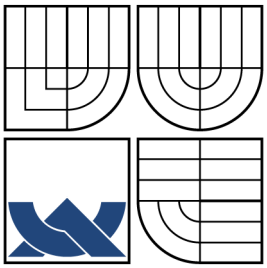
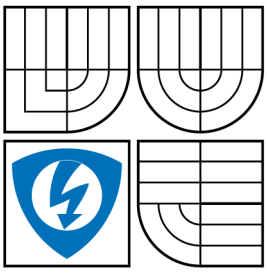


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO SROVNÁNÍ ALGORITMŮ PRO KOMPRESI OBRAZŮ

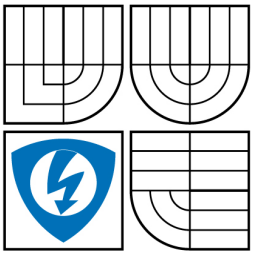
WEB APPLICATION FOR IMAGE COMPRESSION ALGORITHMS COMPARISON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL ŠANDA
AUTHOR

VEDOUCÍ PRÁCE — ING. ZDENĚK PRŮŠA
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Pavel Šanda ***ID:*** 98675
Ročník: 3 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Webová aplikace pro srovnání algoritmů pro kompresi obrazů

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte webovou aplikaci, která bude sloužit jako databáze obrázků různě zkreslených ztrátovou kompresí. Pro vývoj použijte nástroje založené na programovacím jazyce JAVA. Aplikace by měla umožnit současné vizuální i objektivní posouzení kvality obrazů.

DOPORUČENÁ LITERATURA:

[1] DORAY, A. Beginning Apache Struts, From novice to Professional, USA: APRESS, 2006. 536 s. ISBN 1590596048

[2] BAUER, CH., KING, G., Java Persistence with Hibernate, USA: Manning Publications, 2006. 904 s. ISBN 1932394885

[3] BASHAM, B. SIERRA, K.. BATES, B. Head First Servlets and JSP, USA: O'Reilly Media, Inc., 2008. 913 s. ISBN 978-0596-51668-0

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 2.6.2009

Vedoucí práce: Ing. Zdeněk Průša

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Bakalářská práce se zabývá návrhem webové aplikace pro vzájemné porovnání originálního a zkreslených obrázků, které jsou znehodnoceny ztrátovou kompresí. Obrázky mohou být posuzovány jak visuálně, tak dle parametrů k nim příslušících. Po přihlášení má uživatel možnost nahrávat obrázky na server a ukládat jejich parametry do databáze, dále je porovnávat, hodnotit, a také mazat. Práva v aplikaci jsou rozdělena dle příslušných rolí.

KLÍČOVÁ SLOVA

TOMCAT, JAVA, HIBERNATE, STRUTS, JSP

ABSTRACT

Bachelor´s thesis deals with concept of web application for comparing original and distorted images, which are deteriorated with losing compression. Images can be critized so visual as according to parameters, which belong to them. After logging on user is able to record images on the server and save their parameters to the database. Then he is able to compare them, and also evaluate or delete them. Rules in application are devided according to corresponding roles.

KEYWORDS

TOMCAT, JAVA, HIBERNATE, STRUTS, JSP

ŠANDA, P. *Webová aplikace pro srovnání algoritmů pro kompresi obrazů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 47 s. Vedoucí bakalářské práce Ing. Zdeněk Průša.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Webová aplikace pro srovnání algoritmů pro kompresi obrazů“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Zdeňku Průšovi, za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne ....

(popis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Bakalářská práce, na téma *„Webová aplikace pro srovnání algoritmů pro kompresi obrazů“*, se zabývá návrhem a realizací aplikace, která umožňuje vzájemné porovnání obrázků různě zkreslených ztátovou kompresí. Porovnání jednotlivých obrázků je možné posoudit subjektivně, ale také objektivně. K tomuto posouzení je nápomocné jak samotné zobrazení obrázků vedle sebe, tak i jejich vlastní parametry. Tyto parametry jsou uloženy v databázi, ale samotné obrázky se nachází na serveru. Uživatel má možnost ukládat, porovnávat a mazat obrázky. Dále je může posuzovat a následně ohodnotit, neboť hodnocení stejného obrázku různými uživateli je jedním z hlavních účelů této aplikace. Důvod proč je toto umožněno je takový, že každý uživatel může zkreslení vnímat odlišně.

Pro realizaci jsou použity nástroje založené na platformě Java. Konkrétně tedy již zmíněná Java, frameworky Struts a Hibernate, dále databáze MySQL a webový server Apache Tomcat. Všechny tyto nástroje jsou použity ve vývojovém prostředí NetBeans IDE. Pro návrh databáze byl využit program CASE Studio.

V první části této práce jsou popsány nástroje použité k vývoji aplikace. Jsou zde rozebrány výhody a nevýhody vybraných frameworků. Mezi nejvíce popisované frameworky patří Struts 1.x a Hibernate, neboť jsou v této práci používány. U serveru Tomcat jsou popsány jeho důležité konfigurační vlastnosti.

Druhá část se zabývá již samotným návrhem a realizací. Jsou zde popsány jednotlivé metody a postupy, jak bylo docíleno výsledné podoby aplikace. Také je z výsledku zřejmé jak frameworky usnadňují vývoj aplikací.

Celá tato práce je součástí rozsáhlejšího projektu, jehož hlavním účelem je práce s obrázky.

1 SROVNÁNÍ WEBOVÝCH FRAMEWORKŮ A POUŽITÉ NÁSTROJE

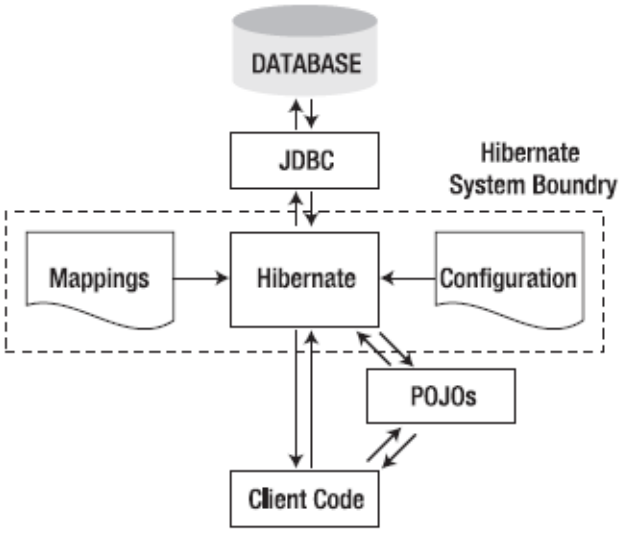
1.1 Webové frameworky

V tomto bodě jsou popsány klady a zápory vybraných frameworků. Nejsou zde ale uvedeny zdaleka všechny, neboť to není hlavním cílem této práce.

1.1.1 Hibernate

Hibernate je nástroj, který umožňuje objektově relační mapování (ORM) pro prostředí Java. Zkratkou ORM je myšlena technika mapování dat, které jsou reprezentovány databázovými tabulkami. Tento framework umožňuje také získávání dat z databáze na základě vlastního plně objektově orientovaného jazyka Hibernate Query Language (HQL). Tento jazyk je velice podobný jazyku SQL. Hibernate poskytuje jednoduché API, které umožňuje mapování objektů do 20 druhů relačních databází. Může být použit v mnoha aplikačních architekturách. Nejčastěji je tento framework použit jako perzistentní vrstva u dvouvrstvé a třívrstvé architektury. [2]

Na následujícím obrázku je zobrazeno, jakou roli hraje Hibernate v aplikaci napsané v Javě [5]:



Obr. 1.1: Postavení Hibernate v Javové aplikaci

1.1.2 Struts 1.x

Tento framework je již poměrně dlouhou dobu vyvinut. To znamená, že má velkou důvěryhodnost, a tím pádem je také často využíván. Z toho také vyplývá velké množství dokumentace, a to může být, v některých případech, rozhodující. Nemluvě o podpoře knihoven, kterých je velké množství. Dále je velkou výhodou zpětná kompatibilita, takže je možné bez větších komplikací přejít na vyšší verzi. [3]

Jde vlastně o ovládací vrstvu, založenou na standardních technologiích jako jsou JavaServlets, JavaBeans, ResourceBundles a XML. Tento framework pomáhá vytvářet rozšiřitelné vývojové prostředí pro určenou aplikaci, založené na volně šiřitelných standardech a osvědčených návrhových vzorech. Struts implementuje svůj vlastní web Controller a také obsahuje další technologie pro práci s vrstvami Model a View. Pro vrstvu Model jsou to standardní technologie pro přístup a práci s daty např.: JDBC a EJB, stejně jako balíčky třetích stran Hibernate, iBATIS a Objektově relační most. Ve vrstvě View pracuje tento framework s JSP stránkami, včetně JSF a JSTL, stejně jako s Velocity Templates, XSLT a dalšími. Web Controller tvoří propojení mezi vrstvou Model a View. Ve Struts je také implementována třída ActionForm, která usnadňuje přenos dat mezi těmito dvěma vrstvami. [12]

1.1.3 Struts 2.x

Další verze tohoto frameworku sebou přináší mnoho výhod a vylepšení. Dle [3] je největším pokrokem oproti předešlé verzi jednoduchá architektura a snadná rozšiřitelnost. Je možné snadno přejít z předešlé verze Struts 1.x. Oproti svému předchůdci však nemá tolik dostupné dokumentace.

1.1.4 JSF

Výhodou uvedeného frameworku je, že má otevřenou architekturu, a tím pádem lze téměř vše doladit podle svých představ. Dále má velké množství komponent a nástrojů. Nevýhodou může být to, že ho nelze použít samostatně a při spojení s jinými frameworky se mohou vyskytnout problémy. [3]

1.1.5 GWT

U tohoto frameworku lze snadno vytvořit i RIA aplikaci, neboť uživatel napíše v jazyce Java zdrojový kód a framework samostatně vygeneruje JavaScript. Tím pádem lze podobnou aplikaci vytvořit i bez znalostí JavaScriptu. [3]

1.1.6 Seam

JBoss Seam je framework pro Java EE 5.0. Z důvodu, že poskytuje důsledné a zároveň snadno pochopitelné modely pro veškeré komponenty ve webových aplikacích se z něj stal jeden z nejvíce používaných frameworků vůbec. Vzhledem k tomu, že je hojně využíván v mnoha firmách a institucích, tak se neustále vyvíjí ke stále dokonalejšímu nástroji pro zjednodušení návrhu a realizaci webových aplikací. Dále Seam integruje knihovny a doporučuje postupy, knihovny a také generuje kostru aplikace. [10]

1.2 Použité nástroje pro realizaci

Z předchozí kapitoly popsaných frameworků jsou v této práci použity Hibernate a Struts 1.x. V této části jsou uvedeny další nástroje, které jsou zde také využity.

1.2.1 Webový server Apache Tomcat

O serveru
Jde o relativně jednoduchý Servlet/JSP Container. Z tohoto důvodu, že jde o referenční Servlet/JSP Container, je možné provozovat JSP stránky jak na tomto serveru, tak i na všech ostatních J2EE certifikovaných serverech. Sám o sobě obsahuje ještě mnoho dalších nástrojů, které usnadňují vývoj a nasazení webových aplikací. Mezi tyto nástroje patří také Java Servlet a JavaServer Pages. Tyto nástroje jsou postupně také vyvíjeny, a čím vyšší verze serveru je, tím vyšší je i verze těchto technologií. Jaká verze serveru obsahuje jaké verze těchto nástrojů je uvedeno na stránkách výrobce. [4]

Konfigurace
Tento server se nijak neodlišuje od ostatních. Tím pádem ho nalezneme také na webové adrese http://localhost:8080/, případně http://127.0.0.1:8080/.

Obsah důležitých adresářů serveru dle [4]:

- /bin/ - spouštěcí a vypínací soubory
- /common/lib/ - zde jsou *.jar knihovny viditelné jak webovým aplikacím, tak internímu kódu.
- /conf/ - umístění konfiguračních souborů serveru. Nejdůležitějším souborem je server.xml, který obsahuje nastavení všech charakteristik - od nastavení portu, přes nastavení SSL až po nastavení kontejneru atd.

- /logs/ - logovací soubory
- /shared/lib/ - obsahuje *.jar knihovny, které jsou viditelné pouze všem webovým aplikacím, ale internímu kódu viditelné nejsou
- /webapps/ - domovský adresář všech nalezených webových aplikací
- /webapps/docs/ - kompletní dokumentace serveru

1.2.2 Java

Java je univerzální, objektově orientovaný, programovací jazyk. Základ, z kterého Java vznikla, je programovací jazyk C++. Došlo zde však k výraznému usnadnění práce a odstranění chyb typických pro C++. Další z hlavních předností Javy je přenositelnost programů mezi jednotlivými počítačovými platformami, a také, že po přeložení může zdrojový kód běžet v libovolném prohlížeči nezávisle na platformě nebo hardwaru. [6]

Základní části javové platformy jsou:

- Java Virtual Machine (JVM), neboli javový virtuální počítač, zajišťující běh přeložených aplikací
- vývojové nástroje
- implementace tříd tzv. Java Core API

1.2.3 MySQL

MySQL je databázový systém s veřejným zdrojovým kódem, což znamená, že ho lze bezplatně využívat, ale také je možné tento kód upravovat. Jde o systém správy databází (DBMS - database management system) určený pro relační databáze (RDBMS - relational database management system). Relační databáze je sbírkou vzájemně provázaných dat uložených v podobě textu, čísel nebo také binárních souborů řízenou právě systémem správy DBMS.

Základem MySQL je server MySQL (spouští a udržuje databáze), klient MySQL (nabízí rozhraní pro správu serveru) a dále obsahuje ještě mnoho nástrojů, ale tyto zde uvedené jsou základem, a také nejdůležitější. Architektura databáze MySQL je odlišná od jiných. Vyhovuje náročným podmínkám např. webových aplikací, uchovává data, podporuje mnoho různých datových typů, má velký rozsah podporovaného hardwaru a mnoho dalšího. [9]

1.2.4 NetBeans IDE

NetBeans je vývojové prostředí pro jazyk Java, ale není omezeno pouze na Javu, nýbrž v něm lze použít téměř jakýkoliv programovací jazyk. Je to vlastně nástroj, ve kterém mohou uživatelé psát, překládat, ladit a distribuovat různé druhy aplikací. Dále lze rozšířit tvůrčí možnosti pomocí tzv. plug-ins, což jsou softwarové moduly, které však nepracují samostatně, ale musí být součástí nějakého již hotového vývojového prostředí, jakým je zde popisovaný NetBeans IDE. [13]

1.2.5 CASE Studio

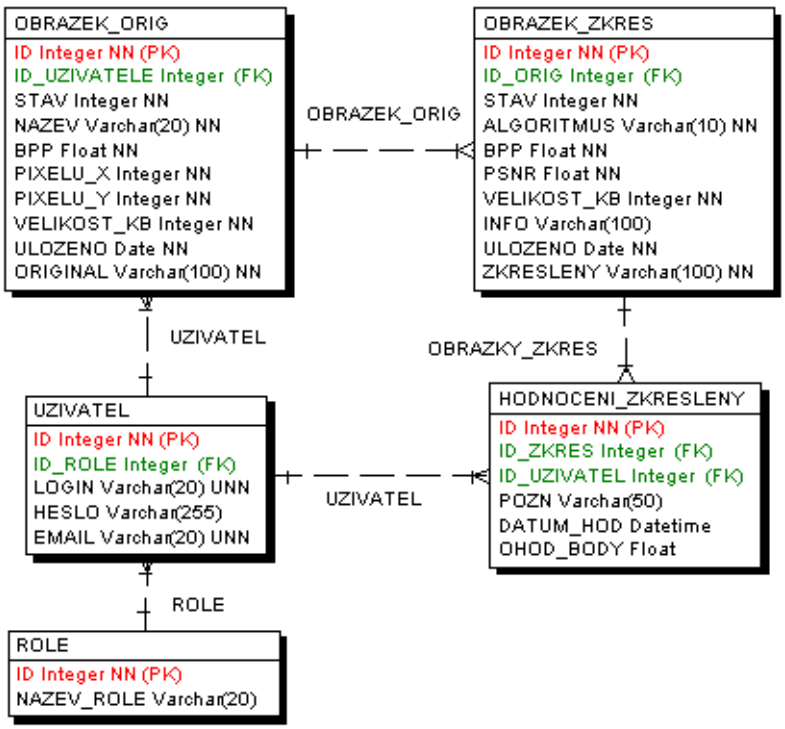
CASE Studio je program, určený k navrhování databázových struktur. Lze zde vymodelovat jak entitně relační diagramy (ERD), tak i data flow diagramy (DFD). Použití je možné pro mnoho různých databázových systémů, např.: Oracle, MSSQL, MSAccess, Firebird, DB2 a v neposldní řadě také MySQL, který je použit v této práci. A to zdaleka nejsou všechny typy, které je možné použít. Program umožňuje také převod mezi jednotlivými databázovými systémy. Další vlastností je kontrola a generování výsledného skriptu. Pomocí tohoto skriptu dojde k vytvoření všech výsledných tabulek a závislostí. Tento program skrývá řadu dalších vlastností, které zde nejsou uvedeny a je možné je využívat.

2 REALIZACE WEBOVÉ APLIKACE

2.1 Návrh databáze

Pro návrh databáze byl použit program CASE Studio. V tomto nástroji byla kompletně navržena celá tabulková struktura, došlo zde k vytvoření jednotlivých entit a jejich procesů, vazeb, ... Tyto entity reprezentují vlastně samotné tabulky. V tomto případě jsou následující: `ROLE`, `UZIVATEL`, `OBRAZEK_ORIG`, `OBRAZEK_ZKRES` a `HODNOCENI_ZKRESLENY`. Poslední z uvedených je tzv. vazební tabulka, neboť mezi entitou `UZIVATEL` a `OBRAZEK_ZKRES` je vazba typu M:N. To znamená, že každý uživatel může hodnotit více zkreslených obrázků, a také jeden obrázek může být ohodnocen více uživateli. Na závěr byl vygenerován skript, který se načetl programem MySQL Query Browser, jehož prostřednictvím došlo k samotnému vytvoření tabulek v databázi.

Na následujícím obrázku je zobrazen obsah jednotlivých tabulek.



Obr. 2.1: ERD diagram

Jak je zřejmé z obrázku 2.1, tak aplikace obsahuje tabulku `ROLE`, která umožňuje každému uživateli přiřadit určité pravomoci a omezení. V této práci jsou tyto role

následující: Administrátor a Uživatel. Práva jednotlivých rolí jsou popsána v dalších částech práce.

2.2 Struktura aplikace

V tomto oddílu práce je popsáno, jak ve skutečnosti vypadají jednotlivé součásti aplikace. Po prostudování této části bude jednoznačná struktura jak z hlediska počtu a názvů webových stránek, tak i jak dochází k přechodu mezi JSP stránkami pomocí `Action` tříd a validací. Ke zpracování zobrazených obrázků, v této sekci, byl použit jazyk UML, který poskytuje mnoho různých možností jak zobrazit strukturu aplikace. Patří mezi ně např.: Activity Diagram, Class Diagram, Collaboration Diagram, Component diagram, ... Více informací o tomto jazyce lze nelézt například zde [8].

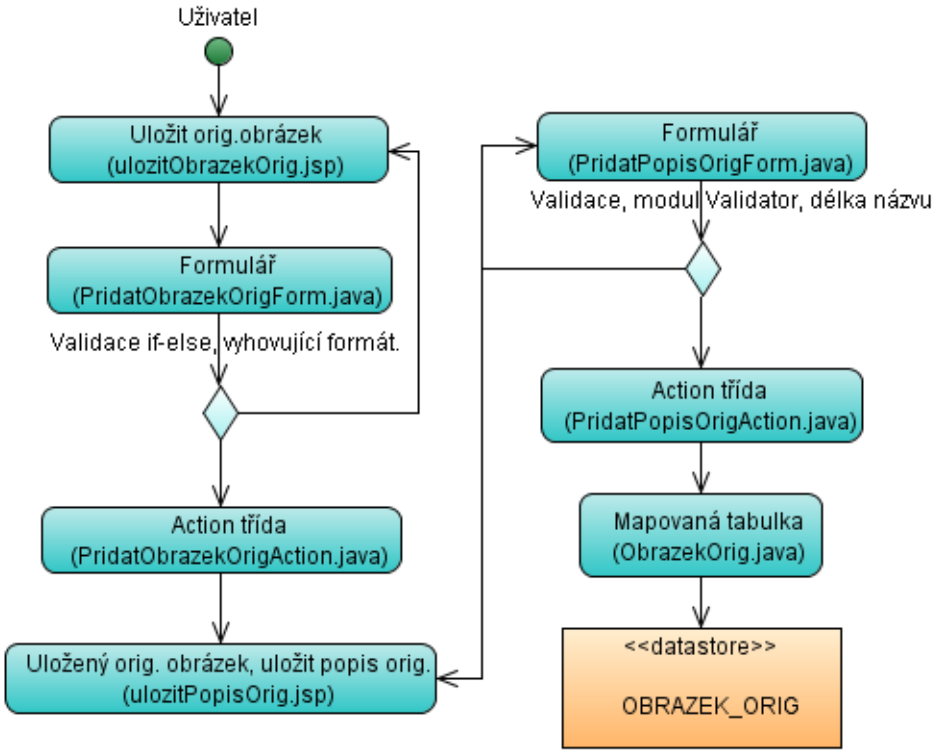
2.2.1 Activity Diagram

Tato část zachycuje možnosti pohybu v aplikaci. Z které stránky je možné přejít na kterou, pojmenování jednotlivých stránek, co se na dané stránce provádí. Pro lepší přehlednost je diagram vytvořen pro obě zmíněné role, jak pro Uživatele (příloha obr. A.1), tak pro Administrátora (příloha obr. A.2). Každý má totiž svá specifická práva pro přístup do jednotlivých sekcí.

2.2.2 Princip aplikace

Pro lepší pochopení jazyku Java je začleněn obrázek 2.2. Je na něm názorně ukázáno, jak probíhá práce s formuláři, třídami a JSP stránkami.

Příklad realizuje uložení originálního obrázku na server a parametrů k němu příslušících do databáze. Uživatel vstoupí na první stránku, na které dojde k výběru příslušného souboru k nahrání na server. Při stisknutí tlačítka pro postup na další stránku proběhne naplnění formuláře daty a následně jeho validace ve smyslu, zda vybraný obrázek má správný formát (*.jpg, *.png, ...). Tato validace se provádí uvnitř formuláře. V případě, že vybraný formát nevyhovuje, tak se vypíše chybové hlášení a aplikace se přesměruje zpět na původní stránku. Když však vše proběhne bez problémů, pokračuje se v dalších bodech, a to provedení `Action` třídy. Zde se provede načtení dat z formuláře, proběhne proces pro upload souboru na server a následně přesměrování na druhou stránku, kde dochází k zadání parametrů obrázku. Většina parametrů je již vypočítaná procesem v `Action` třídě, takže jedinný parametr vyžadovaný od uživatele je Název, tedy jak se daný soubor jmenuje. V dalších



Obr. 2.2: Ukládání originálního obrázku

bodech proběhne vše jak již bylo popsáno dříve, s tím rozdílem, že validace není prováděna ve formuláři, ale za pomoci modulu Validator, viz část 2.9. V závěru se uloží zadané a vypočítané údaje do tabulky `OBRAZEK_ORIG` v databázi prostřednictvím mapované třídy `ObrazekOrig.java`.

Struts pracuje s třívrstvým modelem aplikace `MCV` (`Model-Controller-View`). Jeho prostřednictvím lze docílit toho, že zdrojová data jsou oddělena od jejich zobrazení na stránce. To znamená, že lze například modifikovat vstupní data bez upravování výstupního zobrazení. Částí `Model` je reprezentován například obsah databáze, s kterým se bude v určitém projektu pracovat, `Controller` tvoří jednotlivé `Action` třídy, formuláře, a také soubor `struts-config.xml`, viz část 2.3. Jako poslední je zde vrstva `View` umožňující samotné zobrazení dat na stránce. Praktické použití je také zřejmé z obrázku 2.2. Zde do vrstvy `Model` náleží část `Mapovaná tabulka` spolu s datovým úložištěm `OBRAZEK_ORIG`. Vrstva `Controller` obsahuje jednotlivé formuláře a `Action` třídy. Poslední část `View` tvoří JSP stránky `ulozitObrazekOrig.jsp` a `ulozitPopisOrig.jsp`.

2.3 Soubor `struts-config.xml`

Každý projekt vytvářený pomocí frameworku Struts obsahuje, kromě tříd `Action` a formulářů `Form`, také soubor `struts-config.xml`. Tento soubor je jedním z nejdůležitějších v jakémkoliv projektu tohoto typu. Obsahuje párový tag `<form-bean>`, v kterém jsou uvedeny názvy a umístění jenotlivých formulářů použitých v projektu. Druhým důležitým tagem je `<action-mappings>`. Ten zajišťuje správné přiřazení jednotlivých tříd formulářům, dále obsahuje parametry, přes které je možné k nim přistupovat, a v neposlední řadě, když je to třeba, směrování na jinou stránku, při úspěšném proběhnutí třídy `Action`.

Příklad jednoho příkazu `<action-mappings>`:

```
<action input="/login.jsp" name="LoginForm" path="/login"
     scope="session" type="cz.struts.action.LoginAction">

     <forward name="success" path="/prihlasen.jsp"/>

</action>
```

2.4 Mapování tabulek na perzistentní objekty

Aby bylo možné s tabulkami uloženými v databázi pracovat je nutné namapování na perzistentní objekty. To je realizováno pomocí frameworku Hibernate. Tento framework umožňuje dva způsoby.

Prvním způsob je pomocí soboru *.hbm.xml. Ten obsahuje samotné mapování jednotlivých sloupců tabulky na objekty, a také jsou zde ošetřeny jednotlivé vazby: `One-To-Many`, `Many-To-One`, `One-To-One` a `Many-To-Many`. Dále je třeba vytvořit třídu *.java, ke které se bude tento soubor vztahovat. Tato třída obsahuje `get/set` metody pro každý sloupec tabulky. Oba tyto soubory, mapovací soubor *.hbm.xml a příslušná třída, reptezentují jednu tabulku umístěnou v databázi.

Druhým způsobem jsou anotace. Tento způsob je použit i v této práci. Jedná se o novější způsob, který má výrazně jednodušší zápis. Hlavním znakem je zde `@`, za kterým následuje příslušný text, např.: `@Id, @Entity,` .... Je zde zapotřebí pouze jedinná třída, která obsahuje `get/set` metody, anotace jednotlivých sloupců, a také

vzájemné vazby mezi tabulkami. Tato třída reprezentuje jednu tabulku v databázi. Následuje ukázka mapování pomocí anotací.

```
@Entity
@Table(name = "hodnoceni_zkresleny")
public class HodnoceniZkresleny implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    @Column(name = "ID", nullable = false)
    private Integer id;

    @Column(name = "POZN")
    private String pozn;

    @Column(name = "DATUM_HOD")
    @Temporal(TemporalType.TIMESTAMP)
    private Date datumHod;

    @Column(name = "OHOD_BODY")
    private Integer ohodBody;

    @JoinColumn(name = "ID_UZIVATEL", referencedColumnName = "ID")
    @ManyToOne(cascade=CascadeType.REFRESH)
    private Uzivatel idUzivatel;

    @JoinColumn(name = "ID_ZKRES", referencedColumnName = "ID")
    @ManyToOne
    private ObrazekZkres idZkres;

}
```

Předchozí příklad obsahuje část zdrojového kódu, který vystihuje, že anotace `@Entity` je základním prvkem mapování. Dále pomocí `@Table` se určí k jaké tabulce v databázi se daná třída vztahuje. Znakem `@Id` se označuje primární klíč a v tomto případě je to sloupec `ID`. `@GeneratedValue` způsobí to, že hodnoty tohoto sloupce budou generovány automaticky, a tím pádem není nutné se s těmito hodnotami zabývat. `@Column` je zde z důvodu, že jednotlivé sloupce v databázi mají jiné jméno, než které je používáno pro pozdější práci s nimi v aplikaci. Tedy pokud by byly

tyto názvy stejné, nebyly by tyto anotace třeba. `@JoinColumn` realizuje propojení s jinými tabulkami a určuje jejich vzájemnou vazbu.

Důležitým souborem je také `hibernate.cfg.xml` obsahující cestu k umístění mapovaných tříd v aplikaci. V tomto případě je to `cz.struts.hibernate.mapping.*`, kde hvězdička reprezentuje název třídy (tabulky v databázi). Dalšími parametry pro správnou funkčnost databáze jsou např. URL adresa a port, přístupové jméno a heslo, ovladač, dialect, ... Nastavení těchto parametrů se mění dle použitého druhu databáze. O dalších parametrech, jejich nastavení, významu a celé této problematice více v [5, 11].

2.5 Testování funkčnosti databáze

Kvůli tomu, aby se předešlo problémům v komunikaci s databází, je zde začleněna také část, kdy dojde k testování zda komunikace pracuje správně. Jelikož, když by se přistoupilo přímo k práci s databází přes formuláře, nebylo by možné, nebo velice obtížně, zjistit kde nastala chyba. Proto je zde vytvořena třída `Test.java`, v sekci `Test Packages`, balíčku `cz.struts.test`.

```
public class Test {
public static void main (String[] args) {

     Uzivatel u = new Uzivatel();
     Session session = null;

     try {
          System.out.println("Pred openSession");
          session = HibernateUtil.getSessionFactory().openSession();
          System.out.println("Po openSession");
          Transaction tr1 = session.beginTransaction();
          System.out.println("Po beginTransaction");
          u.setEmail("email");
          u.setLogin("login");
          u.setHeslo("heslo");
          session.save(u);
          tr1.commit();
     }
     finally {
          session.close();
```

```
        }

        System.out.println("Konec");
}
}
```

Na předchozím příkladu dojde k uložení zkušebních hodnot do databáze. V případě, když by se vyskytla nějaká chyba, tak jasně uvidíme jaký text byl vypsán, a který ne. Dle toho lze určit místo chyby.

2.6 Systém přihlašování

Přihlašování do systému probíhá přes grafické rozhraní. Jeho náhled se nachází v příloze obr. B.1.

Vše je realizováno ve třídě `LoginAction.java`. V prvním kroku se provede výběr všech uživatelských účtů z tabulky `UZIVATEL`, které mají uživatelské jmého shodné se zadaným. Výběr je prováděn následujícím příkazem:

```
uList = ses.createQuery("from Uzivatel where login='" +
        lForm.getUsername()+"'").list();
```

Výsledek toho výběru je uložen do objektu typu `List`. Když je tento `uList` prázdný (tzn. uživatelské jméno neexistuje) dojde k výpisu chybového hlášení „Uživatelské jméno neexistuje!“. V opačném případě se postoupí k ověřování správnosti zadaného hesla. Heslo je zabezpečené algoritmem SHA-256, jehož sytaxe se nachází ve třídě `HashPasswd.java`. To znamená, že heslo musí být nejdříve zakódované tímto algoritmem, a až poté porovnáno s tím, které je uložené v databázi. Když je heslo zadané také špatně, dojde opět k chybovému hlášení „Špatné heslo!“ a uživatel není přihlášen. V případě správně zadané kombinace uživatelského jména a hesla dojde k uložení potřebných údajů jako je `ID` role uživatele, `ID` uživatele a uživatelského jména do `HttpSession`. Tyto údaje jsou zde uloženy pro pozdější použití v aplikaci. Při odhlášení se `HttpSession` ukončí, a tím se z ní vymažou všechna data. Při delší nečinnosti uživatele ho systém automaticky odhlásí.

Toto ověřování identity uživatele je realizováno za pomoci podmínky `if-else`. Vypisování chybového hlášení má na starosti objekt typu `ActionErrors`. U něj

dojde k vytvoření nového objektu, dále pomocí příkazu `error.add("spatneHeslo", new ActionMessage("error.spatne.heslo"));` se vypíše chybové hlášení uložené pod zkratkou `error.spatne.heslo`. Definice tohoto hlášení je uvedena v souboru `ApplicationResource.properties` v balíku `cz.struts.messages`. Tento soubor obsahuje všechny zkratky (hlášení) používané v celé aplikaci.

2.6.1 Hlavní menu

Po úspěšném přihlášení je uživatel přesměrován na úvodní stránku aplikace `prihlasen.jsp`. Tato stránka je rozdílná pro každou roli příslušného uživatele. V případě, že se jedná o běžného uživatele, tzn. má cizí klíč `ID_ROLE` roven 2, tak má úvodní menu pestřejší. Mezi jeho hlavní pravomoci patří ukládání nových obrázků a mazání obrázků, které sám uložil do databáze. Dále možnost vzájemně porovnat originální obrázek spolu s více zkreslenými a samozřejmě také tyto zkreslené obrázky ohodnotit, dle tabulky uvedené na úvodní stránce. V případě, že se jedná o uživatele s cizím klíčem `ID_ROLE=1`, tak jde o roli Administrátor. Ten takové možnosti jako běžný uživatel nemá. Mezi jeho hlavní pravomoci, a čímž se nejvíce liší od běžného uživatele, patří možnost smazat jakékoliv obrázky bez ohledu na to kdo je uložil. Dále má možnost zobrazení a porovnání obrázků, stejně jako běžný uživatel, ale nemá možnost hodnotit.

To co bylo popsané v předchozím odstavci je docíleno jednoduchou podmínkou přímo na JSP stránce `prihlasen.jsp`. Otestováním pomocí JSTL tagu `<c:if test="$idRoleUzivatele == 2">` je zjištěno, zda je jedná o běžného uživatele nebo o administrátora. V případě, že jde o běžného uživatele, tak dojde k zobrazení odkazu do příslušné sekce aplikace. Naopak, když tato podmínka splněna není zůstanou možnosti v podmínce ukryté.

2.7 Ukládání obrázků

Toto je stručně popsané již dříve na obrázku 2.2. Celá tato část by se dala rozdělit na dvě části. Zpočátku se uloží pouze samotný obrázek na server prostřednictvím stránky `ulozitObrazekOrig.jsp`. Jeho nahrání probíhá za pomoci tagu `<html:file>`, `<html:submit>` a `<html:errors>`. První dva tagy musí být umístěny uvnitř dalšího tagu, a to `<html:form>`. Obsahem tohoto tagu musí být parametr `action="obrazekOrig"` a `enctype="multipart/form-data"`. První z uvedených parametrů je uveden, také v již zmíněném souboru `strus-config.xml`, který ho propojuje s jednotlivými třídami, ale zde je uveden pod parametrem `path`. Druhý zmíněný je zde z důvodu, že se jedná o upload souboru na server. Výsledkem je, že

se díky tagu `<html:file>` na stránce zobrazí přímo výběrové okénko, které pomocí tlačítka „Procházet“, umožňuje vybírat jednotlivé soubory v počítači. Je zde vyžadován jedinný parametr, a to `property = "nezkresleny"`, jehož hodnota reprezentuje proměnnou typu `FormFile`, která je k uploadu souboru nezbytná. Tato proměnná je uložena ve formuláři `PridatObrazekOrigForm.java` spolu s jejími `get/set` metodami. V případě, že se nevyskytnou žádné chyby, tak po výběru příslušného obrázku dojde k přechodu na následující stránku,`ulozitPopisOrig.jsp`, pomocí tlačítka „Další“. Toto tlačítko je svázáno s již zmíněným tagem `<html:submit>`, který potvrzující tlačítko zobrazí na stránce a po jeho stisknutí dojde k odeslání formuláře. Chybová hlášení jsou zobrazována pomocí tagu `<html:errors>`. Vše co bylo zmíněno by se neodehrálo, kdyby neexistovala třída `PridatObrazekOrigAction.java`, obsahující všechny důležité procesy, které jsou realizovány po stisknutí potvrzujícího tlačítka. Tato třída je umístěna v balíku `cz.struts.action.*` spolu s dalšími třídami a formuláři, plnící tyto funkce. V této třídě se nachází proces realizující upload obrázku na server. Příkazem

```
PridatObrazekOrigForm obOrigForm = (PridatObrazekOrigForm) form,
```

dojde k vytvoření objektu `obOrigForm` typu `ActionForm`. Přes tento objekt lze poté přistupovat ke `get/set` metodě proměnné `nezkresleny`, která, jak již bylo zmíněno, je typu `FormFile`. Vybraný obrázek se rozloží na pole bytů pomocí příkazu

```
byte orig[] = obOrigForm.getNezkresleny().getFileData(),
```

kde `byte orig[]` znamená, že pole je pojmenováno `orig` a je typu `byte`. Následuje vytvoření nového objektu typu `File obrazek`, pomocí kterého se vytvoří nové místo na serveru kam se obrázek uloží. Při tomto procesu se zároveň zjišťuje také cesta k umístění budoucího souboru. Jako výchozí je nastavena `\build\web\...`. V této práci je tato cesta upravena na následující: `\build\web\upload\original\...`. Příkaz pro samotné vytvoření místa pro obrázek je `obrazek.createNewFile()`. Dalším krokem je umístění získaného pole bytů do vytvořeného prostoru na serveru. K tomu je třeba vytvořit objekt typu `FileOutputStream fo` a pomocí příkazu `fo.write(orig)` dojde k naplnění vytvořeného prostoru daty z pole bytů.

Uložení obrázku na server však není jedinným úkolem této třídy. Dochází zde také k výpočtu parametrů obrázku, které budou ukládny do databáze. Toho se docílí

použitím objektu typu `BufferedImage`, který umožňuje práci s obrázky. Za jeho pomoci lze získat mnoho údajů, např.: rozměry obrázku v pixelech, počet bitů na pixel, jednotlivé hodnoty kanálů RGB, ... Zdaleka však v této práci není využito plného potenciálu tohoto objektu, protože jeho využití při práci s obrázky je obrovské. Je zde použit pouze pro získání počtu pixelů v ose X, Y (`getWidth(), getHeight()`) a ke zjištění kolik má obrázek bitů na pixel(`getColorModel().getPixelSize()`). Dalším zjišťovanou informací v této třídě je samotná velikost obrázku. To je realizováno za pomoci již dříve zmíněného objektu typu `File`, a je využita jeho vlastnost `length()`. Po zavolání této funkce získáme velikost obrázku v bytech, ale jelikož získaná hodnota by byla příliš vysoká, je převedena na kB vyděleným číslem 1024. Tato hodnota se již ukládá do databáze.

Na závěr jsou ještě uložené hodnoty uloženy do `HttpSession` (dále jen `HS`), ve které zůstanou uchovány pro pozdější použití. Uložení dat do ní probíhá ve dvou krocích. Nejdříve je třeba vytvořit objekt tohoto typu např.:

```
HttpSession session = request.getSession();.
```

Poté data do tohto objektu jen uložit pomocí příkazu

```
session.setAttribute("námi definovené jméno", hodnota).
```

A samozřejmě je zde zapotřebí nějaká návratová hodnota, a tou je příkaz

```
return mapping.findForward ("SUCCESS"),
```

kde `SUCCESS` je modifikovatelný parametr spojený s přesměrováním na další stránku. Tento parametr je uveden v souboru `struts-config.xml` jako `path`, ovšem tento je uzavřen mezi tagy `<action>`.

Nyní následuje přechod na stránku `ulozitPopisOrig.jsp`, na které dochází k zadání jednotlivých údajů náležících příslušnému obrázku. Tyto parametry je možné zjistit z obrázku 2.1. Většina údajů je na tuto stránku vypsána ze `HS`, do které jsme tyto hodnoty uložily ve třídě `PridatObrazekOrigAction.java`. To je ralizováno za pomoci tagu `<html:text>`, který musí být opět umístěn uprostřed tagu

<html:form>. Tag <html:text> má jeden povinný parametr, a to property, určující, ke které proměnné ve formuláři dané třídy se vztahuje. Další parametry jsou již volitelné. V tomto případě je zde použit jestě parametr readonly="true" zajišťující, že příslušné pole nebude modifikovatelné. Jak již bylo uvedeno, většina údajů bude vypsána do neměnných polí. Mezi tyto údaje patří BPP, PIXELU_X, PIXELU_Y, VELIKOST_KB a ORIGINAL. Poslední uvedený parametr uchovává název uloženého obrázku, kvůli pozdějšímu načtení. Tento údaj je uložen v dalším typu tagu, a tím je <html:hidden>, který má stejnou funkci jako <html:text> s tím rozdílem, že je pro běžného uživatele skrytý. Je viditelný pouze ve zdrojovém kódu. Předposledním parametrem je ULOZENO, který obsahuje datum uložení daného obrázku. K jeho naplnění dojde až přímo při ukládání do databáze. Jedinným parametrem závisejícím na uživateli je NAZEV, má omezení od 5 do 20 znaků. Tato kontrola (validace) je prováděna pomocí Struts Validatoru, viz část 2.9.

Třídou realizující uložení údajů o obrázku do databáze je PridatPopisOrigAction.java. Tato třída má k sobě také formulář PridatPopisOrigForm.java, ve kterém jsou uloženy jednotlivé get/set metody BPP, PIXELU_X, ... Proces, který proběhne v této třídě je obdobný s již uvedeným v části 2.3. Pro uložení dat do databáze je zapotřebí vytvořit proměnnou typu Session, Transaction, objekt typu ObrazekOrig a objekt typu ActionForm. Dále již dochází přímo k ukládání a to příkazem

```
popisOrig.setNazev(popisOrigForm.getNazev()),
```

kde popisOrig je objekt typu ObrazekOrig, popisOrigForm je objekt typu ActionForm. Toto se provede pro všechny položky, které chceme uložit. Vyjímkou je aktuální datum. Zde se to provádí následovně: popisOrig.setUlozeno(new Date()). Příkazem ses.save(popisOrig) dojde k samotnému uložení dat do databáze. Na závěr se uloží do HS ID originálního obrázku pro pozdější použití. Samozřejmostí je návratová hodnota return.

Bylo předvedeno uložení jak samotného obrázku na server, tak i jeho parametrů do databáze. Následovalo by popsání ukládání zkreslených obrázků příslušících k danému originálu, ale jelikož je postup obdobný s několika změnami, tak budou objasněny pouze důležité změny.

PSNR

PSNR je zkratka Peak Signal-to-Noise Ratio. Je jedním ze snadno zjistitelných

a často využívaných koeficientů učujících blízkost obrazu k originálu. Je definován jako:

$$\mathrm{PSNR} = 10 * \log_{10}\left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}}\right) \quad [\mathrm{dB}], \tag{2.1}$$

kde MSE je Mean Square Error, střední kvadratická chyba. MAX je maximální hodnota pro danou bitovou hloubku. Ze vztahu je patrné, že čím menší bude střední kvadratická chyba, tím větší bude hodnota PSNR.[7]

Právě PSNR je jednou změnou mezi ukládání originálního a zkresleného obrázku. Na jeho výpočet je vytvořena vlastní třída v sekci `cz.struts.util.PsnrUtil.java`, které zadáme pouze vstupní parametry `bi` a `bo`.

Třída `PsnrUtil.java`:

```
public class PsnrUtil {
    public static double PSNR(BufferedImage bi, BufferedImage bo){

        double PSNR=0.0;
        float in[]=null,out[]=null;

        in = bi.getData().getPixels(0, 0, bi.getWidth(),
        bi.getHeight(), in);

        out = bo.getData().getPixels(0, 0, bi.getWidth(),
        bi.getHeight(), out);

        for (int i = 0; i < in.length; i++) {
            PSNR += Math.pow(in[i]-out[i],2)/in.length;
        }
        return 10.0 * Math.log10(65025.0 / PSNR);
    }
}
```

Jak je zřejmé z předchozího zdrojového kódu, tak jsou zde opět využity objekty typu `BufferedImage`. Dojde k deklaraci dvou polí typu `double`, do kterých se ukládají hodnoty jednotlivých pixelů originálního a zkresleného obrázku. Následuje využití

již zmíněného typu `BufferedImage`, pomocí něhož dojde k vytvoření pole `in[]` a `out[]`. Cyklem `for` se poté vypočítá MSE. Parametr `return` vrátí, při zavolání této metody, již vypočítané PSNR.

Dalšími změnami oproti originálnímu obrázku jsou `BPP`, `ALGORITMUS` a `INFO`. `BPP` bylo již použito u ukládání originálu, ale zde má možnost zadat tuto hodnotu přímo uživatel, jako to bylo v předchozím případě u parametru `NAZEV`, neboť tato hodnota nemusí u zkreslených obrázků vždy odpovídat hodnotě vypočítané procesem. `ALGORITMUS` vypovídá o tom, jakým způsobem byl daný obrázek zkomprimován, zadání je také na uživateli a je zde omezení na 10 znaků. Poslední změnou je `INFO`, které je na stránce `ulozitPopisZkres.jsp` reprezentováno tagem `<html:textarea>`. Jedná se o obdobu tagu `<html:text>` s jedinným rozdílem, a to, že tomuto poli lze určit rozměry pomocí parametrů `rows` a `cols`. Zde může uživatel napsat případnou poznámku k obrázku.

2.8 Prohlížení a mazání obrázků

Prohlížení obrázků je umožněno přes odkaz v hlavním menu „Prohlížet uložené obrázky“. S tímto odkazem je svázána třída `ZobrazitSeznamOrigAction.java` spolu s jejím formulářem `ZobrazitSeznamOrigForm.java`. Po kliknutí na tento odkaz se provede výběr všech uložených originálních obrázků z databáze, a to z tabulky `OBRAZEK_ORIG`. Příkaz pro tento výběr je následující:

```
list = ses.createQuery("from ObrazekOrig").list();.
```

Jak je již jistě zřejmé z tohoto příkazu, tak dojde uložení výstupu do proměnné `list`, která je typu `List`. Tento samotný list je poté uložen do `HS` pod názvem `originaly`. Jak již bylo zmíněno dříve toto ukládání je realizováno z důvodu pozdějšího použití v aplikaci. Výstup tohoto výběru je vypsán na stránku `prohlizetOrig.jsp` do tabulky. Tato tabulka je vyřešena pomocí tagu `<display:table>`.

```
<display:table name="$originaly" id="original" pagesize="5"
               frame="1" defaultsort="1">
```

Tento tag je velice nápomocný nástroj, neboť bez jakéhokoliv programování umožňuje stránkování a řazení jak vzestupně, tak sestupně. Jedinné co je zapotřebí nastavit jsou vstupní data. Data jsou v tomto případě dodána pomocí proměnné

originaly, která byla v předchozím kroku naplněna listem originálních obrázků. Parametr name určuje právě zdroj těchto dat, id je parametr, přes který se bude pomocí tečkové notace přistupovat k datům uvnitř proměnné originaly, např.: original.bpp. Pagesize určuje počet záznamů, které budou zobrazeny na jedné stránce. V případě překročení tohoto počtu se další záznamy zařadí na další stráku. Frame vykreslí rám okolo tabulky o velikosti 1px. Defaultsort určuje, dle kterého sloupce budou záznamy seřazeny při příchodu na stránku. Jednotlivé sloupce tabulky jsou vykreslovány pomocí tagu <display:column>.

```
<display:column property="nazev" title="Název" sortable="true"/>
```

Parametr property přiřadí sloupci danou proměnnou, title je název sloupce v hlavičce tabulky a nakonec sortable vypovídá o tom, zda bude možné záznamy řadit dle daného sloupce. V posledním sloupci tabulky se nachází radiobutton, jehož vybráním zvolíme příslušný originální obrázek. Výběr probíhá dle ID, které je pro každý záznam jedinečné. Pro toto je využito tagu <html:radio>, který je uzavřen uvnitř <display:column> a všechny sloupce jsou umístěny v <display:table>.

```
<html:radio property="id" value="$original.id"/>
```

V případě, že nejsou uloženy žádné originální obrázky tak dojde k vypsání hlášení „Nejsou zde uloženy žádné obrázky.“ Toto je docíleno JSTL tagy, pomocí kterých je možné vkládat do JSP stránek např. cyklus, podmínky, ... Jsou zde použity tagy <c:choose>, <c:when> a <c:otherwise>. Poslední dva uvedené tagy jsou uvnitř prvního. Vše je realizováno podmínkou <c:when test="$not empty originaly">, která zjistí, zda jsou v proměnné originaly nějaké záznamy. Když je tato podmínka splněna tak dojde k zobrazení tabulky společně s daty. V opačném případě bude provedena druhá část cyklu, a to <c:otherwise>, který uzavírá text „Nejsou zde uloženy žádné obrázky.“ Když zde jsou nějaká data k zobrazení, tak jak již bylo zmíněno, dojde k jejich vypsání do tabulky spolu s dvěmi tlačítky Zobrazit/Ohodnotit a Smazat umístěnými pod tabulkou. Obě tyto tlačítka jsou zobrazena pouze když je přihlášený uživatel Administrátor, neboť běžný uživatel nemá právo mazat obrázky jiným uživatelům. Tím pádem zde opět dochází k testování pomocí <c:if>.

Přechod na další stránku je realizován dle toho, které tlačítko bylo stisknuto, viz příloha obr. A.1. Tato akce je provedena třídou PorovnaniOrigZkresAction.java.

Ke zjištění jaké tlačíko bylo stisknuto zde není použita klasická třída dědící z `Action`, nýbrž Struts nabízí třídu dědící z `LookupDispatchAction`.

`LookupDispatchAction` umožňuje použití dvou tlačítek s různou funkčností v jednom formuláři, a to bez použití JavaScriptu. Využívá toho, že při stisknutí tlačítka `<html:submit>`, obsahující parametr `property`, dojde k vypsání textu zobrazeného na tlačítku, jako parametru, do URL adresy v prohlížeči. Podle toho dojde ke zjištění, které tlačítko bylo stisknuto [1].

Konkrétně je to v této aplikaci takto. Na stránce `prohlizetOrig.jsp` jsou umístěna, již zmíněná, dvě tlačítka. Tyto tlačítka mají stejný parametr `property`.

```
<html:submit property="mazat/zobrazit">
    <bean:message key="submit.zobrazit"/>
</html:submit>
```

Dále je v souboru `struts-config.xml` přídán další parametr do `<action>`, a to `parameter="mazat/zobrazit"`. Tento parametr zajišťuje, že je vytvořeno provázání s oběma tlačítky. Je třeba také doplnit `<forward name="zobrazenePorovnani" path="/zobrazeniPorovnaniOrigZkres.jsp"/>` a `<forward name="zobrazeneHodnoceni" path="/zobrazitHodnoceni.jsp"/>`. Tímto je zajištěno, že při stisknutí různých tlačítek bude aplikace směrována na jinou stránku spolu s jiným procesem, který proběhne. Přímo ve třídě `PorovnaniOrigZkresAction.java` je tato problematika vyřešena následovně.

```
protected Map getKeyMethodMap() {
    Map m = new HashMap();

    m.put("submit.zobrazit", "zobrazit");
    m.put("submit.smazat", "smazat");

    return m;
}
```

Tímto procesem je docíleno „rozdělení“ této třídy na dvě části. Jednotlivé části jsou svázány s tlačítky s odpovídajícím popisem. Není zde klasická metoda `ActionForward execute`, jako je tomu v případě kdy je na stránce pouze jedno tlačítko, ale je zde buď `ActionForward zobrazit` nebo `ActionForward smazat`.

Část `zobrazit`, jak již název napovídá, slouží k zobrazení zkreslených obrázků příslušících originálu. Na začátku procesu je změna ve zdrojovém kódu. V prvním kroku dojde k získání `ID` originálního obrázku příkazem `int vybrId = porOrigZkresForm.getId();`. Poté dojde k výběru originálního obrázku z databáze, který má získané `ID` a jeho uložení do proměnné typu `List`.

```
list = ses.createQuery("from ObrazekOrig where id=" + vybrId).list();
```

Následujícím krokem je získání kolekce zkreslených obrázků příslušících danému originálu. Jedná se o proměnnou `collZkres` typu `Collection`. Toto je jedna z mnoha vlastností frameworku Hibernate. Velice jednoduše lze získat všechny obrázky, které mají cizí klíč (`ID_ORIG`) stejný jako je `ID` originálu.

```
collZkres = list.get(0).getObrazekZkresCollection();
```

Po tomto výběru následuje uložení této kolekce, vybraného originálu a cesty k umístění originálu na serveru do `HS`. Po provedení těchto akcí dojde k přesměrování na další stránku `porovnaniOrigZkres.jsp` a zobrazení vybraných hodnot.

Druhá část této třídy, `smazat`, realizuje vymazání originálního obrázku přímo ze serveru, jeho parametrů z databáze, a také všech přidružených hodnocení, obrázků a jejich parametrů. Tímto se zamezuje vzniku tzv. „sirotků“, jež by mohly zůstat v databázi při neprovázání s originálem. Tohoto je docíleno díky použití databáze typu `InnoDB` a cizích klíčů. Ještě je třeba doplnit příkaz `cascade=CascadeType.ALL` do mapované třídy `ObrazekOrig.java` do závorek části `@OneToMany`. Příkaz s koncovkou `ALL` umožní veškerou možnou manipulaci s přidruženými zkreslenými obrázky. V [5] jsou popsány další možnosti jako například: `MERGE (UPDATE), PERSIST (INSERT), REFRESH (SELECT) A REMOVE (DELETE)`. Zpočátku dojde k získání listu originálních obrázků, které byly vybrány. Dalším příkazem je smazání vybraného originálního obrázku z databáze. To je provedeno následujícím příkazem.

```
ses.createQuery("delete from ObrazekOrig where id=" + vybrId).
                executeUpdate();
```

Tímto však bude smazán pouze záznam v databázi, ale nikoliv soubor samotný na serveru. K tomu se využívá proměnné typu `File`, která je použita již v části 2.7. Zde je také zjištěna cesta k souboru na serveru pomocí `ID`, poté je pomocí příkazu `obrO.delete();`, smazaný obrázek uložený na dané cestě. Následuje smazání kolekce zkreslených obrázků jak z databáze, tak ze serveru. Do proměnné `list2` typu `List` jsou uloženy všechny hodnoty z kolekce. Následuje zjištění velikosti obsahu toho listu, příkazem `int pocetObr = list2.size();`. Dále jsou pomocí cyklu `for` smazány všechny obrázky na serveru.

```
for (int i=0; i<pocetObr; i++){

     File obrZ = new File(getServlet().getServletContext().
                 getRealPath("/") + "/upload/zkresleny/" +
                 list2.get(i).getZkresleny());

     obrZ.delete();
}
```

V tomto cyklu dochází ke zjišťování cesty a mazání obrázků do té doby, dokud je `i` menší než `pocetObr`. Tímto je realizováno smazání všech zkreslených obrázků náležících danému originálu. Na závěr zde je použita klasická návratová hodnota (`return mapping.findForward(smazanyObr);`), která nasměruje výstup po stisknutí potvrzujícího tlačítka na další stránku. Z této stránky má uživatel možnost volby smazání dalšího obrázku nebo návrat do hlavního menu.

V případě, že je provedena první část třídy, `zobrazit`, tak to znamená, že uživatel postoupil k výběru zkreslených obrázků, které chce porovnat s originálem. Tento výběr je umožněn prostřednictvím stránky `porovnaniOrigZkres.jsp`, na které došlo k vypsání jak parametrů vybraného originálu, tak k zobrazení tabulky se zkreslenými obrázky. Na této stránce jsou využity další dva tagy JSTL, `<c:forEach>` a `<c:out>`. První zmíněný má za úkol postupné procházení např.: listu objektů a jejich vyobrazení do tabulky, jak je to použito na této stránce. Použití je obdobné jako v případě tagu `<display:table>`, pouze jsou zde jinak pojmenované vyžadované parametry, bez kterých by se zobrazení neobešlo. Zdroj dat tomuto tagu je dodán pomocí parametru `items="$vybraneZkres"` a přistupovat k jeho jednotlivým částem je umožněno přes parametr `var="vybranZkres"` pomocí tečkové notace. Je zde použita klasická tabulka, jíž odpovídá tag `<table>`. Sloupce a řádky reprezentují

tagy <td> a <tr>. V těchto řádcích tabulky je použit druhý ze zmíněných JSTL tagů, a to <c:out>. Jeho použití je podmíněno parametrem value, neboť tento parametr je vyžadován. Následuje příklad použití.

```
<td align="center"><c:out value="$vybranZkres.algoritmus"/></td>
```

Výběr zkreslených obrázků, které chceme porovnávat (hodnotit), probíhá přes tzv. „chechboxy“. Tyto checkboxy jsou vyobrazeny pomocí tagu <html:checkbox>, jehož jedinným povinným parametrem je property. Zde je použit parametr property="pole" a value="$vybranZkres.id". Druhý ze zmíněných určuje, že po zaškrtnutí daného chechboxu bude vybrán právě ten obrázek, který má dané ID. První ze zmíněných parametrů určuje to, že se vybrané hodnoty ID uloží do proměnné pole, která je definována jako pole hodnot Integer. Výběr je však omezen na 3 položky pro porovnávání a na 1 položku pro zobrazení hodnocení. Opět je zde využit systém dvou tlačítek v jednom formuláři, již zmíněný dříve v této části. Pomocí těchto tlačítek je možné buď přímo zobrazit samotný originál v porovnání se zkreslenými obrázky spolu s jejich parametry (Porovnávat) nebo zobrazit hodnocení jednoho zkresleného obrázku (Ohodnocení obrázku). Opět je zde třída ZobrazitPorovnaniOrigZkresAction.java rozdělena na dvě části, a to na část porovnani a hodnoceni. V případě stisknutí tlačítka Porovnávat, dojde k provedení části porovnani. V této části je postup následující. Nejdříve se získá proměnná pole z formuláře ZobrazitPorovnaniOrigZkresForm.java. V tomto poli jsou uloženy všechna vybraná ID zkreslených obrázků, které byly vybrány na stránce porovnaniOrigZkres.jsp. Následuje podmínka typu if-else, která realizuje získání a uložení parametrů zkreslených obrázků do HS. Samozřejmě je ukládána také cesta, odkazující na umístění obrázku na serveru. Pomocí toho cyklu je také zajištěna validace ve smyslu, když by uživatel nevybral žádný nebo naopak příliš obrázků.

Příklad výběru parametrů a cesty v případě jednoho obrázku:

```
else if (poleId.length == 1){

    list0 = ses.createQuery("from ObrazekZkres where id="
            + poleId[0]).list();

    session.setAttribute("cesta_zkres0", "/upload/zkresleny/" +
                         list0.get(0).getZkresleny());
```

```
    session.setAttribute("vybranyZkres0", list0);

}
```

Na příkladu je realizován výběr z tabulky v databázi `ObrazekZkres`, ve kterém se `ID` shoduje s uloženým v `poleId`. Toto vše je uloženo do proměnné `list0`, která je typu `List`. V dalších krocích dojde k uložení cesty k obrázku na serveru a již zmíněného listu do `HS` pro další použití.

V druhém případě, při stisknutí tlačítka „Ohodnocení obrázku“, proběhne zdrojový kód uložený v druhé části třídy, a to část `hodnoceni`. Dojde zde opět k získání `poleId` obsahující `ID` zkreslených obrázků, ale v tomto případě zde bude uložena vždy pouze jedno hodnota. Znovu je tu použita také podmínka `if-else`. Pomocí této podmínky je zde ošetřeno, že je možné vybrat pouze jeden zkreslený obrázek. Když je toto splněno, provede se obdobný příkaz jaký je uveden v předchozím příkladu. Úprava zdrojového kódu spočívá pouze v uložení kolekce hodnocení uživateli `HodnoceniZkreslenyCollection()`, v které jsou uloženy všechna hodnocení všech uživatelů daného obrázku. A samozřejmě také uložení této kolekce `collHod0` do `HS`.

Zobrazení výstupu stisknutí těchto tlačítek je realizováno stránkami `zobrazeniPorovnaniOrigZkres.jsp` v prvním případě, a ve druhém je to `zobrazitHodnoceni.jsp`. Na stránce `zobrazeniPorovnaniOrigZkres.jsp` dojde k již finálnímu zobrazení originálního a několika (maximálně však tří) zkreslených obrázků spolu s jejich parametry. Jednotlivé obrázky jsou řazeny vedle sebe a pod sebou mají uvedeny parametry. Tohoto je docíleno za pomoci tabulky `<table>`. Jedná se o tabulku, která má pouze jeden řádek a čtyři sloupce. Tímto je velice jednoduše docíleno organizovaného zobrazování obrázků. O zobrazení parametrů se stará již dříve zmiňovaná kombinace tagů `<c:forEach>` a `<c:out>`. Novým tagem je zde `<html:img page="$cestaOrig"/>`. Tento tag slouží k vykreslení uloženého obrázku na stránce, kde parametr `page` určuje relativní cestu k umístění obrázku na serveru. U originálního obrázku jsou zobrazeny pouze jeho parametry a samotný obrázek. V případě zkreslených je zde několik položek navíc. Uvnitř tagu `<c:forEach>` je mimo běžných tagů `<c:out>` umístěn také `<html:hidden>`. Hodnota tohoto tagu je

```
<html:hidden property="id_vybr" value="$vybrany.id"/>.
```

Parametr `property` určuje, že v této proměnné ve formuláři bude uloženo `ID` obrázku, který byl ohodnocen. Toto je důležité, neboť tato hodnota je potřebná pro uložení do databáze. Dále je zde podmínka `<c:if>`, zjišťující identitu uživatele. V případě, že jde o administrátora, tak položky hodnocení, které budou dále popsány, mu na stránce zobrazeny nejsou. Naopak, když se jedná o běžného uživatele, položky mu zobrazeny budou. Patří mezi ně možnost bodového ohodnocení a poznámky k hodnocení obrázku. K bodovému ohodnocení slouží tag `<html:select>`, který po kliknutí umožňuje zobrazení výběru možných hodnot. Parametrem tohoto tagu je `property="ohodBody"`, který se vztahuje k položce v databázi s bodovým ohodnocením. Jedná se o párový tag a mezi položky uzavřené v tomto tagu patří další párový tag, a to `<option>`. Následuje příklad použití tohoto tagu:

```
<option value="1">1 - Velice špatný</option>.
```

Jeho parametr `value` určuje hodnotu, která bude uložena do databáze jako ohodnocení. Mezi další položky, které je možné zvolit patří:

- 2 - Špatný
- 3 - Průměrný
- 4 - Dobrý
- 5 - Výborný

Mezi těmito hodnotami lze volit také desetinné hodnoty (po 0,2), kvůli přesnějšímu vyjádření zkreslení.

Poslední hodnotou, kterou je možné vyplit a následně uložit do databáze, je poznámka k ohodnocení. Ta je reprezentována tagem `<html:textarea>`, již využívaný v závěru části 2.7. A samozřejmě je zde zapotřebí tag `<html:submit>`, po jehož stisknutí dojde k uložení dat do tabulky `HODNOCENI_ZKRESLENY` v databázi.Všechny tyto tagy náleží pouze jedinnému zkreslenému obrázku a proto musí být uzavřeny v tagu `<html:form>`. Jelikož na stránce se mohou nacházet až 3 obrázky, je na stránce tedy umístěno vše 3x. Výsledný náhled na porovnání se nachází v příloze obr. C.1.

Třídou, která zrealizuje vlastní uložení do tabulky v databázi je `PridatHodnoceniZkreslenyAction.java`. V této třídě uložení probíhá klasicky jak

již bylo zmíněno např. při ukládnání parametrů originálního obrázku v části 2.7, proto není nutné tuto problematiku opět rozebírat.

Nyní bude popsáno zobrazení druhého případu, a to stránky `zobrazitHodnoceni.jsp`. Úkolem této stránky je zobrazit uložené obrázky spolu s jejich ohodnocením všemi uživateli. Opět se jedná o zobrazení za pomoci tagů `<c:forEach>` a `<c:out>`. Na levém okraji pod navigačními tlačítky se zobrazí neměnné hodnoty vybraného zkresleného obrázku Algoritmus, BPP, PSNR, Velikost v kB a Info spolu s názvem originálu. V těle stránky se zobrazí vybraný zkreslený obrázek a pod ním měnící se parametry Bodové ohodnocení, Datum hodnocení, Ohodnotil a Poznámka.

Poslední částí je možnost uživatele smazat obrázky, které jsou jeho. K tomu slouží v hlavním menu odkaz „Smazat své obrázky“. Opět jde o klasické načtení dat z databáze z tabulky `OBRAZEK_ORIG`, ale s tím rozdílem, že je zde nastaveno omezení. Toto omezení je realizováno pomocí následujícího příkazu

```
list = ses.createQuery("from ObrazekOrig WHERE idUzivatele=" +
       session.getAttribute("idUzivatele")).list();,
```

který vybere z databáze pouze obrázky obsahující v tabulce ve sloupci `ID_UZIVATELE` číslo opovídající `ID` uživatele uloženému v `HS`. Toto `ID` se uložilo do `HS` při přihlášení. Výsledek tohoto procesu je uložen do proměnné typu `List`, odkud se poté zobrazí na stránce `zobrazeniOriginaluUzivatele.jsp`. Na této stránce jsou jednotlivé hodnoty obrázků zobrazeny do tabulky a pomocí radiobuttonů je možné je označit a následně smazat. Postup smazání byl již uveden dříve, proto není nutné ho opět uvádět.

Po celou dobu pohybu v aplikaci je zobrazené uživatelské jméno přihlášeného uživatele. Je umístěno v horní části obrazovky. Toto je možné díky `HS`, která umožňuje přístup k datům v ní uloženým po celou dobu její životnosti a je viditelná (přístupná) z kteréhokoli části aplikace.

2.9 Modul Validator

Modul Validator je součástí frameworku Struts. Jeho funkcí, jak již název jistě napovídá, je validace neboli kontrola zda uživatel aplikace zadal správně údaje. Není

zde nijak nutné složitě definovat různá pravidla uvnitř tříd či formulářů. Jedinné co je třeba ve formuláři změnit je třídu `Action` na `ValidatorAction`. Pro jeho používání je také nutno připojit do projetku knihovnu `commons-validator.jar`. Jedná se plug-in, který se do aplikace připojí přidáním následujícího kódu do souboru `struts-config.xml`:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
     <set-property property="pathnames"
     value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

Jak je zřejmé i z předchozího zdrojového kódu, hlavní dva soubory, ve kterých se definuje validace jsou `validator-rules.xml` a `validation.xml`. První zmíněný soubor obsahuje předdefinované metody validace, které je možné libovolně rozšířit o své vlastní. Avšak není to nezbytně nutné, neboť obsahuje dostatečný počet možností. Druhý z uvedených souborů je stěžejní pro definici validace a dochází zde již k nastavení co a jak se bude kdy ověřovat.

```
<form name="PridatPopisOrigForm">
   <field property="nazev" depends="required, minlength">

      <arg0 key="nazev.orig"/>
      <arg1 name="minlength" key="$var:minlength" resource="false"/>
      <var>
         <var-name>minlength</var-name>
         <var-value>5</var-value>
      </var>

   </field>
</form>
```

Předchozí příklad vystihuje validaci vstupního parametru při ukládání názvu originálního obrázku do databáze. Tag `<form>` s jeho parametrem `name="PridatPopisOrigForm"` určuje k jakému formuláři se má daná kontrola vztahovat. Následuje tag `<field>` určující jaké proměnné uvedené v parametru `property="nazev"` se bude tato kontrola týkat. Parametr `depends="required,`

minlength" zajišťuje to, že proměnná bude vyžadována a musí mít minimální délku. Tagy <arg0> a <arg1> jsou hodnoty, které budou zobrazeny ve výpisu při nesplnění podmínek uvedených v depends. Za <arg0> to bude text, který je předdefinovaný v ApplicationResource.properties a za <arg1> bude vypsáno číslo 5. Také je možné definovat globální proměnné, které nebudou závislé pouze na jedinném formuláři, ale budou kontrolovat všechny formuláře v aplikaci. Ve výsledku bude, při nesplnění podmínek, hlášení vypadat například takto:

```
Název obrázku: musí být vyplněno!,
```

případně

```
Název obrázku: nesmí být kratší než 5 znaků.
```

Všechna tato hlášení lze upravit v souboru ApplicationResource.properties, dle vlastní potřeby.

2.10 Grafický vzhled

Grafický vzhled byl vytvořen pomocí kaskádových stylů CSS. Tento styl byl získán jako volně šiřitelný. Je zatížen licencí *Creative Commons Attribution 2.5 License*, což omezuje uživatele těchto stylů pouze v tom smyslu, že v jeho aplikaci musí být zpětný odkaz na stránky, ze kterých byl získán. Toto je v této práci splněno.

3 ZÁVĚR

Cílem této práce bylo vytvořit webovou aplikaci, kde by měl uživatel možnost posoudit různě zkreslené obrázky spolu s originálem. Zpočátku byly rozebrány teoretické vlastnosti jednotlivých nástrojů, zde využitých. Následovalo jejich praktické využití v podobě realizace této aplikace. Uživatel zde má možnost se přihlásit do systému přes grafické rozhraní, kde již poté pracuje po celou dobu. Uživateli je umožněno jak nahrávání souborů na server, tak i ukládání jejich parametrů do databáze. S těmito obrázky je poté možné pracovat, jako například zobrazit si originální obrázek a k němu náležící zkreslené obrázky. Tyto obrázky je možné následně ohodnotit na stupnici od 1 do 5. Další možností je smazání uživatelových obrázků, které sám uložil na server. Práva k tomu smazat jakýkoliv obrázek má samozřejmě pouze administrátor.

Je zde dobře patrná pomoc jakou poskytují frameworky Struts a Hibernate při vývoji aplikací. Mezi velkou výhodu Struts patří například modul Validator. Tento modul poskytuje jednoduché a přitom efektivní možnosti validace (kontroly) vstupů uživatele. Není tedy třeba nijak obtížně definovat nějaké cykly. Nepochybně patří mezi jeho výhody to, že je poměrně „starý“, a tím pádem se nevyskytují žádné problémy s kompatibilitou nebo případně s chybami. Další z uvedených frameworků, Hibernate, má nezpochybnitelnou výhodu v práci s databází, k čemuž je také určen. Pomocí anotací umožňuje například jedinným příkazem získat všechny zkreslené obrázky náležící jednomu originálu. Avšak toto není jeho jedinnou výhodou. Celkově je za pomoci tohoto frameworku značně urychlena a zjednodušena práce s databází.

Programovací jazyk Java je již sám o sobě velice užitečným nástrojem, ale spojením s dalšími frameworky se z něj stává velice mocný programovací prostředek. Menší nevýhodou by mohly být složitější začátky, neboť k plnému využití potenciálu těchto nástrojů je třeba umět alespoň základy jazyka Java.

LITERATURA

[1] DORAY, A. *Beginning Apache Struts : From Novice To Professional.* Steve A. 1st edition. [s.l.] : [s.n.], 2006. 536 s. ISBN 978-1-59059-604-3.

[2] HAMTIL, V. *Framework Hibernate* [online]. 2004-12-10 [cit. 2009-05-20]. Dostupný z WWW: <http://nb.vse.cz/zelenyj/it380/eseje/xhamv01/xhamv01.htm>.

[3] KRÁTKÝ, T. Web frameworks v praxi. *Profinit : professionals in IT* [online]. 2008 [cit. 2009-05-20].

[4] MĚRKA, D. *Apache Tomcat - konfigurace, srovnání s jinými servery* [online]. 2005-12-11 [cit. 2008-05-20]. Dostupný z WWW: <http://nb.vse.cz/zelenyj/it380/eseje/xmerd04/Tomcat.htm>.

[5] MINTER, D., LINWOOD, J. *Beginning Hibernate : From Novice To Professional.* Steve A. 3rd edition. [s.l.] : [s.n.], 2006. 360 s. ISBN 978-1-59059-693-7.

[6] PITNER, T. *Java - začínáme programovat, podrobný průvodce začínajícího uživatele.* [s.l.] : Grada Publishing, spol. s.r.o., 2002. 224 s. ISBN 80-247-0295-9.

[7] PRŮŠA, Z., MALÝ, J. Metoda SPIHT pro kompresi barevných obrazů a její implementace. *Elektrorevue* [online]. 2009 [cit. 2009-05-20]. Dostupný z WWW: <http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/0/metoda-spiht-pro-kompresi-barevnych-obrazu-a-jeji-implementace/>. ISSN 1213-1539.

[8] STEIN, R. *Návrh aplikací v jazyce UML* [online]. 2003-2005 [cit. 2009-05-20]. Dostupný z WWW: <http://interval.cz/serialy/navrh-aplikaci-v-jazyce-uml/>. ISSN 1212-865.

[9] ULLMAN, L. *PHP a MySQL, Názorný průvodce tvorbou dynamických WWW stránek.* Bodgan K. [s.l.] : Computer Press, a.s., 2004. 536 s. ISBN 80-251-0063-4.

[10] YUAN, M., HEUTE, T. *JBoss Seam: Simplicity and Power Beyond Java EE.* [s.l.] : [s.n.], 2007. 402 s. ISBN 0-13-134796-9.

[11] *Complete Hibernate 3.0 Tutorial* [online]. c2008 [cit. 2009-05-20]. Dostupný z WWW: <http://www.roseindia.net/hibernate/index.shtml>.

[12] *Struts 1 - Welcome* [online]. 2000-2008 [cit. 2008-12-15]. Dostupný z WWW: <http://struts.apache.org/1.x/>.

[13] *Vítejte u NetBeans* [online]. 2004 [cit. 2008-5-20]. Dostupný z WWW: <http://www.netbeans.org/index_cs.html>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

EJB Enterprise Java Beans

HS HttpSession

J2EE Java Enterprise Edition

JDBC Java Database Connectivity

JSF Java Server Facelets

JSP Java Servlet Pages

JSTL JavaServer Pages Standard Tag Library

RIA Rich Internet Application

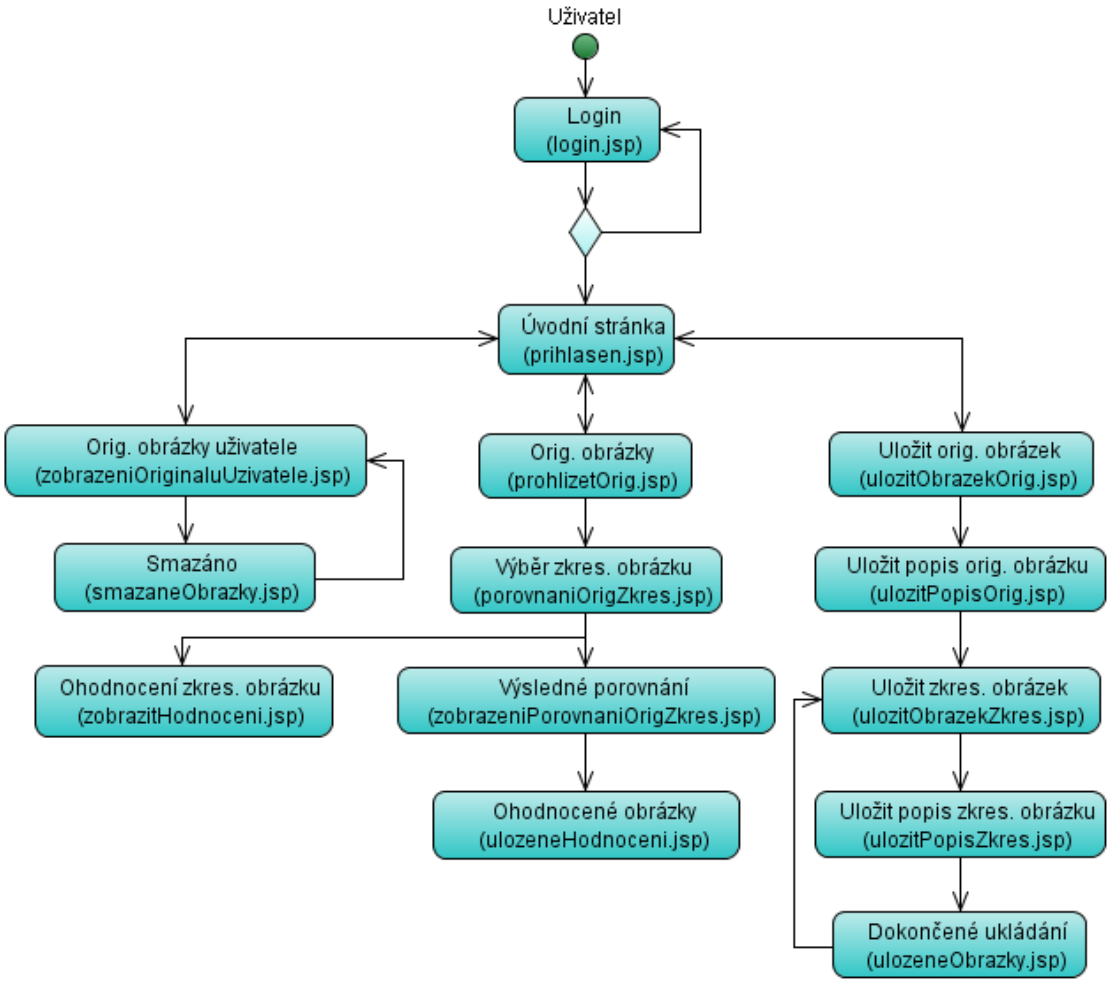
UML Unified Modeling Language

XML Extensible Markup Language

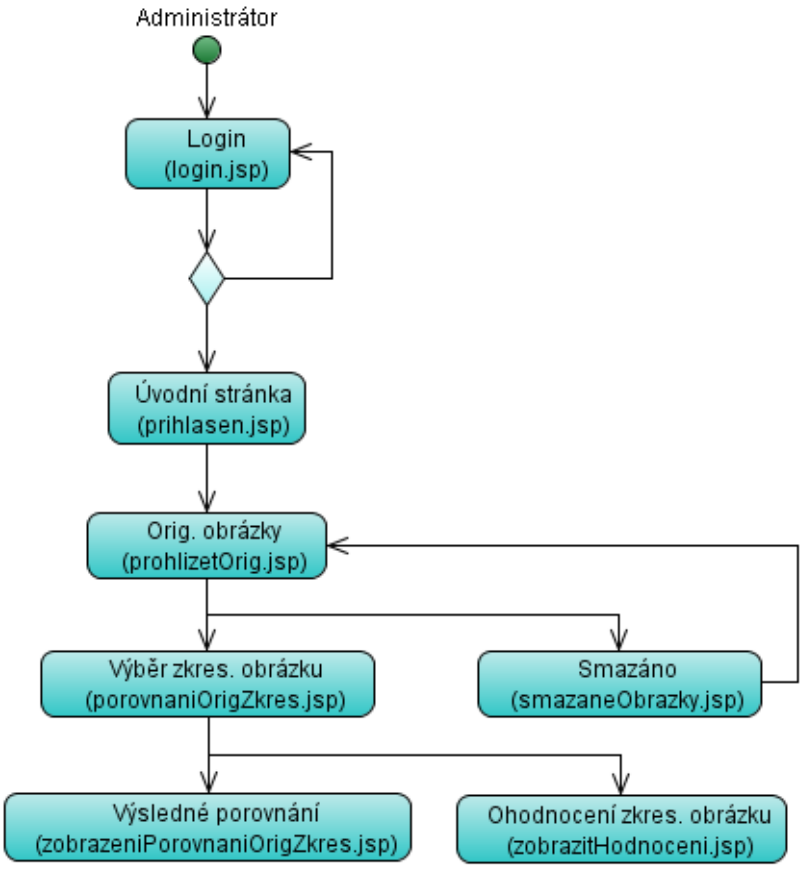
XSLT Extensible Stylesheet Language Transformations

SEZNAM PŘÍLOH

A ACTIVITY DIAGRAMS

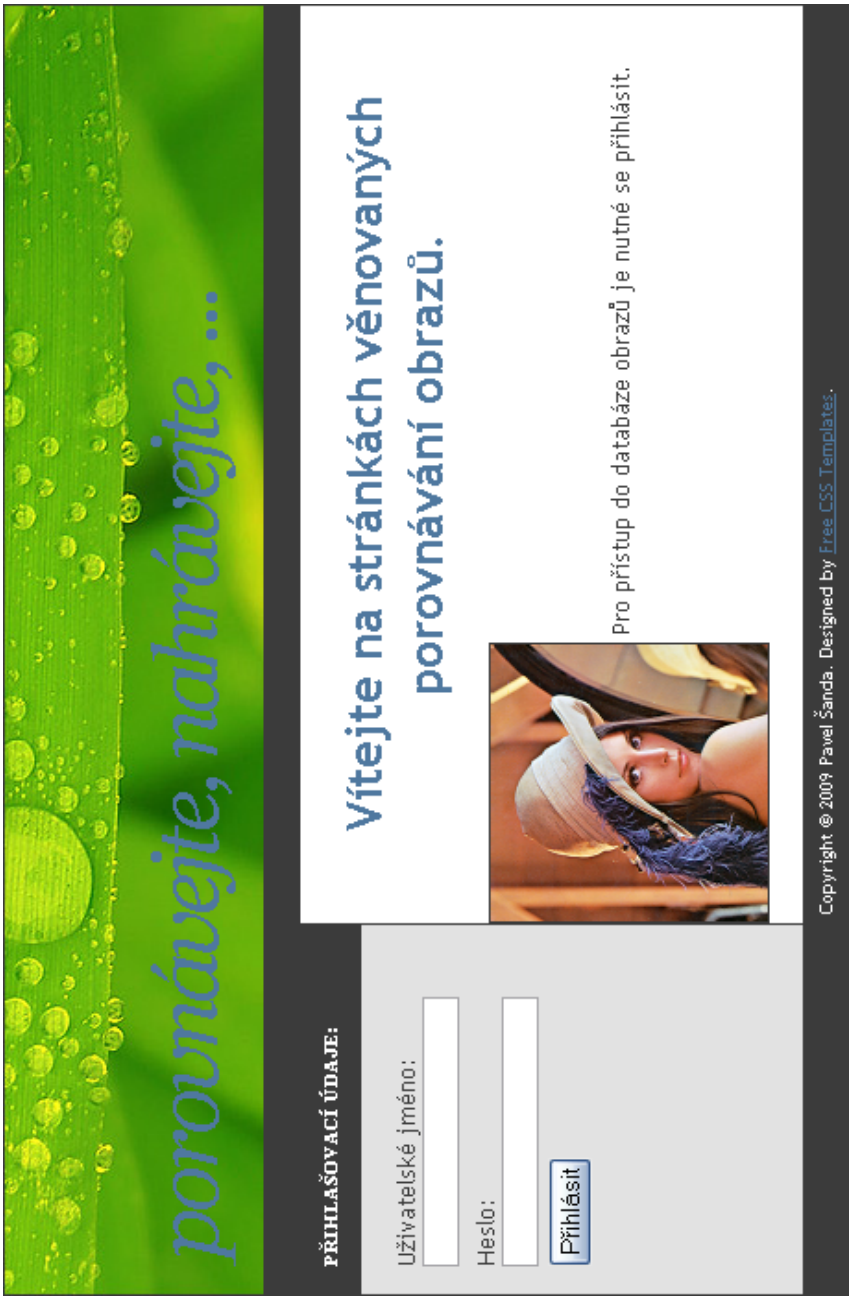


Obr. A.1: Activity diagram Uživatel



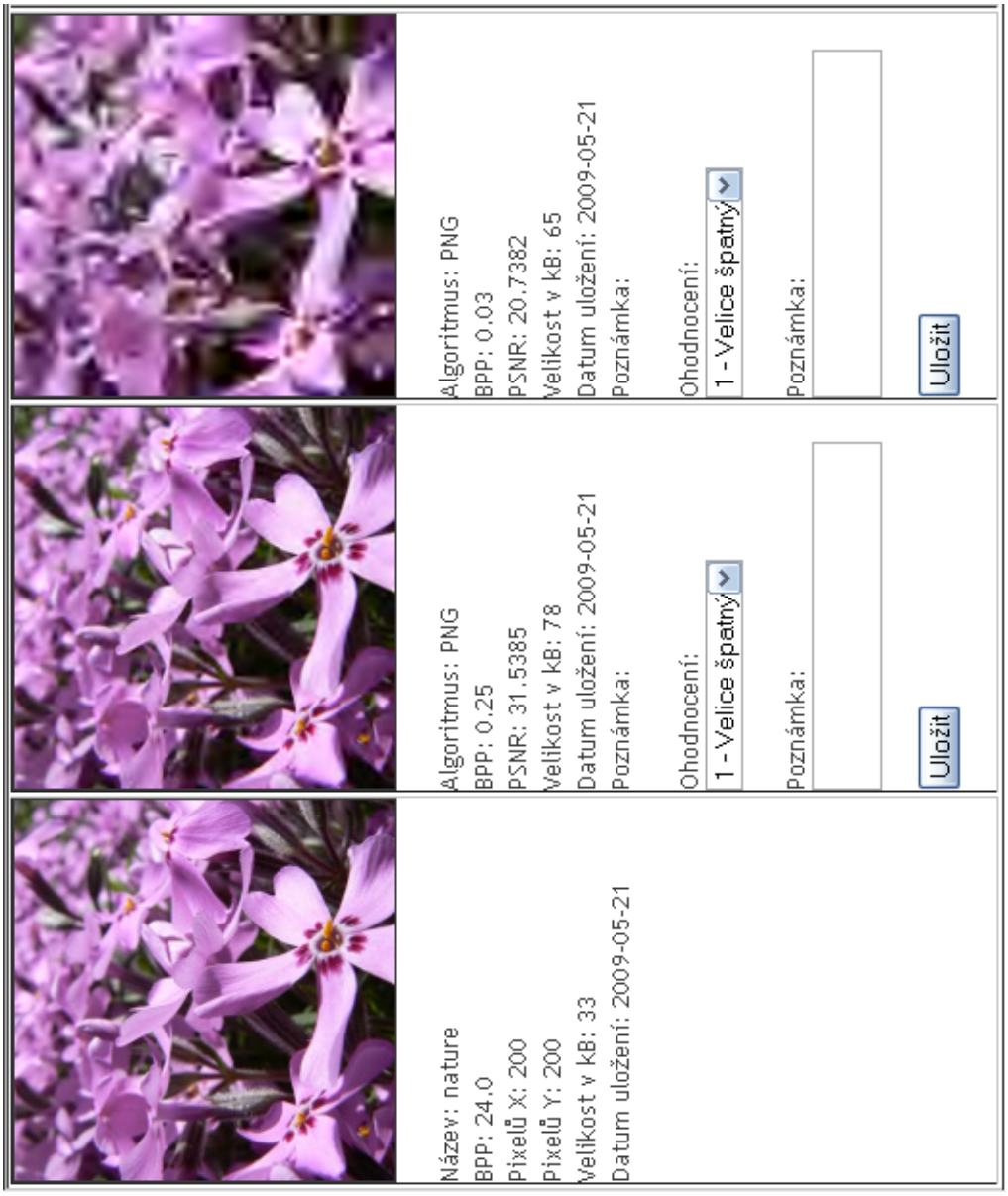
Obr. A.2: Activity diagram Administrátor

B PŘIHLAŠOVACÍ STRÁNKA



Obr. B.1: Náhled přihlašovací stránky

C VÝSLEDNÉ POROVNÁNÍ OBRÁZKŮ



Obr. C.1: Vzájemné porovnání obrázků a jejich parametrů