Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Porovnání možností databází SQL a NoSQL pro využití ve webových aplikacích

Diplomová práce

Autor: Michal Kořínek
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Petra Poulová, Ph.D.

Hradec Králové 04/2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

Hradci Králové dne 20.4.2023 Michal Kořínek

Poděkování:

Děkuji vedoucí diplomové práce doc. RNDr. Petře Poulové, Ph.D. za metodické vedení práce, vstřícnost a za cenné rady při psaní práce. Dále bych chtěl poděkovat své rodině a přátelům za veškerou podporu, kterou mi poskytli.

Anotace

Práce se zabývá porovnáním výhod a nevýhod SQL a NoSQL databází pro využití ve webových aplikacích. Hlavními kritérii pro porovnání bude především návrh a implementace obou databázových řešení v konkrétní webové aplikaci. Praktická část této práce je zaměřena na vytvoření aplikace, která bude plnit úlohu jednoduchého řízeného skladu a následného porovnání výsledků obou řešení. Aplikace by měla obsahovat výpis detailu skladových pozic, možnost manipulace se zbožím pomocí ID a generování objednávek. Na konci práce je popsáno možné vylepšení formou android aplikace, pro snadnou manipulaci skladových pozic pomocí čtečky čárových kódů.

Annotation

Title: Comparison of SQL and NoSQL database capabilities for use in web applications

The thesis deals with the comparison of the advantages and disadvantages of SQL and NoSQL databases for use in web applications. The main criteria for comparison will be the design and implementation of both database solutions in a specific web application. This practical work is focused on the creation of a web application that will fulfill the role of a managed warehouse. The application should contain a detailed list of stock positions, the possibility of handling goods using IDs and generating orders. At the end of the work, a possible update in the form of an Android application is described, for easy manipulation of warehouse positions using a barcode reader.

Obsah

1 Úvod

Databáze je nejdůležitějším prvkem aplikací pro ukládání dat. Jedná se o organizovaný soubor strukturovaných údajů neboli dat uložených elektronicky v počítačovém systému. Databáze je řízena systémem řízení báze dat (DBMS). Existuje mnoho různých typů DBMS. Výběr databáze pro konkrétní organizaci závisí na tom, jak organizace zamýšlí data využít [1]. Tato práce bude zaměřena na relační a nerelační typy databází. Obě tyto varianty mají své výhody i nevýhody, proto budou dopodrobna porovnány. Pro porovnání bude vytvořený praktický příklad webové aplikace, která bude plnit úkol řízeného skladu v malém podniku. Pro SQL byl zvolen databázový systém MySQL a NoSQL varianta databáze byla vyvíjena za pomocí MongoDB. Základ webové aplikace byl naprogramován v jazyce NodeJS, díky kterému bylo možné propojit databázi s aplikací.

2 Cíl práce

Cílem diplomové práce je porovnání výhod a nevýhod databází SQL a NoSQL pro využití v jednoduchých webových aplikacích. Důležitými kritérii pro porovnání budou především jednoduchost implementace, způsob ukládaní dat, jejich použitelnost a náročnost na hardwarové prostředky. V práci bude porovnána vhodnost využití dané databáze v konkrétní webové aplikaci, týkající se řízeného skladu.

3 Databáze

Hlavním úkolem databáze je uchovávat data a učinit je použitelnými dle aktuální potřeby aplikace připojené k databázi. Databáze je obvykle určena pro dlouhodobou perzistenci dat. Účel databáze lze snadno popsat pomocí základních operací tzv. CRUD operace. Zkratka CRUD se skládá ze slov create, read, update a delete. Tato slova v překladu znamenají vytvořit, přečíst, aktualizovat a smazat. [2]

V závislosti na případu použití je možné si vybrat z velkého množství typu databází. Nejpoužívanějšími typy jsou relační (SQL), nerelační (NoSQL), objektové, dokumentové, cloudové a mnoho dalších. Databáze spolu s DBMS tvoří celek, kterému se říká databázový systém.

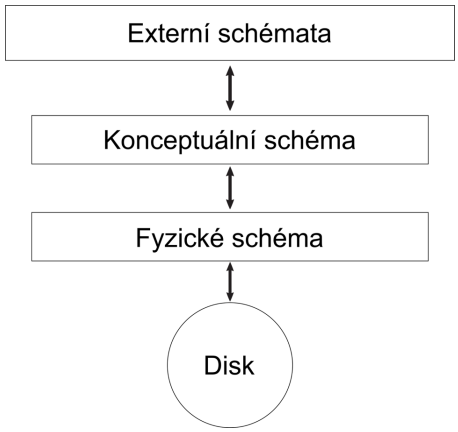
3.1 DBMS – Database Management System

Database Management System neboli systém řízení báze dat má za úkol komunikovat s uživateli, aplikací a databází. Uživatelé systému mají možnost provádět několik druhů operací, buď pro manipulaci s daty, nebo pro správu samotné struktury databáze. DBMS jsou kategorizovány podle jejich datových struktur nebo typů. [3]

Použitím DBMS ke správě dat má mnoho výhod a jednou z nich je datová nezávislost. Aplikační programy by v ideálním případě neměly být vystaveny detailům reprezentace dat a jejich ukládání. DBMS poskytuje abstraktní pohled na data, který detaily skrývá. Další výhodou je efektivní přístup k datům, jejich integrita a zabezpečení. Pro efektivitu se využívá řada sofistikovaných technik k ukládání a získávání dat. Tato funkce je obzvláště důležitá, pokud jsou data uložena na externích úložných zařízeních. Pokud je k datům přistupováno přes DBMS, lze využít omezení integrity. Příkladem je připsání platu zaměstnanci. Před vložením dat do tabulky se zkontroluje, zda se nepřekročil budget oddělení, pokud se tak stane, dojde ke kolizi s daným integritním omezením a do tabulky se plat nepropíše. Do dalších výhod můžeme zařadit administraci dat, zotavení při chybě, při kterém je databáze vrácena do konzistentního a použitelného stavu. [4]

3.1.1 Abstrakce v DBMS

Data v DBMS jsou popsána na třech úrovních abstrakce (obrázek č. 1). Jedná se o schéma externí, konceptuální a fyzické, ke kterým se přistupuje pomocí DDL a DML. [4]



Obrázek 1: Úrovně abstrakce v DMBS [autor dle [5]]

Externí schémata umožňují přizpůsobení a autorizaci přístupu k datům na úrovni jednotlivých uživatelů nebo jejich skupin. Každé externí schéma se skládá z kolekce jednoho nebo více pohledů a vztahů z konceptuálního schématu. Pohled je z konceptuálního hlediska vztah, jehož záznamy nejsou uloženy v databázi, protože se počítají pomocí definice pohledu. Návrh externího schématu se řídí požadavky koncového uživatele. Uživatel může s pohledem zacházet stejně jako s klasickou vazbou a klást dotazy týkající se záznamů v pohledu, i když záznamy v pohledu nejsou uloženy explicitně. [4]

Návrh konceptuálního schématu je proces generování popisu obsahu databáze v termínech na vysoké úrovni, které jsou přirozené pro uživatele databáze. Proces bere jako vstupní požadavky na informace pro aplikace, které budou databázi používat a tvoří schéma vyjádřené v notaci koncepčního modelování, jako je datový model Extended Entity-Relationship (EER) nebo diagramy tříd UML. Při navrhování koncepčního schématu zahrnují přeměnu neformálních informačních požadavků na kognitivní model, který jednoznačně popisuje obsah budoucí databáze a vhodné použití jazyka pro modelování dat pro generování z kognitivního modelu konceptuálního schématu, které jej co nejpřesněji odráží.

Existuje velké množství komerčních nástrojů pro návrh konceptuálních schémat založených na modelu EER nebo diagramech tříd UML. Ty obsahují nástroje pro kreslení, dokumentaci a rozvržení koncepčních schémat, dokonce i automatické generování standardních schémat pro podporu návrhu databáze. [6]

Schéma fyzické databáze určuje, jak jsou data fyzicky uložena v úložném systému nebo na diskovém úložišti ve formě souborů a indexů. Návrh databáze na fyzické úrovni se nazývá fyzické schéma. Analytici mohou obvykle použít fyzický datový model k výpočtu odhadů úložiště a může obsahovat konkrétní podrobnosti o přidělení úložiště pro daný databázový systém.

3.2 Typy databází

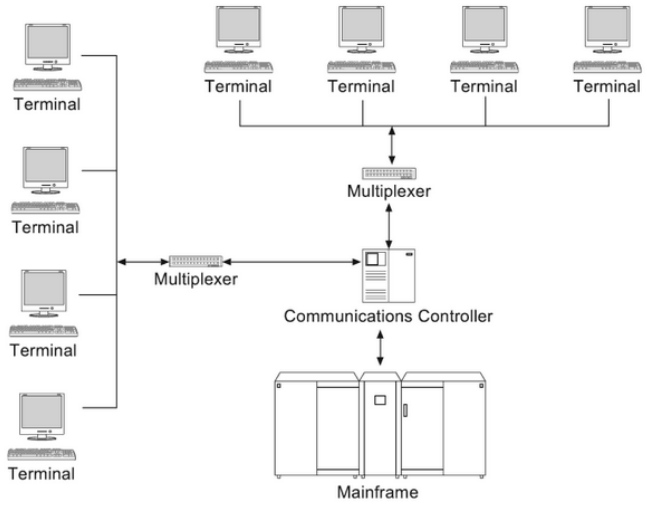
Jelikož se databázové technologie v průběhu let zlepšovaly, zdokonalily se i různé typy databází. Nyní existuje mnoho různých typů databází, z nichž každá má své silné a slabé stránky podle toho, jak jsou navrženy. Pro podniky je obzvláště důležité porozumět různým typům databází, aby bylo zajištěno jejich co nejefektivnější nastavení.

Nejběžnější databáze dle hardwarové architektury:

- Centralizované
- Cloudové
- Distribuované
- Databáze koncových uživatelů

3.2.1 Centralizovaná databáze

V centralizovaných databázích jsou všechna data spravována jedním DBMS a umístěna na jeden uzel, přičemž v síti jsou distribuováni pouze uživatelé (obrázek č. 2). U centralizovaných databází jsou hlavní výhody určeny dobrou integrací dat, která zajišťuje konzistenci dat a snadnou správu transakcí v přísném souladu s vlastnostmi ACID (Atomicity, Consistency, Isolation and Durability). Nevýhodou jsou zejména vysoké náklady na komunikaci a velmi nízká spolehlivost a dostupnost, protože jakákoliv chyba, která blokuje přístup do databáze, přeruší veškerou aktivitu v síti. [7]



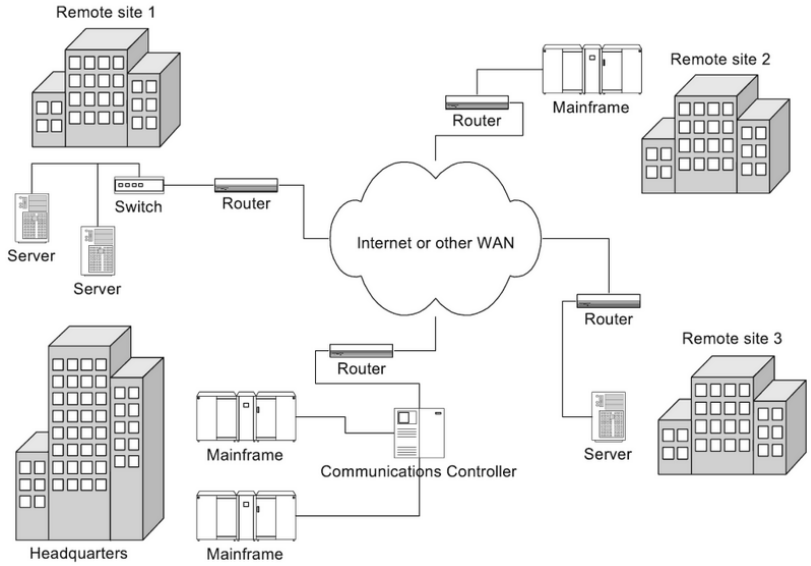
Obrázek 2: Architektura centralizované databáze [8]

3.2.2 Cloudová databáze

U cloudové databáze jsou data uložena na pevném disku nebo serveru, ale informace jsou dostupné online. To usnadňuje přístup k souborům z jakéhokoliv místa s přístupem k internetu. Pro použití cloudové databáze, si ji mohou uživatelé buď vytvořit sami, nebo zaplatit za službu, která pro ně jejich data uloží. Šifrování je nezbytnou součástí každé cloudové databáze, protože všechny informace musí být při přenosu online chráněny. Důležitými kritérii jsou tedy výkon a zabezpečení cloudové databáze. [9]

3.2.3 Distribuovaná databáze

Distribuovaný databázový systém se skládá z kolekce lokálních databází, geograficky umístěných v různých bodech a logicky propojených funkčními vztahy tak, aby na ně bylo možné nahlížet globálně jako na jedinou databázi (obrázek č. 3). Hlavní výhodou použití distribuované databáze je, že sdílením databáze mezi více uzly lze získat rozšíření úložného prostoru a také může těžit z více zdrojů zpracování. Přestože výpočetní výkon v posledních letech značně vzrostl, zpracování velkých dat může vést k celkově špatnému výkonu. Distribucí dat přes více procesorových center lze získat velké výkonnostní výhody díky paralelnímu zpracování dat přes více uzlů, ale zachování vlastností ACID transakcí je mnohem obtížnější. [7]



Obrázek 3: Architektura distribuované databáze [8]

3.2.4 Databáze koncových uživatelů

Koncový uživatel je termín používaný při vývoji produktu, který označuje osobu, která produkt používá. Databáze koncových uživatelů je tedy databáze, kterou primárně používá jedna osoba. Dobrým příkladem tohoto typu databáze je tabulka uložená v uživatelově počítači. [10]

3.3 Relační Databáze

Relační databáze (RDBMS) ukládá a poskytuje přístup k záznamům, které spolu souvisejí. SQL databáze jsou založeny na relačním modelu, intuitivním a přímočarém způsobu reprezentace dat v tabulkách. V relační databázi je každý řádek v tabulce záznamem s jedinečným ID nazývaným primární klíč. Sloupce databázové tabulky obsahují atributy, což usnadňuje stanovení vztahů mezi datovými body. [11]

Nejpopulárnějšími databázovými systémy jsou:

- Oracle Database
- MySQL
- Microsoft SQL Server
- PostgreSQL

3.3.1 Historie relačních databází

V roce 1970 Edgar F. Codd publikoval článek [12] ukazující, jak lze získat přístup k informacím uloženým ve velkých databázích, aniž bychom věděli, jak jsou tyto informace strukturovány nebo kde se v databázi nacházejí. Do té doby vyžadovalo získávání informací poměrně sofistikované počítačové znalosti a služby specialistů, kteří uměli psát programy pro získání konkrétních informací, což bylo časově náročné a nákladné. Díky tomu se otevřela nová možnost nezávislosti na datech. Dle konceptu by uživatelé nemuseli být specialisté, ani by nemuseli vědět, kde informace byly nebo jak je počítač získal. Jednoduchost a flexibilita relačních databází z nich dnes učinila převládající volbu pro finanční záznamy, výrobní a logistické informace a personální data. [13]

3.3.2 SQL – Structured Query Language

SQL (strukturovaný dotazovací jazyk) je standartní dotazovací jazyk pro interakci s RDBMS, který umožňuje správci databáze snadno přidávat, aktualizovat nebo mazat řádky. Dotazy SQL umožňují uživatelům získávat data z databází pomocí pouze několika řádků kódu. Vzhledem k tomuto vztahu jsou relační databáze občas také označovány jako databáze SQL. [14]

3.3.3 Entita, Atributy

Návrh databáze začíná identifikací entit. Entita je libovolný objekt reálného světa, kterou je následně zachycena do datového modelu. Příkladem entity může být osoba, věc, zvíře apod. Entity nemusí být hmotné, protože událost jako je například koncert je také entita. Jednotlivé vlastnosti entit jsou popsány pomocí atributů. Osoba může být popsána osobním číslem, jménem, příjmením a telefonním číslem. Entita koncertu může být popsána názvem, datem, místem a jménem interpreta. Při reprezentaci entity v databázi, se ukládají ve skutečnosti pouze atributy. Každá skupina atributů, která popisuje jeden výskyt entity v reálném světě, představuje instanci entity. [8]

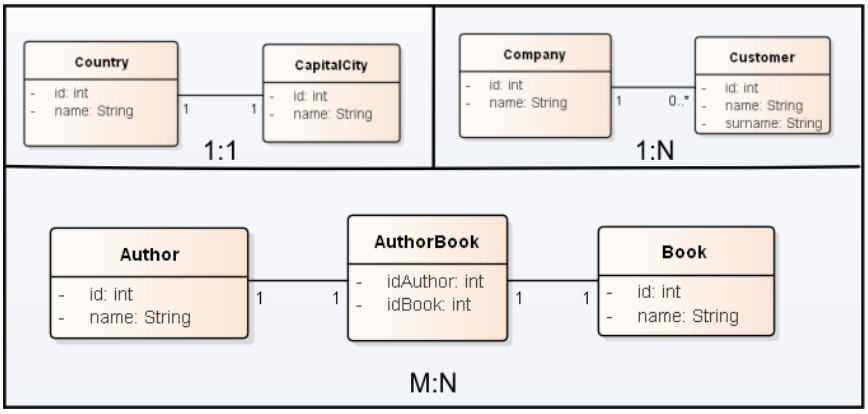
3.3.4 Identifikace entit (instancí)

Jediným účelem vložení dat, která popisují entitu do databáze (instance entity) je pozdější načtení nebo modifikace uložených dat. Pro odlišení jedné entity od druhé jsou použity jedinečné identifikátory tzv. primární klíče. Jednotlivé vazby mezi entitami, se určují pomocí cizích klíčů, které obsahují hodnotu primárního klíče jiné entity. [15]

3.3.5 Vazby

Pokud jsou při návrhu vytvořeny základní entity v databázovém prostředí, pak dalším úkolem je identifikace vztahů mezi těmito entitami. Existují tři základní typy vazeb a to binární, unární nebo n-ární. Binárním vztahem lze popsat mnoho databázových problému reálného světa. Příkladem binárního vztahu je zaměstnavatel a zaměstnanec. Unární vztah má pouze jednoho účastníka a relace je spojena sama se sebou. Pro definici unárního vztahu je nutné určit roli dané vazby. Některé vazby mohou ukazovat na tři nebo více entit, pak se jedná o vztah n-ární. [15]

Pro správnou identifikaci jednotlivých vztahů mezi entitami je důležité nastavit kardinalitu vazby. Je možné narazit na tři typy kardinalit: 1:1, 1:M nebo M:N (obrázek č. 4). Vztah jedna k jedné platí tehdy, když záznam v jedné entitě je spojen přesně s jedním záznamem v jiné entitě. Příkladem vazby 1:1 může být vztah mezi státem a hlavním městem. Stát může mít pouze jedno hlavní město a naopak. Vztah 1:M platí, pokud jeden řádek v tabulce A může být propojen s mnoha řádky v tabulce B, ale jeden řádek v tabulce B je propojen pouze s jedním řádkem v tabulce A. Posledním příkladem je vazba M:N, jejíž definice je vztah M:N tam, kde více než jeden záznam v tabulce souvisí s více než jedním záznamem v jiné tabulce. Pro reprezentaci se využívají pomocné tabulky, pomocí kterých se propojí záznamy jednotlivých tabulek. [15]



Obrázek 4: Příklady kardinalit vazeb [autor]

3.3.6 ACID

ACID transakce zaručují, že databáze bude po spuštění souboru operací v konzistentním stavu. Ne všechny databáze však podporují transakce, které fungují přes více záznamů. Transakce je skupina operací (čtení a zápis) nad databází, která je úspěšná pouze tehdy, jsou-li úspěšné všechny dílčí operace. Transakce mohou ovlivnit jeden nebo více záznamů. ACID je zkratka, která vyjadřuje atomicitu, konzistenci, izolovanost a trvalost dat. Tyto vlastnosti zajišťují, že sada databázových operací ponechá databázi ve funkčním stavu i v případě neočekávaných chyb. [16]

Atomicita zaručuje, že se všemi příkazy, které vytváří transakci, se zachází jako s celkem a buď společně uspějí, nebo selžou (např.v případě nechtěné události, jako je výpadek proudu. Transakce by byla buď úspěšně dokončena, nebo by byla odvolána, pokud by některá část transakce selhala. [17]

Konzistence databáze je definována sadou hodnot, se kterými se musí všechny datové body v databázovém systému řídit, aby byly správně přečteny a přijaty. Pokud do databáze vstoupí data, která nesplňují předem upravené hodnoty, bude to mít za následek chybu konzistence datové sady a celá transakce selže.

Izolace zajišťuje, že všechny transakce fungují v izolovaném prostředí. To umožňuje souběžné provádění transakcí, protože transakce se navzájem neovlivňují. [17]

Trvalost zaručuje, že jakmile je transakce dokončena a změny jsou zapsány do databáze, zůstanou zachovány. Data v systému zůstanou zachována i v případě selhání systému, jako jsou havárie nebo výpadky napájení. [16]

3.3.7 DDL – Data Definition Language

Příkazy DDL se používají k sestavení a úpravě struktury tabulek a dalších objektů v databázi. Po provedení příkazu DDL se změny projeví okamžitě. Pro vytvoření tabulky se používá příkaz *CREATE TABLE* a k její úpravě příkaz *ALTER TABLE*. Příkaz alter table lze použít k určení omezení primárního a cizího klíče a k provedení dalších úprav struktury tabulky. Je možné se kdykoli zbavit jakéhokoli objektu, který byl vytvořen, pomocí příkazu *DROP*. [18]

Nejčastější datové typy jsou znakové řetězce, které jsou nazývány *VARCHAR* nebo *CHAR* pro řetězce s proměnnou nebo pevnou délkou. Datové typy jako *NUMBER* nebo *INTEGER* vyjadřují číslo, ale s rozdílnou přesností. Posledním častým datovým typem je *DATE* vyjadřující datum. [18]

Aby bylo možné pracovat s tabulkami jiných uživatelů, je nutné, aby uživatel „vlastnící“ tabulku udělil práva ostatním pomocí příkazu *GRANT*. Podle udělených příslušných oprávnění je možné manipulovat s daty v ostatních tabulkách (výběr, vkládání, aktualizace nebo smazání). Práva se udělují na každou tabulku zvlášť. [19]

3.3.8 DML – Data Manipulation Language

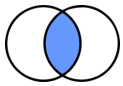
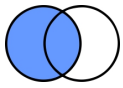
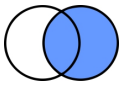
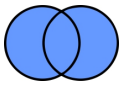
Pro práci s daty v tabulkách se používají příkazy DML. Pokud je uživatel připojený k víceuživatelské databázi, pak ve skutečnosti pracuje se soukromou kopií tabulek. Pro zviditelnění změn ostatním uživatelům se využívá příkaz *COMMIT*. Pro přidávání řádků do tabulky se využívá příkaz *INSERT*, úpravy se provádějí pomocí příkazu *UPDATE* a smazání pomocí příkazu *DELETE*. V případě, že nastane situace, kdy je zapotřebí vrátit změny, provedené v databázi, pak se využívá příkaz *ROLLBACK.* [18]

3.3.9 Příklady operací s tabulkami

Pro práci s relační databází existuje spousty operací, díky kterým lze filtrovat data z databáze. Základní funkcí je operace *SELECT*, pomocí níž lze získat data z tabulky. V případě potřeby filtrování dat, lze využít klauzuli *WHERE.* Ta určuje kritéria, která musí být splněna, pro následné vypsání do výsledků dotazu. V momentě, kdy je nutné mít data seřazená například podle abecedy, lze využít klauzuli *ORDER BY*. V SQL databázi je možnost použití výpočetních operací jako je součet, průměr nebo počet. Do dotazu se zapisují pomocí identifikátorů *SUM, AVG* a *COUNT.* Po využití výpočetní operace se do výsledku dotazu vypíše její výsledek. [20]

Jak bylo zmíněno, v relační databázi jsou data strukturována a organizována do tabulek. Data v těchto tabulkách mají často vzájemné vztahy nebo závislosti a mohou být propojeny pomocí SQL, pomocí příkazu *JOIN*. Při specifikaci příkazu *JOIN* se následně mění výstup spojené tabulky (viz. tabulka č.1).

Tabulka 1: Popis typů spojení v relačních databázích [autor]

Typ spojení	Definice	Vennův diagram
INNER	Spojení záznamů se shodnými hodnotami	
LEFT	Spojení záznamů z levé tabulky se shodnými záznamy z tabulky pravé	
RIGHT	Spojení záznamů z pravé tabulky se shodnými záznamy z tabulky levé	
FULL	Vrácení záznamů, pokud je shoda v levé nebo pravé tabulce	
CROSS	Kartézský součin dvou tabulek – všechny možné kombinace řádků	
SELF	Tabulka spojená sama se sebou	

3.4 Nerelační databáze

Nerelační databáze se od relačních systémů se v mnoha významných ohledech široce odlišují. Nejdůležitější je, že nepoužívá tabulky jako strukturu uložiště, nepoužívá SQL jako svůj dotazovací jazyk, nelze provádět operace spojení,

nezaručuje vlastnosti ACID a lze ji horizontálně škálovat. Většina nerelačních databází používá objekty formou dokumentu, který je specifikovaný v JSON (JavaScript Object Notation) formátu. [21]
Nejpopulárnější nerelační databázové systémy:

- MongoDB
- Redis
- Cassandra

3.4.1 Flexibilní schémata

NoSQL databáze obecně poskytují flexibilní schémata, která umožňují rychlejší a iterativnější vývoj. Díky flexibilnímu datovému modelu jsou databáze NoSQL ideální pro polostrukturovaná a nestrukturovaná data. Flexibilita usnadňuje mapování dokumentů na entitu nebo objekt, protože se může shodovat s datovými poli reprezentované entity, i když se dokument podstatně liší od jiných dokumentů ve sbírce. V praxi však dokumenty v kolekci sdílejí podobnou strukturu a během operací lze vynutit pravidla ověřování dokumentů pro kolekci. [21]

3.4.2 Horizontální škálování

Horizontální škálování se týká propojení dalších uzlů ke sdílení zátěže. To je u relačních databází obtížné, protože je složité rozložit související data mezi uzly. U nerelačních databází je to jednodušší, protože kolekce fungují samostatně a nejsou žádným způsobem propojeny. To umožňuje jejich jednodušší distribuci mezi uzly, protože dotazy je nemusí spojovat. Zatímco vertikální škálování pracuje s přidáváním paměti, horizontální škálování přidává další zařízení. [21]

3.4.3 Key-Value databáze

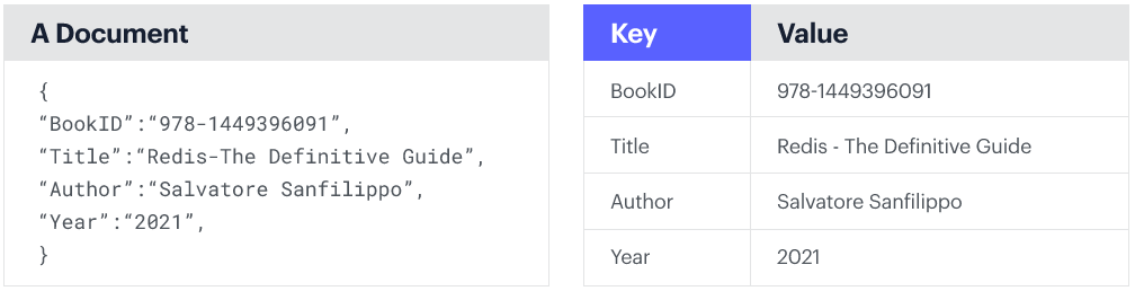
Key-Value databáze, jsou typy databází, ve kterých jsou data uložena ve formátu „klíč-hodnota“ a jsou optimalizována pro čtení a zápis těchto dat. Data jsou načítána pomocí jedinečného klíče, aby bylo možné získat přidruženou hodnotu s každým klíčem. Databáze klíč-hodnota používají kompaktní a efektivní indexové struktury, aby byly schopny rychle a spolehlivě vyhledat hodnotu podle jejího klíče, což je činí ideálními pro systémy, které potřebují být schopny vyhledávat a získávat data

v konstantním čase. Redis je například databáze klíč-hodnota, která je optimalizována pro sledování relativně jednoduchých datových struktur. Tím, že podporuje pouze omezený počet typů hodnot, je Redis schopen vystavit jednoduché rozhraní jejich dotazování a manipulaci s nimi. [22]
Příklady kdy využít typ ukládání „klíč-hodnota“:

- mechanismus mezipaměti pro často používaná data (caching)
- aplikace je navržena na jednoduché klíčové dotazy
- náhodný přístup k datům v reálném čase

3.4.4 Dokumentové databáze

Dokumentová databáze je typ nerelační databáze, která ukládá data jako strukturované dokumenty (obrázek č. 5). Je to moderní způsob, jak ukládat data ve formátu JSON. Dokument může být typu PDF, XML či JSON. [23]



Obrázek 5: Příklad dokumentové databáze [23]

Pro práci s dokumentovou databází se využívá kolekce. Jedná se o skupinu dokumentů, které mají podobný obsah. Ne všechny dokumenty v kolekci musí mít stejná pole, protože databáze dokumentů mají flexibilní schéma. Vývojáři běžně považují práci s daty v dokumentech za jednodušší a intuitivnější než práci s daty v tabulkách. Dokumenty se mapují na datové struktury ve většině oblíbených programovacích jazyků. Vývojáři se nemusí starat o ruční rozdělování souvisejících dat do více tabulek při jejich ukládání nebo jejich opětovné spojování při načítání. Databáze dokumentů mají obvykle API nebo dotazovací jazyk, který umožňuje vývojářům provádět CRUD operace. [24]

V následujících případech se vyplatí použít dokumentové databáze:

- aplikace nepotřebuje tabulková data
- aplikace potřebuje zvládnout spoustu nepřetržitých čtení, zápisů a rychlý přístup do paměti
- aplikace potřebuje zpracovávat širokou škálu přístupových vzorů a datových typů

3.4.5 Grafové databáze

Grafová databáze je definována jako specializovaná jednoúčelová platforma pro vytváření a manipulaci s grafy. Grafy obsahují uzly, hrany a vlastnosti, které se používají k reprezentaci a ukládání dat způsobem, na který relační databáze nejsou vybaveny. Mezi nejpopulárnější modely patří grafy vlastností a RDF grafy. Graf vlastností se zaměřuje na analýzu a dotazování, zatímco graf RDF zdůrazňuje integraci dat. Oba typy grafů se skládají ze souboru vrcholů a spojeními mezi těmito body. Grafové databáze jsou extrémně flexibilní a výkonný nástroj, jelikož díky formátu grafu lze s mnohem menším úsilím určit složité vztahy pro hlubší náhledy. Obecně spouštějí dotazy v jazycích, jako je *Property Graph Query Language.* [25]

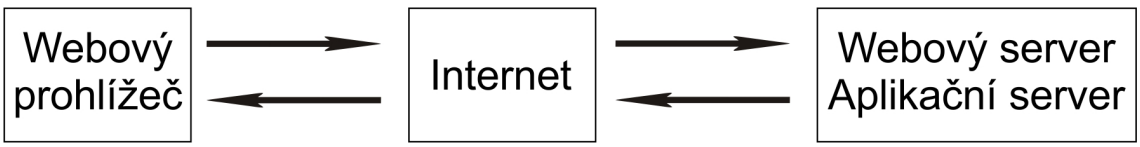
3.4.6 Objektově orientované databáze

Objektově orientovaná databáze je databázový systém, který může pracovat se složitými datovými objekty, tedy s objekty, které se používají v objektově orientovaných programovacích jazycích. V objektově orientovaném programování je vše objekt. Kromě toho je mnoho objektů poměrně složitých, mají různé vlastnosti a metody. Objektově orientované systémy správy databází spolupracují s OOP a usnadňují ukládání a získávání objektově orientovaných dat. Obecně je objektově orientovaná databáze považována za podmnožinu databáze NoSQL. [26]

4 Webová aplikace

Webová aplikace vyžaduje vývojové procesy, které zahrnují sběr požadavků a programování prostřednictvím různých jazyků, což vede k heterogenitě v jejím vývoji. Některé z webových aplikací jsou statické, díky čemuž nevyžadují na serveru vůbec žádné zpracování, zatímco jiné webové aplikace jsou dynamické a vyžadují zpracování na straně serveru v závislosti na akci, kterou vstup uživatele vyžaduje. Díky jejich rychlému rozvoji a výhodám jsou využívány po celém světě pro interakci mezi koncovým uživatelem a obsahem stránky. [27]

Obecně platí, že webová aplikace nevyžaduje jejich stahování, protože se jedná o počítačový program, který se obvykle nachází na vzdáleném serveru. Aplikační server provádí úlohu, kterou požadují klienti, kteří také mohou někdy potřebovat databázi k uložení informací. Každý uživatel může k serveru přistupovat pomocí standartního webového prohlížeče. Princip webové aplikace je znázorněn na obrázku č. 6. Webové aplikace jsou kódovány pomocí jazyků podporovaných téměř všemi webovými prohlížeči, protože to jsou jazyky, které při vykreslování spustitelného programu spoléhají na webové prohlížeče. Většina aktuálně dostupných webových aplikací na internetu je napsána pomocí programovacích jazyků, jako je HTML, CSS a Javascript, které se používají při vytváření front-endu (aplikace na straně klienta). K vytvoření skriptu webových aplikací se programování na straně serveru provádí pomocí programovacích jazyků, jako je Java, Python, PHP, Ruby atd. Mezi běžné webové aplikace patří e-mail, e-shop, online aukce, wiki, sociální sítě a další. Mnoho společností přesouvá své zaměření na webové aplikace, které lze dodávat jako Software-as-a-Service (SaaS). [28]



Obrázek 6: Princip webové aplikace [autor]

Software-as-a-Service je cloudový model dodávání softwaru, ve kterém poskytovatel cloudu vyvíjí a spravuje aplikační software, poskytuje automatické aktualizace a zpřístupňuje software svým zákazníkům prostřednictvím internetu. Zákazníci SaaS tak mohou nasazovat a upgradovat podniková řešení rychleji a předvídat celkové náklady s větší přesností. [29]

4.1 Výhody a nevýhody webové aplikace

Níže jsou popsány některé kritické a užitečné body, které pomohou porozumět výhodám a nevýhodám webové aplikace.
Výhody, které nabízí webová aplikace:

- Každá webová aplikace je přístupná na jakémkoli operačním systému, jako je Windows, Mac a Linux, pokud je prohlížeč kompatibilní.
- Aplikaci obvykle není nutné instalovat na pevný disk, a tím jsou vyřešeny problémy spojené s omezeným prostorem a případnými právy instalace.
- Všichni uživatelé mají přístup ke stejné verzi webové aplikace, což eliminuje všechny problémy s kompatibilitou.
- Po zpřístupnění cloudu je nyní úložný prostor prakticky neomezený, pokud si ho uživatel můžete dovolit.
- Aplikace je vyvinuta pomocí základních programovacích jazyků, jako je HTML, CSS, které jsou kompatibilní a dobře známé mezi IT profesionály.

Každá technologie má své výhody, ale i nevýhody. V každém případě je důležitý návrh a zvážení nevýhod dané technologie.
Mezi nevýhody webových aplikací patří:

- Připojení k internetu je nezbytné pro přístup k webové aplikaci.
- Aplikace musí podporovat několik webových prohlížečů, včetně nových a starých verzí.
- Klienti musí utrácet velké množství peněz, aby udržovali kondici své webové aplikace a poskytoval aktualizaci vždy, když se vyskytne problém.
- Webová aplikace může čelit problémům, pokud není naprogramovaná responzivním způsobem. [30]

4.2 Technologie pro vývoj front-endu webových aplikací

Front-end vývoj zahrnuje tu část vývoje webu, která je viditelná pro koncového uživatele. Proto jsou front-endoví vývojáři z velké části zodpovědní za vytváření uživatelských rozhraní. Vývoj front-endu v podstatě zahrnuje přeměnu back-endového obsahu na něco, co je pro uživatele snadno dostupné prostřednictvím grafického rozhraní. Existuje mnoho front-endových technologií, které se dají použít pro webovou aplikaci a jejich základem jsou jazyky HTML, CSS a JavaScript.

4.2.1 HTML

Koncept HTML vytvořil sir Tim Berners-Lee na konci roku 1991, ale nebyl oficiálně vydán. Dlouho předtím, než byly zavedeny revidované standardy a specifikace, umožnila každá verze svému uživateli vytvářet webové stránky mnohem snadněji a hezčím způsobem. HTML 1.0 bylo vydáno v roce 1993 se záměrem sdílet informace, které mohou být čitelné a dostupné prostřednictvím webových prohlížečů, ale málo vývojářů se podílelo na tvorbě webových stránek. Pak přichází HTML 2.0 vydané v roce 1995, ve kterém se zdokonalovaly základní vlastnosti tohoto jazyka. [31]

Dave Ragget vydal článek [32], ve kterém zahrnul vylepšené funkce HTML a vznikla nová verze HTML 3.0. Problémem ale bylo, že tyto výkonné funkce nového HTML zpomalily prohlížeč po aplikaci dalších vylepšení. Předposlední verzí je HTML 4.01, která byla úspěšnou a velmi používanou verzí. Rozšířením této verze vznikla současná verze HTML 5.0, která byla publikovaná v roce 2012. [31]

HTML (Hypertextový značkovací jazyk) je jazyk pro publikování textových a multimediálních informací na webových stránkách. Používá znaky k vytváření strukturovaných dokumentů prostřednictvím sémantiky pro text, jako jsou nadpisy, odstavce a seznamy, stejně jako pro odkazy a další prvky. HTML také umožňuje autorům vkládat obrázky a objekty do stránek a může vytvářet interaktivní formuláře. Organizace Internet Engineering Task Force zahájila jako první úsilí standardizovat HTML v roce 1995 pomocí HTML 2.0. Úsilí IETF udržovat HTML se zastavilo, takže standardizaci převzalo W3C. [33, 34]

4.2.2 CSS

CSS je jazyk, který se používá ke stylování webové stránky v kombinaci s HTML. Důvodem vzniku tohoto jazyka je ten, že HTML nikdy nemělo obsahovat tagy pro formátování webové stránky. Základní formátování bylo zpřístupněno ve verzi HTML 3.2, ale pro vývojáře se jednalo o zdlouhavý proces formátování. Pro vyřešení tohoto problému vznikl jazyk CSS, který odstranil formátování stylu ze stránek HTML. [33]

Existují tři možnosti, jak lze nastylovat webové stránky pomocí CSS a to řádkově, interně nebo externě. Rozdíly jsou v nadefinování stylů. Řádkové CSS styluje jednotlivé prvky přidáním atributu „*style*“ do počátečního tagu prvku.
Příklad změny barvy nadpisu pomocí řádkového CSS:

```
<h1 style="color:blue;">Nadpis o velikosti H1</h1>
```

Interní styly CSS upravují obsah stránky přidáním tagu „*style*“ do hlavičky dokumentu HTML.
Příklad změny barvy nadpisu pomocí interního CSS:

```
<head>
    <title>Titulek stránky</title>
    <style>
      h1 {
        color: blue;
      }
    </style>
</head>
```

Externí styly CSS upravují obsah jedné nebo více stránek propojením dokumentu HTML s externí šablonou stylů.
Příklad propojení HTML stránky s CSS:

```
<link rel="stylesheet" href="styl.css" />
```

4.2.3 JavaScript

JavaScript je skriptovací jazyk, který umožňuje vytvářet dynamicky aktualizovaný obsah. Jádro jazyka se nachází na straně klienta a skládá se z běžných programovacích funkcí, které umožňují dělat věci, jako je ukládání užitečných hodnot do proměnných, operace s částmi textu, spouštění kódu v reakci na určité události na webové stránce a mnoho dalšího. [35]

JavaScript lze rozdělit do dvou kategorií – kód na straně serveru a na straně klienta, zejména v souvislosti s vývojem webu. Kód na straně klienta je kód, který se spouští na počítači uživatele. Při prohlížení webové stránky se stáhne kód na straně klienta, poté se spustí a zobrazí v prohlížeči. Co nevidí uživatel je kód, který je spuštěn na straně serveru, ve kterém se odehrává veškerá business logika aplikace. Příklady populárních webových jazyků na straně serveru zahrnují PHP, Python, ASP.NET a dokonce i JavaScript. JavaScript je možné použít jako jazyk na straně serveru například v oblíbeném prostředí Node.js. JavaScript se na stránku HTML aplikuje podobným způsobem jako CSS. Zatímco CSS používá prvky „*link*“ a „*style*“, u JavaScriptu se používá pouze prvek „*script“.*

4.2.4 React a React Native

ReactJS je JavaScriptová knihovna vytvořená společností Facebook, která slouží k vývoji uživatelského rozhraní. Umožňuje vývoj velkých webových aplikací, které mohou měnit svá data, bez obnovování stránek. Jeho výkon spočívá v používání virtuálního DOM (Document Object Model), který na základě aktuálního stavu komponenty a změn, ke kterým došlo, rozhoduje, zda se komponenta musí znovu načíst nebo ne. Pro tento účel se používá JSX, které umožňuje vývojářům vytvářet virtuální DOM pomocí syntaxe XML. Nevýhodou je, že výkon může být snížen při navrhování složitých aplikací, kde se aplikace musí vypořádat s velkým množstvím dat. Aplikace může přestat reagovat a dochází k jejímu zpoždění. To je běžným problémem v podnikových aplikacích. [36]

ReactJS i React Native sdílejí stejnou syntaxi. React Native byl vytvořen jako způsob, kterým mohou vývojáři vytvářet mobilní aplikace pro různé platformy s využitím svých stávajících znalostí pro vývoj webových aplikací, jako jsou HTML, CSS, JavaScript a základní knihovna React.

4.3 Technologie pro vývoj back-endu webových aplikací

Back-endové frameworky se skládají z nástrojů a jazyků používaných při programování na straně serveru ve webovém vývojovém prostředí. V back-endu se vytváří základní logika, která zajišťuje správný běh aplikace a dynamicky se integruje s front-endem. V dnešní době je pro společnosti zabývající se vývojem softwaru zásadní najít vhodný back-endový framework pro své projekty, protože se na trhu nachází široká nabídka jazyků a frameworků.

4.4 NodeJS

Node.js je platforma postavená na runtime prostředí V8 JavaScript enginu pro snadné vytváření rychlých a škálovatelných aplikací. Node.js používá událostmi řízený (event driven), neblokující I/O model, díky kterému je efektivní a ideální pro datově náročné aplikace v reálném čase. [37]

Programovací jazyk bez jakýchkoliv knihoven je pro praktické aplikace nevhodný. Proto má Node.js API pro IO operace, základní síťování, moduly HTTP serveru, kompresi a mnoho dalších. Knihovny Node.js, které jsou k dispozici lze rozšířit o další balíčky, které jsou dostupné prostřednictvím veřejných nebo soukromých registrů balíčků. [38]

4.4.1 Využití

Node.js se používá k vytváření škálovatelných desktopových a mobilních aplikací, stejně jako webových, které potřebují zvládnout velké množství připojení bez jakýchkoli problémů. Z tohoto důvodu některé z nejpopulárnějších aplikací po letech používání jiných prostředí přešly na Node.js. Mezi nejznámější aplikace, které jsou naprogramovány v Node.js patří například Netflix, Twitter a Uber.

4.4.2 Výhody a nevýhody NodeJS

V Node.js se JavaScript používá pro front-endový i back-endový vývoj, díky čemuž je jazyk konzistentnější v celé aplikaci. Při používání Node.js je možné vyměňovat kód mezi klientskými a serverovými aplikacemi a pro celý vývojový proces lze používat JavaScript, což umožňuje lepší komunikaci mezi back-endovými a front-endovými týmy. Jednou z největších výhod je Node Package Manager. NPM umožňuje stahovat a používat balíčky kódu poskytnuté jinými vývojáři ve vlastních projektech. Největší registr softwarových knihoven na světě provozuje Node.js. Obsahuje více než 1,3 milionu balíčků v hlavním registru, z nichž všechny byly vytvořeny komunitou Node.js, takže je snadné najít řešení různých problémů. Node.js je známý tím, že používá architekturu jednovláknové smyčky událostí, která je ideální pro mikroslužby. Když se aplikace Node.js spustí, inicializuje smyčku událostí a poté pokračuje v provádění instrukcí. [39]

Největší nevýhodou Node.js je jeho neschopnost rychle zpracovávat úlohy vázané na CPU. Problém nastane, když Node.js přijme úkol vázaný na CPU. Kdykoli do smyčky událostí přijde složitý požadavek, Node.js nastaví veškerý dostupný CPU, aby jej nejprve zpracoval, a pak odpověděl na další požadavky ve frontě. To má za následek pomalé zpracování a celkové zpoždění ve smyčce událostí, což je důvod, proč se Node.js nedoporučuje pro náročné výpočty. Ačkoli byla zmíněna výhoda v množství balíčků, pak se výhoda stává zároveň nevýhodou. Základní moduly Node.js jsou stabilní a lze je považovat za vyspělé, ale v registru NPM je mnoho nástrojů, které jsou buď nekvalitní, nebo nejsou řádně zdokumentovány a otestovány. Navíc samotný registr není dostatečně strukturován, aby nabízel nástroje na základě jejich hodnocení nebo kvality. [39]

4.4.3 Alternativy

Python je interpretovaný, objektově orientovaný, vysokoúrovňový programovací jazyk s dynamickou sémantikou. S Pythonem lze snížit náklady na vývoj na rozdíl od jiných programovacích jazyků. Dosahuje se ho implementací knihoven s hotovými moduly a podporou komunity. [40]

ASP.NET je skriptovací jazyk na straně serveru a umožňuje webovým vývojářům vytvářet dynamické, datově řízené a interaktivní aplikace a webové

stránky. Úroveň výkonu je významným přínosem, díky kterému je ASP.NET alternativou k Node.js. Díky vysokému výkonu ASP.NET lze využít k vytváření webových aplikací a dynamických webů využitím správných služeb pro vývoj webu.

Java je objektově orientovaný, robustní a bezpečný jazyk, který se používá pro vývoj mobilních a webových aplikací, vestavěných systémů a technologií s velkým obsahem dat. Java je z hlediska zabezpečení silnou alternativou k Node.js. Má mnoho knihoven, běžně používaných bezpečnostních algoritmů a protokolů, které mohou vývojáři snadno integrovat do svého kódu. [40]

5 Praktická část

V praktické části práce je popsáno řešení vývoje řízeného skladu formou jednoduché webové aplikace, která bude mít dvě verze. Obě varianty budou vyvíjeny pomocí jazyka Node.js, který bude řídit veškerou business logiku aplikace. Rozdílem těchto dvou verzí je propojení s databází a typem databáze. V jedné variantěbude aplikace připojena k databázi NoSQL (MongoDB) a v druhé bude využívat SQL databázi (MySQL). Aby bylo možné správně vyhodnotit porovnání těchto dvou databází, obě budou spouštěny na lokálním zařízení, pro přesnější porovnání hardwarových nároků. Následující kapitoly se budou zabývat vývojem řízeného skladu, návrhem databází, vývojem samotné aplikace a porovnáním jednotlivých řešení.

5.1 Skladové hospodářství – řízený sklad

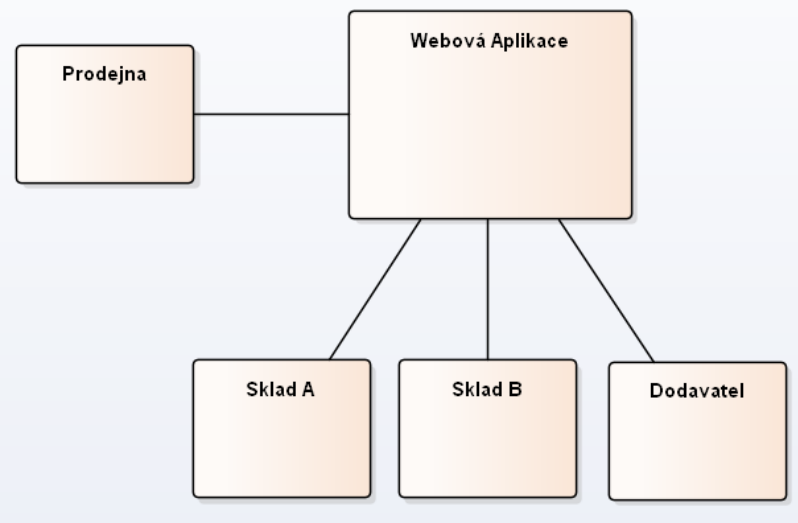
S rychlým rozvojem odvětví logistiky se vysokorychlostní, vysoce efektivní a přesné dodávky staly novými cíli logistického průmyslu. I v případě, že se nejedná o logistickou firmu, každý podnik potřebuje skladové zásoby a jejich efektivní naskladňování pro následné využití (výroba, prodej, …). Spousty malých firem si musí vést evidenci zásob ručně a v některých případech může docházet ke snižování logistické efektivity. Při ručním třídění a přesouvání zásob může docházet k chybám a následným inventurním ztrátám. Zmíněné problémy se dají vyřešit pomocí digitalizace a vývoje řízeného skladu. U řízeného skladu se nemusí jednat o velkou komplexní aplikaci, kterou by si malé firmy nemohly dovolit, protože by byla velmi drahá na vývoj a údržbu, ale může se vytvořit jednoduchá webová aplikace s formou polo automatizace. V následující kapitole jsou rozebrány jednotlivé řešení řízených skladů.

5.2 Možnosti řízeného skladu

Klasickým příkladem řízeného skladu je komplexní inteligentní systém, který nabízí přehled o veškerých zásobách podniku, kontroluje a řídí operace, týkající se zásobování produktů od dodavatelů, až po naskladňování do kamenných obchodů podniku. Vzhledem k tomu, že internet a digitální technologie proměnily způsob, jakým zákazníci nakupují (mění se nákupní vzorce zákazníků), operace

plnění musí čelit změnám pomocí vlastního digitálně propojeného řešení. Tato zmíněná řešení vyžadují komplexní algoritmy pro stálý chod aplikace a jsou vysoce nákladná jak na pořízení, tak údržbu. Proto se takové řešení hodí pouze pro firmy, které mají složitější logistické procesy, vysokokapacitní vychystávání objednávek a velký provoz. Existují cloudová řešení, která lze integrovat do fungujícího podniku. Z počátku je proces složitý a hrozí riziko dočasného zpomalení celkového chodu podniku. Jedním z nejznámějších cloudových řešení je systém SAP. Poskytuje flexibilní, automatizovanou podporu při zpracování všech pohybů zboží a při správě zásob ve skladovém hospodářství. Systém podporuje plánované a efektivní zpracování všech logistických procesů ve skladu.

Jak bylo zmíněno, komplexní řešení je finančně náročné a pro malé podniky nevýznamné. Proto se tato práce zabývá návrhem jednoduchého řešení pro malé firmy, které bude cílené na jednu pobočku s více skladovacími prostory (obrázek č. 7). Webová aplikace bude mít za úkol kontrolovat zásoby na pobočce a vychystávat přesuny zboží formou úkolů v závislosti na požadovaných stavech jednotlivých produktů. Další výhodou aplikace bude generování objednávkových listů, které zjednoduší procházení stavů produktů a jejich doobjednávání v zadaném intervalu. Samozřejmostí aplikace je ruční přesun a manipulace zboží. U vytvořeného řešení bude porovnán databázový výkon SQL a NoSQL.

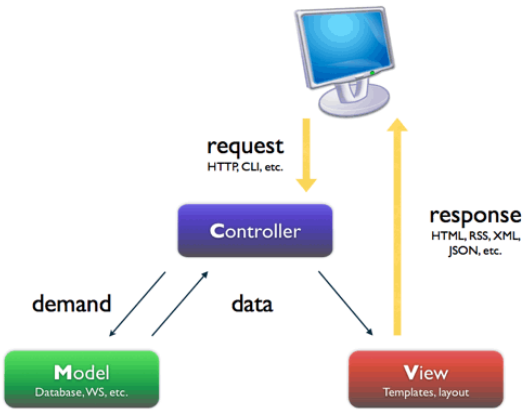


Obrázek 7: Zjednodušený návrh struktury systému [autor]

5.3 Vývoj webové aplikace

Vývoj webových aplikací prošel dlouhou cestu od začátku webových stránek. K vytváření webových aplikací se nyní používá nesčetné množství technologií a programovacích jazyků. Mnoho vývojářů se pokusilo přijít s různými technologiemi, aby se zlepšila uživatelská zkušenost a pomohlo vývojářům budovat rychlejší a výkonnější webové aplikace. V dnešní době se technologie a jazyky zaměřují na vzor MVC (model, view a controller). Vzor MVC poprvé představil Trygve Reenskaug v 70. letech v Xerox Parc. Dle něj „zásadním účelem MVC je překlenout propast mezi mentálním modelem lidského uživatele a digitálním modelem, který existuje v počítači".[41]

Aplikace je rozdělena do tří hlavních kategorií: model hlavní aplikační domény, prezentace dat v tomto modelu a interakce uživatele (obrázek č. 8). Návrhový vzor MVC se tak dobře hodí pro vývoj webových aplikací, protože kombinuje několik technologií obvykle rozdělených do sady vrstev. Interakce uživatele s aplikací MVC probíhá v přirozeném cyklu. Uživatel provede akci a v reakci na to aplikace změní svůj datový model a poskytne uživateli aktualizovaný pohled. To je vhodné pro webové aplikace dodávané jako posloupnost HTTP požadavků a odpovědí (request, response).



Obrázek 8: MVC architektura [41]

5.3.1 Model

Modelová vrstva je zodpovědná za business logiku aplikace, zapouzdří metody pro přístup k datům (databázím, souborům atd.) a zpřístupní opakovaně použitelnou knihovnu tříd. Model je obvykle vytvořen s ohledem na abstrakci dat, validaci

a autentizaci. Model by měl být co nejjednodušší a měl by zahrnovat pouze zpracování dat, které je přísně svázáno s reálným objektem, který je modelován. Například objekt získá informace o zákazníkovi z databáze, bude s nimi manipulovat a aktualizuje je zpět do databáze nebo je použije k vykreslení dat.

5.3.2 Pohled (View)

Pohled je zodpovědný za správu grafického uživatelského rozhraní. To znamená všechny formuláře, tlačítka, grafické prvky a další HTML prvky, které jsou uvnitř aplikace. Oddělením designu aplikace od logiky aplikace se výrazně snižuje riziko, že se objeví chyby, když se designér rozhodne změnit grafické uživatelské rozhraní dané aplikace. Zároveň je práce vývojářů značně omezena, protože již nepotřebuje vidět prvky HTML kódu, designové prvky a grafické prvky. Vrstva View je to, co lze normálně nazvat web design, který řídí způsob zobrazení dat a způsob interakce uživatele
s nimi. Poskytuje také způsoby shromažďování dat od uživatelů. Technologie, které se používají hlavně v pohledech, jsou HTML, CSS a JavaScript.

Většina frameworků webových aplikací dnes používá šablonovací engine, který využívá prvky generátoru, aby udržely HTML kód na minimu a snížilo se tak riziko překlepů. Tyto generátory se obvykle používají k vytváření složitých webových částí. Tímto způsobem může front-endový vývojář vidět konečný kód až poté, co byl vygenerován, a nemá téměř žádný způsob, jak šablonu upravit.

5.3.3 Controller

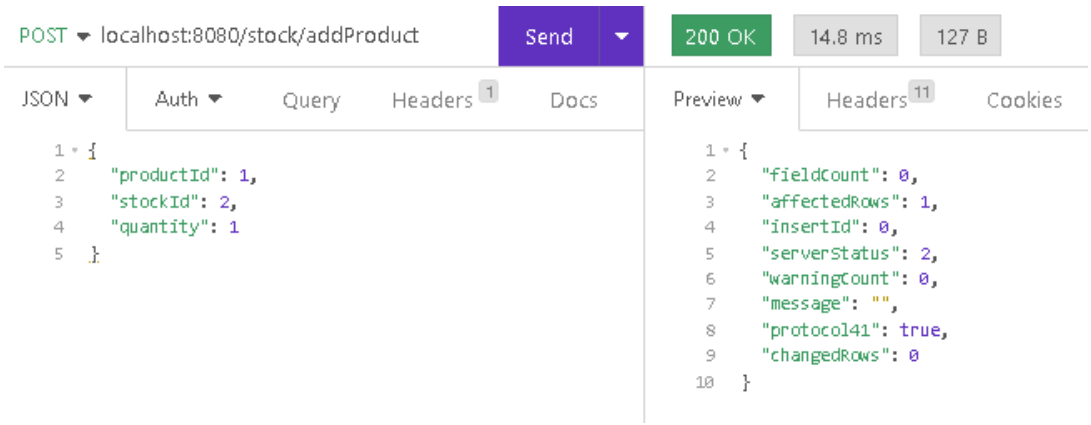
Controller funguje jako rozhraní mezi komponentami Model a View pro zpracování veškerých příchozích požadavků. Controller manipuluje s daty pomocí komponenty Model a provádí interakci s View za účelem vykreslení konečného výstupu. Controller je odpovědný za zpracování událostí, které mohou být spuštěny buď uživatelem při interakci s aplikací, nebo systémovým procesem. Controller přijímá požadavky a připravuje data na odpověď. Když požadavek dorazí na server, framework MVC jej odešle do metody v řadiči na základě adresy URL. Při prací s požadavky se využívá REST API. Když je požadavek klienta proveden prostřednictvím rozhraní, přenese reprezentaci stavu zdroje na žadatele nebo

koncový bod. Tyto informace nebo reprezentace jsou dodávány v jednom z několika formátů prostřednictvím HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP nebo prostý text. JSON je obecně nejoblíbenější formát souborů k použití, protože navzdory svému názvu je jazykově agnostický a je čitelný jak pro lidi, tak pro stroje.

Tabulka 2: Druhy nejpoužívanějších http požadavků

Typ požadavku	Popis
GET	Metoda GET se používá k získání informací z daného serveru pomocí daného URI. Požadavky pomocí GET by měly pouze načítat data a neměly by mít žádný jiný vliv na data.
POST	Požadavek POST se používá k odeslání dat na server, například informací o zákaznících, nahrání souboru atd. pomocí formulářů HTML.
DELETE	Odebere všechny aktuální reprezentace cílového zdroje dané URI.
PUT	Nahradí všechny aktuální reprezentace cílového zdroje nahraným obsahem.

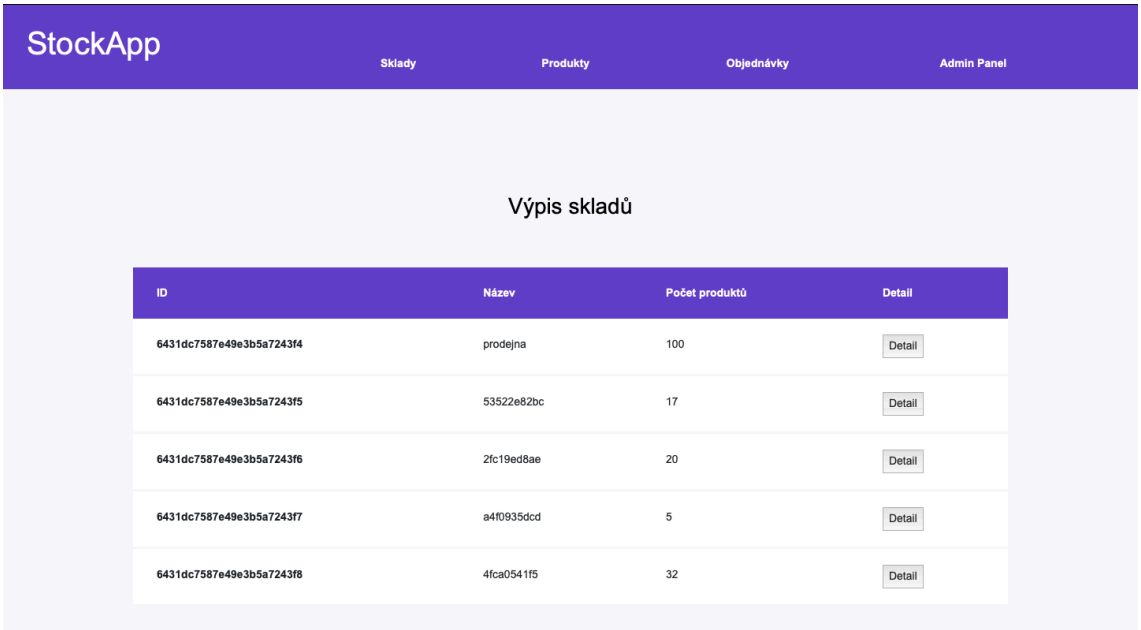
Pro testování jednotlivých požadavků, lze využít front-end nebo nástroje třetích stran, ve kterých se odešle požadavek přímo na back-end. Příklady těchto aplikací jsou Insomnia REST client nebo PostMan. Oba tyto klienti pomáhají při interakci, navrhování, ladění a testování API (obrázek č. 9). Insomnia kombinuje snadno použitelné rozhraní s pokročilými funkcemi.



Obrázek 9: Požadavek v programu Insomnia rest client [autor]

5.3.4 Front-end webové aplikace

Práce se zabývá především back-endovou částí webové aplikace a porovnáním databázových řešení, proto veškeré testování neprobíhalo na front-endu, ale pomocí zmíněného klienta třetí strany Insomnia, který simuluje odesílání požadavků z front-endu a výpis odpovědí z back-endového serveru. Front-endová část je navržena pomocí jazyků HTML, CSS a JavaScriptové knihovny ReactJS. Hlavními stránkami jsou: přihlašovací formulář do skladového systému, zobrazení skladů a skladových zásob, manuální přesouvání zboží a stránka s výpisem objednávek zboží.



Obrázek 10: Hlavní stránka webové aplikace po přihlášení [autor]

Ve výpisu skladů se nachází detail skladů uložených v databázi (obrázek č. 10). Na prvním poli se vždy nachází hlavní sklad (prodejna). U každého řádku je možnost přesměrování na detail skladu, ve kterém jsou vidět veškeré produkty, které sklad obsahuje. Další stránkou jsou produkty, kde se nachází výpis všech produktů. Předposlední stránka objednávky obsahuje výpis vytvořených objednávek, vyfiltrovaných podle datumu. Další funkcí této stránky je tlačítko na vytvoření objednávky. Poslední záložkou je administrační panel, kde je možné přidávat do databáze nové produkty a sklady.

```
async createStock(dtoIn) {
    let commandUri = this.getUri( useCase: "stock/create");
    return Calls.call( method: "post", commandUri, dtoIn);
},

function createStock(header, content) {
    Calls.createStock( dtoIn: {header, content}).then(result => {
            listPosts();
            result.errorCode ? setCreateErrorSnack( value: true) : setCreateSuccessSnack( value: true);
        }
    );
}
```

Obrázek 11: Příklad provolání funkce – vytvoření skladu [autor]

5.4 Vývoj back-end

Důležitou součástí vývoje back-endu je správné roztřídění souborů do složek a jejich konkrétní popis z důvodu přehlednosti kódu aplikace. Soubory jsou popsány vždy podle kolekce, se kterou jsou provázány. Příkladem může být controller. V adresářové struktuře je vytvořena složka controller, ve které jsou zanořeny dílčí soubory. Prvním a základním souborem je index.js. Jedná se o spustitelný soubor, který se nachází v kořenové složce serveru (obrázek č. 12). Ten má za úkol nastartovat server a nastavit mu počáteční konfiguraci. V hlavičce se nachází přiřazení konfiguračních knihoven. V případě využívání některé z knihoven, je nutné ji přiřadit do proměnné pomocí funkce *„require(„název-knihovny“)*.

```
const express = require("express");
const app = express();
const { JsonRoute } = require("json-routing");
const port = process.env.PORT || 8080;
app.use(express.json());
```

Obrázek 12: Hlavnička souboru index.js [autor]

5.4.1 Controller

V aplikaci byla zvolena možnost dynamického zpracování požadavků pomocí knihovny *„json-routing“*. Ta využívá konfigurační soubor, ve kterém je definována:

- Cesta požadavku
- Druh http požadavku
- Cesta ke controlleru a metodě, která má být zavolána
- V případě ověření uživatele, cesta k autentizační metodě

Konfigurační soubor je ve formátu JSON a obsahuje veškeré možné požadavky, které budou následně provolávány z front-endu (obrázek č. 13). Po zavolání http požadavku se v controlleru přiřadí metoda modelu, která obsahuje aplikační logiku serveru. Metoda controlleru obsahuje tři parametry *request*, *response* a *next*. Ve vstupním parametru *request* se nachází vstupní data, se kterými následně model pracuje. V případě bezchybného chodu aplikace metoda modelu vrátí výstupní data a jsou nahrány do parametru *response*, která je následně předána zpátky na front-end. Pokud nastane jakákoliv chyba, pak pro hladký chod aplikace se předá do parametru *next* a server zůstane ve fungujícím stavu. Ten se zpracuje a vypíše podobným způsobem jako *response*.

```
"/product/create": {
  "POST": {
    "route": "./controller/product-controller:createProduct",
    "policy": [
      "authorization-middleware:authenticate"
    ]
  }
},
"/product/getProductQuantity": {
  "POST": {
    "route": "./controller/product-controller:getProductQuantity",
    "policy": [
      "authorization-middleware:authenticate"
    ]
  }
},
```

```
async createProduct(req, res, next) {
  await ProductModel.createProduct(req.body) Promise<any>
    .then((dtoOut : any ) => {
      res.json(dtoOut);
    }) Promise<any>
    .catch((e) => next(e));
}
async getProductQuantity(req, res, next) {
  await ProductModel.getProductQuantity(req.body) Promise<any>
    .then((dtoOut : any ) => {
      res.json(dtoOut);
    }) Promise<any>
    .catch((e) => next(e));
}
```

Obrázek 13: Příklad konfiguračního souboru a Controlleru [autor]

5.4.2 Model

V modelu se nachází hlavní aplikační logika a přebírá data od controlleru, které následně zpracovává. Každá metoda obsahuje vstupní parametr *DtoIn* (Data Transfer Object), ve kterém se nachází tělo požadavku odeslané z front-endu (*request.body*). Vstup zvaliduje (popis validace je v kapitole „validace a odchytávání chyb“) a následuje zpracování dat a provolávání databázových příkazů. Každý model má k sobě přiřazený objekt pro přístup k datům (DAO – Data Access Object).

Jelikož je Node.js asynchronní programovací jazyk, metody vrací tzv. „slib“ (promise). Objekt *promise* představuje případné dokončení nebo selhání asynchronní operace a její výslednou hodnotu. V případě, že je zapotřebí ihned po zpracování příkazu pracovat s výsledkem, pak se musí použít příkaz *await*. Ten se používá k rozbalení slibu a předává výsledek jako výraz. Skoro u všech

databázových operací je doporučené využít příkaz *await*, protože je zapotřebí odpověď, zda se operace provedla úspěšně nebo selhala.

```
1 usage   Michal Kořínek
async createProduct(dtoIn) {
  let result = await DtoValidator.validate(dtoIn,  schemaName: "createProductDtoInType");
  if (!result.isValid()) {
    throw new ProductError.CreateProduct.InvalidDtoIn(result.validationResult);
  }
  return await this.ProductDao.insertOne(result.cleanDtoIn);
}
```

Obrázek 14: Příklad metody na vytvoření produktu a zapsání do DBS [autor]

5.4.3 DAO – Database Access Object

DAO je objekt pro přístup k databázi, který odděluje aplikační logiku od jeho mechanismů pro přístup k datům. Z důvodu přehlednosti a znovu použitelnosti se nachází v aplikaci obecný DAO soubor *base-dao.js*, který definuje připojení a obecné CRUD operace pro přistup do databáze. Pro každý model existuje DAO soubor, který je potomek obecného modelu a přistupuje k datům v návaznosti na přiřazený model. Obecně řečeno v modelu se zpracují data a v databázovém objektu se z příslušných dat složí dotaz (query), který ho následně odešle do databáze. Jelikož se práce zabývá porovnáním SQL a NoSQL databází, tak DAO třídy jsou ty, ve kterých se kód podstatně odlišuje. Porovnání a ukázky těchto tříd se nachází v kapitole 5.8.

5.4.4 Validace a odchytávání chyb

V aplikaci se k validaci používá knihovna *validate.js*, která při každé operaci v modelu ověřuje, zda tělo poslané v požadavku je stejné jako v definovaném schématu. Schéma je definované ve formátu JSON (obrázek č. 15).

```
const createProductDtoInType = {
  name: {
    type: "string",
    presence: true,
  },
  "quantity.minimal": {
    type: "number",
    presence: true,
  },
  "quantity.maximal": {
    type: "number",
    presence: true,
  },
};
```

Obrázek 15: Příklad definice schématu pro validační knihovnu [autor]

Tímto schématem se nastaví podmínka pro vytváření produktů, která definuje parametry, které musí být na vstupu. U předešlého příkladu se jedná o tři parametry: název, minimální a maximální množství produktu. Parametr *type* definuje, o jaký datový typ na vstupu se má jednat a parametr *presence,* zda je parametr povinný při zadávání. V případě, že tělo dotazu bude mít parametry navíc, které nejsou pro danou metodu potřeba, pak validační metoda odebere přebývající parametry a na výstupu zůstanou pouze parametry, které jsou definované v příslušném schématu. Na obrázku č. 14 si výstupu z validační metody lze všimnout pod pojmem *result.cleanDtoIn*.

Pokud při vložení dat dojde k validaci, která neskončí úspěšně, pak se příkaz neprovede a výstupem bude nastavena chyba s příslušným kódem a statusem. Je možné vypsat obecnou chybu nebo si definovat chyby vlastního typu (obrázek č. 16). Pro větší přehlednost v kódu byla zvolena možnost definice vlastních typů chyb. Příkladem, jak nadefinovat chybu, je možnost zdědění vlastností typu *Error.* V něm se poté do konstruktoru třídy nahraje kód, zpráva, parametry a status chyby. Nejběžnější je chyba aplikace, která nese status s číslem 400.

Příklad kódu pro definici chyby:

```
const CreateProduct = {
    Michal Kořínek
  InvalidDtoIn: class extends Error {
      1 usage   Michal Kořínek
    constructor(params, cause) {
      super();
      this.code = `invalidDtoIn`;
      this.message = "DtoIn is not valid.";
      this.paramMap = params;
      this.status = 400;
    }
  },
};
```

Obrázek 16: Příklad definice chyby špatného vstupu požadavku [autor]

Obsah parametru *params* je výstup z validace. Pokud například do parametru „minimal“ bude na vstupu na místo čísla odeslán textový řetězec, pak se vyvolá popsaný error. V parametrech erroru se vypíše hláška, která definuje přesné porušení schématu například: „U parametru quantity.minimal byl očekáván typ number, ale na vstupu byl textový řetězec.“ Spojení této validační knihovny s vlastními pomocnými chybami zjednodušuje vývoj a přehlednost kódu aplikace.

5.5 NoSQL databáze v Node.js

Node.js podporuje všechny druhy databází bez ohledu na to, zda se jedná o databázi relační nebo nerelační. NoSQL databáze jako MongoDB se však nejlépe hodí k tomuto programovacímu jazyku. I když Node.js funguje dobře s databází MySQL, perfektní kombinací je MongoDB. Ukládání dat založené na dokumentech je hlavním cílem použití nestrukturované databáze, jako je NoSQL. MongoDB je distribuovaná databáze, která umožňuje ad-hoc dotazy a integraci v reálném čase. MongoDB je open-source a je ideální pro často se měnící data. Další výhodou je možnost ověřování dat na straně serveru.

5.5.1 Návrh NoSQL databáze

Návrh SQL databáze je podstatně jednodušší oproti SQL. Strukturu musí mít uživatel navrhnutou předem, ale pro následné vkládání dokumentů do databáze stačí zavolat operaci *write* a mít nadefinované schéma pro jednodušší práci s daty. Při zavolání operace na zápis do databáze dojde k ověření databázového schématu a následně se vytvoří nový dokument v přiřazené kolekci.

V aplikaci se využívají především tři hlavní databázové objekty. Jedná se o produkt, sklad a objednávku. V případě NoSQL databáze je struktura objektů následující:

Tabulka 3: Struktura databázových objektů v NoSQL

Název databázového objektu	Struktura
Produkt (product)	_id: ObjectID Název: String Minimální množství: Number Maximální množství: Number
Sklad (stock)	_id: ObjectID Název: String Produkty: Array
Objednávka (order)	_id: ObjectID Datum: Date Produkty: Array

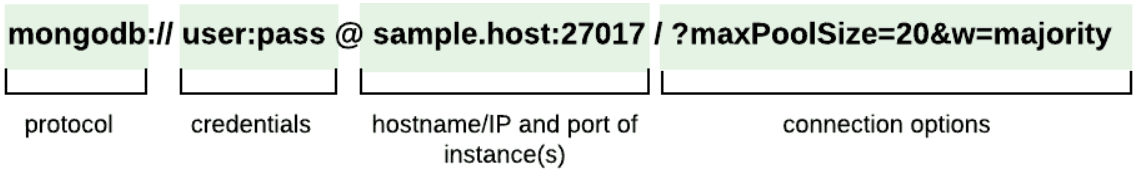
Každý databázový objekt má atribut „*_id:*“, ve kterém se nachází ObjectID daného záznamu. Jedná se o automaticky generovaný identifikátor, který je indexovaný a díky němu dochází k rychlému prohledávání databáze. Oproti relační databázi je rozdíl struktury v objektu Sklad a Objednávka. Mají přiřazené pole produktů, ve kterém je identifikátor produktu a množství na skladě nebo množství, které je potřeba doobjednat. U MongoDB je možné tedy ukládat pole s produkty přímo do dokumentu, zatímco SQL databáze využívají propojovací tabulky s cizími klíči. V této kapitole byly popsány nejdůležitější databázové objekty pro funkci řízeného skladu, dalšími objekty by mohly být: uživatelé, objednávky zákazníků, inventury apod.

5.5.2 Propojení NoSQL s webovou aplikací

V rámci NoSQL části aplikace byla zvolena databáze MongoDB. Jedná se o multiplatformní dokumentovou databázi, která využívá místo tabulek dokumenty ve formátu JSON. Pro integraci a vytváření dat lze využít dynamická schémata, které zjednodušují práci s daty. Pro integrování mongo databáze do projektu je nutné

vložit si název dependence „*mongodb*“ do souboru package.json, díky které po spuštění příkazu *npm install* se přídá do node modulů.

Po nainstalování potřebných modulů, je důležité propojení aplikace s databází. MongoDB využívá k připojení propojovací odkaz tzv. connection URI (obrázek č. 17). Jedná se o sadu pokynů, které ovladač používá k připojení a nasazení databáze a určuje, jak se má aplikace chovat při připojení. [42]



Obrázek 17: Části propojovacího odkazu k MongoDB a jejich popis [42]

Struktura MongoDB se skládá z databáze, kolekce a příslušných dokumentů. Následující kus kódu vyobrazuje obecnou třídu *BaseDao*, která definuje obecné připojení k databázi. Ve hlavičce se nachází propojovací odkaz s názvem databáze. Níže se nachází konstruktor pro napojení na specifickou kolekci, kam se budou vkládat dokumenty. Další třída s názvem *StockDao* rozšiřuje zmíněnou třídu obecnou, ve které se nastavuje již specifická kolekce.

```
const { MongoClient } = require("mongodb");
const uri = "mongodb://127.0.0.1:27017";
const client = new MongoClient(uri);
const DATABASE_NAME = "stock_app";

class BaseDao {
  constructor(collectionName) {
    this.collectionNamme = collectionName;
  }
  async getCollection() {
    await client.connect();
    const database = client.db(DATABASE_NAME);
    return database.collection(this.collectionNamme);
  }
```

```
const BaseDao = require("./base-dao");
const { ObjectId } = require("mongodb");

class StockDao extends BaseDao {
  constructor() {
    super( collectionName: "stocks");
  }
```

Obrázek 18: Propojení databáze a kolekce u MongoDB [autor]

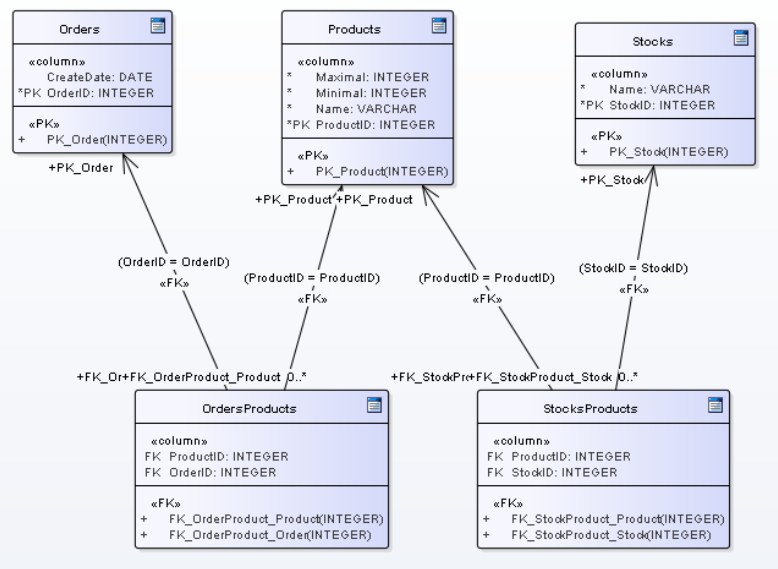
Po nastavení parametrů je aplikace připravena pracovat s databází a veškeré CRUD operace jsou směrovány na příslušný databázový objekt.

5.6 SQL databáze v Node.js

U SQL databáze je dbán větší důraz na přesnou strukturu databázových objektů oproti databázím NoSQL. Zatímco u NoSQL si stačí nadefinovat schéma a následně lze vkládat do databáze, u relační databáze je nutné si definovat tabulky a jednotlivé vztahy mezi nimi.

5.6.1 Návrh SQL databáze

U SQL databáze je zapotřebí mít předem definovanou a vytvořenou strukturu tabulek a vazeb mezi nimi. Pro návrh databázového schématu byl využitý program třetí strany Enterprise Architect (obrázek č. 19), ve kterém si lze namodelovat struktury tabulek (atributy a jejich datové typy) a vazeb (1:1, 1:M, M:N). Enterprise Architect od Sparx systems je modelovací nástroj založený na UML diagramech. V případě, že se jedná o vazbu M:N, je nutné si vytvořit pomocnou propojovací tabulku, ve které jsou propojeny cizí klíče hlavních tabulek.



Obrázek 19: Návrh databáze skladu pomocí Enterprise Architect [autor]

EA nástroj zjednodušuje vytvoření databázové struktury pro MySQL, jelikož po navrhnutí objektů v UML umožňuje vygenerovat script, který po spuštění vytvoří přesnou strukturu do dané databáze. Jelikož se jedná o automaticky generovaný script, tak je vhodné veškeré příkazy scriptu pečlivě zkontrolovat, jelikož hledání chyby v již funkční a používané databázi by mohlo být nákladné a náročné.

5.6.2 Propojení SQL s webovou aplikací

Pro vývoj SQL části aplikace byl zvolen databázový systém MySQL. Dle firmy ORACLE se jedná o celosvětově nejoblíbenější open source relační databázi. Pro implementaci MySQL databáze do aplikace lze využít Node.js modul *mysql*, který po přidání do závislostí (dependencies) a spuštění příkazu *npm install*, nainstaluje potřebné komponenty a knihovny do node modulů.

Na propojení aplikace s databází se využívá funkce *createConnection(config).* Vstupní parametr config obsahuje základní konfiguraci pro připojení do databáze. Parametry pro připojení jsou: host, databáze, uživatel a heslo databázového uživatele. Jelikož v popisované aplikaci je databáze spuštěna na lokálním zařízení, pak je host nastaven na lokální adresu počítače nebo „*localhost*“. U databázového uživatele musí být nastavená práva pomocí příkazu *GRANT*, aby mohl číst a psát z přiřazené databáze specifikované v konfiguraci. Po připojení k databázi lze následně využívat potomky daného DAO a provolávat potřebné dotazy na databázový server.

```
class BaseDao {
  async setConnection() {
    let connection = await mysql.createConnection( config: {
      host: "localhost",
      user: "korinmi2",
      password: "password",
      database: "stock_app",
    });
    try {
      connection.connect();
    } catch (err) {
      console.log(err);
    }
    return connection;
  }
```

Obrázek 20: Propojení SQL databáze s aplikací [autor]

5.7 Funkce a výstupy webové aplikace

Webová aplikace znázorňuje jednoduchý řízený sklad, který se stará o evidenci skladových a produktových zásob, vytváření úkolů na převod zboží z vedlejších skladů na prodejnu a vytváření objednávkových listů. Celá logika je postavena na principu min-max. Jedná se o jednoduchý vzor, který určuje kolik má být minimálně produktů na hlavním skladu (prodejna) a maximálně v celém skladovém systému.

Základ aplikace stojí na CRUD operacích, které se využívají v dalších metodách. Ve většině případů se k databázovým záznamům přistupuje pomocí ID. V případě MongoDB se jedná o ObjectID a MySQL využívá parametr s datovým typem *number* (primární klíč). Následující tabulka obsahuje výpis nejpoužívanějších funkcí webové aplikace.

Tabulka 4: Výpis nejdůležitějších operací aplikace [autor]

Název funkce	Požadavek	Popis	Vstup
createProduct	POST	Přidání produktu do databáze	Název, Min, Max
updateProduct	POST	Úprava existujícího produktu v databázi	Název, Min, Max
findProduct	POST	Výpis produktu/ů z databáze	ID produktu
findProducts	GET	Výpis všech produktů	
findProductQuantity	POST	Výpis množství produktu podle ID	ID produktu
createStock	POST	Vytvoření skladu	Název, Produkty(pole)
addProduct	POST	Přidání produktu do skladu	Produkt (JSON)
listProducts	POST	Výpis produktů ve skladu	ID skladu
updateStock	POST	Úprava existujícího skladu	ID skladu, Produkty(pole)
transferProduct	POST	Přesun produktu mezi sklady	ID skladu1 (z), ID skladu2 (do), ID produktu, množství
refillMainStock	GET	Doplnění produktů na prodejnu	
createOrder	GET	Vytvoření objednávky	
dropCollections	GET	Vymazání databáze	
generateData	POST	Vygenerování testovacích dat	Počet skladů, počet produktů

5.7.1 Vymazání databáze a generování testovacích dat

První důležitou funkcí na porovnání výkonu je vymazání databáze a generování testovacích dat. U vymazání databáze se jedná o poměrně rychlý proces. V případě MongoDB se na každou kolekci zavolá příkaz *dropCollection*, který vymaže veškerá data specifikované kolekce. U MySQL lze využít příkaz *truncate* na vymazání tabulek, ale ponechání jejich struktury. Zde nastává problém s cizími klíči, protože MySQL používá bezpečnostní prvek, který nedovolí smazat záznamy provázané přes cizí

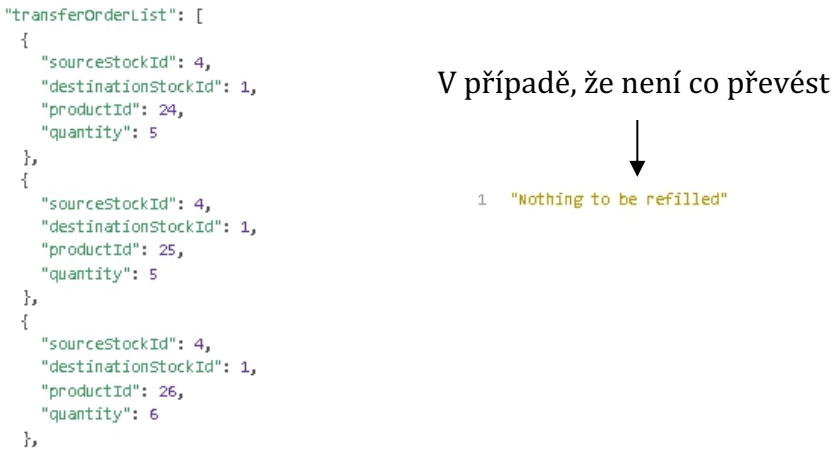
klíče. Proto je nutné vypnout dočasně kontrolu cizích klíčů pomocí příkazu „*SET FOREIGN_KEY_CHECKS = 0*“ a po vymazání ji nastavit zpět na hodnotu 1.

Generování testovacích dat je rozděleno na dvě části – vytvoření produktů a vytvoření skladů. Jak bylo zmíněno, produkt je složený ze čtyř parametrů (ID, název, minimal, maximal) a sklad ze tří (ID, název, produkty). Vstupními parametry funkce na generování dat jsou počet produktů a počet skladů, které chce uživatel nechat vytvořit. V případě generování produktů se vytvoří počet produktů dle vstupního parametru, nastaví se náhodně vygenerovaný min/max a pomocí knihovny *crypto*, se vygeneruje název o pěti náhodných znacích. U vytvoření skladu je funkce podobná s rozdílem toho, že jako první sklad se vytvoří prodejna, která obsahuje veškeré produkty se stavem zásob nula a ostatní sklady jsou vygenerovány náhodně stejným způsobem jako u produktů. Na prodejně jsou ponechány nulové zásoby z důvodu následného otestování doplnění zásob na prodejnu (hlavní sklad).

5.7.2 Doplnění zásob na prodejnu (hlavní sklad)

Funkce převodu zboží na prodejnu funguje na zmíněné logice min-max. Prodejna, jakožto hlavní sklad se vytváří jako první a obsahuje vazbu na veškeré produkty, které jsou evidovány ve skladovém systému. Po zavolání funkce si aplikace vytvoří pomocné pole produktů, do kterého si nahraje z databáze ID produktu a hodnotu *minimal* (minimální stav produktu, který má být na prodejně). Pomocí cyklu se porovnají aktuální stavy na prodejně s hodnotou minimální a v případě, že minimální hodnota je větší než aktuální, tak se začnou prohledávat ostatní sklady. Při každém průchodu (za předpokladu splněných doplnění zboží) se zavolá funkce *transferProduct*, která vyskladní produkt z jednoho skladu do druhého. Jelikož tato funkce naskladní produkty na prodejnu, aniž by byl produkt fyzicky přesunut, vytvoří se úkol tzv. *transferOrderList*, ve kterém je výpis všech produktů s detaily, odkud se zboží bralo a kam se má přesunout. Pokud nastane situace, že není zboží, které by mohlo být převedeno, pak je odpověď požadavku „Nejsou produkty k naskladnění“. Možné zdokonalení této funkce by bylo přidání dalších dvou parametrů. Jedním z nich by mohl být *blockedQuantity*, který by sloužil jako dočasná proměnná, která by vyjadřovala stav „zboží na přesunu“. Dalším možným

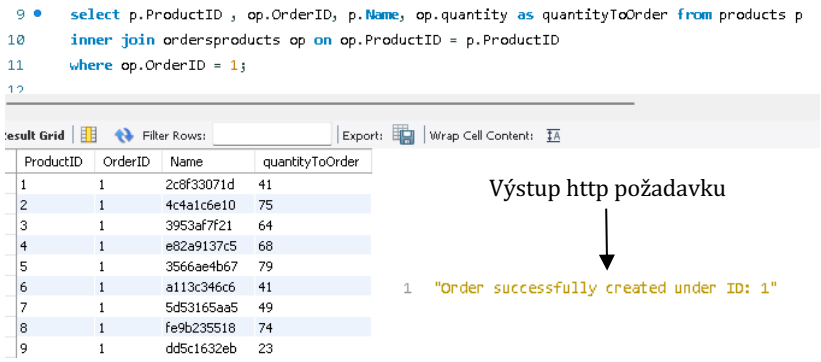
vylepšením by byla priorita skladů. Každý ze skladů by měl nastavenou prioritu, ze kterou by se upřednostňovalo vyskladňování produktů na prodejnu.



Obrázek 21: Výsledek funkce *refillMainStock* [autor]

5.7.3 Vytvoření objednávky

V případě, že na skladech docházejí zásoby, pak je možné vytvoření objednávky vůči dodavateli zboží. Systém si zjistí, jaký má být maximální stav zásob u každého produktu a podle toho prochází a sčítá množství produktů v jednotlivých skladech. Pokud nastane situace, že součet stavů u produktu je menší než nastavené maximum, pak se přidá do objednávky včetně množství, které je potřeba doobjednat. Na konci funkce je se objednávka dokompletuje a vloží do databáze s přidaným aktuálním datumem objednávky. Možným vylepšením by mohlo být generování PDF z vytvořených objednávek. Příkladem knihovny na generování PDF je *pdfkit*.



Obrázek 22: Ověření vygenerované objednávky v MySQL Workbench [autor]

5.8 Porovnání implementovaného řešení

Jak bylo zmíněno aplikace je rozdělena na model a DAO. U implementovaného řešení bylo dbáno na ponechání stejné aplikační logiky modelu pro nerelační i relační část aplikace. Hlavní rozdíly jsou v DAO třídách, které slouží k provolávání a zpracování databázových operací. V následujících kapitolách budou rozebrány kusy kódu databázových tříd týkajících se databázové třídy skladu (*stockDAO*) a porovnání výkonu aplikace.

5.8.1 Implementace MongoDB

Při práci s databází MongoDB se využívají jednoduché příkazy z knihovny *mongodb*, které se provolávají z přiřazené kolekce. Nejpoužívanějšími operacemi jsou *insertOne, replaceOne, updateOne, findOne, find, deleteOne* a pro smazání kolekce příkaz *drop.* Veškeré dokumenty, které se odesílají na vstup databázových příkazů jsou ve formátu JSON. Každá databázová operace má určený vstup a výstup, který se vloží do odpovědi přijatého http požadavku. Při každém zavolání databázové operace se vytvoří připojení do databáze a po odeslání odpovědi se připojení zavře pomocí příkazu *client.close*.

Základní operací je příkaz *insertOne* díky které se vkládá dokument, který je na vstupu do kolekce. Další funkce *replaceOne* nahrazuje první odpovídající dokument v kolekci, který odpovídá filtru. Filtrem se bere *ObjectID* dokumentu, který je vygenerován při vkládání dokumentu do databáze. Příkaz *updateOne* funguje na podobném principu s rozdílem, že nenahrazuje celý dokument, ale upravuje parametry, které jsou zadané do query. V případě aktualizace dokumentu je nutné do query zadat parametr *$set*, který nahrazuje hodnotu pole zadanou ve vstupním dokumentu. U mongoDB je možné nahrát do databáze pole a díky tomu není zapotřebí propojovací dokument v případě vazby M:N. Příkladem využití funkce *updateOne* je na aktualizaci stavu produktů v daném skladu. Na vstupu je dokument, který obsahuje identifikátor *ObjectID* a pole produktu, který je potřeba aktualizovat. V tomto konkrétním případě je na vstupu ještě parametr *options*, kterým se dá například nastavit filtr pro práci s polem.

```
async updateProductQuantity(dtoIn) {
  let filter = { _id: new ObjectId(dtoIn.stockId) };
  const query = {
    $set: { "products.$[product].quantity": dtoIn.quantity },
  };
  const options = {
    arrayFilters: [{ "product.productId": dtoIn.productId }],
    upsert: true,
  };
  return await super.updateOne(filter, query, options);
}
```

Obrázek 23: Příklad metody na aktualizaci množství produktu [autor]

Metoda *findOne* požaduje na vstupu filtr, díky kterému je na výstupu vyfiltrovaný dokument. Pokud se jedná o metodu find, pak na výstupu není jeden dokument, ale pole vyfiltrovaných dokumentů. Pokud není nastavený žádný filtr, pak tato metoda vrátí veškeré dokumenty v kolekci. Příkaz *deleteOne*, požaduje na vstupu opět filtr, pomocí kterého se vymaže vyfiltrovaný dokument z databáze.

5.8.2 Implementace MySQL

Hlavním rozdílem implementace relační databáze je v zadávání query (dotaz). Jelikož knihovna *mysql* nemá předdefinované query, jako to bylo u databáze MongoDB, tak se u každého databázového příkazu musí vytvářet dotaz ručně. Dalším rozdílem je ten, že u relační databáze se nevyužívají dokumenty, ale všechna data se zapisují do tabulek, mezi kterými jsou definované vazby. V případě vazby M:N je nutné vytvoření propojovací tabulky, ve které jsou zapsány cizí klíče propojených tabulek. Pokud je zapotřebí odpověď z databázového dotazu, je nutné si vytvořit *promise* (slib), který po vykonání operace vrátí přímý výsledek nebo chybu. Pokud je potřeba zadávat parametry dynamicky, pak se v dotazu napíše „*?*“, který se definuje při volání dotazu v hranatých závorkách. Obrázek č. 24 zobrazuje příklad zpracování objektu *promise*.

```
async insertStock(name) {
  let connection = await this.setConnection();
  let query = "INSERT INTO stocks (name) VALUES (?)";
  return new Promise( executor: (resolve, reject) => {
    connection.query(query, [name], (err, result) => {
      if (err) {
        reject(err);
      } else {
        resolve(result);
        connection.destroy();
      }
    });
  });
}
```

Obrázek 24: Příklad dotazu na vložení skladu do databáze [autor]

V následující tabulce jsou vypsány hlavní dotazy, které byly využívány v aplikaci s relační databází.

Tabulka 5: Výpis dotazů relační databáze [autor]

Název	Query
insertStock	INSERT INTO stocks (name) VALUES (?)
insertProduct	INSERT INTO stocksproducts (productId, stockId, quantity) VALUES (?, ?, ?)
listProducts	SELECT * FROM products p inner join stocksproducts sp on p.productId = sp.productId WHERE sp.stockId = ?
updateProductQuantity	UPDATE stocksproducts set quantity = ? where productId = ? and stockId = ?
findStock	SELECT * FROM stocks where stockId = ?
updateStock	UPDATE stocks set name = ? where stockId = ?
findProductStock	SELECT * FROM stocksproducts where productId = ? and stockId = ?
findStocksProducts	SELECT * FROM stocksproducts where stockId = ?
findMinimals	select minimal, productID, (select *sum*(quantity) from stocksproducts where stockID = 1 and productID = ?) as mainStockQuantity from products where productID = ?
refillProduct	select * from stocksproducts where StockID != 1 and quantity > 0 and ProductID = ?
findProducts	SELECT * FROM products
findProductQuantity	select *sum*(quantity) as totalQuantity from stocksproducts where productId = ?
insertProduct	INSERT INTO products (name, minimal, maximal) VALUES (?, ?, ?)

5.9 Porovnání výkonu aplikace

Výkon aplikace byl testován na dvou zařízeních s rozdílným hardwarem a operačním systémem. Celá aplikace je testována se spuštěným lokálním databázovým serverem a samotné aplikace ve vývojovém studiu InteliJ IDEA. V průběhu testování nebyla front-endová část aktivní a veškeré výsledky jsou monitorovány od odeslání požadavku po získání odpovědi v aplikaci Insomnia REST client. Porovnávány byly nejnáročnější funkce, které jsou v aplikaci implementovány. Porovnávanými funkcemi jsou: vygenerování dat (vysoká zátěž při zapisování hodnot), smazání databáze, přesun zásob na prodejnu (vysoká zátěž při čtení a aktualizaci hodnot) a vytvoření objednávky. Vygenerovaná data jsou 100 a 1000 produktů pro 10 a 100 skladů.
Dle MongoDB jsou minimální hardwarové požadavky na provoz následující:

- Místo na disku: 10 GB + místo na udržování dat databáze
- RAM paměť: 4 GB
- CPU: 64 - bitová architektura CPU

MySQL uvádí minimální hardwarové požadavky na provoz:

- 2 jádra CPU
- RAM paměť: 2 GB
- I/O subsystém použitelný pro databáze náročné na zápis

Tabulka 6: Minimální požadované místo na disku u databáze MySQL

Platforma	Service manager	Agent minimum
Linux 64x	1,3 GB	800 MB
macOS	1,2 GB	700 MB
Windows 64x	800 MB	500 MB

5.9.1 Výkon aplikace na operačním systému MacOS

Hardwarové specifikace zařízení – MacBook Air early 2015:

- CPU: Intel Core i5, dvoujádrový, frekvence - 1,6 GHz (TB 2,7 GHz)
- RAM: 4 GB, DDR3, frekvence - 1600 MHz
- SSD: rychlost čtení – 1500 MB/s, rychlost zápisu: 700 MB/s

Na zařízení s operačním systémem MacOS se podle výsledků dá říct, že průměrný čas odezvy všech operací je nižší při použití nerelační databáze. Výjimkou jsou operace smazání databáze a převodu zboží na prodejnu. Při velkém množství dat si relační databáze dokázala poradit lépe u funkce převodu zboží na prodejnu.

Tabulka 7: Porovnání výkonu MongoDB a MySQL – MacOS [autor]

Operace	Počet skladů	Počet produktů	Čas operace (ms) MySQL	Čas operace (ms) MongoDB
Clear DB	5	100	5,1	75
Generate data			2640	1390
Refill main stock			3190	1840
Create order			1680	127
Clear DB	50	1000	16	67
Generate data			66600	6010
Refill main stock			235800	242600
Create order			4450	4220

5.9.2 Výkon aplikace na operačním systému Windows

Hardwarové specifikace zařízení – Lenovo Ideapad 3 gaming:

- CPU: Intel Core i5, čtyřjádrový, frekvence – 2,4 GHz (TB 4,2 GHz)
- RAM: 16 GB, DDR4, frekvence - 2666 MHz
- SSD: rychlost čtení – 3500 MB/s, rychlost zápisu: 3000 MB/s

Na zařízení s operačním systémem Windows probíhaly operace mnohem rychleji než na zařízení se slabším hardwarem. S větším výpočetním výkonem si opět aplikace poradila lépe s MongoDB než MySQL. Na systému Windows se vyskytl jeden

velký problém. Jak bylo zmíněno, funkce doplnění zboží na prodejnu používá velké množství databázových operací, které se provolávají rychle po sobě. Dle vyjádření MongoDB a spousty dalších vývojářů, má operační systém Windows problém při práci s virtuální pamětí počítače. V tomto případě dochází k vysokému zatížení a databázový server nemá dostatek paměti na rychlé vykonávání operací. Odpověď dotazu na zařízení s lepším hardwarem se vyšplhala až na devět minut, přičemž zařízení s polovičním výkonem odeslal odpověď za čtyři.

Tabulka 8: Porovnání výkonu MongoDB a MySQL – Windows [autor]

Operace	Počet skladů	Počet produktů	Čas operace (ms) MySQL	Čas operace (ms) MongoDB
Clear DB	5	100	2	20
Generate data			936	545
Refill main stock			1670	1530
Create order			364	76
Clear DB	50	1000	3	21
Generate data			35400	4210
Refill main stock			169200	555600
Create order			2800	1970

6 Shrnutí výsledků

6.1 Dosažené výsledky

Pomocí programovacího jazyka Node.js, databázových technologií MongoDB a MySQL se úspěšně podařil vývoj obou variant aplikací. V navrhnuté aplikaci jsou implementovány veškeré CRUD operace pro snadnou správu skladů, produktů a objednávek. Hlavními výhodami aplikace, je automatická správa skladových zásob mezi prodejnou a vedlejšími sklady. Systém sám automaticky vyhodnotí chybějící zásoby na hlavním skladu (prodejna) a bez manuálního přesouvání automaticky vygeneruje úkol přesunu zásob nedostatkových produktů. Další výhodnou je automatické generování objednávek vůči dodavateli. Díky aplikaci uživatel nemusí ručně procházet skladové zásoby, ale dle zadané maximální zásoby, která je specifikována v parametru produktu, systém automaticky vygeneruje objednávku s množstvím, které je zapotřebí objednat. Pomocí aplikace třetí strany Insomnia REST Client, se podařilo precizně změřit a porovnat výkon aplikace na obou zařízeních.

6.2 Možné vylepšení systému

Jedno z velkých rozšíření systému by bylo vytvoření android aplikace kompatibilní se čtečkou čárových kódu. V databázových entitách by se přidal identifikátor EAN kód (identifikátor zboží – čárový kód), pomocí kterého by docházelo k jednoduché manipulaci se zbožím. Uživatel by nemusel složitě opisovat kódy produktů, ale načítal a prováděl by veškeré požadavky skrz čtečku čárových kódu připojenou k serveru.

7 Závěr a doporučení

Práce je rozdělena na teoretickou a praktickou část. Teorie se skládá z popisu databáze a webové aplikace. Databázové technologie popisuje kapitola č. 3. Nejdůležitějšími podkapitolami jsou 3.3 a 3.4, které se zabývají podrobnostmi a rozdíly mezi relační a nerelační databází. V následující kapitole č. 4 je rozebrán vývoj a architektura webových aplikací. Další kapitolou je praktická část s označením č. 5. Ta se zabývá podrobným návrhem back-endu webové aplikace, nerelační a relační databáze. Webová aplikace plní úlohu řízeného skladu, která se stará o udržování zásob na skladech a prodejně. Aplikace byla vyvíjena z důvodu porovnání výkonu relační a nerelační databáze v závislosti na hardwarové prostředky. Vytvořený model obsahoval jednoduchou strukturu databáze. V tomto případě je vhodnější využití nerelační databáze pro její rychlost. Jak si lze všimnout z výsledků, pokud je provedena náročnější operace, která vyžaduje velké množství připojování, pak je lepší využít databázi relační. Dalším kritériem jsou vztahy mezi jednotlivými databázovými objekty. Pokud by databázové schéma obsahovalo mnohem více vazeb, pak by relační databáze měla vetší výkon. U navržené webové aplikace, která se týká řízeného skladu, je z pohledu vývoje pohodlnější využití relační databáze, díky práci s propojovacími tabulkami. U MongoDB se pro jednotlivé operace musí procházet pole produktů, které se hůře upravuje na aplikační úrovni. Jelikož se může vyskytnout problém s virtualizací paměti u některých verzí operačního systému Windows, pak je doporučené nasazování aplikací na servery, které využívají operační systém Linux.

8 Seznam použité literatury

[1] ORACLE. *What is database?* [online]. 13. leden 2023. Dostupné z: https://www.oracle.com/database/what-is-database/

[2] TRUICA, Ciprian-Octavian, Florin RADULESCU, Alexandru BOICEA a Ion BUCUR. Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. In: *2015 20th International Conference on Control Systems and Computer Science (CSCS)*: *2015 20th International Conference on Control Systems and Computer Science* [online]. Bucharest, Romania: IEEE, 2015, s. 191–196 [vid. 2023-01-13]. ISBN 978-1-4799-1780-8. Dostupné z: doi:10.1109/CSCS.2015.32

[3] IBM CORPORATION. *What is a database management system?* [online]. 13. leden 2023. Dostupné z: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-what-is-database-management-system

[4] RAMAKRISHNAN, Raghu a Johannes GEHRKE. *Database management systems*. 3rd ed. Boston: McGraw-Hill, 2003. ISBN 978-0-07-246563-1.

[5] *Three Schema Architecture of DBMS* [online]. 13. leden 2023. Dostupné z: https://www.tutorialandexample.com/three-schema-architecture-of-dbms

[6] BORGIDA, Alexander a John MYLOPOULOS. Conceptual Schema Design. In: Ling LIU a M. Tamer ÖZSU, ed. *Encyclopedia of Database Systems* [online]. Boston, MA: Springer US, 2009 [vid. 2023-02-07], s. 438–442. ISBN 978-0-387-35544-3. Dostupné z: doi:10.1007/978-0-387-39940-9_642

[7] MAGDALENA IACOB, Nicoleta a Mirela LILIANA MOISE. Centralized vs. Distributed Databases. Case Study [online]. 2015. ISSN 2457-5836. Dostupné z: https://www.academia.edu/41556717/Centralized_vs_Distributed_Databases_Case_Study

[8] HARRINGTON, Jan L. *Relational database design and implementation: clearly explained*. 4th edition. Cambridge, MA: Elsevier, 2016. ISBN 978-0-12-804399-8.

[9] What is a cloud database? *Oracle* [online]. 7. únor 2023. Dostupné z: https://www.oracle.com/database/what-is-a-cloud-database/

[10] What Are the Different Types of Databases? *Indeed* [online]. 25. únor 2020. Dostupné z: https://www.indeed.com/career-advice/career-development/types-of-databases

[11] What is a Relational Database (RDBMS)? *Oracle* [online]. 16. únor 2023. Dostupné z: https://www.oracle.com/database/what-is-a-relational-database/

[12] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM* [online]. 1970, **13**(6), 377–387. ISSN 0001-0782, 1557-7317. Dostupné z: doi:10.1145/362384.362685

[13] IBM100 - Relational Database. *IBM* [online]. 23. únor 2023. Dostupné z: https://www.ibm.com/ibm/history/ibm100/us/en/icons/reldb/

[14] What is a relational database? | IBM. *IBM* [online]. 24. únor 2023. Dostupné z: https://www.ibm.com/topics/relational-databases

[15] STOREY, Veda C. Relational database design based on the entity-relationship model. *Data & Knowledge Engineering* [online]. 1991, **7**(1), 47–83. ISSN 0169023X. Dostupné z: doi:10.1016/0169-023X(91)90033-T

[16] ACID Properties In DBMS Explained. *MongoDB* [online]. 22. únor 2023. Dostupné z: https://www.mongodb.com/basics/acid-transactions

[17] ABRAMOVA, Veronika a Jorge BERNARDINO. NoSQL databases: MongoDB vs cassandra. In: *C3S2E13: International C* Conference on Computer Science & Software Engineering*: *Proceedings of the International C* Conference on Computer Science and Software Engineering* [online]. Porto Portugal: ACM, 2013, s. 14–22 [vid. 2023-04-18]. ISBN 978-1-4503-1976-8. Dostupné z: doi:10.1145/2494444.2494447

[18] ATCHARIYACHANVANICH, Kanokwan, Srinual NALINTIPPAYAWONG a Tanasab PERMPOOL. Development of a MySQL Sandbox for processing SQL statements: Case of DML and DDL statements. In: *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*: *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)* [online]. NakhonSiThammarat, Thailand: IEEE, 2017, s. 1–6 [vid. 2023-04-18]. ISBN 978-1-5090-4834-2. Dostupné z: doi:10.1109/JCSSE.2017.8025930

[19] GUPTA, Ravinder. Capturing DDL Changes. In: Ravinder GUPTA*Mastering Oracle GoldenGate* [online]. Berkeley, CA: Apress, 2016 [vid. 2023-04-18], s. 83–102. ISBN 978-1-4842-2300-0. Dostupné z: doi:10.1007/978-1-4842-2301-7_6

[20] MySQL SQL. *w3schools* [online]. 21. únor 2023. Dostupné z: https://www.w3schools.com/mysql/mysql_sql.asp

[21] NISHTHA, Jatana, Puri SAHIL, Ahuja MEHAK, Kathuria ISHITA a Gosain DISHANT. A Survey and Comparison of Relational and Non-Relational Database. 2012. ISSN 2278-0181.

[22] What is a key-value database? | MongoDB. *MongoDB* [online]. 24. únor 2023. Dostupné z: https://www.mongodb.com/databases/key-value-database

[23] Document Database | Redis. *Redis* [online]. 24. únor 2023. Dostupné z: https://redis.com/nosql/document-databases/

[24] Document Database - NoSQL | MongoDB. *MongoDB* [online]. 24. únor 2023. Dostupné z: https://www.mongodb.com/document-databases

[25] Graph Database Defined. *Oracle* [online]. 7. únor 2023. Dostupné z: https://www.oracle.com/autonomous-database/what-is-graph-database/

[26] What Is an Object-Oriented Database? *MongoDB* [online]. 7. únor 2023. Dostupné z: https://www.mongodb.com/databases/what-is-an-object-oriented-database

[27] MOLINA-RÍOS, Jimmy a Nieves PEDREIRA-SOUTO. Comparison of development methodologies in web applications. *Information and Software Technology* [online]. 2020, **119**, 106238. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2019.106238

[28] What is web application and its Benefits. *TechTarget* [online]. 25. únor 2023. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app

[29] WATERS, Bret. Software as a service: A look at the customer benefits. *Journal of Digital Asset Management* [online]. 2005, **1**(1), 32–39. ISSN 1743-6559. Dostupné z: doi:10.1057/palgrave.dam.3640007

[30] Web aplication | What is a Web Application? - Javatpoint. *Javatpoint* [online]. 25. únor 2023. Dostupné z: https://www.javatpoint.com/web-application

[31] HTML History. *W3C* [online]. 26. únor 2023. Dostupné z: https://www.w3schools.in/html/history

[32] RAGGETT, Dave, Jenny LAM a Ian ALEXANDER. *HTML 3: electronic publishing on the World Wide Web*. Harlow, England ; Reading, Mass: Addison-Wesley, 1996. ISBN 978-0-201-87693-2.

[33] DUCKETT, Jon. *HTML & CSS: design and build websites*. Indianapolis, Indiana: John Wiley & Sons Inc, 2014. ISBN 978-1-118-87164-5.

[34] VAUGHAN-NICHOLS, Steven J. Will HTML 5 Restandardize the Web? *Computer* [online]. 2010, **43**(4), 13–15. ISSN 0018-9162. Dostupné z: doi:10.1109/MC.2010.119

[35] SUN, Kwangwon a Sukyoung RYU. Analysis of JavaScript Programs: Challenges and Research Trends. *ACM Computing Surveys* [online]. 2018, **50**(4), 1–34. ISSN 0360-0300, 1557-7341. Dostupné z: doi:10.1145/3106741

[36] JAVEED, Arshad. Performance Optimization Techniques for ReactJS. In: *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*: *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)* [online]. Coimbatore, India: IEEE, 2019, s. 1–5 [vid. 2023-02-26]. ISBN 978-1-5386-8158-9. Dostupné z: doi:10.1109/ICECCT.2019.8869134

[37] LEI, Kai, Yining MA a Zhi TAN. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. In: *2014 IEEE 17th International Conference on Computational Science and Engineering (CSE)*: *2014 IEEE 17th International Conference on Computational Science and Engineering* [online]. Chengdu, China: IEEE, 2014, s. 661–668 [vid. 2023-02-26]. ISBN 978-1-4799-7981-3. Dostupné z: doi:10.1109/CSE.2014.142

[38] CHITRA, Lakshmi Prasanna a Ravikanth SATAPATHY. Performance comparison and evaluation of Node.js and traditional web server (IIS). In: *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*: *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)* [online]. Chennai: IEEE, 2017, s. 1–4 [vid. 2023-02-26]. ISBN 978-1-5090-3378-2. Dostupné z: doi:10.1109/ICAMMAET.2017.8186633

[39] KHAN, Mohammad Ayoub, Sanjay GAIROLA, Bholā JHĀ a Pushkar PRAVEEN, ed. *SMART COMPUTING proceedings of the 1st international conference on*. S.l.: CRC PRESS, 2021. ISBN 978-1-00-316748-8.

[40] Top 5 Alternatives to NodeJS. *EpamAnywhere* [online]. Dostupné z: https://anywhere.epam.com/business/top-nodejs-alternatives

[41] POP, Dragos-Paul a Adam ALTAR. Designing an MVC Model for Rapid Web Application Development. *Procedia Engineering* [online]. 2014, **69**, 1172–1179. ISSN 18777058. Dostupné z: doi:10.1016/j.proeng.2014.03.106

[42] Connection Guide - Node.js. *MongoDB* [online]. 1. duben 2023. Dostupné z: https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/connect/#std-label-node-connect-to-mongodb

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Michal Kořínek**
Studium:	I2100066
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Porovnání možností databází SQL a NoSQL pro využití ve webových aplikacích**
Název diplomové práce AJ:	Comparison of the possibilities of SQL and NoSQL databases

Cíl, metody, literatura, předpoklady:

Cílem diplomové práce je porovnání výhod a nevýhod databází SQL a NoSQL pro využití ve webových aplikacích. Důležitými kritérii pro porovnání budou především: implementace, způsob ukládaní informací a jejich použitelnost, náročnost na hardwarové prostředky. Dále autor porovná vhodnost využití dané databáze v konkrétní webové aplikaci.

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. RNDr. Petra Poulová, Ph.D.

Datum zadání závěrečné práce: 26.1.2021