VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

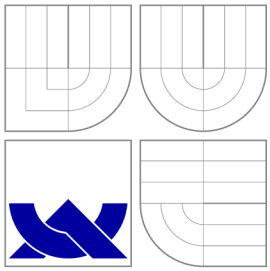
SOFTWAROVÉ ARCHITEKTURY A NÁVRHOVÉ VZORY VE WEBOVÝCH APLIKACÍCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

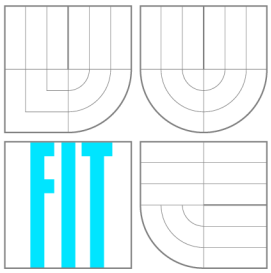
AUTOR PRÁCE
AUTHOR

JAN KAŠPAR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SOFTWAROVÉ ARCHITEKTURY A NÁVRHOVÉ VZORY VE WEBOVÝCH APLIKACÍCH

SOFTWARE ARCHITECTURES AND DESIGN PATTERNS IN WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN KAŠPAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DUŠAN VRÁŽEL

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Kašpar Jan**
Obor: Informační technologie
Téma: **Softwarové architektury a návrhové vzory ve webových aplikacích**
Kategorie: Web

Pokyny:

1. Seznamte se moderními softwarovými architekturami a návrhovými vzory pro informační systémy a webové aplikace.
2. Vyberte tři návrhové vzory a porovnejte jejich výhody a nevýhody při použití v prostředí webu. Výběr konzultujte s vedoucím.
3. Navrhněte vhodnou webovou aplikaci pro demonstraci výhod a nevýhod jednotlivých vzorů. Při návrhu využijte jazyka UML.
4. Implementujte navrženou aplikaci s využitím vybraných vzorů. Rozsah implementace konzultujte s vedoucím.
5. Ověřte funkčnost aplikace na vhodně zvoleném vzorku dat.
6. Zhodnoťte dosažené výsledky. Zaměřte se na porovnání výhod a nevýhod implementace aplikace dle jednotlivých návrhových vzorů.

Literatura:

- Úvod do návrhových vzorů dostupný na adrese http://objekty.vse.cz/Objekty/Vzory.
- Fowler M.: Patterns of Enterprise Application Architecture. Addison-Wesley Professional 2002.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vrážel Dušan, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Kašpar**
Id studenta: 86859
Bytem: Nad Obcí II 50, 140 00 Praha
Narozen: 02. 08. 1983, Praha
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP:	Softwarové architektury a návrhové vzory ve webových aplikacích
Vedoucí/školitel VŠKP:	Vrážel Dušan, Ing.
Ústav:	Ústav informačních systémů
Datum obhajoby VŠKP:	.....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické formě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Práce popisuje charakteristické rozdíly mezi softwarovými architekturami a návrhovými vzory pro webové aplikace. Hlavní teoretickou částí je rozbor a porovnání tří vzorů určených pro stavbu webových aplikací a jejich následovné použití v praxi. V jazyce UML je třeba navrhnout aplikaci (případ užití, schéma databáze a objektů apod.) a následně ji implementovat za použití vybraných vzorů. Výsledkem bude zhodnocení práce s každým vzorem, jeho výhody, nedostatky a nejlepší oblast použití.

Klíčová slova

Softwarové architektury, návrhové vzory, Model-View-Controller, layers, pipes and filters, vrstvy, roury, webové aplikace, web, php, uml

Abstract

This text describes software architecture and design patterns for web applications together with theirs advantages and disadvantages. The main theoretical part is focused on detailed description and comparison of three patterns and their practical use. A schema of an application with all its parts (use-case diagram, database and objects schema) will be created using UML and subsequently this application will be implemented using the tree chosen patterns. Based on this implementation, the best use for each pattern will be evaluated in the final part of this text.

Keywords

Software architektures, design patterns, Model-View-Controller, layers, pipes and filters, web applications, web, php, uml

Citace

Jan Kašpar: Softwarové architektury a návrhové vzory ve webových aplikacích, bakalářská práce, Brno, FIT VUT v Brně, 2008

Softwarové architektury a návrhové vzory ve webových aplikacích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Dušana Vrážela Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

.....
Jan Kašpar
13. května 2008

Obsah

Úvod

Práce je v teoretické části zaměřena na softwarové architektury a návrhové (architektonické) vzory pro webové aplikace. Je rozebírána funkčnost, klady a zápory těchto dvou oblastí v prostředí webu. Závěrem bude aplikace implementovaná za použití tří vzorů a porovnání výhod a nevýhod vybraných vzorů na základě implementace.

V první kapitole je teoretický rozbor vlastností a možností software na základě zadání - softwarové architektury a návrhové vzory. Je zde celkově nastíněna problematika tématu, historické aspekty vzniku a na konec detailní ukázka typických reprezentantů pro každou oblast. Důležité je shrnutí klíčových vlastností každé kategorie.

Druhá část se týká samotné aplikace, ale opět na teoretické úrovni. Zabývá se analýzou a specifikací požadavků na aplikaci, objektově-relačním návrhem pomocí jazyka UML (studijní materiály [2] a [10]), specifikací schématu databáze a základem implementace.

Třetí kapitola se zabývá použitím jednotlivých vybraných vzorů při vytváření konkrétní aplikace. Je zde podrobně rozebrána implementace a srovnání zdrojového kódu vybrané části aplikace při použití jednotlivých vzorů.

Čtvrtá kapitola popisuje proces testování a ověřuje tak implementaci na základě návrhu.

V závěru textu je vyhodnocena použitelnost vybraných vzorů a jejich hlavní klady a zápory při použití v prostředí webu.

Kapitola 1

Teoretická část

1.1 Softwarové architektury

> *Softwarová architektura* je struktura komponent programu/systému, jejich vzájemné vazby, principy a předpisy určující jejich návrh a vývoj v průběhu času.

Toto je znění jedné definice [9] výrazu *softwarové architektury* (SA), ale rozhodně není jediné. Jedná se o strukturu, organizaci a interakci jednotlivých součástí software a programů. S touto definicí je úzce spjata definice kooperace jednotlivých vnitřních komponent. Zároveň se může dotýkat souvislostí s okolním prostředím a vzájemným ovlivněním obou celků. V bodech by se dalo zmíněné shrnout následovně:

- systém součástí
- vztahy jednotlivých částí
- definice struktury
- definice chování

1.1.1 Důvodů je mnoho

S postupným rozšiřováním programování a softwarového inženýrství začaly být programy složitější a jejich organize byla nezbytná. Ze začátku byl software poměrně jednoduchý a jednoúčelový, ale s postupným rozvojem a rozšiřováním se začal stávat složitějším a postupovat na vyšší úrovně - vysokoúrovňové jazyky. A právě v těchto dobách bylo potřeba vnést do programování systém - strukturu, **architekturu**. Časově se jedná o 50. až 60. léta minulého století.

S rozšiřováním software do všech oblastí lidského života a s důrazem na zkoumání potřeb při vývoji programů se začalo rozšiřovat pole programovacích jazyků. Od funkcí a procedur se přešlo na objekty s metodami a prototypování - objektově orientované programování. Tyto přístupy přinášejí možnost vytváření komponent, a jejich znovupoužitelnost. V tomto ohledu se architektura zaměřuje hlavně na důležité současti celku. Není potřeba specifikovat každou drobnou vlastnost všech komponent, ale jasně a přesně vyzdvihnout základní stavební kameny systému.

Důsledek rozvoje a rozšíření software se značně projevil v jeho specializaci - vestavěné, real-time, komponentové nebo se zaměřením na služby (services). Každá oblast se specializovala na jiný typ aplikací s jinými požadavky, což vedlo k rozdílným uskupením architektur. Přestože vzniklo několik různých přístupů k uskupení součástí v jednotlivých

architekturách, některé části a jejich vztahy jsou shodné. V nynější době je zřejmě nejrozšířenejší architekturou typ klient-server, který dosáhl svého rozmachu především díky sítím a internetu (www, ftp, IM) - v tom je jeho podstata. Další hojně používanou je architektura modulární - aktuálně se používá objekt (nebo více objektů) pro reprezentaci jednohu modulu, ale není to podmínkou. Z jiné oblasti bych vyzdvihnul událostmi řízené architektury, jež jsou typické pro grafická rozhraní operačních systémů.

1.1.2 Modulární architektura

> **Vývoj založený na komponentách** („Component-based development“): Systémy mohou být postaveny rychlým a efektivním způsobem převzetím (nebo vygenerováním) velkých externě vyvinutých komponent. Známá architektura umožňuje samostatný a nezávislý vývoj skladebních komponent, kde integrace se stává stěžejním problémem.

Takto zní jedna z mnoha definic [9] modulární architektury. Základním kamenem takové softwarové aplikace musí být nástroj obsluhující jednotlivé zapojené komponenty. Takový základ systému musí mít schopnost připojit nebo odpojit komponentu a další specifické vlastnosti (metody) na základě kontrétní specifikace, napr. přeposílání zpráv a stavů z jedné komponenty na jiné. Komponenta systému je založena na následujících klíčových bodech (zároveň se jedná o kladné vlastnosti).

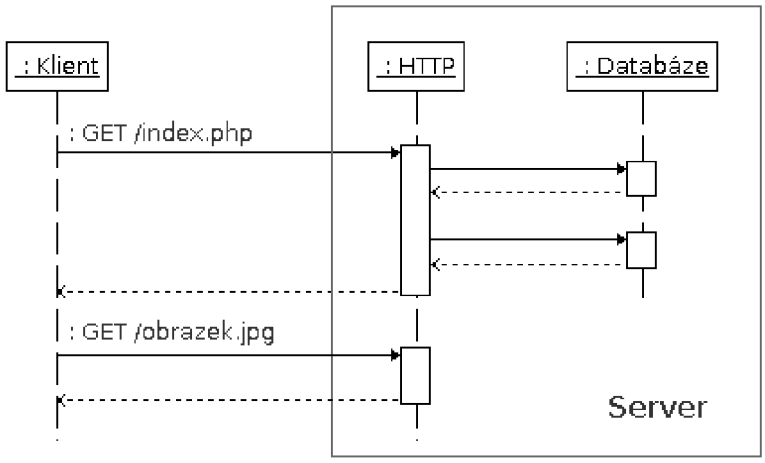
- znovupoužitelnost
- nezávislost na okolí (včetně vývoje)
- použitelnost s jinými komponentami
- zapouzdření - přístupnost přes rozhraní

V dnešním světě jsou modulární programy základem všeho, výrobců jednoho subsystému je více a každý používá jiné rozhraní pro komunikaci. Tento důvod vede k vývoji různých komponent s rozhraním specifickým pro daný produkt, avšak jádro hlavní aplikace může zůstat stejné. Znovupoužitelnost je důležitá charakteristika kvalitní komponenty. Na jedné straně je univerzálnost a míra složitosti použití, naproti tomu je potřeba upravit konkrétní část pro další použití. Při návrhu i implementaci je důležité zvolit adekvátní hranice, ve kterých by se měl modul udržovat - ani moc univerzální a složitý, ani moc konkrétní pro daný případ užití.

1.1.3 Klient-server

Díky sítím a komunikaci mezi počítači vznila architektura typu klient-server. Na jedné straně je server, ke kterému může být připojemo více klientů a každý požadavek klientů je obsluhován samostatně. Klient je aktivní prvek z dvojice, který odešle požadavek na server. Server pasivně čeká na příchozí požadavek, který po obdržení zpracuje a výsledek odešle zpět. Umístění klienta ani serveru není nijak omezené. Oba se mohou nacházet na jednom fyzickém stroji, ale také mohou být vzdáleny přes půl zeměkoule.

Jedná se o centralizovaný systém. Všechny podstatné informace a data jsou umístěna na serveru a každý klient dostane jen ta data, o která požádá. Další dobrá vlastnost se týká hromadné správy, data se nemusí upravovat na více místech ani synchronizovat. Server si



Obrázek 1.1: Ilustrativní schéma klient-server (web server s přístupem do DB).

může, na základě klientem poslaných dat, ověřit jeho totožnost a umožnit, či zamítnout požadavek (např. úpravu kritických dat). Do třetice kladného výčtu: nezávislost softwarového vybavení. Na straně serveru může běžet jiná verze služby (starší), ale klient může pracovat s nejnovějšími prostředky a aplikacemi - za podmínky dodržení kompatibilního komunikačního protokolu.

Samozřejmě není vše tak dokonalé, jak může na první pohled vypadat. Množství přenášených dat nebo zahlcení serveru požadavky od velkého počtu klientů - to jsou hlavní problémy této architektury. Jeden z nedostatků je neefektivní rozdělení zátěže - hlavní výpočetní výkon je směřován na server.

Klíčové výhody modelu klient-server:

- centrální správa dat
- malé objemy přenášených dat
- klient je jednoduchá aplikace
- dostupnost serveru z různých míst

Nevýhody:

- server vykonává většinu práce - možné zahlcení od klientů
- výpadek serveru ovlivní všechny klienty

1.2 Návrhové vzory

Návrhový vzor představuje obecné řešení problému, který se opakovaně objevuje při návrhu softwaru. [4] Jako základní stavební jednotka se uvažuje objekt, ale nejedná se o kontrétní popis struktury. Vzory ukazují obecné vztahy, chování a komunikaci mezi jednotlivými třídami a objekty. Jinak řečeno, ukazují cestu, jakou se vydat při řešení problému ve stádiu návhu aplikace. V žádném případě se nejedná o reálný kód, jež by se dal okamžitě nasadit do vyvíjené aplikace. Nejedná se ani o jedno z následujících:

- **algoritmus** - řeší výpočetní problém (nikoli návrhový), může být reálná část kódu k nasazení
- **komponenta** - je větší část aplikace, může být znovupoužita v několika systémech
- **framework** - jedná se přímo o kolekci komponent nebo zapouzdření nad nižší aplikační logikou a funkčností

1.2.1 Trocha historie

Důvod vzniku vzorů je stálé opakování jednoho problému při návrhu softwarových aplikací. Software na vyšší úrovni se vyvíjí již značnou dobu (přibližně od 50. let), ale řešené problémy při návrhu a implementaci zůstávají stejné. Jako historický milník se v literatuře označuje rok 1987[1], kdy Kent Beck a Ward Cunningham začali experimentovat s aplikací vzoru při návrhu software. Od tohoto roku se na vzorech začalo aktivně pracovat a usilovně se hledaly nové. V roce 1995 vznikla úvodní publikace návrhových vzorů: **Design Patterns: Elements of Reusable Object-Oriented Software**.

> Základním kamenem pro návrhové vzory v oblasti informační technologie se stalo dílo **Design Patterns: Elements of Reusable Object-Oriented Software** GAM95. Autorem byl Gang of Four (Gamma, Helm, Johnson a Vlissides) a poprvé bylo představeno na OOPSLA'94[2], kde se setkalo s velkým úspěchem. Tato práce je nazývána knihou knih v oblasti návrhových vzorů a dodnes se jedná o velmi uznávané dílo. Většina dnešních děl o návrhových vzorech vychází z rozdělení, které zde bylo popsáno. Vzhledem k tomu, že tato kniha si udržuje svoji hodnotu i v rychle se vyvíjejícím světě informačních technologií, jedná se opravdu o unikátní dílo. [5]

1.2.2 Specializace vzorů

První definice a kolekce vzorů vznikaly pro jazyky Smalltalk a později C++. S rozšířením počítačů, jejich schopností a s přibývajícím počtem vyšších programovacích jazyků se vzory šířily napříč všemi oblastmi počítačů - včetně webového prostředí. Originální vzory byly jen malá schémata, která se postupně slučovala do větších celků pro řešení rozsáhlejších částí aplikací, až po řešení návrhu aplikace jako celku. Opět by chtělo zdůraznit, že se nejedná o reálné části kódu, ale pouze o návod, jak nejlépe řešit určitý problém.

Dnes je známo poměrně velké množství vzorů a podle své funkčnosti jsou děleny do tří základních skupin. Zaměření skupin je následující:

- **Creational Patterns** - týkají se problému vytváření objektů na nějakém základě nebo pravidle
- **Structural Patterns** - starají se o vhodné uspořádání objektů v systému a jejich přehlednost
- **Behavioral Patterns** - mají co dočinění se správným chováním systému a spolupráce jednotlivých částí

[1] vzory existovaly již před tímto rokem, jen nebyly detailně popsány a uskupeny
[2] konference Object-Oriented Programmin, Systems, Languages and Applications, rok 1994

Každá skupina obsahuje několik konkrétních vzorů. Při jejich vhodném výběru a kombinaci lze dosáhnout složitějších a komplexnějších vzorů pro řešení celých systémů - **architektonické vzory**. Základní návrhové vzory implicitně neřešily záležitosti týkající se výstupu, s webovou aplikací je to jiné. Uživatel zašle na server požadavek a data, server (webová aplikace) vše odpovídajícím způsobem zpracuje a výsledek (HTML) putuje zpátky k uživateli. Dílčí operace se mohou vykonávat v rámci databáze, či jiných datových zdrojů. Tím se celý systém začíná dělit na jednotlivé části - vrstvy. A právě na základě těchto vrstev vznikaly postupně různé vzory a postupy pro řešení celých internetových systémů.

Následující kapitoly 1.2.4, 1.2.5 a 1.2.6 jsou úvodem do vybraných vzorů, pomocí kterých bude navržena a implementována aplikace. Konkrétně se jedná o vzory **Model-View-Controller**, **Layers** (vrstvy) a **Pipes and Filter** (roury). Porovnání jejich vlastností, výhod a nevýhod je shrnuto v kapitole 1.3.

1.2.3 Architektonické vzory

Architekturou se pak rozumí způsob rozdělení systému do velkých částí a způsob jejich interakce. [11] V praxi se jednotlivé části mohou chápat jako vrstvy, kde každá zabezpečuje určitou část funkčnosti aplikace.

Za základ systému se považuje zpracování vstupního požadavku. Podle toho se dynamicky začnou provádět funkce (v rámci objektů metody) dílčích prvků aplikace (jiných vrstev). Podrobná organizace všech komponent už závisí na konkrétním přístupu. Následující kapitoly popisují různé přístupy tohoto uspořádání.

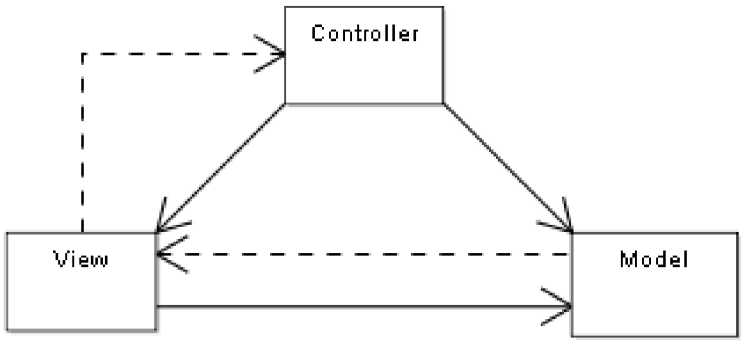
1.2.4 Model-View-Controller (MVC)

MVC je systém složený ze tří hlavních komponent. Obrázek 1.2 ukazuje obecnou komunikaci komponent vzoru MVC (diagram se může lišit v závislosti na implementaci).

Controller (řadič) - přijme požadavek od uživatele a podle jeho obsahu volá další metody, nejčastěji aplikační logiku.

Model - je zmíněnou aplikační logikou, která provádí hlavní výpočetní akce, popř. zasahuje k uloženým datům (databáze). Tato část pracuje, v případě potřeby, s datovou vrstvou - databází.

View (pohled) je poslední komponentou součástí systému. Interpretuje načtená data na požadovaný formát výstupu (např. HTML).



Obrázek 1.2: Obecný diagram komunikace Model-View-Controller. Zdroj [3].

Tento vzor je v poslední době hojně používán při návrhu aplikací a systémů, a to nejen v prostředí webu. Nespornou zásluhu na tom má znovupoužitelnost již napsaného kódu - Model. Další výhody se týkají samotného oddělení vstupu, logiky a výstupu, každou část je možné přepsat bez ovlivnění zbytku systému. Existuje mnoho rozdílých frameworků pro mnoho programovacích jazyků a pro různorodé použití. Zástupci pro PHP jsou například *Zend Framework* a *CakePHP*, pro jiné skriptovací jazyky existují *Ruby on Rails* (Ruby) nebo *Django* (Python). Rozšiřitelnost je v dnešní době velice žádanou vlastností a tento vzor ji umožňje bez kompromisů.

Výhody:

- rozdělení na základní tři části (řízení, logika, prezentace)
- znovupoužitelnost částí
- různá reprezentace dat bez změny aplikační logiky

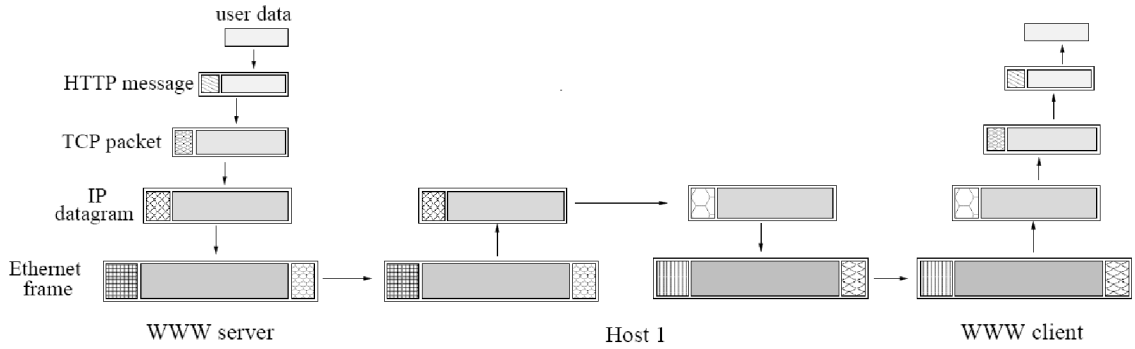
Nevýhody:

- velké systémy mají složitou logiku
- Model je závislý na datovém uložišti (databáze)

1.2.5 Layers - vrstvy

Vrstva je chápána jako dílčí komponenta celého systému, jež vykonává určitou, pro ni specifickou, funkčnost. Kompletní vzor vrstev může vypadat jako lineární obdoba MVC, je zachována logická, prezentační a aplikační vrstva. Zdroj dat - databáze - je zvláštní vrstva. Postupem času se počet vrstev mění - zvětšuje, což je důvod vytváření složitějších aplikací a vhodnou volbou granularity návrhu jednotlivých částí.

Provázanost vrstev je dělena na dva způsoby. První možností je uzavřená komunikace - každá vrstva volá jen metody následující vrstvy. To zabezpečuje lineární závislost vrstev a úpravy v kódu jsou snažší - méně zavislé na okolí. Druhý přístup je otevřená komunikace - vrstvy se mohou volat přes více úrovní. To je možné tolerovat, pokud má návrh hodně specifických vrstev a jejich postupné volání by bylo složité, ne-li nemožné. V případě použití specifického rozhraní se mohou vrstvy měnit, a tím i chování celé aplikace.



Obrázek 1.3: Zapouzdření dat během přenosu po síťových vrstvách. Zdroj [6].

Typické použití vrstev je v prostředí počítačových sítí. Zde jsou vrstvy rozděleny od nejzákladnějších fyzických, až po systémové - zabezpečující konzistenci přenášených dat. Jak ukazuje obrázek 1.3, uživatelská data jsou postupně obalena směrovacími daty a kontrolními součty (HTTP hlavičku, kontrolní součet, IP směrování). Každá vsrtva si přidá potřebná data k již získaným a tento celek dostane následující vrstva. V rámci objektů je tento model chápán jako obalení objektu jiným, jeden objekt (objekt `A`) obsahuje vlastnost s instancí jiné třídy, ke které se přistupuje přes rozhraní objektu `A` - zapouzdření.

Výhody:

- téměř neomezená volnost v rozdělení vrstev do tématických celků
- lineární komunikace vrstev (jednoduchost)
- optimální pro týmový vývoj složitých systémů

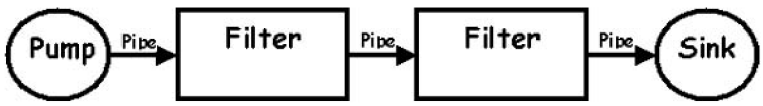
Nevýhody:

- provázanost vrstev a jejich rozhraní
- je třeba zvolit optimální granularitu vrstev
- náročné předávání hodnot přes více vrstev

1.2.6 Pipes and Filters - roury

Každý Filter zpracuje přijatá data a výsledek pošle na vstup dalšího filtru - to je jednoduchá charakteristika tohoto vzoru. Místo spojení filtru a roury se nazývá port. Podle použití a chování portu jsou určeny tři základní typy: vstupní, výstupní a vstupvýstupní. Pojmenování je založeno na použitém portu v sekvenci filtrů.

- **vstupní** - pouze dostává data a neprodukuje je na výstup, většinou data vypíše na obrazovku
- **výstupní** - většinou první Filter, sám si načítá data z globálního prostředí nebo z datového zdroje
- **vstupvýstupní** - pracuje přímo s daty a jejich hodnotami, výsledek předá dalšímu filtru



Obrázek 1.4: Grafické znázornění vzoru Pipes and Filters. Zdroj [8].

Tento vzor dovoluje extrémní nezávislost jednotlivých komponent. Předávají se pouze data a každý filtr si zpracuje pouze jemu určená data. Jednotné musí být pouze rozhraní pro tok mezi filtry.

V dnešní době je model rour (pipes) hojně používán v mnoha prostředích (překladače zdrojových kódů, příkazová řádka nebo celé grafické prostředí) [7]. Sekvence příkazů dokáže

ze vstupních dat, které uživatel zadal, vybrat, filtrovat nebo přeskupit ty podstatné. Obecně se jedné o sekvenci volání událostí, každý filtr je volán po skončení předešlého a dostává jeho výstupní data na svůj vstup.

Výhody:

- flexibilita je umožněna výměnou filtrů
- flexibilita rekombinací - to umožňuje vytvořit novou zpracovávající sekvenci změnou filtrů, nebo přidáním dalších filtrů
- opakované využití komponenty filtru

Nevýhody:

- sdílení stavových (globálních) informací je náročné
- složité zpracování chyb
- jednotné rozhraní všech filtrů

1.3 Porovnání

1.3.1 Architektura versus vzor

Na závěr kapitoly shrnutí a porovnání základních bodů. V obou případech (architektura i vzor) se jedná o práci se software, uspořádání jednotlivých částí i jejich komunikace. Návrhové vzory, jakožto základní prvek systému, jsou speciální případ. Adekvátní porovnání lze provádět až na úrovni softwarových architektur a architektonických vzorů, kde obě oblasti pracují se systémem jako celkem.

Architektura. Pracuje s jednotlivými komponentami systému a poukazuje na jejich vlastnosti, vzájemnou spolupráci a ovlivňování. Vhodně navržená komponenta má široké a snadné uplatnění v dalších projektech - znovupoužitelnost.

Vzor. Uplatňuje se při řešení problému ve fázi návrhu software a je nezávislý na okolním kontextu. Jako základní jednotka je použita třída nebo objekt. Nejedná se o reálnou část aplikace - kód.

Pro přehlednost tabulkové vyjádření vlastností.

	softwarová architektura	návrhový/architektonický vzor
uplatnění	rozvržení komponent	návrh systému
princip	definice, praktické užití	obecné řešení
jednotka	komponenta	třída, objekt
reálné nasazení	ano (komponenty)	ne (nejedná se o reálnou část kódu)

Tabulka 1.1: Výsledky vyhledávání pro různé implementované metody

1.3.2 Porovnání vybraných vzorů

V kapitole bylo probráno teoretické seznámení s klíčovými vlastnostmi vybraných vzorů a bude následovat porovnání hlavních výhod a nevýhod. Model-View-Controller (MVC) a Layers byly vybrány pro jejich rozšířenost a aktuálnost. Na druhou stranu, vzor Pipes and Filters byl vybrán pro svoji částečnou originalitu - zejména v prostředí webu.

První dva vzory jsou již dostatečně ověřeny praxí. Uskupení jejich jednotlivých části je výhodné jak pro vývoj v týmu, tak pro jednotlivce. Striktní určení funkčnosti jednotlivých částí zpřehledňuje vývoj a údržbu systému.

MVC je, jak již název napovídá, rozdělen na tři základní části a pěkně kopíruje cyklus od požadavku od klienta na server až po odpověď - Controller zpracuje vstup, Model se kompletně postará o načtení a zpracování dat a View prezentuje aktualní data jako odpověď serveru. Dalším podstatným kladem v použití s jazykem PHP je šablonování. Možnost začlenění PHP přímo do HTML kódu je snadnou cestou prezentace dat a v tomto vzoru se přímo nabízí. Základní nevýhodou je složitější práce s rozsáhlým modelem, ten je však možné rozdělit na několik základnějších jednotek (datová vrstva, zpracování dat, ošetření vstupních formulářových dat).

Vzor **Layers** může být libovolně rozdělen podle potřeb aplikace na různý počet součástí. Na rozdíl od MVC se jedná o větší počet vrstev - aplikační (datová) nebo prezentační vrstva je moc obecná, a proto se dále dělí. Některé vrstvy mohou být nahrazeny externím programem či aplikací, což může vést k optimalizaci běhu celého systému nebo se celý zdrojový kód zjednoduší. Například použití XSLT[3] jako prezentační vrstvy, ještě důkladněji odliší tuto vrstvu od ostatních a zpřehlední tuto část systému. Hlavní nevýhoda (již zmiňovaná) je volba optimální granularity jednotlivých vrstev (ani moc složitá, ani moc jednoúčelová). Druhý zápor je v průchodu celým systémem formou „tam a zpět“. Prezentační vrstva (chápáno horní) zároveň přijímá požadavek od klienta a musí ho zpracovat. To dělá vrstvu složitější, ale tento neduh může být systémem ošetřen.

Vzor **Pipes and Filters** je používán především v operačních systémech a uživatelských prostředích. V prostředí webu není jeho použití v současné době úplně běžné. Specifické použití je předurčuje přednostně k manipulaci s proudem dat, ale internetové aplikace mohou mít složitější chování. Důležitým krokem při používání tohoto vzoru je volba vhodné formy předávání dat mezi filtry. V případě implementace pomocí objektů je volba jasná, tím se také zajistí dobré předávání hodnot mezi filtry (převzato z [12]). Pojetí je podobné vrstvám (lineární), ale data proudí jedním směrem od prvního filtru k poslednímu.

Shrnutí charakteristik vybraných vzorů.

- **Model-View-Controller** - základní rozdělení součástí, podobnost se schématem klient-server, v současnosti hojně používané v řadě informačních systémů a frameworků
- **Layers (vrstvy)** - rozdělení systému na drobné části, lineární pojetí MVC s drobnějším dělením částí
- **Pipes and Filters (roury)** - jednoduché použití, snadné rozšíření nebo změna funkčnosti, znovupoužitelnost filtrů

[3] The Extensible Stylesheet Language Transformations

Kapitola 2

Návrh aplikace

2.1 Slovníček pojmů

Následující výčet upřesňuje chápání a význam výrazů použitých při návrhu webové aplikace.

- **systém** je serverová část aplikace obstarávající dynamičnost
- **návštěvník** je člověk (požadavek) bez založeného účtu a není přihlášený do systému
- **nepřihlášený uživatel** má založen účet, ale není přihlášen
- **uživatel** má založen účet, na který je přihlášen
- **autentizace** zajišťuje ověření uživatele při požadavku na systém
- **příspěvek** je textový záznam vložený do systému
- **komentář** je text přiřazený právě jednomu příspěvku

2.2 Nefunkční požadavky

Nefunkční požadavky se týkají především návrhu a implementace aplikace. Netýkají se přímo funkcí na rozdíl od požadavků funkčních.

- užití architektonických vzorů (Model-View-Controller, Layers, Pipes and Filters)
- čistě objektový přístup v rámci skriptovacího jazyka
- použití PHP5 a MySQL na serveru (dynamická část a uložiště dat)
- výstup do XHTML a grafické formátování pomocí CSS
- zabezpečení funkcí dostupných pouze uživatelům
- počet uživatelů nepřesáhne 30
- maximálně bude přihlášeno 50 % uživatelů (asi 15)
- příspěvků bude v rámci jednotek tisíců (v závislosti na počtu uživatelů)

2.3 Funkční požadavky

Po analýze a specifikaci požadavků na systém je vhodná doba na shrnutí aktivní funčnosti aplikace - funkční požadavky. Jedná se vlastní funkčnost, kterou bude systém poskytovat.

- návštěvník může pouze prohlížet příspěvky v systému
- přihlášený uživatel může přidávat nové příspěvky a komentáře
- uživatel má k dispozici správu kategorií
- uživatel může upravovat svůj profil (heslo, email a jméno)
- systém dovoluje třídit příspěvky podle kategorie nebo uživatele
- systém dovoluje vyhledávat příspěvky podle textu

2.4 Analýza požadavků

Návrh a analýza aplikace by se měla odvíjet od vybraných návrhových vzorů a jejich schopnosti zvládnout různé požadavky. Rád bych zvolil aplikaci, jež by do poslední maličkosti otestovala schopnosti, klady a zápory daných vzorů v prostředí webového serveru, resp. prohlížeče. Zpracování požadavků od klienta je samozřejmostí. Na základě těch musí aplikace zvládnout vykonat určité kroky a dostat se k požadovanému výsledku - HTML kód.

Práce je soustředěna hlavně na návrhové vzory a aplikace zkoumá jejich nasazení. Neuvažuje se proto zabíhání do oblastí HTML, popř. CSS a grafického pojetí, ani databázových SQL dotazů. Přestože budou v aplikaci použity, text se jim bude věnovat jen minimálně. Taktéž obsahem nebude nápověda pro ovládání aplikace.

Když se vezme v úvahu běžná aplikace pro použití v prostředí webu, je vidět několik základních funkcí:

- **autorizace/autentizace** - ověření přístupu uživatele
- **formuláře** - přenos dat od uživatele systému na server
- **dynamičnost** - ovlivnění výstupu na základě vstupních dat

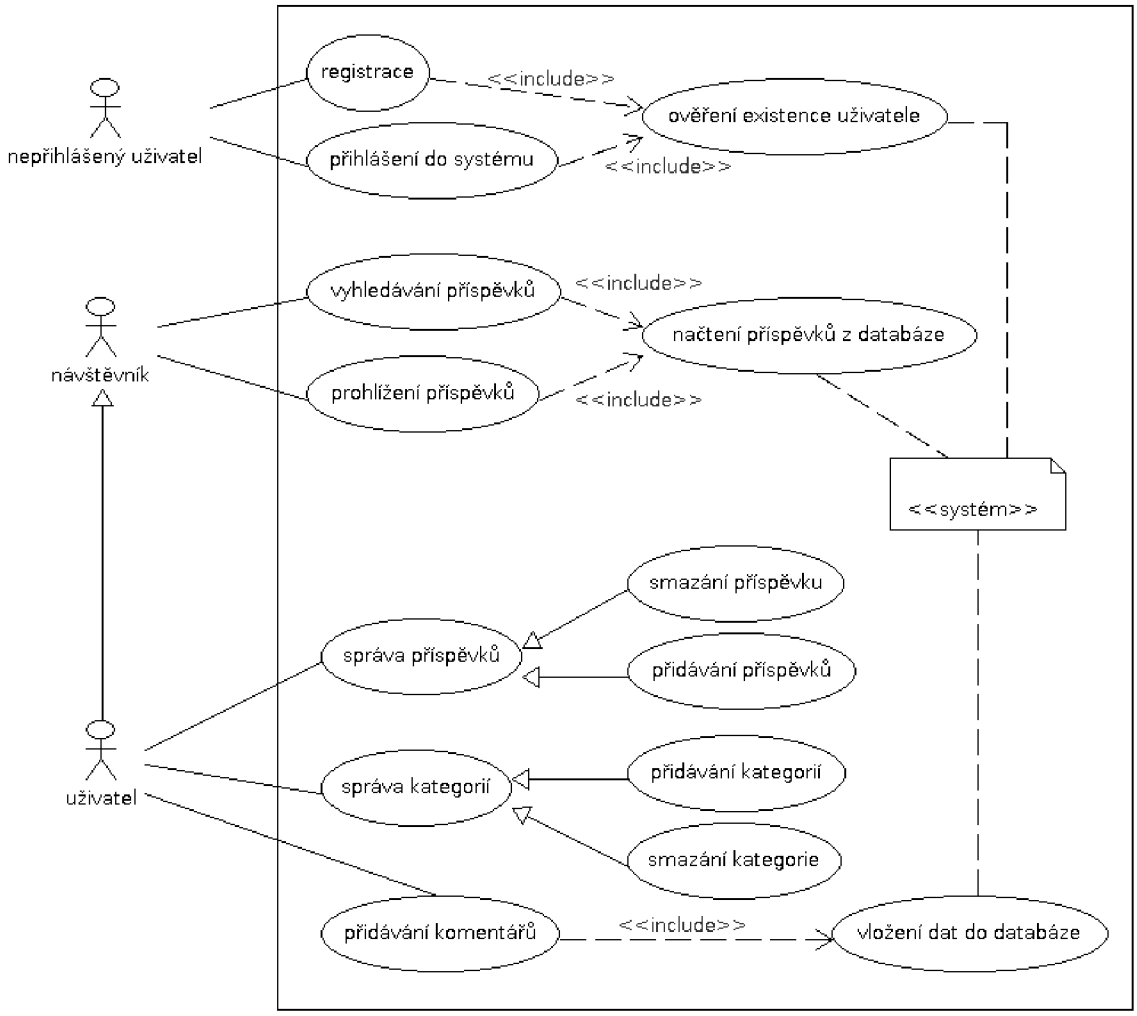
Na základě těchto bodů byla zvolena výsledná aplikace **BLOG** (internetový deník uživatelů). Pro plnohodnotnou funkčnost systému se musí uživatel zaregistrovat a přihlásit - autentizace a formuláře. Formuláře budou dále použity pro vkládání nových záznamů a komentářů do systému. A poslední základní požadavek - dynamičnost - se projeví při prohlížení a filtrování vložených příspěvků.

Volba softwarového vybavení nebude vzhledem k zaměření aplikace žádný složitý krok. V prostředí webu jsou velmi rozšířená open-source řešení, výběr je opravdu široký. Dynamičnost bude obstarávat skriptovací jazyk vyšší úrovně ve spojení s databází na pozadí. Jazyk bude PHP ve verzi 5 a jako uložiště dat poslouží populární MySQL. Novější verze PHP je možné pokládat za jazyk s plnou podporou objektově orientovaného programování s nejmodernějšími vlastnostmi (převzato z [1]). Proto velice dobře poslouží k implementaci všech vlastností a požadavků. Zvolená databáze bude jisto jistě dostačovat, ale má svá omezení v podobě kombinací některých funkcí.

Vzhledem k zaměření použití systému a jeho jednoduchosti, není administrace součástí návrhu ani implementace. Administrace se bude řešit pomocí externí aplikace pro správu databáze uživatelů a příspěvků.

2.5 Případ užití

Základní rámce požadavků na systém jsou tedy dány, nyní je potřeba se vcítit do role uživatele a specifikovat podrobné funkce. V základu bude systém rozdělen na dvě oblasti - návštěvník a uživatel. Rozdělení oblastí je řízeno podle toho, zda je přístup autentizován, nebo ne. Návštěvník nemá založen účet a jeho možnosti v užívání blogu jsou omezené. Naproti tomu je zde registrovaný uživatel, který má ve svých rukách téměř kompletní funkčnost aplikace (speciálním účtem je administrátor). Následující diagram případu užití (obrázek 2.1) ukazuje hlavní aktéry v aplikaci a jejich možnosti.



Obrázek 2.1: Případy užití tvořené aplikace.

Z role návštěvníka vyplývají určité specifické kroky, jež se netýkají uživatele - registrace a přihlášení. Dále jsou některé možnosti společné - prohlížení a vyhledávání příspěvků. V případě přihlášeného uživatele je zde největší volnost pohybu v systému. Takový účastník má možnost přidávat a mazat vlastní příspěvky, vkládat komentáře k příspěvkům ostatních uživatelů, ale ty už se nedají upravit. Další úkony se týkají správy kategorii, do kterých se budou dělit příspěvky v systému. Tato oblast je podobná příspěvkům samotným, uživatel je může vytvářet, upravovat název anebo celé kategorie mazat. O veškerou koordinaci akcí

se stará již zmíněná autentizace, jež po přihlášení změní práva návštěvníka na uživatelská.

2.5.1 Detail případu užití

Na základě diagramu případu užití je zde jeden konkrétní detail registrace návštěvníka (Tabulka 2.1 na straně 16).

Detail případu užití: registrace návštěvníka
Účastníci: nepřihlášený návštěvník
Vstupní podmínky: návštěvník přistoupí na stránky, není přihlášen
Tok událostí: 1. návštěvník klikne na odkaz registrace 2. návštěvník vyplní přihlašovací jméno, heslo a oveření hesla 3. pokud se zadaná hesla shodují a přihlašovací jméno není registrováno, tak 3.1. systém vloží hodnoty do databáze 3.2. proběhne automatické přihlašení do systému 3.3. uživatel je přesměrován na svůj profil 4. jinak 4.1. je zobrazena stavová chyba
Výstupní podmínky:
návštěvník se stává přihlášeným uživatelem

Tabulka 2.1: Detail případu užití: registrace návštěvníka

2.6 Schéma databáze

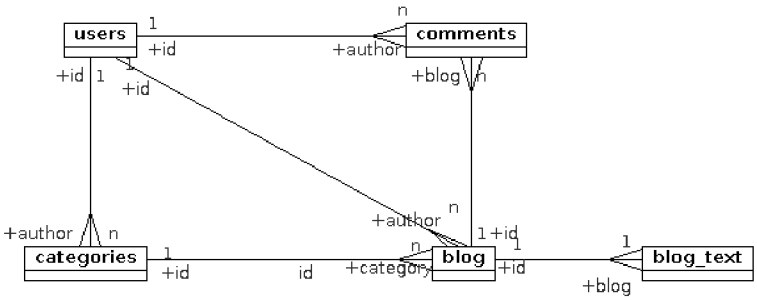
Na základě specifikace požadavků je nyní možné sestavit odpovídající schéma databáze (DB). Jak již bylo zmíněno, v systému se budou pohybovat uživatelé, kteří budou vkládat příspěvky, komentáře a vytvářet kategorie záznamů. Celý systém aplikace bude poměrně jednoduchý, takže i databáze by měla mít pouze několik základních tabulek.

Pro uchování dat uživatelů bude sloužit tabulka `users`, která bude použita pro načítání základních dat pro přihlášení návštěvníka. Sloupce v tabulce musí obsahovat přístupová data do systému - login a heslo (pro zakódování hesla se použije algoritmus MD5), dále email pro kontaktování vlastníka účtu, jeho přezdívku (po vstupu bude uživatel vystupovat pod přezdívkou, která se bude lišit od přihlašovacích údajů) a hodnotu posledního data přihlášení pro monitorování aktivity.

Tabulka s názvem `categories` bude obsahovat názvy jednotlivých kategorií, ID vlastníka a datum vytvoření - pouze informační hodnota. Každý záznam v této tabulce bude jednoznačně vázán na uživatele a nikdo jiný k němu nebude mít přístup. Z tohoto důvodu zde postačuje vazba 1:n (autor:kategorie) a není potřeba vytvářet vazební tabulku. Na ID kategorie odkazuje sloupec z tabulky `blog`, dále tato tabulka pojímá údaje o autorovi a nadpis samotného příspěvku. Datum a čas přidání příspěvku je samozřejmostí. Malinko speciální tabulka `blog_text` bude existovat pouze pro vyhledávání v obsahu příspěvků - pomocí full-textu. Obsahovat bude pouze text příspěvku a cizí klíč do rodičovské entity, což na jednu

stranu drobně zesložiťuje schéma, ale je to optimální volba s přihlédnutím na použité softwarové vybavení.

Poslední součástí aplikace, a tím pádem i databází, jsou komentáře v tabulce `comments`. Obsahem sloupců tabulky jsou ID autora, ID záznamu blogu, text komentáře a datum vložení. Všechny prvky databáze jsou známy a výsledkem je následující ER diagram (obrázek 2.2).



Obrázek 2.2: ER diagram (Entity Relationship Diagram).

Na závěr malé ujasnění situace. Protože vybrané uložiště dat neumožňuje současně používat FULLTEXT a cizí klíče do tabulek, musí se pro vyhledávání zavést speciální tabulka. Tato bude obsahovat text pro vyhledávání výrazů, ale nebude provádět kontrolu platnosti reference na ID záznamu do jiné tabulky. Tento problém bude potřeba řešit aplikačně nebo na úrovni DB pomocí procedur a triggerů.

2.7 Podrobný návrh

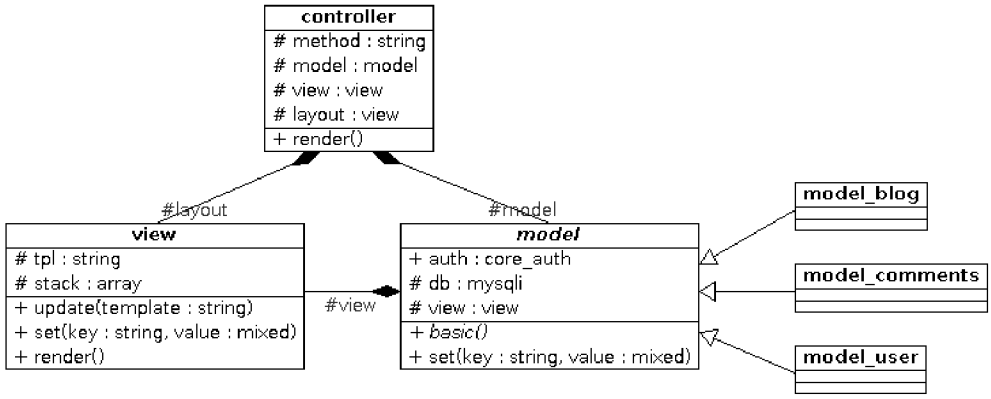
Podrobný návrh se bude odvíjet od každého vzoru zvlášť. Určité základní součásti budou společné pro všechny, např. globální nastavení (databáze) a logování chyb. Hlavním předmětem práce není grafická úprava aplikace. Vzhled bude jednoduchý se zaměřením na funkčnost, ale konzistentní ve všech realizovaných vzorech.

2.7.1 Model-View-Controller (MVC)

Nejprve je potřeba vytvořit základ systému, ze kterého se budou dědit další konkrétní součásti a funkčnost. Jedná se o základní obalující elementy součástí na nižších úrovních (např. databáze). Cyklus začíná zpracováním požadavku od uživatele - ten přijme Controller. Pokračuje přes vytvoření objektů, na základě přijatých dat, pro pohled (View) a aplikační logiku (Model), které jsou propojeny referencí (vlastnost modelu ukazuje na objekt pohledu), viz obrázek 2.3. Společně s identifikačním názvem modelu může být přijat název metody z modelu, ta obsahuje veškeré zpracování dat. Jako poslední akce se provede vytvoření objektu pohledu celé stránky - společného HTML (hlavička, menu, patička). Poslední prováděná akce je vygenerování výstupu pro uživatele - tím cyklus požadavku a odpovědi končí.

Prezentační část vzoru přistupuje k publikování dat přes šablony. Bylo by možné pro každou stránku vytvořit vlastní třídu pohledu, ale pomocí šablon je tento přístup zjednodušen za použití jedné společné třídy. Pohled si na základě nastavených hodnot načte požadovanou šablonu a s tou transparentně pracuje.

2.7.2 Návrh tříd



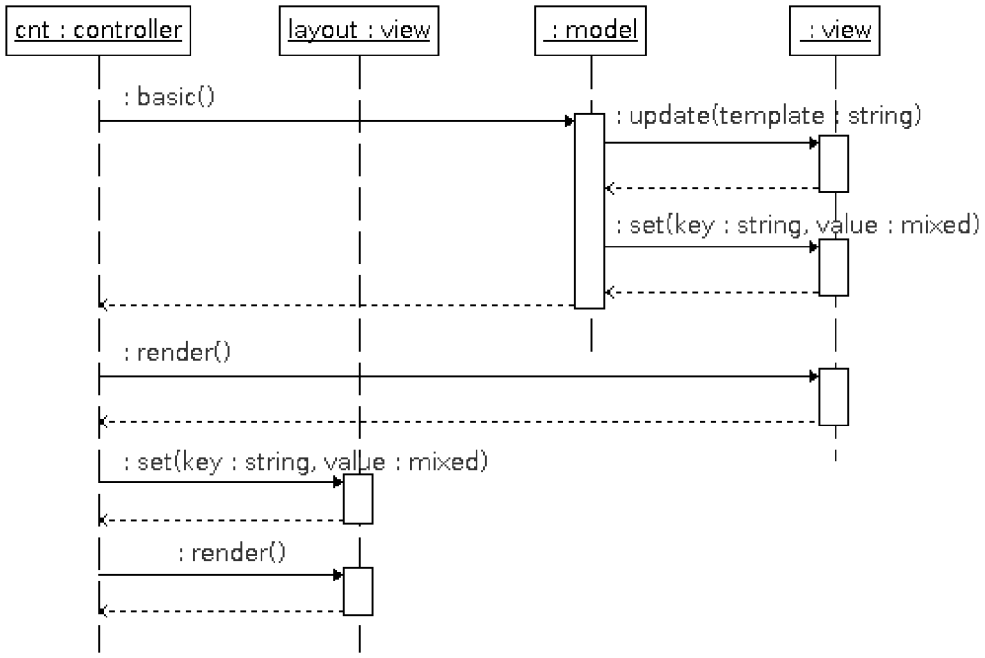
Obrázek 2.3: Schéma tříd aplikace za použití vzoru MVC.

Celá tato podkapitola popisuje programování aplikace na základě obrázku 2.3. Vztahy a interakce tříd reflektují již zmiňované schéma MVC na obrázku 1.2 v kapitole 1.2.4. Hlavním stavebním kamenem a jádrem vzoru je `controller`, ten se jako první stará o požadavek od klienta, provádí inicializaci dalších součástí a ve finále volá požadavky na vygenerování celého HTML. Chráněná vlastnost `$layout` je referencí na objekt pohledu (`view`), které se stará o výpis společných částí HTML na každé stránce - hlavní šablona. Další reference ukazuje na hlavní instanci třídy model - vlastnost `$model`. Trochu speciální přístup zavádí správa přihlášení uživatele, která je vytvářena při zavádění modelu, ale její dostupnost z prostředí controlleru je nezbytná pro výpis menu v hlavní šabloně.

`View`, neboli pohled, není vůbec přístupná z globálního prostředí aplikace. Na jeho instanci je odkazováno jak z modelu, tak controlleru. Controller vytvoří chráněný objekt a použije ho až při výsledném generování HTML obsahu. Tyto dvě komponenty spolu jinak vůbec nekomunikují. View obsahuje zásobník hodnot (`$stack`) zadaných přes metodu `set()`, jedná se o asociativní pole, kde klíč je název proměnné a obsahem je hodnota. Metoda aplikace proměnných pro získání obsahu šablony je `render()`. Šablona se v rámci objektu udržuje v proměnné `$tpl`. Její hodnotu lze měnit i při průchodu Modelem na základě aktuálních hodnot, změna se provede metodou `update()`. V šabloně se přistupuje k proměnné běžným způsobem.

`Model` - hlavní aplikační logika. Obsahuje přístup do databáze, zpracování načtených dat a následný zápis hodnot pro použití v šabloně. Jako parametr při vytváření objektu přijímá referenci na pohled - s tímto pohledem nadále pracuje. Základní metodou, kterou musí obsahovat každý `model`, je `basic()`. Ta je volána v případě, že nebyla explicitně jiná zadána v požadavku. Názvy metod pro volání z prostředí controlleru se přímo rovnají textové hodnotě požadavku. Tyto metody nevrací žádnou konkrétní hodnotu, ale pracují přímo s hodnotami pohledu a nastavují proměnné pro použití v prostředí šablony. Další klíčovou vlastností modelu je autorizace uživatele - `$auth`. Stav přihlášení aktivně ovlivňuje vzhled a dostupnost určitých stránek. Pomocí metody `set()` pracuje transparentně s přiřazeným pohledem. Tato třída je abstraktní a je nutné ji zdědit pro vztvoření konktrétního objektu.

Základní schéma sekvence volání jednotlivých metod objektů znázorňuje následující diagram (obrázek 2.4 - nejedná se o kompletní diagram, ale jen o hlavní metody).



Obrázek 2.4: Sekvence volání metod vzoru MVC.

2.8 Layers - vrstvy

2.8.1 Podrobný návrh

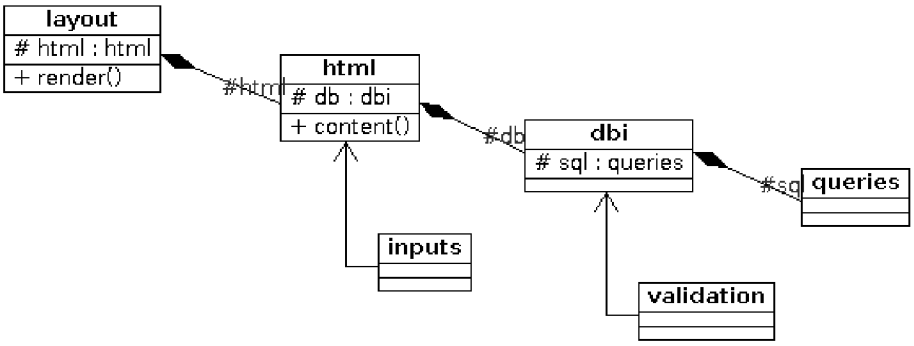
U tohoto vzoru je velice důležitá etapa návrhu. Provázanost jednotlivých vrstev a jejich komunikace jsou klíčovou vlastností a pokud nebude vše správné pracovat, celý systém tím může hodně utrpět. Jednotlivé vrstvy, od datové vrstvy přes střední aplikační vrstvy až k prezentaci dat, je potřeba správně rozvrhnout. Především se jedná o navržení granularity a rozsahu každé součásti. Každá vrstva se stará o svoji oblast a je volána, skrze metody, vrstvou nadřazenou. Tímto způsobem probublávají data od nejnižších vrstev až po výstup do HTML.

V určitých případech nastává odchylka od zavedené konvence volání tříd. Na jisté úrovni je možné mít více tříd, které se volají současně. Tento přístup je východiskem funkčnosti určitých vrstev - jsou „jednosměrné“. Nevyžadují data od jiných vrstev, jsou pouze obalujícím prvkem. Jedná se například o kontrolu vstupních dat (validaci) nebo generování částí HTML kódu.

2.8.2 Návrh tříd

Třídy vzoru Layers jsou navrhnuty s ohledem na funkční zaměření a vhodnou volbu rozsahu. Vrchní vrstva (`layout`) implementuje ošetření vstupních dat a jejich zpracování - na jejich základě jsou použity další vrstvy. Zároveň se jedná o vrstvu generující konečný HTML kód po vykonání nižších vrstev, výsledné HTML je odesláno klientovi jako odpověď serveru.

Následující vrstva `html` už obstarává jednotlivé části aplikace (příspěvky, komentáře, správu skupin). Ve své podstatě se jedná opět o formu výpisu HTML, ale protože jde již o jiné zaměření, byla vytvořena nová vrstva. Interně používá instanci třídy `inputs` pro generování HTML kódu základních formulářových prvků. Tato vrstva je poslední ze skupiny



Obrázek 2.5: Schéma tříd aplikace za použití vzoru Layers (vrstvy).

obstarávající výpis HTML, další vrstvy již pracují s daty v souvislosti s databází (čtení a zápis).

První člen pracující s daty je vrstva dbi. Tato se stará o validaci předaných dat a načítání nebo ukládání dat do DB. Na vstupních datech provede kontrolu formátu - pokud to systém vyžaduje. K validaci slouží třída mimo lineární propojení ostatních vrstev - `validation`. Její funkce je kontrolovat vstupní data před zápisem do databáze, pokud nejsou data validní, je vrácena chyba.

Poslední vrstvou v řadě je samostatný zásobník SQL dotazů pro databázi. Tato třída striktně odděluje text SQL dotazů. Všechny jsou na jednom místě, což zajišťuje přehlednost v případě úpravy.

2.9 Pipes and Filter - roury

2.9.1 Podrobný návrh

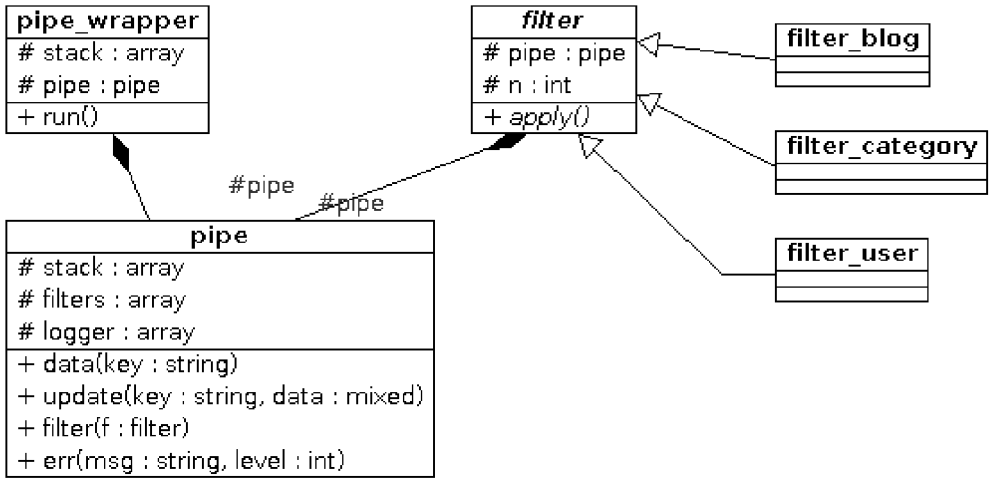
Jádro tohoto vzoru obstarává základní třída pro předávání objektu `pipe` mezi jednotlivými filtry. Tento obal (`pipe_wrapper`) přijme při vytváření objektu pole názvů filtrů, které budou aplikovány na obsah objektu pipe. Při sekvenčním průchodu polem se vezme daný filtr, předá se mu objekt pipe a nechá se provést úprava konkrétních hodnot.

Každý filtr si může zjistit, zdali se má aplikovat v daný okamžik. Objekt `pipe` obsahuje všechna data a `filter` k nim může libovolně přistoupit, zjistit hodnotu podle klíče a zachovat se odlišně pro každou vrácenou hodnotu. Upravená data nakonec uloží zpět do objektu pipe, který je poslán dalšímu filtru a celý proces se opakuje. Jak již bylo zmíněno, rozhraní mezi jednotlivými elementy sekvence musí být pevně definované. Z tohoto důvodu je třída `filter` abstraktní a další konkrétní třídy tento základ rozšiřují na požadované chování.

Samotný předávaný objekt, `pipe`, implementuje funkčnost pro přístup k interním hodnotám a uložení nové hodnoty.

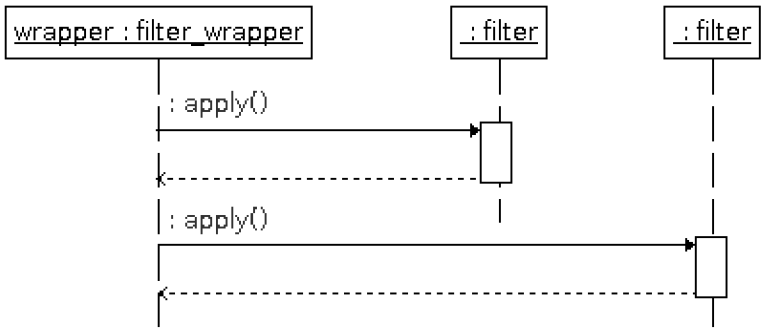
2.9.2 Návrh tříd

Následující text popisuje obrázek 2.6, vztahující se k návrhu tříd vzoru Pipes and Filters. Jak bylo již zmíněno, hlavní funkčnost obstarává obalující třída `pipe_wrapper`. Třída má chráněné vlastnosti `$stack` a `$pipe`. První zmíněná obsahuje pole filtrů, které se budou aplikovat, druhá vlastnost je referencí na objekt předávaný mezi filtry.



Obrázek 2.6: Diagram tříd vzoru Pipes and Filters.

Třída `pipe` má vlastnosti `$stack`, `$filters` a `$logger`. První zmiňovaná vlastnost `$stack` je pole obsahující veškerá data ze superglobální proměnné `$_REQUEST`. S těmito daty se pracuje celou dobu existence instance třídy. Další proměnnou v rámci třídy je `$filters` - pole obsahující počet aplikací každého filtru. Poslední položkou je `$logger`, který obsahuje vzniklé chyby za běhu filtru. Metody `data()` a `update()` se uplatňují ke čtení hodnoty, resp. k zápisu nové hodnoty. `Filter()` inkrementuje počet aplikací daného filtru.



Obrázek 2.7: Sekvenční diagram vzoru Pipes and Filters.

Poslední základní třída diagramu vzoru, `filter`, je abstraktní a obsahuje společný základ pro odvozené třídy. Vlastnost `$pipe` je referencí na instanci putující třídy. Přes tuto vlastnost, resp. metody této instance, se spravuje veškerý obsah. Hlavní abstraktní metoda `apply()` je volána v každém kroku sekvence z objektu `pipe_wrapper`. Každá další odvozená třída musí předefinovat tuto metodu s vlastní požadovanou funkčností.

Konečná posloupnost volaných metod vzoru Pipes and Filters je znázorněna sekvenčním diagramem v UML (obrázek 2.7).

2.10 Adresářová struktura

Součástí návrhu je také struktura adresářů a souborů. Přestože bude aplikace tvořena třemi způsoby, je nutné mít na paměti možnost sdílení určitých částí systému (tříd). Každý vzor bude mít vlastní složku a sdílené součásti bude načítat z adresáře o úroveň výš. Plná odrážka (•) značí adresář a prázdná (◦) je soubor.

- client - soubory načítané klientem (css, sdílené)
- img - obrazky grafiky (sdílené)
- layers - vzor Layers
  - include - všechny třídy
  - ◦ config.layers.php - konkrétní nastavení vzoru
  - ◦ index.php
- mvc - vzor Model-View-Controller
  - controllers - třídy pro ovládání aplikace
  - models - soubory s funkcemi
  - tpl - HTML šablony stránek
  - view - třídy pro obsluhu šablon
  - ◦ config.mvc.php - konkrétní nastavení vzoru
  - ◦ index.php
- pipes - vzor Pipes and Filters
  - filters - adresář s filtry
  - include - ostatní třídy (pipe)
  - ◦ config.pipes.php - konkrétní nastavení vzoru
  - ◦ index.php
- share - třídy společné pro všechny vzory

◦ config.php - sdílené nastavení databáze

◦ index.html - rozcestník vzorů

Kapitola 3

Implementace

Základní stavební jednotkou celého systému je samozřejmě objekt, ale v podstatě se bude podrobnější návrh odvíjet od každého návrhového vzoru. Podrobný rozbor implementace je detailně popsán v následující kapitole o použití vybraných vzorů. Testování je prováděno na vhodně zvoleném vzorku dat - vhodné pro vstup i výstup. Otestovat se musí všechny navržené části aplikace, od autorizace přes formuláře, až k samostatným výpisům dat - HTML. U vstupu a výstupu dat by měl být kladen důraz na potencionálně nebezpečné znaky jako jsou: „, \ a HTML tagy. Ověření funkčnosti jako takové by mělo být zaručeno správnou implementací podle návrhu.

Součástí implementace není administrace uživatelů a příspěvků v systému, tato správa se řeší pomocí externí aplikace. Databáze je navržena tak, aby i přímý zásah zachoval všechna data v konzistenci.

Speciální částí implementace jsou pravidla pro psaní zdrojového kódu. Identifikátory proměnných, funkcí (metod tříd) i tříd budou tvořeny malými písmeny a podtržítky (pro oddělení slov), popřípadě pořadovými čísly. Z jazykového hlediska budou používány názvy anglické. Název třídy bude například `class_name`. V žádném případě nejsou přípustné česko-anglické názvy identifikátorů - např. `$password_uzivatele`. Stejné konvence platí i v prostředí databáze.

3.1 Implementace

3.1.1 Model-View-Controller (MVC)

Jelikož se jedná o webovou aplikaci, požadavek začíná souborem index.php (v tomto případě). Do souboru se načte nastavení vzoru (včetně hodnot spojení do databáze a základních tříd aplikace). Vytvoří se instance třídy `controller` (hlavního ovládacího prvku) a zavolá se metoda `render()` pro vyvolání dalších akcí aplikace. Celý obsah souboru jsou následující řádky.

```
// základní nastaveni vzoru a databáze
require_once('./config.mvc.php');

$cnt = new controller();
$cnt->render();
```

Veškerá další funkcionalita je prováděna již pouze uvnitř objektů. V konstruktoru objektu se vytvoří instance konkrétní třídy `model` (blog, kategorie, uživatel) - podle vstupních

dat. Objekty ze třídy `view` jsou vytvořeny ve dvou instancích. První se předá modelu pro zpracování konkrétní šablony a druhá je vázána na controller v podobě společné šablony každé stránky.

Metoda `render()` volá akci modelu, které pracuje s daty aplikace a nastavuje proměnné do objektu `view` pomocí metody `set()`. V dalším kroku aplikuje na šablonu nastavené hodnoty pohledu, který je svázán s modelem a výsledný obsah šablony je uložen do proměnné layoutu celé stránky - `$content_for_layout`. Poslední akce vytiskne celý obsah stránky a tím ho pošle jako odpověď na požadavek od klienta.

```
// metoda třídy controller
public function render() {
  if (is_callable(array($this->model, $this->method))) {
    call_user_func(array($this->model, $this->method));
  }

  if ($this->view instanceof view) {
    $this->layout->set('content_for_layout', $this->view->render());
  }

  echo $this->layout->render();
}
```

Zajímavou metodou je samotné aplikování hodnot pohledu na šablonu, která je volána z prostředí `controlleru`. PHP funkce `extract()` vytvoří lokální proměnné z klíčů předaného pole s jejich hodnotami. Následně se vloží soubor šablony, ale díky funkci `ob_start()` není obsah vypsán, ale odchycen a jako řetězec je návratovou hodnotou metody.

```
// metoda třídy view
public function render() {
  extract($this->stack);

  ob_start();
  include($this->tpl_file());

  return ob_get_clean();
}
```

3.1.2 Layers - vrstvy

Na základě návrhu tříd je průběh implementace vzoru Layers poměrně jednoduchý. Rozdělení na části udržuje kód celé aplikace přehledný a možné úpravy se provádějí velice snadno. Nejvyšší třída `layout` se stará o celkový vzhled HTML stránky. Metoda `render()` objektu typu `layout` vypíše pomocí vnořených vrstev obsah stránky. Při konstrukci instance třídy se zároveň vytvoří instance vnořené vrstvy, to zaručuje dostupnost nižší vrstvy v požadovaný okamžik.

Druhá vrstva (konkrétní logika aplikace) vypisuje data získaná z datového zdroje a upravená pro výpis. Tuto funkčnost konkrétně zajišťují třídy `html_blog`, `html_categories` a `html_user`. Opět se uplatňuje pravidlo konstrukce objektů. Tentokrát se jedná vrstvu pracující s datovým uložištěm - čtení nebo zápis dat.

Třída `dbi` získává z poslední vrstvy čisté SQL dotazy, které odesílá na databázi. Na této předposlední úrovni se provádí kontrola vstupních dat pro další zpracování. V rámci SQL dotazů, které se získávají z poslední vrstvy, se provádí jak načítání dat z databáze, tak jejich zápis. Následující útržek kódu aplikace ukazuje zpracování při požadavku registrace nebo změně hodnot účtu (kód je vytržen z kontextu a je drobně upraven).

```
  } elseif (!validation::email($arr['email'])) {
    #log

  } elseif ($this->auth) {
    if (empty($arr['pwd'])) unset($arr['pwd']);
    $this->db->user_update($this->auth->get_user(), $arr);

  } elseif (!empty($_POST['login']) && !empty($arr['pwd'])) {
    $arr['login'] = $_POST['login'];
    $this->db->insert('users', $arr);
  }
```

Výsledná data nakonec probublají zpět z poslední datové vrstvy na začátek, kde jsou prezentována jako odpověď serveru.

3.1.3 Pipes and Filters - roury

Celý průběh aplikace opět obstarává soubor index.php v adresáři vzoru. Konstruktor obalujícího objektu přijímá jako první parametr pole názvů filtrů, ty budou následně aplikovány na objekt `pipe`. Vytvoření objektu třídy `pipe_wrapper` nemá žádný vliv na data a jejich hodnoty. Při vytváření instance třídy se vytvoří také instance třídy `pipe` a reference se přiřadí do vlastnosti `$pipe` třídy `pipe_wrapper`.

```
// základní konfigurace vzoru a databáze
require_once('./config.pipes.php');

$wrapper = new pipe_wrapper(array('db', 'auth', 'blog', 'categories',
  'user', 'layout'));
$wrapper->run();
```

Sekvence filtrů se spustí explicitním voláním metody `run()`. Uvnitř metody je cyklus procházející zásobník názvů filtrů. V každém cyklu je vytvořen objekt pro daný filter a zavolána metoda `apply()` pro úpravu dat postupujících přes filtry. Z pohledu násobného použití jednoho filtru může být zbytečné vytváření vždy nového objektu, ale tato drobná vada není v řešené situaci důležitá. Režie pro vytvoření nového objektu je s malou vytížeností systému zanedbatelná.

```
public function run() {
  foreach ($this->stack as $cls) {
    $cls = 'filter_' . $cls;
    $filter = new $cls($this->pipe);

    $filter->apply();
  }
}
```

Kompletní konkrétní úprava dat probíhá v metodě `apply()`, objekt má přístup k potřebným datům a podle hodnoty vybraných dat může měnit svoji funkcionalitu. Tato metoda zároveň ošetřuje situaci, kdy je filtr pasivní, tzn. když dostane hodnotu, na kterou není přizpůsoben, ukončí metodu příkazem *return*.

Někdo by mohl namítat, že se nová hodnota nikde neuloží do promenné `$this->pipe`, ale to není pravda. Obecná práce s objekty spočívá v práci s referencí (odkazem) na objekt, to je případ i PHP. Objekt existuje v jednom místě paměti a všechny proměnné s hodnotou tohoto objektu ukazují do paměti na jedno místo (netýká se nově vytvořených objektů). Proto vše funguje jak má - `filter` dostane jako parametr odkaz od paměti a s tou pracuje.

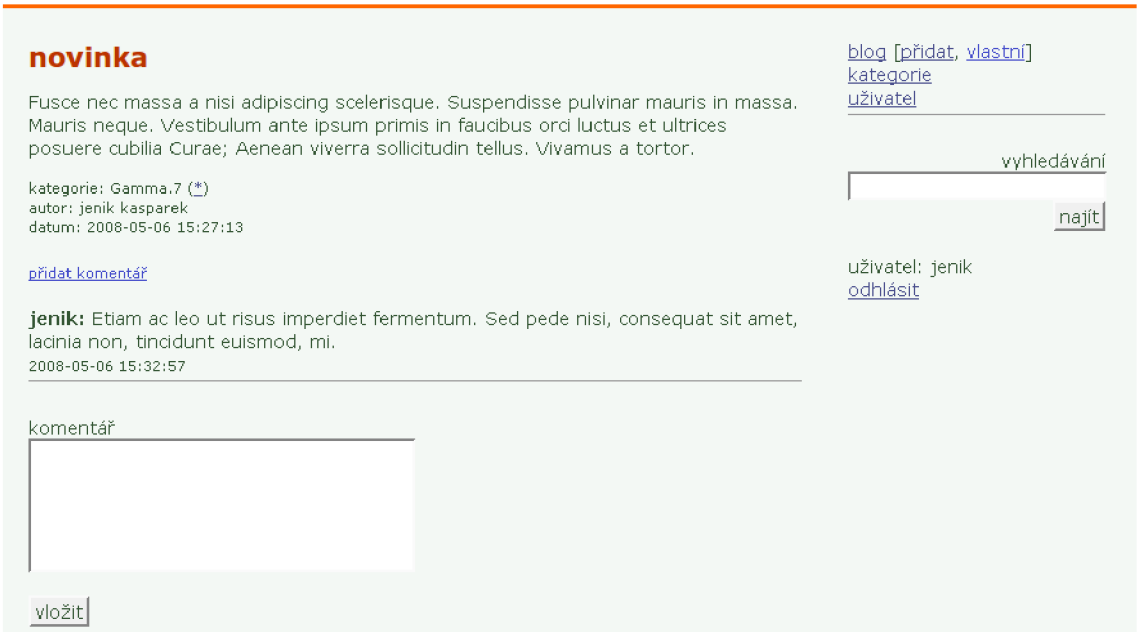
3.2 Vzhled a popis aplikace

Následující obrázky 3.1, 3.2, 3.3 a 3.4 ukazují vzhled jednotlivých klíčových součástí aplikace. Jak již bylo zmíněno, vzhled aplikace není hlavním zadání práce, a proto byla volena jednoduchá grafická úprava. Tato součást aplikace byla řešena pomocí CSS - Cascading Style Sheets.

Obrázek 3.1: Úvodní stránka aplikace s výpisem příspěvků.

Obrázek 3.1 prezentuje vzhled úvodní stránky pro návštěvníka - není přihlášen. Oblast stránky je dělena na tři základní oblasti - nadpis, sloupec menu (pravý) a obsah (levá část). V horní části je název systému (může být nahrazeno grafickým logem), dále je v pravé části aktuální datum a posledním údajem je vzor použitý k implementaci (Model-View-Controller, Layers, Pipes and Filters). Ve sloupci menu je také formulář pro vyhledávání a pro přihlášení do systému. Hlavní obsahová část prezentuje seznam jednotlivých příspěvků v systému s odkazem na detail. Hlavní část zobrazuje také ostatní části aplikace - správu kategorií a formulář pro úpravu účtu.

Výpis detailu příspěvku s přidanými komentáři je na obrázku 3.2. Podrobný výpis zobrazuje také kategorii, autora a datum prispěvku. Po přihlášení je k dispozici formulář



Obrázek 3.2: Detail příspěvku s výpisem komentářů.

pro vložení nového komentáře k příspěvku. Změnilo se menu, které je nyní vypsáno v režimu pro přihlášeného uživatele.

Správu kategorií obstarává jedna stránka se všemi požadovanými prvky (obrázek 3.3). Zobrazuje názvy všech kategorií přihlášeného uživatele a dovoluje jejich odebrání, změnu názvu nebo vložení nové.

Poslední klíčovou vlastností a jejího výpisu je správa samotného účtu přihlášeného uživatele (obrázek 3.4).

3.3 Zdrojové kódy a instalace

V době publikace této technické zprávy ještě nebyla implementována kompletní funkčnost aplikace. Základní informace o aplikaci, rozcestník a aktualizované verze aplikace jsou dostupné na následujících WWW adresách.

Celý zdrojový kód aplikace je možné stáhnout z internetu, viz následující body:

- **rozcestník** - http://jenik.cz/fit/
- **aplikace (zdrojový kód)** - http://jenik.cz/fit/blog.tar.gz
- **schéma tabulek databáze** - http://jenik.cz/fit/schema.sql

Obrázek 3.3: Správa kategorií uživatele (vložení, úprava a mazání).

Obrázek 3.4: Formulář pro správu účtu uživatele a registraci.

Kapitola 4

Testování

V rámci zadání bylo provedeno testování aplikace z hlediska správné implementace návrhu. Důkladné testování proběhlo na zadávání “special characters„ a “html entities„, což jsou znaky se speciálním významem ve značkovacím jazyku HTML. Tyto znaky představují potenciální bezpečnostní riziko při výpisu ve stránce. Hlavní nebezpečné znaky jsou ostré závorky (< a >), uvozovky (“ a ’) nebo zpětné lomítko.

Další část testování byla zaměřena na zátěž celého systému při současných požadavcích uživatelů. Ve specifikaci je předpokládaná hodnota stanovena na 15 současných požadavků, což není mnoho. Takový test absolvoval systém aplikace naprosto bez problémů.

Kapitola 5

Zhodnocení výsledků

V prostředí webu je typický cyklus běhu aplikace: požadavek - zpracování - odpověď. S tímto základem je nutné počítat při návrhu a realizaci aplikace. Nejlepe se s touto skutečností vypořádá vzor MVC - trojúhelníkový tvar, neprochází všemi vrstvami „ve dvou směrech“ jako u Layers. Pipes and Filters mají plně lineární průběh, data jsou předávána z jednoho filtru na další, v základním schématu nejsou žádné odbočky.

Oddělení jednotlivých vrstev aplikace (řídící, logická, datová a prezentační) je důležitým kritériem přehlednosti implementace. Všechny vzory toto podporují, s tím že Layers a Pipes nejsou omezeny na společnou loginku, ale každá vrstva (filtr) obstarává specifickou funkčnost.

Část aplikace obstarávající prezentaci dat může být řešena libovolně. V prostředí webu je tato část zvláště důležitá, protože se jedná o odpověď na požadavek klienta a reprezentaci dat uložených na serveru. Ostatně, tato vrstva může být řešena i jako externí součást aplikace - jak již bylo zmíněno (viz kapitola 1.3.2 na straně 12).

Teoretické nastínění posloužilo k obecnému přehledu vlastností a schopností každého vybraného vzoru a následná implementace aplikace za použití vzoru prověřila jeho použitelnost v praxi. Rozsahem se jedná o malou aplikaci, ale i tak obsahuje základní prvky všech informačních systémů - formuláře, jejich zpracování, vložení do databáze a následné výpisy dat.

Pro návrh bylo použito modelování systému pomocí UML. Všechny vzory byly velice dobře aplikovatelné při návrhu i implementaci zvolené aplikace. Dobrý a kompletní návrh je základem kvalitní aplikace.

Použití PHP s šablonami u vzoru MVC, přineslo velmi jednoduchou a přehlednou prezentaci aplikačních dat. Stejně tak je příjemné oddělení řídící logiky Controlleru od ostatních částí aplikace. Při požadavku na nezávislost datové vrstvy je nutné rozdělit Model na více částí - aplikační logiku a ovládání databáze.

Složitější komunikace se projevila u vzoru Layers, kdy se poustupně systém zanořuje a volá metody nižších vrstev. Taková funkčnost se může projevit ve zhoršené orientaci v kódu aplikace (jak při implementaci, tak při následné správě). Naopak přínosem tohoto vzoru je volnost při návrhu součástí, která je vyvážena komplikovanějším návrhem.

Pipes and Filters se vyznačují až extrémní flexibilitou chování, které lez dosáhnout výměnou jednotlivých filtrů. Nutností při použití vzoru je vhodná volba struktury předávaných dat, ale objektový přístup se může uplatnit téměř vždy.

Testování aplikace ověřilo správnou implementaci návrhu a otestovalo ošetření vstupů a výstupů.

Kapitola 6

Závěr

Osobně si myslím, že nic není černobílé a kombinování různých klíčových vlastností jiných architektonických vzorů může být přínosné. „V jednoduchosti je síla“ - tak se dá označit vzor MVC, je rozdělen do tří základních prvků, které obstarávají chod celého systému.

Osobně věřím ve velký potenciál vzoru rour (pipes), ten je dlouhodobě používán v praxi v prostředí operačních systémů a nikdo zatím nepřišel s jiným řešením (nebo se aspoň neuchytilo mezi lidmi). V informačních systémech pro web může být tento vzor použit k dosažení ještě větší nezávislosti a flexibility, jak jednotlivých součástí, tak celého systému.

Naopak mne moc neoslovil vzor vrstev. Jeho linearita občas koliduje s přehledností celého systému. Na druhou stranu bych viděl přínos v možnosti použití externích prvků jako jednotlivých vrstev, např. prezentace dat, ale i specializovaných výpočetních jednotek na středních vrstvách.

Literatura

[1] Frequently Asked Questions | GoPHP5.org. citováno 6.3.2008 17:15.
URL http://gophp5.org/faq

[2] Jim ARLOW, Ila NEUSTADT: *UML a unifikovaný proces vývoje aplikací*. Brno: Computer Press, 2002, ISBN 80-7226-947-X, 387 s.

[3] Model-view-controller. [online], citováno 7.4.2007.
URL http://en.wikipedia.org/wiki/Model-view-controller

[4] Návrhový vzor. [online], citováno 4.1.2008, 22:36.
URL http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor

[5] PAVLICEK, L.: Historie návrhových vzorů. [online], citováno 16.6.2005, 19:04.
URL http://objekty.vse.cz/Objekty/Vzory-historie

[6] Petr MATOUŠEK: Síťové aplikace a správa sítí, Úvod. Přehled TCP/IP, adresování a konfigurace. 2007.

[7] Pipeline (software). citováno 8.4.2008, 08:45.
URL http://en.wikipedia.org/wiki/Pipeline_(software)

[8] Pipe-And-Filter. změněno 4.8.2007, 22:57.
URL http://www.dossier-andreas.net/software_architecture/p%ipe_and_filter.html

[9] SMOLÍK, T.: Softwarová architektura: Nezúžený „klasický“ úvod. [online], 2007.
URL http://academy.profinit.eu/courses/SWI026/lectureNotes/ProfinitSwArchitectureOverview.pdf

[10] Unified Modeling Language. [online], citováno 21.4.2008, 15:31.
URL http://en.wikipedia.org/wiki/Unified_Modeling_Language

[11] Vladimír SKLENÁŘ, Miloš KUDĚLKA: Návrhové vzory a jejich aplikování. [online], 2003.
URL http://objekty.pef.czu.cz/2003/sbornik/SklenarKudelka2%003.pdf

[12] Windows PowerShell.
URL http://en.wikipedia.org/wiki/Windows_PowerShell#PowerShell_1.0

Seznam příloh

Příloha 1. CD se zdrojovým kódem aplikací, schématem databáze a zdrojovým textem písemné zprávy