

**jQuery - vytvoření 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 Úvod 1](#_Toc352350117)

[1.1 Cíl práce 1](#_Toc352350118)

[2 Použité technologie 1](#_Toc352350119)

[2.1 HTML 5 a CSS 3 1](#_Toc352350120)

[2.2 PHP a MySQL 1](#_Toc352350121)

[2.3 Java Script 1](#_Toc352350122)

[2.4 JQuery (podrobný popis knihovny) 1](#_Toc352350123)

[2.5 Formát JSON 1](#_Toc352350124)

[2.6 Systémy pro správu verzí 1](#_Toc352350125)

[2.7 Standardy pro tvorbu komentářů 3](#_Toc352350126)

[2.8 knihovny 4](#_Toc352350127)

[3 Relevantní práce 4](#_Toc352350128)

[4 Tvorba pluginu jQuery 4](#_Toc352350129)

[4.1 Vytvoření pluginu a připojení ke knihovně 4](#_Toc352350130)

[4.2 Použití obalové funkce 5](#_Toc352350131)

[4.3 Defaultní parametry pluginu 6](#_Toc352350132)

[4.4 Soukromé funkce v pluginu 6](#_Toc352350133)

[4.5 Načítání dat ze souboru 7](#_Toc352350134)

[4.6 Formáty vstupu 7](#_Toc352350135)

[4.7 Obecná použitelnost 7](#_Toc352350136)

[4.8 Výstup pluginu 8](#_Toc352350137)

[5 Realizace pluginu pro rezervační systém 10](#_Toc352350138)

[5.1 Použití frameworku bootstrap 10](#_Toc352350139)

[5.2 Databáze 10](#_Toc352350140)

[5.2.1 Model databáze PHP 11](#_Toc352350141)

[5.2.2 Diagram databáze 12](#_Toc352350142)

[5.2.3 Konfigurační soubor pro databázi 14](#_Toc352350143)

[5.3 Funkce rezervačního systému 15](#_Toc352350144)

[5.3.1 Vygenerování všech elementů 15](#_Toc352350145)

[5.3.2 Přidání a odebrání elementů z vybraných položek a rezervace 16](#_Toc352350146)

[5.4 Popis ukázky 16](#_Toc352350147)

[5.4.1 Posílání zpráv – message - ?? 16](#_Toc352350148)

[5.4.2 Výběr města 16](#_Toc352350149)

[5.4.3 Výběr filmu 16](#_Toc352350150)

[5.4.4 Výběr sálu 17](#_Toc352350151)

[5.4.5 Výběr místa 17](#_Toc352350152)

[5.4.6 Rušení rezervace 18](#_Toc352350153)

[5.5 HTML dokument ?? 18](#_Toc352350154)

[5.5.1 Návrh dokumentu pro vložení pluginu 18](#_Toc352350155)

[5.6 Události 18](#_Toc352350156)

[5.7 O Pluginu 18](#_Toc352350157)

[5.8 Ukázka pluginu 18](#_Toc352350158)

[5.8.1 Použití pro kino 18](#_Toc352350159)

[5.8.2 Použití pro squash 18](#_Toc352350160)

[6 Použitá literatura 18](#_Toc352350161)

[7 Závěr 19](#_Toc352350162)

[8 Příloha 19](#_Toc352350163)

[8.1 Ukázky zdrojových kódů 19](#_Toc352350164)

# Úvod

## Cíl práce

# Použité technologie

## HTML 5 a CSS 3

html, html 5, css, css3

## PHP a MySQL

Php, mysql, normální formy

## Java Script

(podrobný popis) proc jQuery ?

## JQuery (podrobný popis knihovny)

## Formát JSON

## Systémy pro správu verzí

Přidat informace o tom, co to jsou verzovací systémy!

Používání systému pro správu verzí by mělo ulehčit práci na projektu. Nejvíce se to projeví v případě, že na projektu pracuje více vývojářů. Většina systémů pro správu verzí používá tzv. větve (branches). Větve rozdělují projekt do více částí, na kterých vývojáři mohou pracovat samostatně.

Systémů pro správu verzí je celá řada. Rád bych uvedl jen tři, které jsou podle mého názoru nejzákladnější:

* CVS
* SVN
* GIT

Nejstarším systémem je CVS (Concurrent Version System). Jde o pomyslné rozšíření RCS (Revision Control System) (1). CVS verzuje jednotlivé soubory zvlášť bez ohledu na ostatní. Soubory mohou být uloženy v různých adresářích. Adresář v tomto systému není brán jako soubor, a to má špatný vliv na celý systém. V CVS má adresář speciální postavení, soubory se ukládají a organizují podle toho, v jakém adresáři se nacházejí. To znamená, že adresář nejde dost dobře smazat, činí to zbytečné problémy při přesouvání souborů apod. Proto by měl 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. Další funkcí CVS je spravování a slučování větví. Provádí to ale neefektivním programem *diff3*, který po sloučení větví zapomíná informace o vedlejších větvích (2).

SVN (Subversion) nabízí již lepší funkčnost. Hlavní změnou, jak uvádí (3), je nakládání s adresářem. SVN na rozdíl od CVS nerozlišuje rozdíl mezi adresářem a souborem. Také lépe zvládá větvení projektu. Jeho nevýhodou, oproti systému GIT, je rychlost práci s daty.

Jako třetí systém pro správu verzí představuji GIT. Tento program je napsán v jazyce C. Podle (4) byla při vývoji GITu rychlost běhu hlavním cílem. V tabulce 3 převzaté z (4) je uvedeno porovnání běhu GIT a SVN. Z tabulky je zřejmé, že GIT pracuje ve všech testovaných případech rychleji. Porovnání je pouze přibližné, jak sám autor testů uvádí. Také GIT umí dobře pracovat s větvemi, dokáže je vytvářet i slučovat. Po sloučení větví nedochází ke ztrátě dat o vedlejší větvi.

V této práci používám systém GIT pomocí služby github. GITHUB je služba založená na systému GIT, který je možno zdarma používat pro veřejné projekty. Celý projekt je pak veřejně přístupný na webu github.com i se zdrojovými kódy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation |  | Git [s] | SVN [s] |  |
| 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 (4)

## Standardy pro tvorbu komentářů

Komentáře jsou důležitou součástí každého softwaru. Umožňují lepší orientaci autorovi i dalším programátorům, kteří budou kód používat. V této práci jsem se zde snažil používat různé druhy dokumentačních postupů. Postupy jsou závislé na tom, ve kterém jazyce jsou komentáře napsané.

Při psaní Modelu databáze jsem používal komentáře PhpDoc (celým názvem PhpDocumentor). Tento způsob je popsán v (5). Díky PhpDoc komentářům je možno vygenerovat přehlednou dokumentaci celého kódu. Používat lze nejen jednoduché texty, ale i tagy, jako je například seznam. Dokumentaci potom můžu 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 může vývojové prostředí našeptávat komentáře, jako nápovědu.

U samotného pluginu jsem se snažil co nejvíce přiblížit komentářům Jsdoc, které jsou popsané ve zdroji (6). Z komentářů napsaných tímto způsobem se dá rovněž vygenerovat dokumentace ve formátu HTML.

## knihovny

Twitter – bootstrap

PHP Form Builder Class

jQuery Tokeninput

jQuery Nivo Slider

# Relevantní práce

– najit pluginy ktery resi jen cast problemu

<http://techbrij.com/seat-reservation-with-jquery> obrazek

# Tvorba pluginu jQuery

V této kapitole se budu zabývat nejdůležitějšími problémy, se kterými jsem při psaní pluginu setkal. Zároveň zde popíšu, jak jsem je v práci řešil.

## 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 u pluginu pro rezervační systém ukládám do objektu *jQuery.fn.booking\_system* všechny potřebné vlastnosti pluginu. Ukázku rozšíření objektu *jQuery.fn* vidíme ve zdrojovém kódu 1.1.

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

Kód .

K vytvořenému pluginu pak přistoupíme 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 (7), se často využívá u přístupu k objektu jQuery jeho zkrácená verze zápisu $. Tento přístup, který používám i ve své práci, je výhodný z důvodů:

* Kód je kratší.
* Kód je čitelnější.

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. (8) tvrdí, že díky tomu je zajištěna rychlejší odezva stránky mezi začátkem stahování stránky a mezi tím, než stránka začne plně reagovat.

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

Protože je používání zkráceného zápisu oblíbené, může přinést řadu problémů. Například knihovna, která je přidána ke stránce před tímto pluginem, může zkrácený zápis $ využívat jako ukazatel na něco jiného, než objekt jQuery. Z tohoto důvodu jsem v práci použil obalovou funkci. Zdrojový kód obalové funkce je uveden v ukázce kódu 1.2.

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

Kód .

Je to 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. Ve funkci potom můžu $ libovolně používat bez vzniku kolize. Další výhodou tohoto postupu je, že definováním obalové funkce vytvořím uzávěr. Definici uzávěru uvádí (7). Uzávěr mi pomůže s možnou kolizí názvů metod tohoto a ostatních používaných pluginů.

Jazyk JavaScript si dokáže poradit se zapomenutým středníkem na konci řádku (7). K problému dochází v případě minimalizace kódu. Konce řádků se vypouští a může tak docházet k chybám. Proto je důležité na začátku pluginu napsat středník. Tím ošetřím případ, kdy programátor, který psal předcházející kód, na středník zapomněl.

## Defaultní parametry pluginu

Nejjednodušším řešením je přidávat nastavení pluginu, jako parametr funkce. Toto řešení ovšem nebude stačit v případě, 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* porovnám defaultní nastavení a nastavení od uživatele. Údaje, které uživatel neuvedl, metoda sama doplní defaultními údaji.

Výhodou tohoto přístupu je čitelnější kód. Pluginy většinou mívají hodně proměnných, které je potřeba nastavit. Tímto přístupem se vyhnu tomu, kontrolovat všechny proměnné, zda byly nastaveny. Jednoduše kontroluji jen to, co potřebuji. 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);

Kód .

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

Kód 1.4

## Private funkce a načítání dat ze souboru

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 již zmíněnou obalovou funkcí. Uvnitř obalové funkce jsem si definoval funkci, kterou jsem nepřipojil k objektu *jQuery.fn*. Tím zůstane funkce navenek skrytá, ale mohu ji využívat uvnitř pluginu.

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 proto, že načtení souboru, vytvořeného uživatelem, může být zdrojem mnoha chyb. Například daný soubor neexistuje, nebo data byla načtena ve špatném formátu. Zvolil jsem řešení načítat soubor do stránky, kde se dá jednoduše v náhledu zdrojového kódu stránky ověřit, zda se data opravdu data načetla a jsou správná.

## Formáty vstupu

Při zvolení formátů, ze kterých se má vstup načítat, jsem musel vybrat formát dat, který se dá uložit do souboru. Tento formát musel být zároveň známý, aby si uživatel mohl snadno vyhledat, jak do formátu své dat převést.

Pro načtení dat ze souboru jsem si tedy vybral formát CVS. Uživatel si tak může data navrhnout v MS Excelu a může je jednoduše nechat uložit v tomto formátu. Při práci s tímto tabulkovým procesorem jsem přidal ještě jeden podporovaný formát dat. Je jím XLS. Zjistil jsem, že při ukládání dat přímo v XLS (nikoliv v XLSX) se data ukládají do přehledného formátu. Pro editaci dat se i z tohoto formátu dobře načítají. Uživateli používající MS Excel se tak bude lépe pracovat.

Podporovaným formátem je i JSON. Jde o velmi používaný formát a proto jsem ho také začlenil do podporovaných formátů. V ukázce načítám data pro tento formát z databáze a pak je převádím do formátu JSON. Chtěl jsem tak v ukázce naznačit, že převod dat z databáze do formátu JSON. Převod je totiž díky dobré podpoře formátu od jazyka PHP snadný. S daty v databázi se lépe pracuje a myslím, že načítání dat z databáze by v ukázce určitě být mělo.

## Použitelnost

Při psaní pluginu jsem musel dbát na možnost jeho použití u různých rezervačních systémů. Je to zřejmé na příkladech kina a squashového kurtu, které jsem vytvořil, jako ukázky používání pluginu.

Důležité bylo vyhnout se přímému stylování objektů. V pluginu generuji celý systém přímo do stránky a zde nesmí být uveden žádný přímý styl. Přímé stylování by znemožnilo použitelnost pluginu na jakýkoliv jiný případ, protože 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 názvy tříd a identifikátorů volitelné. Mohlo by totiž dojít k tomu, že by kolidovaly s již použitými názvy na stránkách uživatele. Ten by pak musel své stránky nebo plugin ručně předělat. Aby uživatel nemusel nastavovat všechny třídy ručně, při přidávání 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. To objekt graficky mění. Například ho zprůhledňuje. Uživatel může s objektem spojit libovolnou jinou akci. Stačí v nastavení pluginu zadat jméno funkce, která  defaultní funkci v pluginu překryje. Zařadil jsem tuto problematiku do kapitoly Obecná použitelnost, protože uživatel si může napsat funkce nakládající s objektem, jak potřebuje. Jedinou podmínkou pro uživatele je přijímat objekt *events* jako parametr funkce, přes který se dostane k objektu, se kterým je manipulováno.

## Výstup pluginu

Výstup z pluginu se přímo vypisuje do HTML elementu podle daného identifikátoru id. Výpis je možný s použitím elementů table (tabulka) nebo div.

Výstup pomocí tabulky je výhodný v tom, že data se sami řadí do mřížky. Nemusím je jakkoliv ovlivňovat styly. Projeví se to obzvláště u generování více dat, které se pak řadí do řádků a sloupců. Toho můžu využít i pro řazení sedadel do řady. K sedadlům stačí přidat patřičné značky tabulky označující začátek řádku (v případě kina řady sedadel) a dalšího prvku (sedadla). Nevýhoda tohoto přístupu je v tom, že elementy nejde rozmístit nepravidelně. Tedy například chce-li uživatel rozmístit elementy do geometrického obrazce (například kruhu), musí jednotlivé elementy vyhledávat a stylovat každý zvlášť. Z pohledu struktury HTML dokumentu by v tabulce měly být pouze tabulková data, nikoliv celé elementy. Proto je tento přístup nevhodný.

Druhá možnost je ovlivnit rozmístění objektů pomocí značek div. V tomto případě každý element div obsahuje vlastní objekt (v případě kina sedadlo) a uživatel může objekt přesunout, kam potřebuje. Nevýhoda tohoto přístupu je zřejmá. Elementy div se automaticky neřadí do mřížky. Proto je nutné si pro ně vytvořit vlastní kaskádové styly.

Rozhodl jsem se používat značky div. V pluginu vytvářím skupiny objektů. V nich se jednotlivé objekty nachází. V případě kina si je můžeme představit, jako řady sedadel. V kulečníkové herně by to byly stoly. V této skupině se pak objekty mohou stylovat libovolně. Například u kulečníku je můžeme umístit okolo stolu do oválu. Tento případ by se využitím tabulky řešil velmi špatně.

# Realizace pluginu pro rezervační systém

V této kapitole se věnuji popisu návrhu a samotného provedení pluginu. – dopsat co tu opravdu bude

## Použití frameworku bootstrap

Bootstrap je front-end framework od Twitteru. Po vložení do kódu stránky přestyluje všechny základní HTML elementy. Také obsahuje mnoho předpřipravených tříd, jako jsou například třídy pro tlačítka. Vývojář díky tomu může psát pouze HTML kód. Bootstrap také nabízí řadu javascriptových funkcí, jako je například vytváření stylovaných vyskakovacích oken, nebo různé práce s textem. Bootstrapem se více zabývám v kapitole Použité technologie. Více informací je také uvedeno v (9).

Tady by chtělo uvést aspoň základní rysy + případně nějakej obrázek apod. O tomhle se bude dobře psát, tak toho využijte.

Při vytváření ukázky jsem se musel rozhodnout, zda použít pro stylování některý z dostupných frameworků, nebo si vytvořit vlastní styly. Jedním z hlavních kritérií rozhodování byla dobrá dokumentace. Ukázka pluginu má být intuitivní a uživatel by neměl přemýšlet nad ukázkou samotnou. Měl by se zaměřit hlavně na použití pluginu, aby ho mohl co nejdříve využívat na svém webu. To jsou důvody, proč jsem zvolil bootstrap framework. Má velmi kvalitní dokumentaci všech elementů. V dokumentaci je nejen popis stylování, či javascriptu, ale také ukázka jejich použití. Uživatel si tedy může velmi rychle vyhledat informace o tom, s čím právě pracuje.

Z frameworku využívám ve své práci převážně předstylované HTML tagy. Ty používám na celkový vzhled ukázky a ve většině náhledů. Na stylování samotné stránky, kde jsem používal rezervační systém, jsem musel přidat i vlastní styly. Potřeboval jsem ovlivnit konkrétní věci, jako je třeba řazení a pozice sedaček v kině. Díky použití vlastních stylů jen na konkrétní výjimečné situace jsem dosáhl toho, že vlastní styly jsou velmi krátké a uživatel se v nich lépe vyzná.

## Databázová vrstva – realizace s využitím PHP

Při vytváření ukázky použití pluginu bylo důležité dobře realizovat přístup k databázi. Přístup 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.

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

Při tvorbě ukázky jsem nejdříve celý objekt napsal podle návrhového vzoru jedináček. Tedy konstruktor je soukromý a má privátní statickou metodu, která vrací novou, nebo již vytvořenou instanci. Výhodou je, že stačí přes metodu vrátit objekt. Uživatel se nemusí starat o to, zda již připojení v kódu někde použil a je tudíž vytvořené. Uživatel tak má vždy jeden objekt, ke kterému může přistupovat. Toto řešení jsem nevyužil z důvodu, že ukázka má být v první řadě jednoduchá a dobře čitelná. To při použití modelu jedináček nebyla. Uživatel se totiž musel seznámit s napsáním modelu, aby pochopil, jak přistupovat k objektu. Objekt si v ukázce vytvořit nemohl, protože konstruktor byl podle návrhového vzoru privátní. Proto vytvářím objekt 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 dosáhnout 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 jsou konkrétní a každý nový druh přístupu k databázi znamená novou metodu. Například pro přihlášení má objekt metodu *signIn* s parametry email a heslo. Ukázka zdrojového kódu metody je v ukázce 2.1. Metody jsou chráněny před SQL injection. To znamená, že uživatel nemá možnost 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. Tento druh ochrany je vhodný 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();  
}

Kód .

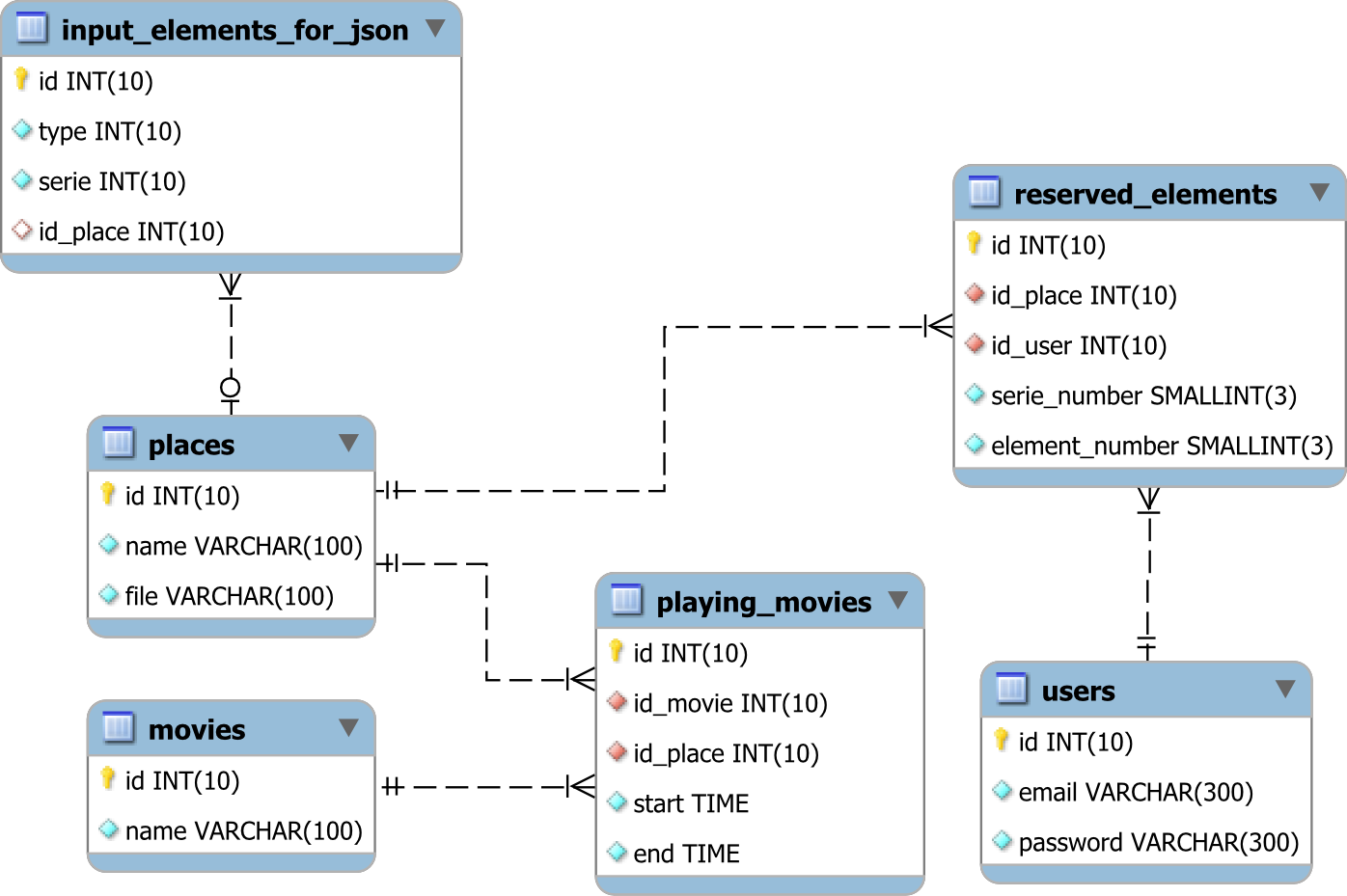
Objekt obsahuje metody na přihlášení a odhlášení k databázi. Tyto metody nemají žádné parametry. Potřebné hodnoty se inicializují už při vytváření objektu. Metody jsou veřejné, aby si je uživatel mohl volat přímo v kódu. Další metody jsou *query* a *fetch*. Tyto metody jen obalují funkce *mysql\_query* a *mysql\_fetch\_object* a řeší případné výjimky.

Metoda, kterou bych chtěl podrobněji popsat, 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 ukázce, 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 pouze k tomu, aby se uložily výsledky z databáze do pole *$rs*. To cyklus vykoná již v podmínce. Když funkce *mysql\_fetch\_assoc* nemá další data, která by mohla převést do asociativního pole a uložit do proměnné *$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);  
}

Kód .

### Diagram databáze



.

Na obrázku 2.1 vidíme diagram databáze používané v ukázce pluginu. Diagram obsahuje pouze vazby N:1. Databáze je ve třetí normální formě. Všechny tabulky obsahují primární klíče, které jsou zastoupeny maximálně desetimístným číslem. Číslo 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 identifikátorem promítaného filmu. Sloupec *id\_place* je identifikátorem sálu, kde se film promítá. Sloupec *id\_place* je v ukázce důležitý hlavně proto, aby se podle sálu 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*. V případě uvedení této ukázky do praxe, by tato tabulka jistě musela být 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, kdy je nastaven vstup ze souboru. Ve sloupci *file* je uvedeno jméno souboru, ze kterého se má vstup načítat. Každé místo má v ukázce právě jeden vlastní soubor, 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, zda jde o jednomístnou sedačku, nebo dvojmístnou sedačku. Sloupec *serie* označuje řadu, ve které se sedačka nachází. Ve sloupci *id\_place* je uvedeno, kterému sálu daná sedačka náleží. Tato tabulka je přes klíč *id\_place* spojená s tabulkou *places*.

**reserved\_elements** – Používá se pro zarezervování elementu. V případě kina si můžeme element představit, jako sedadlo. Kdyby se data načítala pouze z formátu JSON, obsahovala by tabulka pouze identifikátor elementu z tabulky *input\_element\_for\_json*. Protože ale musí existovat i možnost načíst vstup ze souboru, v tabulce musí být informace pro jednoznačné určení elementu. Při načítání dat ze souboru se data neukládají do databáze, ale přímo se generují do stránky. Ve sloupci *serie\_number* je informace o sérii prvku. U 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. Používám ji nejen při registraci a přihlášení, ale také k identifikaci uživatele, který si daný element zarezervoval. Tabulka je spojena s tabulkou *reserved\_elements* pomocí primárního klíče. Ve sloupci *email* je uveden email uživatele. Ve sloupci *password* je hashované heslo uživatele.

### Konfigurační soubor pro databázi

V konfiguračním souboru jsou uloženy veškeré konstanty používané v ukázce. Vytvořil jsem ho proto, aby bylo možné vše dohledat a měnit na jednom místě. Nachází se v něm názvy všech tabulek, uložené jako konstanty. To má speciální význam. Díky tomu se dají jednoduše měnit názvy tabulek a program zůstane stabilní. Uživatel chce například změnit název tabulky *movies* na *filmy*. Změnu stačí udělat pouze v tomto konfiguračním souboru a změny se projeví v celé ukázce. Kromě názvů tabulek soubor obsahuje i údaje potřebné pro přihlášení k databázi. V ukázce se jedná o soubor *config.inc.php*.

## Funkce rezervačního systému

### Vygenerování všech elementů

Po načtení dat ze vstutpu a převední z různých formátů do dvourozměrného pole se spustí generování všech elementů do stránky.

Nejdřív musím projít obě pole. Pro tento průchod slouží konstrukce *$.each*. Tato funkce je obsažena přímo v knihovně jQuery. Slouží podobně jako konstrukce *foreach* z jazyka PHP. Prochází pole a nabízí jeho jednotlivé prvky. Má dva parametry. První je pole, které má procházet a druhý je funkce, která s prvkem bude pracovat. Této funkci předá pomocí parametrů index prvku a jeho hodnotu. Pro procházení druhého rozměru pole tuto funkci použiji ještě jednou. Zde už ji ale místo parametru pole předám parametr *this*. Ten obsahuje … kouknout se jak je to v knizce

V ukázce zdrojového kódu vidíme použití konstrukce $.each při procházení druhého rozměru pole.

$.each(this, function(index1, value){  
 // zde bude volání funkce pro vytvoření elementu  
}

* Doplnit popisek

Při každém průchodu prvního rozměru pole vytvářím novou sérii elementů (v případě kina si sérii můžeme představit jako řadu sedadel). Do té pak následující elementy umísťuji. Každá série má vlastní identifikátor, aby se mohla jednoznačně určit. Pomocí tagů DIV obaluje všechny prvky v ní obsažené.

Když procházím elementy, nad každým z nich volám funkci createElement. Ta podle parametrů rozhodne, o jaký element se jedná (v případě kina o jednosedačku či dvojsedačku) nebo je element prázdný (například ulička). Dále přiřadí elementu pořadové číslo. To slouží hlavně pro ukázku kina, kde potřebuji zobrazit číslo sedadla. Důležité je vědět, že číslo sedadla není číslo elementu. V případě, že se vykresluje ulička se nesmí číslo zvýšit. Pak přiřadím element poslední vložené sérii. Poslední sérii najdu pomocí konstrukce *TŘÍDA:last*. … jak funguje atribut last z knížky

### Přidání a odebrání elementů z vybraných položek a rezervace

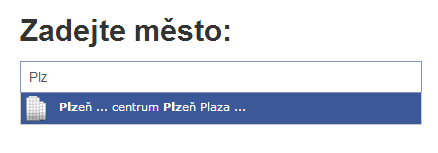
## Popis ukázky

V této části popíšu, co všechno ukázka obsahuje.

### Posílání zpráv – message - ??

### Výběr města

Nejdřív musí uživatel v ukázce zvolit město. Možnost volby města je k dispozici 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 doplnit. *Tokeninput* potom pošle proměnnou s názvem města další stránce. Stránka ho použije pro výběr kina v daném městě. Zajímavou vlastností je i to, že *Tokeninput* používá obrázky vedle své nápovědy, pro lepší orientaci. Více se o tomto pluginu rozepíšu v kapitole Použité technologie. Na obrázku 2.2 vidíme ukázku našeptávání.



2.2

### Výběr filmu

Na této stránce používám galerii obrázků. Tu zajišťuje plugin *Nivo Slider*, který umožňuje jednotlivé obrázky procházet a automaticky galerii promítá. Více o pluginu *Nivo Slider* je uvedeno v kapitole Použité technologie. Dále na stránce najdeme seznam všech právě promítaných filmů. U jednotlivých filmů jsou pak uvedeny informace o tom, kdy se začínají promítat a čísla sálů, ve kterých se promítají.

### Výběr sálu

Výběr sálu probíhá spolu s výběrem filmu. Pro lepší ukázku možností pluginu jsem 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 kapitole Použité technologie.

### Výběr místa

Výběr místa uživatel najde na stránce rezervace. Výběr místa zajišťuje plugin pro rezervační systém. Plugin při vygenerování všech elementů naváže na každý element několik událostí. První je událost *mouse\_over***,** tedy přejetí myší nad elementem. Podle zadání má být kladen důraz na grafické provedení. Proto při přejetí kurzorem myši element plynule mění barvu. Druhá událost je klepnutí myší na element. Ten po této akci opět vyvolá změnu barvy a zároveň zavolá funkci pluginu *addToSelected*. Funkce má jediný parametr *e* značící událost (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*. Zavolá funkci, která vrací element do původního stavu po události *mouse\_over*. Uživatel si může vybrat tolik míst, kolik potřebuje. Všechny informace rezervační systém ukládá do předpřipraveného formuláře. Formulář při dokončení výběru odešle. Elementy, které jsou již zarezervované, mají zvláštní obrázek. Není na ně navázaná žádná z výše uvedených událostí a je tak jednoduše ošetřeno, že nejdou rezervovat. Na obrázku 2.3 je zobrazeno sedadlo v kině při najetí kurzorem myši. Z obrázku je patrné, že sedačka zobrazuje své číslo. Více názorných příkladů a ukázek najdete v kapitole Ukázka pluginu.

Zamyslet se nad lepším obrázkem



2.3

### Rušení rezervace

## HTML dokument ??

### Návrh dokumentu pro vložení pluginu

## Události

## O Pluginu

Používané třídy

Události

jak se to chova

## Ukázka pluginu

### Použití pro kino

### Použití pro squash

# Použitá literatura

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.

# Závěr

# Příloha

## Ukázky zdrojových kódů

Zvýraznění syntaxe