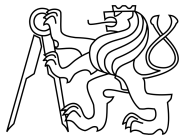
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh a implementace software pro interaktivní práci s objekty MBI pomocí jazyka MBIQL
Student:	Bc. Vojtěch Stránský
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat software umožňující interaktivně procházet a dotazovat objekty MBI pomocí jazyka MBIQL. MBI (Management of Business Information) je otevřená sada metodických pokynů a postupů ohledně strategického rozhodování, plánování i provozního řízení informatického oddělení firmy. Má i svou implementaci - mbi.vse.cz. Návrh jazyka MBIQL vznikl v bakalářské práci Ondřeje Batíka obhájené na FIT v létě 2015.

1. Nastudujte návrh jazyka MBIQL z bakalářské práce O. Batíka [1].
2. Proveďte případnou revizi návrhu.
3. Navrhněte architekturu software pro interaktivní práci s objekty MBI pomocí MBIQL.
4. Zvolte vhodnou implementační platformu.
5. Implementujte funkční prototyp.
6. Podrobte jej uživatelským testům, zhodnoťte výsledky.
7. Proveďte úpravy vycházející z vyhodnocení uživatelských testů a systém zdokumentujte a nasaďte.

Seznam odborné literatury

[1] O. Batík: Dotazovací jazyk pro vztahy MBI objektů. Bakalářská práce oboru ISM na FIT ČVUT. 2015.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 26. října 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Návrh a implementace software pro interaktivní práci s objekty MBI pomocí jazyka MBIQL

Bc. Vojtěch Stránský

Vedoucí práce: Ing. Michal Valenta, Ph.D.

20. června 2017

Poděkování

Rád bych chtěl poděkovat svému vedoucímu diplomové práce za vstřícný přístup a poskytnuté konzultace. Velký dík také patří mé rodině za podporu během studia, ale i při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 20. června 2017 .....

Odkaz na tuto práci

Stránský, Vojtěch. *Návrh a implementace software pro interaktivní práci s objekty MBI pomocí jazyka MBIQL.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat software, který umožní interaktivní procházení a dotazování objektů MBI (Management Byznys Informatiky) pomocí jazyka MBIQL.

Na základě analýzy a návrhu je vytvořena webová aplikace využívající javascriptový framework AngularJS, knihovnu D3.js a jako uložiště dat grafovou databázi Neo4j. Pomocí těchto technologií je v aplikaci naimplementován grafický dotazovací jazyk MBIQL, který je překládán do jazyka databáze Neo4j. Výsledky dotazu jsou zobrazeny knihovnou D3 v podobě grafu.

Výsledkem diplomové práce je tedy softwarový systém, který využívá jazyka MBIQL a umožňuje uživateli nejenom dotazovaní a procházení objektů MBI, ale i ukládání a třídění vzniklých dotazů. Uživateli je nabídnuta alternativní možnost pro získávání informací z MBI a to interaktivním způsobem pomocí nového dotazovacího jazyka a databáze Neo4j.

Klíčová slova MBIQL, Neo4j, javascript, dotazovací jazyk, D3.js, AngularJS

Abstract

The aim of this Master's thesis is to design and implement a software that enables interactive browsing and querying of MBI (Management of Business Informatics) objects.

Based on an analysis and a design, a web application is created using the AngularJS javascript framework, the D3.js library and the Neo4j graph database which is used as the data storage. Using these technologies the MBIQL graph query language, which is translated into the Neo4j database language, is implemented into the application. The query results are displayed by the D3 library in the form of a graph.

The result of this thesis is a software system that uses the MBIQL language and allows the user not only to query and browse MBI objects but also to store and sort the generated queries. The user is offered an alternative option to retrieve information from MBI, which is an interactive method that uses the new query language and the Neo4j database.

Keywords MBIQL, Neo4j, javascript, query language, D3.js, AngularJS

Obsah

Seznam obrázků

Úvod

Motivace

V dnešní době existuje velké množství materiálů a informací o řízení informačních technologií v podnicích. Tyto informace se mění a je potřeba vzniklé materiály inovovat. MBI (Management Byznys Informatiky) je portál, který tyto informace poskytuje a navzájem je mezi sebou propojuje. Na základě změn pak provádí aktualizaci. V portálu tudíž vzniká mnoho vazeb, na které je možné pohlížet jako na síť hustě propojených objektů. Zde lze vidět inspiraci pro použití grafové databáze pro vizualizaci právě těchto objektů.

Možnost propojit portál s grafovou databází byla popsána a pomocí konceptu dokázána v práci „Vizualizace vztahů v informační bázi MBI“ [2], ve které byly objekty nahrány do databáze Neo4j a uživateli poskytnuto rozhraní pro dotazování pomocí jazyka Cypher.

Tento způsob dotazování nebyl příliš vhodný pro netechnicky založené uživatele, tudíž byla vytvořena práce „Dotazovací jazyk pro vztahy MBI objektů“ [3], ve které byl navržen nový dotazovací jazyk MBIQL, který je zaměřen právě na tyto uživatele.

Tato práce je zaměřena na implementaci jazyka a má sloužit jako první prototyp pro náhled na data pomocí grafického dotazovacího jazyka MBIQL.

Struktura práce

Práce se skládá z následujících kapitol.

První kapitola (viz 1) seznamuje s informační bází MBI a shrnuje bakalářskou práci „Dotazovací jazyk pro vztahy MBI objektů“. Uvádí také formální požadavky na vyvíjenou aplikaci a detailní specifikaci cíle. Dále také zmiňuje použitou metodiku vývoje.

Ve druhé kapitole (viz 2) je provedena analýza a uveden návrh vyvíjeného systému. Je tedy popsán způsob importu dat do databáze, ale i architektura

a grafický vzhled celého softwarového systému. Tento systém je v rámci této kapitoly také rozdělen na menší části, které budou postupně implementovány.

Třetí kapitola (viz 3) se zabývá použitými technologiemi a důvodem jejich zvolení. Popisuje tedy javascriptový framework, grafovou databázi a vizualizační knihovnu. Uvadí také jiné - alternativní možnosti a příklady jejich použití.

Ve čtvrté kapitole (viz 4) je uskutečněna samotná realizace softwarového systému na základě předchozího návrhu a zvolených technologií. Nejprve je implementován import dat do databáze a potom jednotlivé dílčí části systému.

V páté kapitole (viz 5) probíhá ověření správnosti realizace. Systém je tedy podroben různým testům, kterými je otestován nejen z pohledu správnosti zobrazovaných dat, ale i zhlediska uživatelské použitelnosti.

V šesté kapitole (viz 6) je provedeno zhodnocení systému.

Kapitola 1

Rozbor zadání

V následující kapitole je popsáno současné řešení prezentace dat MBI a přednesen návrh nového řešení dle výsledků bakalářské práce pana O. Batíka. Z tohoto bodu také vyplyne bližší specifikace cíle této diplomové práce.

1.1 MBI

„MBI (Management Byznys Informatiky) je portál obsahující zobecněná řešení v řízení provozu a rozvoje IT, resp. podnikové informatiky. Jeho smyslem je sdílet a využívat znalosti a doporučení vyplývající z praxe a z dalších zdrojů.“ [1]

Jedná se tedy o souhrn principů a postupů týkajících se podnikové informatiky, které jsou prezentovány pomocí hierarchicky uspořádaných objektů propojených vazbami. Uživatelům jsou tyto informace zpřístupněny skrze webovou stránku MBI [1] nebo přes nově vytvořené datové uložiště v grafové databázi Neo4j, popsané v práci „Vizualizace vztahů v informační bázi MBI“ [2].

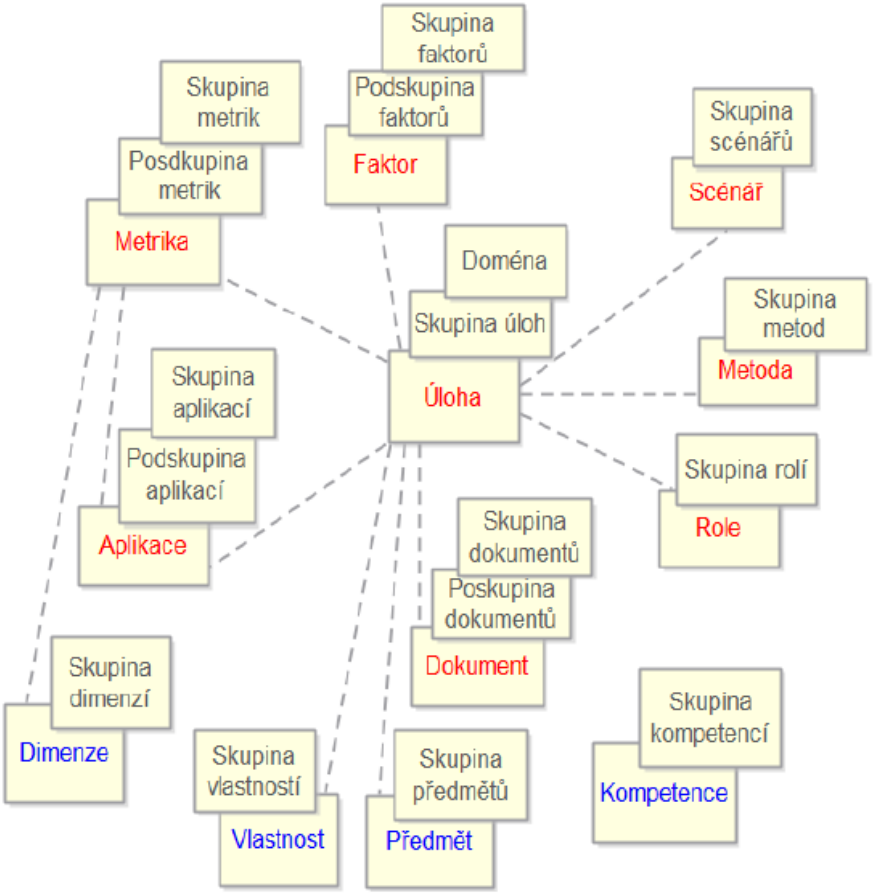
1.2 Současný stav problematiky

Současný stav byl zanalyzován v bakalářské práci „Dotazovací jazyk pro vztahy MBI objektů“ [3]. Nasledující text tedy přibližuje hlavní důvody a motivaci vzniku této diplomové práce a vychází z již zmiňované bakalářské práce. Nejprve je provedeno shrnutí uživatelů a modelu MBI, následně pak popsány dotazovací způsoby a v závěru uveden výsledek celé analýzy.

1.2.1 Analýza uživatelů a vstupních objektů

Uživatelé, kteří přistupují do informačního portálu MBI, můžeme rozdělit na tři skupiny: Firemní zaměstnanci (výjma manažerů), potencionální manažeři a

[1] `http://mbi.vse.cz/`



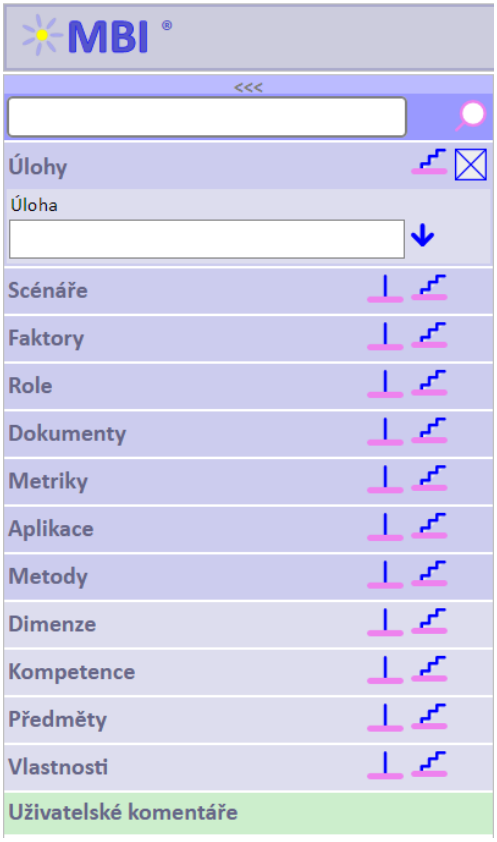
Obrázek 1.1: Struktura objektů a jejich vazeb [3]

ostatní uživatelé se zájmem o daný obor. Dotazy, které dané skupiny mají, jsou směřovány na oblast podnikové informatiky. Uživatelé tyto informace získávají skrze objekty, které jsou propojeny vazbami. Tyto vazby představují závislosti jednotlivých objektů mezi sebou. Hierarchickou strukturu celého modelu lze vidět na obrázku 1.1. Už z obrázku je možné vidět, že nejčastější dotazy budou souviset s objektem Úloha, jelikož je ve středu celého modelu a patří i mezi dva základní objekty (dále Scénář), z nichž je doporučováno začít procházení [4]. Další možností je zvolit si libovolný další objekt.

1.2.2 Dotazovací způsoby

Dotazování na objekty je umožněno skrze menu v levé části portálu [2] zobrazené na obrázku 1.2. Uživatelé mají několik možností, jak přes toto menu

[2] `http://mbi.vse.cz/mbi/index.html`



Obrázek 1.2: Menu pro dotazování v portálu MBI

přistupovat k již zmíněným objektům:

- **Dotazování přes objekt** - Přes grafické prvky přistoupíme rovnou k objektu.
- **Dotazování přes hierarchie** - Ke konkrétnímu objektu se přistupuje s využitím hierarchie. To znamená, že se nejprve vybere Skupina, do které objekt patří, následně pak Podskupina a nakonec samotná instance objektu.
- **Dotazování přes vyhledávání** - V aplikaci je umožněno fulltextové vyhledávání, kdy aplikace vrátí uživateli objekty, ve kterých se zadaný řetězec vyskytuje.

Po nalezení požadovaného objektu jsou uživateli zobrazeny detailnější informace. Je také možné se prokliknout do dalších (souvisejích) objektů.

1.2.3 Shrnutí

V uvedeném způsobu procházení modelu byly nalezeny následující výhody a nedostatky. Ty se budou v pozdějších sekcích dále řešit.

1.2.3.1 Výhody

Jednou z výhod portálu je možnost dotazování skrze menu, které bylo zmíněno výše (viz obrázek 1.2). Další předností je to, že vyhledaný objekt poskytuje velké množství dat, se kterými uživatel může dále pracovat (tisk, vytvoření reportu či přidání poznámek).

1.2.3.2 Nedostatky

Nedostatky jsou shrnuty do dvou navzájem souvisejících bodů:

- Problém nepřehlednosti
- Problém matic

Tyto body znamenají, že uživatel může mít v některém z kroků dotazování/procházení modelem problém zorientovat se. Především se to týká matic, které se dají zobrazit při rozkliknutí detailu nějakého z objektů. Matice znázorňují vazby mezi objekty, kdy při větším počtu souvisejících objektů a Skupin nabývají velkých rozměrů.

1.2.3.3 Zhodnocení

Na základě předchozí analýzy, která je detailněji rozepsána v bakalářské práci pana Batíka [3] je vyvozen následující závěr:

Samotný portál MBI není potřeba upravovat. Avšak pro zmíněné nedostatky bude vhodnější vytvořit samostatnou aplikaci, která by měla vzhledem k níže uvedeným požadavkům 1.4 řešit problém matic a současně s tím i problém nepřehlednosti.

1.3 Řešení pomocí MBIQL

Dotazovací jazyk MBIQL byl navržen na základě předešlé analýzy a detailně je popsán v práci pana Batíka [3]. V následujích kapitolách budou nejdůležitější body návrhu jazyka i samotné aplikace shrnuty.

1.3.1 Neo4j a jazyk Cypher

Grafové databáze mají obecně velkou vizualizační sílu a vzhledem ke struktuře modelu by měly řešit nedostatky portálu MBI (viz 1.2.3.2).

Grafová databáze Neo4j byla použita pro uložiště modelu MBI již v předešlé bakalářské práci zmiňované v kapitole 1.1.Pro novou aplikaci bude ale potřeba stávající schéma upravit.

Dotazovací jazyk Cypher je nástroj, kterým je možné se dotazovat v databázi Neo4j. Z tohoto důvodu bude využit i pro nově vznikající aplikaci. Uživateli však nebude sloužit jako primární nástroj pro dotazovaní - pouze se do něho bude překládat jazyk MBIQL popsaný níže.

1.3.2 MBIQL

1.3.2.1 Definice

"MBIQL (Management of Business Informatics Query Language) je graficky orientovaný a doménově specifický dotazovací jazyk, který je určen pro dotazování se na data MBI. Dotaz jazyka MBIQL je vytvořen pomocí jedné ze dvou metod: Specificky orientované dotazování a Dotazování pomocí vazeb. Výsledný dotaz je následně automaticky překládán do jazyka Cypher." [3]

1.3.2.2 Ovládání jazyka

MBIQL má umožnit uživateli vytvářet dotazy na základě dvou skupin grafických prvků. Po zvolení odpovídajících prvků a vybrání hodnot je jazyk přeložen do grafového dotazovacího jazyka Cypher. Tomuto překladu je věnována část implementace v kapitole 4.

- **Základní grafické prvky** - Uživatel si v průběhu dotazování vybírá jaký ze dvou základních prvků chce zobrazit. Na výběr má ze zobrazení "Objektu" nebo "Vazby".

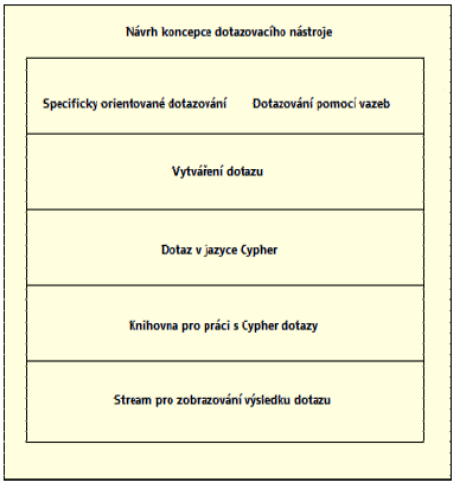
- **Ostatní grafické prvky** - Ostatní grafické prvky umožňují blíže specifikovat "Objekt" nebo "Vazbu". Specifikují také zobrazovaný výsledek dotazu. Mezi tyto prvky patří Seznam, Pole příznaků, Pole logických operátorů a Textové pole.

1.3.3 Dotazovací nástroj

Z obrázku 1.3 je vidět základní struktura návrhu nově vznikající aplikce. Pro frontend byly navrženy jako cílové technologie: HTML, CSS, Javascript. Pro backend pak PHP framework, Java.

1.4 Formální požadavky

Na nově vznikající softwarovou aplikaci jsou kladeny následující požadavky:



Obrázek 1.3: Návrh koncepce dotazovacího nástroje [3]

1.4.1 Funkční požadavky

- importovat entity a vztahy z MBI do grafové databáze
- umožnit dotazování na samostatný jeden objekt
- umožnit dotazování i na širší kontext - vztahy mezi jednotlivými objekty
- podporovat různé uživatele
- zobrazit výsledky dotazů novým způsobem
- umožnit ukládání dotazů

1.4.2 Nefunkční požadavky

- automatizovat importování dat z MBI do grafové databáze
- použít grafovou databázi Neo4j a její dotazovací jazyk Cypher
- implementovat nový grafický dotazovací jazyk MBIQL
- transformovat dotazy do jazyka Cypher
- umožnit dostupnost přes web
- optimalizovat pro webový prohlížeč Google Chrome

1.5 Specifikace cíle

Cílem této diplomové práce je vytvoření softwarového systému, který umožní interaktivně procházet a dotazovat se na objekty MBI pomocí jazyka MBIQL. Návrh jazyka vznikl v bakalářské práci O. Batíka [3]. Tato práce byla v předchozích kapitolách shrnuta a uvedeny stěžejní body.

Pro dosažení tohoto cíle je nutné nejprve zrevidovat a doplnit stávající návrh pro konkrétní implementaci systému. Detailněji tedy budou vypsány uživatelské požadavky na systém, blíže rozebráno dotazování pomocí MBIQL, uveden grafický návrh a zvolena odpovídající architektura i implementační platforma. Úpravou také musí projít stávající schéma uložiště dat z MBI (v Neo4j).

Jednotlivé části softwarového systému budou následně implementovány a podrobeny uživatelským testům, na základě kterých bude provedeno zhodnocení a případné úpravy sytému.

1.6 Metodika vývoje

Na základě předchozí kapitoly lze práci chápat jako softwarově inženýrský projekt. Jeho základní principy budou s využitím zdrojů [5] a [6] popsány.

Softwarové inženýrství se zabývá procesem vzniku softwarového systému. Celý proces se skládá z několika kroků, které je potřeba dodržet.

První z nich je sběr požadavků, kdy probíhá komunikace se zákazníkem, vyjasňování požadavků, různé meetingy a schůzky. To vše s cílem správně pochopit, co má být výsledkem a co přesně je od softwaru požadováno.

Dalším z kroků je analýza a návrh, kdy jsou nashromážděné požadavky detailně rozebrány a vytvoří se z nich návrh řešení, které například ve formě modelu slouží jako podklad pro samotnou implementaci.

Následně je tedy prováděna implementace, při které dochází již ke vzniku funkčního softwaru.

Po jejím dokončení probíhá testování a ověřování kvality produktu. Případné nalezené nedostatky nebo chyby jsou opraveny a produkt je připraven pro reálné použití u zákazníka. Po tomto kroku by již neměly být nalezeny žádné další chyby, protože dále systém „opouští“ firmu a míří k zákazníkovi, kde je jakákoliv nalezená chyba, která je potřeba opravit, velmi drahá.

U zákazníka dojde k nasazení/zprovoznění softwarového systému. Systém je tedy podroben reálné zátěži uživatelů.

Posledním z kroků je krok údržby, kdy je v potřebě opravy nebo přidání nové funkcionality systém upravován.

Tyto kroky vývoje se mohou měnit v závislosti na zvolené metodice. Metodiky se dělí na tradiční a agilní, kdy tradiční metodiky postupně procházejí skrze uvedené kroky, zatímco agilní metodiky upřednostňují rychlý vývoj, bě-

hem něhož je velmi častá komunikace se zákazníkem a detaily návrhu se řeší i v průběhu implementace.

Kapitola 2

Analýza a návrh

V následujících kapitolách je detailně navržen celý softwarový systém. Výsledek nám následně poslouží jako podklad pro realizaci.

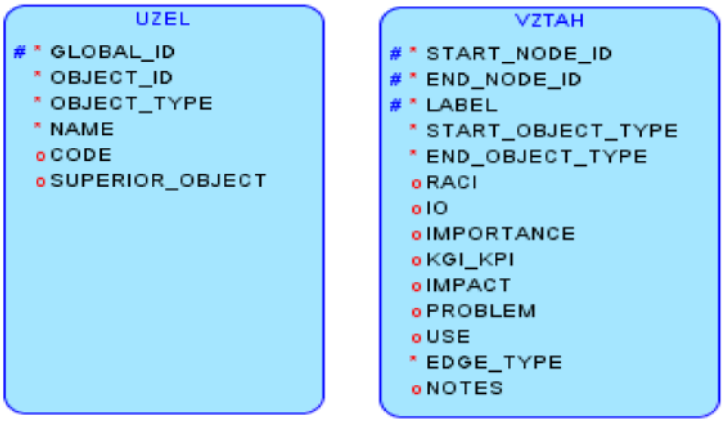
Nejprve se zaměříme na proces importu a transformace dat do grafové databáze Neo4j, dále na samotný jazyk MBIQL, jehož návrh bude primárně vycházet z práce pana Batíka. Podle potřeb dojde k revizi a doplnění. Následně bude specifikován a popsán celý systém jako celek a na závěr uveden grafický návrh.

2.1 Proces transformace dat

Data z portálu MBI jsou již v jednom z řešení (popsané v práci [2]) uložena v grafové databázi Neo4j. Pro potřeby vznikajícího systému je ale nutné vytvořit nové uložiště a to z důvodu změny modelu MBI (viz obrázek 2.2) a potřeb jazyka MBIQL. Kvůli těmto dvěma důvodům vzniklo nové schéma uložiště zobrazené na obrázku 2.1. Schéma se opět skládá z entity Uzel (Objekt) a Vztah (Vazba), jenž jsou základními prvky grafu, do nichž se namapují data z MBI. Ty jsou dodány ve formátu CSV.

2.1.1 Databázové schéma Uzlu

U Uzlu je potřeba zmínit atribut SUPERIOR_OBJECT, který nahradil atributy Skupina a Podskupina a odkazuje na nadřazený objekt. Tudíž u konkrétní instance odkazuje na Podskupinu a u Podskupiny na Skupinu. V databázovém schématu bude tento atribut repretentován vztahem typu INSTANCE_HIERARCHY. Tyto speciální vazby budou zmíněny v následující kapitole. Ostatní atributy zůstávají stejné, jako tomu bylo u dřívějšího modelu: GLOBAL_ID, OBJECT_ID, OBJECT_TYPE, NAME a CODE.



Obrázek 2.1: Základní entity datového uložiště [3]

2.1.2 Databázové schéma Vztahu

U Vztahu došlo k rozšíření dřívějšího atributu POZNAMKY na jednotlivé vlastnosti: RACI, IO, IMPORTANCE, KGI_KPI, IMPACT, PROBLEM, USE a NOTES. Dále se změnil atribut EDGE_TYPE, který nyní definuje typ hrany znázorněné na obrázku 2.2. Vznikly tyto tři typy:

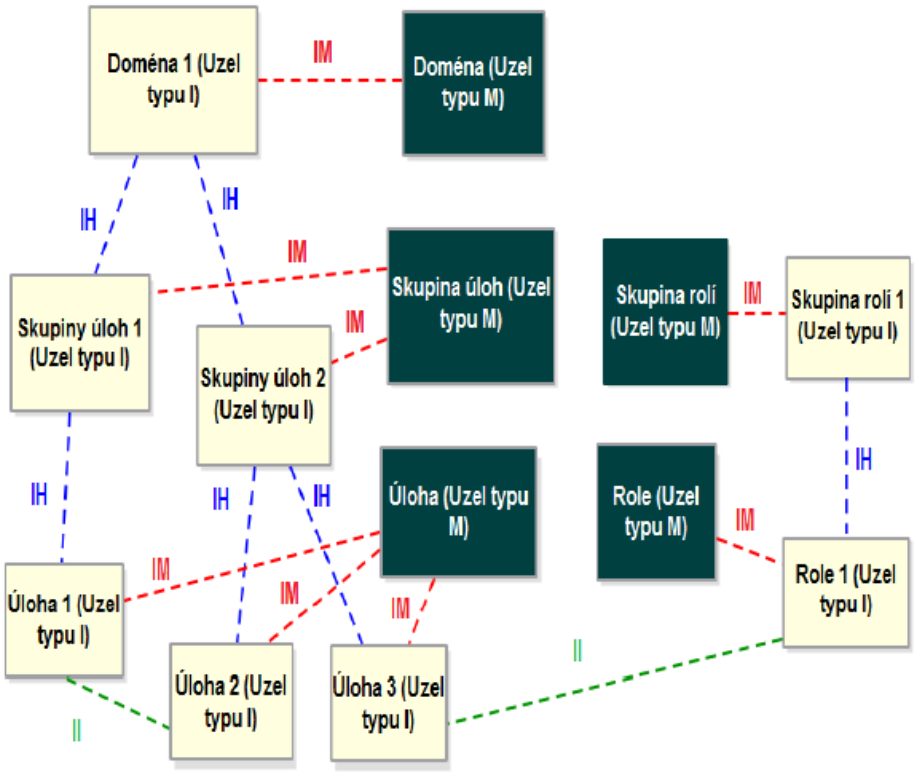
- **II** - INSTANCE-INSTANCE - vztah mezi instancemi objektů (Například objekt TASK a ROLE)
- **IH** - INSTANCE-HIERARCHY - vztah mezi instancí objektu a nadřazeným objektem (Například TASK a TASKGROUP - Skupina)
- **IM** - INSTANCE-MODEL - vztah mezi instancí objektu a modelovým objektem, který spojuje uzly stejné hierarchické úrovně (Všechny instance objektu TASK jsou spojeny s modelovým objektem, který zaštiťuje všechny objekty typu TASK

Ostatní atributy zůstávají stejné: START_NODE_ID, END_NODE_ID, LABEL, START_OBJECT_TYPE, END_OBJECT_TYPE

2.1.3 Import dat do databáze Neo4j

Dřívější způsob importu dat byl prováděn skrze program Batch importer [3], kdy byly připraveny dva CSV soubory a ty naimportovány do databáze. Al-

[3]Program je dostupný z: `https://github.com/jexp/batch-import/tree/20`



Obrázek 2.2: Nové schéma datového uložiště [3]

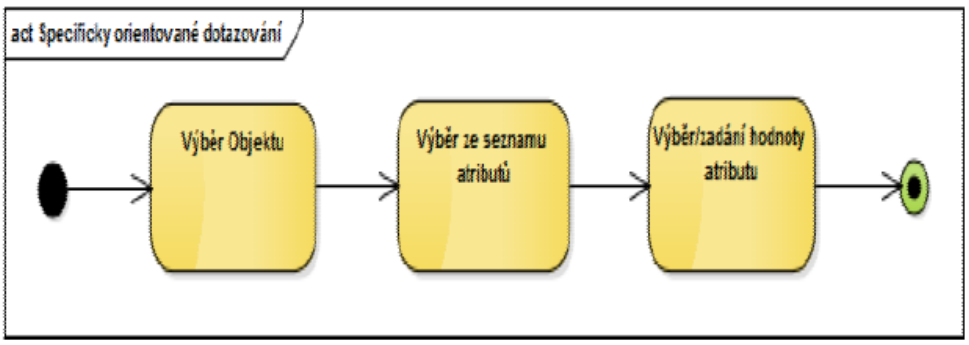
ternativou je vytvořit databázi přímo v javě za pomoci knihovny neo4j [4] a rovnou ji naplnit daty.

Před samotným vytvořením databáze je potřeba Uzly a Vztahy připravit. Vstupní data jsou získána z MBI ve formě CSV souborů, kde je nejprve nutné přeindexovat uzly. Každý tedy dostane svůj unikátní globální identifikátor, který bude dále využit. Další úpravy už se týkají pouze vztahů.

Ve vstupních datech existují pouze vztahy typu INSTANCE-INSTANCE, tudíž je nutné vytvořit i ostatní typy. Ty se lze vytvořit z již připravených Uzlů.

Po těchto úpravách je možné vytvořit novou databázi a naplnit ji připravenými Uzly a Vztahy. Jako poslední úprava je potřeba přidat administrátorský profil. Profily budou popsány v kapitole 2.3

[4] Knihovna je dostupná z: `https://mvnrepository.com/artifact/org.neo4j/neo4j/3.0.6`



Obrázek 2.3: Průběh specificky orientovaného dotazování [3]

2.2 Revize návrhu MBIQL

Jazyk MBIQL je doménově specifický jazyk pro dotazovaní dat z MBI. Po vytvoření dotazu pomocí grafických prvků je dotaz přeložen do jazyka Cypher a přeložený příkaz je následně vykonán nad databázi Neo4j, která vrátí výsledky.

Pomocí MBIQL je možné vytvářet dva typy dotazů, které budou popsány v kapitolách 2.2.1 a 2.2.2

2.2.1 Specificky orientované dotazování

Tento typ dotazu by měl řešit požadavek: Umožnit dotazování na samostatný jeden objekt. Průběh dotazu je znázorněn na obrázku 2.3.

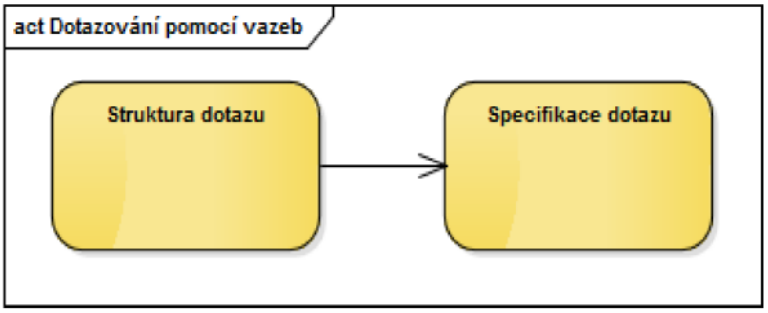
Uživatel si vybere konkrétní typ objektu (OBJECT_TYPE) ze seznamu všech objektů (na výběr má i z hierarchicky nadřazených). Dále vybere jeden atribut ze seznamu. Tento seznam je omezen pouze na 2 atributy (NAME a CODE), jelikož z uživatelského hlediska použitelnosti se neočekává, že by chtěl vyhledávat podle indentifikačních čísel nebo podle typu, který je již předem vybrán. V posledním kroku zvolí hodnotu atributu (opět ze seznamu).

V návrhu práce pana Batíka byla možnost zadat hodnotu do textového pole a využít pomoc našeptávače. Systém tuto možnost bude nabízet až v případných dalších rozšířeních. Momentálně je kladen důraz na splnění základních požadavků.

2.2.2 Dotazování pomocí vazeb

Dotazování pomocí vazeb klade za cíl splnit požadavek: Umožnit dotazování i na širší kontext. Průběh dotazu je znázorněn na obrázku 2.4.

Uživatel může vytvořit komplexnější dotaz skládající se z více objektů a vazeb. U každé z těchto entit je možnost dospecifikovat atributy. Tento typ



Obrázek 2.4: Průběh Dotazování pomocí vazeb [3]

dotazování prošel nejvíce změnami oproti původnímu návrhu. Změny budou probrány v následujících kapitolách níže (viz 2.2.2.1 a 2.2.2.2).

Tvorba dotazu se skládá ze dvou části: Struktura dotazu a Specifikace dotazu (viz obrázek 2.2.2).

2.2.2.1 Struktura dotazu

V prvním kroku si uživatel volí ze základních grafických prvků - Objekty a Vazby, kterými se chce dále zabývat.

V původním návrhu byl vybrán pouze počáteční Objekt a dále uživatel vybíral Vazby, které na sebe navazovaly, až sestavil základní strukturu dotazu.

V novém návrhu uživatel vybírá v cyklu pouze Objekty, ke kterým se automaticky přidávají i Vazby. Uživatel může vybrat pouze takový Objekt, který je spojen s předcházejícím Objektem nějakou Vazbou. Je možnost vybraný Objekt smazat a pokračovat jinou „cestou“.

Na závěr se zvolí, zda budou zobrazeny i Modelové uzly (viz obrázek 2.2).

2.2.2.2 Specifikace dotazu

Ve druhém kroku uživatel blíže specifikuje již vybrané Objekty a Vazby.

V původním návrhu nejprve uživatel označí prvky z prvního kroku, které chce upravovat. Provede jejich prioritizaci podle uzávorkování pomocí Pole logických operátorů a následně v cyklech určuje hodnotu jednotlivých atributů na každém z prvků. Nakonec určí návratové hodnoty dotazu.

Nový návrh umožní uživateli globální pohled na celý dotaz. Při specifikování jednotlivých dotazů vidí ostatní prvky seřazené podle výběru z první části. Má tedy přehled, jak vypadá celý vytvářený dotaz a nejen jeho část při určování atributu na jednom prvku. Současně také vidí již určené atributy.

U každého prvku Objekt na závěr určí, jestli chce zobrazit jeho návratovou hodnotu. V případě dvou zvolených Objektů, které jsou navzájem propojeny vazbou, zobrazí i tuto vazbu.

Aktuálně u tohoto návrhu není možné určovat logické operátory OR nebo AND. Jedná se o případné rozšíření aplikace do budoucna.

2.2.3 Vzorové dotazy

K oběma typům procházení vznikly vzorové dotazy, které budou v závěru práce ověřeny.

1. Zobraz instanci objetku Úloha, která má název: Správa IT infrastruktury.
2. Zobraz všechny Metody, které mají vztah k Úloze: Propojení metrik byznysu a metrik IT.
3. Zobraz všechny Skupiny úloh, které spadají do Domény: Strategické řízení IT.
4. Najdi pouze faktory, které jsou využity u Scénáře: CIO se připravuje na poradu vedení podniku.
5. Zjisti, do jaké Skupiny metrik patří Metrika: Počty spravovaných technických prostředků. Zobraz všechny hrany a uzly.
6. Zjisti, do jaké Skupiny metrik patří Metrika: Počty spravovaných technických postředků. Zobraz všechny hrany a uzly. Zobraz také uzly prvků modelu MBI (Uzly typu M).

Poslední dotaz, týkající se vazby OR, byl vzhledem k návrhu vynechán.

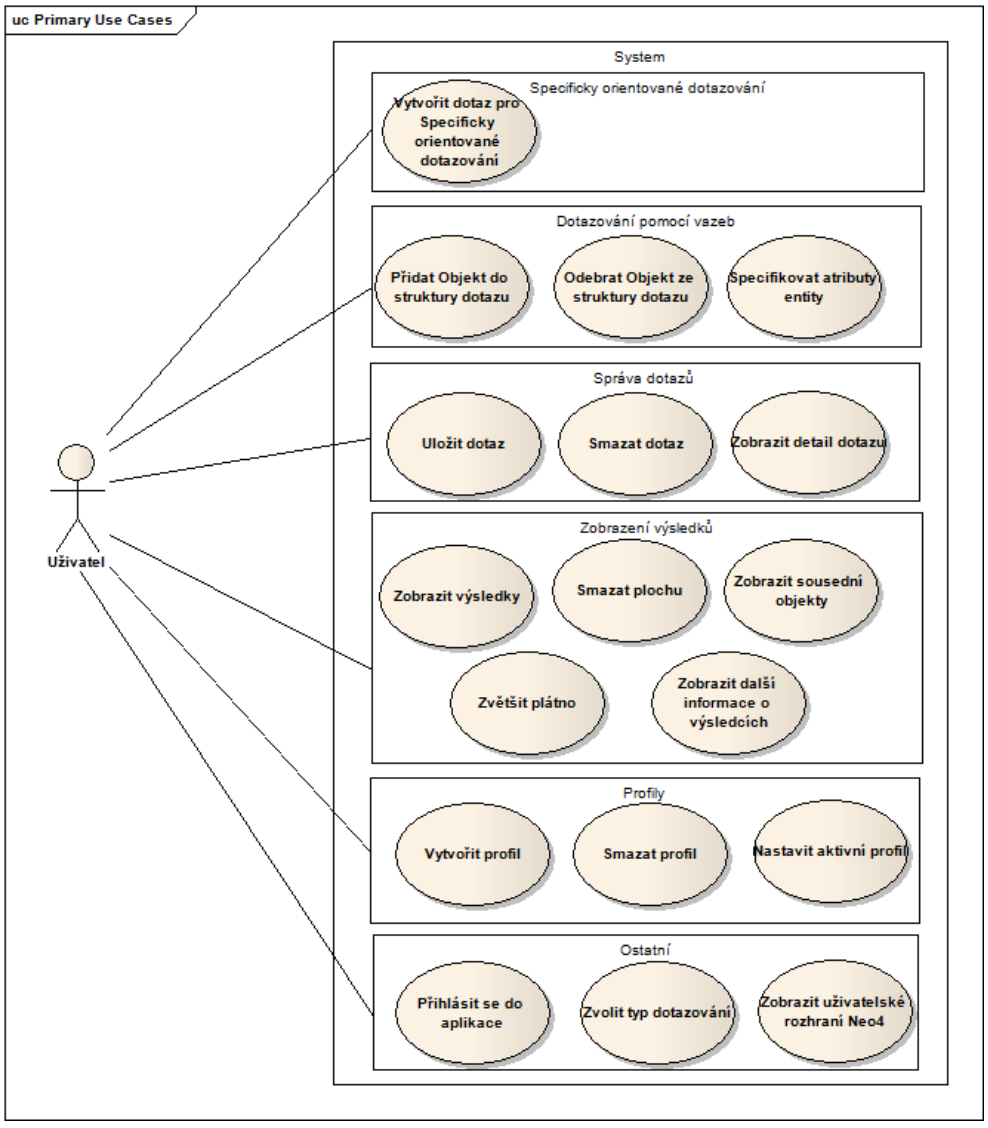
2.2.4 Asistované procházení grafem

Asistované procházení grafem také vychází z práce pana Batíka, ale už se přímo netýká jazyka MBIQL. Toto procházení má za cíl umožnit další interagování s již vytvořeným grafem na základě předešlého dotazování, kdy uživatel chce pokračovat a získávat další informace.

V nově vytvářeném systému bude implementován jednoduchý prototyp v podobě zobrazení všech další souvisejících Objektů. Systém tedy uživateli zobrazí po dvojitém kliknutí na uzel všechny další Objekty, které s daným uzlem mají společnou vazbu. Tento typ procházení bude v budoucnosti možné rozšířit.

2.3 Softwarový systém

V této kapitole je představen celý softwarový systém. Na modelu případů užití 2.5 jsou detailně specifikovány požadavky na systém a v následujících podkapitolách jsou pro přehlednost kategorizovány do menších částí a představeny zvlášť.



Obrázek 2.5: Případy užití

2.3.1 Specificky orientované dotazování

Požadavky týkající se metody Specificky orientované dotazování představené v kapitole 2.2.1. Uživatel se pohybuje mezi třemi grafickými prvky Seznam, z nichž vybírá nabízené možnosti.

2.3.1.1 Vytvořit dotaz pro Specificky orientované dotazování

Uživatel pomocí grafických prvků Seznam vytvoří dotaz, který bude směřován na jednu entitu. Po vybrání jednoho prvku ze seznamu systém nabídne odpovídající možnosti v dalším Seznamu.

2.3.2 Dotazování pomocí vazeb

Požadavky týkající se metody Dotazování pomocí vazeb představené v kapitole 2.2.2. Uživatel má k dispozici dva grafické prvky Seznam, z nichž první obsahuje dostupné entity pro výběr a druhý již vybrané entity pro strukturu dotazu. Po vybrání prvků do struktury dotazu může pomocí ostatních grafických prvků blíže specifikovat dotaz. Při přidání nebo odebrání entity se specifikované atributy smažou.

2.3.2.1 Přidat Objekt do struktury dotazu

Uživatel po kliknutí přidá entitu do struktury dotazu. Mezi dva již přidané Objekty se automaticky přidá i Vazba.

2.3.2.2 Odebrat Objekt ze struktury dotazu

Uživatel odebere poslední přidaný Objekt a současně s ním se automaticky odebere i poslední přidaná Vazba (pokud existuje).

2.3.2.3 Specifikovat atributy entity

Uživatel nastaví atributy vybranému Objektu nebo Vazbě.

2.3.3 Správa dotazů

Požadavky týkající se správou dotazů v systému. Uživatel získá možnost ukládat dotazy a dále s nimi pracovat.

2.3.3.1 Uložit dotaz

Dotaz se uloží do systému pod aktivní profil a obsahuje jméno a překlad do jazyka Cypher. Dotaz lze vytvořit jak při Specificky orientovaném dotazování, tak i při Dotazování pomocí vazeb. Každý dotaz musí mít napříč systémem unikátní název.

2.3.3.2 Smazat dotaz

Smaže dotaz z celého systému. Mazat lze pouze dotazy u aktivního profilu.

2.3.3.3 Zobrazit detail dotazu

Zobrazí detailní informace o dotazu - přeložený Cypher příkaz a možnost zobrazit výsledky nebo dotaz smazat.

2.3.4 Zobrazení výsledků

Požadavky týkající se vizualizace výsledků dotazů a jejich následnou manipulací.

2.3.4.1 Zobrazit výsledky

Na základě přeloženého Cypher dotazu se uživateli zobrazí výsledky v podobě grafu. Graficky znázorněné pomocí kruhů (Objekty) spojených navzájem čarami (Vazby). Objekty jsou barevně odlišeny podle typu entit.

2.3.4.2 Zvětšit plátno

Zvětší plochu pro zobrazení výsledků (plátno) na velikost full HD (1920x1080).

2.3.4.3 Smazat plochu

Smaže aktuálně zobrazené výsledky (celý graf). Plátno zůstane prázdné.

2.3.4.4 Zobrazit další informace o výsledcích

Po najetí myši na Objekt nebo Vazbu jsou zobrazeny další informace, které daná entita obsahuje.

2.3.4.5 Zobrazit sousední objekty

Po dvojitém kliknutí na Objekt se k němu zobrazí všechny jeho Vazby a s nimi i sousední Objekty, u kterých lze také zobrazit sousedy.

2.3.5 Profily

Požadavky týkající se správy profilů. Každý profil může symbolizovat jiného uživatele nebo ho lze využít pro kategorizaci dotazů.

2.3.5.1 Vytvořit profil

Softwarový systém vytvoří nový profil, který primárně slouží ke kategorizaci dotazů. Tudíž každý profil obsahuje různé dotazy. Výjimkou je profil *admin*, který obsahuje všechny vytvořené dotazy v systému. Uživatel tedy nemusí prohledávat všechny profily, aby zjistil, které dotazy už systému jsou a nebo které je potřeba přidat.

2.3.5.2 Smazat profil

Umožňuje smazat daný profil z aplikace a společně s ním se s mažou i vytvořené dotazy pro tento profil (profil *admin* nelze smazat). Dotazy se nadále už nezobrazují ani v profilu *admin*.

2.3.5.3 Nastavit aktivní profil

Umožní nastavit profil jako aktivní. Všechny dotazy, které se vytvoří, budou náležet aktuálnímu profilu.

2.3.6 Ostatní

Požadavky týkající se ostatní funkcionality systému.

2.3.6.1 Přihlásit se do aplikace

Na úvodní stránce by měl systém umožnit uživateli přihlásit se pod uživatelským jménem a heslem nastaveným při prvním spuštění uživatelského rozhraní Neo4j.

2.3.6.2 Zvolit typ dotazování

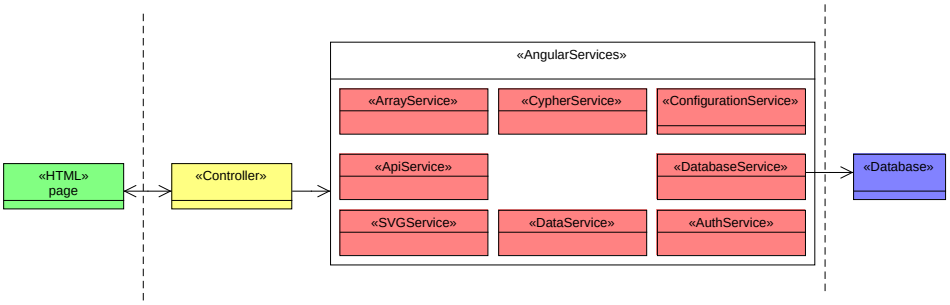
Uživatel má na výběr zvolit mezi dvěma typy dotazovacích metod: Specificky orientované dotazování a Dotazování pomocí vazeb. Mezi těmito metodami může libovolně přepínat.

2.3.6.3 Zobrazit uživatelské rozhraní Neo4j

Systém se přepne do uživatelského rozhraní Neo4j, kde se uživatel může dotazovat pomocí jazyka Cypher.

2.4 Architektura

Systém je navržen jako webová single-page aplikace, pro kterou je typické využívání serveru pouze jako uložiště dat [7]. Vzhledem k tomu, že jsou data vykreslována přes javascript a není potřeba stahovat celou stránku znova, pokud se nezměnil její obsah, jsou tyto aplikace rychlejší.



Obrázek 2.6: Architektura softwaru

Systém je nasazen na webový server Apache Tomcat a je dostupný přes webový prohlížeč (optimalizována pro Google Chrome). K zobrazení dat uživatelům se využívá značkovací jazyk HTML a kaskádové styly (CSS). Použit je i framework Bootstrap.

Pro dynamičnost stránek je použit Javascript, konkrétně framework AngularJS (viz kapitola 3.1) a pro zobrazení výsledků dotazů javascriptová knihovna D3 (viz kapitola 3.2).

2.4.1 Vrstvy aplikace

Na diagramu je vidět základní struktura tříd aplikace (pro přehlednost jsou uvedeny jen jejich názvy), které dohromady tvoří tři vrstvy. Barevně jsou odlišeny rozdílné části aplikace v Angularu:

- HTML stránky - slouží pro vizualizaci dat
- Controllers - spravují data mezi frontendem a backendem
- Services - služby využivatelné na více místech aplikace
- Databáze - nejedná se o část Angularu, ale znázorňuje komunikaci s databází

Více jsou jednotlivé komponenty pospány v kapitole 3.1.

2.4.1.1 Prezentační vrstva

Jedná se o vrstvu, která zobrazuje data uživateli a v diagramu je znázorněna HTML stránkou. Pro jednoduchost není konkretizována, ale zastupuje obecně všechny použité stránky v aplikaci, kdy každá z nich zobrazuje (prezentuje uživateli) data nějaké části aplikace. K HTML stránce také patří CSS soubor, ve kterém jsou nadefinovány styly aplikace, neboli způsob jakým budou data zobrazena (barva, velikost, ohraničení, umístění...).

2.4.1.2 Aplikační vrstva

Tato vrstva se stará o samotnou logiku aplikace. Jsou do ní zařazeny Kontrolery (Controllers) a Služby (Services). Kontrolery komunikují s Prezentační vrstvou (předávají data) a pomocí Služeb i s Datovou vrstvou. Tato vrstva tedy slouží i jako prostředník mezi ostatními vrstvami.

Hlavní funkcionalita jednotlivých služeb:

- **ArrayService** - Poskytuje základní funkce pro práci s polem: Přidávání, mazání nebo filtrování na základě profilu.
- **CypherService** - Poskládá a vytvoří Cypher dotaz na základě vybraných entit a atributů nebo vrací již předpřipravené dotazy.
- **ConfigurationService** - Obsahuje aktuální nastavení proměných. Například profil nebo zobrazené vlastnosti o Objektech nebo Vazbách.
- **ApiService** - Poskytuje prostředníka v komunikaci mezi databází, konfigurací a zbylým systémem.
- **DatabaseService** - Slouží pro komunikaci s databází. Vytváří spojení do databáze, vytváří a maže dotazy a profily, spravuje cache...
- **SVGService** - Zobrazuje výsledky dotazů v podobě interaktivního grafu.
- **DataService** - Nastavuje, zda je zobrazeno uložiště dotazů.
- **AuthService** - Kontroluje, jestli je uživatel přihlášen nebo ne.

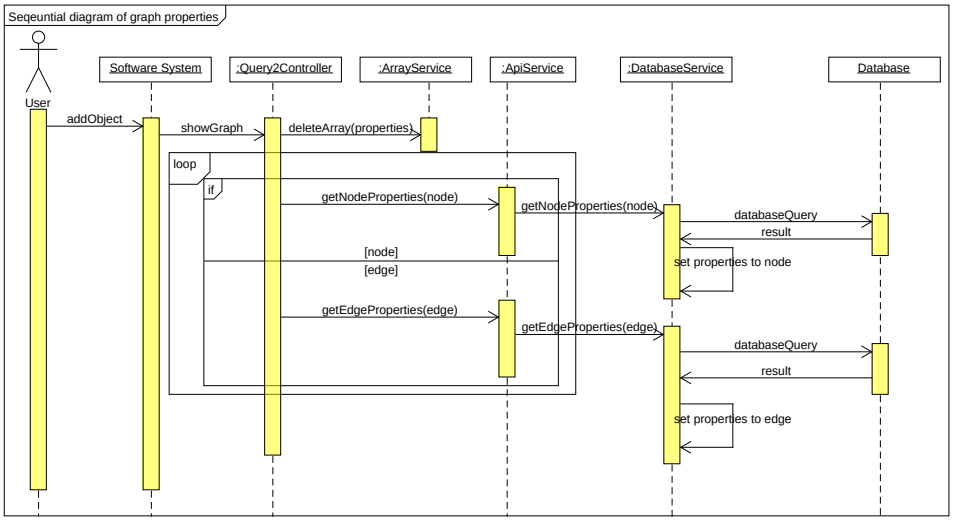
2.4.1.3 Datová vrstva

Datová vrstva slouží jako uložiště dat celého softwarového systému. V tomto případě se data ukládají do grafové databáze Neo4j a přistupuje se k nim pomocí Databázové Služby zmíňené výše.

2.4.1.4 Průchod mezi vrstvami

Pří interagování uživatele se systémem dochází v něterých případech k průchodu přes všechny zmíněné vrstvy. Pro konkrétní ukázku je zvoleno vytváření struktury dotazu při Dotazování pomocí vazeb, kdy uživatel vybere Objekt a jako výsledek se mu zobrazí všechny předchozí vybrané Objekty a Vazby i s jejich vlastnostmi. Průběh je znázorněn na sekvenčním diagramu 2.7 a popsán v následujících krocích:

1. Uživatel vybere Objekt, který chce přidat pomocí grafických komponent na webové stránce.
2. Softwarový systém zareaguje tím, že webová stránka začne komunikovat se svým Kontrolerem, který poslouchá na akci kliknutí na Objekt.



Obrázek 2.7: Sekvenční diagram vytvoření struktury dotazu

3. Kontroler pomocí *ArrayService* smaže současné pole vybraných Objektů a Vazeb a spustí cyklus, kde v závislosti na tom, jestli se jedná o Objekt nebo Vazbu zavolá příslušnou metodu na *ApiService*.

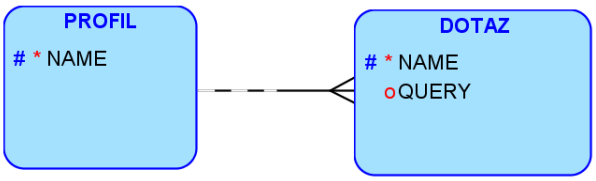
4. *ApiService* přepošle příchozí parametry dál do *DatabaseService*, která se stará o komunikaci s databází.

5. *DatabaseService* na základě příchozích parametrů vytvoří dotaz a dotáže se přímo do databáze Neo4j. Po vrácení odpovědi uloží výsledky do proměnné a díky Angularu, který zareaguje na změny, se výsledek propíše na frontend.

2.4.2 Cache

Pro omezení častého dotazování aplikace do databáze byla vyvořena cache, ve které jsou ukládány profily a k nim uložené dotazy. Cache tedy pod každým profilem obsahuje seznam již vytvořených dotazů. Uživatel v případě změny uživatelského profilu nemusí volat znovu do databáze ale již načte daný profil z cache. Cache je třeba obnovit v případě smazání nějakého z dotazů nebo jeho přidání. Strukura cache:

```
queriesCache =  [{
        "profile": "String",
        "queries": [{
            "name": "String",
```



Obrázek 2.8: Nové entity Profil a Dotaz

```
            "query":"String"
        }]
}]
```

Cache je možné v budoucnu upravit například pro ukládání celých výsledků dotazů (všech uzlů a hran), tudíž uživatel nebude muset čekat na celý proces, kdy je potřeba se dotázat do databáze a na základně výsledků vytvořit soubor z uzly a hranami (viz 3.2).

2.4.3 Nové databázové entity

V kapitole 2.1 bylo navrženo (upraveno) datové schéma pro základní entity pro tvorbu grafu: Uzel (Node) a Vazba (Relationship). Jelikož ale aplikace musí podporovat ukládání dotazů pro různé profily, je potřeba vytvořit další objekty typu Node pro uložení těchto dat. Návrh je možné vidět na obrázku 2.8 (Více o grafové databázi je napsáno v kapitole 3.3).

2.4.3.1 Profil

Schéma Profil je navrženo pro kategorizaci uživatelských dotazů. Každý profil tedy může představovat jiný typ dotazů nebo dotazy pro jiný typ uživatelů. Současné schéma obsahuje pouze jeden atribut: **NAME**. Ten obsahuje jméno profilu, které určuje jaké dotazy se uživateli zobrazí. V systému vždy existuje profil *admin*.

Schéma je připravené pro případné rozšíření. Profil v budoucnu může obsahovat další položky jako například kategorizaci dotazů, historii dotazů, ikonu a další.

2.4.3.2 Dotaz

Schéma Dotaz slouží pro ukládání jednotlivých dotazů. Ty musí být přiřazeny k nějakému profilu (výchozím profilem je *admin*, který nemůže být pomocí aplikace smazán). Schéma obsahuje dva atributy:

- **NAME** - Obsahuje unikátní jméno dotazu v celém systému.

- **QUERY** - Obsahuje dotaz přeložený do jazyka Cypher.

Schéma může být v budoucnu rozšířeno o další atributy. Například o prioritu dotazu, podle které mohou být dotazy seřazeny.

2.5 Wireframe

Wireframe, neboli drátěný model webu, slouží jako návrh pro rozložení prvků webové aplikace, jejich provázanosti a funkcionality [8]. Jedná se o grafický model celé aplikace, kdy pomocí různého softwaru (někdy stačí i papír a tužka) je nakresleno, kde se která komponenta bude v aplikaci nacházet. Model by měl obsahovat všechny stránky dané aplikace a měla by být znázorněna i funkcionalita jednotlivých prvků. Tedy i kam se lze pomocí kterého prvku prokliknout. Nejde však o přesnou grafickou specifikaci a tudíž rozmístění prvků i jejich velikost mohou být ve výsledné aplikaci lehce odlišné.

Následující kapitoly popisují stránky aplikace a funkcionalitu jednotlivých prvků.

2.5.1 Přihlašování

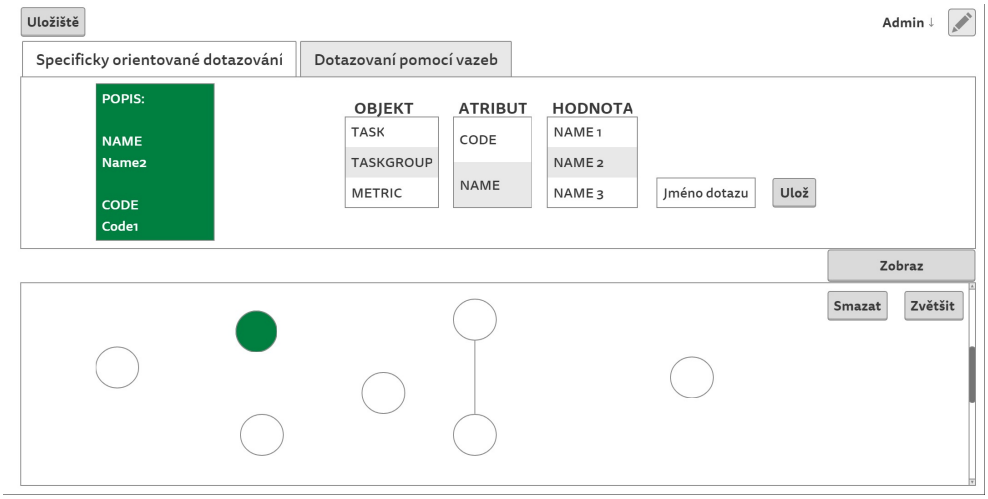
První stránka webové aplikace je přihlašovací formulář. Uživatel musí zadat přihlašovací údaje (jméno a heslo), které jsou současně platné i pro databázi Neo4j. Do té je možné se skrze aplikaci „prokliknout“. Po úspěšném přihlašení se zobrazí úvodní stránka.

2.5.2 Wireframe Specificky orientovaného dotazování

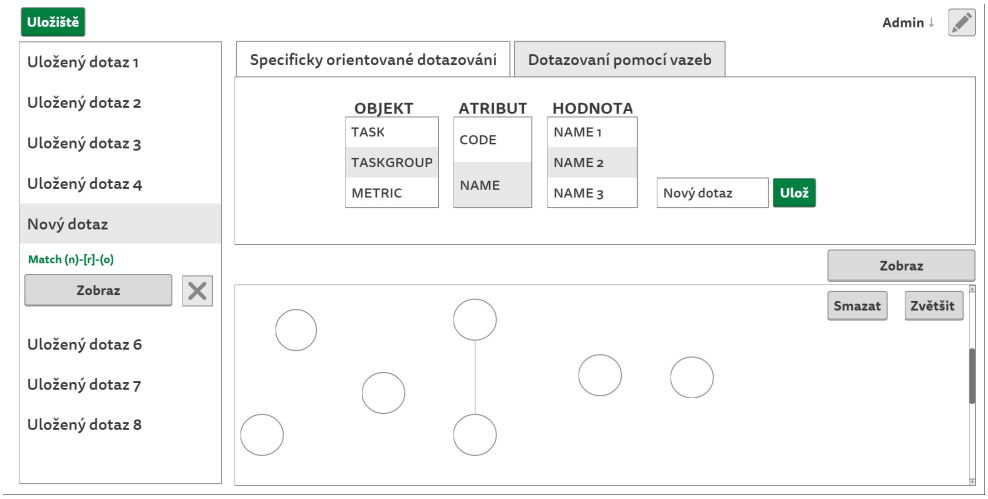
Na obrázku 2.9 je vidět základní strukturu aplikace. V horním panelu je možné pomocí tlačítka vlevo zobrazit uložiště dotazů. Na pravé straně je vypsán aktuálně používaný profil (*admin*) a tlačítko, kterým je možné spravovat profily.

Pod vrchním panelem je okno se záložkami pro oba dva typy dotazů. Vybráno je Specificky orientované dotazování. To se skládá ze tří seznamů, z nichž nejdříve uživatel vybere typ Objektu, po té z druhého atribut a nakonec konkrétní hodnotu atributu. Tím došlo k vytoření dotazu. Uživatel může dále dotaz pojmenovat a uložit stisknutím tlačítka nebo zobrazit výsledky pomocí tlačítka *Zobrazit*.

Poslední část obrázku je okno ve spodní části sloužící pro zobrazení výsledků dotazu. Ty se zobrazí v podobě uzlů a hran. Po „přejetí“ myší přes uzel nebo hranu dojde k zobrazení informací o daném prvku v podobě okna zvýrazněného zeleně. V této části jsou ještě dvě další tlačítka: *Smazat* a *Zvětšit*. *Smazat* slouží pro vyčištění vykreslovacího plátna - smaže všechny uzly a hrany. *Zvětšit* zvětší vykreslovací plochu, aby uživatel nemusel při procházení výsledků scrollovat.



Obrázek 2.9: Wireframe Specificky orientovaného dotazování



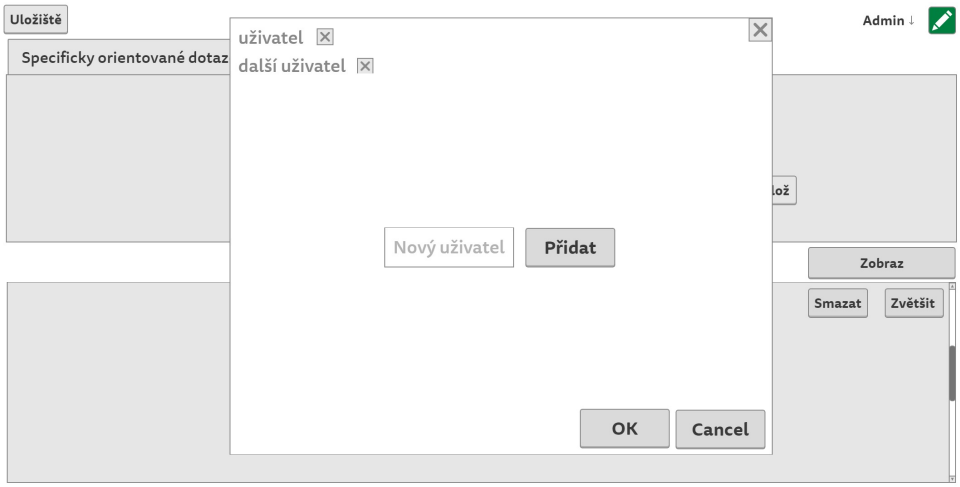
Obrázek 2.10: Wireframe - uložiště dotazů

2.5.3 Wireframe Uložiště dotazů

Na obrázku 2.10 lze vidět chování aplikace v případě kliknutí na tlačítko *Uložiště*. V levé části se zobrazí panel, ve kterém jsou všechny dotazy pro daný profil. V případě profilu *admin* se zobrazí všechny uložené dotazy.

Po kliknutí na dotaz se zobrazí překlad do jazyka Cypher a tlačítka pro smazání dotazu a pro zobrazení výsledků.

Šířce tohoto panelu se musí přizpůsobit dotazovací i vykreslovací okno.



Obrázek 2.11: Wireframe - profily

2.5.4 Wireframe Profily

Na obrázku 2.11 je kliknuto na tlačítko v horním panelu pro správu profilů. Po kliknutí se zobrazí modální okno, ve kterém jsou zobrazeny všechny profily kromě *admina*.

Admin je speciální profil, který není možné smazat. U admina lze pouze mazat dotazy. V případě smazání dotazu, který patřil i pod jiný profil, dojde ke smazání i na tomto profilu.

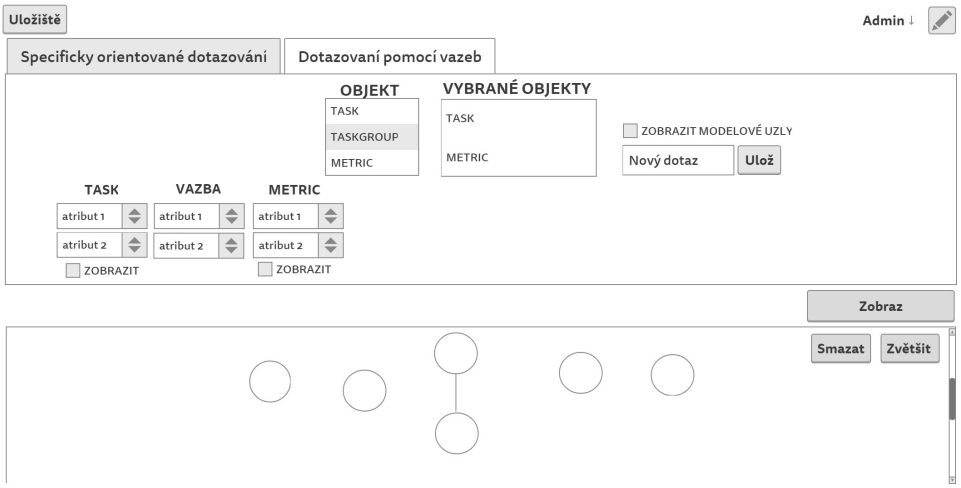
Ostatní profily lze mazat a pomocí tlačítka *Přidat* lze také profily nově vytvářet. Změny se v aplikaci projeví až po stisknutí tlačítka *OK*. V případě stisknutí tlačítka *Cancel* nebo křížku se změny neprojeví. Modální okno poté zmízí.

Po kliknutí na jméno profilu (ne na ikonu) může uživatel mezi vytvořenými profily přepínat. Na základě aktuálního profilu jsou také přístupné příslušné dotazy.

2.5.5 Wireframe Dotazování pomocí vazeb

Na obrázku 2.12 je vidět zvolení možnosti Dotazování pomocí vazeb. Na stránce došlo ke změně pouze v dotazovacím oknu. K dispozici jsou dva seznamy Objektů, pomocí kterých se vytvoří struktura dotazu. Z levého se Objekty vybírají a v pravém jsou už vybrané.

Pro každý vybraný Objekt se dole v oknu zobrazí možnost specifikovat blíže jeho atributy pomocí dropdown menu a zvolit možnost zobrazení ve výsledcích dotazu (checkbox *ZOBRAZIT*). Vybrané Objekty lze také z výběru odstranit dvojitým kliknutím. Odebrat lze ale pouze poslední přidaný.



Obrázek 2.12: Wireframe - Dotazování pomocí vazeb

Mezi dvěma Objekty automaticky vznikne Vazba, jejíž atributy lze také specifikovat. Nelze však zvolit možnost zobrazení, protože Vazba se automaticky zobrazí mezi dvěma zobrazenými Objekty, které spojuje.

Kromě možnosti uložení dotazu, která již byla popsána výše, uživatel zvolí pomocí checkboxu, jestli chce ve výsledcích zobrazit i Modelové Objekty.

Kapitola 3

Technologie

V následujících kapitolách jsou představeny technologie, které jsou využity při tvorbě softwarového systému. Dále jsou také uvedeny konkurenční možnosti a na závěr uvedeny důvody pro zvolení právě oněch vybraných technologií.

3.1 AngularJS

AngularJS (nebo jen Angular) je javascriptový framework určený pro tvorbu velkých rozšířitelných webových single page aplikací [9]. Za jeho vznikem v roce 2009 stojí dva vývojáři: Misko Hevery a Adam Abrons, v současné době je však pod spravou společnosti Google [10]. Základní strukturu aplikace tvoří šablony (templates), tvořené HTML stránkami rozšířené o nové prvky jazyka Angularu. Tyto šablony prezentují uživateli data, která jsou měněna pomocí kontrolerů (controllers). Každá šablona má svůj kontroler, který se stará o logickou funkčnost dané stránky.

Konkrétními prvky jazyka Angularu se zabývají následující kapitoly, které popisují i příklady, jak lze tyto prvky využít ve vyvíjeném softwarovém systému. Kapitoly vycházejí ze zdroje [11].

3.1.1 Moduly

Pod modulem si lze představit kontejner zaobalující ostatní prvky Angularu (šablony, direktivy, filtry...), které spolu navzájem souvisí. Aplikace je rozdělena na moduly podle jednotlivých funkčních částí. Existuje tedy zvlášť modul pro dotazování, zobrazování výsledků, ukládání dotazů a modul zaštiťující služby.

Moduly lze také skládat dohromady. Vkládané moduly jsou zapsány v hranatých závorkách hned za pojmenováním nově vytvářeného modulu. Jde takzvaně o dependency injection [5].

[5] Dependency injection je návrhový vzor, který se stará o správné vytváření komponent a vkládání závislostí napříč celým programem [12]

Následující příklad ukazuje hlavní modul aplikace, který má nainjektované ostatní moduly:

```
var mainApp = angular.module('mainApp', ['ui.router', '
    mainApp.storage', 'mainApp.graph',...]);
```

3.1.2 Direktiva a filtr

Direktivy rozšiřují základní HTML elementy a přidávají jim speciální chování. Například zobrazení nebo skrytí elementu, provedení akce po kliknutí nebo nastavení určité CSS třídy na základě proměnné z Controlleru, jak lze vidět v následující části kódu:

```
<div class="box" ng-class="{resizeGraph : isOpen , graph
    : !isOpen, fullScreen: isFull}">
```

Často používaná direktiva ve vyvíjenémm softwaru je *ng-repeat*, která umožňuje iterovat nad daným polem. Například při zobrazení všech atributů daného Objektu a Vazby:

```
<div class="custom-dropdown" ng-repeat="att in property
    .attributes | orderBy:att.name">...</div>
```

Předchozí část kódu vytvoří v HTML šabloně příslušný počet tříd podle velikosti pole *attributes*. Procházené pole je proměnná v kontroleru příslušné šablony a v případě odebrání prvku direktiva dynamicky zareaguje na změny (viz 3.1.3), tudíž znovu vykreslí už jen zbývající prvky.

Filtr upravuje výsledné hodnoty zobrazené uživateli. Může jít například o přeformátování výsledků - z malých písmen udělá velká nebo nahradí stávající hodnoty jinými. Ve výše zobrazeném kódu se filtr využívá k abecednímu seřazení výsledků na základě vybraného atributu.

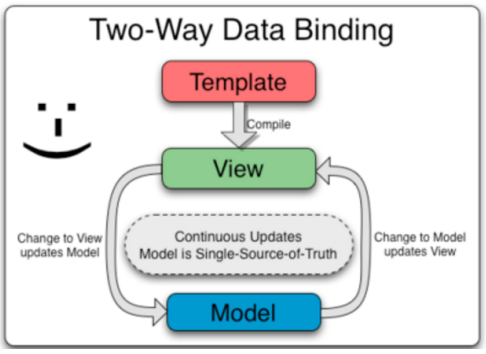
3.1.3 Two-way bindings

Tento koncept zaručuje, že kdykoliv se změní data v kontroleru, která jsou viditelná v šabloně, dojde k úpravě hodnoty v HTML DOMu (Document Object Model) - strom elementů vytvořený na základě HTML stránky. Tudíž každá změna hodnoty proměnné je automaticky ihned vidět na straně uživatele v prohlížeči.

Princip tohoto konceptu je vidět na obrázku 3.1. Šablona je zkompilována do podoby, kterou vidí uživatel v browseru (view) a jakékoliv změny procházejí pouze mezi modelem a view (i obráceně).

Data lze ve view zobrazit pomocí speciálního zápisu a to dvojitých složených závorek:

```
<div> {{property.name}} </div>
```



Obrázek 3.1: Two-way bindings [13]

3.1.4 Services

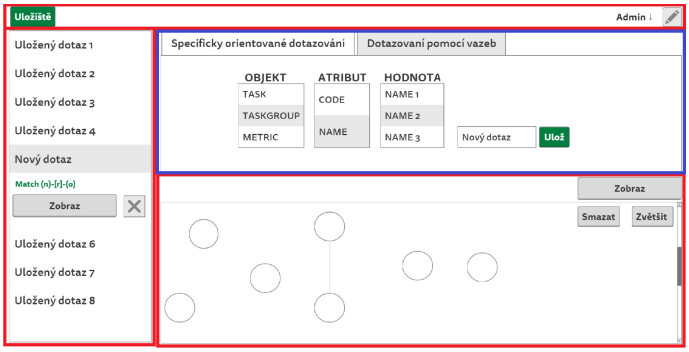
Services (služby) poskytují logickou funkcionalitu pro kontrolery. Narozdíl od direktiv nejsou pevně vázané na jeden modul nebo kontroler, ale jsou znovupoužitelné ve více částech kódu. Na obrázku architektury softwaru 2.4.1 lze vidět všechny používané služby. Typickým příkladem může být *ArrayService*, která je používána napříč všemi kontrolery, kde je potřeba pracovat s polem.

3.1.5 Routing, views a states

Routing (routování/směrování) v angularu umožňuje přepínání mezi jednotlivými stránkami webové aplikace. Pomocí tohoto způsobu (modulu Angularu *ngRoute*) je zaručeno chování webu jako SPA (Single Page Application). Nedochází tudíž k opakovanému načítání celého obsahu stránky - obsah se načte pouze při prvním zobrazení. Vyžadování tohoto chování je především z důvodu zvýšení rychlosti.

Rozložení jednotlivých komponent na stránce je dáno podle nadefinování pohledů (views) a stavů (states). Stav je celá stránka s vlastní šablonou a kontrolerem, která se zobrazí uživateli. Stav nemusí mít pouze jednu šablonu, ale může mít nadefinováno více pohledů - každý s vlastní šablonou a kontrolerem. Stavy se také mohou do sebe vnořovat a přepínání mezi nimi mění jen část stránky a zbytek, kterou tvoří pohledy, zůstává stejný. Toto chování je vysvětleno na obrázku vyvíjené webové aplikace 3.2.

Aplikace se skládá z hlavního stavu, který je dále dělen. Červeně zvýrazněná místa (menu, uložiště a graf) znázorňují views a modře zvýrazněný obdelník uprostřed (specificky orientované dotazování) vnořený state. Při překliknutí mezi záložkami (typy dotazů) dojde k výměně stavu, zatímco views zůstavají nezměněné.



Obrázek 3.2: Rozdělení aplikace na pohledy a stavy

3.1.6 Scope

Jeden z nejdůležitějších prvků angularu je takzvaně scope (do češtiny by se dal přeložit jako „oblast“ nebo „prostor“). Jedná se o objekt existující v kontroleru, který definuje „oblast“ všech dalších proměnných, ke kterým mají přístup šablony a direktivy. Přes scope je tedy možné předávat hodnoty backendu na fontend (do view). V následujícím kódu je vidět konkrétní ukázka použití:

```
$scope.labels = [];
result.records.forEach(function (record) {
  ArrayService.pushToList(record.get(0), $scope.labels)
});
```

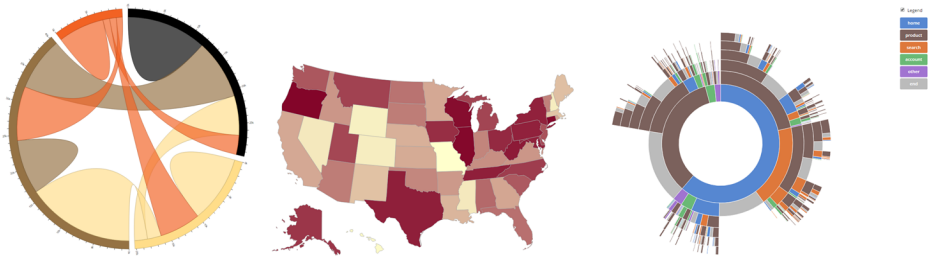
Po vrácení výsledků dotazu z databáze, jsou tyto výsledky předány do proměnné *$scope.labels*. Jelikož je tato proměnná definována ve scopu daného kontroleru, je možné ji v příslušné šabloně procházet například pomocí direktivy *ng-repeat*.

3.2 D3.js

Tato kapitola vychází ze zdrojů [14] a [15].

D3.js je javascriptová knihovna, která poskytuje nástroje pro vizualizaci a „oživení“ uživatelských dat. D3 je zkratka pro Data-Driven Documents, kde data představují poskytované vstupní informace, dokument cokoliv, co může být zobrazeno na webu (webová stránka) a knihovna tyto věci propojuje dohromady tak, že z dat vytváří dokumenty. D3.js tedy pracuje především s daty, které dostane od uživatele a na jejich základě vykreslí/vytvoří nějaký prvek na stránce (například tabulka nebo graf).

Autorem této open source knihovny je Mike Bostock a celý projekt je



Obrázek 3.3: Příklady vizualizace pomocí D3.js [16], [17], [18]

dostupný na GitHubu [6]. D3 bylo vydáno pod BSD licenci, která znamená, že dílo lze volně šířit pouze s uvedením autora. Knihovna umožňuje vykreslování nejrůznějších typů diagramů, koláčových nebo sloupcových grafů, map a sítí. S vykresleným výsledkem je možné dále pracovat - přeskupovat, přibližovat, měnit barvy nebo animovat.

Existuje mnoho různých způsobů k čemu tuto knihovnu využít. Některé z příkladů jsou vidět na obrázku 3.3.

Další jsou dostupné opět z GitHubu [7]. Pro účely vyvíjené aplikace se následující podkapitoly zaměří na tvorbu interaktivního grafu, pomocí něhož bude možné vykreslit právě Objekty a Vazby.

Základní funkce knihovny spočívá v načtení dat, které se připojí na vybraný prvkek na webové stránce. Tento prvek je dále transformován (namapován) na konkrétní formu vizualizace. Například je-li výsledným zobrazením sloupcový graf, data s větší hodnotou mohou mít větší sloupec nebo například sytější barvu. Vizualizace se dále ještě může měnit a to na základě změny dat nebo podnětu od uživatele. Pro vykreslování prvků je použito SVG [8].

3.2.1 Staticky vytvořený SVG obrázek

SVG se používá pro vykreslování obrázků v HTML stránce s tím rozdílem, že se obrázek nevkládá klasickým způsobem, ale je popsán pomocí XML tagů [19]. Jelikož se jedná o vektorový formát, tak je u obrázku při změně velikosti, zachována jeho kvalita. Dále v této kapitole je ukázáno, jak vykreslit jednoduchý obrázek.

Pro vykreslení jakéhokoliv prvku je potřeba vytvořit plátno, do kterého je obrázek umístěn. K tomu použijeme tag *<svg>*, kterému můžeme jako klasickému HTML elementu nastavit šířku a výšku a tím určíme rozměry vykreslovacího plátna:

[6] https://github.com/d3/d3
[7] `https://github.com/d3/d3/wiki/Gallery`
[8] Scalable Vector Graphics

```
<svg width="250" height="250"></svg>
```

Dále je do plátna vykreslen kruh. Tentokrát už je ale využit speciální svg tag *<circle>*, kterému se nastaví průměr a souřadnice, kde se vykreslí. Lze také nastavit další atributy jako u klasického HTML elementu. Například barva, okraje, průhlednost a další. Atributy lze také definovat pomocí CSS třídy:

```
<svg width="250" height="250">
  <circle cx="20" cy="20" r="20" class="kruh"/>
</svg>
```

Další prvky se vykreslí stejným způsobem. Lze tedy přidat další kruhy, obdelníky, elipsy, čáry nebo text. Pomocí těchto prvků lze vytvořit složitější obrazce, sloupcové diagramy nebo například grafy, které se používají při zobrazení výsledků pro oba typy dotazů ve vyvíjené aplikaci.

3.2.2 Dynamicky vytvořený SVG obrázek

Vykreslit graf je možné i dynamicky a to v javascriptu za pomoci zmiňované knihovny D3:

```
1 var svg = d3.select("body").append("svg")
2             .attr("width", 250) .attr("height", 250);
3
4 var dataArray = [1,2,3,4,5];
5
6 svg.selectAll("circle")
7     .data(dataArray)
8     .enter()
9     .append("circle");
```

Na prvních řádcích kódu je vytvořeno plátno a přiřazeno do elementu *body*. Na řádku číslo 4 jsou připravena vstupní data [9].

Na 6. řádku vyhledáme všechny elementy typu *circle*. V tomto momentu žádné v HTML DOMu nejsou. Tudíž funkce vrátí prázdné reference. V následujícím řádku dojde k přečtení vstupních dat a přiřazení klíče každé položce v poli a následně je na 7 a 8 řádku tento klíč spárován s do nynějška neexistujícímu elementem *circle* a připojen k plátnu [20].

Těmto uzlům grafu je možné podobně jako na prvních řádcích přiřadit další atributy (pozice, barva...) nebo vynutit umístění popsané v kapitole 3.2.3.

3.2.3 Force layout

Force layout je funkce knihovny D3.js (nastaví typ vykreslování/uspořádání prvků v plátně), která umožňuje dále upřesnit chování vykresleného grafu. Lze

[9] Vstupní reálná data budou výsledky dotazů, tedy Objekty a Vazby.

nastavit specifické vlastnosti pro graf jako je například automatické pozicování způsobem, aby se uzly po vykreslení nebo přidání dalšího uzlu nepřekrývaly, ale automaticky vypočítaly své nové souřadnice. V grafu lze také uživateli umožnit manipulaci s uzly, proto je může libovolně přeskupovat. Obrázek tedy přestává být statickou reprezentací dat, ale stává se interaktivním nástrojem pro uživatele.

V následujících částech kódu jsou vysvětleny základní principy vytváření grafu pomocí Force layoutu:

```
var force = d3.layout.force()
    .linkDistance(100).charge(-500)
    .size([width, height]).on("tick", tickFunction);
```

Na prvním řádku je nastavení layoutu na typ force. Dále jsou nastaveny délky hran, „odpuzování“ uzlů (záporné číslo znamená, že se uzly odpuzují) a šířka a výška. Poslední funkce definuje reakci na událost „tick“. Je tedy definována funkce *tickFunction*, která se provádí jednou za určitý okamžik a nejčastěji je využita k překreslování, kdy se vypočítávají nové souřadnice uzlů a hran mezi sebou navzájem. Výsledkem je plynulá animace, ve které je vidět, jak se každý z uzlů přesouvá po plátně a snaží se získat pozici vyhovující všem ostatním dle počátečního nastavení.

Jelikož je tento typ layoutu specializován pro práci s grafem, poskytuje funkce pro definování uzlů i hran, jak je možné vidět v následujícím kódu:

```
force
    .nodes(jnodes).links(jlinks)
    .start();
```

Jnodes i *jlinks* jsou pole objektů ve formátu json. Vykreslování lze poté spustit pomocí funkce *start()*. Jakým způsobem budou uzly a hrany vypadat je možné nastavit stejně jako v kapitole 3.2.2.

Uzlům i hranám lze nastavit reakci na určitou událost:

```
node.on("click", function (currentNode) {...})
```

Například uzel může po kliknutí zobrazit detailnější informace, může dojít ke změně grafického desingu nebo odstranění dotyčného uzlu. Nejedná se však pouze o událost kliknutí, uzel/hrana může reagovat na „přejetí“ myší, „táhnutí“ nebo dvojité kliknutí.

Měnit lze i celkové chování Force layoutu. Například lze vykreslovaní po kliknutí zastavit:

```
d3.behavior.drag()
                .on("click", function(){force.stop()})
```

3.3 Neo4j

„Today's world is no longer driven by data – it's driven by the connections between them.“ [21]

Výše zmíněný citát popisuje současný stav, kdy největší roli nenesou data, ale právě propojení mezi nimi, čemuž se věnuje grafová databáze Neo4j.

Grafové databáze patří do skupiny NoSQL [10] databází a poskytují alternativu ke klasickým relačním databázím. NoSQL proto, protože jejich data nejsou uložena v klasické relační databázi a pro dotazování není použit jazyk SQL. Grafové databáze také nejsou pevně vázány principy ACID [11], jelikož na úkor konzistence poskytují větší výkon[22].

Jak už bylo zmíněno výše, grafová databáze nepoužívá jako uložiště dat tabulky relačních databází, ale data ukládá v podobě grafu (uzlů a hran). Vzhledem k využívané struktuře se tento typ databází využívá především pro vztahové problémy, kdy uživatel chce vědět, souvislosti mezi dvěma entitami (uzly) na rozdíl od relačních databází, které řeší spíše problémy agregační [23].

Neo4j je leader mezi grafovými databázemi [21] a Community verze pro nekomerční použití je dostupná pod licencí GPL.

V této práci je použita Communit verze 3.0.6. Pro komunikaci mezi javascriptem a databází byla použita knihovna *neo4j-driver* [12].

3.4 Konkurenční technologie

V této kapitole jsou zmíněny podobné (konkurenční) technologie, které také řeší danou problematiku. Blíže jsou také popsány další javascriptové knihovny a vizualizační nástroje pro zobrazení výsledků dotazů.

3.4.1 React

React je javascriptová knihovna sloužící pro tvorbu uživatelských rozhraní, opensourcovaná roku 2013 Facebookem [24]. React také řeší problém překreslování celého HTML DOMu umožňuje vykreslovat pouze potřebné části. Při tvorbě stránky dochází ke skládání jednotlivých komponent s vlastním stavem, až ve výsledku vznikne celá stránka. Výhodou tohoto frameworku je nezávislost na DOMu, ale pouze na svých komponentách [25].

3.4.2 Bobril

Jedná se o javascriptový framework napsaný Borisem Letochou, inspirovaný Reactem a Mithrilem [9]. Bobril tudíž pracuje s Virtuálním DOMem a s komponentami mající vlastní stav.

[10] Not only Structured Query Language

[11] Atomicita, Konzistence, Izolovanost, Trvalost

[12] Dostupná z `https://neo4j.com/developer/language-guides/`

Virtuální DOM je stromová reprezentace objektů na webové stránce (abstrakce HTML DOMu), jejíž změna vyvolá překreslení změněné části stránky (stromu) [26]. V aplikaci tedy existuje aktuální Virtuální DOM, který se při změně stavu některé z komponent porovná s nově vytvořeným Virtuálním DOMem. Pokud se navzájem liší, dojde v závislosti na rozsahu změny k překreslení části nebo celé stránky.

O změny stavů se stará bobflux pomocí akcí. Pro uživatele jsou vytvořeny různé akce, například kliknutí na tlačíko, které přidá položku do seznamu. Uživatel tedy klikne na tlačítko, to vyvolá akci a změní stav komponenty seznam. Dále dojde k porovnání Virtuálních DOMů a následně k překreslení, kdy se vytvoří nová položka seznamu.

Aplikace v Bobrilu se píší v TypeScriptu.

3.4.3 Sigmajs

V této kapitole bylo čerpáno ze zdroje [27].

Sigmajs je opensourcová javascriptová knihovna zabývající se vizualizací grafů. Stejně jako D3 dokáže vynutit uspořádání uzlů a hran, které se načítají opět pomocí formátu json. Pro vykreslení lze využít Canvas nebo WebGL render. Uživatel může interaktivně procházet graf a pomocí zoomu se zaměřit jen na určitou část grafu. Funkce jsou velmi podobné knihovně D3. Nejnovější verze je dostupná z githubu autora [13].

3.4.4 Alchemy

Alchemy je vykreslovací aplikace vytvořená knihovnou D3. Cílem je usnadnit uživateli vykreslování grafů a pomoci mu jednoduše zobrazit výsledek. Pro vykreslení grafu je pouze potřeba nakonfigurovat parametry chování grafu a ty pak předat konstruktoru objektu Alchemy.

V konfiguraci se nastaví všechny potřebné parametry. Spustit vykreslení lze i pouze se vstupními daty (uzly i hrany jsou v jednom souboru formátu json). Nastavit se však dájí i styly hran nebo uzlů, reakce na kliknutí nebo „táhnutí“ a mnoho dalších věcí, které lze nastavit v D3.js.

Pokud by nějaká funkcionalita v Alchemy chyběla, je možné ji pomocí D3.js doplnit a aplikaci tak rozšířit. Momentálně se na stránkách projektu na githubu[14] hledá nový udržovatel projektu.

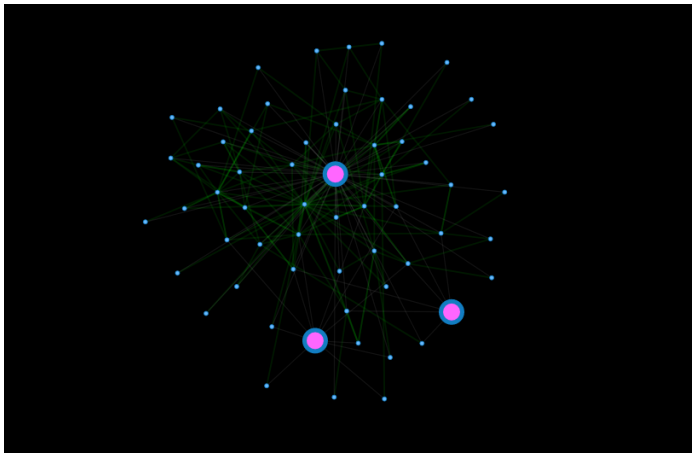
Tato kapitola čerpala ze zdroje [28]. Příklad grafu lze vidět na obrázku 3.4.

3.4.5 Cytoscape

Cytoscape je opensourcová platforma původně určená pro práci s biologickými daty. Například s interakcí mezi sítěmi molekul. V současnosti lze zobrazovat

[13] https://github.com/jacomyal/sigma.js/releases/

[14] Dostupné z https://github.com/GraphAlchemist/Alchemy



Obrázek 3.4: Vizualizace grafu: Alchemy [29]

ale i nejrůznější sítě a grafy. Cytoscape je možné použít jako aplikaci, ale dostupná je i jako javascriptová knihovna, kterou lze jednoduše naimportovat do projektu.

Po importu (podobně jako u předchozí knihovny) se vytvoří konfigurační soubor (json), který se předá kontruktoru. V konfiguraci se opět nastaví uzly, hrany, styly a element, na kterém se má graf vykreslit. Po vytoření grafu jsou možné další úpravy, jako například přidávání dalších uzlů. Příklad grafu je možné vidět na obrázku 3.5.

Kapitola čerpala ze zdroje [30].

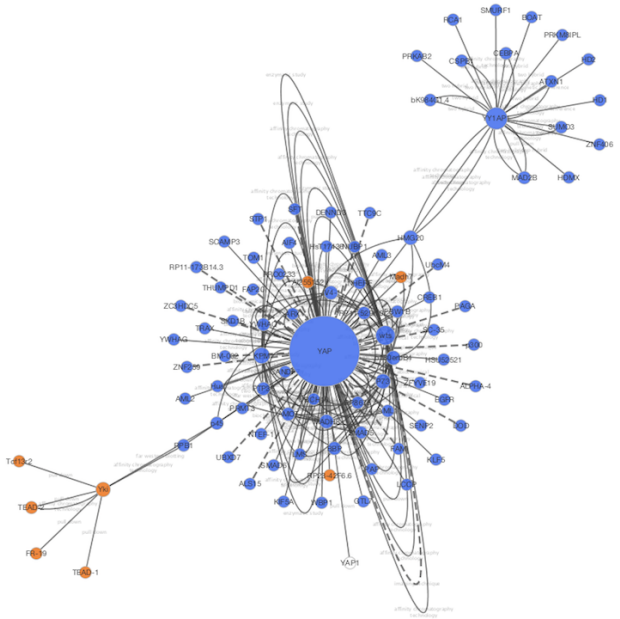
3.4.6 Linkurious

Jedná se interaktivní webový nástroj, který je napsaný s využitím knihovny Sigma.js [32], tudíž má podobné vlastnosti. Stejně jako Cytoscape se jedná o celou aplikaci umožňující pracovat s grafem, dokonce je možné ji propojit s databází Neo4j.

Kromě aplikace je dostupná i knihovna Linkurious.js, která je jednou z vývojových větví Sigma.js. V současnosti se ale jedná o zastaralou knihovnu a Linkurious vydalo novou knihovnu Ogma [33] zmíněnou níže 3.4.7.

3.4.7 Ogma

Ogma je javascriptová knihovna pro vizualizaci grafů vyvinutá společnoustí Linkurious. Vykreslování je založeno (stejně jako u knihovny Sigma.js) na enginu WebGL, ale podporuje i HTML Canvas. Stejně jako ostatní knihovny umožňuje interagovat s grafem (pohybovat s uzly, přidávat uzly, vynutit příslušný layout). Ogma se však pyšní vlastností zobrazit oproti ostatním vizualizačním knihovnám více než 100000 uzlů a 100000 hran. Vzhledem k množství



Obrázek 3.5: Vizualizace grafu: Cytoscape [31]

dat se snaží nezacházet do přílišných detailů a uživateli zobrazit jen potřebné informace. Tudíž při zobrazení několik desítek tisíc uzlů není nutné zobrazovat jejich popis.

Data lze importovat v různých formátech nebo se připojit rovnou do databáze Neo4j. Zobrazené výsledky je také možné i exporovat v podobě SVG, PNG a dalších.

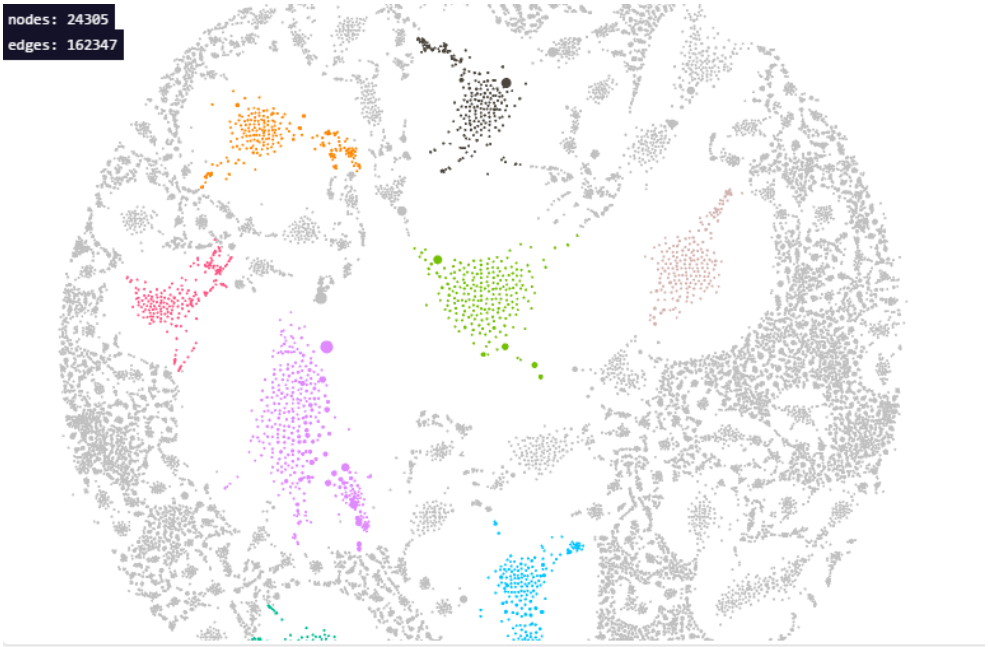
Jedná se ale o proprietární software, kdy poskytovatel upravuje používání softwaru na základě podepsané smlouvy.

Ukázku je možné si prohlédnout na obrázku 3.6, dostupného ze zdroje [34], z něhož čerpala i tato kapitola.

3.5 Důvod zvolených technonologii

3.5.1 Neo4j

Jako uložiště dat byla zvolena databáze Neo4j. Jedná se o databázi použitou i v předešlých pracích a k dotazování využívá jazyk Cypher, do kterého se má výsledný dotaz překládat. Navíc je Neo4j leader v grafových databázích jak bylo zmíněno výše 3.3, proto není důvod tuto technologii měnit.



Obrázek 3.6: Vizualizace grafu: Ogma

3.5.2 Angular

Jelikož aplikace pracuje s Javascriptem, bylo vhodné zvolit nějaký framework, který usnadní vývoj, namísto selektování a vkládání HTML elementů ze samotného Javascriptu.

Angular byl zvolen, jelikož jeho funkce a možnosti jsou pro aplikaci dostačující, navíc autor s ním má zkušenosti na rozdíl od ostatních frameworků.

3.5.3 D3.js

Pro vizualizaci výsledků dotazů bylo na výběr z mnoha možností. Nakonec byla zvolena knihovna D3.js, jelikož nabízí mnoho způsobů vizualizace a to nejenom grafů (pro případné grafické rozšíření systému).

Protože se nejedná o žádnou aplikaci s naprogramovanými funkcemi, ale o open source knihovnu, tak je možné ji dále rozšiřovat nebo přizbůsobit ji pro specifické chování. Knihovna má také velké množství ukázkových řešení a příkladů (viz. GitHub [15]), kterými je možné se inspirovat při tvorbě vlastního řešení.

Knihovna D3.js je také použita pro vizualizaci dat přímo v Neo4j Browseru [35]. Ten slouží jako primární dotazovací nástroj pro databázi Neo4j.

[15] `https://github.com/d3/d3/wiki/Gallery`

Kapitola 4

Realizace

Kapitola realizace se zabývá implementací softwarového systému. Nejprve je popsán import dat a vytvoření databáze Neo4j, dále pak jednotlivé typy dotazování a i princip vytváření a překlad dotazů. Uveden je i způsob zobrazení výsledků a ukládání. V poslední kapitole je zmíněno přihlašování do aplikace.

4.1 Import dat do databáze Neo4j

Před samotnou realizací aplikace musí být nejprve získána a upravena vstupní data, se kterými aplikace pracuje. V práci „Vizualizace vztahů v informační bázi MBI“ [2] byla data obdržena ve formátu XML a za pomocí XSLT transformace převedena na CSV soubory, které bylo potřeba ještě přeindexovat. Pro tento systém není nutná transformace, protože data jsou dodána z MBI v podobě CSV souborů (soubor pro Uzly a druhý pro Vazby).

Nejprve je zpracován soubor s uzly, kdy se každému uzlu přiřadí unikátní identifikátor. Výsledek je nový soubor *reindexNodes.csv*.

Další část programu zpracuje tento soubor a vytvoří z něj v paměti pole uzlů. Jelikož dle návrhu schématu datového uložiště (viz obrázek 2.2) vznikly Modelové uzly (uzly typu M), které nejsou obsaženy v obdržených CSV souborech, je potřeba je nyní vytvořit. Z pole jsou tedy vybrány zástupci za každý typ uzlů a vytvořeny Modelové uzly pouze s typem a identifikátorem (ostatní atributy jsou prázdné).

Nyní jsou k dispozici všechny uzly a je tedy možné začít vytvářet hrany. Celkem je potřeba vytvořit tři typy hran, které budou společně uloženy v poli.

První typ hran jsou hrany propojující instance Objektů mezi sebou (II). Projdou se tedy všechny hrany v CSV souboru, porovnají se s uzly a dojde k přeindexaci. Výsledek se uloží do pole hran. Druhým typem jsou hrany hierarchické (IH), kdy se pomocí cyklů vytvoří každému nadřazenému Objektu potomci. Posledním typem jsou modelové hrany (IM) vytvořené na základě průchodu Modelových a ostatních uzlů. Všechny hrany jsou uloženy v souboru *relations.csv*.

Jelikož přeindexace využívá programovacího jazyka java a vzhledem k možnosti využití knihovny neo4j (zmiňované v kapitole 2.1.3) pro import dat do databáze, je dána přednost tomuto způsobu na místo použití programu Batch importer. Ten byl sice použit v předchozí práci, ale pro nižší verzi databáze Neo4j. Pomocí nového způsobu nebude nutné spouštět jiný program. Navíc je potřeba vytvořit v databázi ještě další typ uzlu a to profil administrátora.

Následně tedy program na zadané cestě (pomocí parametru) vytvoří databázi. Pro jistotu zkusí smazat uzly i hrany, které by mohly v databázi existovat. Potom projde pole uzlů a pro každý uzel vytvoří jeden objekt v databázi s uloženými atributy. Jediná výjimka se týká atributu SUPERIOR_OBJECT, který oproti návrhu nebude v objektu obsažen, jelikož byl nahrazen hranou IM.

Hrany jsou vytvořeny podobným způsobem, jen u každé z nich musí být uložen odkaz na počáteční a koncový uzel. Na závěr je vytvořen jeden uzel typu Profil jako administrátor. Program se spustí následujícím příkazem:

```
java -jar ./Convertor.jar ./MBInodes.csv ./MBIrelations.
    csv C:/databaze
```

Prvním parametrem je soubor s uzly, druhý s hranami a třetí udává místo, kde se vytvoří databáze.

4.2 Specificky orientované dotazovaní

Specificky orientované dotazování je první možnost dotazování, kdy uživatel vybírá hodnoty ze tří grafických prvků Seznam. Po načtení stránky se v prvním z nich zobrazí všechny možné typy Objektů, skrze dotázání se do databáze pomocí dotazovacího jazyka Cypher:

```
MATCH (n) WHERE NOT n:savedQuery AND NOT n:Profile
RETURN distinct labels(n)
```

Dotaz vrátí všechny typy uzlů bez profilů a uložených dotazů. Tyto typy jsou pak pomocí callbacku [16] uloženy do proměnné a pomocí Angularu rovnou vypsány do Seznamu.

Ve chvíli, kdy uživatel vybere hodnotu z prvního Seznamu (klikne na jednu z položek), dojde opět k volání do databáze tentokrát na základě vybraného Objektu. Zpět se vrátí dostupné atributy daného Objektu a uloží se do druhého Seznamu pomocí callbacku.

Třetí Seznam funguje na stejném principu. Po kliknutí na konkrétní hodnotu atributu daného Objektu dojde k připravení Cypher dotazu, který lze dále uložit nebo zobrazit.

[16] Callback je funkce, která je předána jako parametr jíné fukci a její kód může být vykonán až později. Například volání do databáze je asynchronní, tudíž se na toto volání „nečeká“. Callback tedy zajistí, že funkce je zavolána v okamžiku, kdy už jsou dostupná data z databáze.

4.3 Dotazování pomocí vazeb

Tato podkapitola je rozdělena na dvě části. V té první je vysvětlen základní princip dotazování a v té druhé popsáno přeložení dotazu do jazyka Cypher.

4.3.1 Princip dotazování

Druhý typ dotazování využívá dvou Seznamů, které pracují na stejném principu popsaném výše. Z prvního Seznamu (OBJEKT) se Objekty vybírají a přidávají do druhého (VYBRANNÉ OBJEKTY). Při výběru (dvojité kliknutí) se opět zavolá Cypher dotaz do databáze, kde se vyhledají všechny navazující typy Objektů, které se pomocí callbacku uloží do prvního Seznamu a vykreslí Angularem. Tímto způsobem je zaručeno, že uživatel má na výběr vždy jen dostupné Objekty, které obsahují Vazbu k poslednímu vybranému.

Při mazání Objektu z druhého Seznamu (mazat lze pouze poslední prvek) je automaticky aktualizován i první Seznam.

Pro každý vybraný Objekt nebo Vazbu (dohromady tvoří graf) lze v dolní části okna blíže specifikovat jejich atributy, u kterých lze také zvolit reálné hodnoty získáné z databáze. To znamená, že pro každý dotaz na prvek grafu je zavoláno několik dalších Cypher dotazů pro jednotlivé atributy.

Celý graf se automaticky po přidání Objektu překreslí. Další variantou do budoucna je vytvoření cache pro graf, aby bylo zabráněno příliš častému volání do databáze. Namísto toho bude graf uložen v paměti a aktualizován pouze pro poslední přidaný Objekt a jeho Vazbu.

4.3.2 Přeložení dotazu

Dotaz vytvořený v jazyce MBIQL je uložen do následující struktury:

```
Graf = [Uzel, Vztah, Uzel...]

Uzel = {
    "name": "String",
    "type": "String",
    "attributes": [Atribut],
    "returnChecked": "Boolean"
}

Vztah = {
    "name": "String",
    "type": "String",
    "nodeStart": "String",
    "nodeEnd": "String",
    "attributes": [Atribut]
}
```

```
Atribut = {
    "name": "String",
    "values": [],
    "selected": "String"
}
```

Všechny Uzly a Vztahy jsou uloženy do jednoho pole (*Graf*). U Uzlu jsou zaznamenány tyto vlastnosti:

- **name** - V tomto případě se jedná o typ (název) Objektu, který uživatel vybral a který mu je zobrazen v aplikaci. Například TASK, ROLE,...
- **type** - Rozlišení, zda se jedná o Objekt nebo Vazbu. Na základě této vlastnosti je použito příslušné grafické zvýraznění.
- **attributes** - Pole atributů Objektu.
- **returnChecked** - Příznak, zda uživatel chce daný Objekt zobrazit ve výsledcích.

U Vztahu jsou vlastnosti podobné:

- **name** - Jméno Vazby zobrazené uživateli. Momentálně mají všechny Vazby stejné jméno, ale v budoucnu může dojít k rozšíření, kdy například jméno Vazby bude dáno nějakým atributem.
- **type** - Rozlišení, zda se jedná o Objekt nebo Vazbu.
- **nodeStart** - Počáteční Objekt dané Vazby.
- **nodeEnd** - Koncový Objekt dané Vazby.
- **attributes** - Pole atributů Vazby.

Vlastnosti Atributu:

- **name** - Jméno daného atributu: CODE, RACI,...
- **values** - Pole hodnot atributu
- **selected** - Uživatelem vybraná hodnota atributu.

Dotaz do jazyka Cypher z MBIQL se postupně sestavuje při průchodu polem *Graf*. U každého Uzlu i Vztahu se sestavuje zvlášť pro každou klauzuli: MATCH, WHERE, RETURN.

V klauzuli MATCH se naspecifikuje, se kterými Uzly a Vztahy se pracuje a jakým způsobem jsou spolu navzájem propojeny. Podle zadané struktury se dále hledá vhodná část grafu. Jedná se tedy o pravidla, které Uzly a Vztahy musí splňovat. Jelikož výsledek dotazu je souvislý graf, je i klauzule pro překlad

postupně řetězena až ve výsledku zobrazí celou strukturu grafu. Příklad je vidět na následujícím kódu:

```
MATCH (uzel1: Typ)-[vztah1]-(uzel2: Typ)-[vztah2]
     -(uzel3: Typ)-[vztah3]-(uzel4: Typ)-...
```

Postupně, jak se prochází *Graf*, přidává se do klauzule pro každý Objekt i Vazbu její název a typ. Výjimkou jsou Modelové uzly. Pokud je tato možnost zvolena, tak jsou přidány až nakonec.

Do klauzule WHERE se přidávají podmínky. Každému Uzlu nebo Vztahu, obsažených v klauzuli MATCH, je možné přiřadit hodnotu nějakého atributu. Tuto hodnotu vybírá uživatel při tvorbě dotazu a je uložena do vlastnosti *Atributu selected*. V případě prázdného řetězce se podmínka nepřidá. Příklad je zobrazen v následujícím kódu:

```
WHERE uzel1.atribut1 = hodnota1 AND vztah1.atribut2 = hodnota2
      AND uzel2.atribut1 = hodnota3
```

Klauzule RETURN popisuje výsledek dotazu. Tedy to, co se uživateli vrátí v odpovědi. Pomocí příznaku vlastnosti *returnChecked* je Uzel označen v případě, že se má zobrazit. Vztahy tento příznak nemají, protože sami o sobě nemůžou být zobrazeny. Vztah je tudíž zobrazen vždy mezi dvěma zobrazenými Uzly. Pokud uživatel nevybere žádný Uzel k zobrazení, je upozorněn, že nic nevybral a že takový dotaz nemá smysl.

Po naplnění všech klauzulí je výsledek složen do jednoho dotazu, který je následně uložen nebo zobrazen uživateli. Příklad celého dotazu:

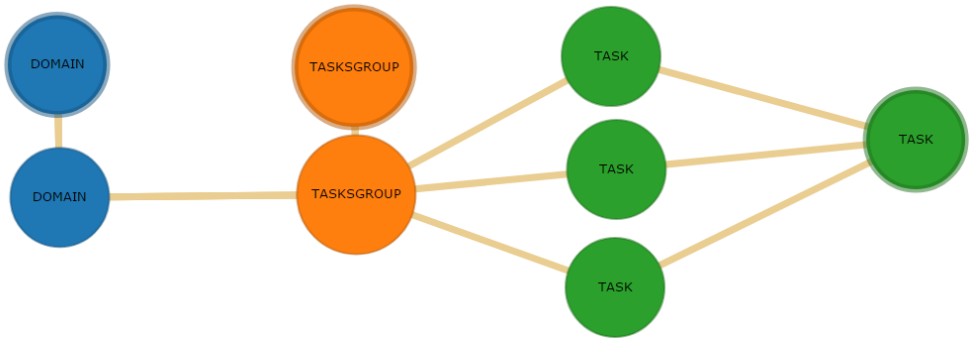
```
MATCH (uzel1: Typ)-[vztah1]-(uzel2: Typ)
WHERE uzel1.atribut1 = hodnota1 AND vztah1.atribut2 = hodnota2
RETURN uzel1, uzel2
```

Překlad z MBIQL do jazyka Cypher je ověřen na vzorových dotazech z kapitoly 2.2.3 níže v kapitole 5.

4.4 Zobrazení výsledků

Výsledky dotazů jsou vizualizovány v dolní části okna a zobrazí se po stisknutí tlačítka *Zobrazit*. Stisknutím tlačítka je zavolán Cypher dotaz do databáze s nadefinovaným dotazem. Jako výsledek se vrátí (ve formátu json) nalezené Uzly a Vztahy, které musí bý mezi sebou odděleny do dvou polí. Jelikož se ve výsledku nacházejí redundatní informace, při oddělování dochází ještě k drobným úpravám.

Do prvního pole jsou uloženy Uzly s identifikátorem (globální id). V druhém jsou Vztahy. Jednotlivé položky pole, ale musejí obsahovat atribut *source* a *target* s identifikátory Uzlů, kterým Vztah náleží. Pro každý Vztah je tedy



Obrázek 4.1: Vizualizace grafu: MBIQL

potřeba nalézt přišlušné identifikátory. Po naplnění obou dvou polí je možné tyto pole předat jako parametr pro knihovnu D3, která z nich vytvoří svg elemenety pro uzly a hrany grafu.

Dále je také potřeba nadefinovat grafický vzhled pomocí stylů a reagování systému na různé události. Nastaveno je například chování na „táhnutí“ myší, kdy dojde k zastavení automatického překreslování uzlů. To se znovu obnoví až uživatel uzel „pustí“.

Na dvojité kliknutí na uzel dojde k dotázání do databáze na všechny sousedy daného uzlu. Po vrácení výsledků dojde k aktualizaci obou dvou polí a vykreslení změn.

Uživatel také může kliknutím na *Smazat* vymazat všechny aktuálně zobrazené výsledky. To znamená z polí odstranit všechny uzly a hrany a odstranit také všechny k nim vykreslené svg elementy.

Pro zvětšení zobrazovací plochy (stisknutí tlačítka *Zvětšit*) dojde k přepnutí mezi styly, kdy je nahrazena výška okna.

Výsledek dotazování je vidět na obrázku 4.1.

4.5 Uložiště dotazů a profily

Stejně jako Objekty a Vztahy jsou také v databázi uloženy Profily a Dotazy. Každý Profil pod sebou může obsahovat různé Dotazy jak bylo uvedeno v kapitole 2.4.3.1. V této části systému se opět volá do databáze a pro uložení dat se využívají z callbacky. Změna oproti jiným částem je ale ve využití cache, která byla popsána v kapitole 2.4.2. Cache je implementována jako objekt, na který si ostatní části systému, které ji využívají, drží referenci.

4.6 Přihlašování

Pro přihlášení uživatele jsou využity přihlašovací údaje do databáze Neo4j. Při potvrzení uživatelského jména i hesla je proveden pokus o vytvoření spojení do databáze. V případě úspěchu je uživatel zalogován do systému. V případě neúspěchu je vyzván authentizaci opakovat.

Kapitola 5

Ověření

V této kapitole jsou provedeny testy, které ověřují kvalitu vyvíjené aplikace. Nejprve jsou provedeny datové testy, kontrolující správnost. Dále proběhlo ověření vzorových dotazů z kapitoly 2.2.3. V předposlední kapitole je aplikace otestována pomocí Nielsenovy heuristické analýzy a v závěru je pak otestována samotnými uživateli.

5.1 Import dat do databáze

V ověření importu dat do databáze se kontroluje, zda žádná importovaná data nebyla ztracena. Porovnává se počet importovaných uzlů a hran s počtem skutečných dat obsažených v databázi. To znamená, že výsledky obdržené v průběhu vytváření uzlů a hran by měly odpovídat výsledkům Cypher dotazů na dané prvky v databázi.

```
1632 nodes were created
29 nodes of type M were created
12108 relations II were created
1545 relations IH were created
1632 relations IM were created
Admin profile has been created
```

```
MATCH ()-[r:INSTANCE_HIERARCHY]->() RETURN count(r)        1545
MATCH ()-[r:INSTANCE_MODEL]->() RETURN count(r)            1632
MATCH ()-[r]->() WHERE r.‘EDGE TYPE‘="II" RETURN count(r) 12105

MATCH (n) RETURN count(n)                                  1662
```

Z předchozích výsledků lze vidět, že hodnoty u všech prvků kromě hran typu II odpovídají (uzly byly sečteny dohromady: 1632 + 29 + 1). Příčina tří ztracených hran, které se nenaimportovaly do databáze Neo4j, byla dohledána.

Bylo zjištěno, že v původních CSV souborech nejsou uzly k těmto hranám dodány, tudíž tyto hrany s chybějícími uzly nemohly být vytvořeny. Tento nález bude nahlášen členům MBI. Import dat do databáze Neo4j tedy proběhl správně.

5.2 Vzorové dotazy

V této kapitole jsou ověřeny vzorové dotazy z kapitoly 2.2.3. Při vytváření dotazů pomocí jazyka MBIQL je pro každý dotaz odsimulováno chování uživatele a následně je proveden překlad tohoto dotazu do jazyka Cypher. Zkontrolovány jsou také zobrazené výsledky s uloženými vstupními daty z databáze.

1. *Zobraz instanci objektu Úloha, která má název: Správa IT infrastruktury.*

   Tento dotaz je typickým zástupcem Specificky orientovaného dotazování. Uživatel tedy vybere tuto záložku. Z prvního Seznamu Objekty vybere typ Úloha (TASK). Z druhého Seznamu vybere atribut Název (NAME) a ze třetího už konkrétní hodnotu: Správa infrastruktury. Potom nechá výsledek zobrazit nebo uložit. Z vybraných parametrů je poskládán Cypher dotaz:

   ```
   MATCH (node:‘TASK‘)
   WHERE node.‘NAME‘ = "Správa IT infrastruktury"
   RETURN node
   ```

   Výsledkem je jeden uzel s názvem: Správa IT infrastruktury. *(Správně)*

2. *Zobraz všechny Metody, které mají vztah k Úloze: Propojení metrik byznysu a metrik IT.*

   Tento i zbylé dotazy používají metodu Dotazování pomocí vazeb. Uživatel tedy vybere ze záložek tuto metodu.

   Z prvního Seznamu dvakrát klikne na Objekt Metoda (METHOD). Metoda se zobrazí v druhém Seznamu a první se zaktualizuje, tak že nabízí pouze Objekty, které sousedí s Objektem typu Metoda. Uživatel tedy dvojitým kliknutím vybere Úlohu (TASK). Nyní dochází ke specifikaci dotazu, kdy uživatel zobrazí nabízené atributy vybraných Objektů, které chce blíže specifikovat. V tomto případě rozklikne Objekt TASK a vybere u atributu Název (NAME) „Propojení metrik byznysu a metrik IT“ a odškrtne příznak ZOBRAZIT, protože tento Objekt nechce ve výsledku zobrazovat. Naopak u Metody zkontroluje zaškrtnutí, protože tento typ Objektu ho zajímá.

   Jelikož uživatel pracuje se dvěmi souvisejícími Objekty, které mají společnou Vazbu, vytvoří se pro klauzuli MATCH oba dva uzly a hrana.

V klauzuli WHERE je nastavený atribut Název a v klauzuli RETURN je pak Objekt typu Metoda, který uživatel chce zobrazit:

```
MATCH (n0:METHOD)-[r1]-(n2:TASK)
WHERE n2.‘NAME‘ = "Propojení metrik byznysu a metrik IT"
RETURN n0
```

Výsledkem jsou 4 uzly typu METHOD: MIT Sloan, Total Cost of Ownership, ABC, CPM. *(Správně)*

3. *Zobraz všechny Skupiny úloh, které spadají do Domény: Strategické řízení IT.*

   Uživatel z prvního Seznamu vybere Doménu (DOMAIN) a po aktualizování má jedinou možnost vybrat jen Skupiny úloh (TASKGROUP), protože Domény jsou nadřazené Objekty pouze Skupinám úloh. Dále nastaví atribut Název (NAME) u Domény na „Strategické řízení IT“. Tento příklad je velmi podobný předchozímu, vybrány jsou jen jiné Objekty a hodnoty. Nastaven je také Příznak pro zobrazení u obou Objektů. Proto jsou ve výsledku dotazu zobrazeny i hrany. Přeložený Cypher dotaz:

   ```
   MATCH (n0:DOMAIN)-[r1]-(n2:TASKSGROUP)
   WHERE n0.‘NAME‘ = "Strategické řízení IT"
   RETURN n0, r1, n2
   ```

   Výsledkem je 6 uzlů Skupin úloh připojené hranami k Doméně. *(Správně)*

4. *Najdi pouze Faktory, které jsou využity u Scénáře: CIO se připravuje na poradu vedení podniku.*

   Uživatel si může vybrat od kterého Objektu chce začít, protože hrany mezi uzly při dotazování nemají přiřazen směr. Vybere tedy například Scénář (SCENARIO). Mezi dalšími Objekty ze Seznamu ale není na výběr Faktor (FACTOR), protože Scénař s ním přímo nesousedí. Je tedy nutné vybrat jiný Objekt, přes který se k Faktoru lze dostat. Ze znalosti MBI nebo z úvodního obrázku 1.1 je potřeba vybrat Objekt Úloha (TASK). Po té je už dostupný i Faktor. Při speficikaci dotazu se nastaví Název (NAME) u Scénáře na „CIO se připravuje na poradu vedení podniku.“ Jelikož mezi zobrazenými výsledky mají být pouze Faktory, Příznak u ostatních se odškrtne. Přeložený Cypher dotaz:

   ```
   MATCH (n0:SCENARIO)-[r1]-(n2:TASK)-[r3]-(n4:FACTOR)
   WHERE n0.‘NAME‘="CIO se připravuje na poradu vedení podniku"
   RETURN n4
   ```

Výsledkem je 24 Faktorů. *(Správně)*

5. *Zjisti, do jaké Skupiny metrik patří Metrika: Počty spravovaných technických postředků. Zobraz všechny hrany a uzly.*

   Dotaz je podobný jako předchozí. Vybrány jsou ale jiné prvky a hodnoty. Ve výsledku jsou také zobrazeny všechny prvky. Přeložený Cypher dotaz:

```
MATCH (n0:METRICGROUP)-[r1]-(n2:METRICSUBGROUP)-[r3]-
      (n4:METRIC)
WHERE n4.‘NAME‘="Počty spravovaných technických prostředků"
RETURN n0, r1, n2, r3, n4
```

   Výsledkem je hierarchie 3 uzlů: Metrika, podskupina Metrik a skupina Metrik. *(Správně)*

6. *Zjisti, do jaké Skupiny metrik patří Metrika: Počty spravovaných technických postředků. Zobraz všechny hrany a uzly. Zobraz také uzly prvků modelu MBI (Uzly typu M).*

   Tento dotaz je uplně stejný jako dotaz číslo 5 s tím rozdílem, že mají být zobrazeny i Modelové uzly. Uživatel tedy musí zaškrnout i Příznak pro zobrazení Modelových uzlů. Při překladu dotazu do jazyka Cypher je tedy potřeba přidat další vazbu a uzel pro každý Objekt. Ty se do klauzule MATCH přidají za vytvořenou strukturu dotazu. Přidat se musejí i do klauzule RETURN:

```
MATCH (n0:METRICGROUP)-[r1]-(n2:METRICSUBGROUP)-[r3]-
      (n4:METRIC), (n0)-[rM0:INSTANCE_MODEL]-(nM0),
      (n2)-[rM2:INSTANCE_MODEL]-(nM2),
      (n4)-[rM4:INSTANCE_MODEL]-(nM4)
WHERE n4.‘NAME‘="Počty spravovaných technických prostředků"
RETURN n0,rM0 ,nM0, r1, n2,rM2 ,nM2, r3, n4,rM4 ,nM4
```

   Výsledkem je hierarchie 3 uzlů jako u předchozího dotazu, ke kterým jsou navíc přiřazeny ještě další 3 uzly - Modelové. *(Správně)*

5.3 Nielsenova heuristická analýza

Nielsenova analýza pomůže otestovat systém bez přítomnosti uživatelů a to pomocí deseti pravidel, které kontrolují dodržení použitelnosti a ovladatelnosti systému [36]. V případě nedodržení některého z nich bude systém na základě daného pravidla upraven.

1. *Viditelnost stavu systému*

   Při práci se systémem by měl uživatel vědět, co v danou chvíli systém dělá. To znamená, zda například nečeká na data nebo zda se nevykreslují výsledky. Signalizuje tak, že je potřeba počkat na dokončení operace. V sytému by tedy chtělo nějakým způsobem toto chování zvýraznit. *(Upravit)*

2. *Shoda mezi systémem a realitou*

   Prvky v systému fungují tak jak uživatel předpokládá. Například mazaní je znázorněno křížkem nebo nastavení ozubeným kolečkem. *(OK)*

3. *Uživatelská kontrola a svoboda*

   Vzhledem k velikosti systému nejsou nutná tlačítka *zpět* nebo *dopředu*. Uživatel by měl ale být kontrolován a v případě pokusu smazání dotazu vyzván, zda si opravdu přeje dotaz smazat. V případě mazání a přidávání profilů má možnost změny smazat a nechat původní nastavení pomocí tlačítka *Cancel*. *(Upravit)*

4. *Shoda s použitou platformou a obecnými standardy*

   Systém odpovídá webové aplikaci a chová se konzistentně. Tlačítka pro zobrazení výsledků vypadají stejně v sekci uložiště i na hlavním stránce. Bude ale potřeba upravit barvu (na červenou) tlačítka pro mazaní profilů, které je narozdíl od tlačítka pro smazání dotazů, nevybarvené. *(Upravit)*

5. *Prevence chyb*

   Aby bylo zabráněno chybám, měl by být uživatel upozorněn v případě, že se pokusí vykonat operaci, při které by mohl ztratit data. To samé platí i při úpravě uživatelských profilů. Uživatel by měl být také upozorněn v případě nevyplnění povinného pole názvu dotazu. *(Upravit)*

6. *Rozpoznávání namísto vzpomínání*

   Systém je intuitivní a umožňuje uživateli snadno dosáhnout cíle. Například při prohledávání uložených dotazů lze vybrat pouze profil, který uživatele nejvíce zajímá a není obtěžován ostatními dotazy. *(OK)*

7. *Flexibilní a efektivní použití*

   Systém je jednoduchý v používání, ale přitom poskytuje zkušenějším uživatelům pracovat se složitějšími dotazy. *(OK)*

8. *Minimalita*

   Uživatel není zatěžován zbytečnými informacemi: Uložené dotazy lze skrýt, stejně tak zobrazené výsledky je možné schovat. V případě úpravy

profilů je zobrazeno modální okno a uživatel v tu chvíli nemůže vytvářet nebo ukládat dotazy - k dispozici má pouze úpravu profilů. *(OK)*

9. *Smysluplné chybové hlášky*

   Systém obsahuje smyslupné chybové hlášky, které uživateli chyby vysvětlí a navedou ho na možné řešení. Například při zadání duplicitního názvu dotazu se zobrazí text: „Dotaz s tímto názvem již v systému existuje. Zvolte jiný název.“ *(OK)*

10. *Nápověda a dokumentace*

    K systému vznikne i uživatelská příručka, která bude popisovat a vysvětlovat chování systému a sloužit jako manuál pro nové uživatele. *(OK)*

5.4 Uživatelské testování

Pro testování s uživateli byl připraven následující scénář. Uživateli je nejprve představena aplikace a její základní funkce. Testovací scénář začíná na uvodní přihlašovací stránce aplikace.

1. Přihlaste se do aplikace pomocí uživatelského jména **neo4j** a hesla **pass**.
2. Pomocí specificky orientovaného dotazování vyberte objekt **Role** (ROLE) s **Názvem** (NAME) **Konstruktér**. Zobrazte tento dotaz.
3. Pomocí specificky orientovaného dotazování vyberte objekt **Aplikace** (Application) s **Kódem** (Code) **AQ037A**. Uložte tento dotaz pod názvem **Dotaz1**.
4. Najděte tento dotaz v uložišti dotazů.
5. Přes dotazování pomocí vazeb zobrazte všechny **Role** (ROLE), které souvisí **Úlohou** (TASK) s **Názvem** (NAME) **Evidence dopravy**. Zobrazte pouze tyto Role.
6. Smažte obsah vykreslovacího plátna.
7. Otevřete nastavení pro správu profilů a vytvořte 2 nové profily: **Nováček** a **Expert**.
8. Zvolte profil **Expert**.
9. Zjistěte, do jaké Skupiny metrik patří Metrika: **Počty spravovaných technických postředků**. Zobrazte všechny hrany a uzly. Zobrazte také uzly prvků modelu MBI (Uzly typu M).
10. Smažte obsah vykreslovacího plátna.

11. Najděte v uložišti dotazů dotaz s názvem **Dotaz20**, který je uložen pod profilem **admin**.

12. Zobrazte ho a zvětšete vykreslovací plátno. Potom tento dotaz smazejte.

Kapitola 6

Zhodnocení a nasazení

V následující kapitole je provedeno zhodnocení testů a celého systému. Druhá část popisuje představení systému členům MBI a proces nasazení.

6.1 Zhodnocení

V rámci testování byly provedeny testy jak bez uživatelů, tak následně i s uživateli. Nejprve tedy byla zkontrolována správnost dat při vytváření datového uložiště v grafové databázi Neo4j. Dálé byly ověřeny vzorové dotazy a jejich správné vytvoření pomocí jazyka MBIQL, přeložení do jazyka Cypher i korektní zobrazení výsledků. V poslední části testování bez uživatelů byla provedena Nielsonova heuristická analýza, na jejímž základě byly provedeny následující úpravy systému:

1. Přidání loaderu („načítací kolečko“) v případě, že uživatel ukládá nebo zobrazuje data. Uživatel tedy má přehled o stavu systému. (*Řeší bod číslo 1*)

2. Při mazaní prvků ze systému se uživateli zobrazí hláška, která kontroluje, zda si uživatel opravdu přeje daná data smazat. (*Řeší bod číslo 3 a 5*)

3. Tlačítko pro smazání profilu bylo nahrazeno stejným tlačítkem, jako které je u smazání dotazu. (*Řeší bod číslo 4*)

4. Nevyplněné pole s názvem dotazu je po chybném zadání zvýrazněno červeně. (*Řeší bod číslo 5*)

V druhé části proběhlo uživatelské testování na základě zmiňovaného scénáře v kapitole 5.4, ze kterého vyplynul jeden požadavek na přidání možnosti upravovat styly zobrazovaných objektů. Uživatel by tedy měl mít například možnost zvolit typ textu, který se mu zobrazí v objektu nebo zvolit rozměry zobrazovaného uzlu.

Systém je tedy připraven pro veřejné používání a nabízí uživatelům alternativní možnost pro procházení objektů báze MBI interaktivní formou. Uživatelé mají také možnost dotazy ukládat a nejsou tak nuceni dotaz opakovaně vytvářet znovu.

6.2 Nasazení

Softwarový systém byl v závěru předveden tvůrci informační báze MBI panu doc. Ing. Janu Pourovi, CSc. Výsledkem byla pozitivní reakce, kdy si pan Pour systém pochvaloval a porovnával ho s předchozím řešením uvedeným v práci „Vizualizace vztahů v informační bázi MBI“ [2]. Konzultovaly se s ním i drobné úpravy systému. Zrušeno bylo například z důvodu bezpečnosti tlačítko pro prokliknutí do uživatelského rozhraní Neo4j.

Po domluvě bylo rozhodnuto nasadit tento systém na webový server Fakulty informačních technologií s pomocí vedoucího diplomové práce. Z důvodu jeho časového vytížení proběhne nasazení v příštích měsících. Do systému se bude možné přihlásit ze stránek MBI [17]. Systém pak bude podroben uživatelskému provozu, ze kterého vyplynou další náměty na případné úpravy a vylepšení.

[17] Dostupné z https://mbi.vse.cz

Závěr

Cílem diplomové práce bylo navrhnout a implementovat software pro procházení a dotazování objektů MBI. Pro tyto účely využívá systém jazyk MBIQL jehož návrh je uveden v bakalářské práci Ondřeje Batíka.

V první kapitole (viz 1) byl nastíněn současný stav problematiky a uvedeno řešení pomocí jazyka MBIQL. Z těchto uvodních podkapitol vzešly formální požadavky a také detailnější specifikace cíle na softwarový systém.

Ve druhé kapitole (viz 2) bylo výše zmíněné řešení zanalyzováno a následně zrevidováno. Na jeho základě se vytvořil grafický návrh jednotlivých částí systému včetně architektury i databáze. Pro importování bylo tedy potřeba upravit i program vytvořený v bakalářské práci „Vizualizace vztahů v informační bázi MBI“, aby výsledná databáze odpovídala novým změnám.

Ve třetí kapitole (viz 3) byly popsány technologie použité při vývoji softwarového systému-zejména tedy framework Angular pro tvorbu webových single page aplikací, knihovna D3.js využita pro procházení objektů a databáze Neo4j, ve které jsou uložena data z MBI. V kapitole byly také zmíněny další-konkurenční technologie, mezi kterými bylo před začátkem vývoje vybíráno.

Ve čtvrté kapitole (viz 4) byla provedena samotná realizace systému. Z obdržených dat od MBI v podobě CSV souborů byla vytvořena databáze Neo4j a naimplementovány jednotlivé části systému na základě předchozího návrhu: Oba dva typy dotazování, vykreslování a práce s výsledky, přihlašování, uložiště dotazů a správa profilů.

V páté kapitole (viz 5) proběhlo otestování kvality systému a v závěrečné kapitole(viz 6) pak celkové zhodnocení.

Cíl diplomové práce byl i se všemi formálními požadavky splněn a vytvořený systém může být pro uživatele zpřístupněn v příštích měsících. Tento systém poslouží jako prototyp pro interaktivní procházení objektů MBI, jak již bylo uvedeno v úvodu.

Za osobní přínos lze pokládat seznámení se s javascriptovým frameworkem AngularJS a knihovnou D3.js, díky které bylo možné vizualizovat data z MBI. Přínosem jsou také získané zkušenosti při práci na rozsáhlejším projektu.

Literatura

[1] Pour, J.: *MBI [online]*. [Cited 2017-5-4]. Dostupné z: `http://mbi.vse.cz`

[2] Stránský, V.: *Vizualizace vztahů v informační bázi MBI [online]*. [Cited 2017-5-4]. Dostupné z: `https://dip.felk.cvut.cz/browse/pdfcache/stranvo1_2014bach.pdf`

[3] Batík, O.: *Dotazovací jazyk pro vztahy MBI objektů*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

[4] MBI: *MBI, Management Byznys Informatiky - Koncepce a návod k použití [online]*. [Cited 2017-5-4]. Dostupné z: `http://mbi.vse.cz/mbi/file/help.pptx`

[5] Arlow, J.; Neustadt, I.: *UML2 a unifikovaný proces vývoje aplikací*. Brno: Computer Press, a.s., 2007, ISBN 978-80-251-1503-9.

[6] Sommerville, I.: *Software Engineering (6th edition)*. Harlow, England: Addison-Wesley., 2001, ISBN 0-201-39815-X.

[7] Jahoda, B.: *Single page applicationí [online]*. 2015, [Cited 2017-5-10]. Dostupné z: `http://jecas.cz/spa`

[8] Comerto: *Drátěný model webu (wireframe) [online]*. 2016, [Cited 2017-5-11]. Dostupné z: `https://www.comerto.com/tvorba-webovych-stranek-a-internetovych-obchodu/jak-tvorime-weby/drateny-model-webu-wireframe`

[9] Botelho, L.: *Learn Angular, The Essentials [online]*. 2017, [Cited 2017-5-13]. Dostupné z: `http://www.learn-angular.org/#!/lessons/the-essentials`

[10] Tutorials Point: *What is AngularJS? [online]*. 2017, [Cited 2017-5-13]. Dostupné z: https://www.tutorialspoint.com/angularjs/angularjs_overview.htm

[11] Google: *AngularJS, Developer Guide [online]*. 2017, [Cited 2017-5-13]. Dostupné z: https://docs.angularjs.org/guide

[12] Mrozek, J.: *Začínáme s AngularJS [online]*. 2012, [Cited 2017-5-14]. Dostupné z: https://www.zdrojak.cz/clanky/zaciname-s-angularjs/

[13] Google: *AngularJS, Data Binding [online]*. 2017, [Cited 2017-5-13]. Dostupné z: https://docs.angularjs.org/guide/databinding

[14] Bostock, M.: *Data-Driven Documents [online]*. 2017, [Cited 2017-5-15]. Dostupné z: https://d3js.org/

[15] Murray, S.: *Interactive Data Visualization for the Web, Introducing D3 [online]*. 2013, [Cited 2017-5-15]. Dostupné z: http://chimera.labs.oreilly.com/books/1230000000345/index.html

[16] Bostock, M.: *Chord Diagram [online]*. 2017, [Cited 2017-5-15]. Dostupné z: https://bl.ocks.org/mbostock/4062006

[17] Pasha: *US State Map [online]*. 2017, [Cited 2017-5-15]. Dostupné z: http://bl.ocks.org/NPashaP/a74faf20b492ad377312

[18] Rodden, K.: *Sequences sunburst [online]*. 2017, [Cited 2017-5-15]. Dostupné z: https://bl.ocks.org/kerryrodden/7090426

[19] Jahoda, B.: *SVG [online]*. 2016, [Cited 2017-5-15]. Dostupné z: http://jecas.cz/svg

[20] Knowledge Stockpile: *Understanding selectAll, data, enter, append sequence in D3.js [online]*. 2012, [Cited 2017-5-15]. Dostupné z: http://knowledgestockpile.blogspot.cz/2012/01/understanding-selectall-data-enter.html

[21] Neo4j: *Your Enterprise is Driven by Connections [online]*. 2017, [Cited 2017-5-16]. Dostupné z: https://neo4j.com/product/

[22] Ramba, J.: *Grafová terminologie a dostupné technologie [online]*. 2013, [Cited 2017-5-16]. Dostupné z: https://www.zdrojak.cz/clanky/grafova-terminologie-a-dostupne-technologie//

[23] Holý, J.: *Grafové databáze a neo4j [video]*. 2012, [Cited 2017-5-16]. Dostupné z: https://www.youtube.com/watch?v=lLuqIcYvWUQ

[24] Mikšů, V.: *React - Úvod [online]*. 2016, [Cited 2017-5-16]. Dostupné z: https://www.dzejes.cz/react-uvod.html

[25] Dostál, A.: *Vývoj webových aplikací: React a Angular 2 [online]*. 2016, [Cited 2017-5-16]. Dostupné z: http://www.aspectworks.com/2016/09/vyvoj-aplikaci-react-angular-2/

[26] Růt, T.: *Bobril - I - Getting Started [online]*. 2017, [Cited 2017-5-16]. Dostupné z: https://www.codeproject.com/Articles/1044425/Bobril-I-Getting-Started

[27] Jacomy, A.: *SigmaJS [online]*. 2016, [Cited 2017-5-16]. Dostupné z: https://github.com/jacomyal/sigma.js/wiki

[28] Hedinger, H.: *Alchemy [online]*. 2015, [Cited 2017-5-16]. Dostupné z: http://graphalchemist.github.io/Alchemy/#/docs

[29] Hedinger, H.: *Examples, Communication Data [online]*. 2015, [Cited 2017-5-18]. Dostupné z: http://graphalchemist.github.io/Alchemy/#/examples

[30] The Cytoscape Consortium: *What is Cytoscape? [online]*. 2017, [Cited 2017-5-18]. Dostupné z: http://www.cytoscape.org/what_is_cytoscape.html

[31] The Cytoscape Consortium: *Cytoscape User Manual [online]*. 2017, [Cited 2017-5-18]. Dostupné z: http://manual.cytoscape.org/en/stable/Cytoscape.js_and_Cytoscape.html

[32] Neo4j: *Graph Visualization for Neo4j [online]*. 2017, [Cited 2017-5-18]. Dostupné z: https://neo4j.com/developer/guide-data-visualization/

[33] Linkurious: *Linkurious.js [online]*. 2017, [Cited 2017-5-18]. Dostupné z: https://github.com/Linkurious/linkurious.js/

[34] Linkurious: *INTRODUCING OGMA, THE JAVASCRIPT LIBRARY FOR LARGE-SCALE GRAPH VISUALIZATION AND INTERACTION [online]*. 2017, [Cited 2017-5-18]. Dostupné z: https://linkurio.us/blog/ogma-js-library-large-scale-graph-visualization/

[35] Neo4j: *Graph Visualization for Neo4j, Screencast: The Neo4j Browser [online]*. 2017, [Cited 2017-5-18]. Dostupné z: https://neo4j.com/developer/guide-data-visualization/

[36] Svoboda, V.: *Nielsen's Heuristic Evaluation[online]*. 2011, [Cited 2017-6-3]. Dostupné z: http://blog.vojtasvoboda.cz/nielsens-heuristic-evaluation

Příloha **A**

Seznam použitých zkratek

MBI Management of Business Informatics

MBIQL Management of Business Informatics Query Language

XML eXtensible Markup Language

CSV Comma Separated Value

NoSQL Not Only Structured Query Language

JSON JavaScript Object Notation

HTML HyperText Markup Language

CSS Cascading Style Sheets

PHP PHP: Hypertext Preprocessor

SVG Scalable Vector Graphics

DOM Document Object Model

ACID Atomicity, Consistency, Isolation, Durability

XSLT eXtensible Stylesheet Language Transformations

II Instance-Instance

IH Instance-Hierarchy

IM Instance-Model

IT Information Technology

Příloha B

Konfigurační a instalační příručka

B.1 Systémové požadavky

- Java JRE.
- Doporučený operační systém Windows 7.

B.2 Konfigurace softwarového systému

1. Vytvořit databázi pomocí programu src/Instalace/Convertor/Convertor.jar:

   ```
   java -jar ./Convertor.jar MBInodes.csv MBIrelations
       .csv C:/databaze
   ```

2. Stáhnout a nainstalovat Neo4j (verze 3.0.6) z `https://neo4j.com/`.
3. Spustit Neo4j a v aplikaci nastavit nové heslo.
4. Stáhnout Apache Tomcat z `https://tomcat.apache.org/download-80.cgi` a nahrát do složky /tomcat/webapps aplikaci (src/Instalace/app) a fonty (src/Instalace/fonts).
5. Otevřít url adresu s aplikací `http://localhost:8080/app`.

Příloha C

Uživatelská příručka

Uživatelská příručka je přiložena na CD.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `manual`
  - `Uživatelská příručka.pdf` ..... Uživatelská příručka ve formátu pdf
  - `Uživatelská příručka.docx` ... Uživatelská příručka ve formátu docx
- `src`
  - `Converter` ..... projekt pro vytvoření databáze Neo4j
  - `MBIQL Application` ..... projekt pro MBIQL aplikaci
  - `Instalace` ..... programy pro spuštění systému
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Stránský_Vojtěch.pdf` ..... text práce ve formátu PDF