

**jQuery - prototyp pluginu pro**

**rezervační systém**

[***KIV/BPINI***](https://portal.zcu.cz/wps/myportal/!ut/p/c5/hc7LDoIwFATQT-rQ0ofLBkmpVhCMiGwIC2MwAi6M328JiQsSde7yZDKX1MTf0L66a_vsxqG9k4rUopFZbth-TZEprEC5EHlcHOHj_SyayOgklA4wknLQlB_oTpeAZX_ap2nvd39yfInG7J8FlEkMKjV3sjABIrbwxQdGzL51scoiFcIw79aqTSBjF0wfpsnYX8ijr9DZW_gGF-KUgw!!/dl3/d3/L0lDU0lKSmdrS0NsRUpDZ3BSQ2dwUkNncFJDU0EvWU9VSUFBSUlJSU1NSUtFRUFBQ0dPR09DR0lCSkZKRkJKTkROREJOTEhMSEJMQW9FQVBQQS80QzFiOVdfTnIwZ0RFU1pJSlJERVNaTUpRaWt5VVNqRkprNGxCSWlSLUEhIS83XzdPUUczUEQyMEdWMDAwMjVWR0g4NDExME01L19VZTluODg4MDAwOTcvaWJtLmludi8yMTYzNDIxODA3OTgvUGxuZW5pUm9rVmFyaWFudHkvMjAxMi9QbG5lbmlaa3JhdGthRmFrdWx0eS9LSVYvUGxuZW5pWmtyYXRrYVByZWRtZXR1L1BSSjU!/#prohlizeniAnchor) ***– Bakalářská práce***

student: Petr Kukrál

studijní číslo: A10B0618P

email: kukral@students.zcu.cz

datum: 15. 12. 2012

Obsah

[1 Tvorba pluginu jQuery 1](#_Toc351123672)

[1.1 Verzovací systémy 1](#_Toc351123673)

[1.2 Komentáře 2](#_Toc351123674)

[1.3 Vytvoření pluginu a připojení ke knihovně 2](#_Toc351123675)

[1.4 Použití obalové funkce 3](#_Toc351123676)

[1.5 Význam středníku před skriptem 3](#_Toc351123677)

[1.6 Defaultní parametry pluginu 3](#_Toc351123678)

[1.7 Soukromé funkce v pluginu 4](#_Toc351123679)

[1.8 Načítání dat ze souboru 4](#_Toc351123680)

[1.9 Formáty vstupu 4](#_Toc351123681)

[1.10 Obecná použitelnost 4](#_Toc351123682)

[1.11 Výstup pluginu 5](#_Toc351123683)

[2 Realizace pluginu pro rezer. system 6](#_Toc351123684)

[2.1 pouzival jsem bootstrap 6](#_Toc351123685)

[2.2 Databáze 6](#_Toc351123686)

[2.2.1 Databázový model - MySQL – obrázek s popisem 7](#_Toc351123687)

[2.2.2 Konfigurační soubor - názvy tabulek jako řetězce 7](#_Toc351123688)

[2.2.3 Model databáze PHP – z pohledu MVC + model jedinacek 7](#_Toc351123689)

[2.3 Popis ukazky 7](#_Toc351123690)

[2.3.1 Posílání zpráv – message - ?? 7](#_Toc351123691)

[2.3.2 Výběr sálu 7](#_Toc351123692)

[2.3.3 Výběr místa 7](#_Toc351123693)

[2.3.4 Rušení rezervace 7](#_Toc351123694)

[2.4 HTML dokument ?? 7](#_Toc351123695)

[2.4.1 Návrh dokumentu pro vložení pluginu 7](#_Toc351123696)

[2.5 Události 7](#_Toc351123697)

[2.6 O Pluginu – požádat o radu co vypíchnout 7](#_Toc351123698)

[2.6.1 Používané třídy ??? 7](#_Toc351123699)

[2.6.2 Události ??? 7](#_Toc351123700)

[2.6.3 jak se to chova 7](#_Toc351123701)

[2.7 Ukázka pluginu 7](#_Toc351123702)

[2.8 Použití pro kino 7](#_Toc351123703)

[2.9 Použití pro squash 7](#_Toc351123704)

[3 Použitá literatura 8](#_Toc351123705)

[4 Příloha 8](#_Toc351123706)

[4.1 Ukázky zdrojových kódů 8](#_Toc351123707)

# Tvorba pluginu jQuery

Touto kapitolou bych chtěl sdělit nejdůležitější problémy, na které jsem při psaní pluginu narazil a popsat, jak jsem je v práci řešil.

## Verzovací systémy

Verzovací systém by měl ulehčit práci na projektu. A to hlavně v případě, že na projektu pracuje více vývojářů. To by mělo být zajištěno pomocí tkz. větví (anglicky branches). Ty by měly rozdělit projekt to více částí, na kterých vývojáři pracují samostatně. Nakonec by měl systém umět tyto větve spojit do jedné a tím sloučit projekt. Má obsahovat správu verzí projektu a nejlépe automaticky projekt verzovat. Při vývoji této práce by měl hlavně soužit pro verzování projektu. Podpora práce pro více vývojářů, bude v tomto projektu zbytečná.

Verzovacích systémů je celá řada. Já bych rád uvedl jen tři, které jsou podle mého názoru nejzákladnější.

* Prvním, z těchto tří nejjednodušším, je CVS (Concurrent Version System). Podle (1), jde o pomyslné rozšíření RCS (Revision Control System). CVS je verzovací systém pro skupinu souborů. Tyto soubory mohou být uloženy v různých adresářích. Adresář zde není brán jako soubor, ale jde o zvláštní typ. Jak uvádí (2) „V CVS měl adresář speciální postavení, soubory se ukládaly a organizovaly podle toho, v jakém adresáři se nacházely. To znamená, že adresář nejde dost dobře smazat, činí to zbytečné problémy při přesouvání souborů apod. Proto by měl v našem IVS adresář pokud možno co nejvíce vystupovat jako obyčejný soubor, u kterého si pouze poznamenáme, že zároveň obsahuje i nějaké jiné soubory. Měli bychom pak možnost adresáře přesouvat, přejmenovávat, vytvářet a mazat stejně jako všechny ostatní soubory.“ CVS dokáže spravovat a slučovat větve. Provádí to ovšem neefektivním programem *diff3* a zapomíná informace o vedlejší větvi, jak uvádí zdroj (2).
* SVN (Subversion) nabízí již více než CVS. Hlavní změnou, jak uvádí zdroj (3), je nakládání s adresářem. SVN totiž nerozlišuje rozdíl mezi adresářem a souborem. Také zvládá větvení projektu. Jak ale budu psát v dalším odstavci, jeho nevýhodou, oproti systému GIT, je rychlost práci s daty.
* Jako poslední verzovací systém představím GIT. Tento program je napsán v jazyce C. Jak tvrdí zdroj (4), hlavním cílem při vývoji GITu byla rychlost běhu. V tabulce uvádím porovnání běhu GIT a SVN. Porovnání je pouze přibližné, jak sám autor testů uvádí. GIT umí dobře pracovat s větvemi, dokáže je vytvářet i slučovat.

Pro mojí práci jsem si vybral systém GIT. Pro veřejně přístupné projekty je možné využít systému github, který je zdarma. Celý projekt je pak dohledatelný přímo na webu github.com i se zdrojovými kódy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation |  | Git | SVN |  |
| Commit Files (A) | Add, commit and push 113 modified files (2164+, 2259-) | 0.64 | 2.60 | 4x |
| Commit Images (B) | Add, commit and push 1000 1k images | 1.53 | 24.70 | 16x |
| Diff Current | Diff 187 changed files (1664+, 4859-) against last commit | 0.25 | 1.09 | 4x |
| Diff Recent | Diff against 4 commits back (269 changed/3609+,6898-) | 0.25 | 3.99 | 16x |
| Diff Tags | Diff two tags against each other (v1.9.1.0/v1.9.3.0 ) | 1.17 | 83.57 | 71x |
| Log (50) | Log of the last 50 commits (19k of output) | 0.01 | 0.38 | 31x |
| Log (All) | Log of all commits (26,056 commits - 9.4M of output) | 0.52 | 169.20 | 325x |
| Log (File) | Log of the history of a single file (array.c - 483 revs) | 0.60 | 82.84 | 138x |
| Update | Pull of Commit A scenario (113 files changed, 2164+, 2259-) | 0.90 | 2.82 | 3x |
| Blame | Line annotation of a single file (array.c) | 1.91 | 3.04 | 1x |

Tabulka zdroj (4)

## Komentáře

V této práci komentáře hrají velkou roli. Práce obsahuje nejen plugin, ale i ukázkové použití. Proto jsem se zde snažil používat různé druhy dokumentačních postupů v závislosti na tom, v kterém jazyce jsem komentář psal.

* V práci jsem při psaní Modelu databáze používal komentáře PhpDoc (celým názvem PhpDocomentor) popisované ve zdroji (5). Díky PhpDoc komentářům se dá vygenerovat přehledná dokumentace celého kódu. Dají se zde používat nejen jednoduché texty, ale mohou obsahovat i tagy jako je například seznam. Tato dokumentace se potom dá vygenerovat v různých formátech jako je HTML, PDF či CHM. Další velkou výhodou těchto komentářů je, že se dají propojit s některými vývojovými prostředími (například s vývojovým prostředím Eclipse). Při použití zdokumentovaného kódu pak může vývojové prostředí našeptávat komentáře.
* Pro psaní samotného pluginu jsem se snažil co nejvíce přiblížit komentářům Jsdoc popsaným ve zdroji (6). Z takto napsané dokumentace se dá rovněž vygenerovat dokumentace ve formátu HTML.

## Vytvoření pluginu a připojení ke knihovně

Knihovna jQuery nabízí velmi jednoduchý způsob, jak k ní připojovat další funkce. Stačí rozšířit objekt *jQuery.fn*. Tedy například v případě pluginu pro rezervační systém budete ukládat do objektu *jQuery.fn.booking\_system* všechny potřebné vlastnosti pluginu. Ukázku vkládání vidíme ve zdrojovém kódu 1.1.

$.fn.booking\_system = function(options){  
 /\* zde je vše potřebné pro běh pluginu \*/  
}

ukázka zdrojového kódu .

K vytvořenému pluginu pak přistoupím jednoduše tak, jako by to byla jakákoliv jiná funkce jQuery. V mém případě to bude *jQuery.booking\_system*.

## Použití obalové funkce

Při používání knihovny jQuery je zvykem, jak uvádí (3), využívat k přístupu k objektu jQuery jeho zkrácenou verzi zápisu $. Tento přístup, který jsem použil i ve své práci, má řadu výhod.

Kód je díky tomuto zápisu kratší. Jelikož se javascript stahuje spolu s načtením stránky a teprve potom je spuštěn v prohlížeči, je kladen velký důraz na to, aby byl kód co nejkratší. (4 str. 141) tvrdí, že je tím zajištěna rychlejší odezva stránky mezi tím, když se začne stránka stahovat a mezi tím, než začne plně reagovat.

Kód je čitelnější. Jelikož se objekt jQuery používá v pluginu opravdu často, zkrácený zápis může být v mnoha případech čitelnější.

Protože je ale používání zkráceného zápisu oblíbené, může přinést řadu problémů. Například knihovna, která je přidána ke stránce před touto, může zkrácený zápis $ využívat jako ukazatel na něco jiného než objekt jQuery. Z tohoto důvodu jsem v práci použil obalovou funkci. Obalovou funkci najdete v ukázce kódu 1.2.

(fuction($) {  
 /\* celý kód pluginu \*/  
})(jQuery);

ukázka zdrojového kódu .

Jde o velmi jednoduché řešení, kdy definuji obalovou funkci, ve které se nachází celý plugin. Její parametr bude $. Přímo za dodefinováním funkce ji zavolám a jako parametr vložím objekt jQuery. Jazyk JavaScript pak vymezí rozsah platnosti této proměnné na celou funkci. V té potom můžu $ volně používat bez vzniku kolize. Další výhodou tohoto postupu je, že definováním obalové funkce vytvořím uzávěr. Ten mi pomůže s možnou kolizí názvů metod tohoto a dalších používaných pluginů.

## Význam středníku před skriptem

Jak tvrdí (3), jazyk JavaScript si dokáže poradit se zapomenutým středníkem na konci řádku. Ovšem v případě minimalizace kódu se konce řádků vypouští a může tak docházet k chybám. Proto je důležité, na začátku pluginu napsat středník. Tím ošetříte případ, kdy programátor, který psal kód přilinkovaný před Vaším na středník zapomněl.

## Defaultní parametry pluginu

Do pluginu by jsme mohli předávat argumenty jednoduše jako parametry funkce. Toto jednoduché řešení ovšem nebude stačit v momentě, chceme-li správně nabídnout uživateli pluginu defautní možnosti.

V tomto případě je dobré, definovat si vlastní defaultní hodnoty a připojit je k pluginu. Pak jednoduše pomocí metody extend porovnáte defaultní nastavení a nastavení od uživatele. Metoda sama zajistí doplňění údajů od uživatele defaultními údaji.

Výhoda tohoto přístupu je jistě čitelnější kód. Pluginy většinou mívají hodně proměnných, které je potřeba nastavit. Tímto přístupem se tedy vyvarujete tomu, kontrolovat všechny proměnné, zda byly nastaveny. Jednoduše kontrolujete jen to, co potřebujete. V ukázce zdrojového kódu 1.3 je vidět, jak se pomocí funkce extend načítá defaultní nastavení a nastavení které si zvolil uživatel. V druhé ukázce zdrojového kódu 1.4 je samotné připojení defaultního nastavení k pluginu.

/\* doplneni nevyplnenych nastaveni o deafultni nastaveni \*/  
var opts = $.extend({}, $.fn.booking\_system.defaults, options);

ukázka zdrojového kódu .

$.fn.booking\_system.defaults = {  
 /\* formát vstupu \*/  
 format: "cvs",  
 /\* další nastavení \*/  
};

ukázka zdrojového kódu 1.4

## Soukromé funkce v pluginu

Při psaní pluginu jsem potřeboval napsat hodně funkcí, které nebudou viditelné pro uživatele pluginu. Kdybych je připojil k pluginu, mohl by uživatel tyto funkce překrýt. Tento problém jsem vyřešil právě již zmíněnou obalovou funkcí. Uvnitř obalové funkce jsem si definoval funkci, kterou jsem nepřipojil k objektu pluginu. Tím ji mohu uvnitř pluginu využívat, ale zvně pluginu zůstane skrytá.

## Načítání dat ze souboru

Při načítání dat ze souboru je možné použít Javascript a knihovnu jQuery. Slouží k tomu funkce *get* která se volá nad objektem jQuery. V případě mého pluginu jsem použil funkci *file\_get\_contents* která je v jazyce PHP. Rozhodl jsem se tak pro to, že načtení souboru vytvořeného uživatelem může být zdrojem mnoha chyb, např. že daný soubor neexistuje. Proto se mi zdálo lepší, načítat soubor do stránky, kde se dá jednoduše v náhledu zdrojového kódu ověřit, zda se opravdu data načetla a jsou správně.

## Formáty vstupu

Jaké všechny formáty podporuji a proč? JSON, xls, cvs

## Obecná použitelnost

Při psaní pluginu jsem musel dbát jeho použitelnost v různých projektech. Je to zřejmé na příkladech kina a squashového kurtu, které jsem vytvořil jako ukázky používání pluginu.

Prvnímu, čemu jsem se musel vyhnout, je přímé stylování objektů. V pluginu generuji celý systém přímo do stránky a zde nesmí být uveden žádný přímý styl. To by znemožnilo jeho použitelnost na jakýkoliv jiný případ. Je to kvůli tomu, že přímé stylování má nejvyšší prioritu vytváření vzhledu objektu a nešlo by překrýt. Všechny styly jsem tedy přesunul do vnějšího souboru, kde si je uživatel pluginu může lehce změnit.

Dalším důležitým prvkem u stylů bylo nechat třídy a identifikátory volitelné. Mohlo by totiž dojít k tomu, že by kolidovali s již použitými názvy na stránkách uživatele a ten by musel své stránky nebo plugin ručně předělat. Aby ovšem uživatel nemusel nastavovat všechny třídy ručně při zavedení pluginu do své stránky, ponechal jsem všem třídám a identifikátorům výchozí hodnoty. Uživatel si tak změní jen ty názvy, které mu na stránkách kolidují.

Jelikož plugin potřebuje pro svůj správný běh uvést cestu k obrázkům, které používá (jako je třeba obrázek sedačky v kině), i tato cesta zůstala v pluginu nastavitelná pro všechny obrázky.

Při psaní pluginu jsem dbal na jeho vizuální stránku. Při najetí kurzorem myši nad objekt, či kliknutí na objekt se spouští různé akce, které objekt graficky mění. Například ho zprůhledňují. V případě, že by uživatel chtěl s objektem vykonat jinou akci, má možnost v nastavení zadat jméno funkce, která objekt funkci v pluginu překryje. Proto i toto řadím k obecné použitelnosti, protože uživatel si může napsat funkce nakládající s objektem, jak potřebuje. Jedinou podmínkou pro uživatele je přijímat objekt *events* s parametrem *target*, přes který se dostane k objektu se kterým je manipulováno.

## Výstup pluginu

Výstup z pluginu se přímo směruje do stránky do zadaného identifikátoru. Při rozhodování o struktuře výstupu z pluginu jsem se rozhodoval, jestli použít výstup pomocí tabulky, nebo pomocí divů.

Výstup pomocí tabulky by mi pomohl v tom, že by objekty byly sami řazeny do mřížky bez toho, abych je musel jakkoliv ovlivňovat styly. To by se projevilo obzvláště u generování více elementů, jako jsou sedadla v řadě. Stačilo by totiž k sedadlům přidat patřičné HTML tagy tabulky označující začátek řádku (v případě kina řady sedadel) a dalšího prvku (sedadla). Nevýhoda tohoto přístupu by byla v tom, že nejdou elementy rozmístit nepravidelně. Tedy třeba v případě, že by chtěl uživatel rozmístit elementy do geometrického obrazce, například kruhu, by musel jednotlivé elementy vyhledávat a stylovat každý zvlášť.

Druhá možnost byla ovlivnit rozmístění objektů pomocí divů. Tedy každý div element by obsahoval vlastní objekt (v případě kina sedadlo) a uživatel by mohl objekt přesunout, kam by potřeboval. Nevýhoda tohoto přístupu je zřejmá. Musely by se udělat obecné styly pro všechny divy, nebo stylovat každý div zvlášť.

Zvolil jsem kompromis. V pluginu vytvářím skupiny objektů, ve kterých je jednotlivé objekty nachází. Můžeme si je představit například v případě kina jako řady sedadel, v kulečníkové herně by to mohly být stoly. V této skupině se pak objekty mohou stylovat libovolně. Například v případě kulečníku je můžeme umístit okolo stolu do obdélníku. Tato možnost by se s využitím tabulky řešila velmi špatně.

# Realizace pluginu pro rezer. system

V této kapitole najdete popis návrhu a samotného provedení pluginu. – dopsat co tu opravdu najdou

## Použití frameworku bootstrap

Bootstrap je front-end framework. Po vložení na stránku přestyluje všechny základní HTML elementy a vývojáři pak stačí pouze psát HTML kód. Také nabízí řadu javascriptových funkcí jako je například vytváření stylovaných vyskakovacích oken nebo různé práce s textem. Více se o bootstrapu dozvíte v kapitole Použité technologie nebo zde (9).

Při vytváření ukázky jsem se rozhodoval, jestli použít pro stylování některý z dostupných frameworků, nebo si vytvořit vlastní styly. Jeden z hlavních kritérií rozhodování byla dobrá dokumentace. Ukázka pluginu by totiž měla být sama dosti návodná a uživatel by nemněl přemýšlet nad ukázkou samotnou. Měl by se zaměřit hlavně na použití pluginu, aby mohl plugin co nejdříve využívat na svém webu. Proto jsem zvolil tento framework. Má velmi kvalitní dokumentaci všech elementů. V dokumentaci je nejen popis stylování či javascript, ale také ukázka použití. Uživatel si tedy může velmi rychle vyhledat, co bude právě potřebovat.

Ve své práci jsem z frameworku využíval převážně předstylované HTML elementy. Ty jsem používal na celkový vzhled ukázky a ve většině náhledů. Na stylování samotné stránky rezervaci systému jsem musel používat i vlastní styly. Je tomu tak proto, abych mohl ovlivnit konkrétní věci, jako je třeba řazení a pozice sedaček v kině. Díky použití vlastních stylů jen na konkrétní vyjímečné situace se docílilo toho, že vlastní styly jsou velmi krátké a uživatel se v nich lépe vyzná.

## Databáze

Při vytváření ukázky použití pluginu jsem se zaměřil na dobrý přístup k databázi. Přístup k databázi je realizován přes model databáze. Ten má metody psané přímo pro konkrétní výběry dat či jejich ukládání. Model disponuje obranou proti útoku SQL injection. Obsahuje konfigurační soubor pro centralizaci dat, která jsou potřebná při práci s databází.

### Model databáze PHP

Celý přístup do databáze probíhá přes jediný objekt *BookingSystemDatabase*. Z pohledu MVC (Model View Controller) architektury je připojení k databázi bráno právě jako model. Ten zajišťuje veškerou komunikaci s databází. Objekt *BookingSystemDatabase* se musí před prvním použitím vytvořit a nainicializovat.

Při tvorbě ukázky jsem nejdříve celý objekt napsal podle návrhového vzoru jedináček. Tedy konstruktor byl soukromý a měl privátní statickou metodu vracící novou, nebo již vytvořenou instanci. Výhoda byla ta, že si stačilo přes metodu vrátit objekt a uživatel se nemusel starat o to, zda již připojení v kódu někde použil a je tudíž vytvořené. Uživatel měl tak vždy jeden objekt, ke kterému mohl přistupovat. Od této varianty jsem upustil kvůli tomu, že ukázka by měla být v první řadě jednoduchá a dobře čitelná. To při použití modelu jedináček nebylo. Uživatel se totiž musel seznámit s napsáním modelu, aby pochopil, jak přistoupit k objektu. Sám si objekt v ukázce vytvořit nemohl, protože konstruktor byl podle návrhového vzoru privátní. Proto jsem se vrátil k jednoduchému vytvoření objektu přes konstrukci *new*. Vytvoření a inicializaci objektu jsem nechal ve zvláštním souboru *booking\_system.class.php*, který se musí pro používání databáze do stránky přidat. Tím jsem se snažil docílit také toho, aby uživatel vytvořil objekt právě jednou.

SQL dotazy se skládají přímo v objektu, nikoliv v kódu stránky. K tomu slouží metody, které přímo vykonají SQL dotaz a popřípadě vrátí výsledek. Metody nejsou psané obecně na libovolný dotaz skládaný v kódu stránky. Jsou konkrétní a každý nový druh přístupu k databázi znamená novou metodu. Tedy například pro přihlášení má objekt metodu *signIn* s parametry email a heslo. Ukázku metody vidíte v příkladu 2.1. Metody jsou chráněny před SQL injection. To je schopnost uživatele modifikovat SQL dotaz pomocí předaných dat. Ochrana spočívá v tom, že všechny parametry jsou escapovány funkcí *addslashes*. Ta se používá pouze tehdy, není li zaplá direktiva *magic\_quotes\_gpc*, která by escapovala řetězec podruhé. Je to ochrana hlavně pro hostingy a servery používající starší verze PHP. Metoda je napsaná podle (10).

public function signIn($email, $password)  
{  
 $email = $this->gpc\_addslashes($email);  
 $password = $this->gpc\_addslashes($password);  
   
 return $this->query(  
"SELECT id FROM ". TABLE\_USERS . " WHERE email = '" . $email . "' AND password = '" . md5($password) . "'")  
 ->fetch();  
}

ukázka zdrojového kódu .

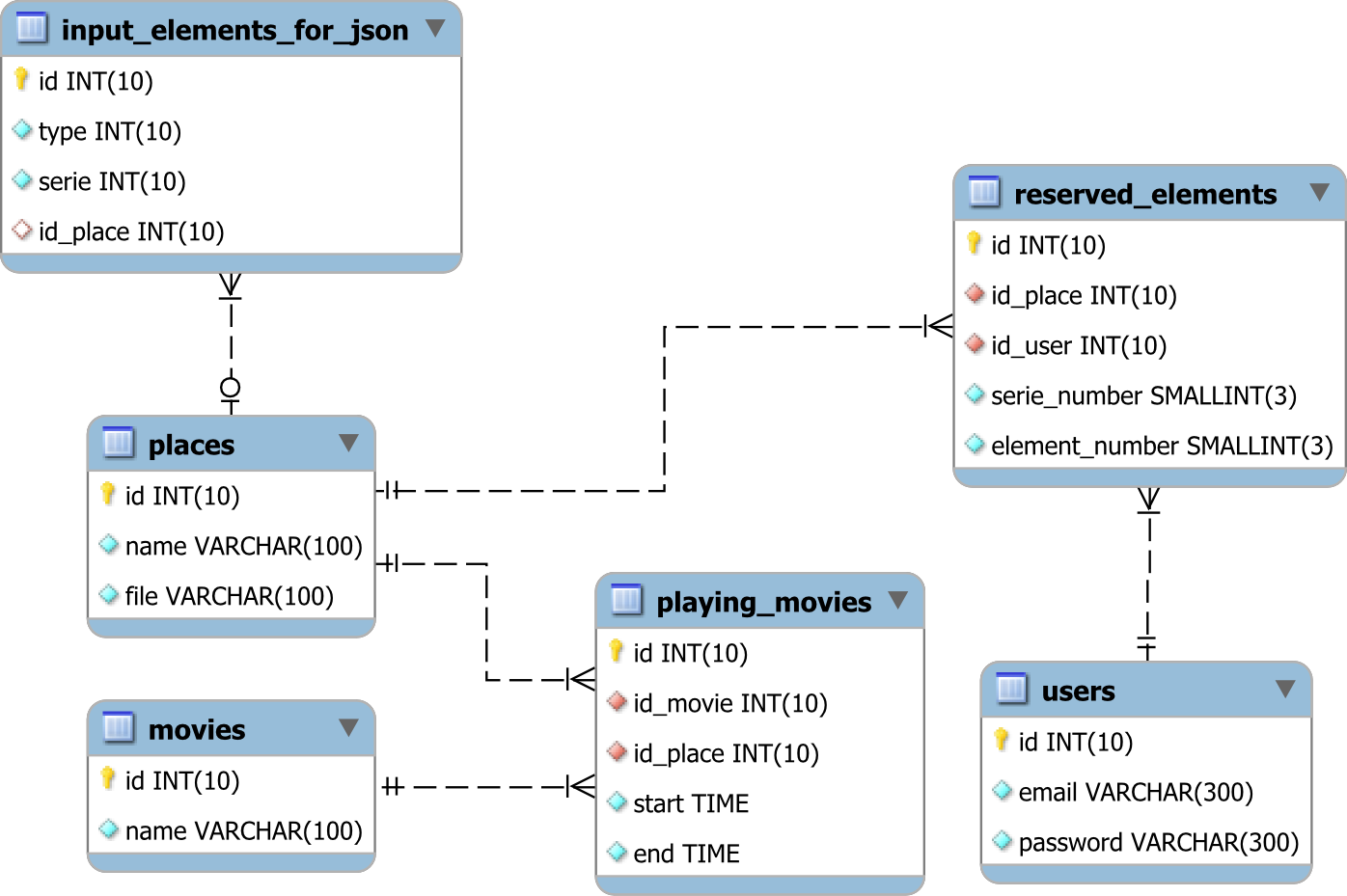
Další funkce objektu zajišťují metody na přihlášení a odhlášení. Tyto metody nemají žádné parametry, potřebné hodnoty se inicializují už při vytváření objektu. Jsou veřejné, aby si je uživatel mohl volat přímo v kódu. Další metody jsou *query* a *fetch*. Tyto metody jen obalují funkce mysql\_query a mysql\_fetch\_object a řeší případné vyjímky.

Metodou, nad kterou bych se rád pozastavil je metoda *getJSON*. Tato metoda slouží pro výběr dat z databáze a uložení do formátu JSON. Využívá funkci *json\_encode* která slouží k převádění dat do formátu JSON. Tato metoda slouží převážně k demonstraci, že plugin může přijímat i data v moderním formátu JSON. Jak můžeme vidět na ukázce 2.2, metoda má velmi zajímavou implementaci. V těle cyklu totiž není vůbec žádný kód. Cyklus slouží totiž jenom k tomu, aby se uložily výsledky z databáze do pole *$rs* a to vykonává již v podmínce. Když funkce *mysql\_fetch\_assoc* nemá další data, která by mohla převést do asociativního pole a uložit do proměnné *$rs*, vrátí hodnotu FALSE a cyklus se ukončí.

public function getJSON()  
{  
 $rs = array();  
 while($rs[] = mysql\_fetch\_assoc($this->rows)) {  
 // zde se skutečně nemá nic dělat  
 }  
 return json\_encode($rs);  
}

ukázka zdrojového kódu .

### Diagram databáze



.

Na obrázku 2.1 vidíme diagram databáze používané v ukázce pluginu. Diagram neobsahuje žádné vazby N:M. Databáze je ve třetí normální formě. Všechny tabulky obsahují primární klíče, které jsou zastoupeny maximálně desetimístným číslem. To je každému záznamu přiřazeno automaticky. V následujících odstavcích si popíšeme jednotlivé tabulky a vztahy mezi nimi.

* **playing\_movies** – představuje filmy které se budou v kině promítat. Sloupce *start* a *end* představují začátek a konec promítání. Sloupec *id\_movie* je identifikátor promítaného filmu. Sloupec *id\_place* je identifikátor sálu, kde se film promítá. Ten je v ukázce důležitý hlavně proto, aby se správně vybralo rozložení sedadel. Tabulka movies je propojená s tabulkou *places* pomocí cizího klíče *id\_place* vazbou N:1. Dále je propojená s tabulkou *movies* pomocí cizího klíče *id\_movie* také vazbou N:1. Tabulka má význam pouze v ukázce použití pluginu pro kino. V ukázce pro squash se tato tabulka nepoužívá.
* **movies** – znázorňuje samotné filmy. Jelikož nejde o opravdový systém kina, tabulka kromě svého identifikátoru obsahuje pouze název filmu. Ten je uložen ve sloupci *name*. Kdyby se tato ukázka uváděla do praxe, tato tabulka by byla jistě doplněna o další hodnoty jako je délka filmu, jestli se jedná o přístupný film do 18 let a jiné. Je propojená s tabulkou *playing\_movies* přes primární klíč.
* **places** – obsahuje místa. V ukázce kina tyto místa můžeme chápat jako jednotlivé sály. Tato tabulka se používá hlavně v případě, že je nastaven jako vstup soubor. Obsahuje ve sloupci *file* jméno souboru, ze kterého se má načítat. Každé místo má v ukázce vlastní soubor a to právě jeden, proto tabulka nebyla rozdělena na dvě. Ve sloupci *name* nalezneme jména jednotlivých míst např. „sál 1“.
* **input\_element\_for\_json** – slouží pro ukázku načtení dat ve formátu JSON. Data ukládám do této tabulky a pak je pomocí metody modelu databáze převádím do formátu JSON. Data která tato tabulka obsahuje jsou jednotlivé rezervovatelné elementy. Pro příklad kina to jsou sedačky. Sloupec *type* označuje, jestli jde o jednosedačku nebo dvojsedačku. Sloupec *serie* označuje řadu, ve které se sedačka nachází. Sloupec *id\_place* říká, kterému sálu daná sedačka náleží. Tato tabulka je přes klíč *id\_place* spojená s tabulkou *places*.
* **reserved\_elements** – se používá pro zarezervování elementu. V případě kina si můžeme element představit jako sedadlo. Kdyby se načítaly data pouze z formátu JSON, obsahovala by tabulka pouze identifikátor elementu z tabulky *input\_element\_for\_json*. Jelikož ale musí být možnost náčítat vstup i ze souboru, v tabulce musí být informace pro jednoznačné určení element. Ve sloupci *serie\_number* je informace o sérii prvku. V případě kina to bude řada sedadel. Sloupec *element\_number* obsahuje číslo elementu. Sloupec *id\_user* představuje identifikátor uživatele, který si daný element zarezervoval. Spojuje tuto tabulku s tabulkou *users* vazbou N:1. Poslední sloupec označuje místo, ve kterém si uživatel element zarezervoval. Pomocí tohoto sloupce je tato tabulka spojena s tabulkou *places*. Vazba je N:1.
* **users** – obsahuje informace o uživatelích. Používá se nejen při registraci a přihlášení, ale také k identifikování uživatele, který si daný element zarezervoval. Tabulka je spojena s tabulkou *reserved\_elements* pomocí primárního klíče. Ve sloupci *email* najdeme email uživatele. Ve sloupci *password* se nachází hashované heslo uživatele.

### Konfigurační soubor - názvy tabulek jako řetězce – config

Konfigurační soubor v této ukázce je soubor, kde jsou uloženy veškeré konstanty používané v ukázce. Vytvořil jsem ho proto, aby bylo vše dosažitelné a měnitelné na jednom místě. Nachází se v něm názvy všech tabulek uložené jako konstanty. Zde má uložení názvů jako konstanty speciální význam. Je tomu tak proto, aby se dali jednoduše měnit názvy tabulek a program zůstal stabilní. Chtěli by jsme například změnit název tabulky *movies* na *films*, stačí to udělat pouze v tomto souboru a změny se projeví v celé ukázce. Kromě názvu tabulek soubor obsahuje i údaje potřebné k přihlášení k databázi. V ukázce se jedná o soubor *config.inc.php*.

## Funkce rezervačního systému

### Vygenerování všech elementů

### Přidání elementů do vybraných

### Odebrání elemntů z vybraných

### Označení elementu jako rezervovaný

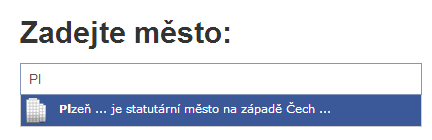
## Popis ukázky

V této části si popíšeme, co všechno ukázka obsahuje.

### Posílání zpráv – message - ??

### Výběr města

První co uživatel v ukázce musí zvolit je výběr města. Tato možnost je pouze v ukázce použití pluginu pro kina. Na této stránce jsem použil plugin *Tokeninput*. Ten reaguje na formulářové pole s textem a nabízí různé nápovědy jak text dopsat. Ten potom pošle proměnnou s názvem města další stránce, které slouží pro výběr filmu. Zajímavou vlastností je i to, že používá obrázky vedle své nápovědy pro lepší orientaci. Více se o tomto pluginu dozvíte v části Použité technologie. Na obrázku 2.2 vidíme ukázku našeptávání.



2.2

### Výběr filmu

Na této stránce je použita galerie obrázků. Tu zajišťuje plugin *Nivo Slider*, který umožňuje jednotlivé obrázky procházet a automaticky ji promítá. Více o pluginu *Nivo Slider* se dozvíte v části Použité technologie. Dále na stránce najdeme seznam všech právě promítaných filmů, kdy se začínají promítat a jméno sálu, ve kterém se promítají.

### Výběr sálu

Výběr sálu probíhá již na stránce výběr filmu. Pro lepší demonstraci pluginu jsem ale výběr sálu umístil i přímo do stránky, kde se plugin pro rezervační systém používá. Změnu sálu pak zajišťuje formulář s výběrem sálu pomocí konstrukce *select*. Pro snazší práci je formulář generován pomocí *PHP Form Builder Class*. Jde o nástroj sloužící k objektovému návrhu formuláře. Využívá HTML 5 a Ajax. Více se o *PHP Form Builder Class* dozvíte v části Použité technologie.

### Výběr místa

Výběr místa najdete na stránce rezervace. Zajišťuje ho přímo plugin pro rezervační systém. Pro vygenerování všech elementů naváže na každý element několik událostí. První je událost *mouse\_over*tedy přejetí myší nad elementem. Jelikož dle zadání měl být kladen důraz na grafické provedení, při přejetí myši nad elementem element změní barvu. Druhá událost je klepnutí kurzorem myši na element. Ten opět vyvolá změnu barvy a zároveň zavolá funkci pluginu *addToSelected*. Ta má jediný parametr *e* značící událost (anglicky event) která nastala. Více o této funkci naleznete v podkapitole *Funkce rezervačního systému*. Při kliknutí na označený element se pak spustí funkce *removeFromSelected* se stejným parametrem, který daný element odebere z výběru. Poslední událostí je *mouse\_out*, která volá funkci vracící element do původního stavu po události *mouse\_over*. Uživatel si může vybrat míst, kolik potřebuje. Všechny informace rezervační systém ukládá do předpřipraveného formuláře, který se při dokončení výběru odešle. Elementy, které jsou již zarezervované, jsou označené zvláštní barvou. Není na ně navázaná žádná z výše uvedených událostí a tudíž je tak jednoduše ošetřeno, že nejdou objednat. Na obrázku 2.3 vidíte sedadlo v kině při najetí kurzorem myši. Můžete si všimnout, že sedačka zobrazuje své číslo. Více názorných příkladů a ukázek najdete v kapitole Ukázka pluginu.



2.3

### Rušení rezervace

## HTML dokument ??

### Návrh dokumentu pro vložení pluginu

## Události

## O Pluginu – požádat o radu co vypíchnout

### Používané třídy ???

### Události ???

### jak se to chova

## Ukázka pluginu

### Použití pro kino

### Použití pro squash

# Použitá literatura

Neznámý autor

1. **Baudiš, Petr.** Výlet do říše verzí: CVS. *root.* [Online] 15. duben 2003. [Citace: 11. březen 2013.] http://www.root.cz/clanky/vylet-do-rise-verzi-cvs/.

2. —. Výlet do říše verzí: ideální verzovací systém? *root.* [Online] 3. květen 2004. [Citace: 11. březen 2013.] http://www.root.cz/clanky/vylet-do-rise-verzi-idealni-verzovaci-system/.

3. **Faigl, Jan.** Systémy pro správu verzí. [Online] [Citace: 11. březen 2013.] http://lynx1.felk.cvut.cz/pte/doc/vcs.pdf.

4. **Chacon, Scott.** Small and Fast. *git.* [Online] 29. červen 2009. [Citace: 11. březen 2013.] http://git-scm.com/about/small-and-fast.

5. **Teutsch, Moshe.** Introduction to PhpDoc. *phpmaster.* [Online] 9. leden 2012. [Citace: 11. březen 2013.] http://phpmaster.com/introduction-to-phpdoc/.

6. **neznámý, autor.** Project Home. *jsdoc-toolkit.* [Online] [Citace: 11. březen 2013.] https://code.google.com/p/jsdoc-toolkit/.

7. **Resig, John.** *jQuery Kuchařka programátora.* Brno : Computer Press a.s., 2010. ISBN 978-80-251-3152-7.

8. **Odell, Dem.** *JavaScript Průvodce programováním ajaxových aplikací.* Brno : Computer Press a.s., 2010. ISBN 978-80-251-2733-9.

9. **neznámý, autor.** Home. *Bootstrap.* [Online] [Citace: 15. březen 2013.] http://twitter.github.com/bootstrap/index.html.

10. **Vrána, Jakub.** Obrana proti SQL Injection. *PHP triky.* [Online] 2. březen 2005. [Citace: 15. březen 2013.] http://php.vrana.cz/obrana-proti-sql-injection.php.

# Příloha

## Ukázky zdrojových kódů

Zvýrazněni syntaxe