Technická Univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Ústav nových technologií a aplikované informatiky

Bakalářská práce

Tvorba modulárního systému internetových prezentací

Tomáš Macik

Vedoucí práce: Ing. Petr Kretschmer

Studijní program: Elektronické informační a řídící systémy

15. června 2011

Poděkování

Poděkovat bych chtěl svému vedoucímu práce, Ing. Petrovi Kretschmerovi za příležitost vytvořit tento projekt a za jeho rady ohledně mé práce. Dále bych chtěl poděkovat své rodině za podporu ve studiu a ve tvorbě projektu. Můj dík paří taky slečně Lucii Kociánové za gramatickou korekci práce.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat náhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Praze dne 19. 5. 2011 .....

Abstract

The task of this thesis is to develope modular web-based content management system. This thesis describes the character of modular web applications and the development of the optimal varsion of modular system. This thesis also compares open-source licences which can be used to publish the developed system.

Abstrakt

Cílem táto práce je vytvořit modulární systém na správu obsahu jako webovou aplikaci. Tato práce popisuje modulární webové aplikace a vývoj optimální varianty modulárního systému. Tato práce taky porovnává open-source licence, pod kterými by bylo možné vytvořený systém publikovat.

Obsah

Kapitola 1

Úvod

Cílem táto práce je seznámení se se stavbou webových modulárních aplikací. Ze získaných znalostí pak vybrat správní model modulární aplikace a tento model implementovat. Na závěr bude práce zhodnocovat existující open-source licence a jejich aplikaci na výsledný projekt.

Kapitola 2

Webové aplikace a jejich stavba

Každý z nás pozná aplikace a denně se s nimi setkává. Počítače zasáhly do našich životů ve velké míře. V poslední době se aplikace stávají dostupnější díky internetu. Tato dostupnost se zvětšila tím, že se uživatelům již nenabízejí kompletní aplikace ale jenom možnost jejich využití. To má na jedné straně výhodu v rychlé aktualizaci programového a datového vybavení aplikace. Na druhou stranu je zde větší bezpečnostní riziko, protože data se již neskladují lokálně ale vzdáleně.

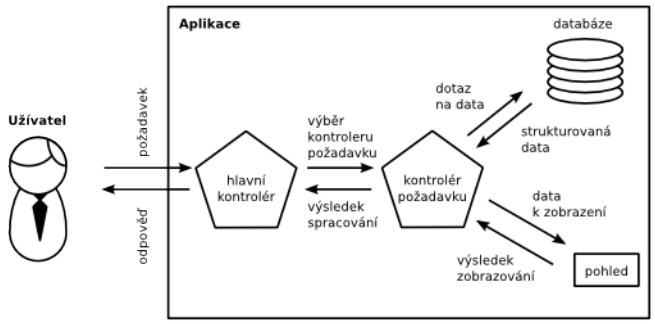
Webové aplikace mají jinou strukturu než aplikace, které jsou tzv. desktopové. Programátoři desktopových aplikací jsou zvyklí na mírně jiný životní cyklus aplikace. Aplikace začne fungovat od svého spuštění a uživatel ji může obsluhovat. Svou paměť si může uchovat a měnit během celou dobu své existence, tj. dokud nebude vypnuta. Před vypnutím si může některé důležité části uložit i do dlouhodobé paměti počítače. Webové aplikace mají mnohem kratší dobu životního cyklu. Aplikace existuje stejně od svého zapnutí do vypnutí, jenom s tím rozdílem, že zapnutí aplikace je vyvoláno požadavkem od uživatele, který odeslal na server. Její vypnutí nastává po vygenerování a odeslání odpovědi zpátky uživateli.

2.1 Životní cyklus webové aplikace

Webové aplikace jsou založeny na bezstavovém protokolu **HTTP**[1] . Pro potřeby většiny aplikací je ale nutné vědět, v jakém stavu se nachází z hlediska komunikace s uživatelem. Skoro všechny webové aplikace jsou koncipovány pro práci více uživatelů, a proto je další

[1] hypertext transfer protocol - protokol pro přenos dat ve formátu HTML

důležitou informací identifikace uživatele.



Obrázek 2.1: Životní cyklus aplikace

1. **Zadání požadavku** Uživatel pomocí prohlížeče odešle požadavek s parametry na server. Zadání požadavku může být například napsání adresy stránky nebo aplikace, zmáčknutí tlačítka nebo přechod kurzorem nad určitou oblastí stránky. Požadavky se také můžou odesílat nepřímo a to scriptem na pozadí stránky, takže si uživatel ani nemusí všimnout, že probíhá komunikace mezi prohlížečem a serverem.

2. **Předání** Dle charakteru požadavku určí aplikace svojí část, která bude následně požadavek a jeho parametry zpracovávat a generovat výstup pro uživatele. Některé aplikace v tomto kroku přibalují k parametrům požadavku také data uložená pro aktuální sezení. Tyto data se ukládají do **session** (sezení), kterou obhospodařuje http server. Sezení je jednoznačně identifikováno pro každé připojení na server (reprezentuje uživatele, prakticky však jenom jeden prohlížeč) a pro každou webovou aplikaci, která jej využívá.

3. **Zpracování** Část aplikace, která byla vybrána při předání, je inicializována a zahájí zpracování požadavku. V této části může probíhat komunikace s databázovým serverem a provádění operací nad daty. Databázový server je využíván jako permanentní úložiště dat, které je společné pro všechny instance aplikace a taky pro všechny uživatele. To však nutně nemusí znamenat, že každý uživatel aplikace má přístup ke všem datům aplikace.

4. **Vytvoření a odeslání výstupu** Tato část cyklu může a také obvykle bývá vytvářená paralelně s předchozí částí. Výstupem je pro člověka čitelná reprezentace dat aplikace. Podstatnou a někdy i jedinou reprezentaci výstupu tvoří **HTML**[2] formát, který je následně prohlížečem zobrazen do čitelné podoby. Některé pokročilejší aplikace tvoří taky různé jiné formáty výstupu jako **json**[3], který se používá v technologii **ajax**[4], nebo formát **xml**[5] pro komunikaci s jinými systémy.

2.2 Konstrukce aplikace v PHP

Pro vytvoření webové aplikace je nutné vytvořit její základ. Pro desktopové aplikace je tento základ poskytován operačním systémem přímo, nebo prostřednictvím virtuálního stroje. Webové aplikace nejsou tak náročné na rozsah tohoto základu. Nepotřebují vytvářet procesy nebo vlákna a také obecně nejsou závislé na jádře operačního systému. Tuto závislost odstraňuje aplikační server nebo interpretr. Tento základ se obvykle nazývá **framework**. Zahrnuje v sobě sadu konstant, funkcí a objektů, které zjednodušují a unifikují řešení určitých problémů. Čím lepší je daný framework, tím víc se může programátor soustředit na logickou a aplikační podstatu problému a může více ponechat technologickou stránku na již vytvořených komponentech frameworku. To ale vyžaduje od programátora, aby pro řešení určitých problémů používal postupy předepsané stavbou frameworku.

Pro jazyk PHP existuje velké množství frameworků. Odlišují se svým určením, technologickým provedením, přístupností, kvalitou provedení a dokumentace a také podporou pro vývojáře. Některé frameworky jsou určeny jenom pro specifické účely. Jsou ve větší míře vytvořené přímo pro potřeby aplikace, ve které jsou použity. Přístup k těmto frameworkům mají jen vývojáři, kteří pracují na dané aplikaci. Další skupinou jsou všeobecnější frameworky. Mohou být určeny pro malý okruh aplikací (internetové obchody, redakční systémy, jednoduché internetové prezentace, atd.). K všeobecným frameworkům patří i univerzální frameworky. Ty zabezpečují jenom základní funkce aplikace a zbytek je

[2] hypertext markup language - značkovací jazyk používaný pro tvorbu webových stránek

[3] javascript object notation - lidsky čitelná reprezentace objektů v jazyku javascript

[4] asynchronous javasctipt and xml - technologie asynchronní komunikace klient server pomocí javascriptu a xml

[5] extensible markup language - značkovací jazyk pro textovou reprezentaci strukturovaných dat

ponechán na programátorech. Všeobecně platí, že zobecnění frameworku vede ke změně portfolia funkcí, které řeší za programátora.

2.2.1 Výběr frameworku pro PHP

Pro potřeby této práce jsem se zaměřil na všeobecné frameworky. Tuto skupinu jsem ještě zúžil na frameworky volně dostupné a poskytované pod jednou z variant open-source licence. Dále jsem se zaměřil na prozkoumání frameworků, které jsou známé a mají dobrou podporu pro vývoj. Mimo už existujících frameworků je zde dále varianta vlastního vývoje. Výhodou této varianty je, že programátor se nemusí učit strukturu frameworku, ale může ji naprogramovat přesně podle svých potřeb.

Zend framework je jeden z neznámějších frameworků vůbec. Pro tvorbu webových aplikací je hojně využíván. Mimo jiné poskytuje vlastní vývojové prostředí, které nabízí širší využití vlastností frameworku při vývoji. Tento framework je na druhou stranu hodně rozsáhlý a pokouší se pokrýt hlavní potřeby pro velké spektrum aplikací.

Nette framework je dílem Davida Grundla[6] . Na tomto frameworku je postaveno mnoho českých portálů. K frameworku se dají připojit další komponenty, které jsou tvořeny komunitou programátorů. Pro připojení k databázovému serveru je nejčastěji využíván framework **dibi**, který usnadňuje práci s jednoduchými dotazy.

Některé další frameworky také poskytují kompilaci PHP kódu do binárních, spustitelných souborů, a tím zvyšují rychlost odezvy na požadavky uživatele. Tuto technologii využívá například hodně známý komunitní portál **facebook** nebo framework **DooPHP**.

Aby bylo možné najít ten správný framework, je nutné stanovit si, co bude výsledný projekt potřebovat. Je potřeba vzít také v úvahu příští vývoj projektu nebo jeho potenciální derivace[7] . Nejlepší volbou je samozřejmé již známy framework, ve kterém má programátor praktické zkušenosti, a který je pro danou aplikaci vhodný.

2.2.2 Potřeby a framework pro projekt IVY

IVY je systém na správu obsahu webu jinak též označovaný jako **CMS** neboli **Content Management System** . Tento systém ulehčuje uživateli práci s webovým obsahem.

[6] David Grundl - programátor a publicista, známy tvorbou frameworku Nette a dalšími přínosy do obce programátorů webových aplikací

[7] Derivace v oblasti software nemají matematický význam. Jedná se o aplikace, které jsou založeny na mateřské aplikaci a upravují její funkčnost a/nebo zaměření.

Umožňuje mu tvořit, měnit a aktualizovat webové prezentace, blogy a jiné informační druhy webových aplikací.

Potřeby webové aplikace lze rozdělit do tří skupin:

1. uživatelské,
2. administrační,
3. vývojové.

Uživatelské potřeby aplikace určíme s ohledem na cílovou skupinu uživatelů, pro které je aplikace vyvíjena. Uživatelé nejsou zpravidla programátorsky zdatní, a proto je nutné, aby aplikace nabízela takové funkčnosti, které umožní vykonat potřebné úkoly s minimálním rozšířením už nabytých znalostí. Intuitivnost je v tomto případě velmi důležitá. I když je všeobecná intuitivnost uživatelského rozhraní značně diskutabilním tématem ve vývojářské komunitě, pro jednotlivé cílové skupiny ji lze definovat snadněji. Cílové skupiny se většinou rozdělují podle toho, jaké aplikace už používaly a jak velké zkušenosti s těmito aplikacemi mají.

Administrační potřeby aplikace specifikujeme jako uživatelské podle té skupiny uživatelů, která bude mít za úkol aplikaci udržovat. Udržování aplikace v sobě může zahrnovat péči o data a jejich filtrování, správu o hardware, na kterém aplikace běží. Potřeby této skupiny uživatelů jsou z části potlačovány samotnými potřebami administrace aplikace.

Vývojové potřeby aplikace se orientují na jednoduchost rozšiřování aplikace. Aplikace se staví podle očekávaných i předvídatelných změn a úprav tak, aby tyto změny upravovaly co nejmenší část. Někdy je ovšem nutné upravit podstatnou část aplikace proto, aby další změny splňovaly zmíněná kritéria. Tyto změny jsou však méně časté, protože příliš časté velké změny svědčí o špatném návrhu aplikace.

Cílovou skupinou projektu IVY jsou uživatelé, kteří mají zkušenosti s publikováním článků na internetu. Uživatelské prostředí by mělo poskytovat CRUD[8] operace nad články. Syntaxe psaní článků by měla být co nejjednodušší a intuitivní. Aplikace nebude od uživatelů očekávat znalost HTML.

Administrace aplikace by měla umožňovat podobný komfort pro práci s daty jako pro práci s články. Je totiž pravděpodobné, že uživatel (autor článků) bude zároveň

[8] Create Read Update Delete - operace nad daty: vytvoření, čtení, úprava, smazání

administrátorem aplikace. Z tohoto pohledu je vhodné, prostředí pro práci s články a jinými daty sjednotit co nejvíc.

Z hlediska budoucího vývoje je nutné aplikaci koncipovat modulárně tak, aby každá část aplikace byla samostatně rozšiřitelná a zároveň toto rozšíření nenutilo vývojáře upravovat jiné části aplikace. Moduly by měly být vytvořeny tak, aby využívaly maximální potenciál ostatních modulů a samy přidávaly jenom funkčnost, za kterou zodpovídají. To znamená, že kompetence jednotlivých částí se nebudou překrývat.

Po definování potřeb a vlastností projektu můžeme pro něj najít vhodný framework, který je nejlépe pokryje. V úvahu připadají již vytvořené frameworky a taktéž možnost vytvoření vlastního frameworku. Zmíněné frameworky, které mají výhodu už existující komunity, šetří práci jenom ohledně jeho vývoje. Nenašel jsem dosud žádný framework, který by poskytoval již základní koncept aplikace. Také požadavek na modulárnost není zcela zakomponován ani do jednoho z nich. Do aplikace by musely být tyto požadavky naprogramovány zvlášť. Základy, které bych mohl z těchto frameworků využít nijak nekompenzují skutečnost zbytečného nastavování dalších vlastností a tak jsem se rozhodl vytvořit si framework vlastní, který bude mít možnost modulárního programování zakomponovanou přímo v sobě.

Kapitola 3

Tvorba frameworku SFW

3.1 Funkcionalita a modulárnost

Pro jednoduchost jsem všeobecné kompetence jednotlivých částí aplikace rozdělil podle životního cyklu webové aplikace. Z těchto základních celků vznikl minimalistický framework **SFW**[1] . Dalším úkolem tohoto frameworku je postarat se o modulární rozdělení aplikace.

Objekty jsou rozděleny od čtyř skupin a ke každé skupině je vytvořeno příslušné rozhraní. Jsou to tyto skupiny:

1. **Moduly** Kontrolní objekty, které se starají o běh aplikace
2. **Entity** Datové objekty na udržování strukturovaných dat
3. **Pohledy** Zapouzdřovací objekty, které se starají o výstup aplikace
4. **Připojení** Komunikační objekty pro propojení aplikace se službami jiných aplikací, nejčastěji pro spojení s databázovým serverem

Zvláštní skupinu tvoří popisy pluginů[2]. Tyto objekty nemají na funkčnost aplikace vliv a při jejím standardním běhu se nevyužívají. Jde o objekty, které v sobě udržují informaci o daném pluginu.

[1] simple framework - jednoduchý framework

[2] odnímatelná část aplikace, pro SFW zabezpečuje modularitu aplikace

3.2 Základní prvky frameworku

3.2.1 Autoloading

Jazyk PHP nemá podporu automatického slučování souborů, resp. vyhledávání tříd v jiných souborech než aktuálním. Existují příkazy, které slučují soubory a tím virtuálně vytvářejí jeden obří PHP script při zpracování. Je ale velmi obtížné, z hlediska programátorské práce, přidávat tyto příkazy a udržovat je aktuální. Při přesunu jednoho souboru do jiného adresáře je totiž nutné změnit všechny odkazy na něj v dalších souborech, které jej využívají. To má za následek buď nutnost fixní adresářové struktury, nebo obřího adresáře, kde jsou uloženy PHP soubory. To ale není flexibilní metoda.

Protože framework SFW by měl být co nejuniverzálnější, je v něm použita technologie autoloadingu. V PHP lze nadefinovat funkci, která je volána při instancování nebo dědění tříd. Jmenuje se **autoloading**. Co bude vykonávat a jak se bude chovat, je na samotném programátorovi. Jejím účelem je zjistit blíže nespecifikovaným způsobem, kde se třída nachází a pokud je pro interpretr scriptu neznámá, připojit soubor s její definicí.

V SFW je tato funkce použita velmi liberálně tj. prohledává všechny dostupné adresáře a hledá požadovanou třídu. Jedinou podmínkou je shoda jmen souboru a třídy včetně velikosti písmen. Pokud je tedy hledaná třída *ArticleView*, funkce bude prohledávat adresáře a hledat soubor *ArticleView.php*.

Další technologií použitou v SFW je **cache** pro třídy. Cache je reprezentována *ini* souborem[3], kde jsou vypsány třídy, které byly aplikací již někdy použity a cesta k souborům s jejich definicí. Když je v aplikaci požadována dosud neznámá třída, nejdříve je prohledána cache a až pak je prohledáván adresářový strom. Když je definice nalezena prohledáváním adresářů, je její poloha zaznamenaná do cache. Jestli ale zaznamenaný soubor z cache neexistuje, je celá cache vymazána jako chybná a bude postupným hledáním sestavena znova. Hledání probíhá jenom pro požadované definice tříd. Protože prohledávání celé adresářové struktury je systémově náročná činnost, je použití cache výrazně zrychlující prvek.

[3] ini soubor - soubor sloužící pro konfiguraci resp. pro inicializaci

3.2.2 Moduly

Rozhraní *IModule* specifikuje jedinou metodu – *run()*. Toto rozhraní je implementováno v abstraktní třídě *Module*. Tuto třídu dědí třída *ActionModule*. V *ActionModule* je implementována také metoda *ADefault()*. Tento modul má veřejný parametr *action*. Tento parametr obsahuje řetězec, který reprezentuje akci požadovanou od modulu. Akce modulu je definována metodou objektu s prefixem *A*. Pokud tedy po modulu budu chtít, aby provedl akci *List*, provede metodu *AList()*. Pokud ale tato metoda nebude definována, provede se standardní akce a to metoda *ADefault()*. Ta se také provede v případě, že bude na modulu zavolaná metoda *run()* a nebude nastavena žádná akce, kterou by měl modul vykonat. Tento způsob usnadňuje volání modulů s jedinou, nebo jednou nejpoužívanější metodou, což je většina modulů.

V modulech je možné vypsat požadovaný výstup. Jsou první a poslední vrstvou mezi uživatelem a aplikací. Není vhodné, aby výstup vypisoval jiný objekt než modul. Také ostatní funkce, které vyžaduje životní cyklus aplikace, jsou delegovány na jiné objekty prostřednictvím modulu.

3.2.3 Entity

Rozhraní *IEntity* popisuje rozsáhlejší kontrolu nad daty. Popis jednotlivých metod by byl příliš zdlouhavý, a proto jsou k nalezení v příloze jako popis frameworku SFW. Tyto metody implementuje třída *Entity*. K uložení jednotlivých dat se používá parametr *data*, který reprezentován asociativním/indexovaným polem. Jazyk PHP nemá silnou typovou konvenci, a proto umožňuje do pole vkládat hodnoty různých typů od primitivních typů, až po objekty.

Kontrola nad daty spočívá v jednotě návratové hodnoty a to samotné hodnoty pokud existuje, nebo hodnoty *false*, pokud hledaný klíč neexistuje. Přidávání hodnot do entity je také usnadněno. Pro vložení hodnoty je nutné definovat hodnotu samotnou. Klíč je pouze volitelný. Pokud není zadán, je použit číselný klíč o jedna větší než nejvyšší použitý.

3.2.4 Pohledy

Rozhraní *IView* specifikuje metodu *Render()*. Toto rozhraní je implementováno v abstraktní třídě *View*. Jenom zdědění této třídy by vrátilo jako výstup prázdny řetězec. Tuto

třídu však dědí třída *TemplateView*. Ta implementuje zobrazování výstupu na základě šablony a tagů.

Šablona je libovolným textovým souborem. Při generování výstupu v ní dojde k nahrazení tagů jejich hodnotami. Tagy mají specifický způsob zápisu. Jméno a atributy tagu jsou ohraničeny dvojitými složenými závorkami. Vnitřní řetězec mezi závorkami je rozdělen podle teček a vložen do pole, přičemž první prvek výsledného pole udává jméno tagu a zbytek pole je volitelně použit jako argumenty. Hodnota tagu je získána primárně ze stejně pojmenované metody objektu a sekundárně z dat pohledu. Data pohledu jsou reprezentována entitou. Při získání hodnoty z metody objektu jsou použity i argumenty tagu, při hodnotě z entity nikoliv.

Pohled je koncipován pro použití na generování HTML výstupu. Tím však jeho použití nekončí. Jeho schopnost generovat libovolný řetězec s předem definovanými hodnotami je také využita například u generování SQL dotazů.

3.2.5 Připojení

Rozhraní *IConnection* specifikuje tři základní metody pro práci s připojením a to *Connect()* pro vytvoření připojení, *Query([query])* pro zaslání dotazu přes připojení a *IsConnected()* pro zjištění existence spojení se vzdálenou aplikací. Tyto metody implementuje abstraktní třída *Connection()*. Ve frameworku SFW neexistuje žádná reálná implementace, resp. zdědění *Connection*. To je ponecháno na nadstavbě v podobě pluginů, které budou vytvořeny specificky pro daný účel.

3.3 Rozšíření frameworku

3.3.1 Seznamy

Při práci s frameworkem SFW jsem zjistil, že funkčnosti, které by programátor ocenil, chybí. Bylo ale v rozporu s filosofií frameworku, aby tyto funkčnosti byly zakomponovány přímo do jádra, a proto jsem vytvořil soubor pro užitečné nástroje, který je při použití SFW volitelný.

V této fázi jedinou funkčností, která mi ve frameworku chyběla, byl seznam. Seznam je popsán rozhraním *ISet*. Funkčnost tohoto rozhraní je implementována ve třídě *Set*. Seznam je rovněž rozšířením entity. Všechny akce spojené s entitou se odehrávají nad aktuál-

ním prvkem seznamu, jestli existuje, nebo vracejí *false*, jestli je seznam prázdný. Kromě funkcí zděděných od entity, seznam umožňuje standardní akce jako pohyb o jeden prvek vpřed nebo vzad, přidání nebo mazání prvku. Také obsahuje metodu *Each(callback)*, která provádí určitou akci pro každý prvek v seznamu. Akce je definována jako zpětné volání, v PHP obyčejně definované jako dvouprvkové pole s třídou a její metodou. Metoda pro zpětné volání musí brát jako vstupní parametr entitu. Výsledkem tohoto volání na všechny prvky je pole výstupů se stejnou délkou jako sám seznam.

3.3.2 Pluginy

Plugin je obvykle distribuován společně s moduly, pohledy, připojeními a entitami, které reprezentují jeho funkčnost. Pluginy jsou rovněž definované ve specifickém souboru. To znamená, že modulárnost aplikace založené na SFW je také volitelná. Objekt pluginu se použije jen při jeho instalaci. Popisuje plugin verzí a závislostí na jiných pluginech. Obsahuje metodu, kde je možno definovat akce, které se při jeho instalaci provedou. V pluginu je možné definovat i akce, které se mají vykonat při spuštění aplikace. Ty jsou definovány jménem modulu, jménem jeho akce a krokem, ve kterém se mají spustit. Spuštění aplikace je pak inicializováno jedinou metodou *ElectTheDead()*. Ty projde v jednotlivých krocích a spustí definované moduly a jejich akce. Kroky spouštění jsou v tomto případě důležité, protože pluginy mohou být na sobě závislé a jejich spuštění může být podmíněno startovací akcí jiného modulu z jiného pluginu.

Kapitola 4

Aplikační framework BaWIS

Aplikační framework **BaWIS**[1] je postaven na frameworku SFW. Využívá všechny jeho prvky jako jádro, nástroje a pluginy. Samotný framework je jenom soubor pluginů, které tvoří základ aplikace. Na programátorovi pak zůstává jen rozhodnutí, které z pluginů bude využívat. BaWIS rovněž poskytuje základní uživatelské rozhraní. Účelem tohoto rozhraní je zjednodušit správu dat aplikace, což je velmi vhodné pro administraci aplikace.

4.1 Core Plugin

Po nastartování aplikace je vždy jako první spuštěn plugin jádra. Ten se stará o základní směřování dat po aplikaci a její strukturalizaci. Z požadavku rozezná, jaký modul má být určen ke zpracování požadavku. Také se stará o předdefinované nastavení aplikace. Tady lze nastavit jméno aplikace, její znakovou sadu i jiné parametry potřebné pro ostatní pluginy.

Jako standardní výstup produkuje tento plugin jednoduché HTML. Výstup je generován pomocí pohledu. Tento výstup je možné změnit nainstalováním jiného pluginu, který bude výstup přetěžovat. Tím umožňuje vzhledově upravit aplikaci podle požadavků zadavatele.

[1]- základní webový informační systém

4.1.1 Zpracování parametrů volání

Všechny parametry volání jsou reprezentovány pomocí entit. Parametry volání aplikace jsou získávány ze třech různých míst a to:

1. z parametrů předaných protokolem HTTP metodou **get**[2] nebo **post**[3] ,
2. ze sezení vytvořeného aplikačním serverem (**session**),
3. ze statické konfigurace aplikace.

K parametrům z různých míst se přistupuje různě a při zpracovaní mají různou prioritu. Parametry ze statické konfigurace aplikace jsou konstantami, které by se v průběhu aplikace neměly měnit. Jejich změna je charakteristická jen pro instalaci pluginů, které si potřebují nastavit výchozí hodnoty pro svůj běh. K jiným změnám dochází při instalaci nebo úpravě aplikace.

Parametry získané ze sezení mají nejmenší prioritu a slouží hlavně k snížení objemu dat pohybujících se mezi uživatelem a aplikací. Takovéto ukládání parametrů je nezávislé na existenci databázového připojení. Proto je jej možné využít i pro jednoduché aplikace.

Parametry získané z požadavku jsou většinou aktualizující, a proto jsou nadřazeny nad parametry ze sezení. Z této skupiny jsou upřednostňovány parametry odeslané v požadavku metodou **post**. Oproti odesílání pomocí metody **get**, jsou tyto parametry více chráněné před ruční změnou uživatelem.

4.2 Database Plugin

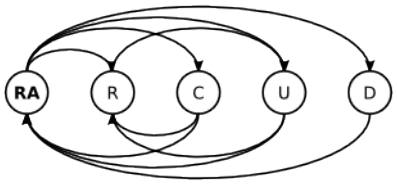
Pro komunikaci s databázovým serverem byl vytvořen plugin *Database*. Tento plugin existuje ve dvou variantách a to pro databáze typu **MySQL**[4] a **MSSQL**[5] . Je postaven jako připojení. Mimo standardního dotazu poskytuje různá zapouzdření pro přidání nebo mazání prvku v databázi, výběr prvků v podobě seznamu nebo výběr jen prvního prvku , rovněž vkládání prvku a případně jeho aktualizaci s parametry ve formě entity, kde je nutné dodržet pojmenování parametrů v entitě shodné s pojmenováním sloupců v tabulce.

[2] parametry jsou předávány přímo v url volání
[3] parametry jsou předávány voláním, ale nejsou viditelné v url
[4] populární relační databáze pro tvorbu webových aplikací
[5] relační databáze od společnosti Microsoft

4.3 CRUD operace s daty

Zkratkou **CRUD** je označován soubor operací nad daty, který zahrnuje operace **Create** – vytvoření prvku, **Read** – načtení prvku, **Update** – aktualizaci prvku, **Delete** – smazání prvku. Zapouzdření těchto operací poskytuje plugin *CRUD*. Tento plugin však není soběstačný a veškerá potřebná funkčnost, která se netýká přímo CRUD operací je dodávána jinými pluginy. Standardní operaci **Read** rozděluje tento plugin na dvě operace a to **ReadAll**, pro načtení seznamu položek a **Read** pro načtení detailu.

Plugin *CRUD* poskytuje jednotné uživatelské prostředí. Pro každou akci kromě mazání poskytuje formulář. Formuláře jsou ve velké míře upravovány podle potřeb obsluhované datové struktury, ale hlavní prvky (tlačítka akcí, nadpis, rozložení) zůstávají pořád stejné, takže uživatel může používat aplikaci s vědomím, že jeho akce mají stejný dopad v každé její části.



Obrázek 4.1: Graf možných přechodů mezi akcemi

Plugin *CRUD* je postaven pro vytváření **agend**. Agenda je část aplikace, která se stará o jednu datovou strukturu, nebo o jednu datovou strukturu z jednoho úhlu podhledu. Agendy jsou postaveny nad databázovými strukturami. Databázové struktury jsou prezentovány jednak tabulkami a rovněž pohledy. Použití pohledů minimalizuje počet databázových dotazů z aplikace. Nechávají totiž zodpovědnost za vytvoření datové struktury závisející na více tabulkách přímo na databázovém serveru. Pohledy se používají při získání dat pro akce seznamu a detailu. Databázové tabulky se pak využívají při zbylých akcích jako přidání, aktualizace a mazání.

V agendě je pak nutné definovat jak tabulky a pohledy, tak sloupce, jejichž hodnoty potřebujeme z databáze použít. Tabulky resp. pohledy je možné definovat zvlášť pro seznam a pro detail. Pro ostatní akce se používá jednotná definice tabulky. Sloupce je možné definovat pro každou akci zvlášť, nebo je možné použít dědění definice.

V tomto pluginu je zakomponována speciální akce **SubReadAll**, která vrátí mini-

malizovaný seznam prvků. Tato akce se používá u detailu prvky na zobrazení navázaných prvků. Když máme například uživatele a otevřeme jeho detail, je možné k němu prostřednictvím této akce zobrazit také jeho role, objednávky nebo jakékoliv jiné záznamy, které jsou k němu navázány.

4.3.1 Uživatelské prostředí

Formuláře pluginu *CRUD* jsou vytvořeny pomocí *TemplateView*. Formuláře ale obsahují rovněž kontrolní prvky jako tlačítka, popisky vstupních polí a různorodá vstupní pole. Pro každou agendu je potřeba tyto pole definovat podle požadavků datové struktury. To znamená vytvořit textové pole pro prvky přímo z databáze, vytvořit výběrová pole pro prvky z jiných tabulek, které jsou nebo budou navázány na aktuálně zobrazený prvek atd. Pro tento účel byl vytvořen plugin *Controls*. Obsahuje zapouzdření ovládacích prvků HTML formuláře.

Protože prostředí BaWIS by mělo být co nejuniverzálnější a zároveň zastávat úlohu unifikovaného prostředí, nastala otázka, co přizpůsobovat a co ne. Prostředí a ovládání zůstalo stejné pro všechny varianty. Popisky, myšlena komunikace s uživatelem tak nemohla zůstat konstantní. Prostředí BaWIS je připraveno i na vícejazyčnou úpravu a to s pomocí pluginu *Translator*. Pomocí konfiguračního nástroje jádra BaWIS je při instalaci nastavena výchozí jazyková varianta, v tomto případě česká mutace. Tento plugin obsahuje možnost nastavit jazykovou variantu za chodu aplikace a uchovávat si ji v session a tím umožnit použitelnost aplikace pro různé jazykové skupiny uživatelů.

4.3.2 Systémová funkčnost

Pro operace nad daty je v hojném množství využívána část jádra BaWIS a to spojení s databází. Pro zjednodušení práce s dotazy je použit plugin *SqlBuilder*, který využívá *TemplateView* k vygenerování dotazu. Pro výběr seznamu prvků je také použit stránkovač *Pager*, který je součástí *SqlBuilder* pluginu. Stránkovač poskytuje jednoduchou práci s velkým množstvím dat, které by bylo samo o sobě pro uživatele nepřehledné. Dokáže vygenerovat parametry dotazu, které určí přesnou část prvků, které se má z databáze vybrat. Přechody na jiné stránky jsou řešeny pomocí parametrů požadavků, které udávají aktuální stránku, která se má zobrazit.

Pro urychlení komunikace s databází, je stejně jako při autoloadingu použito cache-

ování. Toto cacheování globálně nabízí plugin *ObjectCacher*. Má všeobecnou funkčnost – ukládat serializované objekty na základě klíče a mazat je podle maximálního stáří záznamů, nebo podle maximálního počtu cacheovaných záznamů. Zatím je použit při odesílání dotazů pomocí pluginu *SqlBuilder*. Jako klíč pro cache je použit hash řetězce dotazu a jako výsledek je dodána serializovaná hodnota výsledku. Pokud se žádná hodnota pod zadaným klíčem nenachází v cache, je vrácena hodnota *false*. Hodnoty záznamů jsou ukládány do souborů pod jménem shodným s klíčem záznamu.

4.4 Aplikační komponenty BaWIS

Pluginy ve frameworku BaWIS jsou rozděleny do dvou kategorií. První je systémová a do ní spadají všechny výše vyjmenované pluginy. Jsou to pluginy, které jsou s dodržením všech závislostí použitelné i mimo BaWIS. Další skupinu pluginů tvoří aplikační pluginy, které dodávají aplikaci funkčnost využitelnou uživatelem.

4.4.1 Uživatelé

Pro organizaci a správu uživatelů je vytvořena agenda v pluginu *User*. Agenda je postavena nad jedinou tabulkou a to *user*. Plugin *User* dále poskytuje přihlašovací formulář a akce pro ohlášení uživatele. V systémovém pluginu *CRUD* bylo k této příležitosti vytvořeno zapouzdření hlavního modulu do modulu *AuthCrudModule*. Ten zjišťuje přítomnost definice entity *User* v session. Jestli se tam tato definice nenachází, je požadavek přesměrován na přihlašovací formulář. Jinak je použit požadovaný modul. Tato kontrola probíhá pro každou akci v každém požadavku.

4.4.2 Menu

Plugin *Menu* poskytuje horní hlavní menu aplikace. Poskytuje také agendu pro úpravu tohoto menu. Menu reprezentuje série odkazů na moduly aplikace. Je možné je upravovat co do obsahu, tak i do pořadí. Jednotlivé položky menu jsou definovány jménem modulu, jménem akce a dodatečnými parametry odkazu.

4.4.3 Rozložení

Tento plugin je ukázkovým přetížením standardního výstupu poskytovaného jádrem BaWIS. Plugin *Layout* obohacuje standardní šablonu o nezbytné připojení kaskádových stylů a javascriptů potřebných pro správné zobrazení stránky. Používá se jako obálka výstupu jiných modulů. K tomuto výstupu připojuje také menu, které je poskytováno pluginem *Menu*, a standardní hlavičku s logem frameworku. K samotnému menu připojuje jméno aktuálně přihlášeného uživatele a odkaz na odhlášení.

Kapitola 5

Projekt IVY a blog webroute.bigant.eu

Jako demonstraci funkčnosti CMS projektu IVY jsem se rozhodl udělat vlastní blog zaměřený na tvorbu webových aplikací, design a programování. Blog by měl obsahovat základní vlastnosti jako seznam článků, jejich detail a komentáře uživatelů.

Pro usnadnění práce s aplikací jsem se rozhodl rozdělit aplikaci na dvě. Obě budou fungovat nad stejnou databází, tudíž budou obě sdílet stejná data. Na obě části bude použit framework BaWIS. Založení obou aplikací na stejném frameworku usnadní jejich údržbu a kompatibilitu. Zároveň jejich rozdělení zpřehlední kompetence jednotlivých pluginů. Pokud se nedopatřením jedna aplikace stane nefunkční, neohrozí to chod druhé aplikace.

5.1 Projekt IVY

Jednou ze dvou aplikací je projekt CMS IVY. Bude sloužit jako administrace blogu. Z frameworku BaWIS je použito maximum. Je doplněn o další pluginy s agendami na správu obsahu. Projekt IVY je aplikací, která může být využita k vytvoření i jiných blogů.

5.1.1 Datový model

Datový model aplikace je poměrně primitivní. K standardnímu datovému modelu frameworku BaWIS, který obsahuje tabulky pro uživatele a menu, je přidána tabulka, která

obsahuje data související s články blogu. Článek obsahuje identifikátor, titulek, obsah, datum vytvoření, příznak publikace a id uživatele, který ho vytvořil.

Pro formát textu článku jsem využil formátovací nástroj **Texy**[1] . Formát je vzdáleně podobný formátu článků používaném na portálu **Wikipedia**. Jedná se o strukturovaný text s jednoduchými značkami. Text napsaný pomocí Texy může vypadat například takto:

```
Základné UIDP
-------------

**Exchange - Sprostredkovanie**

* Užívateľ používa natívnu formu pre poskytnutie
  informácie systému, akú používa na štandardnú
  výmenu informácií (text, statický obraz, video,
  reč, ...) - vstupné prvky majú maximálnu variability
* Samotná informácia nijak nemení beh systému
* Systém poskytuje informáciu v nezmenenej podobe - autenticky
```

Překlad do HTML zápisu bude mít následující podobu:

Základné UIDP

Exchange - Sprostredkovanie

- Užívateľ používa natívnu formu pre poskytnutie informácie systému, akú používa na štandardnú výmenu informácií (text, statický obraz, video, reč, ...) - vstupné prvky majú maximálnu variability
- Samotná informácia nijak nemení beh systému
- Systém poskytuje informáciu v nezmenenej podobe - autenticky

[1] Texy - formátovací nástroj od Davida Grundla, `http://texy.info/cs`

5.1.2 Pluginy

Jako jediný nový plugin do frameworku BaWIS přibyla agenda na správu článků. Plugin *Article* obsahuje *AuthCrudModule* s definicí dat a tabulek a také potomky formulářů, které definují zobrazení některých prvků. Při zobrazení detailu článku je použitý opět nástroj Texy. Ten vygeneruje z článku jeho podobu v HTML. To je zasazeno do formuláře a uživatel může vidět přibližnou podobu výsledné prezentace článku na blogu. Na blogu mohou (a z pravidla taky jsou) použité jiné kaskádové styly než v IVY, a proto se výsledek může lišit v různých aspektech jako např. použitý font, velikost písma, barva popředí a pozadí, atd.

5.2 Blog webroute.bigant.eu

Pro prezentaci článku pro veřejnost je vytvořena druhá aplikace, která nese jako jméno adresu samotného blogu čili **webroute.bigant.eu**. Z frameworku BaWIS už nebylo použito tolik. Bylo použito jeho jádro a pluginy pro komunikaci s databází.

5.2.1 Pluginy

Stejně jako v projektu IVY byl i zde vytvořen plugin *Article*. Tento plugin má na starosti kompetence ohledně zobrazování článků pro uživatele. Uživatelem se v tomto smyslu rozumí jakýkoliv návštěvník blogu.

V tomto pluginu je rovněž použit nástroj Texy pro zobrazení obsahu článku. Nástroj sám generuje stejný HTML výstup jako v případě administrační aplikace. Výstup je však ovlivněn kaskádovými styly, které jsou na stránce použity.

Obálku pro výstup pluginu *Article* tvoří plugin *Layout*. Tento plugin je specificky navržen právě pro tento blog. Mimo standardní HTML strukturu jsou přidány kaskádové styly stránky. Tento plugin přidává na výslednou stránku logo blogu a blok odkazů na mnou doporučované stránky. Tyto části jsou definovány staticky a nelze je dynamicky měnit. Jejich změna musí být provedena přímo v šabloně, která se používá pro vygenerování výstupu.

5.2.2 Externí služby

Aby bylo uživatelům umožněno reagovat na obsah článků a sdílet je na sociálních sítích, byly k článkům přidány formuláře pro komentáře prostřednictvím služby **disqus**[2]. Tato služba poskytuje skript, který komunikuje se vzdálenou webovou aplikací. Tato aplikace poskytuje řadu možností ohledně spravování reakcí uživatelů. Uživatelé v těchto reakcích můžou vystupovat anonymně, nebo svoji totožnost prezentovat pomocí jednoho ze svých účtů na sociálních sítích. Svůj názor pak mohou vyjádřit pomocí kladného nebo záporného ohodnocení, připsáním komentáře, nebo sdílením článku. Výpis komentářů je připojen v přesně specifikovaném místě, v případě blogu *webroute.bigant.eu*, pod článkem v jeho detailu. Skript, který komunikuje se vzdálenou webovou aplikací, pod detail článku, dodává plugin *Article*.

Monitorováni aktivit na stránce pomocí aplikace **Google Analytics** je zajištěno pluginem *Layout* a to přidáním skriptu do obálky výstupu. Ten potřebné informace o aktivitě předává vzdálené aplikaci. Aplikace **Google Analytics** dokáže z těchto informací generovat grafy a statistiky o návštěvnosti stránky, geografickém místě, odkud byla stránka navštěvována a také informace o návštěvnostech jednotlivých částí stránky.

[2] http://disqus.com

Kapitola 6

Licence

Výsledek mé bakalářské práce, stejně jako mnou vytvořené frameworky, bych chtěl publikovat a poskytnou ostatním programátorům. Aby tato aktivita nebyla v rozporu s univerzitním nařízením a se zákony České Republiky, jako součást práce jsem měl za úkol vypracovat rešerši na téma licencování software. V následujícím srovnání licencí se zaměřím na nejdůležitější open-source licence, které upřesňují šíření software volně, bez poplatků spojených se samotným použitím nebo distribucí software.

Licence pro software obvykle vycházejí z autorského zákona. Dílem se v licencích rozumí zdrojové, potažmo kompilované kódy programu. Licence pro software oddělují samotný program od jeho výstupu, dokud jeho výstupem není stejný program, nebo jeho pozměněná verze. U webových aplikací je tedy možno uplatnit licenci přímo na program a přímo na jeho výstup. U databází se tato licence aplikuje zvlášť na strukturu a na data, jelikož autor těchto dvou částí nemusí být stejný.

6.1 GNU General Public Licence

Historie GPL[1] se odvíjí od projetku GNU. Nejvýznamnějšího použití se jí dostalo od Linuse Torvaldse, který ji použil pro linuxové jádro. Tuto licenci začalo využívat velké množství open-source projektů a v dnešní době ji využívá více než polovina.

Znění licence je dostupné každému, kdo získá software šířený pod touto licencí, protože je vždy u takovéhoto produktu přiložena. Licence opravňuje majitele, který s ní souhlasí k modifikování, kopírování a distribuování tohoto díla i jakéhokoliv odvozeného

[1] http://www.gnu.org/licenses/gpl.html

díla. Mimo binární verzi díla je autor nucen poskytnout k dílu i jeho zdrojové kódy. Licence neukládá uživateli ani autorovi povinnost toto dílo distribuovat bezplatně, nýbrž umožňuje stanovit jakékoliv poplatky. Dílo, ani pozměněné, se nesmí distribuovat pod licencí, která by byla v rozporu s GPL.

6.2 Eclipse Public Licence

Licence EPL[2] je odvozena od svého předchůdce CPL (Common Public Licence). Licenci CPL publikovala firma IBM. EPL je orientována více na trh se software. Její pravidla neukládají tolik povinností autorovi díla. Majitel této licence může dílo modifikovat, kopírovat a distribuovat. Někdy má však povinnost vydat své vlastní změny tohoto díla. Licence není kompatibilní s GPL, a proto části díla vydané pod EPL a části pod GPL nemůžou být z právního hlediska publikovány společně.

6.3 MIT Licence

Licence byla vytvořena na MIT[3] (Messachusetts Institute of Technology). Software vydaný pod touto licencí může být distribuován společně jak s proprietárním software, tak se software publikovaným pod GPL, protože licence GPL explicitně povoluje šíření společně s MIT licencí.

6.4 Creative Commons

Creative Commons[4] poskytuje 16 licencí, přičemž tyto licence jsou kombinací následujících čtyř základních prvků, které můžou být považovány za samostatné licence.

Attribution (by) Povoluje majiteli licence kopírovat, modifikovat a distribuovat dílo jedině společně se jménem autora originálního díla.

Noncommercial (nc) Majitel licence může kopírovat, modifikovat, distribuovat a používat dílo jen pro nekomerční účely.

[2] http://www.eclipse.org/legal/epl-v10.html
[3] http://www.opensource.org/licenses/mit-license.php
[4] http://creativecommons.org

No Derivative Works (nd) Majitel licence může kopírovat, distribuovat a používat dílo, ale nesmí vytvářet jeho modifikace.

Share-alike (sa) Majitel licence může distribuovat dílo jen pod stejnou nebo kompatibilní licencí.

Tyto čtyři části ukládají majiteli licence různé povinnosti. Autoři děl si můžou vlastní licenci nakombinovat a vydat dílo pod výslednou kombinací. Licence je upravena pro více států s ohledem na právní úpravy daných zemí.

6.5 Licence pro SFW, BaWIS, IVY a webroute.bigant.eu

Pro tyto čtyři projekty bych zvolil jednu z kombinací licencí Creative Commons. Je to kombinace **Attribution** a **Noncommercial**, ve zkratce **bync**. Každý může projekty se jménem autora originálního díla pro nekomerční, tedy i výukové, účely. S těmito licencemi mám již zkušenosti při publikování jiných děl jak z oblasti software, tak z oblasti počítačové grafiky. Tato kombinace si neodporuje s prohlášením z úvodu této práce.

Kapitola 7

Závěr

V mé práci se mi povedlo vytvořit vhodný model pro modulární webovou aplikaci. Tento model byl dále rozdělen na jednodušší části, ze kterých vzešly dva frameworky SFW a BaWIS. Na jejich základě jsem pro demonstraci vytvořil blog webroute.bigant.eu a jeho administrační rozhraní IVY.

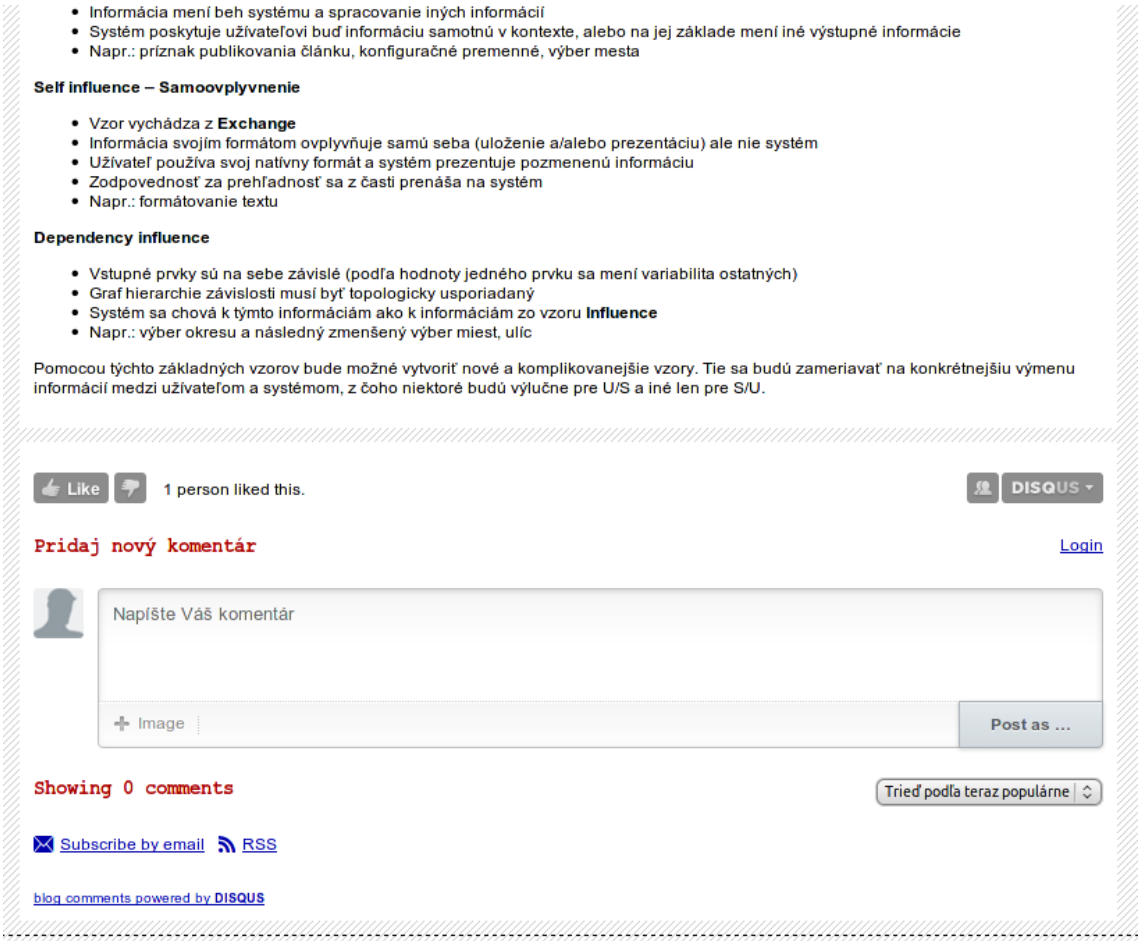
Všechny tři části je možné dále upravit a rozvinout pro jiné účely. Pomocí frameworků SFW a BaWIS lze vytvořit mnoho variant informačního systému v podobě webových aplikací. Z projektu IVY a blogu webroute.bigant.eu lze zase vytvořit prezentační a redakční portál.

Příloha A

A.1 Blog webroute.bigant.eu

A.1.1 Titulní stránka

A.1.2 Komentáře ze služby disqus



- Informácia mení beh systému a spracovanie iných informácií
- Systém poskytuje užívateľovi buď informáciu samotnú v kontexte, alebo na jej základe mení iné výstupné informácie
- Napr.: príznak publikovania článku, konfiguračné premenné, výber mesta

Self influence – Samoovplyvnenie

- Vzor vychádza z **Exchange**
- Informácia svojím formátom ovplyvňuje samú seba (uloženie a/alebo prezentáciu) ale nie systém
- Užívateľ používa svoj natívny formát a systém prezentuje pozmenenú informáciu
- Zodpovednosť za prehľadnosť sa z časti prenáša na systém
- Napr.: formátovanie textu

Dependency influence

- Vstupné prvky sú na sebe závislé (podľa hodnoty jedného prvku sa mení variabilita ostatných)
- Graf hierarchie závislosti musí byť topologicky usporiadaný
- Systém sa chová k týmto informáciám ako k informáciám zo vzoru **Influence**
- Napr.: výber okresu a následný zmenšený výber miest, ulíc

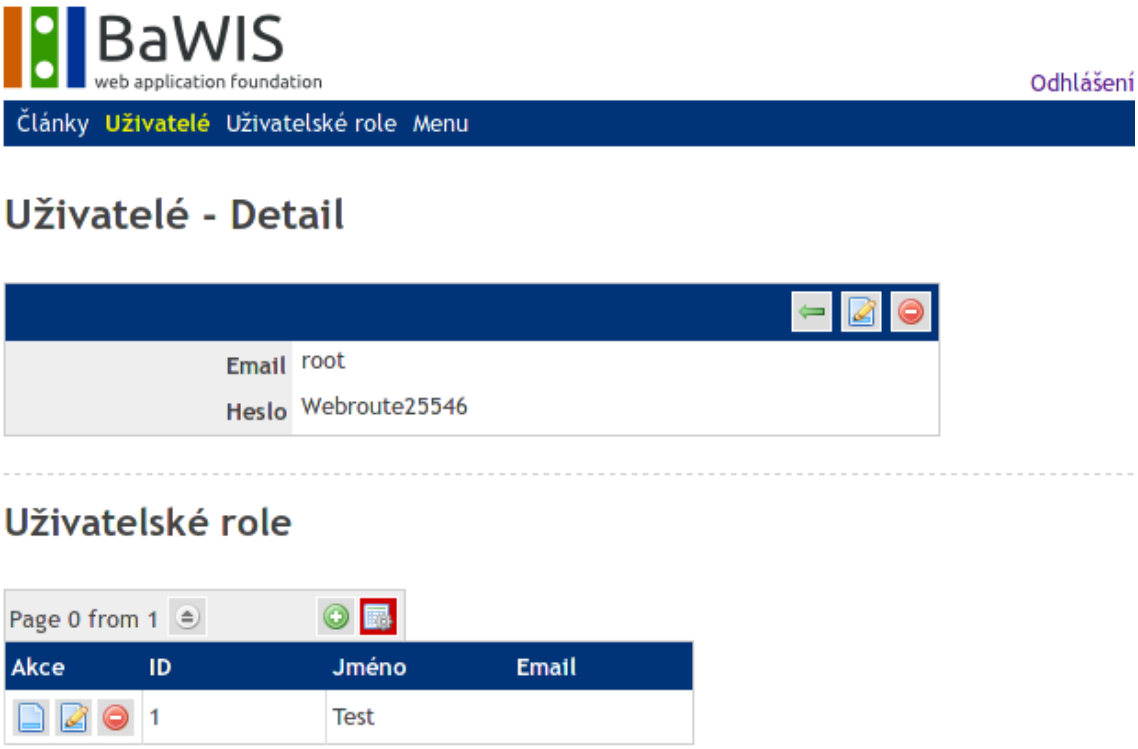
Pomocou týchto základných vzorov bude možné vytvoriť nové a komplikovanejšie vzory. Tie sa budú zameriavať na konkrétnejšiu výmenu informácií medzi užívateľom a systémom, z čoho niektoré budú výlučne pre U/S a iné len pre S/U.

A.2 Administrační rozhraní BaWIS, IVY

A.2.1 Seznam článků

A.2.2 Editace článků

A.2.3 Detail uživatele se seznamem rolí



Příloha B

B.1 Ukázky kódu

B.1.1 CRUD Modul ArticleModule z pluginu Article

```php
<?php
class ArticleModule
extends AuthCrudModule {
    public $table = 'article';
    public $readAllFields = array('title', 'create_dt', 'published');
    public $readFields = array('title', 'create_dt', 'content', 'published');
    public $updateFields = array('title', 'content', 'published');
    public $order = 'create_dt DESC';
    public $formTitlePrefix = 'article';
}
class ArticleCUForm
extends CUForm {
    function content() {
        $value = $this->entity->GetVal(__FUNCTION__);
        $out = "<textarea class=\"richtext\" name=\"content\">$value</textarea>";
        return $out;
    }
    function title() {
        $value = $this->entity->GetVal(__FUNCTION__);
        $fld = new InputField();
        $fld->type = 'text';
        $fld->entity->SetVal('title', 'class');
        $fld->value = $value;
        $fld->name = __FUNCTION__;
        return $fld->Render();
    }
    function published(){
        $value = $this->entity->GetVal(__FUNCTION__);
        $name = __FUNCTION__;
        $checked = '';
        if($value == 1)
            $checked = 'checked="checked"';
        return "<input type=\"checkbox\" name=\"$name\" $checked value=\"1\" />";
    }
}
class ArticleReadForm
extends ReadForm {
    function content() {
        $value = $this->entity->GetVal(__FUNCTION__);
        $texy = new Texy();
        $content = $texy->process($value);
        $out = "<div style=\"width:500px;display:inline-block;\">$content</div>";
        return $out;
    }
}
```

Příloha C

C.1 Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře:

bawis framework BaWIS a SFW

ivy administrační rozhraní

webroute blog webroute.bigant.eu

install popis instalace a scripty pro naplnění databáze

bp elektronická verze této práce