VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

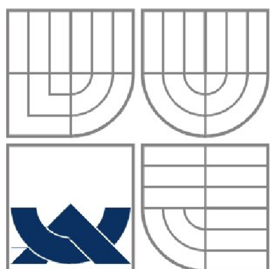
# MODERNÍ TECHNOLOGIE PRO VÝVOJ WEBOVÝCH APLIKACÍ A JEJICH VÝKON

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

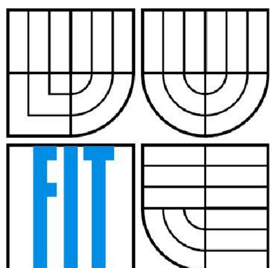
AUTOR PRÁCE
AUTHOR

BC. ZDENĚK SMIŠTÍK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# MODERNÍ TECHNOLOGIE PRO VÝVOJ WEBOVÝCH APLIKACÍ A JEJICH VÝKON

MODERN TECHNOLOGIES FOR WEB APPLICATIONS DEVELOPEMENT AND THEIR PERFORMANCE

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. ZDENĚK SMIŠTÍK
AUTHOR

VEDOUCÍ PRÁCE — DOC. DR. ING. PAVEL ZEMČÍK
SUPERVISOR

BRNO 2009

# Moderní technologie pro vývoj webových aplikací a jejich výkon

## Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Doc. Dr. Ing. Pavla Zemčíka

Další informace mi poskytl Ing. Marian Beszédeš, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Zdeněk Smištík
20.květen.2009

## Poděkování

Speciální poděkování za vedení při práci Doc. Dr. Ing. Pavlu Zemčíkovi a za odbornou pomoc a radu Ing. Marianu Beszédešovi, Ph.D.

© Zdeněk Smištík, 2009.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Abstrakt

Práce se zabývá nástroji pro tvorbu webových aplikací, které jsou založeny na návrhovém modelu Model-View-Controller. Mezi tyto nástroje spadá například Zend Framework, Ruby on Rails a Spring Framework. Vysvětluje jejich funkci, vlastnosti, způsob manipulace s daty a jejich reprezentaci uživateli. Práce obsahuje i příklad aplikace.

# Klíčová slova

Framework, Model-view-controller, MVC, Zend, Ruby, Ruby on Rails, Spring

# Abstract

The thesis focuses on such tools for web applications development that are based on the Model-View-Controller design model. These tools include e.g. Zend Framework, Ruby on Rails, and Spring Framework. The thesis explains the functionality of the tools, their features, methods of data manipulation, and presentation of data to the users. The work contains also an application example.

# Keywords

Framework, Model-View-Controller, MVC, Zend, Ruby, Ruby on Rails, Spring

# Citace

Zdeněk Smištík: Moderní technologie pro vývoj webových aplikací a jejich výkon. Brno, 2009, diplomová práce, FIT VUT v Brně.

# Obsah

Obsah....5
1 Úvod....6
2 Framework pro webové aplikace....8
2.1 Zend Framework....10
2.2 Ruby on Rails....20
2.3 Spring Framework....30
3 Způsoby prověřování frameworků....35
3.1 Uživatelská přívětivost frameworku....35
3.2 Výkon webové aplikace....36
3.3 Vzorová aplikace....37
4 Implementace....38
4.1 Implementace pomocí Zend....42
4.2 Implementace pomocí Ruby on Rails....47
4.3 Implementace pomocí Spring....49
5 Průběh a vyhodnocení testování....53
5.1 Měření výkonnosti....53
5.2 Zobrazení výsledků....57
5.3 Pohled uživatele....61
6 Závěr....62
Literatura....63

# 1 Úvod

Osobní počítače se za dobu svého působení stali nedílnou součástí našich životů. Pomáhají nám v naší profesní, ale i osobní sféře života. Jedním z článků mohutného řetězce využití výpočetní techniky je informační systém. Informační systém je soubor informací, který je jistou formou nabízen uživateli. Jednou z forem jak uživatel může přistupovat k informacím, které systém nabízí je přes webové rozhraní pomocí webové aplikace.

Webová aplikace je aplikace, poskytovaná uživatelům z webového serveru přes počítačovou sít - Internet nebo z vnitropodnikové sítě - Intranet, popřípadě z našeho vlastního počítače. Webové aplikace se těší stále větší oblibě a jejich působení v oblasti informační technologie pomalu, ale jistě vytlačují desktopové aplikace z pole jejich působnosti. Výhodou webových aplikací je všudypřítomnost webového prohlížeče coby klienta. Nezáleží na typu operačního systému, ani na jeho verzi. Jediným úskalím, které nám občas brání jeden projekt bezproblémově přenést na tisíce počítačů, je nekonzistentní implementace HTML, CSS, DOM a nastavení webového prohlížeče. Další bezesporu důležitou vlastností je schopnost aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat jakýkoliv software. S webovými aplikacemi se setkáme na místech nám všem známých ze stránek na Internetu, ale jejich užití je stále širší. Setkáme se s nimi například jako s uživatelským rozhraním pro nastavení síťových komponent (servery, směrovače) i pro reklamní, firemní CD s informacemi, nabídkou dané firmy.

Touto prací bych chtěl upozornit na výhody webové aplikace jako takové. Věřím, že její znalost bude přínosem nejen pro mě, ale i pro všechny, kteří se webovou aplikací budou zabývat ať již v osobním životě či na profesní úrovni.

Tvorbou vlastních webové aplikací se uživatelé zabývají bez ohledu na to, zda mají zkušenosti s programováním či jsou úplnými začátečníky. Začátečníci při vytváření webu používají buď přímo kódu HTML případně v kombinaci s jednoduchými skripty v jazyce PHP či Per. Existuje však celá řada systému, kde je tento proces automatizovaný, a tak mohou vývojářům nabídnout popis programu na mnohem vyšší úrovni. Jejich užití zjednodušuje a zpřehledňuje kód, snižuje počet chyb v aplikacích a programátor se tak může soustředit na důležitější části kódu. Užití těchto systémů často zjednoduší a zpřehlední kód, čímž sníží počet chyb v aplikacích a programátor se pak může soustředit na důležitější části kódu. Proto se nadále v textu věnuji těmto systémům, abych laiků i programátorům přiblížil nové přístupy k tvorbě webových aplikací. Vybral jsem systémy, které jsou postaveny na různých programovacích jazycích, aby si každý mohl vybrat podle svého uvážení, který pro něj bude nejvíce vyhovující.

Úvodní část práce je zaměřena na vysvětlení základních pojmů, které jsou v práci použity a jejichž správná prezentace je nezbytná pro pochopení celé práce. Dále následuje teoretická část, věnovaná každému systému jednostlivě. Jako první je uveden Zend Framework. Jedná se o systém postavený na jazyce PHP, za jehož vznikem stojí silná firma Zend Technologies Ltd., která je považovaná za tvůrce samotného PHP. Získáte informace, jak systém Zend pracuje, jaké má vlastnosti, jaké obsahuje balíčky, co je třeba k jeho chodu a nakonec příklad pro objasnění celé logiky. Dalším uvedený systém je Ruby on Rails, což je soubor několika užitečných webových knihoven napsaných v moderním, interpretovaném jazyce Ruby. Informace o Ruby on Rails jsou strukturované obdobným způsobem jako v případě Zend Frameworku, a stejně tak jsem postupoval i u posledního z vybraných, systému Spring Framework. Spring je komplexní aplikační systém postavený na programovacím jazyce Java, to znamená, že pomocí frameworku Spring lze vyvíjet i desktopové aplikace. Tomto dokument se věnuje pouze jedné z jeho částí a to částí zabývající se webovou vrstvou. Po získání základních informací o vybraných frameworcích, následuje kapitola věnovaná způsobům, jak testovat webové aplikace a na jaké aspekty se soustředit při hodnocení jejich užitečnosti. Toto měření je realizováno v navazující kapitole, kde je popsána tvorba vybrané aplikace v každém ze zmíněných frameworků a detailně je zaostřeno na problematická místa vývoje. Poslední kapitola se zabývá shrnutím získaných dat a jejich posouzením, jak si stojí mezi porovnávanými frameworky. Právě tato poslední kapitola ukáže jak silný nástroj se skýtá v našich rukách (hlavě).

# 2 Framework pro webové aplikace

Cílem této práce je seznámit čtenáře se základními pojmy, které se budou nadále v textu často vyskytovat a jejich znalost je pro pochopení celé práce zcela nezbytná.

## Pojem framework

Framework je softwarová struktura, která pomáhá programátorovi při vývoji jiného softwarového produktu. Cílem takovéto struktury je převzetí typických problémů, s kterými se při řešení častěji potýkáme a plně je zautomatizovat. Vývojář se nemusí zdržovat zbytečnou režií kolem známých a už několikrát řešených věcí a může se plně věnovat implementaci svého konkrétního problému. Proti této výhodě stojí tvrzení, že používáním frameworku se stává kód pomalý a celková činnost neefektivní. Protože čas, který získáme použitím cizího kódu, využijeme pro jeho nastudování. Na druhou stranu, pokud budeme s frameworkem pracovat častěji, budeme znát jeho syntaxi i sémantiku, a tak ušetříme čas, který je potřeba na jeho studii, tím zefektivníme svou činnost, nehledě na to, že se stane programování pohodlnější.

Existují dva základní druhy webových frameworků: komponentové a akční. Oba druhy jsou postaveny na odlišných principech, a každý z nich je vhodný pro jiné typy aplikací.

- **Komponentové frameworky:** Komponentové frameworky automatizují spoustu programátorských činností. Komponenty si samy ukládají svůj stav do sezení, samy mění stav na základě parametrů v URL, samy vykreslují svoji podobu do HTLM atd. Nevýhodou je krádež paměti, neboť stav uložený v sezení ji zabírá. Málokterý komponentový framework umožňuje změnit HTML podobu komponenty jinak než jejím přepsáním.
- **Akční frameworky:** Akční frameworky vycházejí z návrhového vzoru Model View Controller. Tím oddělují výkonovou část od grafické podoby. I zde můžeme aplikovat znovupoužitelnost kódu a používat parametrizované uzavřené fragmenty kódu, ale o komponenty jako takové už nejde. Programátor se musí sám postarat o ukládání stavu do sezení a o správná URL, na druhou stranu má plnou kontrolu nad grafickou podobou všech částí.

Pro lepší pochopení rozdílu mezi těmito frameworky se zaměříme na jeden aspekt a to sice jak framework chápe stav aplikace a především kam ho ukládá. Uložení stavu webové aplikace může proběhnout dvěma způsoby:

1. **URL** – stav aplikace pro jednoho uživatele je určen URL, sezení není potřeba
2. **Sezení** – stav aplikace pro jednoho uživatele je v sezení, v URL se přenáší změny stavu

Komponentové frameworky ukládají svůj stav do sezení. To znamená, že jejich URL jsou jednoduchá, ale nevyžadují kompletní stav, což může znepříjemnit práci robota internetového vyhledávače. Naproti tomu akční frameworky ukládají stav do URL, přesněji je ukládá programátor. Výsledná URL jsou oblíbená u vyhledávačů, ale u složitějších stránek se spoustou ovládacích prvků musí být programátor velmi pozorný, jak má URL vlastně uložit.

Z předešlého textu je vidět i obecné použití jednotlivých frameworků. Komponentové frameworky se hodí pro aplikace se složitým ovládáním a pro intranetové aplikace. Akční frameworky se hodí na různé grafické podoby webu a obecně pro internetové aplikace.

# Model View Controller (MVC)

MVC architektura je považována za návrhový vzor, její vlastnosti jsou dobrým podkladem pro vývoj aplikací. Jak už je z názvu patrno, je MVC architektura rozdělena do tří oddělených části, na model (model), view (pohled) a controller (řadič). Toto dělení je srdcem architektury. Deklaruje jasný rozdíl mezi doménovými objekty, které modelují naše vnímání skutečného světa (modelové objekty) a prezentační objekty, což jsou grafické elementy, které my vidíme na obrazovce (pohledové objekty).

Obecně řečeno aplikace, které jsou navrhovány podle MVC architektury, vyžadují implementaci těchto tří komponent:

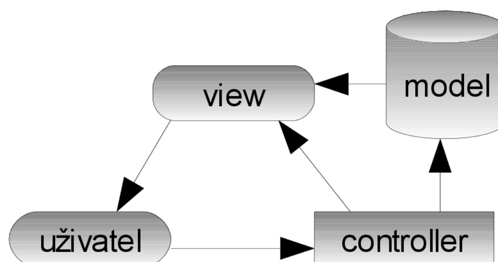
**Model (model)** definuje data a logiku pro zacházení s daty. Jako příklad si můžeme uvést entitu student, která je modelový objekt, má svoje data, jako je jméno a příjmení a svoje metody, jako nastavení a získaní hodnoty. Modelové objekty nejsou přímo zobrazovány, jsou často upravovány kvůli různorodosti platforem.

**View (pohled)** představuje viditelné objekty na obrazovce jako je například panel nebo tlačítko, nebo-li převádí data reprezentované modelem na objekty k prezentaci pro uživatele.

**Controller (řadič)** stoji mezi modelem a pohledem, poslouchá události a reaguje na ně změnou v modelu čí pohledu. Například řadič zprostředkuje jméno studenta z modelu k viditelnému textovému poli v pohledu, kde může dojít k editaci a znovu za pomoci řadiče se jméno přenese do modelu kde dojde k uložení nové informace.

Přestože model MVC může být pojat různými způsoby, vždy musí obsahovat tyto tři komponenty. Nastínění funkce MVC architektury je znázorněno na ilustraci [2.1]. Komunikaci v MVC odstartovává uživatel vybranou akcí nabízenou z uživatelského rozhraní, čímž poskytne controlleru informace, ten pak přistupuje do modelu. V modelu zaktualizuje data na základě akce provedené z uživatelského rozhraní. Po aktualizaci přichází na řadu komponenta view, která zobrazí

zaktualizovaná data z modelu. Tímto krokem je akce ukončena a může započít nová s obdobným scénářem.




*Ilustrace 2.1: Model-View-Controller schéma architektury*

Pro nejširší záběr a uplatnění této práce jsou položeny proti sobě nejpoužívanější jazyky a pro ně nejznámější frameworky na tvorbu webových aplikací na straně serveru. V případě jazyka PHP jde o framework firmy Zend se stejnojmenným názvem, v případě jazyka Java se jedná o framework s názvem Spring a v neposlední řadě jazyk Ruby a s ním spojený framework Ruby on Rails. Jmenovaní představitelé mají svou architekturu založenou právě na MVC.

## 2.1 Zend Framework

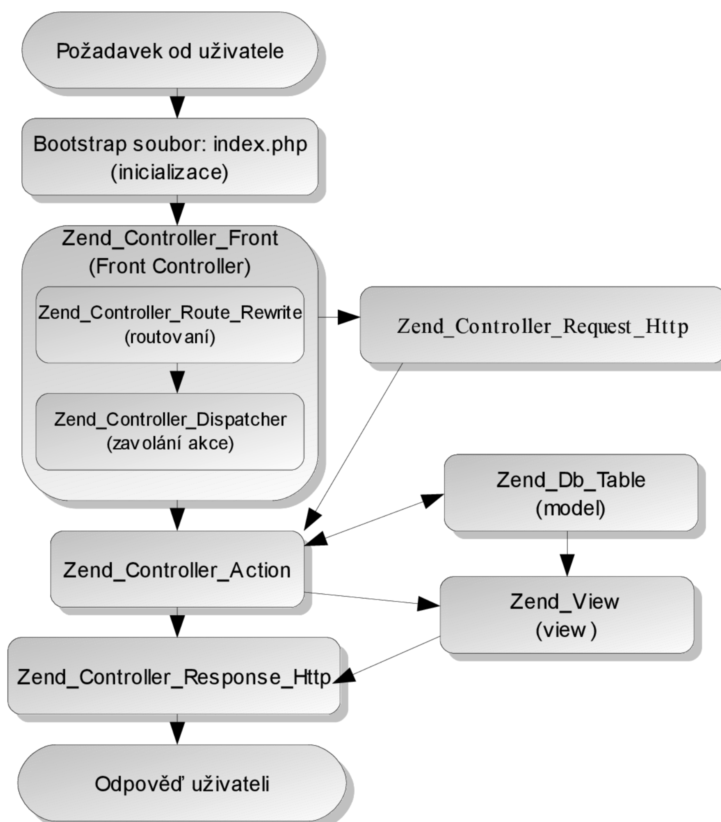
Zend Framework (ZF) je PHP MVC framework vyvinutý společností Zend Technologies Ltd., která stojí za vznikem PHP. Zend Framework je soubor knihoven, které mají výrazným způsobem ušetřit práci všem tvůrcům PHP aplikací.

V současnosti existuje celá řada PHP frameworků, které se liší hlavně mírou využití objektově orientovaného programování (OOP) při stavbě aplikací. Zend Framework klade na OOP mimořádný důraz a to je jedna z vlastností proč se v této práci budeme zaobírat právě jím z celé škály PHP frameworků. Dalším aspektem, proč je Zend tím správným krokem, je silné zázemí frameworku zastoupené firmou Zend, která ho neustále vyvíjí, opečovává a dodává kvalitní dokumentaci k jednotlivým prvkům frameworku z čehož vyplývá i silná komunita vývojářů. Mezi další výhody patří: „Convertion over configuration“, v překladu: „Konvence nad konfigurací“, jedná se o softwarový vzor, který eliminuje složité konfigurační procesy a zachovává potřebnou flexibilitu. V podstatě se jedná o doporučení (konvence) jakým způsobem psát kód. Například, mějme třídu „prodej“, pak odpovídající tabulka v databázi se bude nazývat „prodej“. Při nedodržování těchto konvencí je zapotřebí psát konfigurační soubory, jejichž počet při stoupající složitostí aplikace, roste.

### 2.1.1 Jádro Zend Frameworku

Pomocí Zend Frameworku lze vyvíjet naprosto kompletní aplikace, neboť se skládá z balíčků, které obsáhnou velikou část měřítka využitelnosti, jako například generování PDF, autorizace, autentizace, persistence aj. Celkový systém je neinvazivní, což znamená že lze balíčky používat samostatně.

Jádrem Zend Frameworku je Zend_Controller. Je to balíček, který se stará o obsloužení požadavku, vytvoření modelu a vyrendrování view, neboli odeslání odpovědi. Je postaven na návrhovém vzoru Front Controller, což je v podstatě aplikace architektonického vzoru MVC. Pro tuto konkrétní aplikaci platí pravidlo, že kontrolér (a potažmo i aplikační logika) je volán ještě před vstupem do view. Tedy všechny požadavky komunikují přes jeden objekt, který rozhoduje, který z action controllerů požadavek obslouží. Na ilustraci 2.2 je vidět životní cyklus obsluhy požadavku.




*Ilustrace 2.2: Životní cyklus obsluhy požadavku v ZF*

I. V prvním kroku, hned po požadavku od uživatele, přichází na řadu bootstrap soubor (index.php), ve kterém je vytvořena instance třídy Zend_Controller_Front. Nad ním se zavolá metoda run nebo dispatch, která začne obsluhovat požadavek.

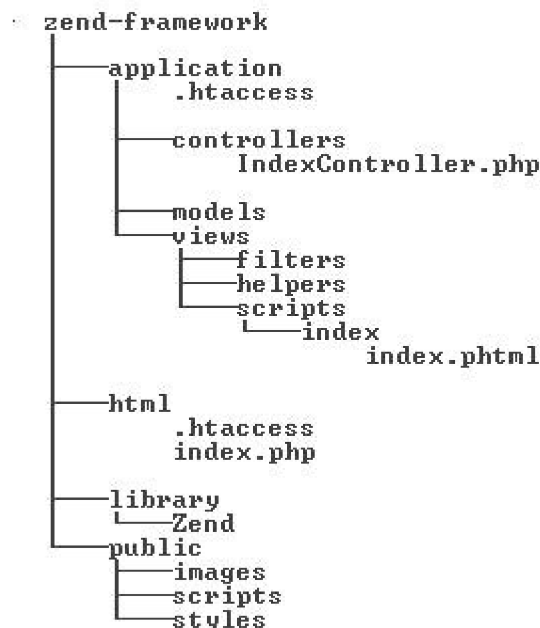
II. V dalším kroku dojde k vytvoření instancí objektů Zend_Controller_Request a Zend_Controller_Response. Tyto objekty nahradí globální proměnné.

III. Po vytvoření nastává proces routování. Routovací proces získá z URL adresy název controlleru (Zend_Controller_Action) a název akce (příslušná metoda), plus uloží další proměnné, které uživatel popsal v požadavku. Pro lepší pochopení se jedná o třídu (místo v kódu), která nám obslouží daný požadavek.

IV. Nyní vykoná svoji práci Zend_Controller_Dispatcher, který na základě názvu action controlleru a jeho akce vytvoří jeho instanci a zavolá příslušnou metodu. V této metodě dojde k naplnění modelu dat a poté následuje vyrendrování view a jeho nastavení do Zend_Controller_Response a odpověď je odeslána klientovy.

### 2.1.2 Uspořádání projektu v ZF

Tato kapitola vychází z vlastnosti konvence nad konfigurací. V Zend Framework je standardní adresářová struktura, která ulehčuje orientaci v projektu, jak samotnému vývojáři (což zjistíte samotným programováním), tak i vývojářům, kteří přebrali cizí projekt. Celá adresářová struktura je vidět na obrázku 2.3.




*Ilustrace 2.3: Zend - uspořádání projektu*

#### Popis jednotlivých adresářů včetně jejich uplatnění

- `application` – adresář, který obsahuje všechny třídy, objekty, soubory jenž souvisí s daným projektem.
- `controllers` – adresář, obsahující action controllery, v kterých je kód na obsluhu daného požadavku.

- `models` – adresář slouží k reprezentaci perzistentních dat.
- `views` – a podadresář `scripts` jsou obzvláště důležité, nacházejí se v nich námi vytvořené html stránky s koncovkou `.phtml`, která slouží pro rychlé rozlišení (typ koncovky lze změnit konfigurací).
- `helpers` – adresář, slouží pro uložení helperů, přesněji pro uložení view helperů, které formátují výstup pro view stránky.
- `filters` – adresář slouží k uložení filtrů.
- `html` – je jediný adresář, který je přístupný z internetu pro uživatele. Obsahuje nejdůležitější soubor `index.php` (bootstrap soubor, viz 2.1.1) a soubor s označením `.htaccess`, pro konfiguraci. V případě ostatních adresářů, je dobré uvést soubor s označením .`htaccess` s kódem `deny from all`, pro zablokování uživatelům vstoupit do adresáře.
- `library` – adresář, kde je místo pro knihovny a hlavně pro samotnou knihovnu Zend.
- `public` – adresář, ve kterém jsou uložené podporované obrázky, scripty a CSS soubory.

### 2.1.3 Součásti ZF

V úvodní části o ZF byly zmíněny nejdůležitější komponenty, které se podílí na obsluze požadavku od uživatele, jako např.: Zend_Controller, Zend_Db, Zend_view, avšak ZF je mnohem rozsáhlejší, a tato kapitola poukazuje na to co všechno ještě skýtá.

- **Zend**

Slouží především pro nahrávání ostatních tříd a práci s registrem, tedy o něco chytřejší obdobou globálních proměnných.

- **Zend_Controller**

Zend_Controller je postaven na návrhovém vzoru Front Controller a je považováno za jádro celého systému. Pracuje s objekty *Zend_Controller_Request*, jenž obsahuje volanou cestu, název controlleru, název modulu, název akce a další parametry a *Zend_Controller_Response*, který obsahuje hlavičky, tělo odpovědi a chybové informace, které nastali během dispatchovacího procesu. Na Zend_Controller_Front lze nastavit pluginy. Pluginy jsou kusy kódu, které se mohou napojit v určitých momentech na životní cyklus obsluhy požadavku. Určitými momenty jsou myšleny události jako, začátek routování, konec routování, začátek dispatchování, začátek dispatchovací smyčky, konec dispatchovací smyčky a konec dispatchování. Tyto pluginy mají přístup k objektům Zend_Controller_Request a Zend_Controller_Response a můžou tak modifikovat požadavek.[1]

---

1 Nastavovat do jeho kontextu přihlášeného uživatele, pracovat s cache, logovat aj.

Obdobou pluginů jsou Action Helpery, které taky mají přístup k událostem našeho controlleru, avšak na rozdíl od pluginů pouze ve třech případech: inicializace action controlleru, predispatch a postdispatch (před a po zavolání metody v action controlleru). Jeden z hotových action helperů se nazývá ViewRenderer, který se stará o automatické vytvoření Zend_View při inicializaci action controlleru a automatické vyrenderování příslušného view skriptu. Dalším action helperem je Redirector, který umožňuje přesměrovat na určitou akci (controller). Poslední věcí o které se zmíním v souvislosti se Zend_Controllerem jsou View Helpery. View Helpery se používají ve view skriptech a slouží k usnadnění opakujících se akcí (formátování textu). Zend má několik View Helperů připravených a jedním z nich je Url Helper, který slouží ke generování internetové adresy a dalším je Form helper generují vstupy a různá tlačítka.

- **Zend_Db**

Balíček Zend_Db je dalším stěžejním balíčkem v Zendu, a to čistě z důvodu, že v dnešní době většina webových stránek nějaký způsobem manipuluje s daty, nebo přesněji komunikuje s databází a právě Zend_Db poskytuje abstrakci nad přístupem k databázi. Zend_Db je třída realizující vytváření instancí adaptérů. Adaptéry jsou objekty, které slouží pro přístup k specifickým databázím (Oracle, MySql, Postgre, IBM apod.) a jsou dvojího typu.

První používající rozšíření PDO a druhý, používající jiné typy rozšíření (MySqli). Dalším typem objektu je *Zend_Db_Statement*, který se stará o kvótování dat od uživatele.[1]

Metody nad Zend_Db_Statement:

- `fetchALL('sql',$parametry);` – vrací pole objektů dle fetching_strategy
- `fetchAssoc` – vrací vždy asociativní pole
- `fetchCol` – vrací pole hodnot (první sloupec z řádku)
- `fetchPairs` – vrací první a druhý sloupec
- `fetchRow` – vrací jednu řádku
- `fetchOne` – vrací jednu hodnotu

```
$result = $db->fetchAll('SELECT * FROM ucty WHERE ucet_id = ?',2);
```

Jeden z případů jak se dá použít Zend_Db_Statement, kde za `ucet_id` je vyplněná dvojka, což chrání kód, před útoky z requestu. Spojení `db` je instance příslušného databázového adaptéru a všimněme si, že jsme nevytvářeli žádnou instanci Zend_Db_Statement, protože pracujeme přímo s adaptérem, který si jí v sobě sám vytvoří. Cílem tohoto zápisu je co nejvíce ulehčit vývojáři práci, co se týče množství kódu, a mimo jiné má také výhodu v tom, že si můžeme na objektu

1 Příklad použití: Nastavíte si SQL na insert (update apod.) a místo proměnných dáme otazníky a statement je vyplní.

Zend_Db_Statement_Mysqli volat další metody, kterými můžeme upravit výsledek požadovaným způsobem.

```
$sql = 'SELECT * FROM ucty WHERE nastaveni_u = ? AND stav_u = ?';
$statmnt = new Zend_Db_Statement_Mysqli($db, $sql);
$statmnt -> execute(array('klient','FIXED'));
```

Další důležitý objekt je *Zend_Db_Select*, který vytvoří novou instanci Selectu, což přináší výhodu skládání SQL dotaz přesně jak potřebujeme, čehož využijeme například u složitých formulářů. Další výhodou používání Zend_Db_Select je možnost komunikovat se všemi databázemi (ten samý dotaz bude fungovat na Oraclu, Mysql aj.), jelikož při překladu selectu na řetězec vezme v úvahu dialekt daného systému (adaptéru) a převede jej do něj.

```
$select = $db->select()
     ->from(array('k'=> 'klienti'),
                array('klient_id', 'klient_jmeno'))
     ->join(array('z'=>'zamestnanci'),
                k.klient_id = z.klient_id');
```

Předposlední třídou o které je v rámci Zend_Db řeč je *Zend_Db_Table* a s ním související *Zend_Db_Table_Row*. Použitím Zend_Db_Table, vstupujeme do čistě objektového řešení, kde podědíme z této třídy a vytvoříme si vlastní třídu, která bude pracovat s jednotlivými řádky tabulky jako se skutečnými objekty. Zjednodušeně řečeno lze přistupovat k sloupcům tabulky jako kdyby to byly public vlastnosti objektů, bez toho abychom je museli explicitně mapovat. Třída umožňuje i namapování vztahů mezi objekty (tabulkami) – one to many, many to one, many to many.

Poslední třídou je *Zend_Db_Profiler*. Při její aktivaci můžeme přistupovat ke všem vykonaným SQL dotazům a jejich parametrům (např. rychlost vykonání) a odstranit tak případné úzké hrdlo práce s databází.

- **Zend_Auth**

*Zend_Auth* je balíček, který pracuje s autentizací uživatelů. Autentizace je proces ověřování, zda objekt je tím za koho se vydává. Nejpoužívanějším adaptérem je *Zend_Auth_Adapter_DbTable*, který pracuje s databázovou tabulkou jako zdrojem dat o uživatelích.[1]

Ukázka autentizace:

```
$authAdapter = new Zend_Auth_Adapter_DbTable($db);
$authAdapter->setTableName('users');
```

1 Další možnosti adaptérů jsou Zend_Auth_Adapter_Basic, Zend_Auth_Adapter_Digest

```
$authAdapter->setIdentityColumn('username');
$authAdapter->setCredentialColumn('password');
$authAdapter->setCredentialTreatment('MD5(?)');
$authAdapter->setIdentity($username);
$authAdapter->setCredential($password);

$auth = Zend_Auth::getInstance();
$result = $auth->authenticate($authAdapter);

if ($result->isValid()) {
	$data = $authAdapter->getResultRowObject(null,'password');
	$auth->getStorage()->write($data);
}
```

Na první řádku došlo k vytvoření instance `Zend_Auth_Adapter_DbTable`, kde je v konstruktoru uložen `db` adaptér. Poté dochází k nastavení dalších potřebných udajů, jako název tabulky, kde jsou uloženi uživatelé, název sloupce s přihlašovacími jmény, název sloupce s hesly. Užitečnou věcí je příkaz `setCredentialTreatment`, který nás informuje v jaké podobě jsou uložená hesla v tabulce, protože každý komerční systém nemá v tabulce uloženou čistou textovou hodnotu hesla, nýbrž hodnotu hashe. Následuje samotný proces autentizace, který se vykoná zavoláním *`Zend_Auth::getInstance()->authenticate($adapter)`*. Návratovou hodnotou je objekt *Zend_Auth_Result*, který obsahuje informaci o výsledku autentizace, samotnou identitu (obvykle uživatelské jméno) a případně chybové hlášky. Zend_Auth_Result obsahuje jenom hodnotu uživatelského jména, ale může se stát, že bude potřeba mít informaci i o celém jméně přihlášeného uživatele. K tomu slouží příkaz `getResultRowObject`, kde v prvním parametru jsou uvedena jména sloupců, které chceme zobrazit (`null` znamená že chceme vidět všechny) a druhý parametr obsahuje názvy sloupců, které nechceme zobrazit. Po úspěšném přihlášení se údaje o uživateli ukládají do *Zend_Auth_Storage* (v defaultní implementaci do session).

- **Zend_Acl**

Balíček pracující s autorizací. Autorizace je proces, při kterém zjišťujeme, zda daná role (např. přihlášený uživatele) má práva k přístupu k danému zdroji (např. stránka nebo jiný objekt).

Konfigurace probíhá vytvořením všech chráněných objektů a rolí. Pomocí *Zend_Acl* nadefinujeme vazby – explicitně vyjmenuje jaká role má (nebo naopak nemá) přístup k jakým chráněným objektům. Jak chráněné objekty (Zend_Acl_Resource), tak role (Zend_Acl_Role) obsahují dědičnost – role dokonce vícenásobnou. Příklad použití Zend_Acl:

```
$acl = new Zend_Acl();

$acl ->addRole(new Zend_Acl_Role('guest'))
     ->addRole(new Zend_Acl_Role('member'))
     ->addRole(new Zend_Acl_Role('admin'));

$parent = array('guest','member','admin');
$acl->addRole(new Zend_Acl_Role('myUser'), $parent);

$acl->add(new Zend_Acl_Resource('myResource'));

$acl->deny('guest','myResource');
$acl->allow('member','myResource');

echo $acl->isAllowed('myUser','myResource')? 'allowed':'denied';
```

První úsek kódu vytváří instanci třídy `Zend_Acl` a inicializuje role (quest, member, admin). Dále je v kódu vidět vícenásobná dědičnost rolí, kdy vznikne nová role s názvem `myUser`, která dědí ze všech rolí v poli `parent`. Na dalším řádku vzniká chráněný objekt, který má `id` objektu `myRousource`. Příkazy `deny` a `allow` nastavují přístupová práva k chráněným objektům. Metoda `deny` označuje že daná role nemá přístup k danému objektu, v našem případě `guest` nemůže přistoupit k datům `myResource`. Naopak metoda `allow` povoluje přístup k chráněným objektů, čili v našem případě `member` může přistoupit k datům `myResource`. A nakonec poslední řádek kódu, který je podstatně zajímavější co se týče práv, jelikož `myUser` dědí ze všech rolí, je problém jakým způsobem tedy rozhodnout o přístupu. ZF řeší tento problém tak, že se podívá na pole rolí, ze kterého vznikla role `myUser` a postupuje odzadu. První odzadu je `admin`, ten nemá nadefinované práva k objektu `myResource`, proto se postupuje dál na `member`. `Member` má přístup k datům `myResource`, proto skončí prohledávání a pro roli `myUser` se nastaví přístup k těmto datům podle role `Member`, v našem případě může přistupovat. Výsledkem tohoto je, že záleží na pořadí vkládání rolí do pole. Metoda `isAllowed` má kromě parametru `allowed` a `denied` ještě další dva a to typ práva (`read` (čtení), `write`(zápis)) a `assertion`. `Assertion` je důležitý a mocný nástroj založený na nadefinování funkce Bool assert, ve které si můžeme nastavit defakto vše co se přístupových práv týče.[1]

1 Využití assertion např. pro objednávky. Nastavíme assert tak, aby vzalo id přihlášeného uživatele a id produktu, porovnalo je, a pokud jsou stejné pokračovalo se dál. Toto řešení nám odbourá potřebu pro každou objednávku definovat Resource a role.

- **Zend_Cache**

Dočasné zachovávání dat je nedílnou součástí každé webové aplikace, a pro trochu náročnější dotazy a akce nutností. Pomocí balíčku Zend_Cache lze zachovat prakticky vše co uživatel potřebuje (výstup, soubor, funkci, metodu, celou stránku aj.). Zachovalá data můžete ukládat do souborů (a přitom se nemusíte nikdy starat o to, zda cache nevypršela), SQLite, ap. Ukázka použití cache:

```
$id = 'mojeVeta';
if (!($data = $cache->load($id))) {
	$data = '';
	for ($i = 0; $i<160; $i++) {$data = $data.$i}
	$cache->save($data);
}
```

Vidíme, že ke dočasně zachovaným objektům se můžeme dostat přes jejich `id` nebo také použitím určitého tagu. Na dalším řádku je vidět podmínka, která zjistí, zda-li nebylo už jednou uloženo. A poté cyklem vložíme data a uložíme data, v tomto případě nějaké složení řetězců, do cache.

## Další balíčky Zend Frameworku

- Zend_Search_Lucene – pro fulltextové vyhledávání
- Zend_Mail – posílání MIME zpráv včetně příloh
- Zend_Pdf – vytváření PDF
- Zend_Config – umožňuje spravovat nastavení v INI či XML
- Zend_View – vytváří výstup z frameworku. Umožňuje integrovat různé šablonovací systémy, například Smarty.
- Zend_Feed – RSS a Atom klient
- Zend_Json – vytváření a čtení JSON
- Zend_Date – API pro práci s datem. Muže zobrazovat čas a datum ve 130 formátech.
- Zend_Form – zpravování formulářů. V aplikaci můžete definovat pravidla pro formulář a pak v šabloně jedním příkazem celý formulář zobrazit.

Zend Framework obsahuje ještě několik dalších balíčků a s plynoucím časem jich stále více a více přibývá, proto zde je pravá chvíle odkázat na oficiální stránky Zend frameworku (http://framework.zend.com), kde je k nalezení podrobný popis jednotlivých balíčků, jejich použití a samozřejmě aktuální verze frameworku.

### 2.1.4 Nasazení ZF a příklad

ZF pro svůj správný chod potřebuje v první řadě PHP verze 5.1.4 a novější. Poté potřebuje webový server, který bude podporovat mod_rewrite. Mod_rewrite slouží pro překlad adres. Čeká na serveru na každou požadovanou stránku, zkontroluje její adresu a podle seznamu svých pravidel, pokud požadovaná adresa v seznamu je, aplikuje na ni konkrétní přepisovací pravidlo.

K samotné tvorbě aplikace - vytvoříme v kořenovém adresáři webového serveru složku `example` (námi vytvořená aplikace bude tedy na adrese http://localhost/example). V adresáři `example` vytvoříme adresářovou strukturu viz. 2.1.2. Nakopírujeme knihovny Zend frameworku do složky `example/library/`. Tímto krokem jsme přidali do našeho příkladu Zend knihovny a adresářová struktura bude vypadat `example/library/Zend`. Nyní máme aplikaci připravenou k jejímu použití a můžeme začít psát kód.

Nejprve je důležité dodržet myšlenku Front Controlleru (do aplikace je pouze jeden vstup). Bude se jednat o jeden soubor index.php (bootstrap). Abychom splnili tuto podmínku napíšeme do souboru `.htaccess` v adresáři `html` tento kód:

```
RewriteEngine on
RewriteRule !\.(js|ico|gif|jpg|png|css)$ index.php
```

První řádek zapne mod_rewrite pro překlad adres a druhý přesměruje všechny URL na index.php. Nesmíme opomenout, že obrázky, skripty a CSS soubory nemohou být přesměrovaný na bootstrap soubor.

#### Obsah index.php

```
set_include_path('.' . PATH_SEPARATOR . '../library'
.PATH_SEPARATOR . '../application/models/'
.PATH_SEPARATOR . get_include_path());

require_once('Zend.php');

function __autoload($class){
Zned::loadClass($class);}
```

Nejprve se nastaví `include_path` pro model a knihovnu. Pak se zahrne třída `Zend.php`, která je hlavní třídou ZF a slouží pro načítání ostatních tříd a uskladnění globální proměnné. Nakonec ještě nadefinujeme autoload, který načte přes metodu `loadClass()` všechny třídy. Tím máme index.php hotový a přejdeme na nastavení Zend_Controller_Frontu.

```
$controller = Zend_Controller_Front::getInstance();
$controller->setControllerDirectory('../application/controllers');
$controller->dispatch();
```

Zavolá se singleton třídy Zend_Controller_Front a nastavuje se adresář s controllery. Poslední metoda dispatch() spouští celý proces. Nastavili jsme adresář s controllery, a tak je nutné vytvořit příslušný controller IndexController.php a uložit ho do adresáře s controllery (application/controllers)

**Obsah IndexController.php**

```
class IndexController extends Zend_Controller_Action
{
	function indexAction()
	{
		echo 'Hello World!';
	}
}
```

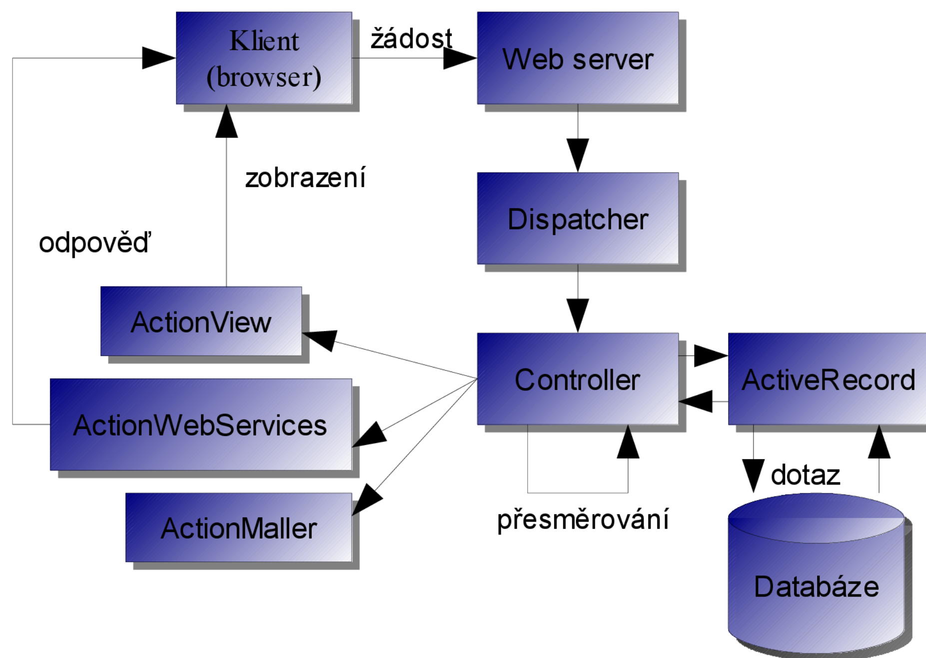
Hotovo, po zadání URL: http://localhost/example v našem webovém prohlížeči dojde k vypsaní programátorsky světoznámé větě: „Hello World!“.

## 2.2 Ruby on Rails

Dalším frameworkem zařazeným do srovnání je Ruby on Rails (RoR). Jedná se o poměrně nový framework, který za svoje poslední dva, tři roky získal takovou slávu, že mnozí o něm mluví jako o dokonalém nástroji na vývoj webu, avšak jiní soudí že jeho popularita je vyvolaná pouze dobrým marketingem.

### 2.2.1 Vlastnosti RoR

Ruby on Rails je vývojový framework postavený na bázi vzoru Model-View-Controller, který automaticky mapuje URL na vnitřní řídící prvky aplikace, abstrahuje přístup k datům v databázi pomocí Object-relational mapping („řádky“ v databázi se převedou na instance objektů, „sloupce“ na jejich atributy) a obsahuje rozsáhlé pomocné knihovny pro snadné generování HTML, práci s Ajaxem, formátování dat a další. Controller, model a view jsou implementovány moduly ActionController, respektive ActiveRecord a ActionView. Způsob propojení modulů a celková komunikace v RoR je znázorněna na obrázku 2.4.




*Ilustrace 2.4: Schéma komunikace v RoR*

Dalším charakteristickým znakem RoR je upřednostnění konvence nad konfigurací, o které byla zmínka u Zend Frameworku 2.1, kde je princip tento princip popsán. Hlavní myšlenka programování v RoR zní: „Nejjednodušší řešení je zpravidla nejsprávnější“, a proto byl na naprogramování frameworku zvolen programovací jazyk Ruby, který je vyhlášen svojí zábavností na programovaní. Jazyk Ruby je základem RoR, a proto abychom Rails mohli aktivně používat, je nutné se s jazykem Ruby seznámit.

### 2.2.2 Jazyk Ruby

Jazyk Ruby započal svojí existenci v roce 1993 pod perem Yukihira Matsumota, který si chtěl napsat svůj plně objektově orientovaný skriptovací jazyk. Jazyk Ruby je interpretovaný, což znamená že odpadá potřeba kompilace, avšak nevýhodou je nižší výkonnost běhu programu.

Syntaxe jazyka je velmi lehce čitelná a i člověku neznalému Ruby dojde co přibližně může daný úsek kódu vykonat.

```
Open(”example”).read count(”/n”)
```

Znázorněný řádek kódu vrací počet řádků v souboru.

#### Proměnné jazyka Ruby

Proměnné jsou referencemi na objekty a jsou jednoznačně určeny svým jménem. Není třeba je definovat, jsou vytvářeny automaticky a jsou beztypové. Rozeznáváme čtyři druhy proměnných:

- $g...globální proměnná
- @ proměnná instance

- @@ proměnná třídy
- [a-z] lokální proměnná
- [A-Z] konstanta

Proměnné self a nil, jsou známé výrazy z jiných programovacích jazyků a jedná se o identifikaci právě zpracovaného objektu a o proměnnou false.

## Pole a haše

Jedná se o indexované kolekce. V případě pole jde o indexy celočíselného typu (integer) a pokud jde o haše, indexem je jakýkoliv objekt.

**Příklad pole:**

```
a = %w{Jarda Mirek Iva Eva}
```

Definovali jsem pole "a" kde a[0] má hodnotu Jarda, a[3] obsahuje řetězec Eva.

```
druh={'Eva'=>'zena','Mirek'=>'muz','Pes'=>'zvire','Jana'=>'zena'}
puts film['Jana']
```

První řádek kódu nám definuje kombinaci klíč=>hodnota. Druhý řádek vrátí hodnotu „zena".

## Objektově orientované znaky jazyka Ruby

Jazyk Ruby je jak jsem již řekl čistě objektově orientovaný jazyk, což znamená že všechna data jsou objekty. Například číslo je instancí třídy Fixnum (Bignum).

**Příklad práce s objekty:**

```
1    Class Ahoj
2    end
3    ahoj = Ahoj.new
4    cau = Ahoj.new
5         def ahoj.hello
6         puts "Hello"
7    end
8    ahoj.hello
9    cau.hello
```

První řádek obsahuje kód `class „název třídy"`, který uvozuje začátek třídy, výraz `end` uvozuje konec třídy. Vytvoření instance je na třetím řádku, vznikla výrazem `„název třídy".new`. Na dalším řádku definujeme metodu `def „metoda"...end`. Zde by se měla dodržovat konvence, názvy tříd jsou psány na začátku velkým písmenem a názvy metod malým.

Poslední dva příkazy v kódu znázorňují volání metody. V prvním případě metoda vypíše „`hello`" a v druhém skončí chybou, jelikož metoda není definovaná. Velikou výhodou je, že definice tříd, modulů, metod jsou vytvářeny až za chodu, což umožňuje dynamické generování kódu.

Mezi klasickými metodami nám jazyk nabízí i speciální metodu initialize, která je volaná při vytváření nové instance třídy. Dalšími metodami jsou tzv. Destruktivní metody, které mění stav objektu. Vyznačují se znakem „!" za názvem metody. Poslední speciální metodou je singleton. Jedná se o instanci metody vázanou na jednu instanci třídy. Použijeme v případech kdy potřebujeme přidat novou metodu instance, ale není výhodné vytvářet odvozenou třídu. K metodám bezesporu patří řízení přístupu. Ruby používá tři stupně přístupů – public, protected a private. Metody jsou standardně public (s vyjímkou konstruktoru, který je private). Přístupy public a protected jsou stejné jako v C++[1]. Metody private mohou volat jen objekty nadřazené třídy s tou vyjímkou, že příjemce musí být jen objekt self. Z toho vyplývá, že dva objekty stejné třídy nemohou volat metody private.

Ruby implementuje pouze jednoduchou dědičnost a to pomocí operátoru „<". Vícenásobná dědičnost je nahrazena pomocí modulů. Modul je uvozený `module...end`. Jsou to kolekce metod, které může každá třída pomocí include naimportovat. Stejný modul může být importován v různých třídách, z toho vyplývá, že úpravou jednoho modulu upravíme všechny třídy, jenž importovali daný modul. Třída, která tento modul importuje se nazývá mix-in.

Operátorem `super` je možné se odkázat na metodu nadřazené třídy. Pokud potřebujeme volat metodu vyššího předka, přiřadíme jí nové jméno pomoci alias.

## Ošetření vyjímek

Víme-li, že v určité části kódu hrozí předčasné ukončení v důsledku vzniku chyby, můžeme tento „nebezpečný" blok označit a v případě vyvolání výjimky na něj patřičným způsobem zareagovat. K tomuto účelu nabízí Ruby následující konstrukci:

```
begin
     #nebezpečný kód vyvolávající vyjímku
rescue Exception
     #ošetření konkrétního druhu vyjímky
ensure
     #kód prováděný vždy
end
```

1 public – přístupné v celém programu
private – přístupné jen v metodách dané třídy a ve spřátelených funkcí
protected – přístupné jen v metodách dané třídy a odvozených tříd a ve spřátelených funkcí

Druh výjimky je reprezentován různými potomky třídy `Exception`. Vlastní výjimku lze vyvolat příkazem `raise`.

### Bloky, iterátory, uzávěry

Bloky kódu patří mezi největší přednosti jazyka Ruby. Umožňují vytvářet bloky vázané na volání metody. Blok tvoří kód uvozený do '{' ... '}' nebo 'do' ... 'end'. Blok a následně i jednoduchý iterátor si ukážeme na příkladu:

```
[1, 2, 3].each do |item| print item end
```

je ekvivalentní tomuto zápisu:

```
array = [1, 2, 3]
i = 0
while i<array.size do
     item = array[i]
     print item
     i+=1
end
```

Jak je vidět iterátory výrazně zkracují délku kódu a usnadňuje tak práci vývojáři. Iterátor each je hlavním důvodem zkrácení zápisu. Může se však stát, že vývojář potřebuje mít vlastní iterátor each.

Bloky v Ruby jsou ve skutečnosti uzávěry, což znamená, že s blokem je možno komunikovat prostřednictvím proměnných, které mohou přistupovat k určitým proměnným bloku. Nejlépe jsou uzávěry vidět na příkladě.

```
"Mam v kapse 5 korun!".sub(/\d+/) {|i| i.to_i * 2}
```

Výsledkem tohoto řádku je: `Mam v kapse 10 korun`. Je vidět, že Ruby obsahuje mocnou implementaci regulárních výrazů.

### Shrnutí vlastností jazyka Ruby

Jak je vidět Ruby přináší spoustu prvků o kterých se dá velmi dlouho a rozsáhle mluvit. Nakonec bych se ještě zmínil o garbage collection, což je vlastnost, kterou Ruby disponuje. Funkce spočívá v automatickém uvolňování paměti, která se nepoužívá (neukazuje na ní žádná proměnná). Ruby navíc implementuje svůj vlastní systém vláken.

Jazyk Ruby je napsaný v jazyce C, a tak i syntaxe je velice podobná, proto je poměrně jednoduché přejít z jazyka C na Ruby. Z této vlastnosti plyne i rozšířitelnost a přenositelnost jazyka.

Ať už je to s frameworkem RoR jakkoliv jazyk Ruby je bezesporu zajímavý a pěkný programovací jazyk v kterém je vývoj aplikací doslova zábava.

### 2.2.3 Implementace MVC v RoR

Struktura aplikace v RoR nese na první pohled prvky architektonického vzoru MVC

```
app/
     controllers/
     helpers/
     models/
     views/
```

Ve složce *app* se nacházejí složky pro *controllery*, *modely*, i *view*. Tyto tři složky jsou srdcem Rails aplikace.

**Součásti z pohledu MVC:**

1. Převod dat z relační databáze na objekty Ruby (toto zajišťuje Active Record)
2. Směrování (routing) - mapování URL na vnitřní řídící prvky (*controllery* a jejich metody) a předávání dat mezi *controllery* a *views*
3. Pomocné metody pro práci s HTML formuláři, Ajaxem, formátování řetězců, data a času atd. (model a view)

**Další klíčové součásti Rails:**

1. Skripty pro generování kódu
2. Konzole pro přímou iteraci s modely běžící aplikace na příkazové řádce (script/console)
3. Komunita a sdílení kódu

Těchto šest oblastí nám dává dobrou představu o tom, jak jsou Rails uspořádány, proto si v následujícím textu postupně projdeme ve zkratce jednotlivé body.

#### Active Record (Model)

Active Record je srdcem každé Rails aplikace. Jedná se o návrhový vzor, který mapuje databázové tabulky na třídy a převádí řádky na objekty (tedy instance tříd) a sloupce na jejich atributy: z názvu třídy (*Article*) odhadne název tabulky (*articles*)[1]. To znamená, že odstiňuje vývojáře od SQL a od vlastnoručního převádění výsledků dotazu na nativní datové entity a umožňuje efektivní práci na daleko vyšší úrovni.

Jak tedy ActiveRecord nahradí takovýto dotaz

```
SELECT * FROM articles WHERE id = 1
```

1 Pro odhad obsahuje jednoduchou implementaci anglického slovníku, která je snadno rozšiřitelná.

V Rails vytvoříme třídu *Article*

```
class Articles < ActiveRecord::Base
end
Article.find(1)
```

Pokud by měl model brát v potaz jinou než standardní tabulku musí se mu to explicitně zadat

```
class Article < ActiveRecord::Base
     set_table_name "clanky"
end
```

Active Record nám dává nástroje nejenom pro získávání dat, ale i pro manipulaci s nimi. Pokud chceme změnit některý atribut daného záznamu, píšeme kód takto

```
article = Article.find(1)
article.author = 'Seneca'
article.save
```

Změníme atribut a ten uložíme. Třídy odvozené od Active Record obsahují metody `create` a `destroy` pro vytváření a mazání záznamů. Active Record obsahuje také metodu `find_by_sql` pro specifické případy složitých dotazů, nebo například dotazů, které je třeba silně optimalizovat.

Active Record umí mimo mapovaní dat z tabulek, také další užitečné věci, mezi ně patří validace vstupních dat a vztahy mezi objekty. Validace vstupních dat je jednou ze základních podmínek bezpečnosti a správné funkce webových aplikací. Vezměme si tedy definici třídy Article:

```
class Article < ActiveRecord::Base
     validates_presence_of :author
end
```

Předešlí kód říká, že author nesmí být prázdný, nebo-li že každý článek musí mít autora. Další validační funkce jsou:

- `validates_lenght_of`
- `validates_uniqueness_of`
- `validates_format_of` (využívající regulární výrazy)

Další ze zmíněných vlastností je podpora pro vztahy mezi objekty. Pro názornost použijeme třídu Article, která obsahuje autora jako articles. Avšak v použitelném návrhu by jsme vymezili na autora vlastní tabulku, pravděpodobně s názvem authors. Článek může mít více autorů, autor může napsat více článků, jde tedy o vztah N:M (many to many). Pro propojení tabulek stačí definovat vztahy v deklaraci tříd.

```
class Article < ActiveRecord::Base
     has_and_belongs_to_many :authors
end
class Author < ActiveRecord::Base
     has_and_belongs_to_many :articles
end
```

Nyní vytvoříme nový článek, nového autora, a autora k článku přiřadíme:

```
article = Article.new
article.title = "Loren Ipsum"

cicero = Author.new
cicero.name = "Cicero"
cicero.save!

article.author << cicero
article.save!
```

Provázali jsme tabulky a o výsledku se dá přesvědčit příkazem `print article.to_yaml` nebo z druhé strany, pokud by jsme chtěli dostat seznam Cicerových článků `print cicero.articles.to_yaml`. Metoda `to_yaml`, pomocí niž dojde k vypsání tabulky, je Rails formát YAML (čitelnější obdoba XML). Tento formát využívá Rails jednak pro konfigurace, tak pro čitelné výpisy datových struktur.

### Action Controller (Controller)

Funkce cotrolleru je obdobná jako v případě Zend Frameworku. Základem je routování (směrování), které zajišťuje, že aplikace má informaci, na jaké prvky má směrovat konkrétní požadavek. Vznikají tzv. hezké URL (pretty URL), což znamená, že uživatel má představu co ho čeká, po přečtení dané URL. Pro představu si uvedeme dvojici URL.

```
http://rails.com/exec/default?action=list&templat=article_list=4
http://www.rails.com/article/show/1
```

Na první pohled je vidět, která z URL je ta hezká. Hezká URL napovídá, že zobrazí `show`, nějaký článek `article` s identifikačním číslem 1.

Direktivy v Rail, které řídí routování jsou obsaženy v souboru config/routes.rb, kde najdeme následující text.

```
map.connect ':controller/:action/:id'
```

Z URL tedy zjistíme jaký controller se použije, jaká metoda tohoto controlleru převezme požadavek a jaké parametry se této metodě předají. Pokud je tato URL strukturu dodržena, aplikace bude vědět co si s požadavkem počít. Neméně důležitý je i výpis adres. Rails obsahují pohodlné a čitelné generátory URL. Routování v Rails je odstiňuje vývojáře od direktiv pro `mod_rewrite` podobně, jako ActiveRecord od SQL. Jedním z konceptů je koncept pojmenovaných routes.

```
map.show_article 'article/show/:id',
                                    :controller => 'article',
                                    :action     => 'show',
                                    :id         => /\d+/
```

Toto pravidlo zachytí URL, které začíná výrazem `article/show`, za kterým následuje číslo (alespoň jedno číslo). Kromě větší flexibility a případně i přehlednosti routovacích pravidel mají pojmenované routes jednu velkou výhodu – lze je volat ve views.

## Propojení controlleru a view

Propojení mezi logikou aplikace a výstupem v HTML stránce je jednou z nejkomplikovanějších oblastí vývoje webu. Rails nepoužívají žádný šablonovací jazyk (template language). HTML šablony Rails[1] obsahují normální Ruby kód, tzv. Embedded Ruby (ERB). Rails rozlišují obecnou šablonu celé aplikace nebo jenom její části.

- layout – obecná šablona celé aplikace
- template – šablony pro konkrétní výstup

Pokud Rails najdou ve složce `app/views/layouts/` soubor `application.rhtml`, použijí jej automaticky jako obecný layout pro všechny controllery, dokud nenaleznou jinou konfiguraci. Je možné tedy vytvářet speciální HTML layouty pro určité controllery. Na následujícím kódu je zobrazeno, jakým způsobem je možné nastavit danému controlleru specialní layout, pro jeho zobrazení.

```
class ArticleController < ApplicationController
    layout "clanek"
    # zbytek logiky controlleru...
end
```

Rails bude layout pro *controller Article* hledat v souboru `app/views/layouts/clanek.rhtml`. Další výhodou je možnost layout deklarovat dynamicky, podle stavu aplikace.

## View v RoR

1 .rhtml soubory ve složce app/views

Jednoduchou korespondencí: „název controlleru“ = „název layoutu“ a „název metody“ = „název šablony“ Rails nešetří jenom psaní kódu, ale i konfigurování těchto vztahů. Pro určitou metodu je možné velmi jednoduše použít zcela jiný layout:

```
def show
    # logika metody...
    render(:template => 'zobraz')
end
```

Předávání dat do view: proměnné, které controller nastaví, jsou přístupné ve view. Představme si typický scénář, kdy chceme mít ve view dostupný objekt, obsahující data o článku s ID 1. (Tomu odpovídá URL ve tvaru: `http://www.example.com/article/show/1`). Pomocí ActiveRecord data přečteme z databáze a uložíme do proměnné `@article`. Zavináč před názvem proměnné znamená, že proměnná bude platit pro celou instanci controlleru a bude dostupná i ve view. Pokud bychom proměnnou nazvali `article` (bez zavináče), platila by pouze v rozsahu metody `show` (viz. proměnné jazyka Ruby 2.2.2). Ve view pak proměnnou `@article` můžeme snadno použít:

```
<h1><%= @article.title %></h1>
<div><%= @article.body %></div>
<hr />
<p>Publikováno dne: <%= @article.published_at %></p>
```

Výraz `<%=` otevírá místo v HTML, kam prostřednictvím Embedded Ruby vypisujeme dynamický obsah.

### 2.2.4 Instalace RoR

Pro použití Rails je potřeba nainstalovat jazyk Ruby (verze 1.8.3 jazyka Ruby vhodná není pro Rails). Poté je nezbytná instalace RubyGems což je standardní manažer balíčků pro Ruby. Poté je možné započít s instalací Rails včetně jejich závislostí pomocí příkazové řádky

```
gem install rails --include-dependencies
```

Nyní již máme vše připraveno pro vývoj našich aplikací. Připavíme si kostru naší aplikace a spustíme server:

```
rails „cesta k naší Rails aplikaci“
cd „cesta k naší Rails aplikaci“
ruby script/server
```

Další možností je použít distribuci, která obsahuje všechno v jednom balíčku (webový server, databázi, Ruby i Rails).

Balíček je možné stáhnout na adrese: *http://instantrails.rubyforge.org*

## 2.3 Spring Framework

Rámec Spring je modulární JEE aplikační rámec, který je vytvořen pro vývoj jakéhokoliv typu aplikace. V podstatě jako u ostatní předem uvedených frameworků, ani tento nevymýšlí nic převratného, pouze poskytuje uživateli velké množství rozšířených softwarových nástrojů a tím tak využít specializovaných a časem prověřených řešení na danou problémovou oblast.

### 2.3.1 Vlastnosti Spring Frameworku

Stěžejními návrhovými technikami ve Springu jsou programování orientované na aspekty (AOP - Aspect Oriented Programming) a návrhové vzory Obrácení řízení a Injektáž závislostí.

#### Obrácení řízení (IoC - Inversion of Control )
#### a Injektáž závislostí (DI -Dependency Injection )

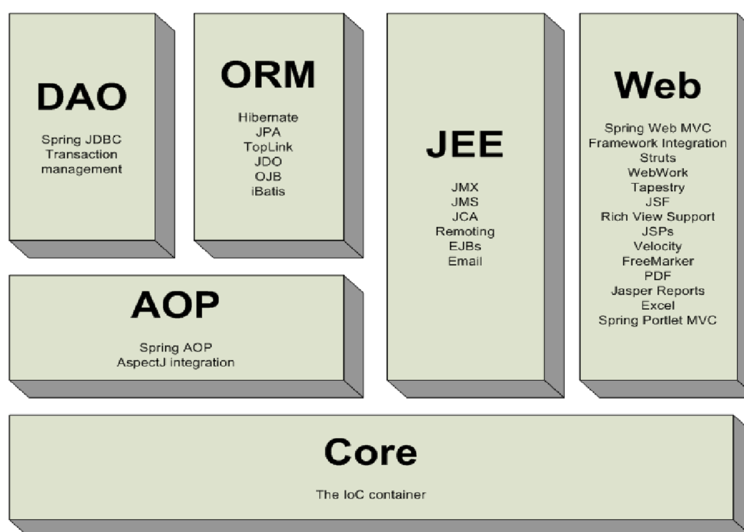
Obrácení řízení je již dlouho používaný návrhový vzor, jenž v původním významu představoval techniku, s jejíž pomocí dochází k přenesení řízení běhu programu z kódu, který navrhuje programátor, na podpůrný aplikační rámec. Jednodušeji řečeno programátor vyvíjí pouze dílčí moduly a o jejich sestavení se postará aplikační rámec. Avšak v tuto chvíli přichází na řadu injektáž závislostí, aby řekl aplikačnímu rámci, jakým způsobem má sestavovat dané moduly, nebo-li vepíše (injektuje) závislosti jednotlivých modulů. Uvedeme si příklad máme závislost dvou objektů. První objekt obsahuje odkaz na druhý. Při použití DI budou oba objekty, při startu kontejneru, vytvořeny a v prvním objektu bude inicializován odkaz na druhý.

Spring Framework, jakožto kompletní aplikační rámec, se skládá ze sedmi organizovaných jednotek, jak je vidět na obrázku 2.5.

**Jádro (Core)** je nejzákladnější část rámce, jejímž základním konceptem je BeanFactory (Továrna tříd), která má na starosti základní funkcionalitu rámce Spring. Zejména provázání objektů pomocí DI a transparentní aplikaci služeb rámce pomocí AOP. Továrna tříd po svém startu drží objekty spravované aplikace, vzájemně je prováže a poskytuje odkazy na tyto objekty svým klientům. To, které objekty jsou továrnou tříd spravovány, je definováno většinou prostřednictvím konfiguračního souboru ve formátu XML.

**DAO (Data Access Object )** poskytuje JDBC-abstraktní vrstvu, která umožňuje odstranění opakujícího se kódu, kterým trpí většina aplikací pracující s JDBC API.

**AOP (Aspect Oriented Programming)** je modul implementující podporu pro aspektově orientované programování. Umožňuje separovat části kódu prolínající se celou aplikací (autorizace, logování, transakce) do takzvaných aspektů a jejich následnou aplikaci na jakýkoli *POJO* objekt[1].




*Ilustrace 2.5: Celkový pohled na Spring Framework*

**ORM (Object Relational Mapping)** je modul určený k podpoře integrace OR frameworků jako Hibernate nebo iBatis.

**WEB** jedná se o modul určený k podpoře integrace s web frameworky jako Struts, WebWork a JSF implementacemi. Obsahuje podpůrné třídy pro webové aplikace (např. práce s cookies, upload souborů) a samozřejmě WEB MVC, což umožňuje vytvářet webové aplikace na principu této návrhové třídy.

## 2.3.2 Web MVC – webová vrstva aplikace

V terminologii webové technologie pod platformou java se často vystytují dva termíny a těmi jsou servlet a JSP (Java Server Pages), proto je důležité si vysvětlit o co se jedná, než přistoupíme k aktivnímu používání těchto termínů. Servlet je program napsaný v jazyce Java, který běží na serveru, a který dokáže zpracovávat požadavky zaslané pomocí HTTP protokolu a dokáže na ně HTTP protokolem odpovídat. V podstatě, jednoduše řečeno, je každý servlet malý webový server. V případě JSP, jde o technologie umožňující směšovat statické HTML stránky s dynamicky generovaným obsahem a je jistou analogií s technologiemi jako jsou ASP či PHP. JSP stránky je možno používat i samostatně, ale podstatně lepší využití najdou ve spolupráci se Servlety. Nyní přistoupíme k samotnému jádru web MVC rámce.

1 Termín *POJO* je zkratkou *Plain Old Java Objects* - objekty, které nejsou svázané technologicky specifickým rozhraním.

Hlavní třídou, pokud jde o webové aplikace, je DispatcherServlet. Je to třída, která je postavena na návrhovém vzoru Front Controller, o kterém byla řeč již v souvislosti se Zend Frameworkem. Z návrhového vzoru plyne, že tento servlet je vstupním bodem každé webové aplikace.

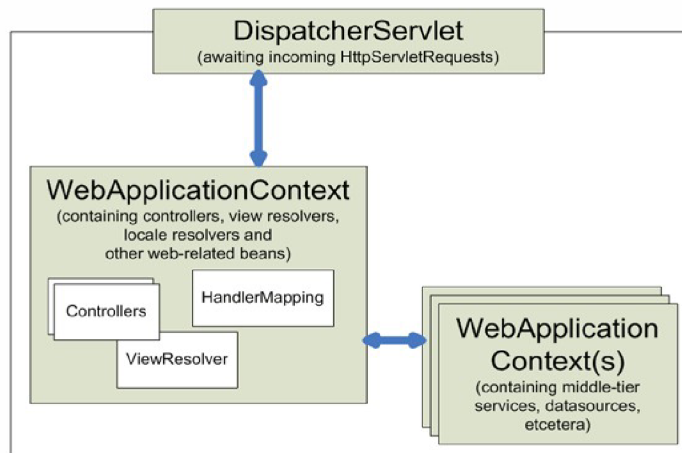
DispatcherServlet je zděděný ze základni třídy HttpServlet a jako takový je deklarovaný ve web.xml webové aplikace. Stejně tak žádosti na použití DispatcherServletu musí být mapovány v tomtéž souboru. Příklad deklarace a mapování DispatcherServetu může tedy vypadat takto.

```
<web-app>
  <servlet>
     <servlet-name>example</servlet-name>
     <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
     <servlet-name>example</servlet-name>
     <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

Tímto jsme definovali servlet s názvem example, který je instancí DispatcherServlet a je mapován na všechny požadavky s koncovkou .html. Navíc je zde možné pomocí contextConfigLocation definovat webový aplikační kontext tohoto servletu, který je inicializovám při startu servletu. Pokud není kontext explicitně zadefinovám použije se implicitní nastavení /WEB-INF/[název servletu]-servlet.xml. V tomto případě tedy example-servlet.xml. Servletů může být v naši aplikaci deklarováno několik a každý bude mít přiřazen svůj aplikační kontext. Tyto kontexty jsou nezávislé a vzájemně neviditelné, čím se zvyšuje míra modularizace webového rozhraní.

Objekty, které jsou součástí aplikační a datové vrstvy, jsou zpravidla společné pro všechny servlety, a proto je jejich definice umístěna v jednom kontextu, který je nadřazený všem ostatním. Pokud není jinak explicitně dáno očekává se definice kořenového aplikačního kontextu v souboru s cestou /WEB-INF/applicationContext.xml.




*Ilustrace 2.6: Hierarchie kontextů ve Spring Web MVC*

*--obrázek převzat z www.springframework.org--*

DispatcherServlet je připraven a očekává požadavek (`httpServletRequest`), jehož obsluha prochází následujícím životním cyklem.

i. Do `ServletDispatch.WEB_APPLICATION_CONTEXT_ATTRIBUTE` je umístěn aplikační kontext daného servletu, pro potřeby dalších objektů v životním cyklu požadavku.

ii. Jako atributy jsou do objektu požadavku přidány detektor národního prostředí (`localeResolver`), sloužící k rozpoznání národního prostředí klienta a detektor motivu (`themeResolver`), sloužící k rozpoznání individuálních uživatelských nastavení klienta

iii. Pokud je v aplikačním kontextu definován objekt s názvem `multipartResolver`, objekt požadavku je zabalen do objektu implementujícího rozhraní `MultipartHttpServletRequest`. Tím je usnadněna práce se soubory, které uživatel posílá serveru. Není-li v kontextu tento objekt, zůstává požadavek nezměněn. V aplikačním kontextu se vyhledají všechny objekty typu `handlerMapping` a v definovaném pořadí jsou dotázány na kontrolér, který obslouží požadavek, nebo na interceptor, který požadavek předzpracuje (postzpracuje). Není-li nalezen žádný objekt `handlerMapping` je vytvořena instance třídy `BeanNameUrlMapping`.

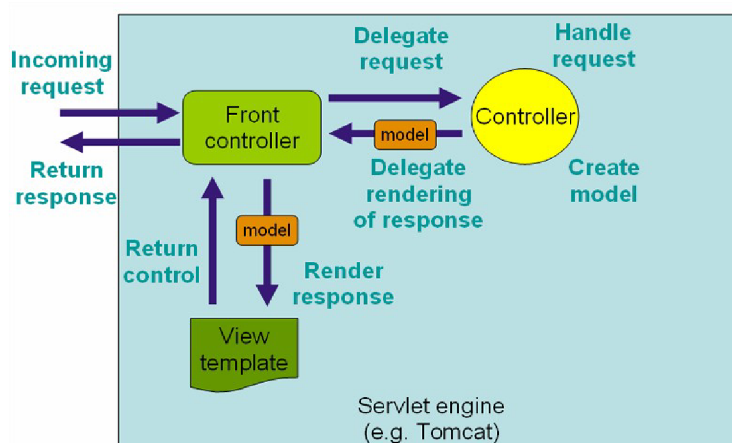
iv. V aplikačním kontextu je vyhledán objekt `handlerAdapter` a je mu předán požadavek. Konkrétní implementace tohoto rozhraní určuje jaký typ kontroléru bude použit v dalším kroku.

v. Požadavek je předán určenému kontroléru ke zpracování. Kontrolér vytvoří model ve spolupráci s aplikační vrstvou a určí logický název pohledu, který bude pro zobrazení modelu použit.

vi. Zde může dojít ke zpracování již zmíněným interceptorem (postzpracování), byl-li nalezen.

vii. V aplikačním kontextu jsou vyhledány všechny detektory pohledu (`viewResolver`). Detektory pohledu převedou logický název pohledu na cestu k souboru, který provede zobrazení. Nejčastěji se pro zobrazení užívají šablonovací technologie, kdy je konkrétním pohledem šablona daného jazyka (JSP).

Zpracování požadavku na vyšší úrovni abstrakce je zobrazeno na obrázku 2.7




*Ilustrace 2.7: Zpracování požadavku ve Spring Web MVC*

*--obrázek převzat z www.springframework.org--*

Pro komunikaci s modelem můžeme použít klasickou JDBC knihovnu, nebo využit ORM nástroje Hibernate. ORM znamená objektově relační mapování. Pro představu se jedná o obdobný nástroj jako je Zend_Db v případě frameworku Zend. Hibernate převádí persistentní data uložená v databázy na objekty, což přináší výhodu manipulace s persistentními daty jako s objekty. Nastavení Hibernate a implementace entit je popsána dále v textu v implementační části o Springu 4.4.

Posledním potřebným nástrojem pro chod Springové aplikace je nastaveni serveru Tomcat. Tomcat je oficiální referenčňí implementace technologií Java Servlet a Java Server Pages (JSP). Aplikace typu Tomcat je jakýsi zásobník servletů, který se stará o jejich spouštění, běh, ukončení, komunikaci s klienty a další věci. Tvoří prostředníka mezi komunikací javovské webové aplikace a webového serveru (Apache).

# 3 Způsoby prověřování frameworků

V úvodní části této práce (Framework pro webové aplikace) bylo zmíněno co je a k čemu slouží framework. Ze škály vlastnosti, které frameworky nabízejí, jsem vybral dva nejdůležitější pohledy na prověření kvality frameworků. Prvním pohledem je zaměření se na uživatelskou přivětivost frameworku a druhým pohledem je testování výkonnosti webové aplikace vytvořené daným frameworkem.

Pro plnohodnotné hodnocení těchto směrů je nezbytné se nejenom seznámit s logikou frameworku, ale i vytvořit aplikaci na jejich základě, jelikož ať již logika, či způsob zápisu, může být snadno pochopitelná a jasná, samotná implementace posléze může přinést nemalé potíže. Proto je nezbytné navrhnout vzorovou aplikaci, která bude demonstrovat um frameworku a tak se blíže seznámíme s tímto nástrojem.

## 3.1 Uživatelská přívětivost frameworku

Do testu uživatelské přivětivost frameworku je možné zařadit veškeré vlastnosti, které souvisí s uživatelem, respektive které uživatele nějakým způsobem ovlivní při tvůrčí činnosti.

Vlastnost, která ovlivňuje uživatele při tvorbě se dá shrnout pod obecné slovo, které je pochopitelnější širší oblasti populace, a tím slovem je problém (úskalí). Proto vlastnosti, které jsem zahrnul do testu mají nějaké potenciální úskalí pro vývojáře. Pro lepší uspořádání těchto problémových vlastností jsem je rozdělil do dvou kategorií. První z nich se bude zabývat problémy při studiu frameworku a druhá problémy, které vzniknou při nasazení frameworku na uživatelův systém. Než se pustím do rozboru každé z kategorií, zmíním věc, která je pro kategorie společná a tou je veličina čas, jenž bude jednou z veličin, která bude určovat míru použitelnosti frameworku.

### Úskalí při studiu

Jak již název napovídá nejpodstatnější vlastností je rozsah, robustnost a složitost frameworku. Samozřejmě čím rozsáhlejší je systém, tím delší dobu zabere jeho nastudování. Je tedy zapotřebí si uvědomit jak náročnou a rozsáhlou aplikaci budeme vyvíjet, abychom zbytečně nepoužívali velmi silný nástroj na prostou webovou aplikaci. Robustní systémy sebou často přinášejí velké množství nastavení, ať již formou například XML dokumentů nebo frameworku vlastní konfigurační metodou. Proto je bezesporu pro úsporu času, jak při implementaci, tak při studii, dát přednost systémům podporující konvence před konfigurací.

Dalším aspektem je dostupnost literatury a s ní související programátorská komunita. Čím silnější je programátorská komunita, tím více informací je k dispozici pro programátora. I kvalita

frameworku se tímto zvyšuje, jež se rychleji upozorní na případné implementační neduhy, nebo dojde na popud uživatelů k rozšíření částí, aby framework ještě lépe padl do ruky programátora. S literaturou je spojena i lokalizace, tím je myšlen dostatek informací, literatury dostupných v mateřském jazyce. Neboť i zde by se časový horizont mohl posunout vlivem náročného překladu logiky frameworku do jazyka jemuž programátor plně rozumí.

Jako dalšího možné úskalí bych uvedl invazivnost systému. Invazivností se myslí, že celková síla systému se integruje do výsledné aplikace a není možné užit pouze některé z vybraných komponent. Tato vlastnost úzce souvisí jak s výkonem výsledné aplikace tak i s náročností nasazení.

A v neposledním řadě rozhoduje o využití frameworku i znalost a zkušenost programátora. Jednak pokud je programátor začátečník, a nechce se věnovat profesionálnímu vývoji webu, neměl by se pouštět, z hlediska úspory času, do rozsáhlých systémů jakými je například Spring. Na druhou stranu pokud je programátor znalý v jednom z jazyků v kterých je framework napsán, nebo dokonce má zkušenosti s programováním v daném frameworku, pak je ideálnější řešení pokračovat v realizaci aplikací právě tímto frameworkem, pokud jeho síla stačí na danou aplikaci.

#### Úskalí pří nasazení

V této kategorii je nejvýznamnějším úskalím zprovoznění frameworku na různých operačních systémech a v jisté míře i na hardwaru. Může se stát že některé z frameworků nebudou mít v daném operačním systému podporu a nebo jen omezenou. Náročnost uvedení frameworku do úplného chodu se pak často zvyšuje z důvodu potřeby podpůrných nástrojů, které často potřebují speciální nastavení.

Zmínil jsem se o potřebě podpůrných nástrojů a ty jsou samozřejmě potřeba i při zavedení frameworku na jakýkoliv systém a hardware, jako například existence web serveru (Apache) nebo databázového serveru (MySQL, PostgreSQL, SQLite). Tyto programy je samozřejmě, také potřeba nastavit a toto nastavení propojit s vaším frameworkem. Pro začínající programátory bych doporučil software, který tyto programy realizuje tzv. all in one (všechno v jednom), kde je základní konfigurace automaticky provedena při instalaci a případná úprava je provedena přes grafické rozhraní, místo editace konfiguračního souboru.

## 3.2 Výkon webové aplikace

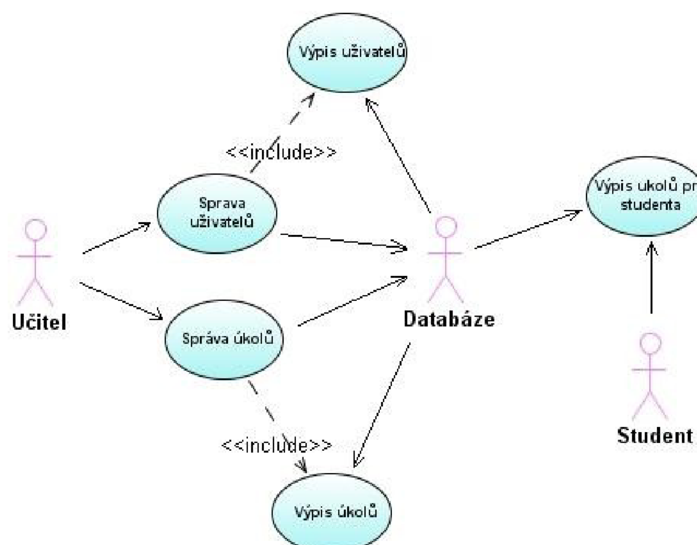
Bezesporu jedna z nejdůležitějších vlastností každé aplikace je její výkon. Ten ovlivňuje několik faktorů, jako je výkon hardwaru, způsob jakým byla aplikace napsána a samozřejmě jaký nástroj byl na vytvoření aplikace zvolen. K tomu abychom nemuseli vlastnoručně psát test rychlosti aplikace, jako například zaznamenat čas před započetím akce a po ukončení akce a jejich rozdílem zjistit čas potřebný na vykonání operace, můžeme použít nástroj jménem JMeter. JMeter je opensource aplikace

vytvořená jako součást projektu jakarta od Apache. Je napsán kompletně v jazyce java a nabízí grafické rozhraní v podobě swing komponent, nebo může běžet v non-GUI módu, kdy nevyžaduje na počítači, ze kterého se testuje, žádné grafické prostředí. Další mód je pro tzv. Remote testing a funguje tak, že na testovaném serveru se spustí pouze serverová část, které odesílá požadavky a výsledky posílá klientovi na jiném stroji. To umožňuje obejít místa mezi klientem a serverem, která zpomalují test. Nastavení JMeteru je popsáno v sekci 5

## 3.3 Vzorová aplikace

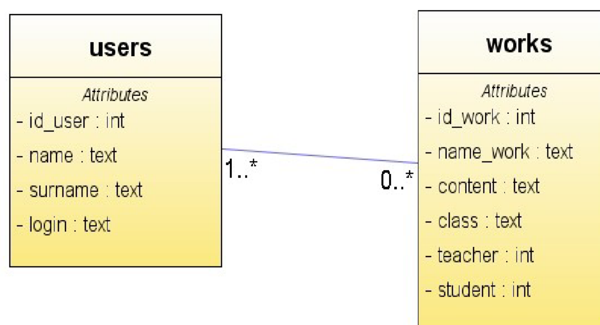
Vzorovou webovou aplikaci jsem navrhl tak, aby implementovala nejčastěji se vyskytující prvky ve webových aplikacích jako jsou informační systémy, které jsou velkou rodinou webových aplikací na poli Internetu. Prvky, které v takovéto aplikaci nesmějí chybět jsou formuláře, databázové dotazy a to hlavně INSERT, UPDATE, DELETE, JOIN a v neposlední řadě omezení přístupu nepovolaným uživatelům (autentizace, autorizace). Všechny tyto vlastnosti jsem shrnul do jedné aplikace s funkcí informačního systému základních a středních škol, kde se evidují domácí úkoly určené pro studenty. Tento informační systém jsem si vybral, neboť je mi tato tématika blízká a je snadno přeimplementovatelný, například na knihovní systém, systém internetového obchodu, nebo jakýkoliv systém, který pracuje s oddělenými přístupy a na základě nich jim podává informace a služby.

Způsob jakým vzorová aplikace komunikuje s klienty je vidět na obrázku 3.1, kde je znázorněn diagram případů použití.




*Ilustrace 3.1: Diagram případů použití vzorové aplikace*

Z obrázku je patrné, že učitel plní roli administrativní, což znamená že se stará o správu uživatelů systému a o správu svých úkolů. To obnáší jejich vytváření, editování a mazání. Student po přihlášení k systému získá informace z databáze o domácích úkolech. Výslednou aplikaci tvoří dvě tabulky, jak je patrné z obrázku 3.2. První z nich je tabulka users, kde jsou uloženi všichni uživatelé tohoto systému. Tabulka uchovává jejich jména, příjmení, popis, který v případě studenta nese informaci o třídě do které chodí a v případě učitele odborné předměty které vede. V neposlední řadě obsahuje ještě informace potřebné k přihlášení uživatele jako je uživatelské jméno, heslo a role pro oddělení administrátorského (učitelského) a studentského přístupu do informačního systému. Heslo v tabulce je kódováno pomocí hashe, konkrétně pomocí MD5. Přestože se jedná pouze o vzorovou aplikaci a tudíž se nepředpokládá její nasazení v běžném životě, je přítomnost hashe v tabulce zbytečná, avšak pro názornost a užitečnost této práce jsem ji zahrnul v práci, neboť každý solidní informační systém má hesla svých uživatelů uložena v databázi v zakódované podobě.




*Ilustrace 3.2: Diagram tříd vzorové webové aplikace*

Druhá tabulka, jménem works, nese informace o domácích úkolech a to přesněji o jeho názvu, obsahu, třídě, které byl úkol zadán, popřípadě jedinci, kterému byl zadán a učiteli jenž úkol vystavil. Tím je objasněna funkce vzorové webové aplikace a můžeme přejít k její realizaci.

# 4 Implementace

Než přistoupím k jednotlivým způsobům realizace, uvedu části, které jsou pro všechny vývojové nástroje společné. Všechny realizace systémů byly prováděny na operačním systému Windows XP Professional verze 2002 servise pack 3, jestliže by byl vývoj provozován na jiném operačním systému můžou se některé úkony mírně lišit (hlavně pokud půjde o nastavení podpůrných programů). Dalším společným prvkem ve vývoji je webový a databázový server. Ze škály webových serverů jsem vybral web server Apache (Apache ver. 2.2.11), který je podle mého mínění nejvíce rozšířen v podvědomí počítačového světa. A v neposlední řadě databázový server MySQL verze 5.1.30.

Veškerý potřebný nelicencovaný (volně dostupný) software k realizaci implementace a vytvořený kód z každém frameworku je přiložen na datovém médiu v této práci.

Po dohodě s konzultantem mé práce jsme uznali za vhodné do porovnání zasadit základní prvek pro tvorbu webových aplikací, čímž je jazyk PHP a prostřednictvím něho nastavit prvotní laťku v porovnání výkonu aplikací vytvořených za pomocí frameworků. Neboť jak již bylo řečeno v úvodu práce jedním z argumentů, které kritizují frameworky jako takové, je snížení výkonu výsledné aplikace vytvořené za pomocí frameworku. Tím, že zapojím do měření i čistý kód PHP se můžeme přesvědčit jak moc pravdivým tento výrok je a to hlavně v porovnání se Zend Frameworkem, který je nad PHP postaven. Pro práci s čistým PHP i na práci s Zend Frameworkem jsem zvolil PHP ver 5.2.8.

## 4.1 Implementace v PHP

Jazyk PHP je velice rozšířen mezi nováčky zabývající se tvorbou webových aplikací a to z důvodu snadné logiky a pochopení, proto se jím nebudu moc zaobírat, pouze ukáži základní práci z databází a ukáži zajímavou implementaci autorizace a autentizace pomoci sezení (session).

Než vůbec můžeme začít pracovat s databází je zapotřebí se k ní připojit.

```
$MC = MySQL_Connect("localhost", "root", "password");
$MS = MySQL_Select_DB("school");
...
MySQL_Close($MC);
```

Prvním příkazem jsme se připojili k databázovému serveru, který se nachází na adrese `localhost` s uživatelským jménem `root` a heslem `password`. Do proměnné `$MC` se uloží identifikační číslo otevřeného připojení, kterého využijeme při zavření spojení s databází na posledním řádku. Identifikační číslo spojení se ukládá do proměnné hlavně pro případ že by bylo otevřeno současně více spojení a tak by se prostřednictví identifikačního čísla odvolávalo na příslušnou databázi.

Druhým příkazem vybereme název databáze, s kterou se má nadále pracovat. V tomto případe `school` (škola). Po otevření spojení můžeme pokládat dotazy na vybranou databázi.

```
$dotaz = "select * from users where (login like '$login')";
$MSQ = mysql_query("$dotaz");
$pole = mysql_fetch_array($MSQ);
$jmeno = $pole["name"];
```

V prvním řádku uložím SQL dotaz do proměnné, čímž lehce zpřehledním kód. V tomto případě žádám databázi o všechny iniciály uživatele, jenž má položku `login` v databázi shodnou s uživatelský jménem uloženým v proměnné `$login`. Příkazem na druhém řádku požádám databázi o provedení dotazu. Výsledek tohoto dotazu se uloží do proměnné `$MSQ` a pomocí funkce `myqsl_fetch_array` naplním proměnnou pole jedním řádkem dotazu. Pole vzniklé touto funkcí je asociativní a klíčem k jednotlivým sloupcům je jejich název. Poslední příkaz uloží do proměnné `$jmeno` jméno uživatele, jenž měl totožné uživatelské jméno s jménem uloženým v proměnné `$login`. Velmi podobným způsobem jsem řešil autentizaci a autorizaci přihlášených uživatelů.

```
if ((IsSet($login)) AND (IsSet($password))){
  $p = MD5($password);
  $MSQ = MySQL_Query("SELECT * FROM users WHERE
         (login LIKE '$login') AND (password LIKE '$p')");
```

Proměnné `$login` a `$password` jsou hodnoty vrácené z přihlašovacího formuláře, kde funkcí `IsSet` se dotazuji, zda-li došlo k jejich naplnění ve formuláři, pokud ano, převedu si hodnotu proměnné `$password`, která symbolizuje heslo, na hash kód a uložím tento kód do proměnné `$p`, což je vidět na druhém řádku. Následuje dotaz do databáze o vyhledání uživatele s tímto uživatelským jménem a heslem. Na ověření správnosti jsem použil následující sadu příkazů.

```
1  if (MySQL_Num_Rows($MSQ) <> 1){
2    echo "Nesprávné uživatelské jméno nebo heslo.";
3  }
4  else {
5    $SN = "autorizace";
6    Session_name("$SN");
7    Session_start();
8    $sid = Session_id();
9    $time = Date("U");
10   $at = Date("U") - 1800;
```

```
11
12  $MSQ = MySQL_Query
13    ("INSERT INTO autorization VALUES ('$sid', $time, 'T')");
14  $MSQ = MySQL_Query
15    ("DELETE FROM autorization WHERE time < $at");
16
17  SetCookie ("lo_uc", $login, time()+3600);
```

Funkce MySQL_Num_Rows prací počet řádků jež byly výsledkem dotazu o hledání shody uživatelského jména a hesla. Pokud došlo k nalezení více nebo žádného uživatele je chyba na serveru nebo došlo k chybě při zadávání údajů (řádky 1-3 v kódu). Při správném chodu je nalezen pouze jeden uživatel s těmito vlastnostmi a ten musí mít do systému přístup. Vytvořím sezení, které bude informovat o tom, že je uživatel přihlášen dokud se neodhlásí (řádky 5-8 v kódu). Pro účely autorizace v PHP jsem vytvořil další tabulku s údaji o autorizaci, kam uložím identifikační číslo sezení, čas kdy došlo k přihlášení do systému a roli pod jakou se uživatel přihlásil (řádky 12, 13 v kódu). A v souvislosti smažu z tabulky všechny uživatele, kteří se přihlásili před více jak půl hodinou (řádky 14, 15 v kódu). Nakonec si uložím informaci o přihlášeném uživateli do cookies, abych ho měl kdekoliv v systému k dispozici (řádek 17 v kódu). Tímto je autorizace hotová. Poté stačí na každé stránce s omezeným přístupem uvést následující úsek kódu.

```
1 $SN = "autorizace";
2 Session_name("$SN");
3 Session_start();
4 $sid = Session_id();
5 $date = Date("U");
6 $ad = Date("U") - 600;
7 $typ = "T";
8 $MSQ = MySQL_Query("SELECT * FROM autorization WHERE
9     (id_aut = '$sid') AND (date >= $ad) AND (type = '$typ')");
10 if (MySQL_Num_Rows($MSQ) < 1){
11    echo "Neautorizovaný prístup";
12    Exit;
13    }
14 else{
15    $MSQ = MySQL_Query("UPDATE autorization SET
16         date = $date WHERE id_aut = '$sid'");
```

Znovu vytvoříme sezení, porovnáme jestli je v tabulce autorizace stejné identifikační číslo sezení, které vzniklo před méně než deseti minutami (řádek 6 v kódu ukládá proměnou, která nese hodnotu, jaký čas byl před deseti minutami) a s právy v našem případě administrátorskými ('T' jako teacher (učitel) řádek 8, 9 v kódu). Pokud byl nalezen takovýto přístup aktualizuje se čas v autorizační tabulce. Tato funkce nám zajišťuje odhlášení uživatele po době nečinnosti deseti minut. Abychom nemuseli tento rozsáhlí kód psát do každé stránky s chráněným obsahem doporučuji ho vložit do souboru, v mém případě autorizace.php a pomocí následujícího příkazu vložit do potřebné cílené stránky.

```
require('autorizace.php');
```

Další úkony na vzorové aplikaci informačního systému v PHP je opakované používání HTML tagů a databázových dotazů, které si myslím, není třeba nadále rozebírat.

## 4.2 Implementace pomocí Zend

Zajímavým měřítkem pro porovnání jednotlivých frameworků je jejich nasazení na daný systém. V případě Zend Frameworkem je nasazení jednoduché a po nastavení Apache a PHP není potřeba jiných programů k jeho úplné funkčnosti. Jediné co je potřeba je připojit knihovny do aplikace. V mé aplikaci je podle adresářové struktury Zend složka /library, ve které jsou potřebné knihovny pro běh aplikace. Implementace vzorové webové aplikace byla realizována na Zend Framework verze 1.7.5.

Při implementaci systému v PHP je v podstatě jedno jaké použijete vývojové prostředí, pro potřeby programátora postačí inteligentní zvýraznění syntaxe, plus například doplňování konce tagů. Zatím co při vývoji v Zend bych doporučil vývojový nástroj Eclipse, který má v sobě integrované pluginy pro práci se Zendem. Využití těchto pluginu usnadní práci a psaní kódu je pohodlnější.

Způsob komunikace v Zend je popsán v teoretické části této práce, proto se zde již nebudu zabývat, jaký kontrolér, metoda nebo jaký pohled se použije. Hlavními pilíři aplikace je práce s databází za pomocí třídy Zend_Db, práce s formuláři pomocí Zend_Form a autentizace, autorizace pomoci Zend_Auth, Zend_Acl.

### 4.2.1 Použití databáze

Nejdřív nastavím jakou databázi chci v aplikaci použít a jaký adaptér má třída Zend_Db použít při konstrukci SQL dotazu. Nastavení jsem provedl v konfiguračním souboru config.ini ve tvaru

```
[default]
```

```
; Nastavení databáze
db.driver            = Mysqli
db.username          = root
db.password          = password
db.dbname            = school-zend
db.host              = localhost
db.encoding          = cp1250
```

Konfigurace není nijak složitá a intuitivně lze odvodit co který řádek konfigurace nastavuje. Řádek `[default]` je pojmenování sekce. Tuto konfiguraci připojím do programu (v bootstrap souboru). Použiji třídu Zend_Registry, abych mohl přistupovat ke konfiguraci všude v aplikaci.

Způsob připojení je vidět dále v textu, kde jako parametry třídy `Zend_Config_Ini` jsem uvedl cestu k souboru s konfigurací a název sekce, kterou chci načíst a výslednou konfiguraci uložím do registru.

```
$config = new Zend_Config_Ini
           ('../application/default/config.ini', 'default');
Zend_Registry::set('config', $config);
```

V poslední řadě získané nastavení předám databázi

```
$db = Zend_Db::factory($config->db);
Zend_Db_Table::setDefaultAdapter($db);
```

Nyní je databáze nastavena a připravena k použití, vytvořím objekt, který bude reprezentovat entitu v tabulce. V adresáři `/models` vytvořím soubor `Work.php` s kódem

```
<?php
class Work extends Zend_Db_Table
  {
    protected $_name = 'works';
    protected $_primary = 'id_work';
  }
```

Objektu sdělím jméno tabulky, tak jak se jmenuje v databázi a určím primární klíč. Nyní pracuji s údaji v tabulce `Works` jako s objekty třídy `Work`. Potom funkce realizující databázový dotaz JOIN vypadá takto.

```
public function getJworks(){
     $select = $this->getJwork()->select();
```

```
    $this->_jwork = new Work();
    $select->from('works',array('id_work', … , 'teacher'))
           ->join('users' , 'users.id_user = works.teacher');
    return $this->getJwork()->fetchAll($select);
}
```

Funkce `getJwork` vrací instanci třídy `Work`, návratovou hodnotou této funkce je pole objektů tabulky `works` doplněnou o informace z tabulky `users`. Této funkce jsem využil při zobrazení domácího úkolu studentovi, ale v tabulce je pouze cizí klíč na učitele a to by studentovi nic neřeklo. Spojením pomocí cizího klíče zobrazím celý úkol plus podrobnosti o učiteli (jméno, příjmení). Po nastavení databáze je vidět, že nám opravdu framework usnadňuje práci s databází. Vyhneme se složitým SQL dotazům a tím že je realizujeme jako metody k třídě je velmi snadné je znovu kdekoliv v kódu využít.

### 4.2.2 Použití formulářů

Pro práci s formuláři jsem použil třídu Zend_Form, použití ukáži na realizaci přihlašovacího formuláře. Tento formulář požaduje pouze přihlašovací jméno a heslo

```
class FormLogin extends Zend_Form
{
    public function __construct($options = null)
    {
     $login = new Zend_Form_Element_Text('login');
        $login->setLabel('Uživatelské jméno');
     $heslo = new Zend_Form_Element_Password('pass');
        $heslo->setLabel('Heslo');
     $submit = new Zend_Form_Element_Submit('submit');
        $submit->setValue('Přihlásit se');

     $this->addElements(array(
            $login,
            $heslo,
            $submit
     ));
    }
 }
```

V konstruktoru třídy jsou jednotlivé položky formuláře instancemi třídy Zend_Form_Element. Zajímavá je implementace třídy Zend_Element_Password, která se nám postará o maskování hesla tečkami zatím co v implementaci pomocí samotného PHP jsem musel tečkovou notaci pracně realizovat pomocí cyklu, zde je opět vidět úspora času. Nehledě na to, že třída Zend_Form nabízí velké množství filtrů a validátorů, které by opět v PHP byly nahrazeny útrpnou prací programátora. Pro názornost ukáži jak k formuláři připojit filtr a validátor.

```
$login->setLabel('Uživatelské jméno');
$login->addFilter('StringTrim');
$login->addValidator('NotEmpty');
```

Přidaný filtr odstraní bílé znaky na začátku a konci řetězce. Validátor zajišťuje neprázdnost položky uživatelského jména. Krom těchto výhod přináší takto vytvořený formulář ještě výhodu znovupoužitelnosti, které jsem využil v případě formuláře pro přidání uživatele do systému, tento formulář jsem využil nejen pro přidání uživatele, ale i pro jeho editaci. Jeden kód, dvojí využití.

### 4.2.3 Použití autentizace a autorizace

Třídy jenž realizují autentizaci a autorizaci, nejsou až tak propracované, jak by programátor očekával od takovýchto tříd. Třídy v podstatě řeší pouze základní funkci zabezpečení, jako je porovnání uživatelského jména a hesla, co se týče autentizace a kontrola role přihlášeného uživatele pro omezení přístupu k systému. Chybí implementace omezení přístupu podle vedlejších kritérií, jako je identifikace v síti (omezení podle IP adresy), nebo časově ohraničeného spojení v době nečinnosti. Tyto funkce si musí každý individuálně implementovat. Je možné, že s následující verzí Zend Frameworku se tyto nedostatky odstraní. Na druhou stranu udržují tyto třídy dostatečnou abstrakci nad těmito operacemi. V následujícím kódu je vyřešena autentizace pomocí třídy Zend_Auth.

```
$form = new FormLogin();
     if (!$this->getRequest()->isPost()) {
          $this->view->form = $form;
          }
     else {
          $db = Zend_Registry::get('db') ;
          $authAdapter = new Zend_Auth_Adapter_DbTable
                         ($db, 'users', 'login', 'password');
          $authAdapter->setCredentialTreatment('MD5(?)');

          if ($form->isValid($_POST)) {
```

```
                $authAdapter->setIdentity($form->login->getValue())
                     ->setCredential($form->pass->getValue());

                $result = $authAdapter->authenticate();
```

Vytvořil jsem instanci přihlašovacího formuláře, o kterém jsem mluvil v části, jak manipulovat s formuláři pomocí Zend_Form. Pokud dojde ke korektnímu zpracování formuláře, vytvořím instanci třídy `Zend_Auth_Adapter_DbTable` a předám jí informaci, kde se vyskytují autentizační údaje systému. Jak je vidět pro přístup k databázi používám databázového adaptéru, který jsem uložil do registru, zde se mi právě hodí. Funkcí `setCredentialTreatment`, oznámím autentizačnímu adaptéru, že heslo je uložené v podobě hash kódu a nakonec prostřednictvím metody `authenticate()` provedu autentizaci. Bezesporu další výhodou třídy je možnost kdekoliv v kódu vyvolat informaci, který uživatel se autentizoval a nemusím se jako programátor starat o ukládání dat do cookies. Získání informace o přihlášeném uživateli

```
auth = Zend_Auth::getInstance();
```

Informace  však vrací pouze uživatelské jméno přihlášeného klienta a to je poměrně málo k rozumnému použití v systému. Proto jsem přidal k informacím o uživatelském jméně i ostatní informace vyjma hesla. Těchto údajů užiji při zobrazení, který klient je přihlášen (jméno, příjmení).

```
auth->getStorage()->write($authAdapter->getResultRowObject(null,
array('password')));

jmeno = Zend_Auth::getInstance()->getIdentity()->name;
prijmeni = Zend_Auth::getInstance()->getIdentity()->surname;
```

Autorizace je založena na nastavení, které naleznete v souboru `acl.php` ve složce `/library/skola`. Stručný výpis kódu nastavení, ukazující jak jednotlivým rolím nastavuji možnosti pohybu po systému je následující.

```
$this->allow('guest' , 'login');
$this->allow('guest' , 'index');
$this->allow('member' , 'student');
$this->allow('admin', 'ucitel');
```

Je vidět, že například uživatel `member` (v mém případě student) má povolen přístup na kontrolér `student`. Dále musím sdělit programu, aby s těmito nastaveními pracoval. Následný výpis úryvku kódu z metody `preDispatch` ověřuje přístup k systému pomoci této konfigurace. Kód je napsaný v metodě `preDispatch`, neboť kontrola přístupu se musí provést při každém routování.

```
$acl = new Skola_Acl($auth);
if ($this->_auth->hasIdentity())
            $role = $this->_auth->getIdentity()->role;
        else $role = 'guest';
if ($this->_acl->has($resource)){
     if ($this->acl->isAllowed($role, $resource, $action)) {
            $module = $this->$request->module;
            $controller = $this->$request->controller;;
            $action = $this->$request->action;
     }
}
```

Vytvořím si instanci třídy, ve které jsem nadefinoval nastavení přístupu do systému, a ujistím se jestli došlo k přihlášení nějakého uživatele. Pokud ne, uživatel, který přistupuje do systému bude mít implicitně definovanou roli jako host (`guest`). Dál se zeptám jestli existuje vůbec nějaký taký zdroj (`resource`), jaký požaduje klient a jestli jsou k němu nastaveno oprávnění pro tohoto klienta. V případě, že ano klienta pustím na jím žádanou adresu, v případě že ne odkáži ho na adresu přihlášení.

Použitím třídy Zend_Acl odpadává použití další tabulky autorizace, jak jsem tomu učinil v případě vývoje systému na samotném PHP, ačkoli věřím, že lze pomocí PHP učinit autorizaci i bez doplňující tabulky, tak považuji variantu s ní za rozumné z důvodu modifikovatelnosti.

Tím jsem shrnul největší úskalí při realizaci systému na frameworku Zend, další práce na systému byla kombinace těchto znalostí, čistého PHP a HTML tagů.

# 4.3 Implementace pomocí Ruby on Rails

Již v implementační části Zendu jsem se zmínil o měřítku nasazení frameworku na daný systém, a proto zachovám strukturu a i v případě Ruby on Rails začnu touto tématikou. V případě frameworku Rails je algoritmus nasazení trochu obtížnější než v případě Zendu. Je to do jisté míry způsobeno absencí jazyka Ruby na systému. V případě Zendu se jednalo o jazyk PHP a ten jsem již instaloval při práci s čistým PHP, tudíž jsem při oživování Zendu, tento krok vynechal.

Prvním krokem je tedy instalace jazyka Ruby (Ruby ver. 1.8.6) spolu s ním Ruby Gems, což je manažer pluginů pro jazyk Ruby. Poté příkazem

```
gem install rails --include-dependencies
```

nainstalujeme Railsy. K tomuto druhu instalace je zapotřebí připojení k internetu, neboť balíčky pro Rails jsou stahovány z repositáře na adrese `gems.rubyforge.org`. Tímto jsou Rails nainstalovány a připravené k použití. Nejtěžším krokem je však jejich nastavení. Rails mají v sobě integrovaný webový server WEBrick, jehož použití je nejjednodušší a často se používá pro vývojovou činnost, neboť odpadá nastavování externích serverů. Jelikož byl předchozí projekt vyvíjen na webovém serveru Apache, považuji za rozumné, tak tomu učinit i v případě Rails. K tomu aby Apache rozuměl souborů, s kterými Rails pracuje je zapotřebí použít FastCGI skriptu. Nainstalujeme fcgi, mod_fastcgi a podporu pro Rails

```
gem install fcgi
```

Po provedení všech instalací je nutné nastavit Apache a to připsáním následujících řádků do `httpd.conf`.

```
<Directory /www/>
     AllowOverrideall
</Directory>

LoadModule fcgid_module modules/mod_fcgid.so

<IfModule mod_fcgid.c>
     AddHandler fcgid-script .fcgi
     IPCCommTimeout 40
     IPCConnectTimeout 10
     DefaultInitEnv RAILS_ENV production
     SocketPath /tmp/fcgidsock
</IfModule>

<VirtualHost *:80>
     ServerAdmin root@localhost.cz
     DocumentRoot /www/rails_skola/public
     ServerName localhost
     ErrorLog /logs/httpd/rails_skola-error_log
     CustomLog /logs/httpd/rails_skola-access_log common
     Options Indexes ExecCGI FollowSymLinks
     RewriteEngine On
</VirtualHost>
```

Je vidět že nastavení Rails na jiný server než WEBrick není jednoduché. Dalším úskalím v nastavení je přístup do databázového serveru. Nastavení pro tento server je uloženo ve složce `/config` v naší webové aplikaci v souboru `database.yml`. Jedná se o YAML konfigurační soubor. Tento soubor po vygenerování generátorem Rails aplikace je standardně nastaven pro databázový server SQLite verze 3. Abych opět zachoval návaznost musím tento soubor přepsat pro MySQL databázy, nebo použít speciální parametr pro generování struktury Rails aplikace

```
rails rails_skola -d mysql
```

Tento příkaz připravil aplikaci na práci s MySQL, ale je potřeba provést ještě její nastavení v souboru `database.yml`. Tento soubor obsahuje tři režimy použití databáze `development` (vývoj), `test` (test) a `production` (předvedení). Databáze je implicitně nastavena používat režim development. Pozor při použití režimu `test` dochází k vymazání stávající databáze.

Až v tomto kroku je Rails připraven na tvůrčí činnost. Je vidět, že nasazení na nestandardní nastavení je poměrně složité a dosti se liší v použitém operačním systému. Zatím co v Unixových distribucí připojíme několik balíčků v případě MS Windows musíme dlouho hledat potřebné instalace a návody na nastavení pro konkrétní problém.

Dále je vývoj pomocí Rails poměrně snadný díky svým generátorům kódu. O administrativní činnosti, jako jsou přidání, editace a mazání uživatele, se postará sám Rails a i pro tuto činnost vygeneruje příslušné formuláře, kontroléry a pohledy. Jediný na co je třeba dát pozor je, aby v době generování byly tabulky v databázi již vytvořeny, neboť by nemohl podle nich scaffold generátor vygenerovat strukturu formuláře. Příkaz jenž vygeneruje vše pro správu domácích úkolu je

```
gem script/generate scaffold Work
```

Takto vytvořený kód není potřeba prakticky modifikovat. I použití autentizace a autorizace je zajištěno generátorem. Jedna se o login generátor a je potřeba ho pomocí správce balíčků gem doinstalovat a pomocí skriptu vygenerovat.

```
gem install login_generator
ruby script/generate login Users
```

Login generátor používá pro hashování hesel SH1, pro modifikaci stačí pouze zaměnit SH1 za MD5. Railsy vygenerovaný kód zbývá jen lehce upravit pro vlastní potřebu a je hotovo. Velikou výhodou právě těchto generátorů je nejenom úspora psaného kódu, ale i to, že Rails za programátora řeší logiku aplikace a on jí pouze ohýbá pro svoje potřeby.

Je vidět, že největším úskalí programování v Rails je jeho nastavení, které zabralo více než tři čtvrtiny času celého vývoje.

## 4.4 Implementace pomocí Spring

Poslední nástroj, který jsem využil pro implementaci vzorové aplikace je Spring. Jedná se bezesporu o nejrozsáhlejší framework z vybraných a práce s ním by vydala na několik separátních prací. Přestože jsem se omezil pouze na jednu jeho část a to webovou vrstvu, zabral vývoj spoustu času a úsilí. Pro používání Springu musíme mít k dispozici jazyk Java (Java ver. 1.6.0), balíčky Springu

(spring.jar, spring-webmvc.jar), balíčky Hibernate, adaptér pro komunikaci s MySQL serverem (určené pro nastavení dialektu) a v neposlední řadě server Tomcat (Tomcat ver. 5.5.27).

Vzorovou aplikaci jsem vyvíjel v Eclipse JEE developers ver 3.4.2. Kam je potřeba všechny výše uvedené balíčky připojit. Balíčky se nacházejí v složce `/lib` webové aplikace. Abych se vyhnul rozsáhlé konfigurace pomocí souboru xml, použil jsem anotací. Anotace přišli jako novinka s verzi 2.5 a vypovídají o druhu objektu ke kterému jsou vázány a konfigurace posléze probíhá automaticky. Základní nastavení je provedeno v aplikačním kontextu, kde informuji Springu o umístění balíčku ve kterém má se mají hledat příslušné anotace. Aplikační kontext obsahuje ještě nastavení detektorů pohledu a umístění zdroje zpráv. Nalezneme zde i nastavení Hibernate a JDBC, kde je řečeno jaký dialekt a jaká databáze bude použita pro aplikaci. Podrobnější nastavení Hibernate je umístěno ve složce `/config` v souboru `hibernate.cfg.xml`, kde jsou namapovány entity s kterými bude Hibernate pracovat. Entity jsou pomocí Hibernate definovány jako javovské třídy, ke kterým můžeme přistupovat obvyklým způsobem. Místo pokládání JDBC dotazů zavoláme metodu třídy ona bez ohledu na databázový server danou operaci provede. Implementace entity je následující

```
@Entity
@Table(name = "users")
public class User {
    private Long id;
    private String username;
    private String name;
    private String surname;
    private String description;
    private String password;
    private String role;

    @Column(name = "id_user")
     public Long getId() {
          return id;
     }

     public void setId(Long id) {
          this.id = id;
     }
```

Na prvním řádku je vidět použití anotace[1]. Teto anotace oznamuje Springu, že se jedná o definici entity a tato entita je reprezentována databázovou tabulkou users. Anotace @Columns mapuje sloupce databázových tabulek na proměnné.

Nad touto entitou je implementováno DAO rozhraní (Data Access Object) – objekt přistupující k datů. Toto rozhraní je implementováno pro úplné oddělení persistentních dat od uživatelské části aplikace, tak aby splňovalo kritéria MVC návrhu. Jsou v něm obsaženy základní metody pro práci s databázemi (BaseDao) a rozšířené metody pro potřeby systému (UserDao). Příklad metody z rozhraní UserDao, metoda vrací uživatele podle jeho uživatelského jména.

```
public User getUserByUsername(String username);
```

UserDao a BaseDao jsou implementovany v UserDaoImpl a BaseDaoImpl, tím jsem docílil dostatečné abstrakce nad DAO rozhraním. Nyní můžu implementaci změnit třeba na použití String JDBC knihovny místo Hibernate, ale DAO rozhraní bude pořád stejné. Příklad UserDaoImpl

```
public User getUserByUsername(String username) {
        List<User> list = (List<User>)getHibernateTemplate().find
           ("FROM User user WHERE user.username = ?", username);
        if (!list.isEmpty() && list.size() == 1) {
            User user = (User)list.get(0);
            return user;
        } else if (list.size() > 1) {
           throw new IllegalStateException("Aplikaci obsahuje
                      dualitu username: " + username);
        }
        return null;
    }
```

Nad DAO rozhraním jsou implementováni manažeři. Manažeři slouží jako prostředníci mezi DAO a kontroléry. Kontroléry v Spring MVC modul by měly být navrženy tak, aby neobsahovaly žádnou aplikační logiku, aby pouze zpracovávaly vstupy z webového uživatelského rozhraní a připravovaly výstupy pro toto rozhraní. Proto musíme implementovat manažery aby právě tuto aplikační logiku vykonávali. Implementace manažera jež získává informace o uživateli podle jeho uživatelského jména.

```
public User getUserByUsername(String username) {
        if (username == null) {
            return null;
        } else {
```

1 Anotace jsou na začátku uvozeny znakem '@'

```
            return getUserDao().getUserByUsername(username);
        }
    }
```

Implementace kontrolérů jsou v balíčku `skola.controllers`. Použitím anotací oznámím Springu, že implementovaná třída je kontrolér a na jakou URL adresu je tento kontrolér namapovaný. Následující kód ukazuje `LoginController`, který se stará o obsluhu přihlášení klienta do systému.

```
@Controller
@RequestMapping("/login.html")
@SessionAttributes("login")
public class LoginController {
     …
     @Autowired
     UserManager userManager;
@RequestMapping(method = RequestMethod.GET)
    public String showForm(HttpServletRequest request, ModelMap
     map) throws Exception {
     LoginForm form = new LoginForm();
     map.addAttribute("login", form);
     return formView;
    }
@RequestMapping(method = RequestMethod.POST)
    public String processSubmit(HttpServletRequest request,
     @ModelAttribute("command") LoginForm form, BindingResult
     result, SessionStatus status) {…}
}
```

Kód není úplný vzhledem k jeho rozsahu, a ani nepovažuji za nutné ho celý znázorňovat. Na vypsaném kusu kódu chci ukázat, jakým způsobem je pomocí anotací nastaven kontrolér, jak jsou předávány data přes formulář a jakým způsobem jsou volány metody. Anotací `@Controller` je Springu řečeno, že se jedná o kontrolér a pomocí `@RequesMapping` je tento kontrolér namapován na URL adresu zakončenou `login.html`. Anotace `@SessionAttribute` vytvoří prostor do kterého se nám uloží data z formuláře, abychom je měli k dispozici po odeslání. Anotace `@Autowired` zpřístupní pro kontrolér beanu, jenž je uložená v paměti aby s ní kontolér mohl pracovat. Posloupnost vykonání kódu v kontroléru je řízen anotaci `@RequestMapping`. V souvislosti s logováním jsem narazil na problém kódování do MD5. V případě Zend i Rails se

jednalo o jednoduchou metodu jenž převedla řetězec na MD5 hash. V případe Springu je to trochu složitější a nalezení řešení zabralo nemálo času.

```
String md5pass = null;
    try {
          MessageDigest md5 = MessageDigest.getInstance("MD5");
          md5.update(form.getPassword().getBytes());
          BigInteger hash = new BigInteger(1, md5.digest());
          md5pass = hash.toString(16);

     } catch (NoSuchAlgorithmException nsae) {  }
```

Z ukázek implementace vzorové webové aplikace je vidět, že framework Spring je poměrně složitý a rozsáhlí systém. To již vyplývá z faktu že se jedná o kompletní aplikační framework a tudíž má velmi propracovanou strukturu. Právě pro tyto vlastnosti je často vybírán jako implementační nástroj pro větší a rozsáhlejší systémy.

# 5 Průběh a vyhodnocení testování

V této poslední kapitole se zabývám testováním vytvořených aplikací podle kritérií jenž jsem popsal v kapitole 3. V první řadě přistoupím k testování výkonnosti každého z frameworků, neboť tyto testy jsou objektivní a bezesporu vypovídají o velmi důležité vlastnosti daného nástroje na tvorbu webových aplikaci. Následně na to zhodnotím jakým způsobem se mi z daným frameworkem pracovalo a kolik času mi zabralo pochopení jeho logiky a práce s ním. Toto měření je velmi neobjektivní a samozřejmě názory na danou problematiku se mohou individuálně rozcházet. Přesto považuji za užitečné toto zhodnocení uvést, neboť může být pro člověka, jenž se rozhoduje, který nástroj použit, důležité.

## 5.1 Měření výkonnosti

Měření výkonnosti jsem provedl za pomoci nástroje JMeteru, který byl vyvinut speciálně pro měření výkonnosti webových aplikací, avšak nabízí i možnosti dalšího využití mimo webové aplikace. Pro docílení co pokud možná nejobjektivnějšího měření jsem testovací program rozdělil na dvě části, klientskou a serverovou (tuto možnost JMeter nabízí). Serverovou část jsem pustil na systému, kde běží webové aplikace, abych klientskou částí JMeteru nezatěžoval tento systém a tak nenarušoval objektivnost naměřených dat.

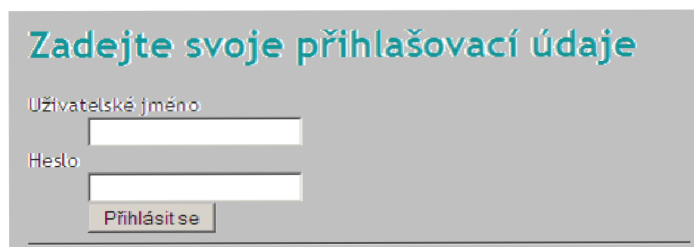
Výpočetní technika jenž byla použita pro účely testování v obou případech pracovala s operačním systémem MS Windows XP Professional SP3. Tyto systémy, však byly využívány pro běžný chod delší dobu, a proto je jejich výkon slabší, jak v případě méně zahlcených systémů. Konfigurace jednotlivých počítačů

- Počítač pro serverovou část
  - Procesor: AMD Atlon 3200 64-bit (Socket 939)
  - Paměť: 1GB, 2xDual DDR400 (4x256 MB)
  - HD: 320GB, (7200 otáček)
- Počítač pro klientskou část
  - Procesor: Intel Pentium M (1,7GHz)
  - Paměť: 512MB, SDRAM
  - HD: 40GB, (4200 otáček)

Počítače byly propojeny přes místní síť 100Mbit/s.

### Testované oblasti

1. Autentizace
    - Měření výkonnosti positivního přihlášení do systému
    - obrázek 5.1




*Ilustrace 5.1: Formulář pro logování uživatelů*

2. Výpis všech studentů
    - Výpis z tabulky users, kde atribut role má hodnotu member
    - obrázek 5.2

Nový student

Zpět

| Jméno | Příjmení | Třída | Login | |
|---|---|---|---|---|
| Wiliam | Shakespeare | 1.A | wilda | Edit Delete |
| Dante | Alighieri | 1.B | dante | Edit Delete |
| Giovanni | Bocaccio | 1.A | gigi | Edit Delete |
| Jaroslav | Foglar | 2.A | fogo | Edit Delete |
| Oscar | Wilde | 1.A | oscar | Edit Delete |
| Alan | Poe | 1.B | alan | Edit Delete |

*Ilustrace 5.2: Výpis všech studentů*

3. Nový domácí úkol
    - Založení nového domácího úkolu (INSERT)
4. Editace domácího úkolu
    - Editace stávajícího uživatele (UPDATE)
    - Měření UPDATE dotazu je zatíženo dotazem na vyhledání úkolu s daným id.
5. Výpis domácích úkolů
    - Výpis domácích úkolů spojen s tabulkou users pro jméno učitele (JOIN)
    - obrázek 5.3

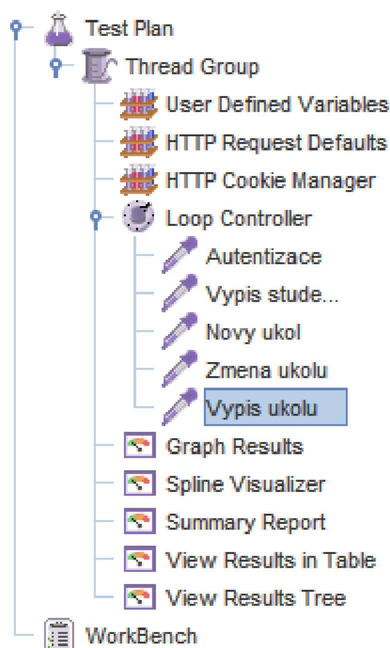
Přihlášen student: Wiliam Shakespeare
Odhlásit se.

| Třída | Název úkolu | Náplň úkolu | Pedagog |
|---|---|---|---|
| 1.A | BMI | Vytvořte program na výpočet BMI | Zdenek Smistik |
| 1.A | Pokus | Udělejte pěkný pokus. | Zdenek Smistik |

*Ilustrace 5.3: Výpis domácích úkolů pro studenta*

## Nastavení programu JMeter

Na serverovém počítači jsem aktivoval serverovou část programu JMeter a tím je nastavení na straně serveru ukončeno. Veškeré nastavení na měřeni výkonu aplikace je provedeno na klientském počítači. K tomuto nastavení jsem použil mód s podporou GUI (grafického uživatelského rozhraní) z důvodu přehlednosti a pro začátečníky pohodlnějšího nastavení, neboť je JMeter do jisté míry intuitivní a obsahuje v sobě postačující nápovědu. Strom testu je zobrazen na obrázku 5.4




*Ilustrace 5.4: Schéma nastaveni testovacího cyklu v aplikaci JMeter*

První položkou, kterou je potřeba nastavit je Thread Group, kde je zadáno kolik uživatelů bude současně testovat, kolikrát opakovat pro jednoho uživatele a jaké má být zpožděním mezi uživateli. Tato položka je kořenem celého testovacího cyklu. Další položkou je User Define Variables. Jedná se

o definici proměnných jenž uživateli při konstrukci testu usnadňují nastavení. V mém případě jsem vytvořil proměnou s názvem *url*, která obsahuje hodnotu základní URL adresy, neboť každý z dotazů v průběhu testu musí být touto adresou definován. V případě, že se změní URL, změnu aplikuji pouze v této proměnné a zbytek kódu zůstane prakticky nezměněn. S touto proměnou pracuje i další položka s názvem HTTP Request Defaults, kde jsou definovány standardní proměnné pro požadavek, jako například použitý webový server, což je v mém případě právě definovaná proměnná *url*. Neopomenutelnou položkou je HTTP Cookie Manager, který simuluje funkci cookies. V případě, že bych tuto funkci neaktivoval, nedošlo by k uložení přihlášeného uživatele do cookies a tudíž by neměl JMeter v testech přístup na testované stránky. Tím je ukončena etapa základního nastavení JMeteru.

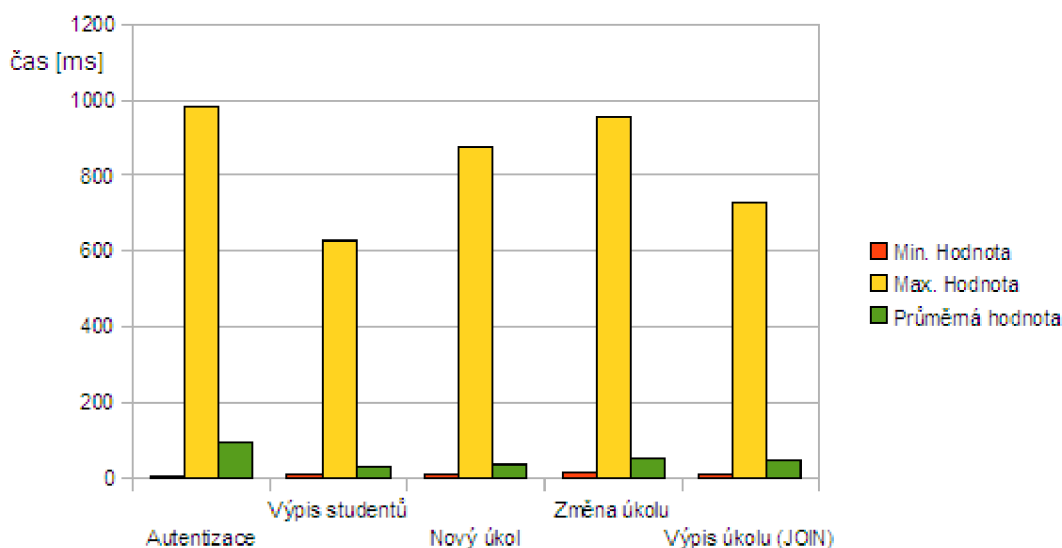
Další položkou je Loop Controller, který obsahuje celý testovací cyklus, přesněji obsahuje sadu HTTP požadavků, u kterých se provede měření doby vykonání. V prvním případě půjde o autentizaci, kde, jsem zadal na které URL adrese najde přihlašovací formulář, tento krok se provede i u všech ostatních HTTP požadavků, s tím rozdílem, že nepůjde o umístění přihlašovacího formuláře, ale o umístění dalšího testovacího úseku. Dále v případě přihlášení musím naplnit přihlašovací formulář a zvolit metodu jakou je formulář odesílán ke zpracování (POST). Analogicky se provede nastavení zbylých požadavků.

Poslední etapou nastavení JMeteru je nastavení tzv. posluchačů, nebo-li objektů pro prezentaci naměřených hodnot. V těchto položkách je uložen a určitým způsobem zobrazen výsledek testu. Mezi nejdůležitější patří View Result in Table a Summary Report. Kde v případě první zmíněné položky je zobrazen každý HTTP požadavek zvlášť, a je možné nahlídnout jakým způsobem byl požadavek zaslán a jakou odpověď na něj server odeslal, čímž je možné zkontrolovat, zda-li byl požadavek obsloužen tak jak měl. V případě položky Summary Report je zobrazena tabulka vyhodnocení celého testu s průměrnými naměřenými hodnotami. Právě tato tabulka je zdrojem veškerých níže zobrazených dat.

## 5.2 Zobrazení výsledků

### 5.2.1 PHP

| Testovaná operace | Počet uživatelů | Min. Hodnota | Max. Hodnota | Průměmá hodnota |
|---|---|---|---|---|
| Autentizace | 50 | 4 | 980 | 71 |
| Výpis studentů | 50 | 8 | 627 | 32 |
| Nový úkol | 50 | 11 | 879 | 35 |
| Změna úkolu | 50 | 14 | 958 | 53 |
| Výpis úkolu (JOIN) | 50 | 10 | 731 | 47 |

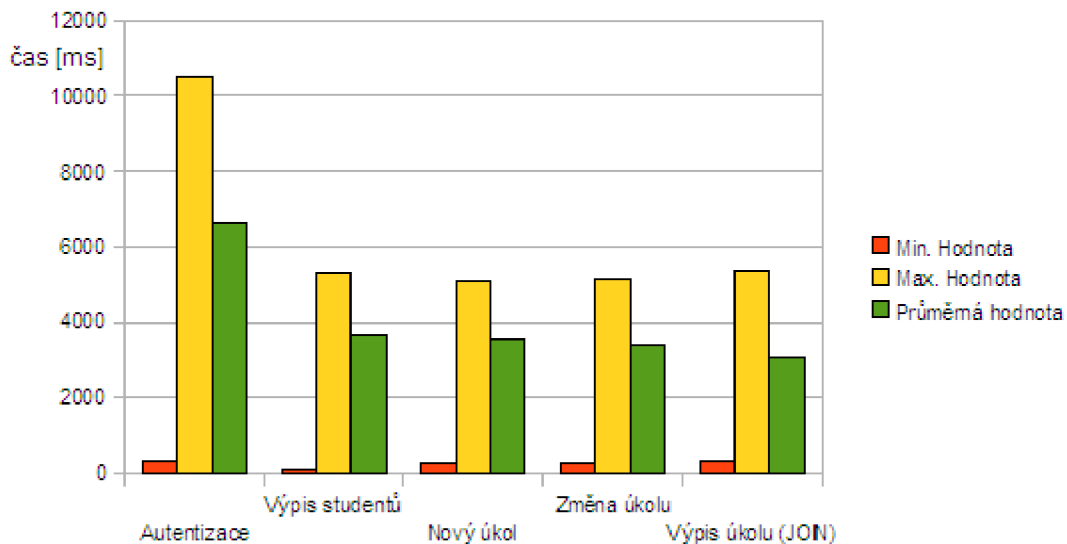



*Ilustrace 5.5: Graf naměřených hodnot u aplikace vytvořené v PHP*

Při měření výkonu aplikace, která je tvořena v PHP, se nejedná o výkon tvůrčího nástroje, ale v podstatě o výkon databáze, nad kterým je systém postaven. PHP nenabízí žádnou abstrakci nad prací s databází a proto nedochází ke zpomalování komunikace vlivem vnitřní logiky PHP. Tímto měřením jsem tedy získal nejlepší možný výsledek testu, jehož je možno dosáhnout na dané konfiguraci. Jak je vidět z grafu, dochází k velkým odchylkám v maximálních a minimálních hodnotách testu. Jelikož jsou průměrné hodnoty podstatně blíže minimálním hodnotám (v průměru do 50ms), je zřejmé že vysoké hodnoty nejsou způsobeny vlivem této aplikace. Vysoké hodnoty bych přisuzoval nahodilému zatížení systému, vlivem chodu jiné paralelní aplikace, jež běží na serveru. Vyšší hodnotu dosahuje pouze část autentizace, kde se spotřebovaný čas dělí mezi práci s databází a ukládání sezení.

### 5.2.2 Zend Framework

| Testovaná operace | Počet uživatelů | Min. Hodnota | Max. Hodnota | Průměmá hodnota |
|---|---|---|---|---|
| Autentizace | 50 | 313 | 10513 | 6612 |
| Výpis studentů | 50 | 133 | 5302 | 3651 |
| Nový úkol | 50 | 255 | 5095 | 3553 |
| Změna úkolu | 50 | 277 | 5171 | 3405 |
| Výpis úkolu (JOIN) | 50 | 324 | 5339 | 3102 |

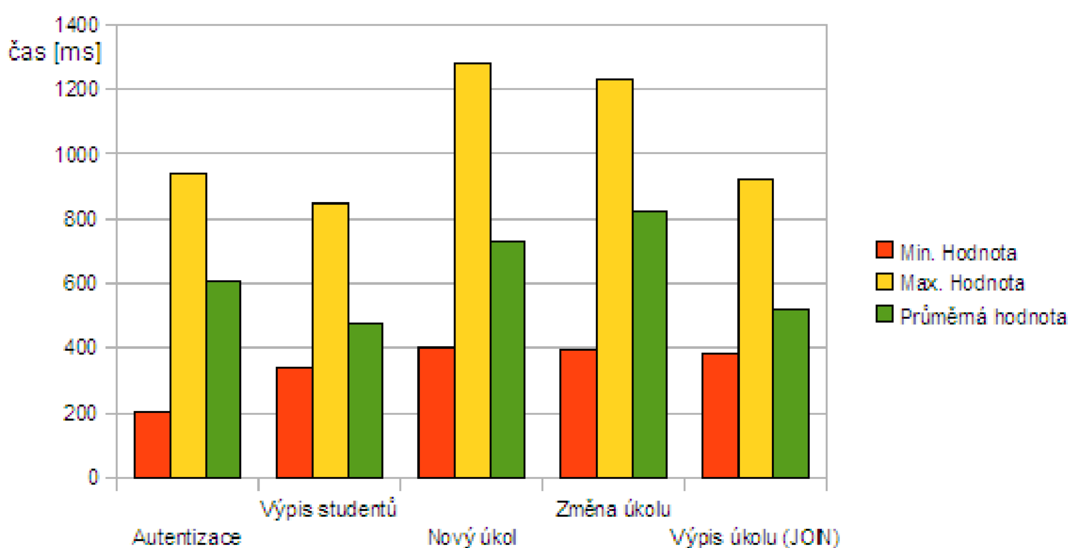



*Ilustrace 5.6: Graf naměřených hodnot u aplikace v Zend Frameworku*

Zend Framework nevychází v testu moc chválihodně. Jak je vidět z tabulky (grafu) autentizace uživatele v průměru zabere více jak 6s. Tato hodnota je při používání v běžném provozu absolutně nepřípustná, málo kdo by používal systém do kterého by se musel hlásit a čekat na odezvu 6 sekund. Navzdory tomu není Zend zatracen, existuji Zend akcelerátory, které několikanásobně zvyšují rychlost frameworku a tím ho staví na pole standardních nástrojů na tvorbu webových aplikací co se výkonu týče.

### 5.2.3 Ruby on Rails

| Testovaná operace | Počet uživatelů | Min. Hodnota | Max. Hodnota | Průměrná hodnota |
|---|---|---|---|---|
| Autentizace | 50 | 205 | 941 | 607 |
| Výpis studentů | 50 | 340 | 850 | 478 |
| Nový úkol | 50 | 401 | 1282 | 730 |
| Změna úkolu | 50 | 396 | 1233 | 826 |
| Výpis úkolu (JOIN) | 50 | 382 | 920 | 520 |

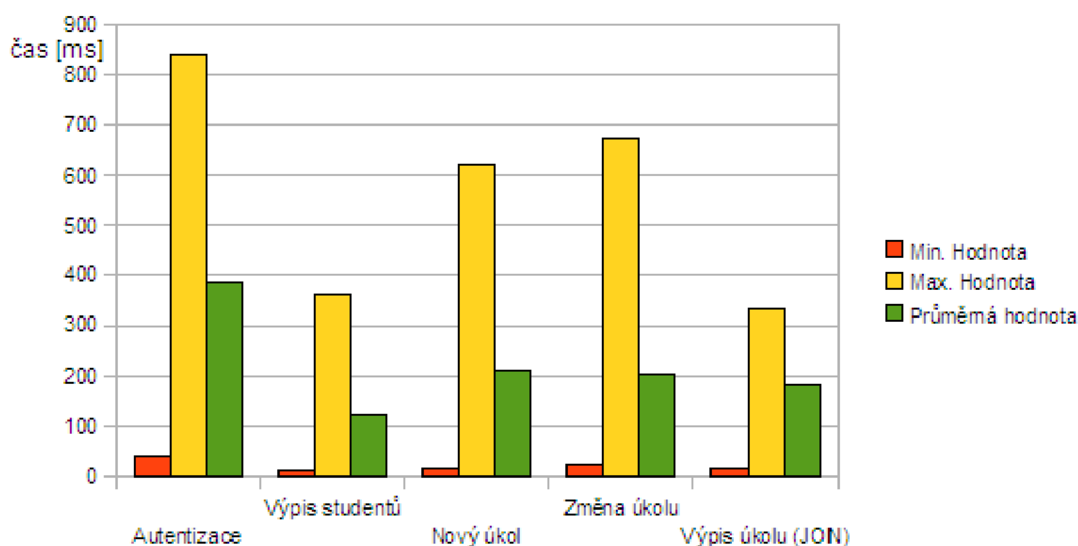



*Ilustrace 5.7: Graf naměřených hodnot u aplikace v Ruby on Rails*

Aplikace vytvořená v Ruby on Rails nijak nevyniká svojí výkoností, ale ani není na kritickém propadu použitelnosti. To co činí Ruby on Rails tak populárním, nebude výkonnost vytvořených aplikací, ale spíše forma s jakou elegancí se aplikace vytvářejí. Veliký výkon spíše Rails předvádí v generování kódu, jenž lahodně svědčí oku, které se dívá jak se aplikace rozvíjí a dostává rychle svoji logickou funkčnost.

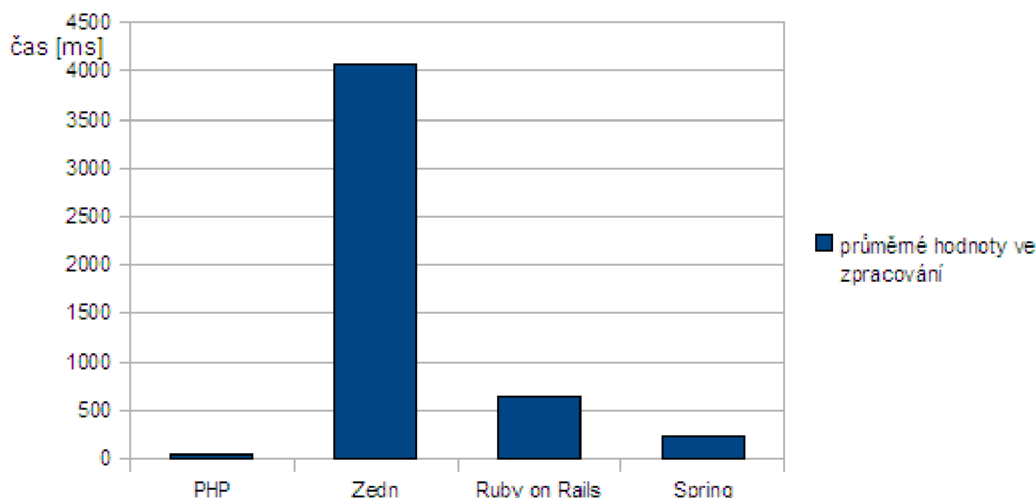
### 5.2.4 Spring Framework

| Testovaná operace | Počet uživatelů | Min. Hodnota | Max. Hodnota | Průměrná hodnota |
|---|---|---|---|---|
| Autentizace | 50 | 42 | 841 | 387 |
| Výpis studentů | 50 | 12 | 362 | 122 |
| Nový úkol | 50 | 17 | 622 | 213 |
| Změna úkolu | 50 | 23 | 671 | 204 |
| Výpis úkolu (JOIN) | 50 | 15 | 334 | 183 |




*Ilustrace 5.8: Graf naměřených hodnot u aplikace v Spring Framework*

Spring Framework je na tom s výkonem trochu lépe, avšak maximální hodnoty jsou občas velmi vysoké, což nejspíše souvisí s poměrně velikou náročností na chod serveru, na kterém přibyl běžící Tomcat. Spring framework se velmi často používá na rozsáhlejší systému, kde je vidět, že svojí úlohu zastane velmi dobře. Avšak spíše než pro svůj výkon je vybírán pro svojí komplexnost a metodiku implementace.




*Ilustrace 5.9: Celkové vyhodnocení v konkurenčním boji*

# 5.3 Pohled uživatele

Porovnávání frameworku z hlediska uživatele je, jak jsem již řekl, velice subjektivní záležitost a proto jí nebudu věnovat tolik pozornosti jako v případě měření výkonu vyhotovených aplikací. Důležitým bodem tohoto testu je informace, že do doby vzniku této práce jsem neměl o žádném z vybraných frameworků žádné hlubší informace, pouze podvědomí o jejich existenci. Z tohoto údaje vyplývá, že s každý z frameworků začínal s čistým štítem, a tudíž nebudu nadsazovat ani jednomu z nich z předchozích znalostí. Snad jen Zend a Spring měli výhodu, že jsou postaveny na mě známém jazyce a tím pádem se některé záležitosti v řešení problému a zapadnutí do logiky usnadnili.

## 5.3.1 Zend Framework

Práce se Zendem je vyloženě příjemná. Mezi veliké klady tohoto frameworku bych uvedl rozsáhlou komunitu programátorů, která velice pomůže při uváznutí na nějakém problému. Dalším aspektem je i dostatek literatury, jak v originálním podání na stránkách Zend Frameworku, tak v českém podání na fóru určeným pro komunitu Zendu. Dále se na českém poli pohybuje veliká škála podařených web blogů založených nad touto tématikou.

V implementační části jsem se setkal pouze s problémem v oblasti autentizace a autorizace, kde jsem pocítil menší absenci příkladů a návodů, jakým způsobem pracovat s třídami jenž se starají o tuto problematiku. Na druhou stranu, je to pravděpodobně způsobeno, ne plně kvalitní implementací těchto tříd. Po pochopení logiky komunikace je programování příjemné a programátor

si velice rychle zvykne na způsob jakým je aplikace tvořena. Dalším působivým aspektem je poměrně malá a nenáročná konfigurace systému pro práci s tímto nástroj. V podstatě se programátor stará jenom o potřebné knihovny, které aplikace využívá a v případě jejího přesunu na jiný server bude fungovat bez jakýchkoliv úprav, neboť málo který server se potýká s absencí PHP. V podstatě jediným úskalím je výkon takto vytvořené aplikace, který se ale dá dohnat akcelerátory.

Jednoduše řečeno pro začínající programátory webových aplikací, kteří chtějí vytvářet slušné webové aplikace s použitelným kódem, který se dá použít i pro rozsáhlejší systémy, je Zend tím správným nástrojem.

### 5.3.2 Ruby on Rails

Framework jenž nabírá velikou popularitu, která je do jisté míry opodstatněná. Z vlastní zkušenosti, můžu říci, že programování pomocí tohoto nástroje mě bavilo a rád jsem se prokousával řádkami kódu, jenž vznikali jakoby sami přímo mě pod rukama. V případě že chce někdo začít programovat webové aplikace, jak pro domácí, tak i komerční účely a nemá zkušenosti s žádným jiným nástrojem na tvorbu webu. Tak jemu bych velice doporučil právě Rails.

Na Rails jsou podle mého mínění nejatraktivnější generátory kódu, které velice šetří programátory jak psaní kódu tak čas nad ním strávený. V případě těchto generátorů je úskalí v tom, že jsou nastaveny do nějaké implicitní hodnoty, která se může s verzí Rails měnit a pak je potřeba nastavit jak s vygenerovaným kódem pracovat. Dalším úskalím je pluralizace. Rails rozumí pouze anglicky a tak používání českých výrazů pro programovaní zbytečně zatěžuje programování neboť, každá odchylka od konvence, je potřeba konfigurovat. Pluralizace znamená že Rails vyžaduje názvy databázových tabulek v množném čísle (users), k této příslušné tabulce generuje kontrolér v jednotném čísle (user). V případě literatury je množství a úroveň informací uspokojivé. Jistá absence je literatura v českém jazyce a vůbec informace v českém jazyce. Výhodou však jsou krátké video šoty, které jsou k nahlédnutí na stránkách Rails, kde jsou předvedeny vybrané akce na určitou tematiku. Do jisté míry se jedná o marketingové snímky, kde se snaží předvést rychlost a lehkost práce s Rails. A jistých případech,pokud se nechceme lišit od konvence tomu opravdu tak je. Nevýhodou je že externí servery většinou Rails nerozumí (neakceptují jeho logiku a soubory). Je tudíž potřeba tyto servery konfigurovat (pomocí FastCGI skriptů). Což není příjemná práce a nese sebou často spousty problému. Hlavně v případě konfigurace pod MS Windows, kde je značná absence informací jak toto nastavení provést. A leckdy nezbývá nic jiného než metoda pokus, omyl.

Abych nějakým způsobem shrnul dojem z Rails, můžu jenom doporučit. Přesto, že se setkáte ve vývoji s nepříjemnostmi na kterých strávíte nemálo času, vývoj je přesto velice příjemný a s každou napsanou řádkou se stáváte více a více nakloněn tomuto frameworku.

### 5.3.3 Spring Framework

V neposlední řadě framework Spring. Jak je z celé práce patrno, jedná se o velmi rozsáhlí framework, který bych asi nezvolil na vývoj jednoduché aplikace, jako tomu bylo v tomto případě. Vezmu-li jenom v úvahu, jak velké množství parametrů se musí nastavit, aby se Spring uvedl do chodu. Na druhou stranu nabízí spoustu možný rozšíření a jeho působnost sahá daleko za rámec web aplikací. Dostupnost literatury je v tomto ohledu podstatně slabší než v předchozích případech. Kvalitní literatura je z většiny pouze v anglickém jazyce a v počtu stran více než odpudivém. V podstatě je logika tvorby velice podobná logice v Zend Frameworku. S tím rozdílem, že Zend za sebou skýtá skriptovací jazyk PHP a Spring čistě objektový jazyk Java. Z tohoto vyplývají i rozdíly mezi nimi.

Spring má hlavní využití při implementaci větších projektů neboť vývoj aplikací je velice přehledný s velmi propracovanou úrovní abstrakce. Snaží se velmi striktně odlišovat jednotlivé vrstvy MVC navzájem od sebe, což je dobrým klíčem v týmovém vývoji aplikací.

### 5.3.4 Shrnutí

Z dosažených zkušeností a naměřených dat, usuzuji, že každý z uvedených frameworků nalezne svoje uplatnění a své příznivce. Frameworky Zend a Rails jsou svým způsobem, co se týče kvality použití, na stejné úrovni. Pro Rails hovoří velmi dobře propracované generátory kódu a pro Zend zázemí známého prostředí jazyka PHP. Nad těmito frameworky se tyčí framework Spring, který svojí robustností výše zmiňované převyšuje, avšak pro jednorázové menší aplikace bych volil snažší cestu a tou je jeden z předcházejících typů.

Srovnání jednotlivých frameworků je vidět v tabulce 1.

**Srovnání jednotlivých vývojových nástrojů**

| Nástroj | Čas potřebný k nastudování | Dostupnost informací v českém jazyce | Úspora psaného kódu | Hardwarová náročnost | Celková doba práce |
|---|---|---|---|---|---|
| PHP | 8 hodin | Velmi dobrá | Velmi slabá | Nízká | 38 hodin |
| Zend | 39 hodin | Velmi dobrá | Dobrá | Nízká | 62 hodin |
| Ruby on Rails | 47 hodin | Postačující | Velmi dobrá | Střední | 80 hodin |
| Spring | 72 hodin | Slabá | Dobrá | Vyšší | 120 hodin |

*Tabulka 1: Celkové hodnocení vývojových nástrojů*

# 6 Závěr

Cílem této práce bylo seznámení se s moderními technologiemi na vývoj webových aplikací a zhodnocení jejich vlastností. Tento cíl byl splněn. Pro dodržení veškerých bodů zadání jsem v prvé řadě nastudoval literaturu na téma tvorba webových aplikací. Na základě konzultací s konzultantem mé práce jsem vybral a nastudoval jednotlivé nástroje na tvorbu webových aplikací a výsledky studia shrnul v kapitole 2). Při práci s těmito nástroji jsem se zaměřil na podstatné vlastnosti, jež jsem si definoval v kapitole 3). Tyto vlastnosti jsem porovnal a vyhodnotil výsledky v kapitole 5). Zadáním bylo specifikováno implementovat aplikaci v jednom z vybraných nástrojů, však na doporučení konzultanta, pro zkvalitnění výsledného měření a porovnání, jsem zvolit vzorovou aplikaci a implementoval ji v každém ze zvolených frameworků. Implementační postupy a kritické sekce ve vývoji jsou uvedeny v kapitole 4).

Výsledkem této práce je porovnání jednotlivých frameworků, kde je vidět, jak si stojí frameworky v oblasti použitelnosti a efektivnosti. Výstupní údaje vypovídají o tom, že použitím frameworku při vývoji aplikace dochází, ve výsledném produktu, ke ztrátě výkonu. Avšak ztráta výkonu nepřekračuje neúnosnou mez a všechny frameworky mají reakční čas kolem 500ms. Což je únosná oběť, proti modularitě, přehlednosti a příjemnosti programování.

Tato práce dává možnost budoucím tvůrcům webových aplikací si na základě zde uvedených informací vybrat, jaký nástroj pro ně má smysl studovat a posléze používat. Pro mohutnější splnění tohoto cíle, by zde měli být zahrnuty i další nástroje zaměřené na tuto tématiku. Jako například CakePHP, Nette, Struts 2 a další. Avšak z důvodu rozsahu celé práce, nebylo možné tuto škálu nástrojů pojmout. S tímto je spojena i aplikace jiných databázových serverů, jako například PostgreSQL, SQLite a porovnání jejich vlastností. Ve všech uvedených, neobsažených tématech jde spíše jen o alternativy k vybraných v této práci, které jsou postavené na podobném základu a jejich uvedení by bylo pouze rozšířením pracovních nástrojů.

Pevně věřím, že tato práce je dobrým základním bodem pro tvorbu na poli webových aplikací.

# Literatura

[1] Porovnání komponentových a akčních webových frameworků, 22.12.2008
http://vyvojari.oxyonline.cz/porovnani-komponentovych-akcnich-webovych-frameworku

[2] Zend Framework, 25.12.2008
http://framework.zend.com/

[3] Vávra, V.: *Vlastův-blog,* 25.12.2008
http://vavru.cz/category/php/zend-framework/

[4] Holzner, S.: *Začínáme programovat s Ruby on Rails,* Computer Press, Brno, květen 2007, ISBN: 978-80-251-1630-2

[5] Objektově orientovaný jazyk RUBY, 28.12.2008
http://nb.vse.cz/~zelenyj/it380/eseje/xjunt04/OOJRuby.htm#1

[6] Ruby – A Programmer's Best Friend, 28.12.2008
http://www.ruby-lang.org/en/

[7] Walls, C.; Breidenbach, R.: *Spring in action*, Manning Publications Co., 2005, ISBN 1-932394-35-4

[8] Matulík, P.; Páral, T., *Moderní JEE™ technologie a nástroje,* 30.12.2008
http://morosystems.cz/java/index.php

[9] The Spring Framework - Reference Documentation, 30.12.2008
http://static.springframework.org/spring/docs/2.0.x/reference/

**Příloha A**

# Elektronický formát této práce

Na přiloženém CD je k dispozici elektronická podoba této práce, konkrétně:

- zdrojový kód této práce ve formátu ODF,
- tato práce ve formátu PDF.

Příloha B

# Zdrojové kódy vytvořených programů

Na přiloženém CD jsou k dispozici zdrojové kódy k vytvořeným programům v této práci, konkrétně:

**PHP**

- zdrojové kódy z vytvořené vzorové aplikace v PHP jsou k dispozici v adresáři /PHP.

**Zend Framework**

- zdrojové kódy z vytvořené vzorové aplikace v Zend jsou k dispozici v adresáři /Zend.

**Rubyon Rails**

- zdrojové kódy z vytvořené vzorové aplikace v Rails jsou k dispozici v adresáři /Rails.

**Spring Framework**

- zdrojové kódy z vytvořené vzorové aplikace ve Springu jsou k dispozici v adresáři /Spring.