ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Yauhen Asanovich

VÝBĚR VHODNÉ DATABÁZE PRO APLIKACI ZDRAVOTNÍ PÉČE PRO CIZINCE

Bakalářská práce

2021

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mi poskytli podklady pro vypracování této práce. Zvláštně pak děkuji Ing. Janě Kalikové a Ing. Janovi Krčálovi za odborné vedení a konzultování bakalářské práce a za rady, které mi poskytovali po celou dobu mého studia a dále bych chtěl poděkovat Ing. Martinovi Šrotýřovi za příležitost projít předmět a zapojit mě do oblasti databází a moderních technologií. V neposlední řadě je moji milou povinností poděkovat svým rodičům za morální a materiální podporu, které se mi dostávalo po celou dobu studia, svým blízkým přátelům za podporu a pomoc v těžkých chvílích, a také své kočce, že mi nepřekážela při psaní této práce.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na ČVUT v Praze Fakultě dopravní.

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

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 09.08.2021

---

podpis

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta dopravní

VÝBĚR VHODNÉ DATABÁZE PRO APLIKACI ZDRAVOTNÍ PÉČE PRO CIZINCE

bakalářská práce

srpen 2021

Yauhen Asanovich

ABSTRAKT

Předmětem bakalářské práce „Výběr vhodné databáze pro aplikaci Zdravotní péče pro cizince“ je analyzovat a povrchně popsat proces vytváření webových aplikací, zvážit stávající typy databázi, včetně relačních a nerelačních, analyzovat více kritérií a následně zvolit vhodnou databázi, implementovat databázi a vytvořit aplikaci.

Klíčová slova: databáze, SQL, NoSQL, webová aplikace, systém řízení báze dat.

ABSTRACT

The subject of the bachelor thesis „Choosing of a Database for the Application of Health Care for Foreigners” is analysis and superficial description of the process of creating web applications, consideration of existing database types, including relational and non-relational, analysis of multiple criteria and subsequent selection of a suitable database, implementing the database and creating the application.

Key words: database, SQL, NoSQL, web application, database management system.

Obsah

1 Seznam použitých zkratek

HTTP	HyperText Transfer Protocol
URL	Jednotný lokátor zdroje (z angličtiny Uniform Resource Locator)
CSS	Kaskádové styly
JS	Multiparadigmový programovací jazyk JavaScript
HTML	Hyper Text Markup Language
DB	Databáze
DNS	Systém doménových jmen (z angličtiny Domain Name System)
Windows NT	Windows New Technology
IIS	Internet Information Server
OS	Operační systém
API	Aplikační Programovací Rozhraní (z angličtiny Application Programming Interface)
ODBC	Open Database Connectivity
CSV	Hodnoty oddělené čárkami (z angličtiny Comma-separated values)
DBMS	Systém řízení báze dat (z angličtiny Database Management System)
RDBMS	Relační systém řízení báze dat (z angličtiny Relational Database Management System)
RAM	Operační, vnitřní neboli hlavní paměť (z angličtiny Random Access Memory)
GIS	Geografický informační systém
PNG	Přenosná síťová grafika (z angličtiny Portable Network Graphics)
JPEG	Joint Photographic Experts Group
WebGL	Knihovna webové grafiky (z angličtiny Web Graphics Library)
AWS	Webové služby Amazon (z angličtiny Amazon Web Services)
SSR	Skriptování na straně serveru (z angličtiny Server-Side Rendering)
HTTPS	HyperText Transfer Protocol Secure
CMS	Systém pro správu obsahu (z angličtiny Content Management System)
REST	Representational State Transfer
SQL	Structured Query Language
O2M	Relace One-to-many (1:N)
M2O	Relace Many-to-one (N:1)
M2M	Relace Many-to-many (N:M)
IDE	Vývojové prostředí (z angličtiny Integrated development environment)
SSH	Secure Shell

2 Úvod

Myšlenka vytvořit užitečnou webovou aplikaci s využitím moderních technologií byla navržena kvůli nedostatku alternativních příležitostí pro cizí občany. Cílem této bakalářské práce je vyvinout webovou aplikaci pomocí databáze, jejíž výběr je založen na řadě kritérií, která jsou přijatelná pro fungování aplikace. Existuje mnoho kombinací technologií pro vývoj webových aplikací. Protože je tato práce založena na znalostech, které autor získal v průběhu studia na ČVUT v Praze Fakultě dopravní, a na znalostech získaných individuálním vzděláváním, budou v této práci požité pouze moderní vývojové technologie a technologie známé autorovi. Důležitým kritériem navrhované práce je absence finančních prostředků na projekt (projekt je nekomerční), proto budou brány v úvahu pouze bezplatné technologie nebo technologie s otevřeným zdrojovým kódem.

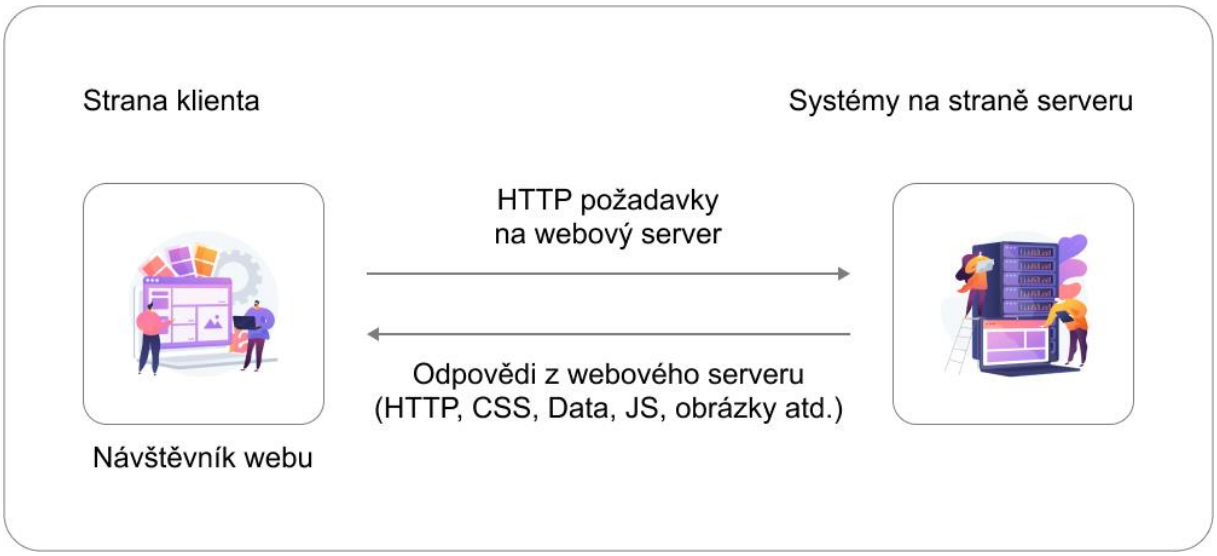
Existuje několik typů databází, které se od sebe liší svými vlastnostmi, strukturou a složitostí práce s nimi. Tato práce proto zváží nejpopulárnější a nejmodernější typy databází, které jsou vhodné pro budoucí webovou aplikaci. Následně bude na základě kritérií popsaných v této práci proveden výběr nejvhodnější databáze. Následná definice struktury databáze, návrh tabulek a vztahů, který vychází z funkčních vlastností aplikace. Vytvoření databáze bude provedeno lokálně na zařízení autorovi a v budoucnu bude přesunuto na webový server, aby byla databáze k dispozici novým uživatelům webové aplikace.

Hlavní ustanovení aplikace vyžadují, aby databáze uchovávala informace o uživatelích (jméno a příjmení, číslo pasu, sjednané pojištěni), o pojišťovnách (název společnosti, adresa, na kterou se klient může obrátit, vztah společností k nemocnicím), o nemocnicích (kteří lékaři jsou v konkrétní nemocnici k dispozici, otevírací doba a adresa). Protože aplikace bude mít uživatelsky přívětivé funkce, databáze by měla rychle reagovat na požadavky ze serveru a vrátit datovou strukturu, kterou lze snadno zpracovat a zobrazit uživateli.

V rámci této bakalářské práce bude brána v úvahu pouze první iterace aplikace. To znamená, že funkčnost aplikace bude omezená, poté je připravena na další rozšiřování nejen možností samotné aplikace, ale také na změny a vylepšení v databázi. První iterace zahrnuje vytvoření webového rozhraní, přidání dotazů do databázi nemocnic a uživatelů. Ve druhé iteraci se plánuje vytvoření systému pro rezervaci návštěv lékaře a sepsání pojistné smlouvy online. A do třetice — navigace na mapě.

3 Základní postupy vývoje aplikací

V současné době webové aplikace mají velký rozvoj v různých oblastech činnosti společnosti. Provoz webové aplikace se provádí prostřednictvím technologie klient-server, kde je prohlížeč klientem a webový server slouží jako server. Nejprve uvedeme obecné schéma webových aplikací. Klient, který přistupuje k webovému prohlížeči, odešle požadavek HTTP na konkrétní adresu URL, která označuje nějaký dynamický zdroj, a to samotnou webovou aplikaci.



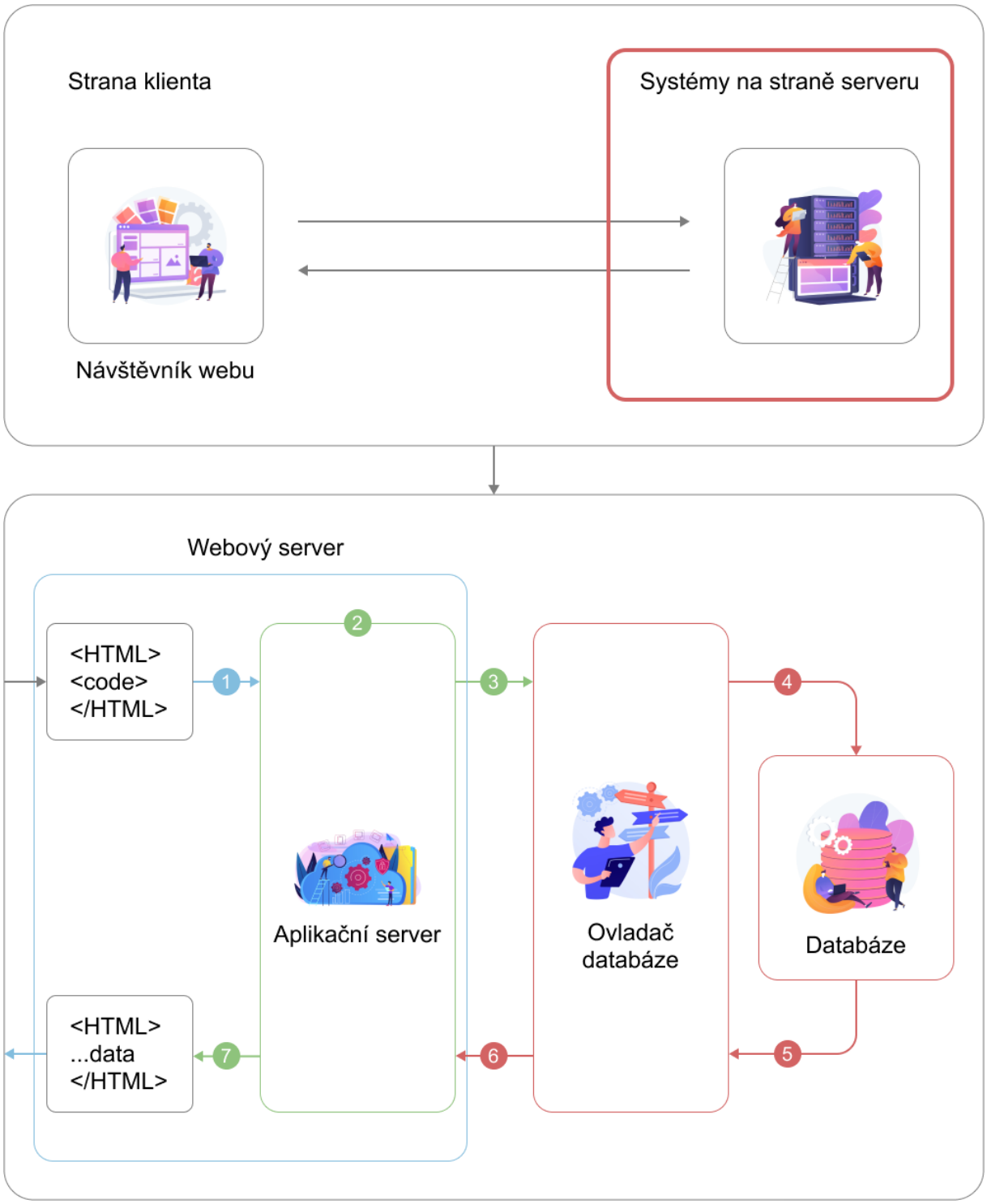
Obrázek 1. Struktura webových aplikací [zdroj autor][1]

V procesu zpracování požadavku uživatele vytvoří webová aplikace odpověď na základě provedení kódu programu spuštěného na straně serveru, webového formuláře, stránky HTML a dalšího obsahu, včetně grafických souborů. Výsledkem je, jak již bylo zmíněno, vygenerování stránky HTML, která je odeslána klientovi. Ukazuje se, že výsledek webové aplikace je totožný s výsledkem požadavku na tradiční web, avšak na rozdíl od ní webová aplikace generuje HTML kód v závislosti na požadavku uživatele a nepřenáší jej jednoduše klientovi ve forma, ve které je tento kód uložen v souboru na straně serveru. To znamená, že webová aplikace dynamicky generuje odpověď pomocí spustitelného kódu — takzvané spustitelné části.[2]

Vzhledem k přítomnosti spustitelné části jsou webové aplikace schopny provádět téměř stejné operace jako běžné aplikace Windows, s jediným omezením, že se kód spouští na serveru, prohlížeč funguje jako rozhraní systému a prostředí prostřednictvím kterého probíhá výměna dat, — internet. Mezi nejběžnější operace prováděné webovou aplikací patří:

- příjem dat od uživatele a jejich uložení na serveru;

- provádění různých akcí na vyžádání uživatele: extrakce dat z databáze (DB), přidávání, mazání, změna dat v DB, provádění složitých výpočtů;
- ověření uživatele a zobrazení rozhraní systému, které odpovídá danému uživateli;
- zobrazení neustále se měnících provozních informací atd.



Obrázek 2. Schéma serverové stránky webové aplikace [zdroj autor][1]

Nyní se podívejme na schéma serverového systému oddělením pravé části obrázku 1:[2]

1. Webový server najde požadovanou stránku a odešle ji na aplikační server.
2. Aplikační server vyhledá na stránce pokyny a připraví ji.
3. Aplikační server odešle požadavek do ovladače databáze.
4. Ovladač provede dotaz na databázi.
5. Sada záznamů je vrácena ovladači.
6. Ovladač předá sadu záznamů aplikačnímu serveru.
7. Aplikační server vloží data na stránku a odešle stránku na webový server.

Všechny operace prováděné webovou aplikací budou přítomny v budoucí aplikaci Health Care for Foreigners.

V současné době existuje mnoho technologií, které implementují logiku webových aplikací na straně serveru.

4 Popis technologií a kritéria pro jejich výběr

K vytvoření aplikace je nutné poskytnout systému všechny technologie nezbytné pro jeho fungování:

- webový server;
- aplikační server;
- ovladač databáze;
- databáze.

Volba názvu domény, připojení hostingu, registrace záznamů DNS a nastavení poštovního serveru nejsou součástí projektu, ale jsou povinné pro doručování plnohodnotné aplikace koncovému uživateli. Vývoj aplikací a připojení k serveru se bude provádět lokálně na výpočetním zařízení vývojáře.

4.1 Webový server

Webový server je server, který přijímá požadavky HTTP od klientů, obvykle webových prohlížečů, a vydává na ně odpovědi HTTP, obvykle spolu se stránkou HTML. Webový server označuje jak software, který funguje jako webový server, tak samotný počítač, na kterém software běží.[3]

Klient, kterým je obvykle webový prohlížeč, odesílá na webový server požadavky na prostředky určené pomocí adres URL. Zdroje jsou stránky HTML, obrázky, soubory, mediální proudy nebo jiná data, která klient potřebuje. V reakci na to webový server odešle požadovaná data klientovi. Tato výměna probíhá přes protokol HTTP.

Název	Výskyt	Autor	Šíření	Open source	Vlastnosti
Apache HTTP Server	1995	Apache Software Foundation	Zdarma	Ano	• Důraz na spolehlivost a flexibilitu • Možnost připojení poštovního serveru
IIS	1995	Microsoft	Součást Windows NT	Ne	• Je součástí balíčku IIS • Podporuje soubor technologií .NET
nginx	2004	Igor Sysoev	Zdarma	Ano	• Byl vyvinut pro náročné velké zatížení serverů • Zahrnuje poštovní proxy server

Tabulka 1. Seznám nejpopulárnějších webových serverů [zdroj autor]

Název	Windows	Mac OS X	BSD	Linux	Solaris	VMS
Apache HTTP Server	Ano	Ano	Ano	Ano	Ano	Ano
IIS	Ano	Ne	Ne	Ne	Ne	Ne
nginx	Ano	Ano	Ano	Ano	Ano	Ne

Tabulka 2. Podpora OS webovými servery [zdroj autor]

4.2 Aplikační server

Softwarová platforma (framework) určená pro efektivní provádění procedur (programů, skriptů), na nichž jsou aplikace postaveny. Aplikační server funguje jako kolekce komponent dostupných vývojáři softwaru prostřednictvím API (Application Programming Interface) definovaného samotnou platformou.[4]

U webových aplikací je hlavní úlohou serverových komponent poskytovat dynamické stránky. Moderní aplikační servery však také zahrnují podporu pro shlukování, zvýšenou odolnost proti chybám, vyvažování zátěže, což vývojářům umožňuje soustředit se pouze na implementaci obchodní logiky.[4]

Název	Autor	Poslední vydání	Open source
Java			
Apache MINA	Apache Software Foundation	02.06.2019	+
GlassFish	Eclipse Foundation	28.01.2019	–
Jetty	Eclipse Foundation	14.08.2019	–
Netty	Netty Project Community	14.01.2016	+
JavaScript			
Wakanda	Laurent Ribardière	29.04.2019	+
Node.js	Ryan Dahl	23.06.2021	+
Python			
Google App Engine	Google	07.04.2008	–
Tornado	FriendsFeed	30.10.2020	+

Tabulka 3. Seznam bezplatných aplikačních serverů s možností rozšíření [zdroj autor]

V případě aplikačního serveru Java se aplikační server chová jako rozšířený virtuální stroj pro spouštění aplikací, který transparentně spravuje připojení k databázi na jedné straně a připojení webového klienta na straně druhé.

4.2.1 Výhody aplikačních serverů

- **Integrita dat a kódu**
  Přiřazením obchodní logiky jednomu serveru nebo malému počtu serverů jsou aktualizace aplikací a vylepšení zaručeny všem uživatelům. Neexistuje žádné riziko, že stará verze aplikace bude k datům přistupovat nebo je bude moci změnit starým nekompatibilním způsobem.
- **Centralizovaná konfigurace a správa**
  Změny nastavení aplikace, například změnu nastavení databázového serveru nebo systému, lze provádět centrálně.
- **Bezpečnost**
  Aplikační server funguje jako centrální bod, ze kterého mohou poskytovatelé služeb řídit přístup k datům a částem samotných aplikací, což je považováno za bezpečnostní výhodu. Jeho přítomnost umožňuje přesunout odpovědnost za ověřování z potenciálně nezabezpečené úrovně klienta na úroveň aplikačního serveru a zároveň skrýt úroveň databáze.
- **Podpora transakcí**
  Transakce je jednotka aktivity, během níž lze atomově (jako nedělitelnou jednotku práce) provádět velký počet změn zdrojů (v jednom nebo různých zdrojích). Koncoví uživatelé mohou těžit ze standardizovaného chování systému, zkrácení doby vývoje a snížení nákladů. Zatímco aplikační server provádí hodně požadovaného generování kódu, vývojáři se mohou soustředit na obchodní logiku.

4.3 Ovladač databáze

Open Database Connectivity (ODBC) je standardní aplikační programovací rozhraní (API) pro přístup k databázovým řídicím systémům (DBMS). Vývojáři ODBC se snažili, aby byl nezávislý na databázových systémech a operačních systémech. Aplikace napsané pomocí ODBC, může být přesunuta na jiné platformy, a to jak na straně klienta, tak na straně serveru, s drobnými změnami v kódu pro přístup k datům.[5]

ODBC poskytuje nezávislost DBMS pomocí ovladače ODBC jako úroveň vysílání mezi aplikací a DBMS. Aplikace využívá funkce ODBC prostřednictvím Správce ovladačů ODBC, se kterým je spojena, a ovladač odešle požadavek na DBMS. Ovladač ODBC lze chápat jako analogové

ovladače tiskárny nebo jiného ovladače, který poskytuje standardní sadu funkcí pro využití aplikace a realizuje specifické pro databázový stroj funkce. Aplikace, která může používat ODBC, se nazývá "ODBC kompatibilní". Jakákoli aplikace kompatibilní s ODBC může přistupovat k jakémukoli DBMS, pro které je ovladač nainstalován. Existují ovladače pro všechny hlavní DBMS, mnoho dalších zdrojů dat, jako jsou systémy adresovatelné knih a Microsoft Excel, a to i textových souborů nebo souborů s hodnotami oddělenými čárkou (CSV).[5]

4.4 Databáze

Databáze je sbírka dat uložených v souladu s datovým schématem, jejichž manipulace je prováděna v souladu s pravidly pro modelování dat. Existuje velké množství databází, které se liší podle různých kritérií. [6]

4.4.1 Klasifikace databází

Existuje obrovské množství druhů databází, které se liší podle různých kritérií. Například v Encyklopedii databázových technologií jsou určeny více než 50 typů DB. Základní klasifikace jsou uvedeny níže.

Klasifikace podle datového modelu:[7]

- Hierarchická databáze
- Síťová databáze
- Relační databáze
- Objektová databáze
- Funkční databáze

Klasifikace podle prostředí trvalého skladování:[7]

- v sekundární paměti (místem ukládaní dat je energeticky nezávislá periferní paměť — obvykle pevný disk);
- DBMS ukládá do paměti RAM pouze mezipaměť a data pro aktuální zpracování;
- v paměti RAM, kde jsou všechna data ve fázi provedení;
- v terciární paměti (data se ukládají do velkokapacitního paměťového zařízení, které je oddělitelné od server);
- v sekundární paměti serveru je uložen pouze datový adresář terciární paměti, mezipaměť souborů a data pro aktuální zpracování; stahování stejných dat samo o sobě vyžaduje zvláštní postup.

Klasifikace podle stupně distribuce:[7]

- centralizované (databáze je plně podporována na jednom počítači);
- distribuované (komponenty databáze jsou umístěny v různých uzlech počítačové sítě podle jakéhokoliv kritéria).

4.5 Systém řízení báze dat

Souhrn softwarových a jazykových nástrojů obecného nebo zvláštního určení, které zajišťují správu vytváření a používání databází.

DBMS je sada programů, které umožňují vytvářet databázi a manipulovat s daty (vkládat, aktualizovat, mazat a vybírat). Systém zajišťuje bezpečnost, spolehlivost ukládání a integritu dat a poskytuje nástroje pro správu databáze.[8]

Hlavní funkce systému řízení báze dat:

- Správa dat v externí paměti (poskytování potřebných externích paměťových struktur pro ukládání dat přímo do databáze a pro úřední účely).
- K dispozici uživateli adresář nebo datový slovník s popisem metadat.
- Podpora transakcí a paralelismu.
- Spravování schránkami pamětí RAM.
- Řízení transakcí.
- Prostředky pro obnovu databáze v případě poškození.
- Podpora autorizace přístupu a aktualizace dat.
- Přístup k podpoře ze vzdálených míst.
- Použití omezení, aby byla zajištěna shoda dat v databázi s určitými pravidly.
- Podpora jazyků databáze.

Složení DBMS:

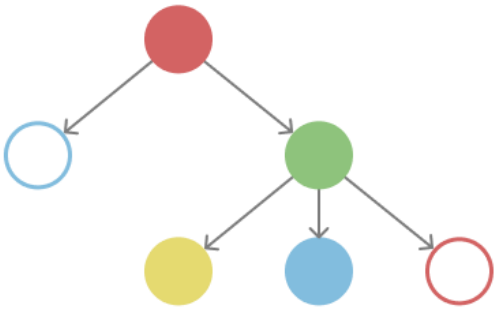
- Jádro, které je zodpovědné za správu dat v externí pamětí i RAM a protokolování.
- Databázový jazykový procesor, který optimalizuje požadavky na extrakci a změnu dat a vytváří typicky strojově nezávislý spustitelný interní kód.
- Subsystém podpory runtime, který interpretuje programy pro manipulaci s daty, které vytvářejí uživatelské rozhraní s DBMS.
- Servisní programy (externí nástroje), které poskytují řadu dalších funkcí pro údržbu informačního systému.

4.5.1 Klasifikace DBMS

Klasifikace podle datového modelu:[9]

- **Hierarchické**

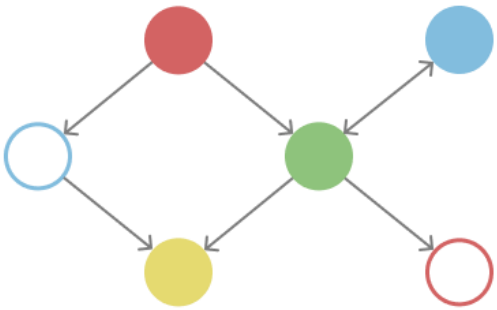
  Datový model, kde se používá reprezentace databáze jako stromová (hierarchická) struktura složená z objektů (dat) různých úrovní. Mezi objekty existují vazby, každý objekt může obsahovat několik objektů nižší úrovně. Takové objekty jsou ve vztahu předek-potomek, přičemž je možná situace, kdy objekt předek má několik potomků, zatímco objekt potomek má pouze jednoho předka. Objekty, které mají společného předka, se nazývají dvojčata. *(Mezipaměť, Google App Engine Datastore API)*



Obrázek 3. Hierarchický datový model [zdroj autor]

- **Síťové**

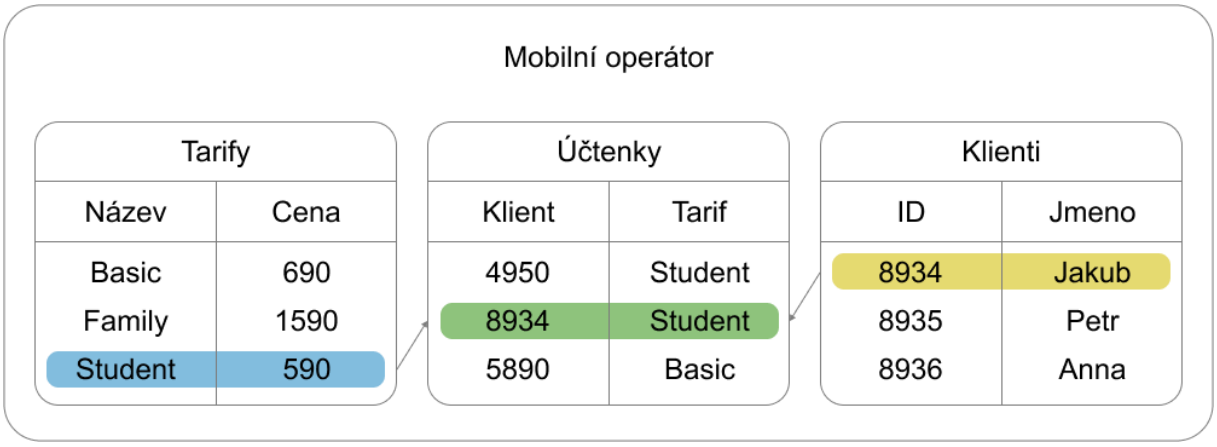
  Rozdíl mezi hierarchickým datovým modelem a síťovým datovým modelem je v tom, že v síťové datové struktuře může mít dítě libovolný počet předků. Síťová databáze se skládá ze sady instancí určitého typu záznamu a sady instancí určitého typu propojení mezi těmito záznamy. *(Mezipaměť)*



Obrázek 4. Síťový datový model [zdroj autor]

- **Relační**

  Logický datový model, aplikovaná teorie konstrukce databáze, což je aplikace na problémy zpracování dat takových oborů matematiky, jako je teorie množin a logika prvního řádu. Teorie normalizace je navíc zahrnuta do modelu relačních dat. *(Oracle Database, Microsoft SQL Server, SQLite, MySQL, PostgreSQL)*



Obrázek 5. Relační datový model [zdroj autor]

- **Objektově orientované**

  Databáze objektů je systém správy databází, ve kterém jsou informace prezentovány ve formě objektů používaných v objektově orientovaném programování. Objektové databáze se liší od relačních databází orientovaných na tabulky. Objektově relační databáze jsou hybrid obou přístupů. *(GemStone)*
- **Objektově relační**

  Tento typ DBMS umožňuje použití objektově orientovaného přístupu prostřednictvím rozšířených databázových struktur a dotazovacího jazyka: objektů, tříd a podobně. *(PostgreSQL, DB2, Oracle Database, Microsoft SQL Server)*

4.6 Popis formátu pro prezentaci geografických dat

Geobjekty v GIS jsou rozděleny podle počtu dimenzí. Skutečné objekty na zemském povrchu jsou vždy trojrozměrné. Jsou však přeneseny do středu GIS podle požadované úrovně zobecnění:[10]

- 0D geobjekty — bezrozměrné objekty, body definované pouze jejich polohou (autobusové zastávky, označení budov)
- 1D geobjekty — jednorozměrné objemy, úseky linek, konečná délka a nulová plocha (silnice, řeky)

- 2D geobjekty — dvourozměrné objemy, mnohoúhelníky s koncovým obvodem plochy (pozemky, pole, jezera)
- 3D geobjekty — trojrozměrné objekty, geometrické těla. Třetí měření v GIS je nejčastěji modelováno pomocí takzvaného digitálního modelu terénu.

4.6.1 Modely geografických dat

Vektorový model dat pracuje se třemi možnostmi objektů: body, čáry a oblasti. Existují různé modely vektorových dat.

Model špagety, kde jsou všechny typy objektů, bez ohledu na počet měření, uloženy v jednom různorodém seznamu. Ukládají se údaje o typu objektu (bod, čára, polygon) a parametrech objektu (jedna nebo více souřadnic). Model špagety neobsahuje žádné informace o topologii (příbuznost, orientace, konektivita, obsah), takže tento model je obtížné použít k analýze geodetů. Navíc existuje redundance dat.[10]

Hierarchický model ukládá data hierarchicky podle počtu měření. Je založen na tom, že polygon se skládá z několika řádků, čáry jsou z několika řádků, čáry jsou spojením dvou bodů. Tyto prvky jsou pak uloženy samostatně v GIS, nejčastěji v databázi geodetů.[10]

Topologický model je kompromisem mezi špagetami a hierarchickým modelem. Zachovány jsou pouze body a čáry a k lince lze připojit informace o její orientaci, podle kterých lze sousední polygon definovat vlevo a vpravo.[10]

Databáze geodetů obsahují tři základní typy datových sad:

- **Třídy prostorových objektů**
  Tabulka souborů obsahujících bodovou, lineární nebo polygonální geometrii pro geografické objekty, kde každý řádek představuje prostorový objekt.
- **Sada bitmapových dat**
  Obsahuje rastry představující kontinuální geografické jevy
- **Tabulky**
  Sada řádků se stejnými poli-třídami prostorových objektů.

Rastrový model dat

Model bitmapových dat pracuje se sadou bodů stejné velikosti pravidelně umístěných v řádcích a sloupcích. Přesnost tohoto modelu je pak dána velikostí bodu, který má vlastnost — barvu. Bitmapová data zahrnují letecké, satelitní a další snímky nebo naskenovaná data.

Kombinace grafických a tabulkových dat a využití softwarových nástrojů GIS umožňuje vizualizovat a prezentovat tyto údaje, připojovat další statistické informace nebo vlastní tabulky a provádět širokou škálu prostorové analýzy. Rozložení informací ve vrstvách je typické pro geografické údaje. Uživatel si může vybrat aktivní vrstvy sám.[10]

4.6.2 Mapbox

Poté, co se Google Mapy staly placenými o rozsahu použití pro své rozhraní API, které se používá pro mapování služeb na webových stránkách a v aplikacích, mnoho vývojářů a dalších platforem, které ji používali, začalo hledat alternativní služby, které by poskytovaly funkčnost Google, ale za nižší ceny. Mapbox přišel na pomoc.

Mapbox je platforma, která nabízí mapové služby prostřednictvím svých rozhraní API. Dvě hlavní knihovny geografických map, které jsou k dispozici vývojářům:

- **Mapbox.js**
  Pro základní mapy Mapbox.js zobrazuje soubory rastrových listů (PNG a JPEG) a zobrazuje je pomocí HTML a CSS.
- **Mapbox GL**
  Zobrazí vektorové dlaždice a zobrazí je pomocí WebGL. Může také zobrazit bitmapové dlaždice.

Mapbox umožňuje provádět přímé a reverzní geokódování. Přímé geokódování převádí text na zeměpisné souřadnice, například přeměnu “Prague“ na „50.075539, 14.437800“. Reverzní geokódování naopak převádí zeměpisné souřadnice na textové popisy.[11]

Jednoduchá implementace

```
let map = new mapboxgl.Map({
 container: 'map', // container id
 style: 'mapbox://styles/examples', // stylesheet location
 center: [50.075539, 14.437800], // starting position [lat, long]
 zoom: 10 // starting zoom
});
```

4.7 Popis kritérií pro správnou volbu technologií

Nejdřív popíšeme standardizovaná kritéria pro webové aplikace.

- Používání populárních řešení a trendů. Pokud se nepoužívají, aplikace rychle zastará a stane se nezajímavou. Aplikace musí nutně podporovat nejnovější verze populárních prohlížečů a nejnovější verze použitých technologií.
- Stabilní aplikace bez pádů. Jakákoli akce, kterou uživatel v aplikaci provede, musí být provedena okamžitě. Jedno zpoždění, druhé, a uživatel již zavře aplikaci a už se k ní nemusí nikdy vrátit. Aby se rychlost odezvy stala bleskurychlou, musíte poskytnout vysoce kvalitní serverovou část, uspořádat stabilní spojení se serverem.

Individuální kritéria

Vzhledem k tomu, že aplikace není určena pro komerční použití (i když komerční použití není vyloučeno a může být implementováno), nejprve zamyslíme na možné náklady (tj. přiblížit je co nejblíže 0). Technologie, které mohou vést k nákladům: webový server a aplikační server, proprietární nebo placené databázové technologie.

Dalším kritériem bude období realizace konečné aplikace. V tomto případě hodně závisí na počtu vývojářů, kteří pracují na aplikaci a jejich kompetenci (jsou požadovány znalosti). Předpokládejme tedy, že na projektu bude pracovat jedna osoba, která bude implementovat klientskou a serverovou část. Podle malé zkušenosti autora může doba vývoje takové aplikace jednou osobou trvat až několik měsíců. Nicméně vydaní hotového produktu a pokrytí potřeb potenciálních uživatelů je třeba dříve, protože za toto období se objeví neurčitý počet podobných služeb, a poté již není vhodné pokračovat ve vývoji této aplikace. V takovém případě přicházejí na pomoc alternativní a současně populární technologie a různé služby.

5 Výběr vhodné databáze (relační x nerelační) a DBMS

Relační DB ukládá strukturovaná data, která mohou představovat objekty ze světa kolem nás. Databáze postavená na takovém modelu tedy může obsahovat informace o skutečné osobě nebo o tom, jaké zboží zákazník vložil do košíku v nákupním centru. Tato data jsou nutně seskupena do tabulek v daném formátu. Nerelační DB jsou strukturovány odlišně. Typy dat přímo závisí na typu samotné databáze. Například v objektově orientované DB data obsažena ve formátu hierarchie a mohou popisovat různé objekty s libovolnými charakteristikami. Databáze umožňuje ukládat obrovské množství informací ve formě jediné entity.[12]

5.1 Vlastností SQL

Všechny informace v databázi jsou vždy přísně strukturované, vždy jsou spojeny s dalšími informacemi. Tabulka musí obsahovat řádky (se záznamy) a sloupce (s datovým typem).

Informace v buňce se vždy zapisuje podle určité šablony.

Integrita informací

Integrita informací znamená, že databáze je přesná, úplná a konzistentní. K zachování integrity v SQL se používají speciální nástroje, včetně primárních a cizích klíčů, a také speciální omezení, například „Default“. Tato omezení umožňují vynutit účinná pravidla pro všechny informace v databázových tabulkách a také zajistit správnost informací.[12]

Transakce

Transakci v databázi jsou uspořádané sekvenci operátorů zpracování informací, které mohou přenášet databázi z jednoho stavu do druhého. Transakce jsou prováděny ve formě sekvenčních operací, které představují jeden úkol. Příkazy musí být provedeny současně a jako celek. Mohou být zapsány do databáze společně nebo nemusí být zaznamenány v jakékoli formě.[12]

Soulad s požadavky

- **Atomicita**
  Podmínka, která umožňuje úspěšné provedení celé transakce, nebo zruší celou transakci, pokud nelze dokončit alespoň jednu její část.
- **Jednotnost**
  V souladu s touto podmínkou musí všechny zaznamenané informace v průběhu jedné transakce splňovat všechna pravidla a předpisy.
- **Izolace**
  Pouze s izolací se může řídit konzistenci prováděných transakcí.
- **Spolehlivost**
  Podmínka, která znamená, že všechny změny provedené v databázi jsou považovány za dokončené.

Relační databáze se vyznačují svou účinností a spolehlivostí, tj. ochrana dat před nejrůznějšími poruchami a ztrátami, data nespotřebovávají více zdrojů, než je uvedeno v parametrech atributů, dostatečně vysoká rychlost čtení.

5.2 Vlastnosti NoSQL

Na rozdíl od SQL jsou v NoSQL všechny informace uloženy bez přesně definované struktury a explicitních vztahů mezi všemi daty. Neukládá žádné strukturované a přehledné tabulky, ale veškeré informace, které lze prezentovat ve formě textového dokumentu, zvukového souboru

nebo publikace na internetu. Protože do takových databází lze ukládat téměř jakákoli data, jsou široce používána v různých aplikacích pro PC a smartphony. Jsou ideální pro všechny případy, kde je flexibilní a snadno škálovatelná databáze, která se také vyznačuje vysokými výkonovými parametry důležitější než struktura srozumitelných dat.

Flexibilita

Díky své vysoké flexibilitě umožňuje NoSQL provádět vývoj mnohem rychleji a lze jej implementovat v několika fázích. Modely s dobrou flexibilitou jsou vhodné pro ukládání nestrukturovaných i neúplně strukturovaných informací.[12]

Škálovatelnost

Nerelační DB jsou navrženy tak, aby škálovaly pomocí distribuovaných klastrů. Někteří prodejci cloudu provádějí nezbytné operace na pozadí, aby zajistili, že služba bude co nejlépe spravovatelná.[12]

Účinnost

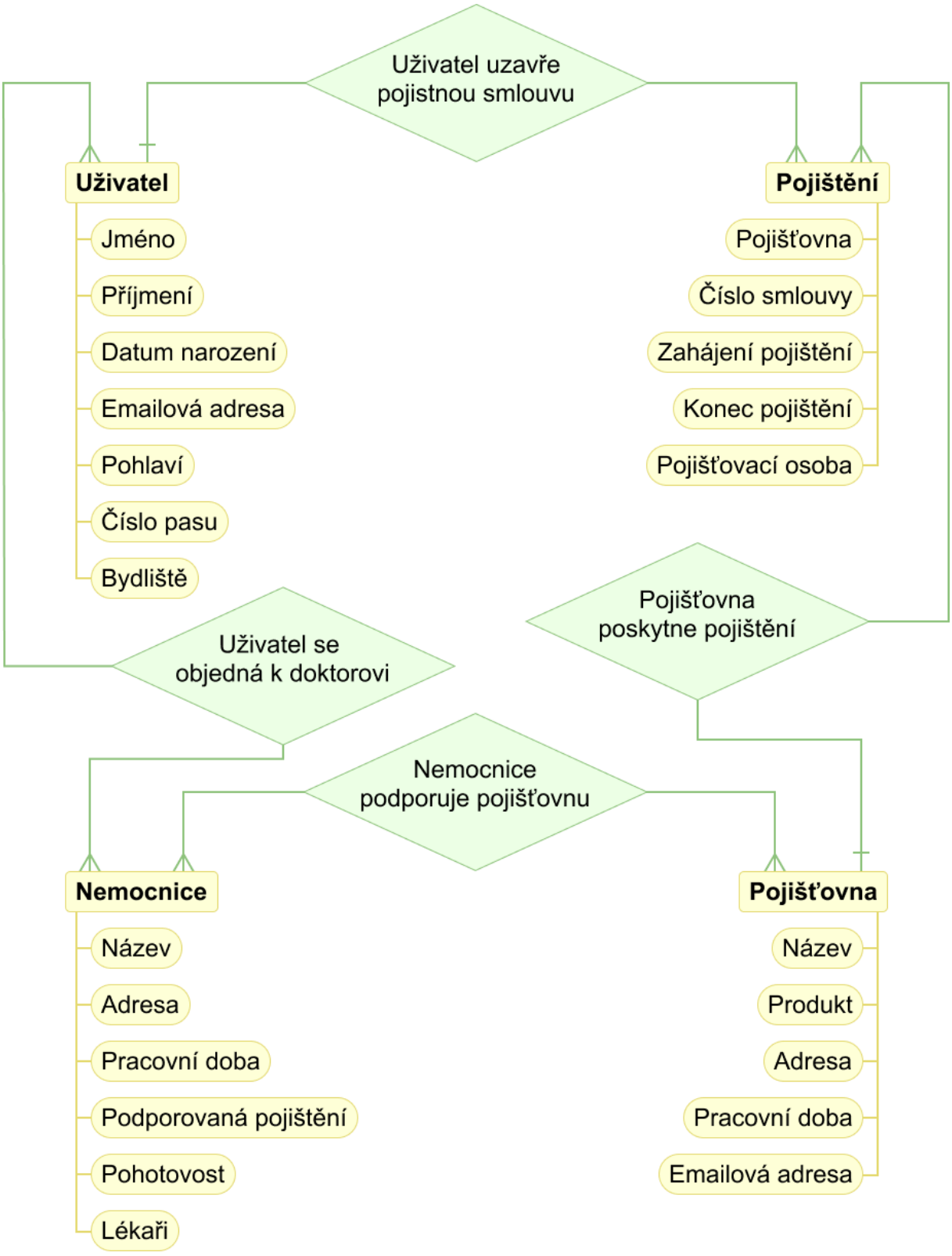
Databáze NoSQL jsou optimalizovány pro ukládání šablon nebo konkrétních dat, čímž dosahují výrazně vyššího výkonu než relačních DB.[12]

Nerelační databáze jsou optimalizovány pro aplikace, které potřebují rychle zpracovat velké množství dat s různými strukturami, s nízkou latencí.

V aplikaci jsou všechny údaje ve struktuře homogenní (například struktura popisující nemocnici A se neliší od struktury nemocnice B). Mezi entitami existují jasná spojení. Proto je v tomto případě rozumnější zvolit relační databázi. Neexistují žádné databáze, které by byly vhodné úplně pro všechno. Mnoho společností používá pro různé úkoly relační i nerelační databáze. Proto jde však přemýšlet o použití řešení SQL a NoSQL, které umožní provozovat databázi v různých konfiguracích, zajistit jejich dobrou škálovatelnost a spolehlivost.

5.3 E-R model aplikace

Správným přístupem k výběru DBMS bude primární návrh dat (databázové schéma), které budou uloženy v databázi. Při první iteraci aplikace se databáze bude skládat z množiny nemocnic, formálních pojistných smluv, pojišťovacích společností a uživatelů. Je však nutné zajistit případné rozšíření databáze. Nejprve je nutné vypracovat E-R model (definice entit, atributů, omezení entit, popis vztahů mezi entitami).



Obrázek 6. E-R model aplikace [zdroj autor]

Jak již bylo zmíněno, databáze se skládá ze čtyř hlavních sad, které mají své vlastní charakteristiky (atributy). Každý uživatel musí mít informace, jako je křestní jméno, příjmení, datum narození, e-mailová adresa, pohlaví, číslo dokladu totožnosti a místo bydliště. Následně

bude mít každý uživatel možnost uzavřít pojistnou smlouvu online, proto již při první iteraci bude tabulka uživatelů evidovat vydané pojistné smlouvy.

Pojišťovna zase bude poskytovat pojistné smlouvy uživatelům. Abyste měli všechny uživatelé potřebné informace o společnosti, i nemocnice by s nimi mohli spolupracovat, každá pojišťovna by měla obsahovat následující údaje: název, nabízený produkt (pojištění), adresu společnosti, pracovní dobu (včetně víkendu), kontakt pro komunikaci (v tomto případě e-mail).

Všechny nemocnice poskytnou uživatelům, kteří jsou členy komplexního zdravotního pojištění (kteří si uzavřeli pojistnou smlouvu od společnosti, z níž nemocnice spolupracuje) možnost se objednat online k doktorovi. Uživateli stačí vybrat nemocnici, lékaře, dostupné datum a čas. Zamluvený termín bude mít nemocnice v jednotném systému. Uživatel může prohlédnout všechny budoucí rezervace přímo v aplikaci (v případě změnit je nebo zrušit, nemocnice ihned dostane aktualizované rezervace). Pro implementaci těchto funkci je nezbytná relevantní informace o nemocnicích (název nemocnice, adresa, otevírací doba, se kterými pojišťovny nemocnice spolupracuje, dostupnost pohotovosti a seznam lékařů).

Uživatelé si také budou moci prohlédnout nejen informace o své aktuální pojistné smlouvě, ale také o předchozích. K tomu existuje tabulka uzavřených smluv, oběma stranám rovněž poskytnou všechny potřebné informace (společnost, která pojistnou smlouvu sepsala, číslo pojistné smlouvy, počátek a konec pojištění, pojištěná osoba (v tomto případě uživatel)). Ve druhé iteraci aplikace bude také možné zobrazit a stáhnout pojistnou smlouvu ve formátu PDF nebo dokonce přejít na elektronické certifikáty.

Poté je z modelu E-R sestavena podrobnější úroveň modelování (logický model). Logický model označuje, jaké sloupce konkrétní tabulka obsahuje, pokud jsou data v těchto sloupcích požadována, a které sloupce obsahují primární klíče.

Model také ukazuje na konkrétní relace mezi databázovými tabulkami. V navrhovaném provedení se používají spojení jako O2M nebo M2O a M2M. Protože existuje mnoho relací, lze se ujistit, že byla relační databázi vybraná správně. Kromě toho byly vytvořeny tabulky:

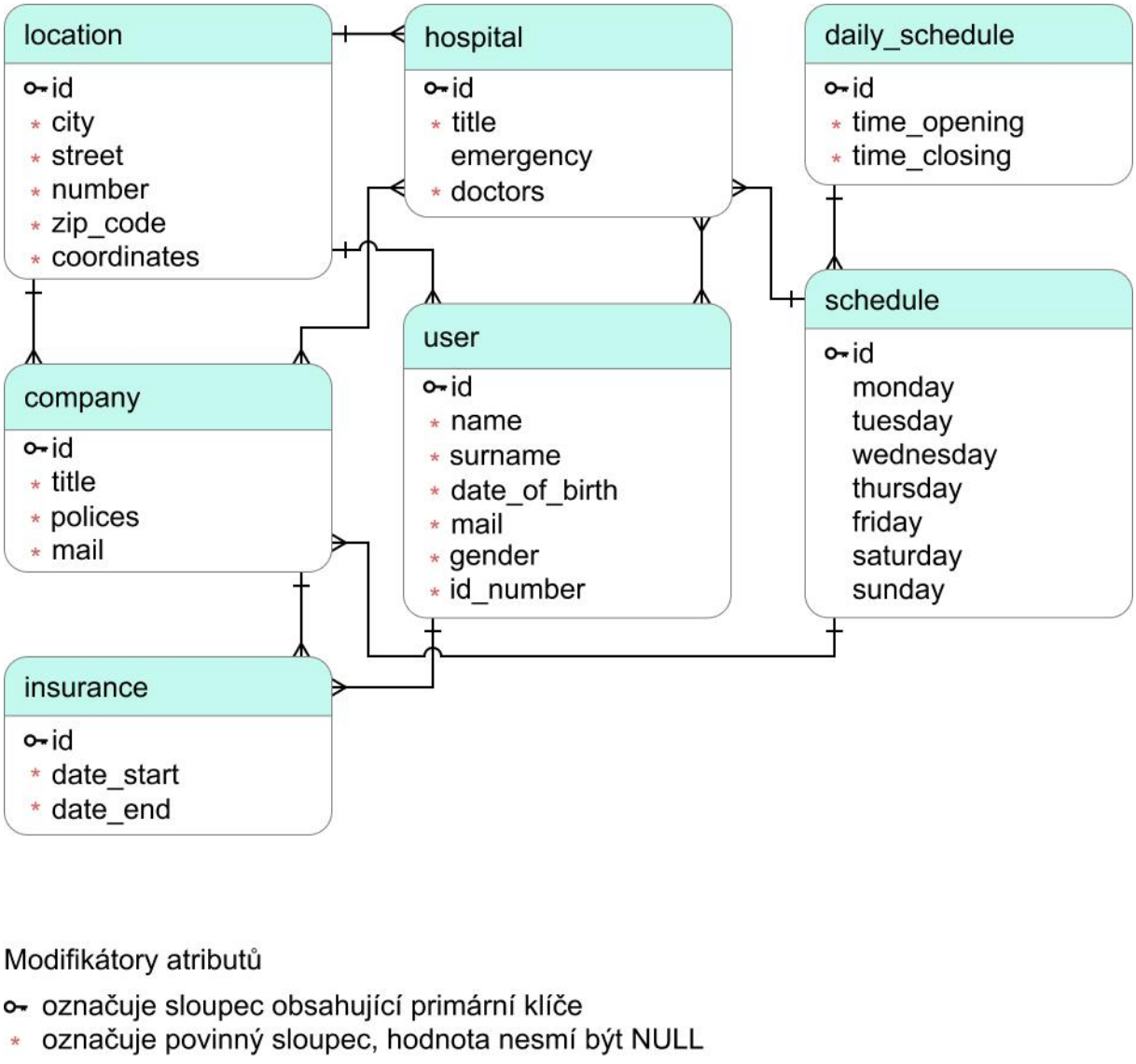
- **Location**
  Nejen nemocnice mají informace o poloze, ale také uživatel a pojišťovna. V některých případech se adresy mohou shodovat a za účelem úspory paměti datového serveru bylo rozhodnuto přesunout atribut adresy do samostatné tabulky. Tabulka zaznamenává hodnoty měst, ulic, popisných čísel, poštovních směrovacích čísel a souřadnic.

- **Schedule**

  Jako slovo “schedule” se rozumí pracovní doba. Stejně jako u adresy se otevírací doba různých nemocnic a společností překrývá ještě častěji. Tabulka se skládá pouze z dnů v týdnu. Hodnota je volitelná. Pokud je hodnota NULL, znamená to den volna.

- **Daily schedule**

  Tabulka je logickou součástí tabulky Schedule, upřesňuje každý den v týdnu. Pole pro otevření a zavření jsou povinná. Tato tabulka byla vytvořena se stejným účelem jako předchozí (úspora paměti). Není tedy třeba zapisovat již existující hodnoty, ale pouze použit relace.



Obrázek 7. Logický datový model aplikace [zdroj autor]

Existují také další varianty logického modelu. Jeho konečný vzhled zpravidla závisí na funkcích, které budou v aplikaci použity.

Dalším krokem je vytvoření fyzického datového modelu. Fyzický model je nejpodrobnější úroveň modelování. Tento model je již závislý na vybrané databázi, protože jednotlivým sloupcům přiřazuje datové typy a jejich délku či přesnost. Proto je před vytvořením fyzického modelu nutné se rozhodnout pro výběr konkrétní databáze. Před vytvořením fyzického modelu je proto nutné se rozhodnout pro výběr konkrétního DBMS. Protože byl definován typ relační databáze, je k dispozici následující volba RDBMS: MS SQL, Oracle Database, MySQL, PostgreSQL, Amazon Aurora, MariaDB a další.

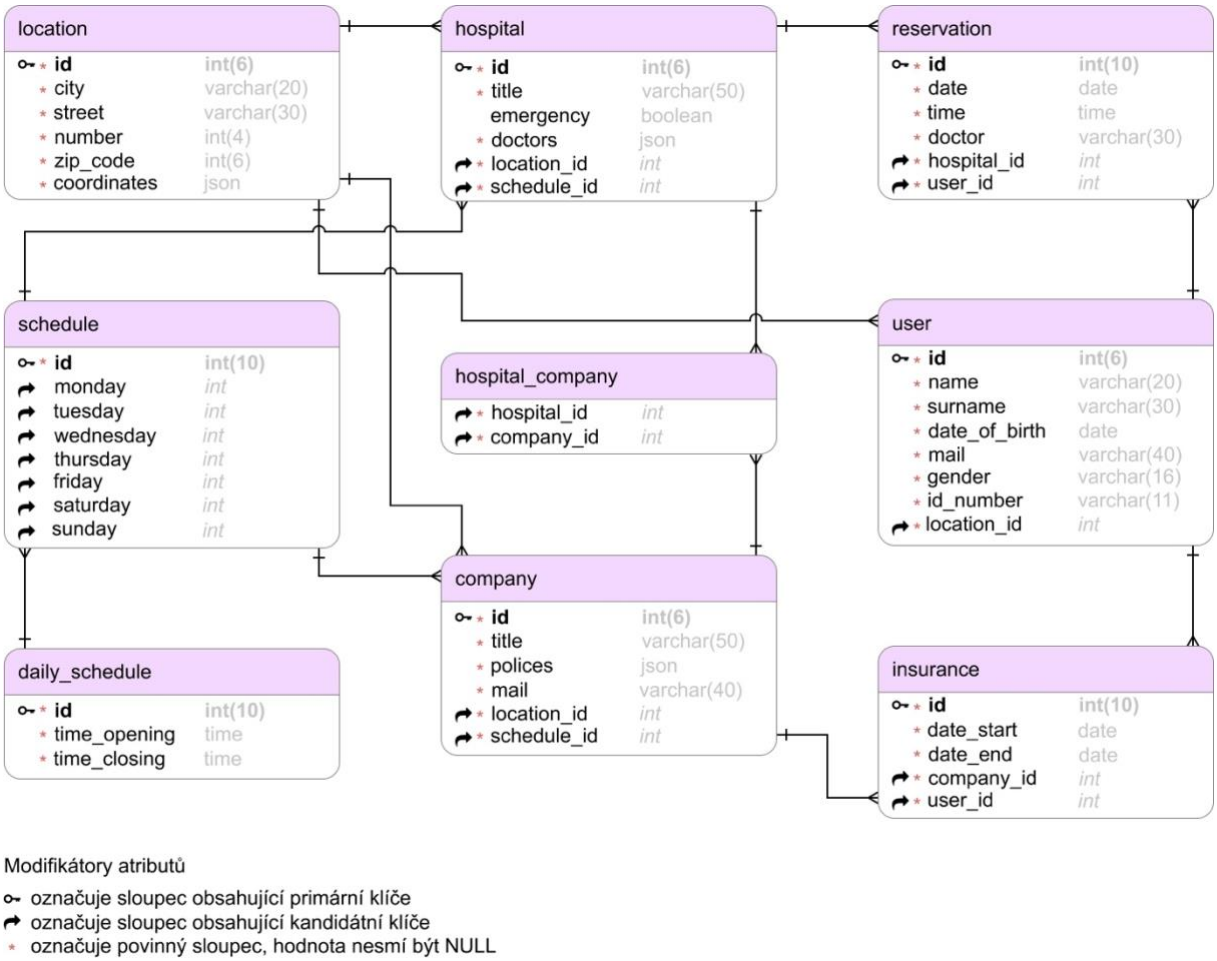
Pro tuto složitost aplikace jsou všechny RDBMS podobné, takže výběr bude proveden podle následujících kritérií: spolehlivost, snadné použití, dostupnost úplné dokumentace. Ve většině případů si vývojáři vybírají DBMS, se kterými jsou zvyklí pracovat, takže tato skutečnost bude také brána v úvahu jako kritérium. Na základě výše uvedeného není třeba „vymýšlet kolo“. Filtr kritérií označuje tři nejvhodnější a nejoblíbenější RDBMS:

- SQLite: velmi výkonný integrovaný RDBMS;
- MySQL: nejpopulárnější a nejčastěji používaný RDBMS;
- PostgreSQL: nejpokročilejší a nejflexibilnější RDBMS.

Protože MySQL je nejpoužívanější RDBMS a má spoustu dokumentace, je logické předpokládat, že pokud vzniknou nejasnosti nebo problémy s jeho prací, řešení lze najít v dokumentaci nebo na fórech. Konečným výběrem databáze bude proto MySQL a následný vývoj aplikace to zohlední.

Ve fyzickém datovém modelu je nutné nasadit všechny relace M2M. V tomto případě je pro relaci mezi nemocnicí a společností vyžadována pomocná tabulka *hospital_company*. Technicky budou vytvořena dvě propojení O2M nebo M2O. Sloupec *hospital_id* z tabulky *hospital_company* bude odkazovat na sloupec *id* z tabulky *hospital*. A sloupec *company_id* z tabulky *hospital_company* bude odkazovat na sloupec *id* z tabulky *company*. Pak tam jsou sloupce *hospital_id* a *company_id* v tabulce *hospital_company* představují kompozitní primární klíč a zároveň jsou cizí klíče pro komunikaci s tabulkami *hospital* a *company*. Stejným způsobem byla vytvořená pomocná tabulka *reservation.* Tato tabulka se také použije, například když nemocnice bude vést záznamy o návštěvnících.

Ve výsledném modelu je nutné upravit typy a velikosti polí podle zvoleného DBMS (MySQL). Kromě toho jsou ve fázi vytváření fyzického datového modelu zavedena pravidla ověřování sloupců, která určují seznamy přijatelných hodnot. Všechny tyto informace jsou k dispozici v Příloze 1. Vlastnosti sloupců tabulek fyzického databázového modelu.[13]



Obrázek 8. Fyzický datový model aplikace [zdroj autor]

6 Tvorba definované databáze

Před spuštěním hotové databáze na serveru je třeba ji vytvořit a otestovat na místním serveru (na zařízení vývojáře). K tomu je třeba nainstalovat požadovaný software. Například spuštění lokálního serveru a následné vytvoření databáze bude provedeno v operačním systému pomocí softwaru:

- Operační systém: Mac OS Big Sur (11.4)
- Databázový server: MySQL Community Server (8.0.26)
- IDE: DataGrip (2021.1.3)

Pokyny pro ostatní operační systémy jsou v dokumentaci MySQL.[14]

6.1 Spuštění serveru localhost

Nejprve je třeba stáhnout instalační soubor MySQL Community Server kompatibilní s architekturou operačního systému a procesoru ze stránky dev.mysql.com/downloads/mysql. Po instalaci stačí spustit server.

```
user ~ % sudo /usr/local/mysql/support-files/mysql.server start
```

6.2 Připojení k serveru pomocí IDE

Aby se připojit k běžícímu serveru a vytvořit na něm databázi, je nutné vytvořit projekt a nakonfigurovat data source a driver. Hostitel bude *localhost*, protože server běží na zařízení, standardní port je *3306*, výchozí uživatel je *root* a heslo, které bylo nakonfigurováno při instalaci staženého souboru MySQL Community Server. Driver a DBMS jsou MySQL.

6.3 Kódování databáze

Poté následuje standardizovaná sekvence pro vytvoření databáze, kde prvním krokem je vytvoření schématu.

```
create schema data;
```

Druhým krokem je vytvoření první tabulky. Podívejme se na tabulku *location* jako příklad (protože tato tabulka nedědí informace z jiných tabulek). Vypracovaní dalších tabulek naleznete v Příloze 2. Kódování databázových tabulek.

```
create table location
(
   id int(6) not null,
   city varchar(20) not null,
   street varchar(30) not null,
   number int(4) not null,
   zip_code int(6) not null,
   coordinates json not null
);
```

Kromě toho je atribut *id* v každé tabulce primárním klíčem, je generovaný databází a musí být jedinečný.

```
create unique index location_id_uindex
   on location (id);

alter table location
   add constraint location_pk
      primary key (id);

alter table location modify id int(6) auto_increment;
```

6.4 Přidání testovacích dat

Aby otestovat fungování databáze, musíme přidat záznamy. Pro nahraní záznamů, můžeme použít online službu pro generování dat mockaroo.com nebo data zadat ručně. V prvním případě je třeba vygenerovaný soubor CSV uložit do zařízení, aby k němu IDE mělo přímý přístup. Pro import dat ze souboru do tabulky:

```
load data local infile '/Users/yauhen/study/BP/data/users.csv'
   into table user
fields terminated by ','
lines terminated by '\n'
(id, name, surname, date_of_birth, mail, gender, id_number, location_id);
```

V případě ručního přidávání musíme použít následující příkaz:

```
insert into user
(id, name, surname, date_of_birth, mail, gender, id_number, location_id)
values
(1, 'Ivan', 'Petrov', '2000-03-08', 'ivan@fd.cvut.cz', 'male', 'PR08032000', 1);
```

Důležité: při přidávání dat do sloupců obsahujících cizí klíč musí být data již přidána do tabulky, na kterou tento klíč odkazuje. V tomto případě, než uložím data o uživateli *Ivan*, musím přidat jeho záznam adresy do tabulky *location* a poté použít *id* vytvořeného řádku jako *lokation_id*.

Protože naše databáze není příliš velká a má citlivé atributy a jejich formáty a také velký počet odkazů, použijeme druhou metodu pro kontrolovanější proces. Vytvořme v každé tabulce asi 10 záznamů s různými informacemi, abychom tato data následně zkontrolovali pomocí dotazů.

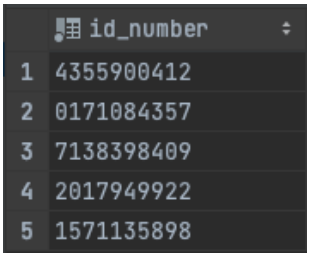
6.5 Testovací dotazy do databáze

Abychom se ujistili, že je databáze vytvořena správně a všechny vztahy fungují podle fyzického modelu, spustíme několik dotazů.

V prvním testovacím dotazu zkontrolujeme vztah mezi tabulkou *user* a tabulkou *location*. Uděláme požadavek na všechny uživatele, kteří žijí ve městě “Prague“, a zobrazíme jejich *id_number*.

```
select id_number from user
inner join location l on user.location_id = l.id
where l.city = 'Prague';
```

V důsledku toho jsme z databáze obdrželi odpověď s tabulkou, která obsahuje jeden sloupec *id_number*. Provedením ruční kontroly můžeme zaručit 100% přesnost relace a tabulky *location*.



	id_number
1	4355900412
2	0171084357
3	7138398409
4	2017949922
5	1571135898

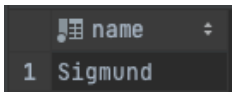
Obrázek 9. 1. Výsledek dotazovaní tabulky User [zdroj autor]

Dále zkontrolujeme konkrétní pojišťovnu, zobrazíme jména všech uživatelů, kteří si u této společnosti zakoupili pojištění.

```
select name from user u
inner join insurance i on u.id = i.user_id
inner join company c on i.company_id = c.id
where c.title = 'HCF Insurance Group';
```

V tomto případě byla odpovědí tabulka uživatele s atributem *name* a pouze s jedním řádkem, který odpovídá zadaným datem v databázi.

Pro větší důvěru ve správnost složení databáze má smysl provést několik dalších testovacích dotazů. V tomto případě to však není nutné, protože pro vývoj webových aplikací budou použity alternativní technologie.



	name
1	Sigmund

Obrázek 10. 2. Výsledek dotazovaní tabulky User [zdroj autor]

7 Alternativní technologie pro vývoj aplikací

Existuje obrovské množství kombinací využití technologií. Každý vývojář si vybírá ty, se kterými je snadné pracovat a které pokrývají potřebnou funkčnost aplikace. Tento projekt se zabývá následující kombinací — Netlify + Directus + AWS.

Obrázek 11. Sada alternativních technologií [zdroj autor]

7.1 Netlify

Netlify je služba hostování a bez serveru pro statické stránky. Služba poskytuje velký výběr nástrojů, které jsou skvělé pro účely a cíle tohoto projektu. Některá důležitá kritéria, která Netlify má:

- zdarma hosting pro vlastní domény;
- snadné použití;
- schopnost podporovat dynamický (nejen statický) obsah;
- vestavěná ochrana HTTPS;
- možnost náhledu webu před spuštěním;
- hotová funkce pro A / B testování.

Další výhodou Netlify je přítomnost Next.js (řeší problém React.js s předběžným vykreslením aplikaci na straně serveru, nikoli klientem — SSR). Netlify tedy pokrývá potřebu webového serveru a zároveň je hostingem budoucího webu.

7.2 Directus

Directus je moderní open source Headless CMS (systém, který zobrazuje obsah pomocí RESTful API), který kombinuje vlastní databáze SQL s dynamickým API a poskytuje intuitivní administrátorskou aplikaci pro správu jeho obsahu.

Klíčové vlastnosti:

- Nastavení knihoven, nástrojů a zásobníku, které jsou pro projekt nejlepší.
- Zcela zdarma prémiový software bez licencí pro osobní nebo komerční použití.
- Directus odráží přesně to, co je v databázi SQL. Žádný proprietární datový model nebo úložiště obsahu.
- Systém rozložení mřížky umožňuje organizovat, seskupovat a umisťovat pole a vytvářet jednotlivé formuláře pro všechny datové sbírky. Více než 50 různých polních rozhraní.
- Spravuje vícejazyčný obsah v libovolném počtu jazyků.
- Flexibilní konfigurace přístupu k datům.

V tomto případě Directus umožňuje provádět přímé požadavky HTTP do databáze, poskytuje pohodlnou aplikaci pro správu dat a umožňuje organizovat skupiny uživatelů s různými úrovněmi přístupu k datům.

7.3 AWS

Amazon Web Services je komerční veřejný cloud spravovaný a vyvíjený společností Amazon. Poskytuje model infrastruktury (virtuální servery, úložiště) i služby na úrovni platformy (cloudové databáze, cloudový middleware, cloudové bezserverové počítače, vývojové nástroje). Amazon poskytuje příležitost vyzkoušet své zdroje po dobu jednoho roku zdarma s určitými omezeními, ale tato omezení budou pro tuto aplikaci stačit.

Minimum prostředků požadovaných pro použití aplikace:

- Elastic Compute Cloud (EC2) — virtuální server, na kterém bude databáze spuštěna.
- Relational Database Service (RDS) — cloudová relační databázová služba.
- Simple Storage Service (S3) — hostování souborů pro ukládání a přijímání velkých objemů dat.

Kombinace těchto prostředků umožní spuštění cloudové databáze. V rámci této práce nebude zváženo nastavení a spuštění služby. Podrobné pokyny najdete v dokumentaci Amazonu.

8 Implementace databáze

Aby databáze fungovala a byla k dispozici uživatelům, musí být umístěna na serveru.

8.1 Spuštění Amazon Elastic Compute Cloud

Spuštění serveru na Amazonu je dostatečně snadné pomocí několika kroků. Podrobné pokyny jsou v dokumentaci Amazon.[15]

1. Amazon Machine Image (AMI)
   Prvním krokem zvolím AMI, což je kolekce softwaru (OS, aplikační server a aplikace potřebné ke spuštění instanci). Některé AMI již mají předinstalované aplikace jako Docker, MySQL, Java, Node a další. Proto pro pohodlí jednoduše vybereme AMI v Linuxu s již nainstalovaným softwarem.
2. Instance Type
   V dalším kroku vybereme typ instance, který potřebujeme, v závislosti na potřebách aplikace. V bezplatné verzi je k dispozici pouze t2.micro, ale bude to stačit (1 GB RAM a dostatečně výkonný procesor na zpracování takového objemu dat).
3. Úložiště
   Dále nakonfigurujeme úložiště, vybereme potřebné množství paměti (8 GB pro obecné účely a 30 GB vyhrazené)
4. Konfigurace skupin zabezpečení
   Aby server přijímal požadavky od uživatelů a správce jej mohl spravovat jako uživatel root, musíme nakonfigurovat připojení SSH a HTTP a stáhnout nový pár klíčů pro root připojení.
5. Připojení
   K připojení použijeme následující příkaz, kde *key* je pár klíčů, *user* je uživatel zadaný během konfigurace a *id* je identifikační číslo instance:

```
user ~ % sudo ssh -i "<key>" <user>@<id>.eu-central-1.compute.amazonaws.com
```

8.2 Spuštění Amazon Relational Database Service

Vytvoření instance DB se také provádí prostřednictvím konzoly AWS. V sekci RDS vybereme standardní způsob vytváření databáze, zvolíme MySQL, vymysleme uživatele root (uživatelské jméno a heslo). Podrobné pokyny jsou v dokumentaci Amazon.[16]

Poté v nastavení RDS připojíme jej k již spuštěnému serveru EC2.

8.3 Instalace Directus a vytvoření projektu

Aby nainstalovat projekt Directus na server, musíte mít nainstalovaný Node.js. Nejprve tedy zkontrolujeme, zda je, v opačném případě jej zařídíme.[17]

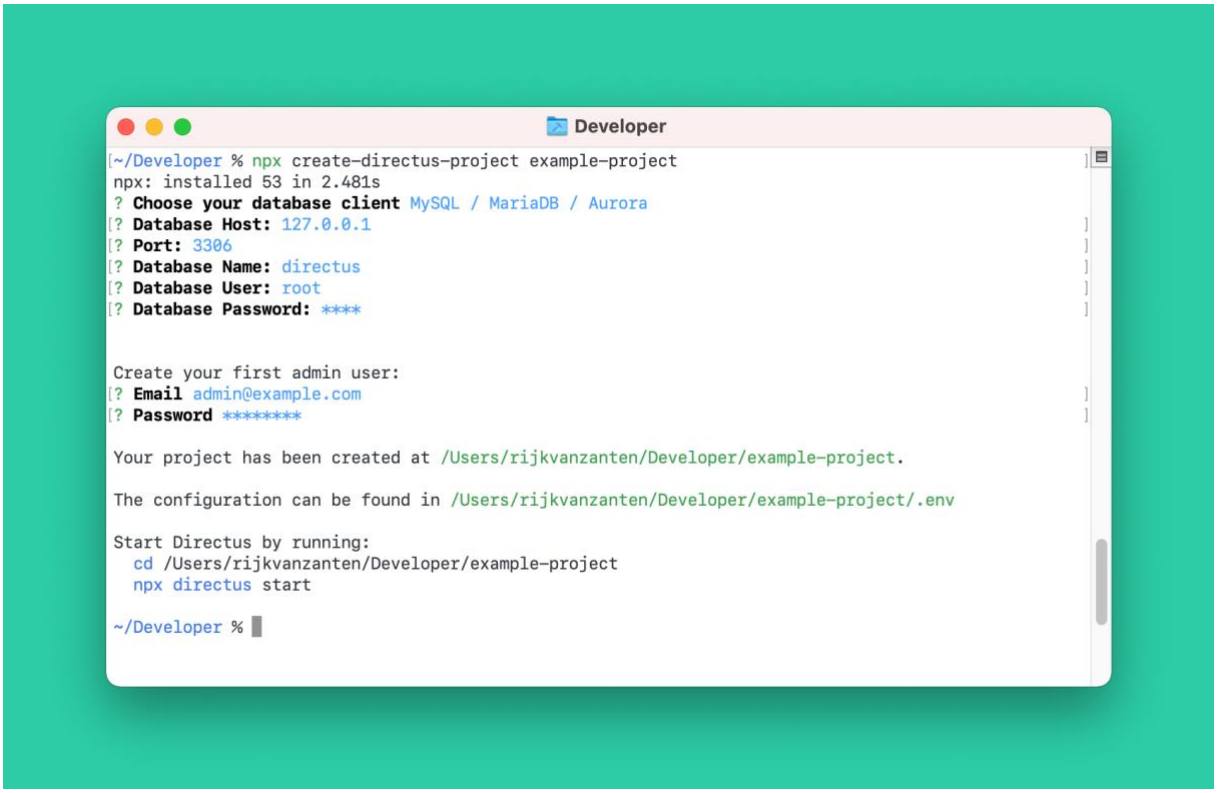
```
~ % node -v
```

V případě existence očekávaná odpověď je *v15.0.1* nebo jiná verze. Aby nainstalovat Node.js, použijeme příkaz

```
~ % sudo apt install nodejs
```

Dále v terminálu spustíme následující příkaz a budeme postupovat podle pokynů.

```
~ % npx create-directus-project hcf-api
```



Obrázek 12. Průběh instalace DB na serveru [19]

Instalace dokončena. Spustíme server pomocí následujícího příkazu (výchozí port 8055).

```
~ % npx directus start
```

Přechodem na veřejnou adresu serveru pomocí portu 8055 se otevře rozhraní pro správu databáze, kde je třeba opakovat tabulky vytvořené lokálně (v systému správy se jim říká kolekce). Zkušenější vývojáři mohou hotovou databázi migrovat do nového projektu. Directus má některé funkce, díky nimž je obtížné provést migraci. Například různé typy dat, tabulky odkazů a pole. Představte si Directus jako systém pro správu obsahu, kde můžete vytvářet nové tabulky, vytvářet odkazy, vytvářet záznamy, upravovat a mazat je.

Také v Directusu existuje kolekce uživatelů a rolí, která umožňuje pohodlně a rychle konfigurovat oprávnění tabulek, záznamů, souborů atd. pro skupiny a jednotlivé uživatele. Podrobný návod, jak vytvářet kolekce a záznamy, najdete na stránce dokumentace Directus.[18]

Operace s databází se provádí pomocí požadavků HTTP (adresa IP serveru, port, kolekce, požadované parametry). Protože byla aplikace napsána v programovacím jazyce JavaScript / TypeScript, pomohla při používání databáze knihovna JavaScript SDK, vyvinutá společností Directus pro pohodlnější a jednodušší práci s databází. V tomto případě není třeba používat axios (slibovaný HTTP klient pro prohlížeč a Node.js) a podobné technologie, knihovna si vše dělá samostatně.

9 DB dotazování pomocí JavaScript SDK

Directus JS SDK umožní získat, načíst nebo aktualizovat data v databázi. Knihovna se velmi snadno používá a nedovolí dělat chyby, protože je napsána v čistém programovacím jazyce TypeScript. Aby nainstalovat knihovnu, musíte v terminálu přejít do složky projektu a spustit příkaz

```
user ~ % npm install @directus/sdk-js
```

Poté musíme nakonfigurovat konstantu, kterou budeme volat při zadávání požadavků do databáze.

```
import { Directus } from '@directus/sdk'

export const API = new Directus('http://127.0.0.1:8055')
```

Nyní můžeme přistupovat k databázi přes konstantu *API*, kde budou zadány parametry dotazu (ke které kolekci přistupovat, kolik záznamů získat, filtrovat podle záznamů nebo třídění). Níže jsou uvedeny skutečné požadavky z aplikace. Důležité. Všechny metody jsou slibované, takže funkce musí být asynchronní (async – await).

Žádost uživatele o autorizaci

V tomto požadavku používáme metodu ověřování *(auth)* s přihlašovacím voláním *(login)*, kde předáváme parametry jako *email* a *password*.

```
await API.auth.login({
  email: payload.email,
  password: payload.password
})
```

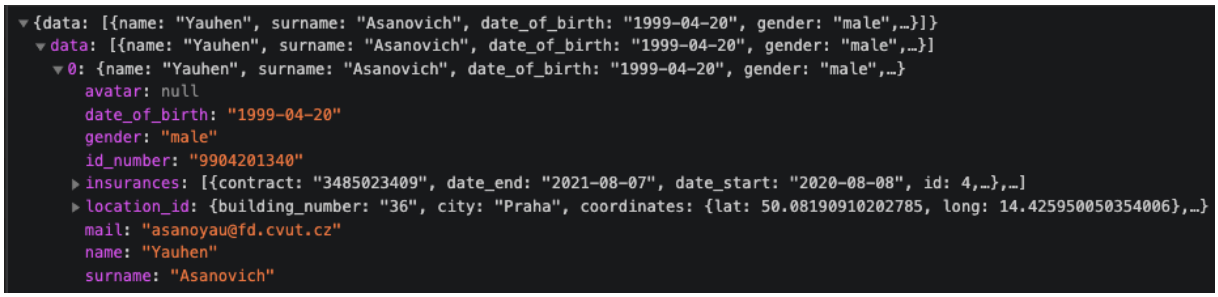
Tento požadavek vrátí přístupový token k databázi pro daného uživatele (*access_token*), klíč k aktualizaci *access_token* a resetování hesla uživatele (*refresh_token*) a dobu vypršení platnosti klíčů.

Žádost o údaje uživatele

Kromě přístupu musí uživatel vidět všechny informace o sobě. Za tímto účelem napíšeme požadavek na shromažďování informací o uživateli *(person)* s poli, která chceme obdržet.

```
await API.items('person').readMany({
      fields: ['name', 'surname', 'date_of_birth', 'gender', 'mail',      'avatar.*', 'id_number', 'location.*',
      'insurance.*.*.*']
})
```

Výraz „.*" znamená vnoření pole, tj. v relačním vztahu například získáme záznamy ze související tabulky. Tímto způsobem obdržíme uživatelské jméno, příjmení, datum narození, pohlaví, emailovou adresu, obrázek, číslo pasu, adresu bydliště a uzavřené pojistné smlouvy. Všechny žádosti o osobní údaje jsou chráněny, proto se používá Request Header Authorization. Databázový server vrací data ve formátu JSON, takže se s ním snadno pracuje. V tomto požadavku budou data vypadat takto:



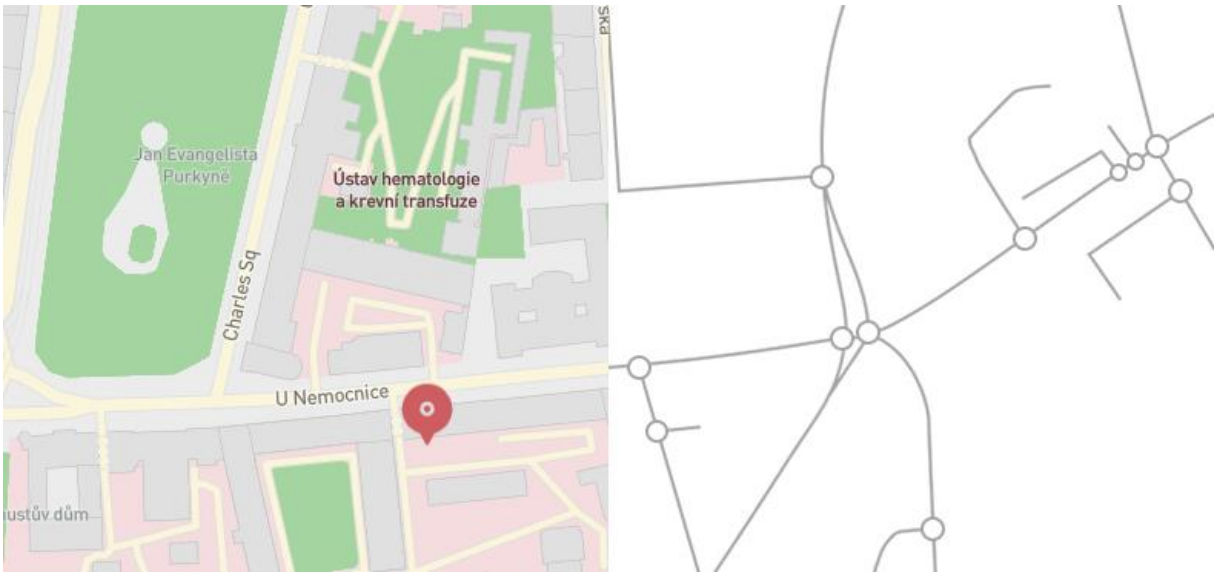
Obrázek 13. Informace o uživateli z databáze [zdroj autor]

A samotný požadavek HTTP vypadá takto:

http://127.0.01:8055/items/person?fields[]=name&fields[]=surname&fields[]=date_of_birth&fields[]=gender&fields[]=mail&fields[]=avatar.&fields[]=id_number&fields[]=location_id.*&fields[]=insurances.*.*.**

10 Navigace pro mapy (další iterace aplikace)

Pro sestavení optimální trasy v jakýchkoli mapách se používají algoritmy k nalezení nejkratší cesty z bodu A do bodu B. Předpokládejme, že se uživatel potřebuje dostat z bodu A do bodu B. Zatímco zadá souřadnice do navigátoru a stiskne tlačítko „sestavit trasu“, systém analyzuje obrovské množství možností cest. Mapové informace jsou grafy silnic, které obsahují velké množství informací: druh provozu v ulicích (jednosměrný, obousměrný), délka ulice, přípustná rychlost atd. Některé systémy (například Google Maps, Yandex Navigator) přidávají k informacím na mapě dopravní zácpy. Tyto informace jsou zpracovávány ve 2 fázích. Ve fázi načítání map prochází silniční graf zpracováním dat a není určen počáteční ani koncový bod trasy. Když systém přijal data z tohoto grafu, začíná fáze požadavku. Toto je systém, který skenuje přijatá data a na základě těchto dat najde nejkratší cestu. Systém používá k nalezení nejkratší cesty určité algoritmy. Pokud vezmeme v úvahu teorii grafů, pak pro tyto účely lze použít několik algoritmů: Floydův, Fordův, Dijkstrův, kde silnici jsou hrany a křižovatky fungují jako uzly. V dnešní době je to často směsice několika algoritmů, nových algoritmů založených na starých nebo zcela nových přístupech, které byly vytvořeny pro specifické potřeby (například metoda Kontrakční hierarchie).



Obrázek 14. Vytváření mapy jako grafu [zdroj autor]

11 Závěr

Účelem této práce bylo vybrat nejvhodnější databázi pro aplikaci pro cizince tak, aby náklady byly minimální a výsledek byl maximalizován. Databáze MySQL jsou velmi populární, protože se objevily v roce 1994, přesto jsou poměrně moderní a pokrývají požadavky neméně moderních webových aplikací. Práce s MySQL je rychlá a snadná. Při spouštění serveru MySQL a vytváření databáze jsem se nesetkal s žádnými problémy. Kroky a metody implementace se však mohou na různých operačních systémech lišit, takže může dojít například k chybě kvůli nedostatku potřebného softwaru. Díky výše popsaným alternativním technologiím pro vývoj webových aplikací si i běžný uživatel může vytvořit vlastní aplikaci (sám již více než rok používám služby MySQL, Amazon atd.). Ve skutečnosti si Directus a Amazon vedly velmi dobře, přičemž doba odezvy serveru byla v průměru 0,17 sekundy. V Directusu jsem ale také našel některé nedostatky, a to nestabilní provoz webového rozhraní a „špatnou“ dokumentaci. Problém byl také s Amazonem — složitá konfigurace služeb (hlavně kvůli jejich velkému počtu).

Vytvoření databáze nebylo mé poprvé a databáze implementovaná jako součást této práce nebyla vytvořena od nuly. Všechny tabulky byly umístěny do stávajícího projektu. Jedná se o projekt internetového portálu o České republice v ruštině (https://thecesko.cz). Z hlediska osobní zkušenosti s databázemi jsem souběžně s psaním této bakalářské práce vyzkoušel NoSQL databáze od Google. Na základě této malé zkušenosti musím říct, že NoSQL skutečně plně implementuje své rozdíly od SQL.

Práce provedená a popsaná v této práci již bohužel není zcela relevantní. Od 2. srpna 2021 došlo v České republice ke změnám zákona č. 326/1999 Sb., O pobytu cizinců na území České republiky. Pozměněn § 180 o pojištění pro cizince. Prezident republiky podepsal zákon, který stanoví monopol na pojištění cizinců společnosti pVZP. To znamená, že nyní pouze tato společnost bude mít právo pojistit cizince. Z tohoto důvodu by současná aplikace vypadala jinak. Změnil by se E-R model (odstranění vztahu), částečně by se změní logický a fyzický model. Mezi tabulkou *hospital* a tabulkou *company* již není potřebná relace, to již nedává smysl. Protože zbývá pouze jedna společnost, je logické předpokládat, že bude spolupracovat se všemi nemocnicemi na seznamu.

Při psaní této bakalářské práce jsem neměl problém se shromažďováním informací. Protože v současné době jsou informační technologie, zejména databáze, nedílnou součástí moderního světa, lze jakékoli informace nalézt v online knihách a na internetových portálech. Věřím, že veškeré poznatky získané při tvorbě bakalářské práce a navržená řešení využiji i v budoucnosti ve své další práci.

12 Použité zdroje

[1] vectorjuice. Grafické prvky, které jsou součástí obrázků. Dostupné z WWW: <https://freepik.com/vectorjuice>

[2] Adobe. Informace o webových aplikacích. 2021 [cit. 2021-05-19]. Dostupné z WWW: <https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html>

[3] Web Server Technology [online]. 1996 [cit. 1996-03-15]. Dostupné z WWW: <https://books.google.cz/books?id=0jExRH3_-hQC&vq=%22Web+server%22+-wikipedia&hl=ru&source=gbs_navlinks_s>

[4] What is an App Server. Co je aplikační server. Dostupné z WWW: <https://www.theserverside.com/news/1363671/What-is-an-App-Server>

[5] Inside ODBC. Co je Open Database Connectivity (ODBC) [online]. 1995 [1995-06-01]. Dostupné z WWW: <https://openlibrary.org/books/OL786609M/Inside_ODBC>

[6] Definice databáze. Dostupné z WWW: <https://aleph.nkp.cz/F/?func=direct&doc_number=000000089&local_base=KTD>

[7] Encyklopedie databázových technologií (RUS). [online]. 2005 Dostupné z WWW: <https://ideas.repec.org/b/scn/updvmx/book-12.html>

[8] Typy SŘDB. Dostupné z WWW: <https://www.oracle.com/ru/database/what-is-database>

[9] INFORMAČNÍ TECHNOLOGIE. DATABÁZE (RUS). [online]. 2009 [2009-11-01]. Dostupné z WWW: <http://www.hi-edu.ru/e-books/xbook690/01/eabout.htm>

[10] Wikipedie. Geografický informační systém. Dostupné z WWW: <https://cs.wikipedia.org/wiki/Geografický_informačn%C3%AD_systém>

[11] Mapbox. Dostupné z WWW: <https://docs.mapbox.com>

[12] Co je relační a nerelační databáze (RUS). Dostupné z WWW: <https://smoff.ru/howitworks/otlichiya-sql-nosql>

[13] Data Types. Typy dat. Dostupné z WWW: <https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

[14] MySQL. Dokumentace MySQL. Dostupné z WWW: <https://dev.mysql.com/doc/mysql-startstop-excerpt/8.0/en>

[15] Amazon Tutorial. Pokyny ke spuštění serveru. Dostupné z WWW: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html>

[16] Creating an Amazon RDS DB instance. Stvoření RDS. Dostupné z WWW: <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html>

[17] Installing from CLI. Instalace Directus a vytvoření funkčního projektu. Dostupné z WWW: <https://docs.directus.io/guides/installation/cli/>

[18] Directus Docs. Dokumentace. Dostupné z WWW: <https://docs.directus.io>

[19] Directus.io. Obrázek zobrazující průběh instalace projektu na serveru. Dostupné z WWW: <https://docs.directus.io/getting-started/quickstart>

13 Seznám obrázků

14 Seznám tabulek

15 Seznám příloh