VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

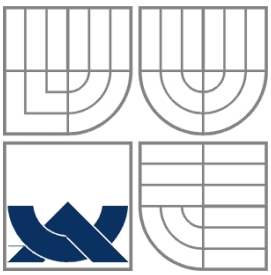
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM AJAX KOMPONENT PRO FRAMEWORK YII

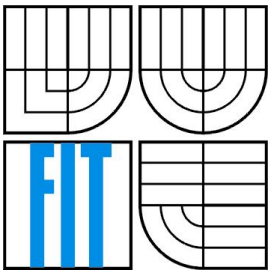
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MAHR
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM AJAX KOMPONENT PRO FRAMEWORK YII

AJAX COMPONENT SYSTEM FOR FRAMEWORK YII

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN MAHR

VEDOUCÍ PRÁCE
SUPERVISOR

ING. LUKÁŠ MÁČEL

BRNO 2010

Abstrakt

Bakalářská práce se zabývá webovými technologiemi PHP, HTML, JavaScript, AJAX a knihovnami jQuery a Yii framework. Výstupem práce je zhodnocení těchto technologií a vývoj rozšiřujících modulů pro knihovnu jQuery a PHP Yii framework. Tato rozšíření dohromady poskytují prostředí pro vývoj komponent uživatelského prostředí, které jsou schopny komunikovat navzájem prostřednictvím technologie AJAX a dovolují tak vytvářet dynamické webové stránky.

Abstract

The bachelor thesis deals with the web technologies PHP, HTML, JavaScript, AJAX, jQuery library and Yii framework. Outcome of this work is to evaluate these technologies and the development of an extension module for jQuery library and Yii framework. These extensions together provide the environment for the development of user interface components that are able to communicate with each other via AJAX technology and allows to create dynamic Web pages.

Klíčová slova

PHP, Yii, framework, JavaScript, AJAX, komponenty, MVC, Web

Keywords

PHP, Yii, framework, JavaScript, AJAX, components, MVC, Web

Citace

Martin Mahr: Systém AJAX komponent pro framework Yii, bakalářská práce, Brno, FIT VUT v Brně, 2010

Systém AJAX komponent pro framework YII

Prohlášení

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

.....
Martin Mahr
19.5.2010

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Lukáši Máčelovi za pomoc, ochotu, názory, věcné připomínky a především trpělivost.

Obsah

1 Úvod

Webové stránky na bázi jazyka HTML jsou ze své podstaty statické. Stránky na práci uživatele reagují vždy stejným způsobem a to znovu-načtením stejné webové stránky nebo přesměrováním na jinou stránku. Východiskem pro vývoj dynamických webových stránek, které dokáží reagovat na činnost uživatele asynchronně bez znovu-načtení celé stránky, je technologie AJAX. Bohužel většina současných knihoven a frameworků pro jazyk PHP tuto technologii nepodporují vůbec, nebo jejich architektura příliš vychází ze statického modelu webových stránek. Cílem této bakalářské práce je proto vytvořit v jazycích PHP a JavaScript programový modul, který umožní vytváření komponent webového uživatelského prostředí, které budou navzájem komunikovat pomocí technologie AJAX a na webové stránce tak vždy dojde k překreslení pouze těch komponent, které změnili svůj obsah. Tyto komponenty jsem pojmenoval Ex komponentami. Mnou navržený modul bude poskytovat prostředky na tvorbu takovýchto komponent, zajistí komunikaci mezi klientskou a serverovou částí komponenty, komunikaci mezi komponentami a správu jejich vnitřního stavu.

Druhá kapitola této práce se věnuje historii a stručné charakteristice jazyka PHP. Obsahuje srovnání s jinými konkurenčními technologiemi a podkapitoly se věnují vybraným aspektům jazyka PHP, které jsou důležité pro další výklad.

Třetí kapitola pokračuje v představení serverových technologií a představuje PHP framework Yii. Vysvětluje jeho architekturu a podrobněji části tvořící jeho základní kameny.

Ve čtvrté kapitole jsou představeny klientské technologie JavaScript a knihovna jQuery. Konec kapitoly vysvětluje pojem technologie AJAX, jakožto spojení doposud probraných technologií a jazyků.

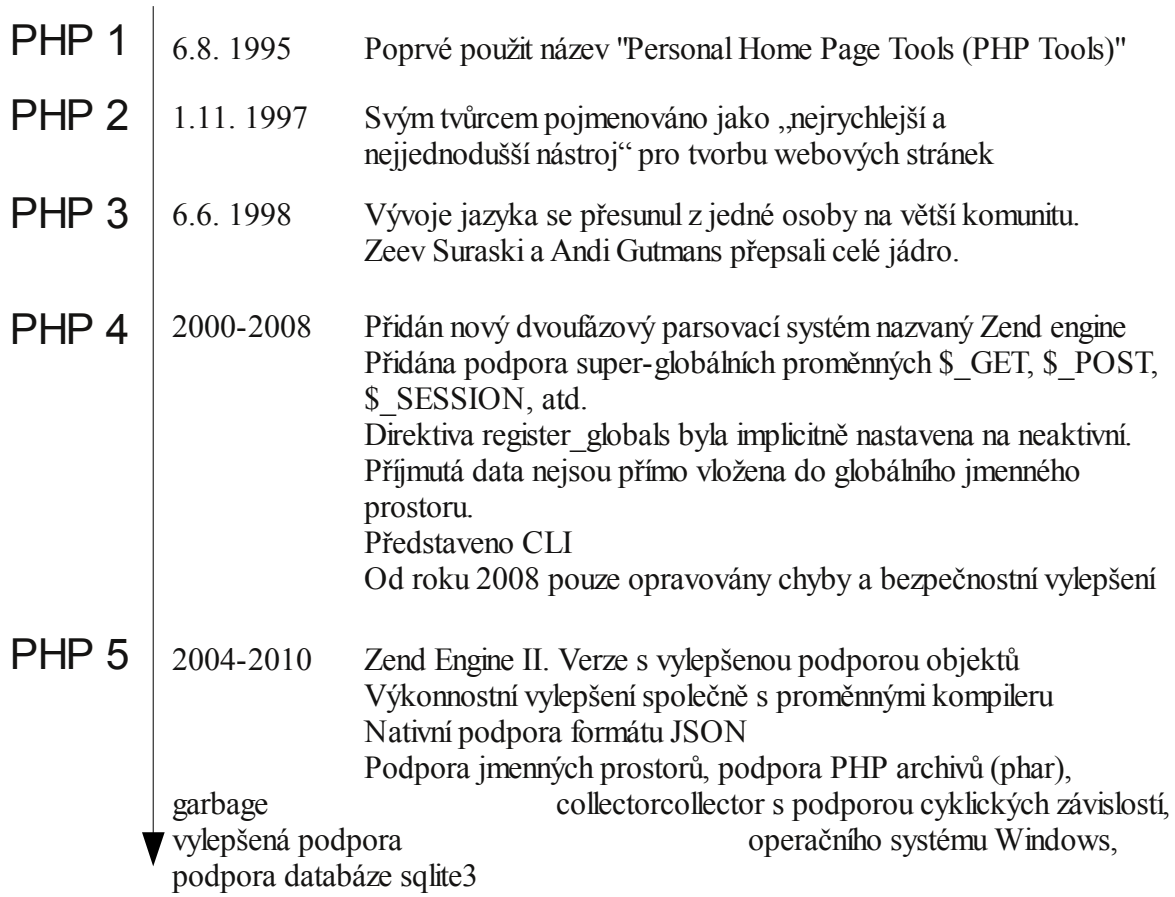
V páté kapitole je popsán systém AJAX komponent pro framework Yii, jaká je jeho architektura, popsány základní třídy a jakým způsobem lze tento systém použít pro tvorbu vlastních komponent.

2 Jazyk PHP

Vznik internetu se datuje rokem 1990, kdy ve švýcarském výzkumném centru CERN vytvořili značkovací jazyk HTML(HyperText Markup Language). Jazyk HTML je určen pro tvorbu hypertextových (provázaných) dokumentů. Spolu s technologiemi HTTP a URL tvoří základ internetu. HTTP (HyperText Transfer Protocol) je přenosový protokol, který zajišťuje přenos HTML dokumentů ze serveru ke klientovi. URL (Uniform Resource Locator) slouží ke specifikaci umístění zdrojů informací na internetu.

Tyto tři technologie umožňovali pouze získání dat ze serveru a jejich zobrazení v prohlížeči HTML dokumentů. Prvním krokem k interaktivním internetovým stránkám byla technologie CGI (Common Gateway Interface), která umožňovala klientské požadavky na www server obsloužit externím programem. Toto řešení je ovšem velice pomalé. Dalšími technologiemi pro tvorbu interaktivních internetových stránek byli SSJS (Server Side JavaScript) společnosti Netscape a ASP (Active Server Pagers) společnosti Microsoft. Obě technologie používají techniku vkládání serverových částí kódů přímo do HTML dokumentu a jejich následnému interpretování. Výhodou je vyšší rychlost a jednodušší použití. Obě tyto technologie byly komerční a svázané s webovým serverem. V případě ASP navíc pouze s operačním systémem Microsoft Windows. [1]

PHP vzniklo jako reakce na tyto technologie. Jedná se o open-source projekt, který je nezávislý na platformě. PHP představuje zástupce dynamicky interpretovaných jazyků. Jeho syntaxe je inspirována jazyky jako C, Java a Pascal. Výhodou jazyka PHP je to, že veškeré nástroje pro vývoj a provoz aplikací jsou dostupné zdarma obvykle jako open-source. Díky oblíbenosti jazyka je také na internetu dostupné obrovské množství návodů, knihoven a otevřených projektů, ze kterých se lze jazyk naučit.



Ilustrace 2.1: Stručná historie vývoje jazyka PHP. Převzato z [2]

2.1 Objektové programování v jazyce PHP

Již před příchodem verze 5 byla v PHP začleněna podpora pro objektově orientované programování (OOP). Bohužel nedosahovala kvalit jiných objektově orientovaných jazyků a v praxi nebyla příliš použitelná. Ve verzi 5 vývojáři přepsali celou tuto část jazyka, která pracuje s objekty a výrazně zrychlili a rozšířili použitelnost objektů. Ve světě PHP je verze podstatný zlom v možnostech programování. Umožňuje to řešit podstatně rozsáhlejší problémy a lepší spolupráci v týmech. PHP verze 5 se tak stalo plnohodnotným objektově orientovaným jazykem, ve kterém můžeme využívat všech objektově orientovaných konstrukcí jako je abstrakce, dědičnost, zapouzdření, rozhraní atd.

```php
class myIterator implements Iterator {
    private $position = 0;
    private $array = array(
        "firstelement",
        "secondelement",
        "lastelement",
    );

    function rewind() {
        $this->position = 0;
    }

    function current() {
        return $this->array[$this->position];
    }

    function key() {
        return $this->position;
    }

    function next() {
        ++$this->position;
    }

    function valid() {
        return isset($this->array[$this->position]);
    }
}

$it = new myIterator;

foreach($it as $key => $value) {
    var_dump($key, $value);
    echo "\n";
}
/*
 *  Výstupem bude:
 *
 *  firstelement
 *  secondelement
 *  lastelement
 */
```

Ilustrace 2.2: Příklad jednoduché třídy. Převzato z [3]

Na ilustraci 2.2 je zobrazen příklad třídy *myIterator* implementující rozhraní *Iterator*. Na ilustraci je vidět, jak v jazyce PHP implementovat třídu, její atributy a metody. Třídy implementující rozhraní *Iterator* lze použít v konstrukci *foreach* místo pole.

2.2 Reflexe

PHP 5 přineslo nové API pro tzv. reflexi. Reflexe umožňuje reverzním inženýrstvím získat informace o třídách, rozhraních, metodách a funkcích. Umožňuje také získat dokumentační komentáře funkcí, metod a tříd [4]. Výhodné je zejména možnost získat z komentářů parametry, na základě kterých je možné se k nim různě chovat. V rámci bakalářské práce využívám reflexy tříd, pro čtení jejich parametrů v dokumentačních komentářích k označení význačných atributů. Toho využívá komponenta *persister* popsaná v kapitole 5.3.

```php
// Vytvoření instance třídy reflexe
$reflection = new ReflectionClass($object);

// Načtení všech atributů objektu
$properties = $reflection->getProperties();

foreach($properties as $property){
    // Získání dokumentačního komentáře atributu
    $docComment = $property->getDocComment();

    // Ověření přítomnosti parametru
    if(preg_match('/@param/', $docComment)){
        // Získání jména atributu
        $name = $property->getName();

        // Získání hodnoty atributu
        $value = $property->getValue($object);

        $this->doSomething($name);
    }
}
```

Ilustrace 2.3: Ukázka práce z třídou reflexe.

Na ilustraci 2.3 je část kódu, na které je ukázáno, jak lze pracovat s třídou reflexe.

2.3 Rozhraní ArrayAccess

PHP 5 umožňuje přistupoval k objektům jako k poli. Aby objekt umožňoval tento přístup musí implementovat rozhraní *ArrayAccess*.

```php
interface ArrayAccess  {
        /**
         * @param offset
         */
        abstract public function offsetExists ($offset) {}

        /**
         * @param offset
         */
        abstract public function offsetGet ($offset) {}

        /**
         * @param offset
         * @param value
         */
        abstract public function offsetSet ($offset, $value) {}

        /**
         * @param offset
         */
        abstract public function offsetUnset ($offset) {}
}
```

Ilustrace 2.4: Rozhraní ArrayAccess. Více příkladů a ukázek lze najít na [5]

Metoda *offsetExists* vrací logickou hodnotu *boolean* podle toho, jestli v poli existuje daný klíč. Tato metoda je volána při použití funkcí *isset()* a *empty()*. Metoda *offsetGet()* slouží pro získání hodnoty z pole podle jejího klíče. Tato metoda je také volána při použití funkce *empty()*, pokud metoda *offsetExists()* vrací logickou hodnotu *true*. Metoda *offsetSet()* slouží pro nastavení hodnoty v poli podle zadaného klíče. Metoda *offsetUnset()* slouží pro odstranění hodnoty z pole. Implementace rozhraní *ArrayAccess* je výhodná zejména kvůli zpřehlednění zápisu.

2.4 PHP frameworky

Framework je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji. [6]

PHP frameworkem rozumíme sadu PHP skriptů, které poskytují vývojáři prostředí pro vývoj webových aplikací a/nebo nabízejí sadu knihoven (tříd), které jsou schopny řešit běžné a časté problémy.

Frameworky přebírají nejčastější problémy dané problematiky a umožňují tak vývojáři soustředit se pouze na své zadání. Ve světě jazyka PHP frameworky nejčastěji řeší tyto problémy:

- architekturu aplikace (MVC, MVP apod.),
- směrování uživatelských požadavků na aplikační komponenty,
- ošetření uživatelských vstupů (GET, POST a COOKIES), ochrana proti útokům na stránky skrze tyto vstupy,
- objektově orientovaný přístup k databázi,
- internacionalizace výstupu,
- tvorba formulářů a jejich validace,
- autentizace a autorizace uživatelů,
- vyrovnávací paměť (cache)
- zpracování chyb a výjímek.

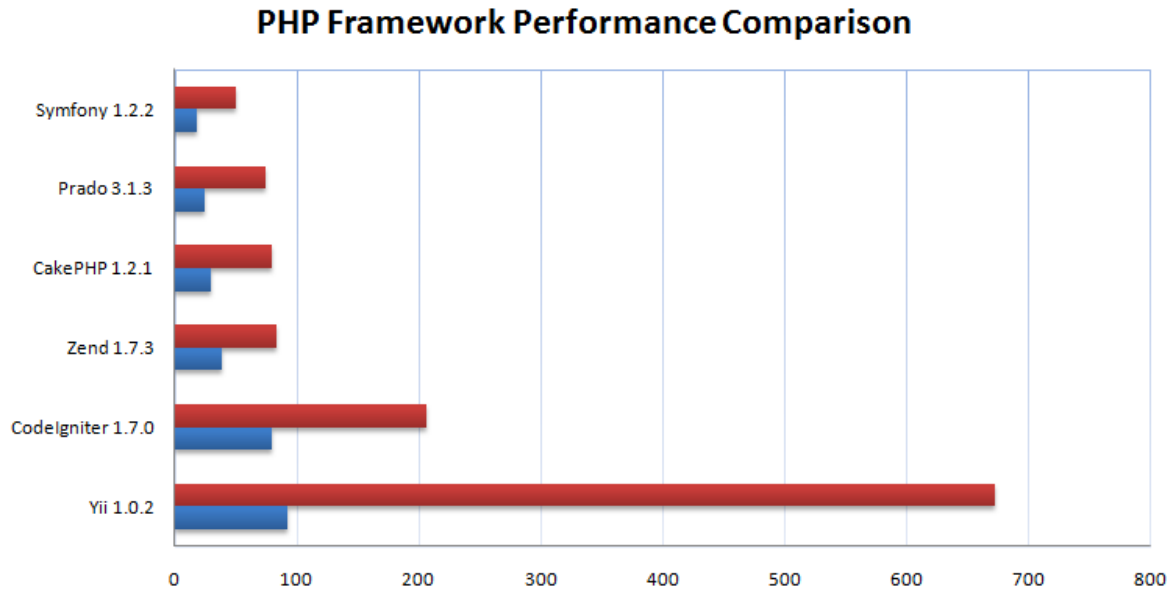
Mezi známé PHP frameworky patří:

- Symphony,
- Prado,
- CakePHP,
- Zend,
- CodeIgniter,
- Yii,
- Nette.

3 Yii framework

Yii je vysoce výkonný komponentě zaměřený PHP framework primárně určený na vývoj velkých webových aplikací. Autorem tohoto frameworku je Qiang Xue momentálně působící na Michigan State University. Historie frameworku Yii započala prvního ledna 2008, jde tedy o velice mladý projekt. Autor Qiang Xue byl také vývojářem úspěšného frameworku Prado a na základě svých zkušeností se rozhodl vytvořit nový framework. V září 2008 vydal první alfa verzi, framework byl oficiálně vydán 3. prosince 2008. Yii tedy myšlenkově vychází především z frameworku Prado, z kterého přejímá především komponentní systém, událostmi řízené paradigma, databázovou vrstvu a mnoho dalších součástí. Z ruby on Rails přebírá myšlenky upřednostnění konvence před konfigurací a návrh databázové vrstvy typu Active Record. [7]

Yii framework je velice výkonný v porovnání s jinými frameworky, což dokládá test zveřejněný na internetových stránkách projektu:



	Yii 1.0.2	CodeIgniter 1.7.0	Zend 1.7.3	CakePHP 1.2.1	Prado 3.1.3	Symfony 1.2.2
RPS with APC	673	206	83	79	75	50
RPS without APC	93	79	38	29	24	18

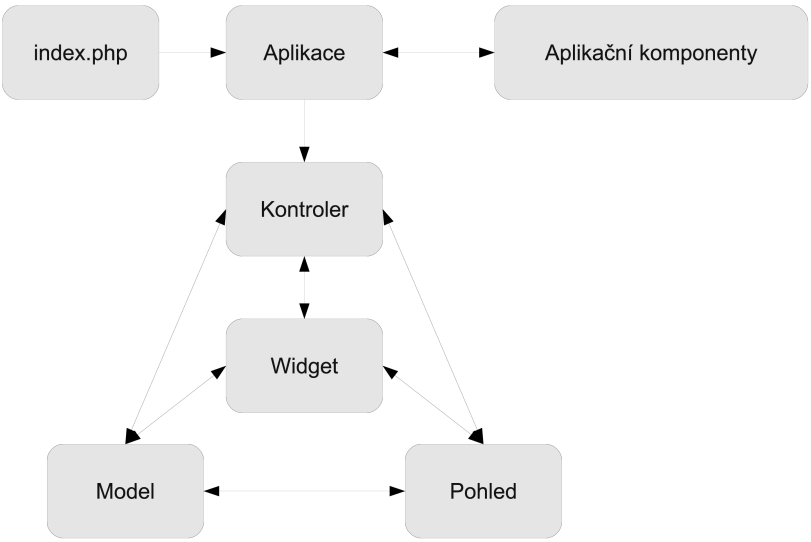
Ilustrace 3.1: Porovnání výkonnosti PHP frameworků. Převzato z [7]

Na ilustraci 3.1 je graf výsledků porovnání výkonnosti některých známých PHP frameworků. Všechny frameworky byly testovány na stejném počítači se stejným nastavením. Pro každý framework byla naprogramována co možná nejjednodušší aplikace, která měla pouze spustit framework a zobrazit stránky s výpisem „Hello world“. Tímto testem se testuje především rychlost

odezvy frameworku. V grafu jsou vyneseny hodnoty počtu obsloužených požadavků za jednu sekundu (Requests Per Second). Pro každý framework jsou v grafu vyneseny dva sloupce. Modrý sloupec je výkon frameworku s vypnutým rozšířením PHP jazyka APC (Advanced Php Cache) a červený sloupec se zapnutým.

3.1 Architektura

Yii je vyvíjeno čistě pro PHP 5 a není zpětně kompatibilní s PHP 4. Nevýhodou je tedy nemožnost tento framework použít na některých starších hostingových serverech, které podporují pouze PHP 4. Výhodou je čistě objektový návrh využívající nejnovějších vlastností PHP 5. Z pohledy architektury je Yii navrženo podle návrhového vzoru Model-View-controller (MVC), který odděluje prezentační vrstvu od logické a datové. Návrh aplikací pomocí modelu MVC značně ulehčuje vývoj a umožňuje také rozdělení práce mezi více vývojářů aniž by mezi jejich prací mohli vznikat konflikty.



Ilustrace 3.2: Architektura Yii frameworku

Na ilustraci 3.2 je znázorněna architektura Yii frameworku. Vstupním bodem celé aplikace je soubor index.php. Ten spustí jádro frameworku, které potom spouští další součásti frameworku a uživatelské třídy a komponenty. Podle návrhového vzoru MVC jsou základními vrstvami aplikace kontroléry, modely a pohledy. Ty jsou v diagramu zobrazeny jako trojúhelník s vrstvou Widget uprostřed. Widget vychází z kontroléru. Má svoji vlastní logiku, předpokládá se, že bude pracovat s modely a bude mít nějaký výstup. Nelze na něj ovšem směřovat uživatelské požadavky jako na kontrolér. Do vrstvy Widget patří třídy reprezentující nějakou ohraničenou součást webové stránky,

která se opakuje napříč webovou aplikací v různých kontrolérech a pohledech. Poslední součástí aplikací frameworku Yii jsou aplikační komponenty. Aplikační komponenty jsou třídy poskytující nějaké služby. Přistupuje se k nim skrze volání jádra Yii frameworku, který zajišťuje jejich spouštění a udržuje vždy jen jednu vytvořenou instanci.

Vstupním bodem aplikace je soubor *index.php*. Všechny uživatelské požadavky jsou směrovány na tento soubor. Tento vstupní script je velice jednoduchý. Pouze načte soubor s bázovou třídou Yii frameworku a vytvoří instanci aplikace voláním metody *Yii::createWebApplication*, které jako parametr předá cestu k souboru s konfigurací aplikace. Příklad souboru index.php je znázorněn v příkladu 3.3

```
$yii=dirname(__FILE__).'/framework/yii.php';
$config=dirname(__FILE__).'/protected/config/main.php';

require_once($yii);
Yii::createWebApplication($config)->run();
```

Ilustrace 3.3: Soubor index.php spouštějící YII framework

Po spuštění aplikace se spustí obsluha chybových stavů a komponenta *autoloader*, která zajišťuje automatické načítání php souborů s požadovanými třídami. Nikde v aplikaci tedy není nutné používat metody *include* nebo *require*. Při použití třídy, která není ještě načtena, *autoloader* automaticky najde a načte příslušný soubor s třídou. Dalším krokem v běhu aplikace je načtení základních aplikačních komponent, načtení konfiguračního souboru a inicializace aplikace, jde především o načtení dalších aplikačních komponent a jejich inicializace podle konfigurace. Aplikační komponentou se rozumí komponenta systému, která nějakým způsobem řídí běh aplikace nebo nesouvisí s žádným konkrétním kontrolérem a poskytuje nějakou službu. V rámci systému je aplikační komponenta globálním objektem dostupným skrze třídu aplikace:

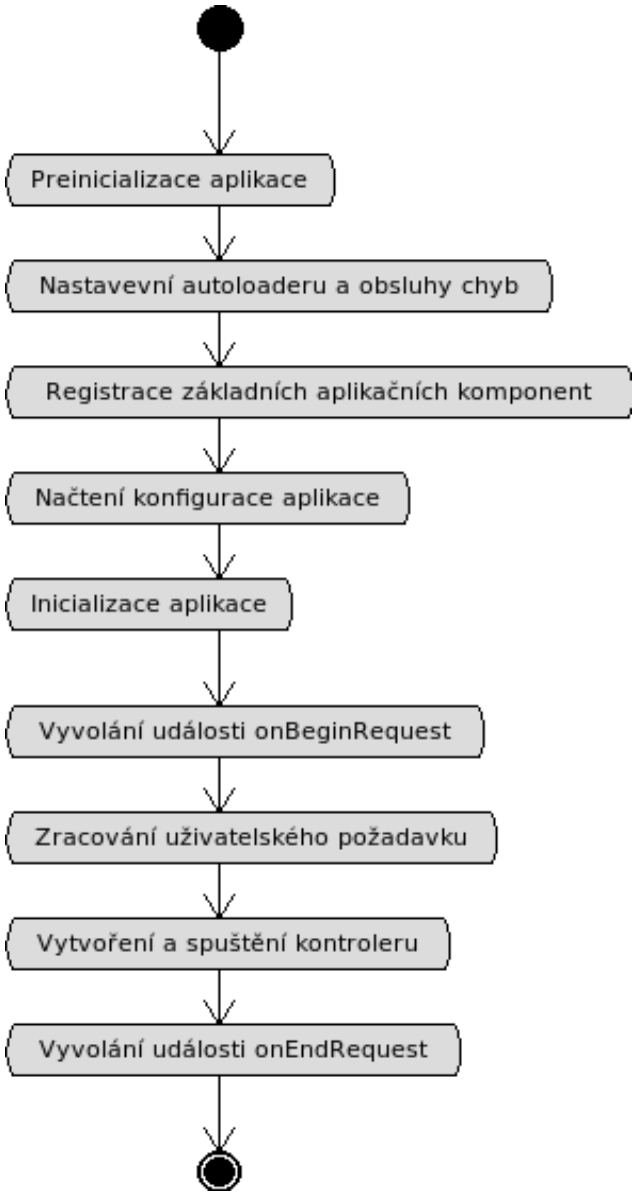
```
Yii::app()->applicationComponentName;
```

Ilustrace 3.4: Přístup k aplikační komponentě

Yii framework načítá některé aplikační komponenty automaticky. Konkrétně jde o komponenty:

- urlManager – zpracovává uživatelský požadavek na aplikaci (vstupní URL)
- request – Obaluje informace o požadavku
- session – Zajišťuje správnou práci s vedením sezení

- assetManager – Zajišťuje publikování statického obsahu jako jsou obrázky, soubory skriptů apod.
- user – Zajišťuje identifikaci uživatele a persistenci informací o uživateli v rámci sezení
- themeManager – spravuje různé vzhledy aplikace
- authManager – implementuje autorizaci operací v rámci systému (Role-Based Access Control)
- clientScript – Zajišťuje publikaci souborů klientských skriptů a souborů kaskádových stylů



Ilustrace 3.5: Životní cyklus aplikace v Yii frameworku

Po načtení všech aplikačních komponent je vyvolána systémová událost *onBeginRequest*. Touto událostí začíná zpracování uživatelského požadavku. Je volána aplikační komponenta *urlManager*, který na základě URL, ze které došlo ke spuštění aplikace, odvodí jméno kontroléru a jeho akce, která se bude spouštět. UrlManager může obsahovat různá pravidla založená na regulárních výrazech na základě kterých pracuje s vstupní URL. Jakmile je známo jméno kontroléru, aplikace vytvoří jeho instanci a spustí požadovanou akci. V akcích kontroléru leží celá aplikační logika. Po dokončení práce kontroléru je volána systémová událost *onEndRequest* a aplikace je ukončena.

3.2 Základní komponenta CComponent

Třída CComponent je bázová třída pro všechny komponenty. CComponent implementuje způsob definování a používání atributů a událostí [9].

3.2.1 Atributy

Pokud je atribut třídy veřejný, lze k němu v přistupovat přímo tímto způsobem:

```
$object->samleAttribute = 'test';
```

Ilustrace 3.6: Běžný přístup k atributu

K atributům třídy by se nemělo podle metodiky OOP přistupovat přímo ale pomocí metod třídy. To nás nutí používat metody pro nastavení a získání atributu třídy tzv. gettery a settery.

```
class SampleClass{
    private $attribute;

    public function getAttribute(){
        return $this->attribute;
    }

    public function setAttribute($value){
        $this->attribute = $value;
    }
}
```

Ilustrace 3.7: Ukázka setteru a getteru

Getter je metoda, která zpřístupňuje obvykle soukromý atribut třídy. Setter naproti tomu atribut třídy nastavuje na nějakou hodnotu. Třída CComponent rozšiřuje možnosti přístupu k atributům třídy a definuje zásady pro vytváření a práci s gettery a settery.

CComponent implementuje metody třídy __get a __set, které jsou volány při pokusu při přístup k jakémukoli atributu třídy. Třída CComponent tak může při přístupu k atributu vyhledat příslušný getter nebo setter a použít jej. Přístup k vnitřnímu atributu třídy tedy vypadá stejně jako kdyby byl veřejný, ale vnitřně se provolávají getter a setter. Tento přístup nám také umožňuje definovat atributy, které jsou pouze pro čtení nebo pouze pro zápis a virtuální atributy. Virtuální atribut je takový atribut třídy, který ve třídě nemá svou definici, ale pouze setter a/nebo getter. Může tedy jít např. o nějaký vypočítávaný atribut.

Setter musí začínat slovem *set* a pokračovat názvem atributu. Např. Pro atribut s názvem *count* se musí jmenovat *setCount()*. Obdobně se pojmenovávají gettery jen s tím rozdílem, že začínají slovem „get“.

```
// Čtení hodnoty atributu
// Kód:
$a = $component->text;
// je ekvivalentní s:
$a = $component->getText();

// Zápis hodnoty aatributu
// Kód:
$component->text='abc';
// je ekvivalentní s:
$component->setText('abc');
```

Ilustrace 3.8: Použití setterů a getterů

3.2.2 Události

Událost je metoda, jejíž název začíná slovem *on*. Název metody odpovídá názvu události [9]. Události se od běžných metod liší tím, že při jejich vyvolání na tuto událost můžou reagovat jiné objekty. Tyto objekty mohou zaregistrovat svou metodu jako obsluhu události (handler) a při vyvolání události se poté zavolají všechny zaregistrované obslužné metody v pořadí v jakém byly zaregistrovány. Obslužné události (handler) musí mít tuto signaturu:

```
function eventHandler($event) {}
```

Ilustrace 3.9: Signatura obslužné události

Jazyk PHP umožňuje vytváření odkazů na metody, což je základní předpoklad pro tvorbu událostí. Možnost vytvářet události a reagovat na ně ovšem jazyku PHP chybí. Třída CComponent tuto funkcionalitu přidává velice jednoduchým způsobem:

```
$component->onClick = $callback;

// nebo

$component->onClick->add($callback);
```

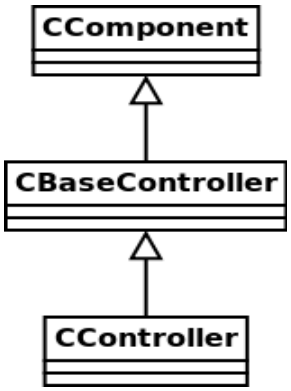
Ilustrace 3.10: Registrace obslužné události

Proměnná *$callback* musí obsahovat validní odkaz na funkci podle PHP5.

3.3 Kontrolér, aplikační logika

Aplikační logika aplikace je primárně uložena ve třídách nazvaných kontroléry (třída *CController*). Každý kontrolér je složen s funkcí, které zastupují jednotlivé akce, které uživatel může provést. Obvykle jde o zobrazení nějaké stránky, nebo provedení nějaké činnosti na serveru. Kontrolér sdružuje související činnosti na jedno místo. Např. *UserController* bude mít na starost činnosti související s uživateli apod.

3.3.1 Kontrolér



Ilustrace 3.11: Diagram tříd pro kontrolér

Kontrolér (třída CController) spravuje sadu akcí, které se zabývají odpovídajícími požadavky uživatelů. Skrze akce CController koordinuje tok dat mezi modely a pohledy. Když uživatel požaduje akci 'XYZ', CController provede jednu z následujících akcí:

1. akce založená na metodě: zavolá se metoda 'actionXYZ' pokud existuje

2. akce založená na třídě: vytvoří instanci třídy 'XYZ', pokud je třída nalezena v mapě tříd (specifikována pomocí metody actions()) a spustí akci.
3. provolá metodu missingAction(), která standardně vyvolá výjimku HTTP 404.

Pokud uživatel nedefinuje žádnou akci, kontrolér může mít definovánu výchozí akci pomocí atributu defaultAction.

CController může být nastaven tak, aby před a/nebo po volání akcí spouštěl tzv. filtry. Filtry před-zpracovávají uživatelský požadavek. Můžou inicializovat zachytávání výstupu, ověřovat práva pro spouštění akcí nebo kontrolovat, zda-li je akce spouštěna metodou get či post. Pokud jsou volány filtry až po provedení akce, mohou různě upravovat HTML výstup akce, nebo se dají využít pro výpis dodatečných ladicích a dalších informací. [11]

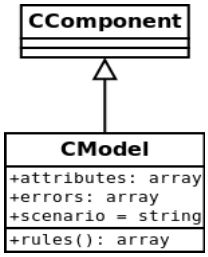
Kontrolér je vytvořen aplikací ve chvíli, kdy si uživatel vyžádá spuštění některé z jeho akcí. Provedení akce obvykle sestává ze získání a zpracování dat z modelů, předání dat pohledu a zobrazení nějakého pohledu.

```
class UserController extends CController{
    public $defaultAction = 'list';

    public function actionList(){

    }
}
```

Ilustrace 3.12: Implementace akce kontroléru a nastavení výchozí akce.

Na ilustraci 3.12 je zobrazen nejjednodušší příklad kontroléru. Při požadavku na vyvolání akce *user/list* kontrolér vyvolá svou metodu *actionList*. Pokud si uživatel vyžádá pouze spuštění kontroléru *user*, bude spuštěna akce *list*, protože je definována jako výchozí.

3.4 Model, datová vrstva



Ilustrace 3.13: Diagram tříd modelu

Model je abstraktní reprezentací dat s kterými aplikace pracuje Model je určen k tomu, aby obsahoval nějaká související data a k nim přidružené funkce. V Yii frameworku jsou k dispozici obecné třídy

pro modely a pro databázově orientované aplikace nabízí model typu *ActiveRecord*. Knihovna *ActiveRecord* umožňuje přistupovat k záznamů v databázových tabulkách jako k objektům, případně jako ke kolekci objektů, a umožňuje také práci s databázovými relacemi.

Základní třídou pro prezentaci dat je třída *CModel*. Pokud budeme mluvit o modelu, myslíme tím konkrétní třídu reprezentující nějaká data, nikoli pouze obecnou třídu. Každý model má svou zvláštní třídu, která definuje především atributy modelu a validační pravidla.

```
class User extends CModel{
    public $username;
    public $password;

    public function rules(){
        return array(
            array('username, password', 'required'),
        );
    }
}
```

Ilustrace 3.14: Jednoduchý model dat

Na ilustraci 3.14 je vidět příklad jednoduchého modelu, který reprezentuje model uživatele systému. Model obsahuje dva atributy: jméno a heslo uživatele. Model dále implementuje metodu *rules()*, která slouží k definici validačních pravidel. Bázová třída CModel obsahuje metodu *validate()*, která slouží k validaci dat modelu. Důležité je, že ke každému atributu modelu se váží validační pravidla, které se použijí při jeho validace a také případné chyby, které při validaci nastaly. Atributy bez definovaných validačních pravidla jsou považovány za nevalidní. Typické vytvoření nového záznamu a jeho validace je zobrazena na obrázku 3.15.

```
$user = new User;
$user->username = 'Admin';
$user->password = '12345';

if($user->validate()){
    $user->save();
}
```

Ilustrace 3.15: Práce s modelem

Na ilustraci 3.15 je ukázáno, jak lze vytvořit nový model uživatele, naplnit jej daty a poté uložit. Uložení by vždy mělo předcházet vytvoření validačních pravidel a volání metody *validate()*. Tím je zajištěno, že ukládaná data odpovídají definovaným předpokladům.

3.5 Pohled, prezentační vrstva

Prezentační vrstva, anglicky nazvaná *View*, pokrývá veškeré výstupy programu, nejčastěji jde o generování výsledného HTML kódu stránky. Pohledy jsou šablony výsledné HTML stránky, které obsahují jednak prostý HTML kód a pak také nějaký řídicí kód. Yii framework pro pohledy používá běžně PHP soubory. Výhodou toho přístupu je především rychlost a variabilnost použití. Yii framework používá pro vykreslení běžných stránek dva pohledy. První z nich slouží pro rozvržení(*layout*) stránky a obvykle obsahuje HTML hlavičku a prvky tvořící vzhled výsledné stránky. Druhým pohledem je již šablona konkrétní stránky, která se vkládá na určené místo v rozvržení. O vykreslení rozvržení a šablony je zodpovědný kontrolér. Kontrolér implementuje metody, které vykreslí danou šablonu v příslušném rozvržení nebo jen samotnou šablonu. To se může hodit, pokud nechceme vykreslovat celou stránku, ale jen určitou část. Typická akce kontroléru končí výpisem nějaké šablony, jak je ukázáno na obrázku 3.16.

```
class SiteController extends CController{

    public function actionIndex(){

        // ...

        $time = date('Y-m-d');

        $this->render('index', array(
            'title' => 'Homepage',
            'author' => 'Martin Mahr',
            'time' => $time,
        ));
    }
}
```

Ilustrace 3.16: Výpis pohledu z kontroléru

Všechny metody pro vykreslení mají stejnou strukturu parametrů. Prvním parametrem se udává název šablony a ve druhém se metodě předává asociativní pole parametrů, které budou šabloně dostupné. Šablony jsou organizovány do složek podle názvu kontroléru, proto stačí metodě *render()* předat pouze název souboru a ne celou cestu.

4 Klientské technologie

Tato kapitola se věnuje klientským technologiím webových projektů. Klientskými technologiemi se myslí ty technologie, které jejichž kód je spouštěn na straně uživatele webové služby. V kontextu mého projektu je zajímavý klientský jazyk JavaScript a knihovna (framework) jQuery napsaná v tomto jazyce.

JavaScript je objektově orientovaný skriptovací jazyk široce využívaný v různých programech. Jeho nejznámější použití je ovšem stále v oblasti HTML stránek. S rozvojem internetu se neustále víc aplikací přesouvá do webového prohlížeče. HTML samotné ovšem neposkytuje žádné mechanismy pro dynamický obsah ani interaktivitou s uživatelem, kterou běžně poskytují klasické aplikace. Na tomto místě nastupuje JavaScript, který umožňuje reagovat na chování uživatele a manipulovat s jinak statickou HTML stránkou.

Stejně jako vznikla potřeba pro PHP frameworky, i při tvorbě JavaScriptových programů se ukázalo, že samotný jazyk neposkytuje nástroje pro v praxi běžné postupy. Většina JavaScriptových frameworků poskytuje podporu pro výběr prvků dokumentu, manipulaci DOM, AJAX, animace apod. Velkou výhodou a motivací pro použití frameworku je také jejich optimalizace pro různé prohlížeče. Prohlížeče používají různé javascriptové interprety, mezi kterými se vždy najdou malé rozdíly, které javascriptové frameworky odstiňují.

4.1 Knihovna JQuery

Práci na knihovně jQuery začal v roce 2005 John Resig, který nyní pracuje pro Mozilla Corporation. Inspirován průkopníky v této oblasti, jako jsou Dean Edwards a Simon Willisonová, Resig dal dohromady sadu funkcí, které usnadňovali vyhledávání prvků ve webové stránce a přiřazovat k nim nějaké vlastnosti a chování. V lednu 2006, kdy poprvé představil svůj projekt veřejně, přidal ještě funkce pro úpravu DOM a základní animace. Dal mu jméno jQuery (query = dotaz), aby zdůraznil jeho podstatu, která tkví ve vyhledávání prvků ve stránce a působit na ně v jazyce JavaScript. Během několika málo let se jQuery rozrostl ve své sadě funkcí, zlepšil svou výkonnost a získal široké přijetí i na některých z nejoblíbenějších stránek na internetu. Zatímco Resig zůstává vedoucí vývojář projektu, jQuery vyrostl v pravý open-source projekt, který nyní může pochlubit jádrem špičkových programátorů stejně tak jako živou komunitou tisíců vývojářů. Knihovna jQuery může vylepšit vaše stránky nehledě na další pozadí vašeho projektu. Poskytuje širokou škálu funkcí, snadno naučitelnou syntaxi a robustní kompatibilitu v jediném kompaktním souboru. Navíc pro jQuery byly vyvinuty stovky rozšíření, což z něj dělá základní nástroj pro téměř každou příležitost. [11]

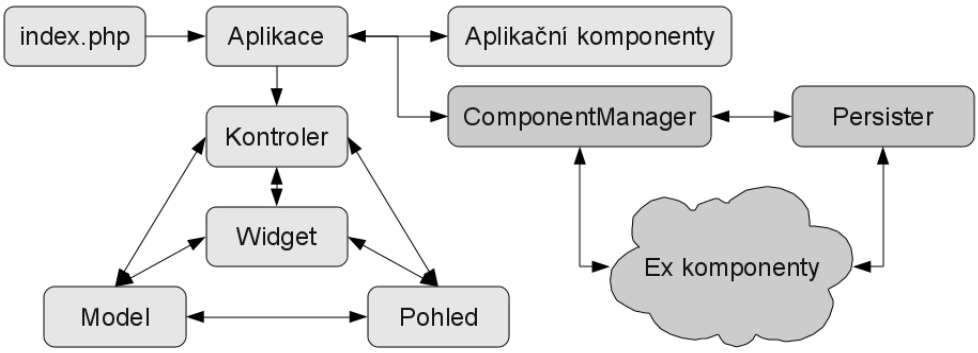
4.2 AJAX

AJAX (Asynchronous JavaScript And Xml) je obecné označení pro technologie vývoje webových aplikací, které dokážou měnit svůj obsah bez nutnosti jejich celého znovu-načtení [12]. K vývoji aplikací na bázi technologie AJAX se využívají technologie (X)HTML a CSS pro zobrazování stránek, DOM a JavaScript pro dynamické změny obsahu a funkci XMLHttpRequest pro asynchronní výměnu dat mezi klientem a serverem. Termín AJAX byl poprvé uveřejněn v dubnu 2005 v článku Jesse James Garretta [13], i když podobné technologie a myšlenky byli publikovány již dříve.

Výhodou použití technologie AJAX je odstranění nutnosti načítat celou stránku znovu při každé akci. Pokud například uživatel klikne na tlačítko přidělující hlas v anketě, v klasickém modelu WWW stránek se musí celá stránka načíst znovu, i když se jen aktualizují výsledky ankety. S využitím technologie AJAX je možné data přenést na pozadí a aktualizovat pouze část stránky zobrazující výsledky ankety.

5 Návrh rozšíření Yii frameworku

Při tvorbě HTML stránek se nejdříve pracovalo se statickými stránkami HTML, kde každé stránce odpovídal jeden zdrojový soubor. S vývojem serverových technologií se začaly stránky vytvářet dynamicky, ale pořád přetrvává myšlenka „internetové stránky“ podobná té statické, kdy jedné zobrazené stránce odpovídá nějaká jedna konkrétní programová jednotka. Může jít o prostou šablonu, funkci, třídu apod. Toto schéma se ale stává čím dál tím více omezující, protože stránky se dnes obvykle skládají z více samostatných interaktivních částí, které mohou být součástí mnoha stránek. Klasická hierarchická struktura stránek se mění spíše na síťovou a to i na úrovni logiky aplikace. Jednou z možností jak překonat toto schéma, je vnímat internetovou stránku jako sadu komponent, které se navzájem mohou ovlivňovat na základě činnosti uživatele v ideálním případě pomocí technologie AJAX. Bohužel dnešní webové technologie nebyly pro takovéto úlohy navrženy. Mým cílem je tedy vytvořit jeden modul pro PHP framework Yii a druhý pro javasciptový framework jQuery, které dohromady vytvoří platformu pro vývoj komponent. Tyto komponenty jsem nazval Ex komponentami. Hlavními požadavky na Ex komponenty jsou možnost komunikovat mezi sebou technologií AJAX a ponechat co možná největší část vývoje nových komponent na serverové straně a jazyce PHP.



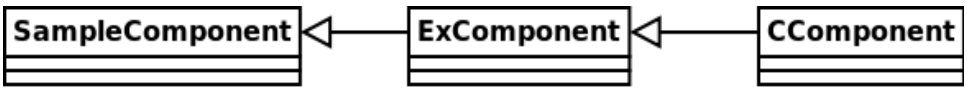
Ilustrace 5.1: Architektura Yii frameworku a Ex komponent

Mnou navržené rozšíření pro Yii framework se skládá z několika hlavních částí: *správce komponent*, *resolveru*, *persisteru* a *Ex komponent*. Na ilustraci 5.1 je znázorněno, jaký je vztah mezi mým rozšířením a Yii frameworkem. Správce komponent je aplikační komponentou a tvoří rozhraní mezi Yii frameworkem a komponentami.

5.1 Ex komponenty

Ex komponenty jsou objekty, které mají jednoznačně definovaný identifikátor, mohou mít atributy, metody a reagovat na události. K Ex komponentám se přistupuje pomocí identifikátoru pouze skrze správce komponent, který zajišťuje jejich vytváření, načítání a ukládání z persistentního zdroje. Každá Ex komponenta je reprezentována svou vlastní třídou, jejíž název odpovídá názvu komponenty

.



Ilustrace 5.2: Diagram tříd ukázkové komponenty

Jak je naznačeno na ilustraci 5.2, komponenta s názvem *SampleComponent* dědí od bázové třídy *ExComponent*, která komponentě dodává metody pro spolupráci se správcem komponent. Komponenty jsou typicky ukládány ve složce */protected/excomponents*, ovšem tato cesta se může změnit v nastavení správce komponent. Ex komponenty jsou až na několik výjimek běžné třídy. Odlišují se možností mít persistentní atributy, voláním jejich událostí přímo z klientské strany a propojením se správcem komponent.

Základním předpokladem pro správnou funkci Ex komponent je jejich schopnost uchovávat si svoje vybrané atributy po celý běh sezení uživatele. Komponenty mohou obsahovat mnoho atributů a ne vždy je potřeba všechny je uchovávat. Pro určení atributů, které je potřeba uchovávat v persistentním úložišti se využívá dokumentačních komentářů. Atributy komponent lze v dokumentačním komentáři označit parametrem „@persistent“. Takto označené atributy *persister* ukládá, ostatních si nevšímá, čímž šetří prostředky.

```
class SampleComponent extends ExComponent{
    /**
     * Tento atribut bude persister ukládat a načítat
     *
     * @persistent
     */
    public $atribute1;

    /**
     * Hodnota tohoto atributu bude ztracena při ukončení aplikace.
     */
    public $attribute2;

    ...
}
```

Ilustrace 5.3: Ukázka persistentních a nepersistentních atributů Ex komponenty

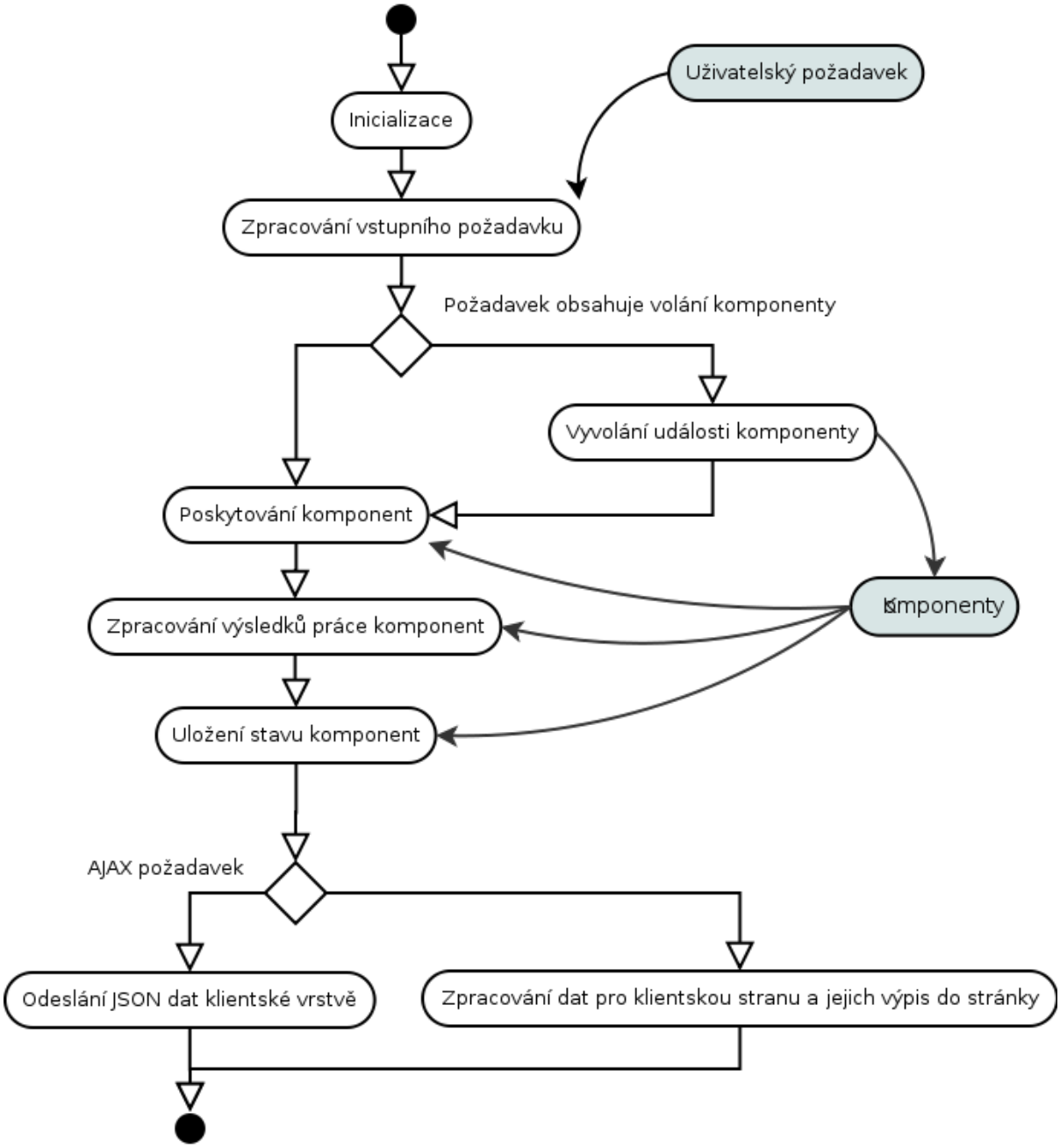
5.2 Správce komponent

Z pohledu architektury Yii frameworku je správce komponent aplikační komponentou, která se načítá při startu aplikace. Aby se správce komponent správně načetl, je potřeba v nastavení aplikace přidat složku modulu do importních cest, nastavit přístupu ke komponentě a důležité je také nastavit, aby se správce komponent načítal vždy při spuštění aplikace. Běžné aplikační komponenty se spouští až při jejich vyžádání, ale správce komponent je potřeba spustit vždy ihned po spuštění aplikace. Správné nastavení je znázorněno na ilustraci 5.4.

```
return array(
    'preload'=>array(
        'excomponents',
    ),

    'import'=>array(
        'application.extensions.excomponents.*',
        'application.extensions.excomponents.components.*',
        'application.extensions.jquery.*',
        'application.excomponents.*',
    ),

    'components'=>array(
        'excomponents'=>array(
            'class'=>'ComponentManager',
        ),
    ),
);
```

Ilustrace 5.4: Nastavení Yii frameworku pro použití Ex komponent

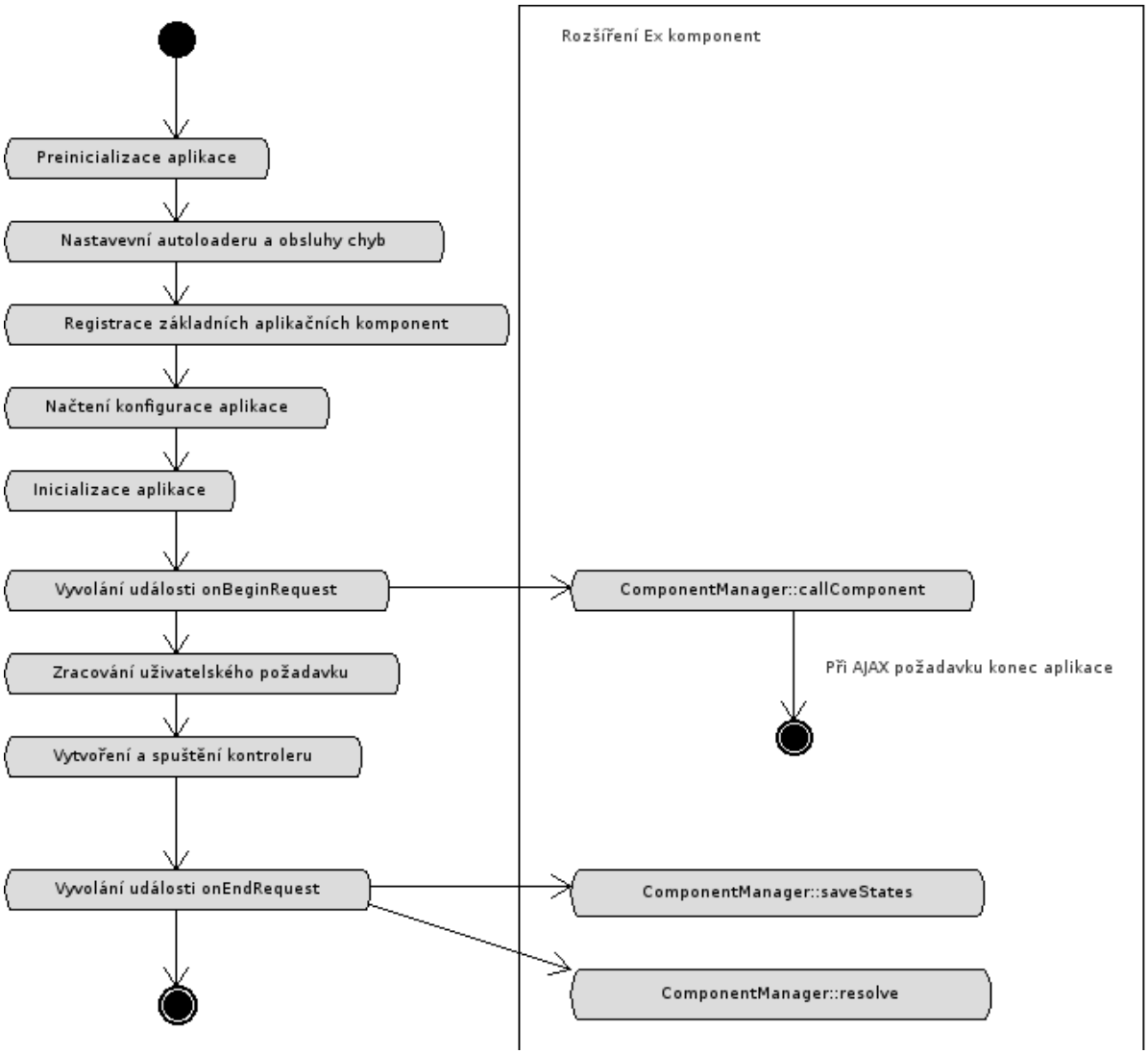
V sekci import jsou všechny potřebné cesty, kde se nacházejí třídy celého projektu. Základní složkou je složka 'application.extensions.excomponents.*', což ve struktuře Yii frameworku odpovídá cestě 'protected/extensions/excomponents/', kam se instaluje celý modul. V podsložce 'components' jsou uloženy bázové třídy pro běžně používané komponenty jaku jsou tlačítka, popisky apod. Do složky 'application.extensions.jquery' se instaluje pomocné rozšíření, které jsem vytvořil, které umožňuje v jazyce PHP pracovat s voláními knihovny jQuery velice podobným způsobem jako v jazyce JavaScript. Toto rozšíření je popsáno v kapitole 5.5. Poslední složkou je 'application.excomponents', kterou je třeba vytvořit v adresáři 'protected' a slouží pro ukládání uživatelských komponent. V sekci 'components' je zaregistrována komponenta *excomponents*, která je instancí třídy ComponentManager a tvoří jádro projektu. Je nezbytné uvést název aplikační komponenty i v sekci 'preload'. To zajistí, že při každém spuštění aplikace bude také inicializován správce komponent. Bez tohoto kroku by správce nebyl funkční.



Ilustrace 5.5: Životní cyklus správce komponent

Na obrázku 5.5 je znázorněn životní cyklus správce komponent. Při spuštění aplikace, ještě před zpracováním požadavku uživatele je správce komponent nečten a inicializován. Při svém spuštění inicializuje další nezbytné komponenty (Persister, Resolver) a načte si seznam použitých komponent. Po inicializaci se pokusí v požadavku uživatele volání komponenty. Pokud jej nalezne, volá událost požadované komponenty. Volaná komponenta provede nějakou činnost, ke které může potřebovat další komponenty, nebo svou činností ovlivňuje vnitřní stav dalších komponent. Přístup k dalším komponentám probíhá vždy skrze správce komponent. K žádné komponentě nelze přistoupit jinak,

než si ji vyžádat od správce komponent. Správce komponent také sbírá změny komponent a další data, které komponenty chtějí publikovat. Jakmile všechny komponenty ukončí svou činnosti, správce komponent zajistí uložení jejich vnitřních stavů a změnu výsledné HTML stránky. Ta může probíhat dvěma způsoby. Pokud byl požadavek proveden technologií AJAX, sesbírané data jsou vrácena klientské straně ve formátu JSON. Pokud se jednalo o běžné zobrazení stránky, kód komponent je standardně zobrazen na jejich místech v šablonách. Správce komponent je na několika místech spojen se strukturou Yii frameworku. Především jde o obsluhu událostí *onBeginRequest* a *onEndRequest*.



Ilustrace 5.6: Propojení Yii frameworku a správce komponent při zpracování požadavku

Na ilustraci 5.6 je znázorněno spojení Yii frameworku a správce komponent. Správce komponent se při inicializaci zaregistruje k obsluze událostí onBeginRequert a onEndRequest. Událost onBeginRequest je volána frameworkem ještě před zpracováním požadavku. Pokud by požadavek nezpracoval správce komponent, Yii framework by podle požadavku spustil příslušný kontrolér a

jeho akci, při které by došlo k výpisu stránky. Správce komponent se pokusí v požadavku najít parametr *do*. Tento parametr obsahuje identifikaci Ex komponenty a její události. Například pokud chceme zavolat událost *click* na komponentě *LoginButton*, v HTTP požadavku se musí objevit parametr *do=LoginButton/click*. Když správce komponent tento parametr najde, načte příslušnou komponentu a spustí její událost. Po spuštění události komponenty a jejím provedení je při běžném požadavku řízení aplikace vráceno zpět Yii frameworku. Při požadavku realizovaném technologií AJAX je běh aplikace ukončen. Při ukončení aplikace je volána událost *onEndRequest*. V obsluze této události správce komponent uloží aktuální stav komponent a zajistí předání dat o změnách komponent klientské straně.

5.3 Persister

O načítání a ukládání komponent se stará objekt nazvaný persister. Persister se stará o ukládání a načítání dat na nějaké persistentní úložiště, které je schopno data udržet alespoň po dobu sezení jednoho uživatele. Jak je vidět na ilustraci 5.7, kde je zobrazeno rozhraní persisteru, musí být persister schopen uložit a načíst seznam komponent, které jsou aktuálně v systému používány a načíst a uložit konkrétní komponentu.

```
/**
 * Rozhraní pro persister.
 * @author Martin Mahr <xmahrm00@stud.fit.vutbr.cz>
 */
/**
 * Persister se stará o ukládání a načítání komponent z nějakého persistentního
 * zdroje dat.
 * @author Martin Mahr <xmahrm00@stud.fit.vutbr.cz>
 */
interface IPersister{
    /**
     * Načte seznam komponent
     * @return array
     */
    public function getComponents();

    /**
     * Nastaví seznam komponent
     * @param array $components
     */
    public function setComponents($components);

    /**
     * Načte komponentu
     * @param ExComponent $component
     */
    public function loadComponent(ExComponent $component);

    /**
     * Uloží komponentu
     * @param ExComponent $component
     */
    public function saveComponent(ExComponent $component);
}
```

Ilustrace 5.7: Rozhraní persisteru

5.4 Resolver

Resolver je aplikační komponentou, která zajišťuje komunikaci mezi serverovou částí Ex komponent a klientskou stranou. Na straně serveru je resolver využíván správcem komponent, který resolveru předává seznam všech komponent a resolver rozhoduje, které z nich je potřeba překreslit. To rozhoduje na základě atributu Ex komponent *rerender*. Pokud je nastaven na logickou hodnotu *true* značí to, že komponenta byla při běhu aplikace změněna a je potřeba její HTML kód znovu vygenerovat a změny zaslat na klienta. Ex komponenty mohou resolveru navíc předávat příkazy v jazyce JavaScript, které budou na straně klienta provedeny a adresu URL na kterou bude stránka po přijmutí požadavku přesměrována.

```
ComponentManager::getResolver()->addCommand("$('#component').show();");
ComponentManager::getResolver()->setLocation('http://www.example.com');
```

Ilustrace 5.8: Volání resolveru

Na ilustraci 5.8 je zobrazeno volání resolveru z komponenty. Resolver je v aplikaci registrován jako aplikační komponenta správcem komponent. K resolveru lze tedy přistupovat přímo skrze volání *Yii::app()->exresolver*, tento způsob se ale nedoporučuje, protože název komponenty lze v nastavení správce komponent změnit. Resolver pro komponenty nabízí dvě metody *addCommand* a *setLocation*. Jak je ukázáno na příkladu, první metoda slouží k registraci kódu v jazyce JavaScript, který bude vykonán ihned po obdržení klientskou stranou. Pro volání jQuery knihovny a jejich objektů lze s výhodou použít rozšíření phpJQuery popsané v kapitole 5.5. Druhou metodou dostupnou komponentám je metoda *setLocation*, která přijímá URL, na které bude stránka přesměrována. Pokud se více komponent pokusí nastavit tuto URL, vždy se přepíše původní hodnota a použije se naposledy přidaná.

Resolver po dobu běhu aplikace sbírá data od komponent a na konci běhu aplikace mu správce komponent předá komponenty k vykreslení. Pokud se jedná o požadavek provedený technologií AJAX, resolver tiskne zpracovaná data ve formátu JSON, která na straně klienta zpracuje rozšíření excomponents popsané v kapitole 5.6. Pokud jde o klasický požadavek jsou nashromážděné kódy javascriptu předány aplikační komponentě *clientScript*, která je vypíše těsně před ukončující značku těla dokumentu (</body>), kde budou vykonány ihned po načtení celé stránky.

5.5 Modul phpJQuery

Mou snahou bylo přenést těžiště mezi jazyky JavaScript a PHP co nejvíce na stranu PHP. Při vytváření komponent pro technologii AJAX je ovšem nezbytná její klientská část. To by bez použití dalších prostředků vyžadovalo udržovat dohromady PHP kód a JavaScript. Resolver umí přijímat části kódu v jazyce JavaScript, které poté spustí na straně klienta, je ovšem na komponentách samotných, aby tento kód sestrojili. To znamená mít v kódu PHP části kódu jazyka JavaScript uložených v řetězcích, což obecně zavádí do programování slabý článek, ve kterém se velice lehce můžou vyskytnout těžko odhalitelné chyby. Z výše uvedených důvodů jsem vytvořil pomocnou knihovnu pro práci s frameworkem jQuery z prostředí PHP. Při vytváření této knihovny jsem se snažil vytvořit takové PHP třídy, které by umožňovali zápis co nejpodobnější volání jQuery. Účelem této knihovny je tedy poskytnout obraz rozhraní knihovny jQuery v PHP a možnost převedení kódu z PHP přímo na jeho javascriptovou podobu. Knihovna si neklade za cíl poskytnout úplné rozhraní pro práci s knihovnou jQuery, neboť to přesahuje rozsah této práce. Knihovna se omezuje pouze na podporu činností s knihovnou jQuery, které jsou typické při vývoji Ex komponent.

```
$('#someElement').method('param1', {
    'key1': 'value1',
    'key2': 'value2'
});
```

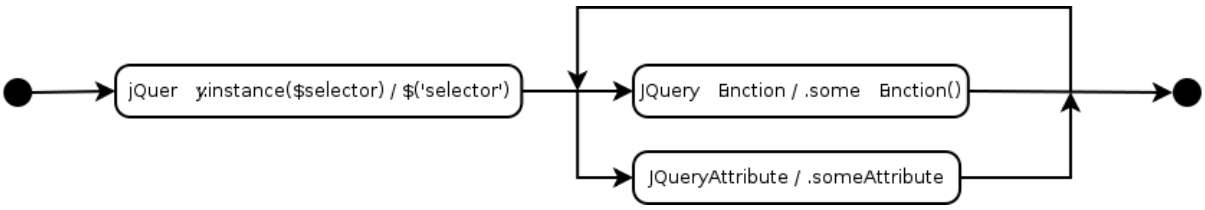
Ilustrace 5.9: Příklad volání metody v jQuery

Na ilustraci 5.9 je vidět smyšlený příklad volání nějaké metody (v terminologii jQuery jde obvykle o plugin) objektu jQuery s dvěma parametry. První je obyčejný řetězec a druhý je pole. Úkolem knihovny phpJQuery je umožnit stejné volání metody objektu jQuery formou volání PHP objektů, převod těchto volání na řetězec v jazyce JavaScript a převod parametrů metod z PHP na javascriptové typy.

```
$js = JQuery::instance('#someElement')->method('param1', array(
    'key1'=>'value1',
    'key2'=>'value2'
));

// Vrací $('#someElement').method('param1', {'key1':'value1','key2':'value2'});
```

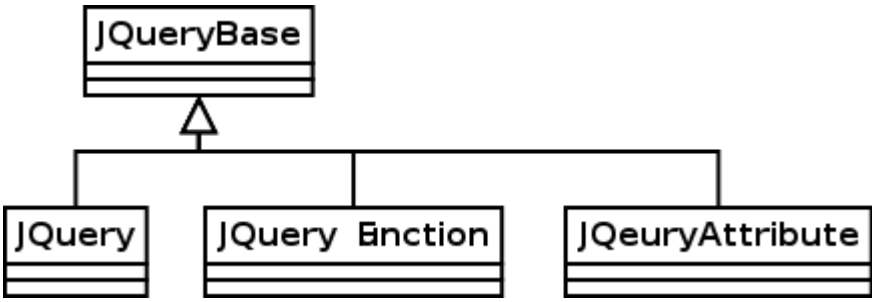
Ilustrace 5.10: Příklad volání metody jQuery v PHP

Na ilustraci 5.11 je znázorněn příklad práce s knihovnou phpJQuery v prostředí jazyka PHP. Jak je vidět, zápis je velice podobný volání přímo v jazyce JavaScript.



Ilustrace 5.11: Graf použití modulu phpJQuery

Z pohledu knihovny phpJQuery se volání knihovny jQuery skládá ze tří částí. První částí je vždy volání objektu jQuery, čemuž v javascriptu obvykle odpovídá objekt „$“. V knihovně tento objekt zastupuje třída JQuery. Na tomto objektu jsou potom volány další metody nebo získávány jeho atributy. Pro reprezentaci volání metody je v knihovně třída JQueryFunction a pro atributy je to třída JQueryAttribute. Instance knihovny jQuery se chová obvykle tak, že při volání metod vrací svou vlastní instanci a příkazy se tak dají řetězit za sebe. Knihovna phpJQuery se chová stejně. Při volání metody(nebo při přístupu k atributu) se vytvoří a vrátí nová třída reprezentující toto volání, která si pamatuje svého předchůdce. Nakonec, kdy je potřeba reprezentovat řetězec volání jako textový řetězec, volaná třída získá reprezentaci svého rodiče a přidá svoji. Výpis tedy probíhá rekursivně.



Ilustrace 5.12: Diagram tříd modulu phpJQuery

5.6 Klientská vrstva

Protože Ex komponenty jsou navrženy tak, aby pracovali jak na straně serveru, tak na straně klienta, je potřeba i na klientské straně vytvořit podporu pro jejich vývoj. V mém rozšíření tato podpora sestává z rozšíření pro framework jQuery. Toto rozšíření se nazývá *excomponents* a jeho úkolem je zpracovávat příchozí odpovědi na AJAX požadavky. Rozšíření *excomponents* využívá toho, že framework jQuery umožňuje nastavit výchozí parametry jeho vrstvy, obstarávající komunikaci skrze technologii AJAX a nastavuje svou metodu jako výchozí metodu pro zpracování úspěšného požadavku, jak je zobrazeno na ilustraci 5.13:

```
jQuery(function($){
    $.ajaxSetup({
        success: $.excomponents.success,
        dataType: 'json'
    });
});
```

Ilustrace 5.13: Nastavení jQuery pro použití excomponents

Tímto krokem je zajištěno, že při každém asynchronním požadavku je odpověď zpracována rozšířením *excomponents*. Na straně serveru se o sestrojí odpovědi stará *resolver*, který od komponent sbírá požadavky na svoje překreslení, spuštění nějakého kódu v jazyce JavaScript a požadavky na přesměrování stránky. Resolver příjmuté požadavky zpracuje a odešle na klientskou stranu ve formě JSON dat.

```
{
    "commands":[
        "document.write('Hello world');"
    ],

    "components":{
        "TextComponent": "Nový text"
    },

    "location":""
}
```

Ilustrace 5.14: JSON data přenášená od persisteru směrem k modulu excomponents

V těchto datech, jak je znázorněno na ilustraci 5.14, jsou přenášeny tří složky. První je pole *commands*, které obsahuje příkazy v jazyce JavaScript, které se mají provést. Druhou složkou je asociativní pole *components*, kde klíčem jsou názvy komponent a hodnotou je jejich HTML kód. Poslední složkou je textový řetězec *location*, ve kterém se nachází URL adresa, na kterou bude aktuální dokument přesměrován. Po přijetí odpovědi od resolveru klientská strana postupně projde tyto data, spustí obsažené příkazy, nahradí obsah komponent jejich novým obsahem a případně provede přesměrování na jinou stránku.

5.7 Vytváření a práce s Ex komponentami

V této kapitole uvedu několik příkladů vytvoření nové komponenty a práci s komponentami. Nejjednodušší komponenty, na kterých lze ukázat princip fungování celého systému je textová komponenta a komponenta tlačítko. Výstupem by měla být jednoduchá stránka s textem a tlačítkem, kde po stisku tlačítka dojde okamžitě ke změně obsahu textové komponenty.

```php
class TextComponent extends ExComponent{
    /** @var string Obsah komponenty */
    public $text = '';

    public function setText($value){
        $this->rerender = true;
        $this->text = $value;
    }

    public function getText(){
        return $this->text;
    }

    public function renderContent(){
        echo $this->text;
    }
}
```

Ilustrace 5.15: Ukázková textová komponenta

Na ilustraci 5.15 je znázorněna textová komponenta s názvem *TextComponent*. Kód této komponenty musí být uložen ve složce, kde správce komponent vyhledává komponenty. V implicitním stavu je to složka *protected/excomponents*. Komponenta dědí z obecné třídy ExComponent, což je obecná třída Ex komponenty. Textová komponenta neobsahuje žádné události a jejím účelem je pouze výpis svého obsahu. Ten je uložen v atributu *text*. V setteru *getText* je důležité označit komponentu jako změněnou a že ji je potřeba překreslit. To se provede nastavením atributu *rerender* na logickou hodnotu true. Výpis komponenty zajišťuje metoda *renderContent*.

```php
$textComponent = Yii::app()->excomponents['TextComponent'];
$textComponent->setText('Hello world');

echo $textComponent;
```

Ilustrace 5.16: Práce s komponentou

Na ilustraci 5.16 je znázorněno načtení komponenty ze správce komponent a její výpis. Správce komponent pracuje tak, že se pokusí načíst komponentu z persisteru a pokud v něm není nalezena, založí se nová instance komponenty. Tímto způsobem lze komponentu získat kdekoli v celé aplikaci, v kontrolérech a jiných komponentách. Komponenty mají implementovánu magickou funkci __toString, takže lze pohodlně získat jejich HTML kód.

```
<div id="TextComponent">HelloWorld</div>
```

Ilustrace 5.17: Výsledný HTML kód komponenty

Na ilustraci 5.17 je zobrazen výstup komponenty. Při výpisu komponenty se výpis metody *renderContent* obalí HTML prvkem *div* s atributem *id* nastaveným na identifikátor komponenty, v tomto případě je to tedy *TextComponent*. Tento obal je důležitý při součinnosti s javascriptovou knihovnou excomponents, která tak může aktualizovat automaticky obsah komponent.

```
class ButtonComponent extends ExComponent{
    public $label = 'Click';

    public function renderContent(){
        echo CHtml::ajaxButton($this->label, $this->createUrl('click'));
    }

    public function onClick(){
        $textComponent = Yii::app()->excomponents['TextComponent'];
        $textComponent->setText('You have clicked the button');

        $resolver = ComponentManager::getResolver();
        $resolver->addCommand('alert("Click!")');
    }
}
```

Ilustrace 5.18: Komponenta tlačítka

Na ilustraci 5.18 je ukázka Ex komponenty reprezentující tlačítko. Komponenta k výpisu svého obsahu využívá knihovnu *CHtml*, která je součástí Yii frameworku. Metoda *ajaxButton* knihovny generuje HTML kód tlačítka a navíc zajistí vygenerování kódu v jazyce JavaScript, který pomocí knihovny jQuery po stisku tlačítka vyvolá požadavek technologií AJAX. V příkladu je použita metoda *createUrl* bázové třídy ExComponent, které je předán název události. Metoda očekává na vstupu identifikátor komponenty a její události oddělený lomítkem. Pokud není zadán identifikátor komponenty, použije se aktuální komponenta.

Po stisku tlačítka se provede požadavek na server pomocí technologie AJAX, požadavek zpracuje správce komponent a zavolá metodu *onClick* v komponentě *ButtonComponent*. V metodě *onClick* se změní obsah textové komponenty *TextComponent* a je zde i ukázáno, jak komponenta může zaregistrovat nějaký kód v jazyce JavaScript. Po dokončení požadavku získá klientská strana nový obsah komponenty TextComponent a příkaz alert(). Javascriptová vrstva vymění současný obsah komponenty *TextComponent* za nový a provede příchozí příkaz. V konečném důsledku tedy po stisku tlačítka dojde ke změně obsahu textové komponenty a uživateli se zobrazí okno se zprávou.

Pro korektní použití Ex komponent je nezbytné, aby kontroléry, které používají Ex komponenty, byli potomky třídy *ComponentController*. Tato třída je dodána s Ex komponentami a jejím účelem je zajistit správné vložení kódů v jazyce JavaScript do výsledné stránky.

6 Závěr

Cílem této práce bylo vytvořit systém AJAX komponent pro PHP framework Yii, čehož jsem podle mého názoru bylo dosaženo. Během studia této problematiky jsem nalezl pouze jediný srovnatelný systém pro tvorbu dynamických webových stránek a to framework Nette. Oproti Nette můj systém používá komponenty, které nejsou závislé na konkrétním kontroléru ani na šabloně.

Systém jsem navrhl se zvláštním zřetelem tak, aby co nejvíce využíval vlastností Yii frameworku a zároveň abych nemusel samotný framework nijak upravovat. Tím by byla vážně poškozena dopředná kompatibilita s novými verzemi Yii frameworku. Splnění tohoto požadavku si vyžádalo detailní studium architektury Yii frameworku. Výsledkem jsou dva samostatné moduly (v terminologii Yii frameowrku jde o „extension“), které stačí pouze nakopírovat do Yii frameworku do složky s rozšířeními a příslušně nastavit konfiguraci.

Architekturu rozšíření jsem navrhoval s důrazem na oddělení odpovědností jednotlivých komponent, aby bylo možné v budoucnu přidávat další funkcionality nebo měnit současné chování systému. Zvláště v oblasti ukládání vnitřních stavů komponent by mohlo být implementováno mnoho různých typů úložišť s různým určením. Např. soubor, XML, CVS nebo různá databázová úložiště.

Během řešení projektu se ukázalo, že při vytváření komponent je často nutné programově v jazyce PHP vytvořit krátký kód v jazyce JavaScript a uložit jej do proměnné v jeho textové podobě. Nejčastěji šlo o volání knihovny jQuery nebo jejích rozšířené. Takováto volání mají ustálený tvar a jednoduchou strukturu, proto jsem nad rámec zadání vytvořil pomocnou knihovnu s názvem PhpJQuery, která se snaží napodobit volání knihovny jQuery v jazyce PHP. V této knihovně jsem implementoval pouze nezbytnou funkcionalitu, kterou jsem potřeboval při práci s komponentami, ale rád bych v práci na této knihovně pokračoval. Obě knihovny bych rád uveřejnil pod některou z všeobecných veřejných licencí např. GNU GPL a k dalšímu vývoji tak přizval širší skupinu vývojářů.

Literatura

1. ZICHA, Vojtěch. Tvorba webu [online]. c2008 [cit. 2009-11-13]. Historie PHP. Dostupné z WWW: <http://www.tvorba-webu.cz/php/historie.php>
2. Wikipedia : the free encyclopedia [online]. 2003, last modified on 13 May 2010 at 16:45 [cit. 2009-12-14]. PHP. Dostupné z WWW: <http://en.wikipedia.org/wiki/PHP>
3. PHP [online]. 2009, Last updated: Fri, 30 Apr 2010 [cit. 2009-12-20]. Iterator - Manual. Dostupné z WWW: <http://cz.php.net/manual/en/class.iterator.php>.
4. PHP [online]. 2010, Last updated: Fri, 30 Apr 2010 [cit. 2010-01-02]. Introduction. Dostupné z WWW: <http://cz.php.net/manual/en/intro.reflection.php>
5. PHP [online]. 2009, Last updated: Fri, 30 Apr 2010 [cit. 2010-05-15]. The ArrayAccess interface. Dostupné z WWW: <http://cz.php.net/manual/en/class.arrayaccess.php>
6. Wikipedie : Otevřená encyklopedie [online]. 2006, Stránka byla naposledy editována 2. 5. 2010 v 13:13 [cit. 2010-01-05]. Framework. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Framework>.
7. Yii PHP Framework [online]. 2008 [cit. 2010-01-08]. About. Dostupné z WWW: http://www.yiiframework.com/about/>.
8. Yii PHP Framework [online]. 2008 [cit. 2010-01-09]. Performance Comparison. Dostupné z WWW: <http://www.yiiframework.com/performance/>.
9. XUE, Qiang. Yii PHP Framework [online]. 2010, 2010-03-05 16:47:11 [cit. 2010-01-18]. CComponent. Dostupné z WWW: <http://www.yiiframework.com/doc/api/CComponent>.
10. XUE, Qiang. Yii PHP Framework [online]. 2010, 2010-04-13 03:51:37 [cit. 2010-01-22]. CController. Dostupné z WWW: <http://www.yiiframework.com/doc/api/CController>.
11. CHAFFER, Jonathan; SWEDBERG, Karl. Learning jQuery 1.3 : Better Interaction Design and *Web* Development with Simple JavaScript Techniques. Olton : Packt Publishing Ltd., 2009. 421 s. ISBN 978-1-847196-70-5.
12. Wikipedie : Otevřená encyklopedie [online]. 2005, Stránka byla naposledy editována 1. 5. 2010 v 09:59 [cit. 2010-05-2]. AJAX. Dostupné z WWW: <http://cs.wikipedia.org/wiki/AJAX>
13. GARRETT, Jesse James. Adaptive path [online]. 2005 : February 18, 2005 [cit. 2010-05-15]. Ajax: A New Approach to Web Applications. Dostupné z WWW: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>

Přílohy

A) Instalační manuál

B) CD se zdrojovými kódy Yii frameworku, rozšířením Ex komponent a dokumentace.

Příloha A – Instalační manuál

Instalační manuál obsahuje návod, jak použít systém AJAX komponent ExComponents v prostředí frameworku Yii. Předpokládá se znalost architektury PHP aplikací. Aplikace byli vytvářeny a testovány v Yii frameworku ve verzi 1.1.1, operačním systému Kubuntu 9.10.

Na přiloženém CD jsou tyto složky:

- ExampleApplication,
- excomponents,
- phpjquery.

Složka ExampleApplication obsahuje nachystanou předváděcí aplikaci i s instalací Yii frameworku ve verzi 1.1.1. Pro spuštění této aplikace je ji potřeba nakopírovat na nějaký hostingový nebo lokální server podporující PHP verzi alespoň 5.1.0. Ukázková aplikace je také dostupná na webu[1]. Složka ExComponents obsahuje samotné rozšíření pro Yii framework a složka phpJQuery pomocnou knihovnu PhpJQuery.

Instalace Yii frameworku

Pro použití v Yii frameworku je potřeba nejdříve vygenerovat novou Yii aplikaci, nebo použít již existující aplikaci. Postup tvorby nové aplikace je popsán na stránkách projektu[2]. Nejprve je potřeba stáhnout zdrojové soubory frameworku z oficiální stránek, poté zkopírovat složku *framework* do umístění, kde lze spouštět PHP scripty. Složka framework obsahuje konzolový program yiic, který umí vytvořit kostru nové aplikace. Na operačním systému linux je potřeba spustit tyto příkazy:

```
chmod +x framework/yiic
php framework/yiic webapp slozka_nove_aplikace
```

Na místě dané druhým parametrem bude vygenerována kostra nové aplikace. Dalším krokem je nastavení práva na zápis do složek:

assets,

protected/runtime.

V tuto chvíli by mělo jít aplikaci spustit. V případě problémů doporučuji spustit soubor *requirements/index.php*, který je součástí balíčku Yii frameworku.

1 http://www.mojefitko.cz/ExComponents

2 http://www.yiiframework.com/doc/blog/start.testdrive

Instalace rozšíření

Dalším krokem pro použití Ex komponent je nakopírování složek *ExComponents* a *phpJQuery* z přiloženého CD do aplikace do složky *protected/extensions* a nastavení aplikace. Nastavení aplikace je znázorněno na následujícím obrázku:

```
return array(
    'preload'=>array(
        'excomponents',
    ),

    'import'=>array(
        'application.extensions.excomponents.*',
        'application.extensions.excomponents.components.*',
        'application.extensions.jquery.*',
        'application.excomponents.*',
    ),

    'components'=>array(
        'excomponents'=>array(
            'class'=>'ComponentManager',
        ),
    ),
);
```

Závěr

V případě problémů se spouštěním aplikace lze využít přiložené funkční verze pro kontrolu cest a nastavení aplikace.