FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh uloženia podnikových dát v manažérskom informačnom systéme
Student:	Bc. Maroš Karas
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cieľom diplomovej práce je navrhnúť riešenie pre manažérsky informačný systém (MIS), ktorý bude prevádzkovaný v cloudovom prostredí a bude škálovaný pre ľubovoľný počet podnikov a používateľov. Riešenie musí zaistiť bezpečnosť podnikových dát.

1) Zhodnoďte existujúce riešenie MIS.
2) Analyzujte možné databázové multi-tenancy riešenia s ohľadom na bezpečnosť podnikových dát.3) Popíšte transformáciu existujúceho riešenia aplikácie na mikroslužby.
4) Porovnajte cloudové a on premises režimy prevádzkovania MIS z technického a ekonomického hľadiska.
5) Navrhnite vlastné riešenie v rámci mikroslužieb.
6) Implementujte prototyp uloženia dát.
7) Dané riešenie otestujte.
8) Zhodnoťte prínosy a ďalší rozvoj.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. prosince 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Návrh uloženia podnikových dát v manažérskom informačnom systéme

Bc. Maroš Karas

Katedra softwarového inženýrství
Vedúci práce: Ing. Petra Pavlíčková, Ph.D.

26. mája 2020

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

Prague 26. mája 2020 .....

Odkaz na túto prácu

Karas, Maroš. *Návrh uloženia podnikových dát v manažérskom informačnom systéme*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Diplomová práca sa zaoberá uložením podnikových dát v manažérskom informačnom systéme, ktorý bude zdieľaný viacerými podnikmi v cloudovom prostredí. Dané riešenie som vytvoril pomocou uloženia dát do jednej databázy a do zdieľaných databázových tabuliek jednotlivými podnikmi. Vytvorené riešenie zaisťuje bezpečnosť podnikových dát a zamedzuje prístup používateľom k cudzím podnikovým dátam. Riešenie takisto berie do úvahy premenlivosť podnikov a umožňuje tak presun podnikových dát medzi jednotlivými podnikmi pri akvizíciach a fúziach podnikov. Prínosom tejto práce je zlepšenie nasadzovania nových verzií manažérskeho informačného systému a možná škálovateľnosť systému pre vyšší počet podnikov, ktoré budú mať záujem o systém.

Kľúčové slová manažerský informačný systém, multitenancy, databáza, cloud, PostgreSQL, REST-api, Kotlin, Spring boot, Hibernate

Abstract

This master thesis deals with the storage of enterprise data in management information system that will be shared between several companies in a cloud environment. The solution is created by storing the enterprise data in one shared database and shared database tables between each companies. The solution provides security and confidentiality of stored enterprise data and forbids users from accessing foreign enterprise data. It also considers variability of companies and it enables transfer of enterprise data between two companies when it comes to acquisitions and company mergers. The benefit of this thesis is improvement in deploying new versions of management information system and possible scalability of the system for higher number of companies that will be interested in the system.

Keywords Management information system, multi-tenancy, database, cloud, REST-api, PostgreSQL, Kotlin, Spring boot, Hibernate

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V diplomovej práci sa budem zaoberať spôsobom ako oddeliť podnikové dáta jednotlivých podnikov v zdieľanom manažérskom informačnom systéme medzi viacerými podnikmi, keďže existujúce riešenie nie je optimalizované pre väčší počet podnikov. Pridanie nového podniku a sprístupnenie manažérského informačného systému pre nový podnik je časovo náročné a veľmi pracné a preto je potreba vytvoriť návrh, ktorý tento proces zefektívni.

Aktuálne je manažérsky informačný systém nasadený pre viaceré podniky samostatne, s vlastnou databázou a aplikačným serverom. Toto riešenie zaisťuje bezpečnosť podnikových dát. Pre každý podnik musí byť zriadený server, buď fyzický alebo virtuálny, na ktorom je daný manažérsky informačný systém nasadený.

Uložením podnikových dát na jedno uložisko sa zníži počet serverov, keďže všetky podniky budú zdieľať rovnaké cloudové prostredie. Zdieľaním sa znížia náklady na zriaďovanie a prevádzku serverov. Centralizujú a zjednotia sa zmeny a nasadzovanie nových verzií informačného systému. Bezpečnosť podnikových dát bude zaistená tak, že používatelia a podniky nebudú mať prístup k cudzím podnikovým dátam.

Riešenie bude škálovateľné pre ľubovoľný počet podnikov a podnikové dáta budú prístupné len pre konkrétny podnik. Používateľ systému nebude mať prístupné dáta ďalších podnikov, ktoré s ním zdieľajú úložisko.

Tento spôsob uloženia podnikových dát bude veľkým prínosom pre projekt manažérsky informačný systém, pretože nastane skrátenie času a znížia sa náklady na založenie novej inštancie manažérského informačného systému pre nový podnik. Pre koncového zákazníka to bude mať takisto kladný dopad, pretože zníženie nákladov bude odzrkadlené v počiatočnej investícii do produktu MIS.

Kapitola 1

Cieľ práce

Cieľom diplomovej práce je navrhnúť riešenie pre manažérsky informačný systém (MIS), ktorý bude prevádzkovaný v cloudovom prostredí. MIS bude škálovaný pre ľubovoľný počet podnikov a užívateľov. Riešenie musí zaistiť bezpečnosť podnikových dát a flexibilitu pohybu dát z dôvodu možnej premenlivosti štruktúry podnikov.

Ďalším cieľom je porovnanie dvoch typov prevádzkovania informačného systému. Prevádzkovanie u zákazníka a prevádzkovanie v cloude bude porovnané po technickej a ekonomickej stránke.

Pre docielenie úspešného návrhu je potrebné vytvoriť prototyp riešenia, ktorý bude dostačujúci ako šablóna pre ďalší vývoj informačného systému. Tento prototyp bude následne otestovaný s ohľadom na využitie pri ostrej prevádzke systému.

Kapitola 2

Manažérsky informačný systém

Manažérsky informačný systém je jeden z informačných systémov, ktoré sú používané pre riadenie podnikov na manažérskej úrovni organizácie. Medzi hlavné funkcie patrí plánovanie, kontroly a rozhodovanie poskytovaním pravidelných prehľadov a hlásení o výnimočných situáciach.

V následujúcich sekciách priblížim pojmy ako podnik a informačný systém a vzápätí naviažem a bližšie popíšem manažérsky informačný systém a jeho vlastnosti a prínosy pre podniky.

2.1 Podnik

Z defínicie podľa knihy *Podniková informatika* [1] je jasné, že podnik je subjekt, v ktorom dochádza k premene vstupných zdrojov na výstupné statky. Je to súbor prostriedkov, zdrojov, práv a iných majetkových hodnôt, ktorý tvorí ekonomickú a právnu jednotku. Jeho hlavnou úlohou je slúžiť podnikateľovi k prevádzke jeho podnikateľských aktivít.

2.2 Podnikový proces

Podnikový proces je sled činností vykonávaných zamestnancami za účelom doručenie hodnoty koncovému zákazníkovi. Podľa [2] tento pojem zahrňuje obchodné procesy, personalistiku, ekonomiku, výskum, výrobu, skladovanie a iné špecifickejšie procesy. Podnikové procesy v existujúcom podniku je možné podporovať informačnými systémami, alebo službami.

2.3 Informačný systém

Informačný systém je množina komponentov spolupracujúca za účelom tvorby, zhromažďovania, spracovania, prenášania a rozširovania informácii a jeho definíciu môžeme nájsť aj v *Ekonomický slovník* [3]. Prvky informačného systému

tvoria ľudia, respektíve používatelia informácii a informatické zdroje. Komponenta je tvorená jedným alebo viacerími prvkami.

Podnikový informačný systém je otvorený systém, ktorého vstupy a výstupy sú informácie. Dochádza k prelínaniu živého a neživého systému v podniku. Podľa [1] v podnikovom informačnom systéme nájdeme jeho tri časti:

- Neformálny informačný systém

  Vvýmena a spracovávanie informácii ľuďmi a ďalšími komunikačnými technikami.

- Formálny informačný systém

  Formalizované pracovné a informačné toky realizované na základe politík, cieľov, stratégii, pravidiel a predpisov.

- Interná logika

  Počítače a stroje, ktoré realizujú kľúčové operácie s informáciami.

2.4 Podnikové dáta

Dáta sú formalizovaný záznam ľudského poznania pomocou symbolov a znakov, ktorý je schopný prenosu, uchovania, interpretácie a spracovania.

Podľa [1] v podnikovom informačnom systéme môžeme dáta rozdeliť na tri skupiny:

- Dáta o spoločenských podmienkach podnikania

  Údaje o politických a štátnych očakávaniach úrovne stability prostredia, demografických, sociálnych a ekonomických trendoch, rozvoji technológii pre vznik nových produktov a služieb a faktory ovplyvňujúce výrobu, ako sú pracovná sila, materiál a kapitál.

- Dáta o trhu

  Dáta o dopyte po komoditách podniku, t.j. tovar a služby, stav konkurencie, hlavne výkonnosť, aktuálne aktivity a plány, rozvoj komodít, očakávané fúzie a akvizície.

- Interné dáta podniku

  Interné dáta podniku tvoria predpoklady k tomu, aby podnik mohol reagovať na svoje okolie. Patria sem plány a predpovede predaja, vrátane finančných plánov, údaje o použití podnikových zdrojov. Ďalej tu patria dáta o vnútornej ponuke zdrojov, čiže pracovnej sile, kapitálu, strojov a zariadení a o ombedzeniach fungovania podniku, procesoch a pravidlách fungovania podniku.

2.5 Bezpečnosť podnikových dát

Bezpečnosť podnikových dát je docielená pri zaistení troch hlavných piliérov bezpečnosti [4] a to dôvernosti, integrity a dostupnosti dát.

Pri zaistení dostupnosti a dôvernosti dát, bez ohľadu na ich integritu, sa docielí to, že podnikové dáta sú oprávnenému používateľovi prístupné, keď ich potrebuje, ale nemá žiadnu záruku, že neboli modifikované.

Keď sa zaistí dôvernosť a integrita dát, bez ohľadu na ich dostupnosť, môže nastať situácia, že používateľ nebude mať prístup k svojim podnikovým dáta, aj napriek tomu, že nebudú modifikované alebo vyzradené.

Ak sa zaistí integrita a dostupnosť dát, bez ohľadu na ich dôvernosť, môžu byť podnikové dáta prístupné neoprávneným používateľom, ktorí by tieto informácie nemali vidieť.

Bezpečnosť podnikových dát je zaistená vďaka kladeniu dôrazu na itegritu, dôvernosť a dostupnosť podnikových dát tak, aby nedošlo k ich modifikácii, vyzradeniu alebo k nedostupnosti.

2.6 Manažérsky informačný systém

Manažérsky informačný systém je podnikový informačný systém, ktorý zbiera dáta z rôznych interných a externých zdrojov, analyzuje ich a vytvára správy, ktoré pomáhajú v rozhodovaní podnikovým manažérom [5].

Cieľom manažérskeho informačného systému je umožniť koreláciu dát z viacerých zdrojov za účelom zlepšenia podnikových procesov a stratégií.

Kľúčovým prínosom manažérskeho informačného systému je vytváranie prevádzkových správ, ktoré zobrazujú výdaje, predaje, výkonnosť zamestnancov, či náklady a efektivitu marketingu.

V každom podniku sú manažéri, ktorí robia rozhodnutia o chode podniku a riadia svojich zamestnancov. Zamestnanci robia svoj diel práce a po dokončení celku vzniká nejaká hodnota, ktorá je potom poskytovaná koncovým zákazníkom. Ak sa jedná o produkt, alebo o služby výsledky tohoto procesu nie sú hneď jasné a jednoznačné. Preto sa všetky informácie zbierajú v informačných systémoch a nad týmito dátami sa robia analýzy, aby následné rozhodnutia pomohli podniku zefektívniť procesy a ideálne zvýšiť zisk.

Zdroje dát delíme na externé a interné, podľa ich pôvodu. Externé zdroje môžu byť napríklad komerčné údaje podnikov alebo údaje o konkurencii. Interné zdroje máme zozbierané v podniku a tie sú napríklad účtovné evidencie, prehľady ziskov a strát, prehľady zásob alebo predajov.

2.6.1 Motivácia používať MIS v podniku

Manažérsky informačný systém zlepšuje rozhodovanie manažérov poskytovaním aktuálnych a presných dát o organizačných aktívach, napríklad ako financie, obsah skladov, personál, harmonogramy projektu, alebo marketing.

Systém zbiera interné a externé dáta, ukladá si ich a následne ich analyzuje. Z týchto analýz sú vytvorené prehľady a správy, ktoré sú prístupné pre manažérov. Manažérom sú tieto prehľady a správy nápomocné pri rozhodovaní o ďalšom smerovaní podniku.

2.6.2 Práca s informačným systémom

Pre správne fungovanie analýz, následným vytváraním prehľadov a správ je, dôležité oboznámenie a školenie všetkých zamestnancov, ktorí prídu do styku s týmto systémom.

Jedným zo základných pilierov fungovania manažérskeho systému je zber interných dát. Dáta je potrebné udržiavať konzistentné a nevynechávať dôležité informácie. Zamestnanci podniku sú pri vzniku týchto informácii a preto je zásadné ich informovať o spôsobe práce a využívaní tohoto systému.

Ide hlavne o evidenciu kontaktov dodávateľov a odberateľov, tvorbu faktúr, správu projektov, zapisovanie dochádzky do správneho projektu a v správnom rozsahu a prípadne zadávanie a plnenie úloh.

Kapitola 3

Manažérsky informačný systém v cloude

Manažérsky informačný systém môže byť prevádzkovaný u poskytovateľa cloudového riešenia a preto prístupný koncovému zákazníkovi, teda podniku cez internet.

Ak bol manažérsky informačný systém a jeho databáza navyše navrhnutá pre používanie viacerími podnikmi, je tento spôsob ideálnym riešením. Jednotlivé podniky budú zdieľať viac než len servery poskytovateľa, ale aj napríklad databázu, prípadne databázovú schému a preto je vynakladaný menší počet výpočtových zdrojov na daný systém.

3.1 Cloud computing

Cloud computing je termín, ktorý popisuje dostupnosť zdrojov, softvérových aj hardvérových, na požiadanie cez internet. Ide najmä o dátové úložiská a výpočtový výkon, bez priameho aktívneho spravovania používateľom. Tento výraz je všeobecne používaný na popis dátových centier, ktoré sú dostupné pre mnoho používateľom prostredníctvom internetu.

Podľa technickej správy z univerzity v Berckley [6] sa cloud computing vzťahuje, tak ako na aplikácie dodávané v podobe služby cez internet, tak aj na hardvér a softvér v dátových centrách, ktoré tieto služby poskytujú. Samotné služby sa označujú slovným spojením *softvér ako služba* (SaaS). Hardvér a softvér dátového centra je nazývaný cloud.

Softvér ako služba znamená, že daný softvér sa dá prenajať koncovým zákazníkom cez internet. Prenajať znamená, že sa platí iba za to, čo zákazník používa. Služba sa často označuje ako služba na požiadanie, pretože je k dispozícii vždy, keď ju zákazník potrebuje. Tento model má podľa [7] dve okamžité výhody a to, že počiatočné náklady sú zvyčajne podstatne nižšie a model poskytuje vyššiu úroveň ľahkej škálovateľnosti. Tieto výhody sa odzr-

kadlia aj na výhodách pre koncových zákazníkov, pretože tým, že poskytovateľ softvéru nemusí stráviť čas a peniaze navyše nákupom a inštaláciou hardvéru, sa počiatočné náklady znížia pre obidve strany.

3.2 Multi-tenancy

Termín multi-tenancy [8] je v softvérovej architektúre používaný na popis zdieľaného prostredia a zdrojov a slúži viacerým nájomníkom či konzumentom.

Nájomník je skupina používateľov, ktorí zdieľaju špecifické časti softvéru a to dáta, konfigurácie, správu používateľov a nefunknčných vlastností.

Takto navrhnutý systém zdieľa prostriedky a služby bez toho, aby jednotliví používatelia mohli zasahovať do prostredia ostatných nájomníkov.

3.3 Spôsoby cloudového riešenia MIS s ohľadom na bezpečnosť podnikových dát

Uloženie podnikových dát sa dá vyriešiť viacerými spôsobmi. Každý variant má svoje výhody a nevýhody, a preto treba zvoliť tú možnosť, ktorá najviac vyhovuje projektovým požiadavkam. V prípade manažérskeho informačného systému to je škálovateľnosť, bezpečnosť dát a zlúčenie, či odlúčenie podnikových divízií.

Ďalšia vec, ktorá sa musí zvážiť, je návrh mikroslužieb a návrh ich databáz, pretože pre mikroslužby je lepšie mať samostatnú databázu pre každú mikroslužbu.

Pri popisovaní jednotlivých spôsobov uloženia podnikových dát do databázy sa bude klásť dôraz na bezpečnosť 2.5 podnikových dát. Podnikové dáta nebudú môcť byť modifikované, vyzradené alebo nedostupné.

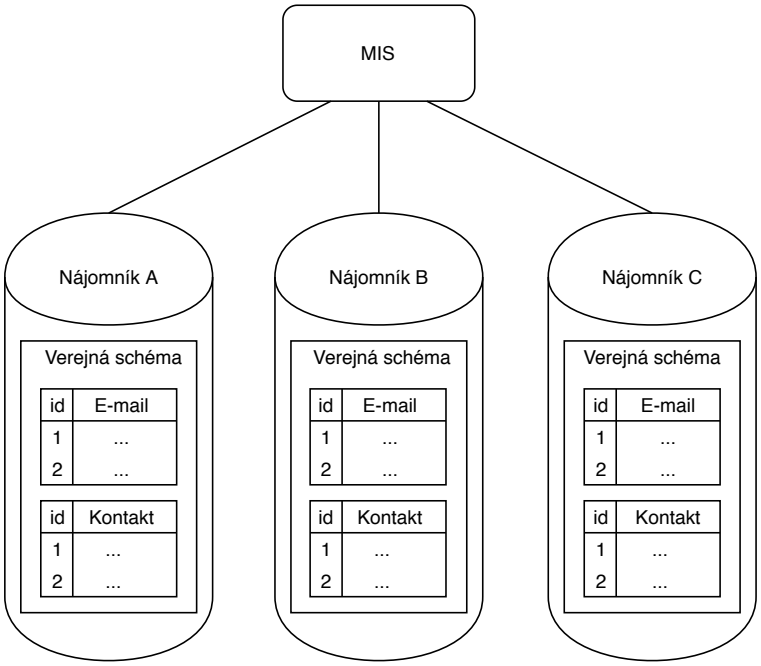
3.3.1 Jedna databáza pre každého nájomníka

Každý nájomník má vlastnú databázu, v ktorej má vlastné databázové schémy, tabuľky a podnikové dáta.

Na obrázku 3.1 je zobrazená aplikácia MIS a pre každý podnik je vytvorená samostatná databáza, v ktorej sú uložené podnikové dáta. Nájomníci zdieľajú len aplikáciu a každý nájomník má vlastné spojenie na svoju databázu.

Dáta sú izolované od ostatných nájomníkov, uložené v rôznych databázach a databázových tabuľkách. Nájomníci nemajú prístup do ostatných databáz. Bezpečnosť prístupu k podnikovým dátam je vysoká, pretože každý nájomník má prístup iba k svojej databáze.

Ak niektorý z podnikov bude požadovať nejaké unikátne zmeny, tak v samostatných databázach toto dosiahneme ľahko pre každého nájomníka, pretože sú databázy oddelené. Možné zmeny sú napríklad pridanie tabuliek do schémy, pridanie stĺpcov do tabuliek, zmeny v schéme, alebo zmeny v tabuľkách.



Obr. 3.1: Samostatné databázy nájomníkov
Zdroj: vlastné spracovanie

Vznik nového podniku je možné vytvorením novej databázy pre ďalší podnik. Takto sa zdvíha hardvérový nárok a náklady na databázový server, na ktorom sú uložené jednotlivé databázy. Preto je škálovateľnosť tohoto návrhu pomerne nákladná, no nie neuskutočniteľná.

Na zlúčenie a odlúčenie divízii je potrebná migrácia dát z jednej databázy do druhej, ktorá bude prípadne vytvorená pri akvizícii, alebo pri fúziach.

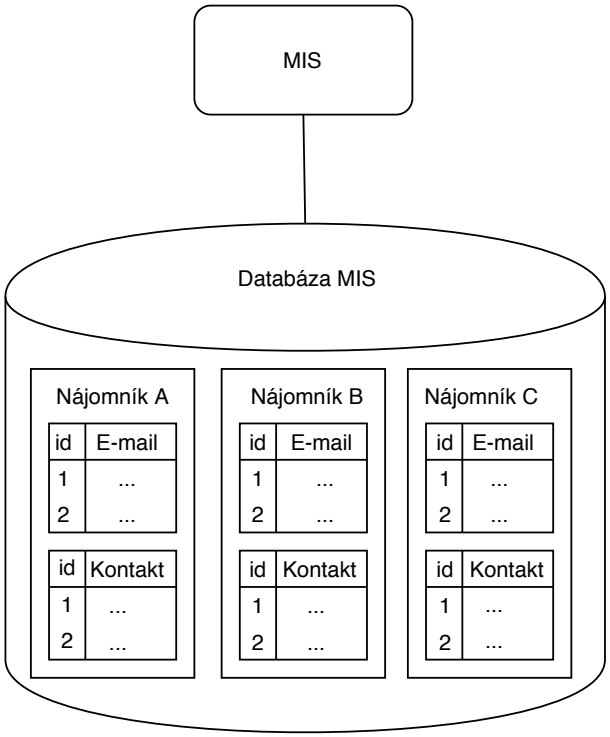
3.3.2 Spoločná databáza, jedna schéma pre každého nájomníka

Databáza je zdieľaná medzi jednotlivými nájomníkmi, ale každý z nich má vlastnú databázovú schému, v ktorej má vlastné tabuľky a podnikové dáta.

Na obrázku 3.2 je databáza zdieľaná a preto všetky náklady na správu databázy sú menšie. Každý z nájomníkov má vlastnú databázovú schému, v ktorej sú uložené ich podnikové dáta.

Pri použití databázových schém sú umožnené unikátné zmeny tabuliek, alebo ich štruktúry pre konkrétneho nájomníka v databázovej schéme.

Bezpečnosť podnikových dát je zaručená pomocou vlastníctva a práv na databázové schémy. Podniková dáta sú izolované a nájomníci nemajú prístup k cudzím schémam.



Obr. 3.2: Zdieľaná DB, samostatné schémy nájomníkov
Zdroj: vlastné spracovanie

Škálovanie je uskutočnené pomocou pridania nových schém, ale má svoje obmedzenia. Pri veľkom počte schém dochádza k spomaleniu databázovej administrácie, pomalým upgradom, exportom, archívu či migráciou a nasadením. Takisto môžu nastať problémy s veľkosťou cache, ktoré sú závislé na konkrétnom softvéry spravovania bazénu pripojení na databázu. Preto je potrebné použiť vhodný nástroj na pripojenie k databáze, aby neiteroval všetky schémy, ale rovno vybral iba tie, ku ktorým má nájomník prístup.

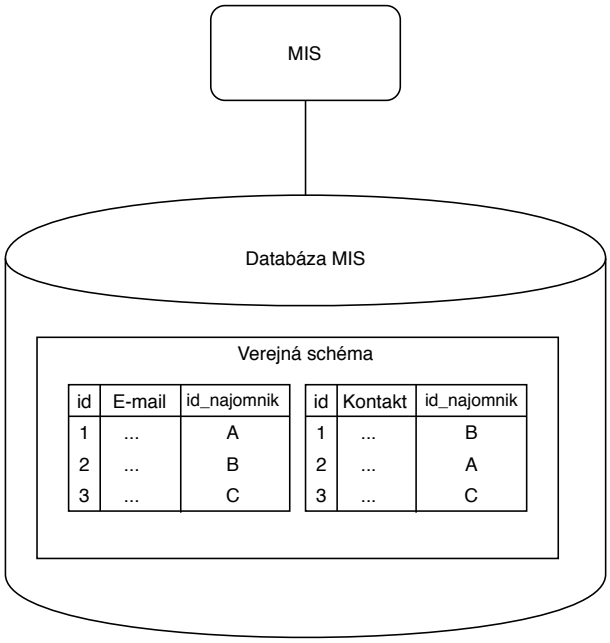
V tomto prípade je vhodné a odporúčené používať univerzálne unikátne identifikátory (UUID) jednotlivých databázových entít.

Zlúčenie a odlúčenie divízií je ľahko uskutočniteľné pomocou povolenia prístupu do ďalších schém.

3.3.3 Spoločná databáza, spoločná schéma

Každý nájomník má označené riadky v zdieľaných tabuľkách, ktoré mu patria na základe prideleného identifikátora.

Na obrázku 3.3 je zdieľaná databáza a zdieľaná databázová schéma. Všetky podnikové dáta jednotlivých nájomníkov sú uložené v databázových tabuľkách jednej databázovej schémy. Výhodou tohoto riešenia je zdieľanie bazénu pripojení k databáze, čo zapríčiní pokles ich celkového počtu.



Obr. 3.3: Zdieľané tabuľky nájomníkov
Zdroj: vlastné spracovanie

Toto riešenie je veľmi dobre škálovateľné pre ľubovoľný počet nájomníkov a podnikov. Z odborného článku o databáze *Google F1* [9] je toto riešenie takisto rozšíriteľné o takzvané rozdelenie databázy na partície alebo trieštenie databázy na úlomky. V anglickom jazyku sa používajú pojmy *database partitioning* a *database sharding*.

Pri spoločnej schéme sa veľmi dobre vykonáva databázová administrácia a správa tabuliek, ako napríklad zmena tabuľky, či vytvorenie nového indexu. Naopak je ťažšie dosiahnuť unikátnych zmien schématu, pretože schéma je zdieľaná všetkými nájomníkmi. Unikátne zmeny schémat sa dajú vyriešiť ignorovaním zmien ostatnými nájomníkmi.

Problém môže nastať pri izolovaní dát, preto je nevyhnutné venovať pozornosť úniku dát. Je potrebné zaistiť, aby každý nájomník videl iba vlastné dáta a tie, na ktoré má práva. Každá požiadavka preto obsahuje identifikátor nájomníka.

Pri dobrom návrhu zdieľanej schémy je jednoduché zaistiť zlúčenie alebo odlúčenie divízii podnikov, pretože podnikové dáta nemusia migrovať. Menia sa len používateľské práva a prístup k týmto podnikovým dátam.

3.4 Mikroslužby

Mikroslužby vychádzajú z konceptu modularizácie, kde každý modul je implementovaný a prevádzkovaný ako malý, ale nezávislý systém [10]. Mikroslužba ponúka prístup k svojej vnútornej logike a údajom prostredníctvom dobre definovaného sieťového rozhrania. To zvyšuje pružnosť softvéru, pretože každá mikroslužba sa stáva nezávislou jednotkou vývoja, nasadenia, operácií, verzovania a škálovania.

Mikroslužby používajú len ľahké technológie ako REST alebo HTTP4 narozdiel od architektúry orientovanej na služby (SOA). Architektúrou orientovanou na služby sa väčšinou chápe integračné riešenie, zatiaľ čo použitie mikroslužieb sa používa skôr na budovanie jednotlivých softvérových aplikácií.

Hlavnými výhodami použitia mikroslužieb je rýchlejšie dodanie výsledného softvéru, lepšia škálovateľnosť a väčšia autonómia modulov. Takisto každá mikroslužba môže byť vyvinutá, nasadená a prevádzkovaná iným tímom, čo umožňuje paralelnejšie zavádzanie nových funkcií.

Kapitola 4

Cloud vs on-premises režimy prevádzkovania MIS

V tejto kapitole sú vysvetlené spôsoby prevádzkovania softvéru a ich porovnanie z technického a ekonomického hľadiska. Prevádzkovanie softvéru budeme rozdeľovať na prevádzkovanie v priestoroch zákazníka, prevádzkovanie v cloude a prevádzkovanie zdieľanej aplikácie viacerími zákazníkmi v cloude.

4.1 Prevádzkovanie u zákazníka

Zákazníci podnikajú v rôznych odvetviach a preto majú rôzne vybudovanú infraštruktúru vnútornej sieti. Všetky dáta a softvér sa nachádza priamo u zákazníka na vlastných serveroch a vlastnej infraštuktúre [11]. Naviac ak má zákazník zabezpečenú sieť s interným firewallom, tieto dáta neopustia vnútornú sieť.

Na prevádzkovanie softvéru zákazníka je potrebné mať hardvér, licencie na softvér, infraštruktúru a správcov serverov. Spravovanie serverov a správneho chodu softvéru majú na starosti správcovia, ktorí pracujú pre zákazníka.

4.2 Prevádzkovanie v cloude

Prevádzkovanie v cloude sa podľa [11] líši od prevádzkovania u zákazníka tým, že celú prevádzku poskytuje dodávateľ, alebo tretia strana.

Dodávateľ softvéru, sa môže rozhodnúť, alebo dohodnúť so zákazníkom na prevádzkovaní softvéru v priestoroch dodávateľa, alebo prevádzkovaním u tretej strany. Takýmto spôsobom zákazník nemusí mať vybudovanú infraštruktúru a potrebný hardvér na prevádzku daného softvéru, pretože tou disponuje dodávateľ, alebo tretia strana. Takisto zákazník nemusí vyčleňovať pracovnú silu v podobe správcov serverov.

Všetky servery a dáta sa nachádzajú v cloude a zákazník má prístup k svojim dátam, prostredníctvom internetu.

4.3 Prevádzkovanie multi-tenancy aplikácie v cloude

Zdieľaná aplikácia viacerími zákazníkmi je prevádzkovaná obdobne ako softvér v cloude, s rozdielom v prístupe ku aplikácii, pretože prístup k aplikácii má viacero zákazníkov naraz.

Toto riešenie je vhodné pre softvér, ktorý je prevádzkovaný u viacerých zákazníkov a líši sa minimálne. Musí ale dôjsť ku úpravám architektúry a návrhu softvéru, aby zdieľanie jedného softvéru viacerími zákazníkmi bolo uskutočniteľné.

Všetky servery a dáta sa nachádzajú v cloude a zákazníci majú prístup iba k svojim dátam, prostredníctvom internetu.

4.4 Technologické porovnanie

Dané spôsoby prevádzkovania softvéru sa líšia v spôsobe komunikácii a preto majú rôzny dopad na životný cyklus podnikových dát. Použitie jednotlivého typu prevádzkovania softvéru môže mať aj ďalšie dôsledky, ktoré sú opísané v tejto sekcii.

4.4.1 Bezpečnosť

Spoločnosti, ktoré pracujú s mimoriadne citvlivými dátami ako napríklad vládne alebo bankové odvetvie musia zabezpečiť určitú úroveň bezpečnosti a súkromia, ktorú poskytuje miestne prostredie. Napriek všetkým výhodám cloudu a vysokej cene prevádzkovania softvéru v priestoroch zákazníka dáva táto voľba väčší zmysel.

Bezpečnostné obavy ostávajú prekážkou číslo jedna pre prevádzkovanie softvéru v cloude. Poskytovateľ prevádzky softvéru na cloude musí zaistiť bezpečnosť dát a zamedziť ich úniku. Po celom svete v IT odvetví došlo k prelomeniu zabezpečenia cloudov a bezpečnostné hrozby sú skutočné. Od úniku osobných údajov zamestnancov, napríklad prihlasovacie údaje až po stratu duševného vlastníctva.

4.4.2 Prístup k dátam

Ak zákazník prevádzkuje softvér vo svojich priestoroch má lepšiu kontrolu nad dátami a čo sa s nimi deje. Spoločnosti vo vysoko regulovaných odvetviach s dôrazom na ochranu súkromia preto preferujú túto možnosť a zdráhajú sa prejsť na cloudové riešenie.

V prostredí cloudu sa dáta nachádzajú v priestoroch dodávateľa softvéru alebo tretej strany. Ak dôjde k neočakávaným situáciám a dôjde k výpadkom poskytovaného softvéru, zákazník nebude mať prístup k týmto dátam.

4.4.3 Nasadzovanie softvéru

Pri prevádzkovaní softvéru u zákazníka v internej sieti je nasadzovanie softvéru v rukách zákazníka. Zákazník je zodpovedný za prevádzku a údržbu riešenia a všetkých jeho procesov. Ak dôjde k vydaniu novej verzie, alebo opravám chýb v systéme je potrebné novú verziu softvéru znova nasadiť na server. Táto nová verzia musí byť dodaná dodávateľom softvéru.

Ak je softvér prevádzkovaný v cloude, o prevádzku a údržbu sa stará dodávateľ softvéru, alebo tretia strana. Zákazník môže pristupovať ku softvéru a všetkým jeho zdrojom v akýkoľvek čas. Pri vydaní novej verzie, či oprave chýb sa tieto zmeny automaticky nasadia na server, na ktorom bola nasadená staršia verzia.

4.4.4 Dostupnosť

Zákazník sa dostane k softvéru z miestnej sieti, alebo cez vzdialený prístup pomocou virtuálnej miestnej sieti(VPN). Vzdialený prístup musí byť zabezpečený proti vonkajším útokom.

Zákazník bude mať prístup ku cloudovému riešeniu neustále, kdekoľvek a z ktoréhokoľvek počítača. Týmto spôsobom je možnosť práce z domova dostupná hneď, bez riešenia prístupu ku sieti.

4.5 Ekonomické porovnanie

Keďže za dané typy prevádzkovania je zodpovedná buď jedna, alebo druhá strana je jasné, že to má vplyv na náklady prevádzkovania a údržbu systému.

Pre poskytovateľa prevádzky softvéru je výhodou, že už má odbornú pracovnú silu a hardvér potrebný na prevádzkovanie systému.

4.5.1 Náklady na prevádzkovanie

Ak si zákazník chce prevádzkovať sám v jeho priestoroch potrebuje spraviť počiatočnú investíciu do hardvéra a to serverov a infraštruktúry. O tieto servery sa budú starať ich správcovia a takisto nasadzovať a udržiavať softvér v prevádzke. Počiatočný kapitál a pravidelné náklady na správcov serverov sú veľké a len ťažko zanedbateľné položky.

Oproti tomu ak dodávateľ softvéru prevádzkuje softvér v cloude, nie je to väčšinou len pre jedného zákazníka a tým sa jeho náklady na hardvér a pracovnú silu rozdelia medzi viacero zákazníkov. Táto cena je oveľa atraktívnejšia pre väčšinu zákazníkov a takisto zákazník nemusí najímať kvalifikovaných

pracovníkov na prevádzku serverov. Softvér v cloude je väčšinou platený ako služba mesačne, alebo ročne. Zákazníci si platia len za zdroje, ktoré naozaj využijú a za čas v ktorý to využívajú. Navyše prevádzkovanie softvéru v cloude umožňuje použitie menšieho počtu fyzických serverov a počtu správcov serverov, pretože je všetok hardvér na jednom mieste.

4.5.2 Náklady na údržbu

Prevažná časť nákladov na podnikový softvér sa vynakladá práve na inštaláciu a údržbu softvéru [7]. Podobne ako u nákladov na prevádzkovanie softvéru, je potrebné mať správcov, ktorý udržiavajú softvér v prevádzke a zaisťujú prípadne nasadzovanie nových verzií od dodávateľa.

Táto položka je pri prevádzkovaní v cloude podstatne menšia pre zákazníka, pretože údržba je jednoduchšia a veľa procesov je zautomatizovaných. Takisto je potrebný menší počet správcov a pri zdieľanom cloudovom softvére sa zdieľajú ako hardvérové zdroje, tak aj finančné prostriedky potrebné na údržbu softvéru medzi jednotlivých zákazníkov.

4.6 Zhrnutie prevádzkovania softvéru

Z technického a ekonomického porovnania prevádzkovania softvéru vyplýva, že obe varianty ako u zákazníka tak v cloude majú výhody a aj nevýhody. Vybrať vhodný typ prevádzky závisí na konkrétnom softvéri a jeho každodennom používaní v praxi.

Hlavnou výhodou prevádzky priamo v priestoroch zákazníka je bezpečnosť. Softvér je nasadený na serveroch na miestnej sieti a pokiaľ je sieť zabezpečená proti útokom zvonku, nie je žiadných pochýb o bezpečnosti. Ak si zákazník cení svoje dáta a prístup k ním natoľko, že je ochotný investovať do hardvéru a infraštruktúry, alebo hardvér a infraštruktúru už má vybudovanú, je pre neho voľba prevádzky v jeho priestoroch dobrá voľba. Ak má zákazník navyše kvalifikovaných pracovníkov, ktorí vedia spravovať servery je to ďalšie plus.

Na druhú stranu, nie každý zákazník pracuje v IT odvetví, alebo má definované IT oddelenie a tak počiatočná investícia do serverov, infraštruktúry a správcov by bola príliš veľká. Bezpečnosť cloudu má na starosti poskytovateľ cloudu a je to jeho veľká priorita, práve kvôli obávam preniknutia bezpečnosti. Môže sa stať, že cloud nie je dostatočne zabezpečený, ale je to ojedinelá výnimka, nie pravidlo. Preto ak sa jedná o zákazníka, ktorý potrebuje finančne dostupný a fungujúci produkt, je cloudové riešenie väčšinou dobrou voľbou.

Z pohľadu dodávateľa softvéru a poskytovateľa cloudového prevádzkovania je výhodou cloudu škálovsteľnosť, takže viacej zákazníkov a menší počet potrebných serverov a správcov.

Kapitola 5

Analýza

V tejto kapitole sa budem zaoberať existujúcim riešením manažérského informačného systému, jeho hlavným modulom, funckiám a použitým technológiam. Identifikujem nedostatky a problémy, ktoré vznikli z návrhu a objavili sa pri používaní systému.

Následne definujem požiadavky na nový manažérsky informačný systém, priblížim transformáciu modulov na mikroslužby a popíšem rôzne možnosti implementovania zdieľaných tabuliek, ktoré som definoval v sekcii 3.3.3.

5.1 Existujúce riešenie MIS

Manažérsky informačný systém slúži na sledovanie firemných procesov, podporuje manažérske rozhodovanie, vytvára manažérske reporty a podklady pre riadenie podniku. Skladá sa z viacero modulov [12], ktoré sú funkčne rozdelené.

5.1.1 Moduly

- Riadenie vzťahov so zákazníkom (CRM)

  Tento modul slúži na riadenie vzťahu so zákazníkom, ponúk, rozpracovaných zakázok.

- Kontakty

  Kontakty poskytujú kompletnú databázu osobných kontaktov, odberateľov, dodávateľov a interných používateľov. Podporuje hierarchickú štruktúru kontaktov.

- Administrácia

  Administrácia ponúka kompletné riadenie a konfiguráciu systému MIS.

- Výkazy zamestnancov

Modul výkazov slúži na evidovanie dochádzky zamestnancov, riešenie projektov a plánovanie dovoleniek zamestnancov.

- Manažérské prehľady

  V manažerských prehľadoch sa dajú zobraziť prehľady rentability projektov, štatistika efektivity pracovníkov, prehľady toku financií a iné prehľady, grafy a štatistiky.

- Dokumenty

  Pomocou modulu dokumenty sa dá vytvoriť interná wiki (stránka, ktorá umožňuje používateľom upravovať jej obsah a štruktúru), ktorá môže obsahovať firemné ifnormácie a know-how, či pravidelné firemné správy alebo noviny.

- Úlohy

  V systéme úloh má každý používateľ nastavené svoje úlohy, ktoré musí splniť a úlohy, ktoré zadal iným používateľom. Systém úloh podporuje určenie priorít jednotlivým úlohám, stanovenie termínu odovzdania úlohy a nastavenie pripomienky na úlohu.

- Financie

  Modul financie zahŕňa kompletnú ekonomiku firmy. Modul obsahuje knihy faktúr, pokladni, bánk, pôžičok, majetku a skladov.

5.1.2 Skupiny a divízie

Existujúci MIS využívajú rôzne podniky, ktorých štruktúra sa môže časom meniť a vyvíjať. Môže dojsť k zlúčeniu dvoch alebo viacerých podnikov, alebo k rozdeleniu podniku na dve časti. Divízia je časť podniku identifikovaná IČ. Zvyčajne každá divízia odpovedá jednému IČ, teda jednej právnickej alebo fyzickej osobe. Je ale možné aby sa jedna právnická či fyzická osoba rozdelila na viacero divízii, typicky podľa rôznych podnikateľských činností daného subjektu. Preto je za najmenšiu časť podniku zvolená divízia a podnik ako taký nazývame skupina divízii, ktorý obsahuje jednotlivé divízie.

Skupina divízii môže mať jednu alebo viac divízii, no nemôže byť prázdna. Postupom času sa skupina divízii môže rozšíriť o ďalšie divízie, buď kúpou, alebo vytvorením novej divízie.

Môže nastať situácia, kedy by dve rôzne skupiny divízií používali náš MIS a chceli sa zlúčiť. Pred zlúčením majú divízie rôznych používateľov a navzájom nemajú žiadne práva na druhú divíziu, zdieľajú iba prostredie. V tomto prípade by toto zlúčenie nemalo narúšať používateľové zavedené procesy a nastavenia na úrovni divízie. Obdobne by to bolo pri rozdelení jednej skupiny divízií.

5.1.3 Technológie

MIS je napísaný v Jave za použitia Spring frameworku a knižnice hibernate. Dáta sú uložené v databáze od PostgreSQL. Aplikačný server, na ktorom MIS beží je JBoss. Pre projekt MIS sa aktuálne používa 5 serverov s n inštanciami aplikačného servera a testovací server. Taktiež existuje samostatný databázový server, kde sú vytvorené databázy pre jednotlivé inštancie MIS a testovací databázový server.

Za pomoci vývojového nástroja Ant sa vytvorí WAR súbor, ktorý je archív a obsahuje všetky časti webovej aplikácie. Rozširuje súbor JAR o adresáre zdrojového kódu a konfiguračného súboru, ktorý hovorí aplikačnému serveru, čo má spustiť a ako to má spustiť [13]. Tento súbor sa potom nahrá na aplikačný server, ktorý odarchivuje WAR súbor a spustí webovú aplikáciu.

5.2 Problémy

V existujúcom riešení MIS sa nachádza viacero problémov, ktoré v nasledujúcich sekciách popíšem. Ide hlavne o problémy, ktoré vyplývajú z technologického návrhu riešenia a jeho použitia. Tento návrh viac preferuje bezpečnosť a samostatnosť nad znovu použitím a možnosťou škálovania.

5.2.1 Vývoj

Problémy pri vývoji nastávajú vtedy, keď existujú duplicitné zdrojové kódy k rovnakému systému, ktoré používajú rozličné skupiny divízií. Jedna zo skupín divízií má vlastný verzovací Git repozitár, ktorý je nezávislý od ostatných skupín. Preto všetky zmeny, ktoré sa budú uplatňovať plošne pre všetky systémy skupín divízií sa musia implementovať duplicitne.

Takýto prístup porušuje princípy neopakovania sa a jednobodového riadenia [14]. Duplicitný zdrojový kód prináša problémy pri údržbe kódu, pretože núti programátorov manuálne sledovať a upravovať každý samostatný opakujúci sa fragment. Taktiež sa môže stať, že sa omylom zavedie do kódu chyba, ktorá nie je úplne jasná a tak sa dostane do ostrej verzie.

5.2.2 Nasadenie

Problémy s nasadením súvisia s počtom všetkých inštancií aplikačného servera. Na každý aplikačný server sa musí nahrať WAR súbor s webovou aplikáciou. Jeden z inštancií má navyše iný zdrojový kód, odlišný od ostatných a tak WAR súbor sa musí vytvoriť iný a nasadiť na daný aplikačný server.

Aj pri automatizovaní nasadzovania webovej aplikácie na aplikačný server môžu nastať rozličné problémy. Napríklad na niektorom z aplikačných serverov nastane chyba pri nasadzovaní, alebo je potreba sa vrátiť do pôvodnej verzie webovej aplikácie.

5.2.3 Škálovanie

Pri zavádzaní MISu do nového podniku je potrebné uskutočniť určité kroky. Predovšetkým vytvoriť novú databázu, nový aplikačný server a ak hardvér nestačí, tak aj fyzicky zabezpečiť nový server. Nakonfigurovať servery, upraviť konfiguráciu webovej aplikácie a vytvoriť používateľa s administrátorskými právami.

Celý tento proces je pracný a časovo náročný. Založenie novej inštancie pre ďalšiu firmu by mal byť ľahký a rýchly proces. Pre firmu, ktorá si kupuje produkt, zakladá a konfiguruje nový systém, musí byť tento proces čo najjednoduchší a rychlý, aby si nákup nerozmyslela.

5.2.4 Zlúčenie a rozdelenie skupín divízií

Zlúčenie a rozdelenie skupín divízií je v existujúcom systéme MIS zložité a ťažko uskutočniteľné. Prevod podnikových dát z jednej skupiny divízií do druhej je uskutočnený extrahovaním dát z jednej databázy a importovaním do tej druhej.

Pri rozdelení skupín divízií nastáva podobný problém ako pri škálovvaní a to z dôvodu založenie novej skupiny pre divíziu či divízie, ktoré sa oddeľujú. Takže okrem prevodu podnikových dát medzi rôznymi databázami sa musí vytvoriť nová databáza a aplikačný server.

5.3 Požiadavky na nový systém

Sekcia 5.2 Problémy popisuje niekoľko základných problémov, ktoré existujúci systém má. Z týchto problémov vyplýva niekoľko požiadaviek na nový systém, ktoré je potrebné popísať.

5.3.1 Bezpečnosť dát

Existujúci MIS zabezpečuje bezpečnosť podnikových dát veľmi dobre. Každý podnik má vlastnú databázu s dátami, ku ktorým nemá nikto iný prístup. Pri inom návrhu uloženia podnikových dát viacerých podnikov v jednom systéme je preto potreba dbať na zabezpečenie dát a prístupu k nim.

Zabezpečenie prístupu k datám nemusí byť interne rovnaká, ale musí byť dostačujúca, aby používateľ informačného systému mal prístup len ku podnikovým dátam daného podniku ku ktorému patrí. Z vonku sa teda bude javiť, že používateľ pristupuje práve do webovej aplikácie daného podniku.

5.3.2 Jednoduchosť vývoja a nasadenia

Pri vývoji a údržbe zdrojového kódu informačného systému sa musí vyvarovať opakovaniu zdrojového kódu a predchádzať tak chybám a stratám na časovom fonde.

Nasadzovať sa bude webová aplikácia len raz a to narozdiel od existujúceho MISu zamedzí chybám a hlavne časovej náročnosti doterajšieho nasadzovania. Už sa nebudú vytvárať rôzne verzie WAR súborov, ktoré sú aktuálne nahrávané na inštacie aplikačných serverov, ale bude sa nasadzovať jediná verzia webovej aplikácie, spoločná pre všetky skupiny divízií.

5.3.3 Škálovateľnosť

Nový informačný systém musí byť dobre škálovateľný ako na interný počet používateľov, tak na celkový počet podnikov, ktoré budú mať o systém záujem. Vytvorenie novej skupiny divízií bude jednoduchý proces, kde vstupom budú základné informácie o danom podniku a výstup bude tvoriť založený nová skupina divízie, divízie a vytvorený používateľ s administrátorskými právami. Ďalej bude nasledovať nastavenie a konfigurácia informačného systému.

Nový MIS musí byť navrhnutý tak, aby založenie novej skupiny divízií bolo rýchle a neboli potrebné žiadne ďalšie kroky. Pri takomto stave si bude môcť ľubovoľný počet podnikov založiť nový manažérsky informačný systém.

5.3.4 Flexibilita

Nový MIS musí poskytnúť riešenie na problematiku zlúčenia a oddelenia divízií a skupín divízií. Aktuálne je táto možnosť zbytočne zložitá, preto s týmto musí návrh nového systému počítať.

Podnikové dáta preto musia byť uložené tak, aby sa divízie mohli prevádzať medzi skupinami divízií ľubovoľne a bez zbytočných zásahov do infraštruktúry databáze.

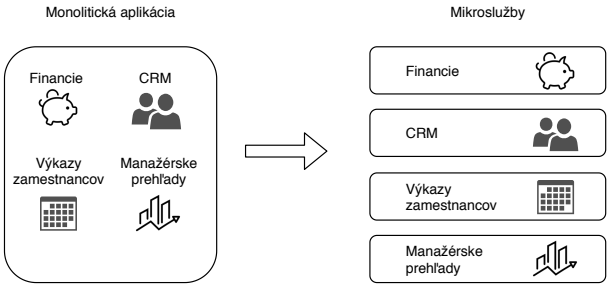
5.4 Transformácia na mikroslužby

Na obrázku 5.1 je zobrazená transformácia monolitickej aplikácie na aplikáciu s využitím mikroslužieb. Všetka logika, ktorá súvisí s nejakou funkcionalitou je presunutá do vlastnej mikroslužby, ktorá je oddelená a samostatná a s ostatnými mikroslužbami komunikuje.

Transformácia existujúcich modulov manažérskeho informačného systému na samostatné mikroslužby má docieliť prehľadnejší vývoj a údržbu zdrojového kódu, rýchlejšie a menej problémové nasadenia a zlepšiť škálovateľnosť webovej aplikácie.

5.4.1 Moduly ako mikroslužby

Každý z popisovaných modulov v sekcii 5.1 Existujúci MIS bude transformovaný na samostatnú mikroslužbu. Táto transformácia musí zachovať funkcionalitu existujúcich modulov a pritom oddeliť moduly tak, aby boli autónomne a vedeli spolu komunikovať.



Obr. 5.1: Transformácia modulov na mikroslužby
Zdroj: vlastné spracovanie

Všetka komunikácia medzi jednotlivými mikroslužbami sa uskutočňuje prostredníctvom sieťových volaní [15], aby sa zaistilo oddelenie medzi službami a zabránilo sa nebezpečenstvu tesného spojenia.

Okrem vymenovaných modulov bude potreba naimplementovať aj mikroslužby slúžiace na autentifikáciu a autorizáciu používateľov a vstupnú bránu.

Z obchodného hľadiska bude dôležité podporovať aktiváciu a deaktiváciu jednotlivých modulov, kvôli poskytovaniu MISu ako samostatných balíčkov funkcionalít.

5.4.2 Mikroslužba pre vstupnú bránu

Vstupná brána je server, ktorý je jediným vstupným bodom pre všetky možné požiadavky do systému. Je podobná ako fasáda z objektovo orientovaného dizajnu. Vstupná brána zapuzdruje vnútornú architektúru a poskytuje vonkajšie rozhranie pre klienta. Následne požiadavky rozposiela na interné rozhrania jednotlivých mikroslužieb.

Pri použití vstupnej brány nemusí klientská strana systému vedieť, ktorú konkrétnu mikroslužbu volá, stačí jej vedieť rozhranie endpointu. Vstupná brána túto požiadavku zaregistruje a predá ju ďalej na spracovanie.

5.4.3 Mikroslužba na overenie identity a autorizáciu používateľov

Na overenie identity a autorizácie používateľov bude slúžiť samostatná mikroslužba, ktorá bude overovať totožnosť prihlasovaného používateľa a overenie jednotlivých práv a rolí, ktoré tento používateľ má.

Prihlásený používateľ dostane po overení token, ktorí slúži na budúce overenie jeho požiadaviek, ktoré bude zasielať na aplikačný server. Tento token obsahuje informácie o prihlásenom používateľovi, prípadne jeho práva a role a čas s platnosťou tokenu. Po vypršaní platnosti tokenu sa musí používateľ znova overiť v tejto mikroslužbe.

5.5 Možnosti použitia zdieľaných tabuliek

Pre nový MIS je dôležité možnosť škálovania, bezpečnosť dát, zlúčenie a odlúčenie divízii podnikov a s možnosťou unikátnych zmien zákaznikom na mieru sa nepočíta, je vhodné vybrať uloženie podnikových dát do prostredia zdieľaných tabuliek.

Toto riešenie zaisťuje zníženie zdrojov, veľkú možnosť škálovania, lepšiu správu a administráciu a pri správnom návrhu a technickom prevedení aj bezpečne izolované podnikové dáta s možnosťou ľahkého presunu divízii medzi podnikmi.

5.5.1 PostgreSQL pohľady s bezpečnostnou bariérou (Security barrier views)

Jedným možným riešením je použiť pohľady na tabuľky, ktoré by mali za úlohu vyriešiť povolenie a zabránenie prístupu k podnikovým dátam. Pohľad je virtuálna relácia [16], ktorá môže slúžiť na filtrovanie informácií, ktoré používateľ nemá vidieť, alebo nemá správnu rolu.

Od verzii 9.2 PostgreSQL implementoval [17] radu bezpečnostných opatrení pre tvorbu pohľadov nad tabuľkami, aby sa zabránilo nevyžiadaným prístupom a únikom dát. Pri tvorbe pohľadu stačí nastaviť atribút *security_barrier*.

Funkcia *current_setting()* vracia hodnotu daného parametru uloženého v session. Túto hodnotu je potrebné nastaviť pri tvorbe session pre prihláseného používateľa.

```
1    CREATE VIEW users_view
2    WITH (security_barrier)
3    AS (
4        Select users.*
5        FROM users
6        WHERE tenant_id :=
7            current_setting("app.tenant_id")
8    );
```

Výpis kódu 5.1: Pohľad s bezpečnostnou bariérou

Zdroj: vlastné spracovanie

Na výpise kódu 5.1 je príklad použitia pohľadu s bezpečnostnou bariérou. Po vykonaní tohoto príkazu bude vytvorený pohľad, ktorý bude filtrovať všetky záznamy z tabuľky používateľov podľa atribútu *tenant_id*.

5.5.2 PostgreSQL bezpečnostná politika na úrovni riadkov (RLS policy)

Ďalšie možné riešenie je použitie filtrovania riadkov databázy. Od verzie 9.5 PostgreSQL zaviedol [18] možnosť vytvorenia takzvanej bezpečnostnej politiky nad tabuľkou. Pri zapnutí tejto vlastnosti a definovaní politiky sa odfiltrujú riadky, ktoré nevyhovujú podmienkam definovaným v definícii politiky.

Okrem nastavenia názvu politiky a názvu tabuľky, nad ktorou je táto politika definovaná, sa dá nastaviť ďalších 5 parametrov.

- AS

  Týmto parametrom sa nastavuje, či daná politika bude liberálna, alebo obmedzujúca a pri vyhodnotení sa budú politiky reťaziť pomocou logických operátorov *alebo*, *a*.

- FOR

  Špecifikovaním parametru FOR obmedzujeme túto politiku len na niektoré SQL metódy zo zoznamu Select, Insert, Update a Delete.

- TO

  Tento parameter udáva na koho, teda na ktoré role sa táto politika bude aplikovať.

- USING

  Podmienka pre existujúce záznamy v tabuľke. Ak je táto podmienka splnená záznamy nebudú odfiltrované. Chyba nenastáva.

- WITH CHECK

  Táto podmienka sa kontroluje v prípade pridávania nového záznamu do tabuľky pomocou metódy Insert alebo Update. Daná podmienka sa kontroluje oproti novo vytváranému záznamu, nie oproti existujúcemu. V prípade nesplnenia podmienky nastáva chyba.

Funkcia z rady systémových informácii *current_user*, ktorá vracia meno prihláseného používateľa v aktuálnom kontextu.

```
1  ALTER TABLE users ENABLE ROW LEVEL SECURITY;
2
3  CREATE policy tenant_policy ON users for all
4  USING (tenant_id := current_user)
5  WITH CHECK (tenant_id := current_user);
```

Výpis kódu 5.2: Bezpečnostná politika na úrovni riadku

Zdroj: vlastné spracovanie

Na výpise kódu 5.2 vidíme bezpečnostnú politiku na úrovni riadku. Táto bezpečnostná politika vytvára filtrovanie záznamov na úrovni riadku, takže riadky, ktoré nespĺňajú danú podmienku vôbec nezískame. V tomto prípade získame zoznam používateľov, ktorí majú atribút *tenant_id* rovnaký ako hodnota prihláseného databázového používateľa.

5.5.3 Hibernate anotáčná trieda Where

Ďalším spôsobom ako implementovať filtrovanie podľa atribútu identifikátora nájomníka je použitie anotácie Where z frameworku Hibernate.

V kombinácii s použitím už zmieňovanej funkcie *current_user* je anotácia where vhodná pre použitie filtrovania.

Na druhú stranu sa obsah tejto anotácie nedá parametrizovať a preto je možné použiť len databázové funkcie na získanie identifikátora nájomníka.

```
1    @Where(clause="tenant_id=current_user")
2    class User(
3        @Column(name="tenant_id")
4        val tenantId: Long
5    )
```

Výpis kódu 5.3: Anotačná trieda @Where

Zdroj: vlastné spracovanie

Na výpise kódu 5.3 vidíme príklad použitia anotačnej triedy where. Táto anotácia sa pridá nad triedu databázovej entity a pri získavaní záznamov z databázy budú záznamy filtrované pomocou tejto podmienky.

5.5.4 Hibernate anotáčné triedy Filter, FilterDef a FilterDefs

Riešenie pomocou hibernate filtrov je založené na vložení podmienok do Hibernate session. Hibernate filtre umožňujú definovať klauzulu podobnú Where, ktorá sa ale dá parametrizovať. Aplikácia sa za behu môže rozhodnúť, či sa majú povoliť filtre a aké budú hodnoty ich parametrov.

Definovať filter môžeme pomocou anotačnej triedy FilterDef, alebo FilterDefs, ak chceme definovať viac filtrov naraz. Definícia filtra musí obsahovať názov filtra a zoznam parametrov. Parameter definujeme pomocou triedy ParamDef, kde sa nastavuje názov parametra a typ. Taktiež môžeme nastaviť prednastavenú podmienku, ktorá bude aplikovaná ak žiaden filter nemá nastavenú žiadnu podmienku.

Po definícii filtra musíme pridať filter na triedu, ktorú chceme filtrovať a pridáme podmienku, ktorá bude filtrovať našich nájomníkov.

```
@FilterDef(
    name = TENANT_FILTER,
    parameters = [
        ParamDef(name = TENANT_ID,
        type = "long")
    ]
)
@Filter(
    name = TENANT_FILTER,
    condition = "tenant_id=:TENANT_ID"
)
class User(
    @Column(name="tenant_id")
    val tenantId: Long
)
```

Výpis kódu 5.4: Definícia filtra s anotáciou @FilterDef a použitie filtra s anotáciou @Filter

Zdroj: vlastné spracovanie

Na výpise kódu 5.4 je zobrazené ako sa v Hibernate definujú a používajú filtre pomocou anotácii *@FilterDef* a *@Filter*. Filter je definovaný pomocou mena filtra a jeho parametrov. Pri použití filtru je potrebné uviesť už definovaný filter a podmienku filtra s prípadným parametrom.

Pre použitie viacerých filtrov naraz je možné použiť anotáciu *@FilterDefs* a jej parameter je zoznam filtrov, ktoré sa majú definovať.

Po definovaní filtra a aplikácii na entitu je potreba v hibernate session zapnúť alebo vypnúť filter respektíve filtre a prípadne nastaviť potrebné parametre.

```
session
    .enableFilter("TENANT_FILTER")
    .setParameter("TENANT_ID", getTenantId())
```

Výpis kódu 5.5: Zapnutie Hibernate filtra

Zdroj: vlastné spracovanie

Na výpise kódu 5.5 je ukázané ako sa dá definovaný a použitý filter zapnúť a prípadne nastaviť hodnotu parametra, ktorý má filter nastavený.

Kapitola 6

Návrh riešenia

Z požiadaviek na systém 5.3 je jasné, že pri návrhu riešenia je potrebné sa zamerať na bezpečnosť podnikových dát, jednoduchosť pri vývoji a nasadzovaní systému, možnú škálovateľnosť používateľov a podnikov a počítať s premenlivosťou podnikov, teda zlúčeniu alebo oddelenia divízii od skupín.

Pre správny návrh zdieľaného úložiska je dôležité si definovať, ako sa budú ukladať jednotlivé záznamy podnikov a následne, ako by prebiehalo zlúčenie, či oddelenie divízii. Pre uloženie a prístup k jednotlivým dátam bude použité filtrovanie záznamou podľa divízii a skupín divízii.

6.1 Mikroslužby a ich dáta

Mikroslužby sa vytvoria podľa modulov definovaných v sekcii 5.4.1 Moduly ako mikroslužby. Pre väčšiu separáciu modulov budú separované aj jednotlivé databázy pre moduly. Takéto riešenie poskytne možnosti rýchlejšieho vývoja, kompletného vývoja modulu od začiatku, či pridania nového modulu, alebo odobrania existujúceho modulu.

Zdieľaná databáza pre mikroslužby môže byť výhodná v minimalizovaní počtu strojov potrebných na prevádzku. Ale podľa [15] týmto zavádzame do systému miesto jediného zlyhania. Ak táto infraštruktúra spadne, môžu byť ovplyvnené viaceré mikroslužby naraz, ktoré by inak mohli fungovať naďalej, čo môže mať za následok katastrofický výpadok.

Používateľský účet a autentifikácia (UAA) bude jedna samostatná mikroslužba, ktorá bude slúžiť pre identifikáciu a autentifikáciu používateľov. Po identifikovaní a overení používateľa prebehne autorizácia, ktorá zistí aké všetky role, prístupy a povolenia má daný používateľ.

Ďalšia mikroslužba, ktorá bude potrebná je vstupná brána (AGW), ktorá bude slúžiť ako rozhranie pre klientov a bude mať za úlohu všetky požiadavky na aplikáciu presmerovať na jednotlivé mikroslužby, ktoré tieto požiadavky naplnia.

6.2 Uloženie skupín a divízii

Dáta budú v databáze uložené podľa toho či patria ku skupine, alebo patria ku divízii. Väčšina entít má navyše stĺpec s identifikátorom divízie. Entita *Používateľ* a *Divízia* zase stĺpec s identifikátorom skupiny. Používateľ patrí do skupiny divízii a má práva na jednotlivé divízie, ktoré budú vyplývať z organizačnej štruktúry.

Používateľova identita sa pri prihlasovaní do systému overuje voči všetkým používateľom naprieč všetkými skupinami divízii. Potom ako sa úspešne overia používateľove prihlasovacie údaje, tak má používateľ dosah na dáta, ktoré sú len v danej skupine divízií, do ktorej patrí.

Keďže divízia je najmenšia nedeliteľná časť podniku, v MISe bude použitá ako identifikátor záznamov pre rozlíšenie, ktorému podniku daný záznam patrí. Pri tomto použití ak nastane zlúčenie, alebo rozdelenie skupiny divízií, tak sa jednoducho divízie presunú na zlúčenú skupinu, alebo na novo vytvorenú rozdelenú skupinu divízií. Používateľ buď ostane, alebo sa takisto presunie do danej skupiny a práva na divízie sa aktualizujú podľa organizačnej štruktúry.

6.3 Zlúčenie a oddelenie divízií

Životný cyklus podniku je premenlivý, a preto je potrebné navrhnúť flexibilné riešenie, ktoré by poskytovalo možnosť podnikom používať manažérsky informačný systém aj pri zlúčení, alebo oddelení divízie.

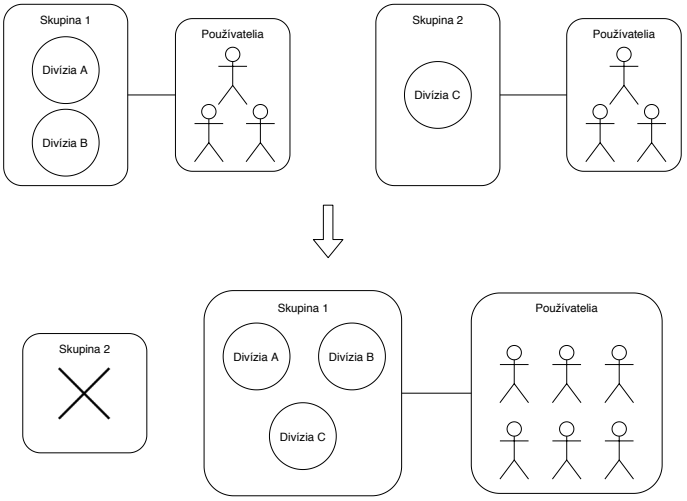
Pri zlúčení a oddelení sa musí dbať na podniky a ich podnikové dáta. Používateľom sa bude meniť príslušnosť k skupine divízií. Skupina divízií buď zanikne, vznikne, alebo ostane vytvorená, no stratí, alebo nadobudne jednu alebo viaceré divízie. Divízia podobne ako používateľ zmení príslušnosť k danej skupine.

6.3.1 Zlúčenie dvoch skupín divízií

Zlúčenie dvoch skupín divízií bude mať dva rôzne výstupy, pretože podnik môže byť zlúčený akvizíciou, alebo fúziou.

Pri fúzii sa dve skupiny zlúčia tak, že vytvoria úplne novú skupinu divízií a staré skupiny zanikajú. Všetky divízie, ktoré patrili pod jednu aj pod druhú skupinu, teraz patria pod novo vytvorenú skupinu. Všetci používatelia z oboch skupín budú patriť do novej skupiny.

Pri akvizícii sa dve skupiny zlúčia tak, že jedna skupina pohltí tú druhú. Druhá skupina zaniká, pretože sa všetky jej súčasti stávajú súčasťou prvej skupiny. Všetky divízie a používatelia odteraz patria pod prvú skupinu, ktorá nezanikla.

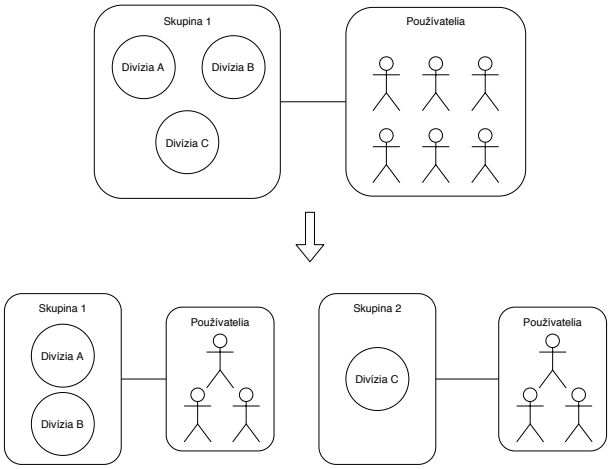


Obr. 6.1: Zlúčenie dvoch skupín divízií
Zdroj: vlastné spracovanie

Na obrázku 6.1 je ukázaná akvizícia skupiny 2 skupinou 1. Samotná skupina divízií zaniká a jej všetky súčasti sa presúvajú pod novú skupinu. Po zlúčení skupín budú všetci používatelia patriť pod jednu spoločnú skupinu divízií. Takisto jednotlivé divízie budú patriť pod jednu skupinu.

6.3.2 Rozdelenie skupiny divízií

K rozdeleniu divízií od skupiny môže dôjsť z dôvodu osamostatnenia divízie a vytvorenie vlastnej skupiny divízií pomocou reštruktulizácie podniku.



Obr. 6.2: Rozdelenie skupiny divízií
Zdroj: vlastné spracovanie

Na obrázku 6.2 je vyobrazené rozdelenie jednej skupiny divízií na dve skupiny, jednej existujúcej a druhej novo vzniknutej. Po rozdelení skupín divízií vzniká nová skupina, ktorá bude obsahovať oddelené divízie a používateľov, ktorí sa od danej skupiny divízií oddelili. Pôvodná skupina bude naďalej existovať s jediným rozdielom, že príde o divízie a používateľov, ktorí sa od nej oddelili.

Ak sa nejaká divízia, alebo viaceré divízie oddelia od skupiny divízií, musí nutne vzniknúť nová skupina divízií, pretože sa nejedná o akvizíciu ani o fúziu. Do tejto novo vzniknutej skupiny budú patriť všetky oddelené divízie a používatelia.

6.4 Uloženie a prístup k dátam v zdieľaných tabuľkách pomocou Hibernate filtrov

Dáta budú uložené v zdieľaných tabuľkách, ktoré budú mať identifikátor divízií, alebo skupín. Takto navrhnuté tabuľky s identifikátormi budú poskytovať škálovateľnosť systému a flexibilitu pri zlúčení a rozdelení skupín divízii.

Pre uloženie a prístup k dátam budú použité filtre z frameworku Hibernate. Toto filtrovanie bude mať na starosti, aby používateľ mohol pristupovať len k dátam, ktoré patria skupine divízií do ktorej patrí, alebo ktoré patria divíziam z jeho skupiny, na ktoré má práva.

Dokopy budú definované dva filtre, jeden pre skupinu divízii a druhý pre divíziu. Týmto spôsobom sa definované filtre budú môcť použiť na filtrovanie výsledkov požiadaviek na databázu, v ktorej sú uložené podnikové dáta v zdieľaných tabuľkách.

```
1     @FilterDef(
2         name = "GROUP_FILTER",
3         parameters = [
4             ParamDef(name = "groupId", type = "long")
5         ]
6     )
7     @Filter(
8         name = "GROUP_FILTER",
9         condition = "group_id=:groupId"
10    )
```

Výpis kódu 6.1: Filter skupiny

Zdroj: vlastné spracovanie

Na výpise kódu 6.1 je definovaný a použitý skupinový filter, ktorý filtruje entitu podľa atribútu identifikátora skupiny divízii. Podmienka definovaného filtra je rovnosť a bude zaručovať príslušnosť k danej skupine divízii.

Pre divíziu bude takisto definovaný filter, ktorý bude filtrovať záznamy, ktoré sú priradené k jednotlivým divíziam. Tieto záznamy patria pod jednu skupinu divízii a jednotlivý používatelia zo skupiny k nim budú mať prístup.

```
1     @FilterDef(
2         name = "DIVISION_FILTER",
3         parameters = [
4             ParamDef(name = "divisionIds",
5             type = "long")
6         ]
7     )
8     @Filter(
9         name = "DIVISION_FILTER",
10        condition = "division_id in (:divisionIds)"
11    )
```

Výpis kódu 6.2: Filter divízie

Zdroj: vlastné spracovanie

Na výpise kódu 6.2 je definovaný a použitý filter divízii, ktorý filtruje entitu podľa atribútu identifikátora divízie. Podmienka definovaného filtra je *in* a tá znamená výskyt danej hodnoty v zozname, ktorý bude priradený parametru *divisionIds*.

Pre prístup k session je použitá metóda entity manažéra *unwrap()*, ktorý sprístupní objekt pre session. Potom je možné aktivovať filter a nastaviť hodnoty parametrom. Ak sa prípadne nebude filtrovať podľa skupín alebo divízií, je treba deaktivovať filtre.

```
1     entityManager.unwrap(Session::class.java)
2         .enableFilter("GROUP_FILTER")
3         .setParameter("groupId", groupId)
4
5     entityManager.unwrap(Session::class.java)
6         .enableFilter("DIVISION_FILTER")
7         .setParameterList("divisionIds", divisionIds)
8
9     entityManager.unwrap(Session::class.java)
10        .disableFilter("DIVISION_FILTER")
11
12    entityManager.unwrap(Session::class.java)
13        .disableFilter("GROUP_FILTER")
```

Výpis kódu 6.3: Aktivácia a deaktivácia filtrov

Zdroj: vlastné spracovanie

Na výpise kódu 6.3 vidíme ako sa filtre zapnú, respektíve vypnú. Pomocou entity manažéra nastavíme tieto filtre v aktuálnej session, ktorá skončí až sa skončí pripojenie k databázovému serveru, alebo je aktuálna session zresetovaná.

Pri použití týchto filtrov bude zaistená bezpečnosť podnikových dát skupiny a divízií, do ktorej používateľ patrí a na ktoré má práva. Ak bude používateľ ukladať nové záznamy, budú uložené do databázových zdieľaných tabuliek s identifikátorom skupiny, alebo divízie.

Aby sa vývoj čo najviac uľahčil a nemuseli byť dané filtre používané pri každej metóde, ktorá bude vytvorená, bude tento filter odtienený od konkrétnej implementácie. Toto sa dá docieliť vytvorením základnej triedy repozitára, od ktorej budú jednotlivé repozitáre databázových entít preberať základné metódy a atribúty. Medzi jeden z atribútov bude patriť entity manažér, cez ktorého sa pristúpi k session a dané filtre budú aktivované.

```
1    class BaseRepositoryImpl: BaseRepository
2    {
3        val entityManager: EntityManager get() = let {
4            if(division){
5                em.unwrap(Session::class.java)
6                .enableFilter(BaseEntity.DIVISION_FILTER)
7                .setParameterList(
8                    BaseEntity.DIVISION_PARAMETER,
9                    divisionIds
10               )
11           }
12           if(group){
13               em.unwrap(Session::class.java)
14               .enableFilter(BaseEntity.GROUP_FILTER)
15               .setParameter(BaseEntity.GROUP_PARAMETER,
16                   groupId
17               )
18           }
19           return em
20       }
21   }
```

Výpis kódu 6.4: Základný repozitár s odtienenou aktiváciou filtra

Zdroj: vlastné spracovanie

Na obrázku 6.4 je návrh ako odtieniť aktiváciu hibernate filtrov. Jedným z možností by mohlo byť vytvorenie metódy, ktorá by aktivovala filter a použitie tejto metódy na začiatku transakcie. Toto riešenie je výhodnejšie, pretože objekt entity manažéra sa využíva pri tvorení databázových požiadavkov a teda ho netreba volať zvlášť, ako by sa volala samostatná metóda na aktiváciu filtra.

6.5 Použité technológie

Pri návrhu riešenia som predpokladal použitie nasledujúcich technológii. Jazyk pre prácu s objektovo-relačno databázou, programovací jazyk aplikačného servera, knižnicu na filtrovanie záznamov podľa skupín a divízii a službu, ktorá má využitie v cloudovom prostredí pri použití mikroslužieb.

Všetky tieto technológie sú zvolené zámerne a zaručujú bezpečnosť podnikových dát uložených v databáze manažérskeho informačného systému. S použitím týchto technológii bude možné implementovať návrh uloženia podnikových dát v manažérskom informačnom systéme.

6.5.1 Databáza PostgreSQL

PostgreSQL je open-source objektovo-relačný databázový systém [19], ktorý používa a rozširuje jazyk SQL. Má mnoho funkcií, ktoré bezpečne ukladajú a sprístupňujú dáta s možnosťou uloženia väčších dát a datových štruktúr.

PostgreSQL je vydávaný pod licenciou MIT a preto sa jedná o softvér s otvoreným zdrojovým kódom a jeho použitie nie je spoplatnené. PostgreSQL vytvára komunita programátorov a nie je vlastnený jednou organizáciou.

PostgreSQL je kompatibilný s operačnými systémami Linux, macOS aj Windows, aj keď je primárne vyvíjaný pre unixové systémy.

6.5.2 Programovací jazyk Kotlin

Kotlin je open-source programovací jazyk vyvinutý spoločnosťou JetBrains [20]. Kotlin je stručný, bezpečný, interoperabilný vďaka JVM a podporuje ho veľká škála vývojových prostredí. Pri použití na vývoj serverovej strany aplikácii Kotlin podporuje framework Spring a preto ponúka stručnujšiu implementáciu API.

Kotlin je syntakticky kompatibilný s programovacím jazykom Java a je navrhnutý tak, aby bol interoperabilný s knižnicami Javy.

6.5.3 Framework Hibernate

Hibernate je framework pre objektovo-relačné mapovanie aplikácie na databázu [21]. Zaoberá sa perzistenciou dát prostredníctvom JDBC. Tento framework je rýchly, spoľahlivý a škálovateľný. Takisto je veľmi konfigurovateľný a rozšíriteľný.

Hibernate používa anotácie a preto je kód, ktorý je vytvorený s použitím tohoto frameworku veľmi prehľadný a seba vysvetľujúci. Anotácia sa používajú primárne na mapovanie databázových entít, ale Hibernate podporuje veľkú škálu ďalších funkcií.

6.5.4 Eureka

Eureka je služba založená na REST [22], ktorá sa primárne používa v cloude AWS na lokalizáciu služieb za účelom vyrovnávania záťaže a zlyhania serverov strednej vrstvy. Služby sa pri spustení registrujú u Eureky a ak chcú komunikovať s ostatnými službami použijú Eureku na ich lokalizáciu. Táto vlastnosť sa dá použiť aj pre internú komunikáciu medzi mikroslužbami.

Kapitola 7

Implementácia prototypu uloženia dát

Podľa návrhu riešenia vytvorím prototyp uloženia dát, ktorý bude spĺňať všetky náležitosti a požiadavky na systém. Hlavnou funkciou implementácie prototypu je ukázať, že daný návrh je správny a bude sa dať použiť v budúcom vývoji.

Pri implementácii prototypu som sa zaoberal, ako budú podnikové dáta uložené v databáze, ako budú tieto uložené záznamy sprístupnené používateľovi manažérskeho informačného systém a ako bude používateľ vytvárať a ukladať nové záznamy do databáze.

7.1 Uloženie dát v databáze

Uloženie podnikových dát v databáze bolo navrhnuté tak, aby bola zaistená bezpečnosť podnikových dát jednotlivých podnikov a aby sa pri akvizíciach a fúziach podnikov podnikové dáta jednoducho sprístupnili pre nové a existujúce skupiny divízii, ktorým tieto dáta budú patriť.

Je dôležité aby používateľ patril do skupiny divízii a záznamy o faktúrach, projektoch, či dokumenty patrili pod divízie. Práve pri tomto rozdelení sa bude dať jednoducho reagovať na premenlivosť podnikov a nebude to ani časovo nákladné.

V databáze sa preto rozlišujú tri rôzne typy entít, ktoré sú uložené v databázových tabuľkách.

- Entita s atribútom skupiny divízii

  Táto entita spadá pod skupinu divízii a je spoločná pre všetky divízie danej skupiny. Prístup k tejto entite bude mať používateľ z ktorejkoľvek divízie.

- Entita s atribútom divízie

  Táto entita sa viaže na konkrétnu divíziu, na ktorú musí mať používateľ právo, ktoré vyplýva z organizačnej štruktúry. Používateľ, ktorý patrí do danej divízie bude môcť pristupovať k tejto entite.

- Entita, ktorá nemá ani jeden z atribútov skupina a divízia

  Nakoniec v databáze existujú entity, ktoré nemajú príznak skupiny alebo divízie. Sú to buď entity spoločné pre všetkých používateľov naprieč celej aplikácie (napríklad skupina divízii), alebo entita má väzbu na entitu, ktorá tento príznak má (napríklad pomocou identifikátora používateľa).

Tieto typy entít je potrebné rozlišovať z dôvodu pristupovania používateľa ku záznamom, vytvárania nových záznamov, alebo ak dôjde k zlúčeniu, alebo rozdeleniu skupiny divízie.

V takom prípade dané divízie a používatelia zmenia skupinu divízii ku ktorej patria, pri zlúčení, alebo rozdelení na existujúcu alebo novo vzniknutú. A žiaden záznam sa nezmení, práve kvôli návrhu typov entít.

7.2 Doťahovanie dát z databázy do aplikácie

Prístup k podnikovým dátam je sprostredkovaný pomocou Hibernate filtrov, ktoré poskytujú používateľom iba tie dáta, ku ktorým majú kompetencie.

7.2.1 Definícia filtru v základnej entite

Aby sa filter nemusel definovať pre každú entitu, alebo pre celý balíček (čo vyžaduje konfiguráciu v .xml súbore) je vytvorená trieda, od ktorej budú jednotlivé triedy dediť. Aby to fungovalo je táto nadtrieda označená anotáciou *@MappedSuperclass*. V základnej entite je navyše možnosť pridať atribúty ako identifikátor, atribút o informácii zmazania, alebo iné spoločné charakteristiky, ktoré môžu byť užitočné pri hľadaní dát podľa týchto atribútov.

```
1     interface BaseEntity<ID> {
2     companion object {
3         const val DIVISION_FILTER = "DIVISION_FILTER"
4         const val GROUP_FILTER = "GROUP_FILTER"
5         const val DIVISION_PARAMETER = "divisionIds"
6         const val GROUP_PARAMETER = "groupId"
7     }
8         val id: ID
9         var deleted: Boolean
10    }
```

Výpis kódu 7.1: Základná entita

Zdroj: vlastné spracovanie

Na výpise kódu 7.1 je zobrazená základná entita a jej atribúty identifikátor s generickým typom a príznak zmazania. Okrem atribútov základná entita obsahuje aj konštanty, ktoré budú následne použité pre zjednotenie názvu filtrov a ich parametrov.

Základná entita je generická z dôvodu flexibility datového typu identifikátora. Obsahuje takisto atribút *deleted*, pretože v manažérskom informačnom systéme je potreba dáta archivovať a uchovávať. Základná entita obsahuje navyše ešte konštanty potrebné pre filtre skupín a divízii.

Základná trieda pre entity s atribútom skupín definuje filter skupín a obsahuje identifikátor skupiny divízii. Definovaný filter má podmienku rovnosti a tento parameter sa doplní do aktúalnej session z informácii o prihlásenom používateľovi z autentikačného tokenu.

```
1    @MappedSuperclass
2    @FilterDef(name = BaseEntity.GROUP_FILTER,
3        parameters = [ParamDef(
4            name = BaseEntity.GROUP_PARAMETER,
5            type = "long")]
6    )
7    @Filter(name = BaseEntity.GROUP_FILTER,
8        condition = "group_id =
9            :${BaseEntity.GROUP_PARAMETER}"
10   )
11   abstract class GroupEntity<ID>
12   : BaseEntity<ID> {
13       abstract var groupId: Long
14   }
```

Výpis kódu 7.2: Entita s atribútom identifikátora skupiny

Zdroj: vlastné spracovanie

Na výpise kódu 7.2 je definovaný a použitý filter skupiny a použitá anotácia mappedSuperclass nad entitou s atribútom identifikátora skupiny. Táto entita rozširuje základnú entitu a má navyše identifikátor skupiny.

Trieda pre entity s atribútom divízie má obdobnú štruktúru ako trieda pre entity so skupinou. Jediný rozdiel je identifikátor dívízii a podmienka, ktorá je *in* namiesto rovnosti. Týmto spôsobom sa vyfiltrujú všetky entity s atribútom divízie z potrebného zoznamu divízii.

```
1    @MappedSuperclass
2    @FilterDef(name = BaseEntity.DIVISION_FILTER,
3        parameters = [ParamDef(
4            name = BaseEntity.DIVISION_PARAMETER,
5            type = "long")]
```

```
6       )
7       @Filter(
8           name = BaseEntity.DIVISION_FILTER,
9           condition = "division_id in
10              (:${BaseEntity.DIVISION_PARAMETER})"
11      )
12      abstract class DivisionEntity<ID>
13      : BaseEntity<ID> {
14          abstract val divisionId: Long
15      }
```

Výpis kódu 7.3: Entita s atribútom identifikátora divízie

Zdroj: vlastné spracovanie

Na výpise kódu 7.3 je definovaný a použitý filter divízii a použitá anotácia mappedSuperclass nad entitou s atribútom identifikátora divízie. Táto entita rozširuje základnú entitu a má navyše identifikátor divízie.

7.2.2 Pridanie hodnoty filtračného paremetru do session

Keďže cieľom práce bolo, čo najviac uľahčiť budúci vývoj a zamedziť chybám pri vývoji, snažil som sa filtre, čo najviac odtieniť, aby sa pri pridávaní nových entít do kódu na nič nezabudlo.

Vytvoril som preto triedu *BaseRepository*, ktorá slúži ako predloha pre ostatné repozitáre. Táto trieda obsahuje atribút *entityManager*, ktorý predtým ako sa k nemu pristúpi zapne, respektíve vypne filtre v závislosti na danej entite. Funguje to v zásade na princípe dedičnosti, kde podľa *supertriedy* zistíme či ide o triedu s filtrom alebo bez neho.

```
1       val entityManager: EntityManager get() = let {
2           val principal = SecurityContextHolder.getContext()
3               .authentication.principal as MisUserDto
4
5           when (domainClass.superclass) {
6               DivisionEntity::class.java ->
7                   em.unwrap(Session::class.java)
8                   .enableFilter(BaseEntity.DIVISION_FILTER)
9                   .setParameterList(BaseEntity.DIVISION_PARAMETER,
10                      principal.authorizedForDivisions
11                  )
12              GroupEntity::class.java ->
13                  em.unwrap(Session::class.java)
14                  .enableFilter(BaseEntity.GROUP_FILTER)
15                  .setParameter(BaseEntity.GROUP_PARAMETER,
16                      principal.groupId
17                  )
18              else -> {
```

```
19                  em.unwrap(Session::class.java)
20                      .disableFilter(BaseEntity.DIVISION_FILTER)
21                  em.unwrap(Session::class.java)
22                      .disableFilter(BaseEntity.GROUP_FILTER)
23              }
24          }
25          return em
26      }
```

Výpis kódu 7.4: Aktivovanie filtrov pomocou entity manažéra

Zdroj: vlastné spracovanie

Na výpise kódu 7.4 je implementácia aktivovania filtrov na základe typu entity. Entity manažér, ktorý sa touto metódou odošle má aktivovaný filter skupiny, filter divízie, alebo nemá aktivovaný žiaden filter, podľa toho o aký typ entity ide.

V repozitáry sa potom pracuje s *entityManager*om a dá sa používať ako normálny entity manager, takže vytvárame query s tým rozdielom, že filter je aktívny.

7.2.3 Prihlásenie do aplikácie

Keď sa používateľ prihlasuje do aplikácie, tak ešte nie je vytvorený žiaden autentikačný token a tým pádom nie sú žiadne informácie o prihlásenom používateľovi, pretože používateľ prihlásený nie je. Takisto pri prihlásení je vyžadované vyhľadávať medzi používateľmi spomedzi všetkým skupín a preto používame entity manager bez filtrov.

Po úspešnom prihlásení sa o používateľovi drží autentikačný token, ktorý má časovú platnosť a obsahuje aj informácie o používateľovi, napríklad do akej skupiny divízii patrí.

Pri prihlasovaní do aplikácie sú všetky filtre vypnuté, aby sa používateľ mohol vyhľadávať z pomedzi všetkých skupín.

```
1   val entityManagerWithoutFilters: EntityManager
2       get() = let {
3           em.unwrap(Session::class.java)
4               .disableFilter(BaseEntity.DIVISION_FILTER)
5           em.unwrap(Session::class.java)
6               .disableFilter(BaseEntity.GROUP_FILTER)
7           return em
8       }
```

Výpis kódu 7.5: Entity manažér bez filtrov

Zdroj: vlastné spracovanie

Na výpise kódu 7.5 je ukázaná výnimka pre špeciálny prípad vypnutia filtrov entity s identifikátorom skupiny. Toto filtrovanie je nežiadúce, pretože prihlášený používateľ ešte neexistuje a nevieme ktorú skupinu máme filtrovať.

7.3 Ukladanie dát z aplikácie do databázy

Použitie frameworku Hibernate na ukladanie dát z aplikácie by sa dalo implementovať pomocou insert a update listeneru [23]. Ale pretože existujú aj jednoduchšie spôsoby ako implementovať ukladanie dát s pridaním atribútu skupiny a divízie, rozhodol som sa využiť práve tieto riešenia.

7.3.1 Ukladanie entity s atribútom skupiny

Pri ukladaní entity s atribútom skupiny sa daný záznam uloží s identifikátorom skupiny, ktorý je naviazaný na prihláseného používateľa.

Anotácia *PrePersist* zaručí pridanie atribútu skupiny do aktuálne ukladanej novej, alebo aktualizovanej entity. Hodnota atribútu identifikátora skupiny bude vychádzať z identifikátora skupiny, ktorý je naviazaný na prihláseného používateľa.

```
1    @PrePersist
2    fun prePersist() {
3        groupId = SecurityUtils.getMisUser().groupId
4    }
```

Výpis kódu 7.6: Ukladanie záznamov pomocou anotácie @PrePersist

Zdroj: vlastné spracovanie

Na výpise kódu 7.6 je zobrazený kód, ktorý zaisťuje, že pri ukladaní entity s identifikátorom skupiny sa identifikátor skupiny uloží s hodnotou skupiny príslušného prihláseného používateľa.

Prvotný používateľ v novo vytvorenej skupine divízii bude vytvorený pomocou procesu vytvorenia nového podniku v aplikácii. Tento používateľ bude mať atribút identifikátora skupiny s hodnotou nove vytvorenej skupiny a bude mať nastavené administrátorské práva. S administrátorskými právami bude môcť založiť nových používateľov, ktorí budú patriť práve pod novo vytvorenú skupiny, t.j. pod rovnakú skupinu divízii ako prvotný používateľ.

7.3.2 Ukladanie entity s atribútom divízie

Keďže používateľ môže mať práva na viaceré divízie, nastáva otázka ako sa vysporiadať s uložením záznamu entity s týmto atribútom.

Automaticky priradiť hodnotu divízie teda nedáva zmysel ako je to pri entite so skupinou a tak pri vytváraní, alebo aktualizovaní danej entity sa posiela parameter identifikátor divízie.

Hodnota identifikátora sa jednoducho overí nájdením danej divízie a overením oproti používateľovým právam. Ak sa hľadaná divízia podľa identifikátora nenájde, alebo používateľ nemá na divíziu dostatočné práva, aplikácia vyhodí výnimku a nedôjde k vytvoreniu alebo aktualizovaniu entity.

```
1       val division = divisionRepository.findById(divisionId)
2           ?: throw entityNotFoundByIdException(
3               updateInvoiceDto.divisionId,
4               Division::class
5           )
6       if (!SecurityUtils.hasAccessToDivision(division.id)) {
7           throw AccessDeniedException(
8               ExceptionCode.ACCESS_TO_DIVISION_DENIED.name
9           )
10      }
```

Výpis kódu 7.7: Ukladanie záznamov entít s atribútom divízie

Zdroj: vlastné spracovanie

Na výpise kódu 7.7 je kód, ktorý popisuje ošetrenie ukladania entity s identifikátorom divízie. Ak daná divízia neexistuje, alebo nepatrí pod skupinu do ktorej patrí prihlásený používateľ nastane chyba. Ak divízia existuje a patrí pod danú skupinu, skontrolujú sa práva prihláseného používateľa na danú divíziu. Ak týmito právami nedisponuje, nastane chyba.

Kapitola 8

Testovanie prototypu

Pre zistenie, či návrh uloženia podnikových dát v manažérskom informačnom systéme funguje, je potrebné implementovaný prototyp otestovať.

Testovať budem dvoma rôznymi spôsobmi. Testované budú filtre pomocou Java unit testov a testované budú konkrétne endpointy aplikačného servera.

8.1 Unit testy

Unit testy sú testy, ktoré testujú individuálne časti kódu, ktoré sú izolované od zbytku aplikácii. Cieľom Unit testov je otestovať, či daný kód pracuje správne a podľa očakávaní.

Testovanie pomocou unit testov som rozdelil na dve časti. Testoval som repozitár, ktorý je nad entitou s identifikátorom skupiny, takže vo výsledku som testoval filter skupiny, ktorý je aplikovaný v tomto repozitáry.

Druhú časť som testoval repozitár, ktorý je nad entitou s identifikátorom divízie, takže som testoval filter divízie.

Vytvoril som zoznam unit testov, pripravil dáta potrebné k týmto testom a následne som testy implementoval a pustil ich nad pripravenými dátami.

8.1.1 Vytvorenie testov

Pre obe filtre som vytvoril sadu testov, ktorými plánujem otestovať funkcionalitu filtrovania záznamov podľa atribútov identifikátora skupiny a identifikátora divízie.

Budem testovať oba filtre a ich výstupom bude buď konkrétny záznam, ku ktorému sa snaží používateľ pristúpiť, alebo zoznam všetkých záznamov, ktorý sa dá filtrovať podľa atribútov a vzťahov.

Unit testy som vytvoril len pre doťahovanie dát z databázy, pretože testujem Hibernate filtre, ktoré som na to použil. Ukladanie dát do databázy budem testovať v sekcii 8.2 testovania endpointami.

Tabuľka 8.1: Zoznam testov filtrovania záznamov

#	Filter	Výstup	Pozitivita
1	Divízie	Záznam	+
2	Divízie	Záznam	-
3	Divízie	Zoznam záznamov	+
4	Divízie	Zoznam záznamov	-
5	Skupiny	Záznam	+
6	Skupiny	Záznam	-
7	Skupiny	Zoznam záznamov	+
8	Skupiny	Zoznam záznamov	-

V tabuľke 8.1 sú zaznamenané jednotlivé testy. Každý test je definovaný troma príznakmi, filtrom, ktorý sa bude testovať, jeho výstup a pozitivita. Ak je pozitivita kladná, predpokladám, že výsledok testu bude kladný, teda že používateľ má právo vidieť daný záznam, alebo záznamy. Ak je pozitivita negatívna, predpokladám, že výstup bude prázdny zoznam, alebo informácia o neexistujúcom zázname. Výstup bude prázdny aj v tom prípade, ak záznamy existujú, ale nepatria danej skupine alebo divízie, na ktoré má používateľ práva.

8.1.2 Testovanie filtra skupín

Pre testovanie filtra skupín som vytvoril viacero testov, ktoré testujú počet a hodnoty záznamov. Vytvoril som pozitívne testy, keď sa používateľ snaží pristupovať k dátam, ktoré patria pod skupinu divízii, do ktorej patrí.

Vytvoril som takisto testy, ktoré sú negatívne a to také, keď sa používateľ snaží pristúpiť ku existujúcim záznamom v databáze, ale tieto záznamy patria inej skupine divízii ako do ktorej používateľ patrí.

8.1.3 Testovanie filtra divízii

Obdobne ako pri testovaní filtra skupín som vytvoril viacero druhov testov, aby som overil funckionalitu a správnosť Hibernate filtrov.

Vytvoril som testy, ktoré overujú či si daný používateľ môže zobraziť záznamy s divíziou, na ktoré nemá právo.

Po implementácii testov na filter divízie som skúšal testy spustiť, ale výstupom testov bola syntaxová chyba.

```
Caused by: org.hibernate.exception.SQLGrammarException: could not extract ResultSet
    ... 60 more
Caused by: org.postgresql.util.PSQLException: ERROR: syntax error at or near ")"
```

Obr. 8.1: Syntaxová chyba pri preklade do jazyka SQL
Zdroj: vlastné spracovanie

Na obrázku 8.1 je zachytená chyba pri preklade definovaného Hibernate filtra a jeho podmienky na jazyk SQL.

Chyba nastala keď Hibernate interpretoval podmienku nastavenú vo filtry na jazyk SQL ako *where division_id in ()*. PostgreSQL berie klauzulu *in* s prázdnym zoznamom hodnôt ako syntaxovú chybu.

Obr. 8.2: Klauzula so syntaxovou chybou
Zdroj: vlastné spracovanie

Obrázok 8.2 zobrazuje debug konzolu s prekladom klauzule do jazyka SQL. S touto klauzulou ale v PostgreSQL nastáva syntaxová chyba.

Potreboval som túto chybu odstrániť z implementovaného problému a prišiel som na dve rôzne riešenia.

Prvým riešením by bolo túto klauzulu nepridávať do výslednej interpretácii do jazyka SQL. Týmto spôsobom by ale používateľ videl zoznam všetkých záznamov, pre všetky divízie. Chyba už nenastala, ale tento výstup nie je žiadaný.

Druhé riešenie, ktoré som aj vo finálnom prototype implementoval, bolo pridanie nesplniteľnej podmienky namiesto pôvodnej klauzule. Pridaním nesplniteľnej podmienky bude výstup vždycky prázdny zoznam záznamov. Tento výstup je prvoplánový, pretože používateľ, ktorý nemá právam na žiadnu divíziu, nemá nárok na zobrazovanie záznamov entity s atribútom identifikátora divízie.

Nesplniteľná podmienka môže vyzerať napríklad ako podmienka rovnosti dvoch prirodzených čísel, ktoré nie sú rovné.

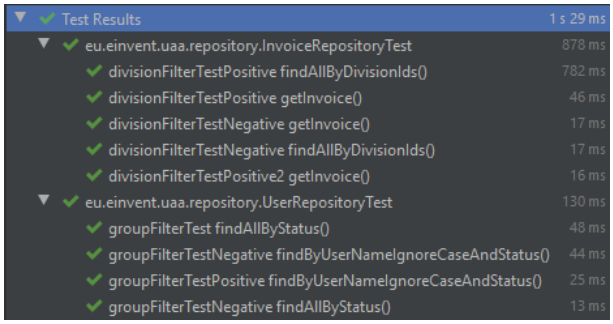
8.1.4 Výsledky testovania

Pri skúšaní implementovaných testov som narazil na menší problém, ktorý popisujem v sekcii 8.1.3. Okrem toho som nenarazil na žiadne problémy a filtrovanie fungovalo podľa očakávaní.

Obrázok 8.3 zachytáva výsledky testov, ktoré všetky prebehli úspešne. Predpokladané výsledky sa zhodujú s reálnymi výsledkami filtrovania. Použitie Hibernate filtrov v prototype som overil Unit testami, ktoré sú izolované od ostatného prostredia.

8.2 Testovanie API endpointov

Na testovanie jednotlivých endpointov som použil aplikáciu Postman. Je to nástroj pre prácu s API a pomáha pri vývoji. Dajú sa v ňom prevolávať jednotlivé endpointy s rôznymi parametrami, autorizáciou a ďalšími možnosťami.



Obr. 8.3: Výsledky Unit testov
Zdroj: vlastné spracovanie

Navyše Postman natívne podporuje testovanie prevolanie endpointov, a preto som sa rozhodol pri volaní jednotlivých endpointov vytvoriť testy, ktoré som následne spustil v prostredí Postmana.

Tabuľka 8.2: Zoznam testov API endpointov

#	**Názov testu**	**Stav odpovede**
1	Prihlásenie používateľa do aplikácie	200 OK
2	Dotiahnutie zoznamu záznamov entity s identifikátorom skupiny	200 OK
3	Dotiahnutie záznamu entity s identifikátorom skupiny	200 OK
4	Dotiahnutie cudzieho záznamu entity s identifikátorom skupiny	404 Not Found
5	Dotiahnutie zoznamu záznamov entity s identifikátorom divízie	200 OK
6	Dotiahnutie záznamu entity s identifikátorom divízie	200 OK
7	Dotiahnutie cudzieho záznamu entity s identifikátorom skupiny	404 Not Found
8	Dotiahnutie cudzieho z vlastnej skupiny záznamu entity s identifikátorom skupiny	404 Not Found
9	Uloženie entity s identifikátorom skupiny	201 Created
10	Uloženie entity s identifikátorom divízie	201 Created
11	Uloženie entity s identifikátorom cudzej divízie	403 Forbidden

V tabuľke 8.2 sú zaznamenané jednotlivé testy, ktoré som testoval pomocou nástroja Postman. Tieto testy sú nazvané podľa toho akú funkcionalitu mali testovať a stav odpovede udáva predpokladanú odpoveď z aplikačného servera pri posielaní daných požiadaviek.

8.2.1 Prihlásenie sa do aplikácie

Jeden z prvých testov, ktoré som vytvoritl bol test prihlasovania používateľa do aplikácie. Očakávam, že sa používateľovi podarí úspešne prihlásiť a ako odpoveď zo servera mu príde autentizačný token spolu s priradenou skupinou a právami na divízie.

8.2.2 Vyhľadávanie záznamov s atribútom skupiny

Keď bude používateľ vyhľadávať záznamy entít, ktoré majú identifikátor skupiny, bude použitý filter skupiny. Ten som otestoval už Unit testami a teraz otestujem rozhranie, ktoré bude využívať používateľ manažérskeho informačného systému.

Pre záznamy s atribútom skupiny som testoval tri rôzne testy. Jeden test na vyhľadávanie zoznamu používateľov. Jeho výsledkom by mal byť zoznam všetkých používateľov, ktorí patria do rovnakej skupiny ako prihlásený používateľ.

Ďalšie testy sú na prístup k dátam používateľa. Jeden z nich má za úlohu otestovať správnosť výsledku a druhý otestovať, že prihlásený používateľ nemôže pristupovať k cudzím podnikovým dátam.

8.2.3 Vyhľadávanie záznamov s atribútom divízie

Používateľ bude môcť vyhľadávať záznamy pre entitu s atribútom divízie a bude mať prístup len ku tým záznamom, ktoré sú vlastnené divíziami, na ktoré má prihlásený používateľ právo. Navyše bude môcť používateľ tieto záznamy filtrovať pomocou identifikátora divízie.

Obdobne ako pri vyhľadávaní záznamov s atribútom skupiny som testoval vyhľadávanie zoznamu všetkých faktúr. Výsledok tohoto testu bol zoznam všetkých faktúr, na ktoré mal používateľ právo.

Pre prístup k detailu faktúry som testoval jednu faktúru, ku ktorej mal prihlásený používateľ právo a ďalšie dve na ktoré práva nemal. Jedna z faktúr, na ktoré právo nemal bola z divízie skupiny, ktorá je zhodná s prihláseným používateľom a druhá divízia je z inej skupiny. K obom týmto faktúrom prihlásený používateľ nemá prístup a preto je predpokladaná odpoveď chybová hláška s nenájdením danej faktúry.

8.2.4 Ukladanie záznamov s atribútom skupiny

Ukladanie záznamov pre entitu s identifikátorom skupiny je implementované pomocou knižničnej funkcie *PrePersist*. Pri ukladaní záznamu je atribútu skupiny priradená hodonta skupiny prihláseného používateľa a následne uložená do databázy spolu s ukladaným záznamom.

Prihlásený používateľ vytvorí nového používateľa, ktorému bude priradená rovnaká skupina ako skupina prihláseného používateľa. Novo vytvorený

záznam je uložený s identifikátorom skupiny a teda nový používateľ patrí pod rovnakú skupinu ako používateľ, ktorý ho vytvoril.

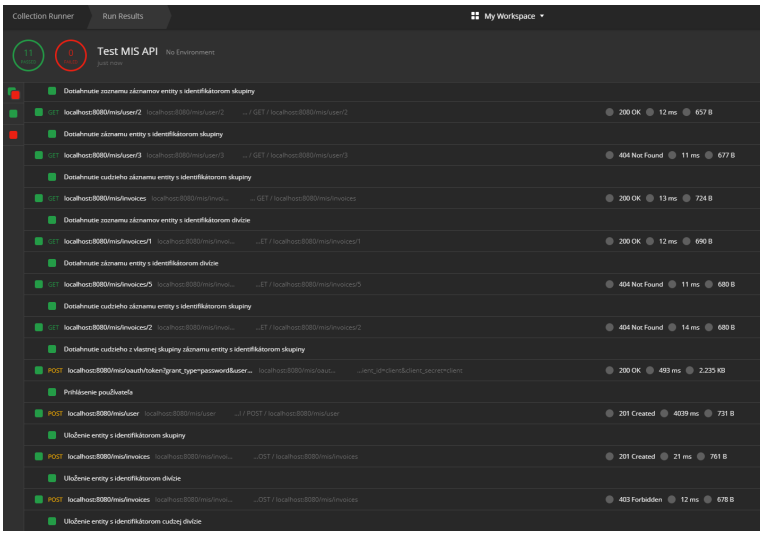
8.2.5 Ukladanie záznamov s atribútom divízie

Pre entitu s identifikátorom divízie sa nový záznam ukladá s hodnotou identifikátora divízie, ktorý je posielaný v požiadavke a vyberá ho prihlásený používateľ. Preto sa musí skontrolovať, či daná divízia existuje, patrí pod rovnakú skupinu a prihlásený používateľ má na ňu právo. Po týchto kontrolách sa záznam uloží s díviziou, ktorú prihlásený používateľ zvolil.

Nová faktúra sa vytvorí pomocou požiadavky, v ktorej bude atribút identifikátora divízie. Táto hodnota sa overí a daná faktúra bude uložená s identifikátorom divízie.

8.2.6 Výsledky testovania

Všetky testy prebehli úspešne a boli dosiahnuté všetky očakávané stavy. Implementovaný prototyp podľa návrhu riešenia je správny a správa sa podľa očakávania. Filtrovanie funguje a používateľ má prístup, len ku vlastným podnikovým dátam.



Obr. 8.4: Výsledky testov API endpointov
Zdroj: vlastné spracovanie

Na obrázku 8.4 sú výsledky testov API endpointov, ktoré boli uskutočnené pomocou nástroja Postman. Všetky testy zo zoznamu testov 8.2 boli otestované s úspešným výsledkom.

Kapitola 9

Zhodnotenie prínosov a budúci rozvoj

Navrhnuté riešenie a implementovaný prototyp má radu prínosov a možností pre ďalší rozvoj. V tejto kapitole tieto prínosy popíšem a bližšie priblížim a zamyslím sa nad tým, čo by sa dalo vylepšiť a ako postupovať pri aplikácii tohoto riešenia do praxe.

9.1 Prínosy uloženia podnikových dát v manažérskom informačnom systéme

Toto riešenie prináša hneď niekoľko výhod, z ktorých budú benefitovať nie len poskytovatelia produktu manažérsky informačný systém, ale aj zákazníci a jeho koncoví používatelia.

Pre poskytovateľa manažérského informačného systému je veľkým prínosom možnosť škálovania tohoto systému. Manažérsky informačný systém bude založený a v prevádzke v podstate na počkanie, práve kvôli spôsobu návrhu jeho riešenia. Takto bude možné rozšírenie základne existujúcich zákazníkov veľmi jednoduché, rýchle a málo nákladné.

Udržovať manažérsky informačný systém tak, aby bol moderný a aktuálny s dobou a v súlade s legislatívnymi zmenami, bude efektívnejšie a proces zmien a nasadzovaní aktualizácií bude menej zasahovať do používania tohoto systému.

S možnosťou škálovania a poskytovania systému pre viacero zákazníkov prichádza konkurenčná výhoda, pretože je toto riešenie navrhnuté na zdieľanie systému medzi zákazníkmi. Poskytovateľ tohoto riešenia bude mať možnosť znížiť cenu manažérskeho informačného systému, pretože systém je uspôsobený na vyšší počet podnikov a týmto sa môže odlíšiť od konkurencie. Ak sa poskytovateľ systému rozhodne pre takúto zmenu, pre zákazníkov to bude dobrá

správa, pretože počiatočná investícia do manažérskeho informačného systému bude nižšia.

Zákazníci a koncoví používatelia systému určite ocenia dostupnosť systému z rôznych zariadení a lokácii. Používateľ nemusí riešiť bezpečnostné prístupy na lokálnu sieť, kde je prevádzkovaný systém, ale o bezpečnosť je postarané zo strany poskytovateľa, keďže manažérsky informačný systém bude prevádzkovaný v cloude.

9.2 Použitie prototypu a jeho rozvoj

Implementovaný prototyp je možné použiť ihneď pri vývoji manažérskeho informačného systému. Pri vhodnej aplikácii na ďalšie databázove entity, bude toto riešenie zaisťovať bezpečnosť a dôvernosť podnikových dát.

Pri použití prototypu bude mať prihlásený používateľ prístup len k podnikovým dátam skupiny, do ktorej patrí. Ostatné dáta budú pre neho nedostupné. Všetky záznamy, ktoré sú naviazané na divízie a prihlásený používateľ na ne nebude mať práva, mu tieto záznamy taktiež nebudú prístupné.

Vzhľadom na požiadavky na systém, ktoré sú kladené, funguje prototyp správne a pri prípadných komplikáciach by sa dalo toto riešenie poupraviť. Napríklad pre funkciu auditovania viacerých skupín by sa mohla filtru skupín zmeniť podmienka z rovnosti jednej hodnoty na zoznam hodnôt, aby auditor mohol vidieť viaceré skupiny naraz. Potom by určite bolo potrebné ošetriť nemožnosť manipulovať s dátami pre auditorov a ďalšie bezpečnostné prvky.

V sekcii 5.5 som sa zaoberal rôznymi možnosťami ako implementovať zdieľaný systém pomocou zdieľaných tabuliek a myslím si, že aj ostatné riešenia, hlavne použitie PostgreSQL bezpečnostnej politiky na úrovni riadkov, sú veľmi vhodné na použitie pri implementácii ďalších mikroslužieb, o ktoré by sa mohol manažérsky informačný systém v budúcnosti rozrásť.

Záver

Cieľom práce bolo navrhnúť uloženie podnikových dát pre manažérsky informačný systém, ktorý bude prevádzkovaný v cloudovom prostredí, bude škálovaný pre ľubovoľný počet podnikov, ktoré budú tento systém zdieľať. Toto riešenie malo zaistiť bezpečnosť podnikových dát jednotlivých podnikov.

Pri skúmaní možných multi-tenancy riešení som prišiel na to, že všetky riešenia majú nejaké výhody a nevýhody a je na konkrétnom projekte a probléme, ktoré z týchto riešení je vhodné použiť. Bohužiaľ ani jedno z týchto riešení neposkytovalo flexibilitu pri ukladaní podnikových dát, tak aby sa jednotlivé skupiny a divízie mohli zlučovať a oddeľovať ľubovoľne, v dôsledku podnikových akvizícii a fúzii.

Preto som vymyslel uloženie podnikových dát v databáze, ktorá je zdieľaná medzi viacerými podnikmi a docielil som to rozšírením multi-tenancy riešenia, ktoré ukladá záznamy do zdieľaných tabuliek s identifikátorom nájomníkov, v tomto prípade podnikov. Rozšírené riešenie berie do úvahy bezpečnosť podnikových dát a ukladá podnikové dáta tak, aby pri premenlivosti podnikov bol zaručený jednoduchý presun podnikových dát medzi podnikmi.

Vytvorený prototyp umožňuje ukladať a pristupovať k podnikovým dátam bezpečne pre jednotlivých používateľov systému. Každý z podnikov má vlastné podnikové dáta, ku ktorým nemajú podniky navzájom prístup, ale všetky sú uložené v zdieľaných tabuľkách a prístupné z aplikácie manažérsky informačný systém.

Pri použití tohoto riešenia bude systém dobre škálovateľný pre vyšší počet podnikov a bude lepšie udržateľný a aktuálny voči legislatívnym zmenám, pretože sa zefektívni proces nasadozvania nových verzií.

Implementovaný prototyp som otestoval v rámci diplomovej práce pomocou Unit testov a testov API endpointov a navyše je oskúšaný v praxi na projekte, pri vývoji manažerského informačného systému.

Pri pokračovaní implementácie manažérskeho informačného systému je naplánované rozšírenie prototypu riešenia o ďalšie entity a mikroslužby.

V budúcnosti je možnosť rozšíriť toto riešenie o podporu používateľov, ktorí budú mať auditorské práva a budú môcť pristupovať k podnikovým dátam viacerých podnikov, kvôli vykonávaniu auditu daných podnikov.

Pri možnom rozšírení o ďalšie mikroslužby je možnosť použiť rovnaký prístup ako pri implementácii prototypu, alebo vybrať jednu zo zvažovaných možností a tie uplatniť pri implementácii ďalších mikroslužieb.

Literatúra

[1] L. Gála, Z. Šedivá, J. Pour: *Podniková informatika: Počítačové aplikace v podnikové a mezipodnikové praxi*. Grada, 2015, ISBN 978-80-247-5457-4.

[2] J. Procházka, C. Klimeš: *Provozujte IT jinak: Agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb*. Grada, 2011, ISBN 978-80-247-4137-6.

[3] R. Hindls, R. Holman, S. Hronová a kol.: *Ekonomický slovník*. C. H. Beck, 2003, ISBN 80-7179-819-3.

[4] M Čermák: *CIA: Důvěrnost-Integrita-Dostupnost*. [online], [cit. 25.5.2020]. Dostupné z: `https://www.cleverandsmart.cz/duvernost-integrita-dostupnost/`

[5] *Business encyclopedia*. [online], [cit. 14.4.2020]. Dostupné z: `https://www.shopify.com/encyclopedia/management-information-systems-mis`

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia: *Above the Clouds: A Berkeley View of Cloud Computing*. [online], 2009, [cit. 6.4.2020]. Dostupné z: `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf`

[7] C.Landis, D. Blacharski: *Cloud Computing Made Easy*. Createspace Independent Pub, 2013, ISBN 978-1482779424.

[8] T. Vitvar: *Cloud Architectures*. [online], [cit. 24.2.2020]. Dostupné z: `http://mdw.vitvar.com/pdf/lecture9-1p.pdf`

[9] Google, Inc.: *F1: A Distributed SQL Database That Scales*. [online], [cit. 24.2.2020]. Dostupné z: `http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41344.pdf`

[10] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov: *Microservices: The Journey So Far and Challenges Ahead. IEEE Software*, ročník 35, č. 3, 2018: s. 24–35, ISSN 0740-7459.

[11] A. Hughes: *On Premise vs. Cloud: Key Differences, Benefits and Risks.* [online], [cit. 15.4.2020]. Dostupné z: `https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud`

[12] e-invent s.r.o.: *Manažerský informační systém.* [online], [cit. 2.4.2020]. Dostupné z: `https://www.e-invent.eu/produkty/manazersky-informacni-system`

[13] P. Niemeyer, D. Leuck: *Learning Java.* O'Reilly Media, 2013, ISBN 978-1449319243.

[14] D. Spinellis: *The Bad Code Spotter's Guide.* 2006, [online], [cit. 2.4.2020]. Dostupné z: `https://www.informit.com/articles/article.aspx?p=457502&seqNum=5`

[15] S. Newman: *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media, 2015, ISBN 1491950358.

[16] M. Valenta: *Jazyk SQL - DML, DCL, TCL.* [online], [cit. 15.5.2020]. Dostupné z: `https://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs_05_sql_3.pdf`

[17] R. Haas: *Security Barrier Views.* [online], [cit. 24.2.2020]. Dostupné z: `http://rhaas.blogspot.com/2012/03/security-barrier-views.html`

[18] The PostgreSQL Global Development Group: *Row Security Policies.* [online], [cit. 24.2.2020]. Dostupné z: `https://www.postgresql.org/docs/12/ddl-rowsecurity.html`

[19] PostgreSQL Global Development Group: *PostgreSQL.* [online], [cit. 24.3.2020]. Dostupné z: `https://www.postgresql.org/about/`

[20] Kotlin Foundation: *Using Kotlin for Server-side Development.* [online], [cit. 26.3.2020]. Dostupné z: `https://kotlinlang.org/docs/reference/server-overview.html`

[21] Hibernate: *Hibernate ORM. Your relational data. Objectively.* [online], [cit. 26.3.2020]. Dostupné z: `https://hibernate.org/orm/`

[22] Netflix: *Eureka.* [online], [cit. 26.3.2020]. Dostupné z: `https://github.com/Netflix/eureka`

[23] Red Hat, Inc.: *Chapter 12. Interceptors and events.* [online], [cit. 12.5.2020]. Dostupné z: `https://docs.jboss.org/hibernate/core/3.3/reference/en/html/events.html`

Dodatok **A**

Slovníček pojmov

Autentifikácia Autentifikácia je proces overenia identity používateľa. Zisťuje sa o akého používateľa ide.

Autorizácia Autorizácia je proces overenia oprávnení pre danú úlohu. Zisťuje sa či na to používateľ má právo.

Cloud Cloud je pojem, ktorý zahŕňa všetko, softvér a hardvér v dátovom centre.

Cloud computing Cloud computing je termín, ktorý popisuje dostupnosť softvérových a hardvérových zdrojov na požiadanie cez internet.

Dátové centrum Dátové centrum je označenie pre špecializované priestory pre umiestnenie počítačovej techniky serverového typu, ktoré sú určené k nepretržitému prevádzkovaniu.

Multi-tenancy Multi-tenancy je odborný termín popisujúci zdieľané prostredie a zdroje, ktoré slúžia viacerým nájomníkom či konzumentom.

Nájomník Nájomník je skupina používateľov, ktorí zdieľajú špecifické časti softvéru a to dáta, konfigurácie, správu používateľov a nefunknčných vlastností.

Divízia Divízia je časť podniku identifikovaná IČ. Zvyčajne každá divízia odpovedá jednému IČ, teda jednej právnickej alebo fyzickej osobe. Je ale možné, aby sa jedna právnická či fyzická osoba rozdelila na viacero divízií, typicky podľa rôznych podnikateľských činností daného subjektu.

Skupina divízií Skupina divízií je podnik, ktorý môže mať jednu alebo viac divízií.

Token Token, alebo inak bezpečnostný token, je spôsob overenia identity používateľa a môže byť hardvérový alebo softvérový. Softvérový token môže byť po overení nositeľom ďalších informácii ako rolí a práv.

Databázová schéma Databázová schéma je popis databázy zapísaný vo formálnom jazyku. Obsahuje aj databázové tabuľky, v ktorých sú uložené dáta.

Samostatné databázy pre každého nájomníka Každý nájomník má samostatnú databázu, v ktorej má vlastné databázové schémy, tabuľky a podnikové dáta.

Zdieľaná databáza, samostatné schémy Databáza je zdieľaná, ale každý nájomník má vlastnú databázovú schému, v ktorej má vlastné tabuľky a podnikové dáta.

Zdieľané tabuľky Každý nájomník má označené riadky v zdieľaných tabuľkách, ktoré mu patria svojím identifikátorom.

Dodatok B

Zoznam použitých skratiek

AGW API Gateway

API Application programming interface

AWS Amazon Web Services

CRM Customer relationship management

DB Database

IČ Identifikačné číslo

JDBC Java Database Connectivity

JVM Java Virtual Machine

MIS Management information system

NIST National Institute of Standards and Technology

ORM Object-relational mapping

REST Representational state transfer

SaaS Software as a Service

SOA Service oriented architecture

SQL Structured Query Language

UAA User account and authentication

UUID Universally unique identifier

VPN Virtual Private Network

WAR Web Application Resource

Dodatok C

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text`
  - `DP_Maros_Karas.pdf` ..... text práce vo formáte PDF