Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Tvorba CRM aplikace v jazyce PHP za použití MVC architektury

Bakalářská práce

Vypracoval: Stanislav Pecka

Vedoucí práce: PhDr. Milan Novák, Ph.D.

České Budějovice 2014

Bibliografické údaje

Pecka S., 2013: Tvorba CRM aplikace v jazyce PHP za použití MVC architektury. [CRM application development in PHP language using MVC architecture. Bc.. Thesis, in Czech.] – 63 p. , Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotation

This bachelor's thesis deals with CRM systems development using PHP language. This work contains a complete analysis which is used as a base for system design and afterwards programming. Application design uses MVC architecture, which is described both generally and for web use. At the end I described methodology of setting the system on production server.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 4. dubna 2014 Podpis autora

Poděkování

Tímto bych chtěl poděkovat především svému vedoucímu práce za jeho trpělivost a odbornou pomoc. Dále děkuji společnosti Wedos Internet a.s. za poskytnutí serveru potřebného k nasazení mé práce do produkčního prostředí. Mé poděkování patří i firmě Orchitech Solutions s.r.o, kde jsem mohl své teoretické znalosti rozvíjet pomocí praxe. V neposlední řadě děkuji své rodině za poskytnutí možnosti studovat a přítelkyni za bezbřehou psychickou podporu.

Obsah

1. Úvod

V poslední době, kdy na trh přichází stále nová zařízení, která je možné využít pro provozování různých aplikací, je zaznamenávána velká poptávka po produktech, které na nich budou fungovat. Vzhledem k tomu, že je velmi složité napsat multiplatformní aplikaci, která bude na všech zařízeních fungovat stejně, jeví se jako řešení tohoto požadavku stále větší využití cloudových aplikací.

Cloud, čili aplikace běžící online, nás zbavuje nutnosti instalace na lokální zařízení, přitom přináší veškeré výhody víceuživatelských serverových aplikací.

Pro aplikace v cloudu je vhodné využití MVC architektury, aby jádro aplikace stále běželo na webu a na nás zbylo jen řešení různých způsobů zobrazení výstupů.

Mezi aplikace, které plně využijí možnosti MVC architektury, patří serverová řešení pro firmy požadující mít k aplikaci přístup z různých druhů zařízení. Takovou aplikací by například mohl být CRM systém, k němuž je třeba přístup z kanceláří firmy, obchodních míst, ale také na služebních cestách manažerů nebo při schůzkách obchodních zástupců.

CRM systém je aplikace, která se stává velmi důležitou součástí softwarového vybavení firem. V poslední době se ukázalo, že pokročilými marketingovými metodami lze zákazníka na produkt firmy nalákat, ale je těžké ho přimět k loajalitě vůči firmě a jeho zavázání k dalšímu nákupu. CRM systém umožňuje udržovat a zpracovávat informace o zákaznících, pravidelně je informovat o novém zboží, akčních nabídkách apod., a to cíleně na konkrétního zákazníka na základě jeho provedených nákupů.

2. Cíle práce

Hlavním cílem práce je vytvoření funkční aplikace CRM systému za použití MVC architektury. Tato aplikace bude následně nasazena na produkční prostředí. V rámci nasazení bude popsán postup spuštění aplikace na serveru Apache v linuxovém prostředí.
Aby tento hlavní cíl mohl být splněn, musely být stanoveny dílčí cíle:

- Konstrukce aplikace bude definována na základě analýzy, v rámci které proběhne dotazníkové šetření a bude provedena obsahová analýza současně nabízených CRM systémů. Šetření bude zaměřeno na zjištění preferencí uživatelů, a to na vlastnosti CRM systému a typy využívaných modulů. Obsahová analýza bude klást důraz na nabízené moduly systému.

- Z nejpoužívanějších modulů vzešlých z analýzy bude vybrán jeden, který bude v rámci práce realizován. Spolu s ním bude naprogramován také modul pro uživatele, jenž je nedílnou součástí podobných systémů.

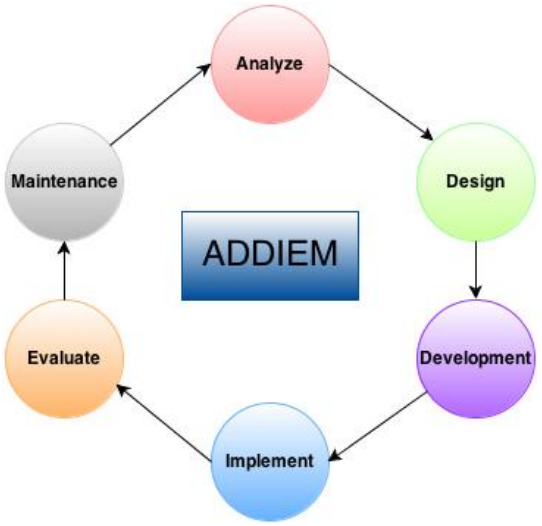
- Návrh struktury a rozvržení aplikace bude proveden pomocí diagramů případů užití a tříd, které výstižně popisují konstrukci a logickou funkčnost aplikace.

- Součástí práce bude kompletní dokumentace kódu pro případné zájemce o další vývoj vytvořené aplikace.

3. Metoda práce

Při realizaci projektu je snahou dodržet vývojový model ADDIE. Tento model je používán především pro návrh elektronických materiálů, nicméně vzhledem k distribučnímu médiu v podobě internetu zde lze spatřovat jistou podobnost. Proto je model ADDIE, resp. jeho inovační podoba ADDIEM, možné uplatnit i při samotné realizaci projektu [14]. Takto rozšířený model ADDIEM představuje jednotlivé fáze vývoje definované počátečními písmeny zkratky:

- **Analyze** (analýza) - ujasnění požadavků na projekt, shromažďování informací. V této fázi vývoje je zapotřebí velká spolupráce ze strany klienta, aby výsledný produkt splňoval všechny požadavky. V této práci bude klienta suplovat dotazníkový průzkum a analýza stávajících CRM systémů.
- **Design** (návrh) - jde o návrh řešení dle požadavků vyplývajících z předešlé analýzy. Do této fáze spadají use-case diagramy, wireframy apod.
- **Development** (vývoj) - vlastní vytváření projektů za použití programovacího jazyka, provádění testů na ověření funkčnosti dle návrhu.



Obrázek 3.1: Diagram modelu ADDIEM.

- **Implement** (nasazení) - nasazení projektu na produkční prostředí.
- **Evaluate** (hodnocení) - hodnocení projektu, z kterého se plynule přechází do analýzy nových požadavků, případně řešení nalezených nedostatků.
- **Maintenance** (údržba) - tato část bývá někdy vynechávána, jelikož se do určité míry překrývá s předchozí fází vývoje. Narozdíl od fáze hodnocení se ve fázi údržby neurčují nové požadavky, ale udržuje se již hotový projekt, aby stále plnohodnotně splňoval svou funkci. Mezi takovou údržbu patří např. u webových stránek SEO, u účetního software legislativní aktualizace apod.

Při vývoji se postupuje ve stejné posloupnosti úkonů, jako je pořadí slov (viz. obr. 3.1) [13]. Samozřejmě, že existují jeho inovace, aby lépe vyjadřoval proces

vývoje. Zejména se jedná o kontrolní mechanizmy. Při vývoji software podle ADDIE modelu je zapotřebí, na rozdíl od vytváření výukových materiálů, provádět hodnocení již v průběhu navrhování a vývoje aplikace, a to formou testů (správná funkčnost) a diskuze se zákazníkem (naplnění představ). Vývoj software je totiž natolik drahá záležitost, že by hodnocení až v konečné fázi vývoje přinášelo velké finanční následky.

4. Úvod do problematiky

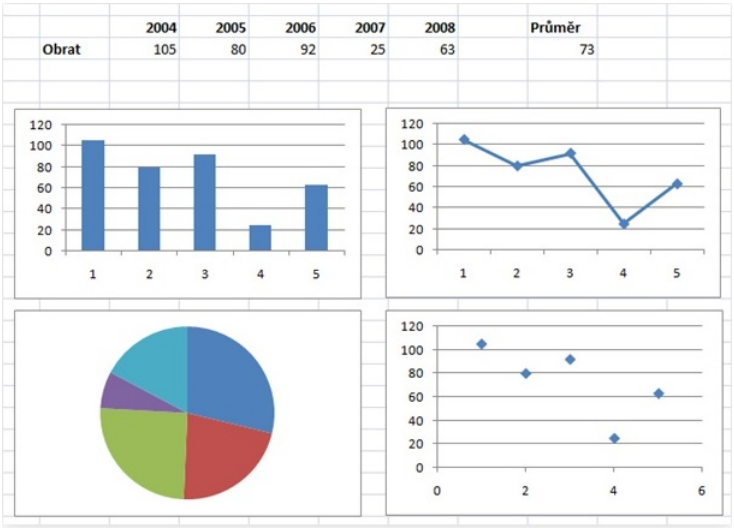
4.1 Architektura MVC

Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně zcela bez dopadu na ostatní části. Tyto tři logické celky jsou:

- **model** - reprezentuje data a business logiku aplikace,
- **view** - zobrazuje uživatelské rozhraní,
- **controller** - řídí tok událostí v aplikaci a obstarává veškerou aplikační logiku.

V odborné literatuře je architektura vysvětlena příkladem na obr. 4.1, kde se jako ukázkového modelu využívá sešitu v programu Excel [7].

Model je datová struktura, která uchovává čísla 105, 80, atd. V aplikaci může být realizována jako jakákoliv datová struktura (pole čísel, proměnné nebo třída). Na příkladu je také uvedený průměr - i tento výpočet je součástí modelu. Zcela netypicky by mohl být model sadou datových objektů bez business logiky. Model ve smyslu MVC je ale tzv. doménový model, který modeluje vztahy reálného světa. V něm jsou kromě dat důležitá také business pravidla, nebo validační pravidla. Důvod je ten, že jednotlivá pravidla potřebujeme ve více pohledech současně, což by v případě uložení ve view vedlo k duplikaci kódu. Ačkoli se o modelu mluví v jednotném čísle, je většinou realizován více objekty.



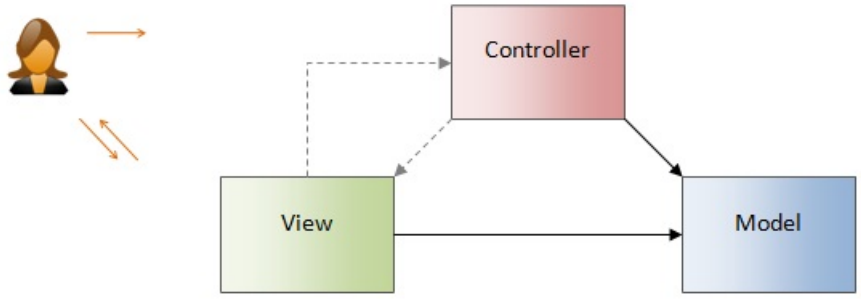
Obrázek 4.1: Příklad MVC architektury uvedený na tabulce v Excelu.

View je v obrázku hned několik - grafy jsou naprosto zřejmým zástupcem této části, ale i samotná tabulka se zobrazenými čísly je dalším view, protože jí mohu určit, aby zobrazovala čísla se znakem měny atp. View je tedy zobrazením modelu a dalších prvků uživatelského rozhraní.

Controller je nejhůře představitelnou součástí uvedeného příkladu. Je to např. reakce na uživatelovu úpravu buňky - controller v tu chvílí musí obstarat aktualizaci modelu, nový výpočet průměru a překreslení všech view. To, jak se view překreslí, ale není záležitost controlleru, ten pouze tuto akci spouští. Controller je tedy ústřední jednotka, která se stará o provázání funkčnosti aplikace.

Návaznost jednotlivých komponent ilustruje obrázek 4.2. Controller má přímou vazbu na model, aby mohl upravovat jeho data. View má přímý odkaz na model, aby mohl jeho data zobrazit. V praxi je pak poměrně častá vazba mezi controllerem a view.

Co nikdy nesmí existovat, je přímá vazba modelu na některou z dalších komponent. To by byla hrubá chyba v návrhu aplikace. View mohou být upozorněna na změnu dat z modelu nějakým notifikačním mechanismem (např. vzor Observer[1]), ale model nesmí mít přímou vazbu na view.



Obrázek 4.2: Architektura MVC - návaznost jednotlivých komponent a interakce s uživatelem.

Pokud do architektury přiřadíme ještě uživatele, dostáváme se k malé komplikaci - výstup aplikace má na starosti vždy view, ale složitější to je s uživatelovým vstupem. Zde máme 2 odlišné možnosti závislé na typu systému:

1. u **widgetových systémů**[2] umí vstup ošetřit komponenty samy (např. tlačítko reaguje na událost Click, textové pole zachytává psaný text apod.). Zde tedy vstup zachycuje view,
2. u **newidgetového systému**, kde žádné komponenty neexistují, zpracovává uživatelský vstup controller.

Činost aplikace založené na architektuře MVC tedy vypadá následovně:

1. uživatel vykoná akci na uživatelském rozhraní,
2. controller tuto akci zachytí,
3. následně controller rozhodne, jak na akci reagovat - obvykle se změní hodnoty v modelu, nebo se přímo ovlivní view,
4. view zobrazí změny uživateli.

Tento koloběh se poté opakuje.

[1]Návrhový vzor Observer umožňuje objektu spravovat řadu pozorovatelů, kteří reagují na změnu jeho stavu voláním svých metod [19].

[2]komponentové frameworky typu Java Swing, Windows Forms, WPF, Silverlight, Flex, ASP.NET Web Forms, PRADO v PHP apod.

4.2 Komponenty MVC v prostředí webu

Pro MVC architekturu používanou ve webovém prostředí byly převzaty vzory z desktopových aplikací. Tím jsou struktura a chování aplikace velmi podobné chování na desktopu.

Rozdíl můžeme nalézt pouze u View komponenty, a to při zapojení AJAXu do zobrazování výstupů. Zatímco u klasické aplikace, která prezentační logiku zpracovává na serveru, se považuje prohlížeč jen za nástroj k zobrazení generovaného výstupu, u AJAXové aplikace se prezentantační logika přesouvá na klienta. Při tomto přesunu se počítá s tím, že se klient zapojí do fungování architektury prostřednictvím JavaScriptu.

V současné době existuje velmi málo plně AJAXových aplikací, nicméně se AJAXu využívá pro interaktivní zobrazení výstupu. To je ale jen doplněk serverového MVC, protože se v JavaScriptu nezpracovává komplikovaná logika, která se řeší na serveru [8].

Pro serverové MVC tedy platí, že:

- **Model** je identický s modelem v desktopových aplikacích.
- **View** je serverový kód, který se stará o generování požadovaného výstupu (např. HTML, XML, JSON apod.).
- **Controller** je složen z více částí. Hlavní z nich je Front Controller, který zachytává všechny HTTP požadavky, ty zpracuje a odesílá dalším controllerům. Konkrétní controller přijme data pocházející původně z HTTP požadavku, uloží je do modelu a s ním prováže konkrétní view, které se postará o vyrenderování výstupu.

4.3 Rozdíly architektury MVC oproti MVP

Ačkoli jsou si tyto architektury velmi podobné, některá literatura je dokonce označuje za totožné [11], několik rozdílů mezi nimi přece jen najdeme. Nejpodstatnějším rozdílem u MVP je přímá vazba View na Presenter. To znamená, že View přímo volá metody Presenteru, čímž může být na View plně obsloužen vstup i výstup pro uživatele, zatímco v případě MVC View předává vstupy Controlleru a ten je následně zpracovává.

Někteří autoři tvrdí, že vazba mezi View a Presenterem je natolik pevná, že by bylo možné Presenter vypustit a udržovat jen monolitické View. Nicméně se zachovává plná MVP architektura z důvodu lepší udržitelnosti kódu [8].

Je tedy třeba tyto rozdíly vzít na zřetel a dodržovat strukturu a vazby zvolené architektury.

5. Analýza

Prvním krokem k realizaci stanovených cílů byla provedena analýza, a to zejména dotazníkovým šetřením, které bylo provedeno podle obsahové analýzy stávajících CRM systémů.

Podle dostupné literatury by funkcionalita měla vycházet z obchodního cyklu, ve kterém se prochází následujícími 4 fázemi [3]:

- **Oslovení zákazníka** - navázání konraktu se zákazníkem. Patří sem různé marketingové analýzy, kampaně a podpůrné nástroje pro kanály, skrz které je zákazník kontaktován.
- **Obchodní transakce** - příprava a dohodnutí kontraktu se zákazníkem.
- Plnění objednávek - dochází k plnění kontraktů. Patří sem dohled nad fakturací, evidence práce, dohled nad projekty atp.
- **Zákaznický servis** - podpora v návaznosti na zakoupené služby nebo zboží. Firma v této fázi posiluje vztah se zákazníkem a nabízí další produkty a služby, čímž se víceméně dostává zpět do fáze Oslovení zákazníka.

Fáze	**Moduly**
Oslovení zákazníka	Marketingové analýzy Řízení kampaní E-marketing Řízení obchodních příležitostí
Obchodní transakce	Prodejní analýzy Řízení prodejních aktivit a řízení kontaktů Řízení projektů Obchod prostřednictvím kontaktního centra Prodej na místě zákazníka E-prodej Stanovení cen a konfigurace produktů na internetu Získání objednávky
Plnění objednávek	Analýza plnění Řízení logistiky Credit management Fakturace
Zákaznický servis	Analýza servisu Péče o zákazníky a HelpDesk Řízení smluv a instalované báze Servis v terénu a dodání produktů E-servis

Tabulka 5.1: Moduly CRM členěné podle částí CRM

Vychází se z tabulky (viz tabulka 5.1 [1]), ve které je popsána funkcionalita (potažmo moduly) rozčleněná právě podle obchodních fází.

5.1 Dotazníkové šetření

Pro zjištění požadavků firem působících v České republice na CRM systém byl sestaven formulář (k náhlédnutí v příloze práce na CD). Aby byly zjištěny jen požadavky těch firem, které CRM skutečně používají, byli vyřazeni respondenti, kteří na otázku, zda používají CRM systém, odpověděli záporně.

Z výše uvedených tezí a z vlastní analýzy nabízených modulů u vybraných CRM systémů (viz tabulka 5.2) byly pro otázku, jaké moduly firmy používají, vybrány moduly nejčastěji se vyskytující v nabízených CRM systémech.

Název systému	**Výrobce**
raynet Cloud CRM	RAYNET s.r.o.
Vistos CRM	Euro Softworks s.r.o.
SugarCRM	SugarCRM Inc.
CRM free	TECHNODAT Develop, s.r.o.
TeamOnline	TeamOnline a.s.
EQIS CRM	Astra Systems, s.r.o.
vtiger	vTiger
BLUEJET	COMPEKON s.r.o.
eWay	eWay System s.r.o.
intouch CRM	ANNECA, spol. s.r.o.
Smart CRM	SMARTCRM GmbH

Tabulka 5.2: Analyzované CRM systémy

Pro zjištění požadavků na CRM byli respondenti dotazováni na preferenci instalace, dostupnost systému, jazykovou mutaci. V rámci dohledatelnosti změn byla položena otázka, zda požadují, aby každý uživatel měl svůj účet a jestli požadují, aby každá změna dat byla zaznamenána.

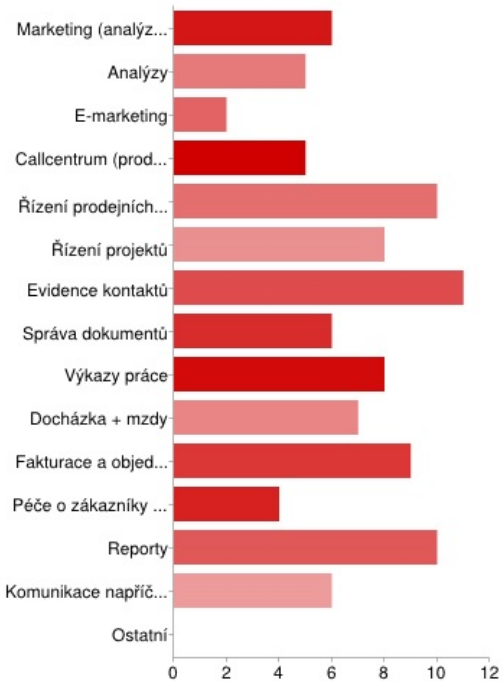
Dále bylo zjišťováno, zda firmy využívají rozšiřování stávajícího systému pomocí modulů, nebo jestli si nechávají systém přeprogramovat dle nových potřeb a požadavků.

Dotazník byl elektronicky zaslán 52 firmám provozujících svou činnost v České republice. Mezi firmami byly zastoupeny reklamní agentury, dále firmy zabývající se vývojem software, nejvýznamější e-shopy, telefonní operátoři, energetické skupiny, poskytovatelé webových hostingů a společnosti zabývající se zjišťováním veřejného mínění. Na dotazník odpovědělo 18 firem.

5.2 Výsledky šetření

Z šetření vyplynulo, že většina oslovených firem (83 %) využívá pro svůj chod CRM systém. U 60 % firem je dokonce systém vytvořený na zakázku, to znamená, že není standardní nabídkou dodavatele.

Zjištěna byla preference instalace - 53 % respondentů upřednostňuje serverovou instalaci s klienty na lokálních počítačích a 33 % si vybere systém bez instalace provozovaný na serverech poskytovatele a dostupný online. S tím také koresponduje požadavek na přístup k CRM systému odkudkoliv, což zvolila 93 % firem. Celkem 73 % dotázaných firem považuje jako důležitou možnost rozšíření



Marketing (analýzy + řízení kampaní)	6	6 %
Analýzy	5	5 %
E-marketing	2	2 %
Callcentrum (prodej služeb nebo produktů)	5	5 %
Řízení prodejních aktivit	10	10 %
Řízení projektů	8	8 %
Evidence kontaktů	11	11 %
Správa dokumentů	6	6 %
Výkazy práce	8	8 %
Docházka + mzdy	7	7 %
Fakturace a objednávky	9	9 %
Péče o zákazníky + Helpdesk	4	4 %
Reporty	10	10 %
Komunikace napříč odděleními	6	6 %
Ostatní	0	0 %

Obrázek 5.1: Nejčastěji využívané moduly CRM systému

systému moduly. Jen 20 % si nechá chybějící funkce doprogramovat na zakázku.

Absolutní samozřejmostí je vlastní účet pro každého zaměstnance, stejně jako dohledání všech změn, které uživatel provedl - všechny firmy se vyslovily pro. Naopak pro firmy není důležité, aby každý uživatel mohl mít nastaven vlastní jazyk - 60 % dotázaných má nastavený jeden jazyk pro všechny uživatele. Vzhledem ke skutečnosti, že byly dotazovány firmy fungující v České republice, není překvapující, že 70% zastoupení má čeština, zbylých 30 % je angličtina.

Z šetření dále vyplynulo, že kromě modulu Fakturace a objednávky, Evidence kontaktů a Reportů je jedním z nejpoužívanějších modulů Řízení projektů (viz obr. 5.1). Tento modul byl tedy vybrán pro realizaci v rámci této práce, jelikož je modul dostatečně komplexní pro předvedení MVC architektury. Další často používané moduly jako Fakturace nebo Evidence kontaktů jsou navíc mnohokrát zpracovány i jako samostatné aplikace a nejsou tolik typické pro CRM systém, jako Řízení projektu.

Detailní výsledky šetření v grafech jsou uvedeny v příloze práce na CD.

5.3 Požadavky na aplikaci

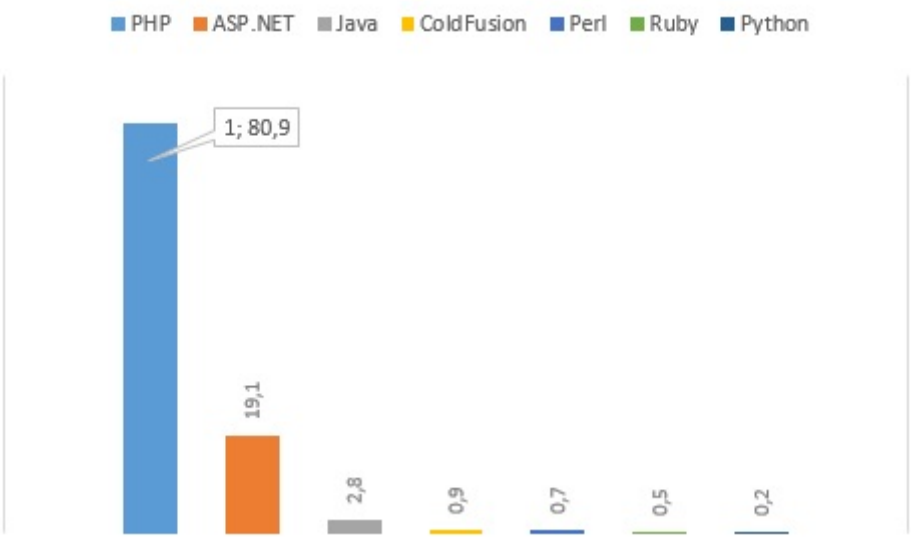
V závislosti na výsledcích dotazníkového šetření byly stanoveny následující požadavky na aplikaci:

- Přístup k aplikaci 24/7 odkudkoliv
- Běh na serveru - bez instalace klienta na lokální PC
- Vícejazyčnost - při nasazení se zvolí jazyk dle preferencí zákazníka
  - Angličtina
  - Čeština
- Každý uživatel musí mít svůj účet, autentizace realizována pomocí uživatelského jména a zvoleného hesla
- Role uživatelů - každý uživatel bude mít jiná přístupová práva
- Logování událostí - sledování změn dat v aplikaci provedených jednotlivými uživateli
- Rozšiřitelnost pomocí modulů

6. Návrh

6.1 Prostředí

Pro běh aplikace bylo zvoleno webové prostředí. Aplikace bude programována v jazyce PHP ve verzi 5.4, jako databázový systém byla zvolena databáze PostgreSQL. Ačkoliv se ve spojení s PHP automaticky nabízí MySQL, rámci tohoto projektu padla volba na PostgreSQL. Tento databázový systém sice za MySQL zaostává v rychlosti při běžných read/write operacích s malým množstvím dat (hlavně kvůli cache použité v MySQL), ale PostgreSQL dokáže být mnohem rychlejší při velkém množství záznamů v databázi, což se u systému CRM dá předpokládat. Navíc absenci cache nahrazuje Doctrine, který je součástí Symfony (viz dále).



Obrázek 6.1: Srovnání procentuálního využití jazyků PHP a ASP.NET na webových stránkách využívajících server-side jazyky k 7. září 2013.

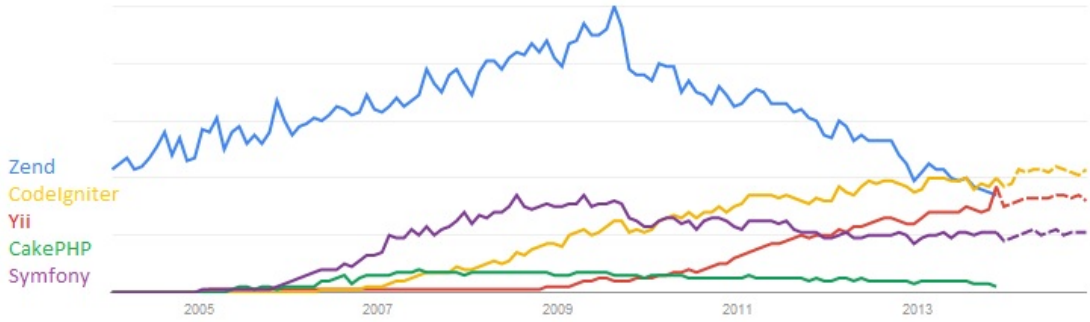
Důvodem, proč dát přednost jazyku PHP před jinými, je jeho výkon a jednoduchá spolupráce s databázovými systémy. Výhodou distribuce jako open-source[1] je na internetu velká podpora komunity. Od svého uvedení na trh zaznamenává PHP exponenciální růst a v roce 2004 předstihl konkurenční technologii ASP, dnes ASP.NET[2] [4] a toto prvenství si drží dodnes, což je velmi zřetelné na obr. 6.1[21].

Z důvodu potřeby snadno udržitelného kódu bude pro práci využit MVC framework. Při volbě konkrétního frameworku jsem vycházel z výběru 5 nejlepších frameworků podle Garga [9], jenž se shoduje se serverem Indiesphp.com [12],

[1] Open-source software je takový software, k němuž zákazník dostane od jeho tvůrce zdrojový kód a může jej dále upravovat [18].

[2] ASP.NET je součást .NET Frameworku pro tvorbu webových aplikací vyvíjeného a podporovaného společností Microsoft [5].

kterými jsou CakePHP, CodeIgniter, Symfony, Yii a Zend. Všechny splňují požadavek na podporu MVC architektury, vícejazyčnost, modularitu a nezávislost na konkrétním databázovém systému pro snadné nasazení na server [15], nicméně s ohledem na skutečnost, že CodeIgniter a Yii nebyly delší dobu aktualizovány, CodeIgniter nepodporuje ORM, zájem vývojářů o CakePHP a Zend podle Google Trends [10] prudce klesá (viz. obr. 6.2), a že pro nás je důležitější spolehlivost než rychlost (Yii a Zend), padla volba na framework Symfony.



Obrázek 6.2: Zájem o PHP frameworky podle procentuálního zastoupení ve vyhledávání Google v období od roku 2004 do listopadu 2013. Čárkovaně je vyznačena předpověď pro rok 2014.

Integrace ORM[3] Doctrine 2 ve frameworku nás v programu oprostí od psaní SQL dotazů, čímž se nám opět kód zpřehlední a neduplikuje na více místech. Díky direktivám v anotacích (které patří mezi nejmodernější přístupy k programování) také nebudeme muset vytvářet schémata v databázi, jelikož je za nás vygeneruje Doctrine z modelu.

Doctrine nám také ve spojení s PostgreSQL umožňuje zbavit se spojovacích tabulek v databázi - v pří-padě referencí typu „many-to-many“ se data uloží jako pole serializované do textového řetězce. Výhoda je zjevná: vystačíme si se 2 tabulkami místo 3. Ačkoliv Beckingham [6] považuje tuto metodu jako špatnou databázovou techniku, vhod-ným ošetřením v programu a využitím Doctrine se dosáhne značného zjednodušení práce s daty v databázi, zvláště pokud se jedná o data stejného typu.

Oddělení view vrstvy od modelu a controlleru umožní integrace šablonovacího systému Twig. Tento systém se vyznačuje jednoduchou syntaxí, která nám zjednoduší zápis HTML kódu a práci s daty získanými z controlleru.

Mimo jiné nám také Symfony umožní plynulý běh aplikace na serveru využitím cachovacího systému - při prvním načtení aplikace si Symfony do cache uloží vygenerované view ze šablony, dotazy generované Doctrine apod., aby je nemusel načítat při každém zavolání. Tím se velmi urychlí další volání aplikace.

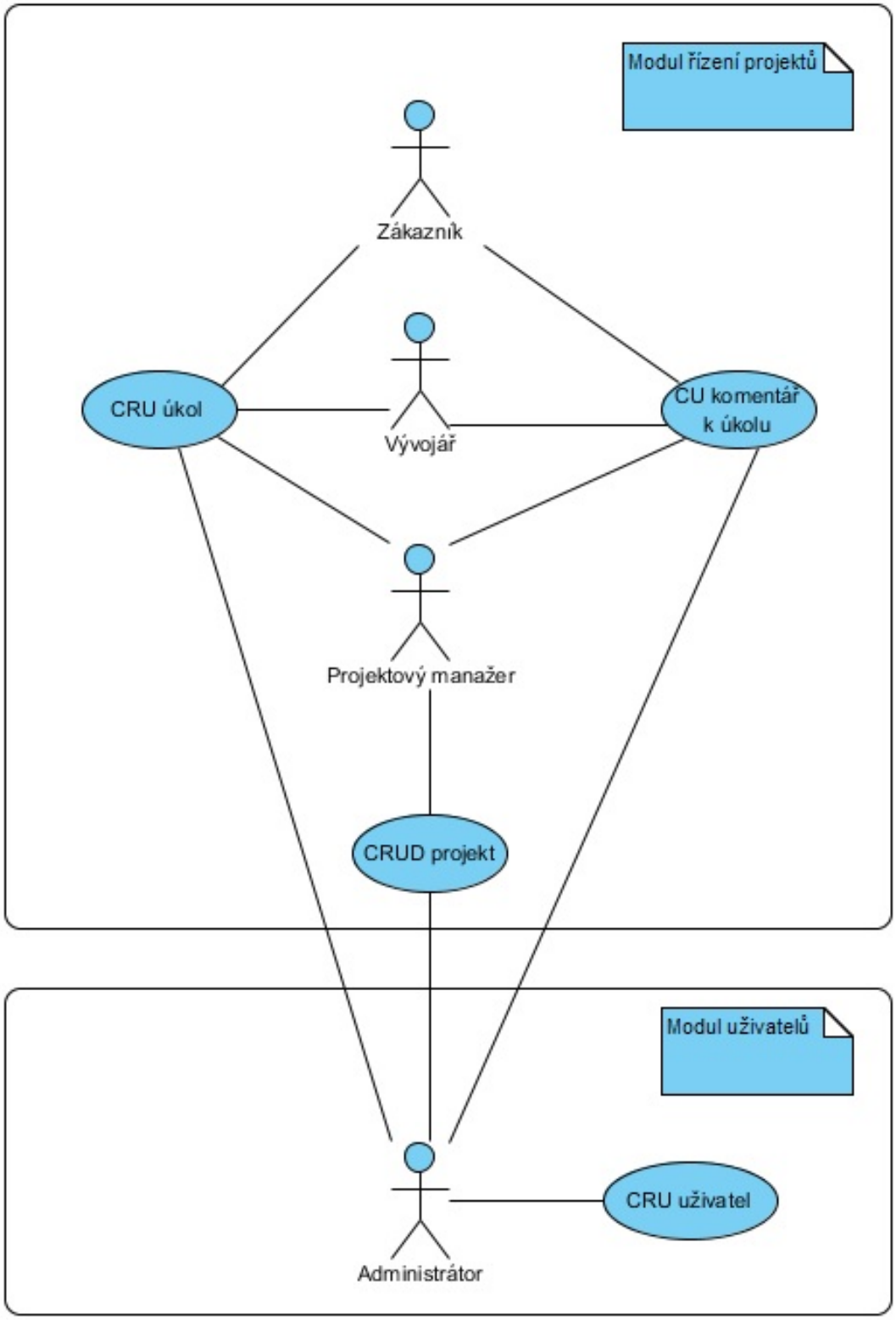
Symfony má vyřešenu i nezávislost jednotlivých modulů. Použitím symlinků

[3]objektově relační mapování - provádí konverzi mezi relační databází a objekty. Tím vzniká abstrakce, která vývojáře zbavuje nutnosti pracovat s dotazy konkrétní relační databáze. ORM usnadňuje provádění tzv. CRUD operací, což je čtení, zápis, úprava a mazání dat. Dále zajišťuje perzistentní uchování dat, tzn. že data uložená v operační paměti zůstanou nepožkozena i při pádu aplikace[2].

ve složce public (v které jsou uloženy soubory viditelné z internetu, tedy i index.php) mohou být uloženy CSS šablony, skripty a obrázky ve složce modulu. Není tedy třeba při nasazení nového modulu měnit obsah public složky. Jen pomocí konzole frameworku Symfony vygenerujeme symlinky.

Posledním, ale neméně důležitým důvodem volby Symfony je rozsáhlá dokumentace doplněna ukázkovými kódy, velikost aktivní komunity, aktivní vývoj a dlouhodobá podpora verze 2.3 až do roku 2016[20]. Tyto výhody zmiňuje na blogu i zakladatel frameworku Fabien Potencier[16].

6.2 Případy užití



6.3 Scénáře

Vytvořit projekt

Krátký popis

Use case umožňuje přidat nový projekt do CRM a informovat o vytvoření členy a sledující e-mailem.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel musí být přihlášen do systému s rolí Projektový manažer
- musí být vybrán modul Projektové řízení

Základní tok

1. uživatel vybere možnost Přidat projekt
2. systém vygeneruje formulář pro přidání projektu
3. uživatel zadá název, popis, členy a sledující projektu a odešle formulář
4. systém zvaliduje data zadaná uživatelem
5. systém uloží nový projekt do databáze
6. mailový dispatcher odešle email o vytvoření nového projektu členům a sledujícím
7. logovací služba zaznamená přidání projektu
8. systém zobrazí notifikaci o úspěšném přidání projektu

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí projekt uložit

4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Nový projekt je korektně uložen do databáze.

Zobrazit detail projektu

Krátký popis

Use case umožňuje zobrazit detail projektu.

Aktéři

- uživatel
- systém

Podmínky pro spuštění

- uživatel musí být přihlášen do systému
- musí být vybrán modul projektové řízení

Základní tok

1. uživatel vybere možnost Přehled projektů
2. systém zobrazí seznam projektů
3. uživatel vybere projekt, jehož detail si chce prohlédnout
4. systém zobrazí detail projektu

Alternativní tok 1

2.1 pokud je uživatel přihlášen jako Zákazník, zobrazí se mu seznam projektů, ve kterých je členem. Tok dále pokračuje 3. krokem základního toku

Podmínky pro dokončení

Detail vybraného projektu je zobrazen.

Upravit projekt

Krátký popis

Use case umožňuje upravit projekt a o změně informovat členy a sledující e-mailem.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel je přihlášen jako Projektový manažer
- je vybrán modul Projektové řízení

Základní tok

1. uživatel vybere volbu Přehled projektů
2. uživatel vybere volbu Upravit u vybraného projektu
3. systém zobrazí předvyplněný formulář pro editaci projektu
4. uživatel změní požadované údaje a odešle formulář
5. systém zvaliduje data zadaná uživatelem
6. systém uloží změny projektu do databáze
7. mailový dispatcher odešle email o změně členům a sledujícícm
8. logovací služba zaznamená úpravu projektu
9. systém zobrazí notifikaci o úspěšném přidání projektu

Alternativní tok 1

5.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí projekt uložit

5.2. uživatel opraví neplatný vstup a tok pokračuje na 5. kroku základního toku

Podmínky pro dokončení

Projekt je v databázi aktualizován.

Smazat projekt

Krátký popis

Use case umožňuje vymazat projekt a podřízené úkoly.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel musí být přihlášen jako Projektový manažer
- je vybrán modul Projektové řízení

Základní tok

1. uživatel vybere volbu Přehled projektů
2. uživatel vybere volbu Smazat u vybraného projektu
3. systém uživatele dialogovým oknem vyzve k potvrzení volby
4. uživatel potvrdí volbu
5. systém vymaže projekt a všechny jeho podřízené úkoly
6. mailový dispatcher odešle e-mail o smazání projektu a úkolů členům a sledujícím projektu
7. logovací služba zaznamená smazání projektu a úkolů
8. systém zobrazí notifikace o úspěšném smazání projektu

Alternativní tok 1

4.1. uživatel zruší svoji volbu

4.2. systém zruší operaci mazání projektu

Podmínky pro dokončení

Projekt a jeho podřízené úkoly jsou vymazány.

Vytvořit úkol

Krátký popis

Use case umožňuje přidat nový úkol.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel je přihlášen do systému
- je zobrazen detail projektu

Základní tok

1. uživatel vybere volbu Přidat úkol
2. systém zobrazí formulář pro přidání úkolu
3. uživatel zadá název, popis, typ, status, prioritu, přiřazeného uživatele, počáteční a konečný datum a odhadovaný čas úkolu a odešle formulář
4. systém zvaliduje data zadaná uživatelem
5. systém uloží nový úkol do databáze
6. mailový dispatcher odešle e-mail o vytvoření nového úkolu členům a sledujícím nadřazeného projektu
7. logovací služba zaznamená přidání úkolu
8. systém zobrazí notifikaci o úspěšném přidání úkolu

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí úkol uložit

4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Nový úkol je korektně uložen v databázi.

Zobrazit detail úkolu

Krátký popis

Use case umožňuje zobrazit detaily úkolu.

Aktéři

- uživatel
- systém

Podmínky pro spuštění

- uživatel je přihlášen do systému
- je zobrazen detail projektu

Základní tok

1. uživatel vybere úkol pro zobrazení detailu kliknutím na jeho název
2. systém zobrazí detail úkolu

Podmínky pro dokončení

Je zobrazen detail úkolu.

Upravit úkol

Krátký popis

Use case umožňuje upravit úkol.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel je přihlášen do systému
- je zobrazen detail úkolu

Základní tok

1. uživatel vybere volbu Upravit úkol
2. systém zobrazí předvypněný formulář pro úpravu úkolu
3. uživatel změní požadované údaje a odešle formulář
4. systém zvaliduje data zadaná uživatelem
5. systém uloží změny úkolu do databáze
6. mailový dispatcher odešle e-mail o úpravě úkolu členům a sledujícím nadřazeného projektu
7. logovací služba zaznamená úpravu úkolu
8. systém zobrazí notifikaci o úspěšné úpravě úkolu

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí úkol uložit

4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Změny úkolu jsou korektně uloženy v databázi.

Přidat komentář k úkolu

Krátký popis

Use case umožňuje přidat komentář k úkolu.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel je přihlášen do systému
- je zobrazen detail úkolu

Základní tok

1. systém zobrazí formulář pro přidání komentáře
2. uživatel zadá text komentáře a odešle formulář
3. systém zvaliduje data zadaná uživatelem
4. systém uloží komentář do databáze
5. mailový dispatcher odešle e-mail o přidání komentáře členům a sledujícím nadřazeného projektu
6. logovací služba zaznamená přidání komentáře
7. systém zobrazí notifikaci o úspěšném přidání komentáře

Alternativní tok 1

3.1. pokud uživatel zadal neplatný vstup, systém na skutečnost uživatele upozorní a nedovolí komentář uložit

3.2. uživatel opraví neplatný vstup a tok pokračuje na 3. kroku základního toku

Podmínky pro dokončení

Komentář korektně uložen v databázi.

Upravit komentář

Krátký popis

Use case umožňuje upravit komentář u úkolu.

Aktéři

- uživatel
- systém
- mailový dispatcher
- logovací služba

Podmínky pro spuštění

- uživatel je přihlášen do systému
- je zobrazen detail úkolu

Základní tok

1. uživatel vybere volbu Upravit u vybraného komentáře
2. systém zobrazí předvyplněný formulář pro úpravu komentáře
3. uživatel zadá změny komentáře
4. systém zvaliduje data zadaná uživatelem
5. systém uloží změny komentáře do databáze
6. mailový dispatcher odešle e-mail o změně komentáře členům a sledujícím nadřazeného projektu
7. logovací služba zaznamená úpravu komentáře
8. systém zobrazí notifikaci o úspěšné úpravě komentáře

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí úkol uložit

4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Změny komentáře jsou korektně uloženy v databázi.

Vytvořit uživatele

Krátký popis

Use case umožňuje vytvořit nového uživatele.

Aktéři

- uživatel
- systém

Podmínky pro spuštění

- uživatel je přihlášen do systému jako administrátor
- je vybrán modul Uživatelé

Základní tok

1. uživatel vybere možnost Přidat uživatele
2. systém zobrazí formulář pro přidání uživatele
3. uživatel zadá jméno, příjmení, e-mail, přihl. jméno, heslo a role uživatele a odešle formulář
4. systém zvaliduje data zadaná uživatelem
5. systém uloží uživatele do databáze
6. systém zobrazí notifikaci o úspěšném přidání uživatele

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí úkol uložit

4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Nový uživatel je korektně uložen v databázi.

Zobrazení detailu uživatele

Krátký popis

Use case umožňuje zobrazení detailních informací o uživateli.

Aktéři

- uživatel
- systém

Podmínky pro spuštění

- uživatel je přihlášen do systému jako administrátor
- je vybrán modul Uživatelé

Základní tok

1. uživatel vybere volbu Detail u vybraného uživatele
2. systém zobrazí detaily o uživateli

Podmínky pro dokončení

Zobrazení detailů o uživateli.

Upravit uživatele

Krátký popis

Use case umožňuje upravit údaje o uživateli.

Aktéři

- uživatel
- systém

Podmínky pro spuštění

- uživatel je přihlášen do systému jako administrátor
- je vybrán modul Uživatelé

Základní tok

1. uživatel vybere volbu Upravit u vybraného uživatele
2. systém zobrazí předvyplněný formulář pro úpravu úkolu
3. uživatel změní požadované údaje a odešle formulář
4. systém zvaliduje data zadaná uživatelem
5. systém uloží změny uživatele do databáze
6. systém zobrazí notifikaci o úspěšné změně uživatele

Alternativní tok 1

4.1. pokud uživatel zadal neplatný vstup nebo vstupy, systém na skutečnost uživatele upozorní a nedovolí úkol uložit

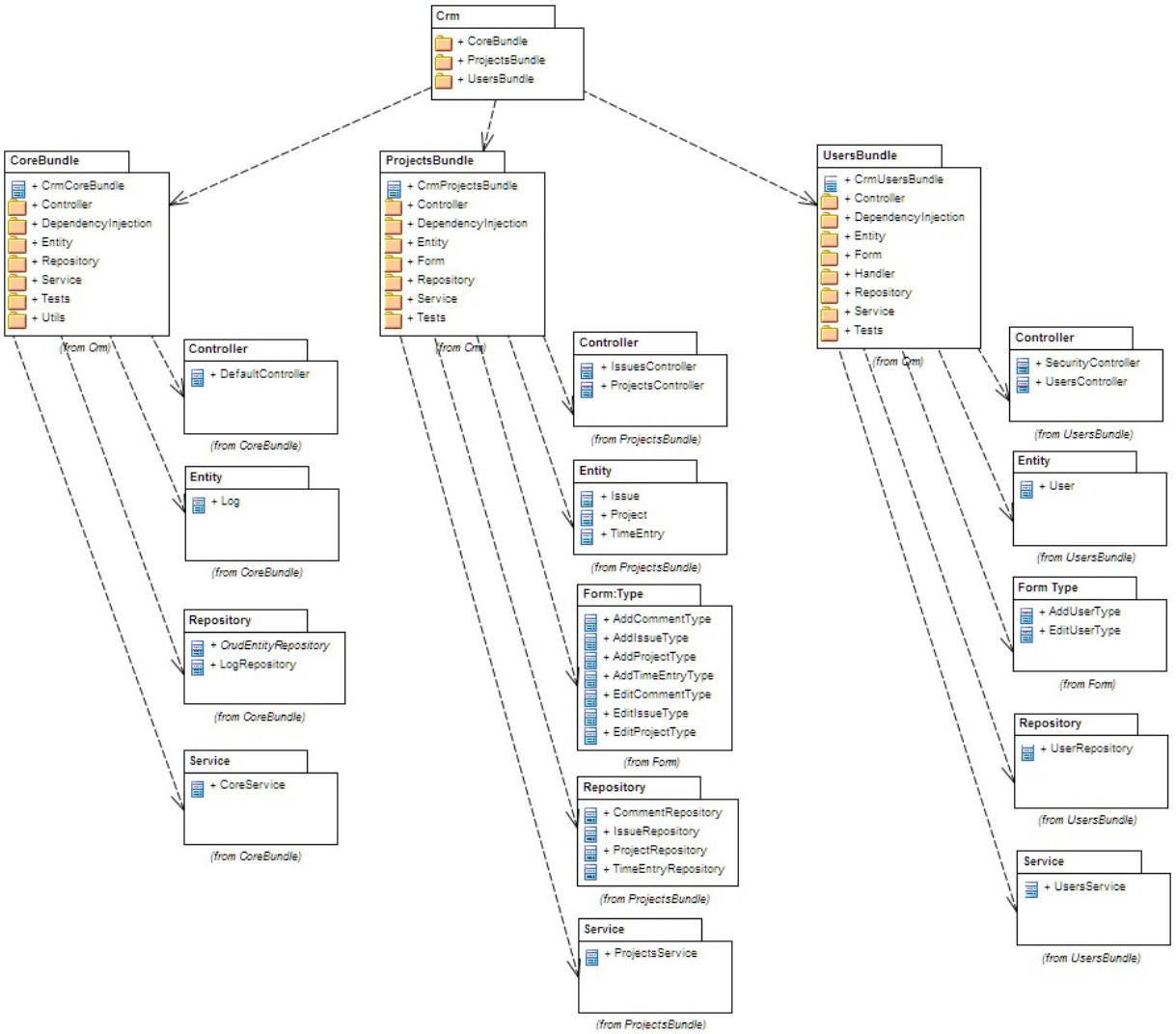
4.2. uživatel opraví neplatný vstup a tok pokračuje na 4. kroku základního toku

Podmínky pro dokončení

Změny uživatele jsou korektně uloženy v databázi.

6.4 Diagram tříd

S ohledem na doporučenou strukturu projektu Symfony a s důrazem na modulární řešení aplikace byla navržena následující struktura aplikace a souborů (obr. 6.4). Kompletní diagram tříd je v příloze práce na CD.



Obrázek 6.3: Zjedodučený diagram tříd

7. Vývoj

7.1 Vývojové prostředí

Jako vývojové prostředí byl zvolen Eclipse. Volba na něj padla z důvodu možnosti nainstalovat PHP developer tools (PDT), které zjednoduší a zrychlí vývoj PHP aplikací a to např. automatickým doplňováním kódu (bez PDT funguje jen pro Javu), zvýrazněním syntaxe apod. Nainstalováním Symfony 2 pluginu funguje doplňování kódu i pro konfigurační soubory, Twig šablony a anotace.

7.2 Adresářová struktura

Adresářová struktura je dodržována podobná, jako doporučují přímo tvůrci Symfony. Je dbáno na přehlednost a jasné oddělení jednotlivých částí aplikace. Z kořenové složky webu je struktura následující:

- app - adresář obsahuje jádro aplikace, nástrojovou konzoli, skripty pro kontrolu správné konfigurace serveru a adresáře pro logy, cache a konfiguraci aplikace.
- bin - umístění pro binární soubory Doctrine.
- src - zde se nachází zdrojové kódy celé aplikace. Ve složce je adresář s názvem aplikace. V něm jsou umístěny další adresáře, pro každý modul jeden. Struktura modelů je následná:
  - Controller - sem se ukládají všechny controllery.
  - DependencyInjection - obsahuje soubory načítající konfigurace z konfiguračních souborů.
  - Entity - zde jsou umístěny jednotlivé modely.
  - Repository - do této složky se umisťují repository třídy pro modely (zajišťují komunikaci s databází).
  - Resources - zde je ve složce config uložena konfigurace konrétního modulu, složka public obsahuje CSS šablony, Javascriptové skripty, obrázky apod. Ve složce translations jsou soubory pro jazykové mutace. Posledním adresářem je views, kde jsou pro jednotlivé controllery a jejich akce odpovídající view.
  - Service - obsahuje servisní třídy pro modul.
  - Tests - složka pro Unit testy.
  - Utils - adresář pro utility jako je např. komunikace s jinou aplikací, email dispatcher apod.
- vendor - složka obsahující všechny závislosti, tzn. i celý Symfony framework. Tento adresář se automaticky vytvoří při stažení závislostí prostřednictvím composeru.

- web - zde se nachází spouštěcí soubor app.php, skript pro konfiguraci přes webový prohlížeč a také se sem generuje složka bundles, ve které jsou pro jednotlivé moduly umístěny CSS šablony, Javascriptové skripty, obrázky apod.

7.3 Routování

Aby v aplikaci byly zachovány tzv. cool adresy, využívá se v Symfony routování. To bylo nastaveno v souboru

`app/config/routing.yml`

Pro výchozí routu bylo nastaveno, aby se zobrazilo view z modulu Projects pro dashboard (přehled pro přihlášeného uživatele):

```
crm_root:
     path:    /
     defaults:
         _controller: FrameworkBundle:Redirect:urlRedirect
         path: /projects/dashboard
         permanent: true
```

Pro modul Projects je pak nastavena obecná routa, která říká, že další routování modulu je nastaveno v controlleru formou anotací:

```
crm_projects:
     resource: "@CrmProjectsBundle/Controller"
     type: annotation
     prefix:   /projects
```

Tímto zápisem bylo zajištěno, že vše, co se týká modulu Projects bude v URI jasně označeno. Tímto způsobem je možno se vyhnout psaní direktiv pro modul rewrite - Symfony tuto práci udělá samo.

7.4 Anotace

Jak už bylo zmíněno, Symfony umožňuje konfiguraci psát do anotací. V první řadě této možnosti bylo využito při nastavení rout. V ProjectsControlleru byla například pro akci Dashboard napsána anotace

```
/**
* @Route("/dashboard")
*/
```

která zajistí, že pro zobrazení dashboardu je třeba zadat adresu:

`http://adresa_serveru/projects/dashboard`

Dále bylo anotací využito v modelech, které jsou zároveň definicí entit pro Doctrine. Například v modelu pro projekt je definována proměnná Name. Anotace pro tuto proměnnou je

```
/**
 * @ORM\Column(name="name", type="string", length=50)
 * @Assert\NotBlank
 * @Assert\Length(max=50)
 */
```

Tato anotace určuje, že entita v databázi bude mít název name, typ řetězec o délce 50 znaků. Pro kontrolu vstupních formulářů slouží direktiva @Assert, která kontroluje, zda je řetězec o délce větší než 0 a menší než 51. Těmito parametry se bude řídit doctrine při generování schématu databáze.

Díky možnosti nastavit entitě v databázi jiný název, než jaký má v PHP kódu, se elegantně řeší jiná konvence psaní názvů proměnných. Zatímco v PHP se využívá tzv. camelCase, kdy se první slovo napíše malým písmenem a další slova se zapisujou s velkým písmenem bez mezer, v SQL je zvykem mezi jednotlivá slova psaná malými písmeny vkládat podrtržítko.

7.5 Simulace MVC

Pro simulaci MVC bylo vybráno přidání projektu. V první řadě byl tedy vytvořen model Project, u jehož atributů byly nastaveny odpovídající anotace:

src/Crm/ProjectsBundle/Entity/Project.php

```
class Project {
/**
* @ORM\Column(name="id", type="integer")
* @ORM\Id
* @ORM\GeneratedValue(strategy="AUTO")
*/
private $id;

/**
* @ORM\Column(name="created_on", type="datetime")
*/
private $createdOn;

...

public function setCreatedOn($createdOn) {
$this->createdOn = $createdOn;
return $this;
}
```

```
public function getCreatedOn() {
return $this->createdOn;
}

...
}
```

Dále byla vytvořena akce addProject pro přidání projektu. Zde byla nastavena anotace pro routování a direktiva @Template(), která udává, že View bude načteno podle výchozího nastavení - tedy šablona psaná Twigem s názvem shodným s akcí a v Twigu bude použito html.

src/Crm/ProjectsBundle/Controller/ProjectsController.php

```
/**
 * @Route("/add")
 * @Template()
 */
public function addAction(Request $request) {
...
$form = $this->createForm(new AddProjectType($users), $project);

$form->handleRequest($request);

if ($form->isValid()){
...
return new RedirectResponse ($this
->generateUrl('crm_projects_projects_overview'));
}

return array(
'form' => $form->createView()
);
}
```

Při zavolání adresy

```
http://adresa_serveru/projects/add
```

se zavolá metoda addAction. Vzhledem k tomu, že v requestu ještě není formulář, zavolá se výchozí šablona a předá se jí vytvořený formulář. Ten se v šabloně použije následně:

src/Crm/ProjectsBundle/Resources/Views/Projects/add.html.twig

```
...

{{ form_start(form, {'attr': {'class': 'add-project-form'}}) }}
<div class='form-group'>
```

```
{{ form_label(form.name) }}
{{ form_errors(form.name) }}
{{ form_widget(form.name, {'attr': {'class': 'form-control'}}) }}
</div>

...

{{ form_label(form.submit) }}
{{ form_widget(form.submit, {'attr': {'class': 'btn btn-default'}}) }}
{{ form_end(form) }}

...
```

což vygeneruje HTML kód:

```
<form method="post" action="" class="add-project-form">
<div class='form-group'>
<label for="addProject_name" class="required">Název</label>
<input type="text" id="addProject_name" name="addProject[name]"
required="required" class="form-control" />
</div>

<button type="submit" id="addProject_submit" name="addProject[submit]"
class="btn btn-default">Vytvořit projekt</button>
</form>
```

Odesláním se znovu zavolá funkce addAction v ProjectsControlleru. Nyní už v requestu formulář je. Pokud jsou všechny zadané údaje správné, zpracuje se formulář, předá informace Modelu a zavolá akci overviewAction přes routu

```
crm_projects_projects_overview
```

která zavolá svoje View a to zobrazí přehled projektů.

Zde je vidět, že Controller má pevnou vazbu na View, na kterém zobrazuje informace uživateli, stejně tak i na Model, kterému předává vstupy zadané uživatelem. Naopak žádná vazba není mezi View a Modelem, což by bylo vážné pochybení při dodržování MVC architektury.

Také View nemá pevnou vazbu na konkrétní Controller, ale musí volat routu, na které se data zpracují.

Jak ukazuje obr. 4.2 v kapitole 4 (Úvod do problematiky), je možné, aby uživatel komunikoval přímo s Controllerem bez použití View. I tato možnost je v addAction zohledněna - pokud uživatel zašle requestem data formuláře, může se zcela obejít bez View. To otevírá možnost použití jakéhokoliv zařízení, dokonce i takového, které není schopno zobrazit HTML kód, nebo propojení s jinou aplikací, která by jen dodávala data do CRM systému.

8. Implementace

Pro nasazení aplikace napsané za pomoci frameworku Symfony je třeba splnit určité požadavky na server. Požadavky jsou následující [17]:

1. Povinné
    - PHP minimální verze 5.3.3
    - povolený JSON modul
    - povoleno ctype
    - php.ini soubor musí mít nastaven parametr date.timezone
2. Volitelné (nicméně pro bezproblémový běh aplikace doporučené)
    - modul PHP-XML
    - modul libxml minimální verze 2.6.21
    - povolený PHP tokenizer
    - povolena funkce mbstring
    - povoleno iconv
    - povoleno POSIX
    - modul Intl s ICU 4+
    - APC minimálně 3.0.17 (nebo jiný cachovací systém)
    - php.ini nastavení
        - short_open_tag = Off
        - magic_quotes_gpc = Off
        - register_globals = Off
        - session.auto_start = Off
    - PDO driver použité databáze

Pro fungování je dále předpokladem, že na serveru poběží webserverová aplikace (v našem případě Apache) a SMTP server.

Pokud tedy máme server připravený, pak již stačí do kořenové složky (např. /var/www u Apache) zkopírovat soubory s aplikací (na CD ve složce source) a nastavit práva pro čtení a zápis webovému serveru (u Apache je to skupina www-data) do složek

```
app/logs
app/cache
```

Po zkopírování souborů je třeba stáhnout aktuální závislosti kódu (tzn. i samotný framework Symfony). Toho dosáhneme zavoláním příkazu

```
php composer.phar install
```

v kořenovém adresáři webu. Zavoláním příkazu

```
php app/check.php
```

zjistíme, zda je server opravdu správně nakonfigurován, případně se nám označí nedostatky. Při volání tohoto příkazu je třeba si uvědomit, že PHP volané z příkazové řádky má většinou jiný konfigurační soubor PHP.ini, než webový server. Je proto dobré zobrazit soubor check.php i ve webovém prohlížeči.

Nyní nastavíme parametry aplikace - v souboru

```
app/config/parameters.yml
```

nastavíme informace o databázi, lokalizaci aplikace, tajný klíč sloužící pro generování csrf kódu formulářů, nastavení SMTP serveru, e-mailovou adresu, z které se posílají e-mailové notifikace, heslo superadministrátora a informace o administrátorovi.

```
parameters:
    database_driver: pdo_pgsql
    database_host: 127.0.0.1
    database_port: null
    database_name: crm
    database_user: uzivatel_db
    database_password: heslo_db

    locale: "cs_CZ"
    secret: eoAbJgDimXCNseiTYCOt

    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: smtp_uzivatel
    mailer_password: smtp_heslo
    addressFrom: email@adresa.cz

    superadminPassword: heslo_superadmina

    adminLogin: admin
    adminPassword: heslo_admina
    adminEmail: admin@email.cz
    adminName: Petr
    adminSurname: Plachý
```

Po nastavení parametrů můžeme vyčistit produkční cache paměť a vytvořit schéma v databázi. Zavoláme příkazy

```
php app/console cache:clear --env=prod --no-debug --no-warmup
php app/console doctrine:schema:create
```

.

Tímto máme aplikaci nasazenou, nyní již pomocí superadministrátorského účtu nastavíme administrátora systému a to tím, že ve webovém prohlížeči půjdeme na adresu služby. Přihlásíme se s uživatelským jménem superadmin a heslem nastaveným v souboru parameters.yml. Klikneme na tlačítko Set default values. Nyní se můžeme odhlásit a nově přihlásit se jménem a heslem administrátora zadaného taktéž v parameters.yml.

9. Závěr

Vzhledem k výsledkům dotazníkového šetření a skutečnosti, že si 60 % dotázaných firem nechává programovat svůj CRM systém na zakázku, je zřejmé, že pro vývojářské týmy má tato oblast velký potenciál.

Aplikace vytvořená v rámci této bakalářské práce poskytuje základ pro vývoj modulárního CRM systému a práce samotná dává vodítko, jak tento vývoj vést. S ohledem na posloupnost prací při využití metodiky ADDIE modelu se omezí situace, kdy vývojář zjistí, že nerozumí zadání zákazníka a tím se vývoj zrychlí a zefektivní. Pokud je navíc návrh schválen zákazníkem, eliminuje se nabobtnávání projektu a zabraňuje se sporům mezi realizátorem a zákazníkem.

Aplikace je připravena pro rozšiřování dalšími moduly a tím na přidání nových funkcí. Takovou funkcí může být například REST rozhraní, které umožní pracovat s funkcemi a daty na jakémkoliv zařízení, které bude schopné prostřednictvím tohoto rozhraní komunikovat.

Jak je vidět na struktuře aplikace, pokud se striktně dodržuje MVC architektura, jednotlivé části aplikace jsou od sebe odděleny a je možné je jednotlivě upravovat, aniž bychom přitom narušili funkčnost jiné části.

Cílem práce bylo tuto architekturu prakticky předvést, čemuž napomohlo použití frameworku Symfony. Díky tomu, že Symfony využívá spoustu moderních nástrojů, je vývoj jednoduchý a svižný. Jakmile je porozuměno jeho struktuře a funkcím, lze snadno vytvářet robustní a stabilní aplikace, jejichž údržba bude rychlá a časově nenáročná.

Literatura

[1] DOHNAL, Jan. *Řízení vztahů se zákazníky: procesy, pracovníci, technologie*. 1. vyd. Praha: Grada, 2002, 161 s. ISBN 80-247-0401-3.

[2] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003, xxiv, 533 p. ISBN 03-211-2742-0.

[3] MAREŠKA, Patrik. *Analýza trhu CRM systémů*. Praha, 2013. Dostupné z: `http://isis.vse.cz/zp/52806`. Bakalářská práce. Vysoká škola ekonomická v Praze. Vedoucí práce Ing. Jana Fortinová.

[4] ULLMAN, Larry. *PHP a MySQL: názorný průvodce tvorbou dynamických WWW stránek*. Vyd. 1. Brno: Computer Press, 2004, 534 s. ISBN 80-251-0063-4.

Elektronické zdroje

[5] ASP.NET: Get Started with ASP.NET & ASP.NET MVC. MICROSOFT CORPORATION. *ASP.NET* [online]. 2013 [cit. 2013-05-04]. Dostupné z: `http://www.asp.net/get-started`

[6] BECKINGHAM, Colin. Take Advantage of Database Field Arrays: Examples Using PostgreSQL. In: OPENLOGIC, INC. *OpenLogic: Helping Enterprises Use Open Source Software* [online]. 2012, Jul 30, 2012 [cit. 2013-10-20]. Dostupné z: `http://www.openlogic.com/wazi/bid/196878/Take-Advantage-of-Database-Field-Arrays-Examples-Using-PostgreSQL`

[7] BOREK, Bernard. MVC a další prezentační vzory: Úvod do architektury MVC. In: *Zdroják.cz* [online]. 2009, 7.5.2009 [cit. 2013-04-20]. Dostupné z: `http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/`

[8] BOREK, Bernard. MVC a další prezentační vzory: Prezentační vzory z rodiny MVC. In: *Zdroják.cz* [online]. 2009, 11. 5. 2009 [cit. 2014-03-16]. Dostupné z: `http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/`

[9] GARG, Osho. Top 5 PHP frameworks for 2013. In: *Crispy Coding* [online]. 2013, May 11, 2013 [cit. 2013-11-04]. Dostupné z: `http://crispycoding.com/top-5-php-frameworks-for-2013`

[10] GOOGLE. Interest over time. In: *Google Trends* [online]. 2013 [cit. 2013-11-04]. Dostupné z: `http://www.google.com/trends/explore?hl=en-US#q=Zend,+Yii,+CodeIgniter,+Cake+PHP,+Symfony&amp;cmpt=q`

[11] HAACK, Phil. Everything You Wanted To Know About MVC and MVP But Were Afraid To Ask. In: *You've Been Haacked: ...and you like it* [online]. 2008, June 16, 2008 [cit. 2014-

03-20]. Dostupné z: http://haacked.com/archive/2008/06/16/everything-you-wanted-to-know-about-mvc-and-mvp-but.aspx

[12] INDIESPHP. Comparison of best PHP frameworks. In: *IndiesServices* [online]. 2013, October 4, 2013 [cit. 2013-11-04]. Dostupné z: http://indiesphp.com/comparison-of-best-php-frameworks

[13] JENNIPHER. The ADDIE Framework is Too Generic. In: *Blogspot* [online]. 2011, 2. 12. 2011 [cit. 2013-05-14]. Dostupné z: http://exploreinstructionaldesign.blogspot.cz/2011/12/addie-framework-is-too-generic.html

[14] NOVÁK, Milan. Fáze výroby stránek. In: *VOXCAFE.CZ* [online]. 2013, 1. 1. 2013 [cit. 2013-09-19]. Dostupné z: http://www.voxcafe.cz/clanky/vyvoj-aplikaci/faze-vyroby-internetovych-stranek.html

[15] PHPFRAMEWORKS. PHP Frameworks. In: *PHP Frameworks* [online]. 2007 [cit. 2013-11-04]. Dostupné z: http://www.phpframeworks.com

[16] POTENCIER, Fabien. Delicious Preview built with symfony. In: *Symfony: Blog* [online]. 2007, October 02, 2007 [cit. 2013-09-01]. Dostupné z: http://symfony.com/blog/delicious-preview-built-with-symfony

[17] POTENCIER, Fabien. Requirements for running Symfony2. In: *Symfony* [online]. 2013 [cit. 2014-03-16]. Dostupné z: http://symfony.com/doc/current/reference/requirements.html

[18] R, Vašek, František KUČERA, David KOLIBÁČ a Petr KOVÁCS. Výkladový slovník. In: *ABC Linuxu* [online]. 20.7.2004, 16.12.2010 [cit. 2013-05-04]. Dostupné z: http://www.abclinuxu.cz/slovnik/open-source

[19] SDRACO. Observer. In: *Devbook.cz: Programátorská sociální síť* [online]. ©2013 [cit. 2013-05-05]. Dostupné z: http://www.devbook.cz/observer-pozorovatel-navrhovy-vzor

[20] Symfony Roadmap. SENSIOLABS. *Symfony* [online]. 2013 [cit. 2013-09-01]. Dostupné z: http://symfony.com/roadmap

[21] W3TECHS. Usage of server-side programming languages for websites. In: Q-SUCCESS. *W3Techs: extensive and reliable web technology surveys* [online]. 2009-2013, 7.9.2013 [cit. 2013-09-07]. Dostupné z: http://w3techs.com/technologies/overview/programming_language/all

Seznam tabulek

Seznam obrázků