http://www.webdevelopersnotes.com/basics/client_server_architecture.php3`

[4] Leach P., Microsoft, Mealling M, Refactored Networks, LLC, Salz E, DataPowe Technology Inc: *RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace* [online]. [červenec 2005], [cit. 2008-12-15], [online], `http://www.ietf.org/rfc/rfc4122.txt`

[5] Fieldng R., UC Irvine, Gettys J., Compaq/W3C, Mogul J., Frystyk H., W3C/MIT, Masinter L., Xerox, Leach P., Microsoft, Berners-Lee T.: *RFC 2116: Hypertext Transfer Protocol – HTTP/1.1*, [červen 1999], [cit. 2008-12-26], [online], `http://www.ietf.org/rfc/rfc2616.txt`

[6] Microsoft Corporation: `MSDN: Guid Structure (System)`. [online], [cit. 2008-12-15], `http://msdn.microsoft.com/en-us/library/system.guid.aspx`

[7] Microsoft Corporation: *Dynamic-Link Libraries*. [online], [cit. 2008-12-18], `http://msdn.microsoft.com/en-us/library/ms682589.aspx`

[8] Microsoft Corporation: *Common Language Runtime*. [online], [cit 2008-01-04], `http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx`

[9] Microsoft Corporation: *ASP.NET State Management*. [online], [cit 2008-01-05], `http://msdn.microsoft.com/en-us/library/y5y3c2c5(VS.80).aspx`

[10] *ECMA-334: C# Language Specification* [online], 4^{th} edition, [cit. 2009-01-03], `http://www.ecma-international.org/publications/standards/Ecma-334.htm`

[11] Thyssen, Anthony: *Hints, Tips and Notes on bulk sorting of images* [online], [cit. 2008-01-06], `http://www.cit.gu.edu.au/~anthony/info/graphics/image_comparing`

[12] Tezaur, Radka: *Co se děje, když se obrázky zmenšují a zvětšují* [online]. 2001-12-10, [cit. 2009-01-08] `http://www.paladix.cz/clanky/co-se-deje-kdyz-se-obrazky-zmensuji-a-zvetsuji.html`

[13] *Barvy: Vlnové délky* [online]. [cit. 2009-01-08]
http://barvy.xf.cz/teorie/vlnove-delky

[14] Žák, Karel: *Historie relačních databází* [online], 2001-10-19, [cit 2008-01-08],
http://www.root.cz/clanky/historie-relacnich-databazi/

[15] Horváth, Tomáš: *Teoretický úvod do relačních databází* [online], 2007-11-08, [cit. 2008-01-09], http://programujte.com/index.php?akce=clanek&cl=2007110801-teoreticky-uvod-do-relacnich-databazi

[16] Reolofs, Greg: *Into to PNG Featurs* [online], 2008-01-30, [cit. 2009-01-10],
http://www.libpng.org/pub/png/pngintro.html

[17] Johnson M.: *Image Hashing.* [online], [cit. 2008-01-12],
http://www.eecs.berkeley.edu/~mjohnson/abstracts/hashing.htm

Přílohy

Přiložené DVD

Na přiloženém DVD se nachází:

- Zdrojové soubory – kompletní zdrojové soubory ke všem modulům v jazyce C, C#, PHP, SQL skripty na vytvoření databáze, zkompilované verze použitých knihoven (libpng, jpeglib)
- Kompilovaná verze – binární spustitelné verze programů (program pro příkazovou řádku, služba, webový projekt pro IIS), navíc je k dispozici jednoduché GUI
- Databáze obrázků – předpřipravené databáze s motivy: auta, příroda, tváře, ovoce, albumarty, fonty pro ASCII art (černobílý i barevný)
- Ukázky vstupu i výstupu – všechny vstupní i výsledné obrázky použité v této bakalářské práci, dále pak i další, vše v původním vysokém rozlišení
- Prezentační video – ukázka funkčnosti bakalářské práce: vložení požadavku, zobrazení průběhu výpočtu a výsledného obrázku
- Video – možné využití této bakalářské práce vytvořené pomocí programu `Video.Animator`