Univerzita Pardubice

Fakulta elektrotechniky a informatiky

WWW formulář pro aktivní verifikaci dat vkládaných do databáze

Jiří Lebduška

Bakalářská práce

2008

SOUHRN

Práce je věnována problematice zpracovávání dat v prostředí webového rozhraní. Hledá a vytváří způsoby jak dosáhnout interaktivity v komunikaci s databází a zajistit zkvalitnění vkládaných dat.

KLÍČOVÁ SLOVA

databáze, webový formulář, Ajax, integrita dat

TITLE

Active WWW form for data verification

ABSTRACT

This thesis deals with a data processing in a sphere of web interface. It tries to find and create the methods which will ameliorate interactivity in a sphere of communication within the database. Among the main purposes is also a better quality of the inserted data.

KEYWORDS

database, form, Ajax, integrity

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

WWW (World wide web - Celosvětová pavučina). Označení pro aplikace internetového protokolu HTTP.

HTTP (Hypertext transfer protocol). Protokol aplikační vrstvy. Zprostředkovává komunikaci mezi uživatelem (prohlížečem) a webovým serverem.

POSIX Portable Operating System Interface. Definuje standardy přenositelného rozhraní pro různé operační systémy.

IEEE The Institute of Electrical and Electronics Engineers. Institut pro elektrotechnické a elektronické inženýrství, který se mimo jiné zabývá vývojem standardů.

SQL Structured query language. Strukturovaný dotazovací jazyk vyvinutý pro práci s daty v relačních databázích.

PL/SQL Procedural Language/Structured Query Language. Procedurální nadstavba nad SQL jazykem.

Ajax Asynchronous JavaScript and XML. Technologie používaná pro vývoj interaktivních webových aplikací.

XML eXtensible Markup Language. Rozšiřitelný značkovací jazyk. Jazyk určený k výměně dat mezi různými aplikacemi a různými platformami.

HTML Hypertext Markup Language. Jazyk používaný k definici struktury, vzhledu a funkce hypertextových dokumentů, zejm. webových stránek.

ASP Active Server Pages. Technologie vyvinutá Microsoftem pro generování dynamického obsahu webových stránke. Vykonává kód na straně serveru, uživateli odešle výsledek.

MS Microsoft. Americká společnost zabývající se vývojem softwaru.

W3C (WWWC) World Wide Web Consortium. Konsorcium pro standardizaci internetových technologií.

PHP Hypertext Preprocessor. Multiplatformní skriptovací programovací jazyk, zpracovávaný na straně serveru.

PSČ Poštovní směrovací číslo. Je součástí adresy.

DNS Domain Name System. Systém doménových jmen.

1 Úvod

Žijeme v době, kdy se s informačními technologiemi setkáváme takřka na každém kroku. Vývoj v této oblasti značně pokročil. Ovládání ve smyslu zadávání a následného zpracovávání dat již není tak složité, jako v době děrných štítků, kdy velký sálový počítač ovládali specialisté ve svém oboru. Dnes je počítač v běžném kancelářském inventáři, ale i v domácnostech, a obsluhovat ho může prakticky každý. Uživatel je doslova hýčkán všemi různými funkcemi, které usnadňují jeho práci (např. jednoduchá úprava dokumentů, automatická kontrola pravopisu). Tím se značně zvyšuje produktivita jeho práce.

Téma mé bakalářské práce je „WWW formulář pro aktivní verifikaci dat vkládaných do databáze“, což souvisí s tím o čem jsem hovořil. Databázi většinou navrhuje specialista, který je podrobně obeznámen se strukturou a formátem dat. Naproti tomu vkládáním či úpravou dat může být pověřen kdokoliv. Kdokoliv, kdo je zvyklý „být hýčkán“. Cílem mojí práce je vytvořit jakýsi framework, jehož aplikací na jakoukoliv Oracle databázi, poskytnu rozhraní, které nejenže usnadňuje vkládání dat, ale zároveň informuje o jejich případné nesprávnosti, či nevhodnosti.

Díky tomu lze minimalizovat rizika vložení nesprávných, tj. nereálných, či nesprávně uvedených dat. Usnadní se tím i práce s jejich pozdějším zpracováváním.

Jako model poslouží evidence osob (např. z oblasti veřejné správy), kde je použití takovéhoto rozhraní více než vhodné.

Pokusím se nastínit některé způsoby získávání většího souboru dat, která poslouží jako šablona pro našeptávače vstupních polí.

2 Analýza problému

Jak již bylo řečeno v úvodu, databázi navrhuje databázový specialista. Ten využívá různé databázové nástroje k zajištění integrity dat. Je obeznámen s formátem a strukturou dat, která je třeba evidovat a na základě těchto informací navrhne databázi.

K datům ve struktuře databáze je přistupováno přes různá rozhraní, ať už přes systémové aplikace, nebo přes webovou stránku. Tato rozhraní navrhuje vývojář. Je vhodné, aby vývojář obdržel dokumentaci k databázi, z níž může čerpat při návrhu rozhraní.

Koncový uživatel (obsluha programu) ovšem nezná strukturu databáze a dat do ní vkládaných. Nezná všechna nastavená omezení, která zabraňují vložení byť jen špatně interpretovaných dat. Není to jeho povinností. Na druhou stranu je vyžadováno, aby jím vkládaná data byla v požadovaném formátu a zapadala do již navrženého kontextu.

2.1 Problém zabezpečení správnosti dat

2.1.1 Na úrovni databáze

Z pohledu databáze lze datovou integritu zabezpečit pomocí různých nástrojů (viz kapitola Integritní omezení), aby vkládaná data byla upravena do požadovaného formátu, je-li to možné, popřípadě aby nebyla vložena vůbec. V takovém případě je vhodné upozornit na situaci chybovým hlášením. Využití databázových nástrojů znamená nejvyšší úroveň zabezpečení, neboť právě databáze je ústím toku dat ze všech rozhraní.

2.1.2 Na úrovni aplikace (uživatelského rozhraní)

Na úrovni aplikace je samozřejmě možné zabezpečit ověřování správnosti dat. Je to ovšem programátorsky náročnější, a pokud se změní omezení v databázi,

je třeba upravit i aplikaci a redistribuovat update všem uživatelům. Řešením může být dynamické využití informací z databáze.

2.1.3 Na úrovni uživatele (obsluhy rozhraní)

Zabezpečení na této úrovni je založeno na pouhé důvěře v uživatele. Je možné zpřístupnit uživateli manuál jak a v jakém formátu vyplňovat jednotlivá pole, ovšem tato politika poskytuje nejmenší úroveň zabezpečení, pokud lze o zabezpečení hovořit.

2.2 Interaktivní rozhraní

Dále je třeba řešit otázku, jak přenést informace o tom, jaká data a v jakém formátu jsou vyžadována, jak do aplikace, tak k uživateli. Databáze samozřejmě nabízí všechna potřebná data ke zmíněné problematice. Problém je v tom, jak z nich vyčíst co největší množství informací a jak je dále prezentovat.

Díky využití vhodné kombinace různých technologií (viz kapitola Použité techniky) lze zabezpečit, aby byl uživatel informován o aktuálně vkládané položce a jejím požadovaném formátu.

2.3 Řešení

Výše uvedené problémy by měla pomoci analyzovat a řešit tato bakalářská práce. Jejím výsledkem není pouhý soubor skriptů ošetřující konkrétní formulář, ale univerzální balík, kde pouhým parametrizováním lze dosáhnout stejného výsledku jako v modelovém příkladu nad jakoukoliv Oracle databází. Takový balík se dá označit jako framework. Jeho předností je jednoduché a hlavně univerzální použití.

3 Použité techniky

Vhodným využitím a kombinací programovacích technik lze vytvořit poměrně inteligentní systém i v takovém prostředí jako web (původně určený k publikaci statických textů).

Ambice frameworku vyžadují přehledný a okomentovaný kód, který by měl být dobře čitelný. Rovněž je preferován objektový přístup, který nejenže celý kód zpřehlední, ale rovněž zjednoduší systém přístupu k jednotlivým metodám a atributům (9).

3.1 Regulární výrazy

Regulární výrazy nalezly v této práci mnohé uplatnění. Ať už jako kontrola formátu vstupních dat, nebo jako výborný pomocník pro třídění, vyhledávání a úpravu starých dat.

3.1.1 Co to jsou regulární výrazy

Regulární výrazy (zkráceně regexp, regex podle anglického „regular expression“) jsou speciálním řetězcem znaků, který představuje určitý vzor (masku) pro textové řetězce (i1, i4). Využívají syntaxi vyvíjenou řadu let, která je součástí POSIX standardu.

Dávají odpověď na otázku „Jak lze popisovat vzor textu?“. Regulární výrazy se poprvé staly široce používané na Unixové platformě, podporované takovými utilitami jako ed, grep a jazyk Perl. Dále začaly být regulární výrazy formalizovány v IEEE POSIX standadru a jsou široce podporované napříč stále rostoucí řadou vydavatelů, emailových klientů, programovacích a skriptovacích jazyků (C#, Java, Visual Basic .NET, Perl, PHP, JavaScript aj.) a nyní i Oracle SQL a PL/SQL (od verze 10g).

3.1.2 Syntaxe

Regulární výraz se skládá z literálů (reprezentace posloupnosti znaků nebo symbolů) textu (i2), které se mají shodovat, a speciálních znaků, sloužících pro popis alternativ, množin, počtů výskytů a přepínačů, které nejsou součástí hledaného textu (i1).

Obyčejné znaky

Pojmem znak je myšlen v „regulárním jazyce“ každý znak, jenž není znakem nového řádku. Obyčejné znaky uvnitř regulárního výrazu reprezentují samy sebe.

Speciální znaky

Speciální znak je takový znak, který sám sebe nereprezentuje. Používají se k reprezentaci celé množiny řetězců. Pokud jej chceme použít jako znak obyčejný, je třeba před něj umístit zpětné lomítko (existují vyjímky – cifry, závorky).

Tečka .

Znak tečka (.) odpovídá kterémukoliv jednomu znaku.

**Zpětné lomítko **

Použitím zpětného lomítka (\) před následujícím speciálním znakem, se docílí navrácení původního významu tomuto znaku (jako by byl znakem obyčejným).

Literární znaky

Syntaxe regulárních výrazů využívá některé neabecední znaky prostřednictvím sekvencí escape, které začínají zpětným lomítkem (\) (3).

Znak	Představuje
Alfanumerický znak	Sám sebe
`\0`	Znak `NULL`
`\t`	Tabulátor
`\n`	Nový řádek
`\v`	Svislý tabulátor
`\f`	Posuv stránky
`\r`	Návrat kurzoru
`\xnn`	Latinský znak zadaný šestnáctkovým číslem
`\uxxxx`	Znak sady Unicode zadaný šestnáctkovým číslem

Tabulka 3.1: Literálové znaky regulárních výrazů

Skupiny znaků

Hranaté závorky

Hranaté závorky definují třídu znaků. Každý znak této třídy odpovídá jednomu znaku. Jestliže je za levou hranatou závorkou jako první uveden znak ^, pak ostatní znaky v hranatých závorkách reprezentují třídu znaků, které v hranatých závorkách uvedeny nejsou (znak ^ zde má tedy význam negace). K definici rozsahu znaků se používá pomlčka (-).

Kvantifikátory

Kvantifikátory se vždy uvádí za množinu znaků, jejíž četnost specifikují. Množinou znaků může opět být jak výčet znaků pomocí hranatých závorek, předdefinovaná množina, tak i samotný znak (i3).

Otazník ?

Znak otazník (?) bezprostředně následující za množinou znaků určuje, že tato množina (či znak) se v řetězci vyskytuje právě jednou, nebo vůbec.

Hvězdička *

Hvězdička (*) může následovat za jakýmkoliv regulárním výrazem, jenž reprezentuje jediný znak. Hvězdička odpovídá nulovému nebo vícenásobnému výskytu zadaného regulárního výrazu. Hvězdička následující za znakem tečka odpovídá jakémukoliv řetězci. Definice třídy znaků následovaná hvězdičkou odpovídá jakémukoliv řetězci znaků, které se vyskytují v definici třídy.

Plus +

Odpovídá jednomu a vícenásobnému výskytu zadaného regulárního výrazu.

Složené závorky { }

Pomocí složených závorek lze numericky vymezit počet opakování. Jedním číslem uzavřeným do složených závorek se definuje přesný počet opakování definované třídy znaků. Dvěma čísly oddělenými čárkou se určí rozsah počtu opakování. Jedním číslem následovaným čárkou se určuje minimum, přičemž maximální počet není stanoven.

Hranice (ukotvení)

Znak stříška ^

Jestliže obsahuje regulární výraz znak stříška (^), může odpovídat řetězci pouze na začátku řádku.

Znak dolar $

Regulární výraz se znakem dolar ($) odpovídá pouze řetězci na konci řádku.

3.1.3 Příklad použití

O síle regulárních výrazů svědčí mj. fakt, že díky jejich kompatibilitě a zmíněné široké podpoře lze stejný regulární výraz použít jak na straně serveru, či klienta, tak i přímo v databázi.

Pro tvorbu složitějších regulárních výrazů lze na Internetu stáhnout návrhové programy či použít on-line testery.

Kontrola rodného čísla

Jako názorná ukázka poslouží regulární výraz představující vzor (masku) pro kontrolu rodného čísla, který je dále v této práci použit.

Rodné číslo se skládá ze šesti numerických znaků, dále lomítka a dalších čtyřech numerických znaků. Jeden z možných vzorů by mohl vypadat takto:

```
^[0-9]{6}/[0-9]{3,4}$
```

Kotvící znaky na začátku (stříška) a konci (dolar) výrazu určují, že na řádku musí být pouze řetězec dle uvozeného vzoru. Pomocí hranatých závorek je vytvořena třída, která určuje rozsah platných znaků. Kvantifikátorem ve složených závorek je určen počet znaků z rozsahu. Dále následuje lomítko, to zastupuje samo sebe a odděluje od sebe dvě numerické posloupnosti. Je tedy patrné, že část za lomítkem má stejný význam jako před ním až na kvantifikátor.

Bohužel významovou správnost rodného čísla (viz. kapitola 6.1.3) pouhým testováním regulárním výrazem docílit nelze.

3.2 Ajax

3.2.1 Co je to Ajax

Jde o zkratku Synchronismus JavaScript and XML za níž se skrývá technologie vývoje interaktivních webových aplikací, které dokáží změnit obsah svých stránek bez nutnosti jejich znovunačítání.

Termín „asynchronous“ značí, že prohlížeč nečeká na odpověď od serveru, ale zpracuje ji až v okamžiku, kdy ji skutečně obdrží. Přenosy dat se odehrávají na pozadí, aniž by musel prohlížeč pozastavit prováděné operace a na něco čekat (aplikace není blokována) (i8).

JavaScript je primární komponentou. Zprostředkovává spojení se serverem, následně přijme odpověď, na kterou dokáže reagovat v prostředí webu.

Aplikace využívající Ajax se vytvářejí tak, aby zpracovávaly odpověď ze serveru ve formátu XML. Čili server předává webové aplikaci XML dokument. Ovšem Ajax dokáže přijmout i obyčejný text (6).

3.2.2 Historie

Vše odstartoval objekt XMLHttpRequest, a to v Internet Exploreru 5 (v roce 1999) jako ovládací prvek Active X. Dále byl v pozměněné podobě podporován Mozillou 1.0 a Safari 1.2 a mohlo dojít k jeho rozšíření. Do té doby se používalo technologie zvané IFRAMEs (stránka je rozdělena do podstránek „FRAMEs“ a ty se mohou jednotlivě obměňovat). K masovému rozšíření Ajaxu došlo po představení Gmailu, Google Maps, Google Suggest, založených právě na této technologii (5).

3.2.3 Výhody / nevýhody

Výhody:

- není nutné znovunačítání stránky při každém volání serveru, což snižuje zátěž na serveru

- tato technika poskytuje široké možnosti při práci na webu – webové prostředí se stává interaktivní (podobně jako desktopové aplikace)
- tuto technologii si lze osvojit a začít využívat ihned, bez nutnosti jakékoliv instalace či učení se novým programovacím jazykům

(i8)

Nevýhody:

- při nevhodné implementaci dochází vlivem častého dotazování k velké zátěži na server
- někdy malá rychlost odezvy v prostředí Internetu může uživatele mást a odradit od používání takové aplikace

(i8)

3.2.4 Použití

Hlavním kandidátem na použití Ajaxu je vyplňování a ověřování formulářů, kdy server může ověřit data ještě před jejich konečným odesláním.

Vhodným kandidátem je jistě již zmíněný Google Suggest (podobná technologie je použita i v této bakalářské práci), do češtiny překládaný jako „našeptávač“. Usnadňuje uživatelům vyplňování formulářových polí ve formě nabídek relevantních hesel.

Mezi novinky patří tzv. „drag and drop“ objekty, někdy překládané jako „táhni a pusť“. Díky nim získávají mnohé webové portály o mnoho více dynamičnosti.

3.3 Java Scripty, aktivní formuláře

3.3.1 Co je to JavaScript

JavaScript je interpretovaný objektově orientovaný programovací jazyk. Nejčastěji se používá jako interpretovaný programovací jazyk pro webové stránky, vkládaný přímo do HTML kódu stránky (i5).

Pro webové programování bylo jádro jazyka rozšířeno přidáním objektů reprezentujících okno webového prohlížeče a jeho obsah. To umožňuje vložit do webových stránek proveditelný obsah. Stránka tedy již nemusí být jen statickým dokumentem HTML, ale může obsahovat dynamické programy, které komunikují s uživatelem, řídí prohlížeč a dynamicky vytvářejí obsah HTML (3, i5).

Program v JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, vytvářet okna mimo obrazovku, číst ani měnit uživatelská nastavení prohlížeče (3, i5, 8).

3.3.2 Syntaxe

Jeho syntaxe patří do rodiny jazyků C/C++/Java. Je jazykem bez typové kontroly, což znamená, že proměnné nemusí mít specifikovaný typ. Rozlišuje velká a malá písmena. Klíčová slova se zapisují malými písmeny. Příkazy se oddělují středníkem, nebo koncem řádku (i5).

3.3.3 Interakce s formuláři HTML

Důležitým aspektem klientského JavaScriptu je jeho schopnost pracovat s formuláři HTML. Tuto schopnost zajišťuje objekt Form a objekty prvků formuláře. Formulářové prvky umožňují číst a zapisovat hodnoty vstupních prvků ve formuláři v dokumentu.

Jiné obvyklé využití klientského JavaScriptu ve spojení s formuláři spočívá v kontrole dat formuláře před jejich odesláním. Pokud je JavaScript na straně klienta schopen provést veškerou potřebnou kontrolu chyb uživatelského vstupu, pak není zapotřebí kontaktovat server, který by musel uživatele upozornit na triviální chyby zadání (3).

3.3.4 Objekty pro práci s regulárními výrazy

Regulární výrazy v JavaScriptu představuje třída RegExp a jak String tak i RegExp definují metody využívající regulární výrazy k vykonání důležitých funkcí hledání vzoru a hledání náhrady znaků v textu (3).

Metody objektu String pro práci s regulárními výrazy

Řetězce podporují čtyři metody využívající regulární výrazy. Nejjednodušší je `search()`. Tato metoda přebírá jako argument regulární výraz a vrací buď pozici znaku první nalezené shody nebo hodnotu `-1`, pokud nebyla shoda nalezena.

Metoda `replace()` vykoná operaci hledání a náhrady. Přebírá regulární výraz jako první argument a náhradní řetězec jako druhý argument. Hledá v řetězci, s nímž je volána, shody podle zadaného vzoru. Má-li regulární výraz nastaven příznak `g`, pak metoda `replace()` nahradí všechny shody v řetězci náhradním řetězcem, jinak nahradí jen první nalezenou shodu.

Metoda `match()` přebírá jako argument pouze regulární výraz a vrací pole všech shod nalezených v řetězci.

Metoda `split()` rozděluje řetězec, s nímž je volána, na pole podřetězců, přičemž jako oddělovač používá svůj argument.

Metody objektu RegExp

Regulární výrazy v JavaScriptu jsou reprezentovány tímto objektem. Konstruktor `RegExp()` přebírá jeden nebo dva argumenty a vytváří nový objekt `RegExp`. Prvním argumentem tohoto konstruktoru je regulární výraz. Druhý argument konstruktoru je volitelný. Je-li zadán, pak indikuje příznaky regulárního výrazu (`g`, `i` , `m`), lze použít i kombinaci.

Objekty `RegExp` definují dvě metody vykonávající operace hledání vzorů, chovající se podobně jako metody objektu `String`. Hlavní metodou hledání vzoru objektu `RegExp` je `exec()`. Tato metoda přebírá řetězec a hledá v něm shodu s regulárním výrazem zadaným v konstruktoru. Pokud shodu nenalezne, vrátí `null`. Pokud ovšem shodu nalezne vrací v poli na indexu 0 podřetězec odpovídající

regulárnímu výrazu. Není-li nastaven příznak `g` (neglobální hledání), vrací, podobně jako `match()`, pole které obsahuje všechny shody.

Další metodou `RegExp` je `test()`. Tato metoda přebírá řetězec a vrací `true`, pokud řetězec odpovídá regulárnímu výrazu, v opačném případě vrací `false`.

Výhodou použití objektu `RegExp` a jeho metod oproti metodám `Stringu` je, že `RegExp`, mimo jiné, má vlastnost `lastIndex`, která určuje pozici prvního znaku za koncem nalezené shody, v případě globálního hledání. Jinými slovy, při prohledávání textu narazí např. metoda `exec()` na shodu, vrátí shodující se podřetězec a nastaví `lastIndex` na první znak za tímto podřetězcem v prohledávaném textu. Při opětovném volání této metody bude text prohledáván od této pozice, neurčíme-li jinak (3).

Využitím této vlastnosti lze např. upozornit na konkrétní znak, který neodpovídá regulárnímu výrazu.

3.3.5 Objekt XMLHttpRequest

XMLHttpRequest (XHR) je rozhraní umožňující webovým aplikacím komunikaci mezi serverem a klientem prostřednictvím protokolu HTTP. Nejčastěji je tento komunikační prostředek používán v interaktivních webových aplikacích založených na technologii AJAX umožňující změnu části obsahu stránky zobrazené v prohlížeči klienta bez nutnosti jejího opětovného kompletního načtení ze serveru.

Objekt XMLHttp byl původně implementován v Internet Exploreru 5 jako komponenta ActiveX. Vývojáři v open source projektu Mozilla začali vyvíjet svoji vlastní XMLHttp. Namísto toho však, aby využívali proprietární technologie MS objektů ActiveX – vývojáři z týmu Mozilla použili podobná jména principiálních metod vlastností ve vlastním objektu webového prohlížeče, XMLHttpRequest. Microsoft se rozhodl díky velké popularitě XMLHttpRequest objektu ho zařadit do svého prohlížeče MS IE 7 (i6, 3).

Objekt XMLHttpRequest není W3C standard proto se jeho chování a funkcionalita může v různých prohlížečích lišit. Ovšem W3C připravuje specifikaci pro XMLHttpRequest, která má sjednotit jeho chování mezi webovými prohlížeči.

Metody a atributy

Metody

Tabulka 3.2: Standardní metody XMLHttpRequest (3) ukazuje, některé nejčastěji používané metody objektu XMLHttpRequest.

Metoda	Popis
`abort()`	Přeruší aktuální požadavek
`getAllResponseHeaders()`	Vrátí všechny hlavičky http požadavku ve formě párů klíč/hodnota.
`getResponseHeader('hlavička')`	Vrátí hodnotu zadané hlavičky ve formě řetězce.
`open('metoda','url')`	Nastaví parametry volání serveru. Argument `metoda` může nabývat hodnot `GET`, `POST`, `PU`. Argument `url` představuje požadované URL, které může být relativní nebo absolutní.
`send(obsah)`	Odešle požadavek serveru.
`setRequestHeader('hlavička','hodnota')`	Nastaví určenou hlavičku na zadanou hodnotu. Metoda `open()` musí být zavolána před jakýmikoli pokusy o nastavování hlaviček.

Tabulka 3.2: Standardní metody XMLHttpRequest (3)

Atributy

Kromě těchto standardních metod obsahuje objekt XMLHttpRequest atributy uvedené v tabulce 3.3: Standardní atributy XMLHttpRequest (3).

Atribut	Popis
`onreadyStatechange`	Ukazatel na obslužný kód, který je spuštěn při každé změně interního stavu objektu, typicky se jedná o funkci JavaScriptu
`readyState`	Stav požadavku. Pět možných hodnot je: 0 – neinicializováno, 1 – načítání, 2 – načteno, 3 – interaktivní, 4 – dokončeno
`responseText`	Odpověď serveru ve formě řetězce.
`responseXML`	Odpověď serveru ve formě XML.
`status`	Stavový kód získaný od serveru: 200 – operace proběhla úspěšně. 404 – prostředek nelze nalézt (not found), atd.
`statusText`	Textová verze stavového kódu (OK, Not Found, atd.)

Tabulka 3.3: Standardní atributy XMLHttpRequest (3)

3.4 PHP

PHP (rekurzivní zkratka PHP: Hypertext Preprocessor, „PHP: Hypertextový preprocesor“, původně Personal Home Page) je multiplatformní skriptovací programovací jazyk, zpracovávaný na straně serveru. Umožňuje vkládání kódu, který generuje dynamický obsah, přímo do HTML.

PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti. Syntaxe jazyka PHP je odvozena z koncepce jazyka C či Perl. PHP je nezávislý na platformě, skripty fungují bez úprav na mnoha různých operačních systémech. Obsahuje knihovny poskytující funkce pro zpracování textu (včetně podpory regulárních výrazů), grafiky, práci se soubory, či pro zajištění konektivity s databázovými servery (mj. MySQL, ODBC, Oracle, PostgreSQL, MSSQL).

Ke spouštění skriptů je třeba, aby na webovém serveru, který stránky generuje byla nainstalována podpora PHP (nejčastěji se používá Apache server). Dále je třeba vhodnou konfigurací, popř. doinstalováním některých modulů zajistit např. konektivitu s databází (i7).

3.4.1 Syntaxe

PHP rozlišuje velká a malá písmena v názvech proměnných. Zatímco v názvech funkcí velikost písmen nerozlišuje. Příkazy se ukončují středníkem. Proměnné se nemusejí předem deklarovat, od ostatního kódu se odlišují tím, že začínají znakem dolaru (`$variable`). PHP podporuje i objektově orientované programování.

3.4.2 Použití

Díky podpoře konektivity s databázovým serverem bylo možné PHP využít právě k účelu získávání dat z databáze. Navíc naprogramované PHP scripty dokáží z přijatých argumentů dynamicky vytvořit SQL dotazy.

3.5 Oracle databáze

3.5.1 Databázová platforma Oracle

Tato bakalářská práce využívá různých prostředků a nástrojů databázového serveru Oracle, a proto je třeba představit i Oracle platformu. Jak již bylo řečeno zahrnuje především databázi. Databázi, ve smyslu uložených dat, a sadu nástrojů, zajišťujících jejich ukládání a manipulaci s těmito daty. Mimo to obsahuje databázový server, což je soubor programových prostředků určených pro práci

s daty, včetně organizace a realizace přístupu k těmto datům. A dále aplikační server, popřípadě další serverové aplikace.

Data jsou ukládána do databázových tabulek. Struktura těchto data jejich uložení jsou nezávislé a od uživatele odstíněné. Přístup k nim a k operacím nad nimi je realizován pomocí SQL jazyka (1, 2).

3.5.2 Integritní omezení

Integritní omezení jsou sadou nástrojů zajišťujících konzistenci dat v databázi. Data jsou konzistentní, vyhovují-li integritním omezením. Jinými slovy, data vyhovující formátu a struktuře specifikované v relačním modelu dat jsou konzistentní. Zabraňují ukládání nesprávných hodnot. V případě, že by mělo dojít k porušení integritního omezení, je jasně stanoveno, co má databáze udělat (1, 2, i9).

K porušení konzistence může dojít několika způsoby, tomu napomáhají předcházet různé druhy integritních omezení:

Entitní

Jedná se o specifikaci primárního klíče tabulky. Primární klíč (`PRIMARY KEY`) je sloupec či skupina sloupců, která slouží pro jednoznačnou identifikaci každého řádku v tabulce. Hodnota pole (polí) primárního klíče musí být v rámci tabulky jedinečná. Pole primárního klíče nesmí obsahovat hodnoty `NULL`.

Doménové

Jde o definici různých omezení na určitý datový typ, popřípadě omezení rozsahu hodnot.

Například uživatelem definované omezení `CHECK`. Lze jím definovat interval přípustných hodnot, či kombinací s jinými funkcemi (pro práci s řetězci, datumy) vymezit přesný formát.

Referenční

Zajišťuje konzistenci dat v rámci více tabulek. Definuje vztah pomocí cizích klíčů (`FOREIGN KEY`). Cizí klíč definuje vztah k primárnímu klíči v jiné (případně té samé) tabulce.

NOT NULL a UNIQUE

Omezení NOT NULL zajistí, že nebude možné vložit do slupce, nad nímž je toto omezení definováno nulovou (prázdnou) hodnotu. UNIQUE zajistí jedinečnost hodnoty v rámci sloupce.

3.5.3 PL/SQL (principy a využití v BP)

PL/SQL je rozšířením jazyka SQL. Rozšiřuje ho o klasické programátorské nástroje. Podporuje řadu konstrukcí jako jsou cykly, podmínky, procedury a funkce. Obecně se zkratka PL vysvětluje jako Trasaction Processing Language (transakční procedurální jazyk). Dává uživateli prostor pro vytváření procedur, větvení toku programu na základě rozhodovacích podmínek a cyklů, deklaraci konstant, proměnných a kurzorů (1).

Využití

Oproti sekvenčnímu zpracovávání SQL, dává procedurální jazyk mnohem více možností. Lze programovat i složité algoritmy, využívající všech prostředků SQL jazyka. Procedury, či funkce se ukládají mezi ostatní objekty uživatelského schématu a lze je opětovně využívat i kombinovat (v těle jedné procedury, či funkce volat jinou).

Formou triggerů lze určit akci, která automaticky spustí provedení programového kódu definovaného v těle triggeru a tím zajistit integritu dat.

Rovněž lze definovat vlastní funkce a ty volat v rámci aplikace či aplikačního serveru. Oddělí se tím aplikační logika od správy dat, která zůstává v kompetenci databázové vrstvy. Má to mnoho výhod jak z hlediska oprávnění a přístupu k datům, tak v rychlosti odezvy, a v neposlední řadě možnost poskytnout stejné funkce pro různé aplikace pracující s databází. Navíc o spoustě funkcionalit zajišťovaných triggery nemusí aplikace vůbec vědět. Provádějí se automaticky (v rámci databáze).

Struktura

Programový jazyk PL/SQL je modulární, tedy také blokový. Základní entitou je programový blok. Typická struktura programového bloku se skládá ze třech sekcí.

Deklarační sekce

Sekce určená pro deklaraci proměnných, konstant, kurzorů apod., které budou využity v sekci výkonné.

Výkonná sekce

Sekce obsahující funkční logiku, tzn. jádro programového bloku. V bloku výkonné sekce je možné manipulovat s daty v databázi, popřípadě o těchto manipulacích vytvářet záznamy či informovat v konzoli.

Sekce pro zpracování vyjímek

Zpracovává a zachytává chyby (vyjímky), které nastaly během provádění výkonné sekce.

Příklad použití v bakalářské práci

Dobrým příkladem je použití triggeru při vkládání dat do databáze. Díky triggerům lze zajistit, že data budou v určitém okamžiku během vkládání dále analyzována a zpracována.

3.6 XML – formát

XML (eXtensible Markup Language, česky rozšiřitelný značkovací jazyk) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Slouží k výměně dat mezi různými aplikacemi na různých platformách a pro publikování dokumentů.

Díky tomu, že je otevřeným formátem, není svázán s žádnou konkrétní platformou, získal širokou podporu. K jeho editaci není třeba žádný speciální program, stačí obyčejný textový editor. Jako znakovou sadu používá implicitně ISO 10646, což je 32bitová znaková sada, která dokáže pojmout všechny znaky dnes

používaných jazyků. Lze ovšem nastavit jinou znakovou sadu vyhovující potřebám konkrétního jazyka (10).

Očekává se, že XML spolu s dalšími příbuznými technologiemi nahradí jazyk HTML (i10).

3.6.1 Syntaxe

Každý XML dokument se skládá z elementů, které jsou do sebe navzájem vnořené. Elementy se v textu vyznačují pomocí tzv. tagů. Většině elementů odpovídají dva tagy (počáteční a ukončovací)

Názvy tagů se zapisují mezi znaky „<“ a „>“. Ukončovací tag má před svým názvem ještě zpětné lomítko („/“), aby se odlišil od počátečního. Jména elementů rozlišují malá a velká písmena. Důležité je, aby dokument obsahoval jeden kořenový (root) element, do kterého jsou ostatní tagy vnořené. Element může obsahovat jeden i více atributů, které jsou součástí otevíracího tagu. Hodnoty atributů se uzavírají do uvozovek (10).

3.6.2 Příklad

Následující vzorek XML dokumentu je ukázkou odpovědi serveru, který vyhledá v databázové tabulce evidující města v ČR názvy měst začínající písmeny „Ko“.

```
<LIST_OF_CITIES>
 <CITY>KOKOŘOV</CITY>
 <CITY>KOLÍN II</CITY>
 <CITY>KOLÍN III</CITY>
 <CITY>KOLÍN IV</CITY>
 <CITY>KOLÍN V</CITY>
 <CITY>KOLÍN VI</CITY>
 <CITY>KOLNÁ</CITY>
 <CITY>KOLONIE 5.KVĚTNA</CITY>
 <CITY>KONČINY 1.DÍL</CITY>
 <CITY>KONČINY 2.DÍL</CITY>
 <CITY>Kostelní Myslová</CITY>
 <CITY>Koterov</CITY>
</LIST_OF_CITIES>
```

3.6.3 Využití

Jak již bylo řečeno, XML formát je ideální pro výměnu dat mezi různými aplikacemi i platformami. Skvěle se hodí pro výměnu dat mezi databází a webovým serverem. Většina programovacích jazyků zahrnuje podporu a práci s XML, JavaScript nevyjímaje. XML je v této práci použit jako hlavní formát pro přenos jakýchkoliv informací z databáze. Velká část dat, získaných pro potřeby našeptávačů byla získána právě v tomto formátu. A díky dobré podpoře XML v PHP jazyce bylo možné napsat jednoduchý script a pomocí něj všechna data z XML dokumentu automaticky zpracovat a naplnit jimi databázi.

4 Pořizování dat

Kvalita výsledného systému závisí ve velké míře na kvalitě dat. Úkolem je ovšem nejen získání těchto dat, ale rovněž vyčtení a vhodné použití maximálního množství informací.

Vynikající zdroj datových souborů je k nalezení na stránkách Ministerstva vnitra České republiky, dále pak na stránkách České pošty, ovšem v různých formátech (xml, xls). Jak tedy těmito daty naplnit databázi?

4.1 Import XLS formátu

Zpracování XLS souboru je poměrně jednoduché. Program SQL developer podporuje import dat z XLS souborů. Proto stačí vytvořit tabulku žádané struktury, vybrat jednotlivé sloupce pro import a potvrdit.

V případě textových polí je nutné experimentovat, neboť při importu může dojít k tomu, že pokud délka řetězce v poli souboru přesahuje nastavenou délku řetězce ve struktuře tabulky, Oracle příslušný řádek nevloží a upozorní na chybu. Proto je třeba zvětšit délku textového pole ve struktuře tabulky nebo pomocí funkcí Excelu zjistit délku nejdelšího řetězce v příslušném sloupci. Zbytečně dlouhé pole je plýtváním pamětí i výkonu, nicméně Oracle neumožňuje zmenšení, pouze zvětšení délky ve struktuře tabulky. Řešením je vytvoření nové tabulky s již předem známou strukturou a její naplnění selectem z původní.

4.2 Import XML formátu

Pro import XML dat do databáze či jiného formátu sice existují různé nástroje, ovšem pro specifické potřeby bylo třeba naprogramovat vlastní script.

4.2.1 XML parser

XML parser, napsaný v PHP vychází z několika frameworků dostupných na stránkách PHP_CLASSES. Úpravou algoritmu procházejícího strukturu dokumentu bylo možné získat jakákoliv data z libovolného XML souboru ve formě SQL.

Zde je uveden příklad takového dokumentu. Eviduje všechny ulice v ČR, přičemž správným algoritmem lze získat i vazbu k městu, ke kterému ulice náleží.

```
<?xml version="1.0"?>
<adresy stav_k="25.únoru 2008">
 <oblast nazev="ČESKÝ TĚŠÍN" okres="KARVINÁ">
  <obec nazev="SLAVONICE" kod="15476">
   <cast nazev="KADOLEC" kod="23950"/>
   <cast nazev="SLAVONICE" kod="15476">
    <ulice nazev="BEDŘICHA SMETANY" kod="2636"/>
    <ulice nazev="BOŽENY NĚMCOVÉ" kod="2935"/>
   </cast>
  </obec>
  <obec nazev="ŽUPANOVICE" kod="1452"/>
 </oblast>
</adresy>
```

4.2.2 Import

Odpověď serveru v následujícím formátu je možné jednoduše vložit do databáze.

```
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','BEDŘICHA SMETANY','2636');
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','BOŽENY NĚMCOVÉ','2935')
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','JANA ŽIŽKY','7882');
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','JIRÁSKOVA','8414');
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','KRÁTKÁ','11044');
  INSERT INTO ULICE (OBEC_KOD,ULICE_NAZEV,ULICE_KOD)
VALUES ('17016','KVASICKÁ','11491');
```

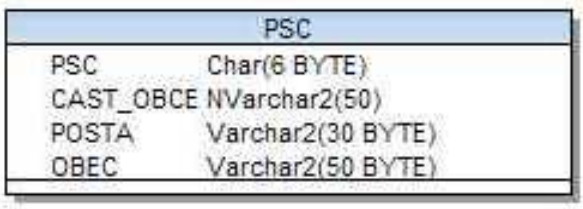
Nejprve je třeba vytvořit tabulku požadované struktury a poté vložit všechny inserty do editačního pole v SQL developeru nebo jiné konzole a naplnit jimy tuto tabulku.

4.3 Propojení dat

Z uvedených pramenů bylo možné získat hodnotná data. Bohužel, získání klíče, který by dokázal logicky spojit data z těchto zdrojů nebylo tak triviální.

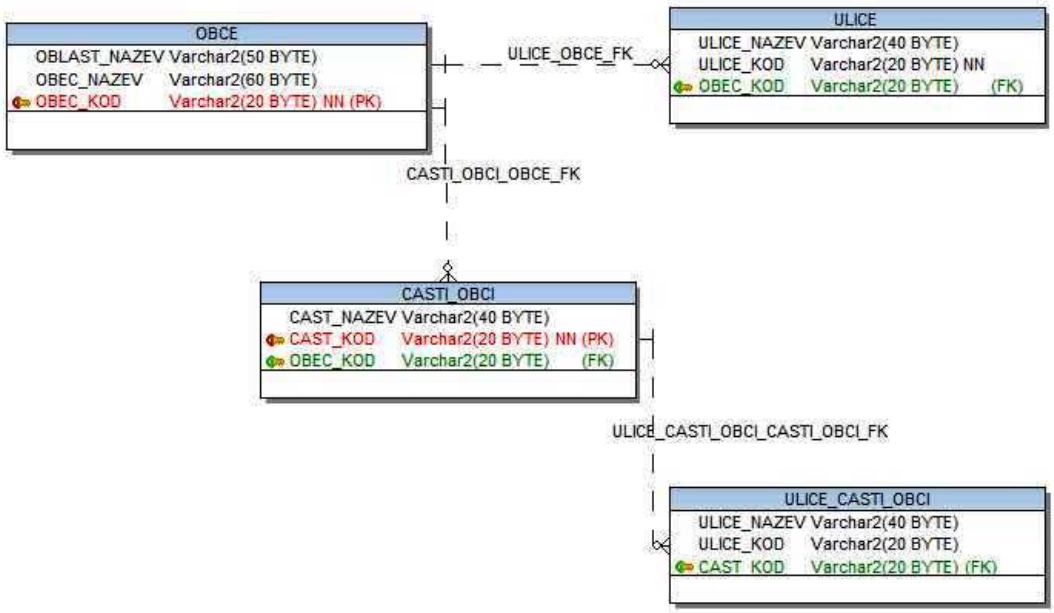
Cílem snažení bylo provázat tabulky tak, aby bylo možné při dotazu na město získat informaci o PSČ a seznam ulic.

Výchozí tabulky vypadaly takto:



Obrázek 1: Struktura tabulky PSC

Tabulka PSC na Obrázku 1, která eviduje obce, jejich části, názvy a směrovací čísla pošt.



Obrázek 2: Struktura tabulek evidujících obce a ulice v ČR

Tabulky, které obsahují jedinečný identifikátor každé obce, příslušnou oblast a seznam ulic znázorňuje obrázek 2.

Otázkou tedy bylo, jakým způsobem provázat tyto dva různé zdroje dat. Za výchozí bod lze, pochopitelně, v tomto případě, považovat sloupce s názvy obcí. Nabízí se tedy jednoduché řešení: Spojit tabulku `OBCE` a `CASTI_OBCI` do jedné a porovnáváním názvů obcí z takto vzniklého spojení s obcemi v tabulce `PSC` získat jednoduše vazbu na data v této tabulce.

Tato teze se ovšem ukázala jako mylná. Nejenže nezanedbatelný počet měst vůbec nebyl nalezen, ale hlavní problém nastal v případě, kdy se název města (i několikrát) opakoval. Takovýto způsob řešení nemohl tudíž poskytnout uspokojivý výsledek.

Bylo třeba zvolit jinou taktiku a zjistit příčiny neshod v názvech (ačkoliv jde o podobná data jen z různých zdrojů). Dále rovněž zajistit přidělení správné vazby v případě měst se stejným názvem.

4.3.1 Modifikace klíčových hesel

Jednoduchý select ukázal, která hesla z množiny, jež by měla zajistit funci klíče pro spojení dvou tabulek, nevyhovují. Jednalo se pouze o rozdíly v interpretaci. A sice hesel představujících městské části Prahy, Brna a Plzně.

```
  select upper(city_loc),zip,region from psc
   where upper(city_loc) not in (select city_name from
cities)
```

Tabulka 4.1 uvádí příklad rozdílné interpretace v tabulkách PSC a OBCE:

PSC	OBCE
Praha 1-Nové Město	NOVÉ MĚSTO
Praha 1-Josefov	JOSEFOV
Bolevec (část)	BOLEVEC
Dolní Vlkýš (část)	DOLNÍ VLKÝŠ
Černá Pole (Brno-sever)	ČERNÁ POLE
Královo Pole (část)	KRÁLOVO POLE

Tabulka 4.1.: Ukázka rozdílů klíčových hesel

Pro úpravu, či lépe sjednocení, formátu řetězců v obou tabulkách k zajištění funkce klíče se nabízelo více možností. Nejjednodušší bylo použití regulárních

výrazů, které mají v Oracle výbornou podporu, a které se i zde jeví jako silný nástroj, zejména s dalšími funkcemi pro práci s řetězci.

Ukázka funkce, která vyhledá a „ořeže“ závorky a jejich obsah, čímž odstraní rozdíly v městských částech Plzně a Brna.

```
  RPAD(condition,REGEXP_INSTR(condition,'\(.*\)$')-1,'
');
```

Další funkce vyhledá řetězce představující městské části města Prahy.

```
  SUBSTR(condition,REGEXP_INSTR(condition,'-.*')
+1,LENGTH(condition));
```

4.3.2 Vytvoření dalšího klíče

Modifikací formátu hesel se sice redukoval počet nenalezených hesel při konstrukci klíčů na nulu, ovšem stále zůstal problém v duplicitách. Totiž název města nelze pokládat ani v jedné z tabulek za primární klíč, neboť skupin měst se stejným jménem je nezanedbatelný počet.

Možným řešením je přidání dalšího klíče, který by v kombinaci s názvem města zajistil jedinečnost. Takovým klíčem může být název okresu.

Okresy sice jako NUTS (statistické územní jednotky Evropské unie) svůj statut v roce 2002 pozbyly, nicméně v použitých zdrojových datech jsou zahrnuty. Pro tvorbu primárního klíče jsou v tomto případě důležité tyto vlastnosti: „Název okresu je jedinečný v rámci celé republiky.“ (i11) A dále: „Název obce by měl být jedinečný v rámci okresu.“ Tím je jasné, že kombinace sloupců NAZEV_OKRESU a NAZEV_OBCE zajistí jedinečnost v rámci tabulky.

Existují vyjímky, které toto pravidlo porušují. Ty lze vyhledat následujícím selectem:

```
   SELECT * FROM (SELECT OBEC, OKRES, COUNT(*) AS POCET
FROM PSC GROUP BY OBEC, OKRES) WHERE POCET > 1
```

Výsledek:

```
OBEC         OKRES            POCET
---------    --------------   ---------
Březina      Brno-venkov      3
Věžnice      Jihlava          4
Mezholezy    Domažlice        2
```

Pouze v těchto případech bylo nutné bližší prozkoumání situace a ruční úprava.

4.3.3 Konečná syntéza

Po konečném propojení klíčů (názvů měst) bylo možné propojit i části měst. Tím byla získána přesná struktura obsahující kód města, název města, dále příslušný okres, PSČ a název pošty.

4.3.4 Připojení seznamu ulic

Celý tento složitý proces byl nutný k tomu, aby bylo možné vytvořit jakousi linii spojující PSČ v tabulce `PSC` a seznam ulic v evidenci měst, obcí a jejich ulic vedoucí přes název města a název pošty.

Pro zjednodušení další práce bylo přiděleno každému jedinečnému názvu ulice jedinečné `ID`. A toto `ID` je dále spojeno s kódem města. Tím se značně zredukoval seznam ulic s identifikátorem o ty, které měly stejný název, ale různý kód.

4.4 Výsledná data

Kromě již zmíněných dat, byla získána data k vytvoření tabulek evidujících křestní jména a příjmení mužů i žen přihlášených k trvalému pobytu v ČR, dále seznam používaných titulů. ER-diagram databáze zdrojových dat je v příloze A. Znázorňuje jednotlivé tabulky a vazby mezi nimy, dále pohledy, které byly vytvořeny pro potřeby našptávačů.

5 Výsledné řešení a implementace

V této fázi lze vysvětlit uplatnění nejen popisovaných technologií, ale i procesu sběru dat. Kombinací všech zmíněných nástrojů bylo možné dosáhnout požadovaného výsledku.

Uživatel přistupující k rozhraní, které zprostředkovává spojení s databází není detailně informován o struktuře ani chování databázového systému, jak již bylo předesláno v úvodu. Je tedy nutné vytvořit takové rozhraní, které jemu poskytne dostatečné množství nástrojů, které zkvalitní a zefektivní jeho práci nejen v tomto rozhraní, ale i v komunikaci s databází.

Existuje řada zaběhnutých a odzkoušených technologií, zdokumentovaných nejen formou komentářů ve zdrojovém kódu, ale i v literatuře, které je možné použít. Hlavní výhodou je efektivní výsledek v podobě přehledného a uživateli dobře známého prostředí.

5.1 Suggest box

Suggest box, neboli česky „našeptávač“ má pravděpodobně své kořeny v Google Suggest. Tato služba měla uživateli usnadnit vyhledávání hesel právě formou našeptávání. Postupně si ji osvojili další vyhledávače a dnes bývá dobrým standardem většiny portálů a nejen tam.

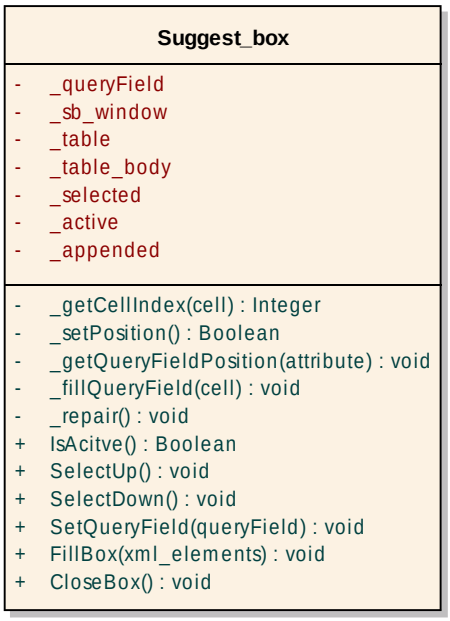
Nasazení této technologie má několik výhod. Především uživateli dává na výběr ze seznamu relevantních hesel. Pokud si uživatel osvojí tuto nápovědu, nemusí dopisovat dlouhá hesla, ba dokonce nemusí dodržovat konvenci ve psaní velkých a malých písmen. Sníží se riziko plynoucí z výskytu chyb v diakritice či pravopisu.

Suggest box používá technologii ajax popsanou v kapitole 3.2. Bez ní by nebylo možné podobného výsledku v prostředí webu dosáhnout. Rychlost odezvy by se v případě nutnosti obnovování stánky při každém stisknutí klávesy prodloužila na tolik, že by použití našeptávače práci spíše ztížilo, než zefektivnilo.

Další podmínkou je dobrý zdroj dat, který poskytne ona relevantní hesla. Proto bylo důležité se věnovat i této fázi práce. Bez příslušných dat by našeptávač nemohl, jak z kontextu jistě vyplývá, plnit svou funkci.

5.1.1 Popis samotného modulu

Jak již bylo zmíněno, použití našeptávače vyžaduje na úrovni implementace rozhraní i technologii Ajax.



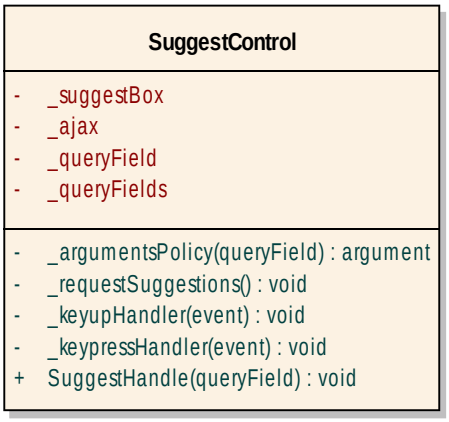
Obrázek 3: Suggest_box

Třída Suggest_box znázorněná na Obrázku 3 je napsaná v JavaScriptu. Poskytuje veškeré metody, které zajišťují nejen dynamické vytvoření elementu představujícího rámec, v němž jsou data našeptávače seřazena, ale dále i metody zajišťující např. posun v našeptávači nahoru a dolu, či reakce na pohyb a potvrzení pomocí myši. Našeptávač rovněž zjistí pozici, kde se má vytvořit a rozměr a to z předem zadaného pole, kterému našeptává.

Parametry předávané našeptávači jsou pouze dva. Prvním je vstupní pole, ze kterého našeptávač zjistí již zmíněné parametry a pozicuje zobrazovaný rámec těsně pod něj a dále jsou to data ve formátu xml elementů, kterými naplní našeptávač.

Správu vstupních polí na stránce, zpracovávání, odesílání a přijímání dat pro účely našeptávače bylo nutné oddělit. Pro případné další použití v různých prostředích je možné jednotlivé moduly měnit, ale přitom využít všech nabízených vlastností a neporušit jejich funkčnost.

Dalším navrženým objektem je `SuggestControl` (Obrázek 4), který zmíněnou správu zajistí. Obsahuje i metodu `_argumentsPolicy`, která vytváří z hodnot v poli argumenty, které jsou předávány serveru.

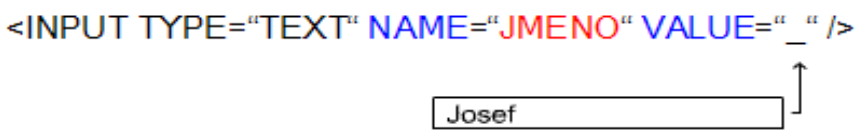


Obrázek 4: SuggestControl

Politika tvorby argumentů předávaných serveru

Server dokáže zpracovat argumenty přijaté z webového rozhraní podobně jako jsou zpracovávány funkcemi v rámci programu. Pro takovou komunikaci je ovšem nutné zvolit vhodnou politiku pro tvorbu, předávání a zpracovávání argumentů. Je důležité klást důraz na disciplinovanost komunikace, neboť zde není kompilátor, který by na případnou chybu upozornil.

Vhodným řešením při volbě formátu a uspořádání argumentů může být použití již zaběhlého sytému, a sice systému odesílání formulářových dat na server. Tento systém využívá dalších vlastností formulářových elementů.



Obrázek 5: Formulářový element

Na obrázku 5 : (Formulářový element) jsou vlastnosti formulářového pole užívané při konstrukci argumentů odesílaných na server. Výsledný argument nese

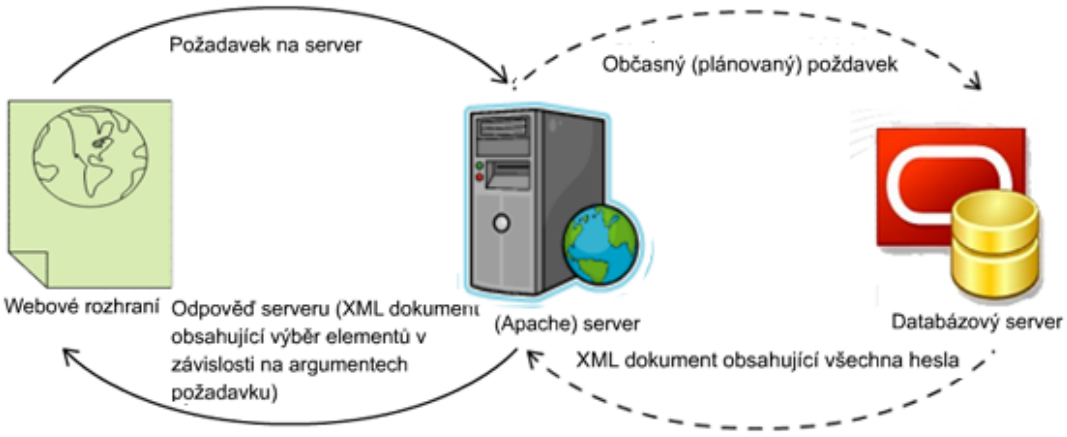
název a hodnotu. Pro název se používá hodnota atributu `NAME` a hodnotou je samotná hodnota formulářového pole (`VALUE`), tedy to, co uživatel zadal.

V případě volby způsobu předávání argumentů pro potřeby našeptávače se popsaný systém jevil jako možné řešení, které nakonec bylo použito. To se dále ukázalo jako výhodné, neboť lze jednoduše přidat další pole argumentů.

Serverová část

Server obdrží požadavek, na který je třeba odpovědět. Hodnoty přijaté popsanou formou argumentů je třeba dále formalizovat ve tvaru SQL dotazu, který je odeslán databázi. Odpovědí je pouhá interpretace odpovědi databáze na tento dotaz.

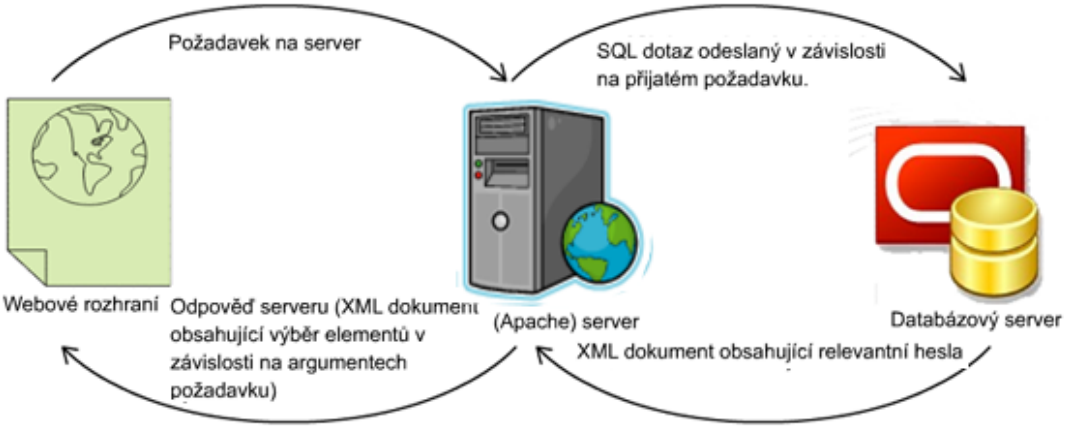
Pro formulaci takového dotazu je již třeba znát strukturu databáze i strukturu jednotlivých objektů. Je třeba znát, ve které tabulce se požadovaná data nachází i název sloupce. Samozřejmým předpokladem pro získání dat je přidělené oprávnění pro přístup k datům.



Obrázek 6: Tok dat (off-line databáze)

Byl volen takový systém, který umožní změnu serverového scriptu, přičemž webové rozhraní zůstane neměnné. To má mnoho výhod. Na argumenty, že při získávání dat pro našeptávač je příliš zatěžován databázový server, lze odpovědět vytvořením serverového scriptu, který stáhne data z databáze pouze jednou či v určených časových intervalech a databázi již dále nezatěžuje. Taková situace je ilustrována na obrázku 6 : (Tok dat (off-line databáze)).

Pro dokonalou názornost a představení všech funkcí však byla volena původní teze, a sice že data jsou získávána v reálném čase přímo z databáze. Jinými slovy, server formuluje přijaté argumenty do SQL dotazu, ihned odešle dotaz databázi a přijatou odpověď v nezměněné podobě přepošle dál.



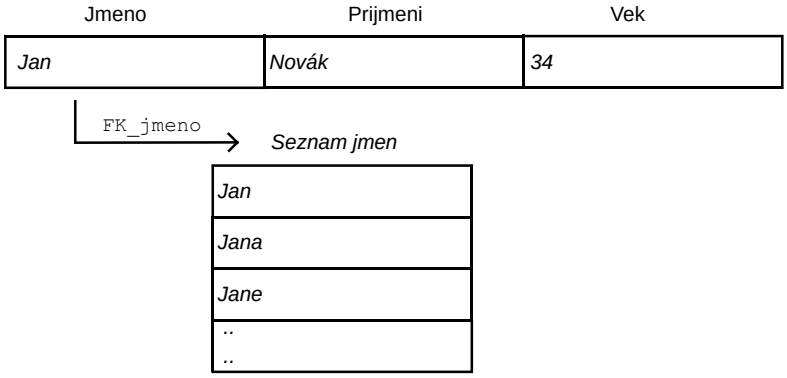
Obrázek 7: Tok dat (on-line databáze)

Hlavní výhodu v tomto případě představuje použití XML formátu. Pokud databázový server odešle data ve formě XML dokumentu, není třeba dále tento dokument jakkoliv upravovat a lze jej jednoduše přeposlat SuggestBoxu, který se naplní jeho obsahem. Navíc pro výběr hesel postačí využití podmínek v SQL dotazu, tudíž není třeba dopisovat jakékoliv vyhledávací či třídící funkce.

Databázová vrstva

Databáze by měla zajišťovat poskytování aktuálních dat pro potřeby našeptávače. Zdroje těchto dat i jejich typy (tabulka, pohled) se různí (s ohledem na potřeby formuláře), ovšem výsledek dotazu by měl mít vždy stejnou formu.

Možným příkladem zdrojových dat může být tabulka obsahující cizí klíče k vyplňovanému poli, jak je znázorněno na obrázku 8.

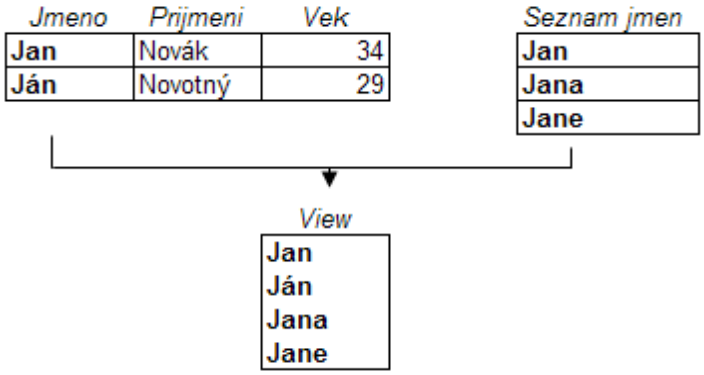


Obrázek 8: Ukázka reference na zdrojová data

Užití cizích klíčů patří mezi integritní omezení. Není možné vložit do příslušného sloupce tabulky hodnotu, která by nebyla v seznamu hesel, představujících cizí klíč, je-li nad tímto sloupcem takové omezení definováno. Proto se použití našeptávače hodí především v tomto případě. Uživateli jsou totiž nabízena hesla, která je možné vložit do tabulky. Pokud není, v závislosti na hodnotě vstupního pole, nabídnuto žádné heslo, znamená to, že hodnota vstupního pole není obsažena mezi hodnotami cizího klíče, a tudíž nelze takovou hodnotu do příslušného sloupce vložit.

Jiným zdrojem dat může být sloupec tabulky, do níž je právě vkládán řádek, či tvořena podmínka pro výběr dat.

V případě výběru zdroje dat v této bakalářské práci byla volena jakási kombinace obou těchto nabízených možností. Cizí klíč nebyl sice definován, přestože existuje vhodný kandidát. Bylo uvažováno, že do vstupního pole bude možné vkládat i jiné (nové) hodnoty, které v potenciálním sloupci cizích klíčů nejsou. Sjednocení těchto množin je prezentováno formou pohledu, čímž je zajištěna aktuálnost.



Obrázek 9: Pohled vzniklý sjednocením

Jako forma odpovědi databáze serveru byl volen XML dokument. Jeho struktura je lehce čitelná nejen ve všech vrstvách aplikace, ale i lidským okem (výhodné při kontrole, či dalším zpracovávání). Následující ukázka představuje obecnou strukturu dokumentu:

```
<LIST_OF_Název tabulky (pohledu)>
<Název sloupce>Vybraný řádek 1</Název sloupce>
<Název sloupce>Vybraný řádek 2</Název sloupce>
..
<Název sloupce>Vybraný řádek n</Název sloupce>
</LIST_OF_Název tabulky (pohledu)>
```

5.2 Aktivní kontrola formátu vymezeného regulárním výrazem

Na některá pole nelze (nebo není vhodné) aplikovat našeptávač (např. rodné číslo). I na tato pole lze (a je vhodné) aplikovat integritní omezení. Toto omezení musí být splněno, jinak data nemohou být vložena. Cílem aktivní kontroly je upozornit uživatele na případné porušení tohoto omezení ihned (ještě před odesláním všech dat). A sice se jedná o omezení typu check, které využívá regulární výraz definování, či lépe vymezování, formátu zadávaných hesel.

Regulárním výrazem lze na úrovní databáze vymezit podmínku formátu vkládaných dat, která musí být splněna. Výhodou regulárních výrazů je fakt, že regulární výraz, definovaný na úrovni databáze, bude vymezovat stejný formát jako na úrovni JavaScriptu. Jinými slovy, je-li provedena kontrola řetězce funkcí databáze, které předáme regulární výraz a řetězec, tato funkce vrátí stejný výsledek, jako odpovídající funkce v JavaScriptu se stejnými argumenty.

Otázkou je, jak regulární výraz v databázi přenést na úroveň JavaScriptu. Při tvorbě databáze lze vytvořit podrobnou dokumentaci zahrnující i tento regulární výraz nad příslušným sloupcem. Programátor si při tvorbě JavaScriptového kódu přečte dokumentaci a regulární výraz prostě opíše. Tím ovšem dochází zdvojování. Nejenže programátor musí hledat zda je, či není příslušný sloupec omezen tímto způsobem, ale regulární výraz v databázi lze kdykoliv změnit a poté by bylo nutné změnit jej i na úrovni JavaScriptu. Nabízí se řešení, které zajistí automatické stáhnutí

požadovaného výrazu (vždy aktuálního) z databáze a jeho následné použití v JavaScriptové funkci.

5.2.1 Načasování a způsob upozornění

Kontrola může být spuštěna jako reakce na jakoukoliv událost v rámci formuláře. Pod pojmem událost je zde myšlen pohyb myši, či stisnutí libovolného tlačítka nebo klávesy na klávesnici v aktivním formulářovém okně.

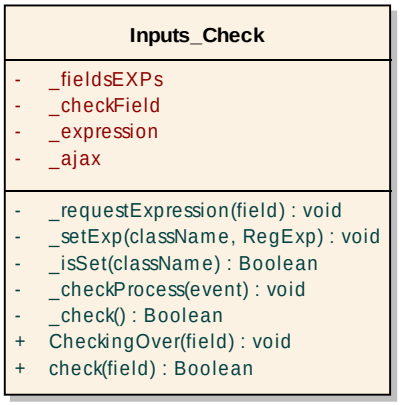
Na výsledek kontroly je třeba upozornit vhodným způsobem. Tzn. takovým způsobem, který neobtěžuje (nevyžaduje další potvrzování jako třeba vyskakovací hlášky), ale pouze informuje.

Nejlepší kandidátskou událostí pro spuštění kontroly formátu pole je každé stisknutí klávesy v aktuálním poli. Jinými slovy, kontrola probíhá neustále, při vyplňování pole.

Upozorňování je realizováno změnou stylu, resp. barvy pozadí. Pokud kontrola proběhne v pořádku, pozadí změní barvu na zelenou. Pokud však nastane chyba, pozadí se obarví červenou. Tak uživatel jasně vidí, zda je možné takové pole vložit do databáze, či nikoliv.

5.2.2 Popis JavaScriptového objektu

`Inputs_Check` (obrázek 10) napsaný pro tyto účely v JavaScriptu poskytuje metody zajišťující součinnost zmíněných operací. Rovněž využívá ajax pro komunikaci se serverem, která je nutná k získání dat z databáze.



Obrázek 10: Inputs_Check

Při aktivaci formulářového pole se odešle požadavek na server a je očekáván regulární výraz náležející danému poli. Pro politiku odesílaných argumentů byla využita vlastnost elementu `className`. Určené `className` přímo v těle stránky je využito na serveru k vyhledání odpovídajícího názvu omezení v databázi. Server odešle požadavek databázi a po obdržení odpovědi vrátí XML dokument obsahující regulární výraz. Ten spolu s ukazatelem na formulářové pole uloží do svého seznamu a v případě pozdější aktivace stejného pole již neodesílá žádost na server, nýbrž odpovídající regulární výraz vyhledá ve svém seznamu.

Metoda `_checkProcess` zajistí provedení kontroly a upozornění (obarvení pozadí) v závislosti na jejím výsledku. Dále tento objekt poskytuje veřejnou metodu `check(field)`, kterou lze využít např. při konečné kontrole formuláře (před odesláním) a to pro každé pole, které je nutné takto kontrolovat.

5.2.3 Realizace kontroly v databázi

Integritní omezení typu check definuje podmínku, která musí být pro ukládané hodnoty splněna (1). Omezení se v tomto případě vztahuje na určitý sloupec. Lze takto vymezit např. rozsah platnosti hodnot, formát atp.

Omezení pro daný sloupec se definuje následovně:

```
  CONSTRAINT "Název omezení" CHECK
(REGEXP_LIKE(Sloupec,'Regulární výraz')
```

Pro získání informací o omezení je třeba vědět, že všechna potřebná data se nacházejí v systémové tabulce `user_constraints`.

Informace o její struktuře lze zjistit následujícím způsobem:

```
desc user_constraints

Name                           Null     Type
------------------------------ -------- ------------
OWNER                          NOT NULL VARCHAR2(30)
CONSTRAINT_NAME                NOT NULL VARCHAR2(30)
CONSTRAINT_TYPE                         VARCHAR2(1)
TABLE_NAME                     NOT NULL VARCHAR2(30)
SEARCH_CONDITION                        LONG()
R_OWNER                                 VARCHAR2(30)
R_CONSTRAINT_NAME                       VARCHAR2(30)
..
..
```

Z výsledné struktury tabulky `user_constraints` lze tedy (mimo jiné) zjistit název omezení (*CONSTRAINT_NAME*), typ (*CONSTRAINT_TYPE*), název tabulky (*TABLE_NAME)* a hlavně `search_condition`, což je argument omezení check (v případě jiného typu omezení hodnota nulová).

Argumentem omezení není pouze regulární výraz, nýbrž řetězec představující volání funkce kontroly včetně argumentů. Problémem tedy zůstává, jak z tohoto řetězce získat samotný regulární výraz. Situaci navíc ztěžuje fakt, že použitý datový typ je `LONG()`, jak je vidět ve výpisu. Pro tento typ neexistuje taková podpora, jako je to v případě obyčejných řetězců (`VARCHAR2`), a proto na něj nelze aplikovat funkce pro práci s řetězci.

I tento problém lze vyřešit na úrovni databáze, i když se způsob může zdát trochu neobratný, či neobvyklý, výsledek poskytuje přesná data.

Nejprve nutné vytvořit funkci, která hodnotu typu `LONG` přetypuje na `VARCHAR2`. Standardní nabídka oracle funkcí pro typovou konverzi takovou metodu nezahrnuje. Princip této funkce je takový, že v deklarační části je sice proměnná použitá pro výraz deklarována jako `LONG`, nicméně jako návratový typ funkce je určen `VARCHAR2`. Tím je získán řetězec požadovaného typu, nikoliv však v požadovaném tvaru.

Tvar řetězce představujícího omezení kontrolující tvar e-mailu vypadá následně:

```
  REGEXP_LIKE(EMAIL,'^[a-zA-Z0-9._-]+@+[a-zA-Z0-9.-]+\.+
[a-zA-Z]{2,4}$')
```

Pro další potřeby je třeba odfiltrovat pouze obsah uvozovek představující regulární výraz. Na to již postačí obyčejná funkce SUBSTR, která vrátí část řetězce určenou polohou první a poslední uvozovky.

Tím lze získat regulární výraz:

```
^[a-zA-Z0-9._-]+@+[a-zA-Z0-9.-]+\.+[a-zA-Z]{2,4}$
```

Získaný výraz je dále formou XML dokumentu odeslán. Tento postup umožní vytvořit jasně čitelnou odpověď, do které lze kdykoliv vložit další atributy.

Příklad odpovědi na dotaz (EMAIL), hledající regulární výraz pro kontrolu e-mailu:

```
  <CONSTRAINT type="REGEXP" like="EMAIL">
  <NAME>CLIENT_EMAIL_FORMAT</NAME>
  <VALUE>^[a-zA-Z0-9._-]+@+[a-zA-Z0-9.-]+\.+[a-zA-Z]
{2,4}$</VALUE>
  </CONSTRAINT>
```

5.3 Aktivní kontrola pomocí uživatelem definovaných funkcí

Některá pole formuláře je třeba kontrolovat i z dalších hledisek nesouvisejících s omezeními definovanými v databázi. Takové kontroly už nemusejí být nutně omezující (do databáze je lze vkládat), ale například pouze doporučující (informují o možném pochybení). Už z tohoto výkladu je patrné, že i otázkou úrovně omezení je třeba se zabývat a ponechat možnost parametrizováním určit, zda kontrola data případně zakáže vložit, či jen upozorní na možnou chybu.

Různé kontrolní funkce lze stáhnout z Internetu, či jen přepsat již hotové v jiném jazyce do JavaScriptu. Proto byl volen systém kontrol tak, aby výsledný framework mohl libovolnou funkci použít (např. načíst z externího souboru) a aplikovat na ní všechny metody, které poskytuje.

5.3.1 Načasování a způsob upozorňování

V tomto případě není nutné uživatele informovat o chybě při každém stisku klávesy, někdy to může být náročné z hlediska složitosti výpočtu, či je nesmyslné

kontrolovat neúplná slova. Přesto je tato možnost ponechána. Lze tedy kdykoliv (pouhým parametrizováním) určit, kdy má být kontrola vykonána. Preferován je okamžik potvrzení aktivního pole formuláře (při přechodu na další pole). V určeném okamžiku je spuštěna kontrola.

Na rozdíl od kontroly uvedené v kapitole 5.2, zde může nastat více možností, než jen správně (TRUE) či špatně (FALSE). V případě složitějších kontrol je třeba konkretizovat chybu ke které došlo např. formou chybového hlášení. Opět by bylo asi nevhodné obtěžovat uživatele vyskakujícími okny, kde sice lze toto hlášení vypsat, ale které je třeba potvrdit a teprve potom pokračovat v práci.

Byl tedy hledán jakýsi kompromis. Uživatel je prvotně informován o výsledku kontroly ikonou vedle formulářového pole. Ikona mění obrázek podle toho, která úroveň omezení byla porušena. Obrázek 11 ukazuje význam jednotlivých symbolů.

	Kontrola proběhla v pořádku
	Varuje před možnou chybou a doporučuje její odstranění
	Vyskytla se chyba zabraňující vstupu dat do databáze

Obrázek 11: Přehled symbolů chybových hlášení

Zbývá vyřešit způsob, jak prezentovat chybové hlášení. Vzhledem k tomu, že toto hlášení má přímý vztah k příslušnému symbolu, je možné použít jej jako výchozí bod. Po přejetí myši nad symbolem se vedle objeví rámec s chybovým hlášením. Po odjetí myši rámec zmizí.

5.4 Funkce Enter klávesy pro potvrzení

Formulářová okna v běžných aplikacích nabízejí jako standard přepínání mezi jednotlivým okny v pořadí, jaké určí programátor, většinou v pořadí vyplývajícího z kontextu.

Ve standardním webovém formuláři bývá pořadí polí určeno pořadím polí v kódu stránky nebo jej lze definovat pomocí tzv. „tabindexu“. Přechod mezi formulářovými poli je realizován klávesou TAB. Klávesa ENTER slouží k potvrzení celého formuláře.

Cílem funkce je přiblížit přepínání kontextu klasické aplikaci. Tzn. klávesa ENTER i TAB plní funkci přepínače. K tomuto účelu poslouží `focus_manager`, který poskytuje metodu `fnext()`. Tato metoda zajistí přechod na další (hodnota tabindexu je je o jednu vyšší než aktuální hodnota tabindexu) pole formuláře při stisku klávesy Enter nad aktuálním polem.

5.5 Funkce „Měli jste na mysli“

Funkce „Měli jste na mysli: “ je nejspíše posledním můstkem, který opraví uživatele a nabídne mu alternativu v případě chyby. Je opět standardem většiny portálů. Omylný uživatel, který ignoroval našeptávač, či jej vůbec nepoužívá může napsat slovo s překlepem. Takové heslo může projít kontrolou formátu a lze jej vložit do databáze. Pokud je možné dané situaci předejít, je nasazení takového nástroje více než potřebné. Opět se redukují chyby v datech vstupujících do databáze.

Princip je jednoduchý: pokud se uživatel při psaní ve slově dopustí chyby ať gramatické, chybě v diakritice, či jen pouhého překlepu, slovo není nalezeno ve slovníku hesel, je potom vyhledáno heslo podobné. Definice a kritéria pro určení měřítka podobnosti záleží na konkrétním algoritmu a jeho kvalitu posoudí především uživatel, kterému je v případě chyby nabídnuto heslo, které měl skutečně na mysli.

5.5.1 Vytvoření algoritmu

Důležitým předpokladem je přístup k slovníku relevantních hesel, tzn. slovníku, jež nabízí alternativy. V tomto případě je slovník shodný se slovníkem, který využívá našeptávač.

Dalším předpokladem je analýza vedoucí k vytvoření algoritmu určujícímu měřítko podobnosti.

Teoretický základ zkoumající podobnost slov

Algoritmus není proces myšlení (využívající zkušenosti a emoce), nýbrž posloupnost naprogramovaných instrukcí, které je možné na základě podmínek větvit. Algoritmus neví, zda je slovo podobné jinému, či nikoliv, pouze aplikuje jakousi matematickou analýzu, která dokáže vzorcem popsat výpočet hodnoty

podobnosti (9). Do vzorce se dosadí obě slova a výslednou cifru lze považovat za měřítko pro porovnávání. Postupným dosazováním všech slov ze slovníku za druhou neznámou ve vzorci lze nashromáždit měřítka a vyhledáním nejvyššího určit „nejpodobnější“, tedy relevantní, heslo.

Vzorec obsahuje řadu kritérií. Jejich vhodnou kombinací lze dosáhnout uspokojivého výsledku.

Prvním kritériem může být délka slova. Slovo, které se skládá z pěti písmen vzniklo těžko ze slova, které má písmen deset. Tudíž čím větší rozdíl v délce slova, tím menší podobnost.

Dalším kritériem je hlásková shoda. Čím více shodných hlásek se vyskytuje v hledaném heslu, tím větší podobnost. To samozřejmě nebere ohled na pořadí hlásek. Například dvojice slov VOSA-SOVA; PLENA-NALEP; SLONI-SILON; SAKO-KOSA vyhovují oběma kritériím, přestože se jedná o naprosto rozdílná slova.

Řešením je přidání třetího kritéria, které nepřímo zohledňuje pořadí hlásek v rámci slova tím, že zkoumá jejich návaznost. Přičemž větší souvislá skupina hlásek má větší váhu, než třeba dvě menší.

Jednotlivým kritériím byl během testování algoritmu přidělen významový koeficient v určitém poměru, který určuje jakou měrou může kritérium ovlivnit výsledek.

Testování algoritmu ukázalo, že pro použití v praxi (kdy je uživateli nabídnuto jediné relevantní heslo) nestačí pouhý výpočet podobnosti, ale je třeba zvážit i kritérium významnosti takového slova. Slovo, které se používá častěji by mělo být nabízeno s větší pravděpodobností, než slovo málo používané, byť je koeficient podobnosti stejný. Proto byla pořízena data ve smyslu četnosti používání konkrétních slov v jednotlivých kategoriích. Např. pro slovo Jiří v kategorii křestních jmen lze použít údaj ze statistické tabulky křestních jmen (zdroj MVČR ke dni 09.05.2007), který uvádí, že celkově je v naší republice registrováno 319593 osob s křestním jménem Jiří. Porovnáním četností lze získat koeficient významnosti, který má zásadní vliv na výsledek algoritmu.

Výsledný algoritmus

Pro implementaci algoritmu byla zvolena databázová vrstva s využitím PL/SQL jazyka, v němž byla postupně vytvořena potřebná funkce `DID_YOU_MEAN(c_sQueryString,c_Category)`, jejímiž parametry jsou slovo, pro nějž je třeba nalézt vhodný ekvivalent, a kategorie. Slovník hesel je totiž pohled spojující všechna možná hesla v databázových tabulkách a kategorie určuje příslušnost hesla k dané tabulce.

Funkce nejprve zkoumá slovník, zda hledané slovo přeci jen není částí určitého hesla pomocí funce LIKE. Pokud nalezne shodu (může být i více), vrátí nejkratší nalezený ekvivalent. V opačném případě pokračuje dále v testování algoritmem, který přiděluje měřítko podobnosti. Funkce v paměti uchovává pouze proměnnou s nejvyšším dosavadním měřítkem a proměnnou s heslem které bylo parametrem dosaženého měřítka. Pokud algoritmus vyhodnotí heslo s měřítkem nižším, obě informace zahodí. V opačném případě zapíše do proměnných měřítko i heslo nové. Po vyhodnocení všech hesel ve slovníku se cyklus ukončí a funkce vrátí nalezený ekvivalent.

Pro širší uplatnění by bylo třeba tento algoritmus více optimalizovat (z hlediska výpočetní složitosti). Možným řešením je např. evidování vyhledávaných hesel (překlepů) v kontextu s relevantními hesly.

5.5.2 Způsob použití

Před voláním funkce DID_YOU_MEAN je testováno, zda se slovo nevyskytuje ve slovníku. Pokud ne, je vhodné funkci použít. Nalezený ekvivalent je prezentován podobně jako ve vyhledávačích. Pod vstupním polem se vypíše hláška „Měli jste na mysli: *hledané heslo*?“ Kliknutím na odkaz se řetězec v poli formuláře nahradí nalezeným.

6 Aplikace frameworku na zadané téma

Tématem této bakalářské práce v obecném pojetí je evidence osob a jejich osobních údajů (titul, jméno, příjmení, rodné číslo, bydliště, telefon, e-mail). Tato

kapitola rozebere jednotlivé položky formuláře a ukáže aplikaci vytvořeného frameworku na tuto položku.

6.1 Formulářová pole

6.1.1 Titul

Při vyplňování titulu je někdy obtížné přesně určit pořadí i dělící znaky jednotlivých zkratek. Zde je tedy příhodné použití našeptávače. Data poskytuje na svých stránkách Ministerstvo vnitra ČR, které ke dni 19. 5. 2006 eviduje 603 různých kombinací titulů. Tituly jsou ve formě souvislého řetězce, takže se nabízí otázka, jak prezentovat případné zkratky uváděné za jménem (např. při tisku obálek, či jiných tiskopisů). Naštěstí Ministerstvo vnitra vedle této spíše statistické tabulky poskytuje i klíč k jednotlivým zkratkám. Podle klíče lze určit váhu titulu, zda se jedná o titul či vědeckou hodnost a zda se píše před, nebo za jméno.

Syntaxe formulářového elementu:

```
<input type='text' id='TITUL'
onfocus="Suggest(this);next(this.form.FIRST_NAME);"/>
```

Z uvedené syntaxe lze vyčíst, že se jedná o formulářové pole (`type='text'`). Atribut `id` dále poslouží pro vyhledání příslušné tabulky a sloupce pro našeptávač. Událost `onfocus` spouští zadané funkce při aktivaci tohoto pole. Funkce `Suggest(this)` je jakýmsi aliasem pro volání metody `SuggestHandle` objektu `SuggestControl`. Argument `this` říká, že hledaná data se vztahují tomuto poli a zde budou našeptávána. Funkce `next(this.form.FIRST_NAME)` volá `focus_manager` a určuje, kterému poli má předat kontext.

SuggestControl reaguje na stisk klávesy (např. uživatel napíše In) a volá server s argumenty v tomto formátu:

```
URL:/PHP_SCRIPTS/SUGGESTIONS.PHP?TITUL=In
```

„In“ odpovídá současné hodnotě formulářového pole TITUL. Na serveru je předem definováno, jaké argumenty může očekávat a jakým způsobem z nich vytvořit SQL dotaz, kterým vyhledává v databázi relevantní hesla. Ukázka definice:

```
    $this->_tables_DESC['TITULY'][1] = "TITUL";
```

Název argumentu se shoduje s hodnotou pole definovaného na serveru. Tato hodnota koresponduje i s názvem sloupce tabulky, který je uveden jako klíčový index (TITULY). Tím server ví, kde hledat i co hledat. Provede automatickou formulaci SQL dotazu:

```
SELECT XMLelement("LIST_OF_TITULY",
(SELECT XMLAGG(XMLforest(TITUL AS TITUL))
  FROM (SELECT DISTINCT TITUL FROM TITULY WHERE
ROWNUM<=12 AND UPPER(TITUL) LIKE UPPER('In%') ORDER BY
TITUL))).getClobVal() as QU from dual;
```

Odpovědí je XML dokument, jehož elementy je dále naplněn našeptávač.

Při stiku klávesy ENTER, či TAB je vstupní hodnota pole potvrzena a FocusManager provede aktivaci pole uvedeného v argumentu.

6.1.2 Jméno a příjmení

Data pro našeptávač jmen a příjmení pocházejí ze stejného zdroje (MVČR), ve kterém je ke dni **9. 5. 2007** evidováno 47404 mužských a 36000 ženských křestních jmen a ke dni 28. 5. 2007 60224 mužských a 63601 ženských příjmení osob s platným pobytem na území ČR včetně cizinců. Do těchto statistik spadá i kombinace křestních jmen, tzn. případy, kdy osoba užívá více jmen (Anna Kateřina). To dává odpověď na otázku, jak zaevidovat osobu s více jmény. Našeptávač pak poskytuje přesný formát takového jména (jména bývají oddělena mezerou, ale i pomlčkou).

Syntaxe, volání serveru s žádostí o data a jejich následná prezentace formou našeptávače je podobná jako v případě titulu.

6.1.3 Rodné číslo

Rodné číslo je jednoznačný identifikátor (měl by být jednoznačný). První šestičíslí popisuje datum narození ve formátu *rrmmdd* , přičemž ženy mají k měsíci připočteno padesát. Zbývající čtyřčíslí je odděleno lomítkem a odlišuje lidi narozené ve stejný den. Na posledním místě je zbytek po dělení předešlého devítimístného čísla číslem jedenáct. Celé číslo by tedy mělo být dělitelné jedenácti.

Na rodné číslo nelze z logického hlediska aplikovat našeptávač. Zato je možné podrobit rodné číslo a jeho části důkladné analýze.

Pro kontrolu rodného čísla a jeho částí poslouží externí JavaScriptová funkce. Ta nejprve zkontroluje formát rodného čísla, zda vyhovuje regulárnímu výrazu. Pro kontrolu rodného čísla byl použit následující formát:

```
^[0-9]{6}/[0-9]{4}$
```

Nicméně v dokumentech popisujících rodné číslo je uvedeno, že do roku 1954 následovaly za lomítkem tři a ne čtyři cifry. Proto byla nutná úprava akceptující obě verze:

```
^[0-9]{6}/[0-9]{3,4}$
```

Pokud proběhne kontrola formátu v pořádku, je možné přistoupit ke kontrole dílčích částí, kontrola měsíce (1-12 či 51-63) a dne (1-31). Pokud i tato kontrola proběhne v pořádku, je provedena zkouška, zda se opravdu jedná o reálné datum (např. 31 .02. by předchozí kontrolou prošlo, ačkoliv má únor maximálně 29 dní).

Nakonec je testována dělitelnost jedenácti. Pokud vše proběhne v pořádku, funkce vyhodnotí rodné číslo jako správné. V opačném případě nastaví zprávu chybového hlášení a příslušnou ikonu.

6.1.4 Obec, část obce, PSČ a ulice

Obec, část obce, PSČ a ulice spolu úzce souvisí. Proto je potřeba jednotlivé vazby respektovat. Pokud například pole Obec obsahuje hodnotu „Čáslav“ a uživatel vyplňuje pole ulice, není vhodné, aby mu našeptávač nabídl seznam všech ulic v ČR, byť abecedně uspořádaných. Jednak by mohl být maten tím, že v našeptávači vybere ulici, která v příslušné obci vůbec není, ale některé ulice mají i různý způsob interpretace, např. B. Smetany, Bedřicha Smetany, Bedř. Smetany, což by mohlo být zcela zavádějící. Cílem tedy je, aby při formulaci podmínek dotazu byly použity všechny dostupné informace.

Framework umí řešit i takto náročnou úlohu, přičemž syntaxe je i v tomto případě jednoduchá:

```
<input type='text' id='CITY'
onfocus="Suggest(this,this.form.PART,this.form.ZIPCODE);
```

```
        next(this.form.ZIPCODE);"/>
<input type='text' id='ZIPCODE'
onfocus="Suggest(this,this.form.CITY,this.form.PART);
        Checking(this);" class='ZIPCODE'/>
<input type='text' id='STREET'
onfocus="Suggest(this,this.form.CITY,this.form.ZIPCODE,t
his.form.PART);"/>
```

Našeptávači se jako argumenty předají vlastní aktivní pole, další pole, která mohou ovlivnit výsledek. Důležité je, aby prvním argumentem bylo aktivní pole, neboť server při zpracovávání argumentů dotazu pokládá první argument za hlavní a vrátí data ze sloupce se stejným jménem jako je id pole.

6.1.5 E-mail

Kontrola e-mailu může probíhat na různých úrovních. Primární úrovní je integritní omezení databáze ve formě regulárního výrazu. Dále se používá kontrola existence MX záznamu. MX záznam (Mail exchanger record) je typ zdrojového záznamu v DNS (systém doménových jmen). PHP pro ověření existence tohoto záznamu používá funkci `checknsrr(string $host [,string $type])`, která vrátí `TRUE` v případě, že záznam nalezne. Zjišťovat MX záznam má samozřejmě smysl pouze tehdy, pokud je e-mail ve správném tvaru (i12).

6.1.6 Číslo telefonu

Od roku 2002, kdy došlo v ČR k přečíslování telefonních linek se používají tvary telefonního čísla uvedené v tabulce 6.1 (Tabulka formátů telefonních čísel) (i13).

Kanonický formát	+420 xxx xxx xxx
Zkrácený formát	xxx xxx xxx
GSM formát	+420xxxxxxxxx

Tabulka 6.1. Tabulka formátů telefonních čísel

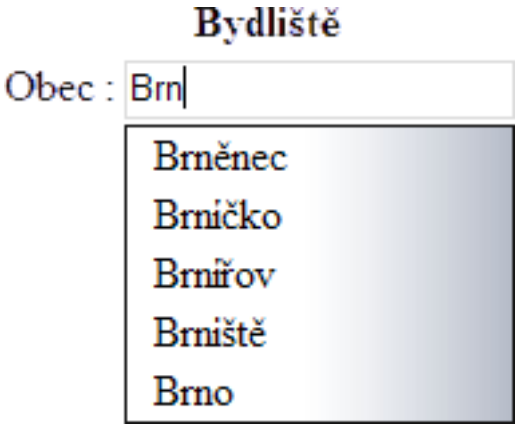
Pro potřeby evidence je použit kanonický nebo zkrácený formát bez mezer, tzn. formát vyhovující tomuto regulárnímu výrazu:

```
^(\+420){0,1}[0-9]{9}$ .
```

6.2 *Ukázky testovací aplikace*

6.2.1 SuggestBox

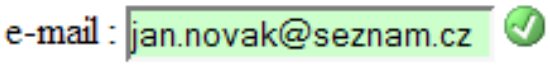
Na obrázku x je ukázka našeptávače (našeptávač pro pole obec). Pohyb mezi položkami našeptávače je realizován kurzorovými klávesami nahoru a dolu, nebo pomocí myši.



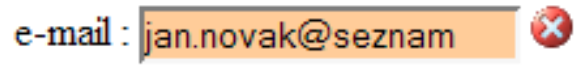
Obrázek 12: Našeptávač (Obec)

6.2.2 Test shody s regulárním výrazem

Na obrázcích 13 a 14 je ukázka testování zadávaného řetězce vůči omezení definovaným nad databází formou regulárního výrazu. Test se provádí při každém stisku klávesy ve sledovaném poli a uživatel je informován změnou pozadí vyplňovaného pole.



Obrázek 13: Ukázka testování řetězce (správný výsledek)



Obrázek 14: Ukázka testování řetězce (byla odhalena chyba)

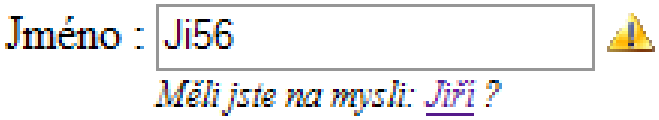
6.2.3 Testování pomocí kontrolních funkcí

Kontrolní funkce definované na úrovni JavaScriptu poskytují další nástroj kontroly (jak již bylo uvedeno v kapitole 5.3). Na obrázku 15 je ukázka výsledku kontroly pomocí funkce ověřující významovou správnost rodného čísla.

Obrázek 15: Ukázka výsledku kontroly významové správnosti rodného čísla

6.2.4 Funkce „Měli jste na mysli?“

V případě, že není v databázi nalezeno zadané heslo, je možné k němu vyhledat a nabídnout vhodný ekvivalent. Volba spuštění této funkce je ponechána na uživateli. V případě, že je ekvivalent nalezen, zobrazí se pod sledovaným polem známá hláška „Měli jste na mysli: ?“.



Obrázek 16: Ukázka funkce "Měli jste na mysli?"

6.3 Konečná kontrola

Před odesláním je formulář podroben celkové kontrole. Pro celkovou kontrolu není třeba dopisovat žádné nové funkce. Lze využít volání ověření formulářového

pole, které již zná ověřovací metody. Před odesláním projde všechny formulářové prvky a ověří správnost. Pokud nastane během kontroly chyba, upozorní na ni chybovým hlášením, kde je uvedena specifikace této chyby pro snadné opravení. Pokud kontrola proběhne v pořádku, data se uloží do databáze.

6.4 Uživatelský test

Aplikaci testovalo 20 uživatelů různých profesí a různými zkušenostmi s IT. Testování odhalilo následné nedostatky:

- pro kontrolu shody s regulárními výrazy je třeba uživatele informovat o požadovaném formátu řetězce
- informační rámce výsledků kontrol pomocí kontrolních funkcí je třeba zobrazit na popředí
- funkci „Měli jste na mysli?“ je třeba spouštět na základě požadavku uživatele, nikoliv automaticky
- funkci „Měli jste na mysli?“ je třeba doplnit o zohlednění významnosti jednotlivých hesel

Nedostatky byly většinou odstraněny na úrovni JavaScriptu (změna uživatelských funkcí). Pro funkci „Měli jste na mysli?“ byla pořízena další data v podobě koeficientů významnosti jednotlivých hesel které tato funkce nadále zohledňuje.

Další testování (na stejném statistickém vzorku) mělo kladné ohlasy a 100% uložených dat bylo validních tj. odpovídalo všem požadavkům.

6.5 Implementace frameworku

Proces implementace je velice jednoduchý. Balík je koncipován tak, aby bylo možné kdykoliv vyměnit jakýkoliv script, ať na úrovni JavaScriptu, nebo PHP scriptů. Framework byl během návrhu i testování ošetřen proti řadě chyb při implementaci. Pokud se přesto nějaká vyskytne, neměla by mít zásadní vliv na výsledné odeslání dat do databáze.

Na úrovni JavaScriptu je třeba do stránky vložit script, či lépe odkaz na externí soubor obsahující příslušný modul. Syntaxe je následná:

```
<script language="javascript" type="text/javascript"
src="JAVASCRIPTS/ajax.js"></script>
```

6.5.1 SuggestBox

Volání funkce našeptávače je jednoduché. Stačí vstupní pole přiřadit do `suggest_control` spolu s parametrem čemu a v závislosti na čem má být našeptáváno. Vhodnou událostí pro přiřazení pole do kontrolleru může být `onfocus()`. Hodnota atributu ID by se měla shodovat s názvem sloupce definovaným v konfiguračním souboru `TABLE_DESC.xml` na serveru. Syntaxe je následná:

```
  <input name='CITY' type='text' id='CITY'
onfocus="Suggest(this,this.form.ZIP,this.form.PART);/>
```

PHP script, který formuluje dotazy pro databázi je třeba informovat o struktuře databáze a určit kde hledat data. Po testování samotného frameworku byl zvolen způsob definování struktury v externím xml souboru.

```
<DATABASE_DESCRIPTION>
 <OBJECT name='SB_FIRST_NAMES'>
   <COLUMN name='FIRST_NAME'>FIRST_NAME</COLUMN>
</DATABASE_DESCRIPTION>
```

6.5.2 Kontrola pomocí regulárních výrazů

Je-li v databázi definováno integritní omezení nad určitým sloupcem využívající možností regulárních výrazů, lze tento regulární výraz přenést na úroveň JavaScriptového kódu. Při formulaci SQL dotazu hledajícího regulární výraz je použit stejný konfigurační soubor jako případě našeptávače. V něm je určena závislost mezi názvem omezení v databázi a atributu `class` vstupního pole formuláře.

Syntaxe pro použití kontroly je potom následná:

```
<input type='text' class='MAIL'
onfocus="Checking(this);"/>
```

6.5.3 Kontrola pomocí uživatelských funkcí

Uživatelská funkce přebírá parametr message, což je objekt, kterému lze nastavit level (stupeň závažnosti chyby) a text (informativní hláška). Funkce sama najde vstupní pole a zkontroluje jeho obsah. Podle výsledku kontroly nastaví level a text.

Dále je třeba určit událost, která spustí proces testování. Událost se určí jako parametr konstruktoru objektu Controller, který přijímá dva parametry: uživatelskou funkci a událost. Syntaxe je následná:

```
var PIN_CONTROLLER = new Controller(PIN_CTRL ,'keyup');
```

Dále je třeba tento objekt přidat do správce kontrollerů a určit, které pole bude kontrolovat. Do správce kontrollerů lze přidat i více kontrolních objektů. Syntaxe je následná:

```
Watch(dataform.ZIP,  ZIP_PASSIVE_CONTROLLER,
ZIP_ACTIVE_CONTROLLER);
```

7 Použití frameworku v praxi

V prostředí Internetu je mnoho webových stránek s různými formuláři (e-shopy, e-government, ...), ve kterých by služba aktivní kontroly data mohla nalézt uplatnění.

Ve většině formulářů je použití takové služby více než vhodné. Otázkou je, jak tuto službu zprostředkovat. Nabízí se spousta možností.

7.1 Užití frameworku jako modulu (balíku)

Je možné poskytnout celý naprogramovaný balík (včetně pořízených dat). Takový balík by byl v konečné fázi součástí zdrojových kódu každé konkrétní webové aplikace. Programátor by byl pouze informován o možné parametrizaci k dosažení požadovaného výsledku, či by mohl měnit jednotlivé funkce či celé moduly. Výhodou je absolutní volnost ze strany programátora při změně kódu či parametrizaci funkcí. Rovněž rychlost odezvy je závislá pouze na hardware provozovatele. Nevýhodou je nutnost instalace Oracle databáze a vytvoření požadováných databázových struktur, včetně naplnění daty.

7.2 Poskytování služby formou webového rozhraní

Bylo by možné dle požadavků vytvořit různá webová rozhraní, kam by byli přesměrovávání potencionální návštěvníci provozovatelů stránek, kteří by měli o tuto službu zájem. Zadaná data by byla odesílána provozovateli. Výhodou je zajištění kompletního servisu, včetně přístupu k datům a funkcím již hotové databáze. Nevýhodou je nutnost přesměrování a přizpůsobování stylu rozhraní stránkám provozovatele.

7.3 Poskytování služby formou programového rozhraní

Nejlepší možnou formou využití služby pro různá využití v prostředí Internetových aplikací je poskytnutí programového rozhraní. Uživatel si objedná balík služeb, je mu zaslán soubor se scriptem zajišťujícím komunikaci s rozhraním. Rozhraní může odesílat jako odpovědi i JavaScriptové funkce, které jsou následně prováděny na straně klienta. Výhodou je snadné začlenění do konkrétních stránek, bez nutnosti instalace vlastní databáze a nezávislost na použití jazyka na úrovni serveru.

8 Závěr

Cílem mé bakalářské práce bylo vytvořit znovupoužitelný systém pro zajištění integrity zadávaných dat v prostředí webového formuláře. Využil jsem k tomu některé již zaběhlé a uživateli Internetu dobře známé funkce (SuggestBox, „měli jste na mysli“) nebo navrhl a implementoval zcela nové, které lze používat stejně intuitivně.

Myslím, že používání všech možných nástrojů pro kontrolu vstupních údajů je velice důležité. Nejen se zefektivní práce člověka (uživatele), který formulář používá, ale zkvalitní se tím i databáze, která bude obsahovat jen správná data. S takovými daty (v jednotném formátu) je později mnohem jednodušší a efektivnější práce.

Během práce se podařilo získat velké množství dat pro evidenci osob jako jsou křestní jména, příjmení, obce, části obcí a ulice. Tato data mají svou hodnotu a lze je použít jak v této konkrétní aplikaci, ale i v dalších, podobných, či jako testovací data.

Cíle se podařilo dosáhnout, neboť systém je nejen obecně použitelný, ale chová se velice otevřeně. Lze naprogramovat jakoukoliv funkci a tu zařadit do systému pro kontrolu konkrétního pole.

Uživatelský test prokázal, že ke zkvalitnění dat opravdu došlo (všechna vložená data byla validní) a ohlasy uživatelů byly velice kladné (rychlé osvojení všech nabízených funkcí).

Tento systém je možné používat podobně jako v případě evidence osob, nebo i dále rozšiřovat o další kontrolní a pomocné funkce.

Použitá literatura

[1] LACKO, Luboslav *ORACLE : Správa, programování a použití databázového systému*. Kadlec Václav; Kocan Marek. 1. vyd. Brno : Computer Press, 2007. 576 s. ISBN 978-80-251-1490-2.

[2] MATIAŠKO, Karol *Databázové systémy*. Šimánková Katarína. 1. vyd. Žilina : EDIS - vydavatelstvo ŽU, 2002. 383 s. ISBN 80-7100-968-7.

[3] FLANAGAN, David *JavaScript : Kompletní průvodce*. Odpovědný redaktor: Ivo Magera; překlad Karel Voráček , David Krásenský. 2. aktualiz. vyd. Praha : Computer Press, 2002. 825 s. ISBN 80-7226-626-8.

[4] POWEL, Thomas A. *Web design : Kompletní průvodce*. Odpovědný redaktor: Libor Pácl; překlad Petr Matějů. Brno : Computer Press, 2004. 818 s. ISBN 80-722-6949-6.

[5] HOLZNER, Steven *Mistrovství v AJAXu*. Odpovědný redaktor: Martin Domes; překlad Jakub Zemánek. 1. vyd. Brno : Computer Press, 2007. 591 s. ISBN 978-80-251-1850-4.

[6] ASLESON, R.; SCHUTTA, N. T. *Ajax : Vytváříme vysoce interaktivní webové aplikace*. Odpovědný redaktor: Ivo Magera; překlad Jakub Zemánek. 1. vyd. Brno : Computer Press, 2006. 269 s. ISBN 80-251-1285-3.

[7] KREINES, David C. *Oracle DBA : Kapesní průvodce*. Odpovědný redaktor: Martin Kysela. 1. vyd. Praha : Grada Publishing, 2006. 156 s. ISBN 80-247-1669-0.

[8] FLANAGAN, David *JavaScript : Kapesní příručka*. 2. vyd. Gliwice, POLSKO : Helion, 2004. 154 s. ISBN 83-7361-466-4.

[9] MCCONEL, Steve *Dokonalý kód : Umění programování a techniky tvorby software*. Odpovědný redaktor: Martin Domes; překlad Bogdan Kiszka. Brno : Computer Press, 2005. 900 s. ISBN 80-251-0849-X.

[10] KOSEK, Jiří. *XML pro každého : Podrobný průvodce*. Praha : Grada Publishing, 2000. 164 s. ISBN 80-7169-860-1.

[i1] *Regulární výraz* [online]. 6. 12. 2004 , 24. 4. 2008 [cit. 2008-04-30]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Regul%C3%A1rn %C3%AD_v%C3%BDraz>.

[i2] *Literál* [online]. 7.2.2006 , 16. 12. 2006 [cit. 2008-12-06].

Dostupný z WWW: <http://cs.wikipedia.org/wiki/Liter%C3%A1l>.

[i3] VÁCLAVÍK, Jiří. *Regulární výrazy : opakování a kvantifikátory* [online]. 18.11.2005 [cit. 2008-04-01]. Dostupný z WWW: <http://www.linuxsoft.cz/article.php?id_article=1018>.

[i4] PECKA, Miroslav. *Základy regulárních výrazů* [online]. 2005-2008 [cit. 2008-04-01]. Dostupný z WWW: <http://www.regularnivyrazy.info/regularni-vyrazy-zaklady.html>.

[i5] *JavaScript* [online]. 12.8.2004 , 12.3.2008 [cit. 2008-04-20]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Javascript>.

[i6] *XMLHttpRequest* [online]. 18.12.2006 [cit. 2008-04-20]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=XMLHttpRequest&limit=500&action=history>.

[i7] *PHP* [online]. 2.6.2004 , 25.4.2008 [cit. 2008-04-20]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/PHP>.

[i8] *Asynchronous JavaScript and XML* [online]. 11.10.2005 , 17. 3. 2008 [cit. 2008-04-21]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/AJAX>.

[i9]*Teorie relačních databází : Integritní omezení* [online]. 2005-2006 [cit. 008-04-21]. Dostupný z WWW: <http://www.manualy.net/article.php?articleID=15>.

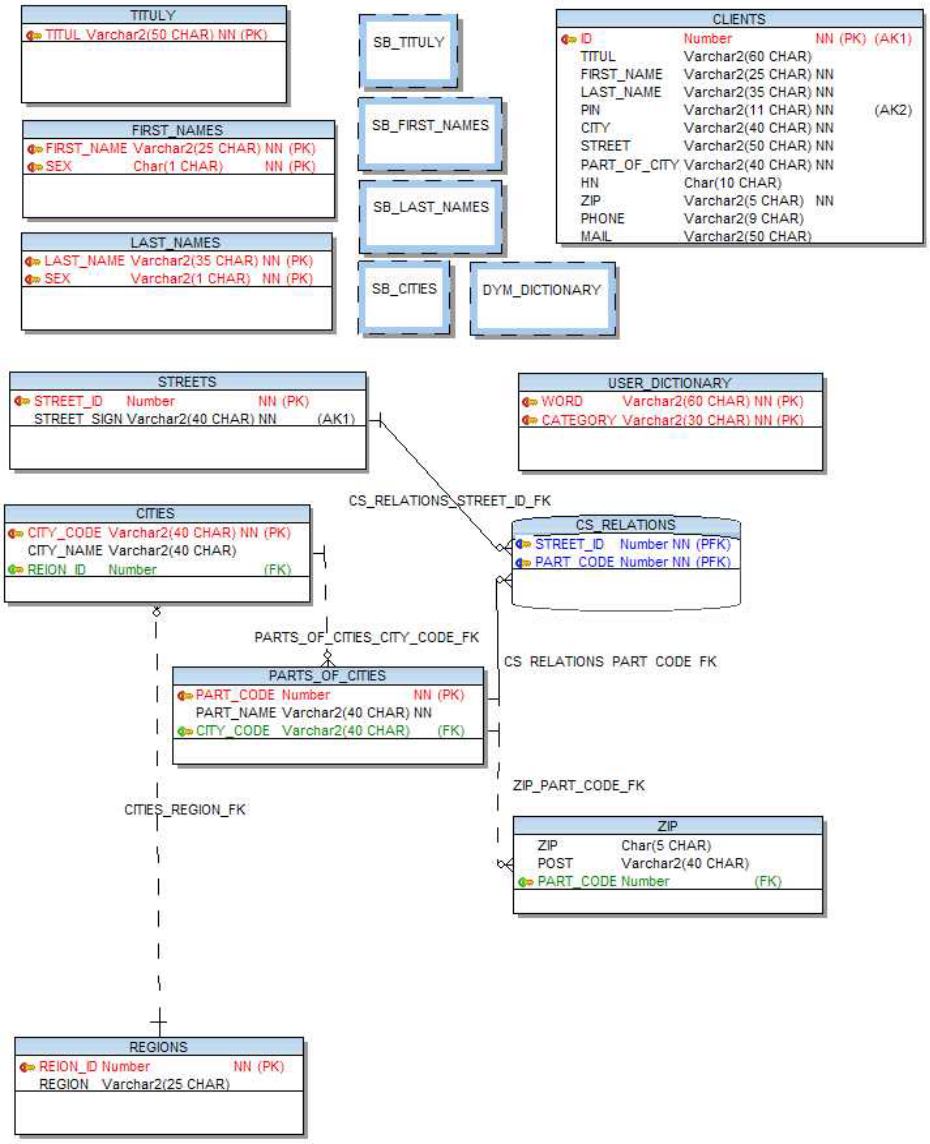
[i10] *Extensible Markup Language* [online]. 13.7.2004 , 14. 4. 2008 [cit. 2008-04-21]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Extensible_Markup_Language>.

[i11] *Okres* [online]. 17.8.2004 , 25. 4. 2008 [cit. 2008-04-20]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Okres>.

[i12] *Checkdnsrr* [online]. 2001-2008 , Fri, 25 Apr 2008 [cit. 2008-04-30]. Dostupný z WWW: <http://www.php.net/manual/en/function.checkdnsrr.php>.

[i13] Vlastnosti a formáty vytáčených telefonních čísel [online].[2004] [cit.2008-05-01]. Dostupný z WWW: <http://www.602.cz/lexik602/msg00124.htm#580dfccd>.

Příloha A


TITULY
TITUL Varchar2(50 CHAR) NN (PK)
SB_TITULY
CLIENTS
ID Number NN (PK) (AK1)
TITUL Varchar2(60 CHAR)
FIRST_NAME Varchar2(25 CHAR) NN
LAST_NAME Varchar2(35 CHAR) NN
PIN Varchar2(11 CHAR) NN (AK2)
CITY Varchar2(40 CHAR) NN
STREET Varchar2(50 CHAR) NN
PART_OF_CITY Varchar2(40 CHAR) NN
HN Char(10 CHAR)
ZIP Varchar2(5 CHAR) NN
PHONE Varchar2(9 CHAR)
MAIL Varchar2(50 CHAR)
FIRST_NAMES
FIRST_NAME Varchar2(25 CHAR) NN (PK)
SEX Char(1 CHAR) NN (PK)
SB_FIRST_NAMES
SB_LAST_NAMES
LAST_NAMES
LAST_NAME Varchar2(35 CHAR) NN (PK)
SEX Varchar2(1 CHAR) NN (PK)
SB_CITIES
DYM_DICTIONARY
STREETS
STREET_ID Number NN (PK)
STREET_SIGN Varchar2(40 CHAR) NN (AK1)
USER_DICTIONARY
WORD Varchar2(60 CHAR) NN (PK)
CATEGORY Varchar2(30 CHAR) NN (PK)
CS_RELATIONS_STREET_ID_FK
CITIES
CITY_CODE Varchar2(40 CHAR) NN (PK)
CITY_NAME Varchar2(40 CHAR)
REION_ID Number (FK)
CS_RELATIONS
STREET_ID Number NN (PFK)
PART_CODE Number NN (PFK)
PARTS_OF_CITIES_CITY_CODE_FK
CS_RELATIONS_PART_CODE_FK
PARTS_OF_CITIES
PART_CODE Number NN (PK)
PART_NAME Varchar2(40 CHAR) NN
CITY_CODE Varchar2(40 CHAR) (FK)
ZIP_PART_CODE_FK
CITIES_REGION_FK
ZIP
ZIP Char(5 CHAR)
POST Varchar2(40 CHAR)
PART_CODE Number (FK)
REGIONS
REION_ID Number NN (PK)
REGION Varchar2(25 CHAR)