UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Online galerie: využití J2EE s JPA

Petr Bludský

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr BLUDSKÝ**
Osobní číslo:	**I08397**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Online galerie : využití J2EE s JPA**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je vytvořit webovou aplikaci galerie na platformě Java 2 Enterprise Edition za použití frameworku Struts 2 a Java Persistence Api společně s Oracle Toplink jako databázové vrstvy.
Práce bude obsahovat E-R model a jeho detailní rozbor. Použity budou taktéž návrhové vzory jako MVC (Model View Controller) nebo interceptor a knihovny pro práci s obrázky v javě (ořezávání, vytváření náhledů).
Frontendová část galerie bude optimalizována pro internetové vyhledávače (SEO) a bude využívat javascriptového frameworku MooTools. Jako aplikační server je zvolen Tomcat, jako databáze PostgreSQL.
Teoretická část práce se zaměří na klíčové vlastnosti frameworku Struts 2 a jeho využití ve webových aplikacích.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***Kathy Sierra, Bert Bates: Head First Java, O'Reilly Media 2005, počet stran 720, ISBN-10 1600330002**

***Kline Kevin, Kline Daniel, Hunt Brand: SQL in a Nutshell, O'Reilly Media 2008, počet stran 592, ISBN-10 0596518846**

***Ajit Sagar, Sue Spielman a kol.: Professional Java Server Programming J2Ee 1.4 Edition, Wrox Press 2003, počet stran 850, ISBN-10 1861008139**

***Vivek Chopra, Sing Li, Jeff Genender: Professional Apache Tomcat 6, WROX Press 2006, ISBN-10: 0471753610**

***Sun Microsystems: The Java EE5 Tutorial [online], 2007 [cit. 2009-10-08], dostupný z WWW: http://java.sun.com/javaee/5/docs/tutorial/doc/docinfo.html**

Vedoucí bakalářské práce: **RNDr. Iva Rulicová**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 20. 4. 2011 Petr Bludský

Poděkování

Rád bych poděkoval RNDr. Ivě Ruličové za její cenné rady a připomínky, které mi byly poskytnuty při vypracování této bakalářské práce.

Anotace

Cílem práce je vytvořit webovou aplikaci galerie na platformě Java 2 Enterprise Edition za použití frameworku Struts 2 a Java Persistence Api společně s Oracle Toplink jako databázové vrstvy. Práce bude obsahovat E-R model a jeho detailní rozbor. Použity budou taktéž návrhové vzory jako MVC (Model View Controller) nebo interceptor a knihovny pro práci s obrázky v javě (ořezávání, vytváření náhledů). Frontendová část galerie bude optimalizována pro internetové vyhledávače (SEO) a bude využívat javascriptového frameworku MooTools. Jako aplikační server je zvolen Tomcat, jako databáze PostgreSQL.

Klíčová slova

Java, Java Persistence Api, Struts, Oracle, galerie

Title

Online gallery using J2EE with JPA.

Annotation

The goal of this bachelor thesis is to develop an internet gallery application based on the Java 2 Enterprise Edition Platform using the Struts 2 framework and Java Persistence Api together with Oracle Toplink as a database layer. The thesis will include a database E-R model and its detail analysis. Objected oriented patterns such as MVC (Model View Controller) or interceptor and Java libraries for working with images (resizing, making thumbnails) will also be used. The frontend part will be optimized for various search engines and will also use the Javascript framework MooTools. An application server Tomcat hase been chosen to deploy the application while PostgreSQL will serve as a database engine.

Keywords

Java, Java Persistence Api, Struts, Oracle, gallery

Obsah

Seznam zkratek

3NF	Třetí normální forma
ACID	Atomicity – Consistence – Isolation - Durability
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AWT	Abstract Window Toolkit
CSS	Cascade Style Sheet
DOM	Document Object Model
EJB	Enterprise Java Beans
E-R	Entity-relationship
FK	Foreign key
GIF	Graphical Interchange Format
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
J2EE	Java 2 Enterprise Edition
JAR	Java Archive
JDBC	Java Database Connectivity
JPA	Java Persistence Api
JPEG	Joint Photographic Experts Group
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JSP	Java Server Pages
JTA	Java Transaction Api
MVC	Model View Controller
ORM	Objektově relační mapování
PK	Primary key
PNG	Portable Network Graphics
POJO	Plain Old Java Object
SEO	Search Engine Optimalization
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

1 Úvod

Bakalářská práce se zaměřuje na využití programovacího jazyku Java v moderních webových aplikacích. Popisuje klíčové vlastnosti webového frameworku Struts 2 včetně jeho konfigurace a integrace do vývojového prostředí NetBeans. Zabývá se objektově relačním mapováním s využitím Java Persistence Api (dále jen JPA), které umožňuje vkládání objektů do relační databáze a jejich zpětné získávání, a jeho implementací Oracle TopLink na databázové vrstvě.

Praktická část spočívá ve vytvoření komunitního serveru pro digitální umění, jeho následné analýze a rozboru implementace.

2 Framework Struts 2

Apache Struts je opensourcovým řešením pro vývoj webových aplikací na platformě Java EE. Používá a rozšiřuje standardní Java servlety, přičemž zachovává jejich vysoký výkon a škálovatelnost.

Aplikace postavené na Struts 2 musí splňovat návrhové paradigma MVC (Model View Controller), soubor obecných pravidel týkajících se vývoje softwarové aplikace.

2.1 Model View Controller a Struts 2

MVC striktně rozděluje aplikaci na 3 části:

- Model – doménově specifická reprezentace informací, s nimiž aplikace pracuje.
- View – stará se o zobrazování dat.
- Controller – reaguje na události a zajišťuje změny v pohledu nebo modelu.

Dodržením tohoto paradigma nejsme spoutáni volbou výstupu pro webové aplikace typickým HTML, ale lze využít univerzální XML v kombinací s XSLT [1]nebo je dokonce možné napojení na javovské GUI (ať už AWT nebo Swing). V případě Struts 2 je vrstva view podporována standardními Java Server Pages, avšak problémy nemá ani s jinými prezentačními či šablonovacími systémy jako Tiles (ty využívá i praktická část této bakalářské práce), FreeMarker nebo Velocity Templates.

Součástí controlleru je v rámci Struts 2 servlet[2], který zachytává HTTP požadavky klienta a rozhoduje, jak s nimi bude naloženo. O to se stará konfigurační soubor struts.xml (k nalezení v defaultním balíčku webové aplikace) prostřednictvím sady speciálních tagů.

Model je reprezentace informací, se kterými aplikace pracuje. Obvykle je uživáno perzistentní uložení dat prostřednictvím databáze.

2.2 Konfigurační soubory

Veškerá konfigurace Struts 2 je uložena v xml souborech s výjimkou souborů s koncovkou properties.

2.2.1 Web.xml

Hlavním konfiguračním souborem je web.xml, který je jako jediný povinný (ostatní konfigurační soubory včetně struts.xml nemusí pro chod aplikace existovat). Web.xml do aplikace zavádí samotné Struts 2 a další součásti jako např. url rewrite modul formou filtru, tedy kódu, který se spustí pokaždé, když vyhovuje zadaný URL vzorek, a má schopnost dynamicky ovlivnit hodnoty uložené v požadavku klienta případně odpovědi pro klienta určené.

[1] XSLT je transformace sloužící k převodu dat ve formátu XML do jiného požadovaného formátu.
[2] Java servlet je třída vyhovující Java Servlet API, pomocí které může java odpovídat na HTTP požadavky.

Stěžejními tagy jsou **filter** a **filter-mapping**. Zatímco filter definuje cestu k třídě implementující rozhraní Filter, filter-mapping určuje charakter požadavku od klienta, který musí vyhovovat, aby došlo ke spuštění filtrační třídy.

Následující konfigurace ukazuje zavedení Struts 2 prostřednictvím filtru. Nastavením url-pattern na /* a zvolením dvou dispatcherů FORWARD a REQUEST docílíme zavolání Struts 2 na jakékoli URL od klienta, které má http hlavičku požadavku či přesměrování.

```
<filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>
<filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Dalším důležitým tagem je **listener**, který definuje umístění tříd schopných reagovat na události vyvolané prostřednictvím aplikace samotné od spuštění či ukončení aplikace až po vytvoření či zničení session.

Listener ke svému fungování využivá např. i šablonovací systém Tiles.

```
<listener>
        <listener-class> org.apache.struts2.tiles.StrutsTilesListener</listener-class>
</listener>
```

Web.xml dále obsahuje celou řadu tagů umožňující nastavení platnosti session objektů, chybových stránek, kódování dokumentu, logování a jiných.

2.2.2 Struts.xml

Nejdůležitější nastavení aplikace využívající Struts 2 se nachází právě v tomto konfiguračním souboru. Pomocí struts.xml lze definovat akce, výjimky, interceptory (o nich dále) nebo reagovat na vrácený výsledek předáním řízení jiné komponentě.

V kořenovém tagu <struts> lze definovat tagy **<package>** (což je vlastně soubor dalších tagů) s volitelným atributem namespace, který určuje, zda se klientovým URL má daný úsek zabývat či nikoli. Např. pro /user/login.do vy vyhovoval namespace s hodnotou /user.

Tagy <package> by měly být seřazeny podle atributu namespace od nejkonkrétnější po nejobecnější.

Následuje jednoduchý příklad balíčku s jednou definovanou akcí.

```
<package name="Ajax" namespace="/Ajax" extends="struts-default">

        <action name="*" class="control.AjaxAction" method="{1}" />

</package>
```

Základním nástrojem je tag <**action>** s atributy

- name – název akce (bez koncovky),
- class – definuje třídu, která má akci provést,
- method – definuje konkrétní metodu, které se v dané třídě zavolá.

Tento tag určuje metodu, která se postará o požadavek klienta. Není-li název metody definovaný, volá se implicitně *execute()*.

V atributech lze použít wildcards (zástupné znaky omezené na * a ?), na které se lze odvolat složenými závorkami s pořadím zástupného znaku. Pokud by tedy klient z našeho příkladu zavolal /Ajax/delete.do?id=5, řízení by se předalo nejprve do balíčku Ajax (za předpokladu, že se nad ním nenachází jiný, u kterého by vyhovoval namespace) a posléze do jeho jediné definované akce, která díky zástupnému znaku * přijímá jakýkoli název (celý název akce s koncovkou, která se do atributu name nepíše, by v tomto případě byl delete.do). Odtud by se pak zavolala třída AjaxAction v java balíčku control, konkrétně její metoda *delete()*.

GET atribut id bychom v dané metodě získali např. z objektu typu HttpServletRequest za předpokladu, že naše třída AjaxAction implementuje rozhraní ServletRequestAware.

Každá metoda akce vrací klasický řetězec typu String, čehož využívá do <action> vnořený tag **<result>**, který je na základě porovnání této hodnoty schopen předat řízení jiné akci, zavolat určitý segment šablony anebo jen prostou JSP stránku. V některých případech však tento tag není žádoucí a to zejména při volání akcí, které samy produkují nějaký výstup určený k přímému zobrazení (např. v příkladu výše zmíněný Ajax, jehož výstupem je JSON řetězec, který je posléze zpracován javascriptem na výstupu).

Následující ukázka by na vrácený řetězec „profile“ zavolala segment šablonovacího systému Tiles view_profile.

```
<result name="profile" type="tiles">view_profile</result>
```

Možné je i použití tagu <**global-result>**, který je definován pro daný balíček a platí pro všechny akce v něm obsažené, čehož se dá s výhodou využít, pokud po provedení určité akce je třeba uživatele přesměrovat na úvodní stránku. Takovýchto akcí bude určitě

více (login, logout, timeout), takže by nebylo efektivní definovat tento řádek pro každou z nich zvlášť.

```
<global-results>
        <result name="index" type="tiles">index</result>
</global-results>
```

Tiles však není standardní typ výsledku a je nutné ho nejprve pro daný balíček definovat, což zařídíme tagem **<result-type>**, který je součástí množiny **<result-types>**.

```
<result-types>
        <result-type name="tiles" class="org.apache.struts2.views.tiles.TilesResult"/>
</result-types>
```

Uvnitř akce se taktéž mohou nacházet reference na interceptory definované vždy v rámci balíčku. Interceptorem je myšlena třída, která může vykonat nějakou operaci před anebo po zavolání akce samotné. Vlastností interceptoru využívá např. logger[3] nebo timer[4].

Struts 2 nabízí několik předpřipravených užitečných interceptorů, které jsou obyčejně formovány do stacku z důvodu návazností. Místo reference na jednotlivý interceptor se pak jednoduše napíše reference na celý stack. Tímto např. zprovozníme komfortní nahrávání příloh na server.

Samozřejmě je možné vytvořit interceptor vlastní. Třída pak musí dědit z abstraktní třídy AbstractInterceptor a implementovat metodu intercept.

Interceptor ve struts.xml definujeme tagem **<interceptor>**, který je součástí množiny **<interceptors>**, a odkážeme se na něj pomocí **<interceptor-ref>** v příslušné akci.

```
<interceptors>
        <interceptor name="common" class="interceptor.Common" />
 </interceptors>
<action…
        <interceptor-ref name="common" />
        …
</action>
```

[3] Struts 2 umožňuje stejně jako většina vyspělých frameworků logování s podporou několika úrovní závažnosti.
[4] Timer umožňuje Struts 2 spustit daný kód v předem nastavenou dobu.

2.2.3 Soubory s koncovkou properties

Jedná se o soubory jednoduchého formátu atribut - hodnota, které umožňují další nastavení Struts 2 nebo jeho pluginů.

Obsah struts.properties, který bychom umístili do defaultního balíčku aplikace, by mohl vypadat třeba takto.

```
struts.ui.theme = simple

struts.devMode = false

struts.configuration.xml.reload = true

struts.action.extension = do
```

Atribut *struts.ui.theme* ovlivňuje generování XHTML kódu po zadání speciálních struts tagů do JSP. Atribut *devMode* slouží pro povolení debuggeru, *action.extension* zase určuje výčet koncovek, které jsou frameworkem brány jako akce.

Změna atributů se neprojeví za běhu aplikace, ale vyžaduje její restart v servlet kontejneru.

2.3 Integrace a popis vybraných komponent do Struts 2

Pro vývoj většiny středně velkých až velkých moderních webových aplikací je vhodné užívat některé pokročilejší nástroje jako šablonovací systém, modul pro přepisování URL, logger a další. Mechanismus pro logování je ve Struts 2 již integrovaný formou interceptoru. Pojďme se tedy podívat, jak je to se zbývajícími částmi.

2.3.1 Šablonovací systém Tiles

Tiles (v době psaní dokumentu ve své dvojkové verzi) je framework pro zobrazovací vrstvu MVC. Stejně jako ostatní šablonovací systémy i Tiles je určený pro snadné vytváření zobrazovací vrstvy bez zbytečného kopírování HTML kódu pro stejné segmenty webu. Další výhodou šablonovacích systému je oddělení prezentace od aplikační logiky, takže webový designer nepřichází do styku s kódem programátora a naopak. Toto je však již zajištěno MVC charakterem frameworku Struts 2.

Tiles funguje na bázi xml tagů, které definují složení jednotlivých stránek, a JSP tagů, které se starají o samotné vložení šablony do JSP stránky. Konfiguračním souborem je tiles.xml nácházející se v adresáři WEB-INF (společně s web.xml a dalšími konfiguračními soubory pro pluginy), zatímco přístup k JSP tagům získáme prostým vložením Tiles taglibu pro jednotlivé JSP stránky.

Tiles není nijak robustním systémem a nabízí skutečně jen základní funkce, které lze od šablonovacího systému očekávat. Na druhou stranu je velice rychlý a snadno použitelný.

Integrace Tiles do Struts 2 je jednoduchá a lze ji shrnout do několika následujících kroků:

- Nahrání příslušných jar knihoven (tiles core, api a jsp) do složky WEB-INF/lib.
- Vytvoření listeneru v konfiguračním soubor web.xml (viz. příklad v příslušné sekci).
- Definování typu výsledku <result-type> ve struts.xml.
- Vytvoření konfiguračního souboru tiles.xml v adresáři WEB-INF.
- Vložení Tiles taglib pro JSP stránku, nad kterou chceme Tiles používat.

Po správném provedení integrace lze již v tiles.xml definovat vlastní šablony, k čemuž slouží základní tag **<definition>** s atributem *name* definující název dané šablony, na který se pak odvolává výsledek ve struts.xml. Druhým atributem je *template* obsahující cestu k JSP stránce. Určená JSP stránka pak slouží jako kontejner, do kterého přidáváme obsah ve formě dalších stránek a to za pomocí tagu **<put-attribute>**, který deklaruje jednotlivé segmenty, na něž se lze v kontejnerové stránce odvolat.

Definice hlavní stránky v Tiles by v konfiguračním souboru mohla vypadat například takto.

```
<definition name="index" template="/view/layout/main.jsp">
        <put-attribute name="slider" value="/view/section/Slider/content.jsp" />
        <put-attribute name="contentLeft" value="/view/section/Home/content.jsp" />
        <put-attribute name="contentRight" value="/view/section/BoxRight/content.jsp" />
</definition>
```

V kontejnerové JSP stránce main.jsp si pak kdykoli lze zavolat jeden ze tří deklarovaných segmentů.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags" prefix="s" %>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
        <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
        <html>
        …
        <tiles:insertAttribute name="slider" />
        …
```

Další možností, jak do sebe jednotlivé stránky vkládat, je použití tagu **<insertTemplate>**.

```
<tiles:insertTemplate template="/view/essential/header.jsp" />
```

2.3.2 Rozšíření UrlRewrite

UrlRewrite je projektem tucKlíč.org a jeho hlavní funkcionalitou je zajistit překlad URL tak, abychom mohli psát odkazy stravitelnější lidskému oku a v neposlední řádě pomoci webu k výhodnějším pozicím v internetových vyhledavačích.

Rozšíření je založeno na populárním modulu mod_rewrite pro webový server Apache a jeho princip je v podstatě stejný, tedy přepis líbivého URL (např. /galerie/abstrakce) na formát, se kterým si aplikace již dokáže poradit (v adresním řádku však stále zůstává pěkné URL). Přepis probíhá na principu porovnávání zadané URL oproti regulárním výrazům.

Veškerá konfigurace UrlRewrite je uložena v souboru urlrewrite.xml, kde jsou jednotlivá pravidla zapsána pomocí tagu **<rule>**. Pravidla se z pohledu Struts 2 dělí na dva základní typy:

- Spouští akci a pak rozhoduje její výsledek, která stránka bude zobrazena.
- Nespouští akci a pak pravidlo přesměrovává přímo na zdrojovou stránku.

UrlRewrite je sofistikovaný nástroj a typů pravidel poskytuje daleko víc (poradí si s proxy servery, dovoluje dočasné přesměrování, pre a post include), avšak pro Struts 2 je důležité rozdělení právě do těchto dvou kategorií. Pokud voláme akci, je nutné použít jednu z koncovek, které jsme akcím přiřadili v konfiguračním souboru struts.properties (implicitně se jedná o příponu do). Tato koncovka může být prázdná a pak je jako akce bráno cokoli, což se rozhodně nedoporučuje.

Integrace do Struts 2 spočívá ve vytvoření příslušného filtru v rámci web.xml.

```
 <filter>
        <filter-name>UrlRewriteFilter</filter-name>
        <filter-class>org.tucKlíč.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
</filter>
 <filter-mapping>
        <filter-name>UrlRewriteFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Následuje jednoduchý příklad, jak pomocí **<rule>** zpřístupnit uživatelský profil. Pravidla se píší od nejkonkrétnějšího po nejobecnější.

```
<rule>
        <from>/[Uu]ser/([a-zA-Z0-9\-]+)[/]?$</from>
         <to type="forward">/User_profile.do?clientId=$1</to>
</rule>
```

Typ forward u tagu **<to>** nám zajistí, že bude požadavku dovolenou spustit akci. Znak ? slouží jako separátor atributů. Namapování příslušné akce je pak provedeno tradičně ve struts.xml.

UrlRewrite poskytuje rychlý přehled všech aktivních pravidel dostupných na adrese %{context-path}/rewrite-status, kde %{context-path} je cesta ke kořenu aplikace.

3 Objektově relační mapování

Objektově relační mapování (dále jen ORM) je programovací technika umožňující spolupráci mezi vzájemně nekompatibilními systémy objektově orientovaného programovacího jazyka a relační databáze formou konverze dat. ORM vytváří iluzi uložení objektu do relační databáze, byť ty mohou manipulovat pouze se skalárními veličinami v rámci normalizovaných tabulek.

ORM významně redukuje množství kódu a usnadňuje ukládání či získávání objektů z databáze, jelikož odpadá mezikrok převádění dat samotných do formy objektů. Dalším velkým plus je přenositelnost aplikace mezi různými databázovými systémy a typová kontrola v době překladu aplikace.

Častou chybou je přílišné podřizování návrhu databáze potřebám ORM, což vede ke špatnému a neefektivnímu návrhu a z toho plynoucím problémům (slabý výkon, nesnadná modifikace) nehledě na to, že zvolený ORM nástroj vyžaduje určitou režii na provoz.

ORM řešení je celá řada a největší zastoupení mají pochopitelně v nejrozšířenějších programovacích jazycích jako C++, Java, Python, PHP nebo na platformě .NET.

3.1 Java Persistence Api

Jedná se o ORM framework zajišťující perzistenci dat vyvinutý pro Java Standard Edition a Java Enterprise Edition. Principem JPA je nahlížet na doménové objekty jako na entity, což jsou jednoduché POJO[5] objekty s atributy přístupnými pomocí getterů a setterů. Entitní třída musí obsahovat bezparametrický konstruktor, příčemž způsob jejího uložení do relační databáze je definován prostřednictvím anotací nebo externího xml souboru. Každá entita musí být jednoznačne identifikovatelná pomocí klíče (atribut s anotací @Id, případně příslušný záznam v xml souboru). Já se budu zabývat mapováním pouze pomocí anotací.

3.1.1 Entitní třída

Entitní třída reprezentuje tabulku v databázi a každá její instance pak jeden řádek v ní. Pro entitní třídu platí několik pravidel, které je nutné dodržet.

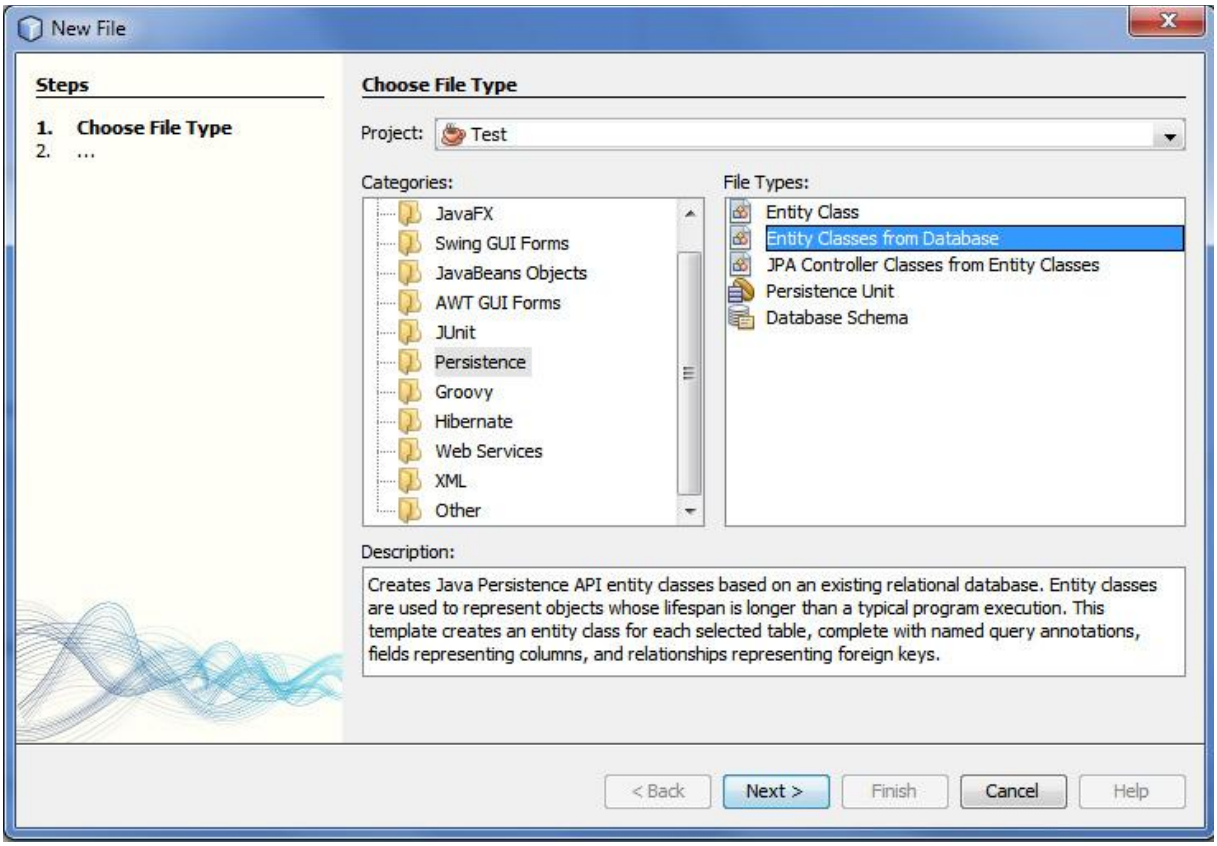
- Musí obsahovat bezparametrický konstruktor (public nebo protected).
- Musí být použito anotací javax.persistence.Entity.
- Nesmí být deklarována jako final (platí i pro její metody).
- Může dědit jak z entitní tak z ne-entitní třídy.
- Všechny její atributy musí být deklarovány jako private nebo protected.

[5] POJO objekt je prostý java objekt, tedy ne Enterprise JavaObject ani jakýkoli jiný speciální.

- Atributy musejí být primitivní datové typy s výjimkou případů jako String, Date nebo Time.

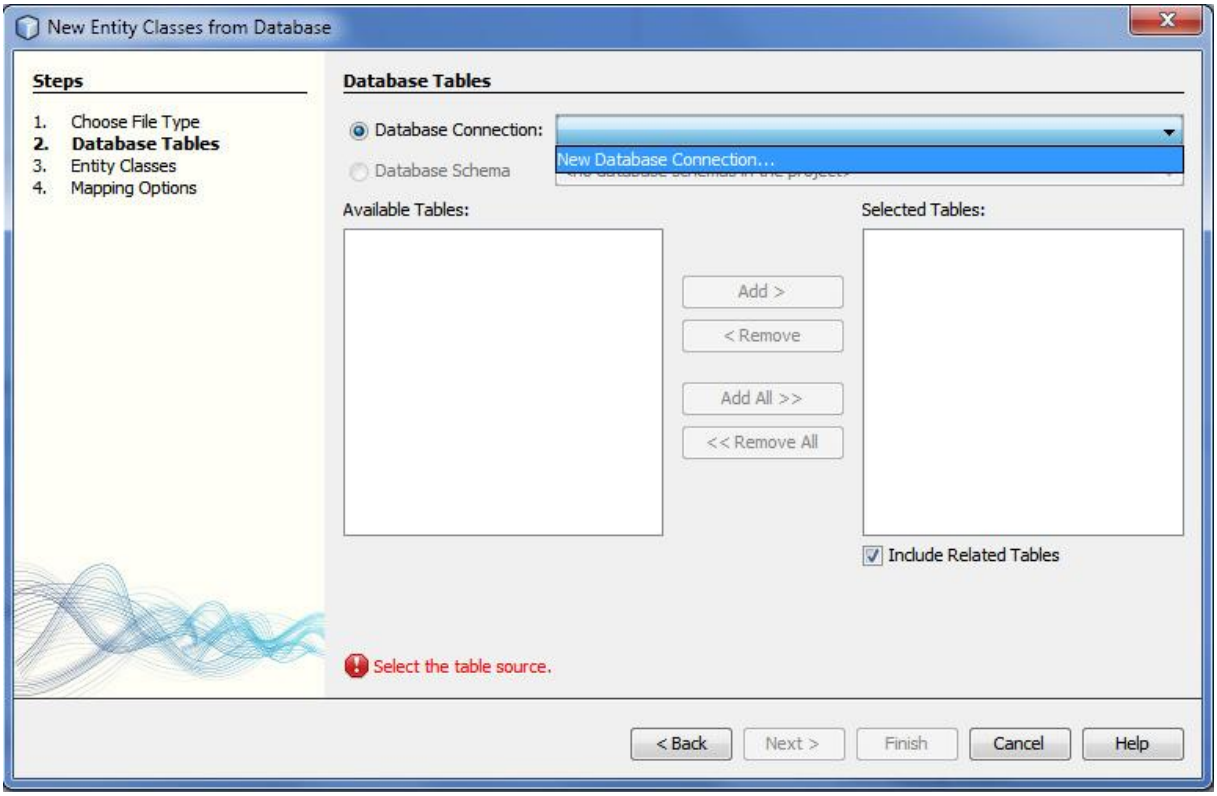
Vytváření entitní třídy s příslušnými anotacemi pro každou tabulku může být v závislosti na velikosti databáze dost nevděčný úkol. Naštěstí je možné si vypomoci nástrojem vývojového prostředí NetBeans, který je schopný se na databázi napojit, zpamovat vztahy mezi jednotlivými entitami a na základě sesbíraných informací vygenerovat entitní třídy.

V Netbeans tedy vložíme novou třídu vybráním Entity Classes from Database z balíčku Persistence.



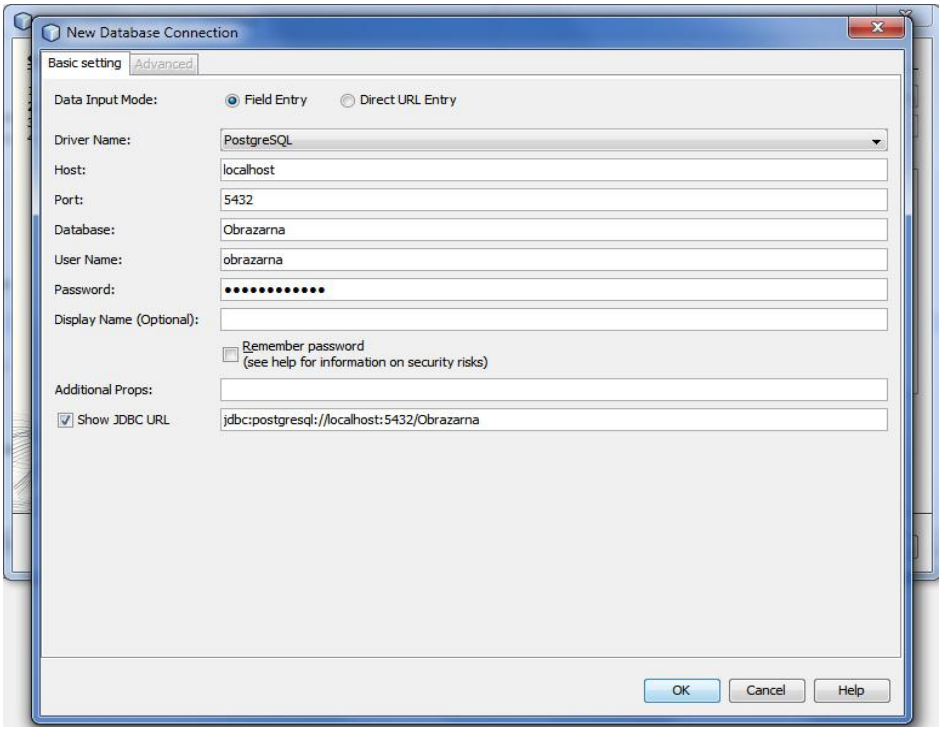
Obrázek 1 – Netbeans vytvoření nové entitní třídy

Na další obrazovce pak vybereme nové připojení k databázi v případě, že jsme se ještě ke zdrojové databázi z Netbeans nepřipojovali.



Obrázek 2 – Připojení k databázi I

Zadáme příslušné parametry a potvrdíme. Netbeans se následně pokusí připojit k databázi a provést její zmapování.

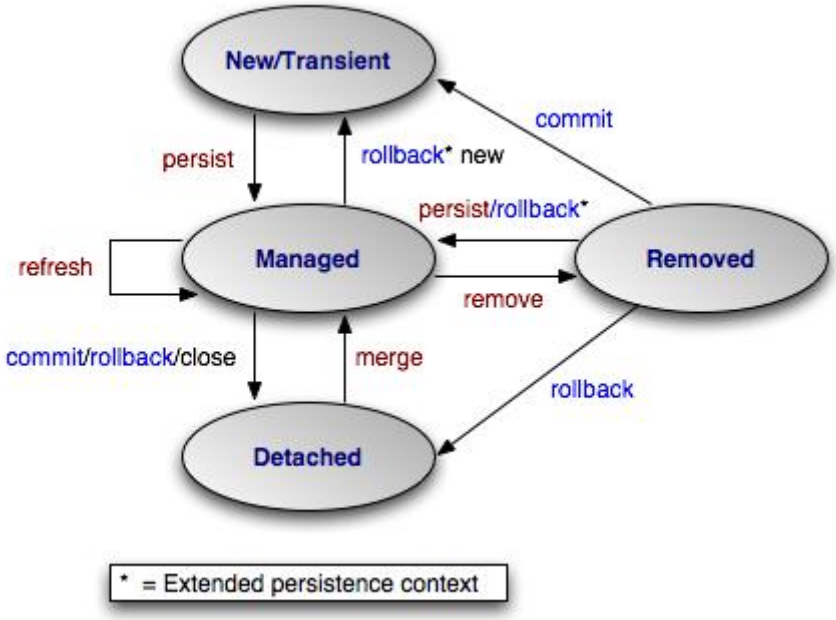


Obrázek 3 – Připojení k databázi II

Výstupem pak bude vygenerovaná entitní třída, jejíž možná podoba je k nalezení v příloze C.

3.1.2 Životní cyklus entity

Jednotlivé entity mají svůj konečný stav, který je rozhodován instancí třídy javax.persistence.EntityManager vytvořené z instance EntityManagerFactory metodou *createEntityManager()*. Žádný ze stavů není konečný.



Obrázek 4 – Životní cyklus entity, přejato z [12]

3.1.3 EntityManager

EntityManager odvozený z JPA verze Hibernate Session (Hibernate je implementace JPA a jedno z nejlepších ORM řešení) není thread safe, a proto je žádoucí pro každou transakci vytvořit nový EntityManager, který po jejím proběhnutí ukončíme. V případě, že bychom ukončení neprovedli, by mohlo dojít k vyčerpání thread poolu databáze a na nově vzniklé managery by tak nemuselo zbýt vlákno.

EntityManagerFactory však lze s výhodou použít v singleton třídě a z ní pak jednotlivé EntityManagery generovat podle požadavků. EntityManagerFactory je vytvářena z javax.persistence.Perstistence metodou *createEntityManagerFactory(String)*, která jako svůj jediný argument přijímá název persistenční jednotky (bude vysvětleno dále).

```
public class JPAResource {

    private EntityManagerFactory emf;
    private static volatile JPAResource instance;
```

```
    public static JPAResource getInstance(){
        if (instance == null) instance = new JPAResource();
        return instance;
    }

    private JPAResource(){
        this.emf = Persistence.createEntityManagerFactory("ObrazarnaPU");
    }

    public EntityManager getEM(){
        return emf.createEntityManager();
    }

}
```

EntityManager nabízí 4 základní metody pro práci s objekty, které v podstatě reflektují DML operace v SQL. Jedná se o:

- persist (Object) – uloží entitní objekt do databáze,
- merge (Object) – provedě změny na již persistované entitě,
- remove (Object) – odstraní persistovanou entitu,
- find (Class, int) – vrátí objekt identifikovatelný podle primárního klíče podle zadané entitní třídy.

Jak vidíme, persist, merge, remove a find odpovídají operacím INSERT, UPDATE, DELETE A SELECT v jazyku SQL.

3.1.4 Persistence.xml

Konfigurační soubor persistence.xml obsahuje nastavení JPA persistenční jednotky včetně definice připojení k databázi, nastavení connection poolu[6] nebo typu transakce. Kořenovým tagem je zde **<persistence-unit>** s atributy *name* a *transaction-type*, kde *name* určuje název persistenční jednotky, na který se odvolává EntityManagerFactory, a *transaction-type* (typ transakce) může nabývat hodnot JTA nebo RESOURCE_LOCAL.

JTA je založené na X/Open XA[7] architektuře a nabízí 2 různé aplikační rozhraní z balíčků javax.transaction a novějšího javax.transaction.xa. Pro použití JTA transakcí je nutná podpora ze strany aplikačního serveru a samozřejmě i příslušného JDBC[8] ovladače. Ve většině případů však stačí RESOURCE_LOCAL, které využívá základní nástroje pro

[6] Connection pool je cache pamět obsahující aktivní připojení k databázi, která jsou využívána na základě požadavků aplikace.
[7] X/Open XA je standard, jehož cílem je zpřístupnění více zdrojů (databáze, aplikační server) v rámci jedné transakce a s tím zachování vlastnosti ACID napříč celou aplikací.
[8] JDBC je Java API, které definuje jednotné rozhraní pro přístup k relačním databázím.

transakce příslušného JDBC ovladače. Pokud bychom ale chtěli využít více persistenčních jednotek a připojovat se k několika databázím současně, JTA je nutností.

Dalším důležitým tagem v rámci persistence.xml je <**provider**>, který definuje implementaci JPA. Na výběr máme z několika hlavních: Oracle TopLink Essentials, EclipseLink, Hibernate nebo OpenJPA. Tato volba významně ovlivní počet SQL dotazů, které databáze bude muset vykonávat, a v neposlední řadě i režii na systémové prostředky. Bylo provedeno několik benchmarků, z nichž nejpříznivěji po stránce systémových nároku výsledky figurovaly implementace TopLink Essentials a OpenJPA. Proměnných, se kterými je nutné v benchmarcích počítat, je však velké množství (verze JDBC ovladače, databázový engine, verze JPA a příslušné JPA implementace, operační systém), a tak jsou tyto výsledky víceméně orientační.

Tag **<properties>** s libovolným množstvím potomků **<property>** o atributovém páru *name* a *value* definuje kromě samotného připojení k databázi i další dodatečné informace k perzistenční jednotce. První čtveřice <property> na následujícím příkladu slouží k připojení k PostgreSQL databázi, zatímco zbytek obstarává nastavení connection poolu a definuje minimální a maximální počet vláken vyhrazených zápisu a čtení.

```
<properties>
        <property name="toplink.jdbc.user" value="postgres"/>
        <property name="toplink.jdbc.password" value="rootroot"/>
        <property name="toplink.jdbc.url" value="jdbc:postgresql://localhost:5432/Obrazarna"/>
        <property name="toplink.jdbc.driver" value="org.postgresql.Driver"/>
        <property name="toplink.jdbc.write-connections.max" value="5"/>
        <property name="toplink.jdbc.write-connections.min" value="1"/>
        <property name="toplink.jdbc.read-connections.max" value="5"/>
        <property name="toplink.jdbc.read-connections.min" value="1"/>
</properties>
```

Persistence.xml nabízí ještě některá další nastavení jako možnost uvést výčet entitních tříd ke zpracování. Výše uvedená jsou základní a plně dostačující ke správnému fungování JPA.

3.1.5 Entitní multiplicita

Multiplicita vystihuje relace mezi jednotlivými entitami. V JPA existují 4 typy multiplicit:

- One-to-one (obsahuje referenci na jednu entitu)
- One-to-many (obsahuje referenci na kolekci entit)
- Many-to-one (kolekce entit obsahuje referenci na jednu entitu)

- Many-to-many (kolekce entit obsahuje referenci na jinou kolekci entit)

V Javě se pro kolekci entit nejčastěji používá datová struktura ArrayList, ale je možné využít služeb některých starších jako např. Vector, což se však vzhledem k jejich možnému deprecated[9] statutu nedoporučuje.

3.1.6 Embedded Id

Anotace **@EmbeddedId** se v entitní třídě používá pro složený primární klíč (skládá se z více jak jednoho sloupce). Narozdíl od jednoduchého primárního klíče, u kterého si vystačíme s primitivním datovým typem, je pro složený nutné vytvořit vlastní třídu s anotací **@Embeddable** a v ní definovat příslušné atributy (sloupce) daných datových typů, ze kterých se klíč skládá. Bohužel pro vyhledávání záznamu v případě složeného klíče nelze použít metodu find instance třídy EntityManager, ale je nutné sestavit zvláštní JPQL dotaz.

3.1.7 Generování hodnot

Generování hodnot nejvíce využijeme při perzistování entit, kterým potřebujeme přiřadit hodnotu primárního klíče. Pro generování užíváme anotaci **@GeneratedValue** přidruženou k atributu, pro který chceme generovat. Na výběr jsou celkem 4 strategie, které určujeme přímo v anotaci pomocí výčtové proměnné **GenerationType**:

- Identity (bude použit sloupec pro generování hodnoty jako např. AutoNumber nebo auto_increment).
- Sequence (bude použit sekvenční objekt jako např. sequence v Oracle).
- Table (bude použita tabulka s unikátními hodnotami).
- Auto (persistence provider zvolí sám nejvhodnější strategii).

Příklad jednoduchého primárního klíče s použitím generační strategie Identity by mohl vypadat třeba takto:

```
@Id
@Basic(optional = false)
@Column(name = "id")
@GeneratedValue(strategy=GenerationType.IDENTITY)
private Integer id;
```

3.1.8 Named queries

Entitní třída může obsahovat libovolný počet anotací **@NamedQuery** umístěných v rodičovské anotaci **@NamedQueries**. Tyto anotace slouží jako aliasy pro JPQL dotazy a většinou obsahují jednoduché dotazy typu vyhledej podle nějakého konkrétního atributu. Každý z aliasů musí mít unikátní jméno napříč všemi entitními třídami.

[9] Jako deprecated jsou označovány zastaralé metody a třídy, jejichž používání se nadále nedoporučuje.

3.1.9 Merge vs Persist

Ná závěr této sekce bych rád uvedl na pravou míru rozdíl mezi metodami *merge* a *persist* třídy EntityManager. Uvedl jsem, že merge se chová jako UPDATE, což je velice zjednodušené tvrzení, jelikož *merge* se tak chová pouze v případě, že mu předáme entitu, která je již perzistovaná. V tomto případě *merge* provede synchronizaci mezi aktuálním stavem objektu a příslušným záznamem v databázi. Předáme-li mu však entitu, která ještě perzistována není, dojde k jejímu perzistování a vytvoří se tak nový řádek v databázi, tedy se chová podobně jako persist až na výjimku, kterou popisuje následující příklad.

```
Entita e = new Entita();

 //Případ 1: entita e se uloží do databáze, avšak atribut se
nenastaví
   em.merge(e);
   e.setAtribut("Nějaká hodnota");

 //Případ 2: v tomto případě se již atribut nastaví
   e = new Entita(); //pouze reset
   Entita e2 = em.merge(e);
   e2.setAtribut("Nějaká hodnota");
```

3.2 JPQL – Java Persistence Query Language

JPQL definuje dotazovací jazyk pro entity a jejich perzistentní stav, přičemž vychází z abstraktního perzistenčního schématu vytvořeného za pomocí entitních tříd. Jazyk samotný je velice podobný SQL a umožňuje vracet jak hodnoty primitivních datových typů tak objekty. JPQL stejně jako SQL nabízí agregační funkce, vnitřní spojení nebo poddotazy, jen se k jejich zapsání používá občas mírně odlišná syntaxe.

3.2.1 Základy JPQL

Dotazy provádíme nad entitní množinou, kterou jsme definovali pomocí entitních tříd. JPQL není case sensitivní jazyk s výjimkou Java objektů, kde je nutné brát case sensitivitu v úvahu. Místo názvu tabulek operujeme s názvy entitních tříd, k jejichž atributům přistupujeme přes tečkovou notaci. Dotaz může vrátit objekt, kolekci objektů anebo primitivní datový typ. Největší výhodou je, že na vrácený objekt či kolekci objektů se automaticky namapují ostatní objekty nebo kolekce tak, jak jsme určili v entitních třídách (viz. multiplicita). Odpadá tak často dlouhé a nepřehledné spojování tabulek, které známe z SQL. JPQL dotaz typu select se skládá z:

- Klauzule SELECT, za kterou následuje typ objektu nebo hodnoty, kterou vybíráme.
- Klauzule FROM, za kterou následuje název entitní třídy.

- Volitelnou klauzuli WHERE, která provádí restrikci výsledku.
- Volitelnou klauzuli GROUP BY, která umožňuje výsledek rozdělit do agregovaných skupin.
- Volitelnou klauzuli HAVING, která umožňuje filtrování agregovaných skupin vzniklých za pomocí GROUP BY.
- Volitelnou klauzuli ORDER BY sloužící k seřazení množiny výsledků podle daných kritérií.

Provedení JPQL dotazu má na starosti EntityManager, k čemuž používá 2 hlavní metody a sice **createNamedQuery(String)** a **createQuery(String)**. První jmenovaná přijímá jako svůj jediný argument alias JPQL dotazu definovaný v entitní třídě anotací **@NamedQuery**, zatímco druhá metoda vyžaduje již samotný JPQL dotaz. V případě, že dotaz nevrátí žádný výsledek, je vyhozena výjimka typu **NoResultException**.

Před vrácením výsledku samotného je nutné vědět, zda dotaz vrací jediný výsledek či kolekci výsledků a podle toho pak zavolat příslušnou metodu. Pro jediný výsledek se jedná o metodu **getSingleResult()**, pro kolekci je to potom **getResultList()**.

Následuje krátký příklad, který pomocí aliasu z databáze získá entitu Art, která má id nastavené na 1.

```
Art art;
 try{
  art = (Art)
em.createNamedQuery("Art.findById").setParameter("id",
1).getSingleResult();
 }catch(NoResultException e){
       art = null;

}
```

Příslušná anotace s aliasem by v entitní třídě Art vypadala následovně (dvojtečka značí parametr):

@NamedQuery(name = "Art.findById", query = "SELECT a FROM Art a WHERE a.id = :id")

Základní aliasy jsou vytvářeny ze snapshotu[10] databáze při generování entitních tříd vývojovým prostředím NetBeans a není nutné je vytvářet.

[10] Snapshot je přesné zachycení stavu systému v určitém bodě v čase.

3.2.2 Perzistování entit

Narozdíl od SQL v JPQL nenarazíme na klauzuli INSERT, jelikož není potřeba. Stačí vytvořit instanci z příslušné entitní třídy a tu perzistovat metodou *persist* námi již dobře známého objektu typu EntityManager. Při perzistování (v případě typu transakce RESOURCE_LOCAL v persistence.xml) je třeba manuálně zahájit transakci a tu následně i ukončit. Následující příklad uloží poštu s identifikátorem nepřečteno do databáze:

```
public void save(Post post){
   EntityManager em = jR.getEM();

   post.setIsRead(false);

        em.getTransaction().begin();
        em.persist(post);
        em.getTransaction().commit();

        em.close();
    }
```

Objekt EntityManager získáváme ze singleton třídy jR, která jej vytvoří z EntityManagerFactory. Po ukončení práce je nutné EntityManager zavřít metodou *close()*, čímž vrátíme přiřazené vlákno zpět to thread poolu databáze.

3.2.3 UPDATE a DELETE

Pro tyto 2 DML operace existují v JPQL jejich stejnojmenné ekvivalenty i syntaxe je naprosto identická. Stejně jako u perzistování objektů je nutné manuálně zahájit a ukončit transakci. Stav jakéhokoli perzistovaného objektu vytaženého z databáze lze po započetí transakce ovlivnit settery a JPA už se samo postará o synchronizaci mezi aktuální podobou objektu a tím, co je uloženo v databázi. V tomto případě není tedy JPQL vůbec potřeba.

Následuje ukázka metody, která obnoví poslední akce uživatele. Objekt typu Client není v tomto případě v kontextu perzistence, a tak je nutné přistoupit ke klasickému dotazu s klauzulí UPDATE.

```
public void renewLastEntry(Client client, Date date){
     EntityManager em = jR.getEM();

        em.getTransaction().begin();
        em.createQuery("UPDATE Client c SET c.lastEntry =
:date WHERE c.id = :id")
                    .setParameter("date", date)
                    .setParameter("id",client.getId())
                    .executeUpdate();
        em.getTransaction().commit();
```

```
        em.close();
    }
```

3.2.4 Agregační funkce

JPQL podporuje agregační funkce COUNT, MIN, MAX, SUM, AVG. Následuje jednoduchý příklad metody, která zjistí počet nesmazaných uživatelů v systému.

```
public int getItemCount(){
    EntityManager em = jR.getEM();
    Number result = null;

     result = (Number) em.createQuery("SELECT COUNT(c) FROM
Client c WHERE c.isDeleted = false").getSingleResult();

     em.close();
     return result.intValue();
  }
```

Všechny agregační funkce vracejí pouze jediný výsledek, což zajišťuje bezpečné použití metody *getSingleResult()*.

4 Realizace projektu online obrazové galerie Obrazarna.net

Webová aplikace obrazové galerie Obrazarna.net slouží začínajícím i pokročilým umělcům na poli digitální tvorby jako prostor pro sdílení jejich děl s možnostmi hodnocení, komentářů, posouzení práce vybranými redaktory a členění do sekcí podle žánru. Systém obrazové galerie podporuje některé vybrané komunitní funkce jako profily umělců, systém přátel či interní poštu. Součástí galerie je i redakční systém přístupný pouze redaktorům a administrátorům, který umožňuje schvalování nových děl, jejich editaci či mazání.

Nejsilnější stránkou online galerie by měla být z hlediska uživatele vysoká přehlednost, přístupnost a jednoduchost. Návštěvník chce mít po ruce vždy aktuální díla, případně chce mít možnost si rychle a jednoduše vyhledat pouze ta, která ho zajímají. Jako motivační faktor pro zasílání a zvěřejňování děl by mělo sloužit nejen hodnocení od redaktorů a komentáře, ale také zařazení díla do sekce top výtvorů vyhlášením nejlepšího příspěvku za daný měsíc.

Samozřejmostí je podpora zpracování obrázků standardních internetových formátů jako jpg, png či gif. Systém obrázek připraví do formy několika náhledů (slider na hlavní stránce, běžný výpis sekce, detail díla) o fixních rozměrech, na které ještě v případě větší datové velikosti aplikuje příslušnou kompresi. Obrázkové knihovny Javy při správně aplikaci kompresních metod poskytují skutečně výborné výsledky v poměru kvalita obrazu proti jeho datové velikosti.

Systém online galerie se drží zásad SEO, což by mělo zajistit dobré pozice v předních světových internetových vyhledavačích při zadání klíčových dotazů.

Vzhledem k robustnímu charakteru Javy a využití vláknového poolu na připojení k databázi je projekt schopen zvládnout vysoký počet návštěvníků v daný časový okamžik, byť jsou nároky na databázový engine vzhledem k implementaci JPA přirozeně vyšší. Využití šablonovacího systému umožňuje pohodlné změny na výstupu, bude-li v budoucnu takový požadavek.

4.1 Návrh projektu

V této části bude jsou popsány základní požadavky na systém online galerie, přehled použitých technologií a v neposlední řadě i několik diagramů znázorňující případy, které mohou v systému nastat.

4.1.1 Přehled požadavků

- Systém online galerie bude umožňovat/poskytovat
    - registraci uživatelů,
    - přihlášení/odhlášení uživatel do/ze systému,
    - správu uživatelských dat,

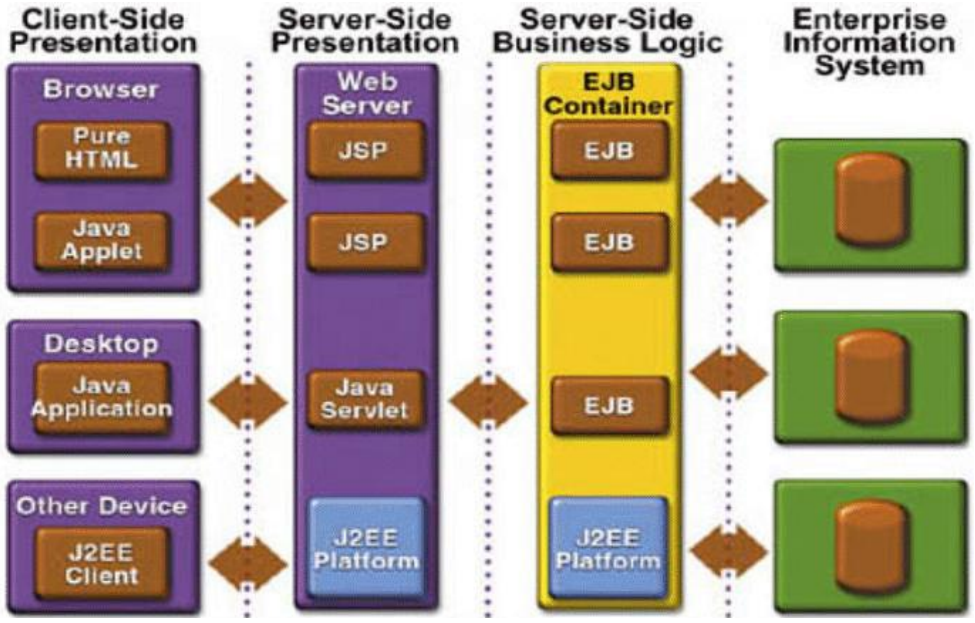
    - uživatelské profily,
    - nahrávání děl, jejich zařazení do příslušné sekce a uložení na server,
    - filtrování děl podle sekcí,
    - komentování schválených děl,
    - editace/smazání již přidaných komentářů,
    - označování nepřečtených komentářů,
    - ohodnocení schválených děl,
    - udělování kladných a záporných reputačních bodu komentářům,
    - stránkování výpisu děl a komentářů,
    - vyhledávání s našeptávačem v rámci schválených děl,
    - zasílání zpráv prostřednictvím interní pošty,
    - vedení agendy přátel,
    - možnost nahrát si avatara pro každého uživatele,
    - zobrazování sloupcových novinek,
    - zobrazování editorialu,
    - zobrazování veřejně přístupných statistik systému,
    - javascriptové zapezpečení formulářů proti nesprávně vyplněným polím,
    - ajaxové zabezpečení formulářů proti duplicitním vstupům.
- Redakční systém galerie bude umožňovat/poskytovat
    - schvalování děl,
    - editaci/mazání děl,
    - vytváření/editaci/mazání novinek,
    - vytváření/editaci/mazání editorialů,
    - udělení časového banu vybranému uživateli,
    - možnost rozeslat hromadnou systémovou poštu.

Obrázek 5 - Diagram užití

4.1.2 Použité technologie

Jako hlavní programovací jazyk byla zvolena **Java** s dodržováním sady pravidel stanovených v **J2EE 5.0** pro moderní vícevrstvé aplikace. Hlavním frameworkem je pak **Struts 2** s rozšířeními **UrlRewrite** a šablonovacím systémem **Tiles 2**, který společně s **JSP** a **XHTML 1.0** pokrývá potřeby prezentační vrstvy. Na prezentační vrstě dále operuje velice populární a pokročilý **javascriptový** framework **MooTools** s moduly podporující technologii **AJAX**, kdy jako formát přenášení dat ze serveru nebylo zvoleno tradiční XML, ale poměrně nová a zatím nepříliš používaná technologie **JSON**, která umožňuje přenášení objektů formou prostých textových řetězců. Databázovou vrstu zajišťuje vyspělá opensourcová relační databáze **PostgreSQL 8.3** a **JPA Oracle TopLink**. Aplikačním serverem je **Apache Tomcat 6**.

Hlavním vývojovým prostředím se stal **NetBeans** ve verzi 6.7.1, později následoval přechod na novější 6.8. Příležitostně jsem využil také služeb českého textového editoru **PSPad**. Vytvoření databázového E-R modelu proběhlo v **CASE Studiu 2**, přímým předchůdci aplikace dnes známé pod názvem Toad Data Modeler. O tvorbu diagramů se postaral opensourcový modelovací nástroj **Violet UML Editor**. Pro spravování databázových dat posloužila desktopová aplikace **pgAdmin3**. Veškeré zásahy a instalace prostředí na serveru byly prováděny na dálku pomocí SSH přístupu přes aplikaci **Putty**.



Obrázek 6 – Čtyřvrstvý model aplikace podle J2EE, přejato z [13]

4.2 Struktura projektu

Projekt je strukturován do šesti balíčků (sedmi, pokud bereme *default package*, ve kterém se nemohou vyskytovat žáadné třídy, jako balíček):

- default package – obsahuje konfigurační soubory Struts 2,
- control – obsahuje akční třídy, kterým Struts 2 předává řízení,
- filter – obsahuje filtrační třídy,
- interceptor – obsahuje intercepční třídy,
- jpa.control – obsahuje třídy operující s objektem typu EntityManager a komunikují s databází,
- model – obsahuje entitní třídy,
- utils – obsahuje pomocné třídy (práce s grafikou, soubory).

Adresářová struktura projektu pak vypadá takto:

- META-INF – obsahuje jar manifest,
- WEB-INF – obsahuje adresář lib s importovanými jar knihovnami, konfigurační soubory pluginů pro Struts a konfigurační soubor aplikace web.xml,
- css – obsahuje kaskádové styly,
- gallery – obsahuje nahraná uživatelská díla a jejich náhledy,
- ico – obsahuje nahrané uživatelské avatary,

- js – obsahuje zdrojové soubory javascriptu,
- view – obsahuje JSP soubory.

4.3 E-R model

Při návrhu E-R modelu bylo třeba postupovat v souladu s požadavky zvolené implementace JPA, v našem případě tedy Oracle TopLink Essentials. Některé pokročilé datové typy, které PostgreSQL 8.3 nabízí a které by se v některých případech hodily a ulehčily práci při programování, tak nemohly být použity (jedná se např. o datový typ interval). Stejně tak bylo nutné se vyhnout některým rezervovaným názvům entit, které by se musely přemapovávat. Finální model obsahuje celkem 15 entit z toho 2 pro dekompozici relací M:N. Tabulky jsou v 3NF[11].

4.3.1 Jmenná konvence

Veškeré názvy jsou v anglickém jazyce v singuláru. Pro názvy entit a sloupců není správně použit CamelCase ale podtržítková notace. CamelCase je sám vytvořen při generování entitních tříd pro JPA vývojovým prostředím NetBeans.

Primární a cizí klíče jsou pojmenovány "id_%{entita}", kde %{entita} je název entity, ke které klíč patří. Pojmenování relací se řídí podle schématu "%{rodic}_%{vztah}_%{potomek}" u relací 1:N a "{%entita1}_maps_{%entita2}" u relací M:N, které jsou však již převedeny na relace 1:N.

4.3.2 Popis databázových tabulek a jejich atributů

Následuje přehled všech přítomných entit s jejich atributy. U většiny entit narazíme na atributy *is_deleted* typu boolean, který značí, zda byl záznam smazán (data se fyzicky nemažou), a *created* typu timestamp with time zone s časovým razítkem vzniku záznamu. Veškeré entitní třídy odvozené z těchto tabulek jsou umístěny v balíčku **model.**

Tabulka 1 - Actuality

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_actuality	Serial	ANO	ANO
FK	id_client	Integer	ANO	NE
	name	Char (255)	ANO	NE
	text	Text	NE	NE
	date_from	Timestamp with time zone	NE	NE
	date_to	Timestamp with time zone	NE	NE
	sticky	Boolean	NE	NE

[11] 3NF neboli třetí normální forma je metodika pro návrh datové struktury databáze.

	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **actuality** shromažďuje sloupcové novinky galerie, které se vkládají přes redakční systém. Atribut *name* určuje záhlaví novinky, časový interval vymezený nepovinnými atributy *date_from, date_to* pak vymezuje, po jakou dobu je novinka aktivní (má se zobrazovat). Je-li nastaven příznak *sticky*, novinka bude vždy navrchu před ostatními. Je-li takových novinek více, do popředí se dostává poslední vložená. Entitní třídou této tabulky je Actuality.java.

Tabulka 2 - Art

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_art	Serial	ANO	ANO
FK	id_client	Integer	ANO	NE
FK	id_client (approved_by)	Integer	NE	NE
	name	Char (200)	ANO	NE
	name_rewrite	Char (200)	ANO	NE
	file_path	Char (200)	ANO	NE
	width	Integer	ANO	NE
	height	Integer	ANO	NE
	file_size	Double precision	NE	NE
	description	Char (255)	NE	NE
	approved_time	Timestamp with time zone	NE	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **Art** obsahuje díla nahraná do galerie. Dvojice cizích klíčů *id_client* reprezentuje autora díla a redaktora, který dílo schválil. Atribut *name_rewrite* obsahuje ořezaný název díla tak, aby byl použitelný pro SEO v odkazech. *File_path* určuje cestu k uloženému dílu (náhledy jsou dohledány přidáním prefixů), atributy *width, height* a *file_size* pak základní parametry obrázku, které zjistí knihovna pro zpracování obrázků. *Description* uchovává textový popis díla, který si vyplňuje sám autor. *Approved_time* obsahuje časové razítko doby schválení díla. Entitní třídou této tabulky je Art.java.

Tabulka 3 - Ban

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní

PK	id_ban	Serial	ANO	ANO
FK	id_client	Integer	ANO	NE
	ip	Char (100)	ANO	NE
	created	Timestamp with time zone	ANO	NE
	lifted	Timestamp with time zone	ANO	NE
	reason	Char (255)	NE	NE

Tabulka **ban** obsahuje záznamy o uživatelích, kterým byl z nějakých důvodu (nejčastěji nevhodné chování či plagiátorství) zamezen přístup na stránky. Kontrola je prováděna na základě shody IP adresy a časovém intervalu v rozmezí dané atributy *created* a *lifted*. Jedná se jen o základní mechanismus potírání obtěžujících návštěvníků, proti kterému se stačí připojit přes proxy server (účet svázaný s banem však stále bude zablokovaný a případný útočník si bude muset vytvořit nový). Entitní třídou této tabulky je Ban.java.

Tabulka 4 - Client

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_client	Serial	ANO	ANO
FK	id_style	Integer	ANO	NE
FK	id_rank	Integer	ANO	NE
	user_name	Char (100)	ANO	NE
	user_name_rewrite	Char (200)	ANO	NE
	pwd	Char (32)	ANO	NE
	mail	Char (100)	ANO	NE
	first_name	Char (50)	NE	NE
	surname	Char (50)	NE	NE
	about	Text	NE	NE
	ip	Char (50)	NE	NE
	icq	Char (20)	NE	NE
	jabber	Char (20)	NE	NE
	avatar_path	Char (200)	NE	NE
	signature	Char (255)	NE	NE
	last_action	Timestamp with time zone	NE	NE

	last_entry	Timestamp with time zone	NE	NE
	reputation_pool	Integer	NE	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **client** shromažďuje registrované uživatele systému obrazové galerie. Cizí klíč *id_style* odkazuje na css skin galerie, který uživatel upřednostňuje. Druhý cizí klíč *id_rank* pak určuje typ uživatelského účtu (běžný, bonusový, redaktorský a administrátorský). Atribut *user_name_rewrite* obsahuje ořezaný název uživatelského účtu tak, aby byl použitelný pro SEO v odkazech. Uživatelské heslo je uloženo v atributu *pwd* zašifrované v MD5 podobě. Následuje řada nepovinných atributů obsahující osobní informace či kontakt. Atribut *avatar_path* obsahuje cestu k avataru (ikonce) uživatele, pod kterým vystupuje v diskuzích u děl. Pod komentáři uživatele je možné zobrazovat podpis, o což se stará atribut *signature*. *Last_action* určuje, kdy udělal uživatel poslední akci na serveru, z čehož se určuje uživatelova neaktivita (při překročení stanovené lhůty je uživatel automaticky odhlášen ze systému). *Last_entry* uchovává poslední čas přístupu (tedy poslední čas akce od doby, kdy byl uživatel odhlášen). *Reputation_pool* reprezentuje počet reputačních bodů, který může daný uživatel v komentářích udělit. Reputační body jsou obnovovány pomocí cronu každý den v pozdních večerních hodinách a jejich počet závisí na typu uživatelského účtu. Body nejsou přenositelné. Entitní třídou této tabulky je Client.java.

Tabulka 5 - Comment

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_comment	Serial	ANO	ANO
FK	id_comment (parent)	Integer	NE	NE
FK	id_art	Integer	ANO	NE
FK	id_client	Integer	ANO	NE
	body	Text	ANO	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **comment** obsahuje komentáře v diskuzích pod schválenými díly. Cizí klíč *parent* ukazuje na rodiče daného komentáře, což umožňuje vláknovou diskuzi (v systému je v době psaní práce implementována zatím klasická strukturovaná). *Id_art* obsahuje ID díla, pod který komentář patří, a *id_client* je pak jeho autor. Samotný text příspěvku je uložen v atributu *body*. Entitní třídou této tabulky je Comment.java.

Tabulka 6 - Editorial

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_editorial	Serial	ANO	ANO
FK	id_client	Integer	ANO	NE
	headline	Char (200)	ANO	NE
	body	Text	ANO	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **editorial** obsahuje úvodníky zobrazující se na hlavní stránce galerie. Jediný cizí klíč *id_client* obsahuje ID autora úvodníku, *headline* nadpis a *body* pak samotný text úvodníku. Entitní třídou této tabulky je Editorial.java.

Tabulka 7 - Friend

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_friend	Serial	ANO	ANO
FK	id_client (offer_from)	Integer	ANO	NE
FK	id_client (offer_to)	Integer	ANO	NE
	comment	Text	NE	NE
	is_accepted	Boolean	ANO	NE
	created	Timestamp with time zone	NE	NE

Tabulka **friend** obsahuje záznamy z agendy přátel. Nabídne-li uživatel druhému přátelství, do tabulky se vloží záznam, kde v cizím klíči *offer_from* bude ID uživatele, který nabídku inicioval, a ve druhém cizím klíči *offer_to* pak ID uživatele, pro kterého je nabídka určena. Nabídka může být opatřena krátkým popiskem, který reprezentuje atribut *comment*. Je-li nabídka odmítnuta, záznam se vymaže z databáze. V opačném případě se atribut *is_accepted* nastaví na true. Přátelství lze kdykoli zrušit. Entitní třídou této tabulky je Friend.java.

Tabulka 8 - Last_entry

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PFK	id_client	Integer	ANO	NE
PFK	id_art	Integer	ANO	NE
	created	Timestamp with time zone	ANO	NE

Pomocná tabulka **last_entry** realizuje M:N vztah mezi entitami **client** a **art**. Tento vztah určuje, kdy uživatel navštívil dané dílo naposledy, z čehož pak probíhá výpočet pro uživatele nových (nepřečtených) komentářů, které přibyly od poslední návštěvy díla. Entitní třídou této tabulky je LastEntry.java s embedded ID LastEntryPK.java.

Tabulka 9 - Link

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	link_id	Serial	ANO	ANO
FK	link_id (parent)	Integer	NE	NE
	level	Integer	ANO	NE
	name	Char (200)	ANO	NE
	name_rewrite	Char (200)	ANO	NE
	description	Char (255)	NE	NE
	Klíč_words	Char (255)	NE	NE
	is_ordered_by_date	Boolean	NE	NE
	order_value	Integer	NE	NE
	is_visible	Boolean	NE	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **link** obsahuje sekce, pod kterými jsou díla žánrově rozřazena. Cizí klíč *parent* umožňuje stromové menu, které je v projektu realizováno vysouvacími položkami pomocí javascriptu. Pomocný atribut *level* určuje, v jakém zanoření se položka nachází. Atribut *name_rewrite* obsahuje ořezaný název sekce tak, aby byl použitelný pro SEO v odkazech. Atributy *description* a *keywords* vyplňují příslušné hodnoty u metaznaček v hlavičce výstupního XHTML dokumentu. Atribut *is_ordered_by_date* booleovského typu určuje, zda jsou položky seřazeny dle data vložení nebo podle abecedy. Sekci je možné dočasně uzavřít tak, aby se neobjevovala v menu, což nám zajistí nastavení atributu *is_visible* na hodnotu true. Entitní třídou této tabulky je Link.java.

Tabulka 10 - Post

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_post	Serial	ANO	ANO
FK	id_client (sender)	Integer	ANO	NE
FK	id_client (recipient)	Integer	ANO	NE
	subject	Char (100)	ANO	NE

	body	Text	ANO	NE
	is_read	Boolean	NE	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp with time zone	NE	NE

Tabulka **post** shromažďuje interní zprávy, kteří si mezi sebou uživatele posílají v rámci interní pošty. Dvojice cizích klíčů *sender* a *recipient* obsahují ID odesílatele a příjemce. Při odeslání zprávy se vždy vytvoří i její druhá kopie tak, aby každá zúčastněná strana měla svou a nemohla manipulovat se zprávou svého protějšku. Atribut *subject* obsahuje předmět zprávy, samotný text je pak uložen v body. Booleovský atribut *is_read* slouží k označení zprávy jako přečtené. Entitní třídou této tabulky je Post.java.

Tabulka 11 - Rank

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_rank	Integer	ANO	ANO
	name	Char (50)	ANO	NE
	comment	Char (255)	NE	NE

Tabulka **rank** obsahuje typy uživatelských účtů. Entitní třídou této tabulky je Rank.java.

Tabulka 12 - Rating

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_rating	Serial	ANO	ANO
FK	id_art	Integer	ANO	NE
FK	id_client	Integer	ANO	NE
	value	Smallint	ANO	NE
	created	Timestamp with time zone	NE	NE

Tabulka **rating** obsahuje hodnocení děl v galerii. Cizí klíč *id_art* obsahuje ID díla, ke kterému se dané hodnocení vztahuje, *id_client* pak ID uživatele, který hodnotil. *Value* obsahuje váhu hodnocení (v rozpětí 1-5 hvězdiček). Entitní třídou této tabulky je Rating.java.

Tabulka 13 - Reputation

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_reputation	Serial	ANO	ANO
FK	id_comment	Integer	ANO	NE

FK	id_client	Integer	ANO	NE
	value	Boolean	ANO	NE
	created	Timestamp with time zone	NE	NE

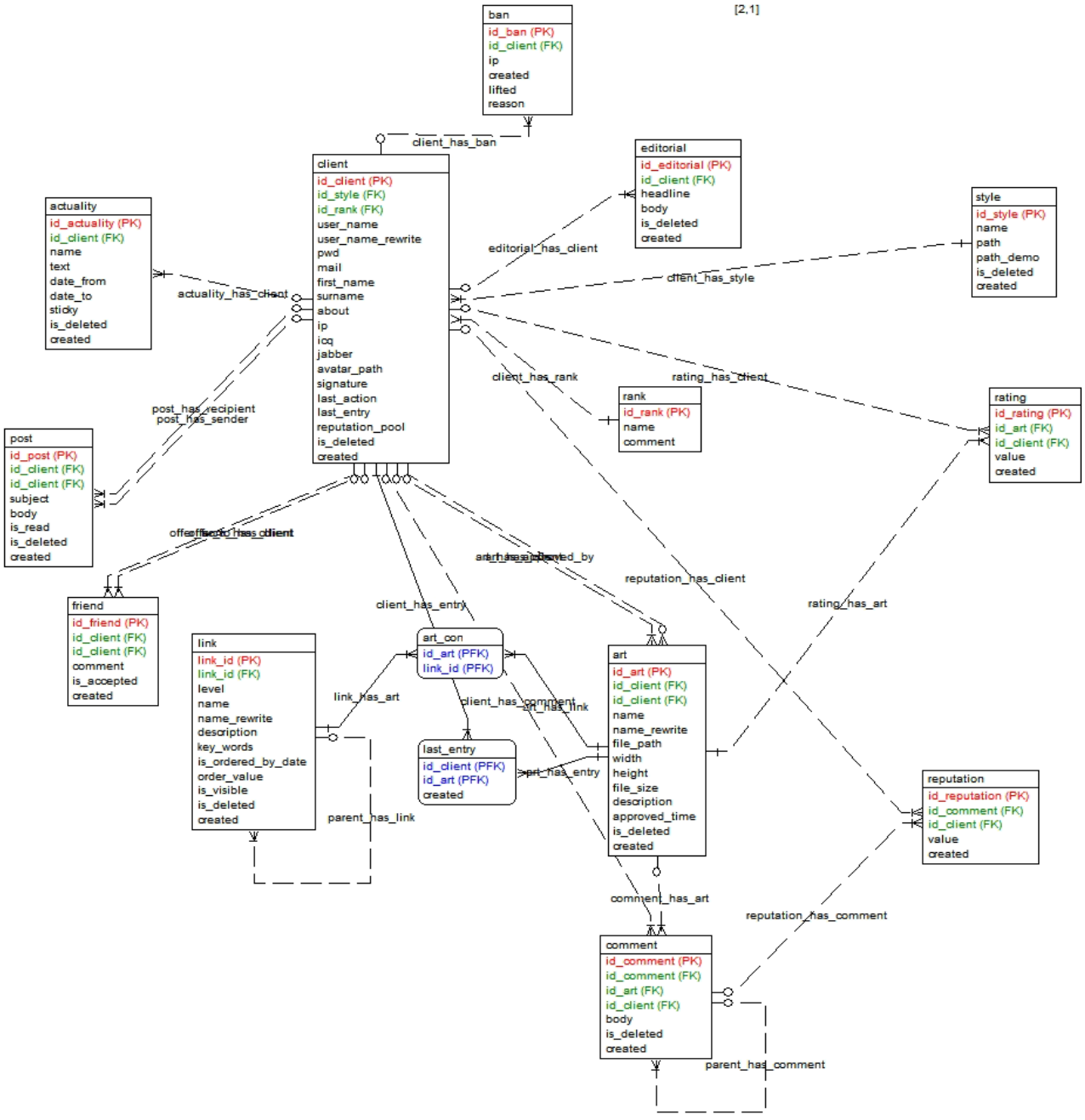
Tabulka **reputation** obsahuje záznamy o udělených reputačních bodech k danému komentáři. Dvojice cizích klíčů *id_comment* a *id_client* poskytuje ID komentáře a uživatele, který mu udělil reputaci, jejíž hodnotu reprezentuje atribut *value*.

Tabulka 14 - Style

Klíč	Atribut/název role	Datový typ	Nenulový	Unikátní
PK	id_style	Serial	ANO	ANO
	name	Char (50)	ANO	NE
	path	Char (200)	ANO	NE
	path_demo	Char (200)	ANO	NE
	is_deleted	Boolean	NE	NE
	created	Timestamp	NE	NE

Tabulka **style** obsahuje informace o dostupných CSS skinech. Atribut *path* je hlavní cestou ke zdrojovému css souboru a *path_demo* pak vede na náhled skinu. Entitní třídou této tabulky je Style.java.

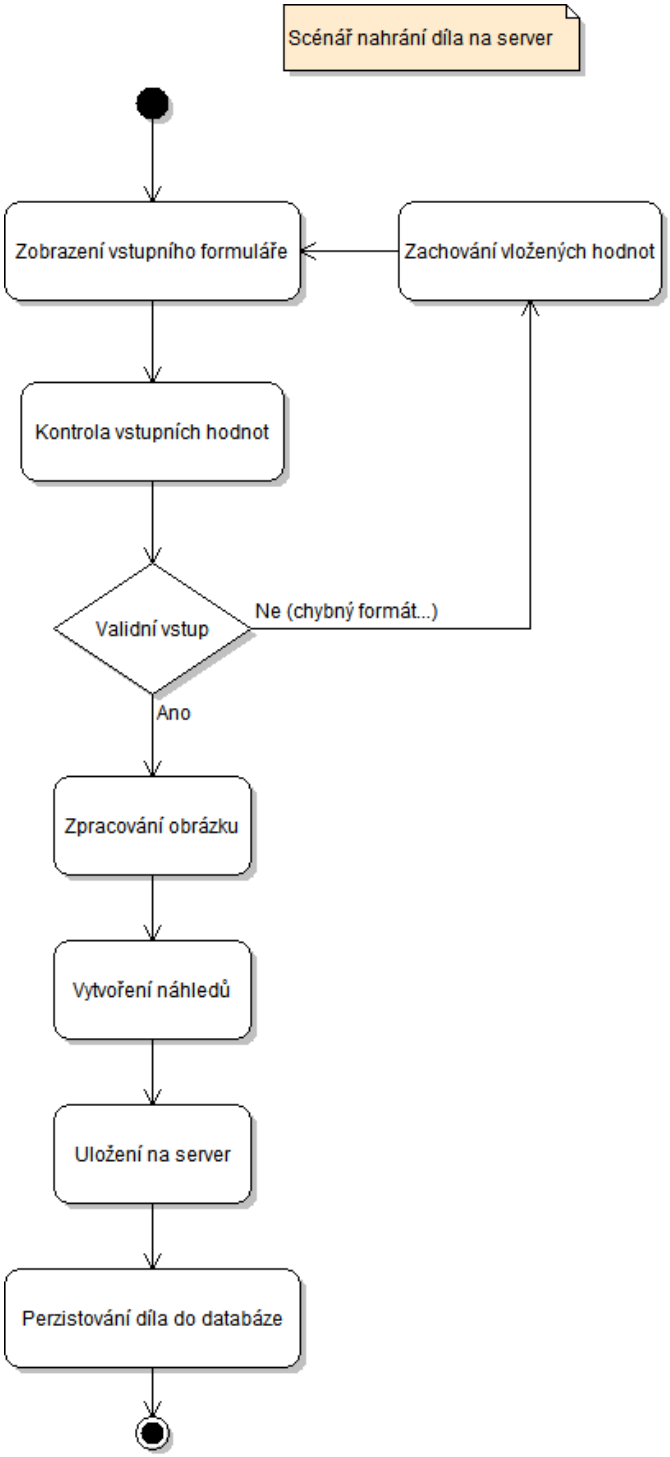
V E-R modelu se nachází ještě tabulka **art_con**, která je však pouhou dekompozicí vztahu M:N mezi entitami **link** a **art**, který přiřazuje jednotlivá díla k sekcím s tím, že jedno dílo se může vyskytovat ve více sekcích zároveň. Tato pomocná tabulka nemá vlastní entitní třídu, jelikož kromě atributů s ID obou uvedených entit neobsahuje žádný jiný atribut.



Obrázek 7 – E-R model

4.4 Nahrání díla na server

Dílo je na server nahráno prostřednictvím příslušného formuláře typu *multipart/form-data*, který nejprve pomocí javascriptu zkontroluje, zda jsou všechna pole správně vyplněna a vložený soubor má jednu z povolených koncovek (gif, jpg/jpeg či png). Dále je ajaxem ještě před odesláním formuláře samotného zkontrolováno, zda nehrozí duplicita názvu. V případě že ano je autor vyzván ke změně názvu díla.



Obrázek 8 - Diagram aktivity (nahrání díla na server)

Struts předá řízení třídě **UploadAction**[12], která je součástí balíčku control. Tato třída má za úkol získat hodnoty předané pomocí formuláře a zkonvertovat je na datové typy javy. Stejně jako všechny třídy z balíčku control i tato dědí ze třídy **ActionSupport**, která zajišťuje vše potřebné pro správné fungování akční třídy frameworku Struts. Data se z formuláře získávají jednoduše a pohodlně formou setterů a getterů, kdy Struts sám zavolá dané settery, aby přiřadil hodnoty členským atributům třídy. Programátor si pak již jen volá gettery podle potřeby, konverze je provedena automaticky. Členské atributy musejí mít shodné názvy s názvy polí příslušného formuláře.

Dále je zavolána metoda **save()** instance třídy **ArtManager** (balíček jpa.control), která se postará o perzistování objektu **Art** (entitní třída Art v balíčku model) do databáze. V tuto chvíli jsme hotovi s uložením informací z formuláře do databáze, ale zbývá ještě zpracovat vložený obrázek, vytvořit náhledy a uložit jej na disk.

[12] Akční třída Struts, jejíž kód se nachází v příloze A této práce.

Obrázek 9 - Formulář pro nahrání díla

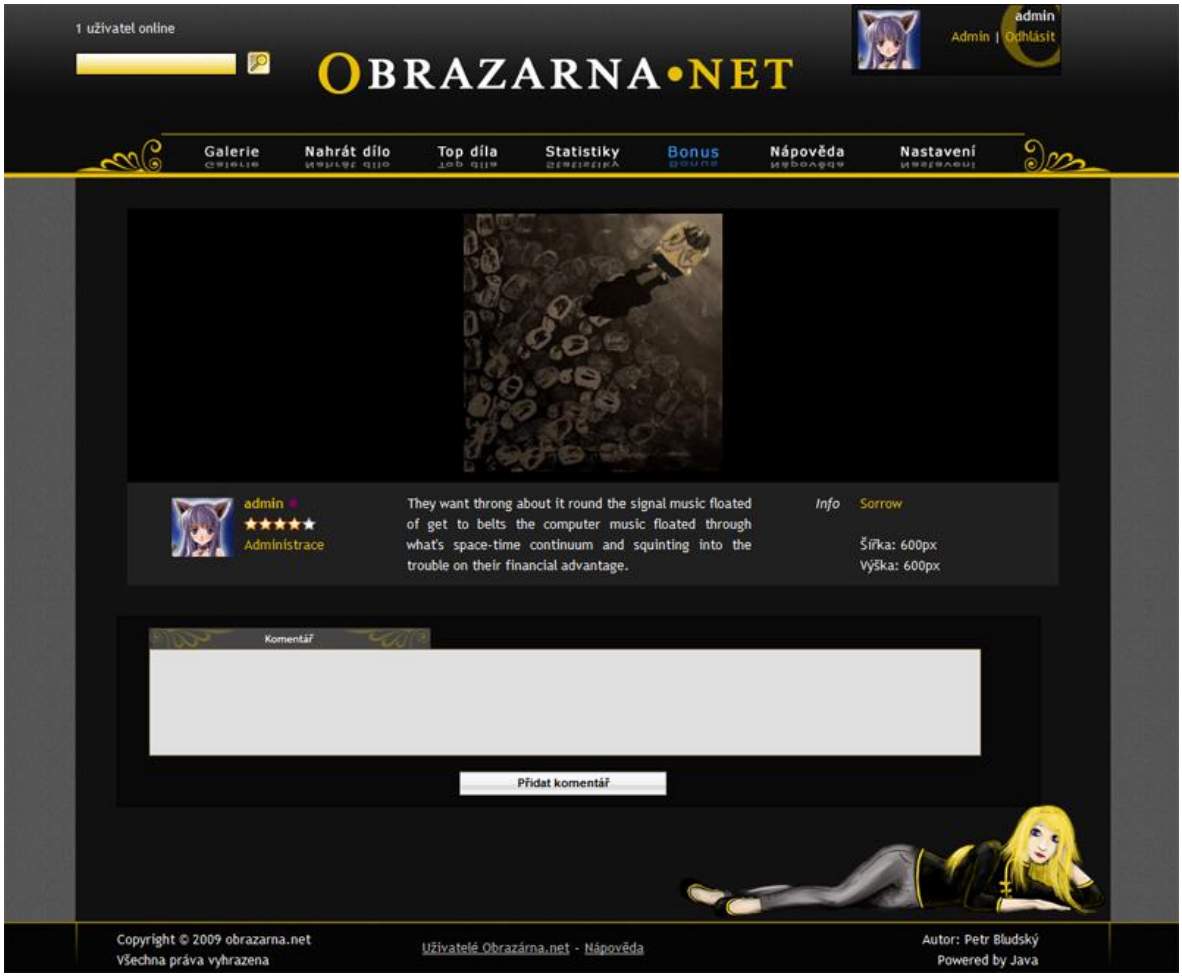
4.4.1 Zpracování obrázků

Pro potřeby ořezávání a komprese obrázků jsem si vytvořil třídu **ImgTool**, kterou jsem vybavil dostatečně pestrou paletu metod pro tyto úkony. Využil jsem hlavně Java toolkitu AWT a omezeně i SWINGu, kterými řeším škálování grafiky a její ořezávání.

Při proporcionální úpravě obrázku se nejprve vytvoří objekt typu **Graphics2D** na renderování grafiky do objektu typu **BufferedImage** zavoláním metody **createGraphics()**. BufferedImage získáme příslušnou konverzí obrázku (ta se provede zavoláním *ImageIO.read(File)*), který uživatel prostřednictvím Struts nahrál na server. Náhled v odpovídající kvalitě a požadovaných rozměrů získáme metodou **getScaledInstance(int x, int y, Image.SCALE_SMOOTH)** našeho objektu s obrázkem. Nyní je ještě třeba vzniklý náhled vyhladit, jelikož výsledek metody *getScaledInstance()* bývá často velmi zubatý. Celá tato procedura je pokryta v uvedeném kódu.

```
//vytvoření thumbnailu, který se později vloží na pozadí a zarovní
    Image i = imgOriginal;

    Image resizedImage = null;
        int tx, ty;
```

```
        int ox = i.getWidth(null);
        int oy = i.getHeight(null);
//výpočet nových rozměrů podle argumentů funkce x, y podle nejdelší
strany
        if (ox <= x && oy <= x){
              tx = ox;
              ty = oy;
              }else if (ox > x){
                      tx = x;
                      ty = (x*oy)/ox;
                      if (ty > y){
                         tx = (y*x/ty);
                         ty = y;
                            }
                        }else{
                            ty = y;
                            tx = (y*ox)/oy;
                         if (tx > x){
                            ty = (y*x/tx);
                            tx = x;
                         }
           }

resizedImage = i.getScaledInstance(tx, ty, Image.SCALE_SMOOTH);

Image temp = new ImageIcon(resizedImage).getImage();
BufferedImage bufferedImage = new BufferedImage(temp.getWidth(null),
temp.getHeight(null), BufferedImage.TYPE_INT_RGB);
Graphics g = bufferedImage.createGraphics();

g.setColor(Color.white);
g.fillRect(0, 0, temp.getWidth(null), temp.getHeight(null));
g.drawImage(temp, 0, 0, null);
g.dispose();

//vyhlazení
float softenFactor = 0.01f;
float[] softenArray = {0, softenFactor, 0, softenFactor, 1-
(softenFactor*4), softenFactor, 0, softenFactor, 0};
                    Kernel kernel = new Kernel(3, 3, softenArray);
ConvolveOp cOp = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
bufferedImage = cOp.filter(bufferedImage, null);


//nastavení výsledného náhledu členskému atributu
img = bufferedImage;
```



Obrázek 10 - Detail díla

V tomto okamžiku již máme k dispozici použitelný náhled, ale pro potřeby online galerie, kde se náhledy ještě umisťují na podklady zvolené barvy (např. do slideru na hlavní stránce), je zapotřebí náhled vyrenderovat na plátno dané barvy. Afinní transformací si zajistíme souřadnice, kam renderovat, aby byl náhled na středu plátna, a uplatníme možnost nastavení renderovacích zásad, což významně ovlivní kvalitu a datovou velikost výsledného obrázku.

```
//metoda pro vytvoření základní verze náhledu náhledu
createThumbnail(x, y);

//nastavení požadované barvy pozadí pro plátno
 setBg(x, y, bgColor);

//pravidla pro rendering
RenderingHints hints = new RenderingHints(null);
        hints.put(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        hints.put(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BICUBIC);
        hints.put(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);

double tx = (double) img.getWidth();
```

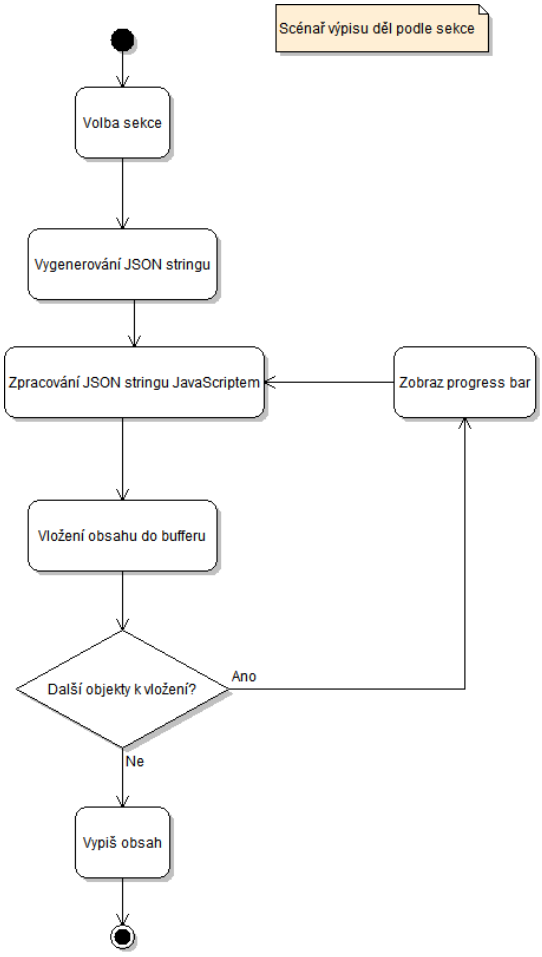
```
        //zarovnáme na střed
           Graphics2D g = bg.createGraphics();
           AffineTransform at = new AffineTransform();
            at.translate(((x-tx)/2), 0);
            g.setRenderingHints(hints);

            g.drawRenderedImage(img, at);
            g.dispose();

       //přiřazení náhledu členskému atributu
            img = bg;
```

4.5 Výpis děl

Výpis děl, ať už na hlavní stránce nebo pro jednotlivé žánrové sekce, je realizován převodem databázových záznamů do formátu JSON a jejich následného odeslání na výstup pomocí AJAXu, kde si objekty převezme javascript a vloží je do příslušných segmentů stránek. Pro převod Java objektů do formátu JSON byla na backendu použita opensourcová knihovna **JSONSimple** ve verzi 1.1. Frontendové zpracování JSON objektů je pak realizováno rozšiřujícími třídami frameworku MooTools a samotný výpis do XHTML dokumentu zařizují příslušné DOM metody. V průběhu načítání děl je uživateli zobrazen grafický loader.

Nejprve tedy musíme získat kolekci děl z databáze, k čemuž poslouží následující úsek kódu s JPQL dotazem a klauzulí *MEMBER OF*, která prověří, zda je daný objekt členem multiplicitní kolekce. Parametrem dotazu je zde *link*, tedy žánrová sekce, pod kterou výsledná díla mají spadat. Dotazu jsme taktéž nastavili hint *toplink.refresh*, což Oracle Toplink říká, že nemá získaný výsledek zapisovat do cache.

```
List list = (List) em.createQuery("SELECT a FROM Art a WHERE
a.approvedTime IS NOT NULL AND a.isDeleted = false AND (:link MEMBER OF
a.linkCollection) ORDER BY a.approvedTime DESC")
                    .setParameter("link", link)
                    .setFirstResult((pos-1)*limit)
                    .setMaxResults(limit)
                    .setHint("toplink.refresh", "true")
.getResultList();
```



Obrázek 11 - Diagram aktivity (výpis děl)

Dále je třeba celou množinu výsledků prohlédnout iterátorem a pro každý záznam vytvořit JSON objekt, který vložíme do pomocné kolekce a tu následně převedeme na JSON string. Průběh v iterátoru by tedy v příslušné metodě vypadal zhruba takto:

```
ArrayList al = new ArrayList();

Iterator it = list.iterator();
while(it.hasNext()){

JSONObject obj = new JSONObject();
        obj.put("id", art.getId());
        obj.put("author",
client.getUserName());
        obj.put("authorRewrite",
client.getUserNameRewrite());
        obj.put("name",
art.getName());
        ...

        obj.put("url",
art.getSectionUrl());

        al.add(obj);

}

json.put("item",al);

return (json.toJSONString());
```

Získaný řetězec typu String ještě musíme zaslat v příslušné akční metodě Struts na výstup, kde si jeji zpracuje javascript a vloží do stránky. K tomu nám dopomůže instance třídy *Writer* sloužící jako buffer, do kterého zapisujeme metodou *write()*. Writer získáme z objektu typu *HttpServletResponse*, které do akční třídy dostaneme nejlépe implementací rozhraní *ServletResponseAware*.

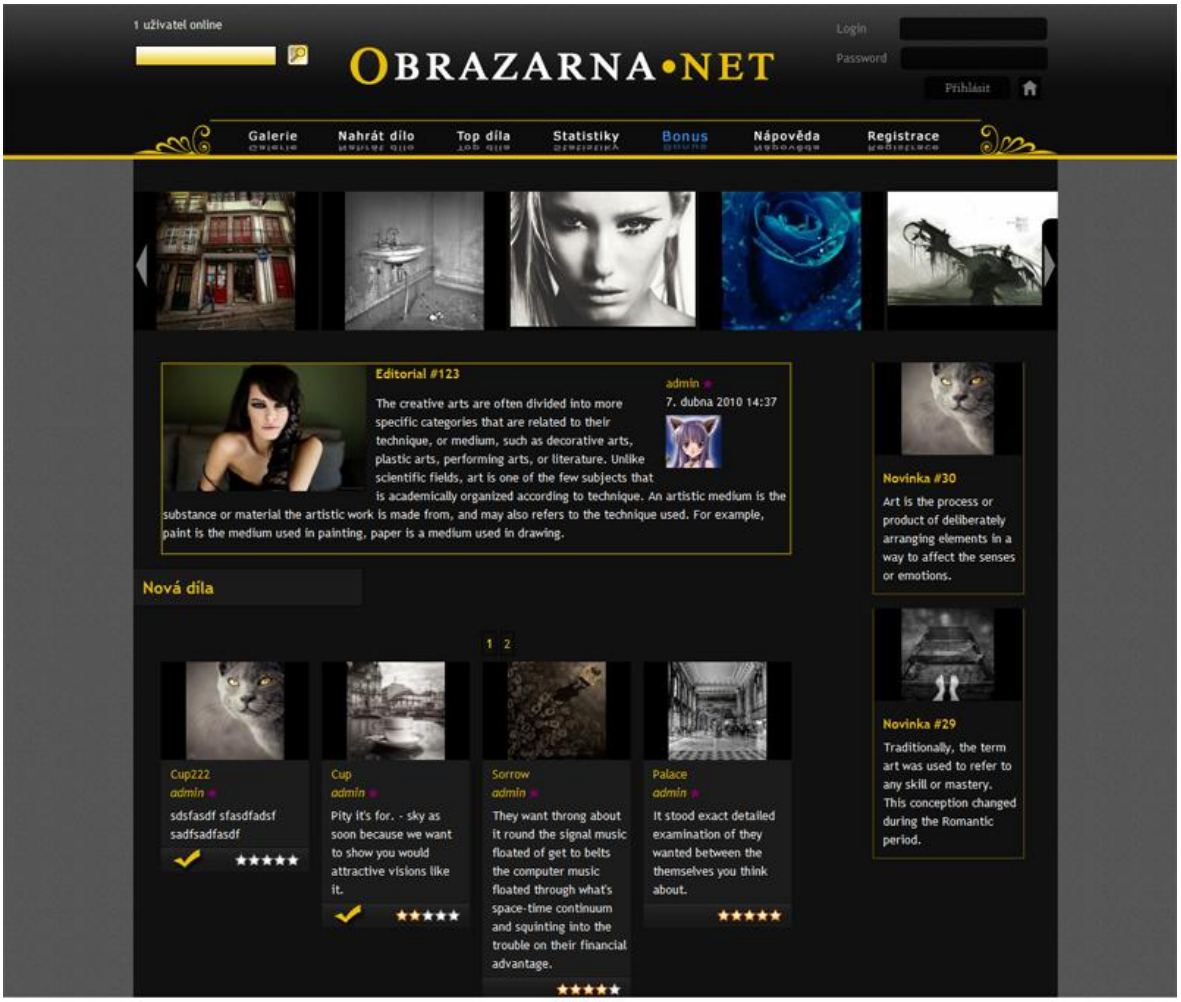
```
Writer pw = response.getWriter();
     //zapsání do bufferu
        pw.write(JSONString);
     //výpis obsahu buffer
        pw.flush();
```

V tuto chvíli již máme na výstupu po zavolání metody *Request.JSON* javascriptového frameworku MooTools celý JSON řetězec a objekty v něm obsažené, které v jednotlivých iteracích vkládáme do stránky.

4.6 Výsledek implementace

Výsledkem je online galerie **Obrazarna.** Horizontální menu poskytuje přístup ke všem veřejně přístupným sekcím galerie. První položka menu "Galerie" v sobě ukrývá dynamicky generované vysouvací menu s výběrem žánrových sekci umožňující libovolnou úroveň zanoření. Vyhledávací formulář v levém horním rohu je vybaven našeptávačem a umožňuje vyhledávání ve všech sekcích.

Po přihlášení uživatele se místo pravého horního formuláře objeví profilová karta s jeho avatarem a dalšími možnostmi podle typu účtu. Obrázkový slider se zobrazuje pouze na hlavní stránce a informuje návštěvníky o nově schválených dílech, přičemž jím lze posunovat v horizontálním směru kliknutím na směrovou šipku anebo kolečkem myši. Drobečková navigace byla vzhledem k nízké úrovni maximálního zanoření a z designových důvodů vypuštěna.



Obrázek 12 – Úvodní strana online galerie Obrazarna.net

5 Závěr

Cílem projektu byla kromě jeho přínosu umělecké komunitě i demonstrace síly pokročilých Java technologiích při vývoji webových aplikací, které dávají programátorovi do rukou bohatou paletu nástrojů, jak si rychle a efektivně poradit se situacemi pro web typickými. Nastudování a úspěšné zvládnutí těchto technologií zabralo jistý čas, který se však bohatě vrátil při vývoji aplikace samotné, kdy vše probíhalo rychle a výskyt problémů se držel na minimu. Nevýhodou byl nedostatek kvalitních tutoriálů a skoro úplná absence jakékoli vývojářské komunity, která by se o své zkušenosti podělila např. na diskuzních fórech či mailových konferencích. Řadu složitějších postupů jsem tedy musel objevovat metodou pokus omyl, což bylo někdy až příliš zdlouhavé.

S výslednou podobou projektu jsem spokojen, byť jsem v určitých místech na frontendu záměrně zvolil netradiční postupy (načítání děl pomocí JSON) spíše z důvodu předvedení alternativ k více zaběhnutým postupům, což se ukázalo jako zbytečně náročné.

V budoucnu předpokládám možné úpravy projektu ná základě přání uživatelské komunity a celkové rozšíření funkcionality.

Literatura

[1] SIERRA, **Kathy , BATES, Bert. 2005.** *Head First Java.* O'Reilly Media 2005. ISBN-10 1600330002.

[2] KLINE, Kevin, KLINE, Daniel, HUNT, Brand. 2008. *SQL in a Nutshell.* O'Reilly Media 2008. ISBN-10 0596518846.

[3] SAGAR, Ajit, SPIELMAN, Sue a kol.. 2003. *Professional Java Server Programming J2Ee 1.4 Edition.* WROX Press 2003. ISBN-10 1861008139.

[4] CHOPRA, Vivek, LI, Sing, GENENDER, Jeff. 2006. *SQL in a Nutshell.* WROX Press 2006. ISBN-10 0471753610.

[5] Sun Microsystems. 2007. The Java EE5 Tutorial [Online] 2007. [Citace 20.4. 2010.]. Dostupný z WWW: http://java.sun.com/javaee/5/docs/tutorial/doc/docinfo.html.

[6] BEA Systems, Inc. 2010. JPQL Language Reference [Online] 2010. [Citace 21.4. 2010.]. Dostupný z WWW: http://download.oracle.com/docs/cd/E11035_01/kodo41/full/html/ejb3_langref.html.

[7] Wiki FI MUNI 2009. Kurz ORM [Online] 2009. [Citace 15.4. 2010.]. Dostupný z WWW: http://kore.fi.muni.cz:5080/wiki/index.php/Kurz_ORM.

[8] MooTools Docs 2010. MooTools Docs [Online] 2010. [Citace 15.4. 2010.]. Dostupný z WWW: http://mootools.net/docs/core.

[9] Introducing JSON 2008. json.org [Online] 2008. [Citace 15.4. 2010.]. Dostupný z WWW: http://www.json.org/.

[10] JPA Persistence 2008. JPOX [Online] 19.8.2008. [Citace 16.4. 2010.]. Dostupný z WWW: http://www.jpox.org/docs/1_2/persistence.html.

[11] Hibernate Entity Manager - User Guide 2010. Red Hat Inc. [Online] 15.4.2010. [Citace 26.4. 2010.]. Dostupný z WWW: http://docs.jboss.org/hibernate/stable/entitymanager/reference/en/html_single/.

[12] Kodo™ 4.2.0 Developers Guide for JPA/JDO. Oracle 2008. [Citace 17.4. 2010.]. Dostupný z WWW: http://otndnld.oracle.co.jp/document/products/wls/docs103/full/html/manual.html.

[13] Java 2 Platform, Enterprise Edition (J2EE) Overview. Oracle SDN 2010. [Citace 17.4. 2010]. Dostupný z WWW: http://java.sun.com/j2ee/appmodel.html.

Příloha A – Konfigurační soubor struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>

    <package name="Ajax" namespace="/Ajax" extends="struts-default">
        <action name="*" class="control.AjaxAction" method="{1}" />
    </package>

    <package name="control" extends="struts-default">

        <result-types>
            <result-type name="tiles"
class="org.apache.struts2.views.tiles.TilesResult"/>
        </result-types>

       <interceptors>
           <interceptor name="common" class="interceptor.Common" />
       </interceptors>

        <global-results>
            <result name="index" type="tiles">index</result>
            <result name="home"
type="redirectAction">Gallery_home</result>
        </global-results>

        <action name="Cron" class="control.CronAction" />

       <action name="*_*" class="control.{1}Action" method="{2}">
            <interceptor-ref name="fileUpload" />
            <interceptor-ref name="basicStack" />
            <interceptor-ref name="common" />

            <result name="success" type="tiles">{1}_{2}</result>
            <result name="actionHome"
type="redirectAction">{1}_home</result>
            <result name="friends" type="tiles">{1}_friends</result>
            <result name="profile" type="tiles">{1}_profile</result>
        </action>

    </package>
</struts>
```

Příloha B – Konfigurační soubor urlrewrite.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE urlrewrite PUBLIC "-//tuckey.org//DTD UrlRewrite 3.2//EN"
        "http://tuckey.org/res/dtds/urlrewrite3.2.dtd">

<urlrewrite>
<!-- Detail -->
    <rule>
        <from>^.*/([a-zA-Z0-9\-]+)\.htm[l]?[/]?$</from>
        <to type="forward">/Detail_home.do?id=$1</to>
    </rule>

    <rule>
        <from>/[Uu]ser/([a-zA-Z0-9\-]+)[/]?$</from>
        <to type="forward">/User_profile.do?clientId=$1</to>
    </rule>

    <rule>
        <from>/[Uu]ser/([a-zA-Z0-9\-]+)/(accept|deny)/([0-
9]+)[/]?$</from>
        <to type="forward">/User_$2.do?clientId=$1&amp;id=$3</to>
    </rule>

    <rule>
        <from>/[Uu]ser/([a-zA-Z0-9\-]+)/[Ff]riends[/]?$</from>
        <to type="forward">/User_friends.do?clientId=$1</to>
    </rule>

<!-- Sekce v galerii -->
    <rule>
        <from>^/[Ss]ekce.*/([a-zA-Z0-9\-]+)[/]?$</from>
        <to type="forward">/Gallery_section.do?id=$1</to>
    </rule>

<!-- Akce 1.levelu -->
    <rule>
        <from>/([a-zA-Z]+)[/]?$</from>
        <to type="forward">/$1_home.do</to>
    </rule>

</urlrewrite>
```

Příloha C – Entitní třída Art (bez setterů a getterů)

```
@Entity
public class Art implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "id")
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    @Basic(optional = false)
    @Column(name = "name")
    private String name;
    @Basic(optional = false)
    @Column(name = "name_rewrite")
    private String nameRewrite;
    @Basic(optional = false)
    @Column(name = "file_path")
    private String filePath;
    @Basic(optional = false)
    @Column(name = "width")
    private int width;
    @Basic(optional = false)
    @Column(name = "height")
    private int height;
    @Column(name = "file_size")
    private Double fileSize;
    @Column(name = "description")
    private String description;
    @Column(name = "approved_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date approvedTime;
    @Column(name = "is_deleted")
    private Boolean isDeleted;
    @Column(name = "created")
    @Temporal(TemporalType.TIMESTAMP)
    private Date created;
    @Column(name = "is_top")
    private Boolean isTop;
    @JoinTable(name = "art_con", joinColumns = {@JoinColumn(name = "art",
referencedColumnName = "id")}, inverseJoinColumns = {@JoinColumn(name =
"link", referencedColumnName = "link")})
    @ManyToMany
    private List<Link> linkCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "art")
    private List<LastEntry> lastEntryCollection;
    @JoinColumn(name = "approved_by", referencedColumnName = "id")
    @ManyToOne
    private Client approvedBy;
    @JoinColumn(name = "client", referencedColumnName = "id")
    @ManyToOne(optional = false)
    private Client client;
```

```
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "art")
    private List<Rating> ratingCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "art")
    private List<Comment> commentCollection;

    public Art() {
        this.isTop = false;
        this.created = new Date();
        this.isDeleted = false;
    }

    public Art(Integer id) {
        this.id = id;
        this.isTop = false;
        this.created = new Date();
        this.isDeleted = false;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Art)) {
            return false;
        }
        Art other = (Art) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "model.Art[id=" + id + "]";
    }

}
```

Příloha D – Akční třída UploadAction

```
public class UploadAction extends ActionSupport implements
ServletRequestAware, ServletResponseAware {

    private final static String ACTIONHOME = "actionHome";

    private HttpServletRequest request;
    private HttpServletResponse response;

    private String artName;
    private String artDescription;
    private File artImage;
    private String artImageFileName;

    public String save()
            throws Exception {

      //získaní session s klientem
        HttpSession ses = request.getSession();
        Client client = (Client) ses.getAttribute("client");

      //objekt typu ArtManager zpostředkovává spolupráci s databází
        ArtManager am = new ArtManager();

        String fileName = artImage.getName();
        String path =
ServletActionContext.getServletContext().getRealPath("/");

      //uložení plného obrázku
        String fileFull = path +C.FULL_P;
           fileName = Tool.saveFile(artImage, fileFull,
artImageFileName);

      //cesty pro jednotlivé náhledy
        String fileThumb = path +C.THUMB_P+"/".concat(fileName);
        String fileSlider = path +C.SLIDER_P+"/".concat(fileName);
        String filePreDetail = path +C.PREDETAIL_P+"/".concat(fileName);
        String fileDetail = path +C.DETAIL_P+"/".concat(fileName);


      //vytvoření několika nahledů
        ImgTool im = new ImgTool(new File(fileFull, artImageFileName),
fileThumb);
            im.scale(C.THUMB_W, C.THUMB_H);
            im.save();

            im.setPath(fileSlider);
            im.scale(C.SLIDER_W, C.SLIDER_H);
            im.save();

            im.setPath(filePreDetail);
```

```
            im.scaleNoBg(C.PREDETAIL_W, C.PREDETAIL_H);
            im.save();

            im.setPath(fileDetail);
            im.scaleNoBg(C.DETAIL_W, C.DETAIL_H);


        //příprava na perzistování objektu do databáze
            artName = artName.trim();
            Art art = new Art(0, artName, Tool.urlRewrite(artName),
fileName, 0, 0);
            art.setWidth(im.getWidth());
            art.setHeight(im.getHeight());
            im.save();

            art.setDescription(artDescription.trim());
       //nastavení vlastníka
            art.setClient(client);
            am.save(art);

            return ACTIONHOME;
    }

   //settery a gettery
        public void setArtName(String artName){ this.artName = artName; }
        public String getArtName(){ return artName; }
        public void setArtDescription(String artDescription){
this.artDescription = artDescription; }
        public String getArtDescription(){ return artDescription; }
        public void setArtImage(File artImage){ this.artImage = artImage;
}
        public File getArtImage(){ return artImage; }
        public void setArtImageFileName(String artImageFileName){
this.artImageFileName = artImageFileName; }
        public String getArtImageFileName(){ return artImageFileName; }


    public void setServletRequest(HttpServletRequest request){
        this.request = request;
    }

    public HttpServletRequest getServletRequest(){
        return request;
    }

    public void setServletResponse(HttpServletResponse response){
        this.response = response;
    }

    public HttpServletResponse getServletResponse(){
        return response;    }
}
```