VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Návrh a implementace webové aplikace pro podporu rozhodování společnosti působící na komoditním trhu

Design and Implementation of Web-based Decision Support System for Company Operating in the Commodity Market

Student: Bc. David Jamárik
Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2021

VŠB – Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. David Jamárik**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T017 Informatika v ekonomice

Téma: Návrh a implementace webové aplikace pro podporu rozhodování společnosti působící na komoditním trhu

Design and Implementation of Web-based Decision Support System for Company Operating in the Commodity Market

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoretická a metodická východiska implementace webových aplikací
3. Analýza současného stavu a vymezení požadavků
4. Návrh a implementace softwarového řešení
5. Migrace dat a testování
6. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

BANKS, Alex a Eve PORCELLO. *Learning React: Modern Patterns for Developing React Apps.* London: O`Reily, 2020. ISBN 1492051721.
HOBERMAN, S., D. BURBANK and Ch. BRADLEY. *Data Modeling for the Business: A Handbook for Aligning the Business with IT Using High-Level Data Models.* New Jersey: Technics Publication, 2009. ISBN 9780977140077.
WALLS, Craig. *Spring in action.* Fourth edition. Shelter Island, NY: Manning Publication, 2015. ISBN 9781617291203.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 20.11.2020

Datum odevzdání: 23.04.2021

Ing. Petr Rozehnal, Ph.D.	doc. Ing. Vojtěch Spáčil, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem celou diplomovou práci, včetně všech příloh, vypracoval samostatně a uvedl všechny použité podklady a literaturu.

Zároveň bych chtěl poděkovat svému vedoucímu diplomové práce, Ing. Vítězslavu Novákovi, Ph.D., za vedení diplomové práce, cenné rady a připomínky.

V Ostravě dne 23. 4. 2021 .....

Bc. David Jamárik

Obsah

1 Úvod

Společnosti čelí v dnešní době velké konkurenci, ať už se jedná o jakýkoliv trh. Neustále se snaží něco optimalizovat, hledat nové nástroje a tím zvyšovat svou konkurenceschopnost na trhu. Mezi moderní pomocníky patří také systémy pro podporu rozhodování. Primárním cílem tohoto typu systému je poskytnout pomoc při analýze podnikového prostředí a tím získat lepší výchozí pozici při rozhodování v oblasti řízení obchodních či marketingových strategií společnosti.

Zatímco někteří odborníci se brání myšlence důvěřovat řešením počítačového softwaru pro podporu rozhodování, já si osobně myslím, že je to vhodná cesta pro pochopení klíčových trendů a ukazatelů. Pokud je takový systém správně zakomponován do prostředí společnosti, může velmi efektivně pomoci při analýze, plánování a rozhodování.

Problém, který v současné době trápí vedení společnosti eCENTRE, a.s., je špatná orientace v podnikových procesech a datech. Řízení a zpracování jednotlivých procesů je rozděleno do relativně velkého počtu malých desktopových aplikací, které jsou následně obhospodařovány většinou malou skupinou specialistů na danou problematiku.

Společnost také čelí velké konkurenci na trhu elektrické energie a zemního plynu, na kterých působí, a je proto nutné dobře reagovat na změny jiných subjektů. Z uvedených důvodů bylo společnosti nabídnuto vytvoření nového analytického nástroje, který vnukne lepší pohled do podnikového prostředí a zefektivní řízení obchodních a strategických cílů.

Hlavním cílem této diplomové práce je tedy navrhnout a implementovat webovou aplikaci sloužící jako nástroj pro podporu rozhodování společnosti. Práce popisuje celý průběh tohoto projektu, včetně teoretických poznatků, návrhu a samotné implementace webové aplikace.

V první kapitole jsou popsána teoretická a metodická východiska návrhu a implementace webových aplikací. Součástí této kapitoly jsou vysvětleny všechny použité technologie, principy a metody, které byly využity v praktické části.

Následující kapitola je věnována analýze současného stavu společnosti, její infrastruktuře a technologiím, které používá. V této kapitole jsou taktéž uvedeny definované cíle a požadavky, které musí výsledná aplikace zajistit.

Třetí kapitola se věnuje návrhu a implementaci webové aplikace. Jedná se o stěžejní část celé diplomové práce. Celé řešení je nutné navrhnout tak, aby splňovalo všechny definované požadavky. Je tedy potřeba vybrat správné technologie a přesně specifikovat všechny funkcionality aplikace a navrhnout uživatelsky přívětivé grafické rozhraní. Navržené řešení je následně implementováno do prostředí společnosti a napojeno na ostatní interní a externí systémy.

V předposlední kapitole je popsán proces migrace dat do systému a testování celé aplikace, zda správně fungují všechny funkcionality a jsou pokryty všechny požadavky.

V závěru práce budou shrnuty všechny poznatky získané pří návrhu a implementaci. V této kapitole je uvedeno i vyhodnocení dosažení jednotlivých cílů a požadavků včetně zhodnocení implementovaného řešení. Případně také návrhy na zlepšení nebo plánované další etapy vývoje.

2 Teoretická a metodická východiska implementace webových aplikací

2.1 Systém pro podporu rozhodování

V dnešní době se organizace stále častěji pokoušejí činit informovanější rozhodnutí, aby zlepšily své konečné výsledky. Někteří tyto snahy označují jako Business Intelligence, jiní zase jako datovou analytiku nebo analytiku obecně. V obou případech je motivem shromáždit správné informace a korektní modely tak, aby pochopily, co se v daném podniku děje, a zvážily problémy z různých úhlů pohledu. Cílem je tedy poskytnout nejlepší podklady pro osoby s rozhodovací pravomocí.

Jedním ze způsobů, jak vytvářet podkladové informace a modely, je použití systémů na podporu rozhodování (dále pouze DSS – z angličtiny Decision Support System). DSS jsou převážně podnikové počítačové systémy, které shromažďují informace z různých zdrojů, pomáhají při organizaci a analýze dat a usnadňují rozhodování podniku díky modelování výsledků. Jinými slovy, tyto systémy umožňují odpovědným osobám získat přístup k příslušným údajům v rámci celé organizace, aby si mohly vybrat z různých alternativních možností. DSS umožňuje analyzovat data z interních informačních zdrojů, poskytuje přístup k informacím mimo organizaci a analyzuje informace způsobem, který bude užitečný pro konkrétní rozhodnutí a poskytne tuto podporu interaktivně.

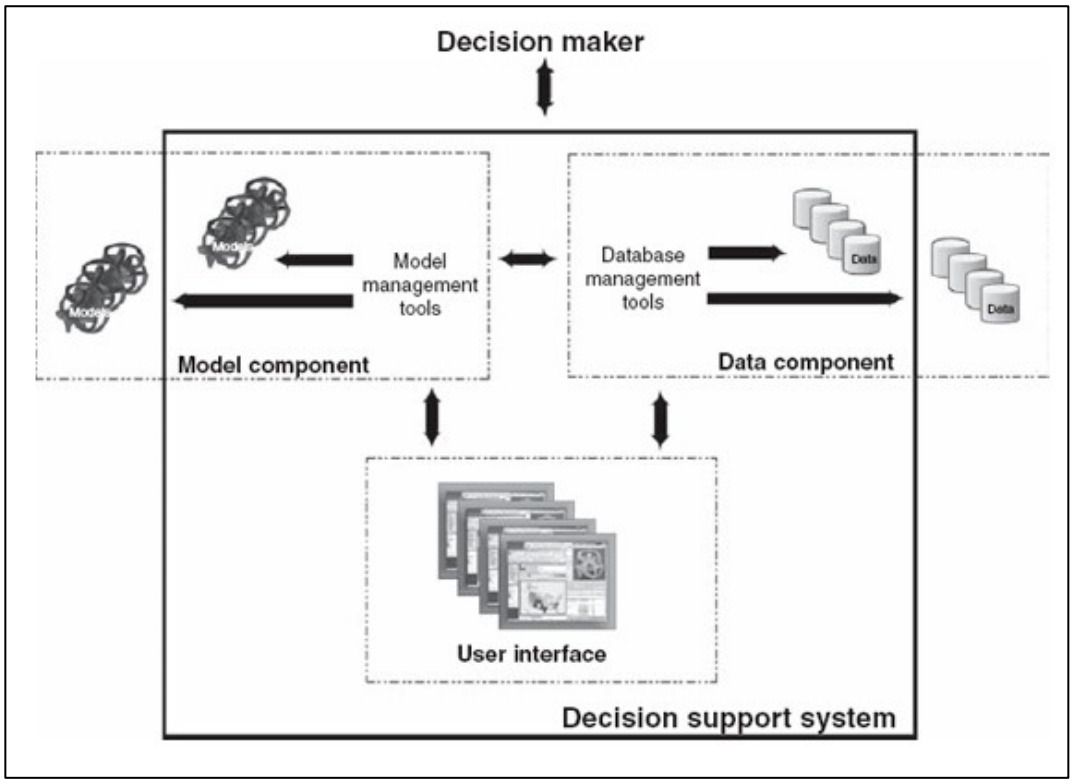
Dostupnost DSS v podniku tedy dává příležitost zlepšit procesy sběru a analýzy dat spojené s rozhodováním. Pokud jde o rozhodování, dostupnost DSS poskytuje příležitost ke zlepšení kvality a citlivosti rozhodování, a tudíž i příležitost ke zlepšení výsledků společnosti. DSS nabízí uživatelům schopnost zkoumat podnikové prostředí efektivně a včas.

2.1.1 Komponenty systému pro podporu rozhodování

Aby bylo dosaženo kvalitní podpory pro rozhodování v podniku, měl by DSS obsahovat tři základní komponenty (viz obrázek 2.1).

První komponentou je systém správy databází (dále pouze DBMS – z angličtiny Database Management System). DBMS poskytuje přístup k datům i ke všem řídicím programům nezbytným pro získání těchto dat ve formě vhodné pro uvažovanou analýzu, bez nutnosti znalosti programování uživatelem. Data zahrnují fakta o interních operacích, trendech, průzkumech trhu a obecně o dostupných podkladech pro transformaci na vhodné informace. Systém DBMS by měl být dostatečně propracovaný tak, aby umožňoval svým uživatelům přístup k datům, i když nevědí, kde jsou data fyzicky umístěna. DBMS navíc usnadňuje slučování dat z různých zdrojů.

Obrázek 2.1 - Komponenty DSS



Zdroj: Sauter, 2010

Systém správy základních modelů (dále pouze MBMS – z angličtiny Model Base Management System) má podobný úkol v DSS stejně jako DBMS, ale je vztažený pro modely. MBMS sleduje všechny možné modely, které mohou být spuštěny během analýzy, a také ovládací prvky pro spuštění modelů. To může zahrnovat syntaxi potřebnou ke spuštění úloh – formát, do kterého je potřeba data před spuštěním připravit, a formát, ve kterém budou data po spuštění úloh. MBMS také vytváří návaznosti mezi modely, tedy výstupem jednoho modelu může být vstup do jiného modelu. Dále poskytuje mechanismy pro analýzu citlivosti modelu po jeho spuštění a nabízí kontextově a modelově orientovanou pomoc, aby pomohl uživateli zpochybnit předpoklady modelů a určit, zda jsou vhodné pro dané rozhodnutí.

Uživatelské rozhraní představuje všechny prostředky, kterými jsou informace zadávány do systému a exportovány ze systému. Zahrnuje všechny vstupní obrazovky, na kterých uživatelé definují požadavky pro získání dat a modelů. Kromě toho zahrnuje všechny výstupní obrazovky, pomocí kterých uživatelé získávají výsledky nebo přehledy. Mnoho uživatelů si uživatelské rozhraní představuje jako skutečné DSS, protože to je ta část systému, kterou vidí. Jedná se však pouze o uživatelsky přívětivou formu vykreslení informací nebo modelů (Sauter, 2010).

2.1.2 Typy systémů pro podporu rozhodování

Nejčastěji se DSS člení podle pohledu na jejich hlavní účel. Obecně je tedy možné takto DSS rozdělit do pěti typů.

Textově orientované systémy katalogizují knihy, periodika, zprávy, poznámky a další písemné dokumenty, tak aby jejich obsah mohl být zpřístupněn odpovědným osobám. Každý dokument nebo část tohoto dokumentu poskytuje určité informace nebo dokonce znalosti, které by mohly být důležité při rozhodování. Systém umožňuje kategorizovat, konsolidovat a slučovat dokumenty, ale také psát komentáře k obsahu a jeho hodnotě. Inteligentní systémy mohou navíc provádět obsahovou analýzu textů a doporučit sekce (a tedy informace), které by rozhodovací činitel jinak nezohlednil.

Databázově orientované DSS jsou podobné textovým systémům v tom, že poskytují popisné informace, které jsou relevantní pro zvažovanou volbu. Místo poskytování textu se však tyto systémy zaměřují na data, která jsou uložena v databázi. Systém ovládající tyto databáze umožňuje manipulovat s daty, propojovat je mezi sebou a prezentovat je tak, aby z toho měli prospěch uživatelé s rozhodovací pravomocí. Obecně takové systémy používají k identifikaci a manipulaci s daty strukturovaný dotazovací jazyk SQL.

Tabulkově orientovaný DSS používá již dostupné tabulkové nástroje k sumarizaci a analýze dat. Namísto pouhého poskytnutí přístupu k datům umožňují tyto DSS uživatelům vytvořit některé základní modely, které rychle a efektivně vyhodnotí.

DSS orientované na pravidla jsou rozhodovací systémy, které uživatelům pomáhají se rozhodovat jako expert. Myšlenkou rozhodovacích nástrojů založených na pravidlech je zpřístupnění znalostí, dostupných z literatury a dovedností, které existují pouze v hlavách několika expertů, uživatelům. Proto se jim také říká expertní nebo znalostní systémy. Tyto znalosti se pak aplikují na informace v konkrétní situaci, aby umožnily ostatním učinit efektivnější rozhodnutí.

Složené DSS jsou hybridní kombinace jednotlivých typů DSS. Takové systémy mají smíšené schopnosti, jako je například kombinace *tabulka – databáze*. Komponenty různých typů DSS existují v jednom systému a jejich použití tak umožňuje úplnou flexibilitu. Jak je možné očekávat, takové hybridní návrhy jsou dnes nejběžnější formou DSS (Sauter, 2010).

2.2 Řízení vývoje webových aplikací

Pro úspěšnost vývoje webové aplikace je nutné celý vývoj dobře řídit. Pro efektivní řízení musí pověřená osoba, v praxi se nejčastěji jedná o projektového manažera, zvolit správnou metodiku řízení vývoje softwaru.

2.2.1 Metodika agilního rozvoje

Ve všech agilních metodách se software vyvíjí v iteracích, které obsahují malé přírůstky nové funkcionality. Existuje mnoho forem/metod agilního vývoje, například SCRUM, krystal nebo extrémní programování.

Výhody – umožňuje vydávání softwaru v iteracích. Iterativní vydání zvyšují efektivitu tím, že vývojovému týmu umožňují včas najít a opravit chyby, případně vyrovnat očekávání zákazníka nebo doplnit funkcionality.

Nevýhody – agilní metody se spoléhají na komunikaci v reálném čase, tudíž novým uživatelům často chybí dokumentace. Jedná se o poměrně časově náročný závazek s velkým pracovním nasazením.

2.2.2 Metoda nasazení DevOps

Nejedná se pouze o metodologii vývoje, ale také o soubor postupů, které podporují organizační strukturu. Nasazení systému DevOps se zaměřuje na organizační změny, které zlepšují spolupráci mezi osobami odpovědnými za různé segmenty životního cyklu vývoje.

Výhody – zaměřeno na zrychlení doby uvedení softwaru na trh, snížení neúspěchu nových verzí a zkrácení doby mezi opravami.

Nevýhody – pro některé typy zákazníků nevhodné, například kvůli striktním požadavkům na vývoj.

2.2.3 Metodika Vodopád

Považováno za nejtradičnější metodu vývoje softwaru. Metoda vodopád je rigidní lineární model, který se skládá ze sekvenčních fází (požadavky, návrh, implementace, ověření, údržba) se zaměřením na odlišné cíle. Každá fáze musí být kompletní, než začne další (Martinů a Čermák, 2018).

Výhody – jednoduchá orientace a správa vývoje softwaru. Tuto metodu je možné využít pro projekty se stabilními cíli a požadavky.

Nevýhody – často pomalá a nákladná kvůli pevné struktuře a přísným kontrolám.

2.3 Architektury webových aplikací

Vývoj webových aplikací těží ze vzniku mnoha nových technologií. Víceúrovňové objektově orientované platformy, jako je například Java, se staly samozřejmostí. Tyto nové nástroje a technologie jsou schopné vytvářet výkonné aplikace, ale nelze je snadno implementovat. Z důvodu špatně zvolené struktury ze strany vývojářů často dochází k selhání při vzniku webových aplikací. Na základě zkušeností z mnoha úspěšně nasazených aplikací vzniklo několik běžně zaváděných architektur. I zde je však potřeba zvolit tu nejvhodnější pro daný typ aplikace.

Tato kapitola představuje několik architektur vhodných pro implementaci webových aplikací. Architektura mikroslužeb, která je zde také zmíněna, byla následně použita i v praktické části pro implementaci webové aplikace. Důvodem především bylo rozdělit aplikaci do dvou funkčních celků, které jsou schopny pracovat odděleně.

2.3.1 Třívrstvá architektura

Třívrstvá architektura je nejběžnějším vzorem používaným k rozdělení software systému. Základním principem je koncepční řazení jednotlivých částí systému k dané vrstvě, kdy jednu vrstvu je možné chápat jako souvislý celek systému. Jeden systém může obsahovat více komponent z každé z těchto tří vrstev.

Třívrstvá architektura snižuje složitost závislosti v systému a tím je současně dosaženo nižší vazby s vyšší soudržností jednotlivých vrstev. To má výhody jak pro pochopení, tak pro údržbu systému. Kromě toho lze jednotlivé vrstvy snadno vyměňovat, aniž by bylo potřeba měnit celý systém.

Prezentační vrstva

Primární odpovědností prezentační vrstvy je zobrazit informace uživateli a interpretovat jeho příkazy do systému. Logika prezentační vrstvy je především o tom, jak zvládnout interakci mezi uživatelem a softwarem. Může se jednat o jednoduchou formu příkazového řádku nebo grafické uživatelské rozhraní webového prohlížeče, případně desktopové aplikace.

Aplikační vrstva

Aplikační vrstva obsahuje obchodní logiku systému. Zahrnuje logické operace založené na zdrojových datech systému a zajišťuje data pro správné zobrazení na prezentační vrstvě v závislosti na příkazech uživatelů. Logika systému je zde sjednocena.

Datová vrstva

Jak už název napovídá, datová vrstva slouží především k práci s daty. Největší část datové vrstvy tvoří ve většině systémech databázový systém, který je primárně odpovědný za ukládání trvalých dat. Prostřednictvím datové vrstvy může však probíhat i komunikace s jinými systémy. Jako příklad je možné uvést monitorovací systémy transakcí, jiné podnikové systémy nebo systémy zasílání zpráv (Fowler, 2003).

2.3.2 Monolit

Pokud je nutné všechny funkce v aplikaci nasadit společně, aby aplikace fungovala, je možné takový systém považovat za monolit. Jinak řečeno, jedná se o architekturu webových aplikací, kde je aplikace zcela soběstačná, co se týče jejího chování. Celá aplikace je obvykle nasazena jako jeden celek. Pokud je potřeba takovou aplikaci škálovat vodorovně, obvykle musí být celá aplikace duplikována na více serverech. Mezi typické znaky této architektury patří samostatný programový kód a aplikace se typicky nasazuje a testuje jako jeden celek. Moduly, z nichž se aplikace skládá, jsou těsně svázány.

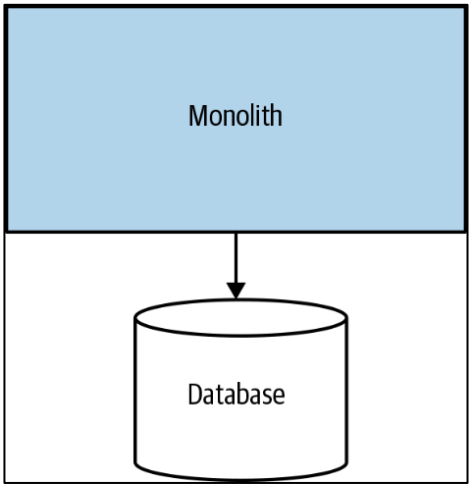
Nejběžnějším příkladem je systém, ve kterém je celý kód nasazen jako jediný proces. Z důvodu robustnosti nebo změny měřítka je možné mít více instancí tohoto procesu, ale v zásadě je celý kód zabalen do jednoho procesu. V praxi mohou být tyto jednoprocesní systémy samy o sobě jednoduchými distribuovanými systémy, protože téměř vždy končí prezentaci informací webovým prohlížečem a načítáním nebo ukládáním dat z/do databáze.

Podmnožinou monolitu s jedním procesem je modulární monolit, ve kterém se jeden proces skládá ze samostatných modulů. Na každém je možné pracovat samostatně, ale pro úspěšné nasazení aplikace je stále nutné moduly zabalit do jednoho celku. Pro mnoho organizací může být modulární monolit vhodnou volbou. Pokud jsou hranice modulu dobře definované, může to umožnit vysoký stupeň paralelní práce.

Aplikace navržené jako monolit mají samozřejmě své výhody i nevýhody. Monolitické aplikace jsou mnohem jednodušší pro nasazení. To může vyústit v mnohem snadnější pracovní postupy pro vývojáře a pro monitorování samotné aplikace. V monolitní aplikaci je snadné opětovné použití kódu. Pokud chceme znovu použít kód v distribuovaném systému, je nutné se rozhodnout, zda kopírovat kód, rozbít knihovny, nebo vytvořit sdílenou funkci mezi službami.

Nevýhody monolitu se projeví především u velkých systémů. Při vydávání nové verze je totiž nutné kompilovat celou aplikaci, což může být komplikované a nákladné. Každá chyba zároveň může teoreticky ovlivnit nebo zničit celou aplikaci (Newman, 2015).

Obrázek 2.2 - Architektura monolitické aplikace



Zdroj: Newman, 2015

2.3.3 Mikroslužba

Architektura založená na mikroslužbách popisuje návrh složitého systému pomocí malých, jednoduchých a nezávisle běžících služeb. Služba zapouzdřuje funkčnost a zpřístupňuje ji dalším službám, nejčastěji pomocí programových rozhraní, jako jsou metoda REST nebo síťový protokol http. Běžně používaným datovým formátem pro přenos dat je JSON.

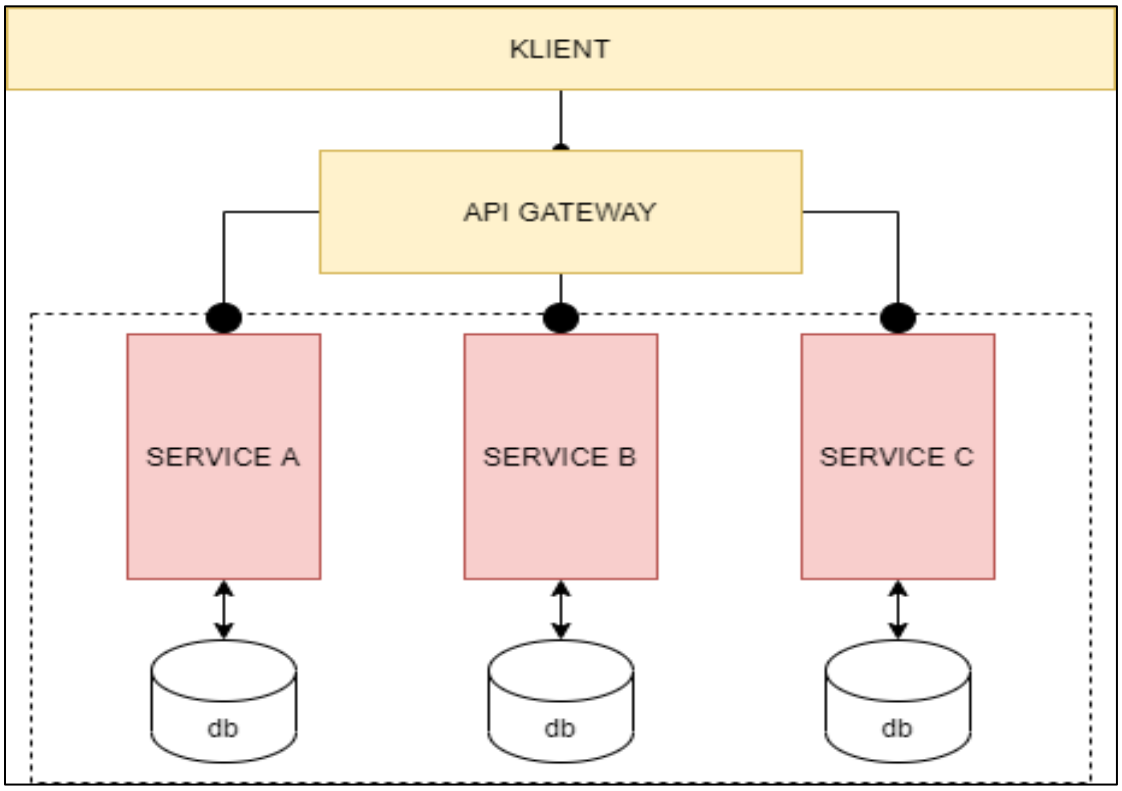
Z vnějšku je jedna mikroslužba považována za černou skříňku. Spotřebitelé, ať už jsou to jiné mikroslužby, nebo externí programy, přistupují k této funkci prostřednictvím síťových koncových bodů. Detaily vnitřní implementace (například technologie, do které je služba zapsána, nebo způsob ukládání dat) jsou před vnějším světem zcela skryty. To znamená, že architektury mikroslužeb se ve většině případů vyhýbají používání sdílených databází. Místo toho každá mikroslužba zapouzdřuje svou vlastní databázi, kde je to požadováno.

Při tvorbě mikroslužby je nutné zvolit správný rozsah funkcionalit, které bude daná mikroslužba poskytovat v rámci celkové architektury systému. Případ příliš rozsáhlého záběru funkcionalit poskytovaných mikroslužeb není vhodný především z důvodu zvyšující se komplexnosti systému, porušování zásady volné vázanosti, nedodržovaní konzistence dat spravovaných danou mikroslužbou a obtížného testování mikroslužby.

Architektura mikroslužeb může poskytnout obrovskou míru flexibility při výběru technologie, organizování týmů a dalších. Tato flexibilita je částečně důvodem, proč mnoho organizací využívá tuto architekturu. Mikroslužby však s sebou přinášejí značnou míru složitosti a je třeba zajistit, aby byla tato složitost oprávněná. Použití musí být ospravedlněno problémy, které je potřeba vyřešit, protože jednodušší přístupy mohou často zajistit stejně kvalitní výsledky.

Mnoho organizací, zejména větších, v praxi nicméně ukázalo, jak efektivní mohou být mikroslužby. Když jsou základní koncepty mikroslužeb správně pochopeny a implementovány, je možné pomocí mikroslužeb vytvořit velké a zároveň kvalitní systémy (Nadareishvilili et al, 2016).

Obrázek 2.3 - Mikroservisní architektura



Zdroj: vlastní

2.4 Databáze

Databázi je možné definovat jako strukturovaný soubor dat ukládaný na paměťovém médiu. Databáze jsou logicky strukturovaná data, a to včetně logiky jejich vzájemných vztahů a vazeb. V současnosti existuje velké množství různých přístupů, v jaké formě data ukládat. V této kapitole budou popsány ty nejpoužívanější (Harrison, 2015).

2.4.1 Relační databáze

Nejčastějším typem databáze využívané v podnicích pro ukládání firemních dat je relační databáze. Tento typ databáze používá strukturu, která umožňuje identifikovat a přistupovat k datům ve vztahu k jinému datu v databázi. Data v relační databázi jsou často organizována do tabulek. Tabulky mohou obsahovat mnohdy až miliony záznamů. Jednotlivé záznamy jsou strukturovány do sloupců, které jsou označeny popisným názvem (např. *jméno*) a mají specifický datový typ (např. *varchar*).

Pro správu relačních databází jsou využívány systémy správy relačních databází (česká zkratka SŘBD). Tyto systémy nám umožňují data vytvářet, aktualizovat a spravovat. Nejčastějším způsobem manipulace s daty je pomocí jazyka SQL. Mezi populární systémy patří MySQL, MSSQL, PostgreSQL nebo Oracle (Harrison, 2015).

2.4.2 Moderní relační databáze

V současné době si vývoj (především webových aplikací) žádá hned několik dalších technologických požadavků, které pomocí tradičních relačních databází nejsme schopni implementovat. Mezi hlavní požadavky na moderní databáze patří:

- decentralizace úložiště dat (Cloud, distribuované databáze),
- práce s velkými objemy dat, velké množství operací (Big Data),
- ukládání specifických/problémových datových typů (objekty, nestrukturované dokumenty, RDF grafy atp.),
- jednoduchá reakce na časté změny schématu dat (v některých případech žádné schéma),
- škálovatelnost (dostupnost, nerovnoměrné rozložení zátěže, neznámé dotazy).

Tyto požadavky mají za následek vývoj moderních relačních databází, které přestávají odpovídat tradičnímu relačnímu konceptu. Jako příklad můžeme uvést úmyslné zanedbání ACID (Atomicity, Consistency, Isolation, Durability) pro zisk větší dostupnosti dat. Důsledkem je také vznik speciálních nerelačních databází, například objektově-relační, objektové nebo XML databáze (Harrison, 2015).

2.4.3 PostgreSQL

PostgreSQL je relační databázový systém s otevřeným zdrojovým kódem. Mezi jeho hlavní vlastnosti patří stabilita, škálovatelnost a bezpečnost. PostgreSQL 13 je nejnovějším vydáním této databáze.

Jednou z největších výhod PostgreSQL je možnost ukládání obrovského množství dat. Jedna instance databázového systému PostgreSQL může obsahovat více než čtyři miliardy jednotlivých databází, každá s neomezenou celkovou velikostí a kapacitou pro více než jednu miliardu tabulek, z nichž každá obsahuje 32 TB dat. Jedna tabulka navíc může mít 1600 sloupců s neomezeným počtem indexů s více sloupci (až 32 sloupců).

Tento databázový systém je plně kompatibilní s ACID a má velmi silný základ v integritě a souběžnosti dat. Dodává se s procedurálním jazykem PL/pgSQL, který lze použít k zápisu opakovaně použitelných částí kódu, jako jsou funkce, rutiny, triggery, pohledy, materializované pohledy nebo dělené tabulky. PostgreSQL lze rozšířit o další vestavěné jazyky jako Perl, Python, Java, a dokonce i Bash.

PostgreSQL běží na téměř každém operačním systému, včetně systémů Linux, Unix, Mac OS X a Microsoft Windows. Instance PostgreSQL se nazývá klastr. Jeden klastr může obsluhovat a zpracovávat více databází. Každá databáze je izolovaným prostorem, kde mohou uživatelé a aplikace ukládat data. Databázi lze uspořádat do jmenných prostorů, které se nazývají schémata. Schéma je mnemotechnický název, který může uživatel přiřadit k uspořádání databázových objektů, například tabulek, do strukturovanější kolekce. Schémata nelze vnořit, takže představují plochý jmenný prostor.

Databázové objekty jsou reprezentovány vším, co může uživatel v databázi vytvořit a spravovat, například tabulkami, funkcemi, triggery nebo datovými typy. Každý vytvořený objekt může patřit pouze do jednoho schématu.

Uživatelé jsou definováni na úrovni celého klastru, což znamená, že nejsou vázáni na konkrétní databázi v klastru. Uživatel se může připojit a spravovat libovolnou databázi v klastru, ke které má povolení (Ferrari a Pirozzi, 2020).

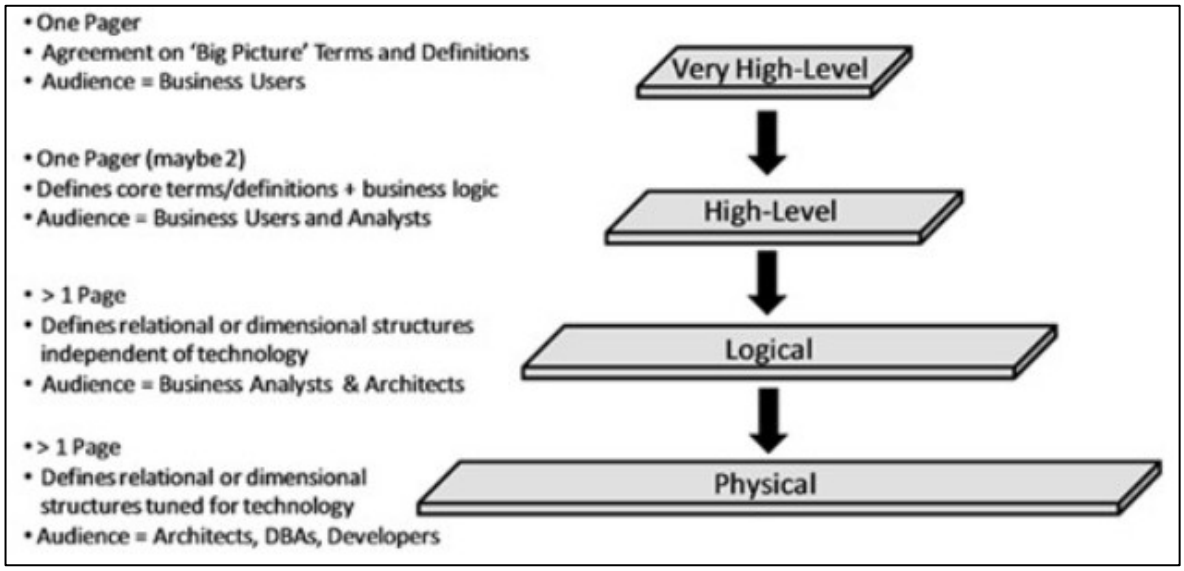
Především díky svým výhodám při práci s velkým objemem dat je PostgreSQL velmi často využíván pro analytické systémy, a proto byl tento databázový systém implementován i v rámci webové aplikace popisované v praktické části této práce.

2.5 Datové modelování

Datové modelování je proces používaný k definování a analýze datových požadavků, které jsou potřebné k podpoře obchodních procesů v rozsahu odpovídajícího systému v podnicích. Výsledkem tohoto procesu je datový model.

Datový model lze definovat jako vizuální reprezentaci objektů reálného světa (lidí, míst a věcí) ovlivňujících podnikové prostředí. Cílem datového modelu je převzít obchodní koncepty a složitá pravidla potřebná k vytvoření databáze a zjednodušit je do intuitivního obrazu. Používá se také k usnadnění komunikace mezi manažery a technickými pracovníky. Datové modely se vytvářejí na několika různých úrovních podrobností, přes základní architektonické uspořádání až po podrobné návrhy datové infrastruktury (Hoberman, Durbank a Bradley, 2009).

Obrázek 2.4 - Úrovně datových modelů



Zdroj: Hoberman, Durbank a Bradley, 2009

2.5.1 Sémantický datový model

Účelem sémantického datového modelu (dále pouze SDM) je poskytnout organizovanou strukturu, ze které lze navrhnout konceptuální datový model a další úrovně modelu. Objekty definované v rámci tohoto modelu představují základní oblasti, subjekty a relace daného podniku. Může jít například o oblasti podnikání, stakeholdery nebo zákaznický segment. Při modelování malých aplikací a jejich datových struktur může být tento model volitelný. Pro velké aplikace je však tento model zásadní.

Hlavním cílem je vytvoření organizované struktury, do které lze mapovat podrobnější datové koncepty na nižších úrovních modelu. SDM je často znázorňován graficky, kde jsou oblasti zobrazovány pomocí rámečků, kruhů nebo jako prostý text. Relace mezi objekty jsou reprezentovány plnou čárou (Hoberman, Durbank a Bradley, 2009).

2.5.2 Konceptuální datový model

Konceptuální datový model (dále pouze KDM) popisuje základní datové koncepty, pravidla a obchodní definice. Výhodou návrhu takového modelu je dosažení společné terminologie a zásad při vývoji datové struktury podniku či aplikace. KDM by měl být vždy dostatečně jednoduchý a jasný, aby mu porozuměl i netechnický pracovník, například vedoucí manažer.

Při vytváření datového modelu na konceptuální úrovni je důležité zvolit správný formát modelování. Jedním z nejběžnějších formátů pro datové modely je ER modelování (Entity Relationship modeling). Tento styl modelování existuje již od 70. let minulého století. ER modelování bylo vytvořeno za účelem popisu logiky dat a vztahů mezi nimi. Relační modely se nejčastěji používají k popisu provozních datových struktur (Hoberman, Durbank a Bradley, 2009).

2.5.3 Logický datový model

Logický datový model (dále pouze LDM) představuje grafické znázornění objektů reálného světa (osob, míst, věcí), základní podniková pravidla, vztahy mezi nimi a všechny jejich charakteristiky, které jsou potřebné pro fungování aplikace (podniku). LDM oproti konceptuální úrovni zobrazuje více podrobností. Datový model je však nezávislý na jakékoliv fyzické implementaci.

2.5.4 Implementační úroveň (Fyzický datový model)

Fyzický datový model (dále pouze FDM) popisuje architekturu skutečné databáze nebo datové struktury rozšířené o některé specifické informace pro konkrétní databázovou platformu. Datový model je nejčastěji prezentován v grafické formě.

Fyzický datový model může obsahovat některé komponenty logického datového modelu, které byly upraveny za účelem optimalizace výkonu databáze a aplikace. Logické entity se mohou stát fyzickými tabulkami a podniková pravidla lze definovat jako databázové omezení. Ve fyzickém datovém modelu mohou být také zastoupeny další databázové techniky pro zlepšení výkonu, jako například indexování nebo partitioning. Datový model je obvykle navržen a optimalizován pro databázový systém, ve kterém bude implementován, například PostgreSQL (Hoberman, Durbank a Bradley, 2009).

2.6 Business Intelligence

Business Intelligence (dále pouze BI) lze popsat jako soubor technik a nástrojů pro akvizici a transformaci prvotních dat na smysluplné a užitečné informace pro obchodní analýzu. Použité technologie v rámci BI jsou schopny zvládnout velké množství dat (strukturovaných i nestrukturovaných), aby pomohly identifikovat a jinak rozvíjet obchodní příležitosti. Hlavním cílem BI je umožnit snadnou interpretaci těchto velkých objemů dat. Identifikace nových příležitostí založených na získaných poznatcích může danému podniku poskytnout dostatečnou konkurenční výhodu či dlouhodobou stabilitu na trhu.

Technologie BI poskytují historický, aktuální a prediktivní pohled na obchodní operace. Běžnými funkcemi technologií BI jsou reporting, online analytické zpracování, analytika, dolování dat, dolování procesů, komplexní zpracování událostí, řízení podnikového výkonu, benchmarking, dolování textu, prediktivní a preskriptivní analýza.

BI je možné použít k podpoře široké škály obchodních rozhodnutí od provozních až po strategická. Ať už se jedná o jakékoliv obchodní rozhodnutí, ve všech případech je BI nejúčinnější, když kombinuje data odvozená z trhu, na kterém podnik působí (externí data), s daty z interních zdrojů společnosti, jako jsou finanční a provozní data (interní data). V kombinaci mohou externí a interní data poskytnout ucelenější obraz, jenž ve skutečnosti vytváří tzv. *inteligenci*, kterou nelze odvodit žádným singulárním souborem dat. V nejčastějších případech užití umožňují nástroje BI organizacím získat přehled o nových trzích, posoudit poptávku a vhodnost produktů nebo služeb pro různé segmenty trhu a měřit dopady marketingového úsilí.

Obecně lze BI definovat jako soubor metodik, procesů, architektur a technologií, které transformují nezpracovaná data na smysluplné a užitečné informace používané k umožnění efektivnějších strategických, taktických a provozních poznatků a rozhodování. Podle této definice zahrnuje BI také technologie jako integrace dat, kvalita dat, datové sklady, správa kmenových dat, analýza textu a mnoho dalších, které jsou někdy shlukovány do segmentu „Správa informací“.

K zvýšení obchodní hodnoty podniku lze BI použít pro mnoho obchodních účelů. Níže jsou popsány ty nejčastější:

- **měření** – soubor procesů, jež vytváří hierarchii výkonnostních metrik a srovnávacích testů, které informují vedení podniku o pokroku směrem k obchodním cílům (řízení podnikových procesů),
- **podniková analýza** – programy či metody, které vytváří kvantitativní procesy pro podnikání, aby dospělo k optimálním rozhodnutím a provádělo zjišťování obchodních znalostí. Podniková analýza často zahrnuje těžbu dat, těžbu procesů, statistickou analýzu, prediktivní analýzu, prediktivní modelování, modelování obchodních procesů, datovou linii, komplexní zpracování událostí nebo preskriptivní analýzu,
- **podávání zpravodajství** – soubor nástrojů a metod, které budují infrastrukturu pro strategické zprávy, jež slouží strategickému řízení podniku. Často zahrnují vizualizaci dat či výkonný informační systém,

- **znalostní management** – program umožňující podniku řídit se pomocí dat prostřednictvím strategií a postupů k identifikaci, vytváření, zastupování, distribuci a umožnění přijetí poznatků a zkušeností, které jsou skutečnými obchodními znalostmi. Řízení znalostí vede k řízení učení a dodržování předpisů.

Kromě výše uvedeného může BI poskytnout proaktivní přístup, například výstražnou funkci, která okamžitě upozorní koncového uživatele, pokud jsou splněny určité podmínky. Například pokud některá obchodní metrika překročí předem definovanou prahovou hodnotu, bude tato metrika zvýrazněna ve standardních sestavách. Obchodní analytik může být upozorněn prostřednictvím e-mailu nebo jiné monitorovací služby. Tento komplexní proces vyžaduje správu dat, kterou by měl zpracovat daný odborník (Bentley, 2017).

2.6.1 Datové sklady

Datový sklad je systém správy databáze, který existuje odděleně od operačních systémů a databází. Stejně jako provozní data jsou datové sklady integrované, předmětově a časově orientované systémy. Rozdílem je však to, že jsou energeticky nezávislé, a proto dokážou konzistentně podporovat různé analýzy. Obecně lze říct, že jsou tyto databáze archivy provozních dat, která byla vybrána na podporu rozhodování a optimalizovány pro interakci s analytickými systémy podniku. Ve většině podniků se jedná o relační databáze, které mohou podporovat širokou škálu dotazů v široké škále formátů. Mohou být složeny ze stovek tabulek optimalizovaných pro typické dotazy.

Vývoj datového skladu je obtížný a časově náročný proces, který většinu podniků stojí značné náklady ve všech ohledech. Zatímco procesy přesunu a optimalizace dat nejsou nijak zvlášť obtížné, procesy identifikace relevantních dat, jejich míchání a zajištění jejich správného čištění jsou velmi obtížné. Jinými slovy, rozhodnutí o tom, která data jsou relevantní pro konkrétní rozhodnutí, jak by měla být data reprezentována a kombinována a jak zajistit, aby byla smysluplná, konzistentní a přesná, jsou obtížnými kroky při budování datového skladu.

Pro efektivní vybudování datového skladu je nutné porozumět potřebám podnikání a pečlivě plánovat. Datový sklad musí být také vnímán a považován za obchodní aktivum a musí být poháněn obchodními potřebami podniku.

Cílem datového skladu je spojit data z různých zdrojů a sloučit je tak, aby byla užitečná pro adekvátní uživatele. Návrh datového skladu tedy znamená přinášení data z interních a externích zdrojů. Firemní databáze neboli interní data poskytují základ rozhodovacího skladu. Tyto systémy obecně poskytují údaje o široké škále transakcí prováděných v běžném obchodním provozu podniku. Interní databáze zaznamenávají informace týkající se prodejů, nákupů, nákladů, personálu, plánů, předpovědí a dalších aspektů organizace. Ačkoli jsou tyto oficiální záznamy společností důležité, nejsou dostatečné k podpoře většiny dnešních rozhodování.

Aby bylo dosaženo co nejlepších výsledků při rozhodování pomocí datové analýzy, je potřeba mít k dispozici informace o jednom nebo více faktorech mimo společnost. Taková externí data mohou být užitečná k získání informací o preferencích zákazníků nebo poptávce po produktech či službách konkurence v konkrétních regionech prodeje. Do datového skladu by měla být také načtena relevantní veřejná data. Jako příklad lze uvést sčítání lidu České republiky, které poskytuje velké množství údajů o změnách populace, počet narozených dětí, úmrtí nebo migrací ze zemí Evropy. Externí data mohou být zakoupena od třetích stran nebo je lze získat z internetu (Sauter, 2010).

Architektura datového skladu

Pro účely dosažení co největší flexibility při používání dat z datových skladů, je využívána technologie on-line analytického zpracování dat (OLAP). Tato architektura poskytuje vylepšený výkon zpracování analytických dotazů. Data jsou ukládána a organizována odděleně od aplikací a odděleně od systémů zpracování transakcí. Jádrem této architektury je tzv. datová kostka. Datová kostka je název pro způsob organizace dat, který rozšiřuje dvojrozměrně tabulkové uspořádání tak, že každá datová dimenze je uložena v jedné ose kostky. Tím překonává některá omezení relačních databází. Takové zpracování dat je obecně označováno jako vícerozměrné online analytické zpracování (MOLAP).

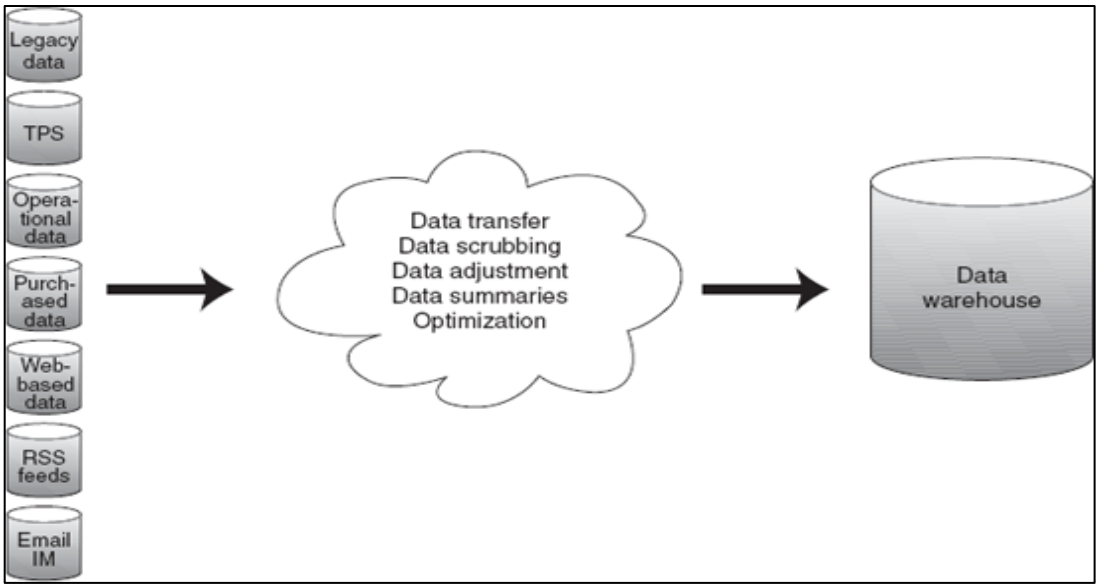
Tyto vícerozměrné produkty OLAP nebo MOLAP obvykle běží rychleji než jiné přístupy, a to především proto, že je možné indexovat přímo do struktury datové krychle a sbírat podmnožiny dat. Řešení MOLAP mohou být problematická pro velmi velké datové sady s mnoha rozměry. Zvyšováním počtu dimenzí se kostka se stává řidší, což má tendenci zvyšovat požadavky na úložiště (někdy na nepřijatelné úrovně). Techniky komprese mohou pomoci, ale jejich použití má tendenci ničit přirozené indexování MOLAP.

Alternativou k architektuře MOLAP je použití relační struktury OLAP nebo ROLAP. Data se shromažďují jako relační tabulky a organizují se jako hvězdné nebo sněhové vločky. Jádrem architektury je tabulka faktů, která je prostřednictvím indexů propojena se specifickými relacemi, jež obsahují konkrétní data. Taková struktura je schopna zpracovat větší objemy dat a podporovat lepší možnosti rozbalení. Technologie ROLAP však nejsou tak rychlé (Sauter, 2010).

Tvorba datového skladu

Hlavním krokem při budování datového skladu je načtení dat z různých datových úložišť. Tím však proces zjevně nekončí. Taktéž důležitým krokem je očištění dat. Data mohou pocházet z nejrůznějších zdrojů a tyto roztříštěné pohledy na prostředí organizace je třeba spojit do jednotného rámce (viz obrázek 2.5). V závislosti na tom, jak jsou data stará a jak pečlivě byla spravována, je nutné určit stupeň čištění dat. Hlavním cílem tohoto procesu je získat konzistentní a organizovaná data.

Obrázek 2.5 - Proces budování datového skladu



Zdroj: Sauter, 2010

Dalším důležitým krokem je správně definovat a standardizovat ukazatelové jednotky pro celý datový sklad. Jako příklad je možné uvést měnu. Měna je jedním z faktorů, které musí být v datovém skladu konzistentní. Většina organizací má nějakou globální součást, jako je dodavatel, zákazník, nebo dokonce část organizace. Tito nadnárodní partneři používají různé měny k vyjádření svých nákladů, výnosů a prodejů. Relativní hodnoty těchto různých měn se v čase mění. Je tedy zásadní, aby při hodnocení prodejců, zákazníků apod. bylo provedené srovnání konzistentní.

Protože data pocházejí z různých systémů, je možné, že budou aktualizována v různých časech. Do datového skladu je důležité přidat identifikační pole, která určují, kdy se data aktualizují, aby uživatelé (analytici) mohli přesně identifikovat časové horizonty bez ohledu na to, kdy byla data zadána.

Cílem všech těchto úprav je poskytnout nejlepší obrázek o podniku, jeho zákaznících, dodavatelích, konkurentech a co nejvíce dalších vnějších vlivů, aby analýzy byly co nejspolehlivější (Sauter, 2010).

2.6.2 Proces ETL

Extract-Transformation-Load (dále pouze ETL) představuje proces pohybu a transformace dat z jednoho nebo více zdrojů do datového skladu nebo datového tržiště. Je také široce používán při integraci nebo migraci dat. Proces je složený ze tří fází – extrakce, transformace a nahrání dat.

První podproces zahrnuje extrakci dat ze zdrojových systémů. Většina takových projektů kombinuje data z různých zdrojových systémů. Každý samotný systém může proto používat jinou organizaci dat nebo odlišný formát. Obecně je tedy cílem fáze extrakce převést data do vhodného a jediného formátu pro transformační zpracování.

Ve fázi transformace dat se na extrahovaná data aplikuje řada pravidel nebo funkcí potřebných pro správné načtení. Jednou z nejdůležitějších funkcí transformace je čištění dat, jehož cílem je předat pouze správná data. Dále jsou v rámci této fáze například sjednoceny číselníky nebo odvozeny nové vypočítané hodnoty.

V poslední fázi jsou data načtena do cílového datového systému. S ohledem na požadavky se tento proces velmi liší v použitých technologiích a pravidelnosti načítání (Bentley, 2017).

2.6.3 Migrace dat

Migrace dat je jednorázový přesun dat ze starých nebo již používaných systémů do nového datového úložiště. Jedná se o jednosměrnou cestu bez aktualizace zdrojových dat. Migrace dat zahrnuje výběr, přípravu, extrakci, transformaci a trvalý pohyb vhodných dat, která mají vhodnou kvalitu. Migrace dat může probíhat mezi různorodými datovými úložišti, formáty či IT systémy.

Proces migrace je obecně brán jako kritická fáze implementace nového systému do podniku. Některé statistiky uvádí, že téměř 40 % projektů, kde bylo potřeba migrovat data, bylo zpožděno, bylo nákladnější nebo zcela selhalo. Rizika migrace dat spočívají v nesprávně nastaveném návrhu toho, co a jak se má přenést, ve ztracených datech, v nevyčištěných datech či v chybně přenesených datech. V dobře provedené migraci dat jsou data přenesena kvalitně, nebo dokonce vyčištěná a bez ztrát. A to vše proběhne s minimálními ručními zásahy. V rámci procesu migrace je tedy potřebné dobře plánovat a využít osvědčených metodik (Morris, 2012).

2.7 Použité programovací jazyky a frameworky

Pokud jde o vývoj webového aplikace, existuje řada jazyků, technologií a frameworků, které je možné využít k dosažení všech cílů a požadavků. Některé z nich jsou vhodné pro webové stránky a jiné zase pro podnikové systémy. Správný výběr je tedy důležitým krokem při návrhu celého řešení.

Vytvoření komplexní webové aplikace zahrnuje technologie jak na straně serveru, tak na straně klienta. Klient neboli tzv. front-end představuje grafické rozhraní k vytváření a zobrazování všeho, s čím koncový uživatel interaguje. Server neboli takzvaný back-end naopak obsahuje obchodní logiku včetně validací, operací a kalkulací.

Tato kapitola zahrnuje teoretická východiska programovacích jazyků a frameworků použitých při implementaci webové aplikace. Jsou zde popsány základní definice, principy a výhody.

2.7.1 Programovací jazyk Java

Programovací jazyk Java, vyvinutý společnosti Sun Microsystems, byl navržen jako nezávislý programovací jazyk, který je dostatečně bezpečný a výkonný, aby nahradil nativní spustitelný kód. Je nutné podotknout, že se Java stala přední platformou pro webové aplikace a webové služby. Díky přenositelnosti a rychlosti je Java oblíbenou volbou pro vývoj obchodních aplikací. Patří mezi objektově orientované programovací jazyky.

Java je kompilovaný a interpretovaný programovací jazyk. Zdrojový kód se promění v jednoduché binární instrukce, podobné strojovému kódu mikroprocesorů. Avšak zatímco zdrojový kód jazyka C nebo C++ je redukován na nativní instrukce pro konkrétní model procesoru, kód Java je kompilován do univerzálního formátu, který slouží jako instrukce pro tzv. Java Virtual Machine (dále pouze VM). Jediné, co cílový systém potřebuje jakmile je kód zkompilován, je mít nainstalované prostředí Java Runtime Environment (JRE).

JRE provádí všechny běžné činnosti jako hardwarový procesor, jen to dělá v bezpečném virtuálním prostředí. Provádí sadu instrukcí založenou na zásobníku a spravuje paměť jako operační systém. Vytváří také primitivní datové typy a načítá a vyvolává nově odkazované bloky kódu. Dohromady to znamená, že kód Java je implicitně přenosný a stejnou aplikaci lze spustit na jakékoliv platformě, která poskytuje JRE.

Jako u většiny objektově orientovaných jazyků je třída základním stavebním kamenem kódu. Třída je skupina datových položek s potřebnými funkcemi, které mohou provádět operace. Datové položky ve třídě se nazývají proměnné a funkce se nazývají metody. Když je aplikace spuštěna, může existovat mnoho jednotlivých pracovních kopií dané třídy. Tyto jednotlivé inkarnace se nazývají instance třídy nebo objekty. Dvě instance dané třídy mohou obsahovat různá data, ale vždy mají stejné metody (Loy, Niemeyer a Leuck 2020).

Výhody programovacího jazyka Java

Java se stala přední platformou pro webové aplikace a webové služby. Tyto aplikace využívají technologie Java Servlet API, webové služby Java, mnoho populárních aplikačních serverů a frameworků Java. Díky přenositelnosti a rychlosti je Java vhodnou volbou pro moderní obchodní aplikace. Servery Java jsou dnes srdcem obchodního a finančního světa. Mezi hlavní výhody programovacího jazyka Java patří:

- podpora více vláknového běhu aplikace,
- existence automatické správy paměti,
- nezávislost na platformě,
- sdílení dat a programů mezi více počítači (distribuovaný jazyk),
- efektivní alokace paměti (Loy, Niemeyer a Leuck 2020).

2.7.2 Spring Framework

Spring je otevřený modulární framework založený na programovacím jazyku Java, který je nejčastěji využíván při tvorbě webových aplikací. Jedná se o kolekci knihoven, jejichž primárním cílem je nabídnout pomoc při řešení běžných problémů s vývojem softwaru. Díky Springu vývojáři nemusí primárně řešit infrastrukturu řešení, ale mohou se zaměřit přímo na obchodní logiku.

Už od svého vzniku se Spring zaměřuje zejména na modularitu. Je to důležitá charakteristika tohoto frameworku, protože především díky ní je skvělou volbou pro různé architektonické styly a různé části aplikací. Spring je tedy možné použít podle potřeby a integrovat jej s celou řadou specifikací a knihoven třetích stran. Například pro webové aplikace Spring MVC podporuje funkci koncových bodů REST, které je možné integrovat do populárního frameworku React.js.

Pokud aplikace potřebuje podporu pro distribuovaný systém, rámec může poskytnout modul s názvem Spring Cloud, který má některé základní funkce pro distribuovaná prostředí, jako jsou registrace služeb, inteligentní směrování a na straně klienta vyvažování zátěže.

Jak už bylo nastíněno, Spring má rozsáhlou funkcionalitu a skládá se z mnoha knihoven, které podporují a zabezpečují implementaci různých funkcionalit. Rámec obsahuje více než 130 Java balíčků a 1200 Java tříd rozdělených do různých modulů. Mezi hlavní moduly patří Spring Core Container, Spring Data, Spring Security, Spring Cloud (Walls, 2015).

Spring Core Container

Modul Spring Core Container je základem celého Spring ekosystému a zahrnuje tři základní komponenty:

- **Spring Core** – je nejdůležitější částí frameworku Spring. Poskytuje funkce jako Inversion of Control (dále pouze IoC) a Dependency Injection (dále pouze DI). IoC uvolňuje pevné vazby mezi objekty a tím zvyšuje flexibilitu programu. DI představuje mechanismus, kdy je do třídy vložena instance jiné třídy. V obou případech je odpovědnost přesunuta na framework Spring, čímž programátor získává větší prostor pro to věnovat se obchodní logice programu,

- **Spring Beans** – poskytuje rozhraní BeansFactory, což je konfigurační mechanismus pro správu objektů, jako je například inicializace, vytváření nebo přístup k objektům z aktuální programové logiky,
- **Spring Context** – zodpovědný za vytváření instancí, konfiguraci a sestavování Spring Beans (Walls, 2015).

Spring Data

Modul Spring Data řeší běžné problémy, kterým programátoři čelí při práci s daty v aplikaci. Pomocí tohoto modulu je možné definovat datová úložiště aplikace jako jednoduchá rozhraní Java. Při definování metod je možné řídit způsob ukládání a načítání dat pomocí konvence pojmenování. Spring Data navíc dokážou pracovat s několika různými druhy databází, včetně relačních nebo NoSQL. Pokud jde o práci s relačními daty, programátoři mají v rámci frameworku Spring několik možností. Dvěma nejběžnějšími možnostmi jsou JDBC a ORM.

JDBC (Java Database Connectivity) je aplikační programovací rozhraní, které umožňuje přistupovat k databázi pomocí standardních tříd Java. Při použití JDBC Spring poskytuje prostředky, kterými je možné provádět operace SQL, aniž by bylo potřeba provádět zdlouhavé kódování související s ověřením připojení nebo správou transakcí. Nejběžnější logiku při používání JDBC API pro přístup k datům zahrnuje třída *Jdbc Template*.

Při vývoji aplikace pomocí frameworku Spring jsou data často ukládána a načítána do/z databáze ve formě objektů. Java objekty se však od objektů v databázi (tabulky) liší a Spring proto poskytuje podporu knihoven pro objektově relační mapování (dále pouze ORM). ORM mapuje reprezentací objektů na parametry příkazu JDBC a následně mapuje výsledky dotazu JDBC zpět na reprezentace objektů. ORM obvykle nefunguje na úrovni SQL, ale spíše se odkazuje na svůj vlastní „Object Query Language“, který za běhu překládá příkazy do SQL. Jednou z nepoužívanějších ORM knihoven je Hibernate (Soni, Ganeshan a Rv, 2017).

Spring Security

Zabezpečení webové aplikace není nic jiného než ochrana zdrojů a umožnění přístupu pouze konkrétním uživatelům. Spring Security by neměl být považován za bránu firewall, server proxy, detekci narušení, zabezpečení JVM nebo něco podobného. Spring Security je primárně zaměřen na webové aplikace založené na Spring. Spring Security zpracovává ověřování a autorizaci na úrovni webových požadavků i na úrovni vyvolání metod. Je vysoce přizpůsobitelný a výkonný pro ověřování a řízení přístupů (Walls, 2015).

Spring Cloud

Svět vývoje webových aplikací vstupuje do nové éry, kde již nejsou aplikace vyvíjeny jako monolitické jednotky, ale jsou složeny z několika jednotlivých jednotek známých jako mikroslužby. Mikroslužby řeší několik praktických problémů týkajících se vývoje a běhu aplikace. Těmto problémům čelí Spring Cloud, modul pro vývoj cloudových nativních aplikací pomocí Spring frameworku.

Spring Cloud je modul implementující sadu běžných vzorů požadovaných distribuovanými systémy ve formě knihoven Java Spring. I přes svůj název není Spring Cloud sám o sobě cloudovým řešením. Spíše poskytuje řadu funkcí, které jsou zásadní při vývoji aplikací zaměřených na cloudová nasazení. Modul skrývá složitosti a poskytuje jednoduché deklarativní konfigurace pro vytváření systémů.

Spring Cloud nabízí vývojářům mnoho možností řešení na základě jejich požadavků. Například lze implementovat pomocí nástrojů Eureka, ZooKeeper nebo Konzul. Komponenty Spring Cloud jsou poměrně oddělené, což přináší požadovanou flexibilitu (Walls, 2015).

2.7.3 Spring Boot

Spring Boot je rozšíření Spring Framework, které nabízí několik vylepšení. Nejznámější z těchto vylepšení je autokonfigurace, kdy Spring Boot umožňuje na základě položek v cestě ke třídě, proměnných prostředí a dalších faktorů provést přiměřené odhady toho, jaké komponenty je třeba nakonfigurovat a zapojit společně. Automatická konfigurace Spring Boot dramaticky snížila množství explicitní konfigurace (ať už pomocí XML, nebo Java) potřebné k vytvoření aplikace. Spring Boot vylepšuje vývoj aplikací natolik, že je těžké si představit vývoj Spring aplikací bez něj.

Kromě automatické konfigurace nabízí Spring Boot také další užitečné funkce. Poskytuje informace o vnitřním fungování aplikace, včetně metrik, informací o výpisu vláken a vlastností prostředí dostupných pro aplikaci. Spring Boot navíc nabízí alternativní programovací model založený na skriptech Groovy, který se nazývá Spring Boot CLI. Pomocí tohoto modelu je možné psát celé aplikace jako kolekci skriptů a spouštět je z příkazového řádku (Walls, 2015).

2.7.4 JavaScript

JavaScript lze definovat jako skriptovací jazyk, který je převážně používán při vývoji grafického rozhraní webových aplikací na straně klienta. S příchodem některých frameworků a podobných technologií však dokáže plnohodnotně figurovat i na straně serveru. Syntaxe JavaScriptu je velmi podobná programovacímu jazyku C, nicméně konvence byly převzaty z programovacího jazyku Java. V době, kdy jsou všechny aplikace transformovány do webové podoby, patří JavaScript mezi nejpoužívanější programovací jazyky na světě. Popularita JavaScriptu neustále stoupá také díky frameworkům, jako jsou React.js, Vue.js nebo Angular.js.

JavaScript umožňuje programátorům efektivně vytvářet grafické rozhraní. Můžou jej využít například pro práci s texty, tvorbu vyskakovacích zpráv nebo validaci vstupů uživatelů bez nutnosti znovunačtení stránky. Mezi největší výhody JavaScriptu patří:

- jednoduchá implementace,
- pracuje na straně klienta přímo v prohlížeči,
- umožňuje vytvářet responzivní grafické rozhraní webové aplikace, které lze přehledně zobrazit na jakémkoli zařízení (tablet, PC, telefon),
- jazyk je možné použít pro opravu chyb způsobených diferenciací prohlížečů.

JavaScript má také své nevýhody. Mezi ty hlavní patří potenciální hrozba zneužití kódu, který je možné zobrazit v prohlížeči, nebo absence možnosti přepisu jiných dat, než jsou cookies (Blumenthal, 2017).

DOM

DOM je rozhraní, které umožňuje JavaScriptu přistupovat k jednotlivým prvkům na webové stránce. Tyto prvky mohou být jednotlivé elementy, tagy, atributy, texty, komentáře nebo vlastnosti jednotlivých prvků. Od dob, kdy toto rozhraní začaly webové prohlížeče podporovat, se otevřel plný potenciál programovacího jazyka JavaScript a je možná daleko vetší a bohatší interakce s webovými stránkami (Blumenthal, 2017).

AJAX

AJAX není samostatný programovací jazyk, jedná se spíše o způsob vytváření interaktivních, lepších a rychlejších webových aplikací. AJAX umožňuje asynchronní aktualizaci webových stránek výměnou dat s webovým serverem v zákulisí. To znamená, že je možné aktualizovat části webové stránky bez opětovného načtení celé stránky (Ford, 2009).

jQuery

JQuery je open-source knihovna, která poskytuje funkcionální programovací rozhraní k JavaScriptu. Celá knihovna je postavena na selektorech jazyka CSS, pracující s elementy modelu DOM.

Syntaxe jQuery je navržena tak, aby usnadnila navigaci v dokumentu, vytváření animací, zpracování událostí a vývoj aplikací Ajax, jQuery také poskytuje vývojářům možnost vytvářet pluginy nad knihovnou JavaScript (Zakas, 2009).

2.7.5 React.js

React.js je javascriptový framework, který je často využíván k vytváření uživatelského rozhraní dynamických webových aplikací. Framework byl vytvořen společností Facebook s cílem vyřešit problémy spojené s velkým objemem dat u rozsáhlých aplikací. Tato knihovna, na rozdíl od některých ostatních tradičních javascriptových knihoven neobsahuje veškeré nástroje, které by každý programátor očekával. React.js se zaměřuje na jednu specifickou část, a to na vytvoření uživatelského rozhraní. V případě potřeby dalších nástrojů pro tvorbu existuje velké množství doplňujících knihoven, které jsou s frameworkem kompatibilní (Banks, Porcello, 2017).

ReactDOM

ReactDOM je doprovodná knihovna, kterou vytvořila (stejně jako knihovnu React) společnost Facebook. Tato knihovna se již nestará o tvorbu uživatelského rozhraní, ale o vykreslení vytvořeného rozhraní do webového prohlížeče uživatele. Jedná se o prostředníka mezi knihovnou React a rozhraním DOM ve webovém prohlížeči. Jednotlivé metody této knihovny nejsou součástí hlavní knihovny Reactu, protože se React nezaměřuje pouze na webové aplikace, ale i na nativní. Pro tyto účely je pak možné použít jinou vykreslovací knihovna (Banks, Porcello, 2017).

JSX

JSX je javascriptové rozšíření, které umožňuje Reactu definovat jeho elementy jednodušeji a také čitelněji, minimálně ve stejné míře jako klasické HTML. JSX bylo vytvořeno spolu s Reactem proto, aby pomáhalo zjednodušit syntaxi při vytváření komplexních DOM stromů s atributy. V JSX jsou elementy znázorněny tagy, stejně jako u HTML. Jednotlivé atributy těchto elementů jsou reprezentovány jako vlastnosti a každý element může mít libovolný počet dětí. Každý element musí mít otevírací a zavírací tag (Banks, Porcello, 2017).

Webpack

Při dokončení aplikace je potřeba vyřešit problém, jak dostat všechny dané soubory do webového prohlížeče. Pro tyto účely slouží právě Webpack, který představuje sdružovač modulů. Tento sdružovač modulů zabalí všechny naše různé soubory, které jsme použili pro vývoj, a promění je v jeden soubor. Tento proces má dvě hlavní výhody, a to modularitu a následný výkon sítě.

Modularita umožňuje rozdělit kód na části neboli moduly, se kterými se poté jednodušeji pracuje. Výkon sítě je následně ovlivněný tím, že jsou načteny pouze moduly, které jsou potřeba (Banks, Porcello, 2017).

Redux

Redux je samostatná knihovna, která byla vyvinutá mimo knihovnu React a následně přijatá jako nejlepší řešení problému se změnou dat, která procházejí aplikací. Redux je malá knihovna, obsahuje pouze 99 řádků kódu. Redux pomáhá vytvořit centralizované místo všech potřebných dat, která jsou potřeba napříč celou aplikací. Pro práci s proměnnými je potřeba dodržovat určitá pravidla, díky kterým je zajištěna jejich změna v předvídatelném běhu (Banks, Porcello, 2017).

2.8 Systémy pro zasílání zpráv – Message Brokers

Systém zasílání zpráv (Message broker) je software, který umožňuje aplikacím, systémům a službám vzájemně komunikovat a vyměňovat si informace ve formě zpráv. Komunikace se provádí překládáním zpráv mezi formálními protokoly zpráv. To umožňuje vzájemně závislým službám/aplikacím spolu komunikovat, i když byly psány v různých jazycích nebo implementovány na různých platformách. Message broker umožňuje ověřovat, ukládat, směřovat a doručovat zprávy do příslušných cílů (Garg, 2015).

Aby se zajistilo spolehlivé ukládání a zaručené doručení zpráv, systémy Message broker často využívají podstrukturu nazývanou frontu zpráv (message queue). Fronta zprávy ukládá, dokud je aplikace nezpracuje. K zasílání zpráv se využívá asynchronní komunikace. Asynchronní zasílání zpráv zabraňuje ztrátě dat a umožňuje aplikacím nadále fungovat i v případě různých problémů, například při ztrátě připojení.

Aktuálně jsou nejčastěji využívanými systémy pro zasílání zpráv Apache Kafka nebo RabbitMQ (Garg, 2015).

2.8.1 Apache Kafka

Apache Kafka je distribuovaný, na oddíly rozdělený a replikovaný systém zasílání zpráv, navržený především s následujícími vlastnostmi:

- **trvalé zasílání zpráv** – při zasílání velkého množství jakéhokoliv druhu dat je potřeba zabránit ztrátě informací,
- **vysoká propustnost** – potřeba zvládnout obrovské množství čtení a zápisů za sekundu z velkého počtu zdrojů,

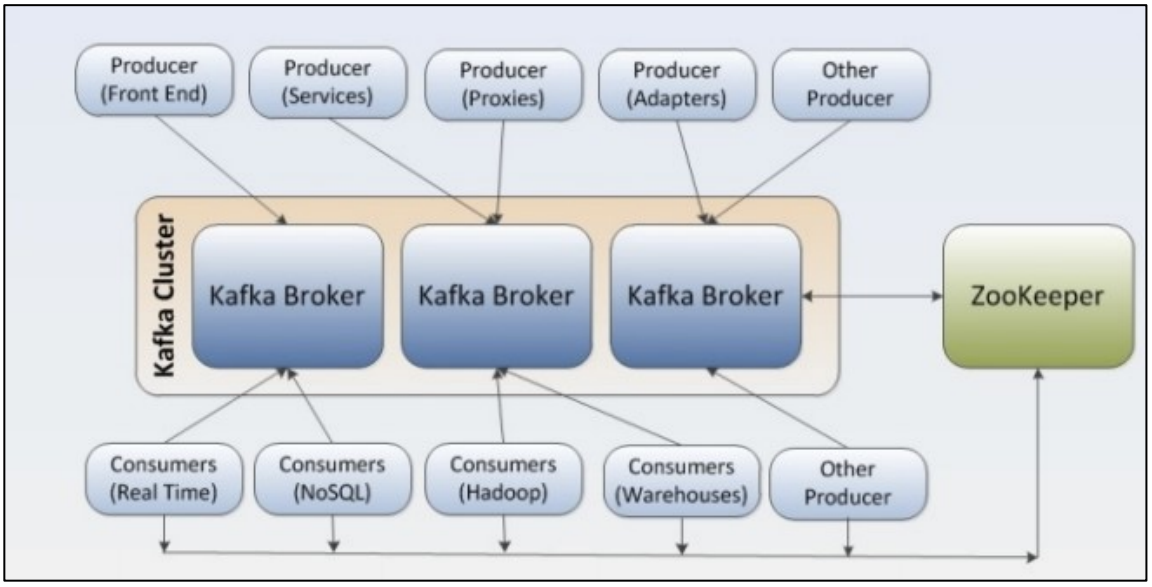
- **podpora více klientů** – snadná integrace z různých platforem, například Java, .NET, PHP, Ruby, Python,
- **zasílání zpráv v reálném čase** – zprávy produkované vlákny odesílatelů by měly být okamžitě viditelné pro vlákna spotřebitelů. Tato funkce je zásadní pro systémy založené na událostech.

Platforma poskytuje řešení publikování a odběru zpráv, které překonává výzvy spotřebování velkého objemu dat v reálném čase. Umožňuje posílat velké množství malých zpráv napříč servery, přičemž umožňuje horizontální škálování a zároveň všechny zprávy replikuje na více serverů – vypadne-li jeden z nich, jiný ho nahradí.

Kafka má tři základní části. *Producer*, který posílá zprávy do Kafky, *broker*, jenž tyto zprávy uchovává, a *consumer*, který tyto zprávy čte. Základní jednotkou, se kterou Kafka pracuje, je téma. Pro různé typy zpráv je možné založit různá témata.

Následující diagram ukazuje typický scénář agregace a analýzy velkých dat podporované systémem zasílání zpráv Apache Kafka (Garg, 2015).

Obrázek 2.6 - Diagram systému zpráv Apache Kafka



Zdroj: Garg, 2015

3 Analýza současného stavu a vymezení požadavků

3.1 Společnost eCENTRE, a.s.

Společnost eCENTRE, a.s., pro kterou je webová aplikace vyvíjena, se zabývá sdruženou poptávkou komodit a optimalizací nákupů pro B2C i B2B sektor včetně zajišťování nákupu komodit v rámci zákona o zadávání veřejných zakázek, tedy pro veřejný sektor. Prostřednictvím elektronických aukcí, které organizuje, se společnosti podařilo ušetřit peníze na elektrické energii a zemním plynu více než 50 000 klientů.

V současné době firma rozšířila svůj seznam „služeb“ o platební terminály, kde se snaží (stejným způsobem jako u energií) také ušetřit peníze zákazníkům. Společnost má celkově 25 stálých zaměstnanců, zaměstnaných na hlavní pracovní poměr, a několik brigádníků, kteří mají podepsanou dohodu o provedení práce.

3.1.1 Infrastruktura společnosti

Společnost funguje převážně na interně vyvíjených aplikacích. Jádrem celého informačního systému je výrobní systém. Jedná se o robustní systém, pomocí kterého je spravována většina obchodních a výrobních procesů. Celý systém tvoří několik menších, avšak vzájemně (minimálně datově) propojených aplikací.

Další část informačního systému tvoří webová aplikace, kterou primárně využívá obchodní síť k evidenci svých zákazníků a generování smluv. Pomocí této webové aplikace každý obchodní zástupce vidí své smlouvy a jejich aktuální stavy. Je také možné vytvářet cenová porovnání potenciálním zákazníkům.

Do značné míry odděleným systémem je tzv. aukční síň. Jedná se o aplikaci určenou pro pověřené zaměstnance a jednotlivé účastníky aukce (dodavatele). Zde probíhá celý proces vyvolávání nabídek.

Správa klientů je řízena pomocí desktopové aplikace, která je součástí výrobního systému. Aplikace umožňuje administrovat klienty, uzavřené smlouvy a obchodní zástupce společnosti.

Společnost také využívá externí softwarové nástroje. Jedním z nich je již nepoužívaný, ale stále udržovaný, CRM systém ZOHO. Pro e-mailovou rozesílku používá marketingové oddělení externí systém SmartEmailing. Pro vedení účetnictví je využíván komerční ekonomický systém, do kterého jsou data pro fakturaci exportována jednou měsíčně z výrobního systému.

3.1.2 Technologické parametry společnosti

Většina aplikací je vyvíjena interními programátory, kteří dané aplikace následně spravují a upravují. Převážně se jedná o desktopové aplikace napsané v programovacích jazycích C#, Visual Basic nebo Java.

Data jsou ukládaná pomocí databázového systému Microsoft SQL Server 2012. Zaměstnanci rovněž využívají firemní intranet, kde jsou ukládány interní soubory a dokumenty. Data pro fakturaci jsou do ekonomického systému exportována pomocí webového API.

Firma v současné době nevyužívá žádné externí úložiště (Cloud).

3.2 Cíle webové aplikace

Hlavním problémem, který v současné době trápí vyšší management společnosti, je špatná orientace v podnikových procesech a datech. Řízení a zpracování jednotlivých procesů je rozděleno do relativně velkého počtu malých desktopových aplikací, které jsou následně obhospodařovány většinou malou skupinou specialistů na danou problematiku. Pro implementaci nového řešení existují tedy dva hlavní cíle:

- **technický** – extrahovat data do jednoho funkčního celku a získat tak souhrnný přehled o situaci v podniku,
- **obchodní** – navrhnout a vytvořit řešení, které podniku napomůže při hodnocení cenové citlivosti zákazníků a rozhodování v rámci strategického řízení.

3.3 Vymezení požadavků webové aplikace

Společnost eCENTRE, a.s. definovala dva základní požadavky pro nové softwarové řešení. Prvním požadavkem je vytvořit uživatelské rozhraní pro správu a získání reportovaných informací z interních i externích zdrojů. Dále je požadováno implementovat samostatný modul, který bude představovat nástroj pro podporu rozhodování, využívaný managementem společnosti.

Společnost eCENTRE, a.s. požaduje:

- implementovat sofistikovaný systém pro zčásti automatické a systematické hodnocení dodavatelů na základě reálných dat,
- vypracovat přehledný reportovací nástroj, který společnosti přinese lepší vhled do aktuálního dění v podniku a na trhu,
- implementovat funkce, které umožní prodejnímu týmu eCENTRE, a.s. snadný přehled o očekávaném dopadu změny ceny pro zákazníky,
- přenesení odpovědnosti rozhodování na vedoucí pracovníky jednotlivých úseků firmy,
- řešení musí být připraveno na budoucí vylepšení o další služby, balíčky nebo modely,
- využití moderních technologií, které jsou vhodné pro tento typ webové aplikace.

4 Návrh a implementace softwarového řešení

V této části diplomové práce budou popsány všechny navržené a implementované funkční záležitosti aplikace. Kapitola obsahuje podrobný popis návrhu celého řešení včetně use-case diagramu a diagramu komponent aplikace. Druhá polovina kapitoly je věnována popisu funkční specifikace implementovaných komponent webové aplikace.

4.1 Použité technologie

Všechny komponenty webové aplikace jsou vyvíjeny na základě specifikovaných požadavků společnosti eCENTRE, a.s. Podle definovaných požadavků byly pro implementaci použity technologie zobrazené v tabulce 4.1.

Tabulka 4.1 - Seznam použitých technologií

Technologie	Verze	Účel/popis	Výrobce
PostgreSQL Server	12.1	Relační databáze	The PostgreSQL
Spring Boot	2.1.6	Aplikační služby	Global Dev. Grup
React	16.8.6	Klientské GUI	Facebook, Inc.
Boostrap	4.3.1	Klientské GUI	Twitter
Recharts	1.6.2	GUI grafy	Recharts Group
Open JDK	1.8u201	Java platform for app services	Oracle
Docker	18.09	Container Platform	Docker, Inc.
Kubernetes	1.14	Řízení řešení	The Kubernertes

Zdroj: vlastní

Každý z funkčních modulů webové aplikace je implementován v třívrstvé architektuře.

Datové úložiště je implementováno pomocí objektově-relační databáze PostgreSQL. Důvodů, proč byl vybrán právě tento databázový systém, je hned několik. PostgreSQL je ideální pro náročné analytické procesy, provádění složitých dotazů a skladování velkého objemu dat. Webová aplikace má navíc sloužit jako analytický nástroj, který je oddělený od transakčních systémů podniku. Data z těchto systémů pouze čte, což umožňuje využít jiný databázový systém, než který je ve firmě doposud používán.

Služby aplikační vrstvy modulu provádějící obchodní logiku jsou implementovány v programovacím jazyku Java jako webové služby REST na platformě Spring Boot.

Prezentační vrstva modulu (GUI) je implementována pomocí rámce JavaScript React a Boostrap. Prezentace grafických dat se spoléhají na komponenty Recharts. GUI rozhraní je velmi efektivní, intuitivní a uživatelsky přívětivé.

Apache HTTP server se používá jako proxy server pro přístup ke službám řešení z uživatelských zařízení. Proxy server také slouží jako ochranná vrstva pro moduly řešení a služby, ke kterým uživatelé přistupují z internetu.

Řešení bylo navrženo pro nasazení aplikace do kontejnerové infrastruktury a implementaci modulů řešení (instance modulů) jako kontejnerů Docker. Kontejnerová infrastruktura bude automaticky spravována platformou Kubernetes.

4.2 Architektura webové aplikace

Jak již bylo zmíněno v předchozí kapitole, webová aplikace je implementována v souladu s principy třívrstvé architektury.

Prezentační vrstva neboli grafické uživatelské rozhraní je optimalizováno především pro webové prohlížeče. Webová aplikace obsahuje poměrně velké množství obrazovek s tabulkami, kvůli kterým je obtížné implementovat responzivní design, a proto aplikace není vhodná pro telefonní zařízení.

Aplikační vrstva zajišťuje výpočty a zpracování dat mezi vstupně-výstupními požadavky pomocí webových služeb založených na REST technologii. Na aplikační vrstvě nejsou žádné objekty relace (session). Jednotlivé moduly aplikační vrstvy odpovídají vzoru mikroslužeb a běží jako kontejnery v kontejnerové orchestraci.

Databázová vrstva zajišťuje ukládání, výběr, agregaci, předzpracování, integritu a audit dat. Automatickou konverzi dat mezi relační databází a aplikační vrstvou zajišťuje objektově-relační mapovací nástroj (ORM) Hibernate. Žádná aplikační logika není implementována v databázové vrstvě.

4.2.1 Komponent diagram

Základní komponenty řešení i se vztahy na interní systémy podniku jsou znázorněny v následujícím schématu (viz obrázek 4.1). Logické komponenty jsou znázorněny růžovou barvou, integrované systémy jsou označeny žlutě.

Webová aplikace je sestavena ze dvou základních funkčních modulů:

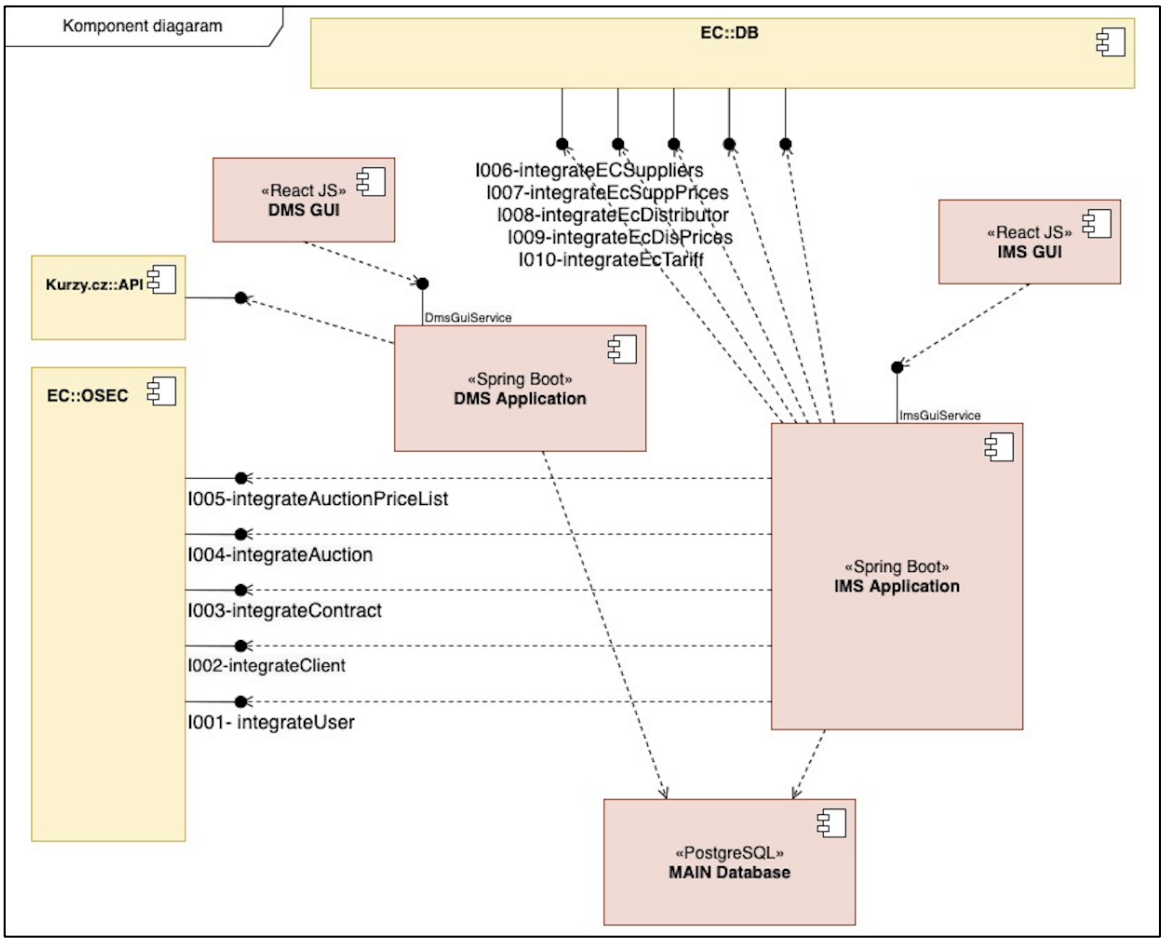
- **správa informací** (Information Management System – IMS) – modul je zodpovědný za správu klientů, dodavatelů, distributorů a aukcí včetně jejich ceníků či smluv,
- **systém pro podporu rozhodování** (Decision Management System – DMS) –modul umožňuje analyzovat tržní prostředí podniku a provádět výpočty cen pro nastavenou skupinu klientů.

Každý z uvedených modulů má své grafické rozhraní (IMS GUI, DMS GUI). Komponenty grafického rozhraní využívají služby poskytované vlastním modulem aplikace.

Řešení aplikace je doplněno o integrační server, který je zodpovědný za komunikaci s podnikovými systémy pomocí Kafka Message brokeru.

Data jsou uložena v jedné databázové instanci. Tabulky jsou rozděleny do schémat podle cílového modulu (*ims_schema*, *dms_schema*).

Obrázek 4.1 - Komponent diagram

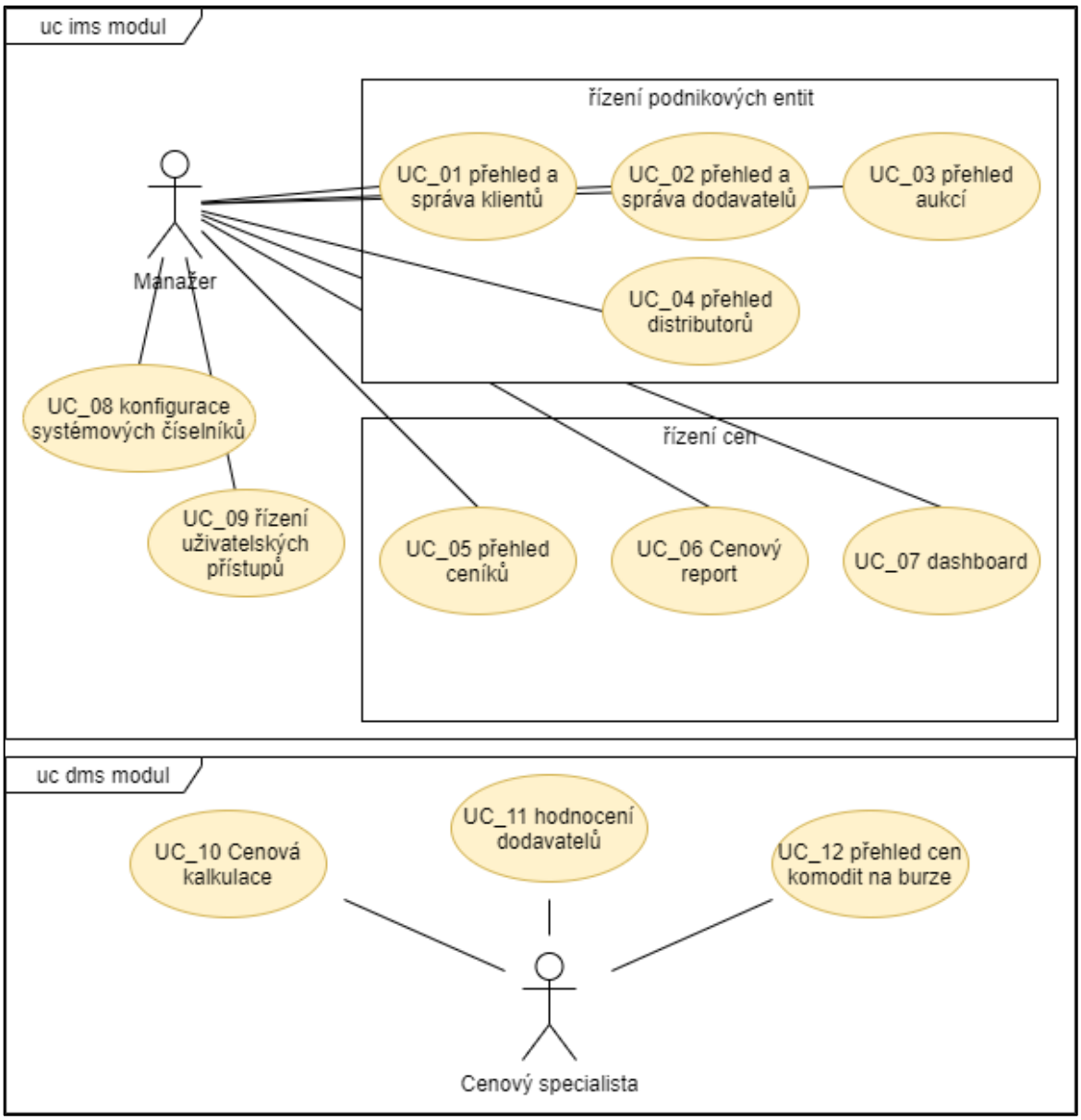


Zdroj: vlastní

4.3 Use Case diagram

Tato kapitola popisuje všechny aktéry a jejich případy užití, které jsou relativní pro specifické funkční moduly webové aplikace. Případy užití byly vytvořeny na základě definovaných požadavků společnosti. Zda funkcionality webové aplikace zajišťují všechny požadavky, je sledováno pomocí vytvořené RTM matice. Vytvořený Use Case diagram pro oba moduly je možné vidět na obrázku 4.2.

Obrázek 4.2 - Use Case diagram modulu IMS



Zdroj: vlastní

4.3.1 Aktéři

Manažer představuje obecnou roli všech zaměstnanců, kteří jsou oprávnění používat modul IMS. Ve většině případů se bude jednat o analytiky, konzultanty nebo vedoucí pracovníky podniku. Tito zaměstnanci jsou zodpovědní za správu všech nástrojů používaných k vytváření a nastavení podkladů pro reporting či kalkulace. Manažer stanovuje limity a pravidla pro další uživatele, například pro marketingové manažery. Tento aktér v aplikaci odpovídá roli *IMS_USER*.

Cenový specialista reprezentuje uživatele webové aplikace, který má plná oprávnění pro práci s modulem DMS. Je odpovědný za vytváření cenových kalkulací a hodnocení dodavatelů. Cenový specialista je jediným aktérem, který používá modul DMS. Tento aktér v aplikaci odpovídá roli *DMS_USER*.

4.3.2 Matice sledovatelnosti požadavků

Matice sledovatelnosti požadavků (dále pouze RTM matice) je dokument, který mapuje a sleduje požadavek uživatele pomocí případů užití. Zachycuje všechny funkční požadavky definované společnosti při návrhu webové aplikace. Hlavním účelem RTM je ověřit, že všechny požadavky jsou kontrolovány pomocí případů užití tak, aby během testování softwaru nebyla zrušena žádná funkce.

Hlavní požadavky jsou složeny z několika detailnějších požadavků, které je možné rozdělit na funkční a nefunkční. Mezi nefunkční požadavky patří například zajištění výkonu, spolehlivosti, rozšiřitelnosti, udržitelnosti nebo bezpečnosti webové aplikace. Tento typ požadavku však není předmětem této kapitoly.

Tato kapitola se zaměřuje na funkční požadavky, které identifikují nezbytné úkoly, akce nebo činnosti pro splnění cílů webové aplikace. Společnost definovala tyto funkční požadavky:

- P1 – správa dodavatelů, distributorů, aukcí a jejich ceníků,
- P2 – správa klientů, smluv a odběrných míst,
- P3 – cenová kalkulace pro definované skupiny klientů,
- P4 – přehled nejlepších cenových nabídek dodavatelů,
- P5 – grafické zobrazení vývoje cen na komoditní burze.

Díky RTM matici je možné ověřit, zda navrhované řešení zohledňuje všechny požadavky. Jak je možné vidět v tabulce 4.2, všechny požadavky byly při návrhu funkční specifikace zohledněny. Případy užití UC_08 a UC_09 nejsou v matici zobrazeny, protože se jedná o systémové funkcionality, které se nevztahují k žádnému z požadavků. Jsou však potřebné pro fungování aplikace.

Tabulka 4.2 - RTM matice

RTM	Případy užití									
Pož.	UC_1	UC_2	UC_3	UC_4	UC_5	UC_6	UC_7	UC_10	UC_11	UC_12
P1		X	X	X	X	X	X			
P2	X					X	X			
P3								X		
P4									X	
P5										X

Zdroj: vlastní

4.4 Návrh databázové struktury

Webová aplikace má primárně sloužit jako analytický nástroj podnikových dat. Odpovědný uživatel má být na základě vhodně prezentované formy schopný efektivně transformovat data na informace. Proto bylo nutné dobře navrhnout a následně implementovat databázovou vrstvu aplikace.

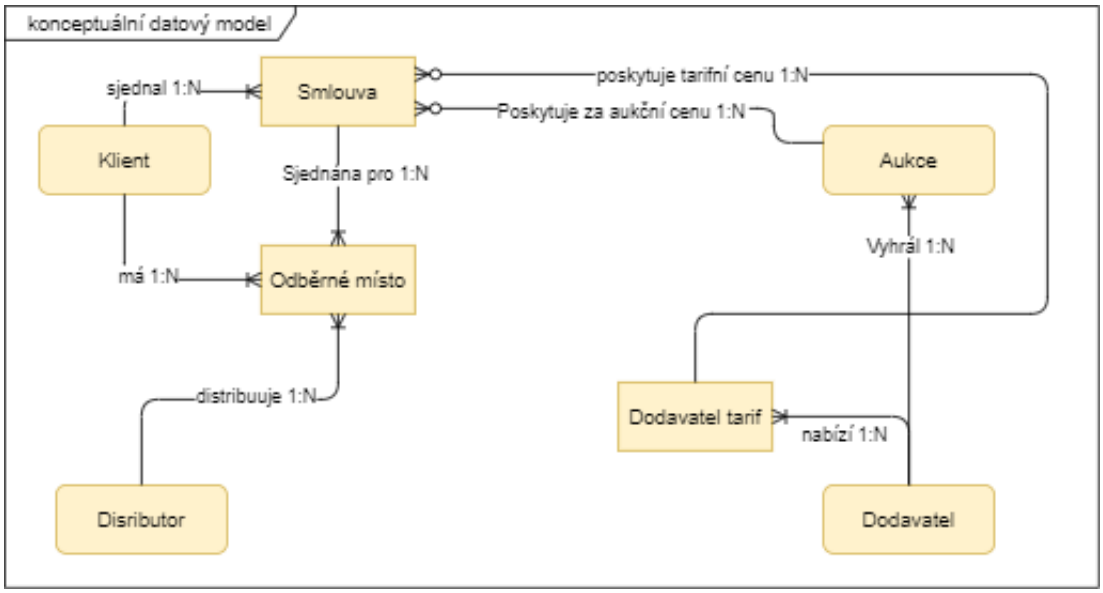
První část této podkapitoly je věnována návrhu konceptuálního datového modelu, pomocí kterého byly definovány potřebné entity ve vztahu k podniku.

Na základě konceptuálního datového modelu byla následně navrhnuta finální databázová struktura obsahující konkrétní tabulky a vztahy mezi nimi.

4.4.1 Konceptuální datový model

Společně s vedením a IT pracovníky podniku, si bylo potřeba určit, co má nová webová aplikace zobrazovat, a podle jakých dat. Na základě této komunikace byl navržen konceptuální datový model, který je možné vidět na obrázku 4.3.

Obrázek 4.3 - Konceptuální datový model



Zdroj: vlastní

Webová aplikace obhospodařuje čtyři základní entity – *klient*, *aukce*, *distributor* a *dodavatel*. Entita *klient* představuje všechny zákazníky podniku, kterým byla sjednána alespoň jedna smlouva. Každý klient může mít několik odběrných míst, pro která byla smlouva sjednána. Smlouvu je možné sjednat pro jedno nebo více odběrných míst.

Každému odběrnému místu je přidělený *distributor* podle regionu, ve kterém se nachází. Distributoři se starají o fyzický přenos komodity od výrobce až do místa spotřeby. Cena za dodávku je regulována Energetickým regulačním úřadem (dále pouze ERÚ).

Entita *dodavatel* představuje obchodníka na komoditním trhu, který přeprodává koncovým zákazníkům elektrickou energii nebo zemní plyn. Dodavatel nabízí zákazníkům nabídky ve formě tarifů.

Dodavatel se může zúčastnit také elektronické aukce, kde nabízí mimotarifní ceny. Nejnižší cenovou nabídku všech účastníků následně společnost poskytuje svým klientům.

Ceny distributorů, dodavatelů a aukcí se liší podle objemu spotřeby, typu subjektu, komodity a dalších pravidel. Souhrnná nabídka těchto cen je klientům prezentována ve formě ceníku. Do konceptuálního modelu to však pro větší přehled nebylo zahrnuto.

4.4.2 Logický datový model

V rámci návrhu logické struktury dat byly entity z předchozí kapitoly transformovány do tabulek obsahující konkrétní atributy, datové typy a další pravidla.

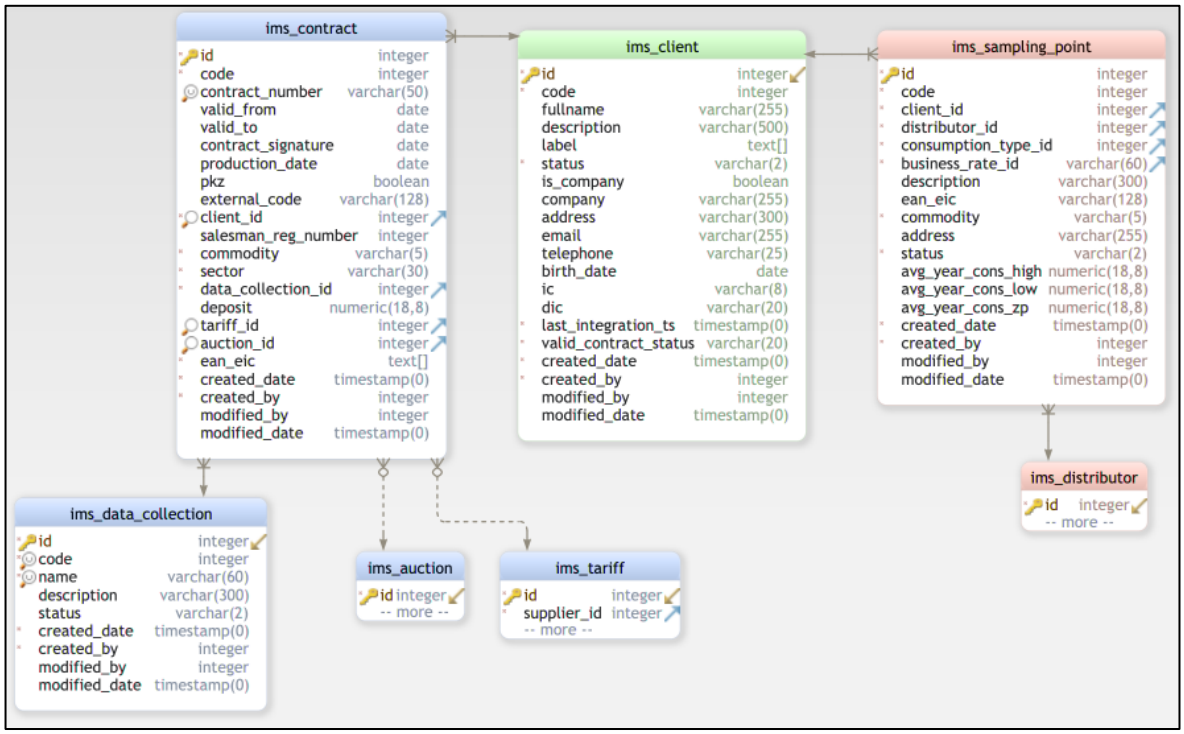
Navržené tabulky jsou rozděleny do schémat podle cílového modulu – *ims_schem*a, *dms_schema*. První zmíněné schéma zahrnuje tabulky reprezentující základní podnikové entity a vztahy mezi nimi. Databázové schéma *dms_schema* obsahuje tabulky určené primárně pro ukládání dat výpočtů cenových kalkulací.

Všechny navržené tabulky obsahují atributy pro sledování toho, kdo vytvořil nebo změnil daný záznam, a v jakém okamžiku k tomu došlo. Atributy *created_by* a *modified_by* obsahují identifikační kód uživatele z tabulky *ims_user*. V této tabulce jsou uložení všichni uživatelé, kterým je umožněn přístup do webové aplikace. Zároveň se zde nachází i tzv. systémoví uživatelé, jako například *integration_user* spravující integraci dat do aplikace. Do atributů *created_date* a *modified_date* je vždy vloženo datum a čas operace.

Logickou databázovou strukturu modulu IMS je možné rozdělit do několika skupin. První skupina představuje konfigurační tabulky obsahující záznamy číselníků, které jsou společností udržovány. Přesný popis dat uložených v tabulkách, je uveden v kapitole 4.7.

Druhá skupina představuje tabulky spravující klientské záznamy. Zde jsou uloženy záznamy všech klientů společnosti, jejich smluv a odběrných míst.

Obrázek 4.4 - Datový model – správa klientů



Zdroj: vlastní

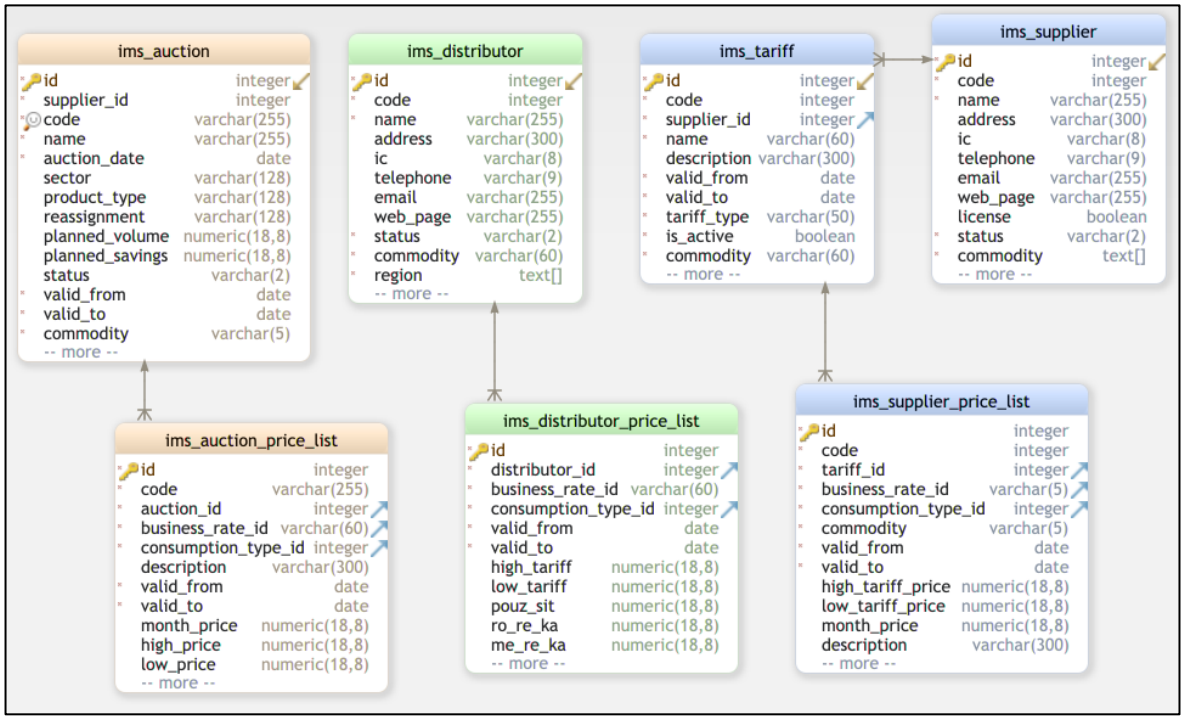
V tabulce smluv *ims_contract* mohou existovat záznamy představující smlouvy, jež klient sjednal jinak než prostřednictvím aukce. Proto byly v tabulce vytvořeny atributy *auction_id* a *tariff_id*, které musí splňovat následující podmínky:

- pokud byla smlouva sjednána prostřednictvím aukce, musí být vyplněn atribut *auction_id* jako cizí klíč k tabulce *ims_auction* a zároveň musí atribut *tariff_id* nabývat hodnoty NULL,
- pokud byla smlouva sjednána jiným způsobem, musí být vyplněn atribut *tariff_id* jako cizí klíč k tabulce *ims_tariff* a zároveň musí atribut *auction_id* nabývat hodnoty NULL.

Atributy *avg_year_cons_high* a *avg_year_cons_high* v tabulce odběrných míst představují průměrnou spotřebu elektřiny nebo zemního plynu. Uložené hodnoty v těchto atributech jsou využívány pří výpočtu úspor v modulu DMS.

Poslední skupina databázového schématu modulu IMS představuje tabulky obsahující záznamy dodavatelů, distributorů, aukcí a jejich ceníků. Data v tabulkách jsou primárně využívána pro kalkulace a souhrnné přehledy.

Obrázek 4.5 - Datový model entit a ceníků



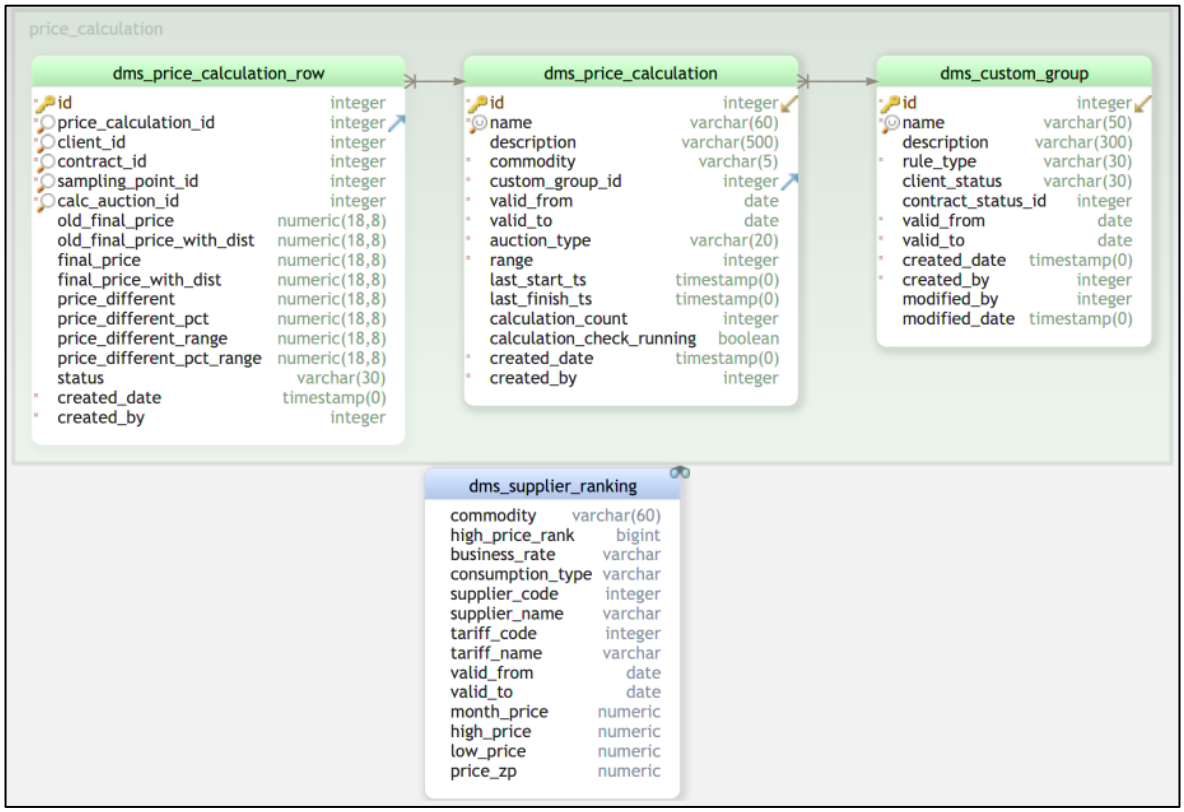
Zdroj: vlastní

Schéma *dms_schema* obsahuje tabulky určené pro ukládání dat ohledně kalkulací a jejich parametrů. Při návrhu datového modelu bylo nutné počítat s velkým objemem dat, a proto bylo potřeba použít některé mechanismy pro zrychlení, například indexování. Celé schéma obsahuje pouze tři tabulky a jeden SQL pohled.

SQL pohled *dms_supplier_ranking* obsahuje seznam všech dodavatelů uvedených v modulu IMS. Ke každému dodavateli je zde zobrazena nejlepší cenová nabídka pro danou komoditu, distribuční sazbu a typ odběru. SQL pohled je volán při každém požadavku uživatele z grafického rozhraní.

Zbývající tři tabulky slouží výhradně pro uložení dat vztahujících se k cenové kalkulaci. Tabulka *dms_custom_group* obsahuje uživateli definované skupiny klientů, pro které bude provedena cenová kalkulace. Nastavené parametry cenové kalkulace jsou obsaženy v tabulce *dms_price_calculation*. Vypočtené hodnoty jsou ukládány do tabulky *dms_price_calculation_row*.

Obrázek 4.6 - Datový model – schéma DMS



Zdroj: vlastní

4.5 Integrace dat

Úvod této kapitoly obsahuje popis implementace integračního serveru. Funkčnost jednotlivých služeb rozhraní, které jsou implementovány v modulu IMS, je popsána v jednotlivých podkapitolách. V druhé části je popsán proces integrace dat z podnikové databáze.

4.5.1 Integrační server

Integrační server zprostředkovává komunikaci mezi platformou *oSec_Kafka* a službami webové aplikace. Integrační server naslouchá tématům Kafka a pro každou obdrženou zprávu vyvolá implementovanou webovou službu v modulu IMS. Poté, co je zpráva službou úspěšně zpracována, integrační server zprávu potvrdí. Po úspěšném potvrzení je zvýšen indikátor fronty zpráv. Jeden integrační server slouží více tématům Kafka.

Funkce integračního serveru je obecná. Směruje zprávy do nakonfigurované služby aplikace bez použití jakékoli logiky, transformace atd. Proces integrace dat je implementován v podobě REST služby v modulu IMS. Každé téma Kafka je obsluhováno vyhrazeným vláknem. Tato vlákna běží v nekonečné smyčce.

Bylo požadováno, aby existoval právě jeden integrační server, který je přihlášen k odběru každého tématu. Pokud dojde k chybě na jednom z definovaných témat, implementace integračního serveru podpoří jeho dočasné pozastavení zpracování zpráv nezávisle na ostatních tématech.

Jeden integrační server se připojuje k jednomu klastru Kafka. V případě, že bude existovat více klastrů Kafka, bude spuštěno více instancí integračního serveru. Mezi integračním serverem a službami modulů je používaná interní identita aplikace k vyvolání služby. Integrační server vyvolá službu rozhraní pomocí požadavku REST/HTTP.

V tabulce 4.3 je možné vidět seznam všech integračních rozhraní včetně názvu témat a krátkého popisu. V příloze 3 je navíc možné vidět implementaci integrační služby *integrateClient()*.

Tabulka 4.3 - Seznam integračních služeb Kafka

ID	Název integrace	Téma Kafka	Popis
I001	integrateUser()	ec.ars.user	Integrace uživatelů
I002	integrateClient()	ec.ars.client	Integrace klientů
I002	integrateContract()	ec.ars.contracts	Integrace smluv daného klienta
I002	integrateAuction()	ec.ars.auction	Integrace aukcí
I002	integrateAuctionPriceLists()	ec.ars.price.list	Integrace aukčních ceníků

Zdroj: vlastní

Implementace integračních služeb

Všechny integrační služby byly implementovány pomocí stejných principů. Služby se tedy mění pouze v konkrétních podmínkách, ale struktura je stejná.

Každá zpráva nese úplný stav entity. Jeden z atributů každé zprávy obsahuje časové razítko změny ve zdrojovém systému (oSec). Pokud je příchozí zpráva starší než aktuální stav entity v databázi, zpráva je ignorována. Pokud příchozí zpráva nese novější časové razítko než záznam v cílové tabulce, záznam je aktualizován obsahem zprávy.

Pokud existují v cílové tabulce kontrolní sloupce (created_by, modfied_by), jsou sloupce vyplněny výchozí hodnotou −100. Tato hodnota představuje ID systémového uživatele *integration_user,* který slouží právě pro tyto účely. V databázi je pak možné dobře rozeznat, co bylo měněno integrací a co manuálně uživatelem.

V případě, kdy existují dvě tabulky se vztahem FK, je zpráva odmítnuta. Není tedy možné uložit záznam, který v databázi webové aplikace nemá všechna potřebná dat. Je tak dosaženo konzistence a úplnosti dat používaných k analýze.

Služba se spouští synchronně a zasílá výsledek zpracování prostřednictvím stavového kódu http na volající server integrace. Logika služby musí být konzistentní. Zpráva je buď zpracována, nebo odmítnuta. V případě selhání jsou všechny změny v databázi (způsobené integrační zprávou) vráceny zpět.

4.5.2 Integrace dat z podnikové databáze

Tato implementovaná logika webové aplikace umožňuje importovat data z podnikové databáze (MSSQL) do interních tabulek webové aplikace. Zdrojová data jsou publikována jako databázové pohledy vytvořené pouze pro tyto účely. Součástí této práce nebylo vyhotovení databázových pohledů. Struktura datových kolekcí byla však specifikována společnými silami na základě komunikace s IT pracovníky podniku.

Proces je spouštěn IT pracovníky jako jednorázová úloha a končí po načtení a uložení dat do databáze webové aplikace. Proces zpracovává následující datové kolekce:

Tabulka 4.4 - Seznam integrací z datového skladu

ID	Název integrace	Cílová tabulka	Popis
I006	integrateEcSuppliers()	ims_supplier	Aktualizace dodavatelů
I007	integrateEcSuppPrices()	ims_supplier_pricelist	Aktualizace ceníku jednotlivých tarifů
I008	integrateEcDistributor()	ims_distributor	Aktualizace seznamu distributorů
I009	integrateEcDisPrices()	ims_distributor_pricelist	Aktualizace ceníků distributorů
I010	integrateEcTariff()	ims_tarrifs	Aktualizace tarifů jednotlivých dodavatelů

Zdroj: vlastní

Pro každý zdroj dat byla implementována samostatná služba. Vzhledem k tomu, že jsou zdrojová data uložena pomocí databázového systému Microsoft SQL 2012, bylo potřeba vyřešit rozdíly v syntaxi oproti PostgreSQL databázi. Z tohoto důvodu jsou služby implementovány v Javě s využitím Spring JDBC pro připojení do zdrojové databáze. Služba se tak dotazuje na danou datovou kolekci přímo pomocí jednoduchého SQL dotazu a data v příslušné formě aktualizuje do interních tabulek aplikace.

Při každém navázaném připojení do databáze se každá služba musí autentizovat prostřednictvím databázového účtu vytvořeného konkrétně pro tyto účely. Tento databázový účet má přidělenou konkrétní roli, které je umožněno pouze data číst.

Konfigurační parametry včetně přístupů jsou uloženy v souboru application.yml.

4.6 Rozvržení webové aplikace

Pro získání lepšího přehledu v aplikaci byl vytvořen Screen-flow diagram. Tento diagram ilustruje tok jednotlivých obrazovek ve funkčních částech webové aplikace.

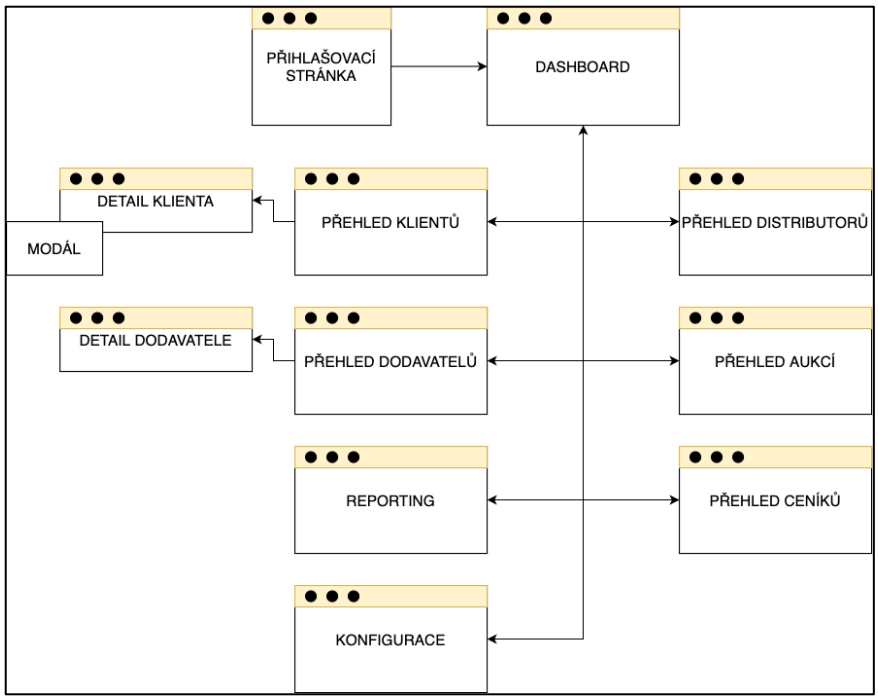
4.6.1 Screen-flow diagram – IMS

Modul IMS je základním kamenem webové aplikace, do kterého je převážně implementována logika pro práci s daty. Většina obrazovek tedy slouží k zobrazení dat a jiných atributů.

Při zadání adresy webové aplikace do internetového prohlížeče je uživateli zobrazena přihlašovací stránka, prostřednictvím které je možné vstoupit do aplikace. Po zadání validních údajů je uživatel přesměrován na hlavní stránku ve formě dashboardu. Z této části webové aplikace může uživatel přejít na dalších obrazovky aplikace: klienti, dodavatelé, distributoři, aukce, ceníky, report a konfigurace.

Hlavní komponentou většiny těchto obrazovek je tabulka. Výběrem záznamu v tabulce klientů nebo dodavatelů je uživatel přesměrován na obrazovku detailu. Pro hromadnou editaci záznamů v tabulce byla do aplikace implementován modál pro validaci.

Obrázek 4.7 - Screen-flow diagram modulu IMS



Zdroj: vlastní

4.6.2 Screen-flow diagram– DMS

Uživatel má umožněn přístup do modulu DMS pouze pokud má veškerá oprávnění. Jednotlivé sekce DMS jsou zobrazena v menu aplikace. V opačném případě tyto sekce uživatel vůbec nevidí. DMS se skládá ze tří hlavních obrazovek, mezi kterými je možné libovolně přecházet. Pokud má uživatel na dané obrazovce nepotvrzené úpravy, bude před odchodem upozorněn.

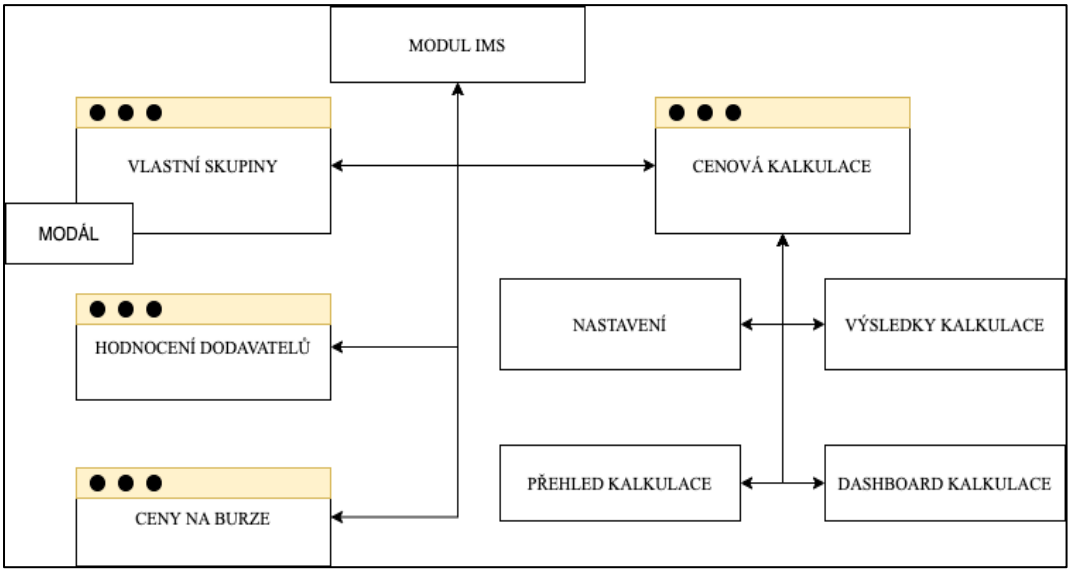
Pomocí obrazovky „cenová kalkulace“ je uživatel schopný administrovat cenové kalkulace klientů, obrazovka je složena ze čtyř částí, které slouží k nastavení, revizi a zobrazení výsledků kalkulací.

Při nastavování cenových kalkulací je možné vybrat konkrétní skupinu klientů, kteří budou zohledněni v kalkulaci. Pro zobrazení všech takových skupin a případné vytvoření nové skupiny slouží obrazovka „vlastní skupiny“. Zvolením možnosti „vytvoření nové skupiny“ se zobrazí se na obrazovce tomu určený modál.

Další obrazovka v rámci modulu DMS slouží k zobrazení setříděného seznamu dodavatelů podle daných parametrů. Uživatel tak může kontrolovat aktuální stav cen na trhu s elektřinou nebo zemním plynem.

Poslední obrazovka „burzovní přehled“ slouží k zobrazení historického vývoje cen dané komodity na evropské burze. Data jsou zobrazena ve formě spojnicového grafu.

Obrázek 4.8 - Screen-flow diagram modulu DMS



Zdroj: vlastní

4.7 Popis implementované logiky webové aplikace

V této kapitole diplomové práce jsou popsány stěžejní komponenty, funkcionality a procesy webové aplikace. Cílem návrhu a následné implementace aplikace bylo pokrýt všechny definované požadavky společnosti eCENTRE, a.s.

V první části kapitoly jsou popsané obecné funkcionality, které jsou využívány napříč všemi částmi webové aplikace. Následně jsou popsány funkcionality a procesy konkrétních modulů, do nichž je webová aplikace rozdělena.

4.7.1 Obecné stavební bloky webové aplikace

Obecnými stavebními bloky je myšlena běžná funkcionalita řešení, která se opakovaně používá ve všech modulech. Stavební bloky nejsou samostatné komponenty, jsou spíše zabudovány do funkčních modulů řešení. Hlavním účelem těchto stavebních bloků je opětovné použití kódu a úspora nákladů na implementaci. Mezi obecné stavební bloky patří:

- **editor zdroje dat** – umožňuje uživateli aktualizovat atributy entit,
- **mechanismus načítání dat** – funkce pro automatické/plánované načítání dat z definovaného úložiště dat do mapovaného cíle,
- **editor vyhodnocení skóre** – umožňuje uživateli definovat atributy a pravidla pro výpočty,
- **nástroj pro výpočet dopadu** – nástroj pro hodnocení a prezentaci dopadu změněných jednotkových cen komodit na klienty,
- **autentizace a autorizace** – obecná komponenta pro autentizaci uživatele systému a autorizaci požadavků přihlášeného uživatele, autorizace požadavků je založena na přiřazení uživatelských rolí,
- **export do souboru ve formátu .xlsx** – obecná funkce pro export konfigurovaných nebo ad-hoc dat ve formátu souboru Excel s použitím nakonfigurovaných pravidel formátování. Implementaci této funkcionality je možné vidět v příloze 2.

4.7.2 Modul správy informací – IMS

Modul správy informací (dále pouze IMS) je primárně zodpovědný za synchronizaci podnikových dat určených k vytváření reportů a analýz klientů, dodavatelů, distributorů, elektronických aukcí a klientů. V IMS je implementována logika zpracování zpráv zaslaných z integračního serveru. IMS spravuje tato data:

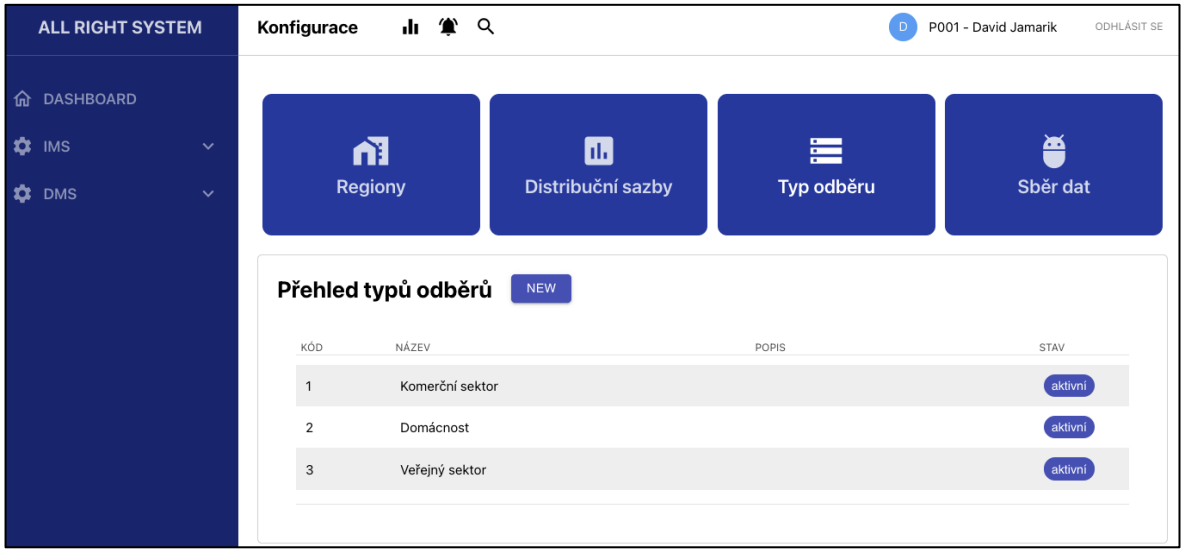
- detailní informace a ceníky dodavatelů a distributorů působících na českém trhu,
- souhrnná data o všech klientech, smlouvách a odběrných místech,
- souhrnná data o elektronických aukcích organizovaných společnosti.

Konfigurace číselníků

Tato část webové aplikace umožňuje uživatelům spravovat seznamy jednotlivých konfiguračních hodnot. U všech těchto hodnot aplikace zobrazuje pouze aktuální hodnoty. Uživatel může přidat do seznamu nový záznam. Hodnoty nejsou historizovány, takže nemají žádnou platnost (*valid_from* a *valid_to* atributy). Uživateli není umožněno upravovat ani mazat žádnou aktuální hodnotu. Na obrazovce jsou zobrazeny tyto seznamy konfiguračních hodnot:

- **regiony (db tabulka: *ims_region*)** – regiony reprezentují všechny kraje České republiky používané v aplikaci. Využívají se především jako atributy k definování území spadajícího pod daného distributora či zjištění regulovaných cen z ceníku daného distributora na základě adresy klienta,
- **sběr dat (db tabulka: *ims_data_collection*)** – sběr dat představuje interní číselník všech stavů, ve kterých se může daná smlouva nacházet,
- **distribuční sazba (db tabulka: *ims_business_rate*)** – distribuční sazba stanovuje, kolik klient zaplatí za spotřebu elektřiny nebo zemního plynu a za jakých podmínek. Ovlivňuje ji druh a počet používaných elektrospotřebičů. Hierarchicky jsou sazby rozdělovány na domácnosti a maloodběratele a dále pak podle využití elektřiny. Sazby pro domácnosti začínají prefixem D, zatímco sazby pro maloodběratele začínají prefixem C,
- **typ odběru (db tabulka: *ims_consumption_type*)** – typ odběru reprezentuje interní číselník, který rozřazuje odběrná místa podle toho, jak jsou využívána (komerční sektor, domácnost atd.).

Obrázek 4.9 - Obrazovka systémové konfigurace,



Zdroj: vlastní

Přehledy podnikových entit

Pro získání lepší orientace v podnikovém prostředí poskytuje webová aplikace souhrnný přehled dodavatelů, aukcí, distributorů, klientů a jejich atributů ve formě stránkovaného seznamu. Zobrazenou množinu záznamů v seznamu je možné dle potřeby filtrovat nebo vyexportovat do souboru aplikace Excel. Pokud je daná entita časově omezena, je umožněno zobrazit pouze aktuální záznamy. Jako příklad je na obrázku 4.10 znázorněna obrazovka přehledu aukcí. Nastavení tabulek pro účely mapování do uživatelského rozhraní je znázorněno v příloze 5.

Obrázek 4.10 - Část obrazovky – Přehled dodavatelů

Zdroj: vlastní

Detail klienta

Výběrem konkrétního záznamu z přehledu klientů je uživatel přesměrován na obrazovku detailu klienta. V této části aplikace jsou zobrazeny detailní informace o klientovi, seznam jeho smluvních závazků a seznam odběrných míst. Informace o smlouvách a odběrných místech jsou zobrazeny prostřednictvím stránkovaného seznamu v podobě tabulky. Nad jednotlivými tabulkami (smluv, odběrných míst) je možné filtrovat či zobrazit aktuální záznamy. V rámci této obrazovky je možné nad některými daty klienta provádět operace:

- změna stavu klienta,
- hromadná de/aktivace odběrných míst.

Každému klientovi náleží jeden ze tří stavů. Klient je v aktivním stavu, jestliže má alespoň jednu platnou smlouvu. Ve chvíli, kdy nemá žádnou platnou smlouvu, náleží mu neaktivní stav. Pokud odpovědný uživatel požaduje, aby byl daný klient ignorován v rámci cenových kalkulací v modulu DMS, může danému klientovi přiřadit stav *suspendován*. Manuálně lze editovat stav klienta pouze do stavu *suspendován* a naopak. Implementovanou logiku změny stavu klienta je možné nalézt v příloze 1.

Uživateli webové aplikace je také umožněno, pro potřeby cenové kalkulace, deaktivovat pouze odběrná místa klienta. Není tedy nutné suspendovat celého klienta, ale pouze výčet některých jeho odběrných míst. Vybraná odběrná místa jsou před samotnou deaktivací validována. Pokud pro dané odběrné místo existuje smlouva ve stavu *znovuzařazení*, nelze jej deaktivovat.

Obrázek 4.11 - Část obrazovky – detail klienta

Zdroj: vlastní

Detail dodavatele

Výběrem konkrétního záznamu z přehledu dodavatelů je uživatel přesměrován na detail daného dodavatele. Na této obrazovce jsou zobrazeny základní informace, seznam nabízených tarifů a výčet aukcí, které dodavatel vyhrál. Data ohledně aukcí a tarifů jsou zobrazena ve formě tabulky, které nelze filtrovat.

Na obrazovce je uveden i status dodavatele, který mu náleží. Webová aplikace spravuje tyto tři stavy:

- **aktivní** – dodavatel nabízí alespoň jeden aktuální tarif a zároveň má platnou licenci od Energetického regulačního úřadu (ERÚ),
- **neaktivní** – dodavatel aktuálně nenabízí žádné služby nebo nemá platnou licenci,
- **suspendován** – dodavatel není zařazen do výpočtu skóre v modulu DMS. Jedná se o jediný stav editovatelný uživatelem.

Obrázek 4.12 - Detail dodavatele

Zdroj: vlastní

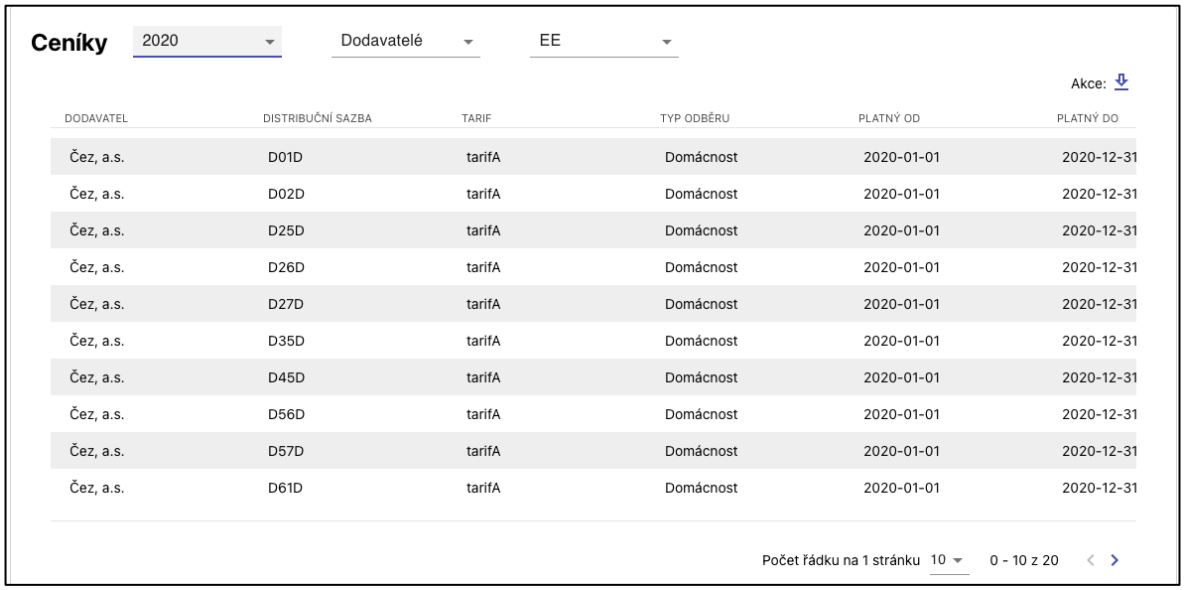
Přehledy ceníků

Jednou z klíčových funkcí IMS modulu je možnost zobrazení historických i aktuálních ceníků dodavatelů, pro každou z dostupných komodit, distributorů a elektronických aukcí. Data jsou v aplikaci zobrazena ve formě tabulky, ve které je možné filtrovat. Každý řádek zobrazené tabulky (viz obrázek 4.13) obsahuje informace o parametrech cen pro kombinaci aukce / dodavatel / distributor vs. distribuční sazba vs. komodita.

Pro zobrazení ceníků je potřeba vyplnit požadovanou entitu a komoditu. Není možné zobrazit ceníky dodavatelů a distributorů nebo elektrické energie a zemního plynu zároveň v jedné tabulce. Data lze také zobrazit v ohraničeném časovém období, například ceníky pouze pro rok 2018. Ve výchozím nastavení jsou zobrazeny ceníky aukcí elektrické energie pro aktuální rok.

Pokud jsou vyplněny všechny parametry výběru, je uživateli umožněno výslednou množinu exportovat do souboru ve formátu .xlsx.

Obrázek 4.13 - Přehled ceníků



zdroj: vlastní

Souhrnný přehled (reporting)

Dalším z dílčích požadavků společnosti bylo implementovat souhrnný přehled dat, která aplikace spravuje, a zobrazit je v adekvátní formě. K tomu, aby byl požadavek úspěšně realizován, byly použity některé techniky Business Intelligence – dimenzionálního modelování podle Ralpha Kimballa.

Na úvod bylo nutné správně definovat úroveň podrobnosti zachycovaných informací z hlediska analýzy vybraných procesů. V logickém datovém modelu aplikace lze nalézt několik tzv. dimenzí – shluků entit, které se mohou vztahovat ke konkrétnímu procesu:

- **správa klientů** – klienti, smlouvy a odběrná místa, ceník aukce,
- **řízení aukcí** – aukce, dodavatelé, ceníky.

Na základě definovaných dimenzí byly následně stanoveny tzv. tabulky faktů, které odpovídají sledovanému procesu. V tabulce 4.4 jsou zobrazeny sledované procesy, dimenze a jejich granularita.

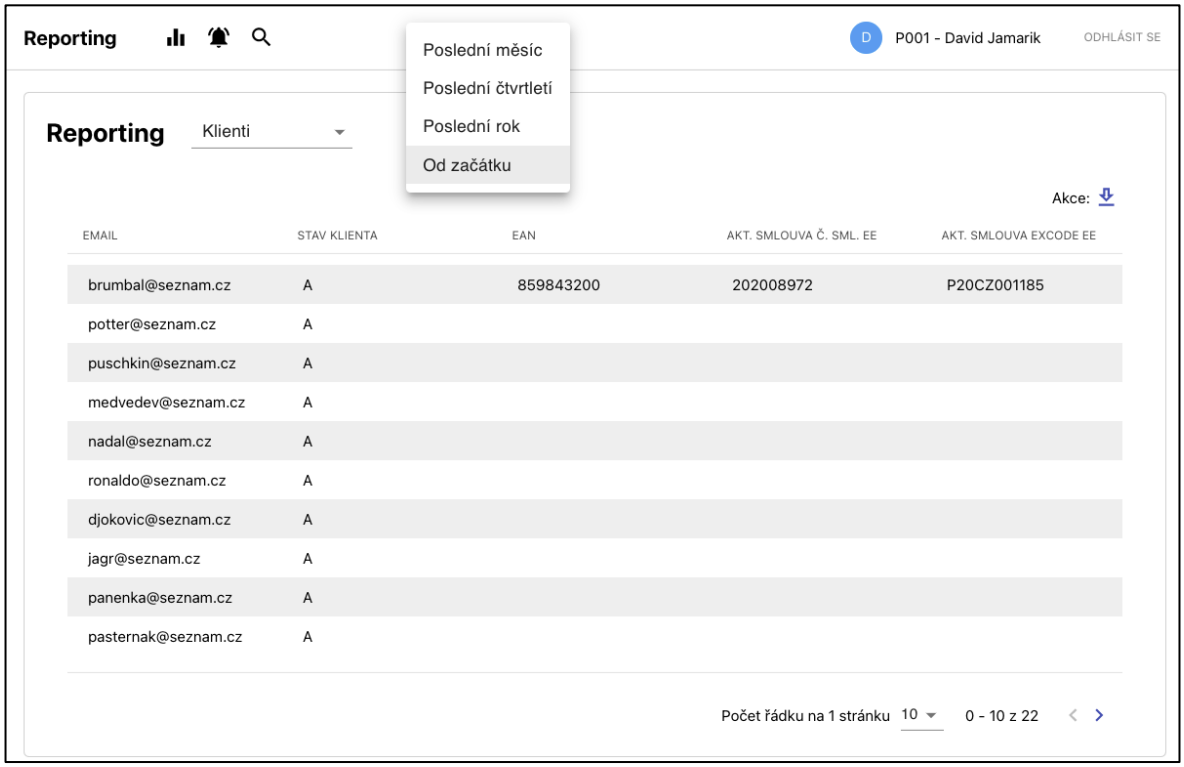
Tabulka 4.5 - Specifikace dimenzionálního modelu

Název procesu	Granularita	dimenze
Správa klientů	1 záznam = 1 odběrné místo	Smlouvy, klienti, odběrná místa,
Řízení aukcí	1 záznam = 1 aukce	Aukce, dodavatel, ceník aukce

Zdroj: vlastní

Dimenze byly implementovány jako SQL dotazy (ims_*report_client_view* a *ims_report_auction_view*) vycházející z určených shluků entit. Pro zajištění plynulého zobrazení byl při implementaci také optimalizován výkon dotazování. Data z těchto tabulek jsou uživatelům zobrazena formou stránkovaného seznamu. Nad seznamem je možné filtrovat a vybrat si zobrazení dat za konkrétní časový úsek.

Obrázek 4.14 - Souhrnný přehled entit



Zdroj: vlastní

4.7.3 Systém pro podporu rozhodování – DMS

Druhým funkčním celkem nové webové aplikace je modul DMS. Hlavním cílem implementace modulu DMS bylo vytvořit nástroj určený vedoucím podniku pro řízení obchodních a marketingových strategií. Mezi klíčové odpovědnosti funkčního modulu DMS lze zařadit výpočet skóre dodavatelů a kalkulaci jednotkové ceny pro definované skupiny klientů.

Jednotlivé operace modulu DMS jsou založeny na integrovaných datech v modulu IMS. Sady zdrojových dat a pravidel výpočtů konfiguruje odpovědný specialista pomocí grafického rozhraní. Grafické rozhraní umožňuje specialistovi:

- definovat a udržovat instance jednotlivých kalkulací,
- spouštět instance jednotlivých kalkulací a algoritmů,
- analyzovat, filtrovat setříděný seznam dodavatelů podle vypočteného skóre,
- analyzovat aktuální situaci na evropské burze jednotlivých komodit.

Zpracované výsledky tohoto modulu jsou interpretovány pomocí tabulkových reportů ve formátu .xlsx.

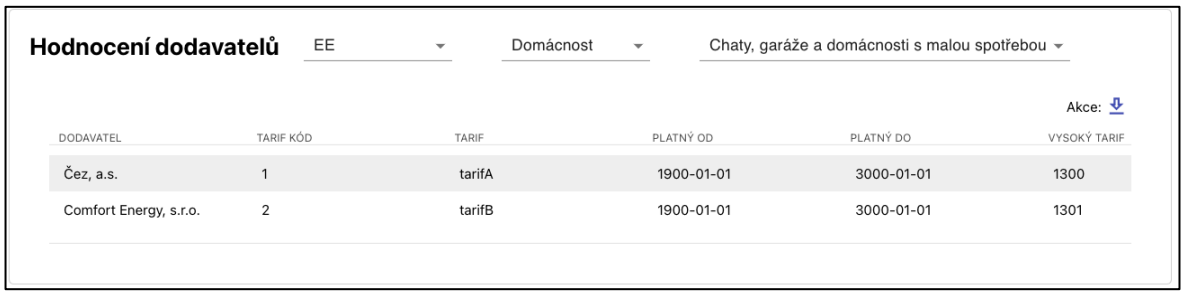
Přehled cenových nabídek dodavatelů

Liberalizovaný trh s elektrickou energií a zemním plynem dává každému klientovi možnost svobodně si vybrat dodavatele energie. Pro většinu klientů při výběru rozhoduje převážně cena, kterou jednotliví dodavatelé nabízejí. Aby společnost byla schopna dobře reagovat na aktuální vývoj trhu, bylo požadováno vytvořit přehledný report dodavatelů utříděný na základě jejich nejlepších cenových nabídek.

Data jsou uživateli zobrazena v aplikaci ve formě tabulky, která je schopna filtrovat výsledky hledání. Cenové nabídky je možné zobrazit pouze podle předem definovaných parametrů – komodita, distribuční sazba a typ odběru. Na základě parametrů vložených uživatelem jsou data vybrána pomocí dynamického SQL dotazu.

Vzhledem k tomu, že se jedná o velký objem dat, bylo nutné optimalizovat výkon dotazování. Z těchto důvodů byly přidány do konkrétních tabulek indexy a partitioning (fyzické rozdělení tabulek na několik částí).

Obrázek 4.15 - Přehled cenových nabídek dodavatelů



Zdroj: vlastní

Cenová kalkulace

Pří výběru vhodného dodavatele hraje u klientů největší rozhodovací element cena. Aby byla firma schopna efektivně řídit své obchodní a marketingové strategie, musí znát míru konkurenceschopnosti vysoutěžených cen oproti jiným dodavatelům na trhu. Jedním z hlavních požadavků webové aplikace tedy bylo implementovat proces výpočtu cen vybraného aukčního ceníku pro definovanou skupinu klientů. Hlavním cílem této části webové aplikace je získat pro každého klienta (resp. smlouvu) výslednou cenu za rok a zjistit úsporu nebo ztrátu na základě předešlé smlouvy.

Celý proces kalkulace je řízen pomocí grafického rozhraní webové aplikace a je složen ze čtyř základních kroků:

- vytvořit nebo smazat kalkulaci,
- nadefinovat parametry kalkulace,
- spustit proces výpočtu,
- analyzovat vypočtené hodnoty v tabulkové nebo grafické podobě.

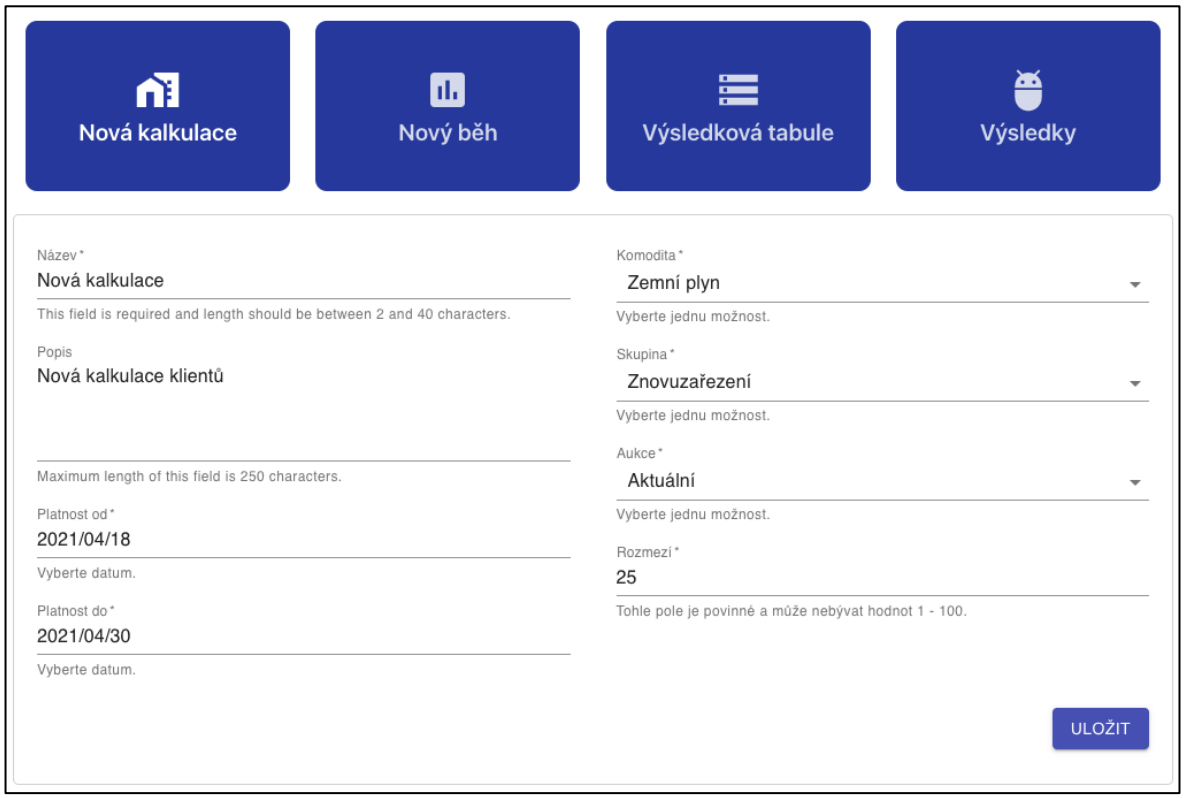
V rámci prvního kroku je potřeba vytvořit cenovou kalkulaci. Každá kalkulace představuje jeden záznam v databázi a musí mít jedinečný název. Počet kalkulací je neomezený. V případě potřeby je uživateli umožněno danou kalkulaci smazat. Pokud tak učiní, smažou se také všechny vypočtené záznamy pro tuto kalkulaci.

Po úspěšném vytvoření je uživatel přesměrován na konfigurační stránku kalkulace. Klíčovými parametry pro následnou kalkulaci jsou:

- **komodita** – výpočet nelze provézt pro obě komodity najednou. Pro každou komoditu je struktura ceníků jiná a zároveň platí i jiné podmínky při výpočtu,
- **skupina klientů** – jedná se o uživatelem definovanou skupinu klientů. Detailní popis implementace je uveden v podkapitole: *Konfigurace vlastních skupin klientů*. Skupinu klientů je také možné časově ohraničit,
- **aukce** – pro výpočet lze vybrat tři typy aukčních ceníků podle časového období – aktuální, předchozí a nadcházející. V případě, že databáze webové aplikace neobsahuje ceník pro nadcházející aukci, je tato možnost zakázána,
- **rozsah** – uživatelem definovaná hodnota, která určuje cenovou toleranci při výpočtu úspory nebo ztráty oproti předešlé smlouvě. V některých situacích je například potřeba vyrovnat růst, nebo naopak propad ceny na burze apod.

Definované parametry je potřeba před spuštěním kalkulace uložit. V případě, že tak uživatel neučiní, jsou úpravy ztraceny a není možné kalkulaci spustit.

Obrázek 4.16 - Nastavení cenové kalkulace



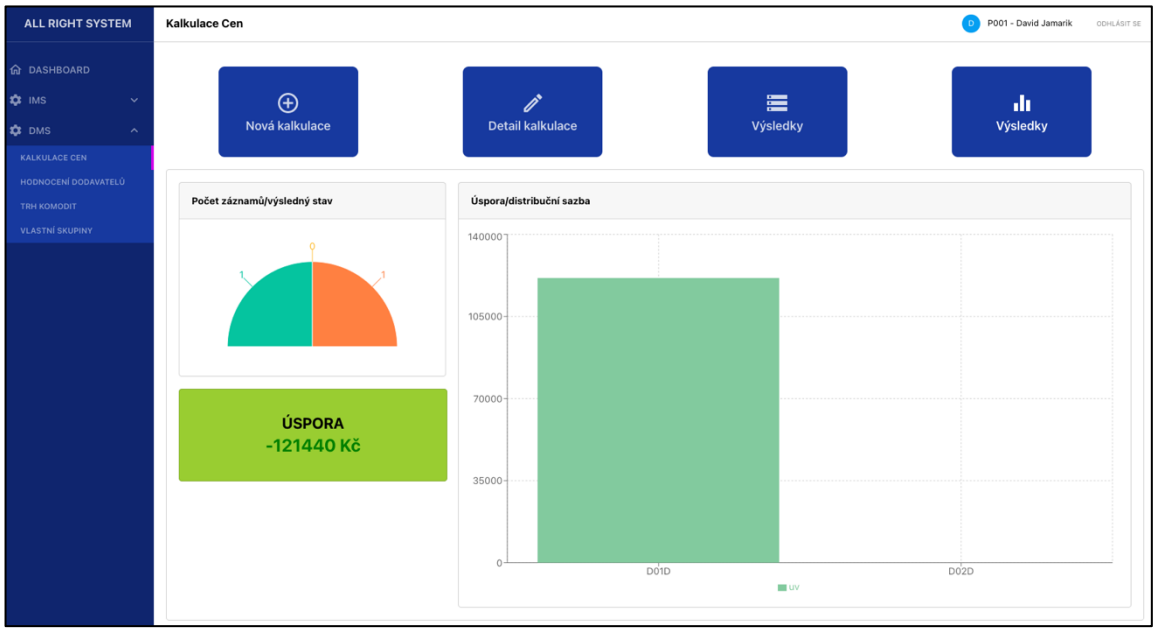
Zdroj: vlastní

Pokud byla cenová kalkulace validně uložena, uživateli je umožněno spustit výpočet cen. Výpočet probíhá jako asynchronní operace, což uživateli poskytuje možnost dále pracovat s aplikací bez omezení. Po doběhnutí výpočtu je uživatel o výsledku informován prostřednictvím notifikačního centra. Implementaci výpočtu cen v programovacím jazyce Java je možné vidět v příloze 4.

Po dokončení kalkulace jsou výsledky prezentovány ve formě tabulky a dashboardu. Zobrazené výsledky v tabulce je možné filtrovat, seřadit nebo exportovat do formátu .xlsx. Dashboard zobrazuje agregovaná data pomocí koláčového, výsečového a sloupcového grafu.

Data jsou také zasílána v rámci marketingové komunikace klientům, kteří přemýšlí o využití služeb společnosti. Díky tomu znají přesnou cenu za daný rok na základě své průměrné spotřeby. Zjistí tak, zda ušetří.

Obrázek 4.17 - Dashboard cenové kalkulace



Zdroj: vlastní

Konfigurace vlastních skupin klientů

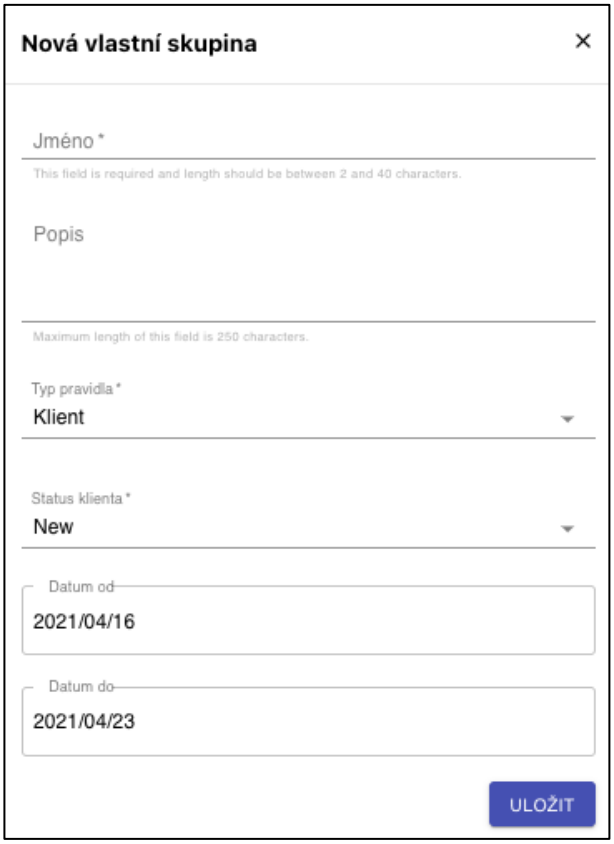
V rámci konfigurace cenové kalkulace je povinným atributem výběr předdefinované skupiny klientů, pro které bude kalkulace provedena. Uživatel má tedy možnost prostřednictvím webové aplikace vytvářet nové skupiny, případně je také smazat.

Při vytváření nové skupiny je uživateli umožněno definovat stav klienta a typ smlouvy. Systémově jsou nastaveny dvě základní skupiny, které nelze odstranit ani editovat. Jedná se o tyto skupiny:

- **noví klienti** – skupina klientů, kteří se rozhodli využít elektronické aukce společnosti pro výběr svého dodavatele,
- **klienti ve stavu „znovuzařazení“** – skupina klientů, kterým v blízké době končí aktuální smlouva. Takoví klienti se často rozhodují, zda nadále budou využívat služeb společnosti, především na základě vysoutěžené ceny. Vedení společnosti tak může předběžně předpovídat situaci na další období a vytvářet obchodní a marketingové strategie.

Na obrázku 4.18 je možné vidět obrazovku s přehledem všech vytvořených skupin.

Obrázek 4.18 - Modál pro vytvoření vlastní skupiny



Zdroj: vlastní

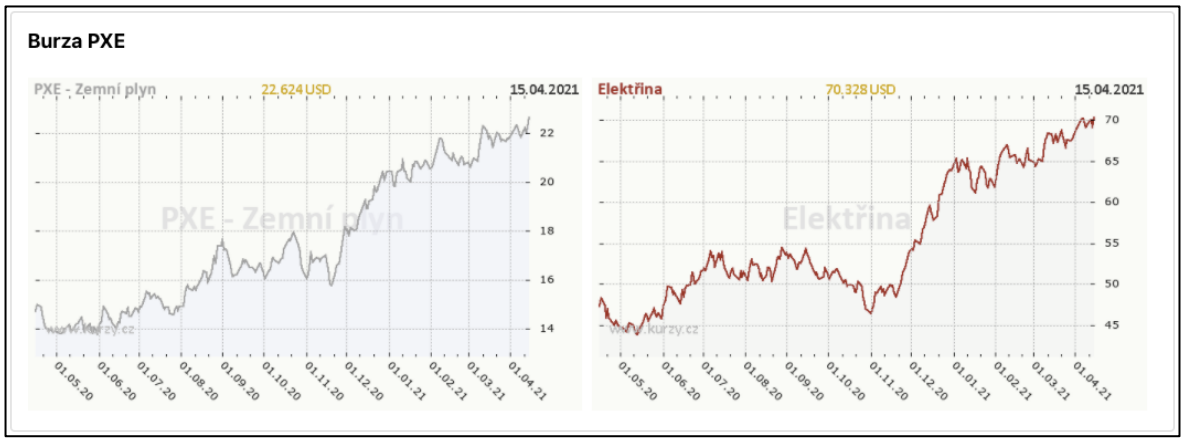
Přehled vývoje cen na burze

Výši neregulované cenové složky zemního plynu nebo elektřiny stanovují její dodavatelé. Ti do ní zahrnují náklady na pořízení komodity, marži obchodníků a další složky, které se mohou u jednotlivých dodavatelů lišit. Celkovou výši neregulované složky ovlivňuje také samotný trh, k němuž patří i energetická burza.

Burza organizuje trh s danou komoditou a umožňuje dodavatelům pohodlnější a rychlejší obchodování. V České republice působí burza společnosti PXE, a. s.

Situace na burze se může projevit také na účtech koncových klientů, a proto je potřeba vývoj ceny monitorovat a případně pružně reagovat. Na základě této potřeby bylo do webové aplikace implementováno grafické rozhraní, které umožňuje situaci sledovat. Data jsou zobrazena prostřednictvím spojnicového grafu z internetové stránky Kurzy.cz. Tento portál poskytuje vygenerovaný HTML kód, jenž byl do aplikace vložen.

Obrázek 4.19 - Obrazovka komoditního trhu



Zdroj: vlastní

4.8 Zabezpečení

Všechny aplikační servery používají standardní rozhraní Spring Security k zabezpečení externích požadavků pomocí informací uložených v kontextu zabezpečení. Kontext zabezpečení obsahuje ID uživatele a jeho interní heslo pro aplikaci. Požadavky na zabezpečení jsou v aplikaci napevno zakódovány a nelze je změnit bez nové kompilace zdrojového kódu.

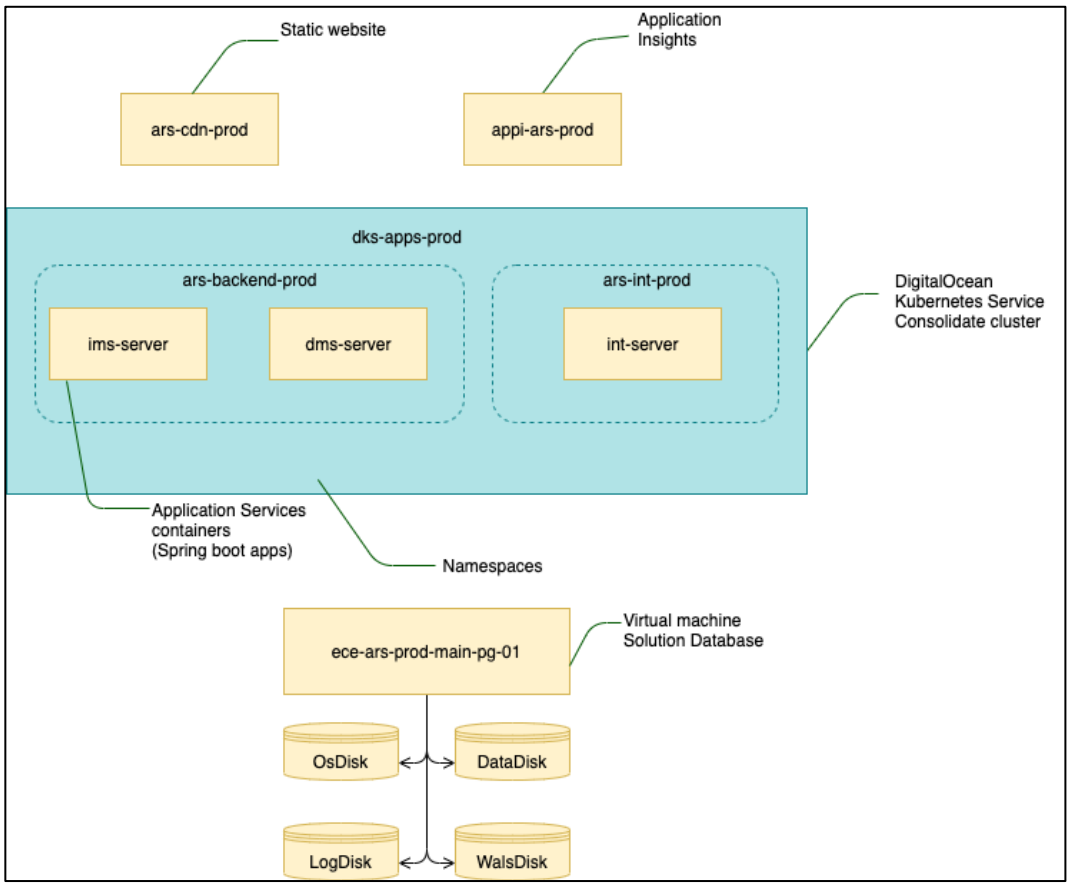
Ověřovací data jsou dekódována z JWT (JSON Web Token) tokenu, u kterého se očekává, že bude nalezeno záhlaví autorizace. Když je JWT dekódován, ověřuje se pouze formát. Neprovádí se žádná validace podpisu vypršení platnosti, aplikace očekává, že ověření proběhne pomocí API Gateway ještě před kontaktováním aplikačního serveru.

Data uživatelů jsou uložena v databázi včetně hesla a přihlašovacího jména. Heslo je zakódováno pomocí Base64. Při každém požadavku je heslo dekódováno a ověřováno, zda je platné a validní.

4.9 Nasazení webové aplikace

Webová aplikace je nasazena a běží na cloudové platformě společnosti DigitalOcean. Nasazení klíčového řešení do cloudových služeb je znázorněno na následujícím obrázku 4.20.

Obrázek 4.20 - Nasazení webové aplikace



Zdroj: vlastní

Grafické rozhraní webové aplikace je nasazeno jako statická webová stránka (ars-cdn-prod). K aplikaci je přistupováno přímo z webového prohlížeče zabezpečeného síťovým protokolem HTTPS. Uživatelé jsou ověřováni pomocí JWT tokenu a přístupových dat uložených v databázi webové aplikace. Grafické rozhraní komunikuje s aplikačními servery pomocí REST API. Ke všem službám REST API se přistupuje pomocí API Gateway, která zajišťuje autorizaci všech požadavků.

Základní funkční moduly webové aplikace (IMS a DMS), implementované jako Java Spring Boot aplikace, jsou nasazeny do cloudové platformy prostřednictvím kontejnerové orchestrace Docker. Každý funkční modul je integrován do samostatného kontejneru Docker a je uložen v Container registry v podobě Docker Image (ims-server, dms-server). Kontejnery funkčních modulů jsou následně nasazeny do Kubernetes klastru (dks-app-prod). Oba funkční moduly běží v jedné instanci (ars-backend-prod).

Integrační server (ars-int-prod) představuje most mezi aplikačními servery a Apache Kafka. Integrační server naslouchá tématům Kafky a odesílá všechny přijaté zprávy na příslušnou službu aplikačního serveru. Integrační server běží v samostatné instanci (ars-int-prod) nasazené do Kubernetes klastru (ars-backend-prod).

Databáze je nasazena na dedikovaný virtuální stroj (ece-ars-prod-main-pg-01). Virtuální počítač používá čtyři disky pro instalaci operačního systému a softwaru VM (OsDisk), logovací soubory (LogDisk), konfiguraci databáze a dat (DataDisk) a WAL soubory databáze PostgreSQL (WalsDisk).

Application Insights aplikace (appi-ars-prod) je nakonfigurovaná tak, aby shromažďovala metriky řešení a klíčové zalogované zprávy. Metriky řešení jsou shromažďovány pro backendové služby aplikace a jejich závislé operace. Hlavní logovací zprávy aplikace (varovné nebo vyšší závažnosti) se shromažďují jako „stopy“ (tzv. traces). Úplné podrobnosti logovacích zpráv se shromažďují v trvalých svazcích Kubernetes klastru.

5 Migrace dat a testování

První část poslední kapitoly je věnovaná migraci dat z interních datových zdrojů podniku do nově implementovaného systému. Druhá část se věnuje testování aplikace, jedné z velmi důležitých části vývoje webových aplikací.

5.1 Migrace dat

Implementovaná webová aplikace slouží jako nástroj pro podporu rozhodování vyššího managementu společnosti eCENTRE, a.s. Aby však bylo možné s aplikací v tomto směru efektivně pracovat, je třeba získat všechna potřebná data.

Pro práci s daty v reálném čase byly do aplikace implementovány některé integrační nástroje, například Apache Kafka Message broker. Do datového úložiště webové aplikace je však potřeba nahrát i historická data klientů a ceníků dodavatelů, distributorů nebo již uskutečněných aukcí. Z tohoto důvodu byl navržen a následně implementován proces migrace dat pomocí ETL (extrakce, transformace a nahrání dat). Procesy transformace a nahrávání dat byly zpracovány prostřednictvím nástroje Talend Open Studio for Data Integration.

5.1.1 Extrakce dat

Prvním krokem při migraci je extrakce dat. Ve zdrojové databázi informačního systému společnosti byly vytvořeny SQL pohledy, díky kterým se podařilo získat data v žádoucí formě. Podstatnou součást extrakce představovala také validace dat, jejímž cílem bylo určit, zda extrahovaná data nabývají správných či očekávaných hodnot. Celý proces extrakce dat je zpracován interními IT pracovníky.

5.1.2 Transformace a nahrání dat

Ve fázi transformace dochází k aplikování různých pravidel na extrahovaná data za účelem jejich přípravy pro nahrání do cílového datového zdroje. Při migraci vyvstává mnoho problémům spočívajícím především v odlišnosti mezi zdrojovým a cílovým datovým úložištěm.

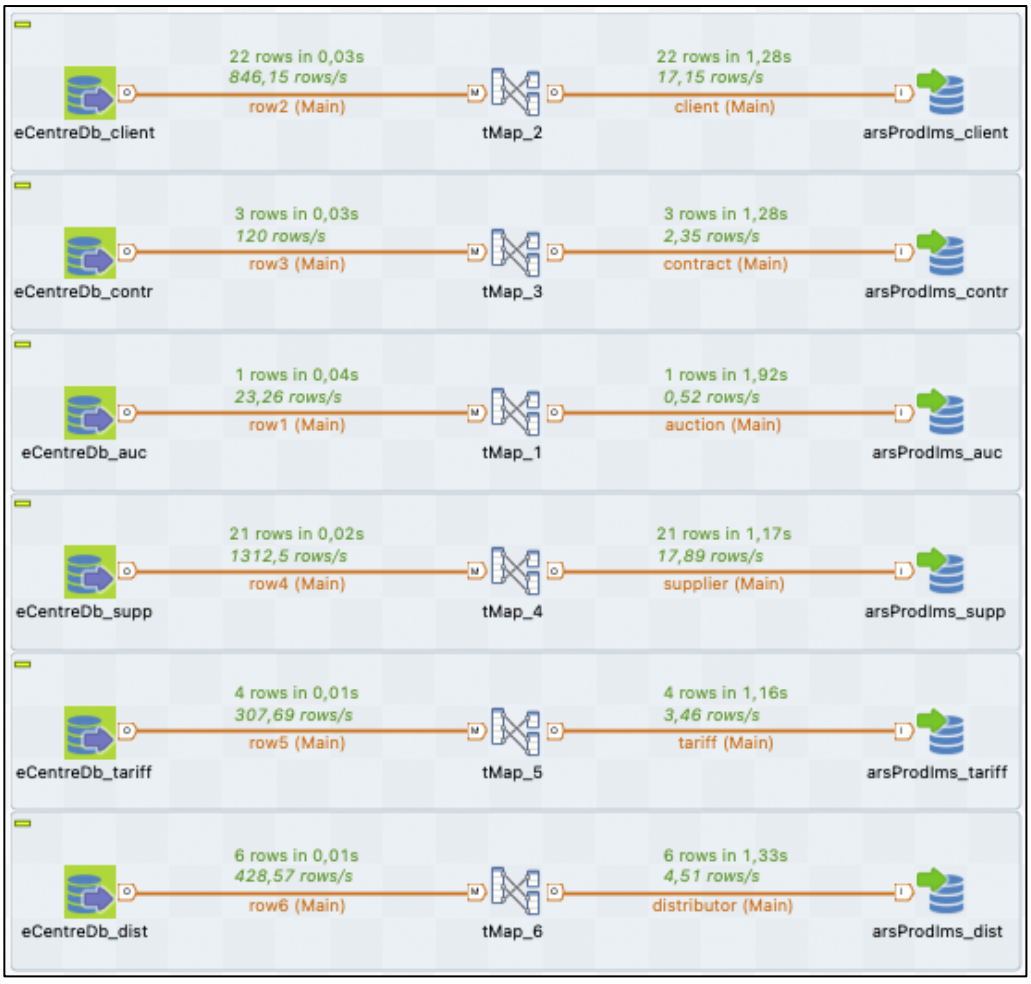
Mezi hlavní problémy, které je potřeba vyřešit, patří:

- **výběr pouze požadovaných sloupců pro nahrání**,
- **převod kódovaných hodnot** – ve zdrojovém systému je pro status některých entit používán datový typ *Boolean*, ale v cílovém systému jsou to textové hodnoty, například *Active*,
- **odlišnost datových typů** – v cílové databázi byl využit datový typ *Array*, který nahrazuje některé číselníkové tabulky ve zdrojové datové struktuře.

Všechny výše uvedené problémy se podařilo vyřešit pomocí nástroje Talend Open Studio for Data Integration. K samotnému mapování zdrojových dat do cílové struktury byla využita komponenta *tMap*.

Posledním krokem migrace je nahrání dat do cílové databáze. Na obrázku 5.1 je možné vidět grafické znázornění migrace dat.

Obrázek 5.1 - Schéma migrace dat



Zdroj: vlastní

5.2 Testování aplikace

Vývoj webové aplikace probíhal agilně. Tvorba aplikace byla rozdělena do několika sprintů. Před koncem každého sprintu, kdy bylo potřeba zákazníkovi dodat funkční část funkcionalit aplikace (tzv. demo) ke kontrole, proběhlo i testování. V rámci tohoto projektu bylo využito testování jednotek, integrační a uživatelské testování.

5.2.1 Testování jednotek a integrační testy

K otestování jednotlivých implementovaných funkcionalit (jednotek) byl do projektu přidán testovací rámec TestNG. Díky tomuto rámci je možné testovat správné chování jednotlivých metod a tříd, které obsahují obchodní logiku webové aplikace. Výhodou TestNG oproti známější knihovně JUnit je jednoduchost a pokrytí širší škály testovacích kategorií, například podpora integračních testů. Pro tento projekt byly implementovány jednotkové a integrační, především pro nejdůležitější funkcionality aplikace, například správa klientů, smluv, ceníků nebo kalkulace cen.

Na obrázku 5.2 lze vidět implementaci jednoduchého testu ověřujícího návratové stavové kódy http při aktualizaci klientů. Integrační server zpracuje zprávu z definovaného Kafka tématu a zašle požadavek na daný server pomocí REST API. Při zpracování zprávy může dojít k různým scénářům, na které je potřeba reagovat odlišně. Testovací scénář implementovaný metodou *testIntegrateClient()* testuje dva konkrétní scénáře. Prvním je vytvoření nového klienta a vracení stavového kódu 201 – Created. Při zaslání stejného požadavku, bez změny časového razítka, testovaná funkcionalita vrátí http status 204 – No Content. Samotná kontrola je provedena voláním metody *assertTrue/False* třídy *Assert.class*. Pro zjednodušení testu nebyl zahrnut integrační server a konkrétní požadavek je imitován voláním metody *beforeCreateClientRequest()*.

Obrázek 5.2 - Integrační test klienta

```
@TestExecutionListeners(MockitoTestExecutionListener.class)
@SpringBootTest
@Transactional
@AutoConfigureMockMvc
@Slf4j
@ActiveProfiles("test")
public class IntegrationServiceTest extends AbstractTransactionalTestNGSpringContextTests

    @Autowired
    private IntegrationService integrationService;

    private ClientRequestDTO testClientRequest;

    @BeforeMethod(alwaysRun = true)
    @Rollback()
    private void beforeCreateClientRequest(){...}

    @Test
    @Rollback()
    public void testIntegrateClient(){
        ResponseEntity<ApiResponseEntity<Void>> res =
                integrationService.integrateClient(testClientRequest);
        HttpStatus firstStatus = res.getStatusCode();

        testClientRequest.setHeaderTimestamp(LocalDateTime.now());
        res = integrationService.integrateClient(testClientRequest);
        HttpStatus secondStatus = res.getStatusCode();

        Assert.assertFalse(firstStatus.equals(secondStatus));
        Assert.assertTrue(firstStatus.equals(HttpStatus.CREATED));
        Assert.assertTrue(secondStatus.equals(HttpStatus.NO_CONTENT));
    }
}
```

Zdroj: vlastní

5.2.2 Uživatelské testování

Po skončení každého sprintu byla aplikace předána k testování vybraným uživatelům podniku. Každý z těchto uživatelů obdržel dokumentaci funkční specifikace daného sprintu a testovací scénáře, u kterých je potřeba otestovat, zda odpovídají zadání.

Testovací scénář obsahuje souhrn všech vstupních požadavků pro své vykonání. Jsou v něm zaznamenány jednotlivé kroky, které mají být při testování provedeny. Pro každý krok je uveden očekávaný výsledek. Uživatel, který testovací scénář provádí, pak postupuje po jednotlivých krocích. U každého postupu vyhodnocuje, zda proběhl podle očekávání. Testovací skript se označí jako úspěšně otestovaný v případě, že každý jeho jednotlivý krok proběhl v pořádku a výsledek odpovídá očekávání. Jako příklad je uveden testovací scénář přihlášení do systému.

Závěr

Hlavním cílem této práce bylo navrhnout a implementovat webovou aplikaci sloužící jako nástroj pro podporu rozhodování vyššího managementu společnosti eCENTRE, a.s. Webová aplikace umožňuje vedoucím pracovníkům společnosti analyzovat různé složky podnikového prostředí.

Stěžejní části aplikace je cenová kalkulace vybraného aukčního celku pro definovanou skupinu klientů. Hlavním cílem bylo získat pro každého klienta výslednou cenu za rok a zjistit úsporu nebo ztrátu na základě předešlé smlouvy. Výstup cenové kalkulace následně slouží jako podklad pro řízení marketingových a obchodních strategií.

Samotnému návrhu a implementaci předcházela analýza současného stavu společnosti společně s definováním jednotlivých cílů a požadavků. Na základě získaných poznatků byl proveden správný výběr technologií, návrh funkční specifikace a koncept uživatelského rozhraní aplikace. Pro prezentaci navrženého řešení byly použity některé UML diagramy, jako například UseCase diagram nebo komponent diagram.

Pro implementaci byly využity moderní přístupy a technologie. Webová aplikace se skládá ze dvou základních funkčních modulů (Systém pro správu informací a Systém pro podporu rozhodování), které odpovídají vzoru mikroslužeb v souladu s principy třívrstvé architektury.

Datová vrstva byla implementována pomocí objektově-relační databáze PostgreSQL, která je ideální pro náročné analytické procesy a skladování velkého objemu dat. Jelikož aplikace bude využívána jako analytický nástroj, který je oddělený od transakčních systémů podniku, bylo možné využít jiný databázový systém, než který je ve společnosti doposud používán.

Prezentační vrstva byla implementována pomocí frameworku JavaScript React.js. Jelikož webová aplikace obsahuje velké množství tabulek, bylo nutné zanedbat některé responzivní prvky. Některé informace jsou prezentovány také ve formě grafů, například koláčový graf nebo histogram.

Aplikační vrstva obou modulů provádí operace a zpracování dat mezi vstupně-výstupními požadavky pomocí webových služeb založených na REST technologii. Veškeré operace, webové služby a logiky byly implementovány pomocí frameworku Java Spring Boot.

Webová aplikace byla úspěšně nasazena na produkční prostředí a splňuje všechny definované cíle a požadavky. Jedná se však o první verzi samotné aplikace a pravděpodobně bude nutné některé části aplikace opravit nebo přizpůsobit na základě zpětné vazby od zaměstnanců společnosti.

V současné chvíli se stále pracuje na zprovoznění všech implementovaných integračních služeb, u kterých je potřeba větší míra spolupráce s interními IT pracovníky. Jedná se o závěrečnou fázi implementace, kdy je nutné zaručit plynulý chod aplikace v rámci podnikového prostředí.

Po kompletním zprovoznění integračních služeb bude následovat proces finální migrace dat do datového uložiště aplikace pomocí nástroje Talend Open Studio for Data Integration. Proces migrace je nicméně otestován a připraven ke spuštění.

Administraci webové aplikace budou provádět interní IT pracovníci společnosti, kteří budou řádně zaškoleni. Požadavky týkající se implementovaných funkcionalit nebo datové struktury bude mít na starosti nadále autor této práce.

Seznam použité literatury

Odborná kniha

BANKS, Alex and Eve PORCELLO. *Learning React: Modern Patterns for Developing React Apps*. London: O`Reily, 2020. 324 p. ISBN 978-1492051721.

BENTLEY, Drew. *Business Intelligence and Analytics*. 2nd edition. Great Britain: Larsen and Keller Education, 2017. 316 p. ISBN 978-1635490565.

BLUMENTHAL, Stephen. *JavaScript For Beginners: Learn JavaScript Programming with Ease in HALF THE TIME: Everything about the Language, Coding, Programming and Web Pages You need to know.* California*:* CreateSpace Independent Publishing Platform, 2017. 122 p. ISBN 978-15-487-9938-0.

FERRARI, Luca and Enrico PIROZZI. *Learn PostgreSQL: Build and manage highperformance database solutions using PostgreSQL 12 and 13*. Birmingham: Packt Publishing, 2020. 650 p. ISBN 978-1838985288.

FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley Professional, 2003. 560 p. ISBN 978-0321127426.

FORD, Jerry Lee. *Ajax Programming for the Absolute Beginner*. Boston: Course Technology, 2009. 320 p. ISBN 978-15-986-3564-5.

GARG, Nishant. *Learning Apache Kafka.* 2nd ed. Birmingham: Packt Publishing Ltd., 2015. 112 p. ISBN 978-1784393090.

HARRISON, Guy. *Next Generation Databases: NoSQLand Big Data*. New York: Apress, 2015. 256 p. ISBN 978-1484213308.

HOBERMAN, S., D. BURBANK and Ch. BRADLEY. *Data Modeling for the Business: A Handbook for Aligning the Business with IT Using High-Level Data Models*. New Jersey: Technics Publication, 2009. 230 p. ISBN 978-0977140077.

LOY, M., P. NIEMEYER and D. LEUCK. *Learning Java: An Introduction to Real-World Programming with Java*. 5th ed. Sebastopol: O'Reilly Media, 2020. 518 p. ISBN 978-1492056270.

NADAREISHVILI, I. et al. *Microservice Architecture: Aligning Principles, Practices, and Culture*. Sebastopol: O`Reily, 2016. 206 p. ISBN 978-1491959794.

NEWMAN, Sam. *Building microservices: Designing Fine-Grained Systems*. 1st ed. Sebastopol: O´Reilly, 2015. 280 p. ISBN 978–1491950357.

MORRIS, Johny. *Practical Data Migration*. 2nd ed. Plymouth: BCS, The Chartered Institute for IT, 2012. 266 p. ISBN 978-1906124847.

SAUTER, L. Vicki. *Decision Support Systems for Business Intelligence*. 2nd ed. New Jersey: John Wiley, 2010. 480 p. ISBN 978-0470433744.

SONI, R. K., A. GANESHAN a R. RV. *Spring: Developing Java Applications for the Enterprise.* Birmingham: Packt Publishing, 2017. 1023 p. ISBN 978-1787127555.

WALLS, Craig. *Spring in action*. 4th ed. NY: Manning Publication, 2015. 624 p. ISBN 978-1617291203.

ZAKAS, Nicholas C. *JavaScript pro webové vývojáře: programujeme profesionálně.* Brno: Computer Press, 2009. 823 p. ISBN 978-80-251-2509-0.

Elektronické dokumenty a ostatní

MARTINŮ, Jiří a Petr ČERMÁK. *Metodiky vývoje software* [online]. Olomouc: Moravská vysoká škola Olomouc, 2018 [cit. 2020-04-30]. Dostupné z: https://1url.cz/nzqkD

Seznam zkratek

ACID	Atomicity, Consistency, Isolation, Durability
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
B2B	Business-to-Business
B2C	Business-to-Consumer
BI	Business Intelligence
CDN	Content Delivery Network
CLI	Command Line Interface
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DBMS	Database Management System
DEVOPS	Development & Operations
DI	Dependency Injection
DMS	Systém pro podporu rozhodování (modul webové aplikace)
DOM	Document Object Model
DSS	Decision Support System
ER	Entity Relationship
ERÚ	Energetický regulační úřad
ETL	Extract, Transform, Load
FDM	Fyzický datový mode
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identification
IMS	Systém řízení informací (modul webové aplikace)
IOC	Inversion of Control
IS	Information System
IT	Information Technology
JDBC	Java Database Connectivity
JRE	Java Runtime Environment
JSON	JavaScript Object Notation

JSX	JavaScript XML
JVM	Java Virtual Machine
JWT	JSON Web Token
KDM	Konceptuální datový model
LDM	Logický datový model
MBMS	Model Base Management System
MOLAP	Multidimensional Online Analytical Processing
MSSQL	Microsoft SQL Server
MVC	Model-View-Controller
NOSQL	Nonrelational Structured Query Language
OLAP	Online Analytical Processing
ORM	Object Relational Mapping
OS	Operation System
PC	Personal Computer
PXE	Power Exchange Central Europe
RDF	Resource Description Framework Schema
REST	Representational State Transfer
ROLAP	Relational Online Analytical Processing
RTM	Requirements Tracebility Matrix
SDM	Sémantický datový model
SQL	Structured Query Language
SŘBD	Systém řízení báze dat
TB	Terabyte
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
REST	Representational State Transfer
VM	Virtual Machine
VŠB	Vysoká škola báňská
WAL	Write-Ahead Log
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Prohlašuji, že

- jsem byl(a) seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou (bakalářskou) práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne: 23.4.2021

.....

Bc. David Jamárik

Seznam příloh

Příloha 1: Zdrojový kód: Změna stavu klienta

```
public ApiResponseEntity<ClientDetailDTO> clientChangeStatus(Integer clientId, Status clientStatus){
    Client client = clientDAO.findById(clientId).orElse(null);
    if(client == null) {
        return ApiResponseUtils.createErrorResponse("Klient s kódem " + clientId + " neexistuje");
    }

    if((Status.A.equals(client.getStatus()) && Status.N.equals(clientStatus)) ||
            (Status.N.equals(client.getStatus()) && Status.A.equals(clientStatus))){
        return ApiResponseUtils.createErrorResponse("Status nelze změnit");
    }

    if(Status.S.equals(clientStatus) && contractDAO.isActiveContractByClientId(client.getId(), LocalDate.now())){
        return ApiResponseUtils.createErrorResponse("Klient má aktivní smlouvu");
    }

    client.setStatus(clientStatus);
    client = clientDAO.save(client);
    ClientDetailDTO clientDetailDTO = mapper.toClientDetailDTO(client);
    return ApiResponseUtils.createOkResponse(clientDetailDTO);
}
```

Příloha 2: Zdrojový kód – export dat ve formátu .xlsx

```
public void exportDistributor(OutputStream out, Commodity commodity){
    List<DistributorDTO> distributorList = distributorDAO.findByCommodity(commodity).stream().map(mapper::toDTO).collect(Collectors.toList());

    try (XSSFWorkbook wb = new XSSFWorkbook()) {
        XSSFSheet sheet = wb.createSheet ("distributor");
        CreationHelper createHelper = wb.getCreationHelper();
        XSSFCellStyle dateCellStyle = wb.createCellStyle();
        dateCellStyle.setDataFormat(createHelper.createDataFormat().getFormat("d/m/yyyy"));

        XSSFRow row = sheet.createRow(0);
        int header = 0;
        for(String h : Arrays.asList(HEADERS)){
            row.createCell(header).setCellValue(h);
            header++;
        }

        if(distributorList == null || distributorList.isEmpty()){
            wb.write(out);
            return;
        }

        int rows = 1;
        for (DistributorDTO d : distributorList){
            writeRow(d, sheet.createRow(rows), dateCellStyle, Arrays.asList(HEADERS));
            rows++;
        }

        wb.write(out);
    } catch (IOException e) {
        log.error("Exporting failed", e);
    }
}

private void writeRow(DistributorDTO distributorDTO, XSSFRow row, XSSFCellStyle dateCellStyle, Collection<String> property){
    AtomicInteger index = new AtomicInteger();
    property.forEach(prop -> {
        Object propertyValue;
        BeanWrapperImpl bean = new BeanWrapperImpl(distributorDTO);
        try {
            propertyValue = bean.getPropertyValue(prop);
        } catch (Exception e){
            propertyValue = null;
        }

        if (propertyValue == null) {
            row.createCell(index.getAndIncrement());
        } else if (propertyValue instanceof LocalDate) {
            XSSFCell dateCell =  row.createCell(index.getAndIncrement());
            dateCell.setCellValue(((LocalDate) propertyValue).format (DateTimeFormatter.ofPattern("dd/MM/yyyy")));
            dateCell.setCellStyle(dateCellStyle);
        }else if (propertyValue instanceof BigDecimal) {
            XSSFCell numberCell =  row.createCell(index.getAndIncrement());
            numberCell.setCellValue(((BigDecimal) propertyValue).doubleValue());
        } else if (propertyValue instanceof String[]) {
            row.createCell(index.getAndIncrement()).setCellValue(String.join(",", (String[]) propertyValue));
        } else {
            row.createCell(index.getAndIncrement ()).setCellValue(propertyValue.toString());
        }
    });
}
```

Příloha 3: Zdrojový kód – integrace dat

```
public ResponseEntity<ApiResponseEntity<Void>> integrateClient(ClientRequestDTO request){
    log.info("Integrace klienta, request: {}", request);
    HttpStatus status;
    ApiResponseEntity<Void> body = ApiResponseUtils.createOkResponse(null);

    Client existClient = clientDAO.findByCode(request.getBody().getCode()).orElse(null);
    if(existClient == null){
        status = HttpStatus.CREATED;

        Client client = new Client();
        IntegrationMapper.mapClientCreate(request.getBody(), client);

        if(contractDAO.isActiveContractByClientCode(request.getBody().getCode(), LocalDate.now())){
            client.setValidContractStatus(ValidContractStatus.ACTIVE);
        } else {
            client.setValidContractStatus(ValidContractStatus.NOT_ACTIVE);
        }
        clientDAO.saveAndFlush(client);
    } else {
        if(existClient.getLastIntegrationTs().compareTo(request.getHeaderTimestamp()) < 0){
            status = HttpStatus.OK;
            IntegrationMapper.mapClientUpdate(request.getBody(), existClient);

            if(contractDAO.isActiveContractByClientCode(request.getBody().getCode(), LocalDate.now())){
                existClient.setValidContractStatus(ValidContractStatus.ACTIVE);
            } else {
                existClient.setValidContractStatus(ValidContractStatus.NOT_ACTIVE);
            }
            clientDAO.saveAndFlush(existClient);
        } else {
            log.info("Data klienta jsou starsi nez v db.");
            status = HttpStatus.NO_CONTENT;
        }
    }
    log.info("Integrace klienta dokoncena, status: {}, request: {}", status, request);
    return ResponseEntity.status(status).body(body);
}
```

Příloha 4: Zdrojový kód – proces cenové kalkulace

```
@Service
@Transactional
@Slf4j
public class PriceCalculationAsyncService {

    @Autowired
    private PriceCalculationDAO priceCalculationDAO;

    @Autowired
    private PriceCalculationJdbcDAO priceCalculationJdbcDAO;

    @Autowired
    private PriceCalculationRowDAO priceCalculationRowDAO;

    @Autowired
    private PriceCalculationStatusService priceCalculationStatusService;

    @Async
    public CompletableFuture<ApiResponseEntity<Void>> priceCalculation(Integer priceCalculationId, List<PriceCalculationRow> oldRows,
                                                                       Integer maxId, Integer userId){
        CompletableFuture<ApiResponseEntity<Void>> completableFuture = new CompletableFuture<>();
        ApiResponseEntity<Void> response = new ApiResponseEntity<>();

        PriceCalculation priceCalculation = priceCalculationDAO.findById(priceCalculationId)
                .orElseThrow(ExceptionUtils.supplyPriceCalculationNotFoundException(priceCalculationId));

        try{
            log.info("zahajeni cenove kalkulace - ({})", priceCalculationId);
            priceCalculationStatusService.setRunStatus(priceCalculationId);
            PriceCalculationRunInfoDTO infoDTO = priceCalculationJdbcDAO.selectInsertCalculationRows(priceCalculationId, userId);
            if(infoDTO.isSuccess()){
                if(Boolean.FALSE.equals(priceCalculationStatusService.getPriceCalculationStatus(priceCalculationId))){
                    priceCalculationJdbcDAO.deleteValues(priceCalculationId, maxId);
                    log.info("Cenova kalkulace ({}) byla pozastavena", priceCalculationId);
                    completableFuture.complete(response);
                } else {
                    priceCalculationStatusService.finishPriceCalculationStatus(priceCalculationId, LocalDateTime.now(), infoDTO.getClientCount());
                    priceCalculationRowDAO.deleteAll(oldRows);

                    log.info("Cenova kalkulace ({}) dokoncena", priceCalculationId);
                }
            }
        } catch (Exception e){
            log.error("Cenova kalkulace ({}) zhavarovala", priceCalculationId, e);
            completableFuture.completeExceptionally(e);

            priceCalculation.setCalculationCheckRunning(false);
            priceCalculationDAO.saveAndFlush(priceCalculation);
        }

        return completableFuture;
    }
}
```

Příloha 5: Nastavení tabulek v React.js

```
export const auctionTable = () => {
    return  {
        columnSetting: [
            {name: "code", showName: "kód"},
            {name: "name", showName: "název"},
            {name: "validFrom", showName: "platné od"},
            {name: "validTo", showName: "platné do"},
        ],
            actions: [
            {icon: <AccountBoxOutlined color={"primary"} />, click: "SPECIAL", tooltipText: "Přejít na přehled"}
        ],
        disableFooter: true,
        disableDetail: true,
        disableExport: true
    }
}

export const tariffTable = () => {
    return {
        columnSetting: [
            {name: "code", showName: "kód", width: "100px"},
            {name: "name", showName: "název"},
            {name: "description", showName: "popis"},
            {name: "validFrom", showName: "platný od"},
            {name: "validTo", showName: "platný do"},
            {name: "tariffType", showName: "typ tarifu"},
            {name: "commodity", showName: "komodita"},
            {name: "isActive", showName: "aktivní"},
        ],
        disableFooter: true,
        disableDetail: true,
        disableExport: true
    }
}

export const confTables = () => {
    return {
        columnSetting: [
            {name: "code", showName: "kód", width: "100px"},
            {name: "name", showName: "název", width: "calc(100% - 500px)"},
            {name: "description", showName: "popis", width: "300px"},
            {name: "status", showName: "stav", width: "100px"},
        ],
        disableFooter: true,
        disableDetail: true,
        disableExport: true
    }
}
```