Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Obecný dotazníkový systém
Diplomová práce

Autor: Lukáš Kmoníček

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Janečka

Hradec Králové listopad 2014

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Lukáš Kmoníček

Poděkování:

Rád bych tímto poděkoval vedoucímu práce Ing. Pavlu Janečkovi za pomoc, ochotu, věcné rady a věnovaný čas mé diplomové práci.

Anotace

Diplomová práce na téma „Obecný dotazníkový systém“ se zabývá tvorbou webové aplikace pro správu dotazníkových výzkumů. Na základě analýzy a zhodnocení vybraných existujících dotazníkových systémů je proveden návrh a implementace vlastního řešení dotazníkového systému. Systém umožňuje přihlášenému uživateli vytvořit dotazníkový formulář obsahující libovolný počet otázek. V každé z otázek uživatel definuje kromě vlastního znění otázky také typ odpovědi, popřípadě její konkrétní možnosti. Aplikace podporuje šest různých typů odpovědí. Odkaz na vyplnění dotazníku jeho tvůrce odešle hromadně všem přiřazeným potencionálním respondentům prostřednictvím emailové zprávy vygenerované systémem. Po vyplnění dotazníku respondenty lze získaná data vyhodnotit a vyexportovat do souboru. Dotazníkový systém byl vytvářen s ohledem na budoucí nasazení na Univerzitě Hradec Králové.

Annotation

Title: Universal questionnaire system

The diploma thesis concerns with creation of web application for administration of questionnaire researches. Proposal and implementation of own solution of questionnaire system is done based on analysis and evaluation of selected existing questionnaire systems. The system enables a signed-in user to create questionnaire forms containing unlimited number of questions. In each question the user defines the wording of the question and also type of answer or other particular options. The application supports six different types of answers. A link for the questionnaire can be sent collectively by its user to all assigned potential respondents via email message which is automatically generated by the system. The gathered data can be evaluated and exported into a file after completion of the questionnaire by respondents. The questionnaire system was created with focus on future deployment on University of Hradec Králové.

Obsah

1 Úvod

Dotazníky a dotazníkové výzkumy jsou potřebným nástrojem například v oblasti průzkumu veřejného mínění, pro marketingové firmy nebo studenty, kteří zpracovávají své práce založené na sběru dat z určité části společnosti. Dotazníkové systémy jsou možností, jak dotazníkové výzkumy realizovat.

Cílem diplomové práce je analyzovat a zhodnotit vybrané dotazníkové systémy a na základě získaných poznatků provést návrh a implementaci vlastního dotazníkového řešení. Místem nasazení nově vzniklého dotazníkového systému by do budoucna měla být Univerzita Hradec Králové, kde by měl sloužit potřebám studentů.

Dotazníkové systémy, zvolené k vyhodnocení, jsou dostupné na internetu a jejich základní koncepce se značně odlišují. Prvním systémem je Survey Simply, který je ve zdarma dostupné variantě značně omezen. Pokud by měl být zvolen jako možné řešení, musela by být zaplacena komerční verze produktu. Druhý vyhodnocovaný systém, Dokumenty Google, je zdarma dostupný pro každého, kdo má zřízený účet u společnosti Google. V kapitole pojednávající o analýze těchto systémů jsou popsány obvyklé i nadstandardní funkcionality systémů spravujících dotazníkové výzkumy a dále jsou vybrané existující systémy porovnány z hlediska použitelnosti dostupných funkcí. Porovnání systémů je realizováno z pohledů dvou odlišných uživatelů. Jedním je náročný uživatel webových aplikací a druhým méně zkušený uživatel internetu.

Následující kapitola se zabývá stručnou specifikací nově vznikajícího řešení dotazníkového systému. Kromě výčtu technologií, které mají být využity k tvorbě systému, jsou popsány jednotlivé části, které by měl systém obsahovat. Specifikace systému je z velké části inspirována dotazníkovým systémem od společnosti Google.

Následuje analýza a návrh samotného systému, ve které je zahrnut datový model, diagram případů užití, detailnější popis částí systému a jejich funkčních specifik.

Samotná implementace je realizována na platformě Java Enterprise Edition s využitím Spring Frameworku a databázového serveru MySQL. Kapitola popisující využití možností použitých technologií a jednotlivé programovací techniky zahrnuje ukázky zdrojových kódů a jejich popis.

2 Analýza existujících dotazníkových systémů

K tvorbě jakékoli aplikace je třeba analýza a návrh, na jejichž základě lze teprve implementovat samotný systém. Nezáleží na tom, zda je systémem myšlena desktopová aplikace, internetový obchod či informační systém pro mezinárodní korporaci, ale v první řadě je třeba zjistit a popsat požadavky zákazníka na systém, hlavně z pohledu funkcionalit systému. Dalším pohledem pak mohou být požadavky na grafické zpracování, které většinou upoutá na první pohled pozornost uživatele.

Jedním z výstupů diplomové práce by měl být dotazníkový systém, který není směřován žádnému konkrétnímu zákazníkovi schopnému specifikovat své požadavky na systém, ale měl by sloužit studentům všech oborů na Univerzitě Hradec Králové. V takovémto případě je vhodné seznámit se s existujícími systémy určenými pro podobné účely, sepsat jejich funkcionality a ohodnotit celkové zpracování, na základě čehož lze specifikovat funkční požadavky na nově vznikající systém. Z tohoto plyne, že bude z nemalé části inspirován zmíněnými existujícími systémy, které mají však značná omezení. Tato omezení by měla být v rámci této práce odbourána, přičemž by se měly využít pouze použitelnější či z pohledu uživatele přátelštější funkcionality.

Pro tento účel byly vybrány nejdostupnější dotazníkové systémy na webu a níže jsou rozebrány jejich funkční vlastnosti z pohledu uživatele.

Všechny zkoumané dotazníkové systémy jsou pro všechny běžné uživatele webových aplikací velmi přívětivým nástrojem pro správu osobních či pracovních dotazníkových výzkumů. A to nejen pro jejich tvorbu, ale hlavně pro určení potenciálních respondentů jejich výzkumu, pro sběr a vyhodnocení dat, vygenerování statistik výzkumu, popř. vystavení výsledků na webu.

2.1 Survey Simply (http://www.surveysimply.cz)

Prvním ze zkoumaných dostupných webových dotazníkových systémů je Survey Simply. Jeho vizuální podoba rozhraní pro správu dotazníků připomíná jednodušší aplikaci určenou pro mladistvé. Tento dojem vytvářejí hlavně ikony akcí. Dle počtu a významu parametrů jednotlivých nastavení vytvářeného dotazníku by však tento systém preferovali spíše pokročilí uživatelé webových aplikací.

Kromě základních funkcí dotazníkového systému, jakými jsou například vytvoření dotazníku, správa existujícího dotazníku, správa uživatelů, správa účastníků výzkumu,

zobrazení statistik odpovědí nebo možnost vyplnění samotného dotazníku přizvanými účastníky, obsahuje tento systém nadstandardní funkce, které například umožňují zkopírovat již existující dotazník, zkontrolovat integritu dat, zálohovat celou databázi do souboru, zobrazit seznam částečných odpovědí, importovat či exportovat celý dotazník a všechny odpovědi, statistiky odpovědí či záznamy účastníků, anebo vytvořit fiktivní záznamy účastníků.

Možnost využití takto rozsáhlého systému však v dnešní době málokterá společnost poskytuje zdarma. Pokud by chtěl potenciální uživatel i přesto tyto služby zdarma získat, má možnost kromě jiných poskytovatelů dotazníkových systémů i u Survey Simply.

Varianta zdarma poskytovaných služeb má zajisté určitá omezení, která nedávají mnoho prostoru pro potřebné kvalitní zpracování výzkumu. První hranicí je počet respondentů, který nesmí překročit 100 účastníků. Dalším problémem je doba, po kterou může uživatel k dotazníku přistupovat, ta činí 2 měsíce od registrace do systému. Mimo to je zde omezena telefonická i emailová podpora.

V případě, že je uživatel ochoten za poskytované služby zaplatit, má k dispozici širokou nabídku služeb, která obsahuje kromě možnosti vytvořit si na webu vlastní dotazník i kontrolu a korekturu dotazníků, převod dodaných tištěných dotazníků do elektronické podoby, vytvoření prezentace výstupů výzkumu a další podobnou podporu individuálního výzkumu. Společnost podporuje také datamining, vícerozměrné analýzy nebo statistické metody. Pokud však zůstaneme u samotné tvorby a využití dotazníku, vyskytuje se v nabídce několik různých cen, závisících na počtu respondentů výzkumu a pohybujících se řádově v tisících korun českých.

2.2 Dokumenty Google (http://docs.google.com)

Společnost Google nabízí svým uživatelům online správu dokumentů od jednoduchých textových dokumentů s obrázky přes tabulky, prezentace, nákresy objektů až po formuláře. Zmíněný nástroj pro správu formulářů v kombinaci se správou tabulek představuje jednoduchý, ale přesto efektivní dotazníkový systém.

Jednoduchý hlavně z toho důvodu, že na uživatelské rozhraní pro tvorbu dotazníku, resp. přímo pro tvorbu otázek, se uživatel dostane ze svého účtu dvěma kliknutími. Přidání či změna pořadí otázek jsou taktéž snadno proveditelné a intuitivní. Jedním z ovládacích tlačítek pro úpravu jednotlivých otázek lze vytvořit duplikát kompletní aktuální otázky včetně typu a obsahu jejích odpovědí. Formulář dotazníku lze vytvořit také například

z některé ze šablon, které jsou zhotoveny různými uživateli Google dokumentů a těmito uživateli veřejně sdíleny v galerii šablon na stránkách této společnosti.

Správa účastníků výzkumu je u této společnosti řešena přes unikátní emailovou adresu uživatelů, kde není třeba vůbec evidovat iniciály ani další údaje uživatelů. Správce výzkumu může zaslat pozvánku k účasti na výzkumu vybraným uživatelům ze svého adresáře anebo na jakýkoli jiný existující e-mail. Součástí této emailové zprávy je URL s přístupem k požadovanému dotazníku a zároveň může obsahovat i samotný dotazník.

Na stejném principu je založena i evidence spolupracovníků. K zaslání pozvánky ke spolupráci na tvorbě a zpracování výzkumu stačí opět pouze emailová adresa kolegy, který následně obdrží informaci o sdílení konkrétního výzkumu s odkazem na formulář s rozpracovaným výzkumem. Pro přístup k tomuto online formuláři musí mít však dotyčný zřízený Google účet spjatý s touto emailovou adresou. Správce může u každého výzkumu nastavit jeho spolupracovníkům oprávnění buď pouze pro čtení anebo pro zápis, přičemž ve druhém případě lze dále nastavit spolupracovníkům práva na přidávání a odebírání dalších uživatelů a na změnu viditelnosti dokumentu. Správce výzkumu, ve webové aplikaci pojmenovaný jako vlastník dokumentu, má však možnost spolupracovníkům povolit pouhou editaci dokumentu bez dalších výše zmíněných oprávnění.

2.3 Porovnání dotazníkových systémů

Pro výběr lepšího či pro uživatele vhodnějšího řešení nějakého typu systému může posloužit srovnání těchto systémů, včetně jejich použitelnosti, zpracování a porovnání jednotlivých vlastností a funkcí, které rozdílně implementované systémy poskytují.

Porovnání existujících dotazníkových systémů lze provést pomocí tabulky vážených kritérií, v níž lze nejlépe porovnat kvantitativní kritéria. Takováto kritéria však nelze v případě zkoumaných systémů jednoduše definovat. Proto jsou pro porovnání těchto systémů volena kritéria kvalitativní, která jsou však převáděna dle dostupnosti nebo stupně použitelnosti na číselnou reprezentaci uvedenou v procentech. Téměř všechna porovnávaná kritéria jsou rozdělena do sekcí dle logického zařazení do systému. Každé sekci je přidělena váha, jež je opět reprezentována procenty, v tomto případě však na základě důležitosti v rozhodování.

Číselné ohodnocení kvalitativních kritérií však může být dosti subjektivně ovlivněno. To z toho důvodu, že jednotliví uživatelé preferují rozdílné funkcionality

systému, a to v tom smyslu, že pro jednoho uživatele může mít jedna konkrétní funkcionalita stěžejní význam pro využívání systému, zatímco druhý uživatel nemusí o této funkcionalitě ani vědět, jelikož ji vůbec nevyužívá, nebo ji při své práci nepotřebuje. I v případě, že oba tito uživatelé zmíněnou funkcionalitu využívají, mohou na ni mít rozdílný názor z pohledu použitelnosti a zpracování. Pak se bude ohodnocení takovéto funkcionality jednotlivými uživateli značně lišit.

Kvůli subjektivním pocitům jednotlivých uživatelů o využitelnosti systémů je zpracována tabulka kritérií z více pohledů (z pohledů uživatelů s rozdílnými preferencemi).

2.3.1 Porovnání z pohledu náročného uživatele

Definujme náročného uživatele jako uživatele jakékoli webové aplikace, který již má značné zkušenosti s využíváním široké škály takovýchto aplikací, rozumí alespoň do jisté míry webovým technologiím a preferuje velké spektrum nastavení dle svých potřeb. Z pohledu popsaného uživatele lze ohodnotit jednotlivá kritéria, dle kterých se porovnávají jednotlivé systémy. Ohodnocení se provádí tak, že složitějším a nastavitelnějším funkcionalitám jsou přiřazeny vyšší hodnoty a naopak funkcionality jednodušší povahy, které nelze nijak konfigurovat, jsou ohodnoceny nižším číselným vyjádřením v procentech.

V následující kriteriální tabulce jsou již konkrétně shrnuty priority náročného uživatele v porovnání dotazníkových systémů. Výsledné hodnoty s razantním rozdílem reprezentují fakt, že dle zvolených preferencí by dal uživatel přednost systému Survey Simply.

Administrace			SurveySimply.cz			docs.google.com		
10%	Vytvoření dotazníku		70%	7%		67%	7%	
		import / kopie	80%		.lss nebo .csv import, kopie	60%		template
		nastavení před vytvořením	70%		široké možnosti nastavení	50%		dva kliky
		ovládání vytváření otázek	60%		dobré	90%		lepší, modernější
5%	Vytvoření / editace uživatele		60%	3%	klasické + email	50%	3%	email
5%	Vytvoření / editace skupiny		60%	3%	klasické	50%	3%	email
5%	Kontrola integrity dat		100%	5%	ano	0%	0%	ne
5%	Záloha celé databáze do sql souboru		100%	5%	ano	0%	0%	ne
25%	Správa dotazníku		68%	17%		51%	13%	
		aktivace / deaktivace dotazníku	100%		ano	100%		ano (přijímání odpovědí)
		počet práv, které je možné přiřadit uživatelům	90%		12	30%		4
		export průzkumu	90%		lss či xml (qoeXML) soubor	0%		ne
		tisková verze průzkumu	100%		ano	0%		ne
		možnost vložit formulář na své stránky přes odkaz	0%		ne	100%		ano
		možnost sdílení odkazu	60%		pouze email	80%		přes Gmail, Facebook nebo Twitter
		zpracování vrácených emailů	100%		ano	0%		ne
		Možnost vypublikovat formulář jako template na web	0%		ne	100%		dostupnost - veřejné / s odkazem / soukromé
20%	Statistiky odpovědí		72%	14%		100%	20%	
		Filtr	100%		ano	100%		ano
		Výstup do	50%		html, pdf nebo excelu	100%		html, pdf, excelu, openOffice, text, csv
		možnost vypublikovat výsledky výzkumu na webu	0%		ne	100%		ano
		grafy	100%		ano	100%		ano
		export / import	80%		ano	100%		ano, import z .xls, .xlsx, .ods, .csv, .txt, .tsv, .tab
		možnosti úprav, přidání nebo smazání odpovědí v seznamu	100%		ano	100%		ano
5%	Obrazovka pro vstup dat		100%	5%	ano	0%	0%	ne
5%	Seznam částečných odpovědí		100%	5%	ano	0%	0%	ne
15%	Správa účastníků		99%	15%		31%	5%	
		seznam účastníků	100%		ano	100%		ano
		záslání pozvánky účastíkům	100%		ano	100%		ano
		zaslání přípomínky účastníkům	100%		ano	0%		ne
		přidání nového účastníka	90%		ano	80%		ano, stačí pouze email
		vytvoření fiktivních záznamů účastníků	100%		ano	0%		ne
		import seznamu účastníků z csv	100%		ano	0%		ne
		import seznamu účastníků z LDAP dotazu	100%		ano	0%		ne
		export seznamu účastníků do csv	100%		ano	0%		ne
		generování přístupových kódů	100%		ano	0%		ne
				79%			49%	

Tabulka 1 – porovnání systémů z pohledu náročného uživatele

2.3.2 Porovnání z pohledu méně zkušeného uživatele

Uživatel, který nemá tak velké zkušenosti s využíváním webových aplikací, bude spíše volit jednodušší funkcionality, které mají intuitivní ovládání a které není nutné složitě konfigurovat. Takovýto uživatel většinou ani nezná v širokém nastavení systému význam parametrů, které by měl nastavit dle svých představ, proto raději volí tu možnost, kdy je v systému vše přednastaveno a nemusí nic měnit, v tom lepším případě žádné neznámé parametry systému měnit ani nelze.

Administrace			SurveySimply.cz			docs.google.com		
10%	Vytvoření dotazníku		30%	3%		87%	9%	
		import / kopie	20%		.lss nebo .csv import, kopie	70%		template
		nastavení před vytvořením	10%		široké možnosti nastavení	100%		dva kliky
		ovládání vytváření otázek	60%		dobré	90%		lepší, modernější
5%	Vytvoření / editace uživatele		30%	2%	klasické + email	60%	3%	email
5%	Vytvoření / editace skupiny		30%	2%	klasické	60%	3%	email
5%	Kontrola integrity dat		20%	1%	ano	0%	0%	ne
5%	Záloha celé databáze do sql souboru		20%	1%	ano	0%	0%	ne
25%	Správa dotazníku		40%	10%		55%	14%	
		aktivace / deaktivace dotazníku	100%		ano	100%		ano (přijímání odpovědí)
		počet práv, které je možné přiřadit uživatelům	20%		12	90%		4
		export průzkumu	20%		lss či xml (qoeXML) soubor	0%		ne
		tisková verze průzkumu	100%		ano	0%		ne
		možnost vložit formulář na své stránky přes odkaz	0%		ne	100%		ano
		možnost sdílení odkazu	60%		pouze email	80%		přes Gmail, Facebook nebo Twitter
		zpracování vrácených emailů	20%		ano	0%		ne
		Možnost vypublikovat formulář jako template na web	0%		ne	70%		dostupnost - veřejné / s odkazem / soukromé
20%	Statistiky odpovědí		72%	14%		100%	20%	
		Filtr	100%		ano	100%		ano
		Výstup do	50%		html, pdf nebo excelu	100%		html, pdf, excelu, openOffice, text, csv
		možnost vypublikovat výsledky výzkumu na webu	0%		ne	100%		ano
		grafy	100%		ano	100%		ano
		export / import	80%		ano	100%		ano, import z .xls, .xlsx, .ods, .csv, .txt, .tsv, .tab
		možnosti úprav, přidání nebo smazání odpovědí v seznamu	100%		ano	100%		ano
5%	Obrazovka pro vstup dat		20%	1%	ano	0%	0%	ne
5%	Seznam částečných odpovědí		20%	1%	ano	0%	0%	ne
15%	Správa účastníků		41%	6%		31%	5%	
		seznam účastníků	100%		ano	100%		ano
		záslání pozvánky účastíkům	100%		ano	100%		ano
		zaslání přípomínky účastníkům	20%		ano	0%		ne
		přidání nového účastníka	50%		ano	80%		ano, stačí pouze email
		vytvoření fiktivních záznamů účastníků	20%		ano	0%		ne
		import seznamu účastníků z csv	20%		ano	0%		ne
		import seznamu účastníků z LDAP dotazu	20%		ano	0%		ne
		export seznamu účastníků do csv	20%		ano	0%		ne
		generování přístupových kódů	20%		ano	0%		ne
				41%			53%	

Tabulka 2 - porovnání systémů z pohledu méně zkušeného uživatele

3 Specifikace nově vznikajícího řešení

Nově vznikající dotazníkový systém bude koncipován tak, aby všechny jeho funkce mohlo využívat co nejširší spektrum uživatelů. To znamená, že bude vyvíjen s ohledem na méně zkušené uživatele webových aplikací a bude oproštěn od spousty různých nastavení, kterým by nemusela většina uživatelů rozumět.

Ovládání by mělo být zkonstruováno tak, aby bylo pro každého uživatele intuitivní a srozumitelné.

Samotná aplikace bude vyvinuta na platformě Java Enterprise Edition s využitím Spring Frameworku. Vizuální část aplikace bude zkonstruována pomocí JSP stránek s podporou JSTL (Java Server Pages Standard Tag Library) a skriptovacího jazyka JavaScript. Úložiště dat bude představovat databáze MySQL, na kterou bude aplikace napojena přes JPA (Java Persistence Api).

Velkou výhodou všech zmíněných technologií je to, že jsou zdarma dostupné ke stažení a využití pro tvorbu libovolné webové aplikace.

3.1 Specifikace jednotlivých obrazovek

Aplikace bude konstruována z několika obrazovek, které využívají stejné či podobné komponenty a stejné styly pro zobrazení jednotlivých komponent.

3.1.1 Přehled dotazníků

Po přihlášení se uživateli zobrazí úvodní stránka s přehledem všech dotazníků, které vytvořil, a s možností vytvořit dotazník nový. Dále je na této stránce zobrazen seznam dotazníků, na kterých je uživatel uveden jako potencionální respondent, to znamená, že by měl na žádost tvůrce uvedené dotazníky vyplnit svými odpověďmi. A ve třetí sekci této stránky bude k dispozici opět přehled uživatelem vytvořených dotazníků, avšak uživatel zde může dotazníky, resp. odpovědi všech respondentů, vyhodnotit nebo vyexportovat do souboru.

3.1.2 Sestavení dotazníku

Při volbě pro vytvoření či editaci dotazníku z přehledu dotazníků se zobrazí obrazovka s komponentami pro editaci jednotlivých atributů dotazníku. Těmito atributy je myšlen například název a popis dotazníku, názvy a popisy jednotlivých otázek včetně volby jejich typů odpovědí a popisů konkrétních možností odpovědí.

Otázky se na formuláři této obrazovky dají přidávat, upravovat i mazat. Je zde podporována validace všech dostupných položek pro povinné vyplnění.

3.1.3 Přidání respondentů

Po uložení vytvářeného či upravovaného dotazníku je možno přiřadit tomuto dotazníku potencionální respondenty. Ty jsou vybíráni ze seznamu uživatelů načteného z databáze. Následně se dají uloženým respondentům zaslat emailové žádosti o vyplnění tohoto dotazníku.

3.1.4 Vyplnění dotazníku

Na obrazovce s konkrétním dotazníkem k vyplnění se jedná o zobrazení otázek obsažených v dotazníku a jejich vyplnění respondentem. Po uložení formuláře se odpovědi uloží do databáze a v tu chvíli lze dotazník tvůrcem vyhodnotit.

K jediné této stránce lze přistupovat bez přihlášení pouze přes adresu obsahující identifikátor dotazníku.

3.1.5 Vyhodnocení dotazníku

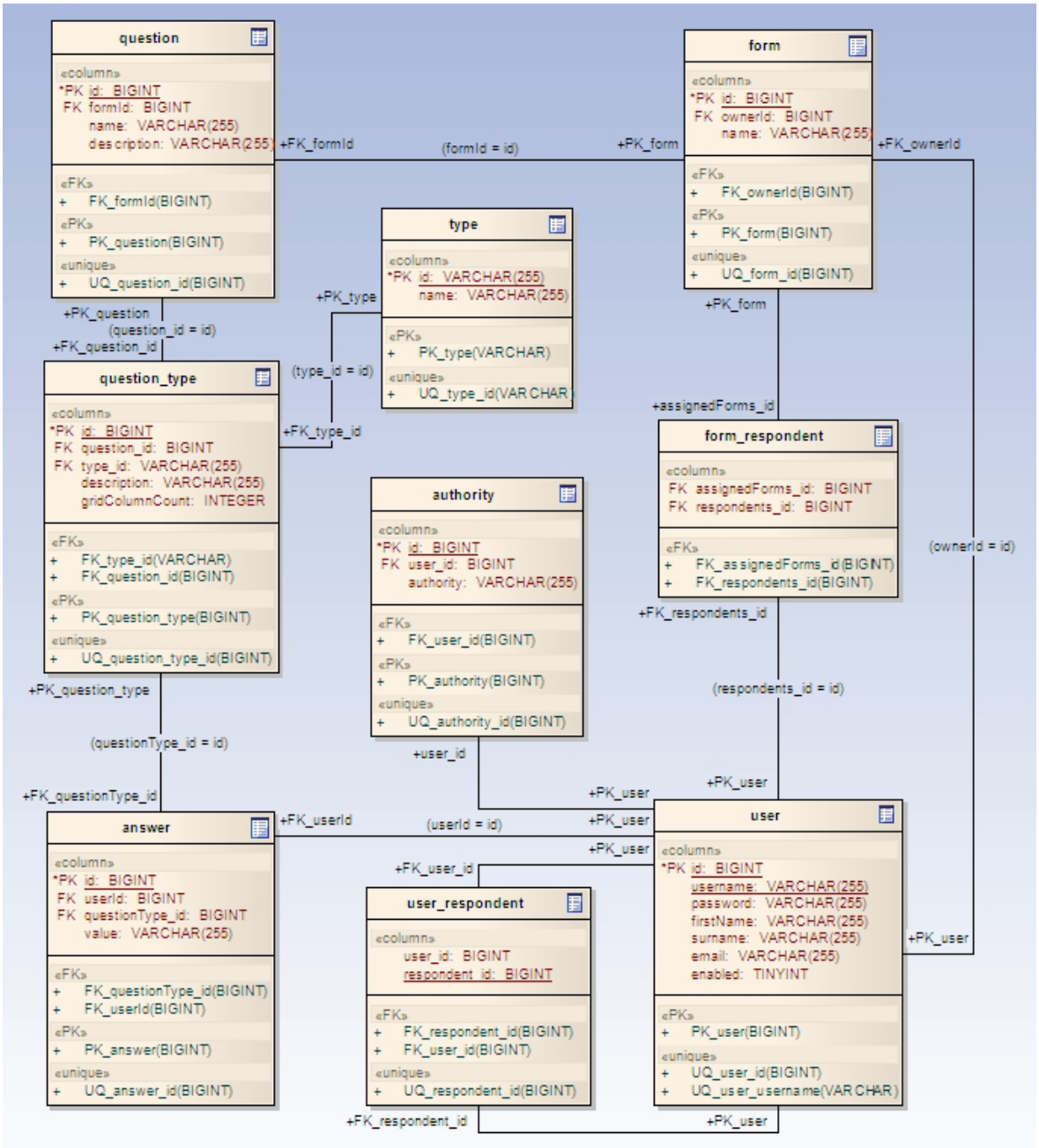
Tato stránka slouží k sumarizaci zodpovězených otázek respondenty, k zobrazení grafů obsahujících data odpovědí a k exportu výsledků do souboru.

4 Analýza a návrh aplikace

4.1 Datový model

Základním stavebním kamenem většiny aplikací je databáze, ve které jsou uložena data, jež daná aplikace využívá. Dále zobrazuje vybraná data uživateli, nebo na základě požadavků uživatele provádí nad daty v databázi změny.

Na následujícím diagramu je popsán návrh databáze vznikajícího systému.



Obrázek 1 – datový model

Schéma databáze je navrženo tak, aby databáze byla schopna uchovávat jednotlivé dotazníky vytvořené konkrétním uživatelem, jednotlivé otázky včetně jejich typů a odpovědí respondentů na dané otázky.

Tabulka *form* slouží k uchovávání informací týkající se dotazníku, tj. jeho názvu a uživateli, který dotazník vytvořil. Pro záznamy uživatelů je zde navržena tabulka *user*, která je sestavena z uživatelského jména a hesla, křestního jména a příjmení uživatele, jeho emailové adresy a příznaku, který říká, zda je uživatel aktivní (povolen). Zároveň je pomocí vazební tabulky *form_respondent* definováno, kteří uživatelé jsou považováni za potencionální respondenty konkrétního dotazníku. Každý dotazník samozřejmě bude obsahovat nějaké otázky, které budou zaznamenávány do tabulky *question*, která obsahuje mimo názvu a popisu otázky také, jakožto cizí klíč, identifikátor dotazníku. Do tabulky *question_type* se budou zapisovat jednotlivé typy odpovědí dané otázky. Například pokud bude otázka obsahovat pět možností odpovědí, kde lze vybrat respondentem odpověď pouze jednu, pak těchto pět možností bude evidováno v této tabulce jako pět záznamů typu „radio_group“, přičemž číselník typů představuje tabulka *type*. Na každou z položek tabulky *question_type* pak může být navázán záznam tabulky *answer*, která zahrnuje všechny odpovědi respondentů.

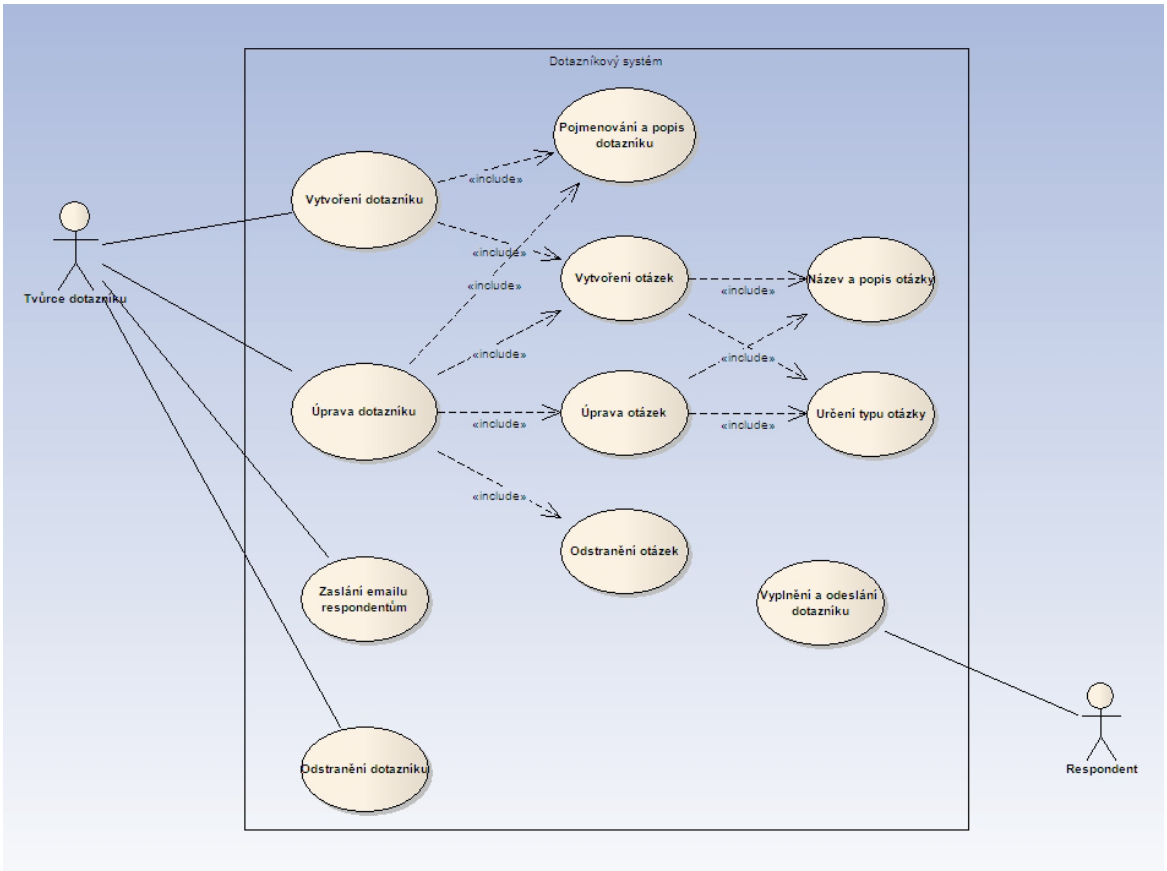
4.2 Use Case diagram

Tento diagram popisuje uživatele systému a jednotlivé funkcionality tohoto systému, které zmínění uživatelé využívají.

Administrátor dotazníkového průzkumu, který je zde pojmenován jako tvůrce dotazníku, může po přihlášení do systému vytvořit vlastní dotazník, kterému přísluší konkrétní název a popis. Každý dotazník samozřejmě obsahuje otázky vytvořené tvůrcem dotazníku. Každá z otázek má také konkrétní název a popis a dále obsahuje definovaný typ odpovědi, popřípadě i popis jednotlivých možností odpovědi. Otázky včetně typů odpovědí lze uvedeným uživatelem jak zakládat, upravovat, tak i mazat ať už při vytváření dotazníku, tak při jeho editaci.

K přizvání potencionálních respondentů průzkumu k vyplnění dotazníku může tvůrce dotazníku využít emailovou pozvánku, která hromadně zašle emailovou zprávu všem vybraným uživatelům. Tito uživatelé mohou následně vyplnit své odpovědi na otázky daného dotazníku a odeslat je na server.

Odpovědi uložené v databázi na serveru budou dále systémem zpracovávány a poskytnuty tvůrci dotazníku včetně jejich statistik a sumarizací.



Obrázek 2 – Use Case diagram

4.3 Analýza obrazovek

Vzhled obrazovek včetně rozložení jejich komponent bude realizován pomocí JSP stránek s dodatečnými funkcemi knihoven JSTL a s navázáním kaskádových stylů (css). Všechny obrazovky budou využívat tentýž soubor s kaskádovými styly, aby bylo zajištěno stejné stylování webových prvků.

4.3.1 Přehled dotazníků

Obrazovka s přehledem dotazníků bude sloužit jako rozcestník aplikace pro přihlášené uživatele. Bude rozdělena do třech sekcí, přičemž první z nich obsahuje dotazníky založené aktuálně přihlášeným uživatelem, druhá zobrazuje list těch dotazníků, na kterých je aktuálně přihlášený uživatel uveden jako potencionální respondent a ve třetím oddílu bude k dispozici seznam dotazníků k vyhodnocení.

V sekci s dotazníky, které uživatel vytvořil, bude moci buď pomocí tlačítka *Vytvořit* sestavit nový dotazník, nebo tlačítkem *Upravit* u každého již vytvořeného dotazníku konkrétnímu dotazníku přidat otázky či upravit název, dále bude u každého existujícího dotazníku tlačítko pro přidání či úpravu seznamu potencionálních respondentů a tlačítko pro zaslání emailové zprávy všem těmto respondentům. Zpráva by měla obsahovat mimo jiné adresu pro vyplnění dotazníku.

Ve druhé sekci budou v seznamu uvedeny všechny dotazníky, na kterých je uživatel uveden jako potencionální respondent. U každého dotazníku bude k dispozici tlačítko *Vyplnit*, pomocí něhož se přihlášený uživatel dostane na obrazovku pro vyplnění konkrétního dotazníku.

V poslední části budou zobrazeny všechny dotazníky k vyhodnocení. To je seznam všech dotazníků vytvořených aktuálně přihlášeným uživatelem s možností vyhodnocení každého z nich. To znamená, že po stisku tlačítka *Vyhodnotit* uživatel přejde na obrazovku se sumarizací všech odpovědí na otázky konkrétního dotazníku.

4.3.2 Sestavení dotazníku

Při vytváření nového či úpravě existujícího dotazníku bude v první řadě třeba zadat název a popis dotazníku do textových polí formuláře.

Dále bude uživatel pracovat s tvorbou či úpravou otázek. Každá otázka bude mít své ohraničení a bude se samostatně ukládat. Základní vstupní pole pro vytvoření otázky bude její název, což je samotná otázka, její popis, který respondentovi při vyplňování dotazníku přiblíží otázku a její typ, na základě něhož se dále zobrazí příslušná pole k vyplnění možností odpovědí.

V aplikaci budou podporovány následující typy otázek:

- Krátký text
- Dlouhý text
- Zaškrtávací políčka
- Jeden výběr ze seznamu
- Vysouvací seznam
- Mřížka

Z těchto typů otázek nebudou mít první dva typy už žádné dodatečné atributy, avšak při výběru jakéhokoliv ze zbylých se pod výběrem zobrazí textové pole pro vyplnění první možnosti odpovědi. Tyto možnosti bude moci tvůrce pomocí tlačítek umístěných vedle možnosti přidávat či odebírat.

Mřížka bude mít ještě specifičtější tvorbu otázky, kdy bude třeba nejdříve definovat počet sloupců mřížky, což odpovídá ve většině případů nějaké škále, ze které se vybírá odpověď, zadají se do vygenerovaných textových polí popisy těchto sloupců a dále bude moci uživatel vyplňovat, přidávat či odebírat samotné popisy řádků.

Následně půjde uložit celou otázku pomocí příslušného tlačítka. Ostatní otázky poté lze editovat nebo pomocí odpovídajících tlačítek u poslední z otázek vytvořit otázku novou anebo vytvořit duplikát té poslední.

Při editaci jedné konkrétní otázky bude moci tvůrce dotazníku tuto otázku nenávratně smazat.

Pokud nebude momentálně otázka v režimu editace, bude její zobrazení vypadat stejně jako na formuláři s vyplňovaným dotazníkem.

Po úspěšném uložení celého dotazníku pomocí tlačítka v zápatí formuláře pro tvorbu dotazníku dojde k přesměrování na stránku s informací o úspěšném uložení dotazníku do databáze a možností přechodu na stránku s přehledem dotazníků.

4.3.3 Přidání respondentů a zaslání emailových žádostí

Existujícímu dotazníku bude moci přihlášený uživatel, který tento dotazník vytvořil, přiřadit potencionální respondenty, které půjde vybrat při přechodu na výběr respondentů pomocí odpovídajícího tlačítka u konkrétního dotazníku.

Na formuláři se všemi načtenými uživateli z databáze vybere ty, od kterých by rád získal odpovědi na otázky vytvořeného dotazníku. Po uložení formuláře se stránka přesměruje zpět na přehled dotazníků, kde může tvůrce následně pomocí tlačítka pro

zaslání emailů rozeslat emailové zprávy obsahující adresu dotazníku všem vybraným uživatelům.

4.3.4 Vyplnění dotazníku

Přechod na stránku pro vyplnění dotazníku půjde realizovat dvěma způsoby. Prvním způsobem bude přechod pomocí tlačítka *Vyplnit* u jednoho z dotazníků v seznamu dotazníků k vyplnění přihlášeného uživatele. Aktuálně přihlášený uživatel zde bude mít k dispozici všechny dotazníky, na kterých je uveden jako potencionální respondent. Druhý způsob spočívá ve znalosti webové adresy pro vyplnění konkrétního dotazníku, která přijde každému potencionálnímu respondentovi v emailové zprávě s žádostí o vyplnění dotazníku. Na základě této možnosti se nemusí respondent do aplikace vůbec přihlašovat.

V obou případech se v první řadě respondentovi zobrazí stránka s názvem a popisem dotazníku, přičemž až po stisknutí tlačítka pro potvrzení, že chce opravdu dotazník vyplnit, se zobrazí dotazník k vyplnění.

Aby nebyly výzkumy zkreslené, budou všechny otázky povinné, což znamená, že u otázky s textovou odpovědí nelze nechat prázdné pole, stejně tak u otázek s výběrem z několika možností je povinností respondenta alespoň jednu možnost vybrat.

Pokud se tak nestane, formulář po stisknutí tlačítka pro uložení zobrazí chybové zprávy u nevyplněných otázek a do databáze se prozatím nic neuloží. Odpovědi se do databáze uloží až v tu chvíli, kdy ukládací proces nenalezne žádné validační problémy, a následně zobrazí respondentovi informaci o úspěšném uložení dotazníku.

4.3.5 Vyhodnocení dotazníku

Tato funkcionalita slouží pouze přihlášeným uživatelům, kteří vytvořili nějaký dotazník. Seznam všech dotazníků k vyhodnocení bude k nalezení na základní stránce přihlášeného uživatele. Po stisknutí tlačítka pro vyhodnocení požadovaného dotazníku, dojde k zobrazení stránky se sumarizačními daty odpovědí na jednotlivé otázky včetně grafů.

Všechny odpovědi lze exportovat do CSV, PDF, či Excelu.

5 Implementace aplikace

Aplikace je vyvinuta na platformě Java Enterprise Edition, která je pro vývoj aplikací zdarma a jako jeden z mála programovacích jazyků nezávislá na operačním systému.

K ulehčení implementace je využit Spring framework, který usnadňuje implementaci MVC (Model-View-Controller) architektury aplikace a zabezpečení přístupu. Detailní informace o problematice lze nalézt v [2,3,4,5,6,7,8,13].

Model v aplikaci představují entity, jež jsou mapovány přes JPA (Java Presistence Api) na relační tabulky v databázi MySQL. View je realizováno pomocí JSP stránek (Java Server Pages) s podporou JSTL knihoven (Java Server Pages Standard Tag Library) a jazyka JavaScript pro obsluhu požadavků na straně klienta. Controller obstarává požadavky uživatele odeslané na server (tzv. requesty) a volá aplikační logiku, přičemž vrací uživateli odpověď (response) v podobě modelu a view, což ve výsledku zobrazí uživateli data na vygenerované HTML stránce.

5.1 Model-View-Controller

5.1.1 Model

Základními stavebními kameny aplikace jsou hned po samotné databázi doménové objekty, které jsou objektovým obrazem databázových relačních tabulek. JPA umožňuje pomocí anotací tyto objekty mapovat na jednotlivé tabulky v databázi, mapovat jejich atributy na sloupce (atributy) příslušných tabulek včetně jejich typů a definovat vztahy mezi atributy jednotlivých objektů na základě vztahů sloupců tabulek v databázi. Tabulky vytvořené využitím cizích klíčů, u kterých je potřeba deklarovat, zda se jedná o vazbu one-to-one, one-to-many (resp. many-to-one) anebo many-to-many. Další informace lze dohledat v [9,12].

```
@Entity
@Table(name = "question")
public class Question implements Cloneable {

  @Id
  @GeneratedValue
  private Long id;

  @ManyToOne
  private Form form;

  @Column
  private String name;

  @Column
  private String description;

  @OneToMany(cascade = CascadeType.REMOVE, mappedBy = "question")
  private List<QuestionType> questionTypes;
```

Zdrojový kód 1 – Entita představující otázku

5.1.2 View

Prezentační vrstva je určena pro interakci uživatele s aplikací. Uživatel pomocí prohlížeče komunikuje s aplikací nasazenou na vzdáleném serveru zasíláním požadavků, na které dostává ze serveru odpovědi.

Tato vrstva je realizována JSP stránkami s využitím JSTL knihoven a Spring Framework tagů. V JSP souborech jsou nadefinovány jednotlivé webové prvky, kterým jsou předávána data v podobě modelu z příslušných controllerů. Ty lze ještě pomocí zmíněných JSTL knihoven modifikovat. Po sestavení celé stránky z dat a prvků na serveru se posílá klientovi pouze čisté HTML, které se uživateli zobrazí v prohlížeči.

Následující ukázka představuje část zdrojového kódu JSP stránky, ve kterém se konstruuje několik textových polí do řádků tabulky v závislosti na velikosti pole *questionTypes*. To je procházeno for-cyklem, přičemž k poslednímu textovému poli se navíc vygeneruje tlačítko pro přidání dalšího textového pole. Pokud není toto poslední textové pole zároveň jediným textovým polem, objeví se na výsledné stránce i tlačítko pro smazání tohoto textového pole. Přidávání a mazání textových polí je zde pak obslouženo JavaScriptem.

```
<c:set var="questionTypesCount" value="0" />
<c:set var="lastQuestionType" value="0" />
<c:if test="${fn:length(question.questionTypes) > 0}">
  <c:set var="lastQuestionType"
    value="${fn:length(question.questionTypes)}" />
</c:if>
<c:forEach var="questionTypes" items="${question.questionTypes}">
  <tr>
    <td class="longer"><form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
      <form:input
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
        cssClass="longest" cssErrorClass="longest error" />
      <form:errors
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
        cssClass="required" /></td>
    <c:if test="${questionTypesCount == lastQuestionType - 1}">
      <td class="buttons btn-group">
        <button type="button"
          class="btn btn-default btn-success plusOnly"
          onclick="addInput(this);"></button>
        <c:if test="${questionTypesCount != 0}">
          <button type="button"
            class="btn btn-default btn-danger crossOnly"
            onclick="deleteInput(this);"></button>
        </c:if>
      </td>
    </c:if>
  </tr>
  <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
</c:forEach>
```

Zdrojový kód 2 – vykreslení textového pole a tlačítek pro přidání a odebrání pole

5.1.3 Controller

Tato vrstva slouží pro obstarávání požadavků jdoucích z klienta na server a navrácení příslušných odpovědí serveru. Mezi těmito dvěma operacemi příslušný controller zajišťuje volání potřebné aplikační logiky.

Každý controller obsahuje anotaci *Controller* a *RequestMapping*, dle jejíž hodnoty se mapuje na konkrétní webovou stránku.

```
@Controller
@RequestMapping("/createForm")
public class CreateFormController {

  @Autowired
  private QuestionDao questionDao;

  @Autowired
  private TypeDao typeDao;

  @Autowired
  private FormDao formDao;
```

Zdrojový kód 3 – ukázka controlleru

Díky anotaci *Autowired* můžeme využívat v controlleru instance tříd, které jsou automaticky nastaveny IoC kontejnerem frameworku Spring. Postačí pouze skenování komponent v požadovaných balících definované v souboru *springmvc-servlet.xml*. Dále je zde povoleno samotné řízení anotacemi.

```
<context:component-scan
  base-package="MavenTestSpring.MavenTestikSpringMVCweb.web,
                MavenTestSpring.MavenTestikSpringMVCweb.domain,
                MavenTestSpring.MavenTestikSpringMVCweb.fileExport" />

<mvc:annotation-driven />
```

Zdrojový kód 4 – konfigurace skenování komponent a řízení anotacemi

Při přechodu na určitou adresu je zachycen GET požadavek, jenž načte příslušná data, která se mají zobrazit uživateli. Dále je sestaven model, který společně s konkrétním view controller vrací. Následně je na serveru zkonstruována HTML stránka odpovídající přiřazenému view z modelu. View může být pro zjednodušení pojato jako šablona pro vzhled stránky a rozmístění jednotlivých prvků a model představuje všechna data, které se mají na výsledné stránce uživateli zobrazit. Model je jakýsi balík instancí objektů, jejichž hodnoty atributů se přiřazují jednotlivým hodnotám HTML prvků. Výsledná HTML stránka obsahující požadované tagy včetně jejich hodnot je odeslána ze serveru klientovi a je zobrazena uživateli.

```
@RequestMapping(method = RequestMethod.GET)
ModelMap getForm(@RequestParam(value = "formId") String formId, ModelMap model) {
  if (model == null) {
    model = new ModelMap();
  }
  model.addAttribute("questTypes", typeDao.getTypes());
  if (formId != null && formId.length() > 0) {
    model.addAttribute("form", formDao.getFormById(Long.parseLong(formId)));
  } else {
    if (model.get("form") == null) {
      model.addAttribute("form", new Form());
    }
  }
  return model;
}
```

Zdrojový kód 5 – ukázka odchycení GET požadavku

Uživatel může následně se stránkou pracovat. Pokud jde například o formulář, který chce uživatel vyplnit a odeslat na server, aby se jeho záznamy uložily do databáze, vyplní na stránce požadovaná pole, odešle formulář pomocí tlačítka sloužícího k odeslání a uložení dat. Následně je odeslán POST požadavek na server, který zachytí příslušná metoda, jíž přijde jako vstupní parametr formulář s uživatelem vyplněnými daty. Takováto metoda většinou zajišťuje po případné dodatečné úpravě uložení dat do databáze. Po

uložení dochází buď k opětovnému volání GET požadavku, který vrátí klientovi původní stránku s aktualizovanými daty, nebo k přesměrování na jinou stránku, například na stránku s informací o úspěšném uložení dat do databáze.

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView save(@Validated Form form, BindingResult bindingResult, ModelMap model) {
  if (bindingResult.hasErrors()) {
    model.addAttribute("questTypes", typeDao.getTypes());
    model.addAttribute("errors", bindingResult.getFieldErrors());
    return new ModelAndView("/createForm", model);
  }
  User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
  String username = user.getUsername();
  formDao.saveForm(form, username, null);
  return new ModelAndView(new RedirectView("savedForm.htm")).addObject("formId", form.getId());
}
```

Zdrojový kód 6 - ukázka odchycení POST požadavku

5.2 Odeslání POST požadavku

POST požadavek slouží například k odeslání dat formuláře, jenž byl vyplněn uživatelem a je třeba data uložit do databáze nebo je jinak zpracovat. Je však více způsobů, jak tuto problematiku řešit, hlavně v případě, že je potřeba na jednom formuláři volat více rozdílných požadavků na základě potřeb uživatele.

5.2.1 Jediný POST požadavek na formuláři

V momentě, kdy se jedná o jediný existující POST požadavek formuláře, je implementace vcelku jednoduchá a pro vývojáře intuitivní. Odeslání takovéhoto požadavku se děje většinou pomocí tlačítka, které je konstruováno v *HTML* kódu jako <*input type=“submit“ />*, a jeho odchycení metodou příslušného controlleru, jež je pomocí anotace označena jako POST metoda.

```
<div>
  <input type="submit" class="btn-success fill"
    value="<spring:message code="form.toFill.saveAnswers"/>">
</div>
```

Zdrojový kód 7 – odeslání POST požadavku

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView save(@Validated Form form, BindingResult bindingResult, ModelMap model) {
  if (bindingResult.hasErrors()) {
    model.addAttribute("errors", bindingResult.getFieldErrors());
    return new ModelAndView("/fillForm", model);
  }
  String username = null;
  if (SecurityContextHolder.getContext().getAuthentication().getPrincipal() != ANONYMOUS_USER) {
    User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    username = user.getUsername();
  } else {
    username = ANONYMOUS_USER;
  }
  formDao.saveAnswersForForm(form, username);
  return new ModelAndView(new RedirectView("filledForm.htm")).addObject("formId", form.getId());
}
```

Zdrojový kód 8 – odchycení POST požadavku

5.2.2 Více POST požadavků na jednom formuláři

Problémy nastávají v případě, kdy je nutné odchytit na jednom formuláři více rozdílných požadavků. Řešením je například implementace klasického HTML tlačítka, na kterém je při stisku zavolána javascript funkce, jež požadovanému formuláři nastaví do skrytých položek potřebné parametry a následně je formulář odeslán na server.

Tlačítko je definováno jako <*input type=“button“ />* a je mu přiřazena funkce s názvem *saveQuestion*, jež se zavolá při každém stisku tohoto tlačítka.

```
<div>
  <input type="button" class="btn-success fill"
    value="<spring:message code="form.question.save"/>"
    onclick="saveQuestion(this);">
</div>
```

Zdrojový kód 9 – volání javascript funkce

Funkce *saveQuestion* zajišťuje přidání potřebných dat odesílanému formuláři. Na formuláři nalezne index otázky, která má být uložena, a přidá mu parametry požadavku. První parametr obsahuje pouze název *saveQuestion* a druhý s názvem *questionIndex* obsahuje ještě hodnotu nalezeného indexu otázky. Následně je formulář odeslán ke zpracování controllerem.

```
function saveQuestion(button) {
        var question = $(button).parent().parent();
        var questionId = $(question).find('input[type="hidden"][class="questionId"]').val();
        var questionIndex = getQuestionIndex(question);
        var questionName = $('input[name="questions[' + questionIndex +    '].name"]').val();
        $(button).closest("form").append("<input type=\"hidden\" name=\"saveQuestion\"/>");
        $(button).closest("form").append("<input type=\"hidden\" name=\"questionIndex\" " +
                        "value=\""+questionIndex+"\"/>");
        $(button).closest("form").submit();
}
```

Zdrojový kód 10 – odeslání POST požadavku přes javascript

Na základě přidaného parametru požadavku s názvem *saveQuestion* se zavolá metoda s anotací *@RequestMapping(method = RequestMethod.POST, params="save Question").* Jako příchozí parametry této metody jsou přidaný index otázky, celý formulář, *bidingResults* obsahující případné validační chyby a model.

Metoda zajistí uložení otázky na základě jejího předaného indexu a formuláře obsahující mimo jiné tuto otázku. Pokud ovšem přijdou metodě nějaké validační chyby, žádné ukládání se neprovádí, přidají se pouze potřebná data do modelu, na základě jehož vrácení se zobrazí stejná stránka s formulářem a se zvýrazněnými validačními problémy.

```
@RequestMapping(method = RequestMethod.POST, params = "saveQuestion")
ModelMap saveQuestion(
    @RequestParam(value = "questionIndex") String questionIndex,
    @Validated Form form, BindingResult bindingResult, ModelMap model) {

  if (model == null) {
    model = new ModelMap();
  }
  if (bindingResult.hasErrors()) {
    model.addAttribute("questionIndex", questionIndex);
  } else {
    User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String username = user.getUsername();

    formDao.saveForm(form, username, Integer.valueOf(questionIndex));
    form = formDao.getFormById(form.getId());
  }
  model.addAttribute("questTypes", typeDao.getTypes());
  model.addAttribute("form", form);

  return model;
}
```

Zdrojový kód 11 – odchycení POST požadavku odeslaného přes javascript

5.3 Validace

K zobrazení informací uživateli o tom, že nevyplnil povinnou položku na formuláři, což by mohlo vést při ukládání k chybě či nekonzistenci v databázi, slouží validátory. Ty kontrolují, zda jsou jednotlivé položky vyplněné nebo zda hodnoty jednotlivých položek odpovídají očekávanému typu, délce, a podobně.

5.3.1 JSP stránka

V JSP souboru příslušejícím konkrétní stránce nadefinujeme vstupní pole například pomocí značek *<form:input />* nebo *<form:textarea />* a případné chybové hlášení značkou *<form:errors />*. Tato chybová hlášení se zobrazí na stránce pouze v tom případě, že validační proces vrátí nějaké chyby, které budou navázány na tyto prvky.

```
<fieldset class="subfieldsset">
  <form:hidden path="form.id" />
  <form:hidden path="form.owner.id" />
  <form:hidden path="form.owner.username" />
  <span class="required">*</span>
  <form:label path="form.name">
    <spring:message code="form.name" />
  </form:label>
  <form:input path="form.name" cssClass="long" cssErrorClass="long error" />
  <form:errors path="form.name" cssClass="required" />
  <br />
  <span class="required">*</span>
  <form:label path="form.description">
    <spring:message code="form.description" />
  </form:label>
  <form:textarea path="form.description" cssClass="long" cssErrorClass="long error" />
  <form:errors path="form.description" cssClass="required" />
</fieldset>
```

Zdrojový kód 12 – chybová hlášení vázaná na formulářové prvky

5.3.2 Chybová hlášení

Při nastartování serveru se mimo jiné načítá soubor s hodnotami chybových hlášení. V souboru *springmvc-servlet.xml* je to zajištěno pomocí definice *messageSource*, která na základě předané první hodnoty do listu *basenames* načítá v české lokalizaci soubor *errors_cs_CZ.properties*, ve kterém jsou uloženy jednotlivé klíče chybových hlášení použitých v aplikaci s jejich hodnotami.

```
<bean id="messageSource"
        class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basenames">
                <list>
                        <value>/WEB-INF/errors</value>
                        <value>/WEB-INF/messages</value>
                </list>
        </property>
        <property name="defaultEncoding" value="UTF-8" />
        <property name="fileEncodings" value="UTF-8" />
</bean>
```

Zdrojový kód 13 – konfigurace načítání souborů s lokalizovanými chybovými hlášeními

5.3.3 Validátory

Každý z validátorů implementuje rozhraní *Validator*, které vyžaduje implementovat metody *supports* a *validate*. První z metod určuje objekt, který je daný validátor schopen zkontrolovat a druhá porovnává již konkrétní hodnoty položek formuláře s požadovanými pravidly.

Konkrétně u třídy *CreateFormValidator* je očekáván pro validaci objekt *Form*, na nějž je v metodě *validate* přetypován vstupující objekt *obj*. Pro kontrolu zadání hodnoty do pole formuláře můžeme využít *ValidationUtils*, které v negativním případě po volání

metody r*ejectIfEmptyOrWhitespace* s předanou instancí objektu *Errors*, hodnotou atributu *name* validovaného pole a klíčem pro chybovou zprávu vloží do seznamu chybových hlášení (instance objektu *Errors*) záznam vázající se na konkrétní vstupní pole formuláře s příslušnou chybovou zprávou. Druhou možností je zvolit podmínku, za které se má konkrétní chybové hlášení uživateli zobrazit, a v případě pravdivé hodnoty podmínky na instanci objektu *Errors* zavolat metodu *rejectValue*, které se podobně předají jako parametry hodnota atributu *name* pole, u kterého se má zobrazit chybové hlášení, klíč pro chybovou zprávu a výchozí chybová zpráva.

```
public class CreateFormValidator implements Validator {

  public boolean supports(Class<?> clazz) {
    return Form.class.equals(clazz);
  }

  public void validate(Object obj, Errors e) {
    Form form = (Form) obj;
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "name", "empty.form.name");
    if (form.getName().length() > 100) {
      e.rejectValue("name", "tooLong.name", "Can't be too long!");
    }
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "description", "empty.form.description");
    if (form.getDescription().length() > 255) {
      e.rejectValue("description", "tooLong.description", "Can't be too long!");
    }
    if (form.getQuestions() != null) {
      for (int i = 0; i < form.getQuestions().size(); i++) {
        new CreateQuestionValidator("questions", i).validate(form.getQuestions().get(i), e);
      }
    }
  }
```

Zdrojový kód 14 – ukázka validátoru

5.3.4 Použití validátoru v controlleru

Požadovaný validátor přiřadíme controlleru pomocí metody *initBinder* implementované přímo v konkrétním controlleru.

```
@InitBinder
protected void initBinder(WebDataBinder binder) {
    binder.setValidator(new CreateFormValidator());
}
```

Zdrojový kód 15 – přiřazení validátoru controlleru

Pro použití validátoru při odeslání formuláře uživatelem stačí přidat vstupnímu parametru metody, která odeslání formuláře odchytává, anotaci *Validated*. Vstupní parametr zde představuje odeslaný formulář. Metoda již zajistí volání validačního procesu přiřazeného validátoru a jako druhý vstupní parametr metody již přijdou všechny nalezené validační problémy, se kterými můžeme v rámci této metody pracovat. To znamená, že se

nebudou data zatím ukládat do databáze, ani nedojde k přesměrování na jinou stránku, ale v případě potřeby se přidají do modelu potřebná data a vrátí se totéž view s předaným modelem. Na základě validačních chyb se poté na stránce generují chybové zprávy u problémových polí formuláře.

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView save(@Validated Form form, BindingResult bindingResult, ModelMap model) {
  if (bindingResult.hasErrors()) {
    model.addAttribute("questTypes", typeDao.getTypes());
    model.addAttribute("errors", bindingResult.getFieldErrors());
    return new ModelAndView("/createForm", model);
  }

  User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
  String username = user.getUsername();

  formDao.saveForm(form, username, null);

  return new ModelAndView(new RedirectView("savedForm.htm")).addObject("formId", form.getId());
}
```

Zdrojový kód 16 – validovaný vstupní parametr odchyceného POST požadavku

5.4 Zprávy

Zprávami (neboli messages) jsou myšlena lokalizovaná informační hlášení, popisky formulářových polí a tlačítek, tituly stránek, nadpisy sekcí na stránce a podobně.

Všechny zprávy vyskytující se na stránkách napříč aplikací jsou podobně jako u chybových zpráv seskupeny v jednom souboru.

Název souboru je *messages_cs_CZ.properties* a načítá se při startu serveru společně se souborem chybových zpráv.

```
<bean id="messageSource"
  class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
  <property name="basenames">
    <list>
      <value>/WEB-INF/errors</value>
      <value>/WEB-INF/messages</value>
    </list>
  </property>
  <property name="defaultEncoding" value="UTF-8" />
  <property name="fileEncodings" value="UTF-8" />
</bean>
```

Zdrojový kód 17 - konfigurace načítání souborů s lokalizovanými zprávami

5.4.1 Použití zpráv v JSP

V každém JSP souboru, kde chceme zprávy použít, je třeba nejdříve určit prefix použité knihovny značek.

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

Zdrojový kód 18 – prefix knihovny značek

A následně můžeme použít na základě definovaného prefixu značku <*spring:message /*>, jejímuž atributu *code* předáme klíč, pod nímž je uložena lokalizovaná zpráva ve výše zmiňovaném souboru.

```
<input type="submit" class="fill" value="<spring:message code="form.save"/>">
```

Zdrojový kód 19 – ukázka vložení lokalizované zprávy

5.4.2 Použití zpráv v Javě

Pokud je třeba použít načtení zprávy v Java třídě, implementace je umožněna přes instanci třídy *MessageSource*.

```
@Autowired
private MessageSource messageSource;
```

Zdrojový kód 20 – instance třídy pro získání lokalizovaných zpráv

Volání metody *getMessage* instance zmíněné třídy vrací lokalizovanou zprávu na základě jejího klíče stejně tak jako při volání z JSP souboru. Této metodě je však nutné předat kromě klíče další parametry. Těmi jsou pole argumentů, které může být *null*, a lokalizace aplikace. Dále to může být nepovinný parametr *defaultMessage*, jenž podle názvu napovídá, že půjde o předanou výchozí zprávu, pokud není zpráva s požadovaným klíčem nalezena.

```
model.addAttribute("message", messageSource.getMessage("form.succes.saved",
    null, Locale.getDefault()));
```

Zdrojový kód 21 – získání lokalizované zprávy a vložení v podobě atributu do modelu

V případě potřeby zobrazení zprávy obsahující hodnotu jakéhokoliv atributu instance třídy, která je v aplikaci momentálně k dispozici, lze předat již nastíněné pole argumentů s konkrétními hodnotami. V použitém kódu se jedná o pole textových hodnot, jimiž jsou název formuláře a jeho adresa.

```
messageSource.getMessage("form.toFill.mail.content",
    new String[] { form.getName(), formUrl }, Locale.getDefault());
```

Zdrojový kód 22 - získání lokalizované zprávy s parametry

Pro požadované zobrazení zprávy musí být v souboru *messages_cs_CZ.properties* následující klíč a hodnota s odkazy na předané položky pole hodnot v zobrazeném formátu.

```
form.toFill.mail.content = Prosím o vyplnění dotazníku s názvem {0} na adrese {1}
```

Zdrojový kód 23 – lokalizovaná zpráva s parametry

5.5 Java entity

Entitami jsou všechny třídy jazyka Java obsahující anotací @Entity. Ta označuje skutečnost, že třída je objektovým obrazem relační tabulky z databáze.

K tomu přidaná anotace *@Table(name="xxx")* říká, že tato entita je mapována na konkrétní tabulku v databázi, a to tabulku s názvem *xxx*.

Pro příklad entita *Form* je objektovým obrazem databázové tabulky s názvem *form*, která má definovány následující sloupce:

- id
  - primární klíč tabulky, identifikátor dotazníku
- owner_id
  - cizí klíč tabulky (reference na tabulku *users*), tvůrce dotazníku
- name
  - název dotazníku
- description
  - popis dotazníku

Tyto sloupce jsou přímo mapovány na jednotlivé atributy entity. Atribut *owner* je zde však speciálním případem, kdy je atributem entity *Form* reference na entitu *User*, jež je objektovým obrazem tabulky *users* a její instance představuje v tomto případě tvůrce dotazníku. Anotace *@ManyToOne* u atributu *owner* určuje kardinalitu vztahu.

Anotace *@GeneratedValue* u atributu *id* zajišťuje ve spojení s *autoincrement* příznakem mapovaného sloupce v databázové tabulce automatické vygenerování hodnoty identifikátoru při ukládání objektu do databáze.

```
@Entity
@Table(name = "form")
public class Form {

  @Id
  @GeneratedValue
  private Long id;

  @ManyToOne
  private User owner;

  @Column
  private String name;

  @Column
  private String description;

  @OneToMany(mappedBy = "form")
  private List<Question> questions;

  @ManyToMany
  @JoinTable(name = "form_respondent")
  private List<User> respondents;

  @Transient
  private List<Respondent> potentionalRespondents;
```

Zdrojový kód 24 – entita s atributy mapovanými na sloupce tabulky v databázi

Atribut anotován *@OneToMany* naznačuje referenci na tento objekt v jiné entitě včetně určení kardinality a atributu, na který je referencován.

Vazební tabulku ošetříme pomocí notace *@ManyToMany*, kde je třeba definovat anotací *@JoinTable* název vazební tabulky.

@Transient atribut je vyloučen ze všech atributů ukládaných do databáze, slouží pouze jako atribut klasické třídy.

Na výňatku z třídy *Form* nejsou obsaženy *gettery* a *settery* jednotlivých atributů.

5.6 Komponenty

Implementace komponent použitých při vytváření, vyplňování a vyhodnocení dotazníku vyplývá do určité míry z návrhu databáze, resp. z analýzy databázového schématu. Jedná se o propojení HTML prvků s třídami jazyka Java, mapovanými na jednotlivé tabulky v databázi, jejich načítání, zobrazení, vyplnění a ukládání.

V implementovaném systému se jedná o konstrukci následujících typů odpovědí, které jsou brány v potaz jak při vytváření dotazníku, při jeho vyplňování, tak i při vyhodnocení.

- Krátký text
- Dlouhý text
- Zaškrtávací políčka
- Jeden výběr ze seznamu
- Vysouvací seznam
- Mřížka

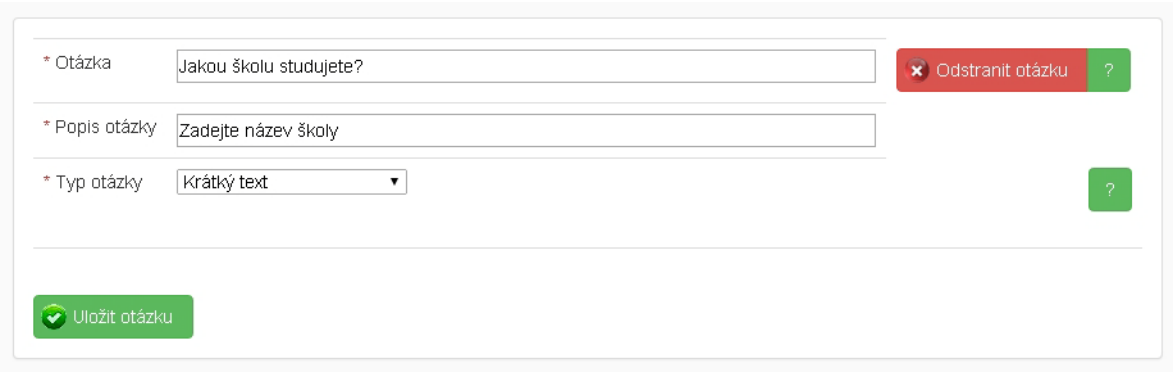
Databáze a od ní se odvíjející entity jsou navrženy tak, aby bylo možné jednotlivé dotazníky obsahující otázky a odpovědi uvedených typů uložit do malého množství databázových tabulek a nemusely se přitom vytvářet tabulky pro jednotlivé typy otázek a odpovědí.

Hierarchicky je konstruován každý dotazník tímto způsobem:

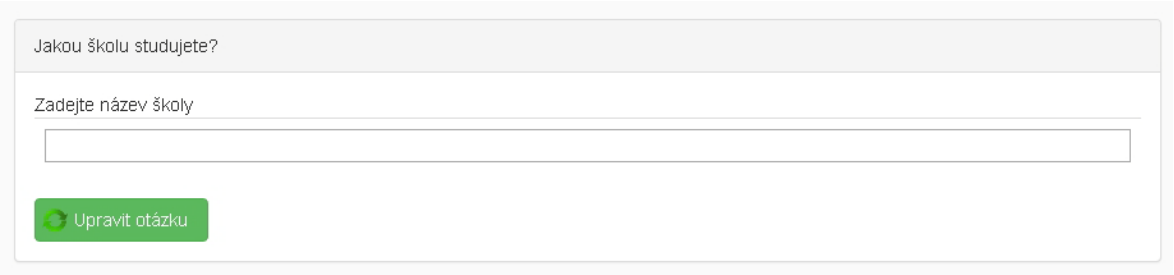
- Dotazník – třída *Form* – obsahuje seznam otázek
    - Otázka – třída *Question* – obsahuje seznam možností odpovědí
        - Možnost odpovědi – třída *QuestionType* – obsahuje seznam a typ odpovědí
            - Odpověď – třída *Answer*

5.6.1 Sestavení dotazníku

Implementace všech vytvořených komponent je rozvrstvena do tří logických celků aplikace, kterými jsou JSP stránka, controller a perzistentní vrstva. Na následujících obrázcích jsou jednotlivé komponenty sestaveny do HTML a zobrazeny prohlížečem.

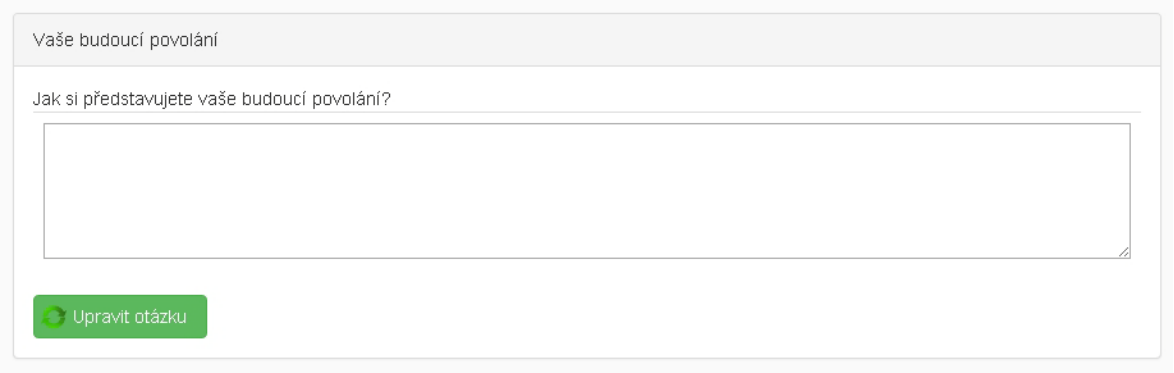


Obrázek 3 - tvorba otázky s odpovědí typu krátký text

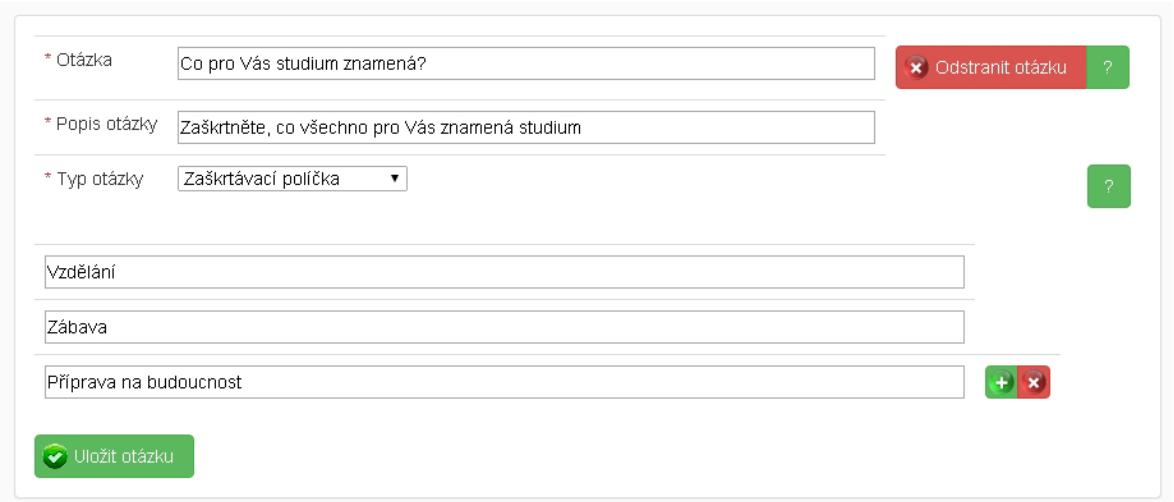


Obrázek 4 - podoba uložené otázky s odpovědí typu krátký text

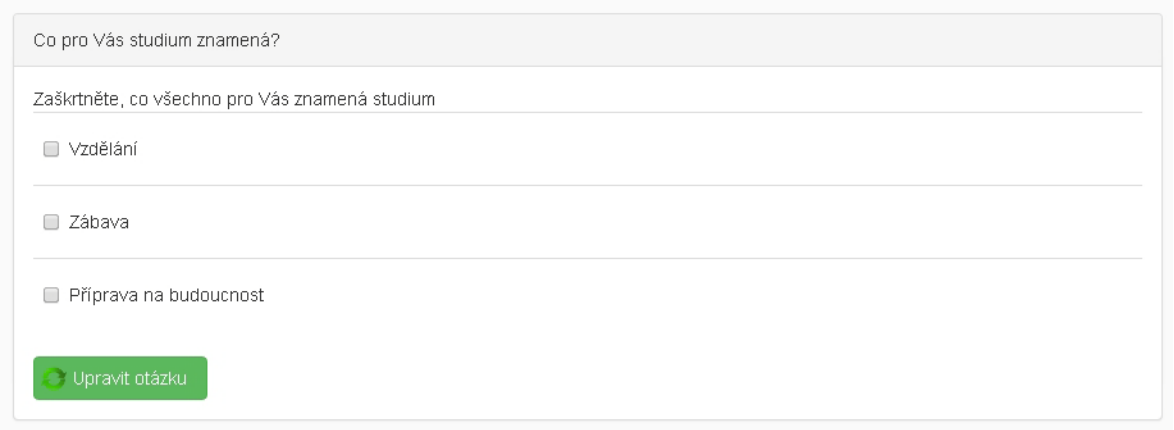
Obrázek 5 – tvorba otázky s odpovědí typu dlouhý text



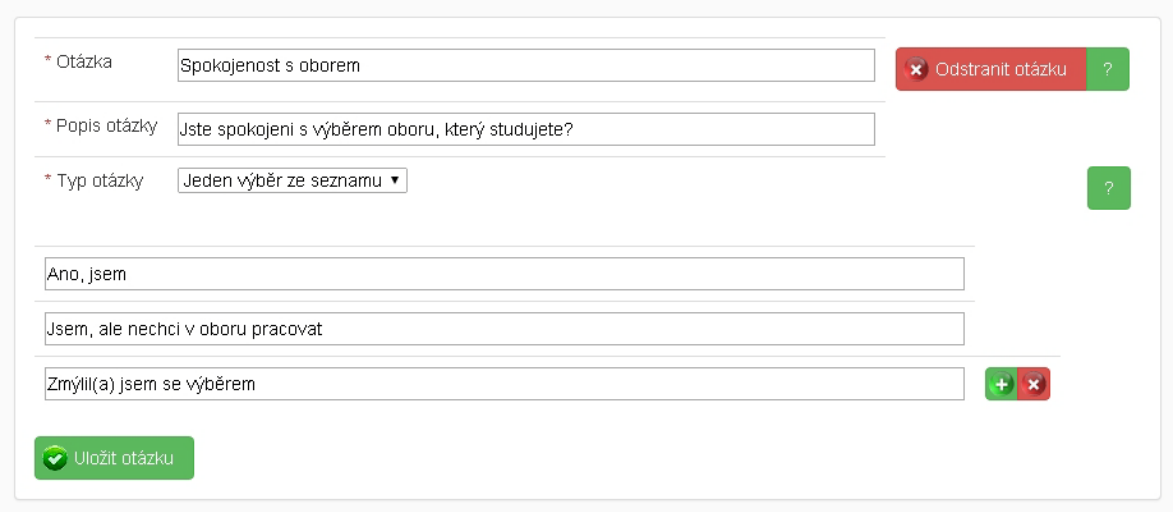
Obrázek 6 - podoba uložené otázky s odpovědí typu dlouhý text



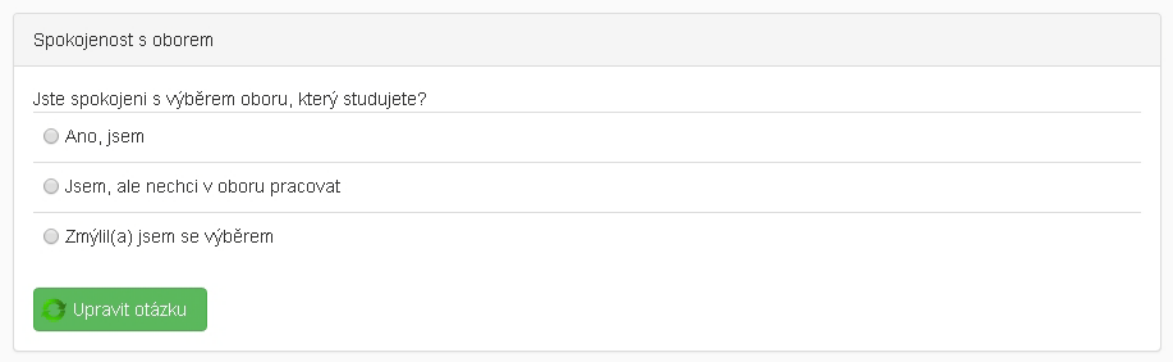
Obrázek 7 - tvorba otázky s odpovědí typu zaškrtávací políčka



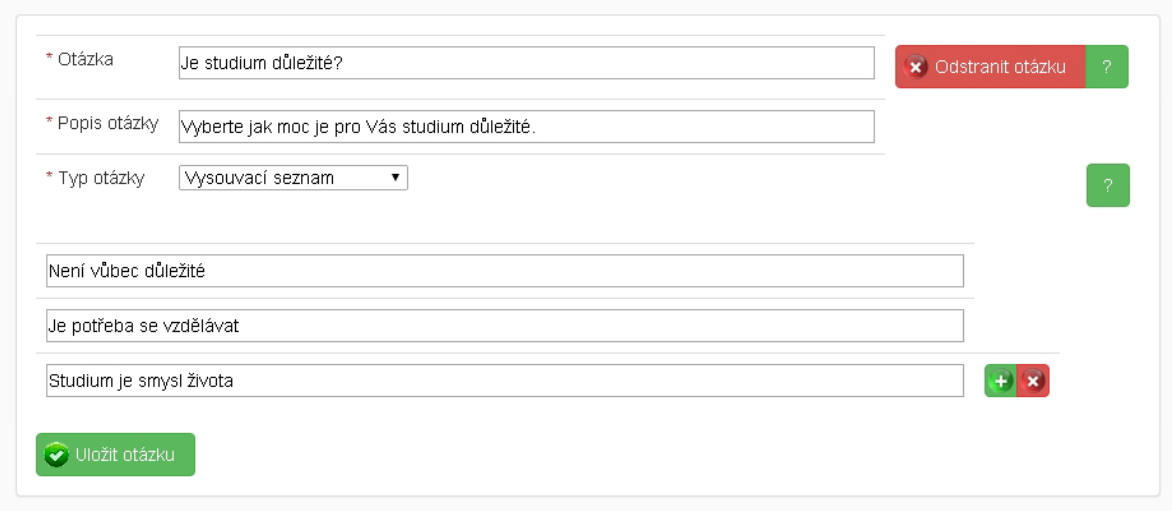
Obrázek 8- podoba uložené otázky s odpovědí typu zaškrtávací políčka



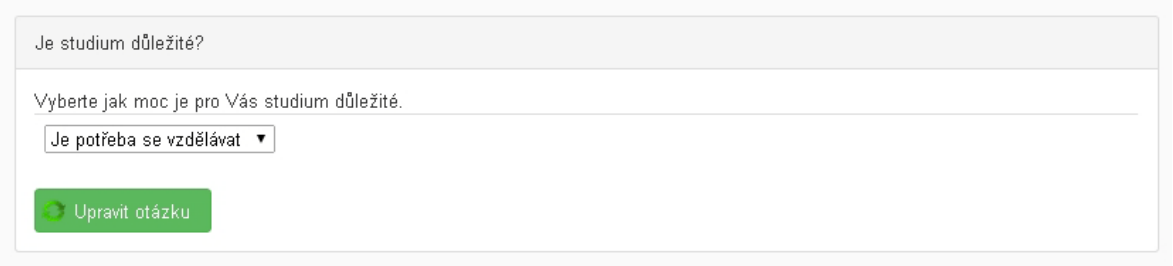
Obrázek 9 – tvorba otázky s odpovědí typu jeden výběr ze seznamu



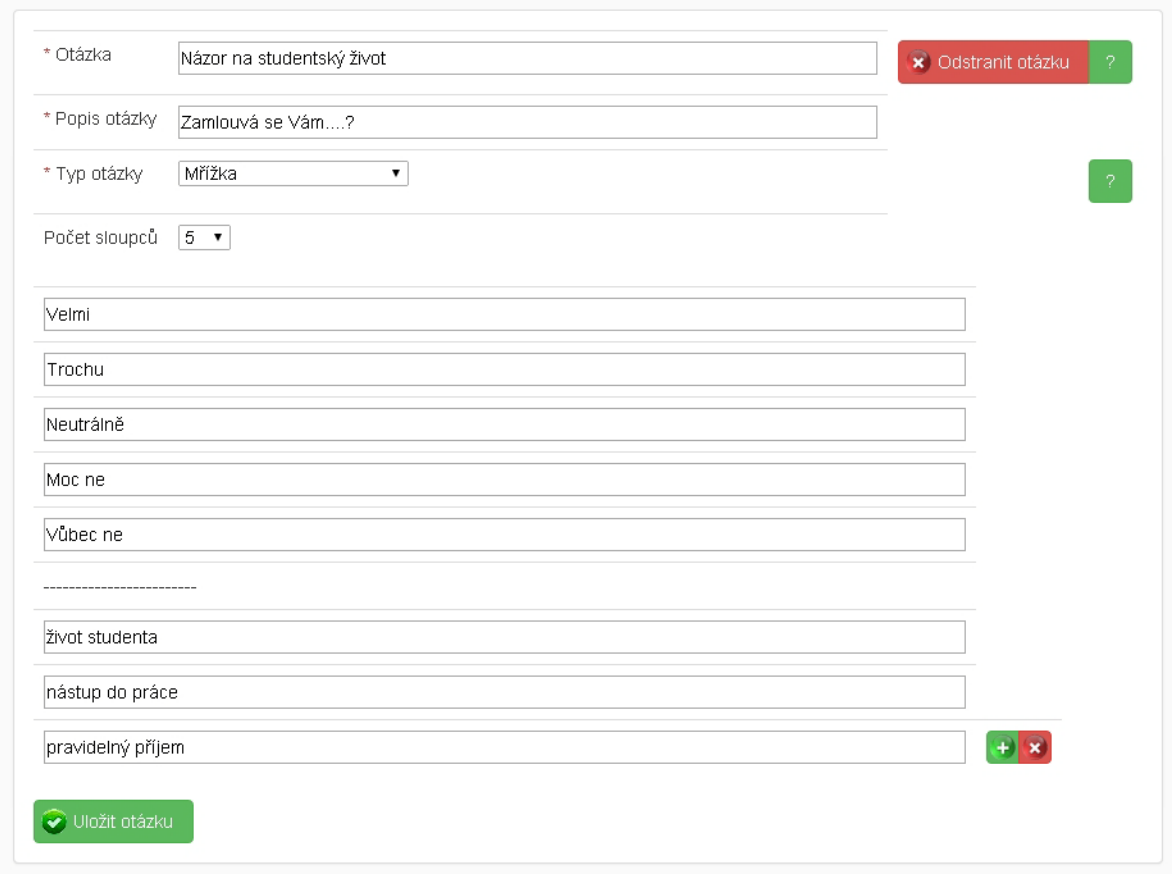
Obrázek 10- podoba uložené otázky s odpovědí typu jeden výběr ze seznamu



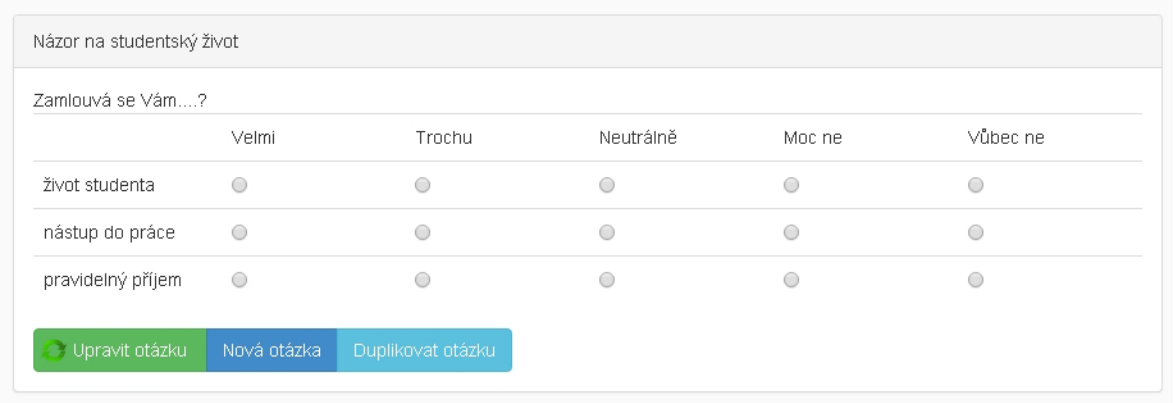
Obrázek 11 - tvorba otázky s odpovědí typu vysouvací seznam



Obrázek 12- podoba uložené otázky s odpovědí typu vysouvací seznam



Obrázek 13 - tvorba otázky s odpovědí typu mřížka



Obrázek 14- podoba uložené otázky s odpovědí typu mřížka

5.6.1.1 Implementace v JSP

Jednoduchým případem je implementace vstupních textových polí pro uložení názvu a popisu dotazníku. Hodnoty těchto polí se napřímo naváží na atributy třídy *Form*. Jedná se o pole definované pomocí značek *<form:input />* a *<form:textarea />*.

Značky *<form:hidden />* jsou zde i v dalších příkladech použity pro uchování hodnot, které nejsou zobrazovány na stránce. Pokud by nebyly hodnoty tímto způsobem uchovávány, ztrácely by se při odesílání dat do controlleru.

```
<fieldset class="subfieldsset">
    <form:hidden path="form.id" />
    <form:hidden path="form.owner.id" />
    <form:hidden path="form.owner.username" />
    <span class="required">*</span>
    <form:label path="form.name">
      <spring:message code="form.name" />
    </form:label>
    <form:input path="form.name" cssClass="long" cssErrorClass="long error" />
    <form:errors path="form.name" cssClass="required" />
    <br />
    <span class="required">*</span>
    <form:label path="form.description">
      <spring:message code="form.description" />
    </form:label>
    <form:textarea path="form.description" cssClass="long" cssErrorClass="long error" />
    <form:errors path="form.description" cssClass="required" />
</fieldset>
```

Zdrojový kód 25 – formulářové prvky v JSP

Podobně je tomu při vytváření popisků a vstupních polí pro zadání otázky a jejího popisu. V ukázce se jedná v prvním řádku tabulky o vykreslení popisku otázky, vstupního pole pro její zadání, pole pro navázání validačních chyb a tlačítka pro odstranění celé otázky. Ve druhém řádku se na stejném principu generují pole týkající se popisu otázky.

```
<tr>
  <td>
    <span class="required">*</span>
    <spring:message code="question.name" />
  </td>
  <td>
    <form:input path="form.questions[${questionCount}].name" cssErrorClass="error" />
    <form:errors path="form.questions[${questionCount}].name" cssClass="required" />
  </td>
  <td>
    <input type="button" class="btn-danger cross"
    value="<spring:message code="form.question.remove"/>"
    onclick="removeQuestion(this);">
  </td>
</tr>
<tr>
  <td>
    <span class="required">*</span>
    <spring:message code="question.description" />
  </td>
  <td>
    <form:input path="form.questions[${questionCount}].description" cssErrorClass="error" />
    <form:errors path="form.questions[${questionCount}].description" cssClass="required" />
  </td>
</tr>
```

Zdrojový kód 26 – konstrukce otázky v JSP

Při sestavování dotazníku a otázek, které má obsahovat, je již nutné znát typy odpovědí na konkrétní otázky a s daným typem odpovědi jednotlivé otázky ukládat do databáze. Pro výběr daného typu při tvorbě každé otázky slouží vysouvací seznam, jehož hodnota je vázána na první možnost odpovědi na konkrétní otázku.

```
<select class="questionTypeType"
  name="questions[${questionCount}].questionTypes[0].type.id"
  onChange="insertElements(this);">
  <c:forEach var="type" items="${questTypes}">
    <c:choose>
      <c:when test="${type.id == question.questionTypes[0].type.id}">
        <option selected="selected" value="${type.id}">
          <spring:message code="${type.name}" />
        </option>
      </c:when>
      <c:otherwise>
        <option value="${type.id}">
          <spring:message code="${type.name}" />
        </option>
      </c:otherwise>
    </c:choose>
  </c:forEach>
</select>
```

Zdrojový kód 27 – konstrukce seznamu pro výběr typu odpovědi na otázku

Při volbě typu krátkého nebo dlouhého textu se ukládá nebo uchovává pouze jedna možnost odpovědi na základě jejího identifikátoru bez popisu a dalších hodnot.

```
<c:if test="${question.questionTypes[0].type.id == 'LNG_TEXT'
           || question.questionTypes[0].type.id == 'SH_TEXT'}">
  <tr>
    <td>
      <form:hidden path="form.questions[${questionCount}].questionTypes[0].id" />
    </td>
  </tr>
</c:if>
```

Zdrojový kód 28 – uchování identifikátoru při volbě typu krátkého nebo dlouhého textu

Po výběru zaškrtávacích polí, jednoho výběru ze seznamu anebo vysouvacího seznamu se pro každou možnost odpovědi generuje kromě skrytého identifikátoru této položky textové pole, které představuje popis možnosti odpovědi.

Dále je definováno pro případ validačních chyb pole pro navázání validační zprávy.

V případě, že aktuálně vykreslovaná možnost odpovědi je poslední ze všech možností přiřazených zpracovávané otázce, je za textové vstupní pole pro popis vloženo ještě tlačítko pro přidání nové možnosti odpovědi na konec seznamu.

Pokud není aktuálně vykreslovaná možnost zároveň jedinou možností v seznamu příslušejícím této otázce, je vedle tlačítka pro přidání nové možnosti vygenerováno také tlačítko pro smazání popisu této možnosti.

```
<c:if test="${question.questionTypes[0].type.id == 'CHCK_BOX'
         || question.questionTypes[0].type.id == 'SELECT'
         || question.questionTypes[0].type.id == 'RAD_GR'}">
  <c:set var="questionTypesCount" value="0" />
  <c:set var="lastQuestionType" value="0" />
  <c:if test="${fn:length(question.questionTypes) > 0}">
    <c:set var="lastQuestionType" value="${fn:length(question.questionTypes)}" />
  </c:if>
  <c:forEach var="questionTypes" items="${question.questionTypes}">
    <tr>
      <td>
        <form:hidden
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
        <form:input
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssErrorClass="error" />
        <form:errors
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssClass="required" />
      </td>
      <c:if test="${questionTypesCount == lastQuestionType - 1}">
        <td class="buttons">
          <input type="button" class="btn-success plusOnly" onclick="addInput(this);">
          <c:if test="${questionTypesCount != 0}">
            <input type="button" class="btn-danger crossOnly" onclick="deleteInput(this);">
        </c:if>
        </td>
      </c:if>
    </tr>
    <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
  </c:forEach>
</c:if>
```

Zdrojový kód 29 – konstrukce textových polí pro definici jednotlivých odpovědí

Složitějším případem je generování mřížky, kdy komponenta zahrnuje sloupce a řádky ukládané ve stejném formátu do databáze.

Implementaci samotných sloupců a řádků mřížky předchází rozdělení uložených či vytvářených možností odpovědí do dvou listů. K rozdělení do listu sloupců a listu řádků napomáhá Java kód vložený přímo do JSP souboru.

```
<c:set var="lastQuestionTypeRow" value="0" />
<c:set var="lastQuestionTypeCol" value="0" />
<c:set var="questionTypesCount" value="0" />

<%
  List<QuestionType> columnList = new ArrayList<QuestionType>();
  List<QuestionType> rowList = new ArrayList<QuestionType>();
%>
<c:forEach var="questionType" items="${question.questionTypes}">
  <c:if test="${(questionType.type.id != null && questionType.type.id == 'GRID_COL')
              || (questionType.type.id == null && questionTypesCount <
                  question.questionTypes[0].gridColumnCount)}">
    <%
      columnList.add((QuestionType) pageContext.getAttribute("questionType"));
    %>
    <c:set var="lastQuestionTypeCol" value="${lastQuestionTypeCol + 1}" />
  </c:if>
  <c:if test="${(questionType.type.id != null && questionType.type.id == 'GRID_ROW')
              || (questionType.type.id == null && questionTypesCount >=
                  question.questionTypes[0].gridColumnCount)}">
    <%
      rowList.add((QuestionType) pageContext.getAttribute("questionType"));
    %>
    <c:set var="lastQuestionTypeRow" value="${lastQuestionTypeRow + 1}" />
  </c:if>
  <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
</c:forEach>
<%
  pageContext.setAttribute("columnList", columnList);
  pageContext.setAttribute("rowList", rowList);
%>
```

Zdrojový kód 30 – příprava konstrukce mřížky za pomocí vloženého Java-kódu do JSP

Vytvoření vstupních polí s případnými přiřazenými hodnotami a validačními chybami sloužících pro definici sloupců mřížky zajišťuje následující kód, v němž se prochází výše naplněný list sloupců.

```
<c:set var="questionTypesCount" value="0" />
<tbody class="gridColumnDescriptions">
  <c:set var="questionTypesColCount" value="0" />
  <c:forEach var="questionTypeCol" items="${columnList}">
    <tr>
      <td>
        <form:hidden
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
        <form:input
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssErrorClass="error" />
        <form:errors
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssClass="required" />
      </td>
    </tr>
    <c:set var="questionTypesColCount"
      value="${questionTypesColCount + 1}" />
    <c:set var="questionTypesCount"
      value="${questionTypesCount + 1}" />
  </c:forEach>
</tbody>
```

Zdrojový kód 31 – konstrukce sloupců mřížky

Stejným způsobem se vytvářejí vstupní pole indikující řádky mřížky při procházení vytvořeného listu řádků. Navíc se zde generují tlačítka pro přidání a smazání řádku mřížky.

```
<tbody class="gridRowDescriptions">
  <c:set var="questionTypesRowCount" value="0" />
  <c:forEach var="questionTypeRow" items="${rowList}">
    <tr>
      <td>
        <form:hidden
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
        <form:input
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssErrorClass="error" />
        <form:errors
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description"
          cssClass="required" />
      </td>
      <c:if test="${questionTypesCount == lastQuestionTypeRow + lastQuestionTypeCol - 1}">
        <td class="buttons">
          <input type="button" class="btn-success plusOnly"
            onclick="addInput(this, null, ${lastQuestionTypeCol});">
          <c:if test="${questionTypesRowCount != 0}">
            <input type="button" class="btn-danger crossOnly"
              onclick="deleteInput(this, null, ${lastQuestionTypeCol});">
          </c:if>
        </td>
      </c:if>
    </tr>
    <c:set var="questionTypesRowCount" value="${questionTypesRowCount + 1}" />
    <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
  </c:forEach>
</tbody>
```

Zdrojový kód 32 - konstrukce řádků mřížky

5.6.1.2 Implementace v controlleru

Po odeslání dat uživatelem na server se zavolá metoda příslušného controlleru, do které vstupují jako parametry instance třídy *Form*, validační chyby a model. Atributy instance třídy *Form* jsou v tuto chvíli již ověřeny, zda neobsahují nějaké validační chyby. Pokud žádné validační chyby nebyly nalezeny, zavolá se mimo jiné metoda pro uložení celého dotazníku, která je již součástí perzistentní vrstvy.

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView save(@Validated Form form, BindingResult bindingResult, ModelMap model) {
  if (bindingResult.hasErrors()) {
    model.addAttribute("questTypes", typeDao.getTypes());
    model.addAttribute("errors", bindingResult.getFieldErrors());
    return new ModelAndView("/createForm", model);
  }

  User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
  String username = user.getUsername();

  formDao.saveForm(form, username, null);

  return new ModelAndView(new RedirectView("savedForm.htm")).addObject("formId", form.getId());
}
```

Zdrojový kód 33 – metoda controlleru pro uložení formuláře

5.6.1.3 Implementace perzistentní vrstvy

Persistentní vrstva obsahuje mimo doménových objektů, tzv. entit, také implementaci jejich načítání a ukládání. K metodám, které tuto funkcionalitu poskytují, se přistupuje přes rozhraní příslušných implementací. Pro příklad je zde umístěna ukázka rozhraní s metodou pro uložení celého dotazníku.

```
public interface FormDao {

  public void saveForm(Form form, String username, Integer questionIndex);
```

Zdrojový kód 34 – metoda rozhraní perzistentní vrstvy pro uložení formuláře

Tato metoda je zároveň součástí třídy *FormDaoImpl* a její implementace je zobrazena na dalším výňatku zdrojového kódu.

Přes třídu *DbConnector* se nejprve zinicializuje a načte *entityManager*. Provede se kontrola, zda je aktivní transakce. V negativním případě se zavolá příkaz pro započetí transakce. Pokud je nalezen existující identifikátor dotazníku, je dotazník již v databázi uložen. Pak je ho třeba upravit a uložit s aktuálními daty. Pokud žádný identifikátor

dotazníku není dostupný, musí se dotazník uložit pomocí volání metody *persist*. Tím zajistíme vytvoření identifikátoru a uložení dotazníku včetně jeho identifikátoru do databáze. To je samozřejmě zajištěno pouze s nastavením *autoincrement* příslušnému sloupci databázové tabulky, v níž jsou uchovávány data dotazníků.

Po uložení dotazníku se ukládá konkrétní otázka nebo všechny otázky v dotazníku obsažené v závislosti na tom, zda je předán index otázky.

Commit transakce provádí trvalé uložení všech objektů do databáze.

```
@Transactional
public void saveForm(Form form, String username, Integer questionIndex) {
  EntityManager entityManager = DbConnector.initEntityManager();
  if (!entityManager.getTransaction().isActive()) {
    entityManager.getTransaction().begin();
  }
  Form existingForm = null;
  if (form.getId() != null) {
    existingForm = entityManager.find(Form.class, form.getId());
    existingForm.setName(form.getName());
    existingForm.setDescription(form.getDescription());
    existingForm.setOwner(form.getOwner());
    existingForm.setQuestions(form.getQuestions());
  } else {
    existingForm = form;
    User owner = userDao.getForUsername(username);
    form.setOwner(owner);
    entityManager.persist(existingForm);
  }

  if (questionIndex != null) {
    saveQuestion(existingForm, form.getQuestions().get(questionIndex),
        entityManager);
  } else {
    for (Question question : form.getQuestions()) {
      saveQuestion(existingForm, question, entityManager);
    }
  }
  entityManager.getTransaction().commit();
}
```

Zdrojový kód 35 – metoda implementace perzistentní vrstvy pro uložení formuláře

Součástí metody pro uložení konkrétní otázky je za podmínky neexistujícího identifikátoru otázky pouze volání metody zajišťující vytvoření nového identifikátoru a uložení otázky včetně možností odpovědí, které otázka zahrnuje.

Ukládání možností odpovědí probíhá tak, že je určen nejdříve typ odpovědi, jenž je načítán z hodnoty vysouvacího seznamu typů odpovědí vygenerovaného na stránce a jeho hodnota je navázána na typ první možnosti odpovědi. Načtený typ bude následně přiřazován všem možnostem odpovědí obsaženým v otázce.

Pokud v tuto chvíli opomeneme ukládání typu mřížka, uložení všech možností odpovědí proběhne tím způsobem, že se každé z možností přiřadí typ a příslušná otázka a jednotlivé možnosti se uloží s automaticky vytvořenými identifikátory.

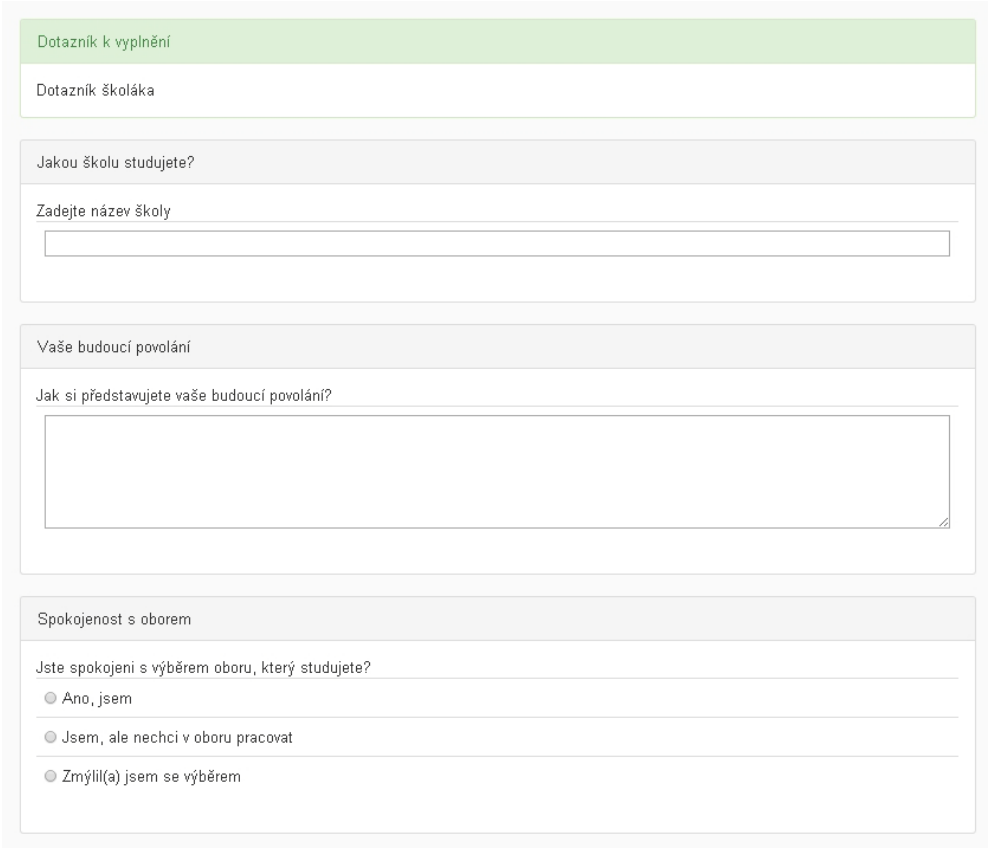
U typu mřížka je implementace o něco složitější z důvodu ukládání řádků a sloupců komponenty do jedné databázové struktury. Načtením hodnoty atributu *gridColumnCount* z první možnosti odpovědi se zjišťuje, jaký počet sloupců pro vytvoření mřížky byl na stránce vybrán. Dále je definováno počítadlo pořadí možností odpovědí. Následně se kontroluje, zda je hodnota počítadla pořadí menší než vybraný počet sloupců. Pokud je kontrola pozitivní, typu možnosti odpovědi bude přiřazen indikátor, který jasně udává, že se jedná o sloupec mřížky. Při negativním výsledku kontroly se přiřadí indikátor odkazující na řádek mřížky.

```
@Transactional
private void createQuestion(Question question, Form form, EntityManager entityManager) {
  question.setForm(form);
  entityManager.persist(question);
  if (question.getQuestionTypes() != null) {
    Type type = entityManager.find(Type.class,
      question.getQuestionTypes().get(0).getType().getId());
    Integer gridColumnCount = question.getQuestionTypes().get(0).getGridColumnCount();
    Integer questionTypesCount = 0;
    for (QuestionType questionType : question.getQuestionTypes()) {
      if (type.getId().equals(Type.GRID_COLUMN) || type.getId().equals(Type.GRID_ROW)) {
        if (questionTypesCount < gridColumnCount) {
          type = entityManager.find(Type.class, Type.GRID_COLUMN);
        } else {
          type = entityManager.find(Type.class, Type.GRID_ROW);
        }
      }
      questionType.setType(type);
      questionType.setQuestion(question);
      entityManager.persist(questionType);
      questionTypesCount++;
    }
  }
}
```

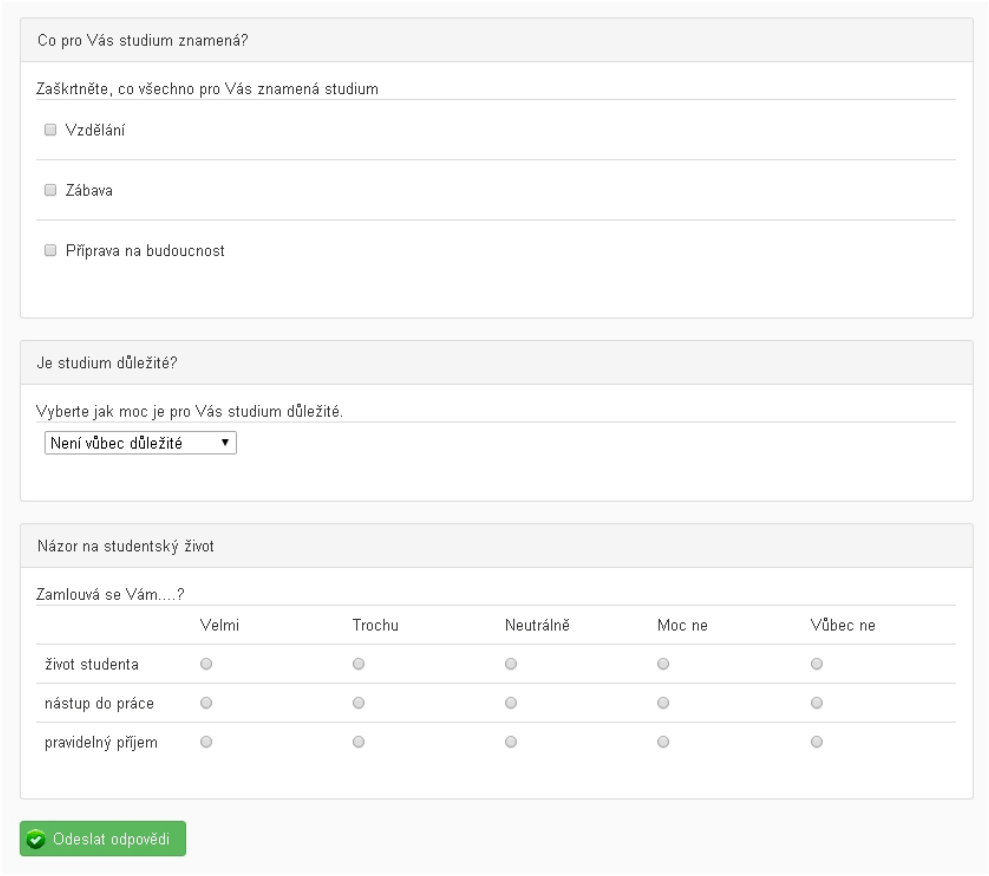
Zdrojový kód 36 – metoda implementace perzistentní vrstvy pro uložení otázky s možnostmi odpovědí

5.6.2 Vyplnění dotazníku

Formulář sloužící k vyplnění dotazníku využívá stejné komponenty, respektive jejich architekturu, které jsou již použity na stránce pro sestavení dotazníku. Pro vyplnění dotazníku musí být však tyto komponenty do značné míry přetransformovány do podoby, která umožňuje vyplnit odpovědi na jednotlivé otázky a zamezuje editaci samotných otázek.



Obrázek 15 – podoba všech dotazníkových komponent na formuláři pro vyplnění dotazníku – 1. část



Obrázek 16 - podoba všech dotazníkových komponent na formuláři pro vyplnění dotazníku – 2. část

5.6.2.1 Implementace v JSP

Celý webový formulář, jenž je zobrazen na stránce, obsahuje identifikátor a název dotazníkového formuláře vázajícího se na třídu *Form*, jednotlivé otázky se vstupy pro zadání odpovědí a tlačítko pro odeslání vyplněného formuláře na server.

Jednotlivé otázky jsou do tohoto webového formuláře vkládány pomocí tagu <*include* /> s odkazem na soubor *fillQuestionForm.jsp*, který obsahuje vykreslení jedné konkrétní otázky jakéhokoliv typu. Oproti formuláři na stránce pro sestavení dotazníku jsou zde zobrazeny jednotlivé položky, které otázka obsahuje, pouze v *read-only* podobě a jediné vstupní formulářové prvky, které lze editovat, jsou zde prvky pro vyplnění odpovědí.

```
<form method="post" action="fillForm.htm">

  <div class="subfieldsset">

    <form:hidden path="form.id" />
    <form:hidden path="form.name" />

    ${form.id} ${form.name}

    <c:set var="questionCount" value="0" />

    <div>
      <c:forEach var="question" items="${form.questions}">
        <%@include file="fillQuestionForm.jsp"%>
        <c:set var="questionCount" value="${questionCount + 1}" />
      </c:forEach>
      <c:if test="${!empty message}">
        <c:out value="${message}" />
      </c:if>
      <div>
        <input type="submit" class="btn-success fill"
          value="<spring:message code="form.toFill.saveAnswers"/>">
      </div>
    </div>
  </div>
</form>
```

Zdrojový kód 37 – JSP s formulářem pro vyplnění dotazníku

Následující ukázky kódu jsou obsahem již zmiňovaného vkládaného souboru *fillQuestionForm.jsp.*

Každá konstruovaná otázka je obalena tagem <*fieldset* /> s příslušnými CSS třídami. Součástí tohoto bloku je název otázky, skryté hodnoty identifikátoru, názvu a popisu otázky. Pro případ zobrazení typu mřížka je uchovávána i skrytá hodnota počtu

sloupců této otázky. Dále je zde zobrazen samotný popis otázky a do naznačené tabulky jsou pak generovány jednotlivé možnosti odpovědí, které jsou popsány níže.

```
<fieldset class="subfieldsset fillQuestion">
  <legend>${question.name}</legend>
  <form:hidden path="form.questions[${questionCount}].id" class="questionId" />
  <form:hidden path="form.questions[${questionCount}].name" />
  <form:hidden path="form.questions[${questionCount}].description" />
  <form:hidden path="form.questions[${questionCount}].questionTypes[0].gridColumnCount" />
   ${question.description}
  <table class="fillTable colapse">
```

Zdrojový kód 38 – uchování potřebných hodnot ve skrytých polích a vypsání otázky s jejím popisem

Pro sestavení otázek, respektive odpovědí typu krátký text, dlouhý text a zaškrtávací políčka slouží následující konstrukt, který dle požadovaného typu generuje v cyklu na základě počtu možností odpovědí buď jednoduché textové pole, textové pole pro zadání delšího textu anebo zaškrtávací políčka.

Jsou zde vytvořeny také skryté hodnoty identifikátorů a popisu jednotlivých možností odpovědí. Popis každé možnosti odpovědi je zároveň zobrazen vedle vygenerovaného vstupního prvku pro zadání odpovědi. V poslední řadě je definováno pole pro validační chyby, které zobrazuje uživateli validační zprávy, pokud nejsou při odeslání formuláře vyplněny některé požadované odpovědi. Jaké odpovědi, respektive jaká pole mají být při odesílání formuláře na server vyplněna je určeno již výše rozebraným validačním mechanismem.

```
<form:hidden path="form.questions[${questionCount}].questionTypes[0].type.id" />
<c:forEach var="questionTypes" items="${question.questionTypes}">
  <tr>
    <td>
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
      <c:if test="${question.questionTypes[questionTypesCount].type.id == 'SH_TEXT'}">
        <form:input
          path="form.questions[${questionCount}]
           .questionTypes[${questionTypesCount}].answers[0].value"
          cssClass="long" cssErrorClass="long error" />
      </c:if>
      <c:if
        test="${question.questionTypes[0].type.id == 'LNG_TEXT'}">
        <form:textarea
          path="form.questions[${questionCount}]
            .questionTypes[${questionTypesCount}].answers[0].value"
          rows="5" cssClass="long" cssErrorClass="long error">
        </form:textarea>
      </c:if>
      <c:if test="${question.questionTypes[0].type.id == 'CHCK_BOX'}">
        <form:checkbox
          path="form.questions[${questionCount}]
            .questionTypes[${questionTypesCount}].answers[0].value"
          value="${question.questionTypes[questionTypesCount].id}"
          cssErrorClass="error" />
      </c:if>
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description" />
      ${question.questionTypes[questionTypesCount] .description}
      <form:errors
        path="form.questions[${questionCount}]
         .questionTypes[${questionTypesCount}].answers[0].value"
        cssClass="required" />
    </td>
  </tr>
  <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
</c:forEach>
```

Zdrojový kód 39 – generování možností odpovědí typu krátký text, dlouhý text nebo zaškrtávací políčka

Níže uvedenou částí zdrojového kódu stránky je sestavena otázka s odpovědí typu vysouvací seznam. Generování otázek tohoto typu má podobnou koncepci jako výše popsané otázky. Využívá se však konstrukce pomocí značky *<form:select/>* a v ní zanořených značek *<form:option/>*, ze kterých je sestaven vysouvací seznam se všemi možnostmi odpovědí na danou otázku.

```
<tr>
  <td>
    <form:hidden path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
    <form:hidden path="form.questions[${questionCount}].questionTypes[0].type.id" />
    <c:forEach var="questionTypes" items="${question.questionTypes}">
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description" />
      <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
    </c:forEach>
    <c:set var="questionTypesCount" value="0" />
    <form:select
      path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].answers[0].value"
      cssErrorClass="error">
      <c:forEach var="questionTypes" items="${question.questionTypes}">
        <form:option value="${question.questionTypes[questionTypesCount].id}">
          ${question.questionTypes[questionTypesCount].description}
        </form:option>
        <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
      </c:forEach>
    </form:select>
    <form:errors
      path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].answers[0].value"
      cssClass="required" />
  </td>
</tr>
```

Zdrojový kód 40 - generování možnosti odpovědi typu vysouvací seznam

Další z typových možností, jak odpovědět na otázku, je pomocí výběru ze seznamu více odpovědí. Tento typ odpovědi je svým významem obdobou vysouvacího seznamu, kdy má uživatel možnost vybrat právě jednu možnou odpověď z několika daných. Visuální podoba výběru je však dosti odlišná.

V ukázce kódu je uveden značkou *<form: hidden />* skrytý identifikátor typu odpovědi na otázku. Dále se v cyklu generuje do řádků tabulky skrytý identifikátor otázky, samotné tlačítko pro výběr jedné z možných odpovědí, jedna z odpovědí na otázku a značka pro zobrazení případných validačních chyb.

```
<form:hidden path="form.questions[${questionCount}].questionTypes[0].type.id" />
<c:forEach var="questionTypes" items="${question.questionTypes}">
  <tr>
    <td>
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
      <form:radiobutton
        path="form.questions[${questionCount}].questionTypes[0].answers[0].value"
        value="${question.questionTypes[questionTypesCount].id}"
        cssErrorClass="error" />
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description" />
        ${question.questionTypes[questionTypesCount] .description}
      <form:errors
        path="form.questions[${questionCount}].questionTypes[0].answers[0].value"
        cssClass="required" />
    </td>
  </tr>
  <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
</c:forEach>
```

Zdrojový kód 41 - generování možnosti odpovědi typu jeden výběr ze seznamu

Pro vygenerování HTML kódu pro zobrazení části formuláře s odpovědí na otázku typu mřížka slouží mimo jiné následující bloky zdrojového kódu.

Na prvním z nich jsou využity ke zkonstruování dané komponenty v JSP stránce kromě JSTL knihoven také objekty jazyka Java. Sestavují se zde dva listy, z nichž jeden představuje sloupce komponenty a druhý její řádky.

```
<%
  List<QuestionType> columnList = new ArrayList<QuestionType>();
  List<QuestionType> rowList = new ArrayList<QuestionType>();
%>
<c:forEach var="questionType" items="${question.questionTypes}">
  <c:if test="${questionType.type.id == 'GRID_COL'}">
    <%
      columnList.add((QuestionType) pageContext.getAttribute("questionType"));
    %>
  </c:if>
  <c:if test="${questionType.type.id == 'GRID_ROW'}">
    <%
      rowList.add((QuestionType) pageContext.getAttribute("questionType"));
    %>
  </c:if>
</c:forEach>
<%
  pageContext.setAttribute("columnList", columnList);
  pageContext.setAttribute("rowList", rowList);
%>
```

Zdrojový kód 42 - příprava generování mřížky za pomocí vloženého Java-kódu do JSP

V následující ukázce se vytváří záhlaví komponenty, kde se v cyklu prochází sestavený list sloupců a popis každého z nich se propisuje do nultého řádku komponenty. Nultý řádek komponenty tudíž představuje škálu možných odpovědí na jednotlivé otázky.

```
<td class="empty">
</td>
<c:set var="questionTypeColCount" value="0" />
<c:forEach var="questionTypeCol" items="${columnList}">
  <td class="gridHeaderColumn">
    <form:hidden
      path="form.questions[${questionCount}].questionTypes[${questionTypeColCount}].id" />
    <form:hidden
      path="form.questions[${questionCount}].questionTypes[${questionTypeColCount}].description" />
    <form:hidden
      path="form.questions[${questionCount}].questionTypes[${questionTypeColCount}].type.id" />
    ${questionTypeCol.description}</td>
  <c:set var="questionTypeColCount" value="${questionTypeColCount + 1}" />
</c:forEach>
```

Zdrojový kód 43 – generování záhlaví mřížky

V dalším cyklu se po jedné položce zpracuje list řádků komponenty, přičemž se do nultého sloupce každého vykreslovaného řádku vkládá popis otázky a do prvního až posledního sloupce je generována značka *<form: radiobutton />*, která ve výsledku představuje tlačítko. To je součástí skupiny, ve které je možné stiskem vybrat právě jedno z nich. Jedna skupina tlačítek je zde vázána na jeden řádek. To znamená, že pokud se nebere v potaz záhlaví komponenty, pak komponenta obsahuje takový počet skupin, jaký je počet jejích řádků.

```
<c:set var="questionTypesCount" value="${questionTypeColCount}" />
<c:set var="questionTypeRowCount" value="0" />
<c:forEach var="questionTypeRow" items="${rowList}">
  <tr class="withHover">
    <td>
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].id" />
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].description" />
      <form:hidden
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}].type.id" />
      ${questionTypeRow.description}
    </td>
    <c:set var="questionTypeColCount" value="0" />
    <c:forEach var="questionTypeCol" items="${columnList}">
      <td class="gridColumn">
        <form:hidden
          path="form.questions[${questionCount}].questionTypes[${questionTypeColCount}].id" />
        <form:radiobutton
          path="form.questions[${questionCount}].questionTypes[${questionTypesCount}]
            .answers[0].value"
          value="${questionTypeCol.id}" cssErrorClass="error" />
      </td>
      <c:set var="questionTypeColCount" value="${questionTypeColCount + 1}" />
    </c:forEach>
    <td>
      <form:errors
        path="form.questions[${questionCount}].questionTypes[${questionTypesCount}]
          .answers[0].value"
        cssClass="required" />
    </td>
  </tr>
  <c:set var="questionTypeRowCount" value="${questionTypeRowCount + 1}" />
  <c:set var="questionTypesCount" value="${questionTypesCount + 1}" />
</c:forEach>
```

Zdrojový kód 44 – generování mřížky – tělo komponenty

5.6.2.2 Implementace v controlleru

Následující metoda, jež je součástí třídy *FillFormController*, zajišťuje odchycení validačních chyb při odeslání vyplněného formuláře uživatelem a předání potřebných parametrů metodě pro uložení formuláře.

Do metody přicházejí vstupní parametry, kterými jsou instance třídy *Form*, výsledky validační kontroly a model. V případě, že formulář obsahuje validační chyby, jsou tyto chyby do modelu přidány a je volán požadavek pro zobrazení stejné stránky s vyplněnými hodnotami, které uživatel zadal. Formulář na stránce bude zároveň obsahovat všechny validační chyby, které byly výstupem validační kontroly.

Pokud validační kontrola proběhne bez chyb, zavolá se metoda perzistentní vrstvy pro uložení zadaných hodnot a proběhne přesměrování uživatele na stránku s informací o úspěšném vyplnění formuláře.

```
@RequestMapping(method = RequestMethod.POST)
public ModelAndView save(@Validated Form form, BindingResult bindingResult, ModelMap model) {
  if (bindingResult.hasErrors()) {
    model.addAttribute("errors", bindingResult.getFieldErrors());
    return new ModelAndView("/fillForm", model);
  }
  String username = null;
  if (SecurityContextHolder.getContext().getAuthentication().getPrincipal() != ANONYMOUS_USER) {
    User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    username = user.getUsername();
  } else {
    username = ANONYMOUS_USER;
  }
  formDao.saveAnswersForForm(form, username);

  return new ModelAndView(new RedirectView("filledForm.htm")).addObject("formId", form.getId());
}
```

Zdrojový kód 45 - Uložení odpovědí (controller)

5.6.2.3 Implementace perzistentní vrstvy

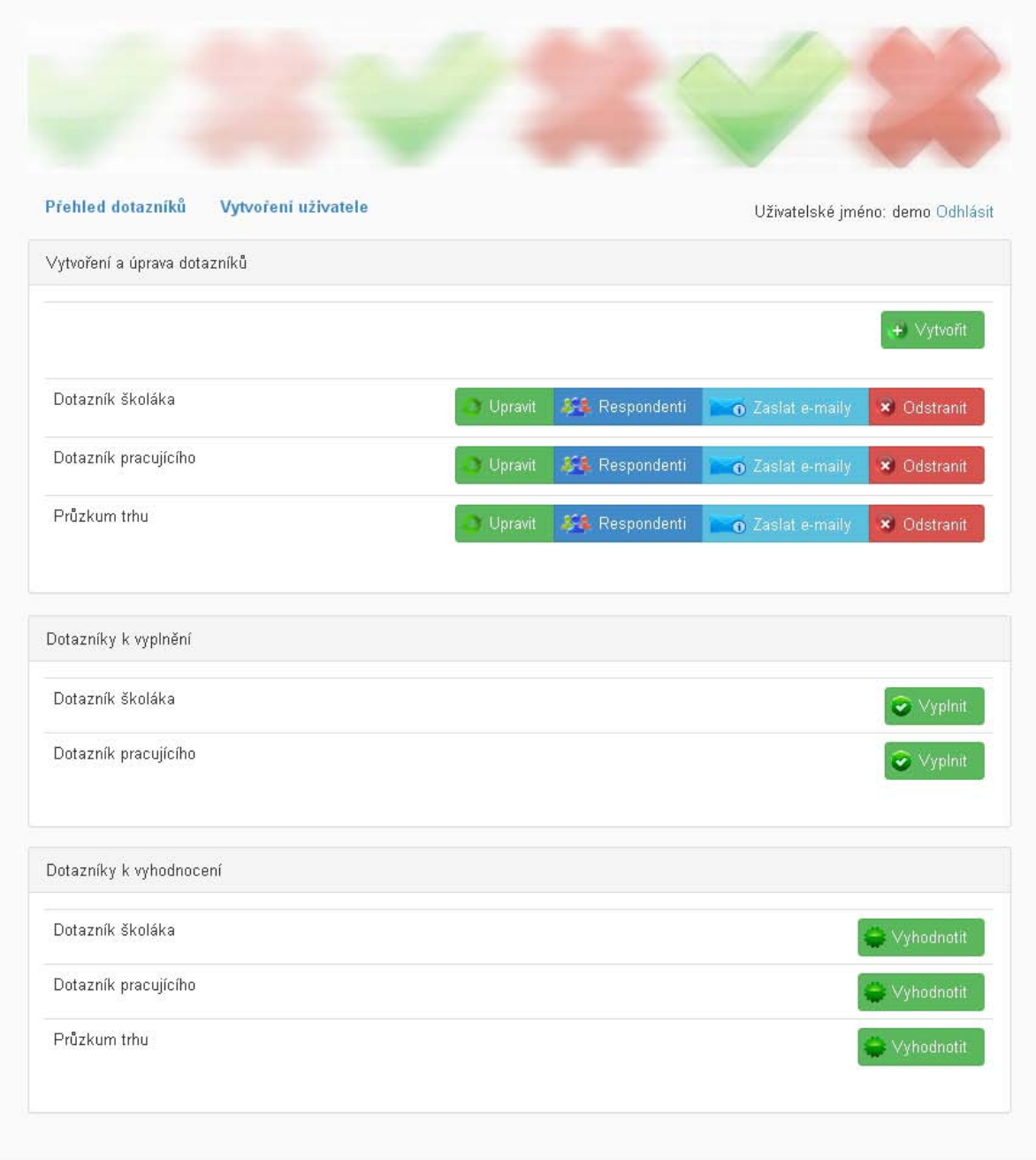
Součástí metody *saveAnswersForForm()* je kromě získání připojení k databázi v cyklu, který prochází všechny otázky, následující kód. Ukázka zajišťuje uložení všech odpovědí vyplněných uživatelem.

```
Type type = question.getQuestionTypes().get(0).getType();
for (QuestionType questionType : question.getQuestionTypes()) {
  if (questionType.getAnswers() != null && questionType.getAnswers().size() > 0) {
    QuestionType foundedQuestionType = questionTypeDao.getQuestionTypeById(questionType.getId());
    for (Answer answer : questionType.getAnswers()) {
      if (type.getId().equals("SELECT") || type.getId().equals("RAD_GR")) {
        foundedQuestionType = questionTypeDao.getQuestionTypeById(
          Long.parseLong(answer.getValue()));
      }
      answer.setQuestionType(foundedQuestionType);
      if (username != FillFormController.ANONYMOUS_USER) {
        answer.setUser(userDao.getForUsername(username));
      }
      answerDao.saveAnswer(answer);
    }
  }
}
```

Zdrojový kód 46 – Uložení odpovědí (perzistentní vrstva)

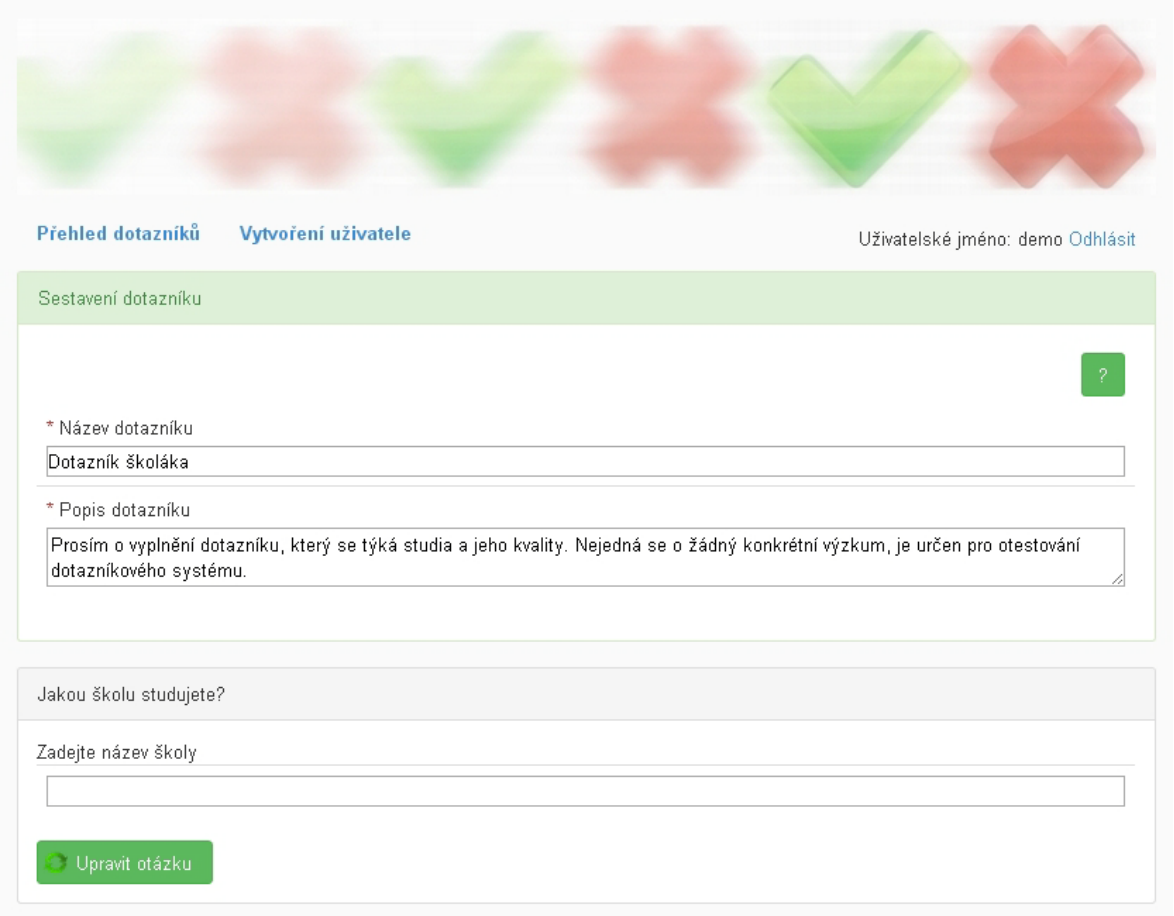
6 Výsledky práce

Následující obrázek představuje úvodní obrazovku aktuálně přihlášeného uživatele s přehledem všech dotazníků určených k úpravě, vyplnění a vyhodnocení.



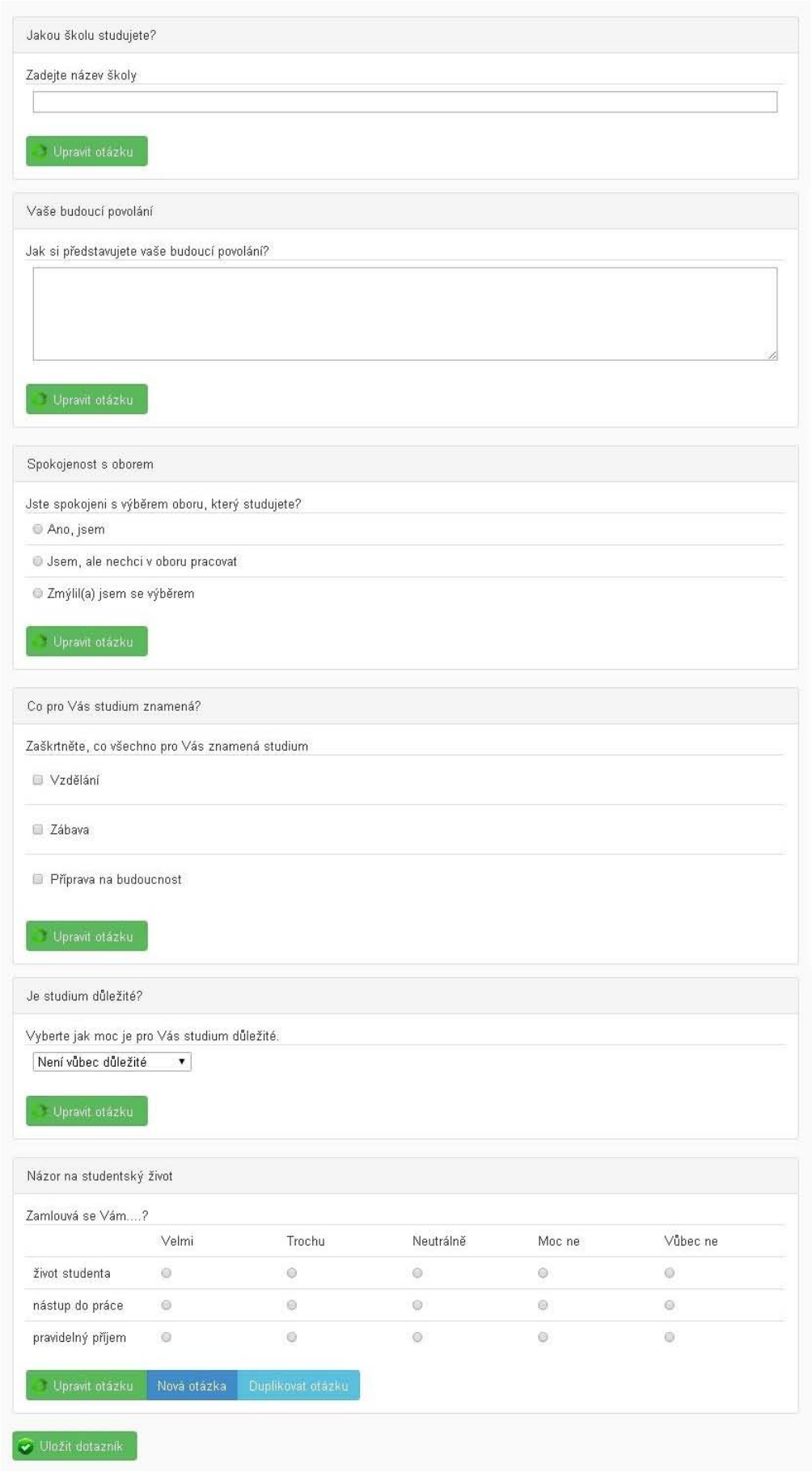
Obrázek 17 - Přehled přihlášeného uživatele

Obrazovka pro vytvoření či úpravu dotazníku obsahuje pole s názvem a popisem dotazníku, tlačítko s nápovědou a první prázdnou otázku. Zde jsou všechny zmiňované položky již vyplněny. Obsah obrazovky je dynamicky rozšiřován dle potřeb uživatele na základě počtu vytvořených otázek.



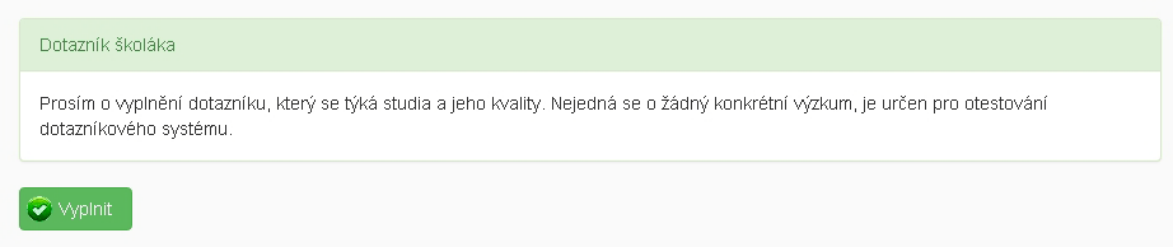
Obrázek 18 – výřez obrazovky pro vytvoření či úpravu dotazníku

Obrazovka obsahuje další uživatelem vytvořené bloky s možností úpravy jednotlivých otázek včetně typu odpovědi. Aplikace podporuje šest různých typů odpovědí na otázky. Na následujícím obrázku je pro ukázku vytvořeno šest otázek, přičemž každá je jiného typu. Novou otázku lze vytvořit buď prázdnou anebo pomocí duplikace poslední otázky na formuláři.



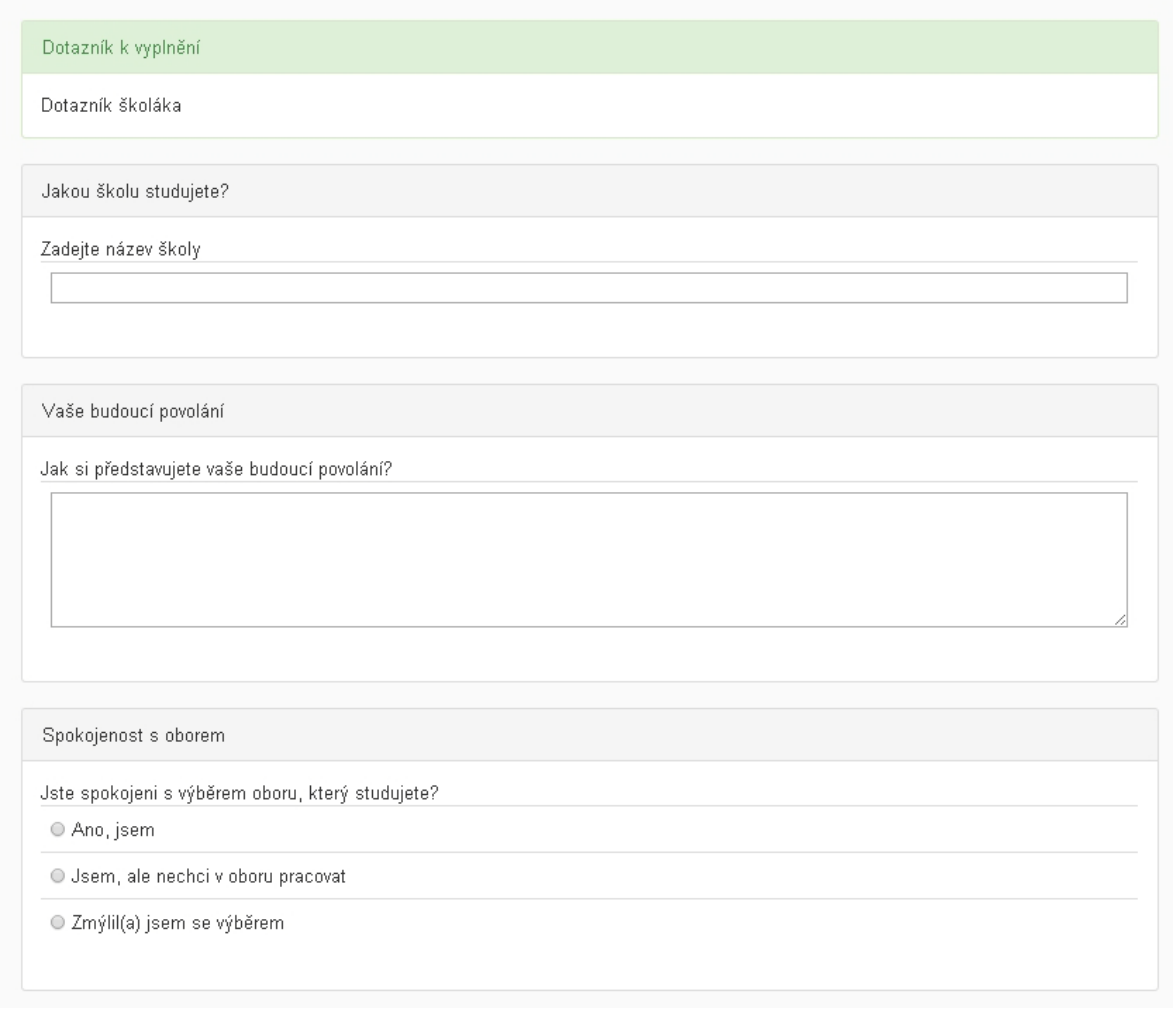
Obrázek 19 - formulář pro vytvoření a úpravu dotazníku

Po uložení vytvořeného dotazníku lze tomuto dotazníku přiřadit potencionální respondenty. Následně je možné hromadně zaslat systémem vygenerovanou emailovou zprávu s odkazem na následující obrazovku. Pokud zná uživatel tento odkaz, může vyplnit příslušný dotazník bez přihlášení do systému.



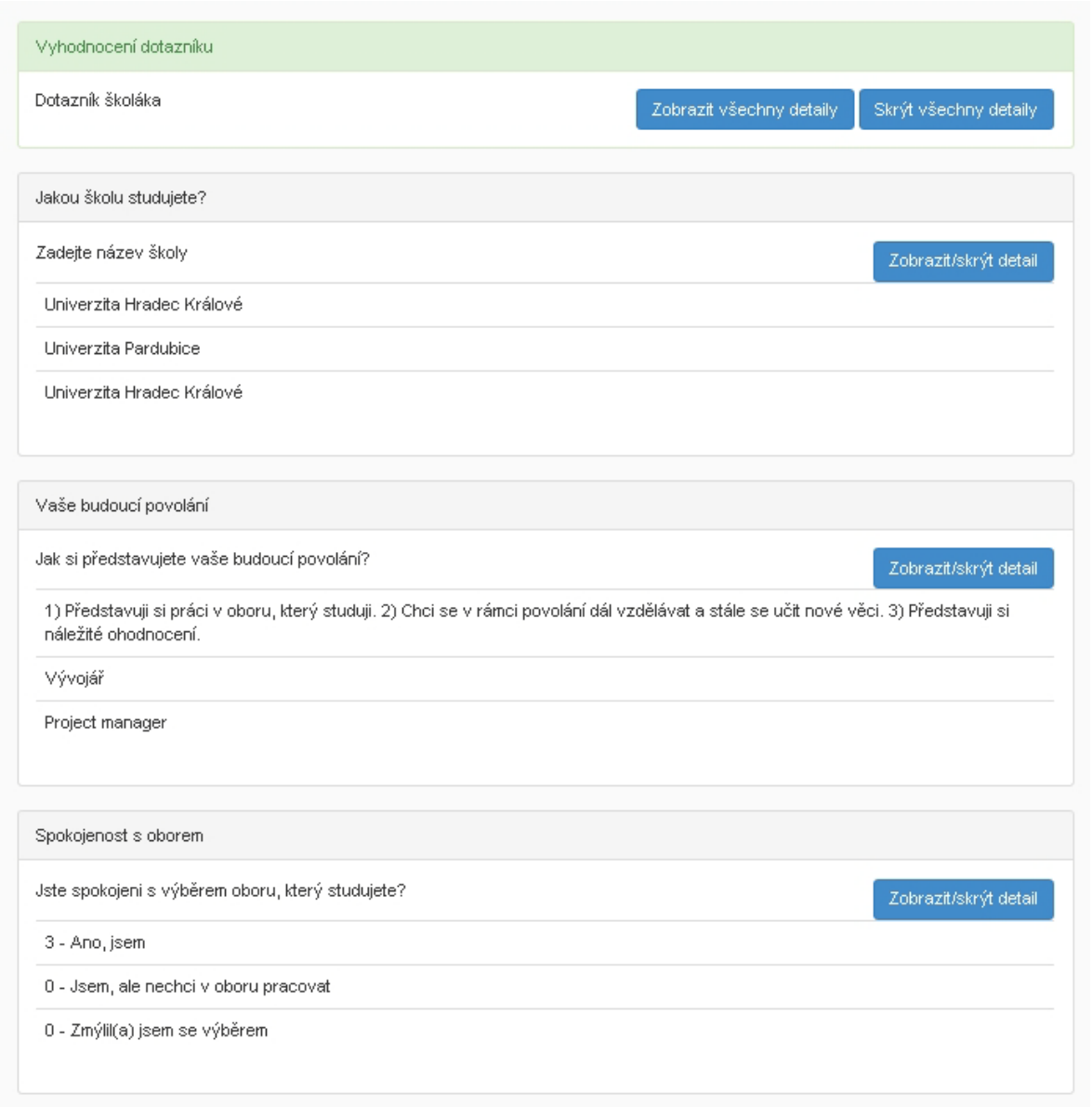
Obrázek 20 – Odkaz na vyplnění dotazníku (není potřeba přihlášení)

Po stisku tlačítka je uživatel přesměrován na formulář pro vyplnění dotazníku.



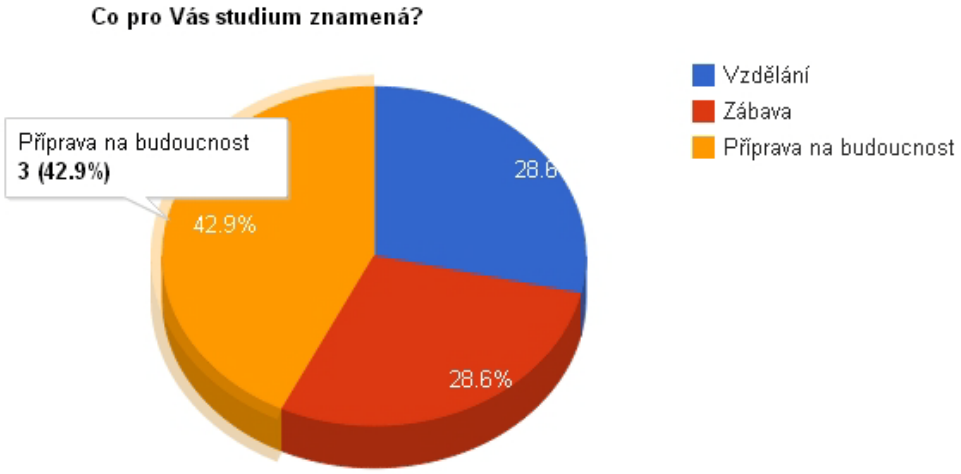
Obrázek 21 – Část formuláře pro vyplnění dotazníku

Při odeslání formuláře jsou okamžitě vyplněné informace ukládány do databáze. Ve chvíli, kdy jsou odpovědi uloženy, tvůrce dotazníku k nim má přístup přes obrazovku s vyhodnocením dotazníku.



Obrázek 22 - Vyhodnocení dotazníku

Po zobrazení detailu některé z otázek se vykreslí graf a tabulka se sumarizovanými daty. Ta jsou založena na počtu odpovědí uživatelů na konkrétní otázku.



Možnost odpovědi	Počet odpovědí
Vzdělání	2
Zábava	2
Příprava na budoucnost	3

Obrázek 23 - detail vyhodnocení jedné z otázek

6.1 Zhodnocení systému uživateli

Systém byl nasazen pro testovací účely na zdarma dostupný server a poskytnut k otestování několika uživatelům. Tito uživatelé se do detailu seznámili s dotazníkovým systémem a následně ho zhodnotili.

Dle poznatků běžných uživatelů internetových aplikací je systém přehledný, jednoduchý a nenáročný na ovládání. Uživatelé ocenili užitečnost nápověd a uvedli, že žádné významné nedostatky systému neshledali a systém by rádi využívali při svých výzkumech a studiích.

Několik uživatelů se dokonce zmínilo, že by se systém dal využít jako prostor pro veřejnou prezentaci univerzity a zároveň vyzdvihovali marketingový potenciál tohoto systému.

7 Závěr

V úvodní části práce byly zhodnoceny vlastnosti vybraných dotazníkových systémů, které jsou zdarma dostupné na internetu. Hodnocení proběhlo z pohledů dvou odlišně zdatných uživatelů webových aplikací. Samotná implementace dotazníkového systému byla založena na variantě přístupnější i méně zkušeným uživatelům. Rozhodování bylo realizováno pomocí tabulky kritérií, díky které bylo v teoretické rovině zjištěno, že dotazníkový systém inspirovaný systémem od společnosti Google bude použitelný pro nejširší možnou skupinu uživatelů.

Na základě hodnocení a rozebrání dílčích funkcionalit existujících dotazníkových systémů vznikal nový dotazníkový systém lépe vyhovující potřebám studentů Univerzity Hradec Králové.

Před samotnou implementací byla provedena analýza a návrh systému, který umožňuje správu dotazníkových výzkumů. Její součástí je datový model, diagram případů užití a analýza jednotlivých obrazovek vznikající aplikace. Dle ní by měl systém podporovat vytvoření dotazníku, přiřazení potencionálních respondentů, kterým je zaslána emailová zpráva s adresou daného dotazníku, vyplnění dotazníku respondenty a vyhodnocení výzkumu.

Systém, který vznikl jako součást této práce, byl vyvinut na platformě Java Enterprise Edition s podporou Spring Frameworku. Ten zajišťuje MVC architekturu aplikace a zabezpečení přístupu. Data systému jsou uchovávána v databázi databázového serveru MySQL. Prezentační vrstva je založena na JSP stránkách s využitím JSTL knihoven a Spring Framework tagů.

Aplikace podporuje šest různých typů otázek, resp. odpovědí. Pokud není vyplněno při vytváření dotazníku jakékoli pole potřebné k uložení dotazníku, uživateli se zobrazí příslušné validační hlášení. Po úspěšném uložení dotazníku může jeho tvůrce vybrat ze seznamu uživatele, které požádá zasláním emailové zprávy o vyplnění dotazníku. Emailová zpráva je systémem automaticky vygenerována a zaslána všem potencionálním respondentům. Výstupem dotazníkového výzkumu jsou sumarizovaná data od všech respondentů a základní grafy sestavené z těchto dat.

Cíl práce byl splněn v celém rozsahu. Systém byl nasazen na veřejný server, kde byl otestován několika běžnými uživateli internetových aplikací. Na základě jejich reakcí lze říci, že systém je zcela vyhovující pro vytváření a správu dotazníkových výzkumů. Tito uživatelé vyzdvihovali hlavně jeho přehlednost, jednoduchost a nenáročnost na ovládání.

Značnou výhodou může být fakt, že lze vlastní implementaci dotazníkového systému kdykoli rozšířit či upravit dle aktuálních potřeb studentů. Často využívané nyní zdarma dostupné dotazníkové systémy mohou být postupem času omezeny z hlediska dostupnosti množství funkcionalit nebo mohou být kompletně zpoplatněny. Proto by mohl být vytvořený dotazníkový systém v budoucnu velkým přínosem pro univerzitu. Systém by se dal využít i pro prezentaci a propagaci univerzity. Tím by se dalo docílit kupříkladu záměnou obrázku v záhlaví systému za komerční sdělení, které bude přes potencionální respondenty dotazníků šířeno mezi širokou veřejnost.

8 Zdroje

[1] Apache. Building a Project with Maven, (on-line). (citace červenec, 10., 2012). Přístup z Internetu: URL: http://maven.apache.org/run-maven/index.html

[2] Apress. *Expert Spring MVC and Web Flow*, 1 .vyd. 2006, 424 s. ISBN-10: 159059584X, ISBN-13: 978-1590595848

[3] Apress. *Spring Recipes: A Problem-Solution Approach*, 1 .vyd. 2008, 700 s. ISBN-10: 1590599799, ISBN-13: 978-1590599792

[4] David Winterfeldt. Spring by Example, (on-line). (citace červenec, 15., 2012). Přístup z Internetu: URL: http://www.springbyexample.org/pdf/SpringByExample.pdf

[5] Manning Publications. *Spring In Action*, 1 .vyd. 2005, 444 s. ISBN-10: 1932394354, ISBN-13: 978-1932394351

[6] Mkyong. Spring Tutorial, (on-line). (citace červenec, 17., 2012). Přístup z Internetu: URL: http://www.mkyong.com/tutorials/spring-tutorials/

[7] Mkyong. Spring MVC Tutorial, (on-line). (citace červenec, 22., 2012). Přístup z Internetu: URL: http://www.mkyong.com/tutorials/spring-mvc-tutorials/

[8] Mkyong. Spring Security Tutorial, (on-line). (citace srpen, 4., 2012). Přístup z Internetu: URL: http://www.mkyong.com/tutorials/spring-security-tutorials/

[9] Mkyong. Hibernate Tutorial, (on-line). (citace srpen, 12., 2012). Přístup z Internetu: URL: http://www.mkyong.com/tutorials/hibernate-tutorials/

[10] Mkyong. Java I/O Tutorial, (on-line). (citace září, 14., 2012). Přístup z Internetu: URL: http://www.mkyong.com/tutorials/java-io-tutorials/

[11] Oracle, MySQL 5.5 Reference Manual, (on-line). (citace červenec, 14., 2012). Přístup z Internetu: URL: http://dev.mysql.com/doc/refman/5.5/en/

[12] Red Hat, Inc. Hibernate Getting Started Guide, (on-line). (citace červenec, 18., 2012). Přístup z Internetu: URL: http://docs.jboss.org/hibernate/core/4.0/quickstart/en-US/html_single/

[13] SpringSource. Web MVC Framework, (on-line). (citace červenec, 15., 2012). Přístup z Internetu: URL: http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Lukáš Kmoníček**

Obor studia: Aplikovaná informatika (2)
Jméno a příjmení vedoucího práce: **Pavel Janečka**

Název práce:
Obecný dotazníkový systém

Název práce v AJ:
Universal questionnaire system

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je zhodnocení existujících dotazníkových systémů a na základě jejich analýzy provést návrh a implementaci vlastního řešení pro potřeby UHK.

Osnova práce:
Analýza existujících dotazníkových systémů
Specifikace nově vznikajícího řešení
Analýza a návrh aplikace
Implementace aplikace
Zhodnocení výsledků

Projednáno dne: 12.11.2011

Podpis studenta

Podpis vedoucího práce