FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Virtuální historický průvodce - jádro
Student:	Vít Černý
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Týmový projekt Virtuální historický průvodce má za cíl uživatelům přívětivě vizualizovat historii architektury českých měst.

1. Analyzujte funkční a nefunkční požadavky na jádro od tvůrců klientských částí. Zaměřte se zejména na komunikační API a datové úložiště.
2. Dle předchozích požadavků navrhněte metodami softwarového inženýrství jádro celého projektu, které bude přes komunikační vysoce dostupné API poskytovat uložená data.
3. Navrhněte administrátorské GUI pro konfiguraci a monitoring jádra.
4. Implementujte prototyp jádra pomocí následujících jazyků a technologií: NodeJS, RestAPI, PostgreSQL.
5. Zaveďte podporu průběžné integrace včetně průběžného testování.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 29. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Virtuální historický průvodce - jádro

Vít Černý

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

14. května 2018

Poděkování

Děkuji především své rodině za podporu, svému vedoucímu za to, že to se mnou vydržel, a stack overflow za to, že mi pomohl s většinou chybových hlášek.

Prohlášení

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

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

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Černý, Vít. *Virtuální historický průvodce - jádro*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce je věnována analýze potřeb a prostředků pro vývoj jádra systému projektu Virtuálního historického průvodce, který má zjednodušit práci odborníkům (zvláště historikům) a zpříjemnit návštěvu historických památek široké veřejnosti. V práci jsou zevrubně analyzovány některé z nejpoužívanějších databázových systémů. Dle hodnotících metrik vybraných na základě funkčních a nefunkčních požadavků a předchozí práce na projektu jsou jednotlivé systémy ohodnoceny a nejvhodnější je pak vybrán pro následnou implementaci. Důkladně je popsána technologie replikace, její potřebnost pro projekt a její možné realizace. V neposlední řadě jsou rozebrány možnosti monitoringu celého jádra. Hlavním cílem této práce není vytvořit plně funkční jádro projektu, ač je funkční prototyp implementován, ale sestavit obsáhlý manuál, základní kámen činnosti pro následující generace řešitelů projektu.

Klíčová slova virtuální realita, historické modely budov, jádro systému, databázový systém, replikace, RestAPI, PostgreSQL, Node.js

Abstract

This thesis is devoted to analysis of needs and resources for developing the core of Virtual historical guide project, which is intended to simplify the work of specialists (especially historians) and to make it mroe pleasant for the general public to visit historical monuments. The thesis analyzes in detail some fo the most used database systems. According to evaluation metrics selected on the basis of functional and non-functional requirements and previous work on project, the individual systems are evaluated and the most appropriate is then selected for subsequent impementation. The technology of replication, its necessity for the project and its possible realizations are described in detail. Last but not least, options for monitoring of the whole core are discussed. The main goal of this work is not to create a fully functional core of the project, although a functional prototype is implemented, but the main objective of this thesis is to create a comprehensive manual, foundation stone for the next generation of project solvers.

Keywords virtual reality, historical models of buildings, system core, database system, replication, RestAPI, PostgreSQL, Node.js

Obsah

Seznam obrázků

Úvod

Projekt Virtuální historický průvodce je zaměřen na vytvoření obsáhlého systému průvodce historickými částmi měst. Tento projekt má návštěvníkovi památek zpříjemnit jeho zážitek a zjednodušit jeho přístup k zajímavým a důležitým informacím. Projekt má též sloužit jako globální nástroj pro historiky a jiné odborníky, co se týče snadného přístupu k informacím a jejich správě. Projekt Virtuálního historického průvodce pracuje s velkým množstvím informací a modelů různých historických budov. Tyto modely je třeba nějak efektivně ukládat, přistupovat k nim, spravovat je a zároveň je vyžadována spolehlivost a vysoká dostupnost tohoto úložiště. K úložišti musí mít možnost přistupovat jak samotná aplikace, kterou budou používat běžní uživatelé, tak grafici, kteří budou vytvářet a upravovat modely budov. Cílem práce je tedy vytvořit nejen datové úložiště, ale i API, jehož prostřednictvím se k němu bude přistupovat. Součástí výsledné práce je i monitoring komunikace mezi API a databázovým systémem. V rámci analýzy se porovnávají různé dostupné technologie, zvláště pak databázové systémy, které jsou následně detailně popsány. Autor při přípravě bakalářské práce spolupracoval s dalšími řešiteli jiných částí projektu Virtuálního historického průvodce.

Kapitola 1

Cíl práce

- Analyzovat funkční a nefunkční požadavky na jádro od tvůrců klientských částí. Zaměřit se zejména na komunikační API a datové úložiště. Po konzultaci s kolegy pracujícími na stejném projektu a z podkladů nasbíraných při práci na projektu v předmětech BI-SP1 a BI-SP2 analyzovat různé technologie pro tvorbu databázového systému a API, se zaměřením především na databázový systém. Dle hodnotících metrik vycházejících z funkčních a nefunkčních požadavků porovnat různé nejpoužívanější systémy a vybrat z nich ten nejvhodnější pro použití v tomto projektu. Hlavním cílem práce je tedy vytvořit manuál. Komplexní analýza prostředků a možností by měla být dostatečným odrazovým můstkem pro další řešitele, kteří budou na projektu Virtuálního historického průvodce pracovat.
- Dle předchozích požadavků navrhnout metodami softwarového inženýrství jádro celého projektu, které bude přes komunikační vysoce dostupné API poskytovat uložená data. Je potřeba navrhnout jak jednotlivé entity, tak strukturu API pro komunikaci s databází. Při návrhu entit se zohlední připomínky od ostatních členů týmu, kteří s daty z databáze nějakým způsobem pracují v jiné části projektu.
- Navrhnout administrátorské GUI pro konfiguraci a monitoring jádra, případně vyhledat co nejvhodnější již existující program s volně šiřitelnou licencí, který monitoring obstará.
- Implementovat prototyp jádra pomocí vhodných technologií.
- Zavést podporu průběžné integrace včetně průběžného testování. Není klíčové otestovat všechny prvky, stačí pouze ukázat, jakým způsobem by se průběžné testy realizovaly.

Kapitola 2

Analýza a návrh

2.1 Analýza

V této části představím různé databázové systémy a technologie, které se pro potřeby projektu využijí. Systémy mezi sebou vzájemně porovnám dle různých hodnotících metrik, která jsou pro tento konkrétní projekt klíčová. Na závěr se věnuji funkčním a nefunkčním požadavkům na systém, které do značné míry vycházejí z mé analytické práce na projektu Virtuálního historického průvodce v předmětech BI-SP1 a BI-SP2.

2.1.1 Databázový systém

Nejprve musíme zvolit vhodný databázový systém. Pro tyto potřeby jsem vybral čtyři nejrozšířenější databázové systémy podporující replikaci (**MySQL**, **PostgreSQL**, **Oracle Database** a **MongoDB**), které jsou zároveň dostupné pod bezplatnou licencí, a vzájemně je níže porovnal. Cílem není zjistit, který z těchto systémů je objektivně lepší - to by bylo zdlouhavé, náročné a patrně nemožné. Cílem této analýzy je vybrat nejlepší databázový systém dle kombinace hodnotících metrik, které jsou pro tento konkrétní projekt nejdůležitější. Za klíčové hodnotící metriky považuji:

- Replikace

  Jednou z hlavních podmínek na jádro systému projektu je vysoká dostupnost. Je třeba, aby případný výpadek či nutná údržba databáze neznamenala kompletní výpadek služby. Toto zajistíme zavedením replikace databázového systému - databáze se rozkopíruje na několik vzájemně nezávislých úložišť. V případě výpadku je pak okamžitě k dispozici náhradní, funkční a naprosto totožné úložiště. V rámci replikace rozlišujeme několik různých termínů, které zde před samotným porovnáváním vysvětlím:

  – Master-slave model

Z replikovaných úložišť jedno zvolíme za hlavní a s ním provádíme komunikaci a read-only operace. Toto úložiště označíme za mastera. V případě, že je třeba provést nějakou write operaci, master pak, synchronně či asynchronně, promítne tuto změnu na ostatní datové úložiště, tzv. slaves. V případě, že dojde k výpadku či nutné odstávce mastera, jsou pak na každém slave úložišti k dispozici totožná data v totožném stavu jako na původním master úložišti.

- Master-master model (či multi-master model)

  Narozdíl od master-slave modelu v multi-master modelu reagují na klientské dotazy všechna úložiště a všechna následně updatují data na ostatních úložištích. Toto logicky však může vyvolat nekonzistenci dat, se kterou je v praxi složité se vyrovnat. Takovéto systémy tedy bývají jen částečně konzistentní, neboli líné. Navíc vyhodnocování konfliktů mezi datasety je poměrně složité a náročné, lze tedy očekávat značný nárůst komunikační latence.

- Asynchronní replikace

  Zápis do databázového systému je považován za ukončený, jakmile daná jednotka, s kterou probíhá komunikace, tento zápis potvrdí. Ostatní jednotky jsou pak aktualizovány s určitým zpožděním, což v případě výpadku hlavního úložiště může způsobit nekonzistentnost dat - pokud se některé změny nestihly projevit na replikované datasety.

- Synchronní replikace

  U synchronní replikace na rozdíl od asynchronní jednotka, s kterou probíhá komunikace, čeká na potvrzení aktualizace od všech ostatních replikovaných jednotek, než sama vyšle potvrzení o zápisu. Toto zajišťuje nulovou ztrátovost dat, protože jestliže je write operace potvrzena jako ukončená, je jistota, že je daná informace zapsána na všech jednotkách. Problém nastává v případě výpadku komunikace mezi jednotlivými replikovanými databázemi. Pak, z předpokladu výše, nelze dále provádět žádné write operace (jelikož jedna nebo více jednotek nemůže potvrdit aktualizaci). V praxi se tedy nulová ztrátovost zpravidla potlačuje a v případě výpadku komunikace mezi některými z úložišť se pokračuje pouze lokálním zápisem.

  [1]

- Zabezpečení

  V této části porovnám zabezpečení samotné databáze, šifrování atributů, možnosti autentizace a případné bezpečnostní hrozby daných systémů.

- Dokumentace

Jedním z hodnotících metrik je dokumentace - jak kvalita, kvantita a přehlednost oficiální dokumentace, tak dostupnost různých neoficiálních tutoriálů a návodů.

- Nástroje pro správu a monitoring

  Zde hodnotím kvalitu a dostupnost různých nástrojů pro správu a monitoring databázového systému.

- Podpora

  Podporované operační systémy a podporovaná propojení s programovacími jazyky, konkrétně s jazyky NodeJS, Python, C/C++, Java, PHP

- BLOB

  BLOB, neboli Binary large object je označení pro datový typ blíže nespecifikovaných binárních dat uložených v databázi. Obecně se většinou jedná o obrázky, videa či zvukové záznamy, které běžně bývají uloženy v samostatných souborech. V této sekci tedy budu porovnávat, jak jednotlivé systémy řeší ukládání blobů či pokud to vůbec obecně podporují. [2]

2.1.1.1 MySQL

Obrázek 2.1: MySQL logo

MySQL je široce rozšířený systém řízení báze dat pod vlastnictvím společnosti Sun Microsystems (dceřiná společnost Oracle Corporation). Je k dispozici pod bezplatnou licencí GPL. MySQL je napsáno v C a C++ a vytvořila ho švédská společnost MySQL AB. První verze spatřila světlo světa v roce 1995. V době vzniku této práce byla nejnovější verze MySQL 5.7.21 z 15.1.2018. [3]

- Replikace

  Replikace je v MySQL řešena standardně asynchronně, otroci nemusí být připojeni k masterovi neustále a podle konfigurace lze replikovat všechny databáze, vybrané databáze či pouze tabulky v databázi. MySQL dále

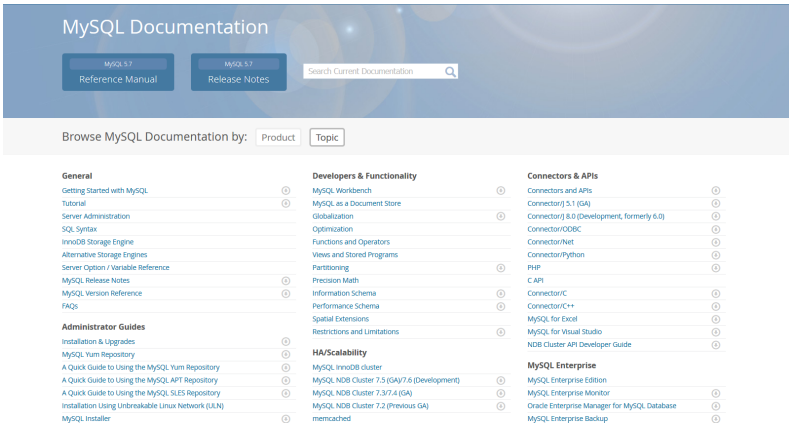
nabízí například MySQL Cluster edici, která řeší replikaci synchronně mezi nody, slibuje 99.999% dostupnost, a bohužel stojí 10 000 $, tudíž pro potřeby našeho projektu ji nebudeme zvažovat.

- Zabezpečení

  MySQL podporuje šifrování komunikace mezi klienty a serverem pomocí TLS protokolu, který se často označuje za SSL protokol (ač MySQL přímo SSL protokol kvůli slabému šifrování nepoužívá). TLS pro verifikaci identity využívá X509 standard. MySQL se vždy nejprve pokouší navázat šifrované připojení, v případě neúspěchu pak naváže nešifrované spojení. Z hlediska autentizace MySQL neumožňuje uživateli, který má práva na vytvoření či zrušení tabulky, odebrat práva na vytvoření či zrušení celé databáze. Dále pak heslo každého uživatele platí globálně na účet. Nelze tedy dané heslo svázat pouze se specifickým objektem, jako je databáze, tabulka nebo rutina. MySQL navíc nabízí řadu bezpečnostních pluginů spravujících autentizaci, connection-control či bezpečné úložiště citlivých informací.[4]

- Dokumentace

  Oficiální dokumentace MySQL[1] je poměrně rozsáhlá, přehledná a snadno se v ní orientuje. Obsahuje jak obecnou dokumentaci o možnostech MySQL, tak i konkrétní návody na vytvoření různých konektorů v různých jazycích, včetně příkladů.[5]



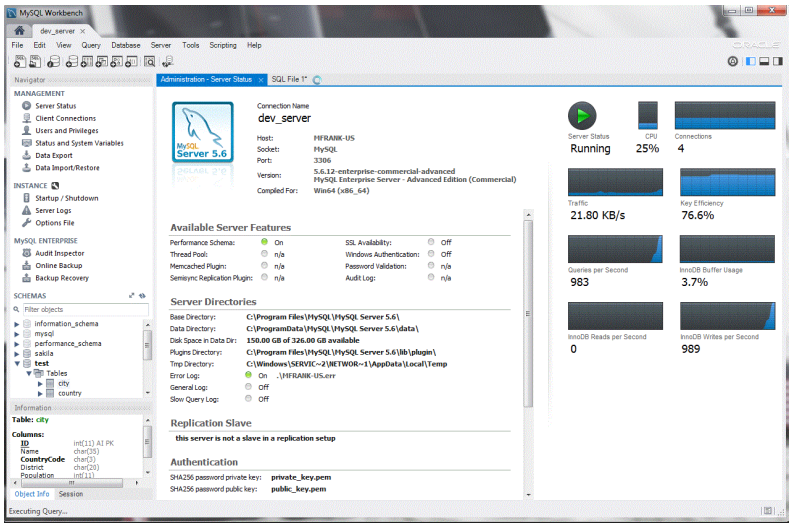
Obrázek 2.2: Dokumentace MySQL

- Nástroje pro správu a monitoring

  Mezi nejrozšířenější programy pro správu a monitoring MySQL databází patří například MySQL Workbench, spravovaný přímo společností

[1] https://dev.mysql.com/doc/

Oracle. Další populární možnosti jsou například HeidiSQL, SQLyog či phpMyAdmin. Celkově je výběr různých programů pro ulehčení práce s MySQL databázemi poměrně veliký.



Obrázek 2.3: MySQL Workbench

- Podpora

  MySQL podporuje většinu nejrozšířenějších operačních systémů - kompletní seznam je dostupný v dokumentaci[2]. V dokumentaci se dále nachází komplexní návody jak implementovat do svého projektu Connector, který se připojí k MySQL databázi. Tyto návody jsou k dispozici pro JDBC, ODBC, .NET, Python, PHP, C, C++,... Co se týče NodeJS, tak jednoduchý návod k připojení je například na w3schools[3].

- BLOB

  MySQL podporuje ukládání dat ve formátu BLOB a pracuje s nimi jako s binárními stringy. [6]

[2]https://www.mysql.com/support/supportedplatforms/database.html
[3]https://www.w3schools.com/nodejs/nodejs_mysql.asp

2.1.1.2 PostgreSQL

Obrázek 2.4: PostgreSQL logo

PostgreSQL je relační databázový systém vyvíjený společností PostgreSQL Global Development Group, který je pověstný svou spolehlivostí a vysokou bezpečností. Je k dispozici pod licencí typu MIT a je napsaný v C. PostgreSQL se vyvinul z projektu Ingres na University of California, Berkeley. Původní název tohoto projektu byl Postgres (po Ingresu - tedy Post-Ingres) a byl vyvíjen mezi lety 1986-1994. V roce 1995 se nahradil Postgresový dotazovací jazyk POSTQUEL za SQL a vznikla první verze dnešního PostgreSQL. Aktuální verze PostgreSQL 10.3 z 1.3.2018.[7]

- Replikace

  PostgreSQL nabízí velké množství různých řešení replikace a clusteringu - synchronní a asynchronní, metodou master-slave aj.

Program	License	Maturity	Replication Method	Sync	Connection Pooling	Load Balancing	Query Partitioning
PgCluster	BSD	Not production ready	Master-Master	Synchronous	No	Yes	No
pgpool-I	BSD	Stable	Statement-Based Middleware	Synchronous	Yes	Yes	No
Pgpool-II	BSD	Recent release	Statement-Based Middleware	Synchronous	Yes	Yes	Yes
slony	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Bucardo	BSD	Stable	Master-Master, Master-Slave	Asynchronous	No	No	No
Londiste	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Mammoth	BSD	No longer maintained	Master-Slave	Asynchronous	No	No	No
rubyrep	MIT	No longer maintained	Master-Master, Master-Slave	Asynchronous	No	No	No
Bi-Directional Replication	PostgreSQL (BSD)	Recent release	Master-Master (no triggers needed)	Asynchronous	No	No	No
pg_shard	LGPL	Recent release	Statement-based Middleware (as an extension)	Synchronous	No	Yes	Yes
pglogical	PostgreSQL	Recent release	Master-Slave	Asynchronous	No	No	No
Postgres-XL	PostgreSQL	Recent release	MPP Postgres, scalable writes & reads	Synchronous	Yes	Yes	Yes
Citus	AGPL	Recent release	MPP Postgres, scalable writes & reads	Asynchronous or Synchronous	Yes	Yes	Yes

Obrázek 2.5: Různé možnosti PostgreSQL

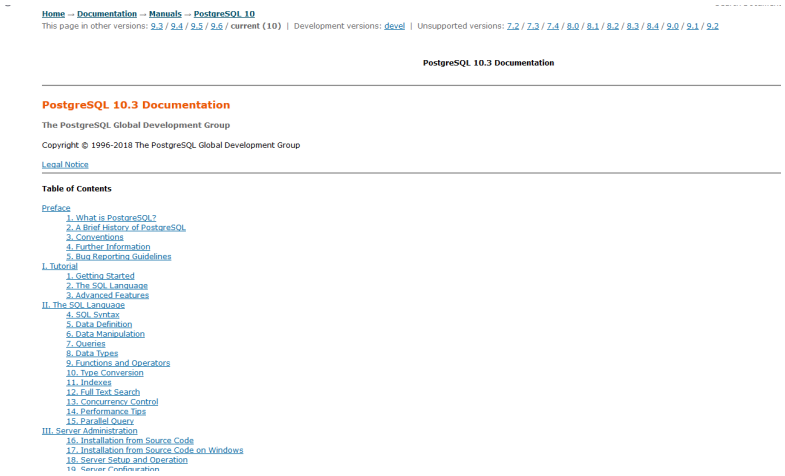
- Zabezpečení

  PostgreSQL má velmi flexibilní permissions systém, který dovoluje uživatele přidělit do nějaké skupiny uživatelů, kteří mají určitá práva. Jeden uživatel může patřit do více skupin. Toto dovoluje přesně vymezit, kam

má který uživatel přístup. Dle výchozího nastavení nemá uživatel přístup pro zápis do databází, které nevytvořil.[8]

- Dokumentace

  Oficiální dokumentace PostgreSQL[4] je podobného rozsahu jako dokumentace MySQL, špatně se v ní však orientuje a celkové GUI je velmi nepřehledné. Za lepší zdroj informací spíše považuji oficiální wiki[5], která obsahuje i různé návody a tutoriály, ač na tom je s přehledností podobně jako oficiální dokumentace. Za zmínku stojí ještě daleko přívětivější stránka se základními návody, tutoriály a fundamentálními informacemi - PostgreSQL Tutorial[6].



Obrázek 2.6: PostgreSQL oficiální dokumentace

- Nástroje pro správu a monitoring

  Na PostgreSQL wiki[7] lze nalézt rozsáhlý seznam dostupného open source software, sloužícího k administraci a monitoringu PostgreSQL databází. Mezi nejrozšířenější patří velmi přehledný pgAdmin, dále stojí za zmínku například TeamSQL či Adminer.

- Podpora

  PostgreSQL lze zprovoznit na všech konvenčních OS. PostgreSQL pak nabízí jedny z největších knihoven různých API dostupných pro různé programovací jazyky. Podporuje tedy všechny z dříve zmíněných jazyků, a ač jsem přímo v dokumentaci nenašel žádné návody či tutoriály, na

[4] https://www.postgresql.org/docs/
[5] https://wiki.postgresql.org/wiki/Main_Page
[6] http://www.postgresqltutorial.com/
[7] https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools

internetu lze takovýchto step-by-step návodů najít mnoho, včetně některých přímo na oficiální PostgreSQL wiki.

- BLOB

  PostgreSQL umožňuje ukládání velkých binárních objektů dvěma metodami - BLOB a bytea. Bytea je binární řetězec, který se od SQL standardu BLOB lehce liší. Není u něj nutné, na rozdíl od blobu, držet si stranou Object ID (a separátní metadata, jakému OID náleží co). Nevýhodou je nutnost při ukládání a načítání escapovat či zakódovat binární objekt. Navíc může lehce narůst paměťová náročnost i na malém vzorku dat.[9]

2.1.1.3 Oracle Database

Obrázek 2.7: Oracle Database logo

Oracle Database (nebo zkráceně Oracle) je systém řízení báze dat vyvíjený společností Oracle Corporation. Vývoj Oracle Database začal roku 1977 pod záštitou společnosti Software Development Laboratories. Oracle je nyní jedna z hlavních společností vyvíjející databázové systémy. Oracle Database má mnoho předností, ať už to jsou rozšířené možnosti zpracování dat, vysoký důraz na bezpečnost s pravidelnými bezpečnostními updaty, podporu multimodelů aj. Co se týče licence, je Oracle Database placený systém, s výjimkou trial verzí a Oracle Database 11g Express Edition, která je v době vzniku práce omezena na 11 GB uživatelských dat, 1 GB paměti a jedno CPU. To je pro tento projekt zatím dostatečné. Poslední stabilní verze je 12c Release 2 (12.2.0.1) z 1.4.2017.

- Replikace

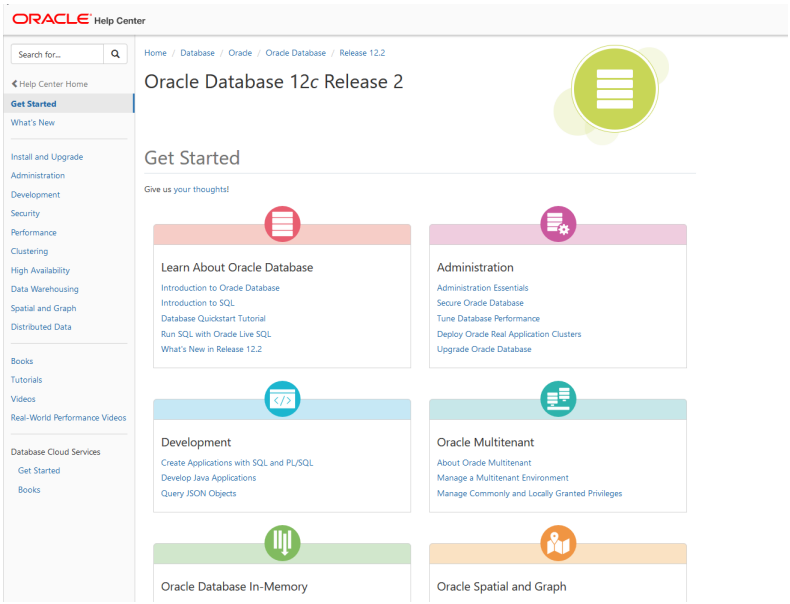
  Oracle defaultně nabízí dva druhy replikace. Základní replikaci (master-slave), kterou podporuje využitím takzvaných read-only table snapshotů, a pokročilou replikaci (master-master), které dosáhne využitím několika konfigurací. U pokročilé master-master metody Oracle databáze automaticky řeší potenciální problém s nekonzistencí dat. Je tedy zaručena jak konzistence, tak integrita replikovaných dat.[10]

- Zabezpečení

Tři základní bezpečnostní prvky Oracle databáze jsou Oracle Advanced Security, Oracle Database Vault a Oracle Label Security. Oracle Advances Security poskytuje nástroje pro preventivní ochranu choulostivých dat u zdroje - šifrování a redakci. Oracle Databse Vault pak řeší práva databázových uživatelů a omezuje vnější i vnitřní útoky. Oracle Label Security zavádí label based access control, pomocí něhož lze přesně určit, kdo může upravovat které sloupce či řádky v tabulce.

- Dokumentace

  Oficiální dokumentace Oracle Database je přehledně rozdělena do kategorií dle funkcionalit. Dále zde najdeme celou sekci věnovanou různým tutoriálům, výukovým videům, dostupné literatuře a dokonce sekci real-world výkonnostních videí.



Obrázek 2.8: Oracle Database Dokumentace

- Nástroje pro správu a monitoring

  Mezi nejpoužívanější nástroje pro správu Oracle databází patří zejména Oracle SQL Developer či Oracle Enterprise Manager přímo od Oracle. Mezi další oblíbené programy musíme zařadit TOAD, DBVisualizer či SquirellSQL. Jelikož je Oracle database primárně placená, je i většina nástrojů pro její správu placená.

- Podpora

  Oracle Database podporuje všechny konvenční operační systémy jako Linux, Windows, Mac OS, Solaris ap. Podporuje zároveň i všechny výše

zmíněné jazyky, konkrétně pak pro Python, Node.js a PHP existují přímo Oracle Database drivery, dostupné z oficiálních stránek.

- BLOB

  Oracle DB kromě standardních blobů podporuje i datový typ BFILE. Jedná se v podstatě o link, ukazující na místo ve filesystému, kde je daný binární soubor uložen.

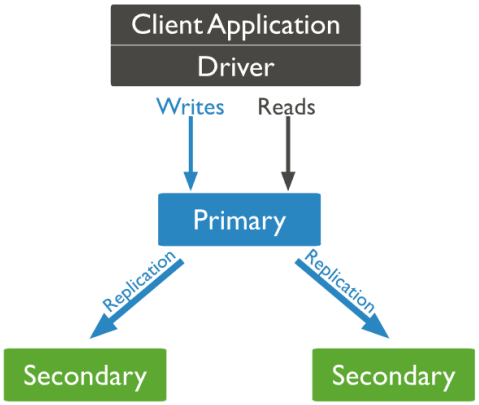
2.1.1.4 MongoDB

Obrázek 2.9: MongoDB logo

MongoDB je dokumentová NoSQL databáze vyvíjená společností MongoDB Inc. v jazyku C++. Z porovnávaných systémů se jedná o nejmladší databázový systém, jehož vývoj započal v roce 2007. Nejzásadnější rozdíl oproti MySQL a PostgreSQL je v tom, že MongoDB nevyužívá pro uchovávání dat ani tabulky, ani samotný jazyk SQL (odtud termín NoSQL). Namísto toho pro ukládání dat využívá dokumenty, formát podobný JSON a dynamické databázové schéma. MongoDB je k dispozici pod bezplatnou licencí GNU General Public License. Aktuální verzí MongoDB je verze 3.6.3 z 23.2.2018.[11]

- Replikace

  MongoDB řeší replikaci přes tzv. sady replik, což je skupina mongod procesů, které spravují stejný data set. Jedná se o master-slave princip, řešení je asynchronní s možností přidání tzv. arbitra, který nenese kopii data setu, ale v případě výpadku pomáhá při hlasování o novém masterovi. [12]



Obrázek 2.10: Řešení replikace v MongoDB

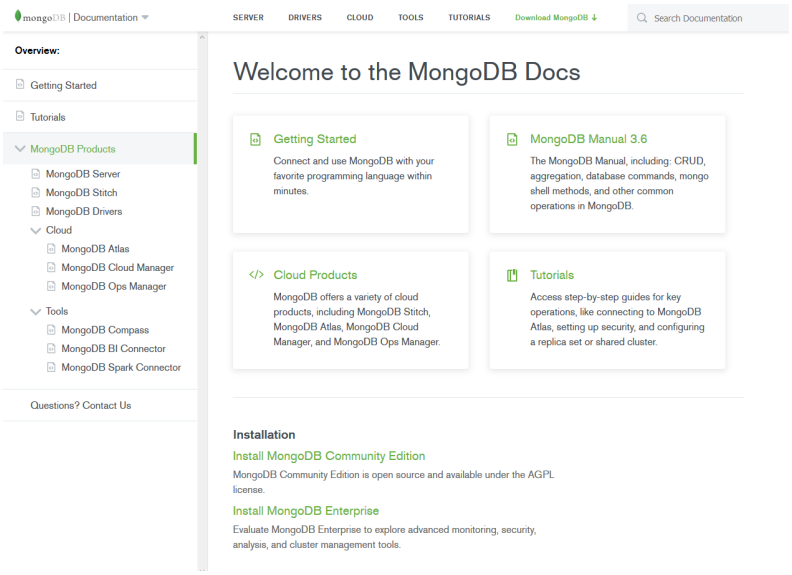
- Zabezpečení

  MongoDB používá TLS/SSL šifrování s využitím OpenSSL knihoven. MongoDB šifrování povoluje pouze silné TLS/SSL šifry s minimálně 128 bitovým klíčem pro všechna připojení. MongoDB využívá pro přístup k datům role - každý uživatel může mít jednu či více rolí, každá role uděluje uživateli různá oprávnění provádět různé akce nad zdroji. Zdrojem rozumíme databázi, kolekci, set kolekcí či cluster. MongoDB dále využívá pro autentizaci ve výchozím nastavení SCRAM, ve kterém používá hashovací funkci SHA-1.[13]

- Dokumentace

  MongoDB dokumentace[8] by se dala nejlépe popsat slovním spojením "user-friendly". Z dokumentací porovnávaných databázových systémů ji považuji za nejpřehlednější a nejobsáhlejší. Obsahuje mimo jiné tutorialy, ukázky kódů, ale i přímo plně funkční a stažitelné drivery pro komunikaci různých jazyků s MongoDB.

[8] https://docs.mongodb.com/



Obrázek 2.11: Dokumentace MongoDB

- Nástroje pro správu a monitoring

  Pro MongoDB obdobně jako pro systémy výše existuje nepřeberné množství různých GUI prostředí, které pomáhají se správou a monitoringem systému. Za zmínku stojí například MongoDB Compass, MongoBooster, Studio 3T či NoSQLclient.

- Podpora

  MongoDB podporuje většinu konvenčních OS - Windows 7 a novější, Mac OS 10.7 a novější, Debian, Solaris, Ubuntu aj. Co se týče podpory propojení s jazyky, již jsem výše zmínil dostupnost různých driverů přímo v oficiální dokumentaci. Tyto oficiální drivery jsou k dispozici pro C, C++, C# a .NET, Javu, Node.js, Perl, PHP, Python, Ruby a Scala.

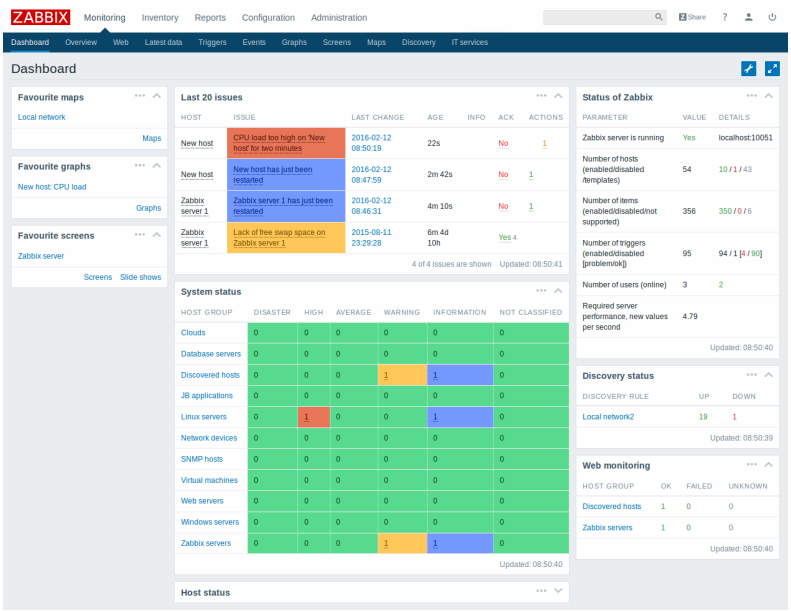
- BLOB

  MongoDB ukládá velké binární objekty pomocí takzvaného BSON (binary JSON). Jelikož běžné MongoDB objekty jsou limitovány velikostí 4 MB na záznam, je blob rozdělen na několik objektů menších než 4 MB. Vedlejší výhodou je větší efektivita při čtení části tohoto objektu.[14]

2.1.2 Monitoring

Jedním z požadavků na systém je i monitoring. Jedná se o součást zabezpečení chodu systému. Kromě možných cílených útoků zvenčí, které má na starosti bezpečnost, může dojít i k vnitřním problémům, které ohrozí či zastaví chod

systému. Tyto potenciální problémy je třeba monitorovat, předejít jim či je odhalit dříve, než se stanou kritickými. Monitorovat lze velké množství různých atributů, od vytížení CPU, paměti, odezvy, místa na disku, zdraví jednotlivých složek systému až k detekci přehřívání atd. Z hlediska softwarového inženýrství je nejvýhodnější využít některý z již existujících open source nástrojů pro monitoring systému, kterých je dostupné poměrně velké množství, jsou přehledné, jednoduché na nastavení, monitorují veškeré možné a potřebné charakteristiky systému a umožňují například i rozesílání emailů či SMS zpráv v případě výskytu kritického problému. Mezi tyto nástroje patří například Zabbix, Icinga, Nagios, Cacti či další. Po dohodě s vedoucím bakalářské práce jsme se rozhodli využít pro monitoring Zabbix.



Obrázek 2.12: Zabbix monitoring

2.1.3 RestAPI

Obrázek 2.13: REST Api

REST znamená Representional State Transfer, tento termín prvně použil a popsal ve své disertační práci Roy Fielding v roce 2000. Jedná se o architektonický styl, který je často používán při vývoji webových služeb. Jeho význačným rysem je přístup ke zdrojům - implementuje 4 základní metody, známé pod zkratkou CRUD - Create, Read, Update a Delete. Rest je služba používající textový formát (XML, JSON,...) a spoléhající především na HTTP jako transportační protokol. RESTové aplikace mají následující charakteristiky:

- Separace klienta a serveru

  Pro architektonický styl REST je příznačné, že je oddělena implementace klienta a serveru. Mohou být na sobě nezávislé a nemusí o sobě nijak vědět. Díky tomu lze upravovat kód klientské části bez toho, aby byla nějak ovlivněna serverová a vice versa. Jedinou podmínkou je, že obě strany musí vědět, jakým typem zpráv se sebou mají komunikovat.

- Bezstavovost

  Systémy pracující pod RESTovým paradigmatem jsou takzvaně bezstavové - server nemusí vědět, v jakém stavu je klient, a naopak. Díky tomu mohou navzájem reagovat na jakékoli zprávy, bez toho, aby znali obsah předchozích zpráv. [15]

- CRUD

  CRUD, tedy Create, Read (Retrieve), Update a Delete (Destroy) jsou čtyřmi základními funkcemi perzistentního úložiště. Ve vývoji RESTful API se tyto funkce nejčastěji mapují na HTTP metody, konkrétně Create na PUT či POST, Read na GET, Update na PUT, POST či PATCH a Delete na DELETE.[16]

Po konzultaci s vedoucím bakalářské práce byla jako technologie pro komunikaci s databázovým serverem zvolena Node.js.

2.1.4 Node.js

Obrázek 2.14: Node.js logo

Node.js vytvořil v roce 2009 Ryan Dahl a jedná se o open-source multiplatformní javascriptový environment, který spouští JavaScript na serverové části. Podporuje vývoj webového serveru čistě pomocí JavaScriptu a kolekce různých modulů pro základní funkcionalitu, jakou je file system I/O, síťování, binární data, kryptografie, datové streamy a další. Node.js lze spustit jak na Linux, Windows, tak na MacOS. Node.js je designově podobný systémům, jako je Event Machine pro Ruby a Twisted pro Python.

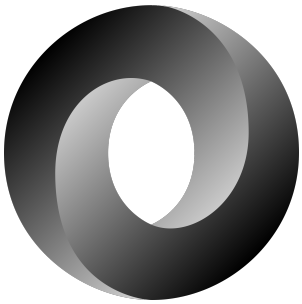
- Threading

  Mezi hlavní výhody Node.js patří fakt, že běží na jednom vlákně, přičemž používá neblokující I/O volání, čímž podporuje desetitisíce souběžných připojení bez nutnosti neustálého přepínání kontextu. Zároveň jsou odstraněny veškeré obavy z deadlocku systému, jelikož k žádnému zamykání nedochází.

- Podpora

  Pro Node.js existují tisíce open-source balíčků, většinou dostupných na stránce npm. Node.js komunita je velmi rozsáhlá, pořádá různá komunitní setkání, vytváří velké množství výukových videí, tutoriálů či fór. Vytvořila též velké množství různých frameworků, které dále usnadňují vývoj aplikací.[17]

2.1.5 JSON



Obrázek 2.15: JSON logo

V práci bude využit objektový zápis JSON (Javascript Object Notation). JSON byl prvně specifikován začátkem druhého tisíciletí Douglasem Crockfordem a slouží k přenosu datových objektů sestávajících z páru atribut-hodnota a datových typů typu pole. JSON je jazykově nezávislý, odvozený z JavaScriptu a většina jazyků nabízí kód pro jeho generování a parsování.[18]

2.1.5.1 Datové typy

JSON podporuje několik datových typů:[18]

- Number

  Číslo se znaménkem, povolena exponenciální E notace, nepovoleno NaN. Číslo musí začínat číslem (nikoliv např. ".5").

- String

  Nula a více Unicode znaků. Řetězce jsou odděleny dvojitými uvozovkami a podporují escapování znaků.

- Boolean

  Podporuje hodnoty true a false.

- Array

  Seřazený list s nula či více hodnotami, které mohou být různých typů. List používá notaci hranatými závorkami a jednotlivé prvky jsou odděleny čárkou.

- Object

  Neseřazená kolekce klíčů-hodnot, kde jsou klíče řetězce. Klíč je od hodnoty oddělen dvojtečkou, kolekce je ohraničena stejně jako Array složenými závorkami. Klíče by měly být unikátní (ale nemusí).

- null

  Prázdná hodnota, klíčové slovo null.

2.1.6 Funkční a nefunkční požadavky

2.1.6.1 Funkční požadavky

- Přístup k datům dle role uživatele

  Je nutné rozdělit přístup do databázového systému dle role uživatele. Koncový uživatel, který bude finální program využívat, potřebuje jen načítat vhodné modely z DB, zatímco grafik, který modely vytváří, musí mít prostředky a prostředí jak pro načítání a ukládání, tak i pro testování svých vytvořených modelů a textur.

- Ukládání velkých multimediálních dat, textur

  Je třeba vyřešit problém ukládání velkého množství poměrně velikých datových struktur - modelů budov a textur.

- Vhodné ukládání informací o modelu

  Je třeba vhodně zvolit strukturu ukládaných dat, protože se zaznamenává velké množství informací o modelu a texturách. Zajímají nás nejen informace o samotné textuře či modelu, ale také při jaké příležitosti se má daný model zobrazit. Je textura zasněžená? Je přizpůsobená zobrazení v nočních hodinách? Jmenovitě nás tedy bude zajímat především: roční doba, denní doba, počasí, výkon zařízení a výkon připojení zařízení, vzdálenost od objektu, datace modelu a v neposlední řadě, jestli se jedná o developerskou, testovací či finální verzi modelu.

- Schopnost nabídnout koncovému uživateli nejvhodnější model a texturu

  Velmi často dojde k situaci, že nebude v databázi přesný model budovy s přesnou texturou, kterou si vyžádá uživatel. Je třeba vyřešit situaci, kdy bude nutné vybrat některou z méně vhodných kombinací model-textura, která co nejlépe splní požadavky. Je potřeba, aby takové řešení bylo rychlé - je tedy nutné zavést nějakou rutinu, která zařídí, aby uživatel dostal k dispozici data, která budou pokud možno co nejbližší původnímu dotazu. Ku příkladu, pokud bude chtít uživatel slunečný model v létě, který v databázi nebude, nejspíše bude vhodné nabídnout mu spíše slunečný model na jaře, než zasněžený model v létě.

- Monitoring komunikace a systému

  Je vhodné monitorovat komunikaci mezi databázovým systémem a jednotlivými API, pro lepší detekci potenciálních problémů.

2.1.6.2 Nefunkční požadavky

- Vysoká dostupnost

  Je potřeba, aby byla služba vysoce dostupná. Pro minimalizování případných výpadků je tedy nutné zavést replikaci, aby se případná plánovaná či neplánovaná odstávka neprojevila na koncovém uživateli.

- Rychlost odezvy komunikačního API

  Je třeba, aby server reagoval rychle. Uživatel nesmí na obdržení nebo aktualizování modelu čekat.

- Datová nenáročnost při komunikaci s koncovým uživatelem

  Je třeba minimalizovat množství dat posílané uživateli, jelikož je aplikace cílena převážně na mobilní uživatele, kteří mají z pravidla omezené množství dat.

- Rozšiřitelnost

  Systém musí umožňovat případné rozšíření o další funkcionality či připojení k dalším API.

2.2 Návrh

V této sekci se budu věnovat samotnému návrhu. Navrhnu zde datové úložiště a API pro komunikaci s koncovým uživatelem. Dále se zde nalézá Big Picture celého projektu a popsaný diagram aktivit nasazení modelu budovy.

2.2.1 Big picture

Takzvaný Big Picture systému, viz Obrázek B.1. Na obrázku je zobrazen nástin kompletního systému. Úložiště sestává z databázového systému a file systému, kam se ukládají modely budov a textury. Toto úložiště komunikuje prostřednictvím API s koncovým uživatelem a dodává mu konkrétní modely budov a textur. Dále má k úložišti prostřednictvím jiného API přístup grafik, který jednotlivé modely vytváří. Posledním API zmapovaným na obrázku je API pro komunikaci s AV ČR, která bude do systému dodávat různé historické informace o budovách, místech, atd. Veškerá komunikace databázového úložiště je monitorována.

2.2.2 Datové úložiště

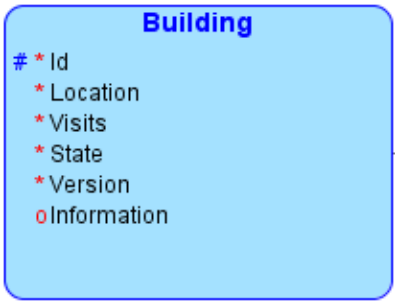
Pro potřeby projektu budeme ukládat informace o modelech budov. Každá budova (tedy určité geografické místo) má jeden či více svých modelů, které jsou vázány právě na tuto budovu. Každý model pak může mít jednu či více různých textur pro konkrétní počasí a roční či denní dobu. Vznikají nám tedy

tři základní entity - budova, model a textura. Entity byly upraveny na základě připomínek Heleny Pavlíkové, která zpracovávala jinou část tohoto projektu. [19]

2.2.2.1 Entity

- Building

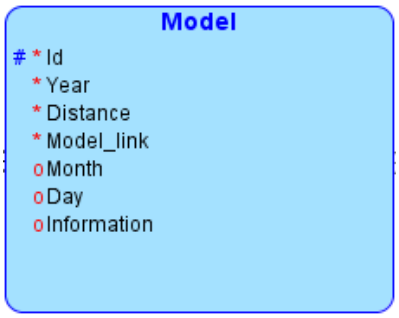
  Každá budova bude mít několik klíčových atributů. Kromě id bude třeba znát přesnou lokaci dané budovy. Nepovinným atributem pak bude textová informace o dané budově a počet navštívení (pro případný žebříček nejoblíbenějších uživatelských destinací). Každá budova bude znát svůj stav, který bude nabývat jedné z hodnot: developerská, testovací či produkční. Jedná se o ukazatel, v jakém stavu je daná budova - pracuje-li se na ní ještě, testuje-li se její funkčnost či je-li možné ji zobrazit cílovému uživateli. Dalším atributem je verze modelu, pro účely verzování.



Obrázek 2.16: Building entita

- Model

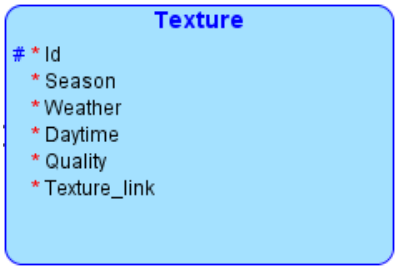
  Model bude určen identifikačním číslem. Dalším důležitým atributem je rok daného modelu (jelikož rokem se většinou budou jednotlivé modely stejné budovy lišit). Pokud by rok nestačil (měli bychom více různých modelů budovy v témže roce), připravíme si pro model i další atributy měsíc a den. Předposledním důležitým atributem je vzdálenost od objektu. Na základě vzdálenosti se také bude model budovy měnit. Samozřejmě nesmí chybět odkaz do file systému na samotný model budovy. Dále, stejně jako budova, bude mít i konkrétní model budovy nějaký textový popis, opět nepovinně.



Obrázek 2.17: Model entita

- Texture

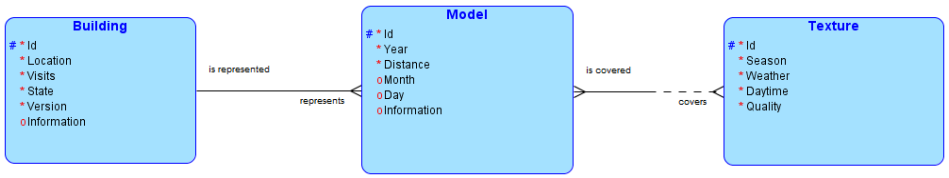
  U textury je identifikačních atributů více. Kromě číselného id texturu jednoznačně určuje roční období, počasí, denní doba a kvalita, tedy je-li to textura ve vysoké kvalitě, či nějakým způsobem zjednodušená (zejména pro nekvalitní připojení či méně výkonné zařízení). Stejně jako u modelu nesmí chybět odkaz na umístění ve file systému.



Obrázek 2.18: Texture entita

2.2.2.2 ER model

Ze zadání nám vyplývá i kardinalita vztahů. Vztah budova - model je vztahem 1:N, model - textura je vztahem N:M. Co se povinnosti vztahů týče, budova musí mít alespoň 1 svůj model a každý model musí být vázán na budovu. Povinný vztah je tedy vztah z obou stran. Na druhou stranu model musí mít alespoň jednu texturu k dispozici, texturu však můžeme v databázi ukládat nenavázanou na konkrétní model.



Obrázek 2.19: ER model

2.2.3 API pro komunikaci s koncovým uživatelem

API pro komunikaci s uživatelem bude koncovému uživateli předávat správné modely budov s korektním potažením texturou, která bude buď přesně vyhovovat aktuálnímu prostředí, v kterém se uživatel nachází, nebo bude alespoň nejbližší možná. API pro komunikaci s koncovým uživatelem bude navrženo dle výše zmíněné RESTful architektury. Bude se jednat o čistě GETovské API, jelikož je samozřejmě nežádoucí, aby uživatel nějak upravoval či dokonce mazal modely budov či nějak upravoval jejich atributy. Pro API tedy budou navrženy dva endpointy, reagující na HTTP metodu GET.

- /buildings

  Tento endpoint slouží pro výpis všech dostupných budov, pro různé potřeby procházení seznamu budov, vypsání informací o budovách, zobrazení počtu návštěv či zobrazení na mapě. RestAPI bude reagovat na metodu GET bez parametrů. Očekávané stavové kódy odpovědí:

  - **200 OK**

    Seznam budov, které jsou v databázi. Informace o budovách, které nás v tomto seznamu budov zajímají (a tedy budou návratovou hodnotou), jsou ID budovy, lokace budovy, počet návštěv budovy, stav vývoje budovy, verse budovy a informace o budově. Žádné další atributy nebudou pro tuto metodu potřeba.

- /building/model

  Tento endpoint vrátí konkrétní model budovy s konkrétní co nejsprávnější texturou. Vrací vždy 1 model, co nejpřesněji odpovídající požadavkům uživatele. RestAPI bude reagovat na metodu GET s parametry co nejpřesněji definujícími požadovaný model. Očekávané stavové kódy odpovědí:

  - **200 OK**

    Vrátí co nejpřesnější objekt, reprezentující model a texturu, dle GET requestu. Tento objekt ponese následující informace - ID modelu, lokace, počet návštěv, stav vývoje, verze, informace o budově,

informace o modelu, počasí, denní doba, roční období, odkaz na model ve file systému a odkaz na texturu ve file systému.

- **400 Bad Request**
  Chyba v syntaxi dotazu.
- **404 Not Found**
  Daný model splňující zásadní atributy (např. lokace) nebyl v databázi nalezen.

2.2.4 Diagram aktivit - nasazení modelu budovy

Každá budova po vytvoření musí projít kolem schvalování. Její stav se postupně mění z developerské, přes testovací až po produkční.To je popsáno v activity diagramu na Obrázku B.2.

- Vytvoření modelu

  Modelář vytvoří model v Blenderu či jiném studiu a vloží ho do systému. Model se tímto nachází v developerské fázi.

- Kontrola integrity modelu

  Systém provede automatizovanou kontrolu integrity modelu. Kontroluje se integrita modelu, výskyt non-manifold geometrie a další možné problémy.

- Kontrola historické správnosti modelu

  Historik si prohlédne model budovy, zkontroluje, že odpovídá historickým materiálům, podle kterých měl být vytvořen.

- Doplnění informací o modelu

  Historik doplní historické informace o modelu. Model přechází do testovací fáze.

- Testování ve vizualizéru

  Tester model převezme a otestuje ho ve vizualizéru, čili tak, jak ho uvidí konečný uživatel. Je třeba ho zkontrolovat ze všech možných úhlů a vzdáleností.

- Nasazení modelu

  Model prošel testováním a je připraven pro zobrazení cílovému uživateli. Přechází do produkční fáze.

- Notifikace modeláře

  V případě neúspěchu testování modelu v některé z fází či v případě velkého množství negativní zpětné vazby na model je notifikován modelář o nutné opravě.

- Oprava modelu

  Modelář opraví vadný model a následně ho nasadí zpět do systému, čímž spustí nové kolo testování.

- Hodnocení modelu a zpětná vazba

  Uživatel má možnost podat zpětnou vazbu na modely. Tato zpětná vazba v podobě hodnocení či slovního popisu je ukládána do systému.

- Sběr zpětné vazby

  V případě většího množství negativní zpětné vazby na model systém notifikuje modeláře, který si zpětnou vazbu projde a v případě nalezení chyby uživatelem model opraví.

Kapitola 3

Realizace

3.1 Diagram nasazení

Diagram nasazení viz Obrázek B.1 ukazuje rozmístění zdrojů a softwarové komponenty, které na nich žijí. Na diagramu můžeme vidět, že databáze poběží na (ze začátku minimálně) třech strojích. Z předpokladu vysoké dostupnosti úložiště se bude databáze replikovat na dva záložní servery, aby se v případě výpadku neomezila dostupnost služby či nedošlo ke ztrátě dat. Klientské API bude komunikovat s databází přes RestAPI napsané v Node.js.

3.2 Instalační příručka

V instalační příručce je popsáno nastavení systému pro úspěšný chod programu. Předpokládáme čistý stroj s operačním systémem Debian GNU/Linux 8.5 či jemu podobný. Předpokládáme, že uživatel instaluje nutné prostředky pod rootem, případně má právo používat sudo.

3.2.1 Node.js

Node.js nainstalujeme dle oficiálního návodu na nodejs.org následujícím způsobem. Pro instalaci je potřeba mít nainstalovaný nástroj curl

```
apt install curl
```

Pak již můžeme nainstalovat samotné Node.js.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
apt-get install -y nodejs
```

Ke zprávnému chodu Node.js budeme ještě potřebovat soubor modulů pro práci s PostgreSQL, které získáme příkazy

```
npm install pg@6.4.0
npm install pg-promise@5 --save
npm install bluebird@3 --save
```

A necháme si vygenerovat strukturu aplikace pomocí Express:

```
npm install express-generator@4 -g
express modely
cd modely
npm install
```

Nyní již máme Node.js nainstalovaný a připravený pro práci s PostgreSQL.

3.2.2 PostgreSQL

PostgreSQL stáhneme jednoduše.

```
apt-get install postgresql-9.4
```

Dále potřebujeme upravit konfigurační soubory PostgreSQL, jelikož jsou defaultně nastaveny příliš přísně pro naše potřeby. Otevřeme si tedy soubor **pg_hba.conf**, který nalezneme ve složce:

```
/etc/postgresql/9.4/main/
```

a přepíšeme řádky

```
local   all             postgres                                peer
```

na

```
local   all             postgres                                md5
```

a

```
local   all             all                                     peer
```

na

```
local   all             all                                     md5
```

Dále ještě upravíme soubor **postgresql.conf** a zde přepíšeme řádek

```
#listen_addresses = 'localhost'
```

na

```
listen_addresses = '*'
```

Tím povolíme připojení z vnějšku.

3.2.3 Účet pro repozitář

Je třeba vytvořit a nastavit nového uživatele, skrz nějž bude probíhat komunikace s repozitářem. Tento uživatel musí mít nastavený přístup pomocí SSH klíče. Nejdříve vytvoříme uživatele bez hesla:

```
adduser --home /home/git-user --disabled-password git-user
```

povolíme mu používat sudo

```
adduser git-user sudo
```

Nyní vygenerujeme dvojici RSA klíčů:

```
ssh-keygen -t rsa
```

a uložíme je do složky nového uživatele:

```
Enter file in which to save the key (...):
/home/git-user/.ssh/id_rsa
```

Nyní stačí přesunout klíč mezi autorizované klíče

```
cat /home/git-user/.ssh/id_rsa.pub >>
    /home/git-user/.ssh/authorized_keys
```

a nastavit potřebná práva

```
chmod 700 /home/git-user/.ssh && chmod 600 /home/git-user/.ssh/*
```

Nyní je uživatel pro přístup k repozitáři nastaven a lze se k němu připojit pomocí private klíče.

3.2.4 Replikace

Po nainstalování databáze je třeba zavést replikaci. Všechny příkazy bychom měli vykonávat jako **postgres** uživatel. Začneme tím, že na master serveru vytvoříme uživatele replication, který bude mít REPLICATION práva.

```
psql -c "CREATE ROLE replication WITH REPLICATION PASSWORD 'heslo'
    LOGIN"
```

Nyní upravíme soubor **pg_hba.conf**, viz výše, a přidáme někam (jinam než na konec) řádek s IP adresou otroka.

```
host    replication    rep    IP_adresa_otroka/32   md5
```

a opět také soubor **postgresql.conf**. Odkomentujeme a přepíšeme následující řádky:

```
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on
```

A restartujeme službu:

```
service postgresql restart
```

Nyní se přesuneme na otroka. Nejprve otroka zastavíme:

```
service postgresql stop
```

Stejně jako v předchozím případě je třeba přidat tento řádek do **pg__hba.conf**

```
host    replication    rep    IP_adresa_mastera/32   md5
```

a znovu upravit **postgresql.conf**

```
wal_level = 'hot_standby'
archive_mode = on
archive_command = 'cd .'
max_wal_senders = 1
hot_standby = on
```

Konfiguraci již máme téměř hotovou, zbývá už jen prvotní nakopírování hlavní databáze z mastera na otroka. To uděláme následujícími příkazy vykonanými na master serveru. Nezapomeneme přepsat IP adresu otroka:

```
psql -c "select pg_start_backup('initial_backup');"
rsync -cva --inplace --exclude=*pg_xlog* /etc/postgresql/9.4/main/
    IP_adresa_otroka:/etc/postgresql/9.4/main/
psql -c "select pg_stop_backup();"
```

Nakonec už jen zbývá vytvořit recovery složku na otrokovi. Přejdeme tedy do složky postgres

```
cd /etc/postgresql/9.4/main
```

a zde vytvoříme nový soubor s názvem **recovery.conf** a obsahem:

```
standby_mode = 'on'
primary_conninfo = 'host=IP_adresa_mastera port=5432
    user=replication password=heslo'
trigger_file = '/tmp/postgresql.trigger.5432'
```

Nyní již zbývá jen spustit otroka:

```
service postgresql start
```

A replikace je zprovozněna.

3.3 Údržbová příručka

3.3.1 PostgreSQL

V případě výpadku PostgreSQL provedeme restart služby pomocí

```
/etc/init.d/postgresql restart
```

V případě výpadku mastera se otrok překonfiguruje na nového mastera, který bude přijímat write operace jednoduše vytvořením souboru **/tmp/postgresql.trigger.5432**. Po opravě původního mastera se pak opakováním kroků výše může tento master zpět připojit jako otrok.

Závěr

V práci byla provedena analýza jednotlivých databázových systémů. Dle stanovených hodnotících metrik byl vybrán nejvhodnější z nich a byl vytvořen prototyp jádra systému Virtuálního historického průvodce. Na základě požadavků ostatních členů projektu byla vytvořena databáze, zavedena replikace a monitoring a definována API pro komunikaci uživatele s touto databází. Výsledek práce a zejména zahrnutá obsáhlá analýza by měly pomoci dalším řešitelům tohoto projektu jako podrobný manuál a komplexní rešerše daného problému. Jako možnost jiné cesty, kterou by se případný další řešitel mohl vydat, je možné zvážit volbu jiného databázového systému, konkrétně MongoDB, který je na trhu databázových systémů nováčkem a nabízí diametrálně odlišné možnosti oproti ostatním systémům.

Literatura

[1] Database Replication. 2018, [Online; accessed 6-April-2018]. Dostupné z: https://docs.oracle.com/cd/A59447_01/nt_804ee/doc/database.804/a58227/ch_repli.htm

[2] Starkey, J.: The true story of BLOBs. 2011, [email; accessed 10-April-2018]. Dostupné z: https://web.archive.org/web/20110723065224/http://www.cvalde.net/misc/blob_true_history.htm

[3] What is MySQL? — MySQL 8.0 Reference Manual. 2018, [Online; accessed 02-April-2018]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[4] The MySQL Access Privilege System — MySQL 5.7 Reference Manual. 2018, [Online; accessed 06-April-2018]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/privilege-system.html

[5] Replication — MySQL 8.0 Reference Manual. 2018, [Online; accessed 10-April-2018]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/replication.html

[6] The BLOB and TEXT Types — MySQL 8.0 Reference Manual. 2018, [Online; accessed 14-April-2018]. Dostupné z: hhttps://dev.mysql.com/doc/refman/8.0/en/blob.html

[7] About PostgreSQL. 2018, [Online; accessed 26-April-2018]. Dostupné z: https://www.postgresql.org/about/

[8] Security — PostgreSQL Documentation. 2018, [Online; accessed 21-April-2018]. Dostupné z: https://www.postgresql.org/docs/7.0/static/security.htm

[9] Binary Data Types — PostgreSQL Documentation. 2018, [Online; accessed 08-April-2018]. Dostupné z: https://www.postgresql.org/docs/9.1/static/datatype-binary.html

[10] Database Replication — Oracle Docs. 2018, [Online; accessed 17-April-2018]. Dostupné z: https://docs.oracle.com/cd/A59447_01/nt_804ee/doc/database.804/a58227/ch_repli.htm

[11] What is MongoDB? 2018, [Online; accessed 08-April-2018]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[12] Replication — MongoDB Manual. 2018, [Online; accessed 17-April-2018]. Dostupné z: https://docs.mongodb.com/manual/replication/

[13] Security — MongoDB Manual. 2018, [Online; accessed 17-April-2018]. Dostupné z: https://docs.mongodb.com/manual/security/

[14] Storing Large Objects and Files in MongoDB. 2018, [Online; accessed 18-April-2018]. Dostupné z: https://www.mongodb.com/blog/post/storing-large-objects-and-files-in-mongodb

[15] What is a REST API? 2018, [Online; accessed 03-April-2018]. Dostupné z: https://www.mulesoft.com/resources/api/what-is-rest-api-design

[16] What is CRUD? 2018, [Online; accessed 03-April-2018]. Dostupné z: https://www.codecademy.com/articles/what-is-crud

[17] About Node.js®. 2018, [Online; accessed 10-April-2018]. Dostupné z: https://nodejs.org/en/about/

[18] Introducing JSON. 2018, [Online; accessed 02-April-2018]. Dostupné z: https://www.json.org/index.html

[19] Pavlíková, H.: *Virtuální historický průvodce - modul virtuální reality*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

GUI Graphical User Interface

BLOB Binary Large OBject

SSL Secure Sockets Layer

TLS Transport Layer Security

MIT Massachusetts Institute of Technology

OID Object Identifier

JSON Javascript Object Notation

XML Extensible Markup Language

HTTP Hypertext Transfer Protocol

REST Representational State Transfer

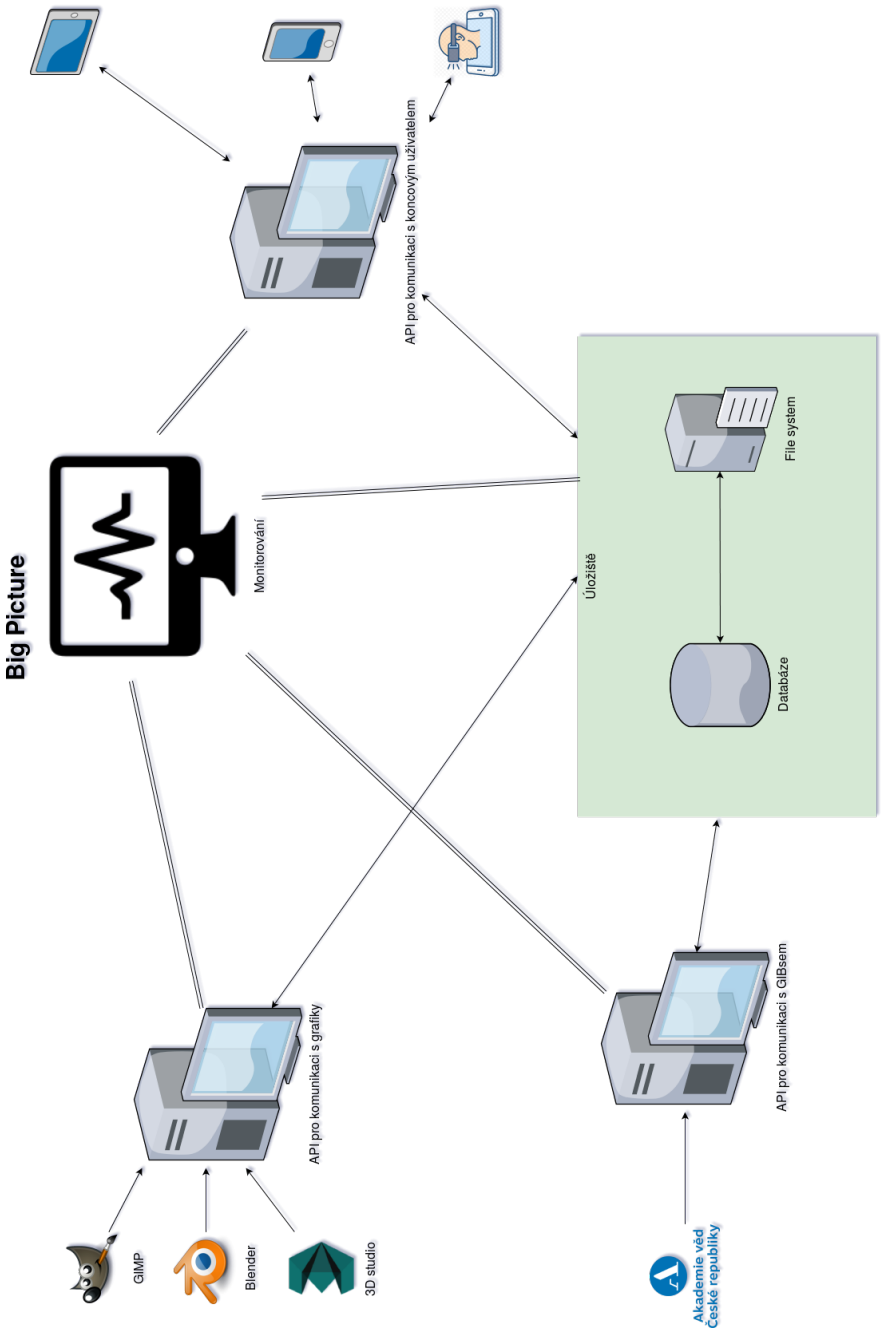
I/O Input/Output

NaN Not a Number

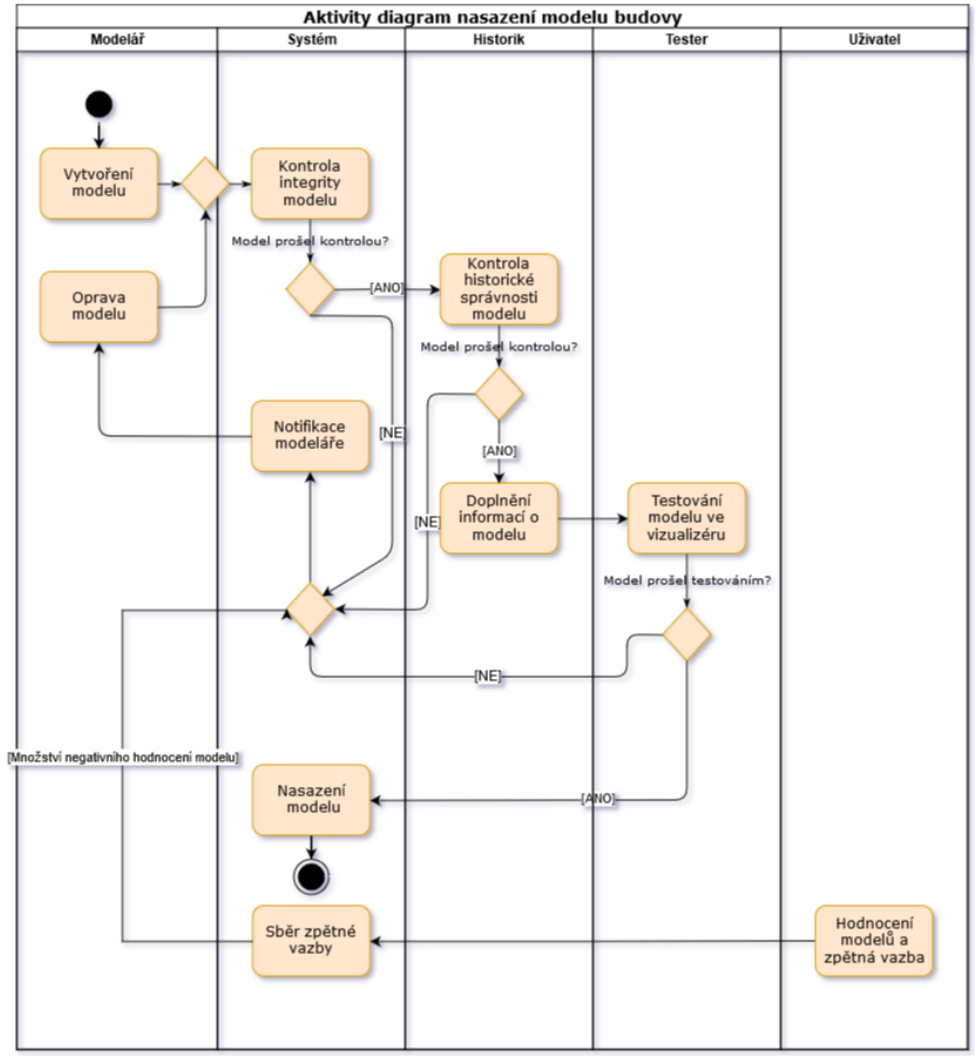
AV ČR Akademie věd České Republiky

Příloha B

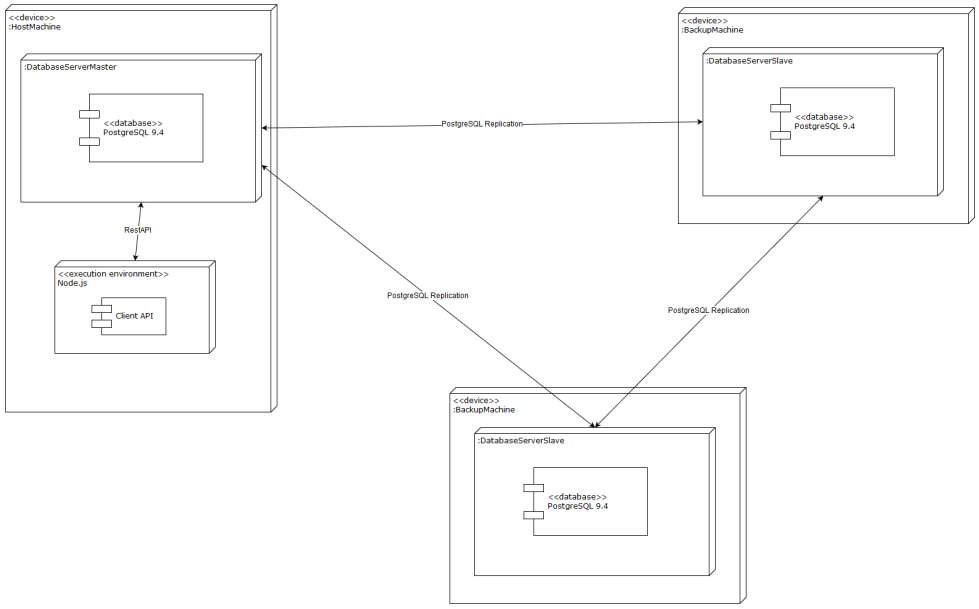
Obrazová příloha



Obrázek B.1: Big picture



Obrázek B.2: Activity diagram



Obrázek B.3: Diagram nasazení

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `bin`
    - `public`
    - `routes`
    - `views`
    - `app.js`
    - `clearDB.sql`
    - `package.json`
    - `queries.js`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF