České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Bruštík**

Studijní program: Otevřená informatika (magisterský)
Obor: Softwarové inženýrství

Název tématu: **Vyhodnocovací systém pro výuku algoritmizace a programování**

Pokyny pro vypracování:

Proveďte rešerši dostupných systému pro spouštění a vyhodnocování programovacích úloh, zejména UVA Judge, ACM ICPC, SPOJ, Dom Judge a dalších systému používaných v programovacích soutěžích. Podle výsledků rešerše navrhněte a realizujte vyhodnocovací systém pro výuku programování a algorimizace na FEL ČVUT. Systém musí umožnit práci registrovaných posluchačů jednotlivých předmětů FEL a také poskytnout funkční rozhraní pro neregistrované uživatele mimo FEL. Systém dovolí učitelům předmětů vkládat a editovat úlohy, nastavovat podmínky vyhodnocování a klasifikace pro skupiny studentů nebo hostů, sledovat statistiky úspěšnosti studentů a jednotlivých úloh. Součástí práce bude webové rozhraní systému a jeho uživatelská i programátorská dokumentace. Zvažte a podle možnosti realizujte rozhraní systému pro spolupráci s informačním systémem KOS, případně dalšími servery FEL. Funkčnost a ovladatelnost systému ověřte zkušebním provozem ve výuce předmětu algoritmizace.

Seznam odborné literatury:

Arlow, J., Neustat, I.: UML 2 a unifikovaný proces vývoje aplikací. Computer Press, ISBN: 978-80-251-1503-9, Praha 2007.
Pressman,R.S.: Software Engineering: A Practitioner's Approach. ISBN 0-07-707936-1, McGraw-Hill, 1992.
Sommerville,I.: Software Engineering. Pearson Education Limited, 2001. ISBN 0-201-39815-X.

Vedoucí: RNDr. Marko Genyk-Berezovskyj

Platnost zadání: do konce letního semestru 2014/2015

[redacted]

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

L.S.

[redacted]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 24. 2. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Vyhodnocovací systém pro výuku algoritmizace a programování

Tomáš Bruštík

Vedoucí práce: RNDr. Marko Genyk-Berezovskyj

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

4. ledna 2015

Poděkování

Tímto bych chtěl poděkovat panu RNDr. Marko Genyk-Berezovskyj za pomoc při vytváření této práce, stejně jaké své rodině a přítelkyni za podporu, kterou mi vykazovali v průběhu vývoje této práce.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 4.1.2015

Tomáš Bu...

.....

Abstract

The goal of this thesis is to explore existing solutions to user programme evaluation and to design and programme a system, which will evaluate sent solutions in chosen languages C, C++ and Java. These solutions will be uploaded to the system via web interface, which will also contain administration properties for managing whole system. There will be a couple of roles, which will have limited and distributed access to resources for management decentralization. Application will also manage users, roles, semesters, courses, tasks, solutions and their evaluation and comparison. It should also enable connection with KOS for importing information of courses.

Abstrakt

Cílem diplomové práce je prozkoumat již existující řešení vyhodnocování zasílaných úloh a navrhnout systém, který bude vyhodnocovat zasílané úlohy ve vybraných programovacích jazycích C, C++ a Java. Tyto úlohy budou zasílané přes webové rozhraní, které zároveň bude obsahovat administrační rozhraní pro správu celého projektu. V systému se bude nacházet několik rolí, jejichž práva budou rozdělená tak, aby umožňovala decentralizaci správy celé aplikace. Systém bude umožňovat správu uživatelů, rolí, semetrů, předmětů, úloh, řešení a jejich vyhodnocování a porovnávání. Zároveň by měl také umožňovat propojení s informačním systémem KOS pro importování informace o předmětech.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Motivace

Důvodem tohoto projektu byla absence aplikačního systému, který by umožňoval lidem mimo systém ČVUT vyzkoušet si vyřešit různé programovací problémy, které by se podobaly těm, které se řeší v různých technických předmětech. Tento systém by mohl sloužit pro propagaci ČVUT na středních školách a skloubil by pohled na studenty vysokých škol. Automatické vyhodnocování by zajistilo co nejvíce podobné podmínky při hodnocení úloh a systém by tak mohl být používán i pro studenty ČVUT. Převedlo by se více zodpovědnosti na nižší role aplikace tak, aby vše nestálo pouze na administrátorech a více lidí tak mohlo spravovat různé aspekty aplikace.

Na škole se v tuto chvíli nachází několik informačních systémů, z nichž jsem přišel do kontaktu hlavně s vyhodnocovacím systémem na Courseware[5]. Tento systém umožňuje omezené automatické vyhodnocování úloh studentů ČVUT, nicméně neumožňuje přístup studentům jiných škol. Také je zde velká zodpovědnost hlavního administrátora, přes kterého chodí všechny požadavky na vytváření dodatečné funkčnosti či správu jednotlivých předmětů a úloh.

1.2 Popis jednotlivých kapitol

1. Úvod - v této kapitole popisuji svoji motivaci k vytvoření tohoto projektu.

2. Popis problému, specifikace cíle - v této kapitole provádím rešerši a vypsání důležitých aspektů podobných projektů této diplomové práci. Hlavně se zaměřuji na projekty Uva Judge, DOM Judge, ACM ICPC a Sharif Judge. Několik z těchto projektů je i vypsáno v zadání této diplomové práce. Vždy se snažím popsat hlavní aspekty a požadavky na spuštění u těchto projektů tam, kde to jde, tedy u těch, které jsou open source a je možné tyto aspekty a požadavky definovat. Zároveň uvádím obecný přehled o těchto projektech. V poslední části této kapitoly se věnuji popsání požadavků z hlediska uživatelských rolí na výsledný systém tak, jak jsme si je definovali spolu s vedoucím práce na začátku a v průběhu vývoje.

3. Analýza částí diplomové práce - zde se věnuji vypsání jednotlivých částí diplomové práce v průběhu návrhu. Také zde popisuji jednotlivé technologie, které byly v průběhu vývoje použity. U každé technologie se snažím popsat její hlavní aspekty a důvody, proč jsem si je vybral.

4. Implementace - této kapitole je věnována hlavní část celé diplomové práce. Popisuji zde hlavní aspekty celé aplikace tak, jak se postupně vyvíjely a popisuji i částečně konkrétní implementace použitých technologií a konfiguraci systému. Objevují se zde také obrázkové ukázky jednotlivých částí aplikace pro lepší pochopení chování aplikace.

5. Testování - v předposlední části diplomové práce se věnuji testování. Uzavřené testování pro několik vybraných uživatelů probíhalo od počátku vývoje aplikace, otevřené testování s několika paralelkami vybraných předmětů (celkem asi 40 studentů) se konalo většinou 1-2x za týden v průběhu 3 měsíců. Zároveň zde také popisuji JUnit testy, které zajišťují otestování hlavní funkčnosti servisní vrstvy aplikace.

6. Závěr - poslední kapitola se věnuje zhodnocení výsledků cílů diplomové práce, úspěšnosti splnění zadání. Zároveň také popisuji možné navrhované vylepšení, které by mohly zvýšit pohodlí ovládání z pohledu uživatele či rozšířit funkcionalitu pro lepší používání.

Kapitola 2

Popis problému, specifikace cíle

2.1 Rešerše podobných projektů

2.1.1 UVa Judge

Uva Judge [30] je online automatizovaný systém pro vyhodnocování řešení programovacích problémů, který byl vytvořen roku 1995. V jeho archivu problémů se nachází více než 4300 problémů. Dá se zde volně registrovat a zasílat vlastní problémy, také řešit otevřené úlohy.

Projekt navštívilo už více než 620 000 unikátních návštěvníků a bylo zde celkově zasláno přes 14 620 200 řešení k vyhodnocení. Kromě samotných programovacích úloh se také pořádají občasné soutěže, kde mají účastníci za úkol vyřešit dané množství programovacích úloh menšího rozsahu v přiděleném časovém okně. Jednotlivé problémy obvykle obsahují několik odstavců popisu úlohy a ukázkový vstup a výstup.

Podporované jazyky

- C;
- C++;
- Pascal;
- Java;

Hlavní aspekty programu a požadavky zde neuvádím, protože Uva Judge není open source a není tak možné determinovat konkrétní specifikace pro spuštění.

2.1.2 DOM Judge

DOM Judge [6] je open source automatizovaný vyhodnocovací systém, na jehož vývoji se podílí v tuto chvíli 12 přispěvatelů. Slouží k pořádání programovacích soutěží. Obsahuje způsob pro zasílání řešení k programovacím úlohám, vyhodnotí je a automaticky umožňuje náhled k výsledkům vyhodnocení skrze webové rozhraní. Jeho hlavní využití je při programovacích soutěžích, kdy jsou řešitelé připojeni přes webové rozhraní a úlohy v přiděleném časovém limitu. Je vyvíjen pod GNU GPL licencí.

Podporované jazyky

- C;
- C++;
- Java;

Hlavní aspekty

- Open Source;
- webové rozhraní pro jednoduchost a přenositelnost;
- škálovatelnost
- modulární systém pro přidávání dalších programovacích jazyků/kompilátorů;
- opakovaná evaluace zaslaných řešení, detailní info odevzdání a výsledků;
- řeší i bezpečnost odevzdávání a vyhodnocování;

Požadavky na spuštění

- alespoň 1 stroj, na kterém běží Linux, s právy administrátora;
- Apache web server s PHP 5 a příkazovou řádkou;
- MySQL nebo MariaDB databázi verze 4.1 či vyšší;
- kompilátory pro programovací jazyky;

2.1.3 ACM ICPC

ACM-ICPC [1] je programovací soutěž mezi univerzitami na celém světě. Koná se každý rok od roku 1977, sponzorována firmou IBM. Je složena z více úrovní, kdy celkové finále se koná formou jedné akce, která trvá 4 dny. Tato soutěž probíhá napříč všemi kontinenty.

Podporované jazyky

- C;
- C++;
- Pascal;
- Java;

Každý tým se skládá ze 3 studentů, kteří mají méně než 5 let výuky na univerzitě před danou soutěží. Během soutěže mají týmy 5 hodin na vyřešení 8-12 úloh. Programy poté běží na testovacích datech, které zjistí korektnost daného řešení. V případě chyb je tým notifikován a může své řešení odevzdat znovu. Vítězem se stává tým, který vyřeší nejvíce úloh v daném čase, a každé odmítnuté řešení je penalizováno. Každý tým má pouze jeden počítač, který může použít k vytváření řešení úloh.

Poslední rok ze soutěže zúčastnilo přes 39 000 studentů z více než 2 000 univerzit a 94 zemí po celém světě. ACM ICPC na lokální úrovni využívá vyhodnocovacích systémů jako jsou Uva Judge, Dom Judge a další ke konání lokální části soutěže.

2.1.4 SPOJ

Sphere online judge neboli SPOJ [25] je vyhodnocovací systém, který běží na internetu. Umožňuje vytváření úloh což vyžaduje komunikaci se správci systému, kdy je potřeba zaslat seznam privilegií dané úlohy. Systém je částečně automatizovaný, kdy libovolný uživatel právě může vytvořit úlohu a podělit se o ní se zbytkem komunity.

Podporované jazyky(je jich až 50, vypisuji jen některé známější)

- C;
- C++;
- Java;
- Python;
- Ruby;
- Python;
- Perl;
- PHP;

Celý projekt má registrováno přes 200 000 uživatelů a je velice aktivní. Samotný projekt je proprietární software, i když z vyjádření administrátorů na fórech jsem vyčetl, že je v budoucnu možné vydat SPOJ pod licencí GPL či pod podobnou licencí. Není tak možné vypsat hlavní aspekty systému ani požadavky na spuštění, stejně jako u Uva Judge.

2.1.5 Sharif Judge

Sharif judge [23] je open source online vyhodnocovací systém. Webové rozhraní je napsáno v PHP, vyhodnocování v pozadí zajišťují BASH skripty. Všechny jazyky kromě Pythonu mají ochrany systému proti zneužití od nahrávaných řešení.

Podporované jazyky

- C;
- C++;
- Python;
- Java;

Hlavní aspekty

- uživatelské role - admin, hlavní učitel, učitel, student;
- bezpečnostní ochrany;
- ochrana plagiátorství - MOSS;
- pravidla pro pozdní odevzdání;
- opakovaná evaluace zaslaných řešení;
- fronta zpracovávaných řešení;

Požadavky na spuštění

- webový server na kterém běží PHP verze 5.3 a novější;
- PHP příkazovou řádku;
- MySQL databázi s mysqli rozšířením pro PHP nebo PostgreSQL;
- nástroje pro kompilaci pro jednotlivé programovací jazyky;

Pro ochranu C/C++/Python zaslaných řešení se využívá program Shield, který pomocí definovaných pravidel zakazuje používání nebezpečných příkazů, které by mohly ohrozit systém - například práce se soubory, fork a další. Pro ochranu Java řešení používá výchozí Java Security Manager. Celkově tedy řeší bezpečnost na úrovni daného zaslaného řešení - kontroluje, zda se používají zákazané metody či přístupy.

2.2 Cíle diplomové práce

Cílem diplomové práce je prozkoumat již existující řešení vyhodnocování zasílaných úloh a navrhnout níže uvedené části systému.

- Administrační rozhraní, které umožní níže uvedené případy užití a rozloží míru zodpovědnosti na více lidí než jen administrátory systému (systém rolí - Admin, Učitel, Student).

- Vyhodnocovací systém pro vybrané programovací jazyky s možností a popisem přidání dalších, definovaný systém porovnávání jednotlivých řešení s možností a popisem přidání dalších, který bude řešit otázky bezpečnosti nahrávaných řešení.
- Umožnit i uživatelům z jiných škol zkusit si vyřešit dané úlohy a vidět jejich výsledky.
- Propojení s jinými systémy ČVUT na případný import dat do systému - hlavně předmětů.

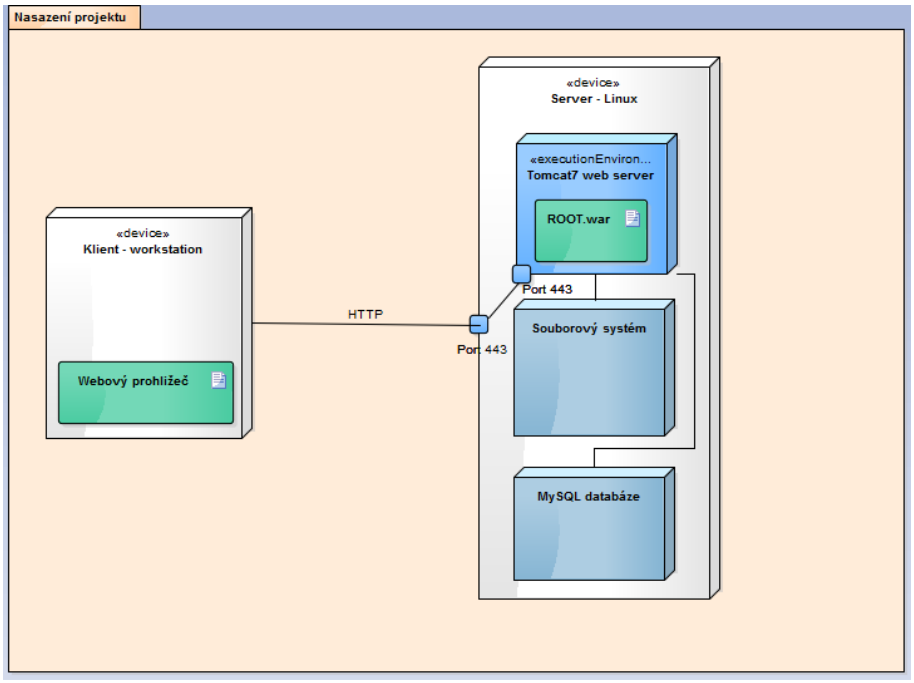
Případy užití

- Administrátor
  - vytváření a spravování uživatelů;
  - vytváření a spravování rolí;
  - vytváření a spravování semestrů;
  - vytváření a spravování předmětů;
  - vytváření a spravování úloh;
  - vytváření a spravování řešení;
  - kontrolovaný přístup k jednotlivým zdrojům (práva, přihlášení uživatele);
  - akce uvedené i u ostatních rolí;
- Student
  - registrovat se na předměty;
  - možnost měnit aktuální semestr;
  - vidět detaily předmětu;
  - vidět detaily úloh;
  - odevzdávání řešení na úlohy;
  - vidět detaily a stav svých řešení a jednotlivých evaluací porovnané vůči referenčnímu řešení;
- Učitel
  - možnost měnit aktuální semestr;
  - možnost vytvářet a spravovat předměty v daném semestru;
  - možnost vytvářet a spravovat úlohy v předmětech;
  - zobrazení hodnocení výsledků řešení vybraných úloh v předmětech;
  - zobrazení všech řešení dané úlohy;
  - rozšířené detaily vyhodnocení (výstupy);
  - zobrazení logu řešení - kdy zasláno, z jaké IP adresy, detaily akce;

Toto byly základní požadavky, které byly postupně konzultovány s vedoucím práce. Další požadavky vyplynuly z vývoje a budou uvedeny v implementační části a vysvětleno jejich řešení. Tyto požadavky se týkají hlavně pohodlnosti práce se systémem, zpětná vazby jednotlivých akcí, přehlednosti zobrazovaných dat, korektnosti přístupu a dalších hledisek. Kompletní seznam práv, které lze přiřadit i nově vytvořeným rolím, je možné naleznout v uživatelském manuálu B.3.

2.3 Návrh struktury projektu

Co se týče struktury projektu, všechny součásti se nachází na jednom serveru. Tento systém musí mít nainstalován distribuci operačního systému Linux Debian Jessie. Co se týče starších verzí, nebo jiných distribucí, velká část těchto systémů by umožnila spuštění celého projektu s jednou výjimkou - LXC. Celkový seznam požadavků na stroj, na kterém se projekt testoval, je možné naleznout v B.1. Na serveru můžeme oddělit 3 součásti projektu.



Obrázek 2.1: Návrh struktury projektu

První částí je samotná webová aplikace, kterou nasazujeme do Tomcatu 7. Pro funkční SSL připojení je potřeba nakonfigurovat Tomcat a umožnit stroji přijímaní požadavků na daném portu. Aplikaci je možné do Tomcatu nasadit přes jeho vlastní webové rozhraní, nebo je možné ji nakopírovat do příslušné složky webapps a restartovat Tomcat. Předtím, než se ale toto bude dít, je potřeba nastavit databázi.

Druhou části je tak samotná MySQL databáze. Je potřeba nainstalovat MySQL server balíček a poté, co je databáze stažená, nakonfigurovat přístup pro aplikaci. Co se týče konkrétních přihlašovacích údajů, je možné se o nich dozvědět více v kapitole 4.1.1. Poté, co

je přístup do databáze nakonfigurován, je nutné ještě vytvořit databázi uploadsystem. Tu je nutné vytvořit ručně a poté spustit SQL skript uploadsystem.sql, který databázi naplní nutnými daty (práva, role, výchozí semestr pro zobrazení a výchozí administrátorský účet).

Třetí částí je souborový systém, do kterého se ukládají nutná data pro správný chod aplikace. Obsahuje všechny nahrané či vytvořené složky a soubory, které aplikace využívá ve webovém rozhraní a vyhodnocovacím systému. Nutné nastavení základní složky se děje v 4.1.1 a do této základní složky je před startem aplikace potřeba zkopírovat složky data, temp, script a log.

V poslední části návrhu si představíme komunikaci mezi klientem a serverem. Klient si otevře webový prohlížeč, zadá příslušnou adresu. Tomcat7 odposlouchává na výchozím portu 443 pro HTTPS požadavky a přesměruje požadavek do webové aplikace. Ta tento požadavek zpracuje a pokud je potřeba, načítá čí ukládá data do databáze a souborového systému. Pokud by v aplikaci vypnulo SSL, to samé platí pro port 80, což je výchozí port pro HTTP požadavky.

Kapitola 3

Analýza částí diplomové práce

3.1 Uživatelské rozhraní

Uživatelské rozhraní by mělo umožňovat pohodlné ovládání, dostatečnou zpětnou vazbu o akcích uživatele a usnadňovat práci. Popíšeme si vybraný případ užití z hlediska zainteresovaných osob.

Z hlediska zadávajícího musí rozhraní umožnit vytvoření předmětu v daném semestru, v tomto předmětu dále vytvořit úlohu, nahrát k ní zadání, referenční řešení, nahrát sadu vstupů a definovat pro ně jednotlivá omezení, jako je třeba časový či paměťový limit. Dále je nutné mít možnost kontrolovat jak jednotlivé výsledky, tak mít přehled o všech výsledcích k dané úloze a přehledné zobrazení celkových výsledků vybraných úloh všech studentů, kteří mají daný předmět zapsán.

Z pohledu studenta rozhraní musí umožňovat zapsání předmětu, následné odevzdávání úloh k vybrané úloze a kontrolování jejich stavu. Poté, co je řešení vyhodnoceno, by měl mít student zpětnou vazbu o tom, jak jeho řešení dopadlo, jak si vedlo v porovnání s referenčním řešením, mít možnost zobrazit si i soubory, které nahrál, aby si mohl zkontrolovat, že je opravdu vše v pořádku. Pokud dojde k nějaké chybě při kompilaci, systém by se měl pokusit specifikovat za jakých okolností k chybě došlo, aby tak student měl přehled o tom, co potřebuje opravit.

Je zde potřeba zmínit, že kromě obecných práv přístupu k vybraným zdrojům, která se vztahují na celé role, je potřeba implementovat vybraný přístup k daným zdrojům. Co se týče studenta, tak jsem definoval následující pravidla.

- Student má přístup pouze k zapsaným předmětům.
- Student má přístup pouze k úlohám v zapsaných předmětech.
- Student má přístup pouze ke svým vlastním řešením.

Z hlediska učitele je potřeba definovat pravidel víc. Tato pravidla slouží k ujištění, že učitelé nebudou moci ovlivňovat zdroje (předměty, úlohy) ostatních učitelů.

- Student má přístup pouze k zapsaným předmětům.

- Student má přístup pouze k úlohám v zapsaných předmětech.
- Student má přístup pouze ke svým vlastním řešením.

Celkově tato pravidla brání zneužití systému a umožňují spravování pouze těch zdrojů, ke kterým by měl mít daný uživatel přístup. Ke konkrétní implementaci těchto pravidel a celého administračního rozhraní se dostaneme v kapitole 4.

3.2 Vyhodnocovací systém

Vyhodnocovací systém má za úkol na požádání rozbalit dané řešení, zjistit, v jakém jazyce je řešení napsáno, zkontrolovat, zda jsou splněny podmínky odevzdání a v případě úspěšné kontroly dané řešení zkompilovat. Poté, co je řešení zkompilováno, se pouští s definovanou sadou vstupů a generují se tak výstupy. Výchozí evaluátor bude fungovat tak, že tyto výstupy porovná s výstupy referenčního řešení a v případě, že se do posledního znaku shodují, vyhodnotí daný testovací případ jako úspěšný a přičte za něj příslušný počet bodů. Při implementaci dalších typů evaluátorů je možné stanovit jiná pravidla.

Vzhledem k tomu, že je možné paralelní vyhodnocování úloh, je potřeba definovat systém, který bude řešení, která je potřeba vyhodnotit, řadit do fronty a postupně vyhodnocovat. Toto omezení je potřeba implementovat z důvodu zachování integrity procesu. Pokud by se dané řešení vyhodnocovalo spolu s dalšími 20 najednou, mohlo by to ovlivnit hlavně časové trvání celého procesu a tím by tam mohlo dojít k nerovným podmínkám mezi vyhodnocováním jednotlivých řešení. Řešení tohoto problému je uvedené v kapitole 4.13.

3.3 Bezpečnost odevzdaných uživatelských řešení

Je potřeba nezapomenout na bezpečnost nahrávaných řešení a odstínění jejich běhu vůči zbytku systému, který se je snaží vyhodnotit. Vzhledem k tomu, že systém spouští libovolně vytvořené programy je zde velké nebezpečí zneužití systému. Obzvlášť je potřeba ochránit zbytek systému, ať už souborový systém, databázi či ovládací prvky a programy systému na kterém aplikace běží. Pokud budeme konkrétní, hlavní problémy spuštění uživatelských řešení spočívají v práci se soubory, přístup k síti, pojmenované roury, sockety, zařízení a sdílená paměť. Je potřeba navrhnout řešení, které se pokusí odizolovat celý překlad a spuštění uživatelského řešení od ostatních zdrojů, ke kterým nemá mít přístup.

3.3.1 Sandbox

Sandbox [22] je označení pro proces oddělení běhu programu od ostatních zdrojů, tedy bezpečnostní mechanismus, který slouží k odizolování běžícího procesu od součástí, ke kterým nemá mít přístup. Pokud budeme mluvit specificky o jednotlivých jazycích, na dané konkrétní programovací jazyky existují implementace tohoto prostředí. Nicméně jejich využití přichází s podmínkami. Hlavním nedostatkem je, že dané implementace se liší, ne vždy jsou dokonalé a hlavně je potřeba pro každý programovací jazyk mít danou implementaci. Ne ke všem programovacím jazykům taky tato možnost existuje. Typicky se v těchto jednotlivých

implementacích kontroluje, zda proces, který běží v sandboxu, nepoužívá určitou sadu zakázaných metod (přístupů). Toto je také nepříjemná vlastnosti, kdy se může v budoucnu objevit nová metoda nebo přístup a v konkrétní implementaci sandboxu na daný programovací jazyk tak nemusí být tato metoda či přístup kontrolována. To pak představuje riziko.

Specifickým příkladem sandboxu je tzv. virtualizace. Vytvoříme virtuální systém, který se chová jako reálný, nicméně má omezené možnosti přístupu k zařízením. Jako zařízení zde beru výše zmíněné možnosti přístupu k souborovému systému, síti, pojmenovaným rourám, paměti a dalším. Toto je vcelku ideální možnost - chci vytvořit stejné podmínky pro všechny uživatele - tedy stejný virtualizovaný systém se stejnými pravidly. Tato pravidla pak definuji tak, aby uživatelé nemohli ohrozit zbytek reálného systému, který tento virtuální systém simuluje. Implementací tohoto přístupu existuje více, nicméně je potřeba brát v potaz systém, na kterém aplikace běží.

3.3.2 AppArmor

AppArmor [2] nebo "aplikační brnění"je bezpečností modul linuxového jádra, který běží pod GNU GPL licencí. Umožňuje administrátorovi systému specifikovat konfigurační soubor pro každý proces, ve kterém se definují pravidla a možné zacházení s tímto procesem. Kromě manuálního vytvoření konfiguračního souboru je také možnost zapnutí tzv. "učení", kdy uživatel zachází s daným procesem tak, jak by ho používal za normálního chodu. Poté, co projde všechny případy, ukončí tento mód a vytvoří se konfigurační soubor, který odpovídá danému využívání, které uživatel předvedl. AppArmor je nabízen jako součást SELinux, který je založen na označování cest k souborům.

Problémem AppArmoru pro náš systém je definování pravidel pro konkrétní program (proces). Pokud bych se snažil AppArmor nastavit pro konkrétní uživatelské řešení, musel bych pokaždé přidávat další a další konfigurační soubory. Zde by byla relace konfiguračního souboru k nahranému řešení 1:1. Jinou možností by bylo definovat pravidla přímo pro spouštění například programu java. Problém ale je, že tato pravidla by se pak aplikovala na všechny programy, které jsou javou spouštěny a tedy i třeba naše java aplikace. Nepodařilo se mi najít způsob jak tyto programy rozlišit, proto jsem AppArmor zavrhl. Našel jsem také totiž ideálnější řešení - tím je LXC.

3.3.3 LXC

LXC neboli Linux containers je virtualizace na úrovni operačního systému, která běží na jádře linux. Díky ní je možné nechat běžet vícero izolovaných Linuxových systému (kontejnerů) v jednom reálném systému. K určení pravidel a omezení pro dané kontejnery se využívá technologie cgroups, která běží na linuxovém jádře. Cgroups také poskytují izolaci jednotlivých jmenných prostorů. LXC prošlo dlouhým vývojem, kdy ve starších verzích nebylo tak zabezpečené, jako podobné systémy v dané době. Nicméně v nejnovějších distribucích umožňuje virtualizaci konkrétní linuxové distribuce a kvalitní oddělení tohoto systému od systému, který ho spouští.

Konkrétní přístup je zde takový, že se vybere distribuce linuxu, kterou se snažíme virtualizovat, a poté se k ní definuje sada pravidel. V této distribuci nainstalujeme potřebné balíčky pro překlad a běh programů tak, jako bychom to dělali v reálném systému. Navrhl

jsem řešení tak, že pro každý programovací jazyk by existoval jeden předpřipravený kontejner. V případě vyvolání akce vyhodnocení řešení naprogramovaném v daném jazyce by se tento kontejner naklonoval, dal by se mu přístup k dočasné složce, do které by se nakopírovala všechna potřebná data - zdrojový kód řešení, testovací vstupy a případné další soubory, které by byly nutné k vyhodnocení. Poté by se provedl zmíněný postup 4.13 a výsledné výstupy by se nakopírovaly do příslušné složky. Porovnaly by se podle konkrétního evaluátoru s referenčními výstupy a došlo by k výsledku.

Krása tohoto řešení spočívá v izolaci nebezpečné části procesu vyhodnocování uživatelského řešení (tedy spuštění uživatelského řešení s testovacími daty) od zbytku systému. Těmto kontejnerům by se neposkytly ani přístupy k síti a případným dalším zdrojům, ke kterým bychom chtěli zabránit v přístupu. Výsledné porovnání evaluátoru už není nebezpečné vůči zbytku systému a je tak možné provádět bez omezení.

Ke navrhovanému řešení tohoto problému je možné nahlédnout v kapitole 4.6.

3.4 Použité technologie

V této kapitole představím jednotlivé technologie, které jsem v diplomové práci použil. Při vybírání jsem se snažil soustředit na používání volně přístupných či open source technologií, aby tak nedošlo k případným problémům v budoucím nasazení aplikace. Konkrétní využití technologií budu uvádět až u daných situací v kapitole 4.

3.4.1 Java EE

Java EE [9] je součást platformy Java, která je určená pro vývoj informačních systémů včetně síťových a webových služeb. Poskytuje API pro vývoj velkých, několikavrstvých, škálovatelných a bezpečných aplikací. Rozšiřuje tak standardní edici platforma Java SE a umožňuje hlavně vývoj systémů skládajících se z modulů, běžících na webovém serveru. Systémy jsou vyvíjeny zejména v programovacím jazyku Java, důraz je kladen hlavně na konvence oproti konfiguraci a využívá anotací k označení chování jednotlivých částí programu. Často se využívá jazyku XML pro konfiguraci aplikace.

Java EE prošlo od roku 2000 velkým vývojem. Postupně se přidala podpora pro JSP, JSF , REST, MVC architekturu, Java Servlet, Java Message Service, práci s formátem JSON a další. Přes stálou dominanci řešení webových aplikací pomocí LAMP, Java EE získává postupně na popularitě. Její nevýhodou je hlavně vysoká cena za počáteční konfiguraci a rozjezd aplikace. Oproti výše zmíněnému LAMP přístupu, který je relativně rychle nastaven a připraven k nasazení aplikace do produkčního prostředí, Java EE vyžaduje několik souborů, obsahujících konfiguraci pro webový server, práci s databází, zabezpečení, transakce, RESTful URL a další.

Nicméně tuto nevýhodu vynahrazuje hlavně u větších aplikací, kde počáteční cena za konfiguraci je vynahrazena udržitelností aplikace, konzistencí a čitelností. Jak bylo výše zmíněno, aplikace se skládá z modulů a tak se dá i lépe rozšiřovat. Existuje dnes mnoho aplikačních rámců, které operují nad Java EE a spolu s jejím standardem umožňují rychlý a kontinuální vývoj webových aplikací. Je zde lepší podpora pro nové technologie, vzniká mnoho open source projektů nad touto platformou. Jeden z těchto aplikačních rámců jsem využil i při vývoji této práce - Spring Framework.

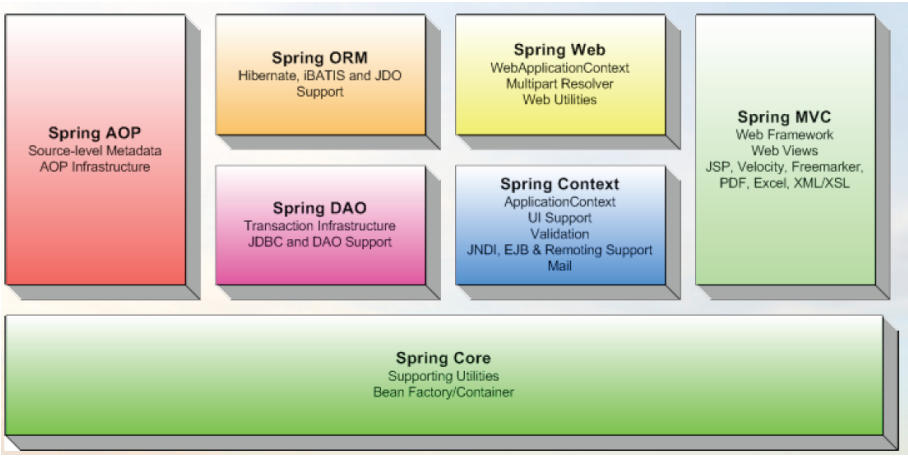
3.4.2 Spring Framework

Spring framework [26] je open source aplikační rámec pro Java platformu. Je vývijen pod Apache licencí, iplementuje návrhový vzor IoC, kdy jednotlivé části programu dostávají tok zpracování požadavků v programu od obecné, znovupoužitelné knihovny. Je opakem klasického procedurálního programování, kdy konkrétní napsané části programu volají obecné knihovny ke zpracování požadavku.

Tento aplikační rámec sice nenutí uživatele do nějakého konkrétního návrhového modelu, nicméně se stal velice populární alternativou k zaběhnutému EJB modelu. Jádro tohoto rámce mohou použít všechny Java programy, nicméně obsahuje rozšíření pro práci s webovými aplikacemi.

Hlavní aspekty

- Umožňuje vývoj webových aplikací za použití POJO objektů. Výhoda POJO objektů je v tom, že není potřeba použít separátní EJB kontejner jako aplikační server, ale lze použít pouze robustní servletový kontejner jako je například Tomcat7.
- Skládá se z jednotlivých modulů, které je možné použít jako části nebo i jako celek. Pokud je tedy potřeba použít pouze určité části, je možné použít pouze ty, které jsou potřeba a ostatní ignorovat.
- Využívá několik existujících technologií jako například několik ORM aplikačních rámců, logovacích prostředků a další.
- Testování aplikace napsané ve Springu je také velice jednoduché, protože kód, který je závislý na prostředí, je přesunut do tohoto rámce. Podporuje testování s JUnit testy a ulehčuje tak výrazně vývoj celé aplikace.
- Injekce závislostí - stará se tvorbu, údržbu a zánik jednotlivých instancí anotovaných tříd, kdy tyto třídy jsou implementované v podstatě jako singletony a vkládají se tak do jiných tříd, které jsou na nich závislé. Odpadá tak tedy potřeba údržby jednotlivých instancí a závislosti jednotlivých tříd na jiných je jednoduché implementovat i udržovat.
- Webový aplikační rámec je MVC rámec, který poskytuje alternativu k webovým aplikačním rámcům jako je Struts nebo jiným méně populárním rámcům.
- Poskytuje podporu pro technologie pro práci s databázemi jako jsou JPA, Hibernate a další.
- Poskytuje vhodné API pro práci s výjimkami, které vyhazují specifické technologie jako JDBC, Hibernate a další. Je možné tak s těmito výjimkami pracovat konzistentně jako s klasickými, nekontrolovanými výjimkami.
- Poskytuje také rozhraní pro konzistentní práci s transakcemi. Tyto transakce je možné škálovat i pro větší množství databází.



Obrázek 3.1: Přehled modulů v aplikačních rámci Spring Framework

Více popíšu injekci závislostí, protože to je hlavním znakem IoC ve Springu. Implementuje se způsobem, že se napíše rozhraní pro třídu, kterou chceme později injektovat do jiných. Daná konkrétní třída pak toto rozhraní bude implementovat, anotuje se pomocí jedné z anotací, které značí Spring komponentu(@Service, @Component, @Repository a další). Poté na místě, kde chceme vložit závislost na danou třídu, se vloží anotace @Autowired a pod tuto anotaci se vloží objekt typu daného rozhraní. Pokud existuje pouze jedna třída, která implementuje dané rozhraní, Spring si tuto komponentu vyhledá a dosadí správnou instanci na dané místo při běhu programu.

Spring framework má vlastní AOP aplikační rámec z důvodu přesvědčení, že je možné poskytnout základní AOP vlastnosti bez velké komplexnosti ať už v návrhu, implementaci či konfiguraci. Je založen na proxy návrhovém vzoru, který je konfigurován až za běhu aplikace. Odebírá to tak nutnost kroku kompilace, nicméně je možné používat pouze veřejné metody, které jsou na dostupné na existujících objektech.

3.4.3 Spring Security

Spring security [27] je open source projekt na Java EE platformě, který slouží k zabezpečení zdrojů v aplikaci před neoprávněným používáním. Je vyvíjen pod Apache licencí a od roku 2003, kdy byl založen, doznal mnoho změn a stal se jedním z nejoblíbenějších nástrojů při zabezpečování webových aplikací, zvlášť při spolupráci se Spring Frameworkem.

Hlavní aspekty

- Autentifikace a Autorizace - jeden z klíčových konceptů rámce, nejdříve autentifikovat uživatele podle jím zadaných údajů a poté mu zpřístupnit konrétní zdroje, tedy autorizovat jeho akce.
- Bezpečnostní filtry - autorizování akcí autentifikovaného uživatele probíhá přes sadu filtrů definovanou buď v konfiguračním XML souboru nebo je možné kontrolovat přístup

k jednotlivým definovaným metodám ve Spring komponentách přes anotace (@PreAuthorize a další).

- Stará se o definování přihlášení uživatele, kam bude přesměrován po přihlášení, odhlášení uživatele, kam bude přesměrován při odhlášení, definuje třídu, která se stará o zamíntuté pokusy o autentifikaci či autorizaci.
- Stará se i o chování při vypršení relace.
- Je možné definovat chování i pro zabezpečený protokol HTTPS.

Je výborným pomocníkem při zabezpečování zdrojů pomocí povolených práv pro autentifikované uživatele, nicméně z mého pohledu není dostatečný co se týče přístupu na vybrané zdroje pro vybrané uživatele. Lze si představit příklad, který jsem uváděl v kapitole 2.2, kdy student má mít přístup pouze ke svým řešením a ne k řešením ostatních studentů. Nestačí tak pouze napsat povolení pro přístup k detailu řešení, ale je potřeba další vrstva autorizace akce a to na úrovni požadavku konkrétního uživatele.

3.4.4 SSL/TLS

Secure Sockets Layer(SSL) a její následník Transport Layer Security (TLS) [29] se používají jako ochranná vrstva mezi vrstvou transportní a aplikační. Slouží k šifrování a zabezpečení komunikace hlavně na internetu. Pro šifrování a autentizaci uživatelů se používají dva typy klíčů - soukromý a veřejný. Veřejný klíč je známý všem, nicméně soukromý klíč zná pouze jeho majitel. Proces autentizace popisuje obrázek 3.2.

V tomto projektu je používán protokol HTTPS což je nadstavba protokolu HTTP, která je rozšířena o bezpečnostní složku. Tato složka zaručuje šifrování dat právě pomocí protokolu SSL/TLS. Slouží tak k zabezpečení komunikace před případnými útočníky, kteří by se snažili odposlouchávat citlivé informace, jako jsou například data k přihlášení.

3.4.5 MySQL

MySQL [17] je jedním z nejpoužívanějších relačních databázových systému. Jednou z jeho výhod pro náš projekt je, že je open source a běží pod GNU GPL licencí. Je napsán v C++ a je možné jej pohodlně používat v operačních systémech Linux. Pro mě osobně je to jednoznačná volba relační databáze, mám s ní dobré zkušenosti za několik posledních let a umožňuje jak jednoduché používání pro méně zkušené uživatele, tak mnoho pokročilých nástrojů pro práci s daty.

Hlavní aspekty

- Pro práci s daty používá obecně zaběhnutý jazyk SQL.
- Podporuje práci s mnoha jazyky, pro tento projekt je důležitá podpora pro jazyk Java.
- Pracuje velice rychle a dokáže si poradit i většími sadami dat, zatím jsem neměl problémy v řádech desetitisíců dat s operacemi nad těmito daty (podle oficiální dokumentace je možné v jedné databázi mít až desítky milionů záznamů).

- Výhodou open source je volná dostupnout a používání, není tedy potřeba za používání platit a navíc je možné si kód databáze upravit podle svých představ.

Co se týče práce s MySQL, je možné s ní pracovat přes příkazovou řádku nebo je možné využít mnoho externích nástrojů pro připojení do databáze a zde operovat nad daty. Zálohování databáze lze dělat přes nástroj mysqldump.

3.4.6 Hibernate

Hibernate [8] je technologie pro mapování objektů na relační data. Je to open source projekt, který byl založen pod LGPL licencí. Slouží jako aplikační rámec k perzistentnímu ukládání dat do relačních databázových systémů. Používání Hibernate vede k velké redukci problémů co se týče perzistentního ukládání dat, snižuje riziko nekonzistence chování a v případě chybných operací poskytuje srozumitelnou zpětnou vazbu. Dá se představit jako most mezi aplikační částí projektu a perzistentní částí. Hibernate je implementace JPA aplikačního rozhraní, specificky nejnovější verze implementují JPA 2.0 rozhraní, které bylo přijato v JSR 317.

Hlavní aspekty

- Mapování Java objektů do databázových tabulek lze dosáhnout pomocí anotací nebo konfiguračního XML souboru.
- Poskytuje jednoduché a srozumitelné API pro ukládání a obnovení dat přímo z databáze.
- V konfiguračním souboru lze nastavit, jak se bude aplikace chovat vůči databázovému schématu při spuštění - vytvoření, smazání a vytvoření, validace, aktualizace.,
- K operacím nepotřebuje aplikační server.
- Snaží se optimalizovat dotazy do databáze pro snížení zátěže na databázový systém.
- Poskytuje jazyk HQL pro jednodušší dotazy přes Java objekty.

Je důležité zmínit podporu pro databázi MySQL, kterou v tomto projektu používám. Také neopomenu podporu pro transakční zpracování požadavků, která je v projektu potřeba.

3.4.7 Primefaces

Primefaces [21] je populární JSF UI aplikační rámec, který se používá při vývoji webových aplikací k přehlednému uživatelskému rozhraní. Je to open source projekt, který byl založen pod Apache licencí. V tuto chvíli nejnovější verze 5.1 obsahuje velké množství nástrojů pro práci s daty při vývoji. Uvedu zde důležité vlastnosti a nástroje a poté v implementační části se zaměřím na konkrétní implementaci a použití těchto nástrojů.

Hlavní aspekty

- Malé množství nutné počáteční konfigurace k rozběhnutí Primefaces v aplikaci.
- Vysoká komunitní základna při testování, řešení chyb v aplikačním rámci a velké množství aktualizací.
- Přes 100 komponent, které usnadňují práci s daty, jejich zobrazení a následné zpracování.
- Možné AJAXové zpracování požadavků od uživatele.
- Zobrazování a práce s dialogy.
- Validace formulářů na straně klienta.
- Velká variace zobrazení celé aplikace - vše stojí na principu témat, kdy se na jednom místě zvolí příslušné téma a tím se změní vzhled celého uživatelského rozhraní.
- Vázání hodnot v uživatelském rozhraní na data v entitách v dalších vrstvách aplikace.

Klíčové komponenty, které využívám v aplikaci

- DataList - slouží k zobrazení seznamu dat.
- Dialog - slouží k zobrazení dialogu (ať už interaktivního, modálního nebo klasického).
- Form - slouží k zobrazení formuláře, případně s předvyplněnými daty.
- CommandButton, CommandLink.
- FileUpload - slouží k přehlednému nahrávání souborů.
- FileDownload - slouží k možnosti stahování souborů, soubory je možné načítat i formou streamu.
- MenuBar a MenuItem - slouží k zobrazení seznamových menu.
- UI-Include - slouží k nahrání jiného souboru, umožňuje tvoření View vrstvy z několika znovupoužitelných zdrojů (tedy šablony).
- AJAXové eventy - na velké množství akcí, které jsou propojeny s ostatními komponentami.
- Media - slouží k zobrazování různých typů médií jako jsou například PDF soubory, hudba, videa a další objekty.

3.4.8 Pretty Faces

Pretty Faces [20] je open source knihovna, která slouží k přepisování URL. Obsahuje několik dalších výborných nástrojů i mimo URL přepisování, jako je například možnost před renderováním samotné stránky zavolat vybrané metody v aplikaci. Proto se dá využít i jako dodatečná vrstva ochrany zdrojů před neautorizovaným používáním. Ve svém projektu ji používám jak pro přepisování URL do uživatelsky přijatelnější podoby, tak právě jako dodatečnou složku ochrany zdrojů.

Hlavní aspekty

- Velice jednoduchá konfigurace - přidá se závislost v Maven projektu do konfiguračního souboru pom.xml a vytvoří se XML konfigurační soubor pretty-config, ve kterém se definují jednotlivá pravidla.
- URL přepisování (http://example.com/resources/task.xhtml?id=85 naproti příjemnější podobě z pohledu uživatele http://example.com/task/85).
- Možnost zavolání akcí v různých fázích JSF renderování, jejíž URL se přepisuje
- Injekce dat přímo do proměnných v aplikaci.
- Zpracování jak parametrů v cestě URL, tak query parametrů.
- Možnost generalizace pravidel, kdy můžou pravidla-děti dědit od pravidel-rodičů a rozšiřovat jejich chování.
- Možnost využití standardních JSF validátorů při práci s parametry.

3.4.9 Log4j a GmailSMTPAppender

Log4j [15] je open source knihovna, která je dostupná pod Apache licencí a slouží k logování v Java projektech. Umožňuje několik forem logování, z nichž důležité pro tento projekt jsou formy logování do souboru a posílání logu e-mailem. Co se týče zasílání logů e-mailem, zde jsem si dovolil využít alternativy k standardnímu SMTPAppenderu, který poskytuje Log4j knihovna. Je to z důvodu problému této knihovny s odesíláním logů na Gmail e-maily, protože neposkytuje SSL podporu. Proto jsem využil přímo GmailSMTPAppenderu [24] s jednou drobnou úpravou od sebe a to na řádku 123, kdy jsem dodal UTF-8 kódování pro správné zobrazování diakritiky. GmailSMTPAppender je v podstatě rozšíření SMTPAppenderu od Log4j, které mění metody na iniciaci spojení a posílání zásobníku logů.

Hlavní aspekty

- Jednoduché připojení do projektu - přidání závislosti do pom.xml souboru a vytvoření konfiguračního souboru log4j.properties.
- Možnost logování v určitém formátu
- Možnost logování do souboru, do konzole, do e-mailu.

- Možnost logování až od určité úrovně chyby(INFO, WARN, ERROR, FATAL), vypnutí logování (OFF), zapnutí režimu detailnějších zpráv (DEBUG, TRACE).

3.4.10 Propojení s ČVUT systémem - KOSApi, REST, OAuth2, Jersey

3.4.10.1 KOSApi

Citace z [13]: *"KOSapi poskytuje aplikační rozhraní (API) v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi KOS. Odstraňuje nutnost zpracovávání exportů, neustálou duplikaci všech dat a potíže s jejich udržováním. RESTové služby staví na osvědčených konceptech webu jakožto distribuovaného prostředí vzájemně provázaných informací. KOSapi umožňuje a podporuje vznik školních i studentských aplikací, které pro svou činnost vyžadují online aplikační přístup k datům souvisejícím s výukou. Tvoří také jednu ze základních komponent nově vznikajícího Informačního systému ČVUT (IS ČVUT). Od verze 3.2 je KOSapi napojené přímo na databázi KOS a poskytuje přístup pro čtení k vybrané podmnožině dat týkající se bílé knihy, rozvrhů, studentů apod. Starší verze KOSapi získávaly data z databázových exportů KOS (již jsou mimo provoz)."*

Pro tento projekt je důležité toto API z hlediska importování dat o předmětech. Je nutné se do KOSApi připojit, autentifikovat a stahovat data o předěmětech v aktuálním semestru. Tato data se poté využívají jako základ při volání vlastních funkcí na vytváření předmětů vzhledem k aktuálnímu semestru v systému. Pro autentifikaci v KOSapi se využívá technologie OAuth2.

3.4.10.2 REST

Citace z [4]: *"Zkratka REST znamená Representational State Transfer. Spoléhá se na bezstavovost, klient-server architekturu a pracuje přes HTTP protokol. REST je styl architektury pro návrh síťových aplikací. Hlavní myšlenkou je využívání dnes již velice zaběhnutého protokolu HTTP pro přenos zpráv, spíše než využívání velmi komplexních mechanismů jako CORBA, RPC nebo SOAP. Můžeme také říci, že v REST využívá možností WWW (World Wide Web) Aplikace postavené na tomto principu využívají HTTP požadavků k posílání, čtení, aktualizování a mazání dat. Tedy využívají HTTP protokol pro všechny 4 operace z CRUD principu práce s daty."*

RESTful principy, které se budu snažit v projektu dodržet

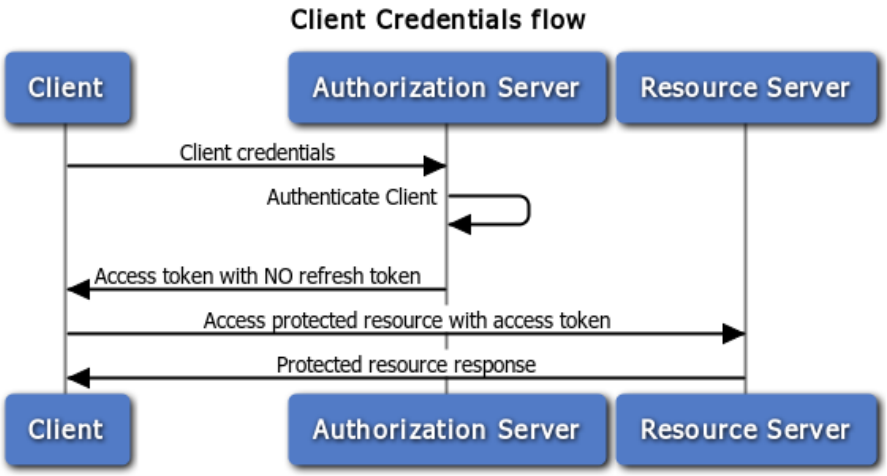
- Klient-server model - klient se serverem komunikuje prostřednictvím HTTP požadavků a webového prohlížeče, server obstarává veškerou logiku.
- Bezestavovost - každý jednotlivý požadavek musí obsahovat všechny data potřebná pro jeho zpracování. Pokud tedy daný požadavek budeme opakovat ve stejných podmínkách, mělo by dojít ke stejnému zpracování a výsledku.
- Možnost cachovat odpovědi - tato možnost existuje implicitně při cachování odpovědí na HTTP požadavky.

- Systém skládající se z vrstev (klient neví, zda je připojen přímo k finálnímu serveru nebo k nějakému prostředníku) - vzhledem k faktu, že load balancer 6.2.1 je navrhován pouze jako případné vylepšení, je klient připojen přímo k serveru, který sám zpracovává požadavky.
- Uniformní rozhraní - jednoznačné definování zdrojů (/task/87), akcí (/task/87/edit).

Ve své bakalářské práci [4] rozebírám architekturu REST mnohem více dopodrobna, pro větší detaily tedy odkazuji do ní.

3.4.10.3 OAuth2

OAuth verze 2.0 [18] je autorizační aplikační rámec, který slouží k autorizaci klienta. V celém procesu figurují 3 strany, klient, autorizační server a server se zdroji. Autorizační server umožňuje klientovi přístup k serveru se zdroji po úspešně autorizaci. Možností této autorizace je několik (Authorization Code, Implicit, Resource owner password credentials, Client Credentials). Pro tento projekt jsem zvolil metodu Client Credentials, jejíž komunikaci je možné vidět na obrázku 3.2. Důvod této volby je jednoduchý - při dotazu na zdroje z KOSapi se nepotřebujeme vázat na konkrétního uživatele, potřebujeme obecně data o všech předmětech. Tato data jsou určitým způsobem filtrována, aby nedošlo ke stahování příliš velkého množství dat a zpomalení aplikace.



Obrázek 3.2: Proces autentizace uživatele

Obrázek krátce popíšu. Nejdřívě je potřeba získat tzv. Client Credentials. Tyto se získají od autorizačního serveru před jakoukoli autentizací. Poté se tyto údaje používají při autorizačním požadavku na autorizační server. Ten vyhodnotí, zda má povolit autorizaci či ne. Pokud se autorizace povede, vrací nám úspěšnou odpověď, která obsahuje Access Token. Tento token má dočasnou platnost a při dotazu na zdroje Resource Serveru se přibaluje vždy do požadavku. Resource server tak ví, že byl klient autorizován od autorizačního serveru a může vrátit data ze zdroje. Konkrétní implementace této technologie se nachází v kapitole 4.10.

3.4.10.4 Jersey

V následující citaci je popsána technologie Jersey [11].

Citace z [4]: *"Jersey už využíváme jako součást serverové části aplikace. Obsahuje také ale Client API pro jednoduchou komunikaci se serverem. Nejdříve je potřeba vytvořit instanci třídy Client. Poté přes tuto instanci klient komunikuje se serverem přes funkce put, get, post a delete. Tyto funkce vytváří HTTP požadavky na server a vracejí odpověď ve formě řetězce. Tento řetězec se pak dále zpracovává."*

V tomto projektu budou pomocí Jersey knihovny vytvářeny požadavky jak na autorizační server pro získaní access tokenu, tak pro požadavky na KOSapi pro získaní potřebných dat pro importování předmětů.

3.4.11 Jenkins

Jenkins [**?**] je open source projekt napsaný v Jave, který je dostupný pod MIT licencí. Slouží jako nástroj při kontinuálním vývoji aplikací. Ve své základní podobě umožňuje definovaní tzv. "Jobs", což jsou opakující se činnosti při vývoji. Je možné zde zmínit opakované nasazování aplikace při změnách kódu, periodické provádění činností, jako je zálohování souborového systému, zálohování databáze, provádění webových požadavků a další. Při instalaci určitých pluginů je tak možné nastavit Jenkins tak, aby se periodicky vše zálohovalo, automaticky se nasazovaly nové verze nebo například rozlišovat testovací a produkční provoz. Konkrétně pro tento projekt je důležitá podpora pro GIT, Maven, Tomcat nasazování aplikace, přehledné zobrazení výsledků JUnit testů, přehledné zobrazení úspěšnosti nasazení aplikace a periodické spouštění určitých operací jako je zálohování dat v souborovém systému a databáze.

Kapitola 4

Implementace

4.1 Konfigurace projektu

4.1.1 application.properties

Níže uvádím a popisuji jednotlivé části konfiguračního souboru application.properties.

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/uploadsystem?characterEncoding=UTF-8&
    transformedBitIsBoolean=true
jdbc.username=root
jdbc.password=root
```

Seznam parametrů, které konfigurují připojení k databázi. Používá se standardní mysql ovladač, url je trochu více konfigurovaná z důvodu kódování UTF-8 a odstranění problému pro překládání hodnot typu Boolean. Prvky username a password zde slouží jako konfigurace uživatele, který se bude připojovat do databáze.

```
hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql=false
hibernate.hbm2ddl.auto=validate
hibernate.generate_statistics=true
```

Konfigurační prvky pro hibernate. Paremeter dialect je zvolen pro komunikaci s MySQL databází, show_sql určuje, zda se budou logovat data z požadavků do databáze, hbm2ddl.auto určuje, co se děje se schématem databáze při spuštění projektu a generate_statistics označuje, zda se budou generovat statistické údaje.

```
root_dir=/home/tomcat7/
```

Parametr root_dir označuje domovskou složku celého projektu, z ní se budou odvíjet cesty do všech ostatních částí projektu.

4.1.2 web.xml

Níže uvádím a popisuji jednotlivé části konfiguračního souboru web.xml.

```
<session-config>
   <session-timeout>10</session-timeout>
   <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Konfigurace relace, je nastaven 10 minutový časový limit pro vypršení.

```
<servlet>
   <servlet-name>Faces Servlet</servlet-name>
   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>Faces Servlet</servlet-name>
   <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<context-param>
   <param-name>primefaces.THEME</param-name>
   <param-value>afterwork</param-value>
</context-param>
```

Konfigurace Faces servletu, kdy se definuje na jakých zdrojích bude tento servlet odposlouchávat a téma afterwork pro vzhled aplikace.

```
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
   <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Konfigurace Spring security filtru, který bude jako první určovat chování při zpracování požadavku.

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
        classpath:spring/applicationContext.xml /WEB-INF/security.xml
```

```
   </param-value>
</context-param>
<context-param>
   <param-name>log4j-config-location</param-name>
   <param-value>classpath:log4j.properties</param-value>
</context-param>
```

Konfigurace kontextových souborů pro spuštění aplikace, Spring security a Log4j komponent.

```
<filter>
   <filter-name>PrimeFaces FileUpload Filter</filter-name>
   <filter-class>org.primefaces.webapp.filter.FileUploadFilter</filter-class>
   <init-param>
      <param-name>thresholdSize</param-name>
      <param-value>51200000</param-value>
   </init-param>
   <init-param>
      <param-name>uploadDirectory</param-name>
      <param-value>${root_dir}temp/</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>PrimeFaces FileUpload Filter</filter-name>
   <servlet-name>Faces Servlet</servlet-name>
   <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Konfiguarce Primefaces filtru pro nahrávání souborů uživatelem na server. Nastavuje se zde velikost a cílová složka pro dočasně nahrané soubory.

```
<error-page>
   <error-code>403</error-code>
   <location>/resources/xhtml/denied.xhtml</location>
</error-page>
<error-page>
   <location>/resources/xhtml/error.xhtml</location>
</error-page>
```

Konfigurace chybových stránek, které se zobrazují uživateli ať už při vyhození výjimky nebo při odepření přístupu na požadovanou stránku.

4.1.3 applicationContext.xml

Nastavuje se základní balíček pro komponenty, v tomto balíčku se Spring bude snažit najít jednotlivé komponenty. Nastavuje se soubor s parametry, které se dále používají v konfiguraci - application.properties. Importuje se zde soubor applicationContext-jpa.xml a nastavuje definice ViewScope, kterou Spring implicitně nemá. Tento ViewScope není moje práce, převzal jsem ho z následujícího zdroje [31]. Část konfigurace ViewScope se také nachází v souboru

faces-config.xml (Třída ViewScopeCallbackRegistrar také převzaná ze zdroje [31] na události PostConstructViewMapEvent a PreDestroyViewMapEvent). Níže uvádím a popisuji jednotlivé části konfiguračního souboru applicationContext.xml.

```
<context:component-scan base-package="cz.cvut.fel" />
<context:property-placeholder location="classpath:application.properties" />
<import resource="applicationContext-jpa.xml"/>
<bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">
<property name="scopes">
    <map>
        <entry key="view">
                <bean class="cz.cvut.fel.helper.ViewScope"/>
        </entry>
    </map>
</property>
```

V souboru applicationContext-jpa.xml se poté nachází konfigurace jednotlivých prvků pro komunikaci s databází.

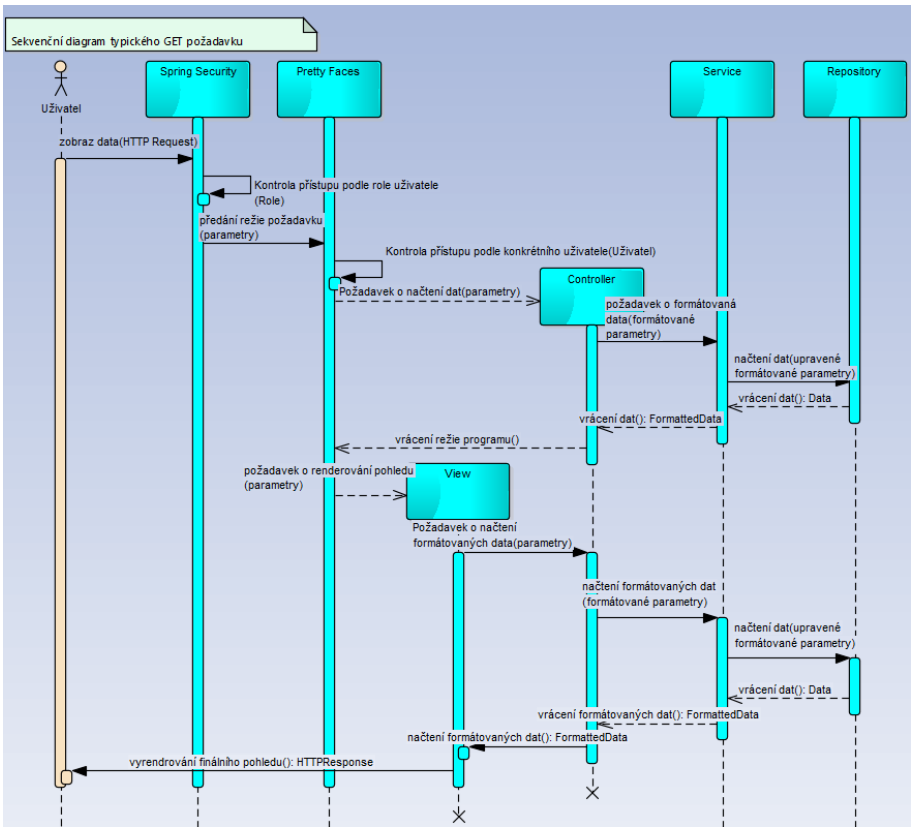
```
<property name="jpaProperties">
        <props>
           <prop key="hibernate.connection.provider_class">
               org.hibernate.connection.C3P0ConnectionProvider</prop>
           <prop key="hibernate.dialect">${hibernate.dialect}</prop>
           <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
           <prop
               key="hibernate.connection.driver_class">${jdbc.driverClassName}</prop>
           <prop key="hibernate.connection.url">${jdbc.url}</prop>
           <prop key="hibernate.connection.username">${jdbc.username}</prop>
           <prop key="hibernate.connection.password">${jdbc.password}</prop>
           <prop key="hibernate.c3p0.min_size">20</prop>
      <prop key="hibernate.c3p0.max_size">100</prop>
      <prop key="hibernate.c3p0.timeout">300</prop>
      <prop key="hibernate.c3p0.max_statements">50</prop>
      <prop key="hibernate.c3p0.idle_test_period">28000</prop>
        </props>
</property>
```

Jako hodně důležitou bych označil konfiguraci c3p0 poolu, která slouží nejen k lepšímu výkonu co se týče dotazů do databáze, ale také řeší problém s dlouhou neaktivitou. Po dlouhé neaktivitě v systému docházelo k vypršení časového limitu spojení do databáze (výchozí hodnota tohoto časového limitu je 28800 vteřin). Po vypršení tohoto limitu a připojení libovolného uživatele do systému chvíli systému trvalo, než se znovu připojil do databáze, a mezitím libovolné požadavky byly odmítány jako neplatné.

4.2 Implementace návrhu projektu

V kapitole 2.3 jsem vytvořil první návrh na rozdělení části aplikace. V této kapitole se budu věnovat konkrétnímu rozdělení aplikace na vrstvy, vysvětlení, co jednotlivé vrstvy dělají a co

zajišťují. Toto rozdělení slouží k jednoznačné definici hranic chování aplikace, a je tedy možné při přidání nových částí aplikace jednoznačně definovat, do jaké části danou komponentu přidat. Budu zde popisovat nejen obecně o co se jednotlivé vrstvy starají, ale také které dříve vyřčené problémy dané vrstvy řeší a jak.



Obrázek 4.1: Proces zpracování požadavku

Obrázek 4.1 popisuje proces požadavku od chvíle kdy ho uživatel poslal na server a jednotlivé části serverové aplikace, které daný požadavek zpracovávají. V celém sekvenčním diagramu předpokládám, že uživatel má dostatečná práva na zobrazení požadovaných dat. Pokud je nemá, uživatel dostane HTTP odpověd s kódem 403 - Přístup zamítnut. V jednotlivých vrstvách vždy budu popisovat Scope(v jakém rozsahu daná komponenty dané vrstvy operují). Tento rozsah může být Application(celá aplikace), Session(relace), View(pohled - požadavek). Také můžu ve zbytku dokumentu používat jak české překlady daných vrstev, tak přímo jejich názvy(Controller - řidící jednotka, View - pohled atd.). Budu snažit jednotně používat názvy těchto vrstev.

4.2.1 Vrstva Spring Security

Scope: Application

Tento filtr zprácovává požadavek jako první v řadě. Jeho úkolem je zjistit, zda má uživatelova role dostatečná práva k zobrazení daného zdroje dat. Pokud uživatel právo nemá,

je přesměrován na stránku /denied. Pokud právo má, je přesměrován do dalšího filtr,u jímž je Pretty Faces filter. Konrétní vybrané pravidlo v konfiguračním souboru security.xml je uvedné níže.

```
<intercept-url pattern="/**" access="isAuthenticated()"
        requires-channel="https" />
```

Přístup na všechny zdroje vyjma jiné uvedené v konfiguračním souboru musí být autentifikován. Zároveň se vyžaduje HTTPS protokol.

```
<intercept-url pattern="/task/*/detail" access="hasAuthority('detailTask')" />
```

HasAuthority kontroluje jednotlivá práva role uživatele a pravidlo se aplikuje na konkrétní vzor URL adresy.

```
<form-login login-page="/login"
        authentication-failure-url="/login"
        default-target-url="/" />
```

Definování stránky loginu a výchozí stránky pro přesměrování z login stránky.

```
<session-management invalid-session-url="/login/expired" />
```

Na tuto stránku se přesměruje uživatel při vypršení relace.

```
<authentication-manager alias="authenticationManager">
   <authentication-provider user-service-ref="userServiceImpl">
      <password-encoder hash="bcrypt" />
   </authentication-provider>
</authentication-manager>
```

Definování třídy, která se stará o správné vyhodnocení login autentifikačních požadavků. Zároveň jsem definoval hashovací algoritmus BCrypt [3] jako ten, který se bude používat pro hashování hesla.

4.2.2 Vrstva Pretty Faces

Scope: Application

Pretty Faces filtr dostává požadavek od Spring security filtru. Jeho úkolem je nejdříve zjistit, zda má uživatel s daným uživatelským jménem právo k zobrazení konkrétních dat. Pokud ne, je přesměrován na stránku /denied. Pokud má, tak dalším úkolem tohoto filtru je zavolat metody v Controllerech pro přednačtení kritických dat. Posledním úkolem je namapovat požadovanou url na reálnou xhtml stránku ve zdrojích. Požadavek na tuto stránku pošle dál Faces Servletu, který zobrazí View(pohled). Níže je uveden příklad pravidla z konfiguračního souboru pretty-config.xml.

```
<url-mapping id="detailTask">
   <pattern value="/task/#{requestedTaskId :
      taskController.requestedTaskId}/detail" />
```

```
   <view-id value="/resources/xhtml/task/detailTask.xhtml" />
   <action>#{taskController.loadTask}</action>
   <action>#{solutionController.loadSolutions}</action>
   <action>#{taskController.detailTaskAccessCheck()}</action>
</url-mapping>
```

Na tomto jednom pravidlu vysvětlím jednotlivé části vytváření pravidel. Url-mapping id pouze označuje unikátní označení pravidla, pattern value označuje vzor url adresy, které toto pravidlo bude překládat na reálný zdroj. RequestedTaskId je parametr, který označuje hodnotu na tomto místě v URL adrese, také se ukládá do proměnné requstedTaskId v Task Controlleru. Prvky action označují vykonání metody před předáním režie požadavku další části aplikace.

4.2.3 Vrstva View

Scope: View

View neboli pohled se stará o zobrazení dat. Jsou to obyčejné xhtml stránky, které navíc využívají injekci dat z a do objektů v Controllerech. Pokud potřebuje nějaká data načíst (GET požadavky), přeposílá požadavek o tyto data do Controllerů. Při POST požadavcích injektuje data vložená do formulářů do příslušných objektů v Controllerech, aby se s nimi mohlo dále pracovat v samotné logice aplikace.

Jednotlivé pohledy se skládají často z více souborů. Využívám zde tzv. šablon, kdy se snažím co nejvíce redukovat opakování kódu. Konkrétně existuje jedna hlavní šablona template.xhtml, která obsahuje většinu kódu na stránce kromě samotného obsahu uprostřed stránky, který se generuje na základě jednotlivých požadavků. Také se snažím používat formuláře na více místech, protože editace a vytváření entit obsahuje často stejné parametry ke zpracování.

Poslední věc,í co zmíním k pohledům, jsou prvky atributy disabled a rendered. Vzhledem k možné injekci dat z řídících jednotek je možné při renderování pohledu určit, zda se konkrétní obsah uživateli má zobrazit nebo zneaktivnit. Tato vlastnost se využívá k tomu, aby uživatel nemohl ať už omylem nebo úmyslně se snažit o činnost, kterou by podle práv neměl mít možnost konat.

4.2.4 Vrstva Controller

Scope: View (Jeden výjimečný Controller SessionController je SessionScope - drží data, která jsou potřebná mít v relaci nebo bez ní nefungují)

Řidící jednotka, která má za úkol nejdříve načíst kritická data, poté na požadavek pohledů načíst dodatečná data a při POST požadavcích zpracovává formulářová data a předává režii do vrstvy Service (služby). Slouží jako obslužná jednotka eventů (akcí), které se vytvářejí v pohledech. Tyto akce se vytvářejí při uživatelových kliknutích, pohybech myši, stisku kláves a dalších možných událostech ve webovém prohlížeči.

4.2.5 Vrstva Service

Scope: Application Servisní vrstva v sobě obsahuje samotnou logiku aplikace. Od řídících jednotek dostává formátované parametry, které potřebuje ke svému chodu. Zde je také transakční logika, kdy potřebujeme většinou několik akcí provádět v jedné transakci tak, aby se buď provedly všechny nebo žádná. Zformátované parametry servisní vrstva využívá k získání dat z vrstvy Repository (repozitář) a nebo vložení do této vrstvy. Po vykonání této logiky (pokud je to potřeba) vrací zformátovaná data zpět do řidích jednotek, kde se ukládají do proměnných a vrací se do pohledů k zobrazení pro uživatele.

4.2.6 Vrstva Repository

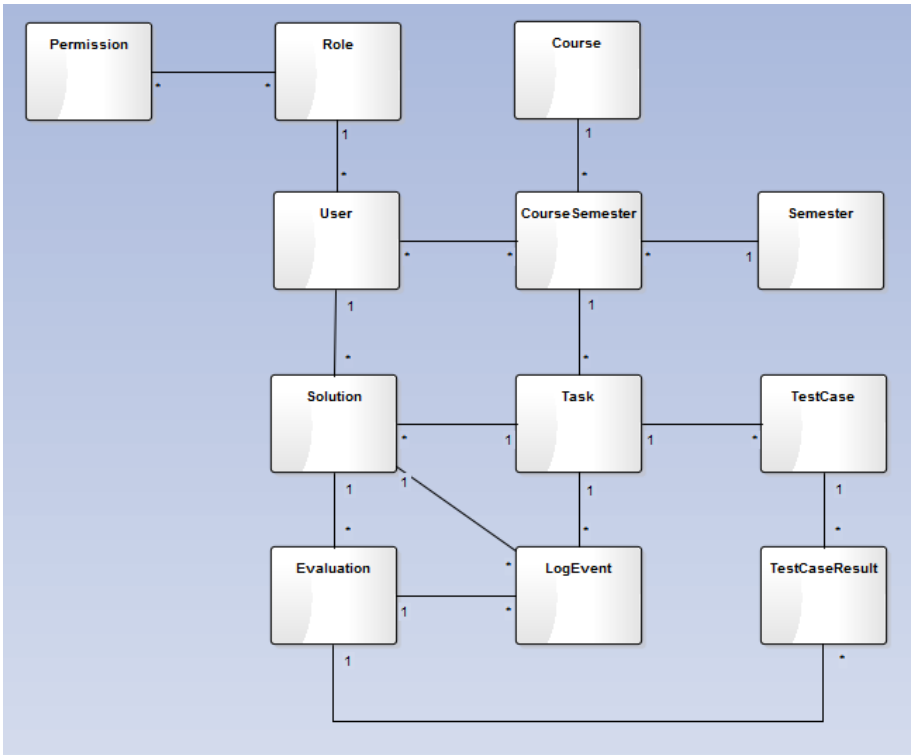
Scope: Application V této vrstvě dochází k napojení se do samotného celého modelu aplikace. Tato vrstva představuje logiku aplikace, která má co do činění s databází. Získává data podle upravených zformátovaných parametrů z databáze a tato data zpět předává ke zpracování do servisní vrstvy. Nebo ukládá nové či pozměněné entity do databáze. Zde je potřeba zmínit, že některé akce vyvolávají změny i mimo databázi. Pokud je potřeba takovouto změnu udělat, zaregistruje se TransactionalCallback (transakční zpětné volání) na danou transakci (technicky se toto provádí v servisní vrstvě, ale vzhledem k tomu, že se to týká modelu a tedy repozitáře, proto to uvádím) a provede se jedině v případě úspěšného vykonání transakce. Hlavně jde o změny v souborové struktuře, kde jsou názvy souborů často závislé na názvech v jednotlivých entitách.

Diagram tříd 4.2 popisuje kardinality jednotlivých modelových tříd.

4.3 CRUD entit, Copier

Pro práci s entitami se aplikuje následující model. Pro uživatele se vždy zobrazují formátovaná zástupná data (DTO objekty), se kterými on může manipulovat aniž by hrozila nechtěná akce na reálných persistentních entitách v databázi. Po zpracování těchto formátovaných dat (tedy přesun z Controlleru do Service vrstvy) se tato data převádí přes univerzální třídu DTOCreator do entit (ať už je to vytváření nebo aktualizace dat). Každá vrstva má svůj vlastní mechanismus jak zpracovat formátovaná data DTO objektů do entit (metody prepareTask, prepareSolution a další). Po připravení entit se tyto entity předávají do vrstvy Repository, kde dochází k metodám jako jsou persist (vytvoření objektu), merge (aktualizace objektu), remove (odstranění objektu) a vybraným dalším metodám jako jsou refresh (obnovení aktuálních dat z kontextu). Vrstva Repository se tak stará o perzistentní uložení připravených dat.

Při práci třídy DTOCreator je potřeba zmínit třídu Copier. Ta slouží pro ulehčené kopírování dat ze dvou tříd, které spolu můžou a nemusí mít vztah přes nějaké rozhraní či společného předka. Metody copyCommonFields a copyCommonFieldsWithExceptions, jak už název napovídá, kopírují jednotlivá pole výchozího objektu do finálního objektu. Pro zjištění, jaká pole má kopírovat, se používá definice tříd daných objektů. Přes reflexi se tak ve smyčce prochází pole výchozího objektu a pokud najde shodu ve finálním objektu (tedy



Obrázek 4.2: Diagram tříd modelových tříd projektu

shodu jak ve jméně, tak v typu pole), tak hodnotu překopíruje. Metoda copyCommondFieldsWithExceptions se používá pro vyjmutí vybraných polí z kopírovacího procesu (například pokud chceme s nějakými poli udělat složitější proces než jen zkopírování hodnoty).

4.4 Zpětná vazba - FacesMessage

Pro zpětnou vazbu směrem k akcím uživatele používám mechanismus FacesMessage, který je v implementaci JSF Primefaces. Tento mechanismus slouží k zobrazení krátké formátované zprávy s její úrovní závažnosti jako zpětná vazba po libovolné akci uživatele. Přidání této zprávy mám definováno v obecném předku Controllerů AController.

```
public void addMessage(Severity severity, String summary) {
    FacesMessage message = new FacesMessage(severity, summary, "");
    FacesContext.getCurrentInstance().addMessage(null, message);
}
```

Přidání této zprávy zde ale nestačí. Co je potřeba ještě udělat, je vytvořit kontejner pro tyto zprávy ve View vrstvě. V tomto kontejneru se jednotlivé zprávy budou objevovat.

```
<p:growl id="messages" showDetail="true" sticky="true"/>
```

Parametr sticky určuje, že zprávy nezmizí, pokud je uživatel nezavře nebo nepřejde na jinou stránku. Je zde z důvodu, že různí uživatelé mají různou dobu zpracování těchto zpráv a proto nejde moc dobře najít ideální časovou hodnotu, po které by zprávy zmizely. Navíc si někdy uživatel chce zprávu přečíst vícekrát. Poslední věc pro zobrazení zprávy je jednoznačná - musí se aktualizovat kontejner messages pro zobrazení nových zpráv. K tomu dochází buď při obnovení/přesměrování stránky ať už programově (například při dokončení odeslání formuláře, akce uživatele) a nebo ručně od uživatele.

```
<p:ajax event="change" update="departments :messages"/>
```

Zde je ukázka události změny na libovolném prvku, který tuto událost podporuje. Důležitý je parametr update, který označuje, jaké prvky v pohledu se mají aktualizovat po dokončení ajax požadavku.

Nakonec ještě přidávám příslušnou změnu životního cyklu zpráv v konfiguračním souboru faces-config.xml. Tato konfigurace je zde z důvodu faktu, že při určitých požadavcích uživatele se zpráva nezobrazila. Takto se zpráva vždy objeví právě jednou při aktualizování prvku messages v šabloně template.xhtml.

```
<lifecycle>
   <phase-listener>
      com.ocpsoft.pretty.faces.event.MultiPageMessagesSupport
   </phase-listener>
</lifecycle>
```

4.5 Logování chyb - výstupy a konfigurace

Konfigurace v souboru log4j.properties se týká hlavně formátu zpráv, jak se budou logovat, a také kanálů, kterými se budou zprávy posílat. Logují se všechny zprávy od úrovně INFO k vyšším prioritám. Úrovně zpráv jsou INFO, WARN, ERROR, SEVERE. INFO představuje informační úroveň zprávy, WARN představuje varování, ERROR představuje chybu, ze které se systém vzpamatuje a SEVERE chybu, která je příliš zavážná a systém se nevzpamatuje.

```
#log to file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=/home/tomcat7/log/app.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
    %c{1}:%L - %m%n
log4j.appender.file.encoding=UTF-8
```

Zde se nastavuje formát zprávy, data kdy se daná událost stala a do jakého souboru se budou zprávy ukládat. Také se nastavuje kódování textu.

```
log4j.appender.gmail=cz.cvut.fel.exception.GmailSMTPAppender
log4j.appender.gmail.SMTPProtocol=smtps
log4j.appender.gmail.SMTPUsername=cvut.us@gmail.com
```

```
log4j.appender.gmail.SMTPPassword=uploadsystem
log4j.appender.gmail.SMTPHost=smtp.gmail.com
log4j.appender.gmail.SMTPPort=465
log4j.appender.gmail.Threshold=ERROR
log4j.appender.gmail.smtp.starttls.enable=true
log4j.appender.gmail.Subject=Logging Message via Gmail
log4j.appender.gmail.To=cvut.us@gmail.com
log4j.appender.gmail.From=cvut.us@gmail.com
log4j.appender.gmail.layout=org.apache.log4j.PatternLayout
log4j.appender.gmail.layout.ConversionPattern=%d{MM/dd/yyyy HH:mm:ss}[%M] %-5p %C
    - %m%n
log4j.appender.gmail.BufferSize=1
```

Druhým prvkem logování je posílání logů přes mail. V konfiguraci se uvádí přihlašovací data uživatele, způsob se k danému uživatelskému email účtu připojit a komu emaily posílat. Také určuje formát zprávy v emailu. Parametr BufferSize určuje, po kolika zprávách se budou maily posílat. Důležité je zmínit, že pro posílání na Gmail používám GmailSMTPAppender, který rozšiřuje klasický SMTPAppender od log4j.

4.6 LXC - návrh implementace

Předtím, než popíšu řešení daného problému chci upozornit, že toto je pouze teoretické řešení. V praxi na testovacím serveru nebylo implementováno z důvodu starší distribuce operačního systému Debian (Wheezy). Pro toto řešení je potřeba nejnovější distribuce Debian (Jessie).

Myšlenka je taková, že se pro každý programovací jazyk vytvoří LXC kontejner, který bude obsahovat všechny prvky potřebné pro spuštění a evaluaci konkrétního uživatelského řešení. Tyto kontejnery budou mít omezené pravomoci - omezí se jim přístup k souborům vyjma na speciální dočasnou složku, do které se nakopírují všechny potřebné soubory k evaluaci a skriptovou složku, která obsahuje vyhodnocovací bash skripty. Omezí se jim také přístup k síti, aby nemohli uživatelé posílat řešení, která se napojí na vzdálený server a budou posílat například vstupní data, která obdrží od vyhodnocovacího procesu.

Poté, co budou tyto kontejnery vytvořeny se při vyhodnocovacím procesu vytvoří dočasná složka, nakopírují se do ní potřebná data k evaluaci (vstupní soubory, zdrojové kódy), uživatelské řešení se vyhodnotí do výstupních souborů v této složce a následně se tyto výstupní soubory přesunou do finální složky výstupních souborů v souborové struktuře projektu.

Co se týče samotného spuštění kontejneru, použije se příkaz lxc-start-ephemereal [16], který se právě nachází až v balíčcích pro nejnovější distribuci Debianu (Jessie). Tento příkaz naklonuje kontejner podle konkrétního šablonového kontejneru a je tak možné dosáhnout stejných podmínek pro všechna uživatelská řešení (stejného kontejneru) a není potřeba vytvářet tyto klony ručně.

Toto řešení jsem zkusil použít na své vlastní distribuci Debianu (Jessie) pouze se zástupnými objekty skriptů, dat, zdrojových kódů, vstupních souborů. Řešení ukazovalo dobré výsledky, fungovalo tak, jak se předpokládalo. Nicméně jak už jsem zmínil, v samotném projektu je vyhodnocování implementováno bez něj.

4.7 Registrace uživatele

Registrace uživatele obsahuje upravený formulář pro vytváření uživatele. V tomto formuláři není možné vybírat roli uživatele, výchozí role pro registrované uživatele je Student. Pro změnu této role později je potřeba kontaktovat administrátora aplikace, který ke změnám rolí jednotlivých uživatelů má přístup.

Pro zástupný DTO objekt UserDTO je role při registraci null. Tato role se doplňuje až v servisní vrstvě. Po odeslání formuláře se v UserControlleru předají zformátovaná data UserDTO objektu do servisní vrstvy, kde se ještě upravuje právě role, pokud je nulová, a heslo, které je potřeba zašifrovat algoritmem BCrypt. Po uložení nového objektu ve vrstvě Repository je uživatel uvědoměn o úspěšné registraci a může se přihlásit.

4.8 Login, Logout

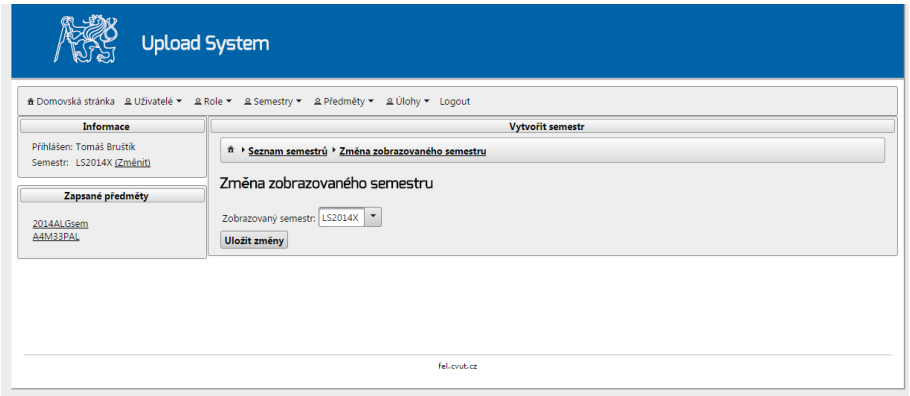
Přihlášení uživatele do systému probíhá přes přihlašovací formulář a odesílá se do UserControlleru. Zde se využívá výchozího mechanismu j_spring_security_check, který přes konfiguraci v security.xml kontroluje přes konkrétní definované metody, zda uživatel s danými iniciály existuje a má být tak autentifikován. V případě neúspěšného přihlášení je uživatel přesměrován zpět na login, v případě úspěšného přihlášení je uživatel přesměrován na stránku, na kterou se pokusil jít předtím. Pokud takováto stránka není (uživatel šel rovnou na přihlašovací stránku), je přesměrován na domácí stránku projektu.

Co se týče odhlášení, může nastat dvojím způsobem. Buď se uživatel sám od sebe odhlásí přes tlačítko Logout v menu, kdy zpracování požadavku na odhlášení je přesměrováno do UserControlleru metody logout. Zde se využívá výchozího mechanismu j_spring_security_logout a přes konfiguraci v security.xml je uživatel odhlášen a přesměrován na konkrétní odhlašovací stránku. Druhou možností odhlášení je vypršení časového limitu pro relaci. Pokud tento limit uplyne a uživatel se pokusí zobrazit chráněná data v aplikaci, je přesměrován na přihlašovací stránku se zprávou, že byl odhlášen kvůli dlouhé neaktivitě.

4.9 Pohled podle semestrů

Po přihlášení uživatele je důležité, že data ve velké části aplikace jsou vázána na konkrétní aktuální semestr. Všechny entity, které jsou vázány na entitu semestr přímo a nebo přes jinou entitu, se zobrazují pouze ve vztahu k aktuálnímu semestru. Konkrétním příkladem je například seznam předmětů nebo seznam úloh a entity vázané na tato data. Tento mechanismus je zaveden pro větší přehlednost, kdy uživatel velice často uvažuje o akcích v rámci semestru (zobrazování či spravování předmětů, výsledků, úloh, řešení).

Pokud se tedy uživatel chce podívat do jiného semestru, v levé části obrazovky má možnost Změnit aktuální semestr. Už při přihlášení tento semestr musí být zvolen pro správný chod aplikace. Pokud uživatel nemá u sebe uveden poslední zvolený aktuální semestr (pole currentSemester v entitě User) tak se zvolí výchozí aktuální semestr podle pole current v entitě Semester. Pro správný chod aplikace musí být vždy právě jeden aktuální výchozí semester v aplikaci. Při spravování semestrů tedy přepínač aktuálního semestru slouží k této podmínce.



Obrázek 4.3: Změna aktuálního semestru

Při změně semestru pro konkrétního uživatele tato změna přetrvává i pro další zacházení s aplikací. Tato změna se totiž propisuje do výše zmíněného pole currentSemester entity User.

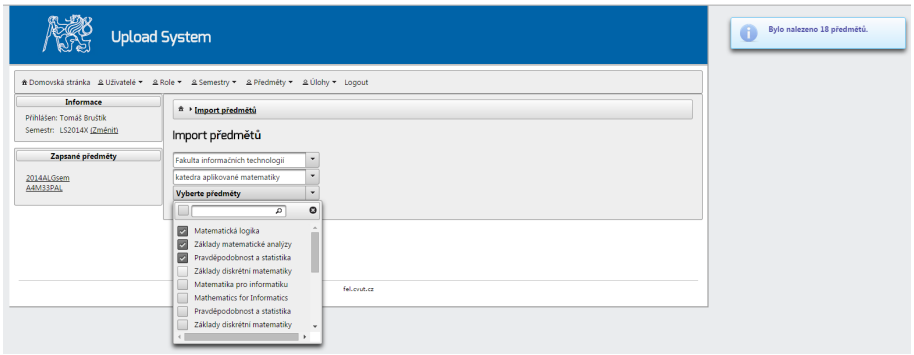
4.10 Vytváření a importování předmětů

Při zapisování předmětů do systému má uživatel několik možností. Pokud už byl předmět vytvořen dříve (a tedy například zahrnut v předchozích semestrech), je možné tento předmět zahrnout i v nově vytvářených semestrech (popřípadě v jejich editaci). V semestrovém formuláři se totiž nachází prvek selectManyCheckbox, který slouží právě k vytvoření nových entit CourseSemester, které budou vázané na již vytvořené staré entity Course.

```
<p:selectCheckboxMenu id="course" value="#{semesterController.selectedCourses}"
    label="Vyberte předměty, které půjdou zapsat"
                          layout="grid" filter="true"
                              filterMatchMode="startsWith"
                              panelStyle="width:250px">
    <f:selectItems value="#{courseServiceImpl.courses}" var="course"
        itemValue="#{course.courseName}" itemLabel="#{course.courseName}"/>
</p:selectCheckboxMenu>
```

Tento ovládací prvek má několik zajímavých schopností z nichž je hodně nápomocná možnost filtrování záznamů a výběr všech zobrazených záznamů přes jedno kliknutí.

Pokud předmět ještě nikdy nebyl vytvořen, je potřeba ho buď vytvořit, nebo importovat. Při vytváření předmětů se vytváří jak entita CourseSemester, tak entita Course. Zároveň se do daného předmětu zapíší vybraní učitelé, které uživatel ve formuláři zaškrtl. Zároveň se automaticky zapíše do předmětu jeho zakladatel. Je to z důvodu takového případu, kdy by uživatel pozapomněl na zaškrtnutí učitelů a po vytvoření předmětu by tak do jeho správy neměl nikdo přístup - do správy předmětu mají přístup pouze uživatelé, kteří jsou v daném předmětu zapsaní a mají příslušná práva ve své roli.



Obrázek 4.4: Importování předmětů do systému

Další možností na vytvoření předmětu je importování dat o předmětech z KOSapi. Je tedy možné importovat například vybrané předměty nějaké katedry v aktuálním semestru. Tento proces probíhá v několika fázích, kdy uživatel na příslušné stránce aplikace vybírá nejdříve fakultu, potom katedru z této fakulty a potom vybrané předměty z dané katedry, jejichž data chce importovat do systému.

Proces uložení importovaných předmětů probíhá stejně jako u vytvářených předmětů. Zde ještě konkrétněji popíšu implementaci vysílání požadavků o daná data z KOSApi. Implementace je ve službě RestServiceImpl.

```
public void createAccessToken() {
   Client client = Client.create();
   AccessTokenRequestDataDTO data = new AccessTokenRequestDataDTO(clientId,
       clientSecret, scope);
   WebResource webResource = client.resource(accessTokenUri);
   Gson gson = new Gson();

   MultivaluedMap formData = new MultivaluedMapImpl();
   formData.add("grant_type", data.grantType);
   formData.add("client_id", data.clientId);
   formData.add("client_secret", data.clientSecret);
   formData.add("scope", data.scope);
   ClientResponse response =
       webResource.type(MediaType.APPLICATION_FORM_URLENCODED_TYPE)
       .post(ClientResponse.class, formData);
   String stringResponse = response.getEntity(String.class);

   accessToken = gson.fromJson(stringResponse, AccessTokenDTO.class);

   if (response.getStatus() != 200) {
      throw new RuntimeException(Došlo k neočekáváné chybě při pokusu o obdržení
          access tokenu : " + response.getStatus());
   }
}
```

Metoda createAccessToken se volá pro každé View nově. Slouží k obdržení access tokenu od autorizačního OAuth2 serveru. Tento access token se bude dále využívat při požadavcích o data z KOSApi. Knihovna GSon [7] se využívá pro překlad dat z JSON [12] formátu do POJO [19] objektů.

Po obdržení access tokenu se v jednotlivých fázích stahují data z KOSApi, nejdříve pro jednotlivé fakulty, poté pro jednotlivou katedru a poté pro předměty z této katedry. Pro příklad uvedu poslední metodu získání dat předmětů.

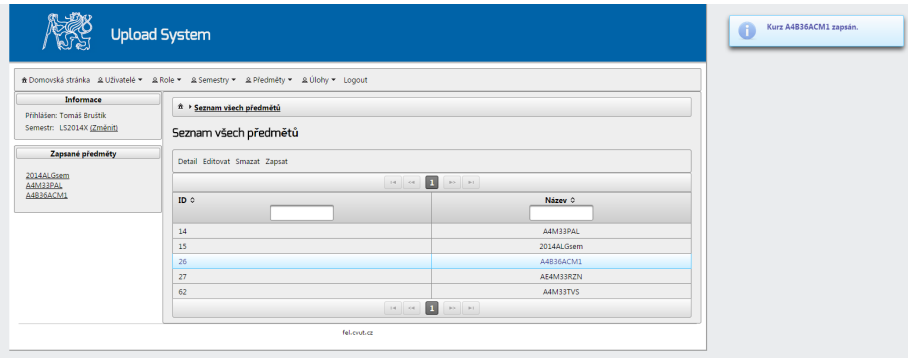
```
public CourseListResourceDTO getCourseListResource(String departmentCode) {
   Client client = Client.create();
   WebResource webResource = client.resource(coursesUri);
   MultivaluedMap<String, String> params = new MultivaluedMapImpl();
   params.add("query", "(department="+departmentCode+")");
   params.add("sem", "current");
   params.add("limit", "1000");
   ClientResponse response =
       webResource.queryParams(params).header("Content-Type",
       "application/json;charset=UTF-8")
           .header("Authorization", "Bearer " +
               accessToken.access_token).accept("application/atom+xml")
               .get(ClientResponse.class);
   JAXBContext jaxbContext;
   CourseListResourceDTO coursesData = null;

   try {
      jaxbContext = JAXBContext.newInstance(CourseListResourceDTO.class);
      Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
      coursesData = (CourseListResourceDTO)
          jaxbUnmarshaller.unmarshal(response.getEntityInputStream());
   } catch (JAXBException e) {
      LOG.error(e);
   }
   return coursesData;
}
```

V této metodě se nejdříve připravuje požadavek, následně se posílá přes Jersey klientskou knihovnu a po obdržení odpovědi se využívá knihovny JAXBContext [10] pro překlad dat z příchozího vstupního proudu odpovědi do POJO objektů.

4.11 Zapsání a zrušení zápisu předmětu

V levé části obrazovky se také zobrazuje seznam zapsaných předmětů v daném semestru. Tento seznam slouží pro rychlý přístup a jednoznačné definování zapsaných předmětů pro daného uživatele. Pro zapsání předmětu je potřeba nejdříve mít vybraný daný konkrétní aktuální semestr 4.9. Poté uživatel naviguje přes hlavní nabídku na Seznam předmětů. Pro zapsání vybraného předmětu je potřeba vybrat ze seznamu konkrétní předmět a poté kliknout na tlačítko Zapsat v menu nad seznamem. Při úspěšném přihlášení předmětu (neúspěšně

může skončit například při opakovaném zapsání již zapsaného předmětu) je uživatel o stavu uvědoměn a v levém seznamu zapsaných předmětů se mu zobrazí nově zapsaný předmět.



Obrázek 4.5: Zápis předmětu z pohledu uživatele

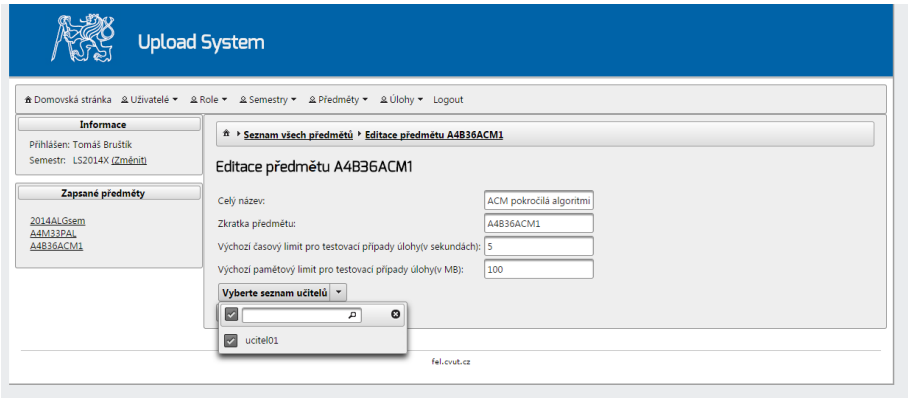
Následující kód popisuje servisní metodu pro zapsání předmětu.

```
public void registerCourse(Integer courseSemesterId, String username) throws
    CustomException {
   User user = userDao.findByUsername(username);
   CourseSemester courseSemester = courseSemesterDao.findById(courseSemesterId);
   if(user.getCourseSemesters().contains(courseSemester)) {
      throw new CustomException("Tento předmět už máte zapsán");
   }
   transactionalCallback.execute(new UserRegisterCourseCallback(user,
       courseSemester));
   user.addCourseSemester(courseSemester);
   courseSemester.addUser(user);
   userDao.merge(user);
}
```

Po registraci uživatele se provádí UserRegisterCourseCallback, který má za úkol v souborové struktuře vytvořit složky toho předmětu potřebné pro uživatele. Jedná se hlavně o složky již existujících úloh.

Zde je důležité podotknout, že je zde hlavní rozdíl mezi entitami Course a CourseSemestr. Uživatel prakticky pracuje s entitami CourseSemester v celé aplikaci, nicméně entita Course je zde pro vybrání společných dat předmětů skrze semestry. Je tedy možné při tvoření nového semestru zahrnout předměty z minulých semestrů. Tyto nově vytvořené entity CourseSemester budou vázané na stejnou entitu Course jako ty předchozí, nicméně ostatní entity jako Task se budou vázat na tyto nově vytvořené entity CourseSemester.

Co se týče zapisování předmětu, tuto pravomoc má pouze Student. Učitel tuto pravomoc nemá, protože při takové možnosti by mu stačilo se zapsat na cizí předmět a hned by měl pravomoci například na upravování toho předmětu, jeho úloh, zobrazování hodnocení úloh a další. Učitele je tedy potřeba přes editaci předmětu do předmětu přidávat. To může dělat libovolný učitel, který už je v daném předmětu registrovaný. Při vytváření předmětu se automaticky uživatel, jež ho vytváří, zapíše do tohoto předmětu, aby měl k němu plná práva.



Obrázek 4.6: Formulář editace předmětu

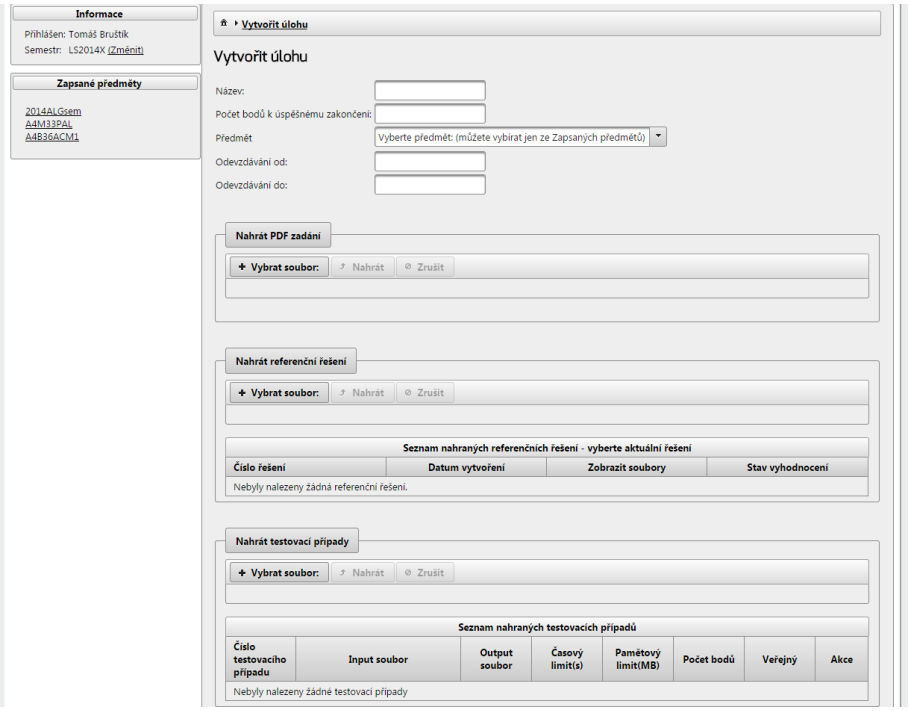
Poslední akci, kterou zmíním v této kapitole, je zrušení zapsání předmětu. Tento mechanismus existuje, dá se zavolat přes metodu CourseSemesterController.unregister() a také se volá při editaci předmětů, když se mění seznam učitelů. Nicméně přímo v aplikaci je zabráněno z pohledu libovolného uživatele zrušit zapsání předmětu pro případné zneužívání. Při zrušení zapsání předmětu by totiž v UserUnregisterCourseCallback došlo ke smazání dat daného uživatele ke konkrétnímu předmětu (tedy například všech jeho řešení na všechny úlohy). Ať už by k tomu došlo nechtěně nebo schválně, rozhodl jsem se, že nepovolím toto přímé zrušení zapsání předmětu. Následující kód popisuje servisní metodu pro zrušení zapsání předmětu.

```
public void unregisterCourse(Integer courseSemesterId, String username) throws
    CustomException {
   User user = userDao.findByUsername(username);
   CourseSemester courseSemester = courseSemesterDao.findById(courseSemesterId);
   if(!user.getCourseSemesters().contains(courseSemester)) {
      throw new CustomException("Tento předmět uživatel "+username+" nemá předmět
          "+courseSemester.getCourse().getCourseName()+" zapsán");
   }
   transactionalCallback.execute(new UserUnregisterCourseCallback(user,
       courseSemester));
   user.getCourseSemesters().remove(courseSemester);
   courseSemester.getUsers().remove(user);
   userDao.merge(user);
}
```

4.12 Vytváření úloh

K vytváření úloh se uživatel dostane přes menu Úlohy - Vytvořit. Samotný formulář se skládá ze čtyř částí. V první části je potřeba vyplnit určitá data úlohy, jako je její název, celkový počet bodů nebo k jakému předmětu patří. Předmět je možné vybrat pouze z těch, které daný uživatel spravuje (má je zapsané). Jako poslední v této části je potřeba zadat

odkdy dokdy bude možné odevzdávat řešení na danou úlohu. Mimo tento časový úsek nebude možné nahrát uživatelské řešení úlohy (v detailu úlohy bude systém hlásit, že odevzdání ještě nezačalo nebo že už termín vypršel).



Obrázek 4.7: Formulář vytváření úlohy

Druhá část se týká zadání úlohy. Vzhledem k tomu, že je prakticky nemožné umístit do webového rozhraní nástroj pro editaci odborného dokumentu, který by umožňoval všechny možné vychytávky, které by si mohl uživatel představit v editoru, jsem se rozhodl pro nahrávání řešení v PDF formátu. Tento formát je široce využívaný a po nahrání na server je možné dané zadání zobrazit (pokud to prohlížeč umožňuje) a nebo stáhnout.

```
<p:dialog id="syntax-pdf" header="Obsah zadání" widgetVar="showAssignmentPdf"
    dynamic="true" resizable="false" modal="true" height="800" width="1000">
                <p:media value="#{sessionController.streamPdf}" width="100%"
                    height="800px" player="pdf" />
        </p:dialog>
<p:commandButton value="Stáhnout zadání
    #{taskController.currentTask.assignmentFileName}" ajax="false"
                      rendered="#{not empty
                          taskController.currentTask.assignmentFileName}"
                          immediate="true">
   <p:fileDownload value="#{taskController.streamPdf}" />
</p:commandButton>
```

Využívají se zde komponenty p:media a p:fileDownload, které poskytují Primefaces na spravování různých typů medií a na stahování souborů.

V další části má uživatel možnost nahrát referenční řešení. Tato referenční řešení budou sloužit při vyhodnocení uživatelských řešení dané úlohu. Referenčních řešení je možné nahrát více, nicméně aktivní zůstane pouze jedno. Toto aktivní referenční řešení je zvýrazněno modrou barvou řádku a textem Aktivní za číslem řešení daného řešení. Je možné toto aktivní řešení změnit poklepáním na jiné. Zde chci podotknout, že nahrávání dat na server se provádí pouze do dočasné složky, permanentně se data propisují až po uložení formuláře.

Poslední část formuláře se týká testovacích případů. Zde se nahrávají testovací vstupy, které se budou postupně při evaluaci předkládat na vstup uživatelského řešení. Také se zde u každého testovacího případu definuje název výstupního souboru, časový limit a pamětový limit, který má řešení na správné vyhodnocení vstupu a vrácení výsledku. Také se zde nastavuje počet bodů za daný testovací případ (některé vstupy mohou být těžší na zpracování než jiné), zda je testovací případ veřejný (veřejné testovací případy si v detailu úlohy může student stáhnout a také při zobrazení detailu řešení vidí výstupy daného testovacího případu jak u svého tak u referenčního řešení). Zde je možné i zobrazit obsah vstupního souboru pro kontrolu - pomocí tlačítka s lupou, které zobrazí okno s obsahem souboru. Více o tom v 4.18.

4.13 Nahrávání uživatelského řešení, vyhodnocovací skripty

V detailu úlohy je možné nahrát uživatelské řešení, pokud k tomu má uživatel právo a datum a čas při zobrazení stránky je v intervalu definovaném ve správě úlohy. Po nahrání tohoto řešení se v servisní vrstvě vytvoří entita Solution s příslušnou entitou Evaluation a po dokončení transakce na propsání těchto entit se zavolá SolutionEvaluationCallback. Toto zpětné volání má za následek umístění dané evaluace do fronty k vyhodnocení.

O celé vyhodnocování se starají třídy TaskThreadPool a TaskWorkerThread. TaskThreadPool má nastavené čtyři vlákna a frontu o velikosti sto objektů řešení, která se snaží postupně zpracovávat. Počet vláken je nastaven vůči dvoujádrovému procesoru testovacího serveru.

```
public void submitSolution(int solutionId, String username) {
   executorPool.execute(new TaskWorkerThread(solutionId, username, solutionDao,
       transactionalWrapper));
}
```

Ve třídě TaskWorkerThread se při vykonání tohoto vlákna nejdříve vytáhne příslušné řešení z databáze a poté se na něm postupně provádí několik operací. Nejdříve je potřeba řešení rozzipovat, k tomu slouží zavolání bashového skriptu unzip.sh.

```
#!/bin/bash
ZIP_FILE=${1}
DESTINATION=${2}
_term() {
   printf "%s\n" "Caught SIGTERM signal!"
   kill -TERM $child 2>/dev/null
}
trap _term 15
unzip $ZIP_FILE -d $DESTINATION &
child=$!
wait $child
```

Poté se prochází složka zdrojových kódu a aplikace se snaží zjistit v jakém programovacím jazyku je řešení napsáno. Toto zjišťuje podle koncovek souborů, musí najít pouze koncovky definované u jednoho programovacímu jazyku, jinak vyhazuje chybu o nalezení více programovacích jazyků. V případě nenalezení žádného podporovaného programovacího jazyku také vyhazuje příslušné chybové hlášení.

```
public static ProgrammingLanguageEnum parseLanguage(String destination) throws
    CustomException {
   File destinationDir = new File(destination);
   Map<ProgrammingLanguageEnum, Boolean> languages = new
       HashMap<ProgrammingLanguageEnum, Boolean>();
   languages.put(ProgrammingLanguageEnum.JAVA, false);
   languages.put(ProgrammingLanguageEnum.C, false);
   languages.put(ProgrammingLanguageEnum.CPP, false);

   for(ProgrammingLanguageEnum language : languages.keySet()) {
         String[] extensions = {language.description};
         List<File> files = (List<File>) FileUtils.listFiles(destinationDir,
             extensions, true);
         if(files != null && files.size() > 0) {
                 languages.put(language, true);
         }
   }

   ProgrammingLanguageEnum resultLanguage = null;
   for(ProgrammingLanguageEnum language : languages.keySet()) {
         if(languages.get(language)) {
                 if(resultLanguage != null) {
                       throw new CustomException("Detekováno více programovacích
                           jazyků v jednom řešení");
                 } else {
                       resultLanguage = language;
                 }
         }
   }

   if(resultLanguage == null) {
         throw new CustomException("Nedetekován žádný podporovaný programovací
             jazyk");
   }
   return resultLanguage;
}
```

Poté je potřeba zjistit balíček/složku v níž se nachází všechny zdrojové kódy. Uživatelům je dána malá volnost při vybírání struktury řešení - můžou mít libovolný počet tříd, v libovolně nazvaném balíčku, ale musí být všechny v jednom balíčku a zároveň třída, ve které se nachází metoda main, se musí jmenovat Main (Main.java, Main.c, Main.cpp). Pokud jsou dodrženy všechny podmínky, validace je úspěšná a daný balíček/složka se zaznamená a dále používá ke kompilování a vyhodnocování.

```
public static String findSourcePackage(String destination,
    ProgrammingLanguageEnum language) throws CustomException {
   File destinationDir = new File(destination);
   String[] extensions = {language.description};
   List<File> files = (List<File>) FileUtils.listFiles(destinationDir, extensions,
       true);
   String pkg = null;
   for(File file : files) {
         if(pkg != null && !pkg.equals(file.getParentFile().getName())) {
                 throw new CustomException("Všechny třídy se nenacházejí v jednom
                     balíčku/složce");
         }
         pkg = file.getParentFile().getPath();
         if((pkg+"/").equals(destination)) {
                 pkg = "";
         }
   }
   if(pkg == null) {
         throw new CustomException("Nebyl nalezen žádný balíček/složka obsahující
             soubory s příponou ."+language.description);
   }
   pkg = pkg.replace(destination, "");
   return pkg;
}
```

Následně se program pokouší o kompilaci řešení. To se děje nejdříve vytvořením parametrů a následným předáním těchto parametrů do kompilačního skriptu příslušného jazyku. Na ukázku zde uvádím kompilační skript pro jazyk C++.

```
#!/bin/bash
ZIP_FILE=${1}
DESTINATION=${2}
_term() {
   printf "%s\n" "Caught SIGTERM signal!"
   kill -TERM $child 2>/dev/null
}
trap _term 15
unzip $ZIP_FILE -d $DESTINATION &
child=$!
wait $child
tomcat7@aetest1:~/script$ cat cpp_compile.sh
#!/bin/bash
SOLUTION_DIR=${1}
_term() {
   printf "%s\n" "Caught SIGTERM signal!"
   kill -TERM $child 2>/dev/null
}
trap _term 15
cd $SOLUTION_DIR;
g++ -std=c++11 *.cpp -o Main &
child=$!
```

```
wait $child
```

Z kompilačního skriptu je jednoznačně vidět, které parametry se předávájí kompilátoru.

Po úspěšné kompilaci dochází k samotnému spouštění uživatelského programu a kontrolování výsledků. Toto se děje ve smyčce pro všechny definované testovací případy a zaznamenává se také čas trvání tohoto vyhodnocování pro statistické účely. Pro ukázku znovu vyhodnocovací skript pro jazyk C++.

```
#!/bin/bash
TASK_DIR=${1}
SOLUTION_SOURCE_DIR=${2}
EVALUATION_OUTPUT_DIR=${3}
SOURCE_PACKAGE=${4}
FILE_INPUT=${5}
FILE_OUTPUT=${6}
MEMORY_LIMIT=${7}
TIME_LIMIT=${8}
_term() {
   printf "%s\n" "Caught SIGTERM signal!"
   kill -TERM $child 2>/dev/null
}
trap _term 15
cd ${SOLUTION_SOURCE_DIR}${SOURCE_PACKAGE};
ulimit -t $TIME_LIMIT;
ulimit -v $MEMORY_LIMIT;
ulimit -m $MEMORY_LIMIT;
ulimit -s $MEMORY_LIMIT;
./Main < $TASK_DIR$FILE_INPUT > $EVALUATION_OUTPUT_DIR$FILE_OUTPUT &
child=$!
wait $child
```

Metoda pro evaluování jednoho testovacího případu.

```
public TestCaseResult evaluate(TestCase testCase, Evaluation evaluation,
    Evaluation referenceEvaluation) throws IOException {
   TestCaseResult testCaseResult = new TestCaseResult();
   String outputFileName = testCase.getOutputName();
   Integer evaluationOrder = evaluation.getOrder();
   Integer solutionOrder = evaluation.getSolution().getOrder();
   String username = evaluation.getSolution().getUser().getUsername();
   String courseSemesterName =
       evaluation.getSolution().getTask().getCourseSemester().getCourseSemesterName();
   String taskName = evaluation.getSolution().getTask().getTaskName();
   Boolean isReference = evaluation.getSolution().isReference();
   Integer referenceEvaluationOrder = referenceEvaluation.getOrder();
   Integer referenceSolutionOrder = referenceEvaluation.getSolution().getOrder();
   String referenceUsername = evaluation.getSolution().getUser().getUsername();
   Boolean referenceIsReference = referenceEvaluation.getSolution().isReference();
   File userResult = new
       File(PathBuilder.getEvaluationOutputFilePath(outputFileName,
       evaluationOrder, solutionOrder, username, courseSemesterName, taskName,
```

```
        isReference));
    File referenceResult = new
        File(PathBuilder.getEvaluationOutputFilePath(outputFileName,
        referenceEvaluationOrder, referenceSolutionOrder, referenceUsername,
        courseSemesterName, taskName, referenceIsReference));
    testCaseResult.setSuccessful(FileUtils.contentEquals(userResult,
        referenceResult));
    if(testCaseResult.getSuccessful()) {
          testCaseResult.setResult(EvaluationResultEnum.SUCCESSFUL);
    } else {
          testCaseResult.setResult(EvaluationResultEnum.UNSUCCESSFUL);
    }
    return testCaseResult;
}
```

V případě úspěšného vyhodnocení testovacího případu se přidávají body k celkové evaluaci.

```
if(testCaseResult.getSuccessful()) {
   solution.getEvaluation().addPoints(testCase.getPoints());
}
```

Poté se všechny nově vytvořené entity uloží, vlákno se vrací do TaskThreadPoolu, kde čeká na nový požadavek o vyhodnocení úlohy. Je nutno podotknout, že tímto procesem vyhodnocení prochází jak uživatelská tak referenční řešení. Referenční řešení se pouze vyhodnocují vůči jim samotným, protože jsou referenční.

Poslední věc, kterou zde zmíním, je že celý tento kód je potřeba provádět transakčně. Protože TaskWorkerThread není komponentou Springu a proto není možné využívat jeho implicitních způsobů pro práci s transakcemi, využívám dvou tříd TransactionalWrapper a TransactionalWrapperProxy. Tyto třídy nejsou mým dílem a se souhlasem autora je v tomto projektu využívám.

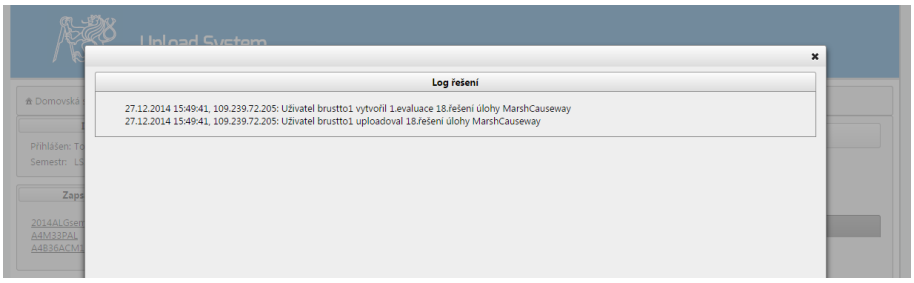
4.14 Přidání nového programovacího jazyku

Pro přidání nového programovacího jazyka je potřeba provést několik úkonů. Nejdříve je potřeba do množiny jazyků ProgrammingLanguageEnum přidat nový programovací jazyk s popisem.Tento popis označuje názvy bashových skriptů, které je potřeba přidat pro správné fungování vyhodnocení. Přidají se tedy dva skripty, popis_compile.sh a popis.sh. Parametry, které jdou do jednotlivých skriptů, je možné vidět z předchozích ukázek. Ve skriptech je možné spouštět cokoli, co by se dalo spustit v konzoli. Správné fungování obou skriptů tedy musí zaručit programátor, který je bude přidávat.

4.15 Logování akcí uživatelů

Všechny akce uživatele, které se týkají úloh, řešení a vyhodnocení se logují. Je tomu tak z důvodu zachování informace kdy, odkud a za jakých podmínek uživatel co udělal, aby

se dalo kontrolovat zneužití systému a předejít dohadům o skutečně provedených akcích uživatelů. Informace, které se logují u jednotlivých akcí, jsou následující. Loguje se typ akce (vytváření, editace a další), IP adresa, ze které uživatel tuto akci vykonal, který uživatel tuto akci vykonal, na jaké úloze, řešení či evaluaci a čas této akce. Je tedy zpětně možné dohledat například, kdy daný uživatel nahrál konkrétní řešení, co bylo jeho obsahem 4.18 a kontrolovat, odkud dané řešení nahrál. To může sloužit třeba jako kontrola při zkouškách, kdy je jednoznačně daná sada IP adres, odkud se mohou řešení nahrávat.



Obrázek 4.8: Log akcí uživatele týkajících se řešení

Pro zobrazení logu v tuto chvíli slouží tlačítko Zobrazit Log na detailu řešení. Po stisknutí tohoto tlačítka uvidí uživatel seznam logovaných událostí seřazených podle data, kdy se udály. Protože to byl požadavek vedoucího práce, zobrazují se události právě jen týkající se daného řešení, pro zobrazení všech událostí na dané úloze či všech událostí obecně by bylo potřeba vytvořit nový dotaz do databáze.

4.16 Zobrazení svých řešení, zobrazení všech řešení

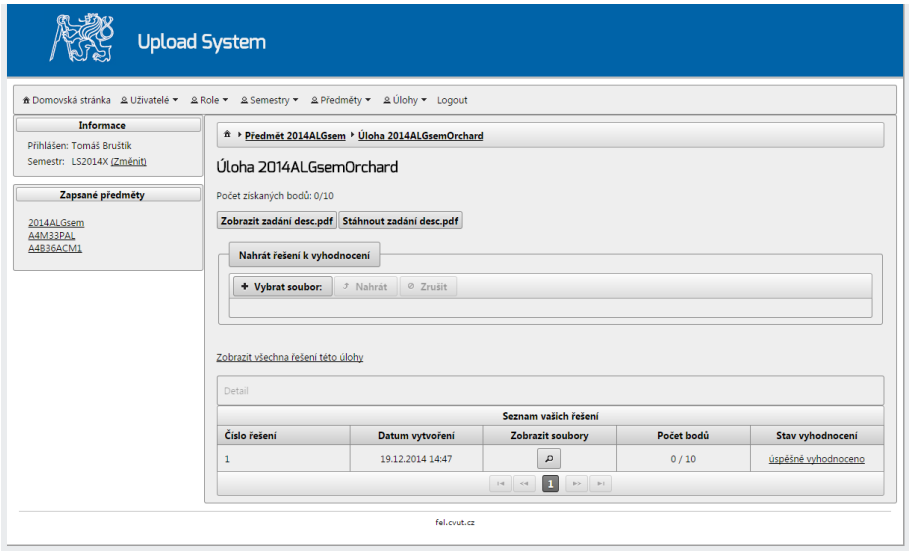
Pro zobrazení seznamu řešení existují dvě možnosti. Studenti mají možnost zobrazovat svoje řešení k dané úloze. Vše, co k tomu potřebují udělat, je podívat se do detailu dané úlohy. Zde se jim od nejnovějšího po nejstarší zobrazují všechna řešení, která k dané úloze zaslali k vyhodnocení. Přes označení řešení a poklepání na odkaz Detail se poté mohou dostat do detailu daného řešení, kde uvidí informace jak o souborech řešení, tak o výsledku jeho vyhodnocení.

Druhá možnost existuje pro učitele. Ti si mohou zobrazit seznam všech řešení všech studentů k dané úloze. Mají také právo dívat se na detail řešení úloh, které jsou zaregistrovány v předmětech, které spravují (mají je zapsané).

Obě tyto možnosti umožňují filtrování podle sloupců stejně jako ve všech zobrazovaných seznamech. Zde se to hodí opravdu výrazně, protože seznam řešení bývá u jednotlivých úloh opravdu dlouhý, a tak je možné rychle dohledat konkrétní řešení.

4.17 Zobrazení výsledků vyhodnocení

V detailu řešení je možné vidět seznam vyhodnocení pod sebou. V detailu konkrétního vyhodnocení je možné vidět seznam výsledků testovacích případů. U každého výsledku se



Obrázek 4.9: Detail úlohy a seznam vlastních řešení uživateleí

zobrazují informace o časovém a paměťovém limitu, čas trvání vyhodnocení u uživatelského i referenčního řešení a zda byl výsledek úspěšný či ne. Pokud na to má uživatel dostatečná práva, tak si může přes tlačítko Output zobrazit a případně stáhnout obsah výsledného souboru jak pro uživatelské, tak pro referenční řešení.

4.18 Zobrazení obsahu souborů

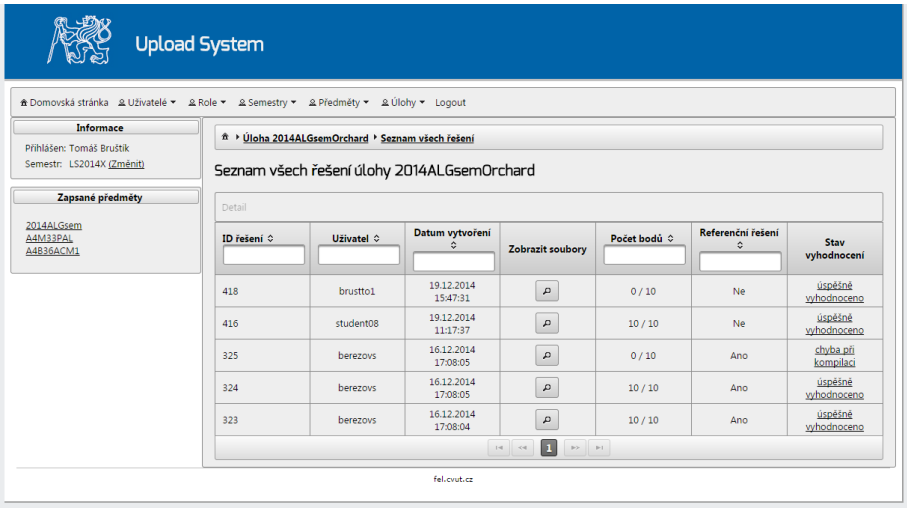
V celém projektu je možné zobrazit si obsah vybraných souborů. Tento obsah se zobrazuje vždy v otevřeném modálním okně. V průběhu této kapitoly jsem už zmínil zobrazování PDF zadání, zobrazování vstupních souborů a zobrazování výstupních souborů. Poslední možnost, kde si je možné zobrazit obsah souboru, je obsah souborů řešení. K této možnost se uživatel dostane na seznamu svých a nebo všech řešení přes tlačítko malé lupy.

V zobrazování obsahů souborů používám open source javascript knihovnu Syntax Highlighter [28]. Ta umožňuje zvýrazňování klíčových souborů a frází podle daných programovacích jazyků. Na obrázku 4.12 je možné vidět konkrétní zobrazení obsahu Main.java a zvýraznění jednotlivých částí tohoto souboru. Uživatel si tak může ověřit, že obsah řešení, které nahrál, obsahuje opravdu to, co má.

4.19 Hodnocení úloh předmětu

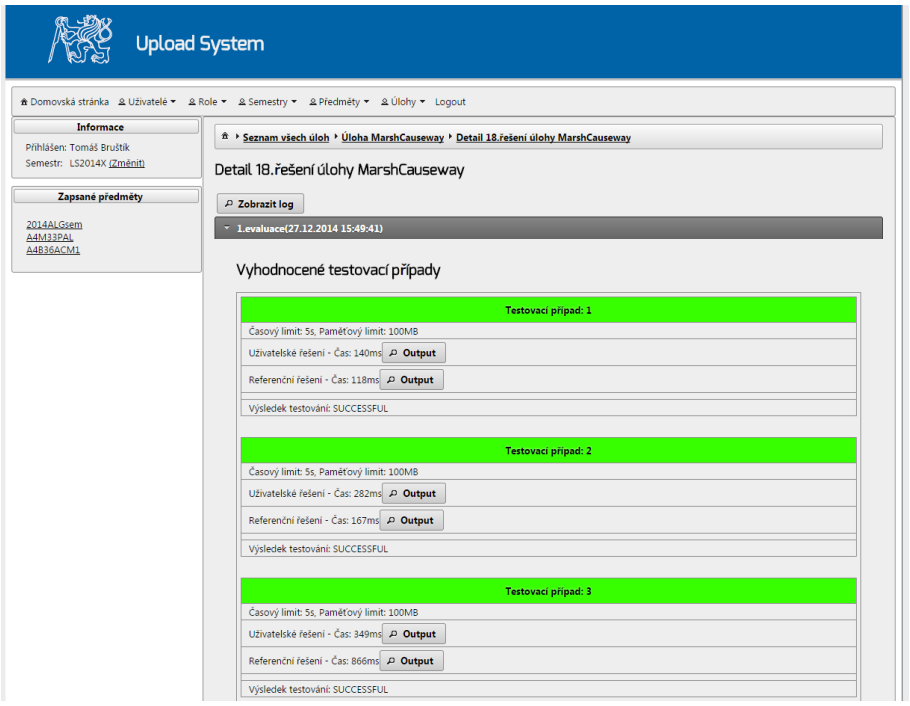
Pro kompletní a přehledné zobrazení výsledků jednotlivých studentů slouží zobrazení hodnocení úloh předmětu. K tomuto hodnocení se dostane uživatel přes detail zapsaného předmětu. Pokud na to má uživatel právo, tak si může zobrazit hodnocení úloh předmětu.

Na této stránce má uživatel možnost filtrovat si jednotlivé úlohy předmětu. Toto filtrování tak slouží k zjednodušení a zkrácení tabulky výsledků, to se může hodit při velkém množství

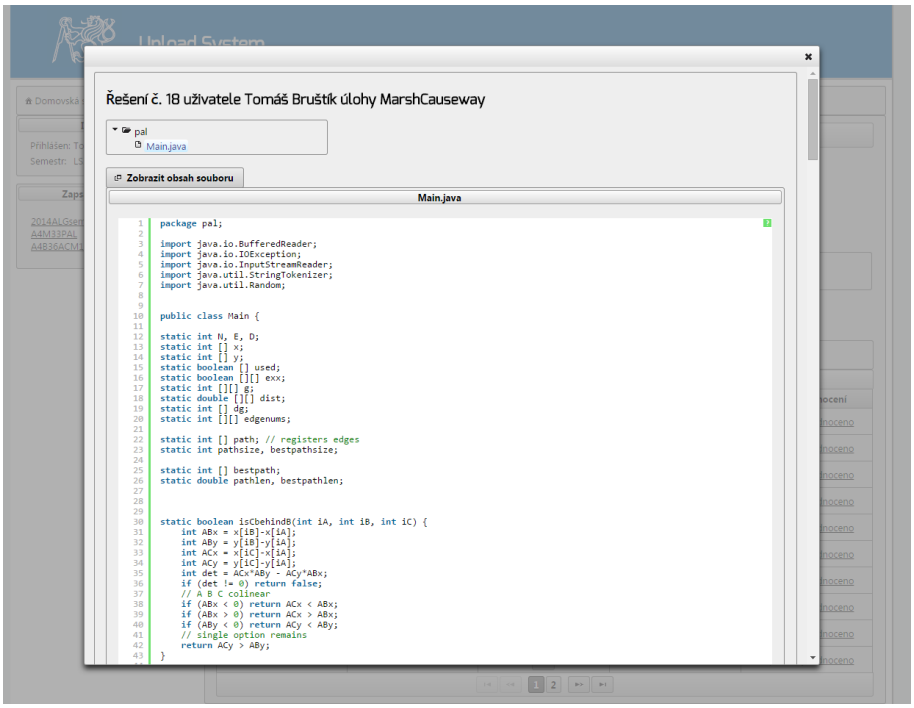


Obrázek 4.10: Seznam všech řešení konrétní úlohy

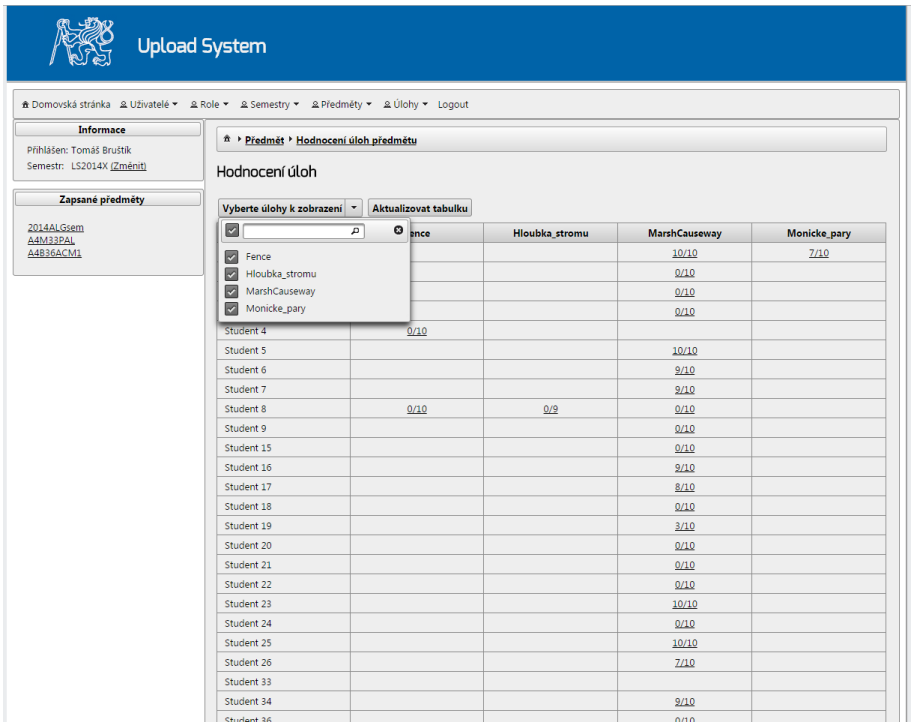
úloh v jednom předmětu. Jednotlivé záznamy obsahují výsledky všech zapsaných studentů v daném předmětu ke konkrétním úlohám. Navíc si uživatel může přes poklepání na konkrétní výsledek zobrazit konkrétní dané ohodnocené řešení.



Obrázek 4.11: Seznam výsledků vyhodnocení uživatelského řešení



Obrázek 4.12: Seznam výsledků vyhodnocení uživatelského řešení



Obrázek 4.13: Hodnocení úloh předmětu

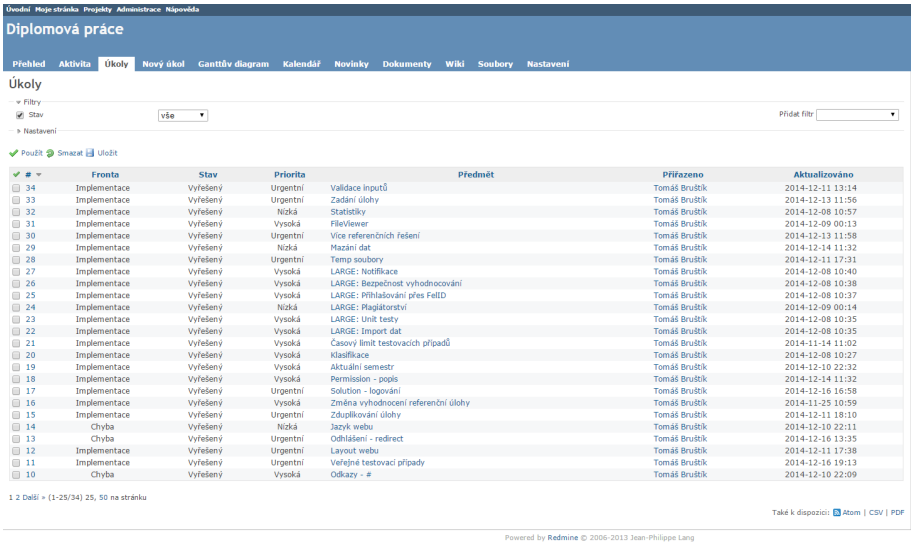
Kapitola 5

Testování

V kapitole testování se budu věnovat jednotlivým fázím testování, jakými projekt procházel. Tyto jednotlivé fáze testování probíhaly průběžně s vývojem aplikace po dobu 4 měsíců.

5.1 Uzavřené testování

Uzavřeného testování se účastnili tři uživatelé, z nichž dva byli v roli Studenta a jeden v roli Učitele. Cílem jejich snažení bylo hlavně odhalit chyby, nedostatky v aplikaci a také vytvářet nové požadavky na implementaci. Z tohoto testování vzniklo přes 30 reportů chyb či nových implementačních požadavků.



Obrázek 5.1: Seznam úkolů vytvořených z uzavřeného testování v programu Redmine

5.2 Otevřené testování

Otevřené testování probíhalo v intervalech během celého vývoje aplikace. Testovacích akcí proběhlo celkově 9, s tím že se testů zúčastnili studenti dvou paralelek předmětu Pokročilé algoritmy (cca 30-40 studentů na test). V průběhu těchto testů měli studenti vždy za úkol registrovat se do konkrétního předmětu a k vybraným testovacím úlohám (k nimž měli připravená řešení) se snažili tato řešení odevzdat a nechat vyhodnotit. V průběhu tohoto testu jsem monitoroval aktivitu a vytížení serveru a po daných testech jsem se snažil najít vždy chyby u nevyhodnocených řešení či u chybně vyhodnocených řešení.

Skrze toto testování jsem odhalil ideální množství počtu vláken ThreadPoolu pro vyhodnocování úloh. Toto ideální množství mi vyšlo na dvojnásobek počtu procesorů na testovacím serveru. Dále jsem se snažil odhalit chyby v nahrávání řešení s tím, že jsem stanovil určitá pravidla pro odevzdávání úloh pro studenty. Tato pravidla zajišťují správný proces kompilace k jednotlivým podporovaným jazykům a také doporučení pro studenty ohledně ukončování výstupů jejich programů. U referenčních úloh se totiž velice často nachází zakončení /r/n na konci výstupu programů, kdežto u studentů se toto nenacházelo. Tyto chyby nebylo jednoduché odhalit, protože při kontrolování výstupů řešení přes administrační rozhraní aplikace tento rozdíl nerozlišuje, dá se zjistit až při stáhnutí výstupů a ručním porovnání.

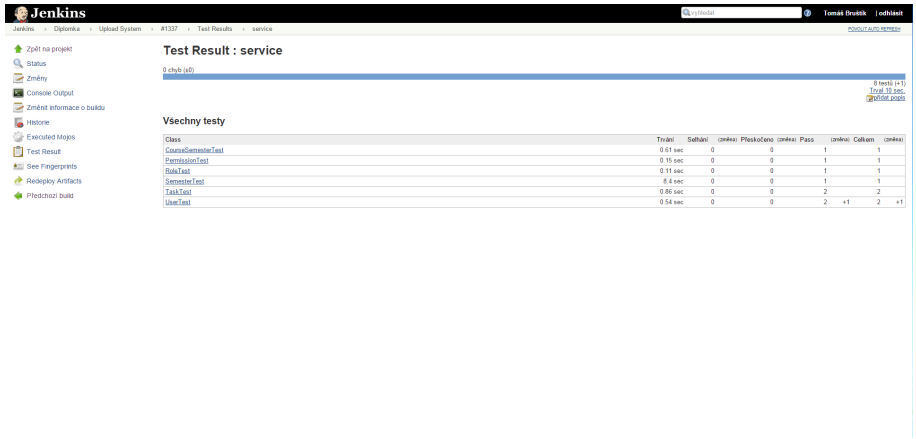
Během tohoto testování došlo k vytvoření více než 40 úloh, odevzdání přes 400 řešení ať už referenčních či uživatelských, více jak 600 vyhodnocení těchto řešení a vygenerování přes 4000 výstupních souborů těchto vyhodnocení. Všechna tato data sloužila k vylepšení vyhodnocovacího procesu. Studenti také vyplňovali na papír svoje připomínky k administračnímu rozhraní a zpětné vazbě systému. Díky těmto připomínkám byla vyvinuta například zpětná vazba systému 4.4 či přepracovaný vzhled systému ohledně seznamů dat,formulářů či rozvržení šablony základní stránky.

5.3 JUnit testy

JUnit testy se v aplikaci nachází pro kontrolování základních funkcí servisní vrstvy aplikace. Tato kontrola předchází každému nasazení aplikaci na server, při špatném vyhodnocení těchto testů se nová verze nenasadí, dokud testy neprojdou úspěšně. Tyto testy testují hlavně základní funkcionalitu servisní vrstvy při vytváření, editování a mazání entit spolu s několika vybranými uživatelskými předměty, jako je například registrování uživatele na vybraný předmět či testování manipulace s entitami při nutnosti unikátních záznamů. V krátkosti popíši jednotlivé části těchto testů.

5.3.1 TestHelper

Tato třída slouží jako pomocník při vytváření testovacích kódů. Obsahuje vybrané části testovacího kódu, které se opakují ve více testech. Protože každý test musí být sám o sobě soběstačný, je potřeba vytvářet všechny pomocné entity vždy znovu. V rámci této třídy se tak nachází pomocné metody pro vytváření testovacích entit User, CourseSemester, Semester, Role a Task. Zároveň pro vyčištění databáze od testovacích dat po doběhnutí testů obsahuje metody k mazání výše zmiňovaných testovacích entit.



Obrázek 5.2: Zobrazení výsledků JUnit testů v programu Jenkins

Celkově se těchto testů nachází v aplikaci 8, kde některé testy jsou postaveny na testování více manipulací s entitami. Většinou se testuje vytváření, editace a mazání testovací entity spolu s pomocnými entitami v jednom testu.

5.3.2 UserTest

Při testování metod UserService se testuje manipulace s entitou User. Testuje se vytváření, editace a mazání této testovací entity. Zároveň se také testuje vytváření entity uživatel s uživatelským jménem již existujícího uživatele. Zde se očekává chyba neunikátního výsledku běhu metody.

```
@Test
public void testUserActions() {
   testCreateUser();
   testEditUser();
   testDeleteUser();
}
@Test(expected=NonUniqueResultException.class)
public void testNonUnique() {
   try {
   testCreateUser();
   testNonUniqueUser();
   } catch (NonUniqueResultException ex) {
      testHelper.deleteDummyUser();
      throw ex;
   }
}
```

5.3.3 RoleTest

V testování entity role se testuje vytváření, editace a mazání testovací entity. Tato role má prázdný seznam práv.

```
@Test
public void testRoleActions() {
   testCreateRole(new ArrayList<String>());
   testEditRole(new ArrayList<String>());
   testDeleteRole();
}
```

5.3.4 PermissionTest

V této testovací třídě se testuje existence všech potřebných povolení ke správnému běhu aplikace. Seznam těchto pravidel a jejich funkcionality popisuji v uživatelské dokumentaci v příloze této práce. B.3

```
@Test
public void testPermissionsExists() {
   testPermissionExists("listUsers");
   testPermissionExists("createUser");
   testPermissionExists("editUser");
   ...
}
```

5.3.5 SemesterTest

V tomto testovacím souboru se testuje klasická manipulace s entitou - vytváření, editace a mazání. Zároveň se také testuje přítomnost alespoň jednoho semestru v databázi, vzhledem k nutnosti minimálně jednoho semestru pro správné fungování celé aplikace.

```
@Test
public void testSemesterActions() throws CustomException {
   UserDTO userDTO = testHelper.createDummyUser();
   userService.createUser(userDTO);
   testCreateSemester(new HashSet<String>(), userDTO.getUsername());
   testEditSemester(new HashSet<String>(), userDTO.getUsername());
   testDeleteSemester();
   testHelper.deleteDummyUser();
}
@Test
public void testSemesterCount() {
   List<SemesterDTO> semestersDTO = semesterService.getSemesters();
   Assert.assertTrue(semestersDTO.size() > 0);
}
```

5.3.6 CourseSemesterTest

Při testování manipulace s entitou CourseSemester se testuje vytváření, editace a mazání testovací entity. Také se testuje zapsání uživatele do testovacího předmětu.

```
@Test
public void testCourseSemesterActions() throws CustomException {
   UserDTO userDTO = testHelper.createDummyUser();
   userDTO.setUserId(userService.createUser(userDTO).getUserId());
   SemesterDTO semesterDTO = testHelper.createDummySemester();
   semesterDTO.setSemesterId(semesterService.createSemester(semesterDTO, new
       HashSet<String>(), userDTO.getUsername()).getSemesterId());
   CourseSemesterDTO courseSemesterDTO = testCreateCourseSemester(semesterDTO);
   testEditCourseSemester(courseSemesterDTO.getCourseSemesterId());
   testRegisterCourseSemester(courseSemesterDTO.getCourseSemesterId());
   testDeleteCourseSemester(courseSemesterDTO.getCourseSemesterId());
   testHelper.deleteDummyUser();
   testHelper.deleteDummySemester();
   testHelper.deleteDummyCourseEdit();
   }

public void testRegisterCourseSemester(Integer courseSemesterId) throws
    CustomException {
   UserDTO userDTO = testHelper.createDummyUser();
   userService.registerCourse(courseSemesterId, userDTO.getUsername());
   Assert.assertTrue(courseSemesterService.isUserRegisteredInCourseSemester
       (userDTO.getUsername(), courseSemesterId));
}
```

5.3.7 TaskTest

V posledním testu testuji manipulaci se základním vzorkem entity Task. Testuje se vytváření, editace a mazání této entity a také manipulace s touto entitou při vytváření úlohy, kdy úloha s tímto názvem už existuje.

```
@Test
public void testRoleActions() throws FileNotFoundException, NonUniqueException,
    CustomException {
   UserDTO userDTO = testHelper.createDummyUser();
   userDTO.setUserId(userService.createUser(userDTO).getUserId());
   SemesterDTO semesterDTO = testHelper.createDummySemester();
   semesterDTO.setSemesterId(semesterService.createSemester(semesterDTO, new
       HashSet<String>(), userDTO.getUsername()).getSemesterId());
   CourseSemesterDTO courseSemesterDTO =
       testHelper.createDummyCourseSemester(semesterDTO);
   CourseSemester courseSemester =
       courseSemesterService.createCourseSemester(courseSemesterDTO);
      courseSemesterDTO.setCourseSemesterId(courseSemester.getCourseSemesterId());
   TaskDTO taskDTO = testCreateTask(courseSemesterDTO);
   testEditTask(courseSemesterDTO, taskDTO.getTaskId());
```

```
   testDeleteTask(taskDTO.getTaskId());
   testHelper.deleteDummyUser();
   testHelper.deleteDummySemester();
   testHelper.deleteDummyCourse();
}

@Test(expected=NonUniqueResultException.class)
public void testNonUnique() throws FileNotFoundException, NonUniqueException,
    CustomException {
   UserDTO userDTO = testHelper.createDummyUser();
   userDTO.setUserId(userService.createUser(userDTO).getUserId());
   SemesterDTO semesterDTO = testHelper.createDummySemester();
   semesterDTO.setSemesterId(semesterService.createSemester(semesterDTO, new
       HashSet<String>(), userDTO.getUsername()).getSemesterId());
   CourseSemesterDTO courseSemesterDTO =
       testHelper.createDummyCourseSemester(semesterDTO);
   CourseSemester courseSemester =
       courseSemesterService.createCourseSemester(courseSemesterDTO);
      courseSemesterDTO.setCourseSemesterId(courseSemester.getCourseSemesterId());
   try {
      TaskDTO taskDTO = testCreateTask(courseSemesterDTO);
      testNonUniqueTask(courseSemesterDTO);
   } catch (NonUniqueResultException ex) {
      testHelper.deleteDummyUser();
      testHelper.deleteDummySemester();
      testHelper.deleteDummyCourse();
      throw ex;
   }
}
```

Kapitola 6

Závěr

6.1 Zhodnocení výsledků práce

V závěru se pokusím popsat splnění jednotlivých cílů v rámci celé diplomové práce.

Proveďte rešerši dostupných systémů pro spouštění a vyhodnocování programovacích úloh, zejména Uva Judge, ACM ICPC, Dom Judge a dalších systémů používaných v programovacích soutěžích. V kapitole 2 se věnuji rešerším výše zmíněných systémů a k tomu navíc systému Sharif Judge. Snažím se vždy obecně popsat to, co jednotlivé pojmy a systémy znamenají, a u těch, které jsou Open Source a jde tedy tak zjistit hlavní aspekty a požadavky systému, i více informací o daných aplikacích.

Podle výsledků rešerše navrhněte a realizujte vyhodnocovací systém pro výuku programování a algoritmizace na FEL ČVUT. Z výsledků rešerše hlavně open source projektů jsem si navrhl formu strukturovaného systému 2.3 pro nahrávání a vyhodnocování úloh. Tento systém také obsahuje velké možnosti v administračním rozhraní pro rozdělení zodpovědnosti a povinností mezi více uživatelů.

Systém musí umožnit práci registrovaných posluchačů jednotlivých předmětů FEL a také poskytnout funkční rozhraní pro neregistrované uživatele mimo FEL. Jak pro studenty ČVUT, tak pro obecné zájemce, je možné registrovat se do systému a zaujmout tak roli studentů (posluchačů). Mohou se zapsat do nabídnutých předmětů a zde odevzdávat řešení k připraveným úlohám. Pokud bude v budoucnu potřeba, je možné rozdělit roli Student na roli Student z ČVUT a obecný posluchač. V rámci této práci ale tato nutnost nevznikla a tak nebyla implementována. Nicméně přidání nové role a přiřazení jí povolených práv je v administračním rozhraní velice jednoduché a je to možné tak udělat v případě potřeby.

Systém dovolí učitelům předmětů vkládat a editovat úlohy, nastavovat podmínky vyhodnocování a klasifikace pro skupiny studentů nebo hostů, sledovat statistiky úspěšnosti studentů a jednotlivých úloh. Veškerou tuto funkčnost je možné vidět v kapitole 4 kde ji dopodrobna popisuji. Učitelé mají všechny tyto možnosti s tím, že hodnocení úloh studentů v daném předmětu jsou všechny v jedné tabulce hodnocení úloh předmětu, kde je možné vybírat konkrétní úlohy k zobrazení, pokud je to pro učitele přehlednější.

Součástí práce bude webové rozhraní systému a jeho uživatelská i programátorská dokumentace. Webové rozhraní je popsáno v práci, co se týče uživatelské dokumentace, tak ta se nachází v příloze této práce. Programátorská dokumentace je v samotném zdrojovém

kódu a z komentářů připojených k jednotlivým třídám a metodám je vygenerován dokument JavaDoc.

Zvažte a podle možnosti realizujte rozhraní systému pro spolupráci s informačním systémem KOS, případně dalšími servery FEL. Funkčnost a ovladatelnost systému ověřte zkušebním provozem ve výuce předmětu algoritmizace. Co se týče spolupráce se systémem KOS, děje se tak přes KOSApi 3.4.10.1. V této kapitole je popsána tato technologie a dále v implementaci je možné najít spolupráci s tímto rozhraním při importování předmětů z databáze KOS. Testování se studenty předmětu algoritmizace probíhalo průběžně 3 měsíce a jeho popis je možné naleznout v kapitole 5.1.

Následují navrhovaná vylepšení, které by mohla řešit budoucí požadavky na systém.

6.2 Návrhy na vylepšení

6.2.1 LoadBalancer a více strojů

Je možné, že v době velké zátěže nebude jeden stroj stíhat. Nebo by mohlo být vhodné právě v očekávaných dobách větší zátěže (třeba v době zkoušky) mít větší výkon systému. Pro tyto požadavky bych navrhoval řešení, kdy by jeden server sloužil jako Load Balancer [14] a rozděloval práci více serverům, které by se registrovaly a spustily v době potřeby. Vznikla by tak možnost většího výkonu celého systému, který by mohl pokrýt potřebné požadavky. Vznikly by tak ale problémy, týkající se držení informace relace, protože jednotlivé požadavky by mohly chodit na různé stroje a bylo by tak potřeba tento problém řešit.

6.2.2 FELid

KOSApi je výborný systém na importování dat ze systému KOS. Nicméně pro přihlašování studentů ČVUT nemusí být vhodný současný způsob (tedy nová registrace) ani importování jejich dat z KOSu. Nejideálnější by bylo asi propojit současný systém se systémem FELID, který umožňuje autentizaci uživatele přes jeho jméno a hlavní přístupové heslo do systému KOS. Tento účet by se pak spároval s účtem v systému a uživatel by se tak mohl přihlašovat přes své údaje z KOSu a dále pracovat v systému pod spárovaným účtem.

6.2.3 WebSocket

Tato technologie by umožnila obousměrnou komunikaci v rámci webového rozhraní mezi serverem a klientem. HTTP požadavky fungují pouze na bázi jednorázového požadavku a odpovědi. Nicméně přes technologii WebSocket by bylo možné posílat uživateli různé notifikace například o vyhodnocení jeho řešení, zprávy učitelů k daným úlohám či cokoli by bylo potřeba dělat asynchronně a nezávisle na akcích uživatele v systému.

6.3 Poslední slovo

Chtěl bych Vám poděkovat za přečtení této diplomové práce a doufám, že se Vám alespoň určité části líbily.

Literatura

[1] ACM ICPC.
http://icpc.baylor.edu/, stav ze 4. 1. 2015.

[2] AppArmor.
http://wiki.apparmor.net/index.php/Main_Page, stav ze 4. 1. 2015.

[3] BCrypt.
https://bcrypt.codeplex.com/, stav ze 4. 1. 2015.

[4] BP, Tomáš Bruštík, Realizce REST rozhraní pro databázový systém ExDB.
https://dip.felk.cvut.cz/browse/pdfcache/brustto1_2012bach.pdf, stav ze 4. 1. 2015.

[5] CourseWare upload system.
https://cw.felk.cvut.cz/ulohy/, stav ze 4. 1. 2015.

[6] DOM Judge.
http://www.domjudge.org/, stav ze 4. 1. 2015.

[7] Knihovna GSon.
https://code.google.com/p/google-gson/, stav ze 4. 1. 2015.

[8] Hibernate.
http://hibernate.org/, stav ze 4. 1. 2015.

[9] Java EE.
http://www.oracle.com/technetwork/java/javaee/overview/index.html, stav ze 4. 1. 2015.

[10] Knihovna JAXBContext.
http://docs.oracle.com/javase/7/docs/api/javax/xml/bind/JAXBContext.html, stav ze 4. 1. 2015.

[11] Jersey.
https://jersey.java.net/, stav ze 4. 1. 2015.

[12] JSON.
http://www.json.org/, stav ze 4. 1. 2015.

[13] KOSapi.
https://kosapi.fit.cvut.cz/projects/kosapi/wiki, stav ze 4. 1. 2015.

[14] Load balancing.
http://en.wikipedia.org/wiki/Load_balancing_%28computing%29, stav ze 4. 1. 2015.

[15] Log4j.
http://logging.apache.org/log4j/2.x/, stav ze 4. 1. 2015.

[16] LXC.
https://linuxcontainers.org/lxc/manpages/man1/lxc-start-ephemeral.1.html, stav ze 4. 1. 2015.

[17] MySQL.
http://www.mysql.com/, stav ze 4. 1. 2015.

[18] OAuth2.
http://oauth.net/2/, stav ze 4. 1. 2015.

[19] POJO.
http://en.wikipedia.org/wiki/Plain_Old_Java_Object, stav ze 4. 1. 2015.

[20] Pretty Faces.
http://ocpsoft.org/prettyfaces/, stav ze 4. 1. 2015.

[21] Primefaces.
http://primefaces.org/, stav ze 4. 1. 2015.

[22] Sandbox.
http://en.wikipedia.org/wiki/Sandbox, stav ze 4. 1. 2015.

[23] Sharif Judge.
http://docs.sharifjudge.ir/, stav ze 4. 1. 2015.

[24] Implementace GMailSMTPAppender.
https://code.google.com/p/log4j-gmail-smtp-appender, stav ze 4. 1. 2015.

[25] SPOJ.
http://www.spoj.com/, stav ze 4. 1. 2015.

[26] Spring Framework, .
http://projects.spring.io/spring-framework/, stav ze 4. 1. 2015.

[27] Spring Security, .
http://projects.spring.io/spring-security/, stav ze 4. 1. 2015.

[28] Syntax Highlighter.
http://alexgorbatchev.com/SyntaxHighlighter/, stav ze 4. 1. 2015.

[29] Transport Layer Security.
`http://http://en.wikipedia.org/wiki/Transport_Layer_Security`, stav ze 4. 1. 2015.

[30] UVa Judge.
`http://uva.onlinejudge.org/`, stav ze 4. 1. 2015.

[31] Spring View Scope for JSF users.
`http://blog.harezmi.com.tr/spring-view-scope-for-jsf-2-users/`, stav ze 4. 1. 2015.

Příloha A

Seznam použitých zkratek a pojmů

AJAX Asynchronous JavaScript and XML

AOP Aspect-oriented programming

API Application Programming Interface

CRUD Create, Read, Update, Delete

ČVUT České vysoké učení technické

EJB Enterprise Java Beans

GPL General Public License

HQL Hibernate Query Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

JDBC Java Database Connectivity

JPA Java Persistence API

JSF Java Server Faces

LAMP Linux, Apache, MySQL, PHP

LXC Linux Containers

MOSS Measure Of Software Similarity

MVC Model, View, Controller

ORM Object-relational mapping

PDF Portable Document Format

POJO Plain Old Java Object

REST Representational State Transfer

SQL Structured Query Language

SPOJ Sphere Online Judge

SSL Secure Sockets Layer

TLS Transport Layer Security

UI User Interface

URL Uniform Resource Locator

⋮

Příloha B

Instalační a uživatelská dokumentace

B.1 Testovací server

V této příloze uvádím informace o serveru na kterém byla aplikace vyvíjena a testována. Konfigurace jednotlivých souborů aplikace je dostupná na přiloženém CD této práce.

- Operační systém: Debian GNU/Linux verze 7.0 (Wheezy)
- Model procesoru: AMD Opteron(tm) Processor 6168
- Počet jader procesoru: 2
- Velikost harddisku: 20GB
- Velikost paměti: 2GB
- IP adresa: 147.32.84.188
- Uživatel 1 (Role Admin): uživatelské jméno - admin01, heslo - admin1
- Uživatel 2 (Role Učitel): uživatelské jméno - ucitel01, heslo - ucitel01
- Uživatel 3 (Role Student): uživatelské jméno - student01, heslo - student01

B.2 Instalační návod

Tento instalační návod předpokládá, že instalace bude probíhat na operačním systému s jádrem Linux, testováno bylo na operačním systému Ubuntu 14.04.1 LTS. Předpokládá instalaci balíčků gcc a g++ pro kompilaci programů napsaných v jazyce C a C++, stejně jako předpokládá instalaci Javy pro překlad programů napsaných v jazyce Java. Dále předpokládá nainstalovaný Tomcat7 a MySQL Server aktuální verze. Předpokládá také nakonfigurovaný Tomcat7 pro přesměrování SSL a tedy i platný certifikát. Pokud není potřeba mít zapnuté SSL, není potřeba tuto konfiguraci v Tomcat7 mít, nicméně je potřeba odebrat atribut requires-channel="https"ze souboru security.xml před vytvořením souboru ROOT.war. Konfigurace aplikačních souborů je stejná jako je na přiloženém CD, při změně libovolných

dat (domácí složky, přihlašovacích dat do databáze či dalších) je potřeba změnit příslušné konfigurační soubory.

1. Vytvořte složku /home/tomcat7/.
2. Nakopírujte obsah složky home do složky /home/tomcat7/.
3. V nainstalovaném MySQL vytvořte databázi uploadsystem, a poté je potřeba spustit uploadsystem.sql, který databázi naplní potřebnými daty.
4. Vytvořte přes libovolný nástroj ROOT.war soubor ze zdrojových kódů aplikace (Warbler, Eclipse a další).
5. Nakopírujte vytvořený ROOT.war soubor do domácí složky s nahranými aplikacemi v Tomcatu 7.
6. Restartujte Tomcat7.
7. Aplikace by nyní měla být dostupná na adrese podle konfigurace Tomcat 7 souboru server.xml.

B.3 Uživatelská dokumentace

V uživatelské dokumentaci se budu věnovat hlavně pravidlům odevzdávání řešení a poté popisům jednotlivých práv v aplikaci.

B.3.1 Odevzdávání řešení

Při odevzdávání řešení je potřeba mít na paměti několik stanovených pravidel. Je možné mít své řešení ve více třídách/souborech, ale všechny tyto soubory musí být v jedné složce/balíčku. Název tohoto balíčku není limitován a název tříd/souborů také není limitován až na jedno pravidlo, kdy v řešení musí existovat soubor Main, který obsahuje hlavní spouštěcí metodu aplikace main. Dále je potřeba řešení nahrávat v archivu zip, název tohoto archivu je rovněž libovolný. Po nahrání řešení je možné, že nastane kompilační chyba. U této chyby může být uvedený i její zdroj - například, že všechny třídy se nenacházeli v jednom balíčku, či že nebyla nalezena třída Main. Také je možné, že při prohledávání řešení bylo nalezeno více použitých programovacích jazyků. To je rovněž nepřípustné.

Kompilace Java

```
javac *.java &
```

Kompilace C

```
gcc --std=c99 -lm *.c -o Main &
```

Kompilace C++

```
g++ -std=c++11 *.cpp -o Main &
```

B.3.2 Práva v aplikaci

V této části se zaměřím na slovní popis jednotlivých práv v aplikaci. Jednotlivá pravidla jsou seřazena podle názvu.

Název pravidla	Role, kterým je toto právo umožněno	Popis pravidla
changeViewedSemester	Admin, Student, Učitel	Slouží ke změně aktuálního semestru.
createCourse	Admin	Slouží k vytvoření předmětu.
createCourseSemester	Admin, Učitel	Slouží k vytvoření předmětu a předmětu ve vazbě k aktuálnímu semestru.
createRole	Admin	Slouží k vytvoření role.
createSemester	Admin	Slouží k vytvoření semestru.
createTask	Admin, Učitel	Slouží k vytvoření úlohy.
createUser	Admin	Slouží k vytvoření uživatele.
deleteCourse	Admin	Slouží ke smazání předmětu.
deleteCourseSemester	Admin, Učitel	Slouží ke smazání předmětu ve vazbě na aktuální semestr.
deleteRole	Admin	Slouží ke smazání role.
deleteSemester	Admin	Slouží ke smazání semestru.
deleteTask	Admin, Učitel	Slouží ke smazání úlohy.
deleteUser	Admin	Slouží ke smazání uživatele.
detailClassification	Admin, Učitel	Slouží k zobrazení hodnocení úloh.
detailCourse	Admin	Slouží k zobrazení detailu předmětu.
detailCourseSemester	Admin, Student, Učitel	Slouží k zobrazení detailu předmětu ve vazbě k aktuálnímu semestru.
detailOutputFile	Admin, Učitel	Slouží k zobrazení výstupních souborů hodnocení.
detailRole	Admin	Slouží k zobrazení detailu role.
detailSemester	Admin	Slouží k zobrazení detailu semestru.
detailSolution	Admin, Student, Učitel	Slouží k zobrazení detailu řešení.
detailSolutionLog	Admin, Učitel	Slouží k zobrazení detailu řešení.
detailTask	Admin, Student, Učitel	Slouží k zobrazení detailu úlohy.
detailUser	Admin	Slouží k zobrazení detailu uživatele.
duplicateTask	Admin, Učitel	Slouží k duplikování úlohy, tedy vytváření nové úlohy s vybranými stejnými daty staré úlohy.
editCourse	Admin	Slouží k editaci předmětu.
editCourseSemester	Admin, Učitel	Slouží k editaci předmětu ve vazbě na aktuální semestr.
editRole	Admin	Slouží k editaci role.
editSemester	Admin	Slouží k editaci semestru.
editTask	Admin, Učitel	Slouží k editaci úlohy.
editUser	Admin	Slouží k editaci uživatele.
importCourseSemesters	Admin, Učitel	Slouží k importování před-

Příloha C

Obsah přiloženého CD

```
│ source
│ ├── resources
│ └── src
├── home
│ ├── script
│ ├── data
│ ├── temp
│ └── log
├── javadoc
├── latex
└── uploadsystem.sql
```

source - složka se zdrojovými kódy a konfiguračními soubory aplikace

source - resources - složka s vybranými konfiguračními soubory aplikace (konfigurace spring, databáze a application.properties

source - src - zdrojové kódy aplikace, v podsložce webapp se pak nachází konfigurační soubory pretty-faces, faces-config a web.xml

home - obsah domácí složky projektu, tento obsah je nutné zkopírovat do domácí složky projektu tak jak je nakonfigurovaná v application.properties

home-script - složka obsahující nutné skripty k vyhodnocování úloh v podporovaných jazycích

home-data - hlavní datová složka aplikace, obsahuje informace o studentech i předmětech

home-temp - složka s dočasnými soubory, slouží jako přechod pro nahrávané soubory do aplikace

home-log - složka se souborem app.log, který obsahuje log celé aplikace

javadoc - složka s vygenerovaným javadocem

latex - složka se šablonou diplomové práce

uploadsystem.sql - sql skript pro naplnění databáze připravenými nutnými daty při instalaci aplikace