

# Agregátor veřejných zakázek

Bc. Antonín Juran

---

Diplomová práce  
2014



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2013/2014

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Antonín Juran**  
Osobní číslo: **A12733**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**  
Forma studia: **kombinovaná**

Téma práce: **Agregátor veřejných zakázek**

Zásady pro vypracování:

1. Prostudujte informační systémy veřejných zakázek a předchozí diplomové práce na téma agregace informací o veřejných zakázkách.
2. Revidujte databázovou strukturu a parsery pro agregaci údajů o zakázkách, která byla použita v předchozí diplomové práci.
3. Implementujte parsery pro načítání údajů z dalších zdrojů.
4. Implementujte portál pro zobrazování a vyhledávání v agregované databázi.
5. Implementujte systém pro notifikaci registrovaných uživatelů o nových zakázkách na základě jimi zadaných kritérií (oborů, klíčových slov, regionu, atd.)

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. SPELL, Brett. Java: programujeme profesionálně. Vyd. 1. Praha: Computer Press, 2002, 1022 s. Programujeme profesionálně. ISBN 80-722-6667-5.
2. HEROUT, Pavel. Java a XML. 1. vyd. České Budějovice: Kopp, 2007. ISBN 978-80-7232-307-4.
3. BAUER, Christian a Gavin KING. Hibernate in action. Greenwich: Manning Publications, xxiii, 408 s. ISBN 19-323-9415-X.
4. OBE, Regina a Leo HSU. PostgreSQL: up and running. 1. ed. Sebastopol, CA: OReilly. ISBN 978-144-9326-333.
5. OBE, O. Regina a Leo S. HSU. PostGIS in action. Revised. Greenwich, Conn: Manning. ISBN 978-193-5182-269.
6. SARIN, Ashish. Portlets in action. Shelter Island, NY: Manning. ISBN 978-193-5182-542.
7. SEZOV, Rich. Liferay in action: the official guide to Liferay Portal development. Shelter Island, NY: Manning, 2011. ISBN 19-351-8282-X.

Vedoucí diplomové práce:

**Ing. Tomáš Dulík, Ph.D.**

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

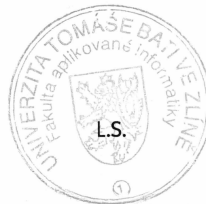
**21. února 2014**

Termín odevzdání diplomové práce:

**20. května 2014**

Ve Zlíně dne 21. února 2014

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



doc. Mgr. Roman Jašek, Ph.D.  
*ředitel ústavu*

### **Prohlašuji, že**

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

### **Prohlašuji,**

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

Ve Zlíně

.....  
podpis diplomanta

## **ABSTRAKT**

Cílem práce bylo vytvořit aplikaci agregující zakázky z e-tržišť do centrální databáze, poskytující portál pro jejich zobrazování a vyhledávání v databázi a nabízející službu zasílání notifikací o nových zakázkách registrovaným uživatelům. V teoretické části je představena legislativa týkající se oblasti veřejných zakázek, zejména zákon 137/2006 Sb., o veřejných zakázkách, dále pak nástroje a technologie použité při vývoji zmíněné aplikace. Je popsána portletová technologie Javy, portálový server Liferay Portal pro nasazení portletu, frameworky Spring a Hibernate použité při vývoji aplikace. Jsou představeny také HTML parser jsoup, který je použit pro získávání dat ze stránek s veřejnými zakázkami, Google Geocoding API využité pro získávání geografických souřadnic adres zadavatelů veřejných zakázek a databáze PostgreSQL pro uložení získaných dat. Praktická část popisuje konkrétní kroky vývoje aplikace využívající nástroje a technologie uvedené v teoretické části. Aplikace je rozdělena do tří samostatných částí, a to systém pro získávání dat z e-tržišť veřejných zakázek a jejich agregaci do databáze, aplikaci zobrazující informace o veřejných zakázkách umožňující vyhledávání zakázek podle zadaných parametrů s možností registrace nových uživatelů a přihlášení registrovaných uživatelů a systém pro zasílání notifikací registrovaným uživatelům o nových zakázkách.

Klíčová slova:

Portlet, Liferay Portal, Spring, Hibernate, veřejné zakázky, jsoup, Google Geocoding API, PostgreSQL.

## **ABSTRACT**

The aim of the thesis was to create an application, which aggregates public contracts from e-markets into a central database, which provides a portal for their visualisation and for database searching and offers notification service of new contracts for registered users. In the theoretic part is introduced the legislation of the public law, especially the act 137/2006 Coll. on Public Contracts, further then tools and technologies used at application development. The portlet technology of Java is described and also portal server Liferay Portal for portlet deployment, frameworks Spring and Hibernate, that were used in application development. HTML parser jsoup, that is used for data gain from pages with public contracts, Google Geocoding API, which is used for gain of geographic coordinates of addresses of the contracting authorities, and PostgreSQL database, which is used for storage of gained data, are also introduced. The practical part describes specific steps of application development with usage of tools and technologies mentioned in the theoretic part. The application is divided into three separate parts, that is system for gain of data from public contract e-markets and their aggregation into the database, application, which displays information of public contracts and enables contracts search according to the given parameters with possibility of new user registration and logging in of registered users, and system for sending of notifications about new contracts for registered users.

Keywords:

Portlet, Liferay Portal, Spring, Hibernate, public contracts, jsoup, Google Geocoding API, PostgreSQL.

Na tomto místě bych chtěl poděkovat panu Ing. Tomáši Dulíkovi, Ph.D. za cenné rady a odbornou pomoc při vedení mé bakalářské práce.

# OBSAH

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>12</b>
<b>1 ZÁKON O VEŘEJNÝCH ZAKÁZKÁCH</b> .....	<b>13</b>
1.1 ZADAVATEL VEŘEJNÉ ZAKÁZKY.....	13
1.1.1 Veřejný zadavatel.....	13
1.1.2 Dotovaný zadavatel.....	14
1.1.3 Sektorový zadavatel.....	14
1.1.4 Centrální zadavatel.....	14
1.2 VEŘEJNÁ ZAKÁZKA.....	14
1.2.1 Veřejná zakázka na dodávky.....	15
1.2.2 Veřejná zakázka na stavební práce.....	15
1.2.3 Veřejná zakázka na služby.....	15
1.2.4 Veřejná zakázka v oblasti obrany nebo bezpečnosti.....	16
1.2.5 Veřejná zakázka podle výše předpokládané hodnoty.....	16
1.2.6 Předpokládaná hodnota.....	16
1.3 ZADÁVACÍ ŘÍZENÍ.....	18
1.4 UVEŘEJŇOVÁNÍ.....	18
1.5 ELEKTRONICKÁ TRŽIŠTĚ VEŘEJNÉ SPRÁVY.....	18
<b>2 PORTLETOVÁ APLIKACE</b> .....	<b>20</b>
2.1 PORTÁL A PORTLET.....	20
2.2 PORTÁLOVÁ INFRASTRUKTURA.....	20
2.2.1 Portletový kontejner.....	20
2.2.2 Portálový server.....	21
2.3 PORTÁLOVÝ SERVER LIFERAY PORTAL.....	21
2.3.1 Instalace Liferay Portal 6.1 CE GA3.....	22
2.3.2 Konfigurace IDE Eclipse pro práci s Liferay Portal.....	23
2.3.2.1 Vytvoření profilu serveru v Eclipse.....	23
2.3.2.2 Specifikace instalačního adresář serveru Tomcat.....	23
2.3.2.3 Konfigurace serveru.....	24
2.3.2.4 Adresářová struktura portletového projektu.....	24
2.4 DEPLOYMENT DESCRIPTOR PORTLETU.....	25
2.5 TVORBA PORTLETOVÉ TŘÍDY POMOCÍ JAVA PORTLET API.....	27
2.5.1 Životní cyklus portletu.....	27
2.5.2 Portletové URL.....	28
2.5.3 Tvorba portletové třídy zděděním třídy <code>GenericPortlet</code> .....	29
2.5.4 Tvorba portletových URL.....	32
2.5.5 Třídy a rozhraní z Portlet API.....	33
2.6 FRAMEWORK SPRING.....	36
2.6.1 Dependency injection a aplikační kontext.....	36
2.6.2 Anotacemi řízený vývoj.....	36
2.6.3 Funkce <code>classpath-scanning</code> .....	37
2.6.4 Dependency injection anotací <code>@Autowired</code> .....	37

2.7	FRAMEWORK HIBERNATE .....	38
2.7.1	Java Persistence API (JPA).....	38
2.8	INTEGRACE PORTLETOVÉ APLIKACE S DATABÁZÍ .....	39
2.8.1	Definice datového zdroje .....	39
2.8.2	Přístup k databázi pomocí Spring s Hibernate .....	41
<b>3</b>	<b>JAVA HTML PARSER JSOUP .....</b>	<b>44</b>
3.1	PŘÍKLAD.....	44
<b>4</b>	<b>POSTGRESQL.....</b>	<b>45</b>
<b>5</b>	<b>GOOGLE GEOCODING API.....</b>	<b>46</b>
<b>6</b>	<b>ANT.....</b>	<b>50</b>
<b>II</b>	<b>PRAKTICKÁ ČÁST .....</b>	<b>52</b>
<b>7</b>	<b>VÝVOJ APLIKACE .....</b>	<b>53</b>
7.1	POPIS APLIKACE .....	53
7.2	APLIKACE TENDER AGGREGATOR DATA MINING .....	55
7.2.1	Persistentní třídy.....	55
7.2.2	Získávání dat z e-tržišť.....	57
7.2.3	Komunikace s databází .....	59
7.3	APLIKACE TENDER AGGREGATOR PORTLET .....	61
7.3.1	Implementace portletový třídy .....	61
7.3.2	Třídy portletové aplikace .....	66
7.4	APLIKACE TENDER AGGREGATOR NOTIFICATION.....	69
	<b>ZÁVĚR .....</b>	<b>73</b>
	<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>74</b>
	JAVA NAMING AND DIRECTORY INTERFACE. JAVA API PRO ADRESÁŘOVOU SLUŽBU, KTERÉ UMOŽŇUJE SOFTWAREM KLIENTŮM JAVY ZJISTIT A VYHLEDAT DATA A OBJEKTY POMOCÍ JMÉNA.....	75
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>76</b>
	<b>SEZNAM TABULEK.....</b>	<b>77</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>78</b>

## ÚVOD

Problematika veřejných zakázek byla a pravděpodobně vždy zůstane velmi diskutovaným tématem. Veřejné instituce a další subjekty hospodařící s prostředky daňových poplatníků nejsou motivovány k tvorbě zisku. Uvedená skutečnost tak často vede k neefektivnímu nakládání s těmito prostředky a mnohdy také k jejich zneužití.

Zboží, služby či stavební práce poptávané ve veřejném zájmu, jinak veřejné zakázky, mají pro jejich dodavatele velkou atraktivitu. Status veřejné instituce může představovat na jedné straně relativně nenáročného a na druhé pak velmi důvěryhodného zákazníka. Plnění prováděné v rámci veřejné zakázky nemusí být vyžadováno s takovou důsledností, jako v případě plnění pro soukromé subjekty. Naopak míra rizika spojená s nesplacením závazků je velmi nízká, protože je krajně nepravděpodobné, že by veřejná instituce zkrachovala. Míra atraktivity může být pro některé potenciální dodavatele tak vysoká, že je vede ke snaze získat veřejnou zakázku všemi možnými prostředky včetně těch nelegálních, jako je korupce.

Předmětem příslušné legislativy, zejména zákona o veřejných zakázkách je tak úprava postupu při zadávání veřejných zakázek, v němž velmi důležitou roli hraje transparentnost. Jedním ze základních nástrojů pro zajištění transparentnosti je uveřejňování veřejných zakázek. Pro jejich elektronické zveřejňování slouží Věstník veřejných zakázek v elektronické podobě a e-tržišť veřejných zakázek.

Hlavním cílem této práce je vytvoření aplikace agregující zakázky z e-tržišť do centrální databáze, poskytující portál pro jejich zobrazování a vyhledávání v agregované databázi a nabízející službu zasílání notifikací o nových zakázkách registrovaným uživatelům.

Teoretická část nejdříve seznamuje čtenáře s legislativou týkající se oblasti veřejných zakázek, obzvláště se zákonem 137/2006 Sb., o veřejných zakázkách, dále představuje nástroje a technologie použité při vývoji aplikace. Je popsána portletová technologie Javy, portálový server Liferay Portal použitý pro nasazení portletu, využití frameworků Spring a Hibernate pro vývoj portletové aplikace. Obsahem teoretické části je také popis HTML parseru jsoup, který je použit pro získávání dat ze stránek s veřejnými zakázkami, popis Google Geocoding API využitého pro získávání geografických souřadnic adres zadavatelů veřejných zakázek a představení databáze PostgreSQL pro uložení získaných dat.

Praktická část popisuje konkrétní kroky vývoje aplikace využívající nástroje a technologie uvedené v teoretické části. Aplikace je rozdělena do tří částí, a to systém pro získávání dat z e-tržišť veřejných zakázek a jejich agregaci do databáze, aplikaci zobrazující informace o veřejných zakázkách umožňující vyhledávání zakázek podle zadaných parametrů s možností registrace nových uživatelů a přihlášení registrovaných uživatelů a systém pro zasílání notifikací registrovaným uživatelům o nových zakázkách.

## **I. TEORETICKÁ ČÁST**

# 1 ZÁKON O VEŘEJNÝCH ZAKÁZKÁCH

Zadávání veřejných zakázek je v České republice upraveno zákonem 137/2006 Sb., o veřejných zakázkách, ve znění pozdějších předpisů (ZVZ). Zapracovává příslušné předpisy Evropské unie a upravuje především proces výběru nejvhodnějšího dodavatele veřejné zakázky. Důvodem pro úpravu jednotlivých kroků při zadávání veřejné zakázky je fakt, že veřejní zadavatelé spravující veřejné finanční prostředky nemají za cíl maximalizaci zisku a nemuseli by tak být dostatečně motivováni k zajištění efektivního využívání veřejných prostředků. [1]

## 1.1 Zadavatel veřejné zakázky

Zadavatelem veřejné zakázky je subjekt, který je povinen postupovat při zadávání veřejných zakázek podle ZVZ. Zákon rozlišuje 3 kategorie zadavatelů:

- veřejný zadavatel,
- dotovaný zadavatel,
- a sektorový zadavatel.

Na jednotlivé kategorie zadavatelů klade zákon odlišné požadavky (např. různé výše finančních limitů) a stanoví jim různé podmínky pro postup při zadávání veřejných zakázek (např. možnost použití omezených druhů zadávacího řízení). [1]

### 1.1.1 Veřejný zadavatel

Kategorie veřejný zadavatel je rozdělena do čtyř dílčích podkategorií. První podkategorii veřejného zadavatele představuje Česká republika. Česká republika zadává veřejné zakázky prostřednictvím svých organizačních složek, jako jsou ministerstva a jiné správní úřady státu, Ústavní soud, soudy, státní zastupitelství, Nejvyšší kontrolní úřad, Kancelář prezidenta republiky, Úřad vlády České republiky atd. a jiná zařízení, o kterých to stanoví zvláštní právní předpis.

Další podkategorii veřejného zadavatele jsou státní příspěvkové organizace hospodařící s majetkem státu.

Třetí podkategorii veřejného zadavatele představují územní samosprávné celky (obce, kraje) a jimi zřízené příspěvkové organizace.

Poslední podkategorií veřejného zadavatele je tzv. jiná právnická osoba, která zahrnuje subjekty, jež splňují specifické podmínky přesně definované ZVZ. Veřejným zadavatelem – jinou právnickou osobou jsou např. Česká televize, Český rozhlas, Česká národní banka, Všeobecná zdravotní pojišťovna a další. [1]

### **1.1.2 Dotovaný zadavatel**

Dotovaným zadavatelem je právnická nebo fyzická osoba, která zadává veřejnou zakázku hrazenou z více než 50% z veřejných zdrojů (např. dotací ze státního rozpočtu, rozpočtů obcí a krajů, grantů Evropské unie atd.), nebo peněžní prostředky poskytnuté na veřejnou zakázku z veřejných zdrojů přesáhnou částku 200 000 000 Kč. [1]

### **1.1.3 Sektorový zadavatel**

Tato kategorie zahrnuje tzv. přirozené monopoly. Jedná se o podnikatelské subjekty vykonávající tzv. relevantní činnost v odvětvích plynárenství, teplárenství, elektroenergetiky, vodárenství, veřejné dopravy a poštovních služeb. Nehospodaří veřejnými prostředky, ale využívají exkluzivních výhod udělovaných státem nebo jsou jím ovládány. [1]

### **1.1.4 Centrální zadavatel**

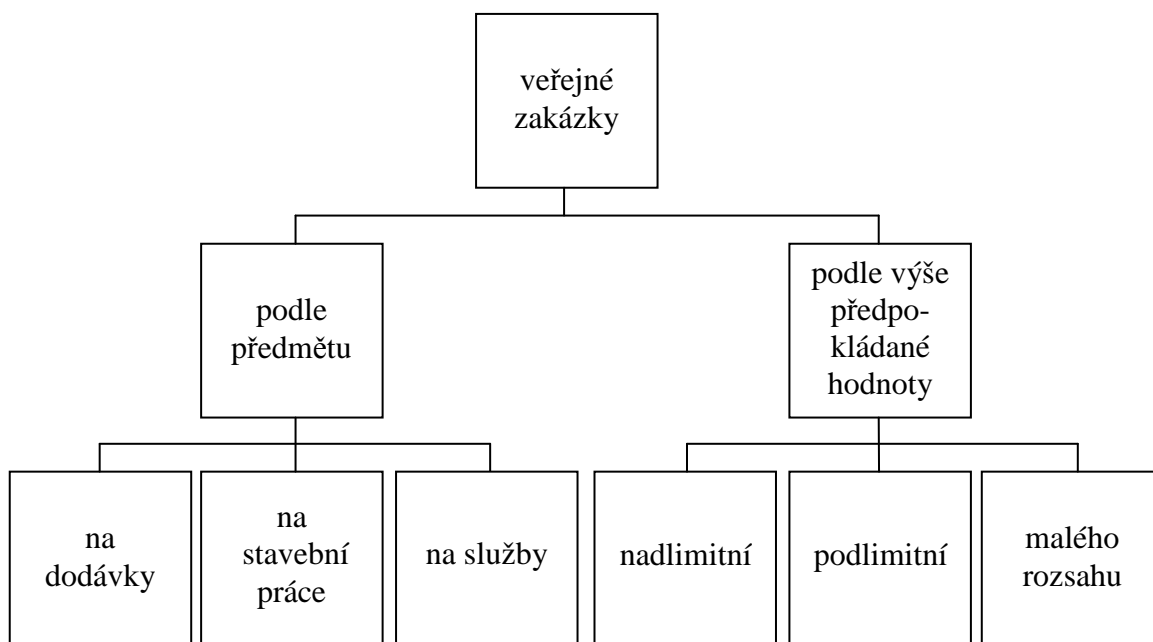
Centrálním zadavatelem může být výlučně veřejný zadavatel. Příkladem centrálního zadavatele může být např. stát nebo kraj disponující patřičnými lidskými zdroji, který bude provádět centralizované zadávání veřejných zakázek pro obce ve svém regionu. [1]

## **1.2 Veřejná zakázka**

ZVZ definuje veřejnou zakázku jako zakázku, která je realizována na základě písemné smlouvy mezi zadavatelem a jedním či více dodavateli, jejímž předmětem je úplatné poskytnutí dodávek či služeb nebo úplatné provedení stavebních prací.

ZVZ dělí veřejné zakázky podle předmětu na veřejné zakázky na dodávky, veřejné zakázky na služby a veřejné zakázky na stavební práce.

Podle výše jejich předpokládané hodnoty se dělí na nadlimitní veřejné zakázky, podlimitní veřejné zakázky a veřejné zakázky malého rozsahu. [1]



Obrázek 1 - Dělení veřejných zakázek [1]

### 1.2.1 Veřejná zakázka na dodávky

Veřejnou zakázkou na dodávky je veřejná zakázka, jejímž předmětem je pořízení zboží a poskytnutí služeb nebo stavebních prací spočívajících v umístění, montáži či uvedení takového zboží do provozu. [1]

### 1.2.2 Veřejná zakázka na stavební práce

K veřejným zakázkám na stavební práce přistupuje ZVZ ze dvou pohledů. Z pohledu jednotlivých činností a z pohledu celkového zhotovení stavby.

První přístup definuje veřejnou zakázku na stavební práce prostřednictvím stavebních prací, které se týkají některé z činností uvedených v příloze č. 3 ZVZ.

Druhý přístup definuje veřejnou zakázku na stavební práce z hlediska výsledku. Ve smyslu tohoto přístupu definice veřejné zakázky na stavební práce zahrnuje stavební nebo montážní práce, jejichž výsledkem je zhotovení stavby, která je jako celek schopna plnit samostatnou ekonomickou nebo technickou funkci. [1]

### 1.2.3 Veřejná zakázka na služby

Tato kategorie veřejných zakázek zahrnuje veškeré zakázky, které nelze zařadit pod definice veřejných zakázek na dodávky nebo stavební práce. [1]

#### 1.2.4 Veřejná zakázka v oblasti obrany nebo bezpečnosti

Do této kategorie spadají veřejné zakázky, jejichž předmět je definován v § 10a odst. 1 ZVZ. Jedná se např. o dodávky vojenského materiálu, dodávky citlivého materiálu a stavební práce, dodávky či služby s nimi související, citlivé stavební práce a další. [1]

#### 1.2.5 Veřejná zakázka podle výše předpokládané hodnoty

Podle výše předpokládané hodnoty ZVZ rozděluje veřejné zakázky do 3 kategorií:

*1. Nadlimitní veřejné zakázky*

Jedná se o veřejné zakázky, jejichž předpokládaná hodnota bez DPH dosáhne nejméně finančního limitu stanoveného prováděcím právním předpisem. Pro limity platné od 1.1.2014 viz tabulku 1.

*2. Podlimitní veřejné zakázky*

Jedná se o veřejné zakázky, jejichž předpokládaná hodnota činí nejméně 1 000 000 Kč bez DPH a nedosáhne finančního limitu nadlimitní veřejné zakázky.

*3. Veřejné zakázky malého rozsahu*

Jedná se o zakázky, jejichž předpokládaná hodnota nedosáhne 1 000 000 Kč bez DPH. Tyto zakázky nemá povinnost zadavatel zadávat podle ZVZ, ale při jejich zadávání musí dodržet zásady transparentnosti, rovného zacházení a nediskriminace. [1]

#### 1.2.6 Předpokládaná hodnota

Předpokládanou hodnotu veřejné zakázky stanovuje zadavatel a rozumí se jí výše peněžitého závazku vyplývajícího z plnění veřejné zakázky. Zákon stanoví poměrně striktní pravidla pro její určení tak, aby ji nebylo možno účelově podhodnotit. Stanovuje se před zahájením zadávacího řízení, aby bylo možno určit, do které z výše uvedených kategorií bude veřejná zakázka spadat a podle této kategorie pak zvolí typ zadávacího řízení. [1]

Tabulka 1 - Finanční limity pro jednotlivé kategorie zadavatelů, oblasti a druhy veřejných zakázek, případně kategorie dodávek a služeb stanovuje Nařízení vlády č. 77/2008 Sb., o stanovení finančních limitů pro účely zákona o zadávání veřejných zakázek (ve znění nařízení vlády č. 456/2013 Sb.) [2]

Zadavatel veřejné zakázky	Finanční limity v Kč
<b>1. Veřejné zakázky na dodávky</b>	
a) Česká republika <sup>1</sup> + státní příspěvkové organizace (§ 2 odst. 2, písm. a) a b) zákona)	3 395 000
b) Územní samosprávný celek nebo jím zřízená příspěvková organizace + jiná právnická osoba (§ 2 odst. 2 písm. c) a d) zákona)	5 244 000
c) Dotovaný zadavatel (§ 2 odst. 3 zákona)	
d) Ministerstvo obrany <sup>2</sup>	
e) Sektorový zadavatel (§ 2 odst. 6 zákona)	10 489 000
<b>2. Veřejné zakázky na služby</b>	
a) Česká republika + státní příspěvkové organizace <sup>3</sup>	3 395 000
b) Územní samosprávný celek nebo jím zřízená příspěvková organizace + jiná právnická osoba (§ 2 odst.2 písm. c) a d) zákona)	5 244 000
c) Česká republika + státní příspěvkové organizace <sup>4</sup>	
d) Dotovaný zadavatel (§ 2 odst. 3 zákona)	
e) Sektorový zadavatel (§ 2 odst. 6 zákona)	10 489 000
<b>3. Veřejné zakázky na stavební práce</b>	
Pro všechny veřejné zadavatele	131 402 000
1) pro Ministerstvo obrany tento finanční limit platí pouze pro zboží uvedené v příloze nařízení vlády 2) platí pouze pro zboží neuvedené v příloze nařízení vlády 3) mimo veřejné zakázky na služby uvedené v příloze č.1 k zákonu v kategorii 5 (jejichž klasifikace odpovídá referenčním číslům CPC 7524, 7525 a 7526) a v kategorii 8; služby uvedené v příloze č. 2 k zákonu 4) pouze pro veřejné zakázky na služby uvedené v příloze č.1 k zákonu v kategorii 5 (jejichž klasifikace odpovídá referenčním číslům CPC 7524, 7525 a 7526) a v kategorii 8; služby uvedené v příloze č. 2 k zákonu	

### 1.3 Zadávací řízení

Zadávací řízení upravuje proces, který vede k rozhodnutí zadavatele o výběru nejvhodnější nabídky a k uzavření smlouvy s vybraným uchazečem. ZVZ upravuje 6 druhů zadávacího řízení:

1. Otevřené řízení
2. Užší řízení
3. Jednací řízení s uveřejněním
4. Jednací řízení bez uveřejnění
5. Soutěžní dialog
6. Zjednodušené podlimitní řízení

ZVZ stanovuje podmínky použití jednotlivých druhů, upravuje průběh zadávacího řízení od okamžiku jeho zahájení, vymezuje lhůty v zadávacím řízení, upravuje zadávací dokumentaci a technické podmínky, stanovuje požadavky na kvalifikaci dodavatelů, náležitosti nabídky, upravuje podmínky otevírání obálek s nabídkami, požadavky na hodnotící komisi a posouzení a hodnocení nabídek a stanovuje podmínky ukončení zadávacího řízení. [1]

### 1.4 Uveřejňování

Uveřejňování je jedním ze základních nástrojů pro zajištění transparentnosti při zadávání veřejných zakázek. Za klíčové lze považovat uveřejnění oznámení o zahájení zadávacího řízení, ZVZ však pamatuje i na další uveřejňovací povinnosti v souvislosti s vyhlášením veřejných zakázek. Místem pro uveřejňování vyhlášení je Věstník veřejných zakázek, který je součástí Informačního systému o veřejných zakázkách podle § 157 ZVZ. Uveřejňování se provádí v elektronické podobě na internetové adrese <http://www.isvzus.cz>.

Vyhlášení týkající se nadlimitních veřejných zakázek se navíc povinně uveřejňují v Ústředním věstníku Evropské unie. [1]

### 1.5 Elektronická tržiště veřejné správy

Elektronické tržiště (e-tržiště) veřejné správy je webová aplikace sloužící k elektronickému zadávání veřejných zakázek ve stanovených zadávacích řízeních. Jedná se o plně elektro-

nický systém určený k rychlým operativním nákupům snadno standardizovatelných komodit. Elektronická tržiště podporují zadávání:

- veřejných zakázek malého rozsahu,
- podlimitních veřejných zakázek zadávaných ve zjednodušeném podlimitním řízení,
- veřejných zakázek zadávaných na základě rámcové smlouvy s více uchazeči dle § 92 odst. 3 ZVZ.

Hlavním smyslem zavedení systému e-tržišť je zvýšení efektivity a úspor při vynakládání veřejných finančních prostředků, podle zákona ZVZ. Systém e-tržišť vznikl v rámci realizace vládní strategie boje proti korupci, jejímž cílem je zjednodušit, standardizovat a zprůhlednit zadávání veřejných zakázek.

V rámci soustavy Národní infrastruktury pro elektronické zadávání veřejných zakázek (NIPEZ) bylo spuštěno celkem 5 e-tržišť, která jsou uvedena v seznamu elektronických tržišť vedeném správcem e-tržišť (Ministerstvo pro místní rozvoj ČR). Pro období od 1. 4. 2012 do 31. 3. 2017 vybralo MMR ČR v koncesním řízení na vytvoření a provoz e-tržišť pět provozovatelů: Českou poštu, Český trh, Sdružení eTenders, Syntaxit a Vortal.

Systém elektronických tržišť byl spuštěn k termínu 1. 5. 2012, povinnost používat elektronická tržiště pro ústřední orgány státní správy a jejich podřízené organizace platí od 1. 7. 2012. [1]

## 2 PORTLETOVÁ APLIKACE

V této části budou vysvětleny pojmy portál a portlet, budou představeny komponenty portálové infrastruktury, portálový server Liferay Portal, popsána jeho instalace a konfigurace v Eclipse IDE. Dále se bude týkat tvorby portletové třídy pomocí Java Portlet API a zmíní se o často používaných třídách z tohoto API.

### 2.1 Portál a portlet

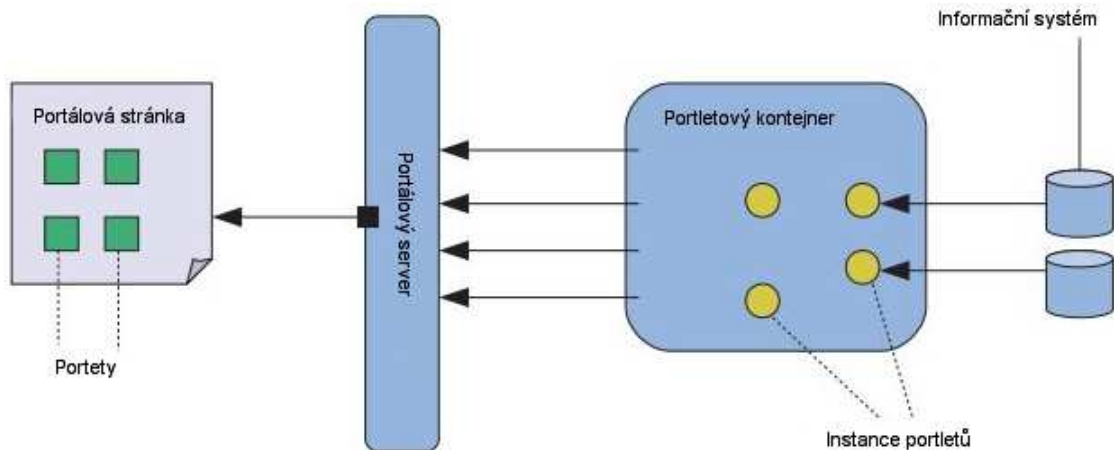
Portál lze charakterizovat jako kolekci webových miniaplikací - portletů. Portlet pak lze popsat jako okenní webovou aplikaci v rámci portálu, kde každé okno na webové stránce portálu představuje portlet. Portlety poskytují uživateli omezené informace a vlastnosti ve srovnání s původními webovými aplikacemi, jež často zastupují. Svá omezení však vyvažují tím, že uživateli nabízí přístup k obsahu z různých zdrojů na jednom místě, což může přispět k větší efektivitě jeho práce. Dalšími benefity mohou být funkce pro přizpůsobení portálu individuálním potřebám uživatele, např. funkce pro sestavení vlastního obsahu nebo funkce pro nastavení vzhledu portletu. [3]

### 2.2 Portálová infrastruktura

Portálová infrastruktura je tvořena portletovým kontejnerem a portálovým serverem. Funkce, které plní tyto komponenty, budou popsány v textu této kapitoly.

#### 2.2.1 Portletový kontejner

Portlet na portálové stránce je reprezentován instancí portletu uvnitř portletového kontejneru viz Obrázek 2. Portletový kontejner zodpovídá za řízení životního cyklu instance portletu a odesílání fragmentů generovaných portlety na portálový server pro jejich sestavení. Portletový kontejner volá metody životního cyklu instancí portletů a poskytuje jim běhové prostředí. [3]

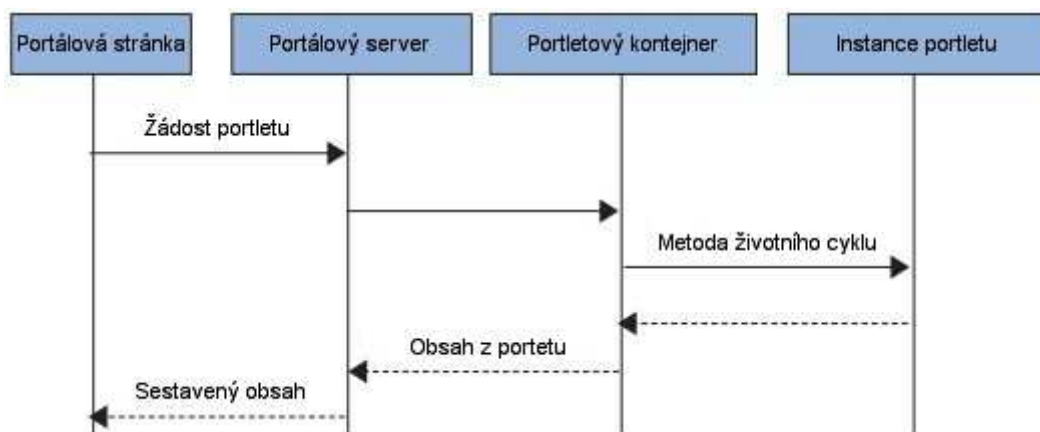


Obrázek 2 – Portálová infrastruktura [3]

### 2.2.2 Portálový server

Portálový server je zodpovědný za předložení požadavku přijatého od uživatele na portálové stránce portletovému kontejneru a za sestavení odpovědi vygenerovaných portlety k vytvoření portálové stránky. Lze jej tedy považovat za komponentu, která je umístěna mezi požadavky uživatele portálové stránky a portletovým kontejnerem.

Portletový kontejner není zodpovědný za vygenerování portálové stránky, ale předává portálovému serveru obsah vytvořený portlety, který tento obsah agreguje a zobrazí na portálové stránce, viz Obrázek 3. [3]



Obrázek 3 – Role portálového serveru a portletového kontejneru při zpracování požadavku portletu [3]

### 2.3 Portálový server Liferay Portal

Liferay Portal je lídrem v oblasti open-source enterprise portálových serverů, je založený na jazyce Java, poskytuje velkou škálu funkcí usnadňujících vytvoření plně funkčního por-

tálu v relativně krátkém čase (např. administraci, správu obsahu, vyhledávání, atd.). Pro vývoj portletové aplikace byl v této práci zvolen server Liferay Portal ve verzi 6.1 CE GA3.

Vytvoření portletu a jeho nasazení na portálový server lze zobecnit do následujících kroků:

1. Instalace portálového serveru a portletového kontejneru - baleny společně jako jedna součást.
2. Vytvoření portálové stránky, lze ji považovat jako prázdnou webovou stránku bez obsahu.
3. Nastavení IDE pro vytvoření projektu portletové aplikace.
4. Vytvoření třídy portletu obsahující logiku pro generování značkovacího fragmentu.
5. Vytvoření konfigurační souborů portletu pro registraci portletu na portálový server.
6. Vytvoření deployment deskriptoru portletu.
7. Zabalení portletu do souboru WAR pomocí sestavovacího nástroje jako je Ant nebo Maven.
8. Nasazení WAR soubor portletu na portálový server.
9. Vložení portletu na portálovou stránku. [3]

Podrobný popis všech zmíněných kroků přesahuje rámec této práce, v dalším textu této kapitoly proto bude soustředěna pozornost pouze na kroky týkající se instalace Liferay Portal, nastavení vývojového prostředí, tvorby třídy portletu, konfiguračních souborů pro registraci portletu na server, deployment deskriptoru a WAR souboru. V ostatních případech je odkázáno na příslušnou dokumentaci k Liferay Portal.

### 2.3.1 Instalace Liferay Portal 6.1 CE GA3

Liferay Portal lze získat jako balík s mnoha používanými aplikačními servery, v této práci je použit s aplikačním serverem Apache Tomcat 7.0.40. Balík lze stáhnout z webové stránky <http://www.liferay.com/downloads/liferay-portal/available-releases>.

Instalace se provede rozbalením staženého zip souboru do lokálního adresáře. Výsledkem bude jediný adresář, `liferay-portal-6.1.2-ce-ga3` (dále na něj bude odkazováno jako na `LIFERAY_HOME`).

Zde je na místě upozornit na dva důležité adresáře v `LIFERAY_HOME` `deploy` a `tomcat-7.0.40`:

- *deploy* se používá pro hot deployment portletů. Po nakopírování WAR souboru portletu do adresáře *deploy* při spuštěném serveru provede server hot deploy portletů.
- *tomcat-7.0.40* je adresář aplikačního serveru Tomcat, který obsahuje spouštěcí a zastavovací skripty aplikačního serveru.

Liferay Portal využívá interní databázi HSQLDB, která obsahuje počáteční konfigurační data, portálové uživatelské informace, portletové preference, atd. Je určena pouze pro vývojové účely, není určena pro produkční použití. Pro svůj běh vyžaduje Liferay Portal Javu SE5 nebo novější a proměnnou prostředí `JAVA_HOME` nastavenou do instalačního adresáře Java SE5. K domovské stránce Liferay Portal lze přistoupit prostřednictvím webového prohlížeče na <http://localhost:8080>. [3]

### 2.3.2 Konfigurace IDE Eclipse pro práci s Liferay Portal

Pro vývoj aplikací v této práci je použito IDE Eclipse verze Kepler Service Release 2. Dále v textu bude uvedeno, jak jej nastavit pro portletový projekt a popsána adresářová a souborová struktura tohoto projektu. Konfiguraci IDE Eclipse pro práci s Liferay Portal lze zobecnit do následujících kroků:

1. Vytvoření profilu serveru v Eclipse.
2. Specifikace instalačního adresáře serveru Tomcat.
3. Konfigurace serveru. [3]

#### 2.3.2.1 Vytvoření profilu serveru v Eclipse

Volba Window > Show View > Servers zobrazí servery dostupné v IDE Eclipse. Po kliknutí pravým tlačítkem myši ve zobrazení serverů se volbou New > Server otevře dialogové okno New Server. V tomto okně je třeba vybrat verzi serveru, který používá balík Liferay Portal (v případě této práce se jedná o Tomcat v7.0 Server). [3]

#### 2.3.2.2 Specifikace instalačního adresáře serveru Tomcat

Nastavení instalačního adresáře serveru Tomcat (dále na něj bude odkazováno jako na `TOMCAT_HOME`) je dostupné v dialogovém okně New Server Runtime Environment, které se zobrazí po kliknutí na odkaz Add... v dialogovém okně New Server. Zde je třeba zadat adresář s instalací serveru Tomcat v rámci instalace Liferay Portal a JDK pro jeho

spuštění. Definice serveru v IDE Eclipse je dokončena po kliknutí na tlačítko Finish v dialogovém okně New Server. Úspěšně vytvořený server se zobrazí ve view Servers. [3]

### 2.3.2.3 Konfigurace serveru

Vytvořený server je třeba nakonfigurovat, aby na něm mohl být projekt nasazen a laděn za běhu. Pro konfiguraci je třeba na něj kliknout pravým tlačítkem myši ve view Servers, ve zobrazeném menu zvolit položku Open, což zobrazí konfigurační stránku serveru. Zde, na kartě Server Locations zvolit možnost Use Tomcat Installation a do textového pole Deploy Path vepsat `webapps`.

Zajištění dostatku paměti pro běh serveru lze provést v dialogovém okně Edit Configuration zobrazeném po kliknutí na odkaz Open Launch Configuration na záložce General Information na konfigurační stránce serveru. V tomto okně lze upravit vlastnosti konfigurace spuštění serveru. Přidání argumentů virtuální paměti `-Xms128m -Xmx512m -XX:MaxPermSize=256m` do pole VM arguments zajistí, že server bude mít pro svůj běh dostatek paměti.

V záložce Classpath je třeba přidat do Bootstrap Entries soubory `servlet-api.jar`, `el-api.jar` a `jsp-api.jar` z adresáře `TOMCAT_HOME\lib` a do User Entries všechny JAR soubory z adresáře `TOMCAT_HOME\lib\ext`.

Posledním konfiguračním krokem nastavení serveru je zvýšení spouštěcího a vypínacího času serveru. Výchozí konfigurace v Eclipse očekává, že se server spustí do 45 sekund, je však bezpečnější časy spuštění a vypnutí serveru Tomcat zvýšit na 180 sekund resp. 120 sekund. Lze to provést na záložce Timeouts na konfigurační stránce serveru. [3]

### 2.3.2.4 Adresářová struktura portletového projektu

V této kapitole budou představeny typické adresáře, zdrojové a konfigurační soubory podílející se na portletové aplikaci. Jsou použitelné pro většinu portálových projektů s malými nebo vůbec žádnými změnami. Některé budou podrobněji popsány v dalším textu. Adresáře v portletovém projektu:

- `src` - obsahuje zdrojový kód projektu, zahrnuje třídy portletu a třídy utilit.
- `test` - prázdný adresář pro testovací třídy projektu.

- *build* - je vytvořen při prvním spuštění nástroje Ant pro projekt, obsahuje WAR soubor vygenerovaný při sestavení projektu pomocí nástroje Ant.
- *css* - obsahuje CSS soubory projektu k definici vzhledu portletů.  
*images* - obsahuje obrázky, které jsou používány portlety v projektu.
- *js* - obsahuje soubory JavaScriptu používané portletem. Portlety mohou také využívat knihovny JavaScriptu, jako DWR, Dojo, a jQuery, které jsou obvykle zabaleny v souboru JAR a přímo přístupné portletům.
- *lib* - obsahuje JAR soubory potřebné v době sestavení, ale nemusí být zabaleny s generovaným souborem WAR, např. JUnit a JAR soubory portletů. Soubor JUnit JAR je nutný ke spuštění unit testů v okamžiku sestavení a soubor JAR portletu je potřebný pro sestavení třídy portletu.
- *WEB-INF/jsp* - obsahuje JSP soubory použité v projektu.
- *WEB-INF/lib* - obsahuje JAR soubory potřebné pro portlety v okamžiku sestavení a za běhu.

Soubory v portletovém projektu:

- Soubor portletové třídy, vyžaduje třídy a rozhraní z Portlet 2.0 API, je tak třeba přidat soubor portlet.jar do classpath, třídou portletu se zabývá kapitola 2.5.
- *portlet.xml* – deployment descriptor portletu, konfigurační soubor, který definuje nastavení portletu, jako je request handler portletu, název portletu, podporované režimy, podporované národní prostředí, inicializační parametry, podporované typy MIME a zdrojový svazek (resource bundle) atd., je popsán v kapitole 2.4.

## 2.4 Deployment descriptor portletu

Webová aplikace je tvořena JSP stránkami, servlety, třídami Javy a deployment descriptor (popisovačem nasazení) webové aplikace - souborem web.xml. Portletová aplikace je webovou aplikací rozšířenou o portlety a portletový deployment descriptor - soubor portlet.xml. Tento soubor obsahuje informace pro nastavení portletů patřících dané portletové aplikaci. Příklad obsahu souboru portlet.xml zobrazuje výpis níže.

```
<portlet-app ... >  
  <portlet>  
    <portlet-name>TenderAggregator</portlet-name>
```

```
<display-name>Tender Aggregator Portlet</display-name>
<portlet-class>
  cz.utb.tenderaggregator.portlet.TenderAggregatorPortlet
</portlet-class>
<init-param>
  <name>defaultEmail</name>
  <value>@liferay.com</value>
</init-param>
<supports>
  <mime-type>text/html</mime-type>
  <portlet-mode>view</portlet-mode>
  <portlet-mode>edit</portlet-mode>
  <portlet-mode>help</portlet-mode>
</supports>
<resource-bundle>content.Language-ext</resource-bundle>
</portlet>
<container-runtime-option>
  <name>javax.portlet.actionScopedRequestAttributes</name>
  <value>true</value>
</container-runtime-option>
</portlet-app>
```

Význam jednotlivých elementů je následující, `<portlet-app>` je kořenovým elementem souboru `portlet.xml`. Portlet je deklarován elementem `<portlet>`. `portlet.xml` může obsahovat více deklarací portletu a všechny tyto portlety soubor tvoří část jediné portletové aplikace. `<portlet-name>` určuje název portletu, musí být v rámci portletové aplikace jedinečný. `<portlet-class>` definuje úplný kvalifikovaný název třídy portletu. `<init-param>` definuje inicializační parametry portletu. Obsahuje dva povinné subelementy `<name>` a `<value>`. Hodnota `<name>` musí být v rámci elementu `<portlet>` unikátní. Inicializační parametry jsou předány portletovým kontejnerem metodě portletu `init` a jsou použity instancí portletu k inicializaci instance. Elementem `<init-param>` lze předat i několik inicializačních parametrů. `<supports>` definuje režimy portletu a MIME typy podporované portletem. `<resource-bundle>` definuje svazek zdrojů použitý portletem

ke zobrazení popisků, zpráv atd. V elementu `<container-runtime-option>` lze provést nastavení kontejneru tak, aby za běhu poskytl portletům další funkce. Nastavením hodnoty `true` volbě `javax.portlet.actionScopedRequestAttributes` je docíleno toho, že atributy nastavené v objektu `ActionRequest` (během zpracování akce), budou následně k dispozici objektu `RenderRequest` (během renderování obsahu). Metoda `init`, objekty `ActionRequest`, `RenderRequest` a další třídy a rozhraní z Portlet 2.0 API budou popsány v kapitole 2.5. [3]

## 2.5 Tvorba portletové třídy pomocí Java Portlet API

V současné době existují dvě verze specifikací Java Portlet API, specifikace 1.0 (popsaná JSR 168) a 2.0 (popsaná JSR 286). Specifikace 2.0 rozšiřuje specifikaci 1.0 o další požadované vlastnosti portletů, jako je obsluha zdrojů, meziportletová komunikace a portletové filtry. Obě specifikace jsou podporovány většinou portletových kontejnerů a specifikace 2.0 je zpětně kompatibilní se specifikací 1.0.

Portletová třída musí implementovat přímo nebo nepřímo rozhraní `javax.portlet.Portlet`. Portlet API obsahuje abstraktní třídu `GenericPortlet`, která rozhraní `Portlet` implementuje a poskytuje výchozí implementaci jeho metod. Portletová třída se tak obvykle vytváří zděděním třídy `GenericPortlet` a přepsáním jejích metod k zajištění specifické implementace. Portletová třída vyžaduje také další třídy a rozhraní z Portlet 2.0 API, je tak třeba přidat do classpath soubor `portlet.jar`. [3]

Pro popis tvorby portletové třídy je důležité objasnit životní cyklus portletu a jádro portletové technologie - portletových URL adresy. Kapitola 2.5.1 tak bude věnována životnímu cyklu portletu, kapitola 2.5.2 portletovým URL adresám.

### 2.5.1 Životní cyklus portletu

Portlety jsou webové komponenty, jejichž životní cykly jsou řízeny portletovými kontejnery. Portletový kontejner je zodpovědný za načtení, vytvoření instance, inicializaci, směrování požadavků k instanci portletu a nakonec za zrušení instance portletu.

Zpracování požadavků v portletech prochází některou z fází (případně více fázemi) zpracování: *render*, *action*, *resource*, *event*. Každá tato fáze má přesně definovanou roli, viz Tabulka 2 – *Fáze zpracování požadavků v portletu*.

Tabulka 2 – Fáze zpracování požadavků v portletu [3]

Fáze zpracování požadavků	Účel
<b>Render</b>	Vygenerování obsahu, např. HTML tagů.
<b>Action</b>	Provedení stav měnící funkce, např. aktualizace databáze.
<b>Resource</b>	Obsluha zdroje.
<b>Event</b>	Reakce na události přijaté portletem.

Pokud portlet, např., přijme požadavek k provedení akce (action request) je na něj aplikována fáze action následovaná vždy fází render, pokud je přijat požadavek na vygenerování obsahu (render request), aplikuje se pouze fáze render.

Každá fáze je mapována na metodu ve třídě portletu, která zpracovává požadavky přijaté v této fázi. Např. fáze render je mapována na metodu `render`, fáze action na metodu `processAction`. [3]

### 2.5.2 Portletové URL

V portletové aplikaci nelze přímo zadat URL adresu, na kterou portlet mapuje, ale portletový kontejner poskytuje objekty, které mohou portlety použít k vytvoření tzv. na sebe odkazujících se (self-referencing) URL adres. Tyto URL adresy odkazují na portlet, který je vytvořil a nazývají se *portletové URL*. Portletové URL se vztahují k typům portletových požadavků. Existují tři typy portletových URL:

- *Renderovací URL (render URL)* – používá se k požadavku na instanci portletu, aby vygenerovala tagy (HTML, XML, WML) v závislosti na jejím aktuálním stavu. Požadavek odeslaný portletu renderovací URL je označován jako *render request*.
- *Akční URL (action URL)* – používá se pro zpracování akce, která má za následek změnu stavu na serveru. Požadavek odeslaný portletu akční URL je označován jako *action request*.
- *Zdrojová URL (resource URL)* – používá se k vykreslení obsahu nebo načtení zdrojů (jako soubory s obrázky). Může být použita pro aktualizaci stavu aplikace. Požadavek odeslaný portletu zdrojovou URL je označován jako *resource request*. [3]

### 2.5.3 Tvorba portletové třídy zděděním třídy `GenericPortlet`

Jak již bylo zmíněno, abstraktní třída `GenericPortlet` z Portlet API implementuje rozhraní `javax.portlet.Portlet`, které musí implementovat portletová třída. Portletovou třídu lze pak vytvořit zděděním třídy `GenericPortlet` a přepsáním jejích metod. Tento způsob tvorby portletové třídy byl zvolen i v této práci.

Metodami definovanými rozhraním `Portlet` jsou `init`, `processAction`, `render` a `destroy`.

#### **Metoda `init`**

Metodu `init` volá portletový kontejner po té, co vytvořil instanci portletu. Má za úkol inicializovat zdroje portletu před přijetím požadavku (např. připojení k databázi, čtení konfiguračních dat ze souboru atd.), které budou potřeba pro jeho zpracování. Tyto zdroje zůstávají po dobu životnosti instance portletu obvykle nezměněny. Signatura metody `init` je následující:

```
public void init(PortletConfig config) throws PortletException
```

Objekt `javax.portlet.PortletConfig` je poskytnut portletovým kontejnerem a poskytuje portletům přístup k jejich běhovému prostředí a deployment deskriptoru (`portlet.xml`).

Následující fragment kódu ukazuje využití metody `init` pro získání hodnoty inicializačního parametru `defaultEmail` z deployment deskriptoru:

```
private String defaultEmail;  
public void init(PortletConfig config) throws PortletException {  
    defaultEmail=config.getInitParameter("defaultEmail");  
}
```

[3]

#### **Metoda `render`**

Metodu `render` volá portletový kontejner, když přijme render request pro instanci portletu. Je zodpovědná za generování obsahu portletu. Signatura metody `render` je následující:

```
void render(RenderRequest request, RenderResponse response) throws  
PortletException, IOException
```

Objekty `RenderRequest` a `RenderResponse` jsou vytvořeny portletovým kontejnerem v době požadavku na zpracování. Objekt `RenderRequest` poskytuje instanci portletu přístup k datům zaslaným v rámci `render request`. V metodě `render` portlet obvykle čte informace ze zdroje dat a k zápisu obsahu používá objekt `RenderResponse`.

Třída `GenericPortlet` poskytuje defaultní implementaci metody `render`, která deleguje její volání na metody `doView`, `doEdit` a `doHelp` v závislosti na režimu portletu, umožňuje však také použití anotace `@RenderMode` pro tvorbu renderovacích metod s vlastním pojmenováním. Následující fragment kódu ukazuje použití anotace `@RenderMode` k označení metody `sayHello` jako renderovací metody pro režim portletu `VIEW`:

```
@RenderMode(name = "VIEW")  
public void sayHello(RenderRequest request, RenderResponse response)  
throws PortletException, IOException {  
    ...  
}  
[3]
```

### **Metoda `processAction`**

Metoda `processAction` je volána v odpovědi na `action request`. Tato metoda reprezentuje akci uživatele, která vede k změně stavu na serveru, např. odeslání formuláře. Signatura metody `processAction` je následující:

```
void processAction(ActionRequest request, ActionResponse response)  
throws PortletException, IOException
```

Objekty `ActionRequest` a `ActionResponse` jsou poskytnuty portletovým kontejnerem v době požadavku na zpracování. `ActionRequest` poskytuje instanci portletu data `action requestu` a `ActionResponse` využívá portlet k odpovědi na `action request`. Touto odpo-

vědí může být např. přesměrování na jinou URL. Metoda `processAction` není určena ke generování obsahu pro portlet a jakýkoli pokus o to je kontejnerem portletu ignorován.

Třída `GenericPortlet` poskytuje defaultní implementaci metody `processAction`, která odesílá action request metodě anotované `@ProcessAction`. Následující fragment kódu ukazuje použití anotace `@ProcessAction`:

```
@ProcessAction(name = "registerUserAction") public void registerUser(ActionRequest request, ActionResponse response) throws PortletException, IOException {  
    ...  
}
```

Anotace `@ProcessAction` označuje metodu `registerUser` jako akční metodu. Atribut `name` je hodnota parametru požadavku s názvem `javax.portlet.action` v objektu `ActionRequest`. Pokud `ActionRequest` obsahuje parametr `javax.portlet.action`, jehož hodnota odpovídá hodnotě zadané pro atribut `name` z anotace akční metody `@ProcessAction`, je tato metoda zavolána. V případě kódu výše bude zavolána metoda `registerUser`, pokud bude hodnota parametru požadavku `javax.portlet.action` v objektu `ActionRequest` rovna `registerUserAction`. [3]

### ***Metoda destroy***

Metoda `destroy` je volána portletovým kontejnerem před odstraněním instance portletu z paměti. Je to úklidová metoda portletu, uvolňuje zdroje držené portletem (např. databázové připojení nebo EJB reference) nebo ukládá jejich stav do trvalého úložiště. Signatura metody `destroy` je následující:

```
void destroy()
```

Může být volána kontejnerem kdykoliv v závislosti na jeho implementaci. Portletový kontejner zabezpečí, že všechny aktuálně zpracovávané požadavky portletu byly dokončeny před zničením jeho instance. [3]

### 2.5.4 Tvorba portletových URL

Portletové URL jsou tvořeny programově pomocí objektu `RenderResponse`, který je k dispozici metodě `render`.

#### *Renderovací URL*

Objekt `RenderResponse` disponuje metodou `createRenderURL` která vrací objekt `PortletURL`. Tento objekt poskytuje metody pro přidávání parametrů, nastavení stavu okna a režimu portletu. Příklad vytvoření renderovací URL ukazuje fragment kódu níže.

```
@RenderMode(name="VIEW")
public void renderForm(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {
    ...
    PortletURL renderUrl = response.createRenderURL();
    ...
}
```

[3]

#### *Akční URL*

Pro vytvoření akční URL poskytuje objekt `RenderResponse` metodu `createActionURL`. Tato metoda vrací objekt `PortletURL`, který disponuje metodami pro přidávání parametrů, nastavení stavu okna a režimu portletu. Tyto parametry mohou být použity akční metodou ke specifikaci akce, která má být provedena. Příklad vytvoření akční URL a nastavení jejího akčního parametru ukazuje následující fragment kódu.

```
@RenderMode(name="VIEW")
public void renderForm(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {
    ...
    PortletURL registerUserActionUrl = response.createActionURL();
    registerUserActionUrl.setParameter(ActionRequest.ACTION_NAME,
```

```
    "registerUserAction");  
    ...  
}
```

V předchozím kódu má konstanta `ACTION_NAME` z objektu `ActionRequest` hodnotu `javax.portlet.action`. Hodnota parametru `javax.portlet.action` je užitečná pro volání metod anotovaných `@ProcessAction`. Pokud bude `registerUserActionUrl` vyrenderována jako hypertextový odkaz nebo jako hodnota atributu `action` HTML tagu `form`, po kliknutí na odkaz nebo odeslání formuláře bude provedena metoda `processAction` (nebo metoda anotovaná `@ProcessAction`, s atributem `name` o hodnotě `registerUserAction`). V případě kódu níže bude akční metoda `registerUser` zavolána tehdy, když bude mít parametr požadavku `ActionRequest.ACTION_NAME` hodnotu `registerUserAction`.

```
@ProcessAction(name="registerUserAction")  
public void registerUser(ActionRequest request, ActionResponse response)  
    throws PortletException, IOException {  
    ...  
}
```

[3]

### 2.5.5 Třídy a rozhraní z Portlet API

V této kapitole budou popsány často používané třídy a rozhraní z Portlet API. Některé jsou použity při tvorbě portletové třídy v portletové aplikaci vyvíjené v této práci.

#### ***Třída PortletConfig***

Instance třídy `PortletConfig` umožňuje přístup portletu k nastavením provedeným v deployment descriptoru, jako jsou podporované režimy portletu, podporovaná národní prostředí, zdrojové svazky, inicializační parametry, runtimeové možnosti kontejneru, atd. [3]

#### ***Třída PortletContext***

Koncepce portletového kontextu poskytuje přístup k prostředí, ve kterém portlet běží. Tímto prostředím je portletová aplikace, ke které portlet patří a portletový kontejner, ve kterém

portlet běží. S portletovou aplikací je spojena jedna instance `PortletContext` a portlety ji používají pro přístup k prostředkům v této aplikaci, kterými mohou být JSP, servlet, nebo jiný portlet.

Třída `GenericPortlet` poskytuje metodu `getPortletContext`, která vrací objekt `PortletContext` spojený s aktuální portletovou aplikací. Objekt `PortletContext` lze také získat pomocí objektů `PortletSession` nebo `PortletConfig`. [3]

### Třída `PortletRequestDispatcher`

Obsah portletu může být vygenerován jeho přímým napsáním do objektu `PrintWriter` nebo v případě, že je značkovací fragment generovaný portletem složitý, je lepší tuto zodpovědnost přenést na JSP stránku, servlet nebo HTML stránku pomocí objektu `PortletRequestDispatcher`.

Objekt `PortletRequestDispatcher` lze získat od objektu `PortletContext` některou z těchto metod:

- `getRequestDispatcher(String path)` - Parametr `path` je cesta k JSP stránce, servletu, nebo HTML stránce v portletové aplikaci. `path` je relativní ke kořenu kontextu portletové aplikace a musí začínat lomítkem (/).
- `getNamedDispatcher(String name)` - Parametr `name` je název servletu (definovaného ve `web.xml`) v rámci portletové aplikace.

Objekt `PortletContext` není možné použít k získání objektu `PortletRequestDispatcher`, který se odkazuje na zdroje (např. JSP, servlety a HTML) mimo portletovou aplikaci.

`PortletRequestDispatcher` definuje následující metody:

- `include` - začleňuje obsah generovaný ze zdroje do odpovědi portletu. Existují dvě verze metody `include`, jednu z nich je míněna čistě pro zpětnou kompatibilitu se specifikací Portlet 1.0, jako parametry metody přijímá objekty `RenderRequest` a `RenderResponse`.
- `forward` - metoda předává požadavek na generování obsahu jinému zdroji.

Následující fragment kódu ukazuje, jak metoda `renderHelp` využívá `PortletRequestDispatcher` k začlenění obsahu vygenerovaného stránkou `help.jsp`:

```
@RenderMode(name="HELP")
public void renderHelp(RenderRequest request,
    RenderResponse response)throws PortletException, IOExcepti-
on {
    getPortletContext().getRequestDispatcher("/WEB-
INF/jsp/help.jsp").
    include(request,response);
}
```

V předchozím kódu jsou objekty `request` i `response` předány začleněné JSP stránce, což umožňuje JSP stránce přistoupit k atributům asociovaným s požadavkem. [3]

### ***Třída RenderResponse***

Instanci třídy `RenderResponse` lze použít např. k vytvoření action (akční) a render (renderovací) adresy URL, které lze pak předat JSP stránce pomocí `request` atributů. Tyto action a render URL adresy jsou součástí obsahu generovaného stránkou JSP, kterým může být hypertextový odkaz nebo mohou být hodnotou atributu `action` HTML tagu `form`. [3]

### ***Třída ResourceBundle***

Instance třídy `ResourceBundle` reprezentuje zdrojový svazek specifikovaný v elementu `<resource-bundle>` deployment descriptoru portletu. Metoda `getLocale` z rozhraní `PortletRequest` vrací objekt `java.util.Locale` identifikující locale uživatele. `ActionRequest` a `RenderRequest` implementují `PortletRequest`, takže metoda `processAction` i renderovací metody portletu mohou zjistit locale uživatele. Objekt `ResourceBundle` lze získat zavoláním metody `getResourceBundle(java.util.Locale)` objektu `PortletConfig`. Následující fragment kódu ukazuje, jak lze získat chybovou zprávu, která odpovídá klíči `email.errorMsg.missing` ze zdrojového svazku:

```
ResourceBundle bundle =
    getPortletConfig().getResourceBundle(request.getLocale());
String message = bundle.getString("email.errorMsg.missing");
```

Metodu `getPortletConfig` vracející objekt `PortletConfig` má k dispozici třída `GenericPortlet`. [3]

### **Rozhraní `PortletSession`**

Rozhraní `PortletSession` poskytuje dva různé rozsahy k uložení atributů do session identifikované konstantami `APPLICATION_SCOPE` a `PORTLET_SCOPE`. Pokud portlet uloží atributy v `PORTLET_SCOPE`, budou dostupné pouze jemu, pokud je uloží v `APPLICATION_SCOPE`, budou dostupné i ostatní webovým komponentám v dané portletové aplikaci. [3]

## **2.6 Framework Spring**

Spring je open-source modulární framework pro vývoj J2EE aplikací. Modul může být v aplikaci použit nezávisle na dalších modulech Springu. Jedinou závislost, kterou moduly mají, je závislost na jádře modulů frameworku Spring, které poskytuje základní funkce frameworku, jako dependency injection (přiřazení závislostí), funkce IoC kontejneru, atd. Tato modularita ve frameworku Spring umožňuje zvolit modul, který vyhovuje požadavkům aplikace bez nutnosti nést zátěž ostatních modulů. [4]

### **2.6.1 Dependency injection a aplikační kontext**

Při použití dependency injection (DI), jsou závislosti mezi objekty aplikace specifikovány v externím konfiguračním souboru (např. XML soubor aplikačního kontextu) nebo v rámci zdrojového kódu (pomocí anotací). Tyto závislosti jsou přiřazeny do závislých objektů IoC (Inversion of Control) kontejnerem, který je zodpovědný za vytváření a správu objektů aplikace a za přiřazování závislosti na základě konfiguračních informací. Objekty vytvořené kontejnerem jsou nazývány beany. DI řeší vlastní způsob přiřazování závislostí mezi objekty. Základními způsoby realizace DI jsou konstruktory a setter metody. [4]

### **2.6.2 Anotacemi řízený vývoj**

Anotace definují metadata (data o datech), bývají uvedeny ve zdrojovém kódu aplikace. Lze je považovat za instrukce překladači a běhovému prostředí o úkolech, které by měly být provedeny při kompilaci nebo při běhu. Podpora anotací byla přidána do Javy v Javě 5. Jsou reprezentovány znakem `@` následovaným názvem anotace, např. `@Controller`. Lze je využít jako nástroje k provedení některé z následujících činností:

- Vytvoření často používaného kódu.
- Napsání konfiguračních informací v konfiguračních souborech.
- Poskytnutí informací o kódu nástrojům aplikace.
- Modifikaci chování programu v době běhu. [4]

### 2.6.3 Funkce classpath-scanning

Funkce Springu classpath-scanning skenuje komponenty, které jsou anotovány anotacemi `@Component`, `@Service`, `@Controller` nebo `@Repository` a automaticky je zaregistruje do aplikačního kontextu, to znamená, že zde není nutno explicitně deklarovat beanu.

Pro nastavení funkce classpath-scanning musíte použít element `component-scan` ze schéma Springu `spring-context`. Použití elementu `component-scan` v aplikačním kontextu portletové aplikace této práce:

```
<context:component-scan base-package="cz.utb.tenderaggregator" />
```

Zde je `context` jmenný prostor s elementy dostupnými ze schématu `spring-context.xsd` v XML souboru aplikačního kontextu. Atribut `base-package` vyjmenovává čárkou oddělená jména balíčků, ve kterých bude kontejner Springu hledat Spring komponenty. V tomto případě element `component-scan` prohledá všechny třídy uvnitř balíčku `cz.utb.tenderaggregator` a jeho podbalíčky a automaticky zaregistruje třídy anotované `@Component`, `@Repository`, `@Controller` nebo `@Service` do aplikačního kontextu. [4]

### 2.6.4 Dependency injection anotací `@Autowired`

Anotace `@Autowired` se používá pro proměnnou, setter metodu nebo konstruktor jako instrukce pro Spring kontejner aby přiřadil závislost. Např. v kódu níže třída `AddBookController` používá anotaci `@Autowired` k instrukci Spring kontejneru, aby jí přiřadil beanu `BookService`.

```
import org.springframework.beans.factory.annotation.Autowired;
@Controller(value="addBookController")
public class AddBookController {
    @Autowired
    private BookService bookService;
```

```
    ...  
}
```

## 2.7 Framework Hibernate

Framework Hibernate ORM (objektově relační mapování) definuje metadaty řízený přístup k mapování Java tříd na tabulky relační databáze a poskytuje API persistence, které spravuje persistenci zbavující vývojáře zodpovědnosti za psaní SQL a podpůrného kódu. Framework Hibernate je zodpovědný za transparentní konverzi Java objektů na relační data a naopak pomocí uživatelem definovaných metadat. Persistentní Java třídu lze považovat jako tabulku relační databáze, atributy třídy reprezentují sloupce v tabulce a každá instance Java třídy představuje záznam v této tabulce. Pro mapování lze vytvořit XML soubor nebo použít anotací. [5]

### 2.7.1 Java Persistence API (JPA)

Java Persistence API definuje standard ORM a API persistence pro Java objekty. Je inspirováno frameworky jako Hibernate, TopLink a JDO. Hibernate implementuje JPA, lze tak při použití frameworku Hibernate, využít všech funkcí JPA společně s dodatečnými funkcemi Hibernate. Hibernate lze považovat za nadmnožinu JPA.

Výpis níže ukazuje persistentní třídu Tender jak je namapována pomocí anotací na tabulku tender.

```
import javax.persistence.*;  
  
@Entity(name="MyTender")  
@Table(name="tender")  
public class Tender {  
    @Id  
    @Column(name="id")  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
  
    @Column(name="name")  
    private String name;  
  
    public Tender() {  
        ...  
    }  
    ...  
}
```

Anotace `@Entity` indikuje, že třída `Tender` reprezentuje persistentní entitu – objekty třídy `Tender` jsou uloženy v relační databázi. Volitelný element `name` anotace `@Entity` přiřazuje unikátní jméno entitě, které musí být v aplikaci unikátní. Pokud není element `name` specifikován, defaultním jménem entity je nekvalifikované jméno třídy. Element `name` anotace `@Table` specifikuje tabulku, do které jsou persistentní atributy entity ukládány. Anotace `@Id` identifikuje persistentní atribut entity, který hraje roli primárního klíče v databázové tabulce. Anotace `@Column` specifikuje sloupec v tabulce, do kterého je namapován atribut `id` z entity `Tender`. Anotace `@GeneratedValue` je použita k definici strategie následované poskytovatelem persistence, kterým je v tomto případě Hibernate, pro generování hodnoty primárního klíče. Element `strategy` identifikuje strategii, kterou se musí poskytovatel persistence řídit při vygenerování hodnoty primárního klíče.

Ve výpise výše anotace `@Entity` specifikuje jméno entity `Tender` jako `MyTender`, což je její jméno zaregistrované v poskytovateli persistence. Anotace `@GeneratedValue` specifikuje strategii generování primárního klíče jako  `GenerationType.AUTO`, což znamená, že poskytovatel persistence je zodpovědný za výběr odpovídající strategie k vytváření hodnot primárního klíče. [5]

## 2.8 Integrace portletové aplikace s databází

Existuje více způsobů, jak integrovat portlety s databází. V portletové aplikaci této práce bylo pro tento účel využito modulu JDBC frameworku Spring a frameworku Hibernate ORM (objektově relační mapování) usnadňující vývoj aplikace v oblasti připojení a interakce s databází.

Modul Springu JDBC obstarává správu databázových připojení a nabízí třídy přijímající objekty typu `javax.sql.DataSource`, zodpovědné za správu objektů `Connection`, `Statement` a `ResultSet`.

Framework Hibernate ORM definuje metadaty řízený přístup k mapování Java tříd na tabulky relační databáze a poskytuje API persistence, které spravuje persistenci zbavující vývojáře zodpovědnosti za psaní SQL a podpůrného kódu. [3]

### 2.8.1 Definice datového zdroje

Definici objektu `javax.sql.DataSource`, reprezentující datový zdroj ke kterému bude portlet přistupovat, lze provést dvěma způsoby:

- Jako beanu `javax.sql.DataSource` v souboru aplikačního kontextu `applicationContext.xml`.
- Navázat jej ke jménu JNDI v aplikačním serveru.

V portletové aplikaci této práce bylo využito definice datového zdroje jeho navázáním ke jménu JNDI v aplikačním serveru a získáním v aplikačním kontextu. Úprava konfiguračních souborů `server.xml` a `context.xml` aplikačního serveru Tomcat 7.0.40 proto bude popsána v následujícím textu.

Vytvoření JNDI navázání datového v souboru `server.xml` uvádí následující výpis.

```
< GlobalNamingResources >
...
<Resource name="jdbc/TenderMiningDB"
  auth="Container"
  type="javax.sql.DataSource" username="postgres"
password="password"
  driverClassName="org.postgresql.Driver"
  factory="org.apache.commons.dbcp2.BasicDataSourceFactory"
  url="jdbc:postgresql://localhost:5432/TenderMiningDB"
  maxIdle="2"
  maxActive="5" />
...
</GlobalNamingResources>
```

Element `<Resource>` musí být definován uvnitř elementu `<GlobalNamingResources>`. Definuje datový zdroj navazovaný k JNDI, atribut `name` určuje název, se kterým zdroj dat je vázán k JNDI. `BasicDataSourceFactory` představuje JNDI factory, která vytváří `BasicDataSource`. Implementuje rozhraní `javax.naming.spi.ObjectFactory`. Atribut `maxActive` udává maximální počet připojení, které může connection pool poskytnout v daném okamžiku. Atribut `maxIdle` indikuje maximální počet připojení, které mohou zůstat v nečinnosti v poolu.

Pro nastavení dostupnosti datového zdroje vázanému k JNDI pro portlet, je třeba nakonfigurovat element `<ResourceLink>` v souboru `context.xml`, jak je znázorněno zde:

```
<Context>
...
<ResourceLink type="javax.sql.DataSource"
  name="jdbc/TenderMiningDB"
  global="jdbc/TenderMiningDB"/>
```

```
</Context>
```

Atribut `name` v elementu `<ResourceLink>` je název JNDI, s nímž je datový zdroj `jdbc/TenderMiningDB` v globálním JNDI kontextu k dispozici portletové aplikaci. Atribut `global` identifikuje globální k JNDI vázaný datový zdroj, který je definován v souboru `server.xml`.

Framework Spring poskytuje schéma `jee`, které usnadňuje získat zdroje z JNDI. Výpis níže ukazuje využití schématu Springu `jee` k získání datového zdroje vázaného k JNDI a jeho zpřístupnění jako beanu Springu v aplikačním kontextu.

```
<beans ...
  xmlns:jee="http://www.springframework.org/schema/jee"
  xsi:schemaLocation="
    ...
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-
3.0.xsd
  ">
  ...
  <jee:jndi-lookup
    jndi-name="java:comp/env/jdbc/TenderMiningDB"
    id="dataSource" />
  ...
</beans>
```

XML soubor aplikačního kontextu odkazuje na `jee` schéma Springu. Element `<jndi-lookup>` vyhledá JNDI zdroje (v tomto případě JDBC). Atribut `jndi-name` specifikuje název JNDI zdroje, který má být získán. Atribut `id` přiřazuje jedinečný název datovému zdroji načtenému z JNDI, který lze použít jako `id` beanu v aplikačním kontextu. [3]

### 2.8.2 Přístup k databázi pomocí Spring s Hibernate

Spring lze použít pro konfiguraci `SessionFactory` jako beanu v aplikačním kontextu a nechat jej tak spravovat transakce. K provedení databázových operací je potřeba objektu `Session` získaného ze `SessionFactory`, proto třídy DAO musí mít přístup k `SessionFactory`. Spring poskytuje beanu `AnnotationSessionFactoryBean`, kterou lze použít k vytvoření objektu `SessionFactory` a injektovat jej do DAO tříd pomocí `dependency injection`. Následující výpis ukazuje XML soubor aplikačního kontextu, který definuje beanu `AnnotationSessionFactoryBean`.

```
<beans ...>
  ...
  <context:component-scan base-package=
    "cz.utb.tenderaggregator" />
  <jee:jndi-lookup jndi-name="java:comp/env/jdbc/TenderMiningDB"
    id="dataSource" />

  <bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="annotatedClasses">
      <list>
        <value>cz.utb.tenderaggregator.domain.Region</value>
        <value>cz.utb.tenderaggregator.domain.Tender</value>
        ...
      </list>
    </property>
  </bean>
  <tx:annotation-driven transaction-manager="txManager" />
  <bean id="txManager"
    class="org.springframework.orm.hibernate3.HibernateTransaction
    Manager">
    <property name="sessionFactory" ref="sessionFactory" />
  </bean>
</beans>
```

Definice beany `AnnotationSessionFactoryBean` v aplikačním kontextu, která je zodpovědná za vytvoření `Hibernate SessionFactory`. Informace o datovém zdroji pro `AnnotationSessionFactoryBean` jsou nastaveny pomocí vlastnosti `dataSource`. Vlastnost beany `AnnotationSessionFactoryBean` `annotatedClasses` specifikuje persistentní entity, které jsou asociovány se `SessionFactory`. Element `<annotation-driven>` z transakčního schématu Springu specifikuje, že je aplikována transakce na metody, které jsou anotovány anotací `@Transactional`. Atribut `transaction-manager` určuje transakčního manažera, který bude použit pro správu transakcí.

Následující výpis ukazuje třídu `HibernateTenderDao`, která poskytuje logiku pro přístup k datům pomocí `SessionFactory`.

```
import org.hibernate.Query;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
```

```
import cz.utb.tenderaggregator.domain.Tender;

@Repository("tenderDao")
public class HibernateTenderDao extends HibernateGeneric-
Dao<Tender, Long> implements TenderDao {

    @Autowired
    private SessionFactory sessionFactory;

    public Tender getTender(Long id) {
        String sql = "from Tender as tender where id= :id";
        Query query = sessionFactory.getCurrentSession().createQuery(sql);
        query.setParameter("id", id);
        Tender tender = (Tender) query.uniqueResult();
        return tender;
    }
    ...
}
```

Anotace `@Repository` je použita ke specifikaci, že třída `HibernateTenderDao` je komponenta Springu, bude automaticky registrována v aplikačním kontextu, protože je použita funkce Springu `classpath-scanning` (element `<context:component-scan base-package="cz.utb.tenderaggregator" />`). Beana `AnnotationSessionFactoryBean` definovaná v aplikačním kontextu je automaticky připojena do objektu `HibernateTenderDao`. Přestože `AnnotationSessionFactoryBean` je třída Springu, implementuje rozhraní `SessionFactory` z Hibernate, což znamená, že může být přiřazena proměnné typu `SessionFactory`.

Metoda `createQuery` ze třídy `Session` je použita k vytvoření objektu `Query`. Metoda `uniqueResult()` objektu `Query` provede dotaz k získání objektu `Tender` z tabulky `tender`. [3]

### 3 JAVA HTML PARSER JSOUP

`jsoup` je knihovna v jazyce Java pro práci s HTML. Poskytuje velmi pohodlné API pro získávání dat a manipulaci s nimi. Implementuje specifikaci HTML5 a parsuje HTML do stejného DOM jako to dělají moderní prohlížeče.

- HTML parsuje z URL, souboru nebo řetězce.
- Hledá a získává data pomocí průchodu DOM nebo CSS selektorů.
- Manipuluje s HTML elementy, atributy a textem.
- Čistí uživatelem předložený obsah, kvůli obraně proti XSS útokům.
- Vytváří čisté HTML.

`jsoup` je navržen tak, aby se vypořádal se všemi druhy volně nalezeného HTML, od dokonalého a validovaného po nevalidovaný, vytváří praktický parsovací strom.

`jsoup` je open source projekt distribuovaný pod licencí MIT. Zdrojový kód je k dispozici na GitHub, aktuální verze 1.7.3.

#### 3.1 Příklad

Načtení HTML z URL a získání textu v elementu `<title>`.

```
Document doc = Jsoup.connect("http://example.com/").get();
String title = doc.title();
```

Statická metoda `connect(String url)` ze třídy `Jsoup` vytvoří nový objekt `Connection` a jeho metoda `get()` zpracuje HTML stránku z předané URL do objektu `Document`. Pokud dojde k chybě při načítání adresy URL, je vyhozena výjimka `IOException`, kterou je třeba obsloužit. Metoda `title()` z objektu `Document` získá text obsažený v elementu `<title>`. [6]

## 4 POSTGRESQL

PostgreSQL je relačním databázovým systémem s otevřeným zdrojovým kódem. Je šířen pod BSD licencí, která je nejliberálnější licencí ze všech open source licencí. Prošel více než sedmnácti lety aktivního vývoje. Je ceněn pro svou spolehlivost a bezpečnost. Běží nativně na všech rozšířených operačních systémech. Splňuje všechny podmínky ACID, podporuje cizí klíče, operace JOIN, pohledy, spouště a uložené procedury. Obsahuje většinu SQL92 a SQL99 datových typů, např. INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL a TIMESTAMP a podporuje také moderní datové typy, jako jsou JSON nebo XML. K systému existuje volně dostupná dokumentace. Výkonnostně nezaostává za srovnatelnými komerčními systémy a v mnohdy je i předčí.

PostgreSQL umožňuje běh uložených procedur napsaných v několika programovacích jazycích, v Perlu, v Pythonu, v jazyku C nebo v speciálním PL/pgSQL, jazyku vycházejícím z PL/SQL fy. Oracle. Existují PostgreSQL varianty JDBC, ODBC, dbExpress, Open Office, PHP, .NET, Perl nativních rozhraní. K PostgreSQL existuje překladač Embedded SQL pro C a C++.

Předností systému PostgreSQL je rozšiřitelnost. Systém lze snadno rozšiřovat o nové datové typy, funkce operátory, agregační funkce, procedurální jazyky. Což dalo vzniknout rozšířením jako PostGIS - podpora pro geografické informační systémy, Slony-I - master to multiple slaves replikace a dalším. [7]

## 5 GOOGLE GEOCODING API

Geocoding je proces převádění adres (např. "1600 Amphitheatre Parkway, Mountain View, CA") do zeměpisných souřadnic (zeměpisná šířka 37,423021 a zeměpisná délka - 122,083739), které lze použít např. k umístění značky na mapě. Google Geocoding API poskytuje přímý způsob přístupu k těmto službám pomocí HTTP požadavku.

Tato služba je hlavně určena pro geocoding statických (předem známých) adres pro umístění obsahu aplikace na mapě.

Omezení použití:

- Bezplatní uživatelé API - 2500 přístupů za 24 hodin.
- Firemní zákazníci - 100000 přístupů za 24 hodin.

Formát Geocoding API požadavku:

`http://maps.googleapis.com/maps/api/geocode/output?parameters`

kde `output` může mít jednu z následujících hodnot:

- `json` indikuje výstup v JavaScript Object Notation (JSON).
- `xml` indikuje výstup jako XML.

Povinné parametry požadavku pro geokódování:

- `address` — adresa, pro kterou jsou hledány zeměpisné souřadnice  
nebo  
`components` – filtr komponent (viz dokumentaci Google Geocoding API), pro který je požadováno získat geocode.
- `sensor` - indikuje, zda požadavek pro geokódování pochází ze zařízení se senzorem umístění (jako GPS lokátor). Tato hodnota musí být buď `true` nebo `false`.

Volitelné parametry požadavku pro geokódování, např.:

- `key` – API klíč k aplikaci. Tento klíč identifikuje aplikaci pro účely správy kvót.
- `language` - jazyk, ve kterém budou vráceny výsledky.

- `region` – dvoumístná hodnota kódu regionu.
- `components` - filtry komponent, oddělené svislou čarou (`()`). Každý filtr komponenty sestává z páru `component:value`.

Odpovědi geokódování jsou vráceny ve formátu uvedeném příznakem `output` v URL cestě požadavku.

### Výstupní formát XML

V příkladu níže Geocoding API požaduje `xml` odpověď na dotaz pro adresu "1600 Amphitheatre Parkway, Mountain View, CA":

```
http://maps.googleapis.com/maps/api/geocode/xml?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false
```

XML vrácené tomuto požadavku je uvedeno níže.

```
<GeocodeResponse>
  <status>OK</status>
  <result>
    <type>street_address</type>
    <formatted_address>1600 Amphitheatre Pkwy, Mountain View, CA
94043, USA</formatted_address>
    <address_component>
      <long_name>1600</long_name>
      <short_name>1600</short_name>
      <type>street_number</type>
    </address_component>
    <address_component>
      <long_name>Amphitheatre Pkwy</long_name>
      <short_name>Amphitheatre Pkwy</short_name>
      <type>route</type>
    </address_component>
    <address_component>
      <long_name>Mountain View</long_name>
      <short_name>Mountain View</short_name>
      <type>locality</type>
      <type>political</type>
    </address_component>
    <address_component>
      <long_name>San Jose</long_name>
      <short_name>San Jose</short_name>
      <type>administrative_area_level_3</type>
      <type>political</type>
    </address_component>
  </result>
</GeocodeResponse>
```

```
<address_component>
  <long_name>Santa Clara</long_name>
  <short_name>Santa Clara</short_name>
  <type>administrative_area_level_2</type>
  <type>political</type>
</address_component>
<address_component>
  <long_name>California</long_name>
  <short_name>CA</short_name>
  <type>administrative_area_level_1</type>
  <type>political</type>
</address_component>
<address_component>
  <long_name>United States</long_name>
  <short_name>US</short_name>
  <type>country</type>
  <type>political</type>
</address_component>
<address_component>
  <long_name>94043</long_name>
  <short_name>94043</short_name>
  <type>postal_code</type>
</address_component>
<geometry>
  <location>
    <lat>37.4217550</lat>
    <lng>-122.0846330</lng>
  </location>
  <location_type>ROOFTOP</location_type>
  <viewport>
    <southwest>
      <lat>37.4188514</lat>
      <lng>-122.0874526</lng>
    </southwest>
    <northeast>
      <lat>37.4251466</lat>
      <lng>-122.0811574</lng>
    </northeast>
  </viewport>
</geometry>
</result>
</GeocodeResponse>
```

XML odpověď se skládá z jednoho elementu <GeocodeResponse> a dvou elementů nejvyšší úrovně:

- Element <status> obsahuje metadata požadavku. Viz status kódy níže.

- Žádný nebo více elementů `<result>`, z nichž každý obsahuje jednu sadu geokódovacích informací o adrese a geometrických informací.

### Status kódy

Proměnná `status` v objektu odpovědi Geocodingu obsahuje stav požadavku, může obsahovat ladící informace, s jejichž pomocí lze vypátrat, proč geokódování nefunguje. Proměnná `status` může obsahovat následující hodnoty:

- "OK" znamená, že nedošlo k žádným chybám, adresa byla úspěšně parsována, byl vrácen alespoň jeden geokód.
- "ZERO\_RESULTS" znamená, že geokód byl úspěšný, ale nevrátil žádný výsledek. Taková situace může nastat v případě, že geokodéru byla zadána neexistující adresa.
- "OVER\_QUERY\_LIMIT" znamená, že byla překročena povolená kvóta přístupů.
- "REQUEST\_DENIED" znamená, že požadavek byl zamítnut, většinou z důvodu nedostatku parametrů senzoru.
- "INVALID\_REQUEST" znamená, že chybí dotaz (adresa, komponenty, atd.).
- "UNKNOWN\_ERROR" znamená, že požadavek nelze zpracovat kvůli chybě serveru. Požadavek může být úspěšný, při opakovaném provedení pokusu.

Pokud geokodér vrátí jiný status kód než OK, objekt odpovědi může obsahovat navíc proměnnou `error_message` s podrobnými informacemi o důvodech daného status kódu. [8]

## 6 ANT

Ant je zkratka Another Neat Tool. Jedná se o open source nástroj implementovaný v Javě plně přenositelný mezi platformami. Nevyužívá příkazy hostitelského operačního systému, ale nahrazuje je vlastními úlohami (tasks) s podobnou funkčností. Má široké spektrum schopností, viz <http://ant.apache.org/manual/taskoverview.html>, v této práci je využit jako nástroj pro sestavení portletového projektu.

Pro činnost Antu je třeba vytvořit XML soubor obvykle s názvem `build.xml`. Tento XML soubor obsahuje kořenový element `<project>` a minimálně jeden element `<target>`. K dispozici jsou další užitečné elementy jako `fileset` (pro definování skupiny souborů), `property` (k definování vlastností, které mohou být čteny z externího souboru vlastností) atd. V elementech `<target>` jsou logicky seskupeny jednotlivé úlohy a mohou být závislé na provedení úloh v jiných elementech `<target>`. Příklady tasků jsou `war` (pro vytvoření souboru WAR), `delete` (pro mazání souborů) a `copy` (pro kopírování souborů). Seznam úloh, které poskytuje Ant, lze nalézt na: <http://ant.apache.org/manual/taskoverview.html>. Každý `<target>` musí obsahovat atribut `name`, jehož hodnota pojmenovává daný `<target>` a musí být v rámci projektu unikátní. Nejčastěji spouštěný `<target>` je v elementu `<project>` označen jako defaultní. [9]

Výpis níže představuje fragment kódu souboru `build.xml` pro sestavení WAR souboru z portletového projektu této práce, na němž jsou popsány jednotlivé elementy a jejich atributy:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="TenderAggregator" default="build" basedir=". ">
  <property file="build.properties" />
  ...
  <property name="build.dir" value="build" />
  <target name="build" depends="clean,compile">
    <mkdir dir="${build.dir}" />
    <war destfile=
      "${build.dir}/TenderAggregator.war"
      webxml="${web.xml}" />
```

```
        <fileset refid="war.files" />
    </war>
    <copy todir="${liferay.portal.home}/deploy">
        <fileset dir="${build.dir}">
            <include name="**/*.war" />
        </fileset>
    </copy>
</target>
</project>
```

Atribut `name` elementu `<project>` definuje jako název projektu `TenderAggregator`, atribut `default` určuje jako defaultní `<target>` s názvem `build`, atribut `basedir="."` určuje základní adresář pro všechny dále uváděné relativní cesty. V `build.xml` lze definovat vlastnosti (properties) přímo nebo je lze načíst z externího souboru. V tomto případě je zde definována vlastnost s názvem `build.dir` a hodnotou `build`. Atribut `file="build.properties"` elementu `<property>` instruuje Ant k načtení souboru `build.properties` a použití zde definovaných vlastností. Atribut `depends` z elementu `<target name="build"` specifikuje závislé targety. Target `build` je závislý na targetech `clean` a `compile`, které nařizují Antu nejprve provést target `clean` a pak target `compile` před provedením targetu `build`. Element `<war>` specifikuje vytvoření souboru WAR názvem `TenderAggregator` a určuje, že má být zkopírován do adresáře `{liferay.portal.home}/deploy`, což je adresář pro hot deploy Liferay Portalu. Soubor `build.properties` obsahuje vlastnost, která odkazuje na domovský adresář instalace Liferay Portal (`LIFERAY_HOME`):

```
liferay.portal.home=C:/liferay-portal-6.1.2
```

Nástroj Ant je součástí Eclipse IDE použitého pro tvorbu aplikací v této práci. Sestavení WAR souboru a jeho nasazení na server zde lze provést volbou `Run As > Ant Build` zobrazené po stisku pravého tlačítka myši na souboru `build.xml`.

## **II. PRAKTICKÁ ČÁST**

## 7 VÝVOJ APLIKACE

### 7.1 Popis aplikace

Aplikace získává informace o veřejných zakázkách z informačních systémů veřejných zakázek a provádí jejich agregaci do jediné databáze. Zdrojem informací o veřejných zakázkách jsou e-tržišť veřejné zprávy Centrum veřejných zakázek České pošty, Český trh, sdružení TenderMarket, Syntaxit a Vortal.

Údaji o zakázkách, ukládanými do společné databáze, jsou název e-tržště, URL e-tržště, název zakázky, popis zakázky, stav zakázky, URL zakázky na e-tržšti, lhůta pro podání nabídek, termín zahájení, předpokládaná hodnota, měna, datum a čas otevírání nabídek, datum vytvoření zakázky v databázi, datum úpravy údajů o zakázce v databázi, kódy CPV kategorií položek zakázky a v neposlední řadě údaje o zadavateli. Údaje o zadavateli zakázky na e-tržšti nejsou vždy úplné, proto aplikace pro jejich získání využívá XML rozhraní informačního systému ARES Ministerstva financí, kde lze na základě IČO údaje o podnikatelských subjektech získat.

Další funkcionalitou aplikace je získávání geografických souřadnic pro adresy. Smyslem této funkcionality je zamezení redundance adres v databázi. Před uložením nově získané adresy je kontrolováno, jestli již ukládaná adresa se stejnými souřadnicemi v databázi neexistuje. Dalšími možnostmi využití hodnot souřadnic je vyhledání adres, resp. zakázek s adresou zadavatele, v určité vzdálenosti od daného místa nebo jejich zobrazení na mapě. Geografické souřadnice jsou získávány prostřednictvím Google Geocoding API.

Aplikace zobrazuje zakázky seřazené chronologicky, od nejstarší po nejnovější, podle data jejich vytvoření uloženého v databázi. Nabízí uživateli možnost zobrazit detailní informace o zakázce, vyhledávat resp. filtrovat zakázky podle zadaných parametrů, kterými mohou být název zakázky, název zadavatele, IČO zadavatele, stav zakázky, kraj, obor, klíčová slova, rozsah lhůt pro podání nabídek, rozsah termínů zahájení a rozsah předpokládaných hodnot zakázek.

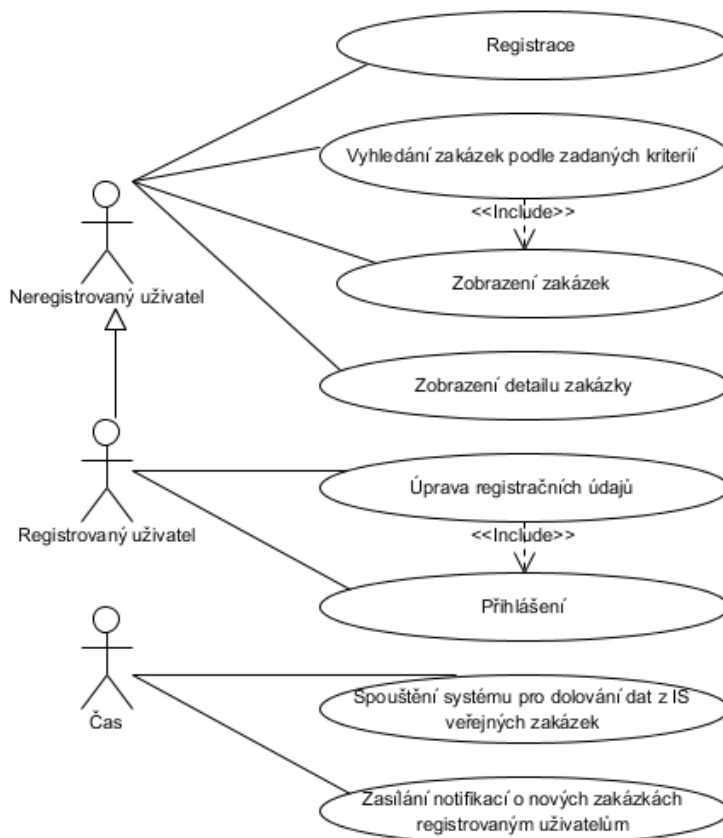
Aplikace poskytuje také rozhraní pro registraci uživatele. Předmětem registrace je, kromě zadání identifikačních údajů, také zadání kritérií pro zasílání notifikací o nových zakázkách a zadání frekvence jejich zasílání. Aplikace při registraci rozlišuje dva typy uživatelů, uživatele – podnikatelský subjekt a uživatele – soukromou osobu. Jejich identifikační údaje vyžadované při registraci se mírně liší. Zatímco uživatel – soukromá osoba je povinen za-

dat jméno a příjmení, od uživatele – podnikatelského subjektu je požadováno zadání obchodního jména a navíc IČO, případně i DIČ, pokud je plátcem DPH. Ostatní registrační údaje jsou pro oba typy uživatelů shodné. Jsou jimi adresa, e-mail, heslo, kriteria pro zasílání notifikací o nových zakázkách a frekvence zasílání těchto notifikací. E-mail slouží jako jedinečný identifikátor uživatele a v aplikaci nesmí existovat více uživatelů se stejným e-mailem.

Uživatel může při registraci nastavit parametry kriterií, podle kterých mu budou zasílány notifikace o nových zakázkách. Pro kriteria má možnost výběru z těchto parametrů: obory (kategorizace podle kódu CPV oborů), krajů, klíčových slov, rozsahu hodnot zakázek, názvu a IČO zadavatele. Posledním registračním údajem je nastavení frekvence zasílání těchto notifikací, a to buď denně, nebo týdně.

Registrovaný uživatel má možnost po přihlášení do aplikace provést úpravu registračních údajů. Pro přihlášení je vyžadován e-mail a heslo zadané při registraci.

Aplikace poskytuje systém pro zasílání notifikací registrovaným uživatelům o nových zakázkách na základě jimi zadaných kriterií.



Obrázek 4 - Model případů užití

Aplikace je napsána v jazyce Java, využívá frameworků Hibernate a Spring a databázi PostgreSQL s rozšířením PostGIS umožňující ukládání prostorových dat. Pro vývoj aplikací v této práci je použito IDE Eclipse verze Kepler Service Release 2. Pro nasazení webové resp. portletové aplikace je použito portálového serveru Liferay Portal 6.1 CE GA3 s aplikačním serverem Apache Tomcat 7.0.40.

Aplikaci lze rozdělit do tří samostatných funkčních celků:

- Aplikaci pro získávání dat z e-tržišť veřejných zakázek a jejich agregaci do databáze, dále Tender Aggregator Data Mining.
- Portletovou aplikaci zobrazující informace o veřejných zakázkách umožňující vyhledávání zakázek podle zadaných parametrů. Obsahuje také administraci pro registraci nových uživatelů a přihlášení registrovaných uživatelů, dále Tender Aggregator Portlet.
- Aplikaci pro zaslání notifikací registrovaným uživatelům o nových zakázkách, dále Tender Aggregator Notification.

Jednotlivým dílčím aplikacím bude věnována zbývající část této kapitoly.

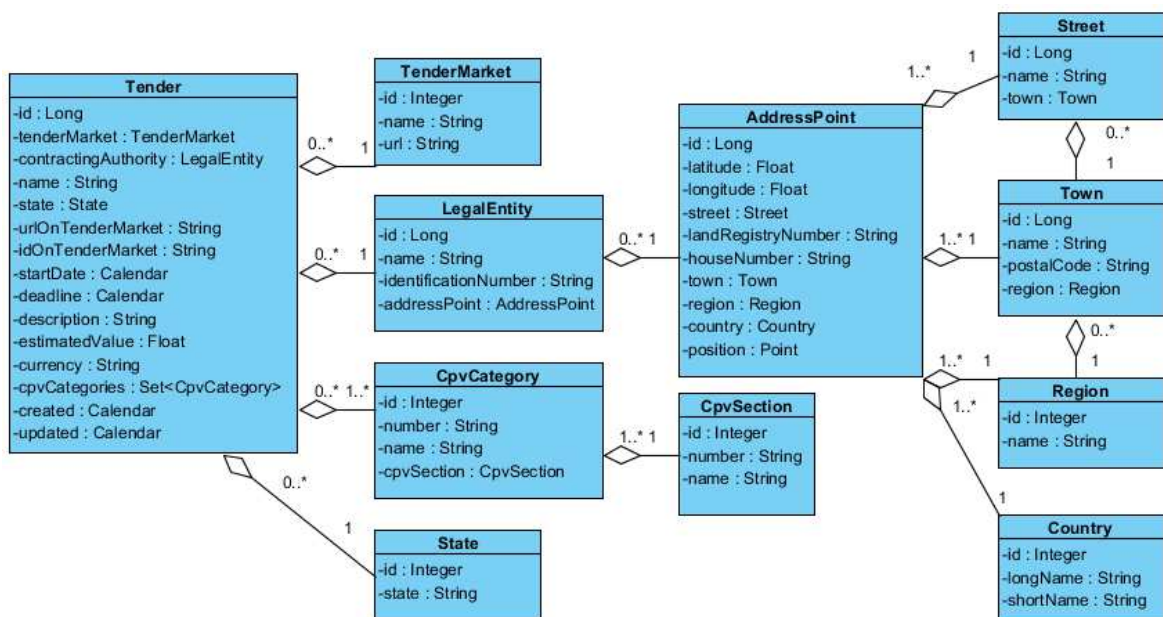
## 7.2 Aplikace Tender Aggregator Data Mining

Aplikace Tender Aggregator Data Mining je konzolovou aplikací, lze ji spustit jakou službu. Provádí, jak již bylo zmíněno, parsování dat z elementů HTML stránek e-tržišť veřejných zakázek a agreguje je do jediné databáze. K parsování je použito HTML parseru jsou popsáno v části 3. Získaná data jsou použita jako hodnoty atributů vytvářených instancí persistentních tříd namapovaných na tabulky databáze PostgreSQL. Obrázek 5 diagramu tříd ukazuje perzistentní třídy použité v aplikaci a jejich asociace (pozn. pro zjednodušení zde nejsou uvedeny getter a setter metody pro jednotlivé atributy).

### 7.2.1 Persistentní třídy

Každá z perzistentních tříd obsahuje atribut `id` (typu `Long` nebo `Integer`), jejich jedinečný identifikátor v databázi. Instance třídy `Tender` představuje konkrétní zakázku. Obsahuje atributy `tenderMarket` třídy `TenderMarket` reprezentující e-tržiště, `contractingAuthority` třídy `LegalEntity` reprezentující zadavatele zakázky, `name` typu `String` pro název zakázky, `description` typu `String` pro popis zakázky, `state` třídy

State reprezentující stav zakázky, `urlOnTenderMarket` typu `String` pro URL zakázky na e-tržišti, `idOnTenderMarket` pro unikátní identifikátor zakázky na daném e-tržišti, `startDate` třídy `Calendar` pro datum zahájení provedení zakázky, `deadline` třídy `Calendar` pro konečné datum příjmu nabídek na zakázku, `estimatedValue` typu `Float` pro předpokládanou hodnotu zakázky, `currency` typu `String` pro měnu zakázky, `cpvCategories` kolekci rozhraní `Set<CpvCategory>` reprezentující CPV kategorie položek zakázky, `created` třídy `Calendar` pro datum vytvoření záznamu o zakázce v databázi a atribut `updated` třídy `Calendar` pro datum úpravy záznamu o zakázce v databázi.



Obrázek 5 – Asociované třídy k persistentní třídě `Tender`

Třída `TenderMarket` reprezentující e-tržišť obsahuje atributy `name` typu `String` pro název e-tržišť a atribut `url` typu `String` pro URL e-tržišť.

Třída `LegalEntity` představuje právní subjekt, obsahuje atributy `name` typu `String` pro název, `identificationNumber` typu `String` pro IČO a atribut `addressPoint` třídy `AddressPoint` reprezentující adresu s geografickými souřadnicemi.

Třída `AddressPoint` reprezentuje adresu společně s jejími geografickými souřadnicemi, obsahuje atributy `latitude` typu `Float` pro hodnotu souřadnice zeměpisné šířky, `longitude` typu `Float` pro hodnotu souřadnice zeměpisné délky, `street` třídy `Street` reprezentující ulici, `landRegistryNumber` typu `String` pro číslo popisné, `houseNumber`

typu `String` pro číslo orientační, `town` třídy `Town` reprezentující město, `region` třídy `Region` reprezentující kraj, `country` třídy `Country` reprezentující stát a atribut `position` třídy `Point` reprezentující geografickou polohu.

Třída `Street` reprezentuje ulici, obsahuje atributy `name` typu `String` pro název ulice a `town` třídy `Town` reprezentující město, v němž se daná ulice nachází.

Třída `Town` reprezentuje město, obsahuje atributy `name` typu `String` pro název města, `postalCode` typu `String` pro PSČ města a atribut `region` třídy `Region` reprezentující kraj, v němž se dané město nachází.

Třída `Region` reprezentující kraj, obsahuje vedle `id` pouze atribut `name` typu `String` pro název kraje.

Třída `Country` reprezentuje stát, obsahuje atributy `longName` typu `String` pro název státu a atribut `shortName` typu `String` pro zkratku státu.

Třída `CpvCategory` reprezentuje kategorii CPV (společného slovníku pro veřejné zakázky). Obsahuje atributy `number` typu `String` pro kód kategorie, `name` typu `String` pro název kategorie, `cpvSection` třídy `CpvSection` pro oddíl CPV, pod který daná kategorie spadá.

Třída `CpvSection` reprezentuje oddíl CPV, obsahuje atributy `number` typu `String` pro kód oddílu a `name` typu `String` pro jeho název.

Třída `State` reprezentuje stav zakázky, obsahuje kromě atributu `id` jediný atribut `state` typu `String` pro popis stavu zakázky.

### 7.2.2 Získávání dat z e-tržišť

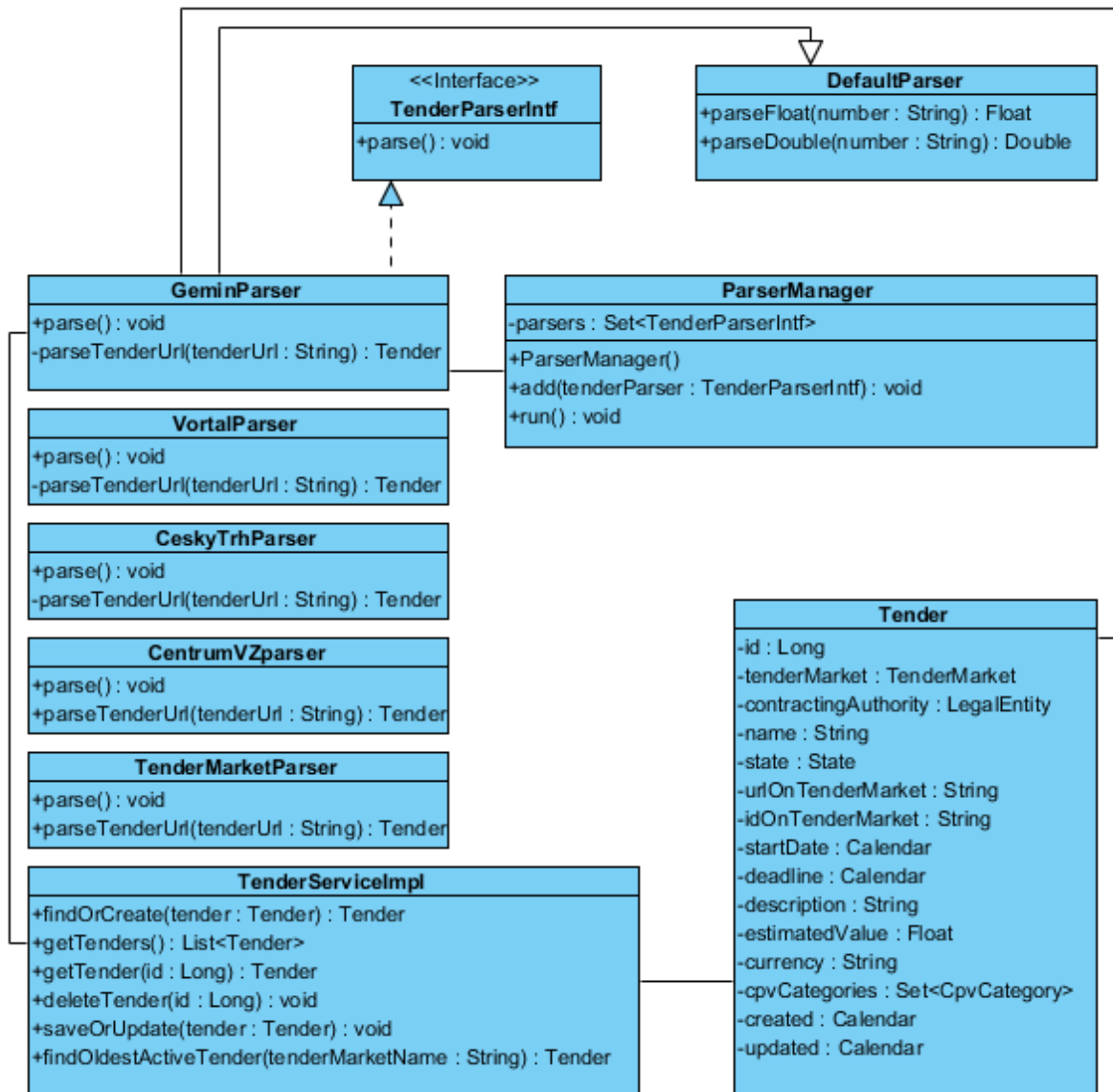
K parsování HTML stránek e-tržiště je používáno instancí třídy parseru pro dané e-tržiště (`CentrumVZparser`, `CeskyTrhParser`, `GeminParser`, `TenderMarketParser`, `VortalParser`). Každá třída parseru dědí ze třídy `DefaultParser` a implementuje rozhraní `TenderParserIntf` a jeho metodu `parse()`.

Metoda `parse()` nejdříve získává z HTML e-tržiště URL jednotlivých zakázek a ukládá je do kolekce `ArrayList`, poté prochází tuto kolekci a parsuje HTML z uložených URL (volá privátní metodu `parseTenderUrl(String tenderUrl)`) a vytváří instance třídy `Tender`, které ukládá chronologicky, od nejstarší po nejnovější, do databáze.

Aby nebylo zbytečně parsováno HTML všech neaktivních zakázek, pokouší se metoda `parse()` nejdříve nalézt v databázi zakázku s nejstarším datem vytvoření (volá metodu `findOldestActiveTender(String tenderMarketName)` instance třídy `TenderServiceImpl`). Pokud takovou zakázku nalezne, je získávání URL zakázek ukončeno při natrefení na identifikátor zakázky shodující se s identifikátorem nalezené zakázky a přejde k parsování HTML kódu na URL adresách zakázek uložených v kolekci `ArrayList`.

Instance třídy `ParserManager` obsahuje metodu `add(TenderParserIntf TenderParserIntf)`. Tato metoda ukládá instance tříd parserů implementujících rozhraní `TenderParserIntf` přijatých jako její parametr do svého atributu `parsers` (kolekce `HashSet<TenderParserIntf>`). Metoda `run()` instance `ParserManager` prochází kolekci `parsers` a volá metodu `parse()` pro každou instanci třídy parseru uloženou v této kolekci.

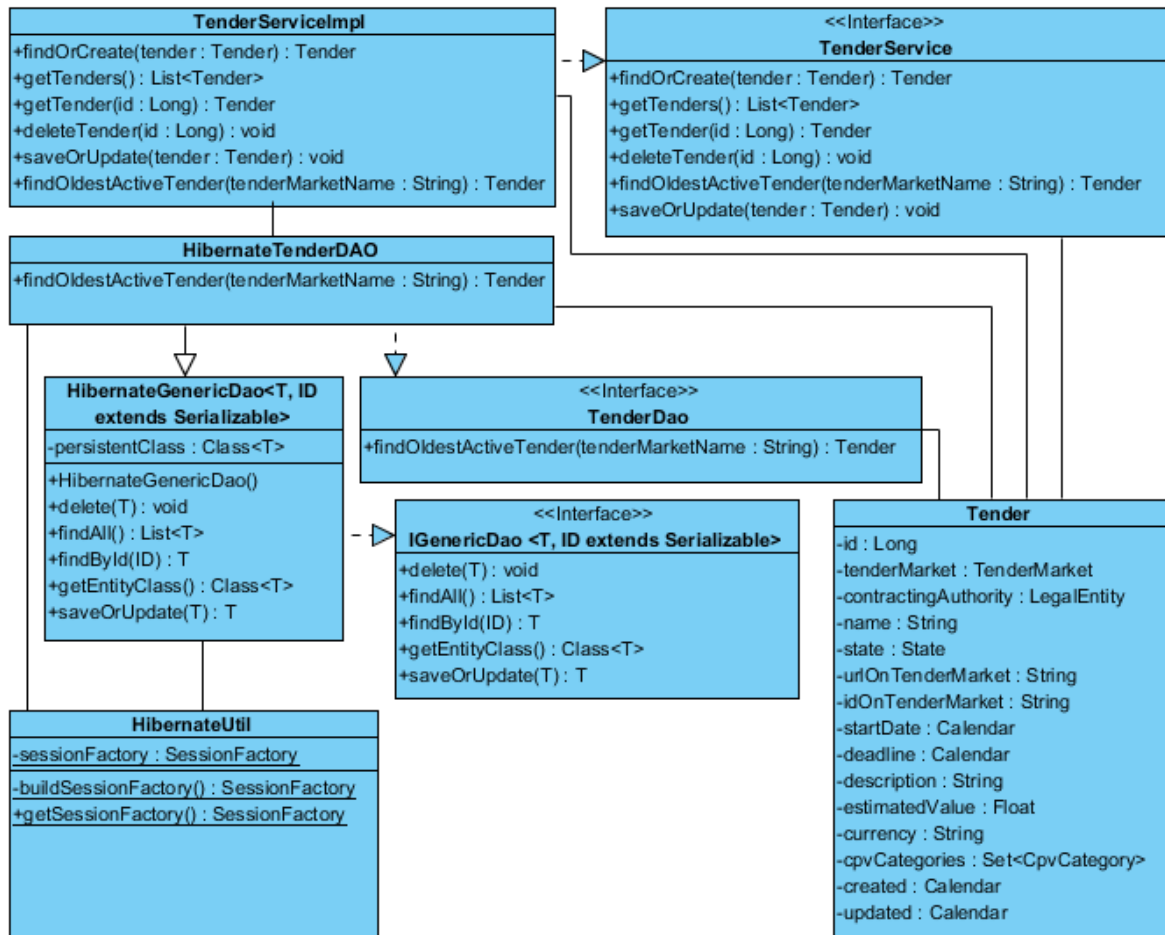
Třída `ParserManager` je správcem pro spouštění parserů. Obsahuje metodu `add(TenderParserIntf tenderParserIntf)` pro přidání instance třídy parseru implementující rozhraní `TenderParserIntf` atributu `parsers` generické třídy `HashSet<TenderParserIntf>` a metodu `run()`, která prochází kolekci `parsers` a volá metodu `parse()` jednotlivých instancí parserů do této kolekce uložených.



Obrázek 6 – Zjednodušený diagram tříd aplikace Tender Aggregator Data Mining

### 7.2.3 Komunikace s databází

Pro oddělení kódu pracujícího s databází do samostatné vrstvy je použit návrhový vzor GenericDAO. Generické rozhraní IGenericDAO deklaruje metody společné pro všechny DAO třídy - operace CRUD. Generická třída HibernateGenericDAO tyto metody implementuje. Pro každou persistentní třídu existuje jedno DAO rozhraní a jedna DAO třída implementující toto rozhraní a dědicí třídu HibernateGenericDAO.



Obrázek 7 - Diagram tříd návrhového vzoru GenericDAO pro třídu Tender

Přístup k DAO objektům je řešen přes servisní vrstvu. Pro každou DAO třídu existuje servisní rozhraní a servisní třída toto rozhraní implementující.

Diagram tříd na Obrázek 7 znázorňuje popsáný způsob použitý pro persistentní třídu Tender. Zde třídy HibernateGenericDAO a HibernateTenderDAO využívají metody getSessionFactory() instance třídy HibernateUtil k získání objektu SessionFactory resp. Session. Třída TenderServiceImpl využívá třídu HibernateTenderDao pro přístup k databázi.

Metoda findOrCreate(Tender tender) třídy TenderServiceImpl je volána metodou parse() z instance parseru před uložením vytvořeného objektu Tender do databáze. Hledá v databázi instanci Tender se stejnou hodnotou atributu idOnTenderMarket jakou má objekt Tender v parametru metody. Pokud takovou instanci nalezne, nastaví její atribut state na stejnou hodnotu, jakou má objekt v parametru metody, atributu updated nastaví aktuální datum a vrátí tuto aktualizovanou instanci. Pokud instanci se

shodnou hodnotou atributu `idOnTenderMarket` v databázi nenalezne, provádí hledání agregovaných objektů instance `Tender` z parametru metody a nastavuje je nové instance `Tender`, která je metodou vrácena. K získání asociovaných objektů jsou volány metody `findOrCreate()` instancí servisních tříd pro dané instance asociované třídy. Smyslem zmíněného postupu je zamezit redundanci v databázi.

### 7.3 Aplikace Tender Aggregator Portlet

Aplikace `Tender Aggregator Portlet` je webovou resp. portletovou aplikací, jak již bylo dříve uvedeno. Prezentuje informace o veřejných zakázkách, poskytuje filtr pro vyhledávání zakázek podle požadovaných parametrů, nabízí možnost pro zobrazení detailních údajů k vybrané zakázce. Aplikace obsahuje také rozhraní pro registraci nových uživatelů a rozhraní pro úpravu údajů na účtu registrovaného uživatele.

Rozhraní pro registraci slouží k vytvoření účtu nového uživatele, což představuje vedle zadání identifikačních údajů uživatele i nastavení parametrů kritériím, podle nichž budou zaregistrovanému uživateli zasílány e-mailové notifikace o nových zakázkách.

#### 7.3.1 Implementace portletové třídy

Portletová třída musí implementovat rozhraní `javax.portlet.Portlet`. Rozhraní `Portlet` deklaruje metody `init`, `processAction`, `render` a `destroy`. Portlet API obsahuje abstraktní třídu `GenericPortlet` implementující rozhraní `javax.portlet.Portlet`.

Implementace portletové třídy `TenderAggregatorPortlet` v aplikaci `Tender Aggregator Portlet` je řešena zděděním třídy `GenericPortlet` a přepsáním jejích metod.

#### Metoda `init`

Metoda `init` je volána portletovým kontejnerem za účelem inicializace zdrojů portletu. Fragment kódu níže ukazuje implementaci metody `init` ve třídě `TenderAggregatorPortlet`.

```
public class TenderAggregatorPortlet extends GenericPortlet {  
  
    private String selectedUserType;  
    private TenderService tenderService;  
    ...  
    private List <Tender> tenders;
```

```
public void init() {
    tenderService = getTenderService();
    regionService = getRegionService();
    cpvSectionService = getCpvSectionService();
    privatePersonService = getPrivatePersonService();
    firmOrBusinessmanService = getFirmOrBusinessmanService();
    userService = getUserService();

    tenders = tenderService.getTenders();
    selectedUserType = "firmOrBusinessman";
}

public TenderService getTenderService() {
    ApplicationContext springCtx = PortletApplicationContextUtils.getWebApplicationContext(this.getPortletContext());
    return (TenderService)springCtx.getBean("tenderService");
}
```

Metoda `init` získává reference na `Service` beany nakonfigurované ve Springu. V ukázce výše je demonstrována implementaci metody `getTenderService` pro získání beany s názvem `tenderService`. K získání objektu `WebApplicationContext` je použito třídy `PortletApplicationContextUtils`. Její metoda `getWebApplicationContext` přijímá jako parametr objekt `PortletContext` a vrací odkaz na objekt `WebApplicationContext`. Jeho metody `getBean` je použito k získání beany s názvem `tenderService`.

Dále inicializuje z databáze kolekci `List <Tender> tenders` voláním metody `getTenders()` objektu `tenderService` a atributu `selectedUserType` nastavuje hodnotu `"firmOrBusinessman"`. Hodnota tohoto atributu je použita jako výchozí volba typu uživatele v radio-buttonu registračního formuláře.

### Metoda `render`

Metoda `render` je volána po vytvoření instance portletu portletovým kontejnerem a vždy následně po akční metodě. S akční metodou komunikuje prostřednictvím atributu požadavku, v případě třídy `TenderAggregatorPortlet` je použito atributu s názvem `actionStatus`. Ve fragmentu kódu níže metoda `render` v podmínce `if` zjišťuje, jestli je hod-

nota atributu `actionStatus` `registration`. V případě že ano, pro renderování využije následný kód.

```
@RenderMode(name = "VIEW")
public void render(RenderRequest request, RenderResponse response)
throws PortletException, IOException {
    ...
    if ("registration".equalsIgnoreCase((String) request
        .getAttribute("actionStatus"))) {
        ...
        getPortletContext().getRequestDispatcher("/WEB-
            INF/jsp/registrationForm.jsp").include(request, response);
        ...
    }
}
```

Pomocí metody `include` objektu `RequestDispatcher` získaného metodou `getRequestDispatcher` objektu `PortletContext` je začleněna JSP stránka `registrationForm.jsp` do renderovaného obsahu.

### Metoda `processAction`

Akční metoda `processAction` je volána v odpovědi na akční požadavek, např. kliknutí na odkaz nebo odeslání formuláře. Defaultní implementace metody `processAction` třídy `GenericPortlet` odesílá akční požadavek metodě anotované `@ProcessAction` na základě hodnoty parametru `ActionRequest.ACTION_NAME` nastaveného v akční URL.

Fragmenty kódů níže ukazují akční metody portletové třídy `TenderAggregatorPortlet` nastavující hodnoty atributu `actionStatus`:

```
@ProcessAction(name = "registerUserAction")
public void registerUserAction(ActionRequest request, ActionRe-
    sponse response) throws PortletException, IOException {
    ...
    request.setAttribute("actionStatus", "registration");
}
```

Metoda `registerUserAction` je volána po kliknutí na odkaz „registrovat“ na domovské stránce portletu. Další akční metody třídy `TenderAggregatorPortlet`:

```
@ProcessAction(name = "loginUserAction")
public void loginUserAction(ActionRequest request, ActionResponse
response) throws PortletException, IOException {
    ...
    request.setAttribute("actionStatus", "loginUser");
}
```

Metoda `loginUserAction` je volána po kliknutí na odkaz „přihlásit se“ na domovské stránce portletu.

```
@ProcessAction(name = "userAccountPageAction")
public void userAccountPageAction(ActionRequest request, ActionRe-
sponse response) throws PortletException, IOException {
    ...
    request.setAttribute("actionStatus", "userAccountPage");
}
```

Metoda `userAccountPageAction` je volána po stisku tlačítka „Přihlásit se“ na stránce s přihlašovacím formulářem.

```
@ProcessAction(name = "finishRegistration")
public void finishRegistrationAction(ActionRequest request, Acti-
onResponse response) throws PortletException, IOException {
    request.setAttribute("actionStatus", "finishRegistration");
}
```

Metoda `finishRegistrationAction` je volána po stisku tlačítka „Registrovat“ na stránce s registračním formulářem.

```
@ProcessAction(name = "filterTendersAction")
public void filterTendersAction(ActionRequest request, ActionRe-
sponse response) throws PortletException, IOException {
    //Pokud bylo stisknuto tlačítko „FILTROVAT“
    if(request.getParameter("action").equalsIgnoreCase("FILTROVAT"))
    {
        ...
        request.getPortletSession().setAttribute("actionStatus",
"filterTenders", PortletSession.PORTLET_SCOPE);
    }
    // Pokud bylo stisknuto tlačítko „ZRUŠIT FILTR“
    if (request.getParameter("action").equalsIgnoreCase("ZRUŠIT
FILTR"))
    {
        ...
        request.getPortletSession().setAttribute("actionStatus", "",
PortletSession.PORTLET_SCOPE);
    }
}
```

```
    }  
}
```

Metoda `filterTendersAction` je volána po stisku tlačítka „Filtrovat“ nebo „Zrušit filtr“ na domovské stránce portletu.

```
@ProcessAction(name = "showTenderDetails")  
public void showTenderDetailsAction(ActionRequest request, ActionResponse response) throws PortletException, IOException {  
    ...  
    request.setAttribute("actionStatus", "showTenderDetails");  
}
```

Metoda `showTenderDetailsAction` je volána po kliknutí na odkaz s názvem zakázky na domovské stránce portletu.

```
@ProcessAction(name = "saveChangesRegisteredUserAction")  
public void saveChangesRegisteredUserAction(ActionRequest request, ActionResponse response) throws PortletException, IOException {  
    ...  
    request.setAttribute("actionStatus", "saveChangesRegisteredUser");  
}
```

Metoda `saveChangesRegisteredUserAction` je volána po stisku tlačítka "Uložit změny" na stránce účtu registrovaného uživatele.

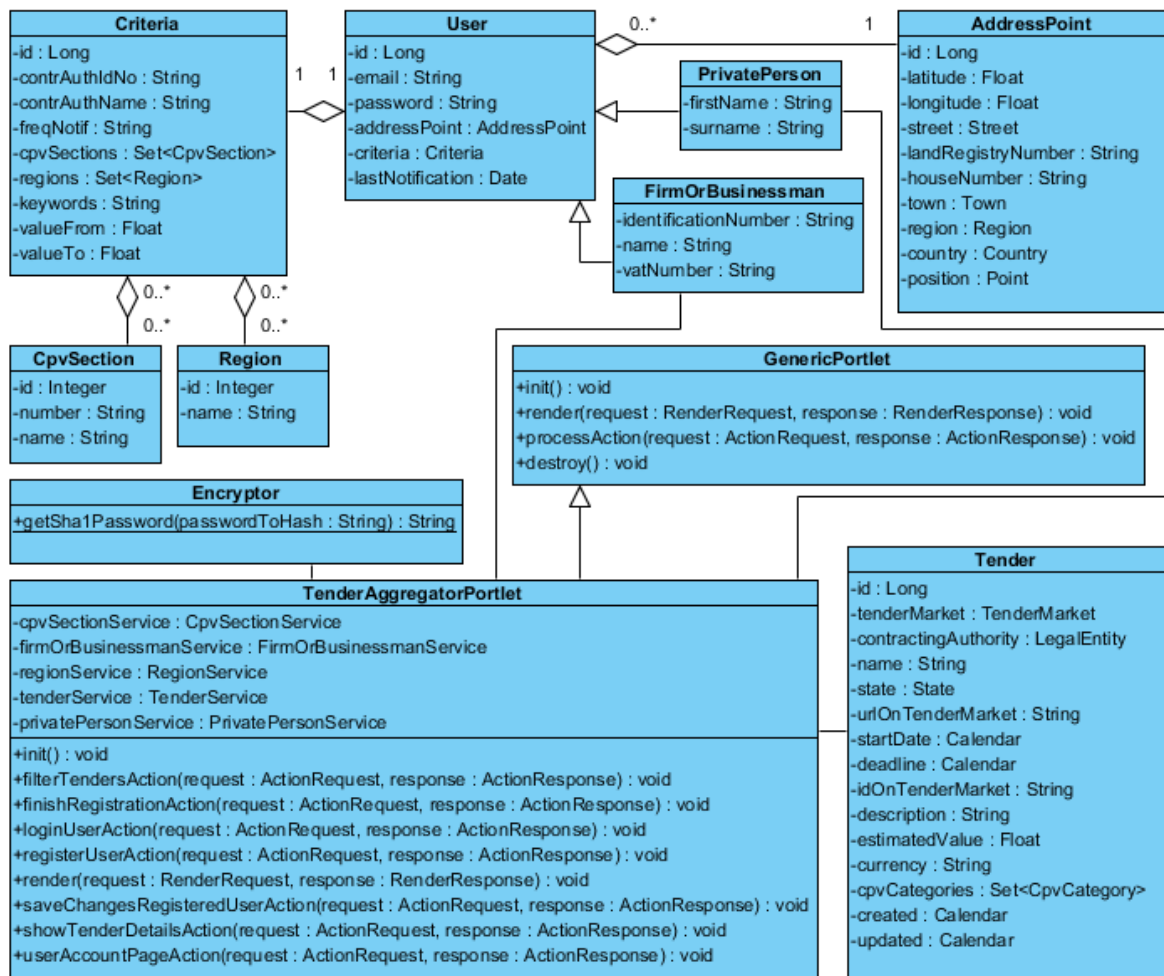
Portlet `TenderAggregatotPortlet` využívá pro zobrazení obsahu JSP stránky, Tabulka 3 obsahuje jejich seznam s popisem.

Tabulka 3 – JSP stránky použité portletem Tender Aggregator Portlet

Název JSP stránky	Zobrazený obsah
homePage.jsp	Zobrazuje seznam zakázek s filtrem pro vyhledávání a odkazy pro registraci a přihlášení uživatele.
registrationForm.jsp	Zobrazuje registrační formulář uživatele.
loginForm.jsp	Zobrazuje formulář pro zadání přihlašovacích údajů registrovaného uživatele.
tenderDetail.jsp	Zobrazuje detaily zakázky.
privatePersonAccountPage.jsp	Zobrazuje účet zaregistrovaného uživatele – soukromé osoby.
firmOrBusinessmanAccountPage.jsp	Zobrazuje účet zaregistrovaného uživatele – podnikatelského subjektu.
successSaveChangesRegisteredUser.jsp	Zobrazuje potvrzení o úspěšném provedení změny registračních údajů.
completeRegistration.jsp	Zobrazuje potvrzení o úspěšném dokončení registrace.

### 7.3.2 Třídy portletové aplikace

Portletová třída `TenderAggregatorPortlet` pracuje jako kontroler, zpracovává příchozí požadavky, připravuje data a odesílá požadavky stránkám JSP. Pro svou činnost využívá také dalších tříd, z nichž některé byly představeny v kapitole 7.2 o aplikaci Tender Aggregator Data Mining, ostatním bude věnován další text. Diagram tříd na Obrázek 8 ukazuje třídy tvořící hlavní strukturu vyvíjené portletové aplikace a jejich vztahy.



Obrázek 8 – Diagram tříd aplikace Tender Aggregator Portlet

## Třída Criteria

Persistentní třída, jejíž instance jsou vytvářeny z parametrů získaných z registračního formuláře uživatele pro uchování kritérií pro zasílání notifikací o nových zakázkách. Jsou také načítány z databáze pro zobrazení zadaných kritérií v registračním účtu uživatele. Tabulka 4 – *Atributy třídy Criteria* uvádí seznam s názvy a popisem jednotlivých atributů (pozn. nejsou uvedeny jejich getter a setter metody).

Tabulka 4 – Atributy třídy Criteria

Třída a název atributu	Popis
Long id	Jedinečný identifikátor instance v databázi.
String keywords	Klíčová slova pro vyhledávané zakázky.
Float valueFrom	Dolní hranice hodnoty zakázky.
Float valueTo	Horní hranice hodnoty zakázky.
String contrAuthName	Název zadavatele.
String contrAuthIdNo	IČO zadavatele.
String freqNotif	Frekvence zasílání notifikací.
Set <CpvSection> cpvSections	Nastavené obory zakázek.
Set <Region> regions	Nastavené kraje pro zakázky.

### Třída User

Persistentní třída, je supertřídou pro třídy `PrivatePerson` a `FirmOrBusinessman`. V Tabulka 5 je uveden seznam s názvy a popisem jednotlivých atributů (pozn. nejsou uvedeny jejich getter a setter metody).

Tabulka 5 - Atributy třídy User

Třída a název atributu	Popis
Long id	Jedinečný identifikátor instance v databázi.
String email	Email uživatele.
String password	Heslo uživatele.
AddressPoint addressPoint	Adresa uživatele.
Criteria kriteria	Kriteria nastavená uživatelem.
Date lastNotification	Datum zaslání poslední notifikace.

### Třída PrivatePerson

Persistentní třída, dědí od třídy `User`, její instance jsou vytvářeny z parametrů získaných z registračního formuláře uživatele pro uchování identifikačních údajů uživatele - soukromé osoby. Jsou také načítány z databáze pro zobrazení zadaných údajů v registračním účtu uživatele. V Tabulka 6 je uveden seznam s názvy a popisem jednotlivých atributů (pozn. nejsou uvedeny jejich getter a setter metody).

Tabulka 6 – Atributy třídy *PrivatePerson*

Třída a název atributu	Popis
String firstName	Jméno uživatele.
String Suriname	Příjmení uživatele.

### Třída **FirmOrBusinessman**

Persistentní třída, dědí od třídy *User*, její instance jsou vytvářeny z parametrů získaných z registračního formuláře uživatele pro uchování identifikačních údajů uživatele - podnikatelského subjektu. Jsou také načítány z databáze pro zobrazení zadaných údajů v registračním účtu uživatele. V Tabulka 7 je uveden seznam s názvy a popisem jednotlivých atributů (pozn. nejsou uvedeny jejich getter a setter metody).

Tabulka 7 – Atributy třídy *FirmOrBusinessman*

Třída a název atributu	Popis
String name	Název podnikatelského subjektu.
String identificationNumber	IČO podnikatelského subjektu.
String vatNumber	DIČ podnikatelského subjektu.

### Třída **Encryptor**

Třída *Encryptor* obsahuje jedinou statickou metodu `String getSha1Password(String passwordToHash)` k provedení šifrování hesla získaného z registračního formuláře před jeho uložení do databáze nebo šifrování zadaného hesla při přihlášení registrovaného uživatele. Šifrování je provedeno algoritmem SHA1. K šifrování je použito API *Java Security* a jeho třídy *MessageDigest*.

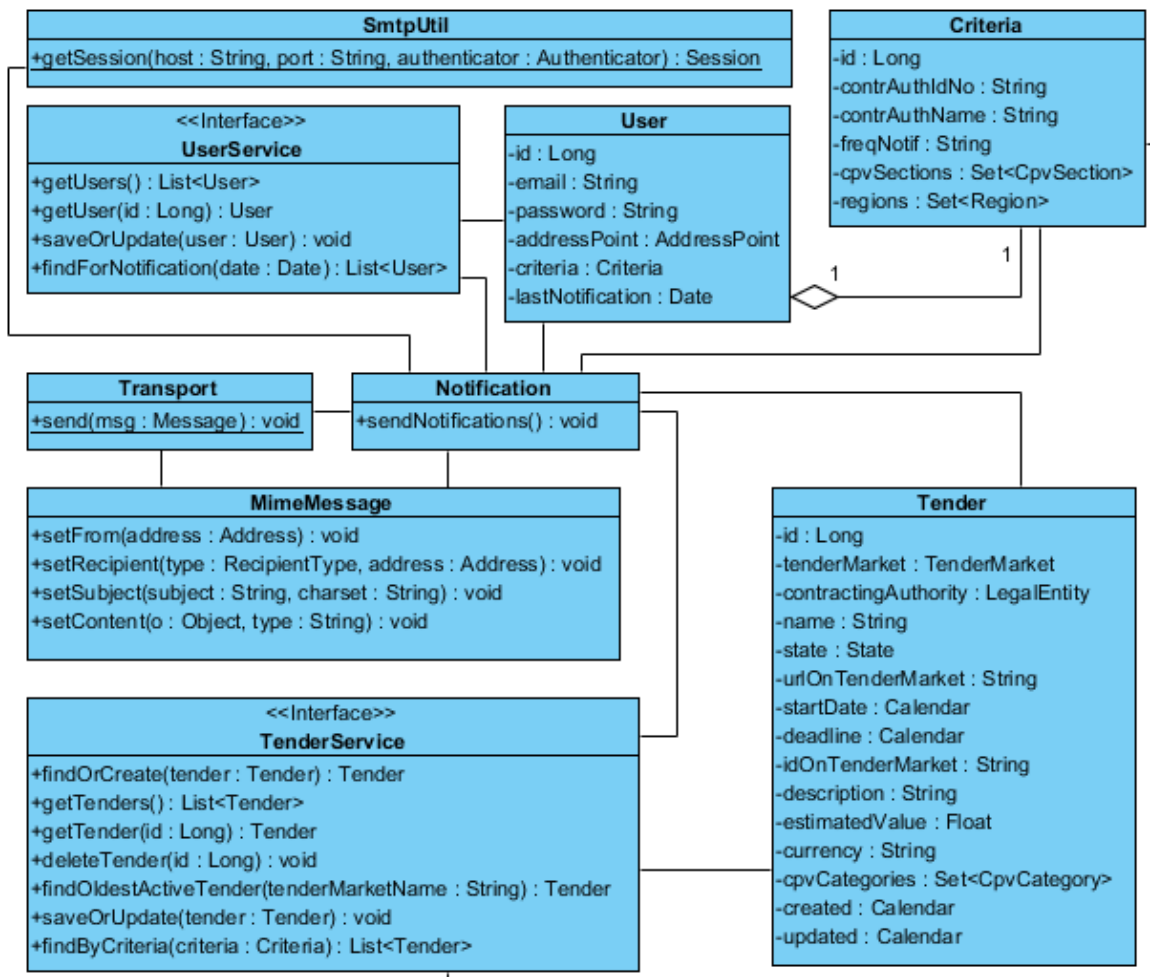
## 7.4 Aplikace **Tender Aggregator Notification**

Úkolem konzolové aplikací *Tender Aggregator Notification* je rozesílat e-mailové notifikace o nových zakázkách registrovaným uživatelům. V databázi nalezne uživatele, u nichž ještě neproběhla notifikace (nově registrovaní uživatelé), dále pak uživatele, kteří mají nastavenou denní frekvenci zasílání notifikací a aktuální datum je o jeden den vyšší než datum jejich poslední notifikace a uživatele, kteří mají nastavenou týdenní frekvenci zasílání notifikací a aktuální datum je o 7 dní vyšší než datum jejich poslední notifikace. Pro tyto uživatele jsou v databázi hledány zakázky vyhovující kritériím, která si nastavili na svém

registračním účtu a jejichž datum vytvoření v databázi je vyšší, než datum poslední notifikace uživatele (aby nebyly zasílány opakovaně totožné zakázky) a zároveň mají aktivní stav (např. stav ZŘ – příjem nabídek), tj. nebyly již zadány nebo zrušeny. Uživatelům, kteří nemají pro zakázky nastavena žádná kritéria, jsou s denní frekvencí zasílány notifikace o všech aktivních zakázkách s vyšším datem vytvoření, než je datum jejich poslední notifikace.

Informacemi zasílanými v rámci notifikace jsou název zakázky jako odkaz na URL zakázky na e-tržišti, popis zakázky a konečné datum pro podání nabídek. Po odeslání notifikace je provedena aktualizace data poslední notifikace uživatele v databázi.

Na Obrázek 9 zjednodušeného diagramu tříd aplikace Tender Aggregator Notification jsou uvedeny klíčové třídy aplikace a jejich vztahy.



Obrázek 9 – Diagram tříd aplikace Tender Aggregator Notification

### Třída `Smtplib`

Třída obsahuje jedinou statickou metodu `getSession(String host, String port, Authenticator authenticator)` vracející instanci `Session` k SMTP serveru. Význam parametrů metody je následující: `host` – IP adresa SMTP serveru, `port` – port, na kterém naslouchá SMTP server, `authenticator` – objekt obsahující uživatelské jméno a heslo pro připojení k SMTP serveru. Využívá framework JavaMail API (framework pro tvorbu emailových aplikací a aplikací pro zasílání zpráv) a jeho tříd `Authenticator` a `Session`.

### Třída `Notification`

Obsahuje jedinou statickou metodu `sendNotifications()`. Tato metoda vyhledává uživatele, kterým bude adresována notifikace o nových zakázkách, zakázky odpovídající jimi zadaným kritériím, sestavuje obsah emailů s využitím atributů nalezených zakázek, rozesílá notifikační e-maily a provádí aktualizaci datům poslední notifikace uživatelům, jimž byla zaslána notifikace.

Pro vyhledání uživatelů, jimž bude zaslán notifikační e-mail (instancí `User`) využívá metody `findForNotification` z rozhraní `UserService`, jejímž parametrem je aktuální datum. Od tohoto data je odečten počet dní odpovídající frekvenci notifikací daného uživatele. Získané datum je porovnáváno s datem jeho poslední notifikace a v případě že se porovnávaná data shodují je daný uživatel přidán do seznamu pro notifikaci.

Pro vyhledání zakázek pro každého nalezeného uživatele využívá metody `findByCriteria` z rozhraní `TenderService`, která vyhledá na základě kritérií (atributu `Criteria`) daného uživatele (instance `User`) zakázky (instance `Tender`), jejichž datum vytvoření v databázi (atribut `created`) je stejný nebo vyšší než datum poslední notifikace uživatele. Jejich atributy `name`, `description` a `urlOnTenderMarket` využije k sestavení obsahu notifikačních emailů.

Pro sestavení a odeslání notifikačního e-mailu je využito tříd frameworku JavaMail API jak ukazuje následující fragment kódu:

```
Session session = SmtplibUtil.getSession("127.0.0.1", "25", null);
try {
    MimeMessage msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress("tenderAggregator@email.cz"));
    msg.setRecipient(Message.RecipientType.TO, new InternetAdd-
ress(user.getEmail()));
    msg.setSubject("Notifikace o nových zakázkách", "utf-8");
    msg.setContent(text, "text/html; charset=utf-8");
    Transport.send(msg);
} catch (MessagingException mex) {
    System.out.println("send failed, exception: " + mex);
}
```

Pro získání instance `Session` k SMTP serveru je použito metody `getSession` třídy `SmtplibUtil`. Třída `MimeMessage` reprezentuje MIME styl e-mailové zprávy, konstruktor přijímá instanci `Session`. Metodou `setFrom` instance `MimeMessage` je nastaven atribut „Od“ ve zprávě. Metodou `setRecipient` je prvním parametrem (`Message.RecipientType.TO`) nastaven příjemce jako primárního příjemce, druhým parametrem je nastaven ve zprávě atribut „Komu“. Metodou `setSubject` je nastaveno pole hlavičky zprávy "Předmět" a jeho znaková sada. Metodou `setContent` je nastaven obsah e-mailu, je nastaven jako html text se znakovou sadou UTF-8. Statická metoda `send` třídy `Transport` odesílá zprávu. Kvůli ošetření výjimky `MessagingException`, kterou může vyvolat metoda `send` třídy `Transport`, je použito bloku try-catch.

Ověření funkčnosti rozesílání notifikačních e-mailů bylo testované pomocí aplikace `smtplib4dev`, fiktivního, tzv. dummy, SMTP serveru.

## ZÁVĚR

V rámci této práce byla vytvořena aplikace, která umožňuje získávat data z e-tržišť veřejných zakázek a agregovat je v centrální databázi. Pro zobrazení informací o zakázkách agregovaných v databázi byl vytvořen portlet, který vedle zobrazení informací o zakázkách nabízí možnost jejich vyhledávání podle zadaných parametrů. Parametry pro vyhledávání mohou být název zakázky, název zadavatele, IČO zadavatele, stav zakázky, kraj, obor, klíčová slova, rozsah lhůt pro podání nabídek, rozsah termínů zahájení a rozsah předpokládaných hodnot zakázek.

Další funkcionalitou portletu je rozhraní pro registraci nových uživatelů a rozhraní pro úpravu registračních údajů registrovaných uživatelů. Předmětem registrace je, vedle zadání identifikačních údajů uživatele, také nastavení kritérií, podle kterých bude dostávat e-mailové notifikace o nových zakázkách a zadání frekvence zasílání těchto notifikací. Nastavenými kritérii mohou být název zakázky, název zadavatele, IČO zadavatele, kraj, obor, klíčová slova, rozsah lhůt pro podání nabídek, rozsah termínů zahájení a rozsah předpokládaných hodnot zakázek.

Služba zasílání notifikací o nových zakázkách registrovaným uživatelům patří k dalším klíčovým funkcionalitám vytvořené aplikace. Registrovanému uživateli jsou na základě kritérií nastavených při registraci adresovány e-mailové notifikace o nových aktivních zakázkách.

Aplikace využívá Google Geocoding API k získávání geografických souřadnic pro adresy zadavatelů zakázek. Jsou využity pro omezení redundance dat v databázi, ale nabízí další možnosti využití, např. pro zobrazení místa zadavatele zakázky na mapě nebo při implementaci funkcionality pro vyhledávání zakázek v zadané vzdálenosti od určitého místa.

Přidanou hodnotou vytvořené aplikace vůči e-tržišťům je koncentrace informací o veřejných zakázkách a možnost jejich vyhledávání na jednom místě, dalším benefitem je pak rozesílání notifikací o nových zakázkách registrovaným uživatelům.

**SEZNAM POUŽITÉ LITERATURY**

- [1] KRUTÁK, Tomáš a Lenka KRUTÁKOVÁ. *Zákon o veřejných zakázkách s komentářem a příklady: k 1.4.2013*. 2. aktualiz. vyd. Olomouc: ANAG, c2013, 639 s. Právo (ANAG). ISBN 978-80-7263-778-2.
- [2] Finanční limity pro zadávání veřejných zakázek. *Odbor strukturálních fondů Ministerstva vnitra ČR* [online]. Praha, 2013 [cit. 2014-01-10]. Dostupné z: <http://www.osf-mvcr.cz/upozorneni-na-nove-financni-limity-pro-zadavani-verejnych>.
- [3] SARIN, Ashish. *Portlets in Action*. Shelter Island, NY: Manning, c2012, xxvii, 612 s. ISBN 19-351-8254-4.
- [4] WALLS, Craig. *Spring in action*. 3rd ed. Shelter Island: Manning, c2011, xxiii, 400 p. ISBN 19-351-8235-8.
- [5] LINWOOD, Jeff a Dave MINTER. *Beginning Hibernate*. 2nd ed. New York: Apress, c2010, xx, 379 p. ISBN 978-1-4302-2851-6.
- [6] HEDLEY, Jonathan. *Jsoup: Java HTML Parser* [online]. Seattle, © 2009 - 2013 [cit. 2014-01-30]. Dostupné z: <http://jsoup.org/>
- [7] HSU, Regina Obe and Leo. *PostgreSQL: up and running*. 1. ed. Sebastopol, CA: O'Reilly, 2012. ISBN 978-144-9326-333.
- [8] *Google Maps API Web Services* [online]. 11.3.2014 [cit. 2014-02-06]. Dostupné z: <https://developers.google.com/maps/documentation/geocoding/>
- [9] HEROUT, Pavel. *Java a XML*. České Budějovice: Kopp, 2007. 313 s. ISBN 978-80-7232-307-4.
- [10] SPELL, Brett. *Java: programujeme profesionálně*. Vyd. 1. Praha: Computer Press, 2002, 1022 s. ISBN 80-722-6667-5.
- [11] SEZOV, Rich. *Liferay in action: the official guide to Liferay Portal development*. Shelter Island, NY: Manning, 2011, xxiv, 351 p. ISBN 19-351-8282-X.
- [12] BAUER, Christian a Gavin KING. *Hibernate in action*. Greenwich: Manning Publications, 2005, xxiii, 408 s. ISBN 19-323-9415-X.
- [13] OBE, Regina a Leo HSU. *PostGIS in action*. London: Pearson Education [distributor], c2011, xxviii, 492 p. ISBN 978-193-5182-269.

## Seznam použitých symbolů a zkratk

API	Application Programming Interface. Programové rozhraní aplikace.
atd.	a tak dále
CPV	Common procurement vocabulary. Odvětvový číselník pro veřejné zakázky.
č.	číslo
ČR	Česká republika
DPH	daň z přidané hodnoty
EJB	Enterprise JavaBeans. Softwarové komponenty serverových aplikací Java.
HSQldb	HyperSQL DataBase. Relační databázový systém napsaný v Javě.
HTML	HyperText Markup Language. Základní značkovací jazyk, ve kterém jsou psány internetové stránky a jež popisuje jejich vzhled.
IDE	Integrated Development Environment. Integrované vývojové prostředí. Standardní součástí IDE jsou editor zdrojového kódu, kompilátor, případně interpret a debugger.
JNDI	Java Naming and Directory Interface. Java API pro adresářovou službu, které umožňuje softwarovým klientům Javy zjistit a vyhledat data a objekty pomocí jména.
JSP	Java Server Pages. Dokument kombinující statické HTML, dynamický kód a další značkování, na jehož základě je kompilátorem vygenerován Servlet.
Kč	Koruna česká
MIME	Multipurpose Internet Mail Extension. Víceúčelové rozšíření pošty v síti Internet. Přenosový protokol umožňující prostřednictvím pošty na Internetu přenášet i osmibitové (národní) znaky a binární soubory.
MMR ČR	Ministerstvo pro místní rozvoj České republiky
např.	například
NIPEZ	Národní infrastruktura pro elektronické zadávání veřejných zakázek. Modulární členěná soustava informačních systémů podporujících procesy elektronizace zadávání veřejných zakázek.
odst.	odstavec
pozn.	poznámka
resp.	respective
SMTP	Simple Mail Transfer Protocol. Jednoduchý protokol elektronické pošty. Standardní protokol na síti TCP/IP.
tvz.	takzvaný
URL	Uniform Resource Locators. Jednotný popis umístění zdroje. Nejužívanější schéma specifikace dokumentu (jeho umístění a typ) v Internetu.
WML	Wireless Markup Language. Značkovací jazyk založený na jazyce XML umožňující tvorbu online dokumentů pro mobilní zařízení.
XML	eXtensible Markup Language. Rozšiřitelný značkovací jazyk, umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a různé typy dat.
ZVZ	Zákon o veřejných zakázkách.

**SEZNAM OBRÁZKŮ**

<i>Obrázek 1 - Dělení veřejných zakázek [1]</i> .....	15
<i>Obrázek 2 – Portálová infrastruktura [3]</i> .....	21
<i>Obrázek 3 – Role portálového serveru a portletového kontejneru při zpracování požadavku portletu [3]</i> .....	21
<i>Obrázek 4 - Model případů užití</i> .....	54
<i>Obrázek 5 – Asociované třídy k persistentní třídě Tender</i> .....	56
<i>Obrázek 6 – Zjednodušený diagram tříd aplikace Tender Aggregator Data Mining</i> .....	59
<i>Obrázek 7 - Diagram tříd návrhového vzoru GenericDAO pro třídu Tender</i> .....	60
<i>Obrázek 8 – Diagram tříd aplikace Tender Aggregator Portlet</i> .....	67
<i>Obrázek 9 – Diagram tříd aplikace Tender Aggregator Notification</i> .....	70

**SEZNAM TABULEK**

<i>Tabulka 1 - Finanční limity pro jednotlivé kategorie zadavatelů, oblasti a druhy veřejných zakázek, případně kategorie dodávek a služeb stanovuje Nařízení vlády č. 77/2008 Sb., o stanovení finančních limitů pro účely zákona o zadávání veřejných zakázek (ve znění nařízení vlády č. 456/2013 Sb.) [2] .....</i>	<i>17</i>
<i>Tabulka 2 – Fáze zpracování požadavků v portletu [3] .....</i>	<i>28</i>
<i>Tabulka 3 – JSP stránky použité portletem Tender Aggregator Portlet.....</i>	<i>66</i>
<i>Tabulka 4 – Atributy třídy Criteria.....</i>	<i>68</i>
<i>Tabulka 5 - Atributy třídy User.....</i>	<i>68</i>
<i>Tabulka 6 – Atributy třídy PrivatePerson.....</i>	<i>69</i>
<i>Tabulka 7 – Atributy třídy FirmOrBusinessman .....</i>	<i>69</i>

## SEZNAM PŘÍLOH

P I SCREENSHOTY APLIKACE

P II SCREENSHOT NOTIFIKAČNÍHO E-MAILU

# PŘÍLOHA P I: SCREENSHOTY APLIKACE

## Home Page aplikace

Tender Aggregator

[Přihlášení](#) [Registrace](#)

Název zakázky:  a  Lhůta pro podání nabídek od:  do:

a  Název zadavatele:  a  Termín zahájení od:  do:

a  IČO zadavatele:  a  Předpokládaná hodnota od:  do:

a  Stav: a  Kraj: a  Obor:

Zadávací řízení: ZŘ - Hodnocení, ZŘ - Přijem nabídek  
Hlavní město Praha, Středočeský kraj, Jihočeský kraj  
Produkty zemědělství, hospodářské produkty, produkty akvakultury, lesnictví a související produkty, Ropné produkty, paliva, elektrická energie a ostatní zdroje energie, Produkty těžebního průmyslu, kovové suroviny a související produkty

a  Klíčová slova:

Je zobrazeno 1 - 20 položek z celkového počtu 10 100.

Položek na stránku 20  Stránka 1  505

Název zakázky	Popis	Stav zakázky	Lhůta pro podání nabídky
<a href="#">Dodávky kancelářských a hygienických potřeb pro potřeby ústředí České správy sociálního zabezpečení</a>	Předmětem veřejné zakázky jsou dodávky kancelářských a hygienických potřeb pro potřeby ústředí České správy sociálního zabezpečení. Popis požadovaného předmětu plnění a předpokládané počty kusů jsou uvedeny v Příloze č. 1 této výzvy. Dodávky kancelářských a hygienických potřeb budou probíhat přibližně 6krát do roka na základě dílčích objednávek.	ZŘ - Přijem nabídek	20.05.2014 00:00
<a href="#">Podpora Cisco Ironport</a>	Předmětem této veřejné zakázky malého rozsahu je pořízení roční podpory na zařízení Cisco Ironport pro 600 uživatelů v rozsahu uvedeném níže.	ZŘ - Přijem nabídek	19.05.2014 00:00
<a href="#">Spotřební materiál a náhradní díly pro IT (pro ÚVNV)</a>	Ústřední vojenská nemocnice – Vojenská fakultní nemocnice Praha provádí poptávku na Spotřební materiál a náhradní díly pro IT. Požadujeme dodání zboží max. do 5 dnů od objednání a splatnost faktury 60 dnů.	ZŘ - Přijem nabídek	16.05.2014 00:00

## Stránka registrace uživatele – podnikatelského subjektu

Firma/Podnikatel  
 Soukromá osoba

\* Název:  \* IČO:

\* Ulice:  \* Č. popisné/orientační:  DIČ:

\* Město:  \* PSČ:

\* E-mail:

\* Heslo:  \* Kontrola hesla:

Pole označená \* je nutno vyplnit.

### Nastavení kritérií pro zaslání informací o zakázkách na e-mail

**Nastavení oborů**  
Všechny obory:  
Produkty zemědělství, hospodářské produkty, produkty akvakultury, lesnictví a související produ  
Ropné produkty, paliva, elektrická energie a ostatní zdroje energie  
Produkty těžebního průmyslu, kovové suroviny a související produkty  
Potraviny, nápoje, tabák a související produkty  
Zemědělské stroje  
Oděvy, obuv, brašnářské výrobky a doplňky  
Usně a textilie, plastové a pryžové materiály  
Tiskařské výrobky a související produkty  
Chemické výrobky  
Kancelářské a počítačové stroje, zařízení a potřeby mimo nábytek a balíky programů

Vybrané obory:

**Nastavení krajů**  
Všechny kraje: Hlavní město Praha, Středočeský kraj, Jihočeský kraj, Plzeňský kraj, Karlovarský kraj  
Vybrané kraje:

**Nastavení klíčových slov**

**Nastavení hodnoty zakázek v Kč**  
od:  do:

**Nastavení frekvence zaslání notifikací:**  
 denně  
 týdně

**Nastavení zadavatele**  
Název:  IČO:

## Stránka registrace uživatele – soukromé osoby

Tender Aggregator

Firma/Podnikatel  
 Soukromá osoba

\* Jméno:  \* Příjmení:

\* Ulice:  \* Č. popisné/orientační:

\* Město:  \* PSČ:

\* E-mail:

\* Heslo:  \* Kontrola hesla:

Pole označená \* je nutno vyplnit.

**Nastavení kritérií pro zaslání informací o zakázkách na e-mail**

**Nastavení oborů**

Všechny obory:

- Produkty zemědělství, hospodářské produkty, produkty akvakultury, lesnictví a související produ...
- Ropné produkty, paliva, elektrická energie a ostatní zdroje energie
- Produkty těžebního průmyslu, kovové suroviny a související produkty
- Potraviný, nápoje, tabák a související produkty
- Zemědělské stroje
- Oděvy, obuv, brašnářské výrobky a doplňky
- Usně a textilie, plastové a pryžové materiály
- Tiskářské výrobky a související produkty
- Chemické výrobky
- Kancelářské a počítačové stroje, zařízení a potřeby mimo nábytek a balíky programů

Vybrané obory:

**Nastavení krajů**

Všechny kraje:

- Hlavní město Praha
- Středočeský kraj
- Jihočeský kraj
- Plzeňský kraj
- Karlovarský kraj

Vybrané kraje:

**Nastavení klíčových slov**

**Nastavení hodnoty zakázek v Kč**

od:  do:

**Nastavení frekvence zaslání notifikací:**

denně  
 týdně

**Nastavení zadavatele**

Název:  IČO:

## Stránka zobrazující úspěšné provedení registrace uživatele

Tender Aggregator

REGISTRACE ÚSPĚŠNĚ DOKONČENA

[Zpět na domovskou stránku](#)

## Stránka s přihlašovacím formulářem

Tender Aggregator

Přihlášení:

E-mail:  Heslo:

[Zpět](#)

## Stránka účtu registrovaného uživatele – podnikatelského subjektu

**Tender Aggregator**

\* **Název:** Antonín Juran \* **IČO:** 62191144

\* **Ulice:** Štefánikova \* **Číslo popisné/orientační:** 3264 **DIČ:** CZ7412044123

\* **Město:** Zlín \* **PSČ:** 76001

Pole označená \* je nutno vyplnit.

**Nastavení kritérií pro zaslání informací o zakázkách na e-mail**

**Nastavení oborů**

Všechny obory:  
Produkty zemědělství, hospodářské produkty, produkty akvakultury, lesnictví a související produkty  
Ropné produkty, paliva, elektrická energie a ostatní zdroje energie  
Produkty těžebního průmyslu, kovové suroviny a související produkty  
Potraviny, nápoje, tabák a související produkty  
Zemědělské stroje  
Oděvy, obuv, brašnářské výrobky a doplňky  
Usně a textilie, plastové a pryžové materiály  
Tiskařské výrobky a související produkty  
Chemické výrobky  
Kancelářské a počítačové stroje, zařízení a potřeby mimo nábytek a balíky programů

Vybrané obory:  
Oděvy, obuv, brašnářské výrobky a doplňky  
Usně a textilie, plastové a pryžové materiály

**Nastavení krajů**

Všechny kraje:  
Hlavní město Praha  
Středočeský kraj  
Jihočeský kraj  
Plzeňský kraj  
Karlovarský kraj

Vybrané kraje:  
Zlínský kraj  
Plzeňský kraj  
Středočeský kraj

**Nastavení hodnoty zakázek v Kč**

od: 20000.0 do:

**Nastavení zadavatele**

Název: NÁZEV ZADAVATELE IČO: 33333333

**Nastavení klíčových slov**

klíčová slova

**Nastavení frekvence zaslání notifikací:**

denně  
 týdně

## Stránka účtu registrovaného uživatele – soukromé osoby

\* **Jméno:** Antonín \* **Příjmení:** Juran

\* **Ulice:** Štefánikova \* **Číslo popisné/orientační:** 3264

\* **Město:** Zlín \* **PSČ:** 76001

Pole označená \* je nutno vyplnit.

**Nastavení kritérií pro zaslání informací o zakázkách na e-mail**

**Nastavení oborů**

Všechny obory:  
Produkty zemědělství, hospodářské produkty, produkty akvakultury, lesnictví a související produkty  
Ropné produkty, paliva, elektrická energie a ostatní zdroje energie  
Produkty těžebního průmyslu, kovové suroviny a související produkty  
Potraviny, nápoje, tabák a související produkty  
Zemědělské stroje  
Oděvy, obuv, brašnářské výrobky a doplňky  
Usně a textilie, plastové a pryžové materiály  
Tiskařské výrobky a související produkty  
Chemické výrobky  
Kancelářské a počítačové stroje, zařízení a potřeby mimo nábytek a balíky programů

Vybrané obory:  
Kancelářské a počítačové stroje, zařízení a potřeby mimo nábytek a balíky programů

**Nastavení krajů**

Všechny kraje:  
Hlavní město Praha  
Středočeský kraj  
Jihočeský kraj  
Plzeňský kraj  
Karlovarský kraj

Vybrané kraje:  
Zlínský kraj  
Středočeský kraj  
Hlavní město Praha

**Nastavení hodnoty zakázek v Kč**

od: 10000.0 do: 1000000.0

**Nastavení zadavatele**

Název: Název zadavatele IČO: 11111111

**Nastavení klíčových slov**

**Nastavení frekvence zaslání notifikací:**

denně  
 týdně

## Stránka detailu zakázky

Tender Aggregator

**DETAIL ZAKÁZKY**

Název zakázky: Dodávky kancelářských a hygienických potřeb pro potřeby úřadu České správy sociálního zabezpečení  
Předmětem veřejné zakázky jsou dodávky kancelářských a hygienických potřeb pro potřeby úřadu České správy sociálního zabezpečení. Popis požadovaného předmětu plnění a předpokládané počty kusů jsou uvedeny v Příloze č. 1 této výzvy. Dodávky kancelářských a hygienických potřeb budou probíhat přibližně 6krát do roka na základě dílčích objednávek.

Zadavatel: Česká správa sociálního zabezpečení  
Křížová 1292/25  
15000 Praha

Stav zakázky: ZŘ - Přijem nabídek  
Lhůta pro podání nabídek: 20.05.2014 00:00  
Zadávací lhůta: 31.07.2014 00:00  
Předpokládaná hodnota: 1800000 Kč

URL zakázky na e-tržbě: <https://www.gemin.cz/verejne-zakazky/0/davky-kancelarskych-a-hygienickych-potreb-pro-po?tr=https%3A%2F%2Fwww.gemin.cz%2Fverejne-zakazky>  
E-tržbě: Gemin

[Zpět](#)

## Stránka zobrazující úspěšné provedení změn registračních údajů uživatele

Tender Aggregator

**Změny registračních údajů byly úspěšně uloženy**

[Zpět na hlavní stránku](#)

## PŘÍLOHA P II: SCREENSHOT NOTIFIKAČNÍHO E-MAILU

