Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vytvoření webového uživatelského rozhraní pro stávající CRM systém

Plzeň 2016 Josef Lexa

Originální zadání

Zde bude originální zadání práce. V kopii práce potom oboustranná kopie tohoto zadání.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 13. června 2016

Josef Lexa

Poděkování

Chtěl bych poděkovat Ing. Václavu Strychovi za vypsání této bakalářské práce a za jeho čas při konzultacích. Dále také Ing. Martinu Zímovi, Ph.D. za odborné vedení práce a cenné rady a zdroje, které mi pomohly tuto práci vytvořit. Děkuji také své rodině za podporu při studiu a své přítelkyni za motivování k této práci a pomoc při kontrole jejího textu.

Abstract

Creation of a web user interface for an existing CRM system

The aim of this work is to explain to a reader the problems of the database and application layers of the M/Data program. To analyse the above layers and in detail describe its results and possible solutions. Based on this analysis, the database and application layers have been optimized and the web user interface was created which includes the main features of the current desktop program. The main finding was that the database and application layers can be easily optimized to the point where they are fully applicable for the creation of a web user interface.

CRM system, web application, database optimalization

Abstrakt

Vytvoření webového uživatelského rozhraní pro stávající CRM systém

Cílem této práce je čtenáři vysvětlit problematiku databázové a aplikační vrstvy programu M/Data. Nad těmito vrstvami provést analýzu a detailně popsat její výsledky včetně návrhů možných řešení. Na základě této analýzy byla databáze a aplikační vrstva optimalizována a bylo vytvořeno webové uživatelské rozhraní, které obsahuje hlavní funkce stávajícího desktopového programu. Hlavním zjištěním bylo, že databázovou a aplikační vrstvu lze vcelku snadno optimalizovat do stavu, kdy je plně použitelná pro vytvoření webového rozhraní.

CRM systém, webová aplikace, optimalizace databáze

Obsah

Seznam obrázků

1 Úvod

Cílem této práce je provedení analýzy a vytvoření webového rozhraní ke stávajícímu informačnímu systému M/Data firmy Kadel Data Servis s.r.o.

M/Data je CRM[1] informační systém vyvinutý pro podporu řízení vztahů se zákazníky ve společnostech. V současné době disponuje pouze verzí pro desktop, což je v dnešní době do značné míry nedostatkem. Právě tento problém byl impulsem pro vznik zadání této bakalářské práce.

Výstupem by tedy měla být přehledná a funkčně vyhovující webová aplikace, jenž bude splňovat většinu funkcí stávající desktopové verze. Celá aplikace poběží na frameworku .NET vyvíjeným společností Microsoft. Pro správu dat by se měl využít SŘBD[2] Microsoft SQL Server. Stávající systém komunikuje s databází díky komunikační vrstvě, která by na základě výsledků analýzy mohla být použita i pro webovou aplikaci.

Webová aplikace umožní uživatelům práci s osobami a společnostmi. Konkrétně stránkované výpisy, vyhledávání, přidání, editaci a mazání záznamů a vytváření pohledů na záznamy s možností filtrace a seřazení podle vybraného atributu.

Při tvorbě bakalářské práce jsem vycházel především ze zdrojů získaných od externího zadavatele, internetových zdrojů nebo případně z odborné literatury zabývající se tvorbou webových a databázových aplikací.

[1] Customer relationship management
[2] Systém řízení báze dat

2 Analýza stávající databáze

2.1 Popis jednotlivých tabulek databáze

Jako první část této kapitoly jsem zvolil analýzu jednotlivých tabulek databáze. Tato znalost mi přijde před samotným analyzováním vztahů a ke správnému porozumění databáze důležitá. Primární databázový systém, který jsem používal po celou dobu analýzy, je Microsoft SQL Server. Důvodem bylo doporučení externím zadávajícím na základě dosavadních zkušeností a využití tohoto systému s desktopovou aplikací M/Data a plné kompatibility s touto platformou. V poslední řadě mně v této volbě utvrdily i mé osobní sympatie.

2.1.1 Tabulka mdContact

Významem této tabulky je jeden z hlavních smyslů CRM systémů, čímž je uchovávání kontaktů osob. Z toho důvodu patří společně s tabulkou `mdCompany` ke dvěma nejobsáhlejším v rámci této práce. Primárním klíčem je atribut `synchID`, využívá automatické přiřazení číselné hodnoty `auto_increment`. Další atributy jsou identifikátory sloužící pro programy třetích stran, jejich využití je mimo rozsah zadání této práce. Pro ukládání informací o osobě slouží posloupnost dalších atributů, jsou to např. `stdFirstName`, `stdBirthdate`, `stdHomeCity` a další. Poslední důležitou skupinu jsou atributy pro uživatelské položky. Primárně jsou v databázi tyto atributy dva pro každý datový typ, po vyčerpání lze vytvořit další, všechny jsou zmíněny v Tabulka 2.1. Za zmínku stojí také vysvětlit rozdíl mezi atributy `stdLastActivity` a `synchLastChange`, které by se mohly na první pohled zdát duplicitní. Atribut `stdLastActivity` je uživatelská položka, která určuje, kdy jsme byli s osobou naposledy ve spojení. Druhý zmíněný atribut určuje datum poslední změny osoby. Slouží tedy pouze pro chod programu, nikoli cílovému uživateli.

2.1.2 Tabulka mdCompany

Tato tabulka má stejný význam jako předchozí zmíněná, slouží ovšem k uchovávání společností a informací k nim příslušných. Všechny atributy této tabulky začínají prefixem `"comp_*"`. To slouží k jedinečnosti názvů, pokud by

bylo potřeba vytvořit stejné atributy jako v tabulce `mdContact` a k přehlednosti dotazů při spojování tabulek. Primárním klíčem je atribut `comp_synchID` opět s vlastností `auto_increment`. Také tato tabulka obsahuje identifikátory sloužící ostatním programům. Pro ukládání informací týkajících se společností je několik atributů včetně možnosti ukládání uživatelských položek, identicky jako u tabulky `mdContact`. Opět jsou předdefinované dvě pro každý datový typ s možností přidání dalších. Posledními atributy jsou stejně jako u předchozí tabulky `stdLastActivity` a `synchLastChange` jejichž význam byl již objasněn.

2.1.3 Tabulka mdRelation

Jedná se o rozkladovou tabulku pro vztah M:N mezi `mdContact` a `mdCompany`. Samotný význam vztahu bude vysvětlen v následující podkapitole. Pro názvy atributů je použit prefix `"rel_*"`. Primárním klíčem je atribut `rel_synchID`, který využívá opět `auto_increment`. Dále v tabulce najdeme dva cizí klíče `rel_cont_synchID` a `rel_comp_synchID`, sloužící k realizaci již zmiňovaného vztahu. Jako další obsahuje tabulka atributy pro specifikaci vztahu. Jsou to např. `rel_stdDepartment` a `rel_stdPosition` pro oddělení a pozici nebo `rel_stdMail`, který můžeme v programu využít pokud chceme pro komunikaci na tomto vztahu použít e-mail. Rozkladová tabulka také obsahuje volné atributy pro budoucí uživatelské položky, zmíněné v tabulce 2.1.

Uživatelské položky	
Název položky	Datový typ
usr_str	varchar(4000)
usr_int	int
usr_flt	float
usr_bol	smallint
usr_dat	datetime
usr_tim	datetime
usr_blb	image

Tabulka 2.1: Uživatelské položky

2.1.4 Tabulka mdPredefinedValueGroup

Tato tabulka slouží pro skupiny předdefinovaných hodnot. Díky tomu se můžou hodnoty z tabulky `mdPredefinedValue` slučovat. To poté v programu lze využít pro výběrová pole neboli comboboxy. Primárním klíčem je `groupId`, ten je datového typu `varchar` a uchovává název skupiny. Skládá se z prefixu `"std"` a jména skupiny. Dále jsou v tabulce tři atributy. Prvním je `description` uchovávající popis skupiny. Druhý `external_reference` slouží pro uložení identifikátoru k načtení hodnot z nějakého externího systému. Posledním je `readonly`. Určuje, zda položky spadající pod tuto skupinu mohou být editovány.

2.1.5 Tabulka mdPredefinedValue

Slouží jako číselník pro předdefinované hodnoty. V programu může hodnoty zadat sám uživatel nebo mohou být zadány i administrátorem předem přímo do databáze. Primárním klíčem této tabulky je atribut `valueId`. Dále má tabulka cizí klíč `groupId`, ten slouží pro zařazení předdefinované položky do nějaké skupiny z tabulky `mdPredefinedValueGroup`. Posledním atributem je `predefined_value`, slouží pro uložení obsahu předdefinované hodnoty.

2.1.6 Tabulka mdUserItemsDef

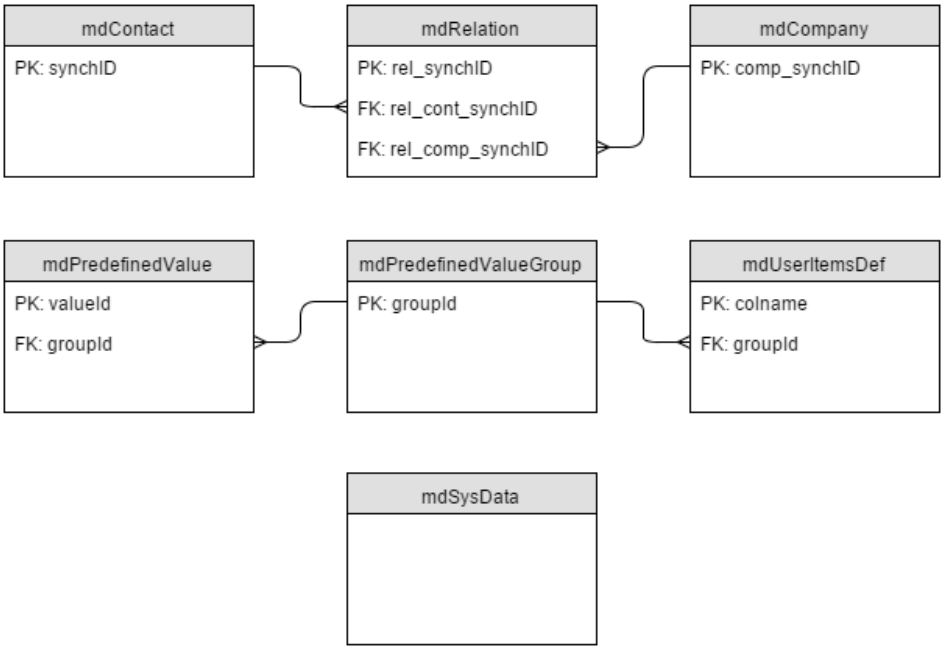
Významem této tabulky je uchování informací o uživatelských položkách. Uživatelskou položkou je v programu míněno např. textové pole nebo výběrové pole. Primárním klíčem je `colname`, nese název atributu uživatelské položky a je datového typu `varchar`. Tento atribut byl zvolen primárním klíčem z důvodu ochrany proti duplicitě názvů uživatelských položek. Tabulka obsahuje také cizí klíč `predefined_groupId`. Do něj se ukládá `id` skupiny předdefinovaných hodnot, pokud chceme, aby uživatelská položka měla nějaké hodnoty na výběr nebo měla nějakou danou hodnotu předvyplněnou. Důležitým atributem je také `valtype` datového typu `integer`. Nese číselnou hodnotu, podle které se později určí datový typ položky. Atribut `relationship` obsahuje číselnou hodnotu, podle níž se určí, pro kterou tabulku je položka vytvořena. Posledním atributem je `predefined_multiselect`. Definuje, zda uživatelská položka bude moci obsahovat výběrové pole.

2.1.7 Tabulka mdSysData

Nakonec je v databázi také tabulka uchovávající data pro správný chod celého programu. Konkrétně nese informace o verzi databáze a o posledních identifikátorech osoby, společnosti, vztahu a předdefinované hodnoty. Tabulka nemá žádné vztahy ani spojitost s ostatními zmiňovanými tabulkami.

2.2 Entity-relationship model

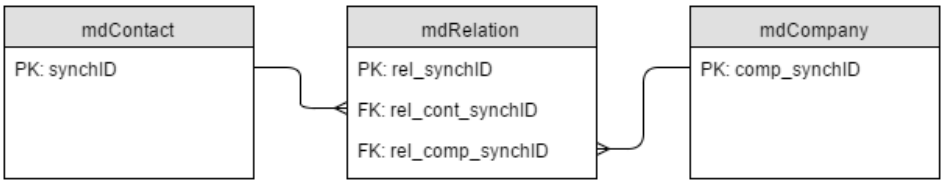
Entity-relationshop model, dále jen ERM, se používá v softwarovém inženýrství pro konceptuální návrhy databází. [1] V tomto případě se dá díky ERM lépe porozumět již vytvořené stávající databázi programu M/Data. Na modelu vysvětlím jednotlivé vztahy mezi tabulkami databáze a jejich význam pro chod programu. ERM také použiji pro návrh a vysvětlení jiných možných řešení vztahů stávajícího modelu.



Obrázek 2.1: Kompletní ER model databáze

2.2.1 Vztah mdContact - mdCompany

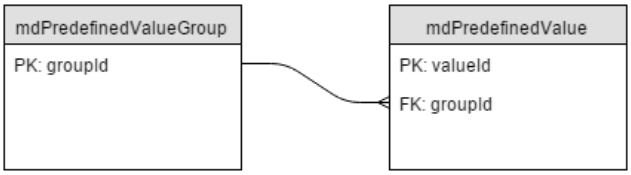
Vztah mezi osobou a společností je typu M:N. Je tedy realizován rozkladovou tabulkou `mdRelation`. Jedna osoba může být přidělena k více společnostem a naopak k jedné společnosti může být přiděleno více osob. V programu to slouží k vytvoření struktury vztahů a návazností mezi zmiňovanými entitami a ukládání informací o těchto vztazích.



Obrázek 2.2: ER model vztahu mdContact - mdCompany

2.2.2 Vztah mdPredefinedValueGroup - mdPredefinedValue

Jedná se o vztah 1:N. Jedna předdefinovaná hodnota může spadat pouze pod jednu skupinu hodnot. Vztah tedy zaručí seskupení určitých předdefinovaných hodnot do jedné skupiny. Tyto skupiny lze poté využívat pro výběrové položky díky následujícímu vztahu.

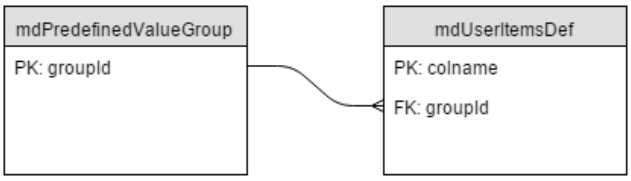


Obrázek 2.3: ER model vztahu mdPredefinedValueGroup - mdPredefinedValue

2.2.3 Vztah mdPredefinedValueGroup - mdUserItemsDef

Jde opět o vztah 1:N. Uživatelské položce může být přidělena nějaká skupina předdefinovaných hodnot a zároveň skupina může být přidělena k více uživatelským položkám. V praxi je tento vztah využíván pro uživatelské položky,

jenž jsou výběrové, tedy mají na výběr z několika předdefinovaných hodnot, které spadají pod danou skupinu, viz předchozí popsaný vztah.



Obrázek 2.4: ER model vztahu mdPredefinedValueGroup - mdUserItemsDef

2.3 Problém uživatelských položek

Zde bych se chtěl vrátit k uživatelským položkám tabulek `mdContact`, `mdCompany` a `mdRelation`. Tento problém není v tuto chvíli z pohledu relačních databází zcela správně vyřešen, ale má to svá odůvodnění. Ty bych zde chtěl objasnit a zároveň uvést jiná možná řešení tohoto problému a obě možnosti navzájem porovnat.

2.3.1 Aktuální řešení problému

Jak jsem již zmiňoval u popisu jednotlivých tabulek, uživatelské položky jsou uchovávány přímo v nich. Tento fakt by nebyl taková přítěž, ale problém nastává ve chvíli, kdy potřebujeme přidat další uživatelskou položku, pro kterou není už volný atribut s příslušným datovým typem. V tento okamžik se musí přidat atribut do tabulky, kde má být položka ukládána. Neboli musí se vykonat databázový příkaz `ALTER TABLE` a tomu by se v relačních databázích za chodu systému mělo předcházet. [2] Toto řešení bylo zvoleno na základě následujících náležitostí:

- Snazší implementace rozhraní mezi databází a programovou částí.
- Snazší formulace dotazů na databázi.

Tento způsob přináší snazší implementaci rozhraní mezi programovou a databázovou částí díky nepotřebě spojování vybraných dat z databáze do jednoho záznamu pro výpis v programu. Další výhodou je snazší dotazování,

kdy nám stačí pouze jednoduchý dotaz `SELECT` pro získání celého záznamu. Tento záznam získáme včetně všech k němu příslušných uživatelských položek. Snazší je také filtrování nad jednotlivými atributy neboť filtruji nad jedním spojeným záznamem.

2.3.2 Jiné možné řešení problému

Jiných a z pohledu relačních databází korektnějších řešení je více. Zde zmíním jedno, které popíši a znázorním na ER modelu (viz obrázek 2.5).

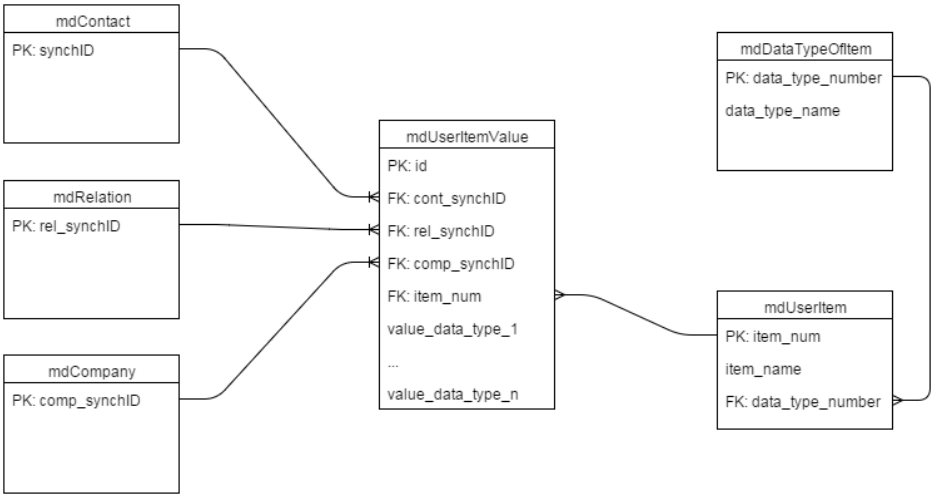
Aby se zabránilo změně struktury databáze za běhu, tím rozumíme přidávání atributů, je potřeba tabulky s uživatelskými položkami a jejich hodnotami oddělit. V existujícím modelu by to znamenalo přidat tabulku pro hodnoty uživatelských položek `mdUserItemValue`. Ta by byla propojena pomocí cizích klíčů s `mdContact`, `mdCompany` a `mdRelation`. Bylo by v ní také `n` atributů pro `n` datových typů k uchování samotných hodnot položek. Do modelu by ještě byly přidány tabulky `mdUserItem` a `mdDataTypeOfItem`. První zmíněná by uchovávala identifikátor a název uživatelské položky a druhá k ní příslušný datový typy.

Uložení záznamů v tomto modelu by muselo být ošetřeno pomocí spouště `TRIGGER`. Bylo by zkontrolováno, zda pro danou osobu, společnost nebo vztah již neexistuje hodnota uživatelské položky, do které má být zapsáno. Dále by také zaručil vyplnění pouze jednoho z cizích klíčů aby hodnota souvisela pouze s jednou tabulkou.

2.4 Předpokládané SQL dotazy systému

Zde vytvořím a popíši několik SQL dotazů, které otestuji na databázi stávajícího programu M/Data. Pro testování předpokládaných dotazů bylo vygenerováno v tabulkách `mdContact` a `mdCompany` přibližně 180 000 náhodných záznamů a v `mdRelation` přibližně 320 000 náhodných záznamů.

Pro vyhodnocení doby provádění byl dotaz změřen vždy pětkrát. Mezi jednotlivými měřeními byl proveden příkaz pro vymazání cache paměti databáze. Pokud by nebylo tak učiněno, dotazy by po opětovném vykonání databáze provedla značně rychleji a průměrný čas by byl zkreslený. Zároveň při této části analýzy byly vytvářeny a popsány vhodně zvolené indexy na atributy



Obrázek 2.5: Jiné řešení problému uživatelských položek

k urychlení jednotlivých dotazů a měřen čas vykonání s nimi. Databáze do této chvíle žádné indexy mimo primárních a cizích klíčů neobsahovala. Na následujícím výpisu je vidět přikaz na vymazání cache paměti databáze.

```
USE SerieM;
GO
CHECKPOINT;
GO
DBCC DROPCLEANBUFFERS;
GO
```

2.4.1 Dotaz 1: Počet osob

Dotaz spočte a vrátí počet záznamů v tabulce `mdContact`. Celkový čas vykonání dotazu byl 1217 ms. Tento dotaz lze zrychlit přidáním indexu na některý z atributů tabulky `mdContact`. Doba vykonání poté byla 22 ms.

```
SELECT
  COUNT(*) AS "number␣of␣items"
FROM mdContact;
```

```
number of items
---------------
177412
```

2.4.2 Dotaz 2: Seznam osob

Vybere všechny atributy z tabulky `mdContact`, které jsou v dotazu zmíněny. Celkový čas vykonání dotazu byl 779 ms. Zde by vytvoření indexu nemělo žádný vliv na rychlost dotazu.

```
SELECT
  stdFirstName,
  stdSurname,
  stdMainMail
FROM mdContact;
```

```
stdFirstName      stdSurname          stdMainMail
----------------- ------------------- --------------------------
Jan               Novák               novak@mail.cz
Věra              Stará
Jmeno             Prijmeni
23188Karel        23188Bukowski       bukowski23188@gmail.com
51373Chandler     51373Bing           chandler.bing51373@mail.us
...               ...                 ...
```

2.4.3 Dotaz 3: Minimální hodnota

Dotaz vrátí minimální hodnotu uvedenou v atributu `usr_int_1` z tabulky `mdContact`. Celkový čas vykonání dotazu byl 1269 ms. Pokud bychom v programu často hledali minimum nějaké položky, k optimalizaci by pomohlo na tento atribut v databázi vytvořit index. Celkový čas vykonání tohoto dotazu s indexem na atributu `usr_int_1` byl 24 ms.

```
SELECT
  MIN(usr_int_1) AS "min␣value␣of␣item"
FROM mdContact;
```

```
min value of item
------------------
50
```

2.4.4 Dotaz 4: Filtrace osob

Stejný jako druhý dotaz s tím rozdílem, že vybere pouze atributy za podmínky uvedené v klauzuli `LIKE`. Výsledkem jsou tedy všechny záznamy, které v atributu `stdFirstName` mají řetězec, jenž začíná podřetězcem `'Jar'`. Celkový čas vykonání dotazu byl 1157 ms. V databázích se nad často filtrovanými atributy vyplatí vytvářet index. Tento dotaz byl s indexem na atribut `stdFirstName` vykonán za 62 ms.

```
SELECT
  stdTitle,
  stdFirstName,
  stdSurname
FROM mdContact
WHERE stdFirstName LIKE 'Jar%';
```

```
stdTitle       stdFirstName          stdSurname
-------------- --------------------- -------------------------
Bc.            Jaroslav              Nový
               Jaromír               Starý
Bc.            Jarmila               Nováková
Ing.           Jarda180              Jágr180
Ing.           Jarda523              Jágr523
...            ...                   ...
```

2.4.5 Dotaz 5: Seřazená filtrace osob

Vybere stejné položky jako předchozí dotaz, ale navíc výsledek vzestupně seřadí podle atributu `stdSurname`. Celkový čas vykonání dotazu byl 1410 ms. Vhodnost indexů na filtrovaných atributech jsem zmínil výše. Stejně tak je tomu u položek, podle kterých se v dotazech často řadí. V tomto případě se dotaz díky existenci předchozího indexu a vytvoření nového na atribut

`stdSurname` vykonal za 69 ms.

```
SELECT
  stdTitle,
  stdFirstName,
  stdSurname
FROM mdContact
WHERE stdFirstName LIKE 'Jar%'
ORDER BY stdSurname;
```

stdTitle	stdFirstName	stdSurname
Ing.	Jarda10	Jágr10
	Jarda101	Jágr101
Bc .	Jarda108	Jágr108
Ing.	Jarda109	Jágr109
Ing.	Jarda110	Jágr110
...	...	...

2.4.6 Dotaz 6: Přirozené spojení

Vypíše atributy ze tří tabulek `mdContact`, `mdCompany` a `mdRelation`. Z nich spojí atributy na základě cizích klíčů a vypíše je. Celkový čas vykonání dotazu byl 5756 ms. Zde se porovnávají pouze primární a cizí klíče, které už indexovány jsou. Přidání dalších indexů by tedy na rychlost dotazu nemělo vliv.

```
SELECT
  c.stdSurname,
  r.rel_stdDepartment,
  m.comp_stdCompany
FROM mdContact AS c
JOIN mdRelation AS r
ON c.synchID = r.rel_cont_synchID
JOIN mdCompany AS m
ON r.rel_comp_synchID = m.comp_synchID;
```

```
stdSurname        rel_stdDepartment     comp_stdCompany
----------------- --------------------- ---------------------
54987Spielberg    85IT                  10000Apple
39485Petka        23IT                  10001Acer
20773Petka        14IT                  10001Acer
11538Nováková     31IT                  10004HP Computers
4641Nováková      55IT                  10004HP Computers
...               ...                   ...
```

2.4.7 Dotaz 7: Skupinový dotaz

Spojí tabulky `mdCompany` a `mdRelation`. Po tomto spojení určí počet osob u jednotlivých společností na základně agregační funkce `COUNT()` na atributu `rel_cont_synchID`. Celkový čas vykonání dotazu byl 3095 ms. Pokud by se klauzule `GROUP BY` opakovaně využívala na jeden atribut je dobré na něj použít znovu index. V tomto případě byl vytvořen na atribut `comp_stdCompany` a přinesl zrychlení na 455 ms.

```
SELECT
  c.comp_stdCompany AS 'company',
  COUNT(r.rel_cont_synchID) AS 'number␣of␣contacts'
FROM mdCompany AS c
JOIN mdRelation AS r
ON c.comp_synchID = r.rel_comp_synchID
GROUP BY(c.comp_stdCompany);
```

```
company                 number of contacts
----------------------- -------------------
65636Kadel Software     1
86792Acer               1
93740HP Computers       2
PrvniSpolecnost3267     1
29114Siemens            2
...                     ...
```

2.4.8 Dotaz 8: Skupinový dotaz s vyhledávací podmínkou

Poslední dotaz vypíše společnost popř. společnosti, které mají k sobě navázány nejvíce osob. Dotaz nejdříve spojí dvě tabulky, stejně jako předchozí. Dále v dotazu musí být použita klauzule `HAVING`. Ta nám umožňuje definování podmínky s využitím agregačních funkcí. V podmínce pak hledáme již zmiňované maximum v počtu osob s využitím poddotazu a hledání jeho maxima na základě dalšího vnořeného poddotazu v něm. Pokud se maximum poddotazu rovná počtu osob navázaných k dané společnosti, právě tato společnost má nejvíce osob. Celkový čas vykonání dotazu byl 2948 ms. Stejně jako u předchozího dotazu i zde provedení urychlil index na atribut `comp_stdCompany` uvedený v klauzuli `GROUP BY`. Dotaz se pak vykonal ještě rychleji než předchozí, a to díky malému počtu vrácených položek. Konkrétně trval 428 ms.

```
SELECT
  c.comp_stdCompany AS 'company',
  COUNT(r.rel_cont_synchID) AS 'number␣of␣contacts'
FROM mdCompany AS c
JOIN mdRelation AS r
ON c.comp_synchID = r.rel_comp_synchID
GROUP BY(c.comp_stdCompany)
HAVING COUNT(r.rel_cont_synchID) =
  (
  SELECT MAX(subquery.c) FROM
    (
    SELECT
      COUNT(rel_cont_synchID) AS c
    FROM mdRelation
    GROUP BY(rel_comp_synchID)
    )
  subquery
  )
;
```

```
company with MAX contacts        num of contacts
-------------------------------- -----------------
14190Lenovo                      3
71202Acer                        3
90384Siemens                     3
```

2.5 Shrnutí analýzy databáze

Vzhledem k faktu, že není možnost měnit databázový model stávajícího programu M/Data přidáním či odebráním některých tabulek, nabízí se možnost optimalizace pouze pomocí indexů.

Jak jsem již zmiňoval u jednotlivých testovacích dotazů, použití indexů přineslo značné zrychlení. Aktuální databáze žádné indexy neobsahuje. Díky tomuto zjištění by bylo dobré vytipovat atributy, na které by se indexy měly doplnit. V CRM systému lze předpokládat, že uživatel bude vyhledávat převážně podle příjmení osoby resp. názvu společnosti. Index by se tedy určitě měl přidat na atribut `stdSurname` z tabulky `mdContact` a atribut `comp_stdCompany` z tabulky `mdCompany`. Další indexy by mohly být doplněny na základě analýzy práce s programem M/Data přímo u konkrétního zákazníka.

Zde je pro rekapitulaci uvedena tabulka s naměřenými časy provedení jednotlivých dotazů. Nejdříve bez indexu a poté s vhodně zvoleným indexem. Jsou zde uvedeny pouze dotazy, u kterých index mělo smysl vytvořit a přinesl nějaké zlepšení.

Dotaz č.	Bez indexu	S indexem
1	1217 ms	22 ms
3	1269 ms	24 ms
4	1157 ms	62 ms
5	1410 ms	69 ms
7	3095 ms	455 ms
8	3160 ms	368 ms

Tabulka 2.2: Měření rychlosti testovaných dotazů

3 Analýza aplikační vrstvy

V této kapitole popíši analýzu stávající aplikační vrstvy programu M/Data. Aplikační vrstvu jsem obdržel jako dynamicky linkovanou knihovnu DLL[3]. Knihovnu jsem analyzoval ze dvou pohledů. Prvním byla rychlost vykonávání jednotlivých funkcí a druhým trasování dotazů funkcí na databázi. Měření funkcí bylo zjištěno z programu Logview jenž je součástí instalace programu M/Data. V tomto logu je uveden čas vykonání úvodního dotazu pro získání identifikátorů všech záznamů a zpracování tohoto dotazu do doby, než začne poddotazování na jednotlivé záznamy a jejich vykreslování v programu. Měření probíhala dvě, při 10000 a 170000 záznamech. Databázi byla mezi jednotlivými měřeními vymazána cache paměť. Trasování dotazů na databázi probíhalo v logu Microsoft SQL Server 2014.

3.1 Funkce k testování

Do testování aplikační vrstvy jsem zahrnul pouze nosné funkce knihovny, jenž pracují s největším objemem dat a při použití v desktopové verzi programu bylo při jejich použití zpozorováno značné zpomalení. Jsou to funkce pro získání objektů:

- ContactList
- CompanyList

První zmíněná funkce se používá pro získání osob a druhá pro získání společností. Pro testování byl ještě důležitý jeden z parametrů obou funkcí a to `RelationsNeeded`. Parametr datového typu boolean určuje, zda mají být osoby načteny i s příslušnými společnostmi, resp. společnosti s osobami, s nimiž mají vztah. Tyto dvě funkce použité bez vztahů i se vztahy si jednotlivě rozebereme.

[3]Dynamic-link library

3.1.1 ContactList bez použití vztahů

Vrací záznamy osob bez navázání na záznamy společností. Trasováním komunikace na databázi byl zachycen dotaz pro získání identifikátorů osob, a to následující:

```
SELECT
  synchID,
  stdName
FROM mdContact
ORDER BY stdName ASC;
```

Tento dotaz má dva defekty, jenž jsou pro rychlost vykonání zásadní. Prvním je dotazování se na atribut `stdName`. Pokud potřebuji pouze identifikátory záznamů postačí mi dotaz pouze na atribut `synchID`. Dotazování se na `stdName` zpomaluje vykonání dotazu a výrazně zvětšuje počet bytů k přenesení, a to až o pětinásobek. To by později až bude databáze na serveru stálo nějaký čas. Druhým defektem je řazení nad atributem `stdName`. V tomto dotazu je řazení zbytečné a čas vykonání tím značně stoupá.

Funkce byla vykonána při 10000 záznamech za 2285 ms, při 170000 byl čas vykonání 2703 ms.

3.1.2 ContactList s použitím vztahů

Vrací záznamy osob s příslušnými společnostmi a atributy k tomuto vztahu. Dotaz pro získání identifikátorů těchto záznamů je následující:

```
SELECT
  synchID,
  rel_synchID
FROM mdContact
LEFT JOIN mdRelation
ON mdContact.synchID = mdRelation.rel_cont_synchID
ORDER BY stdName ASC;
```

Dotaz má opět navíc řazení. Pokud potřebuji pouze identifikátory záznamů pro získání počtu a základní úkony se záznamy, nepotřebuji tyto identifikátory mít seřazeny podle jména osoby. Nezvyšuje se tím počet přenese-

ných bytů jako v předchozím případě, neboť nad jménem se pouze řadí, ale časová náročnost dotazu tím opět stoupá.

Funkce byla měřena při 170000 osobách a 200000 společnostech s existencí 10000 a 320000 vztahů. V prvním případě se funkce vykonala za 3340 ms. V případě druhém to bylo 4615 ms. Je tedy vidět, že počet vztahů má velký vliv na čas vykonání této funkce.

3.1.3 CompanyList bez použití vztahů

Vrací záznamy společností bez navázání na tabulku osob. Stejně jako u předchozí funkce jsem v databázovém logu nalezl dotaz, kterým se funkce dotazuje na identifikátory záznamů:

```
SELECT
  comp_synchID,
  comp_stdCompany
FROM mdCompany
ORDER BY comp_stdCompany ASC;
```

Dotaz obsahuje stejné dvě chyby, které jsem popisoval u funkce pro ContactList. Nepotřebuji se dotazovat na atribut `comp_stdCompany` ani nad tímto atributem řadit. Znovu se upravením dotazu sníží rychlost provedení a počet bytů k přenesení, zde asi o čtyřnásobek.

Funkce byla vykonána při 10000 záznamech za 2651 ms a při 170000 záznamech čas vykonání stoupl na 3061 ms.

3.1.4 CompanyList s použitím vztahů

Vrací záznamy včetně napojení na příslušné osoby. Dotaz pro získání takových záznamů, jenž je funkcí odeslán, má následující podobu:

```
SELECT
  comp_synchID,
  rel_synchID
FROM mdCompany
LEFT JOIN mdRelation
ON mdRelation.rel_comp_synchID = mdCompany.comp_synchID
ORDER BY comp_stdCompany ASC;
```

V dotazu opět není třeba řadit nad atributem `comp_stdCompany`. Bez řazení se tento dotaz vykoná rychleji a tím se zrychlí i celá funkce.

Čas vykonání celé funkce byl změřen při 170000 osobách a 200000 společnostech s existencí nejprve 10000 vztahů a poté 320000 vztahů. V prvním případě se celá funkce vykonala za 3563 ms a v druhém za 4657 ms. Opět je vidět, že počet vztahů mezi osobami a společnostmi zvyšuje čas vykonání dotazu na databázi.

3.1.5 Ostatní chování funkcí

Po prvotním dotazu na identifikátory záznamů se funkce doptávají na jednotlivé záznamy zvlášť poddotazy, ve kterých nebyl nalezen žádný defekt. Problémem je, že si funkce načítají rovnou všechny záznamy najednou a to společně s vykreslením zabírá současnému programu dost času.

3.2 Návrh optimalizací

Vhodné optimalizace, které by mohly vést ke zvýšení rychlosti výše zmíněných funkcí jsou dvě.

3.2.1 Optimalizace dotazů

Z dotazů pro získání identifikátorů jednotlivých záznamů by mělo být odstraněno dotazování na `stdName` a `comp_stdCompany` a řazení nad těmito atributy. Řazení značně zvyšuje čas vykonání dotazu a primárně by se záznamy neměly

vypisovat seřazené. Jméno osoby resp. společnosti také nepotřebuji. Dotaz to příliš časově neovlivní, ale počet bytů k přenesení tím značně stoupá a jak jsem již zmiňoval, v případě, kdy je databáze na serveru, to může mít značný vliv na rychlost. Základní dotaz pro získání identifikátorů by měl mít tedy následující podobu:

```
SELECT
    primary_key_attribute
FROM table;
```

3.2.2 Přednačítání záznamů

Do aplikační vrstvy by se měl implementovat počet záznamů, které by se přednačítaly. V současné době se načítají všechny, a to při jejich velkém počtu stojí hodně času. Na základě definovaného počtu záznamu k přednačítání by se načetl vždy jen tento počet, a pokud bych chtěl záznam mimo načtený interval, načetlo by se opět dalších `n` záznamů. Tento počet by bylo dobré dynamicky nastavovat. Dalo by se tím získat jen tolik záznamů, kolik potřebuji např. na jednu stránku výpisu a aplikační vrstvě by tím bylo co nejvíce ulehčeno načítaní těchto záznamů.

3.3 Shrnutí analýzy aplikační vrstvy

Navrhnuté optimalizace byly externím vedoucím implementovány a přeměřením bylo zjištěno zrychlení minimálně o 50% u obou funkcí. Pro přednačítání záznamů byla na obou objektech implementována vlastnost `PreloadSize`. Díky tomu se dá určit kolik záznamů chceme přednačíst a po překročení načteného intervalu se načte dalších `n` záznamů podle `PreloadSize`. Tím je vykreslení dat z databáze o proti původnímu stavu znatelně rychlejší.

Díky těmto optimalizacím je aplikační vrstva použitelná i pro implementaci webového rozhraní k programu M/Data. Použitím této vrstvy se zvýší také bezpečnost aplikace a dodrží se stejná architektura jako u desktopové verze.

4 Struktura, použité nástroje a implementace

V této kapitole popíši strukturu aplikace, nástroje jenž byly použity pro implementaci a jak jsem v samotné implementaci webové aplikace postupoval. Zároveň bych chtěl zmínit, že aplikace byla vyvíjena tak, aby podporovala všechny nejnovější verze webových prohlížečů Windows Explorer, Mozilla Firefox, Google Chrome a Opera. Celá aplikace byla implementována ve Visual Studiu Express 2015 for Web s využitím Microsoft .NET Framework 4.5 a existujicích knihoven desktopové verze stávající aplikace M/Data. Díky .NET Frameworku a použitým knihovnám využívá celá aplikace třívrstvé architektury.

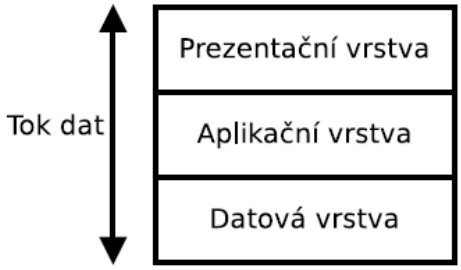
4.1 Architektura aplikace

Vzhledem k využití .NET Frameworku a externích knihoven je celá aplikace implementována v třívrstvé architektuře. Ta je v poslední době hojně využívána, a to hlavně při vývoji středních až velkých informačních systémů. Základní ideou je oddělení aplikačního kódu, uživatelského rozhraní a úložiště dat. U rozsáhlejších systémů to má výhodu při vývoji. Jednotlivé vrstvy lze implementovat odděleně, čímž se zvyšuje jak přehlednost systému, tak efektivnost vývoje.

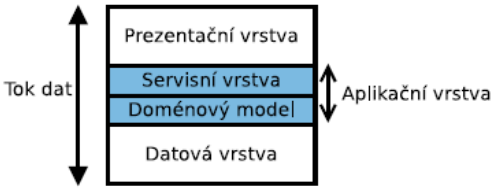
Třívrstvá architektura je obecný a nejhojněji využívaný typ vícevrstvé architektury. Rozděluje celou aplikaci do tří oddělených logických vrstev, které nemusí nutně běžet na stejných zařízeních resp. operačních systémech. Je nutné zmínit, že by každá vrstva měla komunikovat pouze se sousední vrstvou. [3] Rozdělení architektury na jednotlivé vrstvy je znázorněno na Obrázku 4.1.

Nyní popíši jednotlivé vrstvy třívrstvé architektury:

1. **Prezentační vrstva** zajišťuje chod uživatelského rozhraní (UI - User Interface) aplikace. Uživatelské rozhraní je tvořeno ovládacími prvky, které nabízejí uživatelsky přívětivou možnost vyvolání požadavků na aplikaci. Ovládací prvky se liší podle platformy, na které je tato vrstva



Obrázek 4.1: Základní schéma třívrstvé architektury.

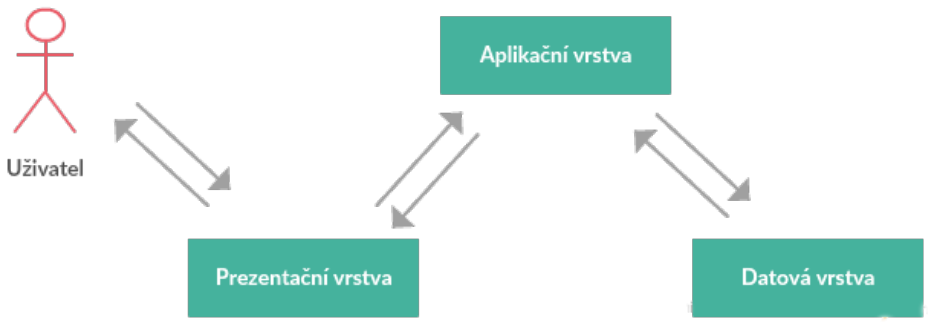


Obrázek 4.2: Rozdělení aplikační vrstvy.

implementována. Obvykle platí to, že čím bohatší ovládací prvky, tím větší je závislost na konkrétní platformě. Webové aplikace jsou většinou na ovládací prvky chudší, ale jsou zobrazitelné na všech platformách, jenž mají k dispozici webový prohlížeč. [4]

2. **Aplikační vrstva** obsahuje samotné jádro aplikace, funkce popř. metody pro výpočty, obsluhu požadavků nebo zpracování dat. Mělo by v ní být soustředěno maximum výkonného kódu a logiky celé aplikace. Existuje spoustu návrhových a architektonických vzorů uplatnitelných při implementaci této vrstvy. U webové aplikace se tato vrstva většinou ještě dělí na doménový model a servisní vrstvu. V doménovém modelu jsou definovány třídy aplikace a jejich vzájemné vazby. Servisní vrstva poté s tímto modelem manipuluje a vytváří jeho instance. Jak je vidět na Obrázku 4.2. servisní vrstva je na vyšší úrovni a vytváří tedy programové rozhraní (API - Application Programming Interface) pro prezentační vrstvu. [3]

3. **Datová vrstva** udržuje persistenci dat. Leží mimo aplikaci a je většinou obsluhována na databázovém serveru specializovaným subjektem jako je SŘBD. Řídí připojení a odpojení od databáze a zajišťuje správný tok dat pro aplikaci. [4]



Obrázek 4.3: Komunikace třívrstvé architektury.

V praxi tato architektura funguje tak, že uživatel vyvolá požadavek na aplikaci. Tento požadavek předá aplikační vrstvě parametry, díky kterým vrstva vyvolá funkci pro obsluhu daného požadavku. Ten provede nějakou změnu nebo výběr dat z datové vrstvy a data předá zpět vrstvě prezentační, která je příslušným způsobem zobrazí uživateli na výstupu. Ukázka této komunikace je na Obrázku 4.3.

4.2 Diagram případů užití

Diagram případů užití (Use Case Diagram) se používá k popisu systému z pohledu uživatele a zachycuje jaké činnosti se systémem uživatel provádí. Umožňuje znázornit funkční požadavky na budoucí systém popř. popsat funkčnost stávajícího systému pomocí interakce mezi ním a uživateli. [5] Diagram vypovídá o tom, co by měl systém umět, ale nepopisuje jak to bude dělat. [6]

Skládá se z případů užití (use cases), aktérů (actors) a vztahů mezi nimi.

Případ užití je sada několika akcí, které vedou k dosažení daného cíle. Může to být např. přidání komentáře k článku, založení nového uživatele systému nebo export dokumentu. Definuje tedy jednu určitou funkcionalitu, kterou by měl navrhovaný systém umět. Tato funkcionalita v sobě obsahuje další akce, např. přidání komentáře bude obsahovat ověření uživatele, validaci a kontrolu textu komentáře, zápis dat do databáze a podobně. To v diagramu případů užití zachyceno již nebude, občas se o tomto diagramu mluví jako o černé skříňce (blackbox), kde vnitřní logiku skryjeme a pracujeme pouze s komponentami. Případy užití vychází z požadavků na systém, hovoříme o tzv. mapování uživatelských požadavků na jednotlivé případy užití. Značíme

je jako elipsy s jejich názvem uvnitř. [6]

Aktér je role, která komunikuje s jednotlivými případy užití. V této roli může být obsazen uživatel systému nebo externí systém. Aktérem tedy může být např. Uživatel, Administrátor, SMTP server apod. Aktér inicializuje nějaký případ užití (např. "Uživatel vloží komentář pod článek"). V tomto případě je aktér aktivní a v diagramu je zapisován nalevo. Může však být i pasivní (např. externí SMTP server je iniciován případem užití "Poslat e-mail uživateli"). V tomto případě se aktér zakresluje v diagramu napravo. Aktéry znázorňujeme jako postavu s jejím názvem napsaným pod ní. [6]

Hranice systému jsou v modelu znázorněny jako rámeček s názvem modelovaného systému kolem případů užití a aktérů. Určují nám co je a co už není součástí systému. [5]

Diagram případů užití webové aplikace M/Data je zobrazen na Obrázku 4.4.

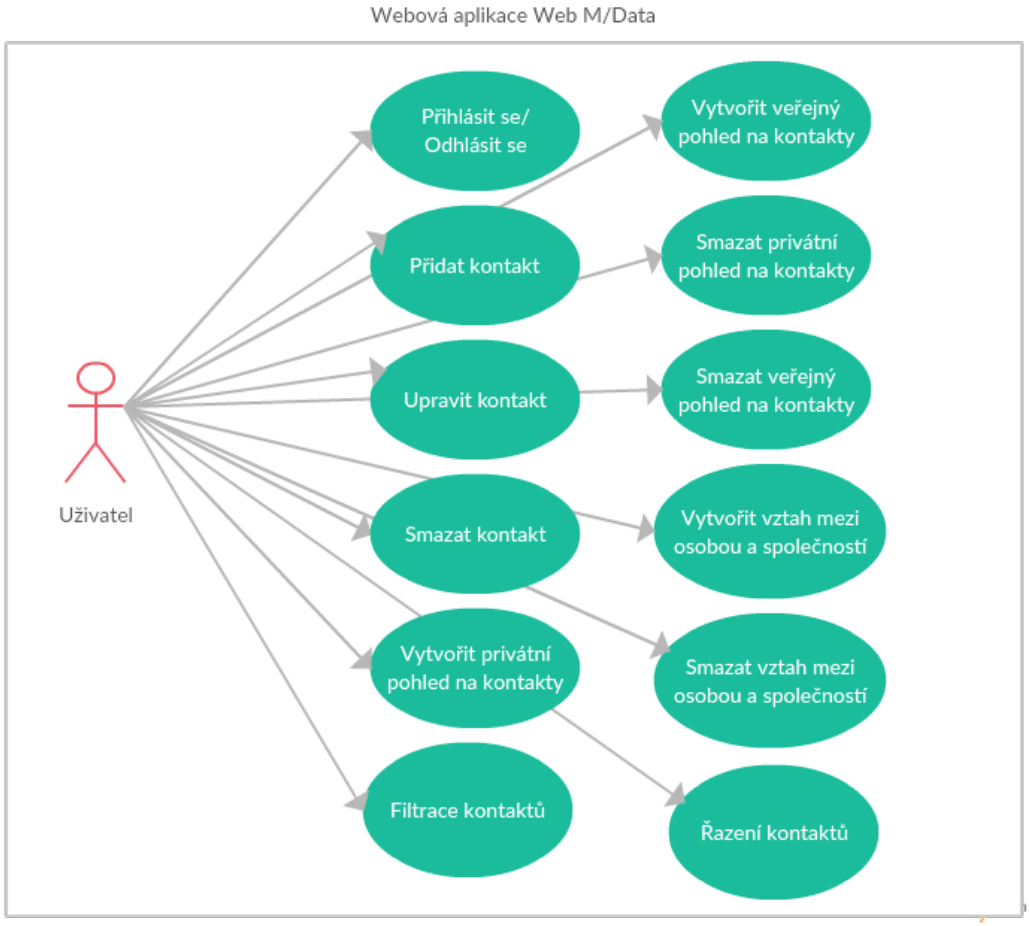
4.3 Popis použitých knihoven

K úspěšné implementaci jsem použil několik DLL knihoven poskytnutých externím zadávajícím. Knihovny využívají technologie OLE DB od Microsoftu. Tato technologie umožňuje přístup k datům z různých zdrojů jednotným způsobem. Existence těchto knihoven ve webové aplikaci přináší možnost využít stejné funkcionality jako v aplikaci desktopové. Zároveň budoucí změny vedoucí např. k optimalizaci těchto knihoven se projeví v obou verzích aplikace. Jednotlivé knihovny podrobněji popíši.

4.3.1 Knihovna M/User

Slouží k administraci uživatelů. Díky ní se dají všechny produkty externího zadávajícího propojit s jedním centrálním úložištěm uživatelů. To jsem využil také pro webovou aplikaci. Díky tomu jsem nemusel řešit správu uživatelů sám a celá aplikace tím dodržuje zavedenou architekturu produktů společnosti externího zadávajícího.

Hlavní a v této práci jedinou použitou funkcionalitou knihovny je objekt `TMUser`. Na tomto objektu jsem pak použil dvě funkce `Login()` a `Logout()`.



Obrázek 4.4: UML diagram případů užití.

Jak už vyplývá z názvů, jedná se o funkci k přihlášení a odhlášení. Funkce k přihlášení vyžaduje jako parametr jméno a heslo uživatele. Na základě pravosti údajů vrací `boolean` hodnotu. Funkce pro odhlášení pak nevyžaduje parametr žádný.

4.3.2 Knihovna M/Data

Jedná se o nosnou knihovnu celé webové aplikace. Její hlavní funkce byly analyzovány v Kapitole 3. Slouží pro získání a přidání dat do databáze, k filtrování a řazení nad těmito daty, pro získání sloupců jak základních položek, tak položek uživatelských a v neposlední řadě také získání předdefinovaných hodnot resp. skupin hodnot např. pro výběrové položky.

Hlavními objekty z této knihovny jsou `MDataContactList` a `MDataCompanyList`. Ty slouží k operacím nad záznamy z databáze. Objekty jsou získány funkcemi `GetContactListEx()` a `GetCompanyListEx()`. Ty mají jako parametry XML řetězce pro řazení a filtrování a `boolean` hodnotu zda chceme do listu zahrnout vztahy. Dále je to objekt `MDataItemList`, jenž slouží k získání všech sloupců databáze. U těchto sloupců lze získat identifikátor v podobě přesného názvu sloupce v databázi, název a datový typ. Posledním využívaným objektem je `PredefinedValueGroup`. Ten lze získat na položce z `MDataItemList` pokud na tuto položku je nějaká skupina předdefinovaných hodnot navázána. To lze na položce ověřit vlastností `HasPredefinedValues`, která vrací `boolean` hodnotu.

4.3.3 Knihovna M/Repository

Knihovna slouží k obsluze repozitáře M/Repository pro ukládání a čtení dat z předem připravených struktur složek. V této práci je repozitář využíván pro uživatelsky definované pohledy, které jsou dále popsány v implementační části.

Pracuje se zde s dvěmi objekty. Jedním je `MRepositoryFolder` reprezentující složku repozitáře a druhým `MRepositoryItem` reprezentující položku obsahující data. U složky jsou hlavními funkcemi `FolderByPath` a `FolderByName`. První zmíněnou funkcí lze získat podsložku pomocí cesty z aktuální složky, druhou pak podsložku na základě jejího názvu. Dalšími funkcemi jsou `AppendFolder` a `AppendItem`. Využívají se pro vytvoření nové složky a po-

ložky ve složce aktuální. U `MRepositoryItem` se využívá pouze funkce `Data`, která navrací její obsah.

4.4 Postup implementace

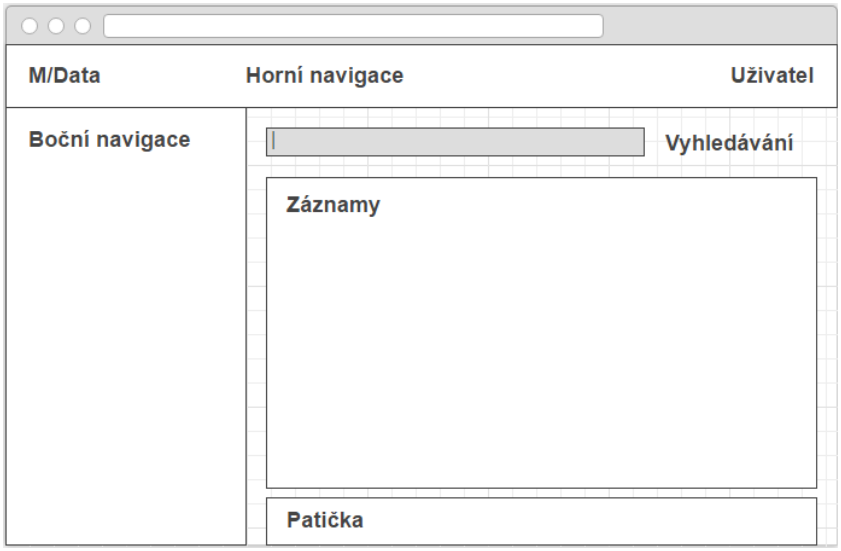
V této podkapitole dojde k popsání postupu, který jsem zvolil v průběhu implementace webového rozhraní. V každé následující vnořené kapitole je vždy jeden větší celek webového rozhraní. Tyto vnořené kapitoly jdou v přesném pořadí, v jakém byly jednotlivé části implementovány. Zároveň bych chtěl objasnit význam použití pojmů *funkce* a *metoda*. Pojem *funkce* je používán u externích knihoven napojených do aplikace a *metoda* pro mnou implementovanou funkcionalitu webové aplikace.

4.4.1 Vizuální návrh webového rozhraní

Prvním krokem k vytvoření přehledného a uživatelsky jednoduchého webového rozhraní byl vizuální návrh webové aplikace.

Nejprve jsem si navrhl rozložení jednotlivých prvků pomocí drátěného modelu. Drátěný model (Wireframe) je dvourozměrný návrh budoucí webové stránky popř. webové aplikace. Konkrétně se zaměřuje na funkci a přidělování prostoru jednotlivým prvkům. Díky němu si lze také lépe představit a pochopit funkcionalitu budoucího webu. [8] Drátěný model webové aplikace M/Data je na Obrázku 4.5. Webová aplikace se bude skládat z hlavičky, která bude obsahovat horní navigaci a jméno uživatele, boční navigace, vyhledávání, pole pro záznamy popř. jiný obsah a patičky. Horní navigace bude sloužit pro akce vztahující se k zobrazenému obsahu. Jméno uživatele bude ve formě rozbalovací navigace, kde bude mít uživatel schovány své volby. Boční navigace bude sloužit pro přechod mezi jednotlivými pohledy na záznamy. Dále bude na stránce prostor primárně vymezen pro záznamy, ale může zde být i jiný obsah. Vyhledávání bude sloužit pro filtrování nad právě zobrazenými záznamy a patička bude sloužit k zobrazení informací o společnosti externího zadávajícího.

Na základě finálního drátěného modelu jsem implementoval vizuální styl



Obrázek 4.5: Základní rozvržení jednotlivých prvků webového rozhraní.

pomocí ASP.NET komponent s využitím CSS[4] stylů. Vzhledem k faktu, že základní rozložení stránky je vždy stejné, jsem využil možnosti ASP.NET `MasterPage`. To umožňuje vytvořit základní styl stránky, do kterého je vždy dynamicky vložen obsah. Tuto funkcionalitu zaručuje komponenta `ContentPlaceHolder`, která se vytvoří na místě, kde má být vložen budoucí obsah. Obsah je poté vložen ze zdrojové stránky pomocí komponenty `Content`, která má vlastnost `ContentPlaceHolderID`, jenž určuje do kterého z `ContentPlaceHolder` na `MasterPage`, má být obsah vložen. K úspěšné implementaci vizuálního stylu jsem také využil knihoven `jQuery`. Ty slouží k jednodušší implementaci `javascript` skriptů, což jsem využil např. pro navigaci, která po kliknutí na její položku zobrazí podnavigaci, na které bude mít uživatel na výběr z několika možností. Nakonec bych chtěl ještě dodat, že celý vizuální styl aplikace se přizpůsobuje šířce okna webového prohlížeče s minimální šířkou 800 pixelů.

4.4.2 Přihlášení uživatelů

Dalším a prvním funkčním krokem implementace bylo přihlašování uživatelů. Jak jsem již zmínil v podkapitole 4.3., k implementaci přihlášení uživatelů jsem využil knihovny `M/User`.

[4]Cascading Style Sheets

Nejprve jsem začal vytvořením stránky s formulářem, jenž slouží k vyplnění jména a hesla uživatele. Je to jediná stránka, která nevyužívá existence `MasterPage` a je implementována samostatně, neboť není potřeba při přihlašování žádných jiných prvků webové aplikace. Pro obsluhu knihovny a ověřování přihlašovacích údajů byla vytvořena třída `MUserHandler`. Třída obsahuje pouze následující tři metody:

- `ExistSession()`
- `Connect()`
- `Disconnect()`

První metoda slouží ke zjištění stavu, zda je uživatel přihlášen a vrací hodnotu `boolean`. To je implementováno pomocí objektu `Session`, jenž udržuje hodnotu jména přihlášeného uživatele. Objekt `Session` je vždy jeden pro jednu komunikaci mezi aplikačním serverem a uživatelem. Pokud tento objekt pro danou komunikaci existuje, víme, že daný uživatel je přihlášen. Druhá metoda slouží k ověření přihlašovacích údajů a vytvoření zmíněného objektu `Session`. Ověření přihlašovacích údajů probíhá pomocí funkce `Login()`, která je obsažena v knihovně `M/User`. Funkce ověří údaje a vrátí `boolean` hodnotu, zda se přihlášení zdařilo či nikoli. Poslední metodou je `Disconnect()`. Jak je již z názvu patrné, slouží k odhlášení uživatele. Volá knihovní funkci `Logout()` a zruší příslušné objekty `Session`.

Po přihlášení do aplikace je uživatel přesměrován na stránku se základním pohledem na záznamy, jejichž zobrazení je popsáno dále.

4.4.3 Zobrazení záznamů

Jedním z nejhlavnějších kroků implementace bylo zobrazení záznamů o osobách resp. společnostech. Zdrojem těchto záznamů jsou funkce z knihovny `M/Data`, popsané v podkapitole 4.3.

Na začátků jsem se nejdříve musel rozhodnout, jak budou záznamy zobrazeny. Jedna možnost byla vlastní implementace komponenty, která by záznamy zobrazila. Druhou možností bylo využití ASP.NET komponenty `GridView`. Výhodou vlastní komponenty by bylo vytvoření pouze funkcí, jež by

byly využity. Nevýhodou pak časová náročnost implementace a ztráta zaručené kompatibility s ostatními komponentami ASP.NET. Proto jsem se nejdříve rozhodl důkladně analyzovat komponentu `GridView` a zjistit, zda bych jí mohl využít. Hlavním problémem bylo jak nahrát záznamy do komponenty. Nejjednodušším způsobem je určení datového zdroje komponenty pomocí vlastnosti `SqlDataSource`, `ObjectDataSource` nebo `XmlDataSource`. První možnost kvůli třívrstvé architektuře nelze realizovat, neboť z aplikační vrstvy není přímý přístup k databázové vrstvě. Druhou možností je vytvoření objektu reprezentujícího strukturu dat. To už by v rámci práce šlo použít, ale problém by nastal v načítání uživatelských položek, které v databázi v podobě sloupců přibývají. Musela by se dynamicky měnit také struktura objektu a od toho jsem ustoupil. Od využití struktury dat pomocí XML jsem také ustoupil. Knihovna `M/Data` funkci k navrácení dat v XML nemá a vytvářet si strukturu po načtení dat sám by bylo vzhledem k uživatelským položkám časově náročné. Další možností, jak určit datový zdroj `GridView` je programově pomocí jeho vlastnosti `DataSource` a objektu `DataTable`, reprezentující tabulku. Tuto tabulku lze jednoduše dynamicky vytvořit a v mém případě to byla asi nejjednodušší volba. Tabulce se nejdříve nadefinují sloupce, které bude obsahovat a poté se k ní přistupuje jako k dvourozměrnému poli, kde na první pozici je řádek tabulky a na druhé sloupec. Po této analýze následovala implementace vytvoření tabulky a napojení na `GridView`.

Nejprve jsem musel implementovat metodu, která získá data skrze knihovnu `M/Data` a poté tyto data transformovat na tabulku a přiřadit do `GridView`. Získání dat je realizováno pomocí metody `GetMDataValues` třídy `MDataHandler`, která obsluhuje zmíněnou knihovnu. Tato metoda má několik vstupních parametrů, jenž udávají identifikátor pohledu (význam popsán u Uživatelských pohledů), sloupec a typ řazení, index stránky listu se záznamy a filtrovací hodnoty v případě vyhledávání nad záznamy. Na základě těchto parametrů metoda pomocí funkce `GetContactListEx` resp. `GetCompanyListEx` získá data v podobě objektu `ContactList` resp. `CompanyList` a předá je metodě `BindContactListData` resp. `BindCompanyListData`. Ta z dat udělá již zmiňovanou tabulku. Tabulce je vygenerováno přesně tolik řádků, kolik udává vlastnost `PreloadSize` ze zmíněných objektů. To zaručí, že nemusí být přednačítán velký počet záznamů a urychlí se tím vykreslení. Tabulku už pak stačí pouze přiřadit ke komponentě `GridView` a načtení záznamů je hotové.

Dalším krokem implementace bylo stránkování nad záznamy. Vzhledem k tomu, že do komponenty `GridView` jsou vždy načteny pouze záznamy, které mají být zobrazeny, tak jsem stránkování musel implementovat zvlášť. Pro

stránkování slouží panel pod záznamy a v případě změny stránky, se pouze vygenerují záznamy k zobrazení a uloží se poslední index stránky. Stránkovat lze po jedné stránce vlevo a vpravo nebo je umožněn skok na první resp. poslední stránku.

Řazení záznamů jsem také implementoval vlastní silou. Komponenta `GridView` umí řazení nad záznamy automaticky, ale musela by mít všechny záznamy v sobě nahrané a to v mém případě nemá. Řazení se děje zavoláním funkce `GetContactListEx()` popř. `GetCompanyListEx()` s parametrem nesoucím XML řetězec, jenž je popsán dále. Tato funkce je zavolána po kliknutí na název sloupce v hlavičce `GridView`. Seřazené záznamy jsou pak vypsány již zmíněným způsobem.

Posledním krokem bylo filtrování nad záznamy neboli vyhledávání. To je realizováno v podobě textového pole nad komponentou se záznamy. Řetězec z tohoto pole je vložen do XML řetězce pro řazení, který je opět popsán dále a tento řetězec je použit jako parametr funkce pro získání záznamů. Záznamy pak zobrazíme znovu stejným způsobem. Filtruje se nad příjmením osoby nebo nad názvem společnosti, záleží který pohled na záznamy je aktuálně zobrazen.

4.4.4 Uživatelské pohledy

Další funkcionalitou je vytváření uživatelských pohledů na záznamy. To sdružuje funkce filtrování, řazení a zobrazené položky z desktopové verze programu do jedné funkcionality. Informace o těchto pohledech jsou uchovávány v repozitáři M/Repository jako XML řetězce. Popis vytváření pohledu a strukturu repozitáře včetně XML řetězců si detailněji popíšeme.

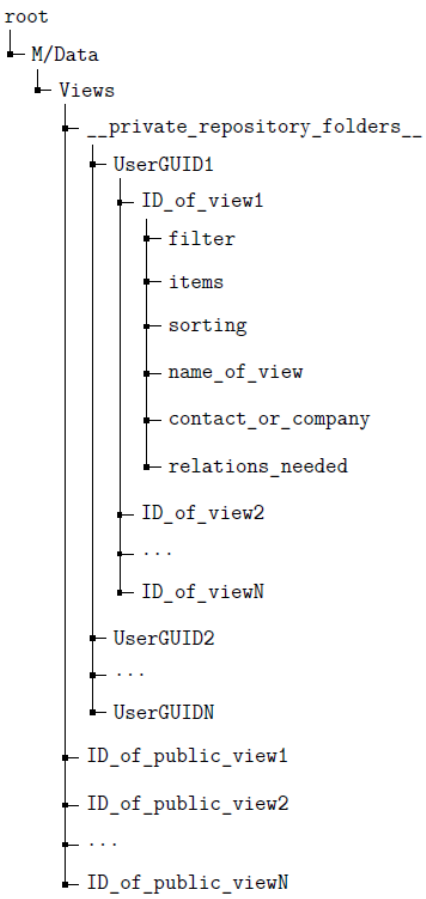
Na začátku této části implementace jsem si musel nejprve rozvrhnout, jak bude vytváření pohledu uskupené. Vzhledem k návaznosti některých vyplněných hodnot a také k jednoduchosti, jsem vytváření pohledů rozdělil na několik kroků. Nejdříve se zvolí vzorový pohled. Na základě toho se v dalších krocích načtou hodnoty zvoleného pohledu. To jednak samotné vytvoření pohledu usnadní a také poslouží např. při detailnějším filtrování nad existujícím pohledem. Dalším krokem je nastavení filtru. Ten může obsahovat až 5 filtrovacích pravidel. Při nastavení každého z nich se nejdříve vybere kategorie položek. To určuje zda se jedná o položky k osobě, společnosti nebo vztahu. Na základě této kategorie se vyplní výběrové pole položka, kde jsou na výběr jednotlivé položky. Dále se vybere operace, na základě které se bu-

dou záznamy filtrovat a jako poslední se zvolí hodnota filtru. Po nastavení pravidel se ještě zvolí jejich typ operátoru, podle kterého budou záznamy filtrovány. Buď A reprezentující `AND` popř. NEBO reprezentující `OR`. Třetím krokem vytvoření uživatelského pohledu je zvolení položek k zobrazení. Zde se pouze zvolí položky jenž mají být v pohledu zobrazeny a je zde možné vygenerovat náhled na budoucí vzhled pohledu. Na posledním kroku je nastaveno řazení, název pohledu a volba, která určuje, zda pohled bude veřejný. Řazení je možné nastavit nad jedním z vybraných sloupců k zobrazení a lze nastavit buď sestupné popř. vzestupné řazení. Pokud by uživatel chtěl sdílet tento pohled s ostatními uživateli, má možnost nastavit pohled jako veřejný. Primárně ale tato možnost byla implementována na požadavek externího zadávajícího s ohledem na budoucí existenci práv v aplikaci. Veřejný pohled by pak měl možnost vytvářet pouze administrátor popř. jiná pověřená role.

Pro uchovávání dat uživatelských pohledů jsem musel navrhnout strukturu repozitáře. Desktopová verze programu repozitář M/Repository již využívá, tudíž v něm již existuje složka M/Data. V této složce jsem si vytvořil složku `Views`. V ní se nachází složky pro veřejné pohledy, které jsou stejné pro všechny uživatele a složka `__private_repository_folders__`, ve které se nachází složky pro jednotlivé uživatele identifikované podle jejich `GUID`. V nich se vytvářejí složky s daty k jednotlivým pohledům. Celá stromová struktura repozitáře pro uchování dat pohledů je vidět na Obrázku 4.6.

Desktopová verze programu M/Data již využívá XML řetězce pro uchování dat o filtrování a řazení. S těmito strukturami také umí pracovat aplikační vrstva, tudíž jsem strukturu těchto dvou řetězců pouze převzal. Jedinou strukturu XML řetězce, kterou jsem si sám navrhl byl řetězec pro uchovávání položek pohledu.

XML řetězec pro filtrování má několik elementů. Hlavním je `FilterRoot`, který obsahuje jednotlivé úrovně filtrování `FilteringLevel`. Ty slouží k uskupení jednotlivých pravidel filtrování `FilteringRule`, ty mají několik atributů. Jsou to `ItemId` určující položku pravidla, `Condition` nesoucí číslo podle kterého se určuje podmínka pravidla a za zmínku stojí ještě `Value`, jenž nese hodnotu filtrovacího pravidla. Pokud XML řetězec obsahuje více filtrovacích úrovní, lze si to představit jako "uzávorkování"několika pravidel v SQL dotazu. Posledním elementem je `FilteringOperator`. Ten slouží k určení operátoru `AND` resp. `OR` jak mezi filtrovacími pravidly tak úrovněmi. Ukázka XML řetězce pro filtrování je vidět v následujícím výpisu.



Obrázek 4.6: Stromová struktura repozitáře.

```
<?xml version="1.0" encoding="UTF-8"?>
<FilterRoot>
  <FilteringLevel>
    <FilteringRule ItemId="stdSurname" Condition="8"
        CaseSensitive="False" Value="Ja" />
    <FilteringOperator OperatorType="0"/>
    <FilteringRule ItemId="stdMainFax" Condition="8"
        CaseSensitive="False" Value="03" />
  </FilteringLevel>
</FilterRoot>
```

Řetězec XML pro řazení obsahuje pouze dva elementy. Prvním je `SortRoot`, který obsahuje jednotlivá pravidla pro řazení `SortingRule`. Samotné

pravidlo obsahuje dva atributy. Prvním je `ItemId`, jenž udává nad kterou položkou se bude řadit. Druhým je `Direction`, ten může nabývat buď hodnoty 0 značící vzestupné řazení popř. 1 značící sestupné řazení. Ukázka XML řetězce řazení je v následujícím výpisu.

```
<?xml version="1.0" encoding="UTF-8"?>
<SortRoot>
  <SortingRule ItemId="stdSurname" Direction="0" />
</SortRoot>
```

Posledním XML řetězec slouží pro zobrazené položky. Jeho strukturu jsem si navrhl sám, neboť jeho čtení obsluhuje také pouze webová aplikace bez využití externích knihoven. Stejně jako předchozí řetězce obsahuje element `ItemsRoot`. V něm jsou obsaženy elementy `Item`, které v jejich atributech `ItemId` uchovávají název položky k zobrazení. Ukázka takto vytvořeného řetězce je v následujícím výpisu.

```
<?xml version="1.0" encoding="UTF-8"?>
<ItemsRoot>
    <Item ItemId="stdFirstName" />
    <Item ItemId="stdSurname" />
    <Item ItemId="stdMainMail" />
</ItemsRoot>
```

4.4.5 Přidání a úprava záznamů

Poslední částí implementace bylo vytvoření formuláře pro přidání a úpravu záznamů. Obě funcionality jsou si velmi podobné, tudíž je pro ně vytvořena stejná struktura formuláře, kterou detailněji popíši. Rozdíl je při úpravě, kdy se do formuláře načte již vytvořený záznam a upravují se jeho data, kdežto u přidání se vytvoří nový záznam a data se do něj z formuláře uloží.

Formulář je rozdělen do několika bloků podle typu informace, který se v nich nachází. Vzhledem k velkému počtu dat, které lze u záznamu zadávat, jsou tyto bloky skrývatelné. Po otevření formuláře je vždy zobrazen pouze blok se základními informacemi. Skrývání bloků formuláře je implementováno pomocí `javascript` skriptu s využitím knihovny `jQuery`. Formulář má několik typů vstupních polí. Je to pole textové, číselné, zaškrtávací a pole pro

datum.

U přidání záznamu se nejdříve vytvoří objekt `Contact` funkcí `CreateNewContact` resp. `Company` funkcí `CreateNewCompany`. Oba objekty mají funkci `SetValue`, pomocí které se upravují jeho data. Funkce obsahuje dva parametry, prvním se určuje jaká položka má být upravena a druhým je nová hodnota této položky. Dále mají objekty funkci `Store` a `Refresh`. Po zavolání první funkce se uloží změněná data objektu, je tedy volána při ukládání formuláře. Druhá funkce změněná data navrátí do stavu před změnami a je tedy volána při zavření formuláře bez uložení.

Posledním blokem k implementaci bylo přidání a správa položek vztahů. Tyto vztahy se získají na objektu `Contact` resp. `Company` vlastností `Relations` dle indexu daného vztahu. Jeho položky se pak získávají a editují stejným způsobem jako u osob a společností. Pro přidání vztahu jsou ve formuláři dvě pole. První textové pole určuje hodnotu filtru. Na základě něj se vyhledají osoby resp. společnosti a nahrají se do druhého pole, které je výběrové. V tomto poli si uživatel vybere osobu resp. společnost, pro kterou chce vytvořit vztah. Přídání vztahu probíhá opět na objektu `Contact` resp. `Company` vlastností `AddRelation`.

5 Testování aplikace

Před závěrečným testováním byla aplikace v použitelnosti a přehlednosti testována průběžně během schůzek s externím zadávajícím. Po dokončení implementace byla nejdříve webová aplikace otestována mnou na vygenerovaných datech, které jsem používal pro testování databáze. Poté jsem aplikaci testoval se dvěma uživateli, kteří nikdy program M/Data nepoužívali. U nich jsem zjišťoval, jak se dokáží ve webové aplikaci zorientovat a jak jim přijde uživatelsky přívětivá. U testu s těmito uživateli jsem využil testovacích scénářů. Podobné testování jsem provedl také ve společnosti externího zadavatele, kde jsem se zaměřil hlavně na zlepšení a použitelnost v porovnání se stávající desktopovou verzí programu.

Osobně jsem webovou aplikaci testoval hlavně v zatížení na datech, na kterých jsem testoval databázi a aplikační vrstvu. Zároveň jsem testoval veškerou funkcionalitu s různými vstupními daty a ladil shodnou zobrazitelnost ve všech základních webových prohlížečích. Na základě funkcionality jsem také vytvořil dva testovací scénáře, které jsem následně testoval na uživatelích a popsány jsou dále. Webová aplikace reaguje na požadavky i při velkém počtu dat v databázi poměrně rychle a zlepšení o proti desktopové verzi je znatelné. Zobrazitelnost ve všech základních prohlížečích je téměř shodná a nebyly zaznamenány žádné velké niance.

Testování s uživateli, jenž program M/Data nikdy předtím nevyužívali probíhalo podle dvou testovacích scénářů. První testovací scénář byl zaměřen na přidání a editaci záznamů a obsahoval následující kroky.

1. Přihlaste se do aplikace.
2. Přejděte na přidání nové osoby.
3. Vyplňte základní informace.
4. Přidejte vztah ke společnosti "ZČU Plzeň".
5. Uložte záznam.
6. Vyhledejte vytvořený záznam a přejděte na jeho editaci.
7. Přidejte kontaktní informace k záznamu a uložte jej.
8. Odhlaste se z aplikace.

Druhý scénář byl zaměřen na vytvoření uživatelského pohledu na záznamy. Tento scénář obsahoval následující kroky.

1. Přihlaste se do aplikace.
2. Přejděte na vytvoření nového pohledu.
3. Vyberte jako vzorový pohled "Společnosti".
4. Přidejte filtrovací pravidlo nad položkou "Společnost", která začíná řetězcem "ZČU".
5. Přijdete jako položku k zobrazení "Společnost stát".
6. Přidejte sestupné řazení nad položkou "Společnost".
7. Uložte pohled pod názvem "zču".
8. Zobrazte tento nově vytvořený pohled.
9. Odhlaste se z aplikace.

Oba uživatelé se vcelku rychle dokázali zorientovat v uživatelském rozhraní. Testování dle prvního scénáře jim nedělalo větší problémy až na pochopení k čemu slouží vztahy a jak se s nimi pracuje. Po objasnění této problematiky však dokázali vztah k osobě bez větších problémů přidat. Při druhém scénáři jsem nejdříve uživatelům vysvětlil k čemu uživatelské pohledy slouží a jakou funkcionalitu přináší. Poté se dokázali v jednotlivých krocích vytváření pohledu zorientovat a po chvíli pohled vytvořit. Oba uživatelé si pochvalovali hlavně jednoduchý a přehledný vzhled a snadné ovládání.

Uživatelé externího zadavatele si pochvalovali hlavně dostupnost pro kohokoli ze společnosti bez nutnosti instalace desktopové verze programu a vytváření uživatelských pohledů na záznamy. To jim ušetří oproti desktopové verzi programu dost času, neboť tato funkcionalita tam byla rozdělena na jednotlivé funkce. Zároveň se jim také líbil vzhled, uskupení jednotlivých prvků a jednoduché ovládání. Ve webové aplikaci se dokázali zorientovat poměrně rychle.

6 Závěr

Cílem této práce bylo analyzovat stávající strukturu databáze společně s aplikační vrstvou programu M/Data a navrhnout a implementovat vhodné uživatelské webové rozhraní pro tento program.

Z analýzy databáze a aplikační vrstvy bylo získáno několik výsledků. Některé z nich byly v této práci pouze popsány pro možné budoucí implementování. Zbylé, které bylo možno externím zadávajícím ihned do aplikační vrstvy přidat, byly implementovány a přinesly aplikační vrstvě značné zrychlení.

Po dokončení analýzy bylo navrženo a implementováno uživatelské webové rozhraní. Ze závěrečného testování vyplynulo, že webové rozhraní bylo navrženo velmi přehledně a jednoduše a splňuje požadavky externího zadávajícího, které byly na začátku práce požadovány. Velmi kladně externí vedoucí také hodnotil, že kromě vytvoření webové aplikace, vznikla dokumentace k databázi programu M/Data.

Na základě analýzy, výběru programů, implementace a testování tvrdím, že práce splňuje zadání v plném rozsahu.

Výsledný program umožňuje uchovávání informací o osobách a společnostech, vytváření vztahů mezi těmito entitami, vyhledávání nad nimi a možnost uživatelem specifikovaných pohledů na ně. Aplikace by se dala dále rozšiřovat o další funkce, které z desktopové verze programu nebyly převzaty.

Práce pro mě byla velice přínosná a to nejen z hlediska nových zkušeností s reálnou praxí v mém oboru a programováním v jazyce C#, ale i z hlediska nových zkušeností s databázovým systémem od Microsoftu, využití externích knihoven a konzultací na reálném projektu. Byla pro mě přínosná i z pohledu získání detailnějších informací s problematikou CRM systému, což v mém studijním oboru je určitě výhodou.

Literatura

[1] Entity-relationship model *searchsqlserver.techtarget.com* [online]. [cit. 2016-06-02]. Dostupné z: `http://searchsqlserver.techtarget.com/definition/entity-relationship-model`

[2] STANEK, William R. *Microsoft SQL Server 2012: kapesní rádce administrátora.* Brno: Computer Press, 2013. Microsoft (Computer Press). ISBN 978-80-251-3797-0.

[3] GEMELA, Lukáš. *Mobilní aplikace pro rezervace objektů* Plzeň, 2013. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd.

[4] Vícevrstvé architektury aplikací *robertdresler.cz* [online]. [cit. 2016-04-22]. Dostupné z: `http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html`

[5] Příklady použití diagramů UML 2.0 *uml.czweb.org* [online]. [cit. 2016-04-23]. Dostupné z: `http://uml.czweb.org/pripad_uziti.htm`

[6] UML - Use Case Diagram *itnetwork.cz* [online]. [cit. 2016-04-23]. Dostupné z: `http://www.itnetwork.cz/navrhove-vzory/uml/uml-use-case-diagram/`

[7] OLE DB *searchsqlserver.techtarget.com* [online]. [cit. 2016-06-02]. Dostupné z: `http://searchsqlserver.techtarget.com/definition/OLE-DB`

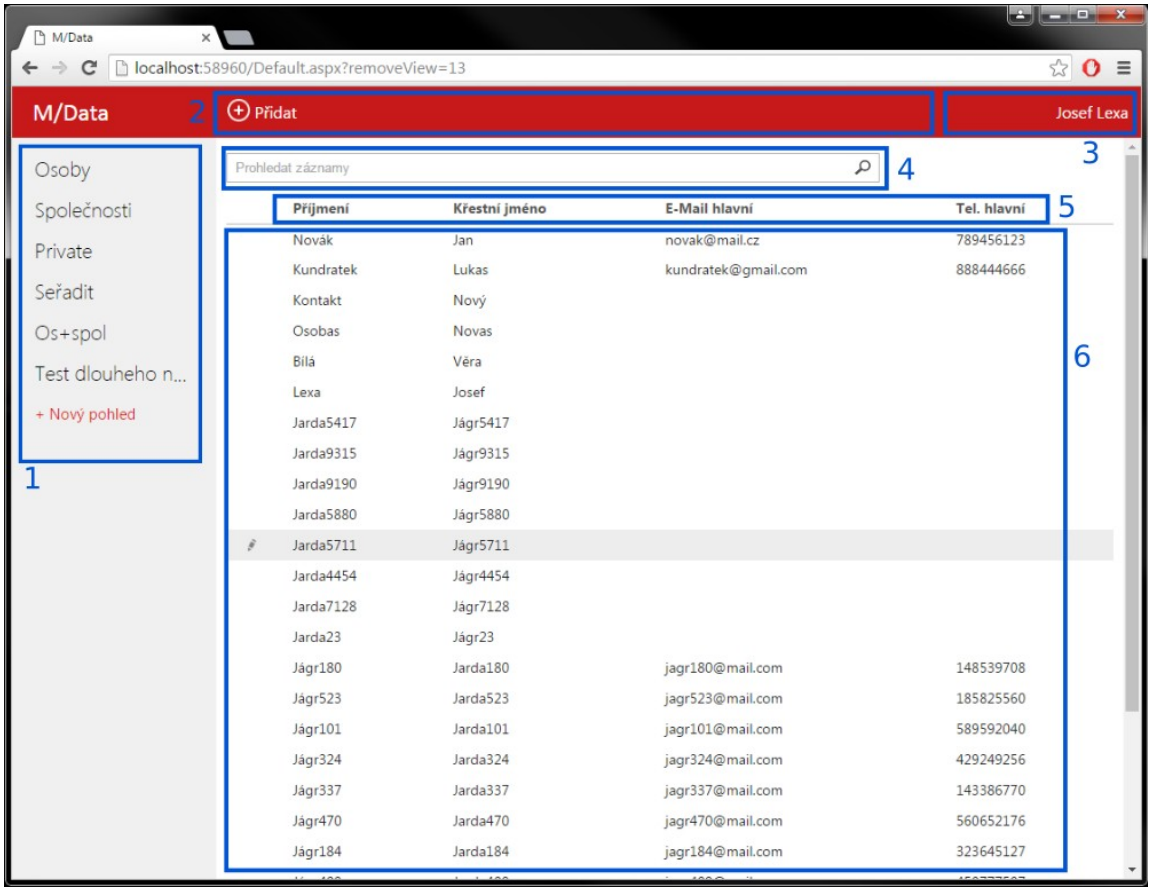
[8] Drátěný model *usability.gov* [online]. [cit. 2016-04-28]. Dostupné z: `http://www.usability.gov/how-to-and-tools/methods/wireframing.html`

Přílohy

Uživatelská příručka k vytvořené aplikaci

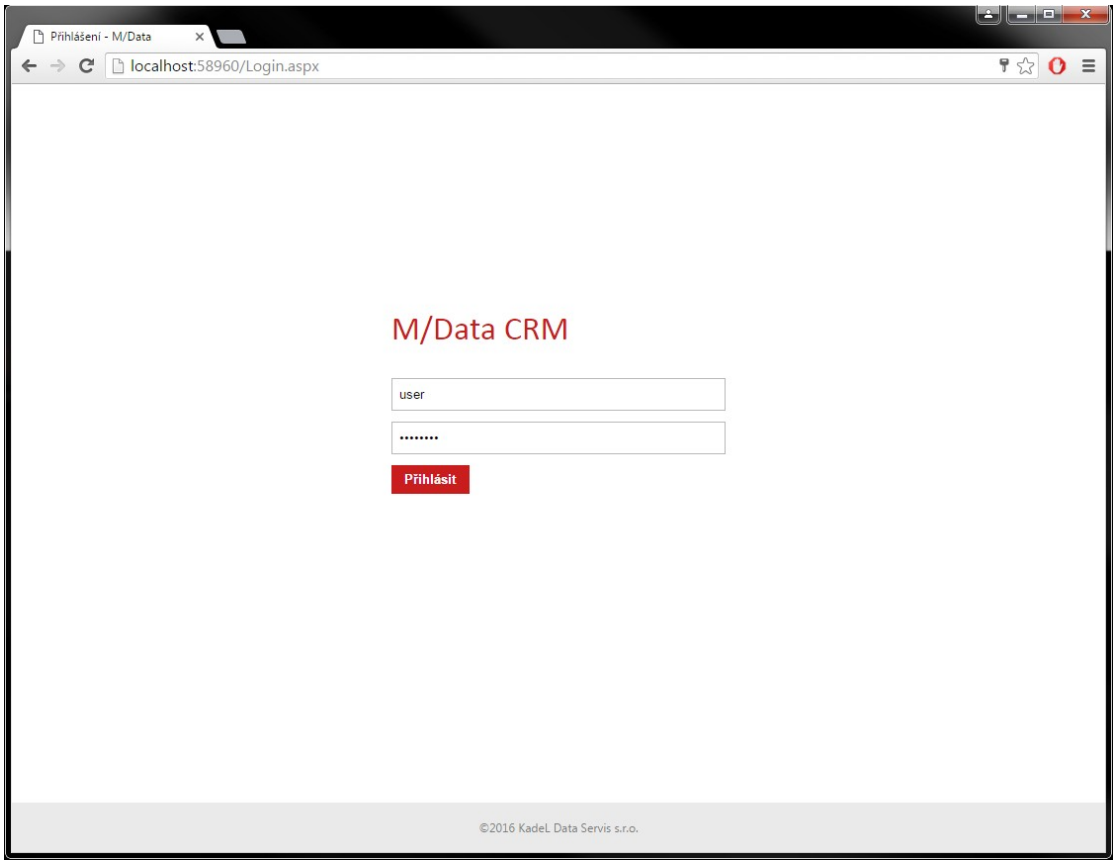
Na následujících screenshotech jsou názorně uvedeny jednotlivé funkce webové aplikace.

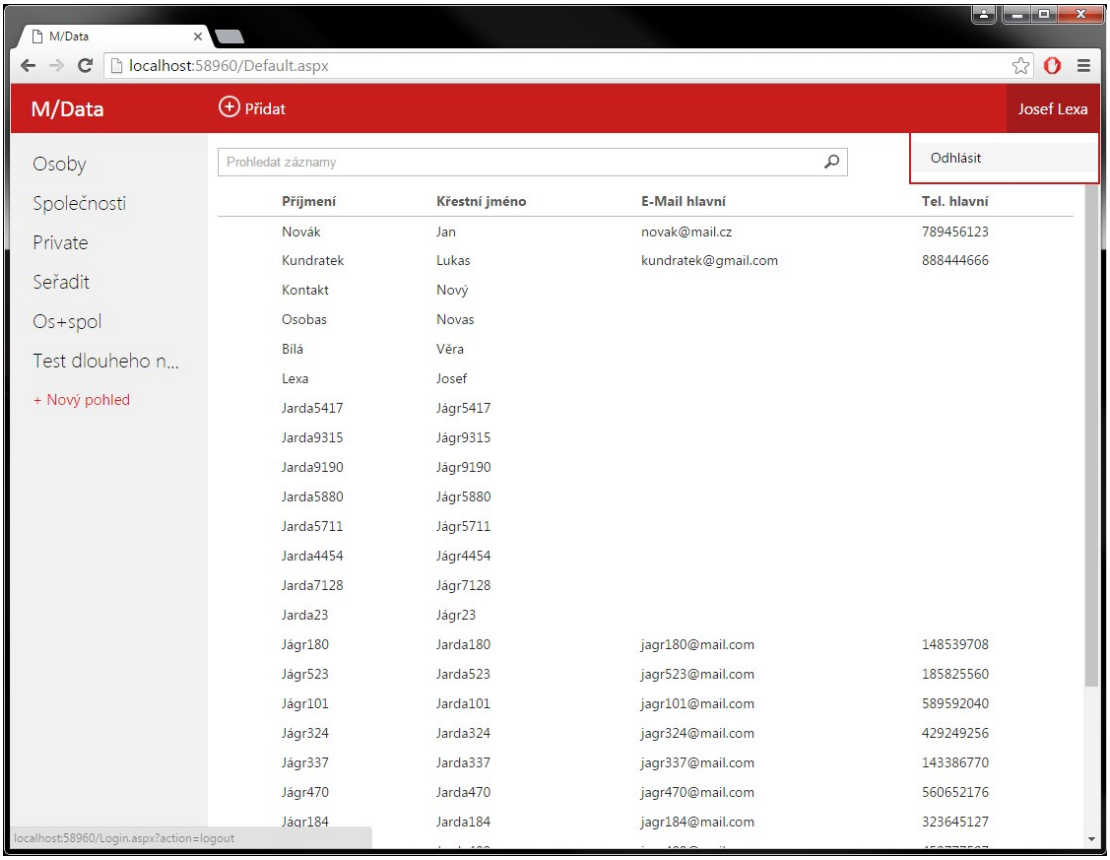
Rozvržení aplikace



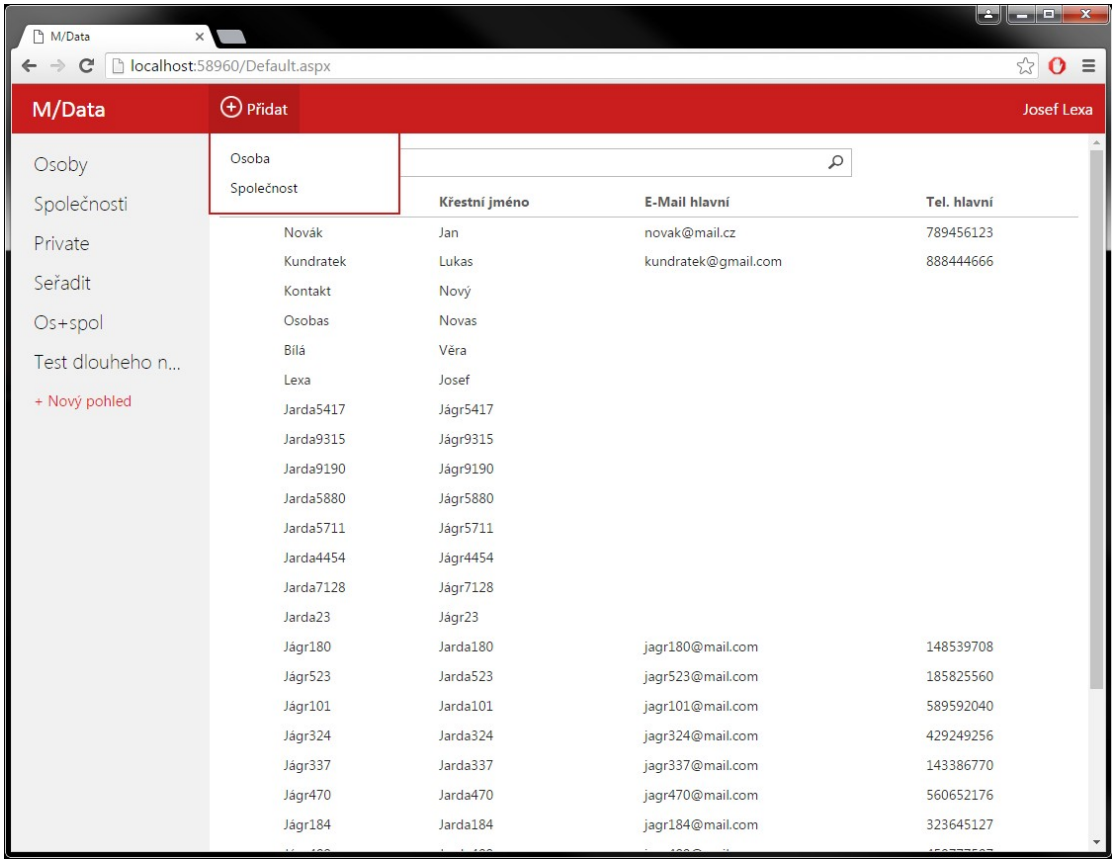
1. Boční navigace
2. Horní navigace
3. Uživatelská navigace
4. Pole pro vyhledávání
5. Jména sloupců zobrazených záznamů
6. Zobrazené záznamy

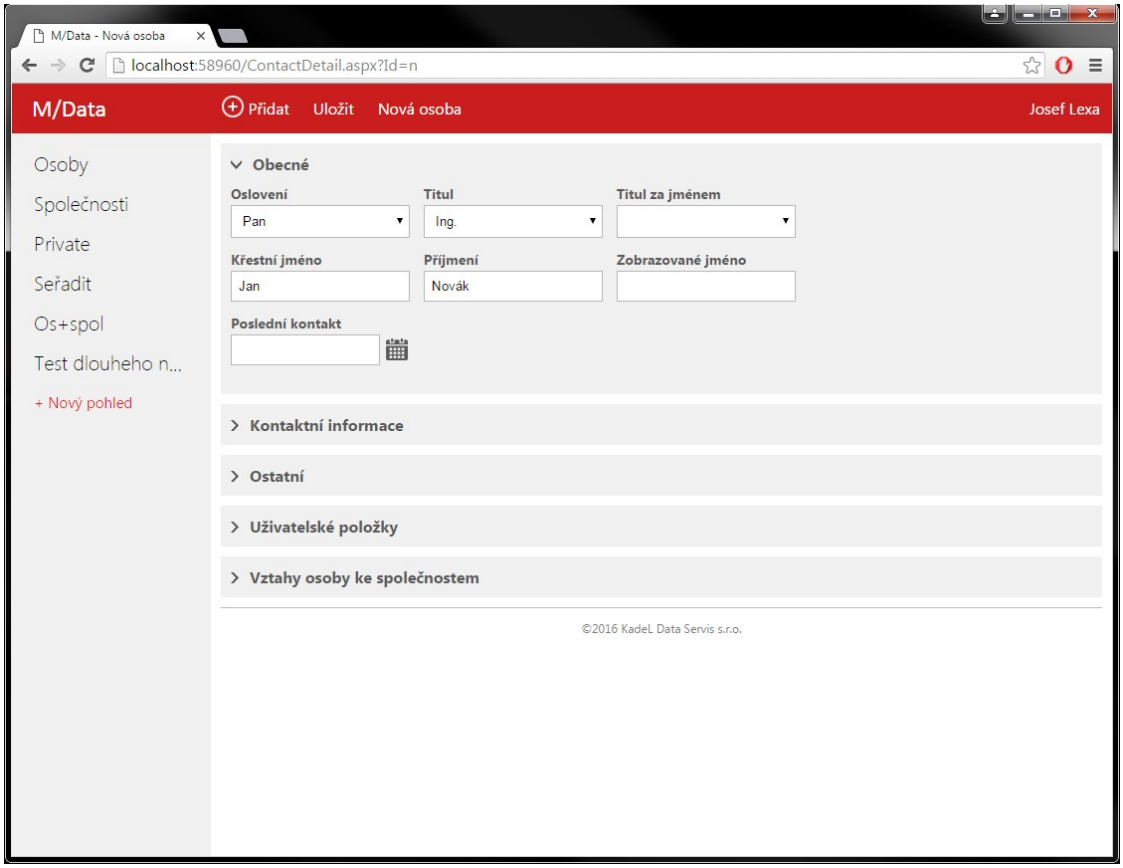
Přihlášení a odhlášení aplikace





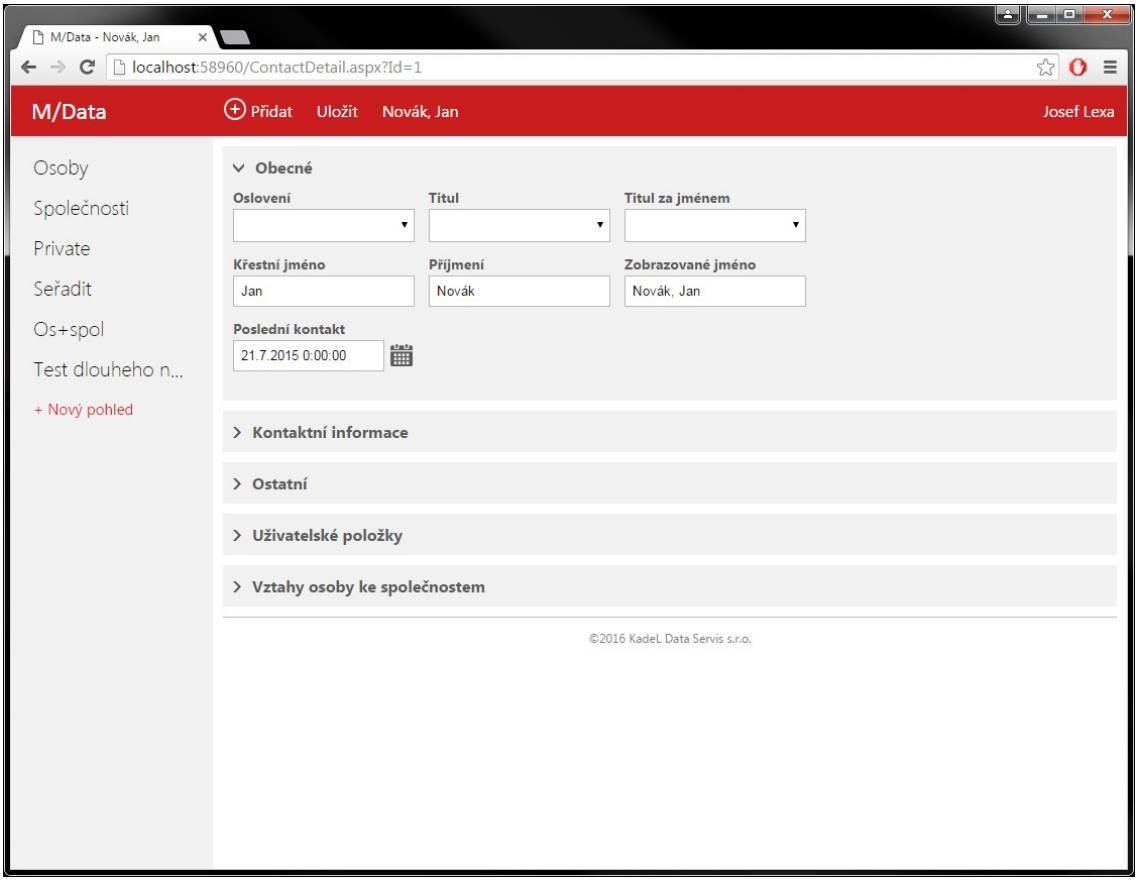
Přidání záznamu osoby a společnosti





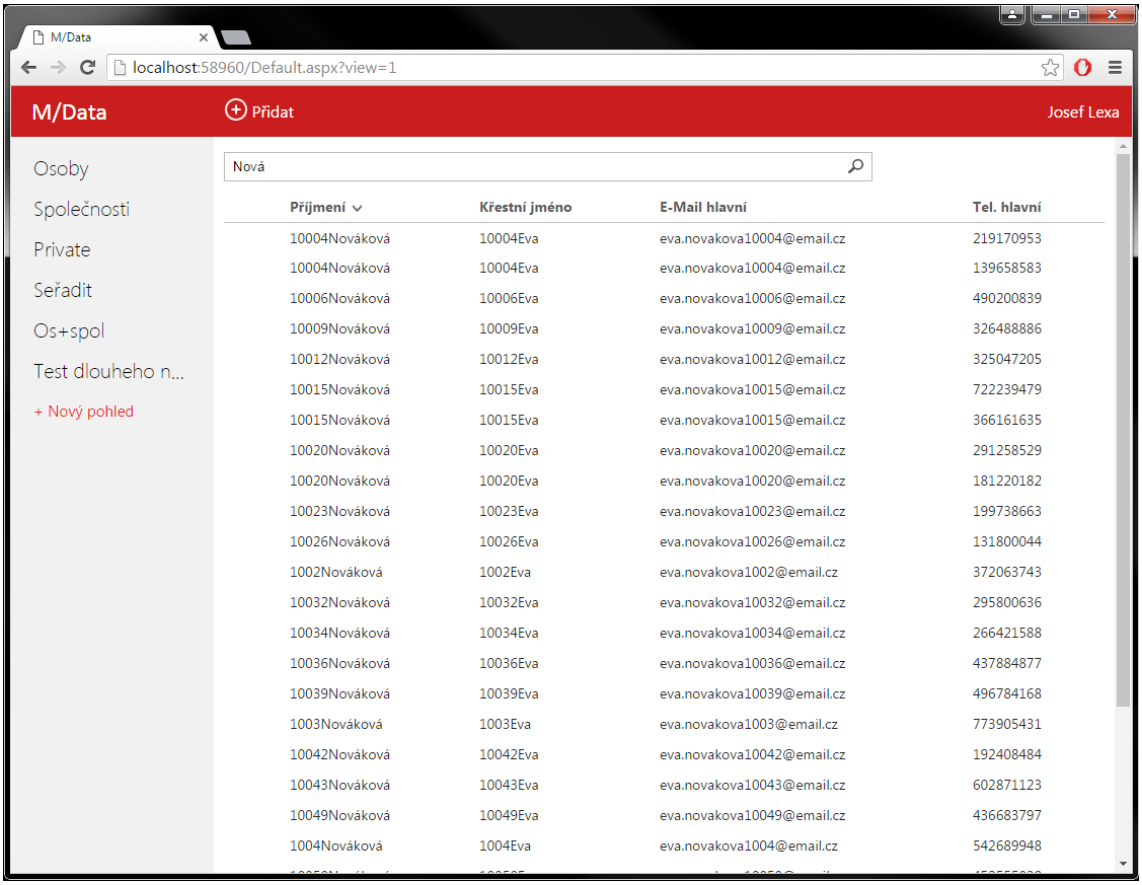
Detail záznamu

Na detail se dostaneme po kliknutí na řádek v poli se zobrazenými záznamy. Jednotlivé skupiny informací o záznamu jsou skryté v šedých blocích. Kliknutím na blok si lze informace zobrazit a vyplnit.



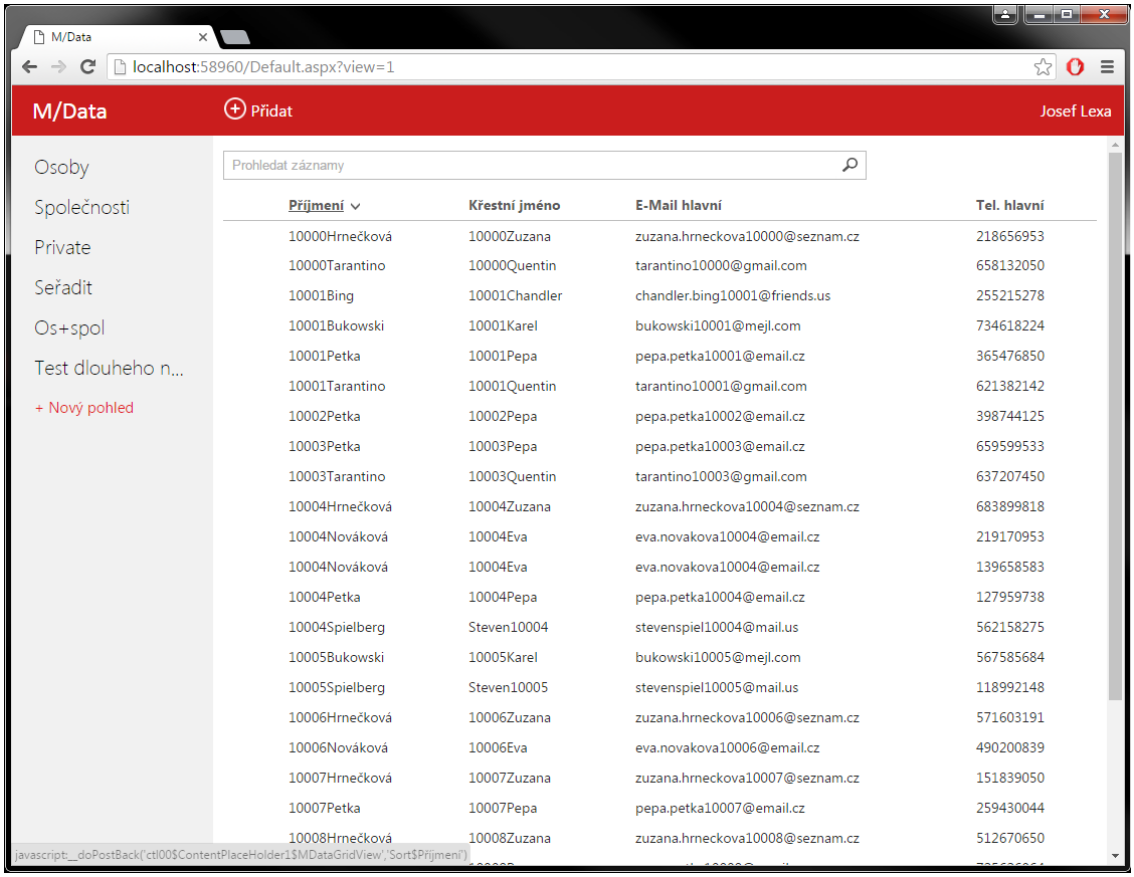
Vyhledávání

V záznamech je možno vyhledávat. Vyhledávání je realizováno textovým polem nad zobrazenými záznamy, kde se zadává řetězec, podle kterého se má vyhledat. Vyhledává se nad příjmením osoby resp. nad názvem společnosti.



Řazení

Řazení je prováděno kliknutím na název sloupce v poli zobrazených záznamů. Řazení se vždy nastavuje pouze na jeden sloupec. První kliknutí nastaví vzestupné řazení, druhé kliknutí na stejný sloupec sestupné řazení.



Vytvoření pohledu

Vytvoření pohledu je posloupnost čtyř kroků. Na každém kroku je zadána určitá informace, na základě které se poté pohled bude zobrazovat.

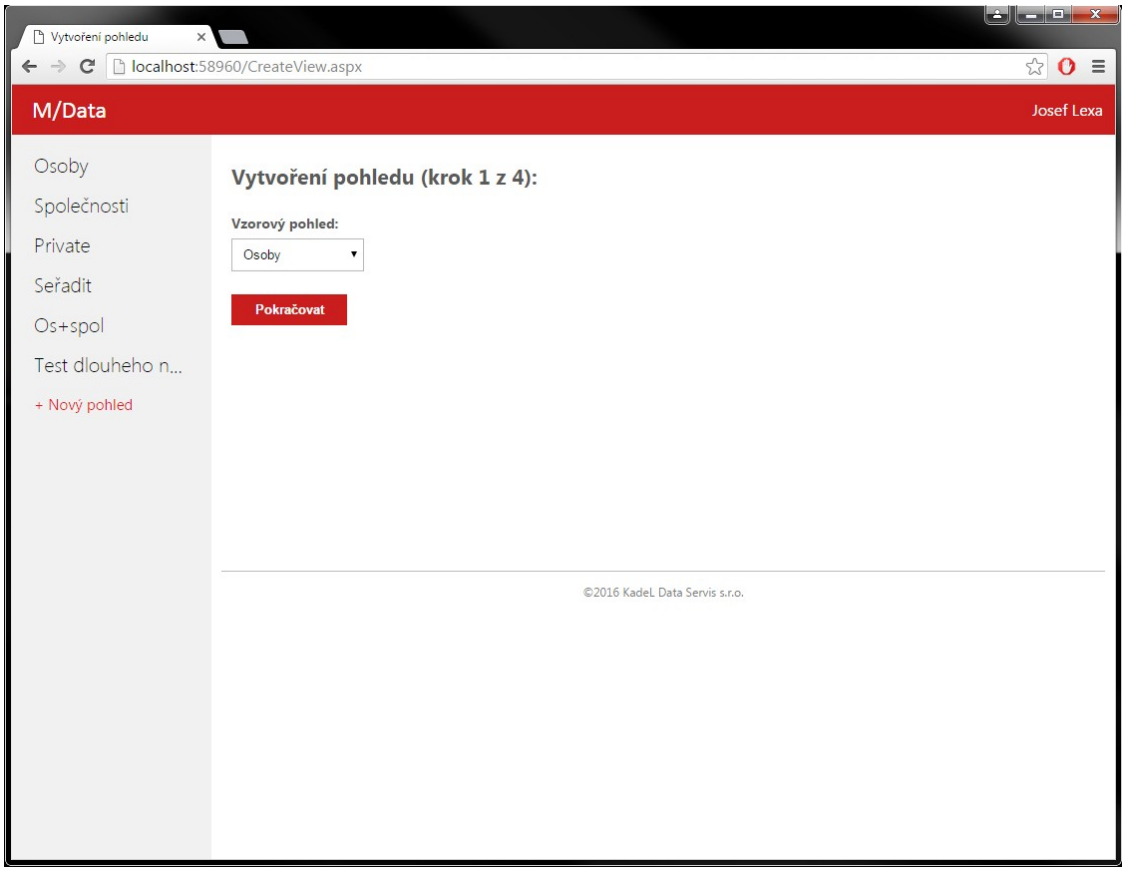
Na prvním kroku se nastavuje „Vzorový pohled“. To slouží k načtení hodnot jiného pohledu. Díky tomu je vytvoření snazší a rychlejší.

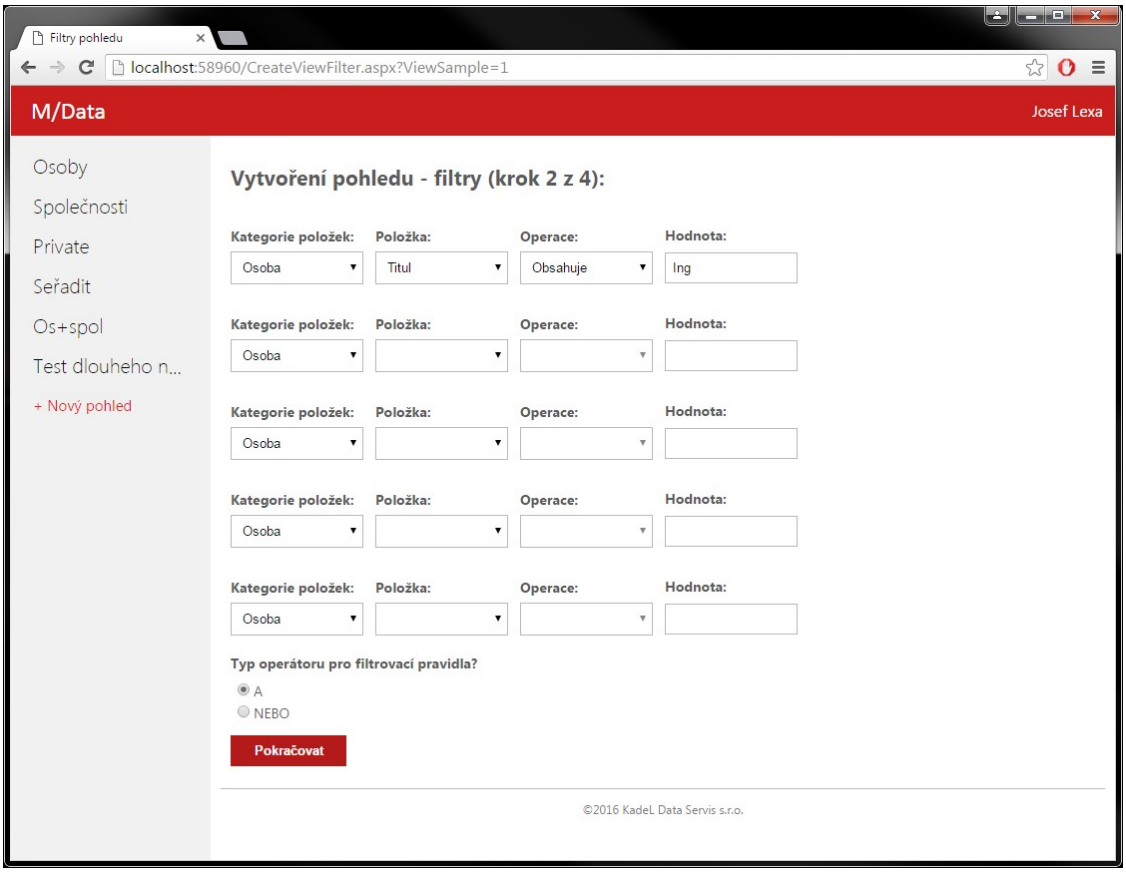
Na druhém kroku se nastavuje filtrování. Pohled může obsahovat maximálně 5 filtrovacích pravidel. U každého pravidla se nastavuje kategorie, jenž udává zda se jedná o položku osoby, společnosti nebo filtru. Dále položka nad kterou se bude filtrovat, operace filtrování a hodnota filtru.

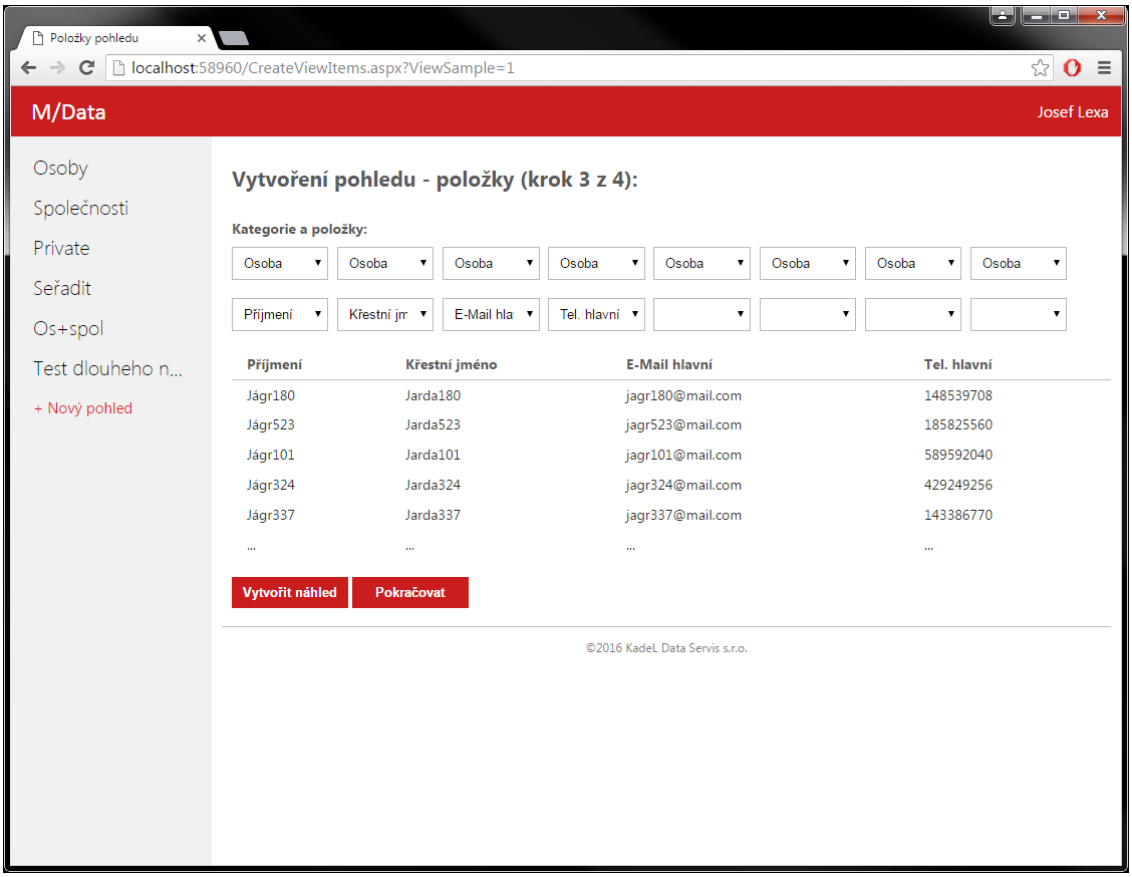
Třetí krok slouží k nastavení položek (sloupců), které bude pohled obsahovat. Zároveň je na tomto kroku možnost zobrazit náhled na budoucí pohled.

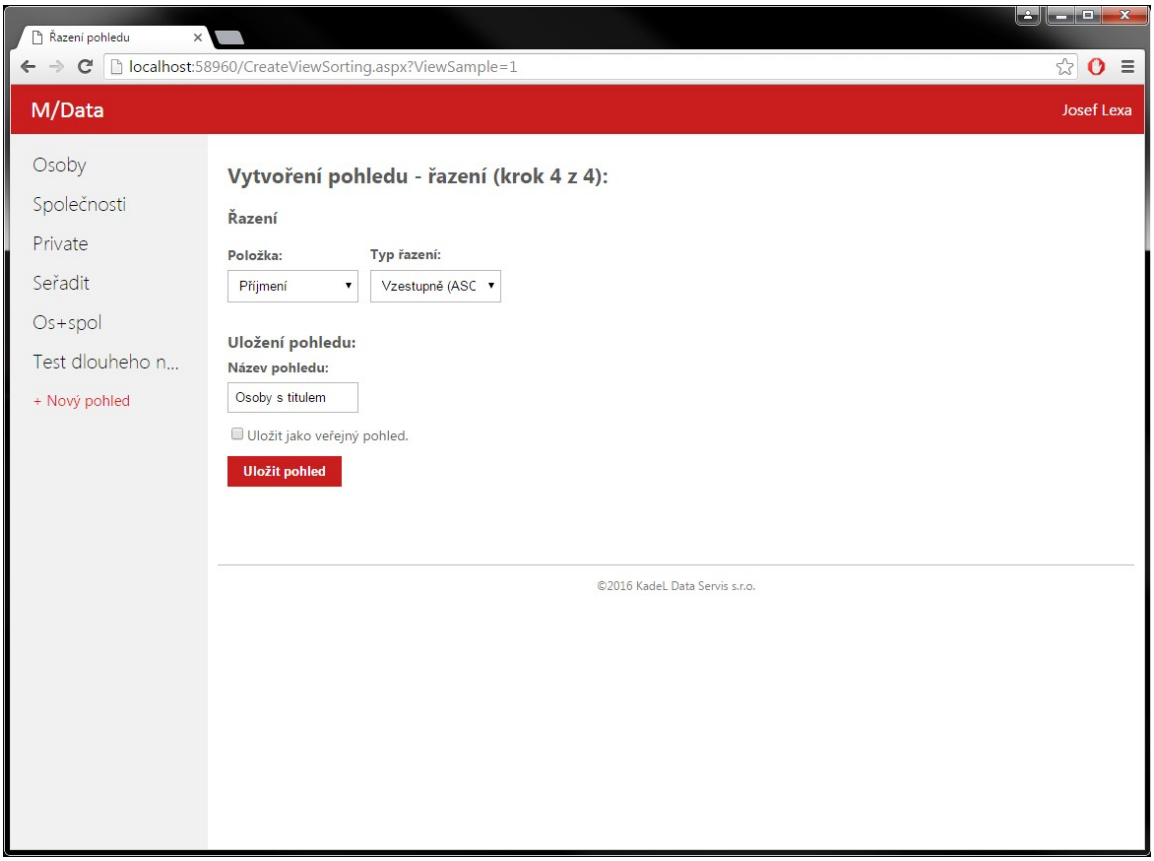
Na posledním kroku dojde k nastavení řazení pohledu, kde se vybere položka a typ řazení. Dále se zde nastaví název pohledu a zda se jedná o veřejný pohled či nikoli.

Postup vytvoření pohledu je vidět na následujících screenshotech.




Filtry pohledu
localhost:58960/CreateViewFilter.aspx?ViewSample=1
M/Data
Josef Lexa
Osoby
Společnosti
Private
Seřadit
Os+spol
Test dlouheho n...
+ Nový pohled
Vytvoření pohledu - filtry (krok 2 z 4):
Kategorie položek:
Položka:
Operace:
Hodnota:
Osoba
Titul
Obsahuje
Ing
Typ operátoru pro filtrovací pravidla?
A
NEBO
Pokračovat
©2016 KadeL Data Servis s.r.o.


Položky pohledu
localhost:58960/CreateViewItems.aspx?ViewSample=1
M/Data
Josef Lexa
Osoby
Společnosti
Private
Seřadit
Os+spol
Test dlouheho n...
+ Nový pohled
Vytvoření pohledu - položky (krok 3 z 4):
Kategorie a položky:
Osoba
Příjmení
Křestní jm
E-Mail hla
Tel. hlavní
Příjmení
Křestní jméno
E-Mail hlavní
Tel. hlavní
Jágr180
Jarda180
jagr180@mail.com
148539708
Jágr523
Jarda523
jagr523@mail.com
185825560
Jágr101
Jarda101
jagr101@mail.com
589592040
Jágr324
Jarda324
jagr324@mail.com
429249256
Jágr337
Jarda337
jagr337@mail.com
143386770
Vytvořit náhled
Pokračovat
©2016 KadeL Data Servis s.r.o.


Řazení pohledu
localhost:58960/CreateViewSorting.aspx?ViewSample=1
M/Data
Josef Lexa
Osoby
Společnosti
Private
Seřadit
Os+spol
Test dlouheho n...
+ Nový pohled
Vytvoření pohledu - řazení (krok 4 z 4):
Řazení
Položka:
Příjmení
Typ řazení:
Vzestupně (ASC
Uložení pohledu:
Název pohledu:
Osoby s titulem
Uložit jako veřejný pohled.
Uložit pohled
©2016 KadeL Data Servis s.r.o.