České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **David Polák**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Měření pedagogického výkonu**

Pokyny pro vypracování:

Na základě existujícího prototypu vytvořte pro potřeby vedení ČVUT FEL aplikaci, která bude umožňovat sledovat:

1. pedagogický výkon jednotlivých vyučujících,
2. pedagogický výkon zkonzumovaný studenty jednotlivých studijních programů.

Vývoj provádějte iterativním způsobem. V každé iteraci proveďte sběr požadavků, analýzu, návrh, implementaci, testování a nasazení.

Seznam odborné literatury:

[1] LARMAN, Craig a Chris RUPP. Applying UML and patterns: introduction to object-oriented analysis and design and iteractive development. 3rd ed. New Jersey: Prentice-Hall, 2005, xviii, ISBN 01-314-8906-2.
[2] FOWLER, Martin. Destilované UML. Praha: Grada, 2009, 173 s. ISBN 978-80-247-2062-3.

Vedoucí: Ing. Martin Komárek

Platnost zadání: do konce letního semestru 2014/2015

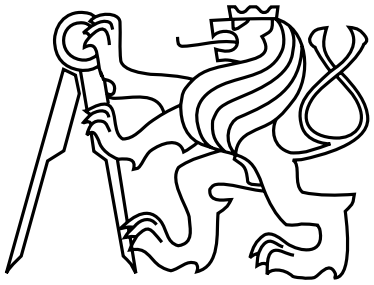
doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 28. 2. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Měření pedagogického výkonu

David Polák

Vedoucí práce: Ing. Martin Komárek

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

21. května 2014

Poděkování

Rád bych poděkoval Ing. Martinovi Komárkovi za skvělé vedení a cenné rady, bez kterých by tato práce nemohla vzniknout. Děkuji také rodině a přátelům za podporu při studiu a psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. 5. 2014 .....

Abstract

This thesis explores the issue of measuring pedagogic performance. The outcome is a deployed application written in Java. The application collects data from KOS, by means of several data exports and from a restful service KOSapi. The data is used for calculation of pedagogic performance for individuals and also for calculating teaching effectiveness.

Abstrakt

Tato bakalářská práce se zabývá problematikou měření pedagogického výkonu. Výsledkem je nasazená funkční aplikace napsaná v jazyce Java, která sbírá data z externí služby KOSapi a z několika datových exportů systému KOS. S těmito daty pak provádí výpočty pedagogického výkonu jednotlivců a efektivity výuky.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem této práce je vytvořit aplikaci, která vyhoví požadavkům vedení ČVUT FEL na sledování pedagogického výkonu jednotlivců. Aplikace staví na metodice Kometa[10], která definuje pedagogický výkon katedry, konkrétně obsahuje kvantitativní výpočet, který pedagogický výkon dané katedry vyjadřuje. Při zachování smyslu metodiky rozšiřujeme definici výpočtu pedagogického výkonu, aby ho bylo možno použít pro výpočet výkonu jednotlivých vyučujících, nejen pro katedry jakožto celky. Aplikace navíc umožňuje sledovat pedagogický výkon zkonzumovaný studenty jednotlivých studijních programů.

Historie vývoje projektu

Aplikace Hodnocení je pokračováním nedokončeného prototypu aplikace, na níž jsem se podílel v létě 2013 spolu s Michalem Řežábkem pod vedením Ing. Martina Komárka. Vývoj proběhl na žádost děkana FEL ČVUT. Tento prototyp je popsán v kapitole 2. Prototyp měl sloužit vedoucím kateder k osobnímu hodnocení výkonu jednotlivých vyučujících ze zdrojových dat přístupných pomocí restových služeb KOSapi[12] a VVVSapi[21].

Pro vedení ČVUT FEL nastala potřeba sledovat pedagogický výkon jednotlivců a tak bylo zadání aplikace uprostřed zimního semestru 2013 změněno a aplikace byla přepracována, aby vyhověla novým požadavkům. Jelikož KOSapi neposkytuje citlivá data, která potřebujeme, přistoupili jsme k jejich získávání na metodu jednorázových exportů ze systému KOS. Na konci zimního semestru dále přišel požadavek na zprovoznění výpočtu zkonzumovaných hodin studenty jednotlivých studijních programů.

Aplikace je schopna vypočítat a exportovat započitatelné hodiny pro libovolného vyučujícího z FEL. Výpočet probíhá zvlášť pro přednášky, cvičení, bakalářské a diplomové práce, zkoušení a zápočty z projektových předmětů. V druhé části aplikace jsou rozpočítávány započitatelné hodiny mezi jednotlivé studenty v seskupení jednotlivých studijních programů. Pro přihlašování je použit centralizovaný systém FELid[4], který stojí na technologii Shibboleth. Do budoucna se budu nadále zúčastňovat vývoje aplikace, na jejíž pokračování byl udělen nový grant RPMT 2014 Měření pedagogického výkonu[7].

Kapitola 2

Problematika: osobní hodnocení

Tato kapitola popisuje historickou část vývoje aplikace. Funkcionalita se již v aplikaci nenachází, ovšem kostra architektury a získané znalosti z vývoje řešení této problematiky byly hojně použity při následujícím vývoji.

2.1 Popis problému

Hodnocení výkonu pracovníků, většinou probíhá na základě osobních pohovorů a zhodnocení množiny dat, která jsou směrodatná pro daný typ práce. V době psaní (léto 2013) na ČVUT neexistovala žádná jednotná a především kvantitativní metrika, dle které by se dal výkon zhodnotit. Aplikace měla za úkol ulehčit hodnocení výkonu předkládáním a agregací kvantifikovatelných dat.

2.2 Zadání

Vytvořte aplikaci, která bude měřit osobní výkon jednotlivých pracovníků s využitím restových služeb KOSapi[12] pro pedagogické výkony a VVVSapi[21] pro výkony akademické. Aplikace vytvoří hodnocení pracovníka za libovolný časový interval (zaokrouhlený na semestry), ve kterém se sečtou body za jednotlivé aktivity. Aktivity zahrnují přednášky, cvičení a položky ze systému VVVS (RIV body)[20][18]. Body za aktivity budou vynásobeny koeficienty, které si nastaví vedoucí katedry, případně zakladatel hodnocení.

V hodnocení bude vestavěn nástroj pro komentáře, aby se hodnotící i pracovník mohli vyjádřit k předchozímu období a zhodnotit podané výkony. Systém bude umožňovat vkládání jednotlivých aktivit, které nebyly systémem zohledněny. Dále bude pracovník moci specifikovat plán na budoucí období s možností schválení hodnotícím a tím usnadnit plánování do budoucnosti.

Aplikace bude autentizovat uživatele pomocí centralizovaného systému FELid[4].

2.3 Analýza

Součástí každého hodnocení je kvantifikovatelná a nekvantifikovatelná část. Naše aplikace měla především sbírat měřitelná data a poskytnout prostor k vyjádření se ke zbytku. Existují dva typy měřitelného výkonu.

- Pedagogický výkon
- Akademický výkon

2.3.1 Aktéři

Do našeho hodnotícího systému zasahují dva aktéři.

- Hodnocený – pracovník jehož výkon se hodnotí.
- Hodnotící – většinou vedoucí katedry, či jiný přímý nadřízený.

2.3.2 Pedagogický výkon

Vytvoření metriky pedagogického výkonu je obtížné. O existenci Komety[10] jsme v době vývoje osobního hodnocení nevěděli a museli jsme si tedy vytvořit metriku vlastní.

Aktivita

Atomickou jednotkou pro naší metriku se stala aktivita, která se vztahuje na vyučovaný předmět. Aktivity mohou být různé, v naší analýze jsme dospěli ke třem typům pozice vyučujícího.
Worktype:

- Cvičící
- Přednášející
- Garant

Tedy za každý předmět mohly být přiděleny až tři aktivity. Za aktivity byly přidělené body dle hodnot následujících atributů.
Atributy:

- Počet vyučovaných hodin [Pvh]
- Počet studentů [Ps]

Pro daný worktype a atribut existuje koeficient, který ho upravuje. Koeficienty nastavuje hodnotící při vytváření hodnocení.
Koeficienty:

- Fixní koeficient [kFix]
- Koeficient počtu studentů [kPs]
- Koeficient počtu hodin [kPvh]

Výpočet celkového počtu bodů tedy probíhá takto:

$$wt = worktype$$
$$Body[wt] = kFix[wt] + Pvh * kPvh[wt] + Ps * kPs[wt]$$

Díky možnosti volby těchto koeficientů je tento systém velmi flexibilní.

2.3.3 Akademický výkon

Metriky akademického výkonu jsou lépe definované, než metriky výkonu pedagogického. Systém VVVS eviduje výsledky vědy a výzkumu, (publikační výsledky, výsledky aplikovaného výzkumu), akce (granty, výzkumné záměry a smlouvy) a další aktivity vědecko-výzkumných pracovníků ve vědecké komunitě.[20]

V systému VVVS jsou jednotlivým publikacím přiřazeny body body RIV[18], které slouží jako hlavní metrika pro daný výkon. RIV body jsou přidělovány za celou publikaci, tedy i pro více autorů. V době vývoje nebylo možné získat poměr rozdělení RIV bodů na jednotlivé pracovníky a tedy skutečná hodnota akademického výkonu zůstala na osobní konzultaci s hodnotícím.

2.3.4 Dodatečné bodování

Jelikož jsme nemohli získat dodatečné faktory při udělování bodů (obtížnost, příprava, zvláštní okolnosti) za jednotlivé aktivity, tak jsme se rozhodli zavést systém dodatečného bodování. U každé aktivity je možné upravit počet bodů, tuto akci dělá hodnocený a je poskytnut prostor pro komentář. Hodnotící poté zváží změněný počet bodů a má možnost ho upravit a schválit.

2.3.5 Výměna komentářů

Webová stránka zobrazení hodnocení by měla poskytovat možnost pro výměnu komentářů mezi hodnotícím a hodnoceným. Nebylo naším úmyslem implementovat chatovací systém a tedy počet komentářů je omezený.

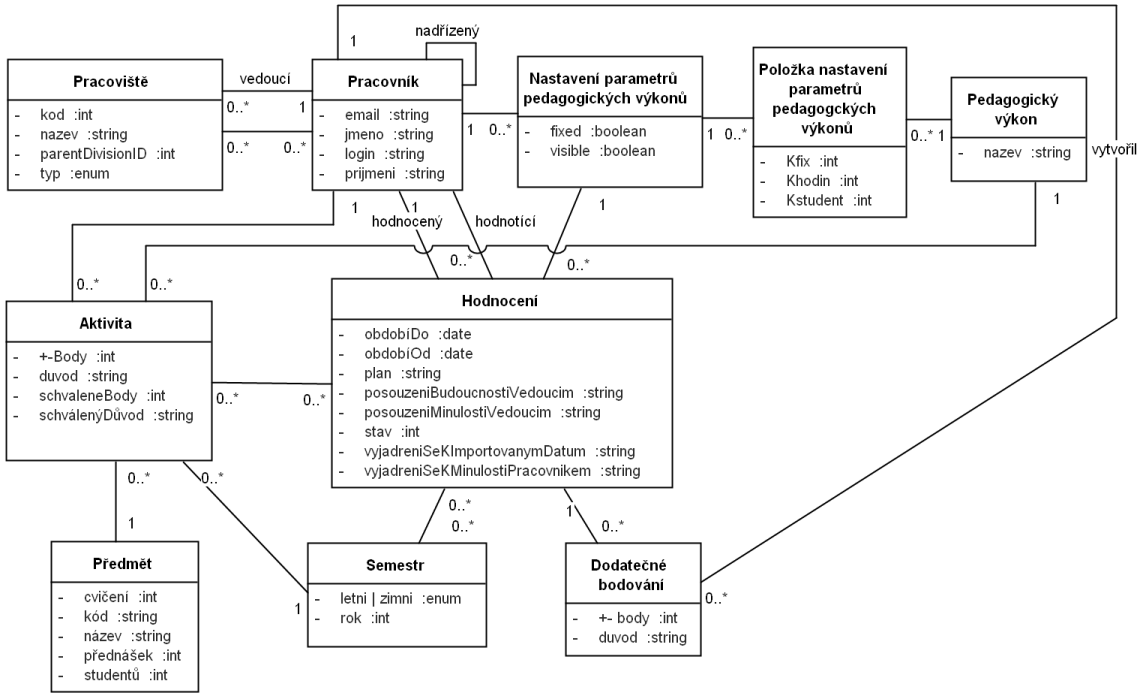
V jednom hodnocení byla možnost vytvořit tyto komentáře:

- Vyjádření hodnoceného k minulému plánu hodnoceného.
- Vyjádření hodnotícího k minulému plánu hodnoceného.
- Budoucí plán
- Vyjádření hodnotícího k budoucímu plánu.
- Vyjádření hodnoceného k předchozímu vyjádření.
- Finální vyjádření hodnotícího k celému hodnocení.

Tyto komentáře měly poskytnout dostatečný prostor pro většinu potřebné komunikace, ovšem neměly nahradit osobní pohovor, ve kterém se dá probrat více věcí do větší hloubky.

2.3.6 Uzavírání hodnocení

Hodnotící má možnost natrvalo uzavřít hodnocení tak, že se budoucí plán bude zobrazovat jako jeden z minulých plánů při zakládání nového hodnocení. V případě předčasného uzavření hodnocení je jedna možnost uzavření revertovat.



Obrázek 2.1: Diagram tříd pro problematiku osobní hodnocení

2.4 Průběh vývoje

Nejdříve byla nutnost seznámit se s KOSapi a VVVSapi. Jedná se o restové služby, které dosud nemají vzorové implementace pro žádný jazyk. V rámci vývoje byla vytvořena implementace dostačující pro účel naší aplikace v Javě.

Zažádali jsme o server, který je nyní hostovaný na školou poskytnutém virtuálním stroji a je dostupný na doméně hop.fel.cvut.cz. Na tomto serveru je také zprovozněn autentizační klient systému FELid, běžící jako modulární součást serveru Apache. Po úspěšném přihlášení jsou požadavky přesměrovány na server Apache Tomcat, na kterém běží aplikace.

2.5 Problémy

Nedostatky systému KOSapi a nesprávné údaje v systému KOS byly největším problémem v návrhu a implementace aplikace.

- Nesprávné zařazení pracovníků do kateder – Náš systém byl navržen pro vedoucí kateder, tedy při zakládání hodnocení byl zobrazen seznam pracovníků dané katedry. Očividný problém nastal při kontrole správnosti zobrazených pracovníků. V systému KOS jsou někteří pracovníci k dnešnímu datu úmyslně nebo neúmyslně zařazeni špatně.
- Neexistence historie zařazení pracovníků do kateder – Při zakládání hodnocení na předchozí období nemohly být zjištěny předchozí stavy pracovních poměrů. Tedy

seznam pracovníků do kateder nebyl aktuální.

- Nemožnost zjistit stav pracovního poměru – Všichni pracovníci v historii KOSu se v KOSapi tváří jako stále zaměstnaní, tedy při zakládání hodnocení mohl v závislosti na katedře seznam obsahovat několik stovek uživatelů. Jedinou možností bylo vytvořit manuální blacklist pracovníků pro katedru. Takové řešení není ideální.
- Množství dat nutné k importování z KOSapi – V důsledku separace datových položek v KOSapi (předměty, studenti, paralelky) a nemožnosti se na KOSapi dotazovat jako na databázi, například pomocí složených dotazů, bylo nutné importovat značné množství dat při zakládání hodnocení. Finální čas založení hodnocení se pohyboval v řádu několika minut.

2.6 Závěr

Po třech měsících vývoje byla práce na osobním hodnocení zastavena a rozestavená infrastruktura byla použita jako základ nové funkcionality, která je popsána v kapitole 3. Aplikace na konci léta byla ve stavu funkčního prototypu, který byl nasazený na server v plném provozu. Aplikace může být v případě zájmu může být v budoucnu lehce obnovena a dopracována.

Kapitola 3

Problematika: výpočet pedagogického výkonu

3.1 Popis problému

Na ČVUT FEL existuje metodika Kometa2[10] pro přerozdělování finančních prostředků mezi katedrami. Tato metodika definuje kvantifikovatelný údaj započitatelných hodin, který určuje přibližný počet hodin, které jsou spotřebované za určitý typ aktivity.

Kometa2 ovšem primárně neřeší rozpočet hodin mezi jednotlivé zaměstnance. Děkan FEL ČVUT projevil osobní zájem na implementaci rozpočtu na zaměstnance.

3.2 Zadání

Vytvořte aplikaci, která dle poslední verze metodiky Kometa2 vypočítá započitatelné hodiny za přednášky, cvičení, bakalářské a diplomové práce, zkoušení a projektové předměty pro jednotlivé učitele v daném semestru. O každém výpočtu vytvořte podrobnou textovou zprávu popisující průběh výpočtu. Zobrazte tento výpočet v řaditelném a přehledném seznamu pro vybrané pracovníky. Umožněte export této tabulky do .xls a .csv souborů.

3.3 Průběh vývoje

Na základě osobní žádosti děkana ČVUT FEL, byla práce na osobním hodnocení přerušena na začátku zimního semestru 2013 a aplikace byla přesměrována na výpočet pedagogického výkonu jednotlivých pracovníků. Na rozdíl od osobního hodnocení, které lehce definovalo flexibilní výpočet, je nová část výpočtu pedagogického výkonu pevně založená na metodice Kometa2.

3.3.1 Metodika Kometa2

Metodika pro stanovení pedagogického výkonu kateder, materiálové náročnosti studia a pro rozdělování finančních prostředků za výuku na katedry.

Na začátku analýzy výpočtu jsme se seznámili s metodikou Kometa2, která vypočítává započitatelné hodiny jednotlivé katedry. V zadání ovšem bylo provést výpočet přes jednotlivé pracovníky, bylo tedy nutné metodiku poupravit tak, aby byly výpočty aplikovatelné.

3.3.2 Externí aplikace Kometa2

Krátce po začátku analýzy jsme objevili aplikaci[11], která úspěšně implementuje metodiku Kometa2 tak, jak je sepsána a poskytuje použitelné exporty v .xlsm formátu.

3.4 Kometa

V kometě je definováno několik vzorců pro celkový výpočet započitatelných hodin za katedru. Většina vzorců obsahuje součet přes všechny předměty katedry v daném semestru. Tedy úprava je jednoduchá, zpracovávat každý předmět samostatně. Ovšem některé vzorce jsme museli modifikovat, tak aby zohlednili výpočet, případě podíl výuky jednotlivce.

Počet paralelek a počet studijních skupin

Definováno v kometě takto. Počet paralelek přednášek P_{par} a počet studijních skupin cvičení P_{ss} se stanoví z počtu studentů zapsaných na předmět P_{stud} a stanovených koeficientů P_{max} (standardní počet studentů v paralelce či studijní skupině) a P_{min} (minimální počet studentů v paralelce či studijní skupině započítávaných již jako celá paralelka či studijní skupina) podle vztahů:

$$M = \left\lfloor \frac{P_{stud}}{P_{max}} \right\rfloor, N = P_{stud} \bmod P_{max} \tag{3.1}$$

$$\text{pro } 0 \leq N < P_{min} \text{ je } P_{ss} = M + \frac{N}{P_{min}}, \text{ jinak je } P_{ss} = M + 1 \tag{3.2}$$

$$\text{pro } 0 \leq P_{stud} < P_{min} \text{ je } P_{par} = \frac{P_{stud}}{P_{min}}, \text{ jinak je } P_{par} = 1 \tag{3.3}$$

kde parametry P_{min} a P_{max} se určí dle tabulky 3.1.

Program	Jazyk	P_{min}	P_{max}
BSP a MSP	Česky	10	20
BSP a MSP	Anglicky	5	20
DSP	Nerozhoduje	4	10

Tabulka 3.1: Stanovení koeficientů P_{min} a P_{max}

Započitatelné hodiny za přednášky

V kometě definováno pro všechny paralelky přednášek přes všechny semestry takto.

$$ZH_{pr} = \sum_{predmety} P_{par} \cdot H_{ps} \cdot K_p \cdot K_j \tag{3.4}$$

Pro naše potřeby počítáme paralelky, ve kterých se vyučující angažoval. Tedy náš použitý vzorec pro jednu paralelku vypadá následovně.

$$ZH_{pr} = P_{par} \cdot H_{ps} \cdot K_p \cdot K_j \cdot M_{pv} \tag{3.5}$$

kde:

P_{par} je počet paralelek

H_{ps} je počet hodin přednášek za semestr

K_p je koeficient přednášek, $K_p = 4$

K_j je koeficient jazyka, dle tabulky 3.2

M_{pv} je náš modifikátor podílu výuky. Pokud vyučuje předmět více učitelů, pak vyjadřuje jejich podíl na výuce. Tento údaj se získává dvěma způsoby. Buď je uveden v KOSu nebo je propočtený rovnoměrně na celkový počet učitelů.

$$M_{pv} = \text{počet učitelů}^{-1} \tag{3.6}$$

Předmět vyučovaný v češtině	$K_j = 1.0$
Předmět vyučovaný v angličtině	$K_j = 1.2$

Tabulka 3.2: Stanovení koeficientu jazyka K_j

Započitatelné hodiny za cvičení

V kometě je definováno pro všechny paralelky cvičení přes všechny předměty takto.

$$ZH_{cv} = \sum_{predmety} (ZH_{cvu} + ZH_{cvp}) \tag{3.7}$$

Kde ZH_{cvu} jsou započitatelné hodiny za přítomnost učitele a ZH_{cvp} jsou započitatelné hodiny za technickou podporu.

Technická podpora je určena pomocí počtu týdnů strávené v laboratorních pracovištích a je připočítávána katedře zaštiťující učebnu. Jelikož je tento parametr obtížné vypočítat z dat z KOSapi, přímo se nevztahuje ke spotřebovaným započitatelným hodinám za pedagogický výkon učitele a nepřipisuje se vždy katedře pod kterou vyučující spadá, rozhodli jsme se ho vynechat.

Samotný vzorec pro výpočet ZH_{cvu} je v kometě definován takto.

$$ZH_{cvu} = P_{ss} \cdot H_{cs} \cdot K_{cv} \cdot K_{pnp} \cdot K_j \tag{3.8}$$

Náš upravený vzorec vypadá následovně.

$$ZH_{cvu} = P_{ssm} \cdot H_{cs} \cdot K_{cv} \cdot K_{pnp} \cdot K_j \cdot M_{pv} \tag{3.9}$$

kde:

H_{cs} je počet hodin cvičení za semestr

K_{cv} je koeficient cvičení, $K_{cv} = 2$

K_{pnp} je průměrný koeficient pedagogické náročnosti předmětu

M_{pv} je již známý modifikátor podílu výuky P_{ssm} je modifikovaný počet studijních skupin. Počet studijních skupin je v kometě počítán přes součet studentů ve všech paralelkách. Pro správné rozpočítání započitatelných hodin za paralelky cvičení kterých se učitel zúčastní musíme počet studijních skupin upravit podílem studentů přítomných na paralelkách vyučujícího.

$$P_{ssm} = \frac{P_{spv}}{P_{sp}} \cdot P_{ss} \tag{3.10}$$

kde:

P_{spv} je počet studentů v zahrnutých paralelkách vyučujícího

P_{sp} je počet studentů v předmětu

P_{ss} je počet studijních skupin, vypočítaný dle vzorce 3.2

Započitatelné hodiny za zkoušení v předmětu

V kometě je definováno pro všechny předměty takto.

$$ZH_{zk} = \sum_{predmety} P_{stud} \cdot K_{zk} \cdot K_j \tag{3.11}$$

Pro naše potřeby jsou započitatelné hodiny za zkoušení připočítávány přednášejícímu. Je-li více přednášejících je použit modifikátor M_{pv}.

$$ZC_{zk} = P_{stud} \cdot K_{zk} \cdot K_j \cdot M_{pv} \tag{3.12}$$

kde:

P_{stud} je počet studentů zapsaných na předmět

K_{zk} je koeficient zkoušení, určený dle tabulky 3.3

Zkouška ve všech ročnících a formách studia	$K_{zk} = 0.8$
Klasifikovaný zápočet ve všech ročnících a formách studia	$K_{zk} = 0.2$

Tabulka 3.3: Stanovení koeficientu zkoušení K_{zk}

Započitatelné hodiny za diplomové a bakalářské práce

V kometě je definováno pro všechny práce takto.

$$ZH_{ost} = K_j \cdot ZH_o \cdot P_{stud} \tag{3.13}$$

V našem případě započítáváme a zobrazujeme každou práci zvlášť, ale jinak je přepis vzorce přímočarý.

$$ZH_{dpbp} = K_j \cdot ZH_o \tag{3.14}$$

kde:

ZH_o koeficient určený dle tabulky 3.4

	BSP	**MSP**
individuální projekt	12	14
týmový projekt	6	7
	BP	**DP**
Vedení + posudek vedoucího	20	25
Posudek oponenta	3	4
SZZ + obhajoba	6	8

Tabulka 3.4: Stanovení koeficientu ZH_o

Započitatelné hodiny za ukončené projekty projektových předmětů

V kometě je definováno pro všechny projekty takto.

$$ZH_{ost} = K_j \cdot ZH_o \cdot P_{stud} \tag{3.15}$$

V našem případě započítáváme a zobrazujeme každý projekt zvlášť, ale jinak je přepis vzorce přímočarý.

$$ZH_{pro} = K_j \cdot ZH_o \tag{3.16}$$

3.5 Analýza

Slučování paralelek

V běžném rozvrhování jsou některé paralelky různých souvisejících předmětů vyučovány ve stejném časovém slotu se stejnými učiteli. Pro každý takovýto případ jsou paralelky sloučeny dle definice níže.

Slučování probíhá při výpočtu ZH_{pr} a ZH_{cv}.

Při slučování paralelek vznikají výjimky.

- Paralelky jsou vyučovány ve stejný čas, trvají stejnou dobu a mají dva vyučující. Z toho jeden vyučující je sdílený mezi oběma paralelkami a ten druhý je v každé paralelce jiný.

- Paralelky jsou vyučovány stejným učitelem a ve stejný čas. Ovšem netrvají stejnou dobu.
- Paralelky jsou vyučovány stejným učitelem a trvají stejný čas, ale jedna z nich začíná o hodinu později a tedy překrývají se pouze jednu hodinu.

Při jakékoliv z těchto výjimek naše aplikace paralelky neslučuje.

Definice postupu slučování

Postup slučování: Paralelky vyučujícího v daném semestru vyučované ve stejný den a po stejnou dobu sluč do jediné paralelky, která měla největší počet studentů a nastav jí součet studentů všech slučovaných paralelek.

Externí import: aplikace Kometa2

Součástí importované tabulky z externí aplikace Kometa2 jsou veškeré koeficienty, které se používají při výpočtů hodnot z Komety. Dále tam jsou také dodatečné údaje o vypočítávaných předmětech.

Některé údaje, především počet studentů a jazyk výuky jsou v KOSapi nepřesné, ale v exportu z aplikace Kometa2 jsou správné. A dále například parametr materiálové náročnosti za cvičení nelze z momentálně dostupných dat v KOSapi vypočítat. Naše aplikace tedy přebírá tyto a další údaje z importu a je tedy pro svůj chod na aplikaci Kometa2 závislá.

Externí import: Diplomové a Bakalářské práce

Z důvodu ochrany citlivých dat, nejsou v KOSapi uvedeny hodnocení jednotlivých diplomových ani bakalářských prací. Museli jsme tedy přistoupit k externímu importu dat ze systému KOS.

Externí import: projektové předměty

Ze stejných důvodů ochrany soukromí jako u diplomových a bakalářských prací nemohou být v KOSapi uvedeny ani hodnocení projektových předmětů, respektive projektů v nich zpracovávaných. Dostáváme externí import dat ze systému KOS.

Externí import: poměr výuky

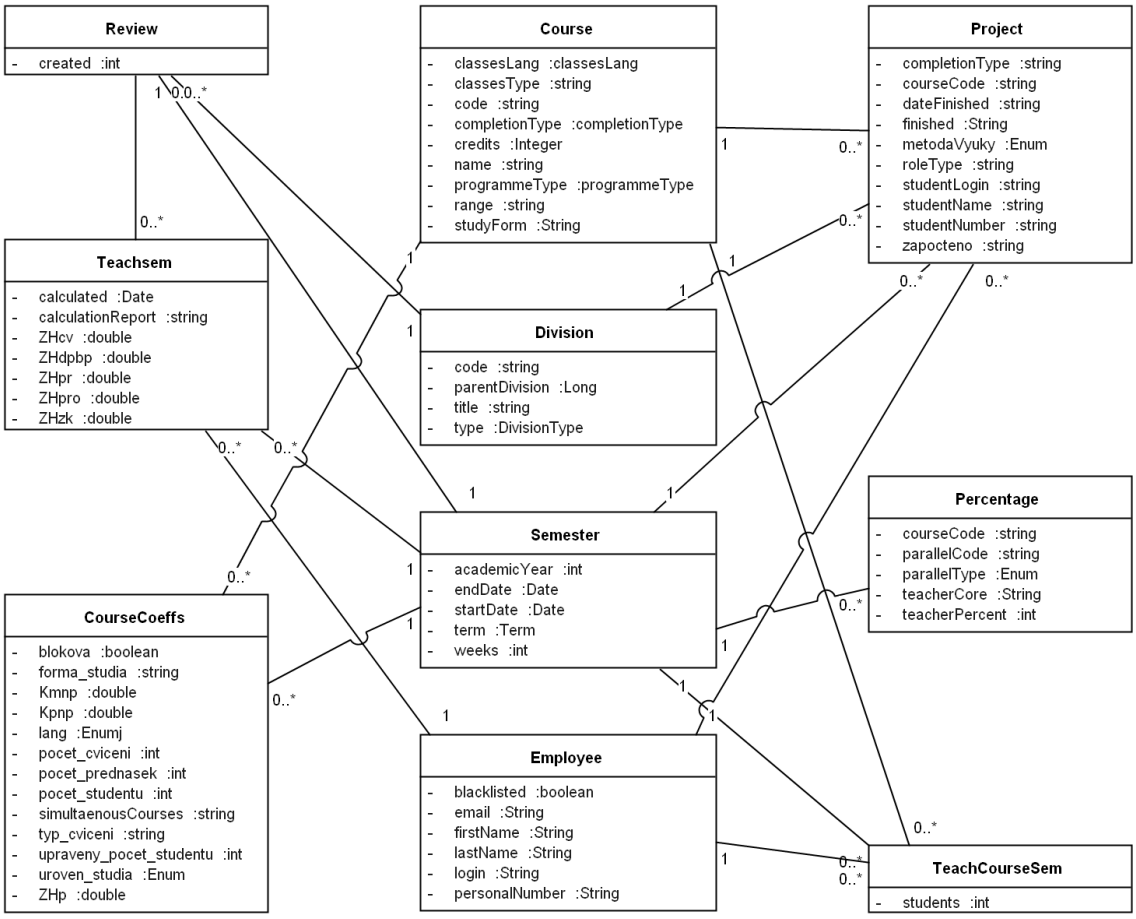
V případě většího počtu vyučujících na vyučované paralelce je zaznamenán podíl jejich výuky v procentech. Tato informace je důležitá při přepočtu započitatelných hodin mezi jednotlivé vyučující. KOSapi tuto informaci neposkytuje. Dostáváme tedy externí import dat ze systému KOS. Tyto data nejsou citlivá a do budoucna by mohly být zpřístupněny v KOSapi.

3.6 Implementace

3.6.1 Diagramy

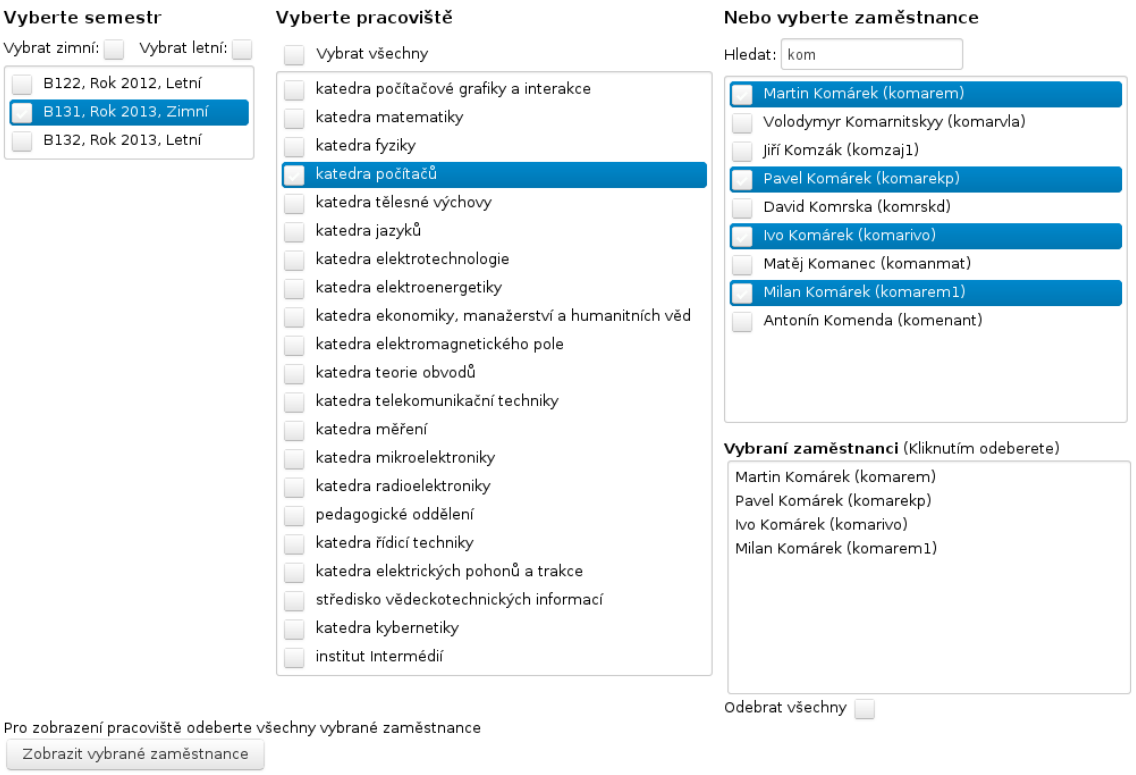
V implementačním diagramu tříd pedagogického výkonu 3.1 se nachází rozložení tříd.

- `Review` – Třída zaštiťující jednotlivé iterace výpočtů
- `Teachsem` – Třída uchovávající vypočtené hodnoty započitatelných hodin a textový záznam výpočtu
- `TeachCourseSem` – Třída uchovávají počet studentů vyučovaných učitelem v předmětu za semestr.
- `CourseCoeffs` – Třída uchovávající data importovaná z Komety
- `Project` – Třída uchovávající importovaná data projektů
- `Percentage` – Třída uchovávající importovaná data poměru výuky



Obrázek 3.1: Diagram tříd pedagogického výkonu

3.7 Uživatelské rozhraní

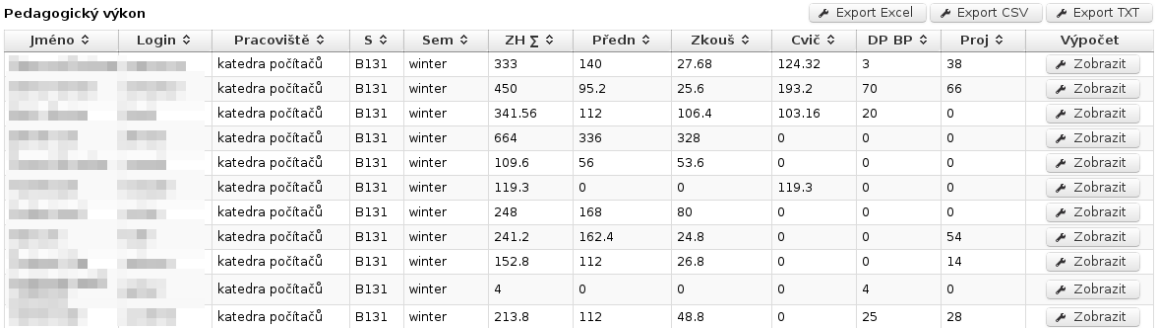


Obrázek 3.2: UI pedagogického výkonu – výběr

Uživatelské rozhraní je strohé a soustředí se především na přehledné zobrazení výsledků výpočtů. Na úvodní stránce aplikace (obrázek 3.2) se nachází výběr dat která mají být zobrazena. Výběr je dělený dle semestrů a pracovišť. Pokud si uživatel přeje, může podle jména vybrat jednotlivé uživatele a zobrazit je ve zvolených semestrech.

Na obrázku 3.3 je vidět zobrazení výsledků po vybrání dat na úvodní obrazovce. Jedná se o tabulku se jmény a přihlašovacími loginy jednotlivých vyučujících, jejich pracoviště a zvolený semestr. Následuje zobrazení vypočtených započitatelných hodin za jednotlivé faktory. Na této stránce je možné zobrazit detailní zprávu výpočtu3.4 pro jakýkoliv záznam.

V detailní zprávě se nachází záznam výpočetního algoritmu, na kterém jsou uvedené jednotlivé elementy výpočtu. V případě vyučování jsou uvedeny jednotlivé započítávané paralelky a koeficienty, které vstupují do finálního výpočtu a celkového součtu součtu.

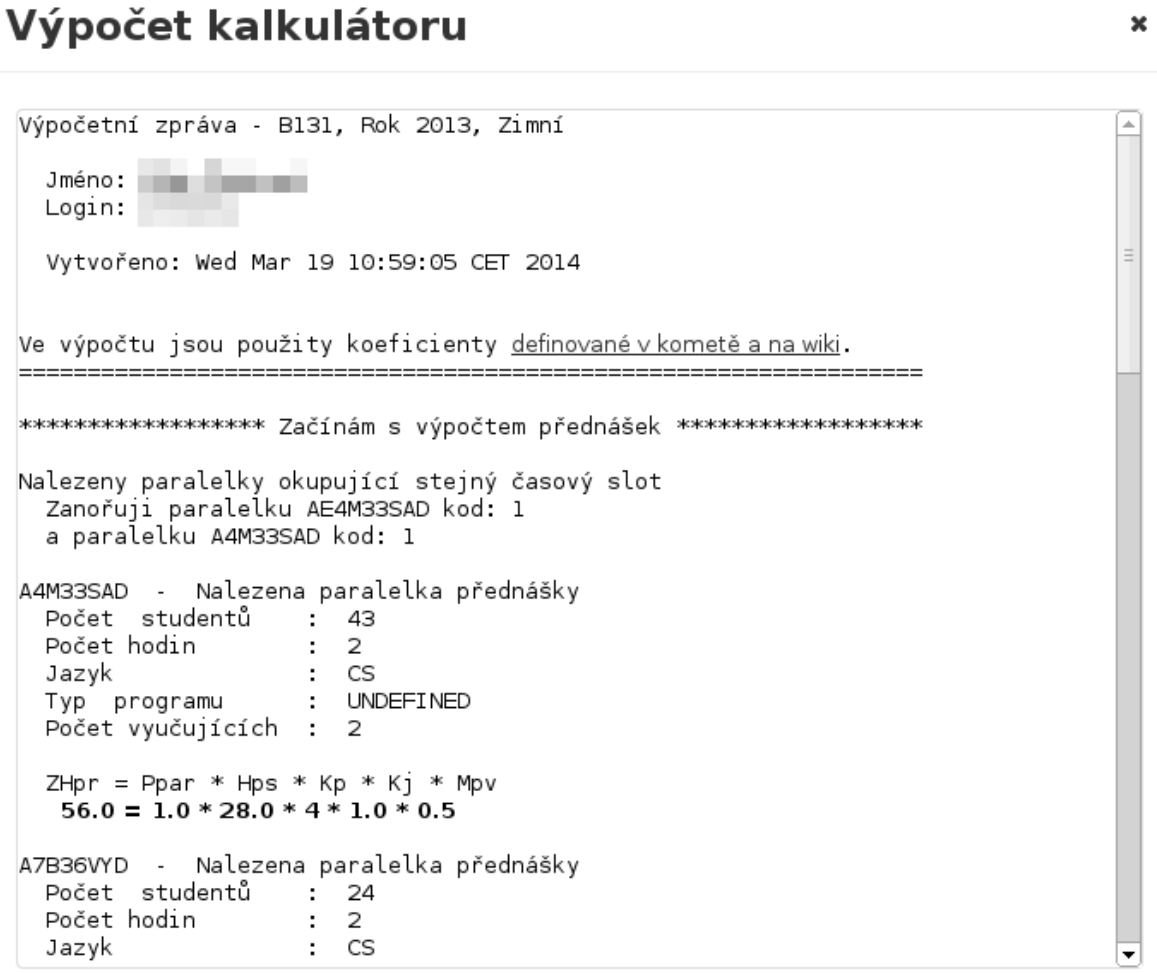


Pedagogický výkon

Export Excel | Export CSV | Export TXT

Jméno	Login	Pracoviště	S	Sem	ZH ∑	Přednˇ	Zkouš	Cvič	DP BP	Proj	Výpočet
[redacted]	[redacted]	katedra počítačů	B131	winter	333	140	27.68	124.32	3	38	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	450	95.2	25.6	193.2	70	66	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	341.56	112	106.4	103.16	20	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	664	336	328	0	0	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	109.6	56	53.6	0	0	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	119.3	0	0	119.3	0	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	248	168	80	0	0	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	241.2	162.4	24.8	0	0	54	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	152.8	112	26.8	0	0	14	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	4	0	0	0	4	0	Zobrazit
[redacted]	[redacted]	katedra počítačů	B131	winter	213.8	112	48.8	0	25	28	Zobrazit

Obrázek 3.3: UI pedagogického výkonu – zobrazení



Výpočet kalkulátoru

```
Výpočetní zpráva - B131, Rok 2013, Zimní

  Jméno: [redacted]
  Login: [redacted]

  Vytvořeno: Wed Mar 19 10:59:05 CET 2014


Ve výpočtu jsou použity koeficienty definované v kometě a na wiki.
======================================================================

********** Začínám s výpočtem přednášek **********

Nalezeny paralelky okupující stejný časový slot
  Zanořuji paralelku AE4M33SAD kod: 1
  a paralelku A4M33SAD kod: 1

A4M33SAD  -  Nalezena paralelka přednášky
  Počet  studentů    :  43
  Počet hodin        :  2
  Jazyk              :  CS
  Typ  programu      :  UNDEFINED
  Počet vyučujících  :  2

  ZHpr = Ppar * Hps * Kp * Kj * Mpv
   56.0 = 1.0 * 28.0 * 4 * 1.0 * 0.5

A7B36VYD  -  Nalezena paralelka přednášky
  Počet  studentů    :  24
  Počet hodin        :  2
  Jazyk              :  CS
```

Obrázek 3.4: UI pedagogického výkonu – zpráva kalkulátoru

Kapitola 4

Problematika: efektivita výuky

4.1 Popis problému

Efektivita výuky vyvstala jako vedlejší požadavek k výpočtu započitatelných hodin pro katedru. Jedná se o výpočet a zobrazení započitatelných hodin rozpočítaných dle jednotlivých studijních programů.

4.2 Zadání

Rozpočítejte započitatelné hodiny za semestr mezi jednotlivé studijní programy podle počtu studentů.

4.3 Průběh vývoje

Na konci zimního semestru 2014 byla zahájena práce na efektivitě výuky na přání děkana ČVUT FEL. Vzhledem k jednoduchosti zadání následoval celkem přímočarý vývoj.

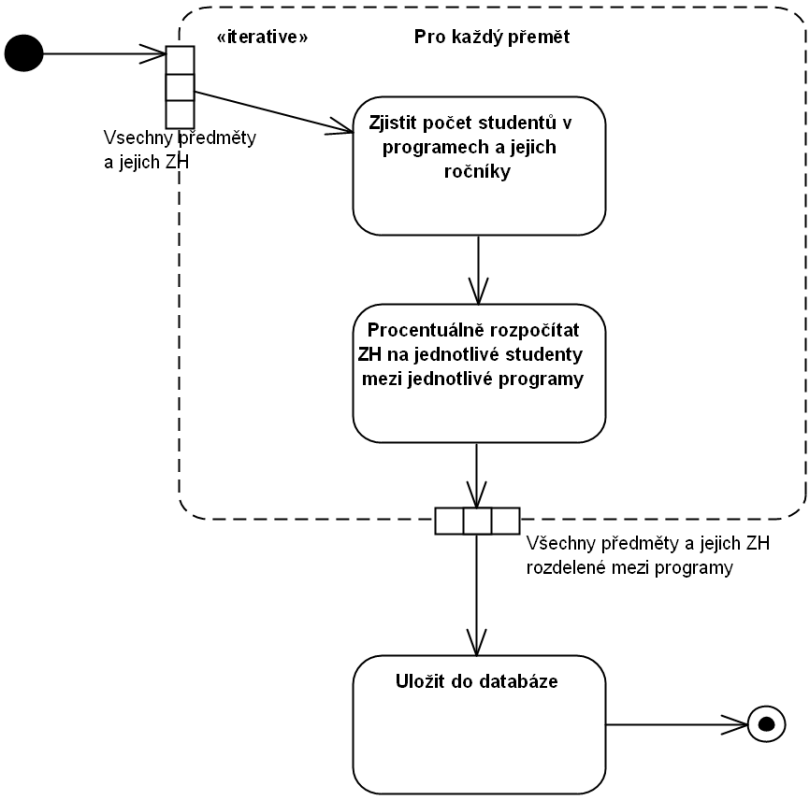
Po skončení vývoje nastal požadavek rozpočítávat i koeficient průměrné materiálové náročnosti předmětu (Kmnp), získaný z exportů aplikace Kometa2[11].

4.4 Analýza

4.4.1 Výpočet

Do výpočtu vstupují data importovaná z Komety[10], přesněji započitatelné hodiny za jednotlivé předměty a koeficient průměrné pedagogické náročnosti předmětu. Postup výpočtu je znázorněn v diagramu 4.1.

Pro každý předmět z Komety zjistíme z KOSapi počet studentů v jednotlivých programech a jejich ročníky. Následně mezi jednotlivé studenty rozpočítáme započitatelné hodiny a Kpnp.



Obrázek 4.1: Diagram aktivit výpočtu efektivity

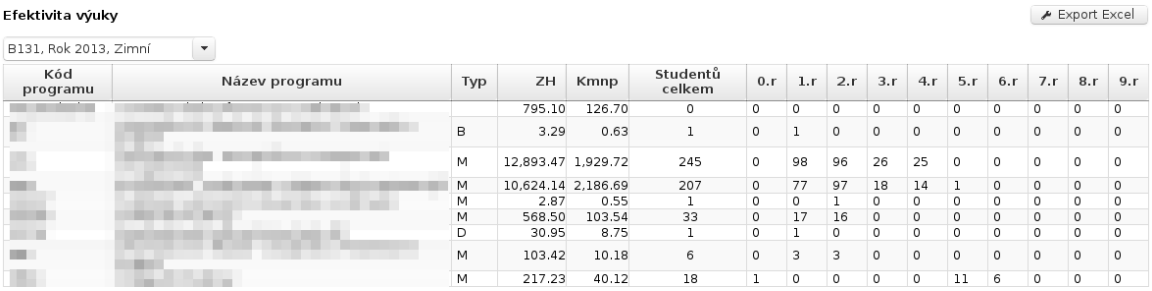
4.4.2 Archiv studentů

KOSapi neposkytuje možnost získat ročník studenta v předchozích semestrech, musel bý tedy vytvořen lokální archiv studentů. Tento archiv je implementován pomocí celkových a částečných snapshotů, kdy při částečném jsou uloženy jen data o studentech jejichž příslušná pole se změnila od posledního celkového snapshotu.

V archivu jsou ukládána taková data o kterých se předpokládá že se budou s postupem času měnit, v případě studentova ročníku se jedná o změnu pravidelnou, ovšem mohou nastat i změny nepravidelné jako například ukončení či přerušení studia nebo přechod na jiný studijní obor.

4.4.3 Zobrazení

Uživateli zobrazíme data v jednoduché a přehledné tabulce řazená dle jednotlivých programů. Jak lze vidět na obrázku 4.2.


Efektivita výuky

Export Excel

B131, Rok 2013, Zimní

Kód programu	Název programu	Typ	ZH	Kmnp	Studentů celkem	0.r	1.r	2.r	3.r	4.r	5.r	6.r	7.r	8.r	9.r
			795.10	126.70	0	0	0	0	0	0	0	0	0	0	0
		B	3.29	0.63	1	0	1	0	0	0	0	0	0	0	0
		M	12,893.47	1,929.72	245	0	98	96	26	25	0	0	0	0	0
		M	10,624.14	2,186.69	207	0	77	97	18	14	1	0	0	0	0
		M	2.87	0.55	1	0	0	1	0	0	0	0	0	0	0
		M	568.50	103.54	33	0	17	16	0	0	0	0	0	0	0
		D	30.95	8.75	1	0	1	0	0	0	0	0	0	0	0
		M	103.42	10.18	6	0	3	3	0	0	0	0	0	0	0
		M	217.23	40.12	18	1	0	0	0	0	11	6	0	0	0

Obrázek 4.2: Uživatelské rozhraní pro efektivitu výuky

4.5 Implementace

4.5.1 Diagramy

V implementačním diagramu tříd efektivity výuky 4.3 se nachází rozložení jednotlivých tříd, tak jak jsou použity v aplikaci.

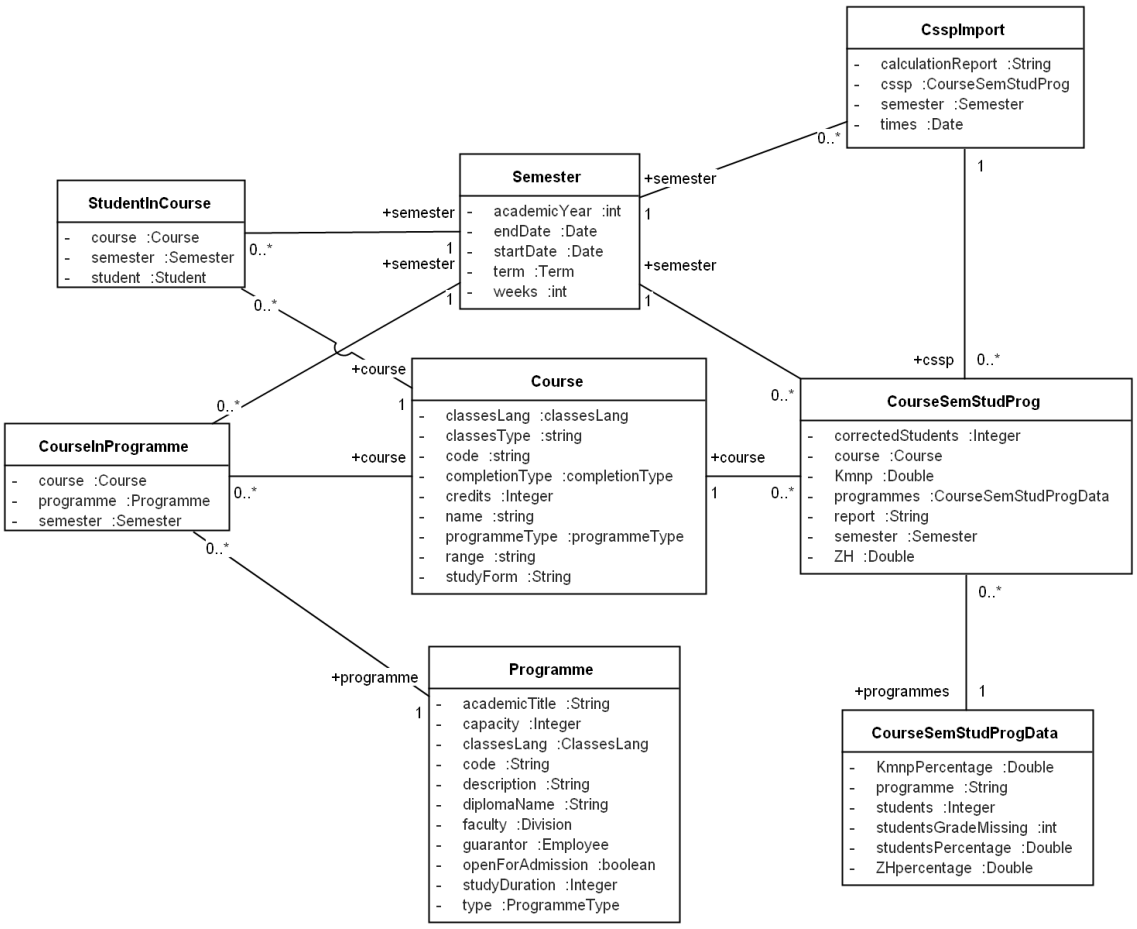
- `Course` – Třída reprezentující předmět (kopírující strukturu /course z KOSapi)
- `Programme` – Třída reprezentující studijní program (/programme z KOSapi)
- `Semester` – Trda reprezentující semestr
- `CourseInProgramme` – Mapuje zařazení předmětu do programu v danném semestru.
- `StudentInCourse` – Mapuje účast studenta na předmětu v danném semestru.
- `CourseSemStudProg` – Hlavní třída, pro předmět v danném semestru ukazuje počet zapsaných studentů a jejich spotřebované ZH přes všechny programy.
- `CourseSemStudProgData` – Třída uchovávající hodnoty pro program.
- `CsspImport` – Třída která seskupuje jednotlivé importy (založení) výpočtu efektivity. Určuje datum a čas výpočtu.

V implementačním diagramu tříd archivu studentů 4.4 se nachází rozložení tříd.

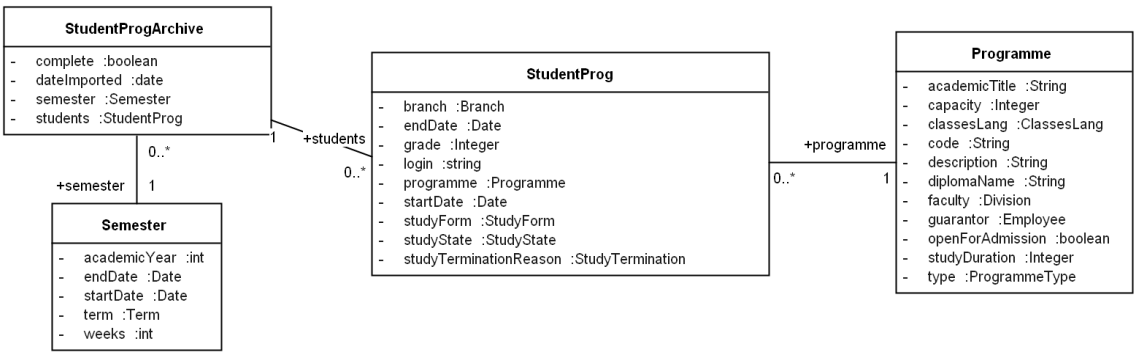
- `StudentProg` – Třída uchovávající stav studenta v době importu archivu
- `StudentProgArchive` – Třída uchovávající jednotlivé importy stavů studentů do archívu.

4.6 Závěr

Funkcionalita problematiky efektivity výuky je nasazená a použitelná. V době psaní této práce je dostupná z pomocného menu v patičce aplikace.



Obrázek 4.3: Diagram tříd efektivity výuky



Obrázek 4.4: Diagram tříd archivu studentů

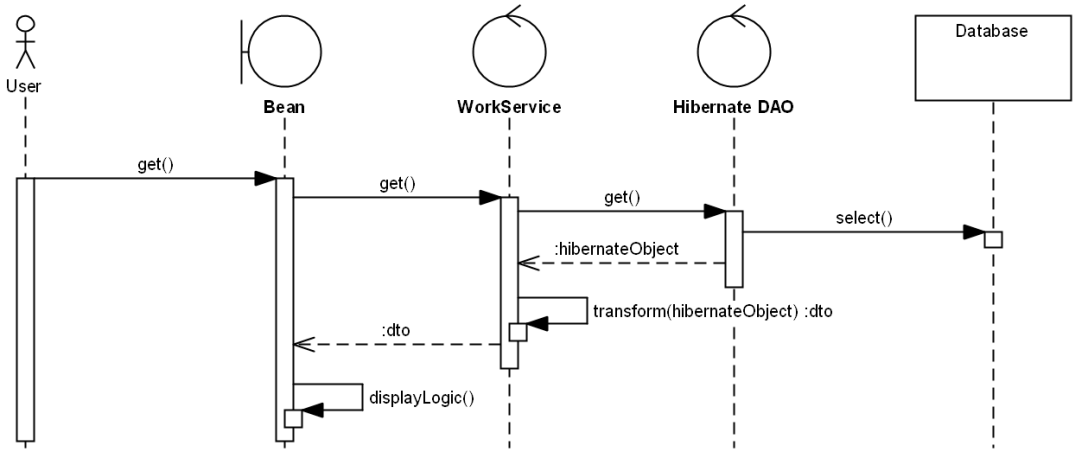
Kapitola 5

Architektura kódu

Aplikace je postavena na Model-View-Controller[5] architektuře. O komunikaci s uživatelem, tedy o vrstvu view, se stará JSF[9] v kombinaci s komponentovou knihovnou Primefaces[17]. Obsluha komponent je zajišťována pomocí tříd zvaných beany, ve kterých je prováděna zobrazovací logika a řadí se do vrstvy controller. Business logika a databázová logika je zajišťována v modelové vrstvě.

5.1 Modelová vrstva

Modelová vrstva je co se týče funkcionality aplikace ta nejdůležitější a také nejrozsáhlejší. Skládá se z jednotlivých databázových entit, obsluhovaných knihovnou Hibernate, která se stará o jejich persistenci do zvolené databáze, v našem případě se jedná o databázový server PostgreSQL[16]. Entity jako takové obsahují velmi málo business logiky, obsahují pouze logiku, která je přímo návazná na operace prováděné s jednoduchými datovými typy, které jsou obsažené v dané entitě.

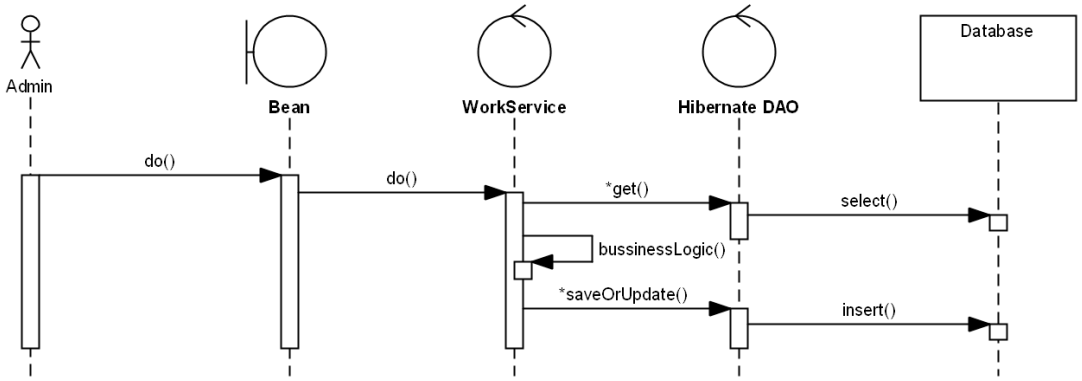


Obrázek 5.1: Získání dat z databáze

Součástí modelové vrstvy jsou servisní objekty. Jsou rozdělené do dvou kategorií, na ty které obstarávají manipulaci s entitami a na ty které zpracovávají výpočty nad nimi. Manipulační servisní objekty obstarávají základní funkcionalitu Create Read Update Delete.

Na obrázku 5.1 je vidět jednoduchá operace získání objektu z databáze. Uživatel vyšle požadavek na naši aplikaci, ten je zpracován beanou, která ho předá modelové vrstvě, přesně servisní třídě. Servisní třída dále vyvolá Hibernate Data Access Object (DAO), ve kterém je uchován potřebný dotaz v jazyku HQL[8], ten je zpracován a přeložen do správné SQL formy knihovnou Hibernate a odeslán na databázi. Hibernate po přijetí odpovědi z databáze vytvoří příslušnou entitu. Entita nemusí být, díky lazy loadingu, plně inicializovaná, proto je servisní metoda součástí transakčního prostředí, tak aby z entity mohla vytvořit plně inicializované DTO, které je dále předáno beaně. Ta dále připraví data pro prezentaci uživateli.

Na obrázku 5.2 je znázorněná výpočetní operace modelové vrstvy. Uživatel, v našem případě pokaždé Admin, vyšle požadavek k zahájení výpočtu. Beana ho předá do výpočetní servisní třídy, ta si postupně z databáze vytáhne všechny potřebné entity. Nad nimi provede požadovaný výpočet a po jeho skončení provede vyžadované operace nad databází, například uloží změněné záznamy, nebo nové entity.



Obrázek 5.2: Provádění výpočtů

5.2 Spring

Aplikace pro pospojování jednotlivých vrstev hojně využívá Spring Frameworku, jehož hlavní funkcí je zajištění a poskytnutí Inversion Of Control principu. V jakékoliv třídě v naší aplikaci můžeme pomocí anotace `@autowired` injektovat jakoukoliv jinou třídu, která je zaregistrována jako komponenta naší aplikace. Například do beany injektujeme servisní objekty modelové vrstvy.

5.3 Adresářová struktura

Adresářová struktura znázorněná v obrázku 5.3 kopíruje architektonické rozvrstvení aplikace. Tak aby se při počtu větším 260ti zdrojových souborů neztrácela orientace.

```
main/
├── java/
│   └── cvut/
│       └── fel/
│           └── hodnoceni/
│               ├── bb/ ... Beany zajišťující komunikaci s
│               │           uživatelem.
│               ├── bo/ ... Zde se nacházejí business objekty
│               │           zajišťující databázovou persistenci.
│               ├── dao/ ... Úschovna data access objektů.
│               ├── dto/ ... Úschovna data transfer objektů.
│               ├── exceptions/ ... Pro celo-aplikační generické výjimky.
│               ├── helpers/ ... Pro pomocné třídy, které kontextuálně
│               │                nepatří do žádného balíčku.
│               ├── jsefa/ ... Úschovna tříd reprezentujících CSV
│               │           objekty.
│               ├── KOSapi/ ... Klientská knihovna pro přístup ke
│               │            KOSapi.
│               ├── oauth/ ... Zde se nachází pomocné třídy pro OAuth,
│               │           především úschovna access tokenu.
│               ├── service/ ... Obsahuje celou servisní vrstvu aplikace.
│               └── VVVS/ ... Klientská knihovna pro přístup k
│                           VVVSapi.
├── webapp/ ... Veřejně přístupná část aplikace
│              obsahující .xhtml šablony.
│   ├── admin/ ... Adresář administrace.
│   ├── efektivita/ ... Adresář efektivity výuky.
│   ├── resources/ ... Grafické a ostatní zdroje aplikace.
│   │   ├── css/
│   │   └── img/
│   └── templates/ ... Úschova kompozitních šablon.
```

Obrázek 5.3: Adresářová struktura kódu

V případě podadresáře `java/` jsou všechny adresáře brány jako Java balíčky.

5.4 Konfigurace

Součástí aplikace jsou dva potencionálně uživatelsky modifikovatelné konfigurační soubory. Ve výsledném WAR archivu se nacházejí v adresáři `/WEB-INF/properties/`. Jedná se o textové soubory klasického konfiguračního formátu `key.subkey=value`.

jdbc.properites

- `jdbc.driverClassName=org.postgresql.Driver` – určuje, který databázový ovladač se má použít, pro snadnou změnu databáze.

- `jdbc.url=jdbc:postgresql://localhost:5432/hodnoceni` – nastavuje cestu a port, na které je dostupný databázový server. Je možné zvolit i server, který se nenachází na lokálním stroji, za předpokladu že je do něj povolený přístup z venčí, případně z omezeného rozsahu IP adres. Na tomto příkladu je vidět že aplikace používá databázi na lokálním serveru, běžící na portu 5432 s názvem hodnoceni.
- `jdbc.username` – přihlašovací jméno ke zvolené databázi. Uživatel by měl být vlastníkem databáze a mít plný přístup ke všem operacím týkajících se dané databáze, pokud by se tak nestalo nemusí aplikace správně fungovat. Hibernate si neporadí se špatnými přístupovými právy a uživatel dostane výjimku.
- `jdbc.password` – heslo ke zvolené databázi.

log4j.properties

V aplikaci je použit logovací systém log4j, který je plně konfigurovatelný pomocí souboru log4j.properties. Množství konfigurací log4j aplikovatelných na náš systém je nemožné popsat jako součást této práce práce, proto odkazuji na dokumentaci systému log4j.

Aplikace v základním nastavení vypisuje do konzole, která je následně odchytávána a zapisována do logovacího souboru serveru Apache Tomcat. Následuje několik konfiguračních hodnot ve formátu `key.subkey=value`, kde values jsou úrovně důležitosti výpisu, od vypnutého stavu `OFF`, přes `ERROR`, `INFO` a `DEBUG` až do vyčerpávajícího výčtu veškerého dění v aplikaci `TRACE`.

- `log4j.rootLogger=trace, CONSOLE` – nastavení hlavního loggeru, v debug provozu je vhodné použít `TRACE`, který zobrazí i vstupy a výstupy z jednotlivých metod.
- `log4j.logger.org.hibernate.engine.transaction` – Zprávy o Hibernate transakcích provedených nad databází.
- `log4j.logger.org.hibernate.engine.jdbc.internal` – Interní stav JDBC konektoru a jednotlivých databázových operací na hlubší úrovni.
- `log4j.logger.org.hibernate.hql` – Popisuje překlad jazyka HQL, se syntaktickým stromem, na kterém je vidět efektivita zapsaného dotazu.
- `log4j.logger.org.apache.http` – Umožňuje debugovat stav jednotlivých HTTP spojení, užitečné například při problémech se službami KOSapi a VVVSapi.

Ve drtivé většině produkčních případů je vhodné nastavit `rootLogger` na `INFO` a všechny ostatní hodnoty na `OFF`

Kapitola 6

Nasazení a vývoj

Pro aplikaci byl vyhrazen virtuální server spravovaný VIC ČVUT. Na serveru je nainstalován operační systém GNU/Linux distribuce Debian 7.1.
Používaná prostředí pro poskytování aplikace:

- Apache Tomcat/7.0.28[19] server slouží pro hostování JAR souboru aplikace.
- Apache/2.2.22[1] spolu s nainstalovaným pluginem pro podporu přihlašovacího systému Shibboleth, slouží Apache server jako autentizační proxy server pro Tomcat.
- PostgreSQL 9.1.9[16]

6.1 Assembla

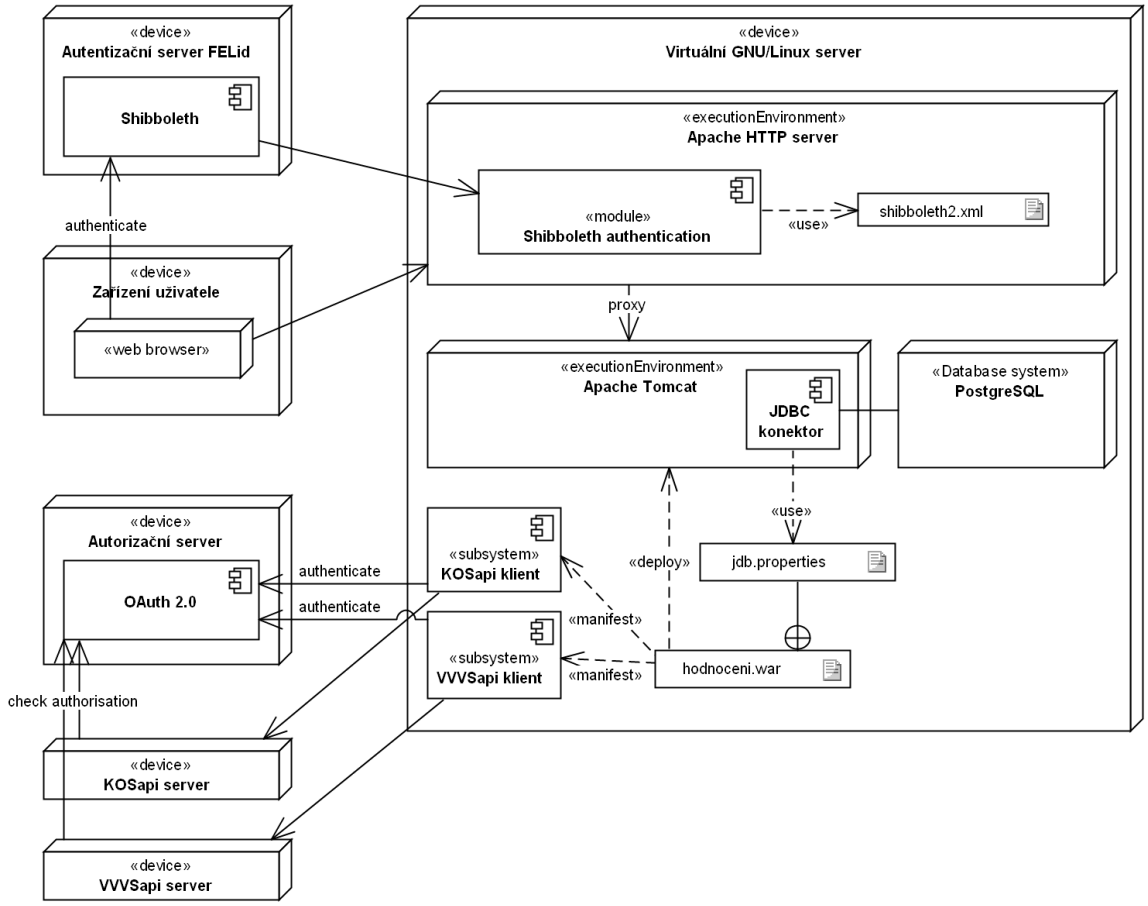
Vývoj probíhá na systému Assembla[3]. Assembla je typický projektový nástroj sloužící ke správě jakéhokoliv projektu, ovšem je orientován na projekty softwarové. Mimo jiné nabízí podporu pro verzovací systém GIT[6], který používáme ke správě a vývoji zdrojového kódu.
Na Assemble je vytvořen veřejně dostupný projekt[2] pro aplikaci Hodnocení.

Tickety

Pro vývoj nové funkcionality a řešení jakýchkoliv změn a požadavků na aplikaci je vytvořen jeden nebo více ticketů v závislosti na složitosti daného úkolu. Ticket se nejčastěji používá k uchovávání poznámek o průběhu práce na úkolu a na sledování času, který člen týmu nad úkolem strávil.

Podpora

V rámci standardní funkcionality Assembly je k dispozici veřejný systém pro přidávání a sledování anonymních ticketů podpory. Kam nám kdokoliv kdo má přístup k aplikaci může napsat libovolnou připomínku nebo návrh. V době psaní této práce je na aplikaci vystaven odkaz na tento systém. Aplikace má denně nenulovou návštěvnost, ale dosud jsme nezaznamenali žádný ticket podpory ze strany uživatelů.



Obrázek 6.1: Diagram nasazení

6.2 GIT

Pro správu zdrojového kódu celého projektu Hodnocení je využíván verzovací systém GIT, s jehož ovládáním a funkcionalitou byli všichni členové týmu seznámeni dlouho před tím než začal vývoj projektu. Práce v GITu je velice intuitivní a rychlá.

Pro projekt byla definována pevná struktura větví a jednotlivých verzí aplikace.

Větev dev

Větev `dev` je vývojářská větev. V době počtu lidí v týmu rovnajícímu se jedné, používám `dev` větev jako hlavní vývojovou větev, do které zavádím veškeré požadované změny ucelené do jednotlivých commitů. V případě paralelního vývoje několika ticketů zároveň občas vytvářím lokální větve založené na `dev`, které pak do `dev` slučuji.

Pokud se tým přistupující k repozitáři rozroste, bude `dev` změněna na společnou vývojovou větev ve které bude ucelená testovatelná funkcionalita. Jednotliví vývojáři pak budou vytvářet lokální větve založené na `dev` a v nich bude probíhat samostatný vývoj.

Větev stable

Větev `stable` je mezistupeň určený k uchovávání plně otestované funkcionality. Zde se nachází nasaditelné verze aplikace, které mohou obsahovat novou funkcionalitu.

Větev master

Větev `master` je určena k uchovávání stabilních, plně otestovaných a nasazených verzí aplikace. Jednotlivé verze jsou označovány tagy s verzovacím číslem.

Pokud se na nějaké verzi na této větvi vyskytne chyba, kterou je nutné okamžitě opravit, pak si vývojář vezme lokální větev z `master`, chybu opraví a sloučí zpět s `master` jako novou verzi.

Verzovací číslo

Verzovací číslo se skládá z prefixu písmene 'v' a ze tří číslic.

- První číslice označuje velké vydání aplikace. Momentálně nejbližší plánované přečíslování na verzi v1.0.0 je nasazení do ostrého provozu ke konci poslední iterace grantu.
- Druhá číslice označuje vydání aplikace s novou funkcionalitou. Funkcionalitou je myšleno jakýkoliv nový větší modul, který vyžaduje nový import a výpočet dat. Posledním příkladem byla efektivita výuky.
- Třetí číslice označuje bugfix nebo malou úpravu funkcionality.

V době psaní této práce byla aktuální verze v0.3.2. Jakékoliv další rozšíření verzovacího čísla je vyhrazeno ale není definováno.

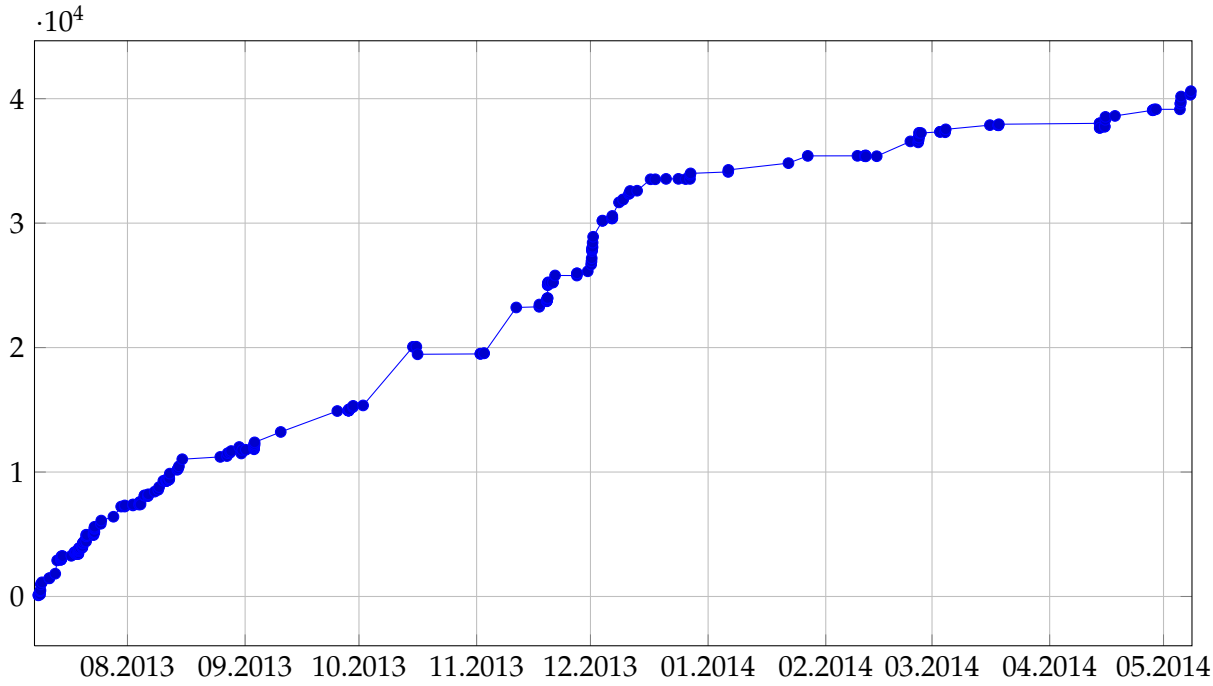
6.3 Statistiky projektu

6.3.1 Odpracované hodiny

V systému Assembla byly zaznamenávány odpracované hodiny na projektu. Od samého začátku jsem prací na aplikaci strávil 350 hodin.

6.3.2 Počet řádků kódu

Počet řádků kódu je díky využití verzovacího systému Git jednoduché zjistit. V grafu 6.2 je znázorněn vývoj počtu řádků kódu za celou dobu repozitáře.



Obrázek 6.2: Počet řádků kódu v závislosti na čase

Kapitola 7

Plán testování

7.1 Unit testování

Unit testování bude prováděno pomocí frameworku JUnit a Mockito. Pro složitější testování budou využity mock objekty.

Bude testováno

- Funkcionalita výpočet pedagogického výkonu
  - Import z KOSapi – Pomocí mock KOSapi otestovat správné ukládání dat.
  - Výpočet
    - Výpočet přednášek
    - Výpočet cvičení
    - Výpočet zkoušení
    - Výpočet Bakalářských a Diplomových prací
    - Výpočet projektových předmětů
    - Slučování paralelek
- Funkcionalita efektivity výuky

Nebude testováno

- Funkcionalita osobní hodnocení – alespoň do té doby než o ní někdo projeví zájem
- DAO objekty využívající Hibernate – objekty jsou zatím jednoduché a přímočaré.
- Importy z externích souborů
- KOSapi

7.2 Integrační testování

Integrační testování v našem případě znamená ověření celkové správnosti výsledků výpočtů funkcionalit. Testy budou spouštěny na živých datech z externích zdrojů (KOSapi a importy). Budou vytipovány případy pro něž bude přesně stanoven správný výsledek a porovnán s výsledkem aplikace.

- Funkcionalita výpočet pedagogického výkonu
- Funkcionalita efektivity výuky

7.3 Systémové testování

Do systémového testování bývá obvykle zahrnováno testování uživatelského rozhraní. Naše aplikace v době psaní práce poskytuje tři stránky dat, kde je naprosté minimum uživatelských aktivních prvků. Tedy za předpokladu budoucí jednoduchosti uživatelské rozhraní nebude testováno.

Předpokládané vytížení aplikace z pohledu současných uživatelů je počítáno v jednotkách uživatelů. Tedy provádění zátěžových testů není potřeba. V případě rozšíření aplikace o více aktivní funkcionality nebo reaktivování osobního hodnocení budou muset být navrhnuty a provedeny zátěžové testy.

7.4 Současný stav

V době psaní této práce je implementována malá část unit testů. Rozšiřování testů a naplnění testovacího plánu je součástí pokračování projektu v rámci grantu.

Kapitola 8

KOSapi a VVVSapi

8.1 Seznámení

KOSapi

KOSapi[12] poskytuje aplikační rozhraní v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi KOS. KOSapi poskytuje necitlivé údaje o studentech, předmětech a zaměstnancích. Naše aplikace hojně využívá poskytovaných služeb. V aplikaci je implementována provozní klientská knihovna pro přístup a získávání dat z KOSapi.

VVVSapi

VVVSapi[21] poskytuje aplikační rozhraní v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi VVVS. Tato databáze obsahuje všechny informace o výzkumu, vědě a vědeckých skupinách na ČVUT. Funkcionalita osobního hodnocení využívala přístupu do VVVSapi, ovšem nové funkcionality počínající Kometou v sobě přístup do VVVSapi neobsahují. V aplikaci je implementována provozní klientská knihovna pro přístup a získávání dat z VVVSapi.

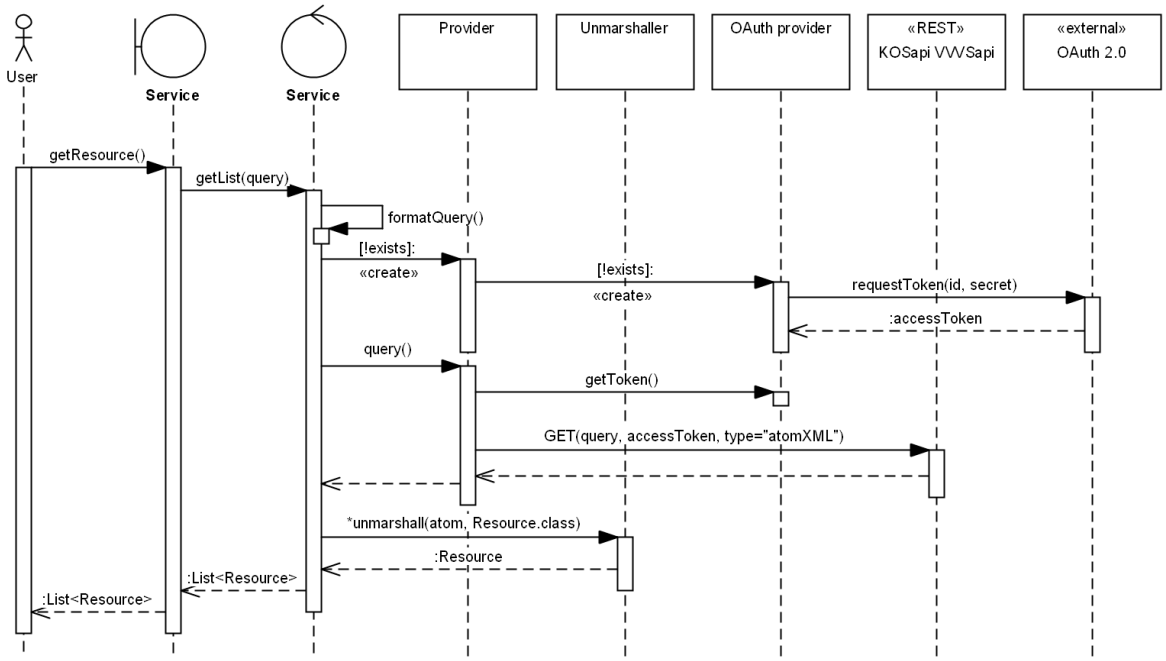
OAuth

KOSapi i VVVSapi jsou přístupné pomocí centrálního autorizačního serveru využívajícího technologie OAuth[13]. Server je provozovaný fakultou informačních technologií[14]. OAuth zabezpečuje přístupy k RESTovým zdrojům KOSapi a VVVSapi, pro získání přístupu je nutné mít správný klíč. Klíč jsme získali ze správce klíčů pro autorizační server[15].

8.2 Architektura klientské knihovny

Jelikož jsou KOSapi a VVVSapi ve své architektuře identické a používají stejný OAuth autorizační server, je možné je popsat současně. Klientské knihovny pomocí kterých v naší aplikaci získáváme data jsou z venčí prezentovány *uživatelsky* přístupnou servisní

vrstvou a množstvím Plain Old Java Objektů (POJO), které reprezentují jednotlivé zdroje poskytované KOSapi a VVVSapi. Servisní vrstva poskytuje `public` metody pro získání některých aplikačně potřebných zdrojů, pomocí specializovaných dotazů.



Obrázek 8.1: Sekvenční diagram znázorňující získání zdroje

Jak je patrné z obrázku 8.1, existuje také vnitřní servisní vrstva, která se stará o samotné vykonání požadavku. Z api je možné vytáhnout jeden nebo `List` zdrojů, vrstva se stará o správné formátování dotazu, přidání určení jazyka a validní získání většího množství dat. Api mají limit na maximální možný počet resource objektů, které je možné v jednom dotazu získat. Vrstva zajišťuje správnou aplikaci parametrů `offset` a `limit` v několika dotazech.

Součástí funkce servisní vrstvy je také inicializace autorizačního tokenu. Při prvním spuštění se vyvolají Providery a provede se autentizace k externímu autorizačnímu serveru OAuth, ze kterého získáme autorizační token. Tento token je pak posílán na RESTovou URI odpovídající požadovanému dotazu. Z api se vrací ATOM XML, které je pak následně unmarshallováno do tříd odpovídajících uživatelskému požadavku.

8.3 Nedostatky KOSapi

Následuje výčet nedostatků na které jsme narazili v průběhu vývoje aplikace. Některé návrhy jsou maličkosti, které by bylo dobré mít a jiné jsou větší nedostatky případě chyby KOSapi, které v době psaní této práce nebyly opraveny.

- Některé předměty mají špatně uvedený jazyk výuky. Viz ticket 107[2].
- Vyučující jsou špatně řazeni pod katedry. Viz ticket 134.

- KOSapi indukuje špatný počet studentů a studenti musejí být počítání *manuálně* přes jejich výčet z KOSapi. Viz ticket 135.
- Předměty s neregulárním kódem nejsou v KOSapi dostupné. Viz tickety 136 a 137.
- U předmětu by měla být uvedena etapa studia. Viz ticket 102.
- Chybí informace o prerekvizitách předmětů. Viz ticket 102.
- Pokud je u přednášky nebo cvičení více učitelů, tak by se mel ukazovat i poměr jejich výuky. Nejedná se o citlivá data a v KOSu jsou. Viz ticket 102.
- KOSapi neposkytuje pohled do minulosti, není tedy například možné zjistit studentův ročník v průběhu studia. Viz ticket 31.
- KOSapi neparsuje rozsah studia, který v minulosti nebyl vždy parsovatelný, ale nyní už je jeho formát dostatečně standardizovaný na správné rozparsování.

Kapitola 9

Závěr

Výsledkem této bakalářské práce je funkční aplikace Hodnocení, která je momentálně v ostrém provozu a využívána pro interní účely fakulty elektrotechnické. Podařilo se nám splnit všechny body zadání, aplikace počítá započitatelné hodiny pro vyučující s použitím dat ze služby KOSapi a několika exportů systému KOS. Aplikace také provádí výpočet efektivity výuky. Aplikaci od začátku roku navštívilo přes 40 unikátních uživatelů.

9.1 Budoucí vývoj projektu

Projekt Hodnocení je na dobré cestě stát se součástí softwarové výbavy FEL ČVUT. Během následujících iterací definovaných v grantu[7] budou opraveny stávající chyby, implementovány nové požadavky a projekt bude důsledně dotestován. Pracovní verze klientské knihovny přístupu ke KOSapi a VVVSapi, kterou jsme v rámci řešení vytvořili, může posloužit jako základ vzorové implementace klientské části systémů KOSapi a VVVSapi pro jazyk Java. Těším se na pokračující spolupráci s Ing. Martinem Komárkem.

Literatura

[1] *Apache HTTP server* [online]. Dostupné z: <https://httpd.apache.org/>.

[2] *Projekt Hodnocení v Assembla* [online]. [cit. 20.4.2014]. Dostupné z: <https://www.assembla.com/spaces/hodnoceni-pracovniku-fel>.

[3] *Assembla* [online]. [cit. 20.4.2014]. Dostupné z: <https://www.assembla.com/home>.

[4] *FELid* [online]. [cit. 6. 4. 2014]. Dostupné z: <https://wiki.fel.cvut.cz/net/felid/about>.

[5] FOWLER, M. *Patterns of Enterprise Application Architecture*. 1. : Addison-Wesley Professional, 1st edition, 2002.

[6] *GIT Source Code Management* [online]. Dostupné z: <http://git-scm.com/>.

[7] *Přijaté projekty RPMT* [online]. Dostupné z: <http://www.fel.cvut.cz/aktuality/vyzva-RPMT-prijate.html>.

[8] *Hibernate Query Language* [online]. Dostupné z: <http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html>.

[9] *Java Server Faces* [online]. Dostupné z: <https://javaserverfaces.java.net/>.

[10] *KOMETA2, Metodika pro stanovení pedagogického výkonu kateder, materiálové náročnosti studia a pro rozdělování finančních prostředků za výuku na katedry* [online]. [cit. 6. 4. 2014]. Dostupné z: <https://wiki.fel.cvut.cz/kometa2/index>.

[11] *Aplikace implementující Kometu2* [online]. [cit. 20.4.2014]. Dostupné z: <https://zp.feld.cvut.cz/kometa2.3/index1.php>.

[12] *KOSapi* [online]. [cit. 6. 4. 2014]. Dostupné z: <https://kosapi.fit.cvut.cz/projects/kosapi/wiki>.

[13] *OAuth protokol* [online]. Dostupné z: <http://oauth.net/2/>.

[14] *Autorizační server OAuth 2.0* [online]. Dostupné z: <https://rozvoj.fit.cvut.cz/Main/oauth2>.

[15] *Správce autorizačních přístupů k OAuth* [online]. Dostupné z: <https://auth.fit.cvut.cz/manager/index.jsf>.

[16] *PostgreSQL* [online]. Dostupné z: <http://www.postgresql.org/>.

[17] *Primefaces – Knihovna komponent pro JSF* [online]. Dostupné z: <http://primefaces.org/>.

[18] *Rejstřík informací o výsledcích* [online]. [cit. 7. 4. 2014]. Dostupné z: <http://www.vyzkum.cz/FrontClanek.aspx?idsekce=986>.

[19] *Apache Tomcat* [online]. Dostupné z: <http://tomcat.apache.org/>.

[20] *Aplikace VVVS* [online]. [cit. 7. 4. 2014]. Dostupné z: <http://www.vvvs.cvut.cz/zakladni_info.html>.

[21] *VVVSapi* [online]. [cit. 6. 4. 2014]. Dostupné z: <https://kosapi.fit.cvut.cz/projects/vvvsapi/wiki>.

Příloha A

Obsah přiloženého DVD

A.1 Adresářová struktura

Adresářová struktura znázorněná v obrázku 5.3 kopíruje architektonické rozvrstvení aplikace. Tak aby se při počtu větším 260ti zdrojových souborů neztrácela orientace.

```
/
├── jar/
│   └── hodnoceni.jar ... Soubor aplikace připravený k nasazení na
│                         server Apache Tomcat.
├── src/ ... Složka obsahující zdrojové soubory aplikace.
├── text/
│   ├── latex/ ... Zdrojový kód této práce v LaTeXu.
│   └── pdf/ ... Tisknutelná forma této práce v pdf.
├── virtual/ ... Složka obsahující virtuální disk s
│               nainstalovanou aplikací spustitelný v
│               programu VirtualBox.
├── INSTALL ... Instalační manuál.
└── README ... Textový soubor obsahující obecné informace.
```

Obrázek A.1: Adresářová struktura přiloženého DVD