Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Tadeáš Palusga

Systém pro kontrolu domácích úkolů a plagiátorství

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Irena Mlýnková, Ph.D.

Studijní program: Informatika

Studijní obor: Programování

Praha 2012

Na tomto místě bych rád poděkoval vedoucí mé bakalářské práce RNDr. Ireně Mlýnkové, Ph.D. za čas, odborné rady a náměty, které mi během vedení této práce poskytla. Děkuji také své rodině a přátelům za nesmírnou trpělivost.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 24.05.2012 Tadeáš Palusga

Název práce: *Systém pro kontrolu domácích úkolů a plagiátorství*

Autor: *Tadeáš Palusga*

Katedra / Ústav: *Katedra softwarového inženýrství*

Vedoucí bakalářské práce: *RNDr. Irena Mlýnková, Ph.D.*

Abstrakt: *Cílem této práce je systém pro správu skupin, studentů a jejich domácích úkolů. Aplikace poskytuje studentům a pedagogům grafické uživatelské rozhraní (GUI) pro zadávání, odevzdávání a kontrolu zadaných úloh. Kontrola domácích úkolů je řešena pomocí samostatných modulů. Tyto moduly se vyznačují jednotným rozhraním a je možné je instalovat a de-instalovat za běhu. Primárně je program určen pro potřeby předmětu „Databázové systémy", ale díky modulární koncepci je snadno přizpůsobitelný i pro ostatní předměty. Každý modul může implementovat jednoduchou analýzu podobnosti odevzdaných domácích úkolů. Celý systém je napsán v programovacím jazyku Java s využitím frameworku Tapestry. Jako úložný systém je použita databáze MySQL.*

Klíčová slova: *správa domácích úkolů, kontrola správnosti domácích úkolů, kontrola plagiátorství, správa studijních skupin.*

Title: *Homework and Plagiarism Management System*

Author: *Tadeáš Palusga*

Department: *Department of Software Engineering*

Supervisor: *RNDr. Irena Mlýnková, Ph.D., Department of Software Engineering*

Abstract: *The goal of this thesis is creation of a group management system involving the management of students and their home assignments. The application provides graphical user interface for defining and assigning, handing in and verifying of home assignments. Homework check is implemented by standalone modules. These modules are based on unified interface and can be installed or removed on the fly. The primary design of the program is intended for the needs of the subject called "Database systems", but a great emphasis is put on modularity of the solution that should, by the means of easy scalability, enable its use within other subjects. The whole system is written in Java programming language using a MySQL database.*

Keywords: *management of homework, check the validity of the homework, check plagiarism, management of study groups*

Obsah

Úvod

Na většině cvičení jsou dnes zadávány rozličné domácí úkoly. U filozofických oborů se může jednat například o vypracování eseje či pojednání na předem dané téma, v případě matematických oborů se jedná o výpočty zadaných příkladů, u IT oborů pak může jít například o vypracování programu, případně o odevzdání výstupu z programu, nákresu či schematu modelu databáze. Všechny úkoly mají několik společných jmenovatelů:

- Učitel by měl zveřejnit přesné zadání. Toto zadání musí být jednoduše přístupné studentovi.
- Student má možnost úkol vždy nějakým způsobem odevzdat.
- Student musí mít jistotu, že učitel daný úkol obdržel a že se tento úkol neztratil například v síti některého z tichých spamových filtrů.
- Učitel musí ohlídat, zda je úkol odevzdán včas.
- Učitel si musí vést evidenci odevzdaných úkolů a jejich bodového ohodnocení. Student musí mít o tomto hodnocení přehled. V případě nesplnění zadání by měl učitel studentovi vždy vysvětlit, jakých chyb se v úkolu dopustil.
- Student, odevzdává-li cizí dílo, se již z principu k plagiátorství nikdy nepřizná. Odhalení podvodů je tedy čistě záležitostí učitele. Pokud navíc učitel nemá přístup k řešením stejných úkolů zadaných v jiných skupinách, je možnost odhalení kopie prakticky mizivá.
- Učitel by měl odevzdané a splněné úkoly archivovat pro případ, že by student žádal o uznání úkolu z předchozích semestrů nebo o přehodnocení daného řešení.

Hlavním cílem této práce, jak již její samotný název napovídá, je vytvořit nástroj pro

správu a kontrolu domácích úkolů, správu studentských skupin a alespoň základní kontrolu plagiátorství. Nasazení a používání této aplikace vede k usnadnění řešení výše uvedených bodů. Implementovaný systém, o kterém tato práce pojednává, by měl být velmi jednoduše aplikovatelný pro potřeby jakéhokoliv předmětu. V původním znění zadání této práce je základní implementace primárně určena pro potřeby předmětu Technologie XML. Až časem jsme se ovšem s RNDr. Irenou Mlýnkovou, Ph.D. shodli na tom, že aktuálně používaný a odladěný program XMLCheck[1], vypracovaný v rámci bakalářské práce Jana Konopáska, prozatím není třeba nahrazovat. Proto jsme došli k závěru, že by bylo vhodné základní možnosti programu prezentovat na jiném tématu. Základní implementace modulů je tedy namísto předmětu „Technologie XML“ zaměřena na předmět „Databázové systémy“.

Práce je rozdělena do několika kapitol. V první kapitole nalezneme vysvětlení používaných pojmů a stručný úvod do problematiky. V druhé kapitole se seznámíme s existujícími řešeními – programy XMLCheck[1] a CodEx[2] a provedeme jejich stručné srovnání. V kapitole třetí nalezneme uživatelskou dokumentaci – popíšeme instalaci aplikace a její grafické uživatelské rozhraní. V poslední kapitole je zpracována programátorská dokumentace. Rozebereme rozvrstvení aplikace, její strukturu, deteailněji popíšeme zajímavé třídy a vysvětlíme princip kontroly domácích úkolů a kontrolu plagiarity.

1. Základní koncepce aplikace

V této kapitole vysvětlíme základní pojmy a ve stručnosti představíme tři základní koncepce aplikace HWChecker:

- **použitelnost v různých předmětech**
- **kontrola podobnosti odevzdaných úkolů**
- **zjednodušení práce učitelům**

1.1. Základní terminologie

- **HWChecker** – název systému pro správu a kontrolu domácích úkolů implementujícího tuto práci.
- **Skupina** – množina studentů docházejících na stejné cvičení v rámci jedné přednášky.
- **Modul** – za běhu instalovatelná a odebíratelná část aplikace sloužící ke kontrole správnosti a podobnosti odevzdaných domácích úkolů.
- **Definice domácího úkolu** – předdefinované zadání domácího úkolu. Definice jsou definovány pomocí kontrolních modulů.
- **Instance domácího úkolu** – zadání úkolu v konkrétní skupině. Instance je definována definicí domácího úkolu, skupinou a mezním datem pro odevzdání.
- **Nepotvrzené řešení** – řešení, jež bylo nahráno do aplikace, ale nebylo zasláno učiteli ke kontrole.
- **Potvrzené řešení** – řešení, které bylo nahráno do aplikace a bylo zasláno učiteli ke kontrole.

- **Zkontrolované řešení** – řešení, které bylo nahráno do aplikace, zasláno učiteli ke kontrole, a tímto učitelem zkontrolováno.
- **Uživatel** – pokud mluvíme o uživateli obecně, máme na mysli uživatele s libovolnou uživatelskou rolí.
    - **Administrátor** – uživatel s rolí administrátor. Správce aplikace.
    - **Učitel** – uživatel s rolí učitel. Spravuje skupiny, definuje domácí úkoly, zadává instance domácích úkolů, opravuje úkoly.
    - **Student** – uživatel s rolí student. Řeší a odevzdává domácí úkoly.

1.2. Použitelnost v různých předmětech

Pokud máme naprogramovat aplikaci, která má mít podobné využití v různých prostředích bez zásadních úprav, musíme o návrhu takové aplikace uvažovat již dopředu. K takovému návrhu lze přistupovat ze dvou stran.

První způsob, návrhově i implementačně jednoduchý a o to více neefektivní, je programování **hrubou silou**. V takovém případě bychom jakoukoliv funkcionalitu začlenili přímo do vyvíjeného software. V případě potřeby rozšíření by pak bylo nutné libovolnou novou funkcionalitu vepsat přímo do programu. Ta by poté musela projít zdlouhavým testovacím cyklem, abychom ověřili, že úprava nezpůsobuje neočekávané chování v jiných částech takové aplikace a v neposlední řadě by bylo nutné provést novou instalaci. Takový vývojový cyklus může být u jednoduchých aplikací výhodný a rychlý, pro složitější aplikace se však stává zbytečně komplikovaným a netestovatelným.

Druhý způsob je z pohledu návrhu složitější. Spočívá v rozdělení aplikace do funkčních celků, kdy každý z nich je samostatně testovatelný a má na starosti pouze jeden konkrétní problém, který řeší. V takovém případě znamená rozšíření

funkcionality naprogramování nového celku, jeho samostatné otestovaní a **přiroubování** k již existujícímu a nainstalovanému programu. Odpadá tedy nutnost testování celé aplikace vždy od začátku a problémem není ani umožnění přidání nových funkcionalit třetím stranám. Složitost návrhu pak vyváží efektivnost vývoje a testovatelnost.

Návrh naší aplikace vychází z druhého popsaného způsobu. Aplikaci jsme rozdělili do tří částí. První z nich zahrnuje grafické uživatelské rozhraní, které je pro libovolné předměty identické. Druhou částí je veřejné rozhraní modulů zajišťujících kontrolu správnosti domácích úkolů a jejich podobností. Třetí částí jsou tyto moduly samotné.

1.3. Kontrola podobnosti domácích úkolů

Domácí úkoly z různých předmětů mohou být velice odlišné. Podobnost dvou XML souborů nelze kontrolovat stejně jako podobnost dvou textových souborů. Podobnost obrázků zase nelze kontrolovat obdobně jako podobnost zdrojových kódů v jazyce Java. Navíc každá kontrola souborů stejných typů může mít svá specifika – při kontrole jednoho druhu XML nás nemusí zajímat elementy určitého typu nebo u jiných XML souborů chceme ignorovat určité atributy. Podobností a plagiátorstvím se zabývají mnohé studie a detailní rozbor či implementace by byly tématem pro samostatnou bakalářskou práci.

Jednoduchou kontrolu podobnosti ovšem v naší aplikaci nalézt lze, a to v předpřipraveném rozhraní pro tvorbu kontrolních modulů. Implementace takové kontroly není v modulech povinná, avšak v dodaných testovacích modulech je implementovaná. Detailnímu popisu modulů je věnována kapitola 4.9.

1.4. Zjednodušení práce učitelům

Jak již bylo v úvodu předestřeno, hlavním cílem této práce je implementovat systém, který by (hlavně) učitelům co nejvíce zjednodušil kontrolu a hodnocení domácích úkolů. Implementovaná aplikace je proto koncipována pro co nejintuitivnější použití a snaží se eliminovat počet externích aplikací nutných pro rozbalování, otevírání a prohlížení a pro jinou práci spojenou s kontrolou odevzdaných domácích úkolů.

2. Analýza existujících řešení

V této části textu se pokusíme rozebrat existující řešení a nastínit hlavní rozdíly oproti naší implementaci. V první podkapitole se zaměříme na program **XMLCheck**[1], který byl dokončen v roce 2012 jako bakalářská práce Jana Konopáska na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze. V podkapitole druhé jsou nastíněny stěžejní rozdíly mezi naší aplikací a aplikací **CodEx**[2] (The Code Examiner), která je vyvíjena od roku 2006 rovněž na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze.

2.1. XMLCheck Assignment Manager

Program **XMLCheck Assignment Manager**[1] (český název: Systém pro správu úkolů a testů. Dále jen XMLCheck) je výsledkem bakalářské práce Jana Konopáska. V letech 2011 a 2012 prošel testovacím provozem a od letního semestru 2012 je využíván jako hlavní systém pro správu domácích úkolů v předmětu **Technologie XML**. Jedná se o systém, který vykazuje s námi vytvořeným systémem těchto několik společných rysů:

- Spravuje studijní skupiny a studenty v těchto skupinách.
- Skrze tento systém jsou zadávány, odevzdávány a kontrolovány domácí úlohy z daného předmětu.
- Systém je možno jednoduchým modulárním způsobem rozšířit o další funkcionality.

Na druhou stranu je systém v mnoha ohledech koncipován odlišně. Jednotlivé odlišnosti rozebereme v následujících podkapitolách.

2.1.1. Rozdílná koncepce uživatelských rolí

XMLCheck má velmi pokročilou správu uživatelských rolí. Role je možné přidávat, a každé z nich je možné definovat určitá privilegia. Nemyslíme si ovšem, že v programu tohoto určení je editovatelnost rolí právě nezbytnou vlastností. Na defaultním nastavení aplikace XMLCheck ukážeme, že již toto rozdělení uživatelských rolí vykazuje známky redundantnosti a složitosti.

Úroveň **student** je v obou programech téměř identická. Úroveň **teacher/učitel** je ovšem v aplikaci XMLCheck reprezentována dvěma. První z nich je **lecturer**, tedy jakýsi učitel, který má na starosti celý předmět. Lecturer se stará zejména o správu daného předmětu/přednášky, vytváření domácích úkolů a o nahrávání modulů. Druhou úrovní je **tutor**. Ten má na starosti správu jednotlivých skupin, kterým nadefinované úkoly zadává a kontroluje je. V programu HWChecker vlastníka přednášky není nutné brát v potaz. Pro studentovu orientaci ve skupinách to je svým způsobem redundantní informace. Pro učitele v tomto případě není problémem, pokud se rozhodne nadefinovat například speciální domácí úkol v rámci dané skupiny (může se jednat o úkol náhradní či alternativní). Navíc z hlediska bezpečnosti je kupříkladu samotné nahrání neotestovaných modulů do aplikace velkým bezpečnostním rizikem, jelikož každý modul je samostatnou spustitelnou aplikační jednotkou. Takové riziko je lepší směrovat až na administrátora dané aplikace.

2.1.2. Podpora učitele při kontrole odevzdaných řešení

Při vytváření programu podobného typu je potřeba neopomínat potřeby obou zúčastněných stran – v tomto případě učitelů a studentů. Podpora pro studenta je v aplikaci J. Konopáska dostatečná – student se přihlásí do předem nadefinované skupiny, posléze v této skupině zkouší odevzdat domácí úkoly a pokud projdou validací, student je potvrdí a zašle ke kontrole učitelem. V tomto směru se naše

implementace výrazně neliší. Liší se ale v koncepci podpory pro kontrolu domácích úkolů učitelem. Aplikace XMLCheck umožňuje učiteli pouze omezenou kontrolu odevzdaného úkolu. Je možné prohlédnout si výsledek kontroly (případně detailnější výpis chyb) a pro důkladnější revizi je zde možnost domácí úkol si stáhnout a zkontrolovat lokálně. To je samozřejmě možnost korektní, avšak zcela eliminuje jeden z požadavků na kontrolní aplikaci, a to požadavek na co nejméně úkonů spojených s manuální kontrolou. Vhodnější způsob spočívá v umožnění náhledu do souborů v odeslaném úkolu přímo z běžící instance aplikace. Jak již bylo uvedeno v úvodu této práce, učitel pak není omezen na jemu známé prostředí, ale může úkoly v téměř identické míře hodnotit i z jiných počítačů, tabletových zařízení, zajdeme-li do extrému, například i z mobilních zařízení s vestavěným prohlížečem.

2.1.3. Zdůvodnění bodového ohodnocení ze strany učitele

Dalším specifikem, kterým se obě aplikace liší, je možnost odůvodnění ztráty/zisku bodového ohodnocení odevzdaného úkolu. V aplikaci XMLCheck je možno po stáhnutí a lokální kontrole daný úkol ohodnotit, chybí ovšem jakákoliv možnost přiřadit ke zkontrolovanému úkolu vhodný komentář. Tuto možnost naše implementace zahrnuje a zjednodušuje tím komunikaci ve směru učitel-student, kdy student je o důvodech svého neúspěchu/úspěchu vhodně informován.

2.1.4. Kontrolní moduly

Design kontrolních modulů je v obou srovnávaných aplikacích (tedy v aplikaci HWChecker i XMLCheck) podobný, i tak má ale každá aplikace svá specifika. V obou případech se jedná o balíčky s určitou strukturou, které konzumují vstup ve formě adresáře s rozbaleným úkolem a produkují výstup. V případě XMLCheck se jedná o cestu k souboru s výpisem vygenerovaných výstupů, v naší

implementaci pak o jednoduchý objekt se zprávami z průběhu kontroly. Na jedné straně je generování speciálních souborů určitě užitečné, na straně druhé není žádný důvod v případě potřeby neukládat výstupní soubory přímo do adresáře s rozbaleným domácím úkolem. V takovém případě pak může učitel generované výstupy procházet skrze identické webové rozhraní, přes které prochází samotný domácí úkol.

2.1.5. Design versus účel

Design aplikace J. Konopáska je velice zdařilý. V nastavení nalezneme dokonce i možnost přepínání mezi více předdefinovanými styly. V programu HWChecker jsme však zvolili přístup poněkud odlišný. Rozhodli jsme se jít cestou co nejjednoduššího grafického layoutu s důrazem na funkčnost a rychlost aplikace na všech možných zařízeních s internetovým prohlížečem. Základní funkce naší aplikace jsou tedy dostupné i v prohlížečích bez zapnutého JavaScriptu.

2.1.6. Provázanost jednotlivých komponent

V aplikaci J. Konopáska chybí provázání mezi jednotlivými kompo-nentami. V naší aplikaci je například možnost prokliku z detailu učitele na detail jeho skupiny, kde můžeme vidět seznam všech studentů a zadaných úkolů. Odtud se opět jednoduchým proklikem můžeme dostat na detail uživatele, kde jako vyučující (jsme-li jimi) vidíme studentovy odevzdané úkoly a také to, do jaké skupiny patří. Proklikem na odevzdaný úkol vidíme řešení, které můžeme rovnou opravit, případně se můžeme prokliknout na instanci úkolu nebo jeho rodičovskou definici, ze které se lze prokliknout na detail samotného modulu.

2.2. CodEx

CodEx[2] je aplikace určená pro odevzdávání a automatické vyhodnocování správnosti úkolů z kurzů programování. Domácí úkol je v kontextu systému CodEx soubor se zdrojovými kódy v daném jazyce, který musí v co nejkratším čase s co nejmenší paměťovou náročností pro předem nadefinované vstupy vygenerovat odpovídající výstupy. Na základě výše zmíněné paměťové a časové náročnosti a korektnosti výstupů je domácí úkol automaticky ohodnocen odpovídajícím počtem bodů.

HWChecker sdílí s aplikací CodEx společný koncept – obě slouží jako rozhraní mezi studenty a učiteli, kde učitelé zveřejňují časově omezená zadání, na něž studenti v daném termínu odevzdávají vypracovaná řešení. Správa skupin je v obou případech tříúrovňová.

2.2.1. Modulárnost

Program CodEx je jednoúčelový – v aktuální verzi slouží k zadávání, odevzdávání a automatizovanému vyhodnocování úkolů v programovacích jazycích Pascal, C/C++, C# a Haskel. Veškeré kontrolní funkce jsou do programu implementovány napevno. HWChecker veškeré kontrolní algoritmy implementuje až v kontrolních pluginech a díky tomu je lze vyvíjet nezávisle.

2.2.2. Kontrola odevzdaných řešení

Koncepce kontroly a vyhodnocování úkolů je velmi rozdílná. Při kontrole výstupu zkompilovaného zdrojového kódu je možné očekávat konkrétní výstupy. Takové výstupy nelze očekávat u nealgoritmických zadání. HWChecker se snaží minimalizovat počet úkonů potřebných pro zkontrolování jednotlivé úlohy.

3. Uživatelská dokumentace

V této kapitole se dozvíme vše potřebné pro bezproblémovou instalaci a provoz programu HWChecker. V první podkapitole popíšeme požadavky, které by měl hostitelský stroj splňovat, postup samotné instalace a konfigurace aplikace. V dalších podkapitolách potom nalezneme rozbor povolených akcí z jednotlivých uživatelských účtů.

3.1. Instalace webového GUI rozhraní aplikace

3.1.1. Požadavky

Předpokladem pro úspěšnou instalaci programu HWChecker jsou nainstalované následující programy:

- Databázový server **MySQL** ve verzi 5.5 nebo vyšší (aplikaci by nic nemělo bránit fungovat i s nižšími verzemi, na nichž ovšem nebyla otestována).
- Nainstalovaný kontejner **Apache Tomcat** ve verzi 6 nebo vyšší.
- **JRE 7** (Java Runtime Environment).
- **SMTP** klient pro odesílání pošty.
- **Textový editor** pro úpravu konfiguračních souborů.

3.1.2. Vytvoření adresářové struktury a rozbalení aplikace

Pro potřeby instalace si na hostitelském serveru vyhradíme adresář, do kterého budeme umisťovat veškeré soubory a podadresáře nutné pro korektní běh

aplikace. Zástupný symbol *„{app_root}“* bude ve zbývající části tohoto textu označovat absolutní cestu právě k tomuto adresáři. Doporučujeme (ačkoliv to není bezpodmínečně nutné) přesně dodržovat adresářovou strukturu uváděnou v tomto textu.

Front-endová GUI (graphical user interface) část aplikace je distribuována ve formě WAR[1] archivu, který je pod názvem *„web2.war“* dostupný v adresáři *„bin“* na přiloženém disku. Archiv obsahuje veškeré spustitelné zdrojové kódy aplikace a příslušné konfigurační soubory. Tento soubor je třeba rozbalit do předem připraveného adresáře. HWChecker ukládá nahrané moduly a odevzdané domácí úlohy do adresářů na pevném disku. Samotná adresářová struktura může vypadat obdobně jako struktura znázorněná na obrázku 3.1.

obr. 3.1 - vzorová adresářová struktura

- *context* – adresář, do kterého rozbalíme WAR archiv web2.war
- *modules* – adresář, do kterého budou instalovány kontrolní moduly
- *homeworks* – adresář, do kterého budou ukládány odevzdané domácí úkoly

1 Web Application aRchive – podobně jako JAR (Java ARchive) je archiv formátu ZIP, používaný pro distribuci desktopových programů psaných v Jazyce Java, WAR archiv je archiv formátu ZIP, používaný pro distribuci webových aplikací psaných v jazyce Java.

3.1.3. Založení uživatele v MySQL a vytvoření prázdné databáze

V MySQL je potřeba založit uživatele, jehož prostřednictvím bude aplikace s databází komunikovat. Dále je potřeba vytvořit databázi, ke které bude mít daný uživatel plný přístup. Uživatele i databázi nazveme jednotně *„checker“*.

Pro vytvoření uživatele použijeme následující skript:

```
CREATE USER 'checker'@'localhost' IDENTIFIED BY '*****';
GRANT ALL PRIVILEGES ON `checker`.* TO 'checker'@'localhost';
FLUSH PRIVILEGES;
```

Pro vytvoření prázdné databáze použijeme následující skript:

```
CREATE DATABASE `checker` DEFAULT CHARACTER SET utf8 COLLATE
utf8_czech_ci ;
```

3.1.4. Nastavení konfiguračního souboru aplikace

Konfigurační soubor *„application.cfg“* se nachází v adresáři *„{app_root}/context/WEB-INF/“*. Struktura tohoto souboru je velmi jednoduchá. Jakýkoliv řádek začínající znakem *'#'* nebo prázdný řádek je ignorován. Validní konfigurační řádek je uspořádaná trojice [sekce, klíč, hodnota], kde sekce je na řádku uvedena jako první následovaná znakem *'.'* a názvem klíče. Poté následuje znak *'='* a libovolný řetězec za tímto znakem až do konce řádku je považován za hodnotu pro danou dvojici [klíč, hodnota].

V konfiguračním souboru je třeba nastavit následující tři povinné parametry:

- **MODULES.DIR** – jako hodnutu uvedeme absolutní cestu k adresáři, do kterého budou nahrávány nainstalované moduly. V případě, že jsme dodrželi adresářovou strukturu zadefinovanou v kapitole 3.1.2, jako hodnotu zadáme *„{app_root}/modules“*
- **SUBMITTED_HOMEWORKS.DIR** – jako hodnotu uvedeme absolutní

cestu k adresáři, do kterého budou ukládány odevzdané domácí úkoly. V případě, že jsme dodrželi adresářovou strukturu zadefinovanou v kapitole 3.1.2, jako hodnotu zadáme *„{app_root}/homeworks“*

- **MAIL.FROM –** jako hodnotu uvedeme e-mailovou adresu, která se bude zobrazovat v poli *„OD“* uživatelům, kterým aplikace z různých důvodů zasílá e-mail. (např. *„no-reply@hwchecker.dnrs“*)

3.1.5. Vytvoření konfiguračního XML souboru pro inicializaci aplikace a jejího spojení na databázi z kontejneru Tomcat

Vytvoříme soubor *„{tomcat_root}/conf/Catalina/localhost/checker.xml“*, v němž nadefinujeme kontext[2] k naší aplikaci, cestu ke zdrojovým souborům naší aplikace a datový zdroj *„jdbc/checker“*, který HWChecker očekává. Pokud jsme dodrželi adresářovou strukturu zadefinovanou v kapitole 3.1.2 a jméno uživatele a databáze zadefinované v kapitole 3.1.3, konfigurační soubor bude vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/checker" docBase="{app_root}/context">
 <Resource name="jdbc/checker"
  auth="Container"
  type="javax.sql.DataSource"
  username="checker"
  password="*****"
  driverClassName="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost:3306/checker?characterEncoding=utf-8"
 />
</Context>
```

2 kontext / context / context path - tato cesta se porovnává s URI requesty přicházejícími na server. Aplikace zpracuje příslušný požadavek právě tehdy, pokud URI request začíná řetězcem definovaným jako context path pro příslušnou aplikaci.

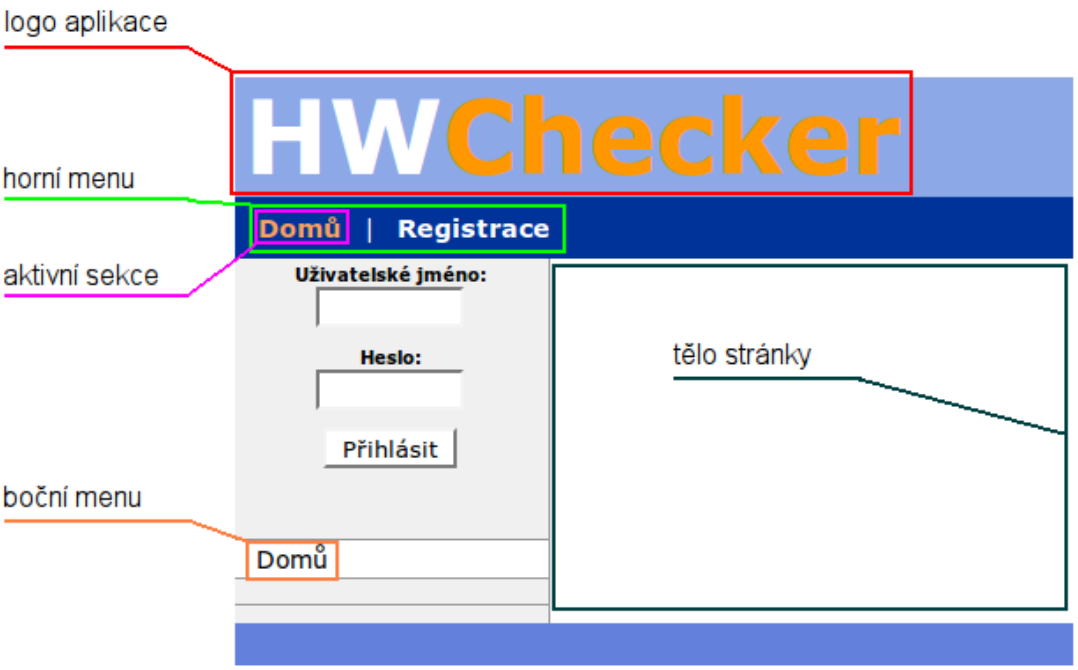
3.1.6. První spuštění aplikace

Pokud nyní nastartujeme kontejner Apache Tomcat, spustí se také poprvé HWChecker. Při prvním spuštění budou vytvořeny v databázi všechny potřebné tabulky. Od této chvíle je aplikace připravena k použití na adrese *„http://localhost:8080/checker“*.

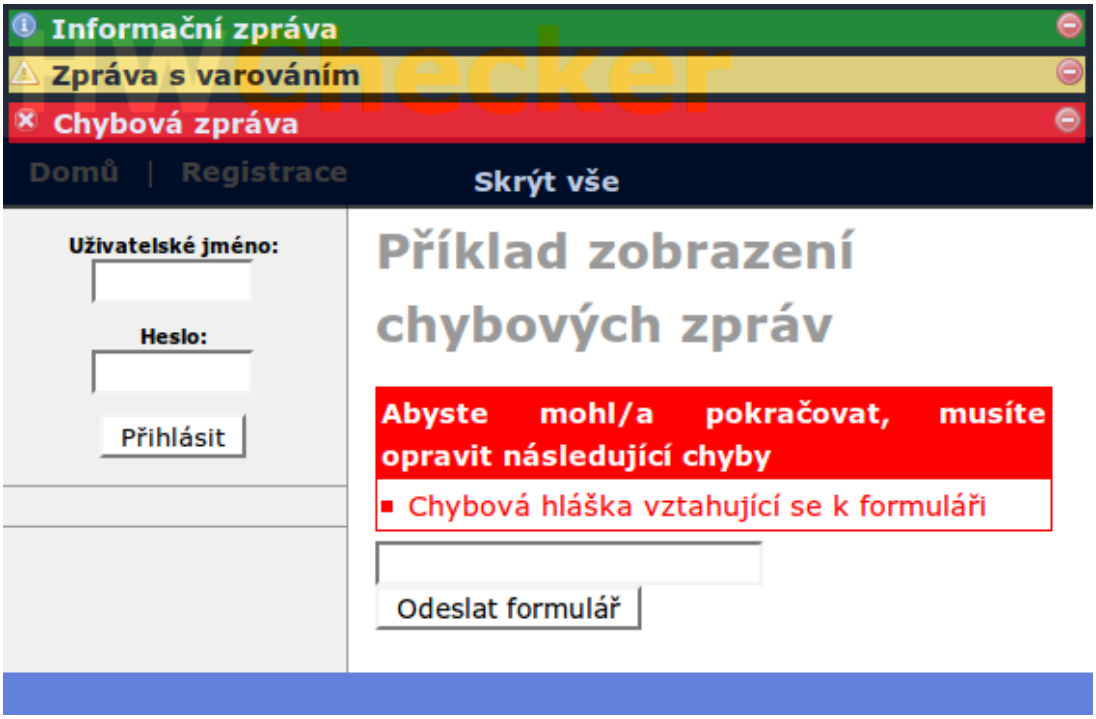
3.2. Rozvržení aplikace

Pro rozvržení aplikace byl zvolen jednoduchý dvousloupcový layout s hlavičkou a patičkou. Hlavním prvkem hlavičky je logo „HWChecker“ s možností prokliku na úvodní stránku. Ve spodní části hlavičky se pak vyskytuje menu pro navigaci po jednotlivých **sekcích** aplikace (dále jen **horní menu**), v levém sloupci stránky se vyskytuje menu pro navigaci po jednotlivých **podsekcích** dané sekce (dále jen **boční menu**) a v pravém sloupci stránky se vyskytuje samotné tělo stránky. Název aktuálně zobrazené sekce (dále jen **aktivní sekce**) je v horním menu zvýrazněn oranžovým písmem. Hlavní interakce aplikace s uživatelem probíhá vždy v tělu stránky. Layout aplikace je odvozen od volně dostupného layoutu Sinorcaish[3]. Rozvržení aplikace je ilustrováno na obrázku 3.2.

Pro chybová, varovná a informativní hlášení je vyhrazena horní část aplikace, kde je zpráva v případě potřeb ajaxově zobrazena. Chybové hlášky z formulářů jsou zobrazovány v červeném poli nad konkrétním formulářem. Zobrazování zpráv je ilustrováno na obrázku 3.3.



obr. 3.2 – rozvržení layoutu aplikace

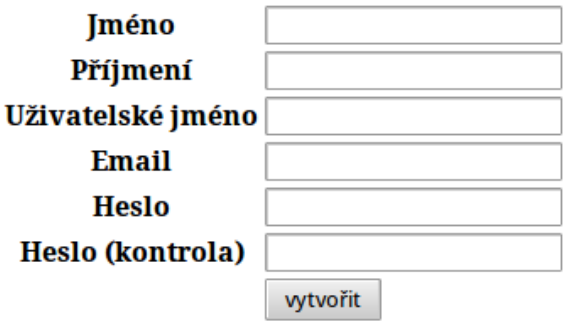


obr. 3.3 – chybové zprávy

3.3. Práce s aplikací v roli administrátora

3.3.1. Vytvoření administrátorského účtu

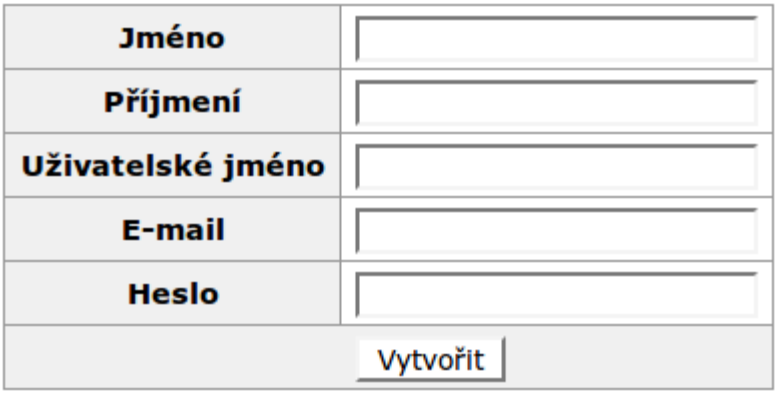
Po základní instalaci v aplikaci není vytvořen administrátorský účet. Pro jeho vytvoření navštivte stránku na adrese *„http://localhost:8080/install.html“*. Tato stránka obsahuje formulář pro vytvoření administrátorského účtu (obrázek 3.4). Po korektním vyplnění tohoto formuláře a jeho odeslání je uživatel přesměrován na domácí stránku aplikace a je informován o úspěšném vytvoření administrátorského účtu. Pod takto vytvořeným účtem se uživatel nyní může přihlašovat. Od této chvíle je stránka *„install.html“* pro přístup trvale blokována.



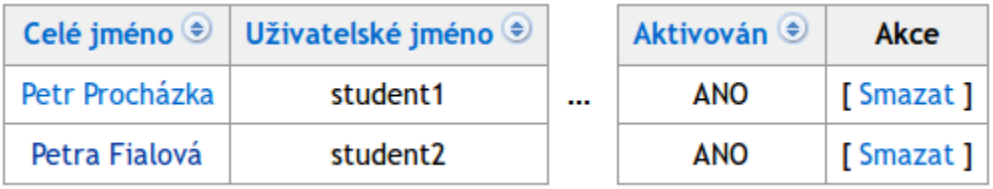
obr. 3.4 – vytvoření administrátorského účtu

3.3.2. Založení učitelského účtu, mazání uživatelů

Administrátor, jako jediný, má možnost vytvářet nové učitelské účty. Pro vytvoření takového účtu nejprve přejděte do podsekce *„Vytvořit učitele“* sekce *„Uživatelé“*. Zde vyplňte potřebné údaje a potvrďte stlačením tlačítka *„Vytvořit“*. Vytvoření učitelského účtu je znázorněno na obrázku 3.5. Pouze administrátor má právo mazat existující uživatelské účty. Toho lze docílit v podsekci *„Uživatelé“* sekce *„Uživatelé“* kliknutím na příslušný odkaz *„Smazat“* (obr. 3.6).



obr. 3.5 – vytvoření nového učitelského účtu

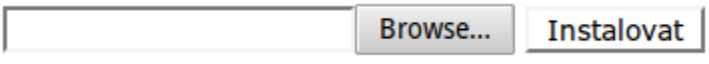


obr. 3.6 – smazání existujícího uživatele

3.3.3. Instalace, odinstalace, aktivace a deaktivace modulů

Uživatel s administrátorskými právy může jako jediný instalovat, odinstalovat, aktivovat nebo deaktivovat kontrolní moduly. Zmíněné akce lze provádět v příslušných podsekcích sekce *„Moduly“*.

Podsekce pro instalaci nového modulu se nazývá ***„Instalovat nový“*** a instalační formulář je znázorněn na obrázku 3.7. Zde administrátor vybere příslušný soubor modulu na disku a stiskem tlačítka *„Instalovat“* jej nainstaluje. V případě neúspěchu bude informován klasickou chybovou tabulkou nad instalačním formulářem. V případě úspěšné instalace bude přesměrován na stránku se seznamem nahraných modulů.

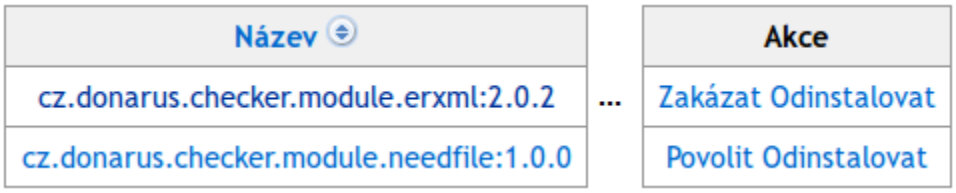


obr. 3.7– instalace nového modulu

Modul se může nacházet v aktivovaném/povoleném nebo také v deaktivovaném/zakázaném stavu. Pokud deaktivujeme modul, který je používán v některé z existujících definic domácích úkolů, pak v této definici zůstane nedotčen do doby její modifikace oddefinováním tohoto modulu nebo jeho úplné odinstalace. Modul v deaktivovaném stavu ovšem není možné v žádné nové ani existující definici domácího úkolu přidat. Standardně je modul instalován jako deaktivovaný.

Odinstalaci, aktivaci či deaktivaci modulu provedeme v podsekci „Moduly“ kliknutím na příslušný odkaz (obr. 3.8).

Pokud je modul odinstalován, je odstraněn ze všech stávajících definic domácích úkolů, v nichž je použit. Před smazáním úkolu je tedy doporučeno modul nejprve deaktivovat, a teprve po ujištění, že již není používán v žádné ze zadaných definic, jej smazat. Zda je v některých definicích modul používán, lze zjistit po zobrazení jeho detailu, který vyvoláme kliknutím na jeho název.



obr. 3.8 – odinstalace, aktivace/povolení a deaktivace/zákaz modulu

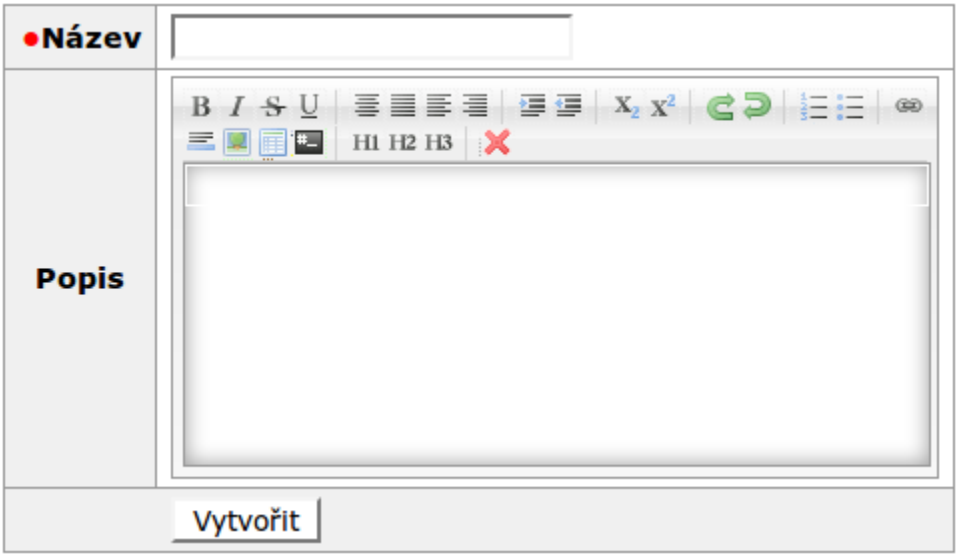
3.3.4. Ostatní možné úkony

- **Mazání a editace existujících skupin** – Tuto akci lze provést kliknutím na příslušný odkaz v sekci *„Skupiny“*.
- **Mazání a editace existujících definic domácích úkolů** – Tuto akci lze provést kliknutím na příslušný odkaz v sekci *„Definice domácích úkolů“*.
- **Mazání a editace existujících instancí domácích úkolů** – Tuto akci lze provést kliknutím na příslušný odkaz v sekci *„Instance domácích úkolů“*.

3.4. Práce s aplikací v roli učitele

3.4.1. Vytvoření skupiny

Učitel může mít na starosti libovolný počet skupin. Název skupiny v rámci daného systému musí být unikátní a měl by být volen s ohledem na co největší jednoznačnost. U každé skupiny může její autor vyplnit jednoduchý popis – doporučeným obsahem popisu je čas a místo konání cvičení dané skupiny, požadavky na zápočet, konzultační hodiny a kontakt na učitele. Tyto údaje jsou viditelné všem uživatelům. Skupinu je možné vytvořit pomocí formuláře v podsekci *„Vytvořit novou“* sekce *„Skupiny“*. Ilustrace k vytvoření je na obrázku 3.9.



obr. 3.9 – založení nové skupiny

3.4.2. Ostranění a editace existující skupiny

Skupinu, jejíž je autorem, může učitel smazat nebo upravit. Pro smazání slouží odkaz *„Smazat“* v podsekci *„Skupiny“* sekce *„Skupiny“*, pro editaci slouží odkaz *„Upravit“* v téže podsekci. Mazání a editaci skupin ilustruje obrázek 3.10.

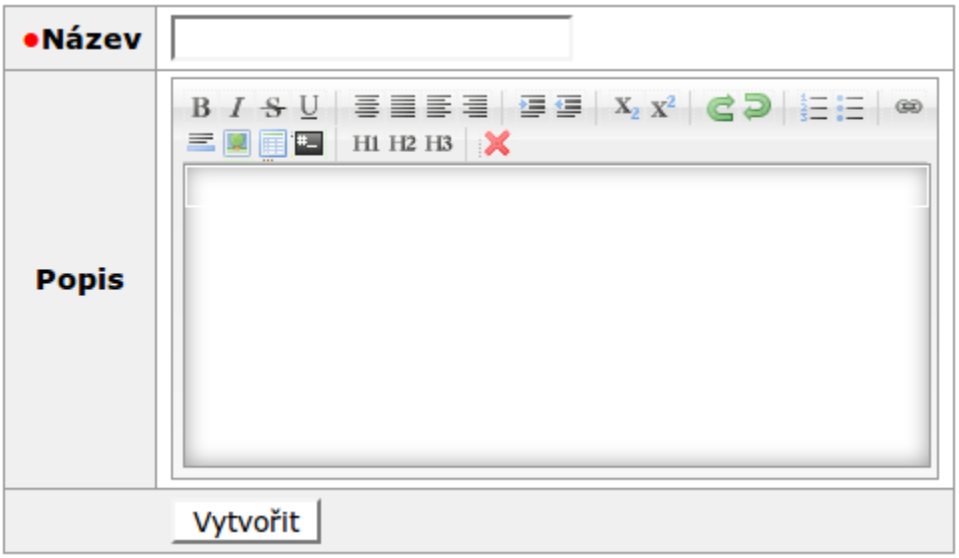
Je-li skupina odstraněna, veškeré instance domácích úkolů v této skupině budou zrušeny a všichni studenti budou z takové skupiny vyloučeni.

Název		Akce
CVUT1 pondělí 18:00 - Veselý		[Smazat] [Upravit]
CVUT2 - úterý 19:00 - Veselý	...	[Smazat] [Upravit]
MFF1 - pondělí 08:00 - Konečná		
MFF2 - středa 06:00 - Konečná		

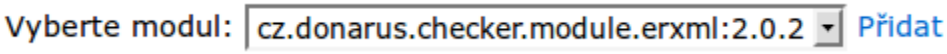
obr. 3.10 – smazání a editace skupiny

3.4.3. Vytvoření definice domácího úkolu

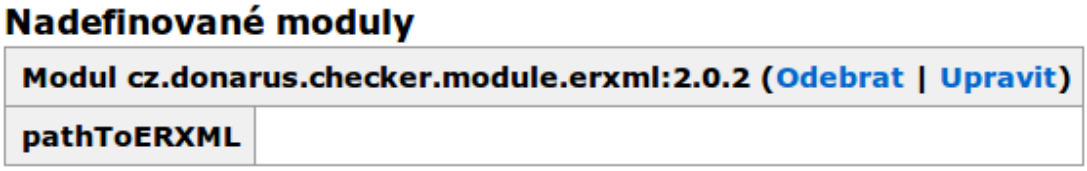
Definici domácího úkolu je možné vytvořit pomocí formuláře v podsekci *„Přidat novou definici“* sekce *„Definice DÚ“* (obrázek 3.11). Zde je třeba uvést název definice úkolu (například: „*Úkol č.1, varianta 1, Vytvoření ER modelu v aplikaci ER Modelář*“) a detailní zadání domácího úkolu. Popis by měl být vyplněn co nejdetailněji. Použijeme-li kupříkladu modul, který kontroluje přítomnost daného souboru s názvem „*erxml.xml*“ v odevzdaném ZIP souboru (domácím úkolu), může popis vypadat následovně: „*V archivu domácího úkolu je očekáván uložený výstupní soubor z programu ER Modelář. Soubor pojmenujte erxml.xml.*“. Po kliknutí na tlačítko *„Vytvořit“* bude úkol vytvořen a dojde k zobrazení stránky s rozšířeným formulářem pro editaci úkolu. Na této stránce je možné nadefinovat, které moduly budou v definici domácího úkolu využívány (obrázek 3.12). Po přidání se modul zobrazí v části *„Nadefinované moduly“* (obrázek 3.13). V této části je možné stiskem tlačítka *„Odebrat“* modul z definice odstranit, případně stiskem tlačítka *„Upravit“* modul upravit. Zvolíme-li možnost upravit, bude zobrazen příslušný editační formulář, ve kterém je možno nastavit modulu potřebné parametry (obrázek 3.14). Kontrolu správnosti zadaných parametrů provedeme stisknutím tlačítka *„Zkontrolovat“*. Nastavení uložíme kliknutím na odkaz *„Uložit“*.



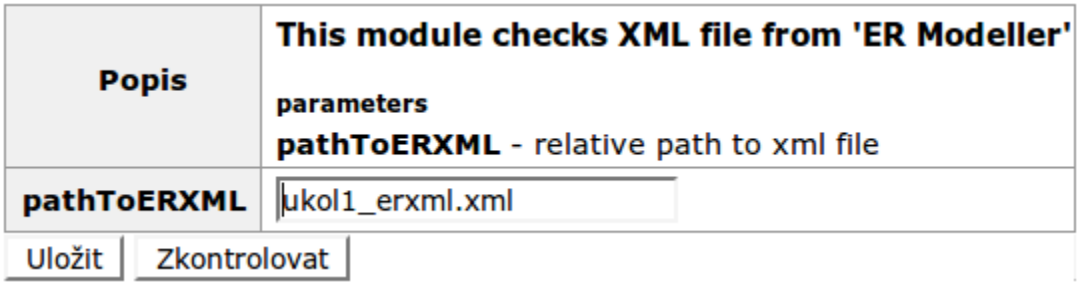
obr. 3.11 – Vytvoření nové definice domácího úkolu



obr. 3.12 – Přidání modulu do definice



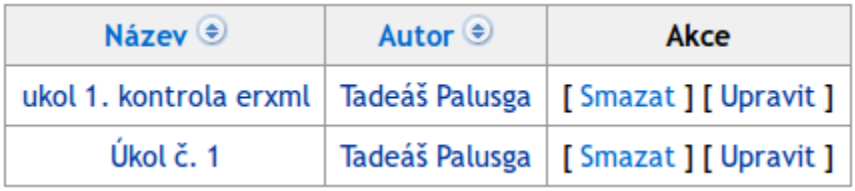
obr. 3.13 – volba editace nebo odstranění modulu



obr. 3.14 – editace nastavení modulu

3.4.4. Odstranění a editace existující definice domácího úkolu

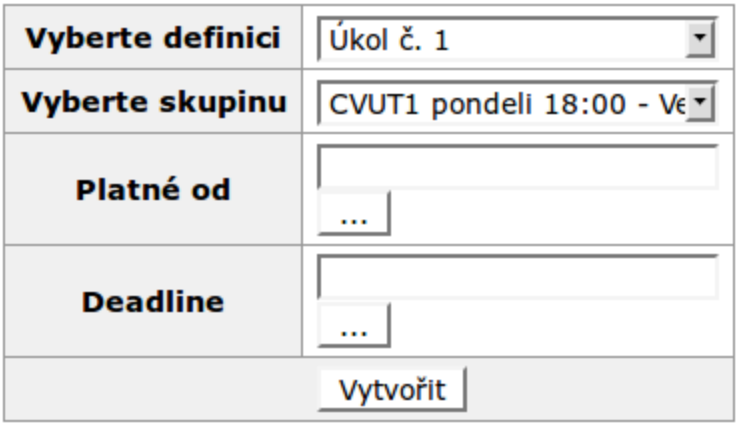
Definici, jejíž je autorem, může učitel smazat nebo upravit. Pro smazání slouží odkaz *„Smazat“* v podsekci *„Definice DÚ“* sekce *„Definice DÚ“*, pro editaci slouží odkaz *„Upravit“* v téže podsekci. Mazání a editaci definic ilustruje obrázek 3.15. Odstraníme-li definici úkolu, budou všechny instance odvozené od této definice odstraněny společně se všemi odevzdanými úkoly pro tyto instance.



obr. 3.15 – Smazání a editace definice domácího úkolu

3.4.5. Zadání domácího úkolu ve skupině

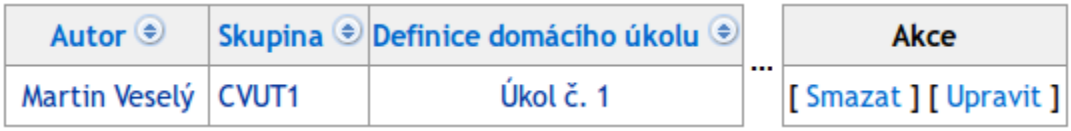
Pro zadání konkrétního domácícho úkolu určité skupině je třeba vytvořit instanci definice domácího úkolu. To lze provést v podsekci *„Zadat instanci DÚ skupině“* sekce *„Instance DÚ“*. Zde nalezneme formulář, ve kterém je potřeba vybrat skupinu, definici úkolu, který chceme zadat, datum zveřejnění a mezní datum pro odeslání. Vytvoření instance je ilustrováno obrázkem 3.16.



obr. 3.16 – Zadání instance definice domácího úkolu

3.4.6. Odstranění a editace existující instance domácího úkolu

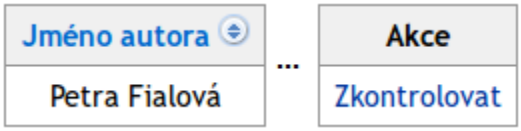
Instanci, jejíž je autorem, může učitel smazat nebo upravit. Pro smazání slouží odkaz *„Smazat“* v podsekci *„Instance DÚ“* sekce *„Instance DÚ“*, pro editaci slouží odkaz *„Upravit“* v téže podsekci. Odstraníme-li instanci definice úkolu, budou odstraněny i všechny instance této definice a k nim odevzdané úkoly. Mazání a editaci instancí ilustruje obrázek 3.17.



obr. 3.17 – Smazání a editace instance definice domácího úkolu

3.4.7. Prohlížení a kontrola odevzdaných domácích úkolů

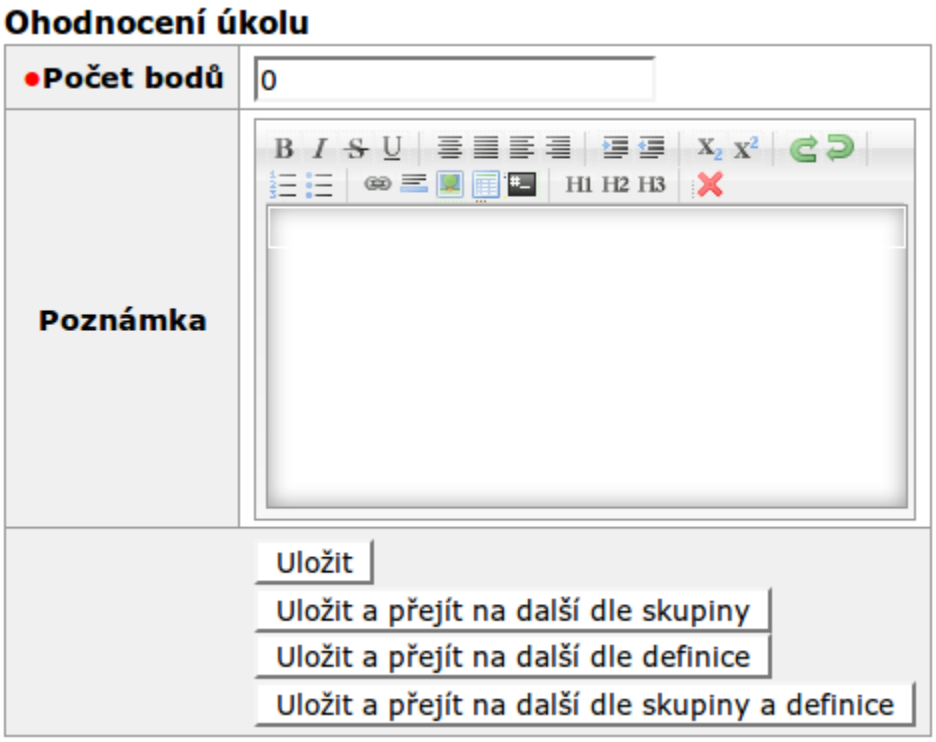
Existuje-li nějaký domácí úkol ke kontrole, je zobrazen v podsekci *„Nezkontrolované domácí úkoly“* sekce *„Domácí úkoly“*. Pokud chceme takový úkol zkontrolovat, stačí kliknout na příslušný odkaz *„Zkontrolovat“* (obr. 3.18).



obr. 3.18 – Výběr úkolu ke zkontrolování

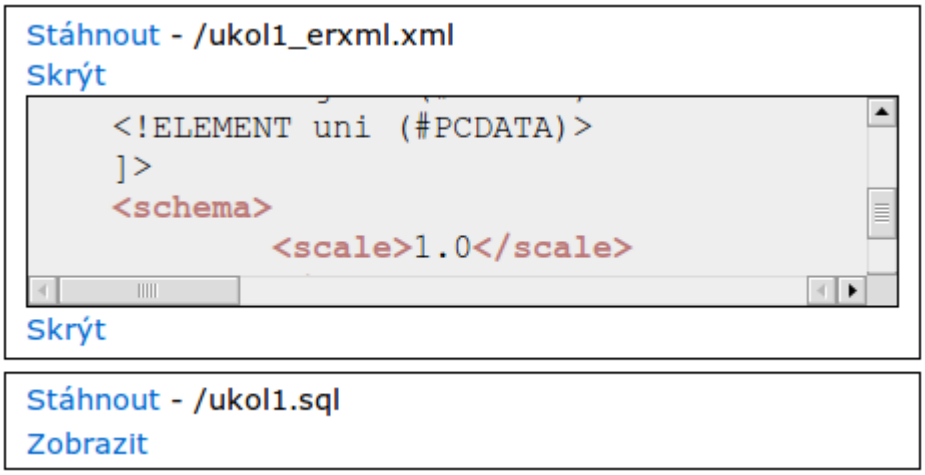
Zobrazí se detailní stránka pro kontrolu domácího úkolu. V její horní části se nachází detailní výsledek kontroly a informace o zadání, řešiteli a jeho skupině.

Ve střední části má učitel možnost obodovat úkol, připsat zdůvodnění zisku/ztráty pro řešitele, kontrolu uložit a přejít ke kontrole dalšího úkolu, pokud takový existuje (obr. 3.19).



obr. 3.19 – Formulář k vyplnění vysledku kontroly úkolu

Ve spodní části obrazovky jsou pak skrze webové rozhraní zpřístupněny jednotlivé soubory odevzdaného domácího úkolu. Jednotlivé soubory jsou nabídnuty ke stažení – pokud se navíc jedná o soubory typu **SQL**, **TXT**, **JPG**, **JPEG**, **BMP** nebo **GIF**, pak jsou nabídnuty k přímému zobrazení (obr. 3.20).



obr. 3.20 – Soubory domácího úkolu

3.4.8. Upozornění na možný plagiát

Při kontrole plagiátů je nutné kontrolovat každý úkol s každým úkolem. A právě z tohoto důvodu může být výsledek kontroly zobrazen až za nějaký čas i pro již obodované a zkontrolované úkoly. Obecně lze však říci, že jakékoliv podezření bude zobrazeno v horizontu několika málo minut po odeslání úkolu ke kontrole. Je-li zjištěno podezření na plagiát úkolu k dané definici, pak je zobrazeno učiteli studenta, který úkol odevzdal později. Ten může vzniklou situaci adekvátně řešit a v případě falešného podezření kliknutím na odkaz *„Ignorovat“* varování vyrušit. Zobrazení varování o možném plagiátu ilustruje obrázek 3.21.

Definice domácího úkolu	First Author	Plagiate Author	Poznámka	Akce
Úkol č. 1	Petr Procházka zobrazit podezřelý úkol	Petra Fialová zobrazit podezřelý úkol	.....	Ignorovat

obr. 3.21 – Zobrazení varování o možném plagiátu

3.4.9. Zobrazení detailní statistiky konkrétní skupiny

Libovolným proklikem na název skupiny, nejlépe přímo proklikem ze sekce *„Skupiny“*, může učitel zobrazit detail dané skupiny. V něm jsou zobrazeni studenti dané skupiny a jejich bodové hodnocení. Učitel má také možnost vygenerovat si CSV soubor se statistikou dané skupiny. Tento formát je vhodný pro tisk - učitel tímto způsobem může snadno archivovat studijní výsledky jednotlivých studentů v daném semestru.

3.4.10. Zobrazení detailní statistiky konkrétního studenta

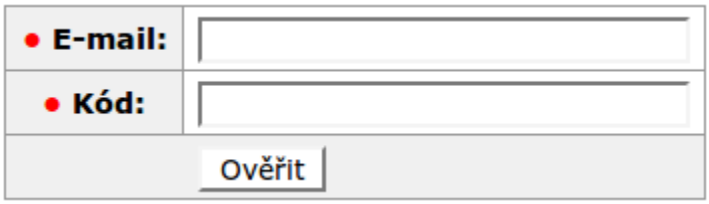
Libovolným proklikem na jméno studenta, nejlépe přímo proklikem ze sekce *„Uživatelé“* nebo z detailu skupiny, může učitel zobrazit detail daného studenta. Na této stránce je mimo jiné zobrazena i tabulka odevzdaných úkolů s přímým odkazem na jejich kontrolu. Není-li učitel vedoucím skupiny, pro kterou byl úkol odevzdán, je místo odkazu na kontrolu zobrazen pouze odkaz na detail odevzdaného úkolu.

3.5. Práce s aplikací v roli studenta

3.5.1. Registrace studenta do systému

Pro zřízení a aktivaci nového uživatelského účtu slouží nepřihlášenému uživateli sekce *„Registrace“*. V podsekci *„Nová registrace“* nalezneme registrační formulář (obr. 3.22). Zvolené přihlašovací jméno a e-mail musí být unikátní v rámci celého systému. Pro úspěšné dokončení registrace je nutné zadat platnou e-mailovou adresu. Po odeslání formuláře dojde k přesměrování do podsekce *„Potvrzení registrace“* (obr. 3.23), kde je nutné zadat registrační kód, doručený na e-mailovou adresu zadanou v registračním formuláři. Po ověření e-mailu je účet aktivován a student se může přihlásit do systému. Zadá-li uživatel třikrát špatný potvrzovací kód, musí si nechat vygenerovat a zaslat kód nový. Může tak učinit v podsekci *„Zažádat o nový potvrzující kód“* (obr. 3.24).

obr. 3.22 – Registrační formulář



obr. 3.23 – Potvrzení registrace

obr. 3.24 – Vygenerování nového potvrzujícího registračního kódu

3.5.2. Vstup do skupiny a vystoupení z ní

Po přihlášení do aplikace student může vstoupit do studijní skupiny. To lze učinit v podsekci *„Skupiny“* sekce *„Skupiny“* kliknutím na odkaz *„Vstoupit do skupiny“* (obr. 3.25). Po vstupu do skupiny bude studentovi umožněno odevzdávat domácí úkoly v ní zadané. Pokud student již členem některé skupiny je, může z ní vystoupit v téže sekci kliknutím na odkaz *„Vystoupit ze skupiny“*.

obr. 3.23 – Přihlášení do skupiny

obr. 3.24 – Odhlášení ze skupiny

3.5.3. Odevzdání domácího úkolu

Domácí úkoly student odevzdává v podsekci *„Domácí úkoly“* sekce *„Domácí úkoly“*. Zde se zobrazují zadané, nahrané, odevzdané i zkontrolované úkoly. Aktuální stav domácího úkolu student vidí v sloupci *„Stav“*. Pokud student řešení ještě nenahrál, případně nahrál a neodeslal ke kontrole učitelem, může nahrát nové řešení po kliknutí na odkaz *„Odevzdat nové řešení“* (obr. 3.25). V takovém případě bude zobrazen formulář pro nahrání nového souboru s vyřešeným úkolem. Po jeho odeslání bude student přesměrován opět do podsekce *„Domácí úkoly“*.

Nahraný a zkontrolovaný úkol si student může prohlédnout po kliknutí na datum odevzdaného řešení ve sloupci *„Odevzdané řešení“* (obr. 3.26). Zde je zobrazen detailní výsledek kontroly (a v případě již ohodnoceného úkolu i počet bodů a komentář od učitele). Zde má také student možnost stáhnout si odevzdaný úkol.

Je-li student spokojen s výsledkem automatické kontroly, má možnost domácí úkol odeslat ještě před uplynutím hraniční doby odevzdání. To může učinit kliknutím na odkaz *„Odeslat ke kontrole“* opět ve sloupci *„Odevzdané řešení“* v

podsekci *„Domácí úkoly“* (obr. 3.26). Tato akce je nevratná.

Pokud není student s výsledkem automatické kontroly spokojen, může zde úkol, pokud ještě nebyl odeslán ke kontrole učitelem, smazat.

Název použité definice	...	Stav	Akce
Úkol č. 1		Zatím neodesláno ke kontrole	Odevzdat nové řešení

obr. 3.25 – Odevzdání domácího úkolu

Odevzdané řešení
05.21.2012 15:37:29 (Odeslat ke kontrole , Smazat)

obr. 3.26 – Zobrazení detailu automatické kontroly, odeslání ke kontrole učitelem, smazání nahraného úkolu.

3.6. Práce s aplikací v roli přihlášeného uživatele s libovolnými právy

3.6.1. Změna hesla

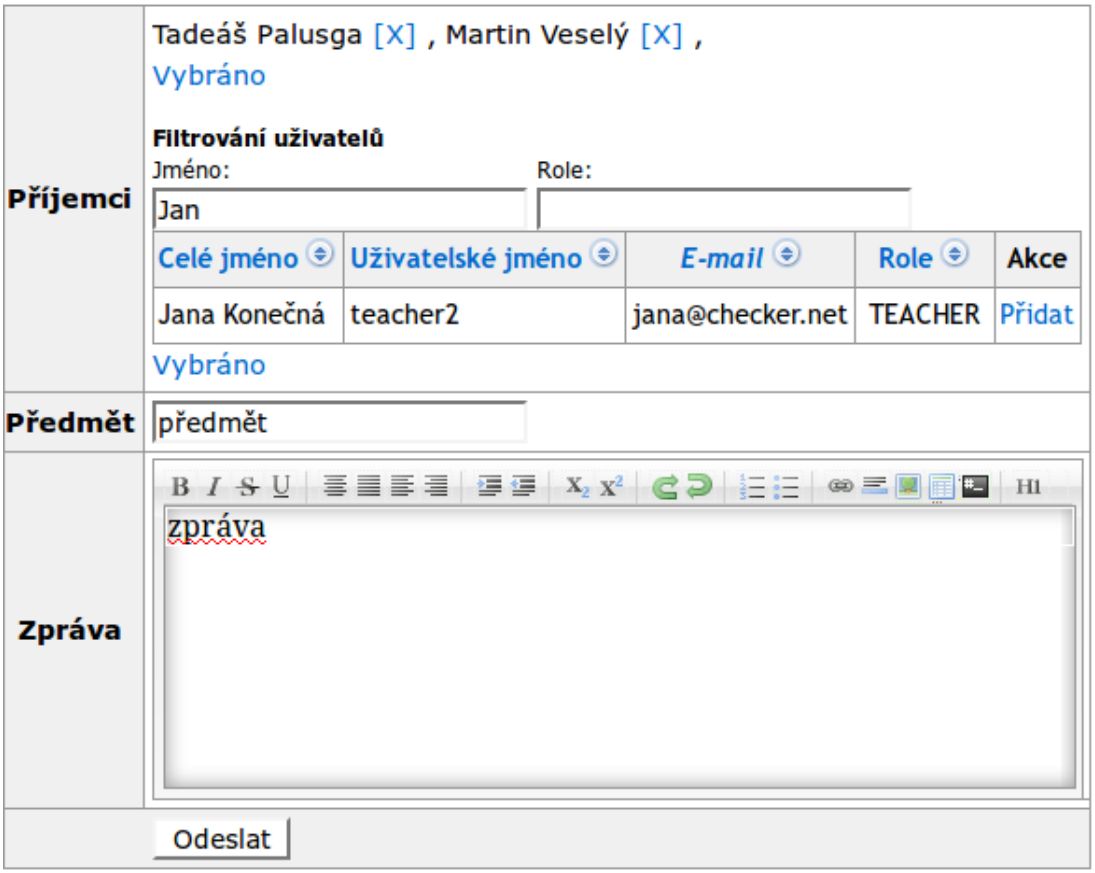
Změnu hesla lze po přihlášení provést v příslušném formuláři v podsekci *„Změna hesla“* sekce *„Nastavení účtu“*.

3.6.2. Změna e-mailové adresy

Změnu e-mailové adresy lze po přihlášení provést v příslušném formuláři v podsekci *„Změna e-mailu“* sekce *„Nastavení účtu“*.

3.6.3. Komunikace pomocí uživatelských zpráv

Uživatelům je k dispozici velice jednoduchá možnost komunikace. Po přihlášení mohou v podsekci *„Vytvořit novou“* sekce *„Zprávy“* odeslat zprávu jinému uživateli. Odeslání zprávy je ilustrováno na obrázku 3.27 .



obr. 3.27 – Vytvoření a odeslání zprávy.

Odeslané a přijaté zprávy lze zobrazit v podsekcích *„Odeslané zprávy“* a *„Přijaté zprávy“* sekce *„Zprávy“*.

4. Programátorská dokumentace

4.1. Použité technologie

V této kapitole se seznámíme se základními technologiemi, s jejichž pomocí je aplikace implementována.

4.1.1. Java

Samotná aplikace je psána v jazyce Java – konkrétně ve verzi 7. Důvodů pro výběr tohoto jazyka je hned několik:

- **Bezpečnost –** Java má vlastnosti, které v síťovém prostředí zabraňují nepřátelskému kódu napadat hostitelský operační systém.
- **Typová kontrola** – veškeré proměnné musí mít předem definovaný datový typ, což také částečně souvisí s bezpečností celé aplikace.
- Java je **objektově orientovaná** – s výjimkou základních primitivních datových typů (boolean, byte, char, double, float, short, int, long) jsou všechny datové typy definovány čistě objektově.
- Java je **nezávislá na architektuře** – aplikace napsaná v programovacím jazyku Java je spouštěna skrze JVM[3]. Díky tomu je aplikaci možné s identickým chováním spustit na libovolném systému, kde je nainstalována JVM.
- Podpora **vícevláknových aplikací –** nativně jsou podporovány aplikace vícevláknové – tedy aplikace, které umožňují souběžný běh dvou či více částí programu.

3 Java Virtual Machine – virtuální stroj, který provádí mezikód (bytecode) Javy

- **Rychlost** – nověji implementované verze JVM již dokáží interpretovat Java bytecode srovnatelně rychle s jazykem PHP.

4.1.2. Tapestry, Tapestry5-JQuery, Tapestry-security

Tapestry[4] je komponentově orientovaný open-source framework určený k vytváření robustních a vysoce škálovatelných webově založených aplikací v jazyce Java. Základním stavebním kamenem každé Tapestry aplikace jsou **komponenty**, **stránky** a poskytované **služby**, které mohou být definovanými komponentami a stránkami využívány.

Komponenty a stránky lze definovat jako klasické POJO[4] objekty s příslušně nadefinovaými HTML šablonami. Interakce mezi šablonami a objekty probíhá na základě metod a proměnných pojmenovaných dle určitých jmenných konvencí, případně s využitím anotací.

Služby jsou singleton[5] instance tříd, které poskytují podporu pro interakci mezi šablonami a jejich POJO reprezentacemi, případně interakci s jinými vrstvami aplikace. Je možné definovat vlastní služby i předefinovávat standardní. Závislosti mezi službami a jejich samotné vkládání do komponent obstarává samotný framework Tapestry skrze IoC[6] kontejner.

Základní JavaScriptový framework využívaný v Tapestry je Prototype. Tapestry5-JQuery[5] je knihovna nahrazující Prototype za JavaScriptový framework JQuery.

Tapestry framework v základu neobsahuje podporu pro autentizaci. Tapestry-security[6] tuto podporu přidává s využitím knihoven Shiro.

4 Plain Old Java Object – většinou se jedná o nejjednodušší Java objekty s bezargumentovým konstruktorem a přístupem pomocí get/set metod dodržujících jednoduché jmenné konvence.
5 singleton – návrhový vzor, kdy v programu běží pouze jedna instance dané třídy
6 Inversion of Control – návrhový vzor, kdy služby a objekty jsou svázány s volající instancí teprve za běhu, při požadavku na první použití. Tyto navázané objekty v době překladu typicky nejsou známy.

4.1.3. OSGi a Apache-Felix

OSGi[7] (Open Services Gateway initiative) je specifikace pro vytváření dynamických modulů pro platformu Java. Tato specifikace umožňuje instalaci a odinstalaci částí aplikace (modulů) za jejího běhu. OSGi moduly jsou distribuovány jako klasické JAR balíčky. Každému nainstalovanému OSGi modulu je vytvořen unikátní classloader. Modul může aplikaci či jiným modulům poskytovat služby, stejně tak mohou být aplikací služby danému modulu poskytovány. Poskytované a očekávané služby jsou definovány v MANIFEST.MF souboru daného modulu. Jako implementace této specifikace byl zvolen OSGi kontejner Apache-Felix[8]. Moduly implementované v této bakalářské práci nevyužívají žádných poskytovaných služeb, pouze služby poskytují.

4.1.4. Hibernate

Hibernate[9] je jedním z mnoha ORM[7] frameworků. Představuje abstraktní rozhraní pro práci s různými datovými zdroji – ty mohou být různorodé. Od InMemory databází přes XML databáze až po plnohodnotné SQL databáze. V implementaci této bakalářské práce slouží pro komunikaci s databází MySQL. V programu HWChecker je použita verze Tapestry-hibernate, která umožňuje snažší imlpementaci ve frameworku Tapestry..

4.1.5. MySQL

MySql[10] je multiplatformní databáze typu SQL. Mezi nesporné klady této databáze patří jak široká a stabilní komunita open-source vývojářů, tak silná podpora ze strany hlavního vývojáře, totiž firmy Oracle, která má na starosti vývoj v

7 Object Relation Mapping – technika pro konverzi dvou navzájem nekompatibilních typových systémů.

komerční sféře velmi rozšířené SQL databáze Oracle Database[11]. Tato databáze v současné době podporuje transakční zpracování a umožňuje definovat vlastní triggery, procedury a funkce.

4.1.6. Apache Maven

Apache Maven[12] je nástroj pro automatickou správu závislostí projektu na externích knihovnách, spouštění automatizovaných zdrojových testů, výslednou kompilaci a buildování. Definice každé Maven aplikace se nachází v souboru *„pom.xml“* v kořenovém adresáři daného projektu. V tomto souboru definujeme název a verzi aplikace, použité knihovny a jejich verze, buildovací pluginy, typ výsledného package (JAR,WAR...).

4.2. Struktura aplikace

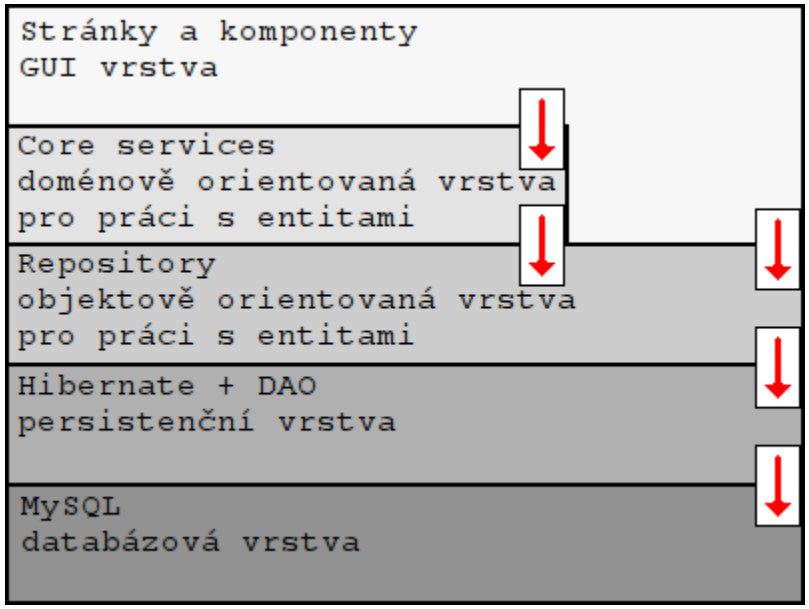
- Nejspodnější vrstvou aplikace je **MySQL**. Tato databázová vrstva slouží jako trvalé úložiště mezi dvěma starty aplikace. Veškeré akce typu *insert*, *update*, *delete* jsou hlídané cizími klíči. Tím je zaručeno, že databáze by se neměla nikdy vyskytovat v nekonzistentním stavu.
- O úroveň výše je použita **Hibernate** vrstva. Tato vrstva je implementována pomocí Hibernate DAO[8] objektů. Díky ní je možné pracovat s databázovými entitami na úrovni objektů. DAO objekty poskytují nejzákladnější rozhraní pro CRUD[9] operace na entitách.
- Další úroveň tvoří **repository**. Tato vrstva rozšiřuje předchozí vrstvu a přidává další neatomické operace usnadňující práci vyšším vrstvám. Jedná se o první viditelnou vrstvu z úrovně programátora stránek, core služeb a

8 Data Access Object – objekt, který poskytuje rozhraní pro manipulaci s konkrétním úložištěm dat.
9 Create-Read-Update-Delete – nejzákladnější operace pro manipulaci s entitami.

komponent.

- Vrstva o stupeň výše je tvořena **core službami**. Tato úroveň abstrahuje operace na entitách tak, aby odpovídaly danému doménovému modelu[10]. Zde již metody mohou využívat zrovna tak služeb z úrovně *repository*, jako i služeb nesouvisejících s databází.
- **Prezentační** vrstvu tvoří samotné komponenty a stránky. Každá stránka i komponenta může obsahovat libovolný počet vnořených komponent a využívat jakékoliv služby.

Jednotlivé vrstvy jsou ilustrovány obrázkem 4.1



obr. 4.1 – Jednotlivé vrstvy aplikace

4.3. Databázová vrstva

Jak již bylo řečeno, aplikace využívá služeb databáze MySQL. Veškeré definované tabulky používají InnoDB[11] úložný engine. Připojení do databáze je

10 Domain model – úhel pohledu, kdy na entity nahlížíme jako na objekty z reálného světa.
11 InnoDB byl navržen pro zpracování mnoha krátkodobých transakcí, které se málokdy anulují.

definováno jako datový zdroj přímo v kontejneru, z něhož je aplikace spouštěna. Schéma databáze je ilustrováno v příloze [B] této práce.

4.4. DAO a Hibernate

DAO objekty/služby jsou určeny pouze k základním CRUD operacím. Tyto objekty pracují s Hibernate session. DAO objekty jsou umístěny v package „**cz.donarus.checker.web.services.db.dao“**. CRUD operace poskytované DAO objekty jsou následující:

- **int count()** - vrátí z databáze počet všech entit daného typu
- **int count(DetachedCriteria criteria)** - vrátí z databáze počet všech entit daného typu, které vyhovují zadaným kritériím
- **void delete(DetachedCriteria criteria) –** smaže z databáze entity daného typu, které vyhovují zadaným kritériím
- **void delete(List<T> entities)** – smaže z databáze konkrétní entity
- **boolean exists(DetachedCriteria criteria)** – zjistí, zda v databázi existuje entita, která vyhovuje zadaným kritériím
- **T find(DetachedCriteria criteria)** – vyhledá v databázi entitu daného typu, která vyhovuje zadaným kritériím
- **List<T> list()** – vrátí z databáze všechny entity daného typu
- **List<T> list(DetachedCriteria criteria)** – vrátí z databáze všechny entity, které vyhovují zadaným kritériím
- **List<T> list(DetachedCriteria criteria, int from, int count)** – vrátí z databáze entity, které vyhovují zadaným kritériím a jsou v rozmezí *„from“* - *„from + count“*

- **void persist(List<T> entities)** – uloží nové/změněné entity daného typu do databáze
- **void persist(T entity)** – uloží entitu daného typu do databáze
- **T refresh(T entity)** – vrátí identickou entitu spojenou s hibernate session (odpovídající entitu v databázi dohledává podle **id** entity z parametru)

Metody **persist** a **delete** jsou anotovány jako **@CommitAfter**, což znamená, že libovolné volání těchto metod je uzavřeno v transakci. Konfigurace (mapování) pro hibernate je řešena pomocí anotací na příslušných entitách. Entity lze nalézt v package **„cz.donarus.checker.web.entity“**.

4.5. Repository

Repository jsou služby, které využívají a obalují DAO služby. Slouží pro manipulaci s objekty bez ohledu na doménový model. Jsou umístěny v package „**cz.donarus.checker.web.services.db.repository“.** Většina metod pouze opětovně volá příslušné DAO metody s identickou hlavičkou:

- **void persist(T entity)**
- **void persist(List<T> entities)**
- **void delete(T entity)**
- **void delete(DetachedCriteria criteria)**
- **int count()**
- **int count(DetachedCriteria criteria)**
- **boolean exists(DetachedCriteria criteria)**
- **T find(DetachedCriteria criteria)**
- **List<T> list()**
- **List<T> list(DetachedCriteria criteria)**

- **List<T> list(DetachedCriteria criteria, int from, int count)**
- **T refresh(T entity)**

Některé další metody však přibyly nově:

- **T findById(long id)** – vyhledá v databázi entity daného typu s příslušným identifikátorem
- **List<T> listByProperty(String property, Object value)** – vrátí z databáze entity daného typu dle parametru a požadované hodnoty
- **T findByProperty(String property, Object value)** – vrátí z databáze první nalezenou entitu daného typu dle parametru a požadované hodnoty

Konkrétní repository pro konkrétní entity implementují další podpůrné metody.

4.6. Core služby

Core služby implementují metody, které nejsou atomické. Core služby jsou umístěny v package „**cz.donarus.checker.web.services.db.repository“**. Tyto služby mají nejednotné rozhraní - to se u každé služby odvíjí speciálně od doménového modelu příslušné entity. Základní služby jsou:

- **AuthenticationService** – služba pro autentikaci uživatelů. Obsahuje metody pro přihlašování a odhlašování uživatelů a kontrolu jejich práv. Využívá zejména služby *SecurityService* poskytované knihovnou Tapestry-security.
- **GroupService** – obstarává vytváření, mazání, updatování, vyhledávání skupin a generování CSV reportu
- **HomeworkDefinitionService –** zajišťuje vytváření, mazání a updatování definic domácích úkolů a jejich nastavení

- **HomeworkInstanceService –** poskytuje metody pro vytváření a mazání instancí domácích úkolů
- **HomeworkUploadedService –** obsahuje metody pro odevzdávání, prohlížení a mazání domácích úkolů
- **MessageService –** poskytuje podporu pro odesílání zpráv
- **ModuleService –** zajišťuje instalaci, odinstalaci, aktivaci, deaktivaci a správu modulů
- **PlagiatesCheckService –** kontroluje plagiaritu
- **UserService –** služba definuje metody pro registraci studentů, vytváření učitelů, mazání uživatelů a přihlašování/odhlašování studentů do/ze skupin. Dále definuje metody pro změnu hesla, e-mailu, potvrzení a zopakování registrace.

4.7. Validátory

Validátory jsou služby, které kontrolují entity vkládané do databáze. Lze je nalézt v package **„cz.donarus.checker.web.services.validators“**. Implementovaných je pět následujících validátorů:

- **GroupValidator** – kontroluje minimální a maximální délku názvu na kontrolované skupině
- **HomeworkDefinitionValidator** – kontroluje minimální a maximální délku názvu na kontrolované definici
- **HomeworkInstanceValidator** – kontroluje, zda nechybí datum nejzazšího odevzdání ani datum zadání a zda datum odevzdání není dřívější než datum zadání. Dále kontroluje, zda je vyplněn autor a skupina pro danou instanci.

- **MessageValidator** – kontroluje, zda je vyplněn odesilatel, příjemce, předmět a text
- **UserValidator** – kontroluje délku jména, příjmení a uživatelského jména. Dále kontroluje shodu hesel, validitu e-mailu, znaky obsažené ve jménu, příjmení a v uživatelském jménu

4.8. Další důležité služby

- **cz.donarus.checker.web.services.mail.Mailer** – slouží k odesílání e-mailových zpráv
- **cz.donarus.checker.web.services.configuration.Configuration** – slouží k načítání konfigurace z konfiguračního souboru, kde jednotlivé konfigurační hodnoty jsou reprezentovány trojicí „**sekce.klíč=hodnota“** na samostatných řádcích. Prázdné řádky a řádky zažínající znakem **„#“** jsou ignorovány.

4.9. Kontrolní moduly

4.9.1. Vlastnosti kontrolních modulů

Kontrolní moduly jsou samostatné části aplikace definující předem dané rozhraní a slouží k poloautomatické kontrole odevzdaných domácích úkolů. Kontrolní moduly by měly být koncipovány jednoúčelově. Pokud má modul na starosti například kontrolu, zda je v domácím úkolu validní XML dokument, již by neměl mít na starosti kontrolu, zda se v druhém souboru se vyskytuje spustitelný SQL script (pokud ovšem tyto dvě kontroly jsou na sobě závislé, pak je definice obou kontrol v jednom modulu v pořádku). O vlastnostech modulu více napoví

definice jeho rozhraní.

4.9.2. Interface kontrolních modulů

Každý kontrolní modul musí rozšiřovat základní abstraktní třídu **„cz.donarus.checker.module.CheckerModule“**. Tato třída je extrahována do samostatné knihovny **„module“** (v zkompilované podobě k nalezení na disku v příloze A v adresáři *„bin“* pod názvem *„module.jar“*). Knihovna je zahrnuta v classpath spuštěné webové části aplikace.

Metody třídy **cz.donarus.checker.module.CheckerModule** jsou následující:

- **abstract String getHtmlDescription()** - Řetězec vracený touto metodou by měl obsahovat základní popis funkčnosti modulu, popis parametrů, které daný modul očekává, způsob vyhodnocování podobnosti, případně zmínku o autorovi a popis možných problémů. Výstup z této metody může obsahovat HTML formátování, je však dobré omezit se pouze na základní tagy *BR, HR, B, I, U, STRONG, FONT, H1, H2, H3, P*. V GUI části aplikace bude tato hodnota zobrazena v popisu modulu.
- **abstract List<ParameterDescription> getAvailableParameters()** - Metoda vrací seznam všech parametrů. V každém parametru je definován jeho název a datový typ. Obsahuje rovněž definici, zda je povinný či nikoliv. Třída *ParameterDescription* je popsána dále v této kapitole.
- **Result validateParameters(Map<String, Object> parameters)** – Tato metoda dostane na vstupu mapu, kde klíčem je název parametru a hodnotou je hodnota daného parametru. Metoda slouží ke kontrole, zda zadané typizované parametry jsou validní. Metoda je již v abstraktní třídě *CheckerModule* implementována – kontroluje, zda jsou přítomny všechny

povinné parametry, zda jsou správného typu a dále volá metodu *userDefValudate(..)* pro uživatelsky definovanou kontrolu nastavení. Implementace této metody však může být v modulu libovolně předefinována. V případě nalezených chyb jsou odpovídající chybové hlášky vyplněny do objektu typu *Result* (popsán dále v této kapitole), který je vracen jako výstup z metody. V případě předefinování není doporučeno vracet *null* hodnotu.

- **abstract void userDefValidate(Map<String, Object> parameters, Result result) –** Uživatelsky definovaná kontrola parametrů. Pokud je to nutné, lze v této metodě nadefnovat dodatečnou kontrolu parametrů - například kontrolu rozmezí hodnot typu *Integer*, kontrolu hodnoty typu *String* vůči regulárnímu výrazu nebo kolize hodnot různých parametrů. Případné chybové hlášky je potřeba nahrát do dodaného objektu *result*.
- **abstract Result run(Map<String, Object> parameters, File homeworkDir) throws Exception –** Tato metoda slouží ke kontrole domácího úkolu, který je rozbalen v adresáři *homeworkDir*. Metoda je z aplikace volána pouze tehdy, projde-li validací pomocí metody *validateParameters(..)*. Lze tedy předpokládat, že vstupní parametry jsou korektní. V těle metody by měly proběhnout veškeré požadované kontroly a případné informativní, varující a chybové hlášky musí být vráceny pomocí objektu typu *Result*. Je dovoleno v případě velmi závažných chyb vyhodit výjimku, výrazně to však **nedoporučujeme**.
- **abstract Result checkSimilarity(Map<String, Object> parameters, File hwSourceDir1, File hwSourceDir2)** – Zkontroluje podobnost dvou domácích úkolů v zadaných adresářích.

Objekt typu **cz.donarus.checker.module.ParameterDescription** je příkladem jednoduchého POJO objektu:

- **ParameterDescription(String name, Type type, boolean req)** – Vytvářecí konstruktor, ve kterém nadefinujeme jméno parametru, jeho typ a určíme, zda je povinný. **cz.donarus.checker.module.ParameterDescription.Type** je enumerační typ s definovanými hodnotami **STRING**, **INTEGER**, **BOOLEAN**.
- **String getName(), Type getType(), boolean isRequired()** - Gettery pro hodnoty nadefinované konstruktorem.

Chybové hlášky vzniklé při validaci parametrů, kontrole domácího úkolu nebo podobnosti dvou domácích úkolů jsou nahrávány a vraceny v objektu typu **cz.donarus.checker.module.Result**. Jedná se o jednoduchý objekt, který má následující metody:

- **void record(Message message)** – přidá k výsledku novou zprávu typu Message (typ popsán níže)
- **List<Message> getMessages()** - vrátí seznam všech nahraných zpráv
- **boolean hasErrors()** - kontroluje, zda byla do objektu nahrána alespoň jedna chybová zpráva
- **void printHtmlReport(PrintWriter printWriter)** – vytiskne do vstupního printWriteru HTML naformátovaný výpis chybových hlášek typu INFO, ERROR, WARN. Zprávy typu EXCEP nejsou zahrnuty.
- **void printExtendedHtmlReport(PrintWriter printWriter)** – vytiskne do vstupního printWriteru HTML naformátovaný výpis chybových hlášek všech typů
- **String toString()** - vrátí řetězec s naformátovaným HTML výpisem

chybových hlášek bez zpráv typu EXCEP

Třída **Result** implementuje rozhraní **Iterable<Result.Message>** a skrze jednotlivé zprávy lze tedy procházet cyklem.

Objekty typu **cz.donarus.checker.info.Result.Message** reprezentují chybové zprávy. Třída má privátní konstruktor a jednotlivé zprávy mohou být vytvořeny voláním statických metod **info(..)**, **warn(..)**, **error(..)**, **excep(..)**.

4.9.3. Vytvoření balíčku modulu

Jednotlivé moduly jsou distribuovány v balíčcích dodržujících specifikaci OSGi. V této části se nebudeme zabývat plnou specifikací OSGi, tu lze nalézt v uživatelské dokumentaci [7]. Zde vysvětlíme základní požadavky pro definici nového balíčku.

OSGi bundle/balíček je obyčejný JAR archiv s několika speciálními specifickými hlavičkami v souboru **MANIFEST.MF**.

Důležité hlavičky v **MANIFEST.MF**:

- **Bundle-ManifestVersion** – Verze hlaviček manifestu, aplikace HWChecker očekává moduly s verzí **2**.
- **Bundle-Activator** – Třída, která implementuje rozhraní **org.osgi.framework.BundleActivator**. Tato třída inicializuje modul a registruje exportované služby.
- **Bundle-Version** – Verze balíčku skládající se z částí **major** version, **minor** version, **micro** version. (Například: **1.2.4**. Je tedy možné v aplikaci použít více stejných modulů různých verzí.)
- **Bundle-Name** - Jméno jednoznačně identifikující modul. Doporučujeme používat standardizovaný tvar *groupId:artifactId*

- **Export-Package –** V této direktivě uvádíme název *package*, v němž se nachází služby, které chceme zpřístupnit, rozsah použitelných verzí a package tříd v těchto službách používaných.
- **Import-Package** – V této direktivě explicitně uvádíme názvy *package*, které chceme modulu zpřístupnit z rodičovského kontejneru a rozsah jejich verzí.

Každému modulu je při instalaci vytvořen vlastní classloader, v němž jsou viditelné pouze zdroje, třídy a knihovny definované v něm samotném. Dále jsou v modulu viditelné zdroje z těch package, které jsou explicitně nadefinované v souboru **„MANIFEST.MF“** direktivou **„Import-Package“**.

Protože ruční vytváření OSGi balíčků by bylo velmi zdlouhavé a museli bychom sami ohlídat závislosti mezi balíčky, využíváme plugin Maven-Compiler-Plugin pro systém Maven. Díky tomu nám odpadne hlídání závislostí, vytváření MANIFEST.MF souboru a značně se nám zjednoduší kompilace do výsledného JAR balíčku modulu. Nic ovšem nebrání ve využití jiných nástrojů, které dodržují OSGi specifikaci. Detailní postup vytváření OSGi balíčků pomocí pluginu **Maven Bundle Plugin**[13] je popsán v jeho dokumentaci.

4.9.4. Služba pro instalaci a odinstalaci kontrolních modulů

Pro instalaci a odinstalaci kontrolních modulů je implementována služba **„cz.donarus.checker.web.services.osgi.CheckerModuleOSGiLoader“**. Její rozhraní je následující:

- **void loadModulesInDir(File dir)** – Metoda nainstaluje všechny OSGi balíčky modulů v daném adresáři. Metoda je volána po startu aplikace.
- **String loadModule(File file) throws Error** – Nainstaluje OSGi balíček modulu z daného souboru.

- **void unload(String moduleName)** – Odinstaluje modul s daným jménem.
- **List<String> getLoadedModuleNames()** – Vrátí seznam všech nainstalovaných modulů.
- **CheckerModule getLoadedModule(String name)** – Vrátí nainstalovaný modul dle jeho jména.
- **boolean isInitialized()** – Indikuje, zda již byla volána metoda l*oadModulesFromDir(..)*.

4.9.5. Implementované moduly

Jako součást aplikace jsou dodány dva plně implementované moduly, určené pro kontrolu domácích úkolů z předmětu *„Databázové systémy“*.

První modul se nazývá **ModuleERXML** a kontroluje výstupní XML soubor z programu ER Modelář [14]. Zkompilovanou verzi se jménem „module.erxml-2.0.2.jar“ lze nalézt v adresáři „bin“ na disku z přílohy A . Modul ověřuje, že model obsahuje alespoň pět entit, pět relací, pět atributů, jednu relaci typu 0:N, jednu relaci typu 1:N, jednu relaci typu 1:1, jednu ISA relaci, jednu rekurzivní relaci a jednu ternární relaci. Havní kontrola podobnosti je implementována s použitím knihovny XMLunit [15] a kontroluje podobnost koster XML souborů. Dále ověřuje výskyt identických slov v názvech entit. Je-li nalezeno dva a více identických názvů (bez ohledu na velikost písmen), jsou dokumenty vyhodnoceny jako podezřelé. Modul používá jediný parametr **pathToERXML** – cestu k souboru s XML výstupem z programu ER modelář.

Druhý modul se nazývá **ModuleNeedFile**. Kontroluje přítomnost souboru s názvem definovaným v parametru **pathToFile**. Pokud je zadána hodnota **true** pro parametr **checkDuplicity**, je podobnost souborů kontrolována vyhledáním nejdelších společných podřetězců o délce větší než 10 (prázdné znaky jsou předem

odstraněny a velikost písmen je ignorována). Pokud jsou nalezeny alespoň dva takovéto podřetězce, je úkol vyhodnocen jako podezřelý. Zkompilovaný modul lze nalézt na disku z přílohy [A] v adresáři „bin“ pod názvem „module.needfile-1.0.0.jar“.

4.9.6. Neimplementované moduly

Původně jsme uvažovali o implementaci třetího kontrolního modulu. Ten by vyhodnocoval SQL dotazy oproti reálné databázi Oracle. Výstup by ukládal do adresáře s domácím úkolem. Po zkontrolování úkolu modulem by byla databáze uvedena do předchozího stavu. Bohužel, účet s právy vytváření a mazání schémat nám nebyl k dispozici a bez takových práv by bylo nutné se při čištění databáze spoléhat na programatická řešení – ta existují, ovšem jsou značně nespolehlivá a ve všech případech vyžadují kontrolu a případné dočištění schématu uživatelem. Pokud bychom přeci jen chtěli účet s plnými právy, bylo by nutné vytvořit a spravovat vlastní Oracle databázový server. Tento návrh jsme nakonec zavrhli s tím, že řešení bylo nastíněno, avšak implementace by byla nad rámec této práce.

4.9.7. Původní implementace načítání modulů

Stávající implementace s využitím specifikace OSGi nebyla první funkční implementací. OSGi technologie byla původně zamítnuta z důvodu přílišné složitosti. V první funkční verzi byly moduly definovány jako nezkompilované JAVA soubory definující předem daný interface. Vývoj nových modulů tím byl velmi usnadněn. Bylo ovšem možno využívat pouze knihoven, které byly nahrány classloaderem front-endové části aplikace, což znamenalo značné omezení. V průběhu času byla implementace upravena. Modul byl implementován jako jednoduchý JAR balíček. V tomto balíčku musela být přítomna třída dědící od třídy

CheckerModule. V případě, že aplikace knihovnu nepoužívá, bylo možné vytvořit pomocí Maven One-Jar pluginu JAR balíček, který knihovnu obsahoval. Vytváření One-Jar balíčku však bylo velmi složité. Na straně front-endové části aplikace byly implementovány třídy pro rozbalování JAR balíčku a jejich classloading a unloading pomocí vytvořeného child-first classloaderu (defaultní chování všech classloaderů implementovaných v Javě je parent-first). Právě implementace child-first classloaderu přinášela velké problémy s pořadím načítaných souborů, s duplicitními verzemi knihoven apod. V té chvíli jsme se dostali do bodu, kdy naše implementace dosáhla větší programatické složitosti než jednoduché použití libovolného z dostupných open-source implementovaných kontejnerů OSGi. Takové implementace jsou ale na rozdíl od naší původní stabilní, lety ověřené a mnoha vývojáři otestované. Tím bylo rozhodnuto o reimplementaci modulů pomocí OSGi specifikace. Jako OSGi kontejner byl zvolen Apache Felix[8].

4.10. Stránky a komponenty

Stránky a komponenty dohromady tvoří GUI vrstvu aplikace. Stránky lze nalézt v package **„cz.donarus.checker.web2.pages“**, komponenty v **„cz.donarus.checker.web2.components“**.

Každou stránku a komponentu definují dvěma až třemi soubory se stejným názvem, avšak odlišnou příponou. Prvním z nich je soubor s příponou *tml*. *Tml* soubor lze jednoduše popsat jako HTML šablonu s některými rozšířeními. Druhým souborem je soubor *java* obsahující objektovou reprezentaci komponenty/stránky. V tomto souboru je dle jednoduchých jmenných konvencí (nebo pomocí anotací) možno definovat proměnné a metody, které mohou být volány z *tml* šablony při generování výsledného HTML kódu pro komponentu. Třetím (nepovinným) je soubor s příponou *properties*. Tento soubor může obsahovat texty, které jsou odkazovány z šablony komponenty/stránky. Detailní popis konvencí pro tvorbu

stránek, komponent a jejich životního cyklu lze nalézt v dokumentaci k Tapestry[4], případně v dokumentaci k Tpestry-JQuery[5].

4.11. Inicializace služeb pro použití v komponentách a stránkách

O správu definovaných komponent a stránek se stará framework Tapestry. Jejich inicializace tudíž probíhá voláním bezparametrického konstruktoru. Stránky i komponenty mohou využívat služeb, které je nutné v aplikaci definovat jako singleton instance. To můžeme (mimo jiné) provádět ve třídě **„cz.donarus.checker.services.AppModule“** a ve všech dalších, které v této třídě nadefinujeme pomocí anotace **@SubModule**. Takové třídy jsou:

- **cz.donarus.checker.web2.services.configuration.ConfigurationModule**
- **cz.donarus.checker.web2.services.core.CoreModule**
- **cz.donarus.checker.web2.services.db.DBModule**
- **cz.donarus.checker.web2.services.mail.MailModule**
- **cz.donarus.checker.web2.services.security.SecurityModule**
- **cz.donarus.checker.web2.services.validators.ValidatorsModule**
- **cz.donarus.checker.web2.services.EditBlocksModule**
- **cz.donarus.checker.web2.services.StartupModule**

O inicializaci služeb se stará **Tapestry IoC** (Inversion of Control). Tím je zaručeno jejich vytvoření a provázání dle potřeby. Pokud není definováno jinak, jsou veškeré služby definovány pro líné vytváření (lazy loading) a jsou instanciovány až

v případě potřeby. To IoC řeší tak, že namísto služeb zprvu vytvoří kontejnery, které tyto služby obalují a ty pak předává namísto přímých referencí buď jiným službám pomocí konstruktorů nebo stránkám a komponentám pomocí **@Inject** anotací nad globální proměnnou daného typu.

4.12. Načasované úlohy

Pro vytvoření časovačů pro pravidelně spouštěné úlohy je nadefinovaný Tapestry modul **cz.donarus.checker.web2.services.StartupModule**. V něm je implementována metoda **runOrScheduleOnStartup(..)**, jejíž parametry mohou být libovolné služby, ať již vlastní, či předdefinované samotným frameworkem. Metoda je anotována **@Startup** anotací – to znamená, že se spustí v okamžiku vytvoření a zaregistrování všech nadefinovaných služeb.

4.12.1. Automatické odeslání úkolu ke kontrole

Tato načasovaná úloha odešle každý načasovaný úkol ke kontrole, neučinil-li tak student a čas pro odevzdání úkolu již vypršel. Je načasována pro spouštění každých patnáct vteřin.

4.12.2. Kontrola plagiátů

Tato úloha zajišťuje každých pět minut volání metody **checkForPlagiates()** služby **cz.donarus.checker.web.services.core.PlagiatesCheckerService**. Ta pro všechny odevzdané úkoly se stejnou definicí spustí kontrolu typu **„každý s každým“**. Kontrola dvou odevzdaných řešení je přeskočena, proběhla-li již v minulosti.

Závěr

Výsledkem práce je implementovaný program, který může usnadnit práci studentům i učitelům. Aplikace poskytuje intuitivní uživatelské rozhraní, které je snadno pochopitelné bez delšího studia uživatelské dokumentace. Součástí práce jsou také dva implementované moduly – ModuleERXML a ModuleNeedFile. Aplikace je dokončena a připravena k nasazení pro veřejné testování. V době psaní tohoto textu je k dispozici testovací verze na adrese http://dbcheck.projekty.ms.mff.cuni.cz.

Vždy existují možnosti, jak by se daná aplikace dala vylepšit. V našem případě uvedu pár příkladů, které by stály za úvahu:

- Možnost pro učitele u každé skupiny nahrávat soubory se studijními materiály.
- Správa docházky a výsledků zkoušek daných studentů.
- Povolení přístupu modulů k některým Repository nebo Core Services.
- Větší konfigurovatelnost aplikace.
- Další kontrolní pluginy.

Referencované zdroje

[1] Konopásek J., *Systém pro správu úkolů a testů*, 2011, http://hon2a.wz.cz/asm

[2] Mareš M., Gavenčiak T., Kruliš M., Svoboda J., Turek L., *The Code Examiner – referenční dokumentace*, http://codex2.ms.mff.cuni.cz

[3] John Zaitseff, *Sinorcaish CSS Stylesheet*, 2007, http://www.zap.org.au

[4] kolektiv vývojářů, *Tapestry 5 – uživatelská dokumentace*, http://tapestry.apache.org

[5] kolektiv vývojářů, *Tapestry5-JQuery – uživatelská dokumentace*, http://tapestry5-jquery.com

[6] Korhonen K., Scandroli A., *Tynamo Tapestry-security – uživatelská dokumentace*, http://tapestry.apache.org

[7] kolektiv vývojářů, *OSGi specifikace*, http://www.osgi.org

[8] kolektiv vývojářů, *Apache Felix – dokumentace*, http://felix.apache.org

[9] JBoss community, *Hibernate – dokumentace*, http://www.hibernate.org

[10] Oracle, kolektiv vývojářů, *MySQL – dokumentace*, http://www.mysql.com

[11] Oracle, *Oracle Database 11g – dokumentace*, http://www.oracle.com

[12] The Apache Software Foundation, *Maven – uživatelská dokumentace*, http://maven.apache.org/

[13] kolektiv vývojářů, *Maven Bundle Plugin – dokumentace*, http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

[14] Petr Liška, ČVUT FEL, *ER Modelář – nápověda a videotutoriály k aplikaci*, http://service.felk.cvut.cz/courses/X36DBS/additions/web_ERM/index.html

[15] Tim Bacon, Jeff Martin, *XMLUnit – dokumentace, javadoc dokumentace*, http://xmlunit.sourceforge.net/

Příloha A – obsah přiloženého CD

bin/	…	zkompilované projekty
web2.war	…	front-end aplikace
module-1.4.jar	…	extrahovaný interface pro moduly
module.needfile-1.0.0.jar	…	modul pro kontrolu existence souborů
module.erxml-2.0.2.jar	…	modul pro kontrolu souborů z aplikace ER Modelář
src/	…	zdrojové kódy
web	…	front-end aplikace
module	…	extrahovaný interface pro moduly
module-erxml	…	modul pro kontrolu souborů z aplikace ER Modelář
module-needfile	…	modul pro kontrolu existence souborů
bp.pdf	…	text této bakalářské práce

Příloha B – struktura databáze