ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

FAKULTA ELEKTROTECHNICKÁ

Integrace map pokrytí mobilní sítě do prostředí AWS

Integration of Mobile Network Coverage Maps into the AWS enviroment

Diplomová práce

Studijní program: Elektronika a komunikace

Specializace: Mobilní komunikace

Vedoucí práce: Ing. Robert Bešťák Ph.D

Luka Jovanović

Praha 2022

Čestné prohlášení

Tímto prohlašuji, že jsem zadanou práci vypracoval samostatně pod vedením vedoucího práce a že jsem uvedl veškeré použité zdroje v souladu s metodickým pokynem o dodržování etických principů při tvorbě vysokoškolských prací.

Datum: 22.5.2022 ……..…

Luka Jovanović

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jovanović** Jméno: **Luka** Osobní číslo: **473459**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Mobilní komunikace**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Integrace map pokrytí mobilní sítě do prostředí AWS

Název diplomové práce anglicky:

Integration of Mobile Network Coverage Maps into the AWS Environment

Pokyny pro vypracování:

Cílem práce je příprava služeb a datových struktur v prostředí AWS pro účely zpracování dat o pokrytí území signálem mobilní sítě. V rámci řešení bude připravena struktura pro systém DynamoDB k ukládání pokrytí sítě a dále pak REST služby, které zpřístupní funkce pro získání informací o pokrytí sítě. Na závěr bude výsledek demonstrován na jednoduché klientské aplikaci běžící v prostředí web prohlížeče.

Seznam doporučené literatury:

[1] B. Marr. Big Data: Using SMART Big Data, Analytics and Metrics To Make Better Decisions and Improve Performance. Wiley. 2015. ISBN: 978-1-11896-583-2.
[2] Martin Sauter. From GSM to LTE-Advanced Pro and 5G: An Introduction to Mobile Networks and Mobile Broadband. Wiley. 2017. ISBN: 978-1-11934-686-9.
[3] B. Baesens. Analytics in a Big Data World: The Essential Guide to Data Science and its Applications, Wiley. 2014. ISBN: 978-1-11889-270-1.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Robert Bešťák, Ph.D. katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **21.01.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

Ing. Robert Bešťák, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání | Podpis studenta

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Robertovi Bešťákovi, Ph.D za ochotu, trpělivost, pomoc a rady při zpracovávání této diplomové práce.

Abstrakt

Tato práce se zabývá vývojem podpůrných funkcí a algoritmů, které slouží ke zpracování a modelování dat pro webovou aplikaci, poskytující informace o pokrytí mobilní sítí v České republice. Jsou v ní popsány základy tzv. Cloud computingu a Edge computingu a jsou také uvedeny principy použitých služeb v prostředí Amazon Web Services. Dále je uvedeno konkrétní řešení a algoritmy zpracování dat, jejichž funkcionalita je předvedena v rámci webové aplikace. V závěru práce jsou uvedeny analýzy řešení, možné alternativy a budoucí směr vývoje aplikace.

Klíčová slova

Mobilní síť, Zpracování dat, Cloud computing, Edge computing, Amazon Web Services, Aplikace, JavaScript

Abstract

The aim of this diploma thesis is the design of backend functions and algorithms, which are used for processing and modeling of data for a web application, which provides information about mobile network coverage in Czech republic. The first part consists of basic principles of Cloud computing and Edge computing as well as principles of used services within the Amazon Web Service platform. The second part contains the applied solution and algorithms for data processing, whose functionalities are demonstrated on a web application. The conclusion of the thesis consists of solution analysis, alternatives and possible future developments of the application.

Key words

Mobile network, Data processing, Cloud computing, Edge computing, Amazon Web Services, Application, JavaScript

OBSAH

Seznam obrázků

Seznam tabulek

Seznam zkratek

AWS	Amazon Web Services
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
HA	High Availability
DR	Disaster Recovery
IAAS	Infrastructure as a Service
PAAS	Platform as a Service
SAAS	Software as a Service
IoT	Internet of Things
MEC	Mobile Edge Computing
VNF	Virtual Network Functions
NFV	Network Function Virtualization
PK	Partition Key
SK	Sort Key
NoSQL	Not Only Structured Query Language
API	Application Programming Interface
RAM	Random Access Memory
CPU	Central Processing Unit
JSTS	JavaScript Topology Suite
OL	Open Layers
EPSG	European Petroleum Survey Group

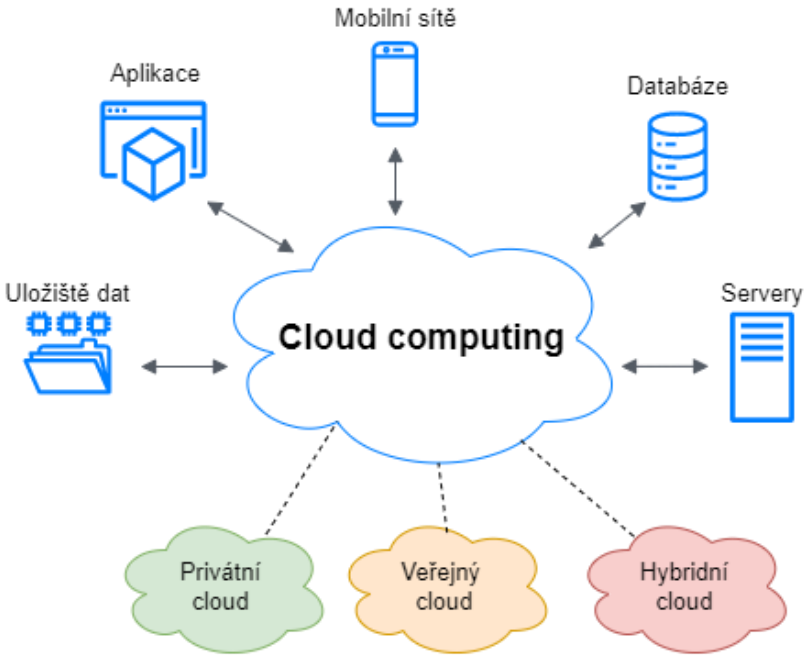
1 Úvod

Vzhledem k vývoji moderních technologií a faktu, že se v dnešní době implementují takřka ve všech branžích a odvětvích s různou funckionalitou, je zřejmé, že množství dat, které je potřeba ukládat a zpracovávat exponenciálně roste. Svět směřuje cestou, kdy každý aspekt našeho života může být analyzován a zlepšován prostřednictvím nashromážděných dat, která jsou sbírána z obrovského množství senzorů a zařízení. Stejný princip je samozřejmě aplikován i v odvětví mobilních sítí, kde data hrají klíčovou roli. Díky sběru tzv. velkých dat, je možné síť a její parametry stále zlepšovat, zpřístupňovat služby většímu počtu uživatelů a zvyšovat i kvalitu samotných služeb (rychlost přenosu dat, kvalita signálu, energetická úspora a mnoho dalších).

Tato práce se zabývá návrhem a implementací podpůrných funkcí pro zpracování dat o pokrytí mobilní sítí v České republice, použitím služeb a cloudových technologií *Amazon Web Services* (AWS). V první části práce je uveden teoretický úvod do problematiky *cloud* a *edge* (okrajových) technologií včetně jejich principu a použití. Dále jsou detailněji popsány konkrétní služby technologií AWS, prostřednictvím kterých byly navrženy algoritmy pro zpracování dat v této práci. V druhé části je nejprve vysvětlena problematika, kterou mají navržené algoritmy řešit. Konkrétně je vysvětlen princip sběru dat o pokrytí a jejich struktura. Dále je pak uvedeno konkrétní řešení, včetně samotných algoritmů pro zpracování dat a je vysvětlena jejich funkcionalita. Poté je navržené řešení demonstrováno v prostředí webové aplikace, která tyto algoritmy aktivně používá. V závěru práce je uveden přínos výzkumu společně s předpokládaným budoucím vývojem dané aplikace.

2 Cloud computing

Vzhledem k vizi a směru, kterým se dnešní technologický svět vydal, je zřejmé, že množství sbíraných a zpracovaných dat bude stále růst. Data jako taková již byla a stále jsou klíčovou součástí takřka každého technologického prvku, jelikož díky nim lze technologii či službu analyzovat, aktualizovat a zlepšovat. Koncepty jako autonomní auta či tzv. chytré domácnosti bazírují nejen na úplné vzájemné konektivitě ale i na analýze velkého množství dat. Vývoj hardware však nepokročil natolik, a je zatím otázka, zda vůbec může, aby bylo možno s veškerými daty pracovat „lokálně“, tj. aby nebylo nutné data transportovat do jiných míst, kde jsou nad nimi prováděny příslušné operace. Z toho důvodu je prakticky ve všech systémech, aplikacích a přístrojích, které pracují s velkým množstvím dat využito tzv. *Cloud computingu* čili zpracovávání dat v „*cloudu*“. Jinými slovy jsou veškeré výpočetní a úložné služby s daty prováděny tzv. *on-demand* (na požádání) prostřednictvím připojení k internetu. Uživatel (či entita využívající cloudové služby) platí poskytovateli za využívané služby a tou cestou dochází k výraznému ušetření jak finančních prostředků, tak času i prostoru [1]. Na obrázku 1 jsou uvedeny nejčastější služby *cloud computing*, přičemž konkrétní pojmy jsou vysvětleny v dalších kapitolách.



Obrázek 1- Služby Cloud computing

2.1 Základní princip

Nejjednodušší způsob, jakým vysvětlit důvod používání *cloud computing*, je srovnáme-li jeho vlastnosti s vlastnostmi lokálního zpracování dat. Aspektů je samozřejmě několik – dále budou vysvětleny ty základní a to:

- Technologické
- Finanční

Scénář k vysvětlení bude použit následující: Existuje firma zabývající se vývojem software, která uvažuje o rozšíření svého podnikání. Software, který se firma snaží vyvinout, může vyžadovat zvýšení výpočetního výkonu (tím pádem více hardware) či rozšíření datového uložiště. Může se jednat například o platformu pro shlížení multimédia podobné Netflixu. Při evaluaci, co vše může tomuto rozšíření bránit se narazí na tři hlavní faktory – omezený počet zaměstnanců, nepředvídatelná poptávka za produktem a omezené prostředky. Před expanzí je tedy brán v potaz scénář, kdy rozšíření proběhne tzv. *on-premise* (při uživateli, bez využití *cloud computing*) a za využití *cloud computing*. Dále jsou uvedeny hlavní aspekty rozvoje, které je nutno brát v potaz.

V následujícím rozboru jednotlivých prvků a výhod použití *cloud computing* je zavedena jednoduchá zkratka vyznačující, zda je použit *cloud computing* (= **CC**) či ne (= **Bez CC**).

2.1.1 Škálovatelnost

Prvním důležitým aspektem je škálovatelnost, neboli schopnost dynamicky měnit (zvyšovat i snižovat) výkon software, náklady, velikost uložiště atd. na základě měnící se zátěže, poptávky apod [1]. Jedná se například o schopnost databází vykonávat procesy při zvýšeném počtu dotazů na data, či jakým způsobem se hardware vypořádá se zvýšeným počtem uživatelů.

Bez CC – Z hlediska škálovatelnosti je řešení při uživateli (*on-premise)* velmi finančně nevýhodné a to proto, že ve chvíli, kdy proběhne tzv. *upscaling* či „škálování nahoru“ (zvýšení kapacity uložiště, výkonu, počtu serverů atd.), je velmi těžké škálovat zpět (bez výrazných finančních ztrát) [5]. Ke škálování nahoru je potřeba rozšířit množství aktivně zapojených serverů a uložišť, což nejen požaduje větší prostor pro tento

hardware, ale zároveň i více energie a úsilí pro provozování. Pokud by poté nastala situace, že v nějaké době bude potřeba opět menšího výkonu, tak již zakoupený hardware bude zbytečně nevyužit a údržba bude stát více peněz [6].

CC – Naopak *cloud computing* nabízí možnost platit pouze za kapacitu, která je v danou chvíli využívána, což umožňuje takzvaný *downscaling* (škálování dolů), tedy snížení množství potřebného uložiště, výpočetní kapacity atd. V této situaci spravuje veškerý hardware poskytovatel cloudových služeb [5]. Zároveň je tento proces dynamické škálovatelnosti vzhledem k dnes dostupným technologiím velmi jednoduchý a rychlý z hlediska uživatele. Požadovanou kapacitu si uživatel stanoví prostřednictvím webového rozhraní, přičemž je tato operace v porovnání s řešením při uživateli (tedy bez *cloud computing*) jednoduchá a finančně výhodná [6].

2.1.2 Datová uložiště

Dalším aspektem je celkové uložiště na serverech. Jelikož většina firem nebo aplikací potřebuje relativně velké množství úložného prostoru pro data, je nezbytné mít možnost kapacitu tohoto uložiště navýšit dle potřeb.

Bez CC– Krom již zmíněných problémů se škálovatelností (tj. navýšení kapacity a poté nastane snížení zátěže – zbytečný hardware), existuje i problém samotného úložného prostoru. Servery fyzicky zabírají relativně dost místa a je potřeba pro ně mít dedikovanou místnost s napájením a dostačující ochranou [5]. Krom toho cena a práce týkající se samotného napájení a údržby není zanedbatelná, proto je i pro serverová uložiště řešení při uživateli nepraktické. Obecně pro toto řešení platí, že jedním z hlavních problémů je neefektivní využití hardware a potřeba za velkými úložnými prostory [6].

CC – I v otázce uložiště umožňují poskytovatelé služeb (Amazon Web Services, Microsoft Azure a další), aby uživatel platil pouze za využívané množství [5]. Zároveň veškerá údržba a spravování serverů je v ceně služby zahrnuta, což ve výsledku ušetří jak peníze, tak i prostor a čas [1][6].

2.1.3 Bezpečnost dat

Otázka bezpečnosti dat je především pro společnosti zacházející s osobními daty klíčová. Zároveň se jedná o téma konstantní debaty vzhledem k vyvíjejícímu se prostředí a výskytu nových hrozeb. Nelze takřka tvrdit, že jakékoliv zabezpečení je 100% bezpečné, avšak cíl je k této hodnotě se co nejvíce přiblížit.

Bez CC – Při rozšíření při uživateli je nutno, aby implementované zabezpečení bylo vskutku správně nastaveno, jelikož je spravováno lokálně. Existuje tedy v rámci společnosti osoba/tým, zodpovídající za zabezpečení a mají nad ním plnou kontrolu. Je-li systém zabezpečení nastavený správně, tak lze argumentovat, že se jedná o lepší řešení než *cloud computing,* ale při chybném zabezpečení mohou být data vystavena hrozbě kybernetických útoků [5]. Zároveň může nastat situace, že o hrozbě či útoku se ani neví. V případě cloudových řešení je tato situace nepravděpodobná, jelikož danou infrastrukturu využívá velké množství různých klientů a otázka bezpečnosti je takřka vždy prioritou [8].

CC – V minulosti nebyla bezpečnost *cloud computingu* dostatečně robustní, avšak dnes je tato možnost stále více využívána. Poskytovatelé služeb se o svá uložiště a servery starají stejným způsobem – existuje dedikovaný tým se striktními procedurami, který má na starost zabezpečení dat, údržbu atd [5]. Je tedy nutno seznámit se s bezpečnostními protokoly daného poskytovatele před tím, než se vybere konkrétní služba. Jelikož je na těchto serverech umístěno obrovské množství dat všelijakého druhu, mohou být pak lákavým cílem pro častější (případně i závažnější) kybernetické útoky [8].

2.1.4 Obnova ztracených dat a údržba

Při návrhu (i rozšiřování podnikání) je nutno brát v potaz i ty nejhorší scénáře. Jakákoliv přírodní katastrofa či závažná mechanická porucha může vést ke ztrátě dat, přičemž této situaci je ideální se vyhnout. Ne vždy je to možné, a proto je třeba mít v záloze i způsob, jak ztracená data obnovit, či minimalizovat škodu.

Bez CC– Pokud neexistuje v jiné lokalitě kompletní záloha všech uložených dat (což je vzhledem k výšce nákladů nepravděpodobné) je složité „zachránit“ jakákoliv ztracená data. Je k tomu potřeba implementovat tzv. HADR (*High Availability Disaster Recovery*) [9]. HA (*High Availability*) je schopnost aplikace vydržet veškeré plánované

(např. softwarová aktualizace) i neplánované výpadky. DR (*Disaster Recovery*) je skupina pravidel, postupů a nástrojů, sloužící k návratu systému zpět do funkce po katastrofě. Zahrnuje procedury kopírování a ukládání klíčových dat do bezpečné lokace a poté i následnou obnovu těchto dat. Jinými slovy je potřeba vždy mít uložiště, sloužící jako záloha (popř. jako aktivní záloha, tj. během chodu aplikace funkční, ale není to primární funkce uložiště) [10]. Z hlediska údržby jsou pak náklady výrazně vyšší, jelikož je potřeba spravovat větší množství serverů a je potřeba financovat kompletní tým lidí s tímto úkolem [8].

CC – V případě *cloud computing* jde paradoxně o principiálně stejný postup, lišící se ve způsobu zálohy. Cloud systémy mají tzv. *Cloud Disaster Recovery* (Cloud DR), který v jednoduchosti „přeskočí“ celý proces zálohování a odstraňuje požadavky na infrastrukturu a hardware [10]. CDR prakticky provede enkapsulaci (zapouzdření) celého serveru – včetně OS (Operační systém), aplikací, dat, aktualizací apod. do virtuálního serveru, který je následně kopírován na geograficky vzdálené uložiště. Jelikož virtuální server nezáleží na hardware, lze takto zapouzdřený server jednoduše a rychle transportovat.

V tabulce 1 je přehledně uveden seznam všech zmíněných aspektů, které je vhodné brát v potaz při rozhodování, zda implementovat *cloud computing* jako řešení pro rozvíjející se firmu/společnost. Pro každý aspekt jsou uvedeny možné výhody a nevýhody, které jsou podrobněji popsány v předchozích odstavcích.

Tabulka 1 - Přehled rozdílů Cloud computing a lokálního zpracování.

Aspekt	**Cloud computing**		**Bez cloud computing**	
	Výhody	***Nevýhody***	***Výhody***	***Nevýhody***
Škálovatelnost	škálovatelné nahoru i dolů, platba pouze za využitou kapacitu	_____	škálovatelné "nahoru"	drahé, nelze škálovat dolů
Uložiště	dynamická velikost uložiště, správa serverů od strany poskytovatele	_____	vhodné pro malá uložiště	potřebný prostor pro servery, cena hardware a údržby
Bezpečnost dat	bez potřeby spravování, podloženo veřejnými protokoly a procedurami poskytovatele	uložiště náchylnější na útoky	vlastní spravování bezpečnosti, plná kontrola	náklady, při chybném zabezpečení velmi zranitelné
Obnova ztracených dat	jednodušší zálohování, nezávislé na hardware, rychlý přenos zapouzdřených dat	_____	potřeba mít duplicitní uložiště sloužící jako záloha – HADR	náklady, údržba
Údržba	výrazně levnější, není potřeba většího počtu prostředků (obstaráno poskytovatelem)	_____	_____	výrazně vyšší cena, potřeba většího počtu prostředků

2.2 Typy modelů Cloud computingu

Klíčový přínos *cloud computing* je tedy to, že přináší veškeré technologické prvky, které nabízí i „tradiční" technologie, avšak ve formě služby. Přístup k těmto cloudovým službám je umožněn prostřednictvím cloudového rozhraní, které poskytuje přístup pomocí REST (*Representional State Transfer*) či SOAP (*Simple Object Access Protocol*) aplikačních rozhraní, popřípadě pomocí webové konzole [11].

Existují dva základní modely *cloud computing*, a to jsou [12]:

- *Deployment model* – model nasazení, popisuje, kdo a do jaké míry má přístup k datům/funkcím dané entity
- *Service model* – distribuční model, který popisuje, jakým způsobem jsou cloudové služby zpřístupněny uživateli

Oba modely se dále dělí dle funkcionality na další části, které jsou vysvětleny v sekcích 2.3 a 2.4.

2.3 Model nasazení

2.3.1 Veřejný cloud

Public cloud neboli veřejný cloud je cloudová služba, která je poskytována jedním z několika poskytovatelů, jenž nabízejí výpočetní a úložní prostředky za poplatek [12]. Jsou mezi nimi například Amazon Web Services (AWS), který je použit v této práci, či Microsoft Azure nebo Google AppEngine [11]. Tento typ cloudu si lze metaforicky představit jako autobus veřejné dopravy – jedná se o veřejně dostupný prostředek, přístupný všem za stejných podmínek.

2.3.2 Privátní cloud

Private cloud neboli privátní cloud je pravým opakem veřejného cloudu – v tomto případě je cloudová infrastruktura spravována pouze jednou stranou, ať už se jedná o osobu, firmu či organizaci [12]. Toho je dosaženo pomocí software jako VMWare, vCloud, nebo OpenStack, což jsou software platformy pro cloudové služby [11]. Údržba samotná je většinou prováděna samotnou organizací či třetí stranou. Z hlediska metaforického přirovnání lze privátní cloud považovat za osobní auto – přístup k němu má pouze majitel, či osoba které majitel auto zpřístupnil.

2.3.3 Hybridní cloud

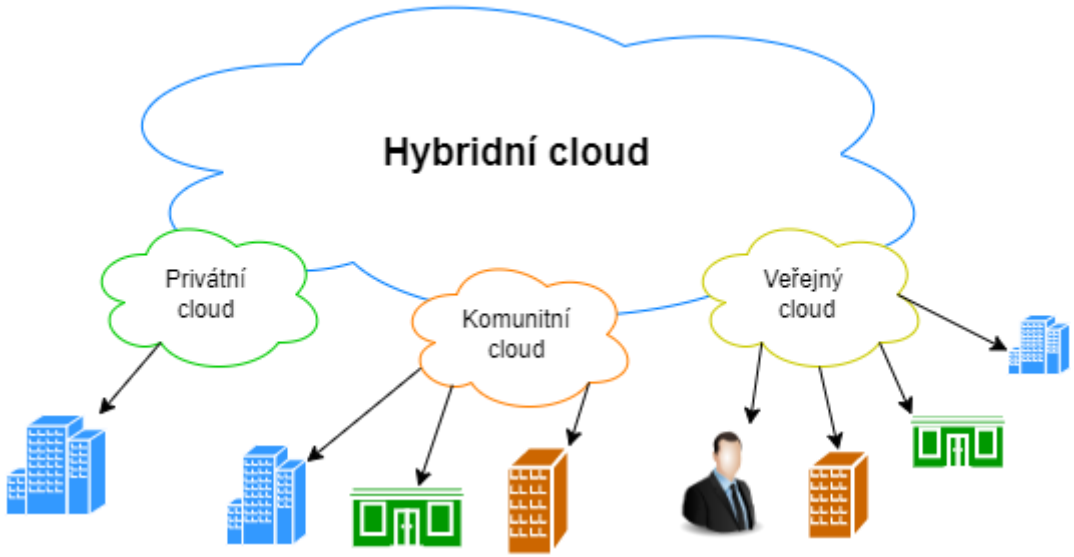
Hybridní cloud je přesně to, čemuž napovídá název – jedná se o mix soukromého a veřejného cloudu [11]. Ten je v porovnání s předchozími modely výhodný v tom, že lze provozovat jako privátní do té doby, než je naplněna jeho kapacita. Nastane-li

taková situace, lze použít veřejný cloud a kapacitu navýšit, tudíž tím není omezena škálovatelnost [12]. Hlavní výhodou hybridního cloudu je možnost rychle se adaptovat na požadavky (především v případě technologických firem), kde dochází ke změnám v potřebách a požadavcích na infrastrukturu.

2.3.4 Komunitní cloud

Community cloud nebo komunitní cloud je zřídka používaný typ provozního modelu. Jedná se o cloud, který je sdílený napříč několika organizacemi a je spravovaný buď interně, či za přítomnosti třetí strany [11]. Využívá se v situacích, kdy určitá skupina má totožné nebo podobné cíle/požadavky a sdílený prostor se může jevit jako nejlepší řešení [12].

Na obrázku 2 je znázorněn rozdíl všech čtyř zmíněných typů modelů nasazení, přičemž každý druh modelu má jinou úroveň přístupu ke cloudu, což je detailněji popsáno v celé sekci 2.3.



Obrázek 2 - Typy modelů nasazení

2.4 Distribuční model

Takzvané *Service models* či distribuční modely určují, jakým způsobem je klientovi zpřístupněna poskytovaná služba [11]. Pokud je veškerá infrastruktura spravována lokálně, musí být spravovány lokálně i její komponenty. Distribuční modely umožnují volnost v podobě spravování pouze některých komponent. Tyto modely se dělí na

základní tři podskupiny, lišící se funkcionalitou. Při porovnání, co vše je potřeba spravovat, je vhodné vědět, o které komponenty se nejčastěji jedná [12]:

- Aplikace
- Data
- *Runtime* – veškerý software potřebný ke kompilaci programovatelných funkcí
- *Middleware* – poskytování funkcionalit nad rámec operačního systému
- Operační systém (OS)
- Virtualizace
- Servery
- Uložiště
- Síťové propojení

Uvedený výpis komponent je vypsán v tomto pořadí, jelikož tvoří jakýsi přechod mezi software a hardware aspekty provozování služeb. Při pohledu na tabulku 2, obsahující úrovně správy pro jednotlivé modely (popsané v sekcích 2.4.1 až 2.4.3), lze pozorovat, že v případě každého modelu se poskytovatel stará o virtualizaci (využívaný hardware ve virtuální podobě), servery, uložiště a síťové propojení. Jinými slovy hardware aspekt služby je vždy obstarán poskytovatelem cloudové služby a modely se liší především v úrovni správy software komponent, kterými jsou aplikace, data, *runtime*, *middleware* a OS.

2.4.1 IAAS

IAAS (*Infrastructure as a Service)* je distribuční model, který poskytuje klientovi přístup k základní výpočetní infrastruktuře [7]. Jedná se o model, který je vhodný v případě, že klient potřebuje přístup k uložišti, *firewallu* (zabezpečení síťového provozu) nebo virtuálním strojům. V IAAS tedy je možno přistoupit k nejnižší úrovni software – k OS virtuálních strojů nebo k bráně *firewallu* [11]. AWS, platforma používaná v této práci, je jedním z největších poskytovatelů IAAS. Co se týče úrovně správy, tak celkové shrnutí je uvedeno v tabulce 2.

2.4.2 PAAS

PAAS (*Platform as a Service*) je model, který vytváří cloud platformu pro vývoj, testování a spravování aplikací [12]. Tento model umožňuje klientovi nasadit aplikaci do běhu, bez potřeby spravování potřebné architektury. Jinými slovy, na základě nasazené aplikace poskytovatel dynamicky alokuje a škáluje potřebné prostředky k správnému běhu aplikace. Typicky PAAS model poskytuje tzv. API (*Application Programming Interface),* neboli jakési rozhraní, sloužící ke spravování využívané platformy [11]. I tento model je nabízený ze strany AWS, přičemž i firma Google nabízí svoji verzi s názvem AppEngine. Úroveň správy PAAS modelu je opět uvedeno v tabulce 2.

2.4.3 SAAS

SAAS (*Software as a Service)* je model, sloužící k tzv. *Hosting*-u (provozování) a spravování software aplikací [12]. V tomto případě se o správu veškeré infrastruktury stará poskytovatel služby, přičemž klient prakticky pouze spravuje svoji aplikaci. V praxi se jedná například o Google Apps, sociální sítě, Dropbox a další. Hlavní rozdíl mezi PAAS a SAAS je, že PAAS je používaný především pro aplikace, které se nacházejí ve stádiu vývoje, zatímco SAAS je využívaný pro již vyvinuté aplikace [11].

V tabulce 2 je přehledně uvedeno, o které komponenty se stará poskytovatel služby či klient pro všechny výše zmíněné modely.

Tabulka 2 - Úrovně správy distribučních modelů

VYSVĚTLIVKA:	**Spravováno poskytovatelem**	**Spravováno klientem**

IAAS	**PAAS**	**SAAS**
Aplikace	Aplikace	Aplikace
Data	Data	Data
Runtime	*Runtime*	*Runtime*
Middleware	*Middleware*	*Middleware*
OS	OS	OS
Virtualizace	Virtualizace	Virtualizace
Servery	Servery	Servery
Uložiště	Uložiště	Uložiště
Síťové propojení	Síťové propojení	Síťové propojení

3 Edge computing

Jak bylo již zmíněno v úvodu práce, veškerá zařízení v dnešní době produkují stále rostoucí množství dat, které je potřeba konstantně zpracovávat. Avšak s postupnou implementací 5G sítí, včetně veškerých předpokládaných inovací, kterými jsou například autonomní vozidla či chytré bydlení není množství dat jediným důležitým faktorem. Klíčovou roli v realizaci těchto inovací hraje i rychlost zpracování dat. Jak je známo, celosvětová síť internetu je enormní a obsahuje nespočet různých tras mezi dvěma destinacemi v síti. Cloud technologie, jelikož se většinou jedná o velká uložiště, která jsou tvořena velkým množstvím hardware, se nemusejí vždy nacházet v obytných oblastech nebo obecně v blízkosti zařízení, produkující data. Skutečnost je taková, že velmi často se nacházejí ve vzdálených lokacích. Pokud je pak nutno implementovat například chytrý semafor, nacházející se v síti autonomních vozidel (kterých může být v provozu v danou chvíli velké množství), je potřeba aby komunikace mezi vozidlem a semaforem probíhala co nejrychleji, což s ohledem na vzdálenou lokalitu cloud serverů nemusí být realitou. Pro takovéto situace existuje tzv. *Edge computing*, neboli zpracovávání dat „na hraně sítě“.

3.1 Princip

Jak i název napovídá, tak *Edge computing* spočívá v tom, že zpracování dat probíhá na periferiích sítě, či jejím okraji. Cílem je zpracování jako takové přiblížit ideálně co nejblíže k samotnému zdroji dat [13]. Jinými slovy, tato technologie se snaží docílit decentralizace zpracovávání dat, která může přinést v určitých aplikacích výrazné benefity. Konvenčním způsobem zpracování je transport dat do centrálního serveru, kde proces proběhne a následného návratu, což z hlediska rychlosti přenosu či výpadku sítě může způsobovat problémy [15]. Princip je tedy poměrně jednoduchý – přiblížit datové centrum ke zdroji dat. Samozřejmě roli hraje množství dat či potřebný výkon hardware. Tyto *edge* technologie bývají často používány u zařízení, která neprodukují enormní množství dat (různé jednoduché senzory, měřiče atd.). Konkrétní oblasti využití jsou uvedeny v sekci 3.4.

3.2 Důvody pro použití Edge computing

Edge computing má za úkol minimalizovat nedostatky *cloud computing* pro určité aplikace. Tyto nedostatky, lze rozdělit na tři základní prvky.

Šířka pásma (*Bandwidth*), která je přímo úměrná přenosové rychlosti, udává v praxi maximální množství přenositelných dat za jednotku času. Samozřejmě platí, že čím větší je dostupné pásmo, tím lepší jsou podmínky pro přenos dat [13]. Veškeré sítě však mají šířku pásma omezenou – to znamená, že existuje určitá přenosová kapacita, která musí být rozdělena mezi daná zařízení. Čím větší jsou požadavky na přenosovou rychlost tím méně zařízení je možno do sítě připojit a naopak. Přiblížením výpočetního hardware ke zdroji dat se minimalizuje využití tohoto pásma (které je většinou maximálně využito pro přenos dat z okolních zařízení), čímž lze dosáhnout stabilnějších a vyšších přenosových rychlostí na krátké vzdálenosti [15].

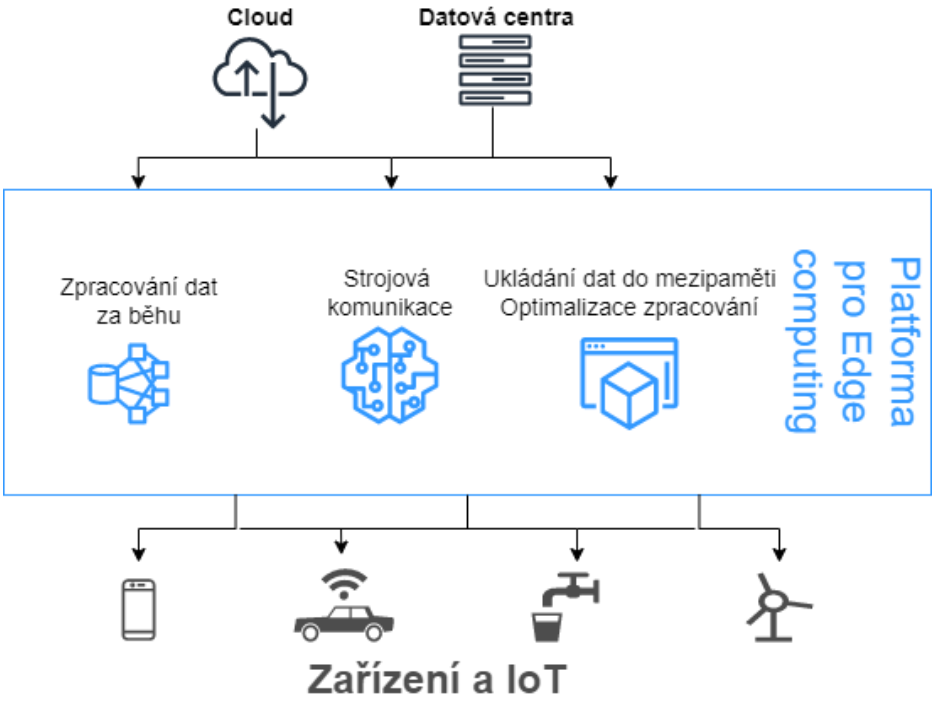
Zpoždění (*Latency*), je definováno jako doba, kterou trvá poslat data mezi dvěma body v síti v jednom směru. Přenos dat v síti sice dosahuje přibližně rychlosti světla, avšak velké vzdálenosti (a tím i přechod přes velký počet síťových prvků), možné výpadky či velká zátěž sítě mohou vést k vyššímu zpoždění [13]. Jak bylo zmíněno v úvodu pro síť autonomních vozidel, kde zpoždění hraje kritickou roli, mohou i malé výkyvy způsobit výrazný problém a v nejzazší situaci i ohrožení života. Zkrácením těchto vzdáleností se umožňuje rychlejší reakce systému na přijatá data a tím pádem i takřka eliminace nečekaného zpoždění [15].

Zatížění (*Congestion*) je parametrem, který je úzce spojený s již zmíněným zpožděním a přenosovou rychlostí. Ačkoliv dnešní internet je tzv. „síť sítí“ a umožňuje stabilní přenos všelijakých dat, není neobvyklý jev, že dochází k tzv. re-transmisím (opětovné poslání stejných dat) [15]. Ty na denní bázi a při každodenních aktivitách nehrají významnou roli a uživatel si jich většinou nevšimne. Z hlediska IoT (*Internet of Things)* však časté re-transmise vedou k většímu zpoždění a v kritických situacích i k výpadkům, čímž se pak tato síť stává nepoužitelnou.

3.3 Rozdíl mezi Cloud a Edge computing

Krom již zmíněného hlavního rozdílu, kterým je fakt, že cloud služby jsou především centralizované na rozdíl od edge služeb, existují i další rozdíly. Jedná se hlavně o místa

implementace. Cloudové služby, jak je zmíněno v kapitole 2 jsou velmi snadno škálovatelné a jsou využívány hlavně pro aplikace, které obsahují velká množství dat a potřebují tato data zpracovávat. Je ale velmi nepravděpodobné, že datové centrum se bude nacházet v blízkém okolí zdroje dat. Edge technologie jsou tedy implementovány pro menší zdroje dat. Dá se paradoxně říci, že edge technologie jsou jakousi komplementární technologií ke cloudu, či tzv. *on-premise* uložištím (uložiště při uživateli, viz kapitola 2.1) [13]. Jedním z příkladů edge technologie je například server umístěný u vzdušné turbíny, který slouží k ukládání a zpracování dat ze senzorů v turbíně. Umístění *edge computing* a jeho primární využití v síti je uvedeno na obrázku 3.



Obrázek 3 – Funkcionalita Edge computing v síti

Na obrázku 3 je znázorněno, jak umístění *edge computing* v síti (tj. blíže k zařízením než cloud, jedná se o mezistupeň), tak i stručné shrnutí, které procesy v rámci *edge computing* platformy mohou probíhat. Obrázek představuje ilustraci toho, že po sběru dat z různých zařízení (senzory, mobilní terminály apod.) jsou nad těmito daty prováděny určité operace už na hraně sítě místo toho, aby data putovala „až“ do datových center či vzdálených cloudů.

3.4 Příklady nasazení Edge computing

Jelikož *edge computing* není vhodný pro velké množství dat, je jeho implementace častá především u menších elektronických zařízení – senzorů, antén apod. Níže uvedené příklady představují jen určitou část možných aplikací *edge computing*.

- **Průmysl** – Jedná se o monitorování, analýzu za běhu a vyhledávání chyb v produkci, které mají vést ke zlepšení a zvýšení efektivity výroby [13]. Data jsou sbírána z různých senzorů, které poskytují například informace o tom, jak je výrobek sestaven či jak dlouho byl odložen ve skladovacích prostorách.

- **Zemědělství** – *Edge computing* je často využíván při pěstování rostlin nebo k analýze půdy. Když rostlina získává méně slunečního světla nebo vody je potřeba použít senzory, sloužící například k měření spotřeby vody, intenzity dopadajícího světla či vlhkosti půdy atd [16]. Data z těchto senzorů slouží k dosažení optimálního pěstování i sběru.

- **Síťový provoz (*Networking)* –** Výpočetní kapacita *edge computing* je aplikována ke zkvalitnění síťového provozu pomocí analýzy provozu a získávání nejadekvátnější trasy pro přenos dat [13]. Tímto způsobem dochází k optimalizaci provozu a omezení výpadků [15].

- **Bezpečnost** – Společné využití senzorů a bezpečnostních kamer, pomocí kterých lze díky analýze dat zajistit bezpečné a přívětivé podmínky v jakékoliv pracovní či jiné budově/lokalitě [13].

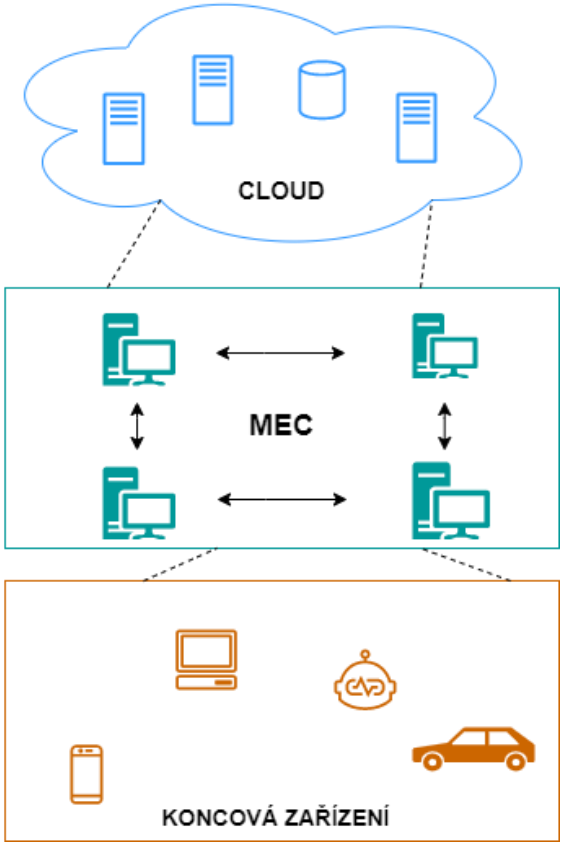
- **Doprava –** Jak již bylo zmíněno, *edge computing* hraje klíčovou roli ve vizi sítě autonomních vozidel. Rychlost vozidel, hustota provozu, lokace, kvalita vozidla, kvalita silnice či stav dopravy jsou několik z mnoha možných parametrů, které bude potřeba zpracovávat a reagovat na ně v tzv. *real-time,* neboli za běhu [15]. Při implementaci takového řešení lze pak dosáhnout například optimální rychlosti a trasování při hustém provozu a zamezit tak zácpám. Jiné implementace mají pak sloužit i k snížení počtu nehod a tím pádem i zvýšení bezpečnosti transportu.

- **Mobilní sítě –** *Edge computing* je jedním z klíčových aspektů aktuálního nasazování a vývoje sítí 5. generace (5G). Operátoři se snaží tzv. virtualizovat čím dál větší části mobilní sítě, pomocí čeho lze výrazně snížit cenu provozování a zvýšit flexibilitu sítě [13]. Jelikož je potřeba vysokého výpočetního výkonu s malým zpožděním je potřeba tento hardware přiblížit co nejblíže zdroji dat. Tato technologie, standardizovaná Evropským ústavem pro telekomunikační normy, je blíže popsána v následující kapitole. *Edge computing* také přispívá ke tvorbě optimální topologie sítě, jejího efektivního využití a spravování připojených zařízení [15].

3.4.1 Mobile Edge Computing

Mobile Edge Computing (MEC) je standardizovaná technologie, sloužící k zvýšení výkonu a flexibility mobilních sítí [13]. Především v nejnovějších 5G sítích má sloužit ke konstantnímu dosažení tzv. *Key Performance Indicators* (KPI) neboli klíčových ukazatelů výkonnosti. Jejich hodnota udává, do jaké míry jsou splněny požadované parametry výkonu [17]. Z hlediska mobilních sítí jde pak především o zpoždění a efektivitu přenosové rychlosti. Jedná se tedy o přirozený krok v procesu evoluce základnových stanic a obecně v konvergenci IT a telekomunikačního sektoru. Krom sofistikovanějších vzdušných rozhraní, mají 5G sítě poskytnout větší přístup programovatelnému software pojetí sítě, které ve spojení s virtualizací informačních technologií a telekomunikační infrastrukturou přinášejí nový přístup k síti jako takové. MEC v této vizí představuje klíčový prvek, sloužící ke transformaci mobilní širokopásmové sítě do programovatelného prostoru se současným dodržením vysokých nároků 5G sítí, konkrétně přenosové rychlosti, zpoždění, škálovatelnosti a automatizace.

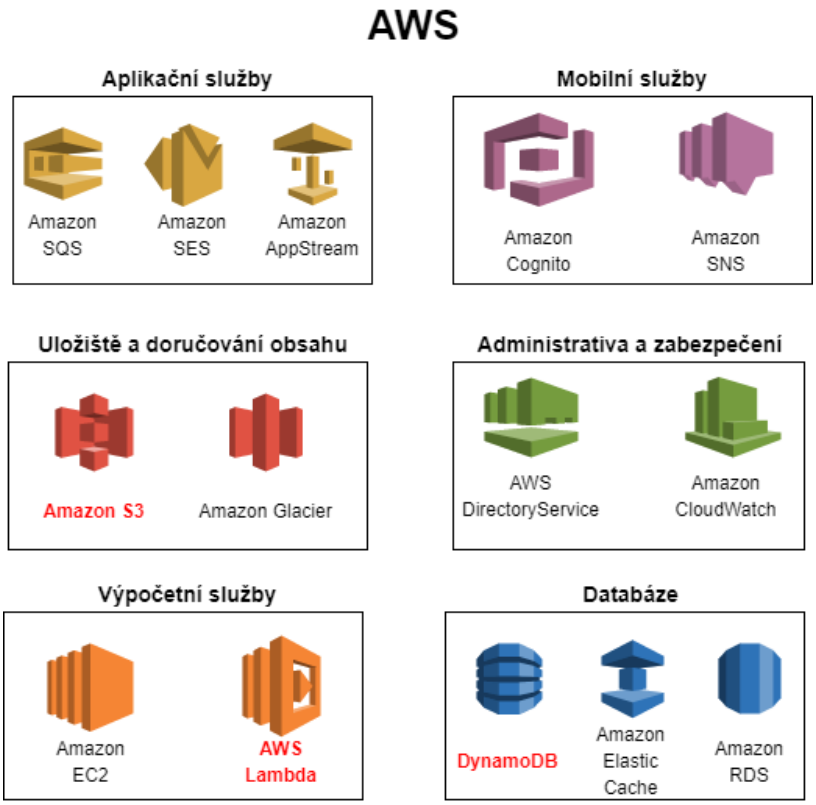
MEC je založeno na virtualizované platformě, podobné takzvanému *Network Function Virtualization* (NFV) [13]. Zatímco NFV slouží především pro síťové funkce, MEC umožňuje provozování aplikací na okraji sítě. Infrastruktury provozující NFV a MEC jsou relativně podobné, a proto bude pro operátory, s cílem ušetřit co nejvíce finančních prostředků, vhodné existující infrastrukturu použít, a to pomocí provozování tzv. *Virtual Network Functions* (VNF) a MEC na stejné platformě [17]. Prostředí MEC lze charakterizovat nízkým zpožděním, blízkostí (*proximity*), vysokou hodnotou přenosové rychlosti a získáváním informací o rádiové síti za běhu (*real-time)*. Na obrázku 4 je ve zjednodušené formě znázorněna lokalizace MEC v síti. V jednoduchosti vyznačují fakt, že v rámci sítě je implementováno prostředí MEC, sloužící ke zvýšení výkonu sítě či její schopnosti rychle zpracovávat data a dosáhnout tak požadovaných parametrů.



Obrázek 4 - MEC v architektuře sítě

4 Nástroje AWS

Amazon Web Services (AWS) nabízí širokou škálu datových, výpočetních a jiných služeb, které jsou vhodné pro jakoukoliv situaci, která může při provozování cloudových aplikací a služeb nastat. Z hlediska modelů *cloud computing*, které jsou uvedené v sekci 2.4, nabízí AWS především služby typu IAAS a PAAS. V této kapitole budou přiblíženy ty, které byly použity pro vývoj algoritmů v této práci. Na obrázku 5 je uvedeno několik druhů služeb/nástrojů, které platforma AWS nabízí. S ohledem na to, že jich je celkově velké množství, jsou zde uvedeny základní typy služeb a několik konkrétních názvů nástrojů. Červeným písmem jsou zvýrazněny ty, které jsou použité v této práci – jedná se o výpočetní službu AWS Lambda, databázi DynamoDB a o uložiště S3.



Obrázek 5 - Typy služeb platformy AWS

4.1 DynamoDB

První důležitou službou, poskytovanou AWS platformou je databáze nesoucí název DynamoDB. Jedná se o NoSQL (*Not Only Structured Query Language*) databázi, která

se vyznačuje jednoduchou a obousměrnou škálovatelností a jedinečným přístupem ke skladování dat [19]. Přístup k datům je založený na identifikaci jednotlivých záznamů v databázi pomocí jedinečných „klíčů“. Tato struktura umožňuje čtení záznamů z jednoduché struktury databáze (například triviální seznam všech zaměstnanců) až po složité struktury, ve kterých jsou záznamy na sobě závislé. DynamoDB je používán pro ukládání menšího množství dat až po obrovská množství s velkým důrazem kladeným na spolehlivost databáze (tj. pokud by došlo k výpadku, nastaly by výrazné finanční ztráty). Příkladem platformy, využívající DynamoDB ke skladování dat je aplikace Lyft, která používá DynamoDB pro ukládání a zpracování všech lokalit pro veškeré jízdy svých řidičů [18]. Další známé platformy, využívající tuto službu jsou například firma Disney s platformou Disney+ pro shlížení multimédia či Dropbox.

4.1.1 Struktura Dynamo DB databáze

Jak bylo zmíněno, záznamy v DynamoDB jsou ukládány na bázi specifických klíčů. Tyto klíče slouží nejen k zjednodušení čtení dat ale i k přehlednému ukládání. Základní ideou je struktura s tzv. *primary key* čili primárním klíčem [18]. Existují dva typy primárního klíče – jednoduchý a složený. Oba typy primárních klíčů obsahují tzv. *partiton key*, označován zkratkou PK. Je-li databáze tvořena pomocí jednoduchého primárního klíče, musí každý záznam obsahovat unikátní *partition key* (PK), který je povinný pro každý záznam. Struktury pouze s PK jsou vhodné v situacích, kdy je určeno, že při čtení z databáze bude vždy požadován jeden parametr vyhledávání. Příkladem je, kdybychom se podívali do seznamu občanů a chtěli získat údaje o osobě na základě rodného čísla (které by bylo v databázi jako PK). Avšak třeba pro situaci, kdy bychom chtěli například data o uživatelském profilu určitého zákazníka a jeho předchozí objednávky (pro situaci, kdy provozujeme internetový obchod), struktura pouze s PK nebude dostačující. K tomu slouží složený primární klíč. Ten se skládá opět z PK a nově je přidán sekundární klíč tzv. *sort key* (označován zkratkou SK), který přidává další úroveň specifikace. Obecně je tento druh primárního klíče mnohem častěji používán, jelikož nabízí větší flexibilitu záznamů. Zároveň je v DynamoDB možno přidat libovolný počet dodatečných parametrů, filtrovat však lze pouze pomocí PK a SK. To umožňuje udržení vysokého výkonu a rychlosti čtení dat pro malé i enormní množství dat, jelikož je proces filtrování záznamů výrazně zkrácen tím, že pokud specifikujeme primární klíč, veškeré záznamy, které tento klíč neobsahují jsou

automaticky ignorovány [19]. Není tedy potřeba procházet každý záznam zvlášť, což ušetří čas a výpočetní výkon. Příklad struktury databáze využívající složený primární klíč se záznamy je uveden na obrázku 6.

Primární klíč		Atributy		
PK	SK			
uzivatel #jannovy	profil#jannovy43	Uživatelské jméno	Jméno a příjmení	E-mail
		jannovy43	Jan Nový	novyJan43@gmail.com
	objednavka#31321	Uživatelské jméno	ID objednávky	Stav objednávky
		jannovy43	31321	PŘIJATO
	objednavka#74231	Uživatelské jméno	ID objednávky	Stav objednávky
		jannovy43	74231	PŘIJATO
	objednavka#21145	Uživatelské jméno	ID objednávky	Stav objednávky
		jannovy43	21145	ODESLÁNO
uzivatel#anamala	profil#anamala91	Uživatelské jméno	Jméno a příjmení	E-mail
		anamala91	Ana Malá	malaana91@gmail.com
	objednavka#89214	Uživatelské jméno	ID objednávky	Stav objednávky
		anamala91	89214	PŘIJATO
	objednavka#99081	Uživatelské jméno	ID objednávky	Stav objednávky
		anamala91	99081	ZRUŠENO

Obrázek 6 - Příklad databáze se složeným klíčem

Z obrázku výše lze pozorovat databázi s uživateli, využívající jakýsi internetový obchod. Pomocí PK a SK lze danému uživateli přiřadit několik záznamů různého druhu – záznam o profilu a záznamy o jednotlivých objednávkách. Platí tedy, že musí vždy být dvojice PK a SK unikátní, což je díky stylu pojmenování SK vždy splněno.

Při pohledu na první řádek, lze vyčíst, že záznam má PK ve formátu *uživatel#jannovy* a SK ve formátu *profil#jannovy43*. Dále má tento záznam tři další atributy – uživatelské jméno, jméno a příjmení a e-mail. Jedná se tedy o záznam v databázi, že osoba se jménem Jan Nový má v rámci internetového obchodu vytvořený profil s uživatelským jménem *jannovy43* (a jsou v záznamu uvedené potřebné informace o profilu). Další řádky (obsahující stejný PK) pak ilustrují jednotlivé objednávky, které uživatel prostřednictvím svého profilu vytvořil (včetně příslušných atributů – ID objednávky a stav objednávky). Dotazování na takovouto strukturu je pak v případě velkého množství dat velmi efektivní, jelikož je dostačující specifikovat pouze chtěný PK a SK.

4.2 Amazon S3

Dalším nástrojem, který sice v této práci není přímo použitý, avšak s danou problematikou úzce souvisí a ve výsledné aplikaci použitý je, je Amazon S3. Název S3 je zkratkou pro *Simple Storage Service* neboli jednoduchá úložná služba. Jelikož se množství dat dynamicky mění a zároveň skladovaná data můžou, ale nemusí být vždy nutně potřebná k aktivnímu používání, je potřeba tato data určitým způsobem skladovat tak, aby bylo možno k nim jednoduše přistoupit a zároveň nebyl plýtván úložný prostor. S3 je takzvané objektové uložiště [20]. Princip spočívá v tom, že data nejsou ukládána „tradičním“ způsobem do složek a souborů, nýbrž do tzv. objektů. Tyto objekty obsahují skupinu dat podobného charakteru, společně s tzv. metadaty („data o datech“) a ukládá je do různých serverů. Tyto bloky dat jsou navzájem nezávisle uložené, což umožňuje rychlé a efektivní čtení i ukládání velkého množství dat, kdykoliv a kdekoliv. Při přístupu je nutno získat pouze určitý objekt a není potřeba procházet celá uložiště s cílem najít konkrétní skupinu dat.

4.2.1 Výhody S3

Z hlediska výhod uložiště S3 je hlavní především škálovatelnost a přístupnost. Jeden objekt v S3 může dosahovat velikosti až 5 TB a z hlediska škálovatelnosti takřka nemá omezení. Zároveň při tvorbě objektu v S3 proběhne automatické zálohování na minimálně dalších dvou fyzických lokalitách, které jsou od sebe vzdálené minimálně 10 km, což zajišťuje, aby nenastala nečekaná ztráta cenných dat [20]. Krom toho jsou data enkryptována třemi způsoby, zamezující hrozbám útoků. Ochraňují uživatelský účet a využívají umělou inteligenci k detekci anomálií při práci s daty, přičemž následně uživatele informují o možné hrozbě. V neposlední řadě je jednou z výhod také jednoduchost spravování databáze. To probíhá prostřednictvím tzv. *user-friendly* (uživatelsky přívětivého) webového rozhraní. Pomocí tohoto rozhraní lze vykonat širokou škálu operací včetně jednoduché migrace dat, transportu dat z nebo do S3 či stažení dat na jakékoliv zařízení.

4.2.2 Rozdíly mezi DynamoDB a S3

Na první pohled se může jevit, v porovnání s S3, databáze DynamoDB jako takřka totožná, jelikož mají podobné charakteristiky. Tyto dvě databáze však mají své rozdíly

a liší se především v tzv. *use-case* neboli v situacích, kdy je vhodné je používat. Základními rozdíly jsou:

Velikost uložených dat – Jak bylo zmíněno, objekt v S3 může dosahovat velikosti až 5 TB, zatímco jeden záznam v DynamoDB má maximální velikost přibližně 400 KB [19]. S3 je tudíž navržena především na množství dat, přičemž není příliš brán zřetel na možné zpoždění při dotazování. S3 je schopno zpracovat enormní množství požadavků z různých zdrojů, zatímco u DynamoDB je kladen důraz na rychlé a efektivní získávání hledaných záznamů. Při dotazování těchto dat je klíčový faktor, o jaká data se jedná. Jak je blíže vysvětleno v dalším odstavci, uložená data mají odlišnou strukturu a každá z technologií je tedy adekvátnější pro jiné situace [20].

Struktura a transport uložených dat – Objekt v S3 jako takový lze považovat za velká, nestrukturovaná data. Při získávání a transportu těchto dat, jsou tyto objekty vkládány do tzv. *buckets* neboli kýblů, které lze považovat například za tabulku s daty, přičemž každý kýbl je spojený s určitým regionem [20]. Kvůli tomu lze pak jednoduše tato data transportovat či zálohovat napříč regiony. Stačí specifikovat region, kde jsou data uložena a uvést informace jako je přístupová kontrola a možnosti spravování dat. Následně je transport dat v „kýblech“ napříč regiony jednoduchý. Z hlediska migrace dat napříč regiony nenabízí DynamoDB takto jednoduchou cestu a operace transportu dat může být poněkud složitá. Avšak uživatelé mohou do dané tabulky DynamoDB zapisovat data ze kteréhokoliv regionu [19].

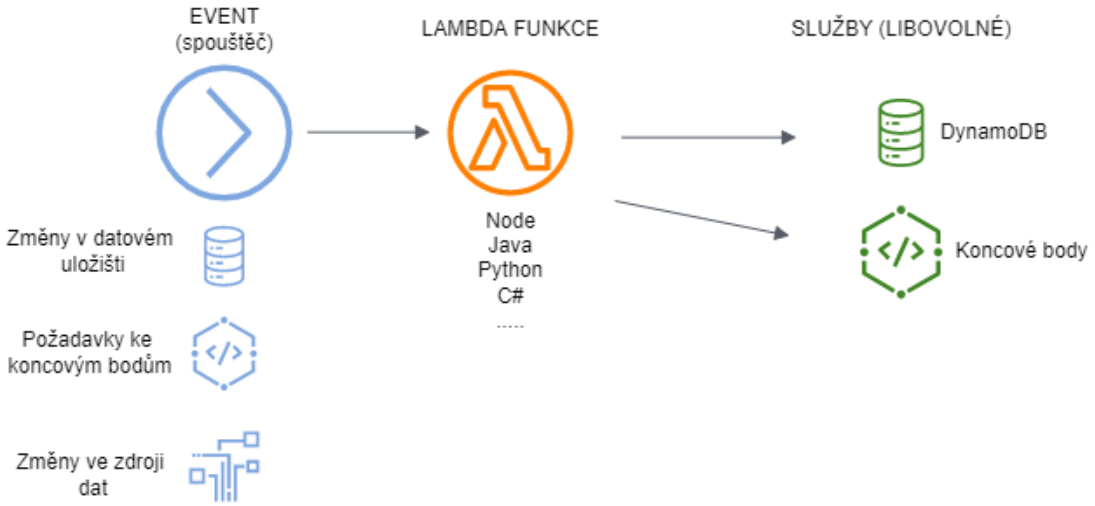
Dotazování dat – Struktura tabulek DynamoDB se záznamy je přehledně uvedena na obrázku 6, přičemž tato struktura umožňuje uživateli jednoduše a efektivně získat libovolné množství uložených dat pomocí vlastní specifikace [19]. Dotazování v S3 je sice také možné, ale S3 je především využívaná pro situace, kdy jsou tato data transportována či zálohována ve větších částech a dotazování na konkrétní data není zdaleka tak efektivní jako u DynamoDB [20]. Na obrázku 7 je uveden jednoduchý princip ukládání dat pomocí S3 a následné využití těchto dat.

Obrázek 7 - Proces ukládání dat v Amazon S3

4.3 AWS Lambda

Posledním detailněji přiblíženým nástrojem z platformy AWS je AWS Lambda. Lambda je tzv. *serverless* (bez potřeby serverů), *event-driven* (spouštěna událostí) výpočetní služba [21]. Využívá se k spuštění takřka libovolného kódu, psaného v jednom z podporovaných programovacích jazyků, bez potřeby správy serverů. Uživatelé vytvářejí funkce či aplikace, které po nahrání do prostředí AWS Lambda jsou vykonány efektivním a flexibilním způsobem. Tyto funkce mohou vykonávat libovolnou činnost – výpočetní proces, spravování webových stránek či zpracovávání toku dat, která jsou „volána“ do API (*Application Programming Interface*) a integrována do dalších AWS služeb. Příkladem funkce může být jev, kdy uživatel chce aktualizovat či přidat položku do nákupního košíku prostřednictvím webového rozhraní na libovolném internetovém obchodě. Daný proces je tedy čistě závislý na způsobu, jakými jsou funkce navrženy a také pro jakou aplikaci mají být implementované.

Princip Lambdy spočívá v tzv. *containers* čili kontejnerech. Každá funkce v Lambda funguje uvnitř svého kontejneru. Při vytvoření funkce uživatele v prostředí Lambda, je tato funkce zabalena do nového kontejneru, který je dále zpracován v tzv. *cluster* čili shluku hardware zařízení, spravovaných poskytovatelem [21]. Před spuštěním je každému kontejneru alokována potřebná paměť RAM (*Random Access Memory*) a výpočetní kapacita CPU (*Central Processing unit*). Na obrázku 8 je uveden jednoduchý princip průběhu změny v aplikaci za použití AWS Lambda, kdy pomocí nějaké události (*Event*) dojde ke spuštění funkce implementované v prostředí AWS Lambda, která následně produkuje na výstupu změnu (může se jednat o uložení/získání dat z databáze, komunikace s koncovým bodem či mnoho dalších jevů).



Obrázek 8 - Princip AWS Lambda

4.3.1 Klíčové vlastnosti AWS Lambda

AWS Lambda se vyznačuje mnoha vlastnostmi. Konkrétních výhod AWS Lambda je velké množství, níže uvedené jsou jen části široké škály nabídky.

Libovolná logika napříč aplikací – Díky funkcím implementovaným v AWS Lambda lze určitou logiku fungování aplikace aplikovat napříč platformou AWS na další služby jako jsou kýble S3 či tabulky v DynamoDB, což umožňuje libovolně manipulovat s daty po dobu procházení cloudem [21]. Toto je hlavní vlastnost, díky které je v této práci umožněno jednoduše implementovat funkce v korespondenci s databází DynamoDB, jak je přiblíženo v kapitole 6.

Tvorba podpůrných (*backend*) služeb – Pomocí Lambda lze vytvořit podpůrné (*backend*) služby aplikací, které se spouštějí při potřebě vývojáře prostřednictvím Lambda API [21]. To umožňuje tyto vytvořené procesy zpracovávat v prostředí Lambda místo vykonávání procesů na straně klienta, což vede k úspoře energie a jednodušším aktualizacím, bez obtěžování klienta.

Vlastní program – Jak již bylo zmíněno, jednou z hlavních vlastností je možnost implementovat vlastní kód, psaný v jednom z podporovaných programovacích jazyků. Mezi ně patří Java, JavaScript (Node.js), C#, Python a další [21]. Díky této možnosti není třeba učit se novou syntaxi při používání AWS Lambda a zároveň poskytuje i tzv. *Runtime* API, která umožňuje implementaci kódu v dalších jazycích.

Správa a ochrana proti výpadkům – Spravování infrastruktury je zcela zajištěno poskytovatelem služby, které zároveň od uživatele nepožaduje žádné vlastní aktualizace OS ani správu uložiště [21]. V jednoduchosti umožňuje službu používat kdykoliv a kdekoliv. Zároveň spravuje dostupnost napříč všemi regiony takovým způsobem, že se takřka nemůže stát, aby z důvodu výpadku hardware či centra na straně provozovatele došlo k výpadku služby.

Automatické škálování a kontrola výkonu – Potřebné množství požadavků či prostředků při zprovoznění funkcí v Lambdě je automaticky škálováno tak, aby bylo možno zpracovat neomezený počet požadavků [21]. Existuje-li funkce, spouštěná např. při interakci uživatele s aplikací, bude potřebný výkon na konstantně vysoké úrovni bez ohledu na počet interakcí, a to právě díky automatickému škálování. Zároveň existuje možnost výkonnost (a tudíž i rychlost reakce na událost) modifikovat dle potřeb, například v situacích, kdy je nutno, aby odpověď byla rychlejší či pomalejší, než je automaticky stanovená hodnota.

Jako příklad obecné implementace v prostředí mobilních sítí a aplikací je na obrázku 9 uveden princip procesů podpůrných funkcí s použitím AWS Lambda. Jedná se o generování notifikace při používání např. sociální sítě, či jiné aplikace. Krom funkčního bloku Lambda se zde vyskytuje i tzv. API *Gateway* čili brána mezi uživatelem a aplikací a Amazon SNS (*Simple Notification Service*), která slouží k distribuci zpráv a notifikací v relaci aplikace-aplikace či aplikace-osoba [21].

Obrázek 9 - Využití AWS Lambda v prostředí mobilní aplikace

5 Použitá data

Algoritmy a řešení uvedena v této práci jsou součástí většího, probíhajícího projektu v rámci univerzity ČVUT. Na tomto projektu se podílí i Technická univerzita v Ostravě a Vysoká škola ekonomická v Praze. Veškeré algoritmy v této práci tedy manipulují s daty, poskytnutými ze strany projektu, přičemž výstupy jsou tatáž data, akorát ve transformované podobě, sloužící pro další vývoj. Konkrétní procesy a výstupy algoritmů jsou uvedeny v dalších kapitolách.

Využitá data se nacházejí v databázi DynamoDB o pokrytí mobilní sítí v České republice na veškerých standardizovaných frekvencích a technologiích. Databáze obsahuje záznamy, které jsou popsány několika parametry. V případě použité DynamoDB databáze je území ČR pomocí záznamů rozděleno do čtverců či pixelů o rozměrech 10000 metrů čtverečních (pixely 100x100m). Příklad záznamu je uveden v tabulce 3.

Tabulka 3 - Příklad záznamu v DynamoDB

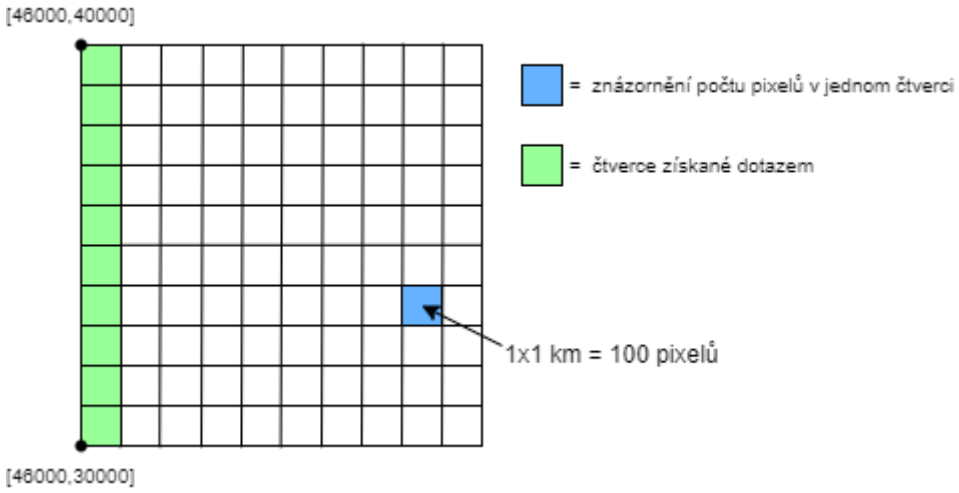
PK	SK	csigs	pix	cnids
cels#202108#460300	4600030019	71772529	100mE46000N30019	KLDacj53AC

Záznam jako takový reprezentuje pixel/čtverec s údaji o pokrytí mobilní sítí. PK je v případě těchto záznamů dán ve formátu *cels#202108#460300,* kde *cels* vyznačuje fakt, že se jedná o buňku mobilní sítě, *202108* je časový údaj, znamenající, že záznam je z 8. měsíce roku 2021 a číslo *460300* udává, že záznam o buňce se nachází v území o rozměru 10x10km, který má souřadnice 46000 na ose X a 30000 na ose Y. SK slouží poté ke konkrétnější specifikaci zkoumaného území. Jeho struktura 4600030019 reprezentuje, že prvních 5 číslic jsou souřadnice na ose X a druhých 5 na ose Y. Při dotazování na buňky nacházející se v uvedeném území s PK *cels#202108#460300* lze dotaz položit například ve formátu:

PK = cels#202108#460300 a *SK = mezi (4600030000 a 4600040000)*

Tímto způsobem se specifikovalo, že rozsah souřadnice Y má být 30000–40000 a souřadnice X má být konstantně 46000, čímž získáme zpět veškeré záznamy s touto

kombinací PK a SK. Konkrétní uvedený příklad by vrátil 1000 pixelů ze zvoleného území 10x10km (tedy 1000 pixelů z možných 10 000). Na obrázku 10 je uvedena vizuální reprezentace takového dotazu.



Obrázek 10 - Vizualizace dotazu na databázi

Celkový velký čtverec tedy reprezentuje území o rozměru 10x10 km. Z obrázku je zřejmé, že jeden čtvereček pak odpovídá území 1x1 km a obsahuje tedy 100 pixelů (s rozměry 100x100m). Při výše uvedeném dotazu bychom se dotazovali na sloupec označený zelenou barvou, tedy na 10 čtverečků obsahujících 100 pixelů, což v součtu činí 1000 pixelů.

Krom zmíněných PK a SK obsahuje záznam z databáze uvedený v tabulce 3 i parametr *csigs*, což je identifikátor konkrétní buňky pokrytí mobilní sítí, parametr *pix*, který je reprezentací daného pixelu, a proto je ve formátu *100mE46000N30019*, který vyznačuje, že daný pixel má počáteční souřadnici (tedy dolní levý roh čtverečku) na uvedených souřadnicích [46000,30019] a velikost pixelu je 100x100m, což je reprezentováno údajem *100m* na začátku parametru. Posledním uvedeným parametrem je parametr *cnids*, což je unikátní identifikátor, nesoucí v sobě informace o frekvenci, technologii a lokalizaci dané buňky. Konkrétně uvedený příklad

KLDacj53AC

znamená, že se jedná o 4G LTE makro buňku provozovanou na frekvenci 2100 MHz s vysílačem umístěným v Kladně.

V databázi se nacházejí i další parametry, jako například veškerá specifická území klasifikovaná pomocí Nomenklatury územních statistických jednotek (NUTS) a jejich vazba na danou buňku, avšak pro tuto práci nejsou důležité.

6 Motivace práce

Jak bylo zmíněno již několikrát, obsahem této práce jsou algoritmy pro zpracování dat o mobilní síti. Tyto algoritmy jsou v rámci projektu zmíněného v úvodu kapitoly 5 aktivně používány ve vývoji webové aplikace, umožňující interakci uživatele s informacemi o pokrytí mobilní sítí v ČR. Principiálně algoritmy umožňují uživateli zjistit úroveň pokrytí a technologie (frekvence, generace sítě) nad libovolným územím a získat zpět seznam buněk mobilní sítě pokrývající toto území. V praxi proces probíhá tak, že uživatel otevře aplikaci ve webovém prohlížeči a zobrazí se mu mapa ČR. Dále pak pomocí tlačítka nakreslí libovolný polygon na místě, kde ho zajímají informace o pokrytí mobilní sítí. Demonstrace vytvořených algoritmů a celková funkcionalita aplikace je uvedena v kapitole 8, společně s vysvětlením přínosu zmíněných algoritmů a dalším plánem vývoje.

Algoritmus A v této práci má za účel transformovat tyto uživatelsky zvolené polygony do podoby, která je adekvátní z hlediska zpracování dat – tedy do jisté míry i formátovat data tak, aby s nimi bylo možno pracovat dále. Algoritmus B má poté za účel transformovaná data dát do souvislosti s databází o pokrytí mobilní sítí, přiblížené v kapitole 5, a získat tím informace o pokrytí zvoleného území. Oba algoritmy jsou psány v programovacím jazyce JavaScript za využití veřejně dostupných knihoven, které umožňují lépe pracovat s daty a rozšiřují možnosti programování. Výsledné kódy těchto algoritmů jsou pak implementovány v prostředí AWS Lambda, pomocí kterého je zajištěna jednoduchá a efektivní funkcionalita v rámci platformy AWS, a to především na relaci AWS Lambda a DynamoDB. To umožňuje rychlé a efektivní získávání dat z databáze DynamoDB a vykonání daných funkcí.

7 Navržené algoritmy

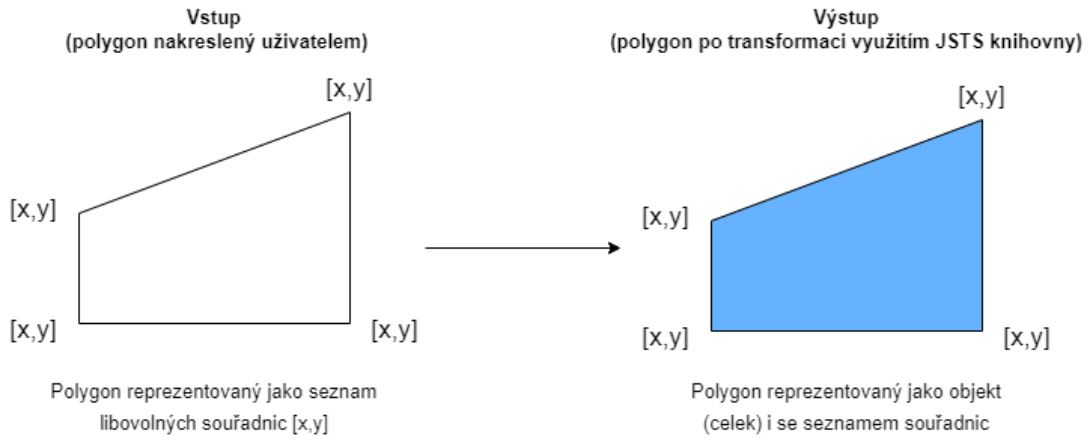
V této kapitole jsou podrobně popsány oba navržené algoritmy. Z hlediska funkcionality je lze označit za transformační algoritmus (algoritmus A) a algoritmus sloužící k propojení s databází DynamoDB (algoritmus B). Algoritmus A upravuje data do vhodné podoby pro další zpracování, zatímco algoritmus B vytváří adekvátní strukturu dat pro dotazování databáze.

7.1 Algoritmus A

7.1.1 Použité knihovny a jejich účel

První vytvořený algoritmus jde z části nazvat transformačním, jelikož jeho principiální podstata je upravit data přijatá ze strany uživatele (osoba využívající webovou aplikaci) a připravit je do takové podoby, aby bylo možno poté je použít v souladu s funkcionalitou databáze DynamoDB o pokrytí mobilní sítí (uvedenou v kapitole 5).

Jak je uvedeno v kapitole 6, vstupními daty pro tento algoritmus je polygon nakreslený uživatelem na mapě. Polygon do systému přichází jako soubor souřadnic, udávající vrcholy tohoto polygonu. Z toho důvodu je potřeba nejprve transformovat takovýto seznam souřadnic do tzv. *JavaScript Topology Suite* (JSTS) struktury. JSTS je název použité knihovny programovacího jazyka JavaScriptu, která obsahuje funkce pro zpracování a úpravu prostorových a geometrických prvků. Zároveň přidává funkcionalitu, díky které jsou funkce knihovny JSTS kompatibilní s funkcemi v knihovně *Open Layers* (OL), což je knihovna, sloužící pro zobrazování map ve webovém prohlížeči. Jinými slovy se jedná o knihovnu, pomocí které je možno v rámci aplikace mít na pozadí zobrazenou mapu a interaktivně využívat souřadnicového systému k dalším úkolům. Rozdíl mezi formátem polygonu na vstupu a potřebným formátem (který je získaný použitím funkcí v JSTS knihovně) je uveden na obrázku 11.



Obrázek 11 - Transformace polygonu pomocí JSTS knihovny

Nad takto transformovaným polygonem lze poté provádět prostorové operace typu: průnik, je/není součástí, rozdíl, součet atd.

Poslední použitou knihovnou je knihovna s názvem *Proj4*, což je zkratka pro slovo projekce. Tato knihovna slouží k transformaci souřadnicových systémů mezi sebou. Tyto transformace mohou být jak kartografické, tak i geodetické. Konkrétní využití této knihovny je uvedeno v následující kapitole.

7.1.2 Transformace souřadnicového systému

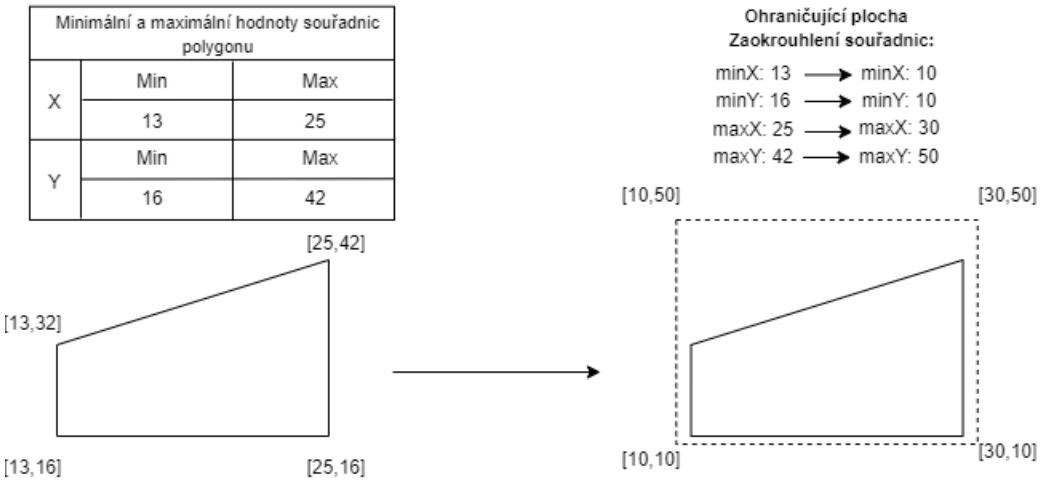
Je známým faktem, že existuje velké množství souřadnicových systémů a všechny mapy nemusejí být nutně používány ve stejném souřadnicovém systému. Proto je nutno sjednotit souřadnicový systém mapy, která je podkladem aplikace a veškerých objektů, které jsou na mapě dodatečně přidány či zobrazovány. Je to z toho důvodu, aby byla umožněna interakce mapy se všemi možnými objekty.

Existuje veřejně dostupná databáze souřadnicových systémů, geodetických údajů a dalších prostorových prvků, nesoucí název *European Petroleum Survey Group* (EPSG). Každý souřadnicový systém v této databázi je určitým způsobem definovaný pomocí zeměpisné délky a šířky, měrné jednotky, středního bodu a dalších parametrů. Z databáze poté lze získat „kódovanou“ podobu souřadnicového systému. V této podobě, za použití knihovny Proj4, lze pak jednoduchou funkcí vytvořit transformaci mezi souřadnicovými systémy. Díky této funkcionalitě lze v tomto algoritmu jednoduše sjednotit používaný souřadnicový systém.

Nyní je tedy přijatý polygon transformován do objektu JSTS, jak je uvedeno na obrázku 11 a veškeré souřadnice jsou převedeny do jednotné (libovolné) formy.

7.1.3 Tvorba ohraničující plochy

Při příjmu již transformovaného polygonu v adekvátním souřadnicovém systému, je nutno v dalším kroku stanovit území, které ohraničuje daný polygon kvůli operacím, které budou nad polygonem prováděny. Jak je zmíněno v kapitole 5, databáze, obsahující data o pokrytí mobilní sítí je strukturovaná na bázi pixelů, tedy čtverečků o určitém rozměru, reprezentující pokrytí sítí. Jelikož je nutno, aby polygon korespondoval s daty v databázi, je potřeba transformovat objekt polygonu do sbírky pixelů. Toho je dosaženo tak, že se nejprve z celkového seznamu souřadnic polygonu získají maximální a minimální hodnoty souřadnic na obou souřadnicových osách X a Y. Tyto hodnoty jsou pak zaokrouhleny (pro maxima nahoru, pro minima dolů) tak, aby vznikl čtverec/obdélník, který v sobě obsahuje celý polygon. Z toho důvodu je minimální hodnota zaokrouhlována dolů a maximální nahoru, čímž je dosaženo jakési „rezervy“, tj nikdy nenastane situace, že by část polygonu nebyla součástí ohraničující plochy. Na obrázku 12 je tento fakt zvýrazněn. Hodnoty souřadnic v obrázku jsou čistě názorné a neodpovídají realitě, jedná se o vysvětlení principu. V reálném světě jsou hodnoty souřadnic řádově vyšší čísla.



Obrázek 12 - Ohraničující plocha polygonu

Probíhá tedy zaokrouhlování nahoru/dolů na jednu platnou číslici. Tím je zajištěno, že polygon je vždy součástí ohraničující plochy. Nabízí se otázka, zda je tím zabrána zbytečně velká plocha, jelikož ohraničující plocha je vždy o určitý kus větší než zkoumaný polygon. Způsob takovéto tvorby ohraničující plochy má svůj důvod, který je vysvětlen v další sekci. Je jím transformace vytvořené ohraničující plochy do pixelizované podoby.

7.1.4 Pixelizace ohraničující plochy

V tomto stádiu je přijatý polygon ve formátu struktury JSTS a existují data o jeho ohraničující ploše, přičemž stěžejními hodnotami jsou minimální a maximální hodnoty na osách X a Y ohraničující plochy. Jak bylo zmíněno v kapitole 5, databáze s pixely obsahuje záznamy, kde každý pixel má rozměr 100x100m. Z toho důvodu je zaokrouhlování, vysvětlené v kapitole 7.1.3, provedeno tak, že se zaokrouhlí na nejbližší hodnotu dělitelnou 100, způsobem, který je uveden v rovnici 1:

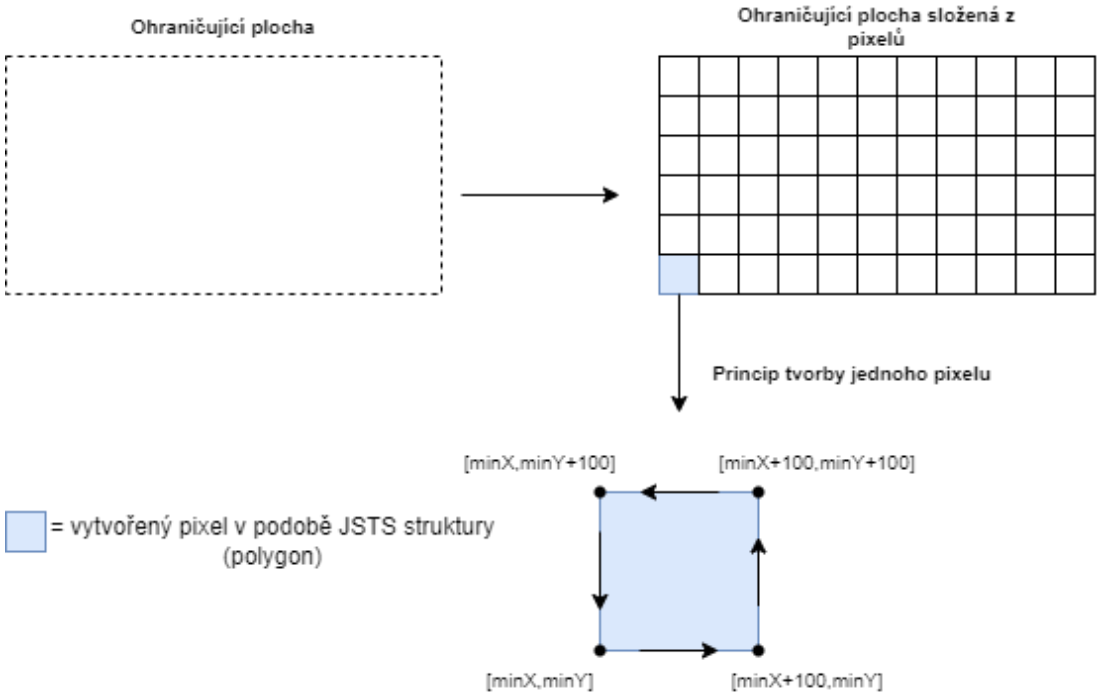
$$noveMinX = zaokrouhlitDolu\left(\frac{minX}{100}\right) * 100 \quad (1)$$

kde *noveMinX* představuje nově spočítanou minimální hodnotu na ose X, *zaokrouhlitDolu* je zjednodušené označení funkce JavaScript, která zaokrouhlí danou hodnotu na nejbližší celé číslo, které je menší, než číslo na vstupu (v JavaScriptu funkce *floor*) a *minX* je minimální hodnota na ose X.

To znamená, že pokud by například byla minimální hodnota na ose X rovna 325, provedlo by se nejprve vydělení čísla na hodnotu 3.25, dále zaokrouhlení čísla 3.25 dolů (na nejbližší celé číslo, tj. na 3) a poté by se výsledek vynásobil číslem 100. Tudíž výsledkem by byla hodnota 300, což odpovídá požadovaným parametrům. Takřka analogický proces je prováděný i pro zaokrouhlování nahoru. Nyní je potřeba získanou ohraničující plochu transformovat do celku složeného z pixelů o rozměrech 100x100m. Nejprve je potřeba zjistit, kolik daná plocha obsahuje pixelů, čehož je dosaženo založením proměnné, jejíž hodnota je rovna rozdílu maximální a minimální hodnotě na dané ose, vydělené číslem 100. Tato operace je provedena pro obě osy, čímž je získána informace, kolik pixelů bude obsahovat jeden sloupec a řádek ohraničující plochy.

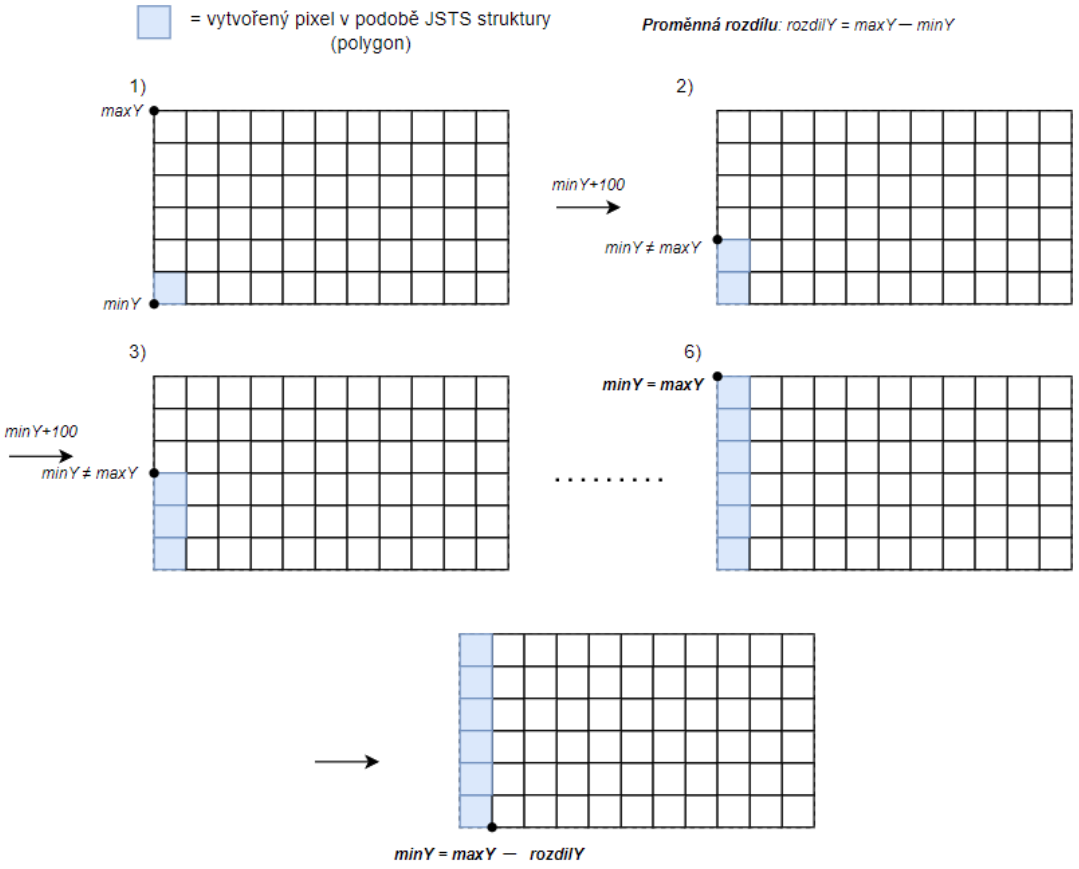
Každý tvořený pixel se skládá z X a Y souřadnic, přičemž celkový počet párů souřadnic jednoho pixelu je 5. Pátá dvojice souřadnic je shodná s první, ale musí být v programu

vytvořena, aby došlo k „uzavření“ čtverečku, jelikož každý pixel je pomocí funkcí v rámci knihovny JSTS poté transformován do polygonu (neboli JSTS struktury kvůli prostorovým operacím). Proces tvorby prvního pixelu ohraničující plochy je ilustrován na obrázku 13.



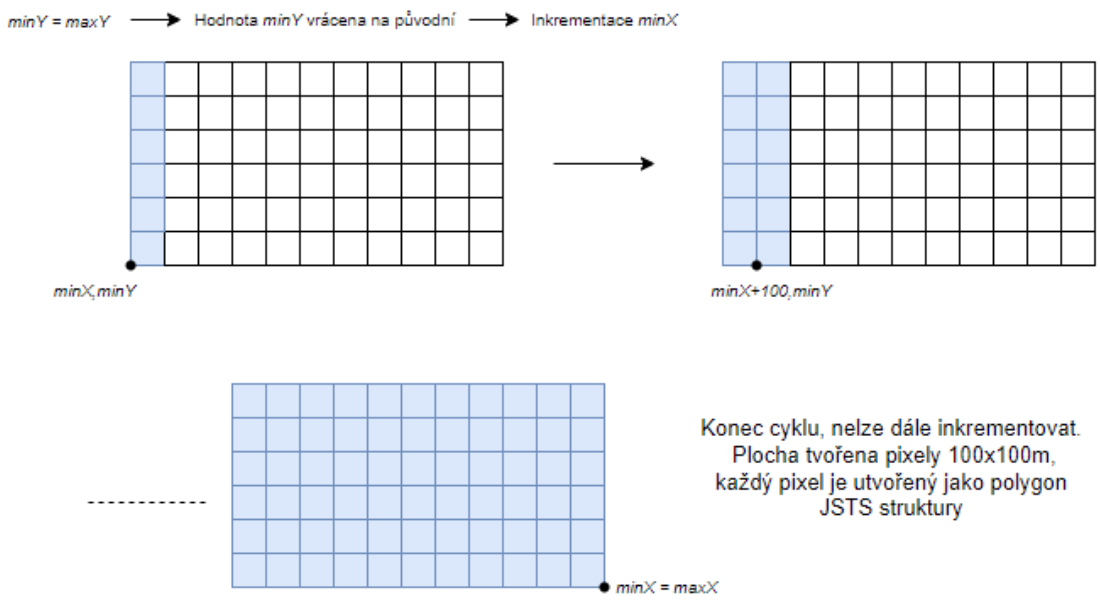
Obrázek 13 - Princip tvorby pixelu

Pixel vyobrazený na obrázku 13 má tedy první a pátou souřadnici rovnou *[minX,minY]*. Při dotvoření pixelu je inkrementována hodnota *minY* o 100. Tím pádem dojde ke tvorbě pixelu s hodnotami na ose X shodnými a na ose Y vyššími o 100. Jinými slovy tento cyklus provádí tvorbu pixelů ve sloupci. Při každém inkrementování hodnoty *minY* je kontrolováno, zda je tato hodnota rovna s hodnotou *maxY*, získanou z rozměrů ohraničující plochy. Ve chvíli, kdy jsou hodnoty *minY* (inkrementované v každém kroku cyklu) *a maxY* rovny, dojde k operaci, která vrátí hodnotu *minY* na původní hodnotu (tj na hodnotu, kterou měla daná proměnná na začátku cyklu). Toho je dosaženo jednoduše odečtením hodnoty rozdílu maximální a minimální hodnoty na ose Y (která byla určena před tvorbou prvního pixelu) od nynější hodnoty *minY*. Princip je ilustrován na obrázku 14.



Obrázek 14 - Princip tvorby sloupce z pixelů

Po takto vytvořeném sloupci pixelů je tedy vrácena původní hodnota *minY* a naopak je o 100 inkrementována hodnota *minX*. Tedy dochází k posunutí na ose X a provádí se tvorba pixelů v druhém sloupci. Takřka analogický proces tedy probíhá i pro osu X – vytvoří se sloupec, způsobem znázorněným na obrázku 14 a posune se hodnota minima osy X o 100. I v tomto případě je při každém kroku kontrolováno, zda je rovna hodnota *minX* (která je inkrementována) hodnotě *maxX*, získanou z parametrů ohraničující plochy. Ve chvíli, kdy je dosaženo situace, že *minX* = *maxX* a nelze tedy inkrementovat dále, cyklus je u konce a celá ohraničující plocha je sestavena z pixelů.

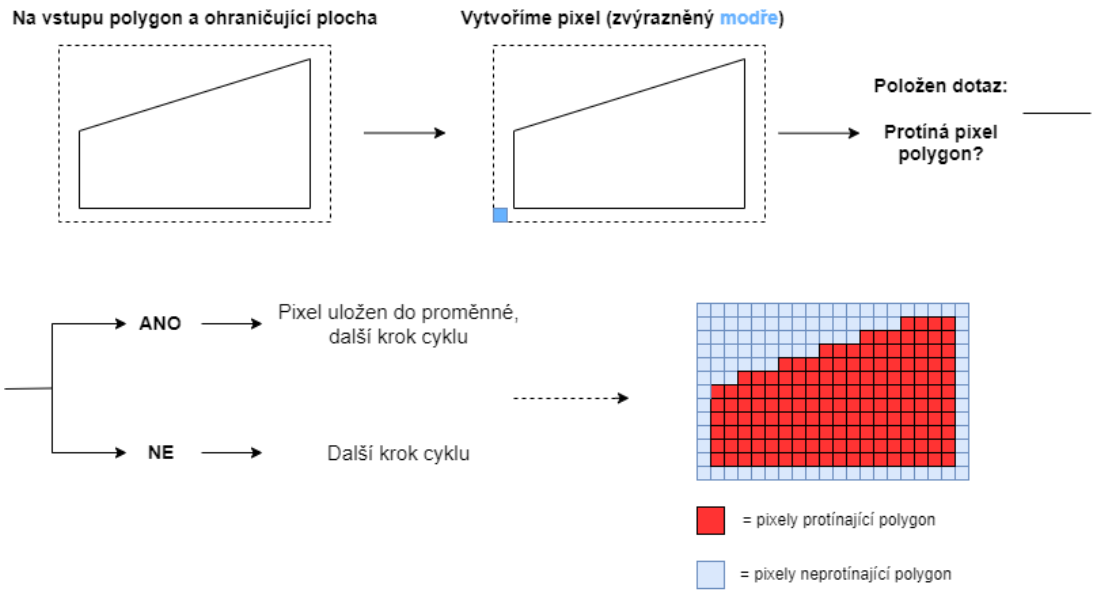


Obrázek 15 - Inkrementace na ose X

Na obrázku 15 je pouze vizualizován princip inkrementace na ose X. Jak již bylo vysvětleno, po každé inkrementaci osy X, proběhne proces tvorby sloupce pixelů (inkrementací osy Y). Po vyčerpání se hodnoty osy Y vrátí na prvotní stav a dojde k posunu na ose X. Proces je u konce ve chvíli, kdy se konstantně zvětšující hodnota *minX* rovná maximální hodnotě *maxX*.

7.1.5 Zisk relevantních pixelů

Jak bylo zmíněno v sekci 7.1.1, použitím knihovny JSTS dochází k transformaci polygonu nakresleného uživatelem do struktury JSTS proto, aby nad tímto polygonem bylo možno provádět prostorové operace. Ze stejného důvodu je i při tvorbě jednotlivých pixelů v kapitole 7.1.4 každý jednotlivý pixel rovněž transformován do polygonu JSTS struktury. Tento proces je dělaný proto, aby bylo možno zkoumat, do kterých pixelů ohraničující plochy zasahuje uživatelem nakreslený polygon. Tato informace je potřebná z hlediska dalšího zpracování dat a zisku dat o pokrytí. Princip procesu získávání potřebných pixelů, které protínají uživatelem nakreslený polygon je uveden na obrázku 16.



Obrázek 16 - Proces získávání relevatních pixelů

Díky knihovně JSTS existují dostupné funkce s názvem *intersects* neboli v překladu „protíná“, které umožňují analyzovat, zda se objekty dotýkají navzájem. Zisk pixelů, které se s uživatelsky nakresleným polygonem protínají, probíhá tak, že ve chvíli, kdy je tvořen pixel ohraničující plochy (kapitola 7.1.4), položí se dotaz, zda se nově vytvořený pixel protíná s nakresleným polygonem. Pokud ano, je tento vytvořený pixel (tzn. objekt JSTS) uložený do oddělené proměnné, která obsahuje ve finále pouze relevantní pixely. Jinými slovy tím získáme seznam všech pixelů, pomocí kterých lze nakreslený polygon reprezentovat. Ostatní pixely jsou více méně zahozeny, jelikož pro další zpracování nemají smysl.

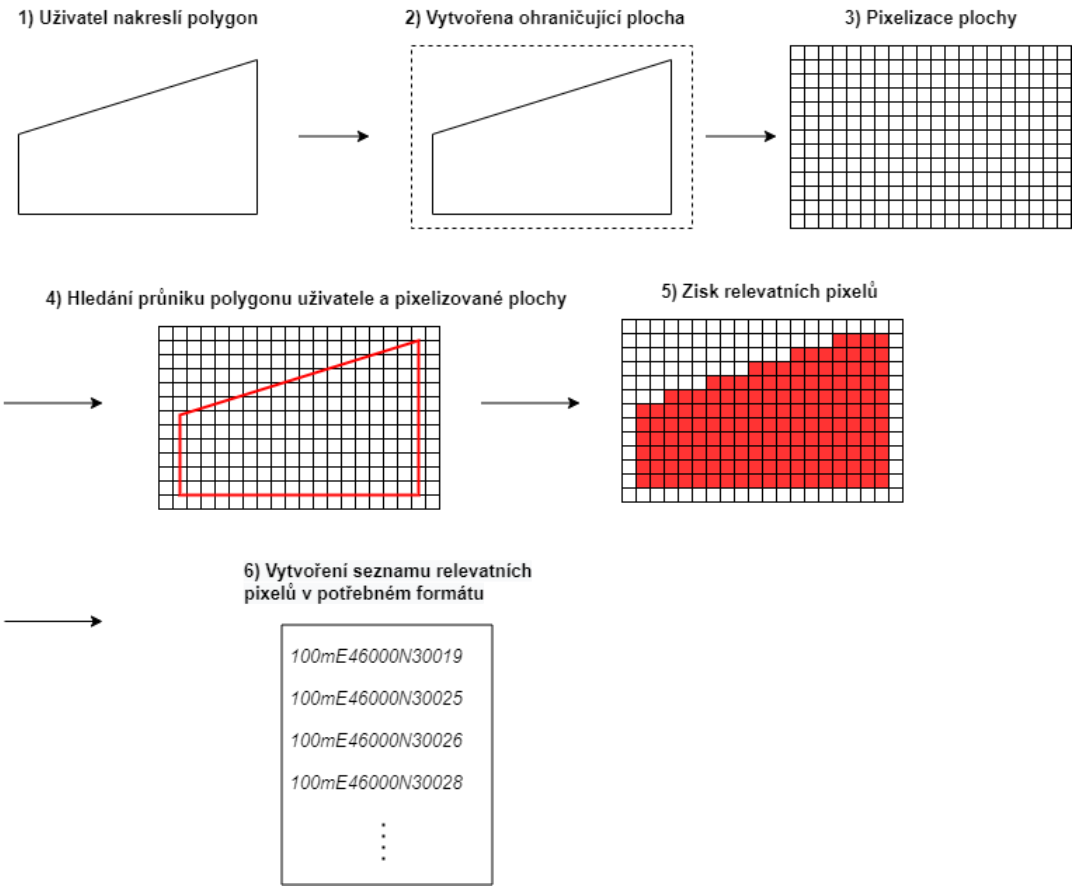
Na konci cyklu je tedy proměnná, nesoucí v sobě data o všech pixelech, které protínají zkoumaný polygon. Tato data jsou cílem toho algoritmu, jelikož slouží pro další zpracování za využití databáze DynamoDB o pokrytí mobilní sítí. Posledním krokem algoritmu je připravit výstup kódu do takové podoby, na kterou lze v dalších krocích navázat. Tím je formátování proměnné s relevantními pixely tak, aby na výstupu byl určitý seznam všech potřebných pixelů. Jak je uvedeno v kapitole 5 a tabulce 3, záznamy v databázi mají jako jeden z parametrů uvedený parametr *pix*. Přesně do toho formátu je potřeba výstupní proměnnou formátovat, aby bylo možno dohledat potřebná data o pokrytí konkrétního území.

V databázi je parametr *pix* dán ve formátu *100mEsouřXNsouřY*, kde *souřX* a *souřY* jsou souřadnicemi bodu pixelu, který má nejmenší hodnoty (tj. dolní levý roh čtverce).

Písmena *E* a *N* společně s předložkou *100m* představují geografické rozložení pixelu, tedy světové strany východ (*E)* a sever (*N*), zatímco *100m* reprezentuje velikost strany pixelu. Jednoduchou funkcí je vytvořen na základě proměnné s potřebnými pixely seznam, který obsahuje veškeré potřebné pixely ve formátu parametru *pix* v databázi. Zároveň jsou s tímto seznamem na výstupu i uvedeny hodnoty maxim a minim ohraničující plochy pro každou souřadnicovou osu.

7.2 Shrnutí a výstup algoritmu A

První algoritmus v této práci slouží k transformaci dat, získaných od uživatele, používající webové rozhraní mapové aplikace. Jeho cílem je transformovat vstupní data takovým způsobem, aby nad nimi bylo možno provádět prostorovou analýzu a slouží k získání relevantních dat, která mají implementaci v dalším chodu aplikace. Celkový zjednodušený proces zpracování je uveden na obrázku 17. Další postupy s daty jsou vysvětleny v algoritmu B.



Obrázek 17 - Shrnutí algoritmu A

7.3 Algoritmus B

Po zpracování geometrických a prostorových dat získaných na vstupu od strany uživatele, je potřeba tato data dále modifikovat v korelaci s již zmíněnou strukturou databáze DynamoDB o pokrytí mobilní sítí. I tento druhý algoritmus má za úkol transformovat vstupní data do vhodné podoby, avšak transformace se tentokrát týká především potřebné struktury klíčů, použité v navazujících dotazech na databázi DynamoDB.

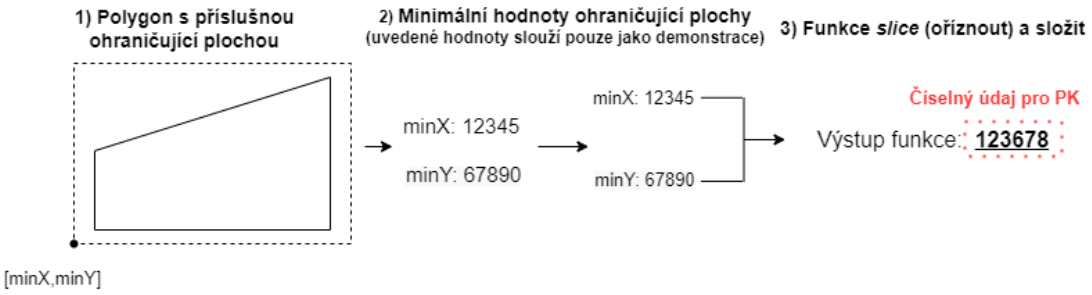
Jak bylo zmíněno na konci sekce 7.1.5, výstupní data algoritmu A jsou ve formě seznamu, obsahujícího údaje o všech pixelech, které protínají uživatelem nakreslený polygon. Nyní je potřeba získat tyto stejné pixely z databáze DynamoDB, jelikož záznamy o těchto pixelech nesou v sobě i informace o pokrytí mobilní sítí, což je cílem celého procesu.

7.3.1 Tvorba PK

Dalším krokem je dotazování se na databázi DynamoDB, přičemž je potřeba získat relevatní záznamy. Struktura databáze je uvedena v kapitole 5, ale z uvedených parametrů záznamů v databázi jsou v tuto chvíli stěžejní klíče PK a SK. PK je ve formátu *cels#202108#xxxYYY*, kde klíčovou částí je údaj *xxxYYY*. Jedná se celkem o šest číslic, kde první tři představují první tři čísla souřadnice na ose X a zbylé tři představují první tři číslice souřadnice na ose Y. Jak je uvedeno na obrázku 10, pokud bychom položili dotaz na databázi čistě jen s PK, získali bychom oblast 10x10km obsahující 10 tisíc pixelů. Takový dotaz je nereálný, jelikož by vrácená data byla příliš velká a zároveň bychom ani nezískali všechny pixely, protože by byl pravděpodobně přesažen limit velikosti vrácených dat. DynamoDB má omezenou velikost odpovědi a v tomto případě bychom obdrželi cca 5000 pixelů z dotazovaných 10000, což však obecně není problém, jelikož dotaz, požadující takové množství pixelů je nereálný (vzhledem k cíli vyvíjené aplikace). Z toho důvodu je nutno v dotazu specifikovat i SK, čímž se zúží zkoumaná oblast. Ten je ve formátu *xxxxxYYYYY,* tedy prvních 5 číslic činí souřadnici na ose X a druhých 5 souřadnici na ose Y.

Prvním krokem je vytvořit funkci pro tvorbu PK. Z výstupu algoritmu A jsou dostupné minimální a maximální hodnoty pro obě souřadnicové osy ohraničující plochy

polygonu. Jelikož se jedná o vícemístná čísla a PK je složený z dvou trojic čísel, je potřeba „oříznout“ hodnoty na první tři číslice a vytvořit tak nové číslo, reprezentující hodnotu PK. K tomu ale stačí pouze upravovat minimální hodnoty, čímž získáme celou zkoumanou oblast – v principu se jedná o dolní levý roh čtverce 10x10km (jako tomu bylo i při tvorbě jednotlivých pixelů v kapitole 7.1.4). Princip je zobrazen na obrázku 18.

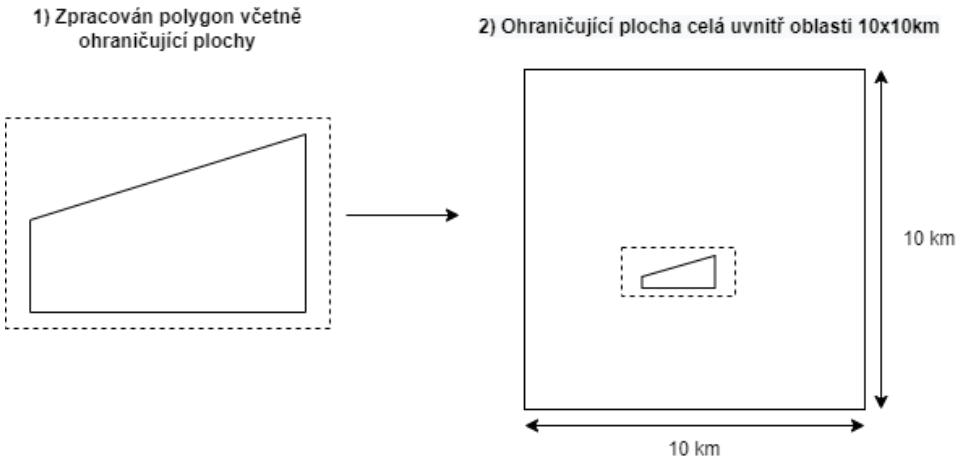


Obrázek 18 - Funkce tvorby PK

Následně pak jednoduchým postupem lze vytvořit celkový textový řetězec, korespondující se strukturou PK v DynamoDB a obsahující aktuální hodnotu PK.

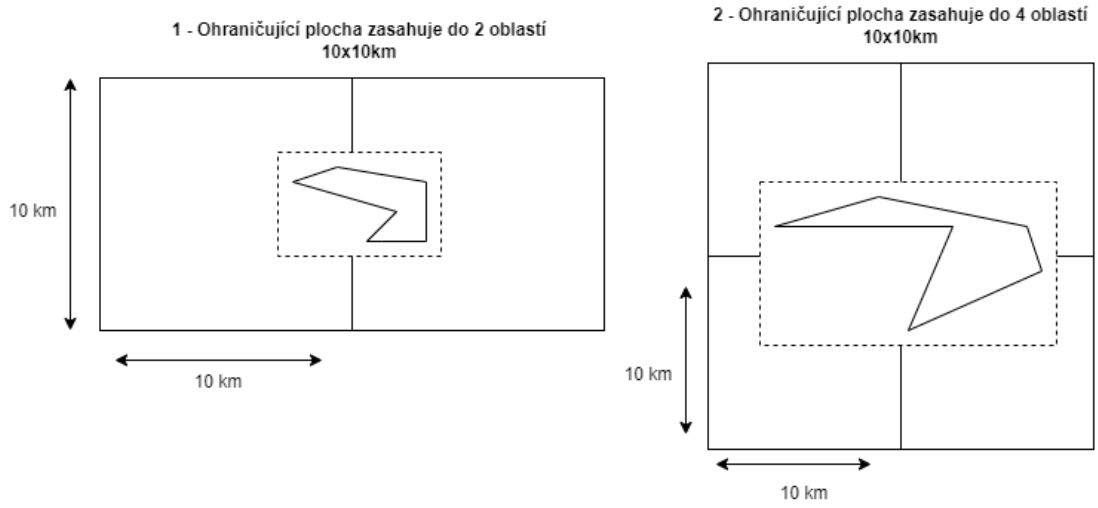
7.3.2 Výběr relevantních PK

Struktura PK je již známa, avšak existuje několik situací, které komplikují výběr, který PK bude součástí dotazu na databázi DynamoDB. Příklad uvedený výše v kapitole 7.3.1 si lze představit tak, že pomocí PK víme, že celá ohraničující plocha zkoumaného polygonu leží uvnitř čtverce daného příslušným PK. Tato situace je zobrazena na obrázku 19.



Obrázek 19 - Polygon a plocha uvnitř území 10x10km

V takovéto situaci nenastává žádná komplikace a dotaz může být se získaným PK bez problémů pokládán. Avšak může nastat situace, kdy nakreslený polygon zasahuje do 2 nebo 4 různých oblastí 10x10km, jak je ilustrováno na obrázku 20.

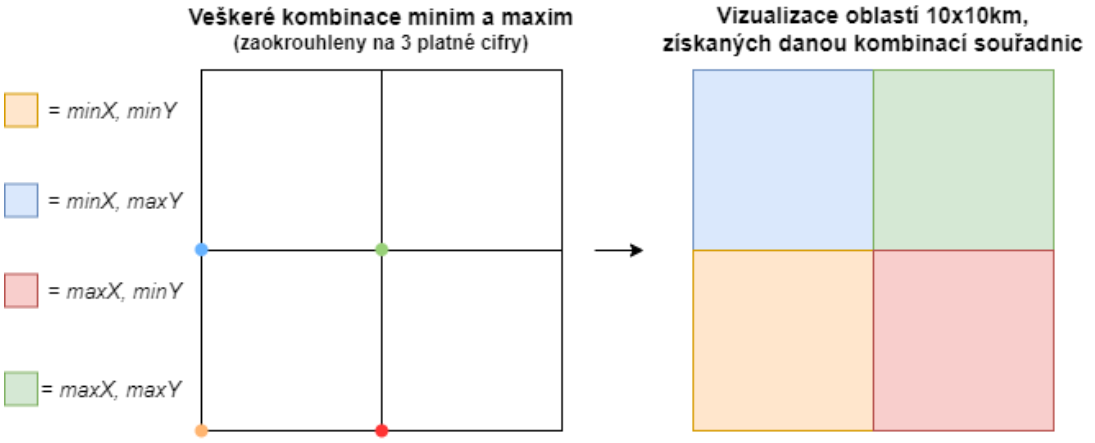


Obrázek 20 - Možnosti zasahování polygonu do více oblastí

V této situaci pak nestačí položit pouze jeden dotaz na databázi a zároveň je nutno dynamicky upravit PK tak, aby zahrnoval všechny oblasti, do kterých polygon společně s ohraničující plochou zasahuje. Z toho důvodu je potřeba vytvořit funkci, která na základě dostupných parametrů stanoví, jaká z uvedených tří situací je pro daný polygon relevantní. Toho je docíleno tak, že jsou nejprve vytvořeny všechny možné PK popisující čtveřici oblastí 10x10km. Vytvoří se tedy vzájemné kombinace minimálních

a maximálních hodnot na osách X a Y (získaných z algoritmu A), čímž získáme 4 různé kombinace PK, kde každý popisuje jednu ze čtyř oblastí 10x10km.

Jak bylo uvedeno v kapitole 7.3.1, PK je vytvořený pomocí funkce, která vytváří novou číselnou hodnotu z prvních tří číslic souřadnice na osách X a Y pomocí hodnot minim a maxim na obou osách (obrázek 18). Do funkce výběru potřebných PK tedy vstupují kombinace, které jsou zvýrazněné na obrázku 21.



Obrázek 21 - Oblasti získané kombinací souřadnic (různé PK)

Jelikož jsou souřadnice maxim a minim dány prvními třemi čísly skutečné hodnoty, nacházejí se všechny 4 body prakticky na hranách jedné oblasti (jednoho čtverce 10x10km). Avšak ne vždy je potřeba se dotazovat na všechny oblasti. Pomocí logických operátorů je vytvořena funkce, která rozhoduje, zda je potřeba dotazovat se na databázi jednou s jedním PK, dvakrát s dvěmi různými PK či čtyřikrát se všemi možnými PK. Principiálně je proces rozhodování následující:

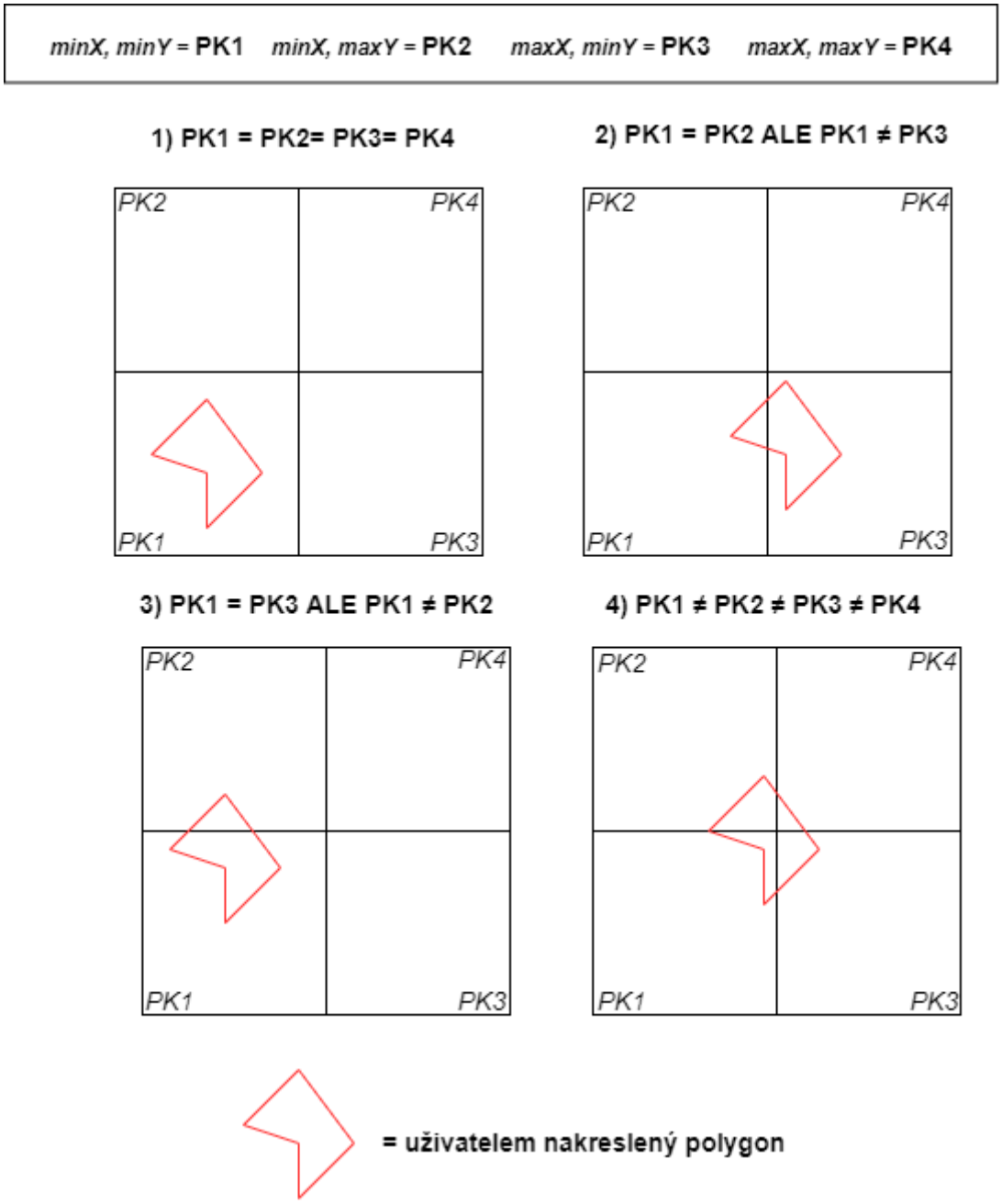
- pokud je PK získaný kombinací *minX, minY* shodný se všemi ostatními kombinacemi, polygon a ohraničující plocha leží celá v jedné oblasti 10x10km a dotaz stačí položit jedenkrát s PK tvořeným právě z *minX, minY*.

- Pokud je PK získaný kombinací *minX, minY* shodný s PK daným kombinací *minX, maxY*, ale odlišný od PK daném *maxX,minY*, polygon leží ve „spodních“ dvou oblastech 10x10km a je nutno dotaz položit dvakrát s dvěmi různými PK

- Pokud je PK získaný kombinací *minX, minY* shodný s PK daným kombinací *maxX, minY*, ale odlišný od PK daném *minX,maxY*, polygon leží ve dvou oblastech 10x10km vlevo a je nutno dotaz položit dvakrát s dvěmi různými PK

- Pokud ani jedna z uvedených možností není pravdivá, polygon leží na pomezí všech čtyř oblastí a dotaz je nutno položit čtyřikrát se všemi PK

Uvedené možnosti jsou znázorněny na obrázku 22, kde jsou uvedeny jak logické operátory pro všechny možné PK, tak i vizualizace, kde v dané situaci leží uživatelem nakreslený polygon. V rozích větších čtverců je uveden PK, pomocí kterého je daná oblast získána.

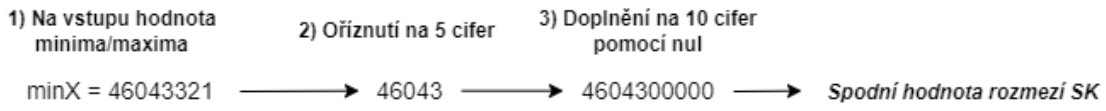


Obrázek 22 - Funkce výběru potřebných PK

7.3.3 Určení hodnoty SK

V tuto chvíli jsou v algoritmu získány potřebné PK, avšak dotaz na databázi DynamoDB v této situaci by pouze s PK nebyl dostatečný. Je to protože bychom se mohli dotazovat na zbytečně velké množství pixelů a zároveň, vzhledem ke zmíněnému omezení databáze, bychom nemuseli získat jako odpověď veškerá chtěná data. Z toho důvodu je potřeba v dotazu specifikovat i hodnotu SK. V tomto případě je způsob určení SK relativně triviální.

Jak je uvedeno v kapitole 5, SK je v databázi ve formátu desetimístného čísla, kde prvních 5 číslic představuje souřadnici na ose X, druhých 5 na ose Y. V případě území ČR jsou však hodnoty souřadnic mnohem větší čísla (konkrétně na jedné ose je souřadnice až desetimístná). To pro tuto práci není příliš velká komplikace, jelikož záznamy v databázi jsou pixely, které mají zaokrouhlené (a pravidelné) rozměry. Z toho důvodu je pro SK potřeba specifikovat pouze rozsah hodnot na dané ose. Pro tuto práci byl vybrán postup, že bude specifikován rozsah hodnot na ose X, avšak analogicky by se to dalo řešit i pomocí osy Y. Princip funkce, tvořící krajní hodnoty rozsahu pro SK je uveden na obrázku 23.



Obrázek 23 - Postup tvorby hodnoty SK

Tento postup je tedy proveden vždy pro obě mezní hodnoty na ose X, čímž je získán potřebný rozsah pro specifikaci dotazu na databázi. Vizuálně prakticky jde o dotazování se na určitý počet sloupců v rámci oblasti 10x10km. Jelikož rozsah osy Y není specifikován, jsou tázány všechny pixely ve sloupci a zmíněnou specifikací SK si „vybereme“ pouze ty potřebné sloupce.

7.3.4 Zúžení výběru pixelů

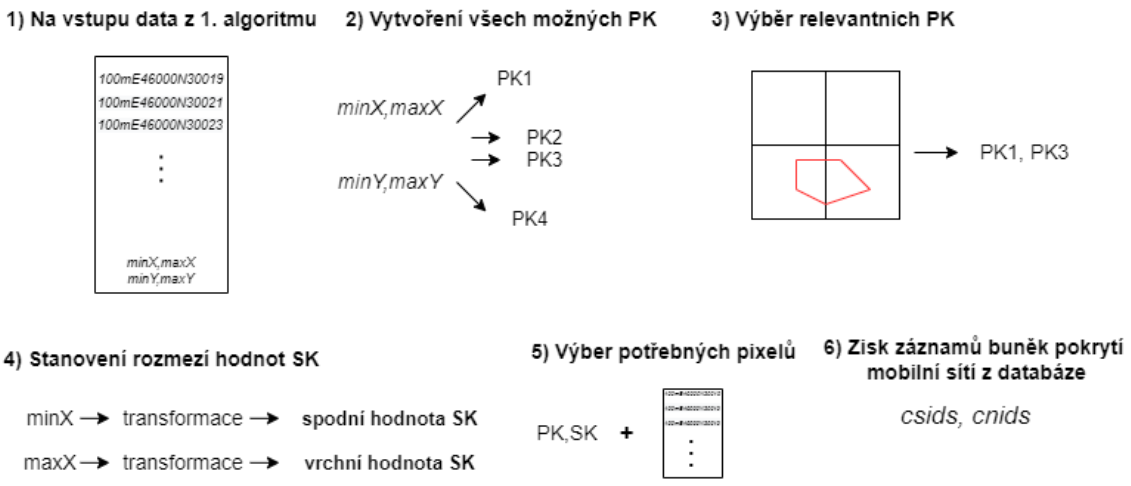
I přes specifikaci PK a SK (relativně často) nastává situace, že položený dotaz vrací příliš velké množství pixelů, které nesouvisí se zkoumaným polygonem. Lze si představit například úzký polygon, který má malý rozsah na ose Y, a naopak velký na ose X. Tím pádem bychom se dotazovali například na 8 sloupců z 10 možných v rámci oblasti 10x10km, avšak z každého sloupce bychom potřebovali jen malý počet pixelů. Tento problém je ošetřený za pomoci získaného seznamu pixelů, který je výstupem algoritmu A, jak je uvedeno v kapitole 7.2 na obrázku 17.

Jak je zmíněno v závěru kapitoly 6, oba algoritmy jsou implementovány v prostředí AWS Lambda, která umožňuje navrhnout funkci, jejímž vstupem jsou určitá data. A právě zmíněný seznam relevantních pixelů (výstup algoritmu A) je pokládaný na vstup

algoritmu B. Jelikož je seznam upraven do formátu databáze DynamoDB, není potřeba s ním nijak manipulovat a je pouze využit v dotazu. Princip spočívá v tom, že v rámci dotazu je implementována jednoduchá funkce, která prohlíží záznamy získané dotazem a porovnává parametr *pix* z daných záznamů databáze se seznamem na vstupu. Ve chvíli, kdy dojde ke shodě, je daný záznam poslaný na výstup dotazu. Pokud ke shodě nedojde, záznam se přeskakuje, čímž je dosaženo filtrace nepotřebných záznamů. Na výstupu dotazu (a tedy i algoritmu B) se pak nacházejí záznamy z databáze DynamoDB o pokrytí mobilní sítí, které pokrývají uživatelem nakreslený polygon. Z hlediska parametrů, které jsou z databáze dotazem získány se pak jedná o parametry *csigs* a *cnids*, které jsou vysvětleny v kapitole 5. V jednoduchosti se jedná o identifikátory buněk pokrytí, pomocí kterých lze pak dané pokrytí vizualizovat, což je názorně předvedeno v rámci aplikace (kapitola 8).

7.3.5 Shrnutí algoritmu B

Druhý vysvětlený algoritmus slouží jako spojovací funkce, která propojuje uživatelský vstup a interakci s daty uloženými v databázi DynamoDB. Po zisku transformovaných dat uživatelsky nakresleného polygonu z algoritmu A jsou tato prostorová data převedena do podoby, která koresponduje s databází a jsou z ní získány potřebné záznamy o pokrytí mobilní sítí. Shrnutí všech dílčích kroků a procesů algoritmu B je uvedeno na obrázku 24.



Obrázek 24 - Shrnutí algoritmu B

7.4 Omezení algoritmů

Oba popsané algoritmy jsou plně využívány v rámci webové aplikace (jak je předvedeno v následující sekci 8). Avšak funkcionalita obou algoritmů je omezena, především implementací.

Co se týče algoritmu A, tak z hlediska návrhu by měl být více méně univerzálně navržený – je potřeba pouze specifikovat souřadnicový systém, ve kterém je používaný a dále jeho celá funkcionalita pracuje prakticky pouze s hodnotami souřadnic. V rámci aplikace je použitelný pouze v oblasti České republiky. To je dáno faktem, že algoritmus B je navržen pro konkrétní aplikaci. Jelikož je algoritmus B závislý na databázi DynamoDB, obsahující data o pokrytí mobilní sítí v ČR, tak je dle toho i navržena funkcionalita algoritmu. Databáze má specifickou strukturu (názvy a typy atributů, struktura klíčů) a proto je algoritmus B použitelný pouze pro tento konkrétní případ, avšak úprava pro odlišnou strukturu databáze by neměla být příliš složitá (záleží na složitosti návrhu databáze). Pokud by například uživatel nakreslil polygon mimo ČR, nestalo by se více méně nic (nemohl by dále získat data o pokrytí, měl by pouze nakreslený polygon na mapě). Z hlediska možných tvarů uživatelem nakreslených polygonů, tak je více méně možno nakreslit libovolný tvar, krom tečky (nástroj na kreslení funguje na principu, že je potřeba nakreslit uzavřený útvar).

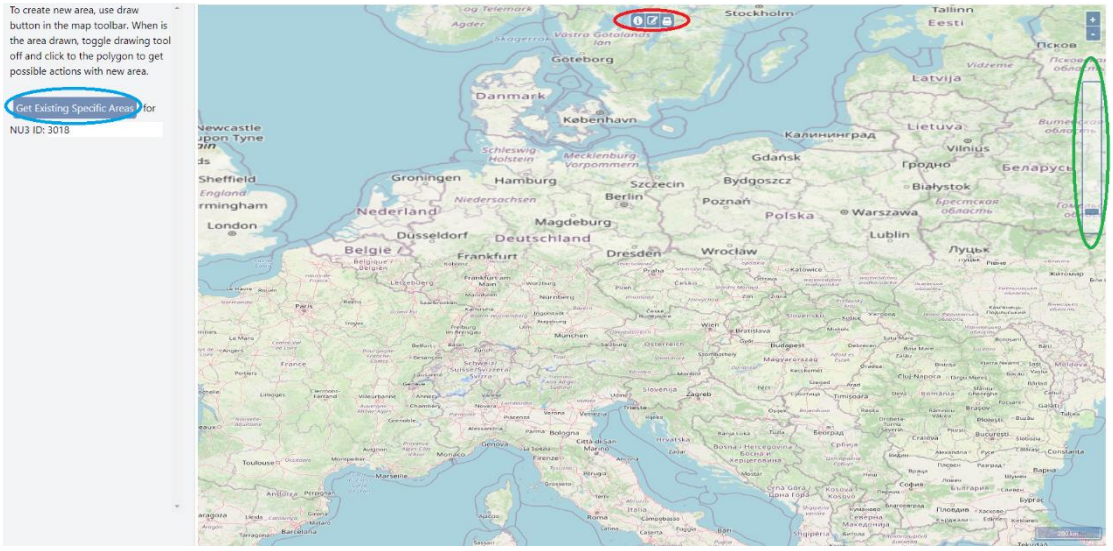
Hlavním „omezením“ je situace, kdy se uživatel rozhodne nakreslit obrovský polygon, například přes celou republiku. Algoritmus A by sice proběhl bez problémů a byly by získány příslušné souřadnice, ale dotazování na databázi by bylo takřka nemožné. Jak je uvedeno v sekci 7.3.2, nakreslený polygon může (v nejzazší situaci) zasahovat do čtyř oblastí 10x10km. Je tedy jaksi stanovené „maximum“, že polygon má mít nanejvýš 400km^2. I takový dotaz by více méně nemusel úspěšně proběhnout, kvůli omezení databáze DynamoDB (uvedené v sekci 7.3.1) na určitou velikost dat, kterou lze dotazem získat. Kdyby chtěl uživatel získat naráz veškeré pokrytí mobilní sítí v ČR, výstupem jeho dotazu by pravděpodobně byla jen část pokrytí, která se do odpovědi „vešla“. Avšak takováto situace je považována za extrémní, jelikož návrh aplikace je dělaný pro zkoumání konkrétnějších (a menších) území a zkoumané oblasti jsou řádově menší než 400km^2, takže v případě konkrétní implementace (tj. pro vyvíjenou aplikaci) nelze tento jev považovat za nedostatek.

8 Využití algoritmů v aplikaci

Jak bylo zmíněno v úvodu kapitoly 5 a 6, řešení popsaná v této práci jsou součástí projektu, který vytváří technickou infrastrukturu pro mapování a ukládání dat o pokrytí mobilní sítí v ČR. V tomto procesu dochází k vývoji interaktivní webové aplikace, která je v této kapitole stručně přiblížena. Oba vysvětlené algoritmy jsou v pozadí této aplikace aktivně využívány společně s velkým množstvím dalších funkcí, procesů a algoritmů. Vytvořené algoritmy jsou v aplikaci používány způsobem uvedeným v následujících sekcích.

8.1 Úvodní obrazovka a seznam území

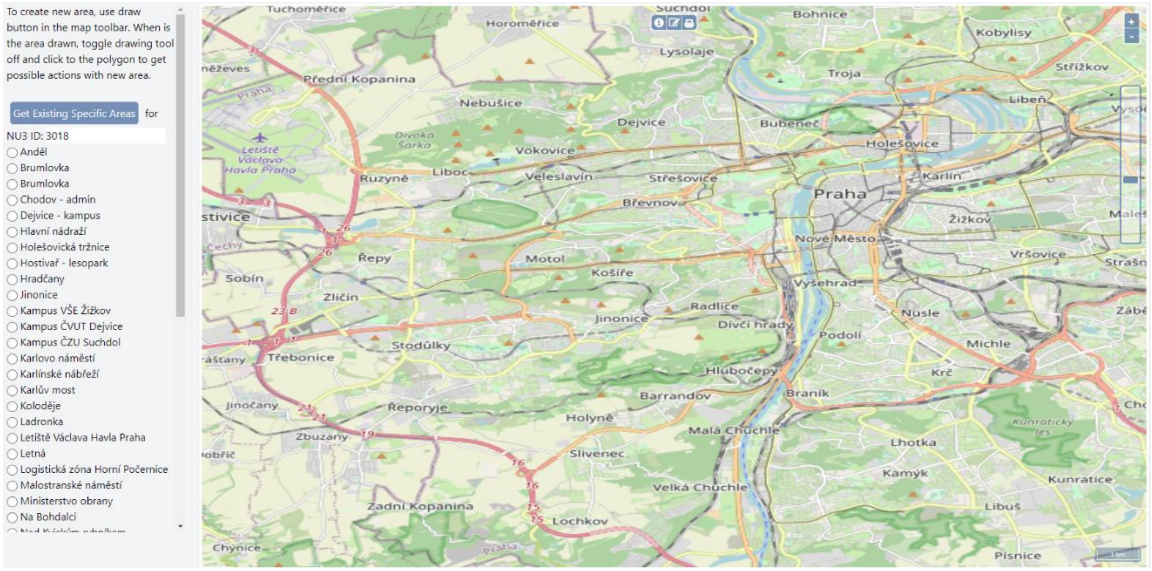
Při spuštění webového prohlížeče s aplikací se uživateli zobrazí mapa, obsahující větší část evropského území. Na levé straně je umístěno modré tlačítko, které slouží k automatickému přiblížení mapy na území Prahy a zároveň k vypsání krátkého seznamu předem vybraných míst v Praze (zakroužkováno modře). Jelikož je aplikace stále ve stádiu vývoje, je vybráno hlavní město jako oblast testování. Úvodní obrazovka aplikace je uvedena na obrázku 25.



Obrázek 25 - Úvodní obrazovka aplikace

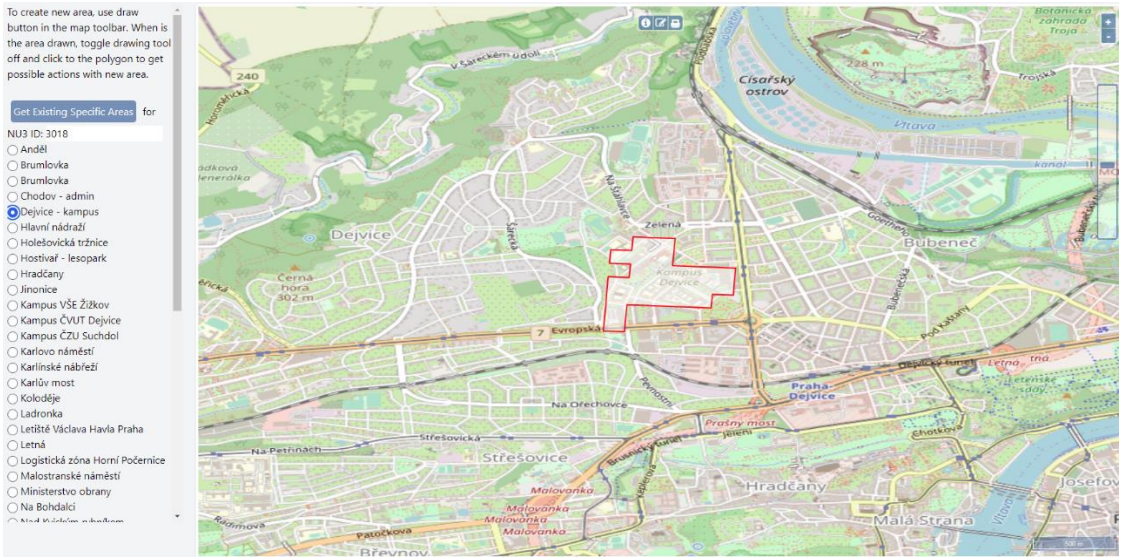
Společně s tlačítkem je na obrazovce několik dalších prvků. Nad zmíněným tlačítkem jsou uvedeny stručné instrukce v anglickém jazyce k používání aplikace společně s možnými funkcionalitami. Pod modrým tlačítkem je uveden text *NU3 ID: 3018*, což je identifikátor území, stanovený podle tzv. Nomenklatury územních statistických jednotek. Zobrazený identifikátor patří Praze, a proto po zmáčknutí tlačítka se zobrazí seznam často dotazovaných území v Praze. Tento identifikátor je měnitelný a lze jej libovolně nastavit na základě toho, které území v České republice má být podrobněji zkoumáno. Dále jsou na horní části obrazovky tři malá tlačítka (zakroužkováno červeně). Ta slouží k výběru typu interakce s mapovým rozhraním. Jejich konkrétní funkcionalita je přiblížena dále. Posledním prvkem je pohybné tlačítko po pravé straně (zakroužkováno zeleně), které slouží k manipulaci s přiblížením na mapě.

Po zmáčknutí modrého tlačítka dojde k přiblížení mapy do centra Prahy a k vypsání seznamu klíčových území. To je zobrazeno na obrázku 26.



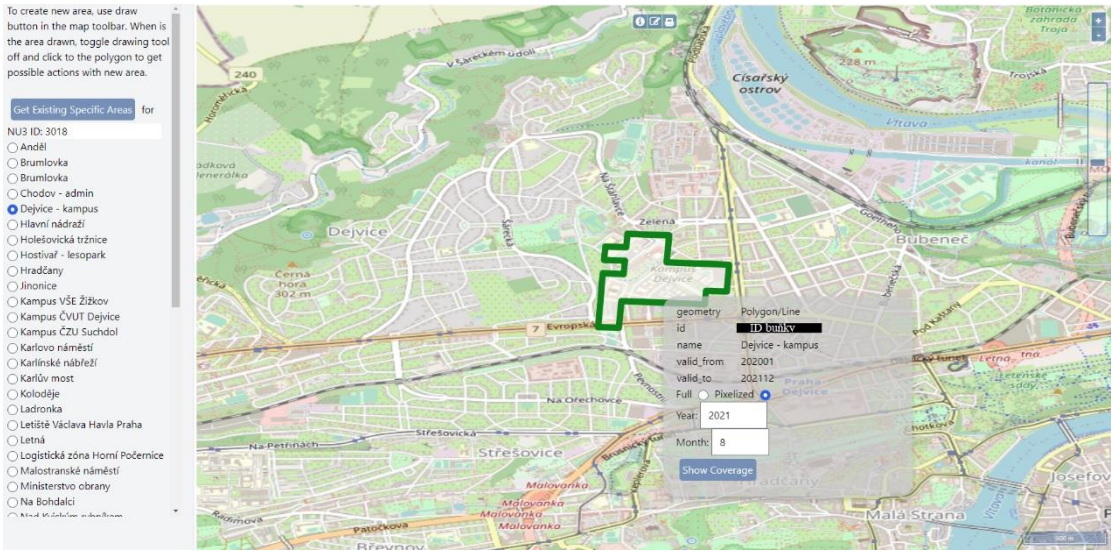
Obrázek 26 - Mapa Prahy se seznamem území

Uživateli je umožněno kliknutím do bílého kolečka vedle názvu území ze zobrazeného seznamu toto území vybrat, čímž se mapa automaticky přesune a přiblíží na zvolené území a zároveň se na mapě objeví polygon pokrývající toto území. Pro názornou ukázku bylo vybráno území kampusu ČVUT v Dejvicích, které je na obrázku 27 znázorněno červeným polygonem.



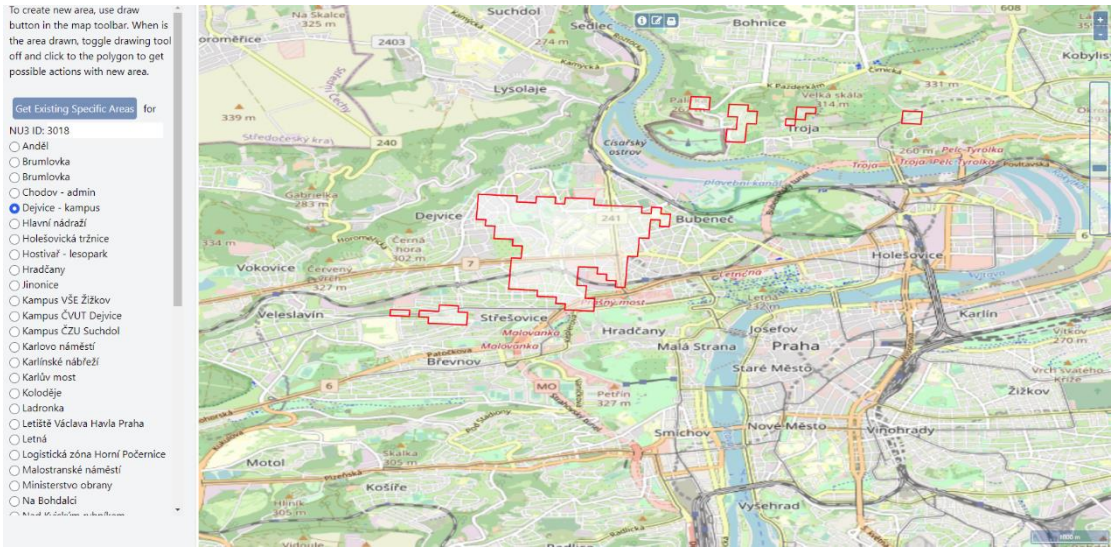
Obrázek 27 - Kampus ČVUT v Dejvicích

Takto zobrazený polygon však vyznačuje pouze zvolené území. Uživatel nyní může kliknout kamkoliv do polygonu, čímž se mu zobrazí okénko, obsahující určité informace a nabízející několik možností. Zároveň polygon změní barvu na zelenou, což naznačuje fakt, že uživatel do něj kliknul (tj. že je označený). V okénku se zobrazí informace, o co se jedná za útvar (polygon), jaký je jeho identifikátor v rámci databáze, název území a rozsah období, od jakého po jaký měsíc a rok je v databázi dostupná informace o pokrytí mobilní sítí. Posledními dvěma prvky je možnost zvolit si vzhled buněk pokrytí (pixelizované, tj hranaté či reálné – hladké čáry, nejedná se o „čtverečkovaný“ vzhled) a kolonky, kde si uživatel může zvolit rok a měsíc, ze kterého by chtěl obdržet informace o pokrytí mobilní sítí. Tato data jsou samozřejmě aktualizována a nebude vždy možné dohledat informace o pokrytí z dávné minulosti, avšak minimálně 2-3 roky by měly být v databázi vždy dostupné. Popsané okénko je zobrazeno na obrázku 28. Políčko uvnitř okénka je na obrázku začerněné z důvodu citlivosti údajů o buňkách pokrytí. Jedná se však o unikátní identifikátor buňky.



Obrázek 28 - Informace o zvoleném polygonu

V poslední řadě je pod kolonkami pro rok a měsíc pokrytí umístěno modré tlačítko s textem *Show Coverage* (zobrazit pokrytí), pomocí kterého se zobrazí pokrytí mobilní sítí ve zvoleném časovém období. Zobrazené pokrytí kampusu ČVUT v Dejvicích v srpnu roku 2021 je zobrazeno na obrázku 29.



Obrázek 29 - Pokrytí kampusu ČVUT v Dejvicích

Lze pozorovat několik jevů. Zaprvé, struktura polygonu pokrytí mobilní sítí je schodovitého charakteru, což je způsobeno již vysvětleným ukládáním dat do databáze DynamoDB, kde jsou tyto záznamy ve formátu pixelů. Při zvolení takto předem daného území v pozadí aplikace prakticky proběhne algoritmus B z této práce, který je

vysvětlen v kapitole 7.3. Území kampusu ČVUT v Dejvicích je tvořeno polygonem na obrázku 27 a tudíž je popsán i určitým „seznamem“ pixelů. Při dotazování se na databázi tedy proběhne dotaz a porovnání tohoto seznamu se záznamy v databázi, čímž jsou získány identifikátory buněk pokrytí, zasahujících do zkoumané oblasti. Ty jsou poté jednoduše vizualizovány do prostředí webové aplikace. Druhým pozoruhodným jevem je fakt, že buňka pokrytí mobilní sítí není jednotný polygon, nýbrž se skládá s několika různě velkých polygonů. To je v mobilních sítích běžná situace, jelikož pokrytí určitého území výrazně záleží na charakteru území. V městských částech je velké množství budov (některých i relativně vysokých), které negativně ovlivňují šíření signálu, a proto je časté, že se signál z vysílače vyskytne na neočekávaných místech (na obrázku 29 se jedná o malé polygony relativně daleko od zkoumané oblasti). Stěžejní informací je však do jaké míry je pokrytá zkoumaná oblast, což při pohledu na obrázek 29, lze usoudit, že tato oblast byla adekvátně pokryta mobilní sítí v daném zkoumaném období.

8.2 Uživatelský polygon

Z hlediska této práce je však důležitější funkce, která umožňuje uživateli nakreslit libovolný polygon, vytvořit si tak vlastní zkoumané území a získat data o pokrytí mobilní sítí z dostupného časového rozmezí. K tomu slouží zmiňovaná tlačítka na horní části obrazovky, která jsou vysvětlena v sekci 8.1.

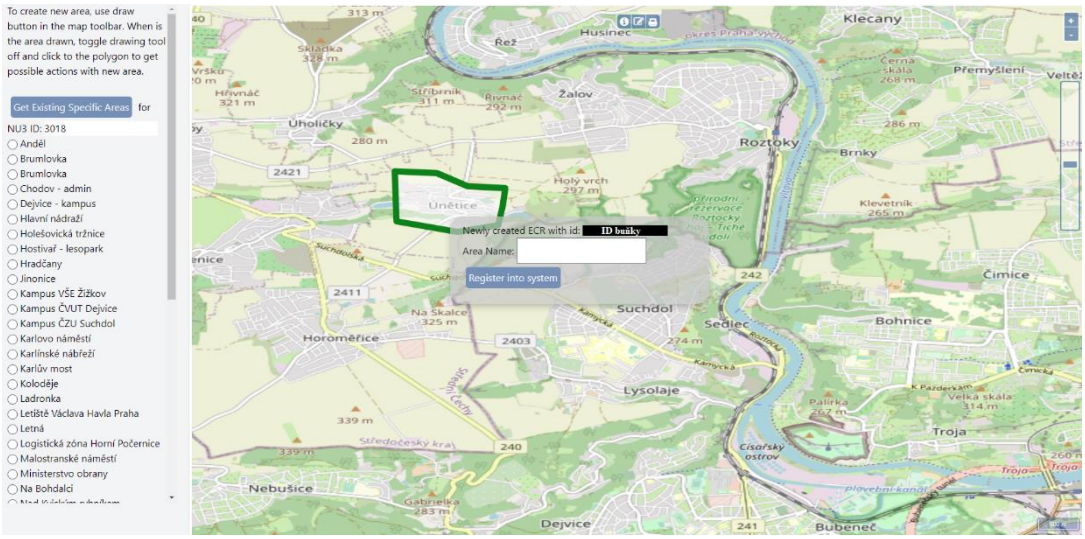
Obrázek 30 - Dostupná tlačítka

První tlačítko na obrázku 30 se znakem *„i“* slouží k získání informací o vybraném území či buňce pokrytí mobilní sítí. Prostřední tlačítko je z hlediska této práce nejdůležitější. Jedná se o spuštění kreslící tužky, pomocí které lze nakreslit na mapě libovolný polygon. V souvislosti s tím probíhají na pozadí oba algoritmy vysvětlené v této práci. Posledním tlačítkem je pouhé tlačítko tisku, které umožňuje vytvořit snímek aktuální obrazovky v podobě připravené k tisku.

Pro tuto práci má však největší význam „kreslící“ tlačítko. Po jeho zmáčknutí se uživateli umožní kreslit na mapě a zvolit tak zkoumané území. Pro demonstraci je na obrázku 31 uveden nakreslený polygon, zkoumající pokrytí na území Únětic.

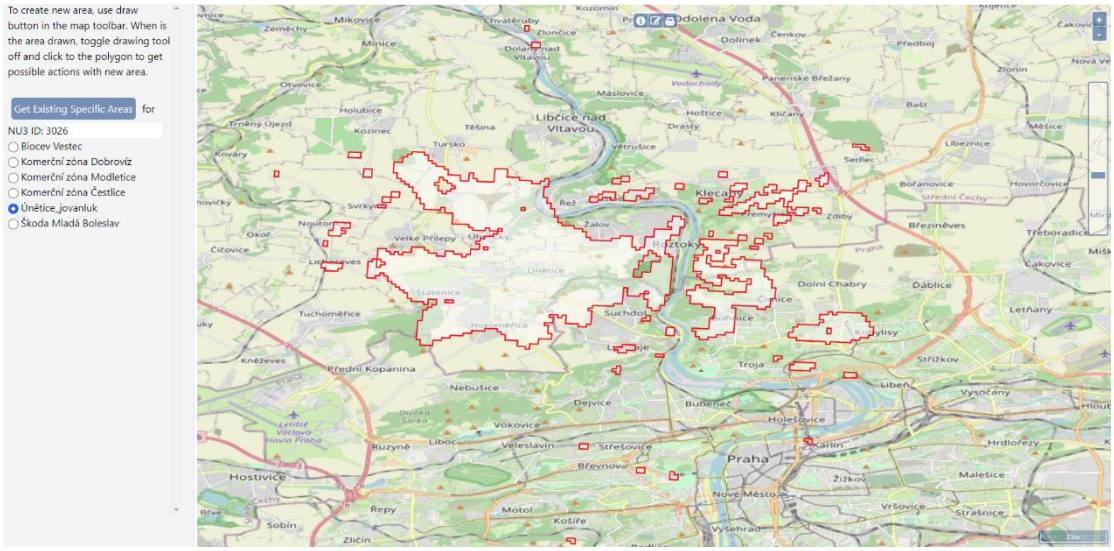
Obrázek 31 - Uživatelem nakreslený polygon v oblasti Únětic

Po dokončení kreslení je potřeba, aby uživatel stisknul opět tlačítko na kreslení, čímž tento nástroj vypne. Nyní v pozadí aplikace prakticky proběhl algoritmus A z této práce (sekce 7.1), kdy jsou souřadnice a data o polygonu upravena do potřebné podoby. K získání pokrytí je však potřeba tuto novou oblast registrovat do databáze území (tj vytvořit nový záznam, aby při zkoumání byl vypsán na seznamu na levé straně obrazovky). Toho je dosaženo opět kliknutím do polygonu, což vyvolá dialogové okno, které nejprve zobrazí text, že probíhá výpočet a poté přidělí tomuto území unikátní identifikátor a vyzve uživatele k pojmenování oblasti. Toto dialogové okénko je uvedeno na obrázku 32. Z důvodu citlivosti údajů je opět začerněno políčko s identifikátorem buňky.



Obrázek 32 - Pojmenování polygonu a tvorba záznamu

Po pojmenování území se vytvoří záznam v databázi a dojde k výpočtu pokrytí. Jinými slovy v pozadí nyní proběhl algoritmus B uvedený v této práci (sekce 7.2). Po získaných datech z klientské strany došlo k přepočtu a transformaci polygonu a zároveň k výpočtu ohraničující plochy. Poté jsou položeny dotazy na DynamoDB s parametry spočítanými pomocí algoritmu B (PK, SK, seznam relevantních pixelů), čímž se získají identifikátory buněk pokrytí mobilní sítí, zasahující do dané oblasti. Posledním krokem je vizualizace těchto buněk. I pro tento případ, tedy když uživatel kreslí polygon, se stejně jako na obrázku 28 zobrazí dialogové okno s informacemi a možnostmi výběru, z jakého časového období zobrazit pokrytí sítí. Stejně jako v případě v sekci 8.1 i pro nově nakreslený polygon v oblasti Únětic byl vybrán měsíc srpen roku 2021. Výsledné pokrytí mobilní sítí, získané za pomocí obou algoritmů této práce a dalších funkcí, které jsou součástí této aplikace je uveden na obrázku 33.



Obrázek 33 - Pokrytí mobilní sítí nakresleného polygonu

Při pohledu na obrázek lze pozorovat, že rozsah pokrytí oblasti Únětic je relativně veliké, což opět může být způsobeno profilem terénu a výškou budou (v okolí Únětic je pravděpodobně méně vysokých budov než v Dejvicích), čímž nedochází k výraznému rušení signálu.

Krom zmíněné vizuální reprezentace pokrytí mobilní sítí zkoumané oblasti je výstupem aplikace zároveň i seznam identifikátorů (a jiných parametrů) daných buněk, které zprostředkovávají pokrytí. Z hlediska následné práce a vývoje v rámci aplikace je tento seznam buněk rovněž klíčový, jelikož je poté nad ním prováděno množství dalších analytických funkcí. I přesto, že aplikace je stále ve stádiu vývoje, poskytuje již vhodné funkcionality pro získávání informací o pokrytí mobilní sítí. Díky platformě AWS je umožněno rychlé a efektivní ukládání a zpracovávání dat, což tvoří páteř celého procesu tvorby aplikace.

9 Závěr

Tato práce se zabývá tvorbou algoritmů pro zpracování dat o pokrytí mobilní sítí v rámci cloudových služeb AWS. Cílem práce bylo vytvořit dva algoritmy, umožňující transformaci vstupních dat získaných od strany uživatele do vhodné podoby, aby nad nimi bylo možno provádět prostorové operace a dále tato data propojit s databází DynamoDB o pokrytí mobilní sítí v České republice.

V první části práce je uveden stručný teoretický úvod, popisující principy *cloud* a *edge computing*, společně s platformou pro provozování cloudových služeb AWS. Zároveň byly podrobněji vysvětleny konkrétní použité nástroje v rámci platformy AWS, pomocí kterých bylo možno aplikovat navržené algoritmy do vyvíjející se webové aplikace.

K realizaci algoritmů bylo použito programovacího jazyka JavaScript, a především prostředí AWS Lambda (vysvětlené v kapitole 4.3), které umožňuje jednoduchou integraci libovolně vytvořených funkcí do platformy AWS. Klíčové to je z toho důvodu, že data o pokrytí mobilní sítí jsou uložena v databázi DynamoDB, na kterou je jednoduché se prostřednictvím funkcí v AWS Lambda dotazovat. V hlavní části práce (kapitola 7) je vysvětlen algoritmus A pro úpravu vstupních dat z klientské strany a algoritmus B pro modifikaci získaných dat do podoby vhodné pro dotazování na DynamoDB. Z hlediska adekvátnosti algoritmů, se navržené algoritmy jeví jako vhodné, jelikož jsou aktivně používány v rámci aplikace a neprojevují víceméně žádné nedostatky (krom omezení uvedených v sekci 7.4). Co se týče alternativních přístupů, tak jednou z možností je navrhovat algoritmy v jiném programovacím jazyce (například Python), avšak navržené algoritmy nepracují s nadměrně složitými procesy a lze předpokládat, že jiná řešení by ve finálním stádiu neprojevovala žádné markantní rozdíly ve funkcionalitě a zátěži systému.

V poslední části jsou funkcionality algoritmů demonstrovány v rámci aplikace v prostředí webového prohlížeče. Aplikace jako taková je stále ve stádiu vývoje a na jejím vývoji se podílí více osob. Sekce 8 obsahuje vysvětlení veškerých dosavadních funkcionalit aplikace, které využívají jak zmíněné algoritmy, tak i množství dalších funkcí.

Z hlediska budoucího vývoje je plánováno rozšířit škálu možností a funkcionalit v rámci aplikace. Krom drobných úprav ve funkcionalitě rozhraní je aktuálně ve stádiu vývoje přidaná funkcionalita získávání statistik chování obyvatel v dané oblasti. Jednalo by se především o statistiku o mobilitě s tím, že by bylo možné tuto statistiku dynamicky spočítat i pro novou oblast (podobně jako algoritmus B v této práci) vytvořením úlohy pro výpočetní centrum.

Seznam zdrojů

[1] MARR, Bernard. *Big Data: USING SMART Big Data Analytics and Metrics To Make Better Decisions and Improve Performance*. 1. West Sussex, United Kingdom: John Wiley, 2015. ISBN ISBN 978-1-118-96583-2.

[2] SAUTER, Martin. *From GSM to LTE-Advanced Pro and 5G: An Introduction to Mobile Networks and Mobile Broadband*. 3rd ed. Hoboken, NJ, USA: Wiley, 2017. ISBN 9781119346906.

[3] BAESENS, Bart. *Analytics in a big data world: the essential guide to data science and its applications*. Hoboken: John Wiley, [2014]. ISBN ISBN978-1-118-89270-1.

[4] ZAIGHAM, Abbas, Javaid ARSLAN a Hammad MUHAMMAD. *CLOUD COMPUTING*. Únor, 2022, 1-13. Dostupné také z: https://www.researchgate.net/publication/358404195_CLOUD_COMPUTING

[5] KIM, Won. Cloud Computing: Today and Tomorrow. *JOURNAL OF OBJECT TECHNOLOGY* [online]. Suwon, S. Korea: ETH Zurich, 2009, **1**(8) [cit. 2022-04-19]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.591.2485&rep=rep1&type=pdf

[6] LAU, Kung-Kiu, Winfried LAMERSDORF a Ernesto PIMENTEL. SECOND EUROPEAN CONFERENCE, ESOCC. *Service-Oriented and Cloud Computing* [online]. Málaga, Španělsko: Springer, 2013, 253 s. [cit. 2022-04-19]. ISBN 978-3-642-40651-5. ISSN 0302-9743. Dostupné z: doi:10.1007/978-3-642-40651-5

[7] SHAHZADI, Sonia, Muddesar IQBAL, Zia Ul QAYYUM a Tasos DAGIUKLAS. Infrastructure as a service (IaaS): A comparative performance analysis of open-source cloud platforms. *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2017, 2017, 1-6. ISBN 978-1-5090-6302-4. Dostupné z: doi:10.1109/CAMAD.2017.8031522

[8] AL MORSY, Mohamed, John GRUNDY a Ingo MÜLLER. *An Analysis of the Cloud Computing Security Problem*. Hawthorn, Victoria, Australia: Computer Science & Software Engineering, Faculty of Information & Communication Technologies Swinburne University of Technology. Dostupné také z: https://arxiv.org/ftp/arxiv/papers/1609/1609.01107.pdf

[9] XIONG, Huanhuan, Frank FOWLEY a Claus PAHL. An Architecture Pattern for Multi-Cloud High Availability and Disaster Recovery [online]. Dublin City University: The Irish Centre for Cloud Computing and Commerce [cit.

2022-05-06]. Dostupné z: https://www.researchgate.net/profile/Claus-Pahl/publication/301670228_A_Database-Specific_Pattern_for_Multi-cloud_High_Availability_and_Disaster_Recovery/links/59df07f1aca27247d7a2488c/A-Database-Specific-Pattern-for-Multi-cloud-High-Availability-and-Disaster-Recovery.pdf

[10] WOOD, Timothy, Emmanuel CECCHET, K.K. RAMAKRISHNAN, Prashant SHENOY, Jacobus VAN DER MERWE a Arun VENKATARAMANI. *Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges*. University of Massachusetts Amherst. Dostupné také z: https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Wood.pdf

[11] GORELIK, Eugene. *Cloud Computing Models*. Massachusetts Institute of Technology, 2013. Dostupné také z: http://hdl.handle.net/1721.1/79811

[12] BRIAN, Olivier, Thomas BRUNSCHWILER, Heinz DILL, et al. Cloud Computing. Zürich: Swiss Academy of Engineering Sciences, 2012, 1-51. Dostupné také z: https://widmer.ch/fileadmin/templates/publikationen/20121106_SATW_White Paper_CloudComputing_EN.pdf

[13] CAO, Keyan, Yefan LIU, Gongjie MENG a Qimeng SUN. An Overview on Edge Computing Research. *IEEE Access*. 2020, **8**, 85714-85728. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2020.2991734

[14] GUPTA, Lav, Raj JAIN a H. Anthony CHAN. *Mobile Edge Computing: An Important Ingredient of 5G Networks*. Washington University, St. Louis: Huawei Technologies, 2016. Dostupné také z: https://sdn.ieee.org/newsletter/march-2016/mobile-edge-computing-an-important-ingredient-of-5g-networks

[15] HU, Yun Chao, Milan PATEL, Dario SABELLA, Nurit SPRECHER a Valerie YOUNG. ETSI. *Mobile Edge Computing: A key technology towards 5G*. ETSI (European Telecommunications Standards Institute), 2015. ISBN 979-10-92620-08-5. Dostupné také z: https://infotech.report/Resources/Whitepapers/f205849d-0109-4de3-8c47-be52f4e4fb27_etsi_wp11_mec_a_key_technology_towards_5g.pdf

[16] GUILLÉN, Miguel A., Antonio LLANES, Baldomero IMBERNÓN, Raquel MARTÍNEZ-ESPAÑA, Andrés BUENO-CRESPO, Juan-Carlos CANO a José M. CECILIA. Performance evaluation of edge-computing platforms for the prediction of low temperatures in agriculture using deep learning. The Journal of Supercomputing [online]. 2021, 77(1), 818-840 [cit. 2022-05-06]. ISSN 0920-8542. Dostupné z: doi:10.1007/s11227-020-03288-w

[17] KEKKI, Sami, Rohit ARORA, Luis M. CONTRERAS, et al. MEC in 5G networks. Francie: ETSI (European Telecommunications Standards Institute), 2018. ISBN 979-10-92620-22-1. Dostupné také z: https://networking.report/Resources/Whitepapers/4c3cff17-94ba-45f0-bf47-ec169d7b762a_etsi_wp28_mec_in_5G_FINAL.pdf

[18] *DEBRIE, Alex. The DynamoDB Book [online]. Version 1.0.1. Gumroad, 2020 [cit. 2022-04-15].*

[19] *Amazon DynamoDB: Fast, flexible NoSQL database service for single-digit millisecond performance at any scale* [online]. Amazon Web Services, 2022 [cit. 2022-04-17]. Dostupné z: https://aws.amazon.com/dynamodb/

[20] *Amazon S3: Object storage built to retrieve any amount of data from anywhere [online]. Amazon Web Services, 2022 [cit. 2022-04-17*]. Dostupné z: https://aws.amazon.com/s3/?nc=sn&loc=1

[21] *AWS Lambda: Run code without thinking about servers or clusters* [online]. Amazon Web Services, 2022 [cit. 2022-04-17]. Dostupné z: https://aws.amazon.com/lambda/