Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

IoT informační systém pro zpracování dat z různých senzorů v multi LPWAN prostředí

Ladislav Herynek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Ladislav Herynek

Informatika

Název práce

IoT informační systém pro zpracování dat z různých senzorů v multi LPWAN prostředí

Název anglicky

The IoT information system for processing data from various sensors within multi LPWAN environment

---

Cíle práce

Cílem práce je navrhnout a vytvořit IoT informační systém, který integruje různé LPWAN sítě a zpracovává data z různých senzorů. Informační systém se bude skládat ze senzorické části snímající fyzikální veličiny, LPWAN síťové vrstvy, RESTful API a webové aplikace k základní prezentaci dat. Klíčovým prvkem systému jsou služby, které budou provozovat senzory v různých sítích a zpracovávat získaná data. Celý systém bude navržen v souladu s architektonickým vzorem Microservices.

Metodika

Základem teoretické části je literární rešerše odborné literatury. Tato část se bude věnovat metodikám vývoje software s důrazem na systémy s maximální dostupností a vysokou komplexitou a také IoT zejména LPWAN sítím. Záměrem této části je popsat moderní přístupy k vývoji software se zaměřením na agilní techniky, reaktivní architekturu společně s technikami Domain-Driven designu a architektonický vzor Microservices.

Praktická část se bude věnovat návrhu a vývoji informačního systému, který se bude skládat z backend a frontend části. Backend bude poskytovat data, integrační vrstvu pro jednotlivé LPWAN sítě a metody pro obsluhu senzorů. Frontend bude vybraná data prezentovat a uživateli umožní základní práci se senzory. Backend bude napsán v jazyce C# a frontend využije nějaký z moderních frameworků ke stavbě moderního UI. Součástí vývoje je také stavba několika senzorů, které budou snímat vybrané fyzikální jevy a testovat očekávané chování systému.

Závěr bude formulován na základě získaných poznatků z průběhu stavby systému a jeho provozu.

Doporučený rozsah práce

60-80 stran

Klíčová slova

IoT, LPWAN, Cloud, Reactive Architecture, Microservices, Domain Driven Design

---

Doporučené zdroje informací

GHOSH, Debasish. Functional and Reactive Domain Modeling. United States of America: Manning, 2017. ISBN 9781617292248.

JAMSA, Kris. Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More. United States of America: Jones & Bartlett Learning, 2012. ISBN 978-1449647391.

NEWMAN, Sam. Building Microservices. 2nd ed. United States: O'Reilly Media, 2021. ISBN 9781492034025.

SALAZAR, Jordi, Santiago SILVESTRE a Jaromír. Internet věcí. Praha: České vysoké učení technické, 2017. ISBN 978-80-01-06231-9.

SELECKÝ, Matúš. Arduino: Uživatelská příručka. Brno: Computer Press, 2016. ISBN 978-80-251-4840-2.

VERNON, Vaughn. Implementing Domain-Driven Design. Amsterdam: Addison-Wesley Longman, 2013. ISBN 978-0321834577.

VURAL, Hulya a Murat KOYUNCU. Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?. IEEE Access [online]. 2021, 9, 32721-32733 [cit. 2022-05-31]. ISSN 21693536. Dostupné z: doi:10.1109/ACCESS.2021.3060895

ZENNARO, Marco a Suresh BORKAR. LPWAN Technologies: Emerging Application Characteristics, Requirements, and Design Considerations. Future Internet [online]. 2020, 12(3), 46-46 [cit. 2022-05-31]. ISSN 19995903. Dostupné z: doi:10.3390/fi12030046

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 11. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 29. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "IoT informační systém pro zpracování dat z různých senzorů v multi LPWAN prostředí" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.03.2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Marku Píckovi, Ph.D., za rady a odborné vedení mé diplomové práce.

IoT informační systém pro zpracování dat z různých senzorů v multi LPWAN prostředí

Abstrakt

Tato diplomová práce popisuje stavbu informačního systému, který se zaměřuje na správu a integraci aktivních prvků z různých nízkoenergetických sítích pokrývajících rozsáhlá území (LPWAN). Podstatná část práce je zaměřena na analýzu a návrh, které využívají moderních návrhových vzorů. Konkrétně se jedná o návrhový vzor Domain Driven Design, jenž komplexní systémy dekomponuje na menší logické nezávislé subdomény, definuje zodpovědnosti a komunikační pravidla mezi vývojovými týmy. Na DDD navazuje architektonický vzor mikroslužby, který pro každou z nezávislých subdomén navrhne vlastní množinu malých služeb dohromady utvářejících aplikační celek. Smyslem práce je dokázat provázanost těchto vzorů s výpočetním modelem cloud computing, který je jednou ze čtyř vrstev architektury IoT systémů. Cloud computing je v této práci reprezentován prostředím Microsoft Service Fabric a vlastní komponenty jsou napsány v jazyce C#.

Klíčová slova: IoT, LPWAN, Cloud computing, Mikroslužby, Návrh řízený doménou, Microsoft Service Fabric, Model, Kontext, Brainstorming, Entita, Služba, DDD

The IoT information system for processing data from various sensors within multi LPWAN environment

Abstract

The thesis describes the construction of an information system that focuses on the integration and management various entities such sensors or actuators from various low-power wide area networks (LPWAN). The main part of the studies is focused on analysis and design within modern design patterns. Specifically, it is a Domain Driven Design pattern, which decomposes complex system into smaller logical independent subdomains, which defines responsibilities and communication rules between development teams. DDD is followed by the microservice architecture pattern. This pattern for the each of the subdomains proposes own set of small services that together create an application. The main purpose of the thesis is to prove the relationship between the patterns and cloud-based computing model. In the thesis, cloud-based computing model is represented by Microsoft Service Fabric environment with .NET framework. Own created components are written in C# language. The computing model is one of the four layers of IoT architecture systems.

Keywords: IoT, LPWAN, Cloud computing, Microservices, Domain Driven Design, Microsoft Service Fabric, Model, Context, Brainstorming, Entity, Service, DDD

Obsah

Seznam obrázků

Seznam použitých zkratek

IoT	Internet of Things
DDD	Domain Driven Design
LPWAN	Low Power Wired Area Network
mMTC	massive Machine-Type communication
uRLLC	ultra-high reliability and ultra-low latency
ICT	Informační a komunikační technologie
IS	Informační systém
ČRa	České radiokomunikace
API	Application Programming Interface
MVP	Minimum viable product

1 Úvod

Ve své bakalářské práci jsem se věnoval tématu IoT v precizním zemědělství se zaměřením na senzorickou vrstvu a vrstvu výměny dat. Praktická část spočívala ve stavbě stanice se senzory využitelnými v zemědělství, které měřily teplotu a vlhkost vzduchu nebo vlhkost půdy. Stanici měla nezávislé napájení a bylo možné ji připojit kdekoliv, kde byl signál vybrané LPWAN sítě. Tato práce navazuje na zmíněnou bakalářskou práci a na problematiku IoT se zaměřuje z pohledu integrační a aplikační vrstvy. Integrační vrstva je typicky server, který ukládá nebo transformuje nasnímaná data, která pak jsou prezentována prostřednictvím aplikace koncovému uživateli. Tyto servery často poskytují provozovatelé nízkoenergetické sítě (LPWAN) a jsou hostována ve velkých datových centrech, protože v IoT očekáváme zpracování velkého množství malých dat. Diplomová práce je vlastně studií, jak hluboké jsou vzájemné vtahy mezi cloud-computing výpočetním modelem a moderními návrhovými vzory, které reprezentují Domain Driven Design a architektura mikroslužeb. Studie je provedena na reálném řešení, které spočívá v návrhu a realizaci systému, který spravuje aktivní prvky připojené do různých LPWAN sítí a poskytuje o nich základní informace, včetně přenesených zpráv. Přičemž návrh je stěžejní pilíř práce.

Práce je rozdělena do dvou částí na teoretickou a praktickou část. Teoretická část se věnuje nízkoenergetickým sítím (LPWAN), cloudovému výpočetnímu modelu, architektuře mikroslužeb a návrhu řízenému doménou (Domain Driven Design). Cílem teoretické části je získat základní poznání o každé z výše uvedených oblastí a vybrat vhodné postupy pro praktickou realizaci. Praktická část se věnuje zejména návrhu v podobě rozborky komplexního problému na nezávislé subdomény a identifikaci mikroslužeb. K popisu je využívána řada diagramů, které jsou součástí obou metodik. Na závěr práce je realizována jedna vybraná subdoména prostřednictvím programovacího jazyka C# a prostředí Microsoft Service Fabric. Reálná data pro testování vytvořeného systému zajistí dva senzory, jeden v rámci sítě Sigfox a druhý v rámci sítě LoRaWAN. Součástí práce není kompletní výpis zdrojového kódu, jsou vybrány pouze relevantní klíčové části.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je navrhnout informační systém, který bude integrovat a spravovat různé aktivní prvky provozované v nízkoenergetických sítích (LPWAN) různého typu. K návrhu systému budou využity návrhové vzory Domain Driven Design (DDD) a architektura mikroslužeb. Na základě výše zmíněného návrhu bude vybrán aspoň jeden ohraničený kontext, který bude realizován jako cloudová služba, protože záměrem je realizovat systém, který je responsivní, elastický a odolný vůči chybám. Dílčím cílem je návrh obrazovek klienta a jejich realizace prostřednictvím vybraného rozhraní a programovacího jazyka.

2.2 Metodika

Teoretická i praktická část práce čerpá ze studia odborné nebo vědecké literatury, odborných internetových zdrojů jako jsou vědecké nebo odborné články, návody k produktům a firemní prezentace. To vše je doplněno o vlastní zkušenosti získané v praxi na pozici .NET vývojáře cloudových informační systémů.

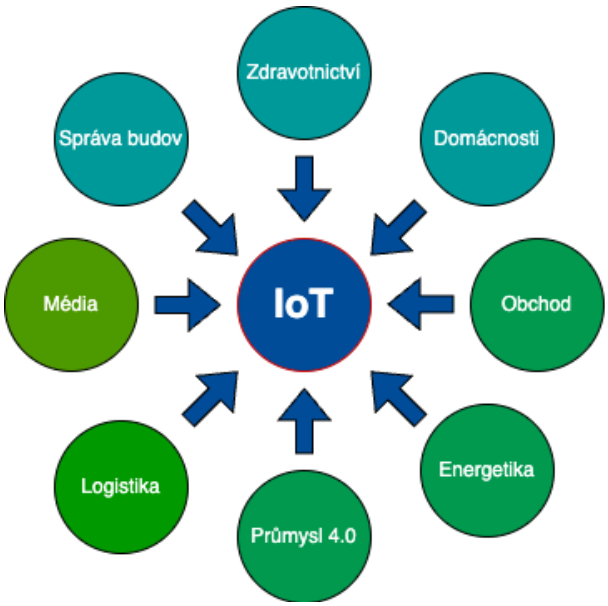
V teoretické části je smyslem získat dostatečné znalosti pro realizaci praktické části. Ke studiu bude využito řady odborných zdrojů většinou v elektronické podobě. Tato část bude zaměřena na základní prvky celého systému, a to zejména na Internet věcí (IoT), výpočetní model cloud computing, návrhový vzor Domain Driven Design a architekturu mikroslužeb. Výchozím bodem teoretické studie je IoT, které má přímou vazbu na cloud computing, jenž je vhodný pro provoz malých služeb, jejichž identifikaci usnadňuje metodika DDD.

Praktická část bude o návrhu systému zakončená jeho částečnou realizací. K popsání návrhu bude využito různých nástrojů vycházejících z metodiky DDD nebo architektury mikroslužeb. Na základě návrhu, bude vybrán aspoň jeden kontext, který bude realizován jako cloudová aplikace. Testováním bude ověřena kompatibilita chování aplikace s definovanými funkčními požadavky a získané poznatky z návrhu, vývoje a testování budou využity k formulování poznání a závěru práce.

3 Teoretická východiska

3.1 Internet věcí

IoT je koncept, který se zrodil mezi lety 2008 až 2009 [1]. V poslední dekádě zaznamenala tato oblast dynamický vývoj, který úzce souvisí s významným vývojem technologií v oblastech cloud computing, umělé inteligence, sítí nové generace, mobilní zařízení, nositelná elektronika nebo oblast zpracování a ukládání dat, a to vše je podpořeno miniaturizací a zvyšováním výkonu hardwarového vybavení. Je to oblast, která zasahuje do každodenních činností člověka v různých odvětvích, jak znázorňuje obrázek č. 3.1. Internet věcí lze charakterizovat jako spojení globální počítačové sítě s její schopností propojit cokoliv kdekoliv kdykoliv a věcí, které mají nějakou vlastní inteligenci, vlastní čip. Mezi věci lze zahrnout počítače, senzory, aktivní prvky podporující automatizaci nějakého procesu nebo také software, který je připojen k internetu. Se vzrůstajícím počtem věcí připojených do Internetu budou přímo úměrně vzrůstat nároky na datová centra ukládající a zpracovávající data, protože IoT bude obrovským zdrojem malých dat. S rozvojem IoT se budou rozvíjet i další obory, pro které jsou data zdrojem nových poznání a zlepšování procesů. Například lze zmínit data mining jenž statistickými metodami hledá netriviální závislosti v datech nebo umělou inteligenci (hluboké učení), která ke svým algoritmům potřebuje data, ze kterých se může učit. Souhrnně lze napsat, že IoT je obor podporující rozhodování s minimalizací chyb a maximalizací získané hodnoty.



Obrázek 3.1 - trhy IoT [2]

Neexistuje jedna vlastní definice IoT. Naopak existuje mnoho organizací, které mají vlastní definice, co je nebo není Internet věcí. Níže jsou uvedeny tři příklady od tří světových organizací.

Organizace IEEE (Institute of Electrical and Electronics Engineers) je světová organizace určující standardy a pečující o rozvoj v technických oborech.

"A network of items — each embedded with sensors — which are connected to the Internet." [2]

ETSI (Evropský ústav pro telekomunikační normy) je organizace, která definuje standardy pro informační a komunikační technologie a v rámci Evropské unie spolupracuje na tvorbě legislativy nebo na evropských standardech. ETSI vymezuje IoT v definici protokolu m2m, což je základní komunikační protokol v Internetu věcí.

"Machine-to-Machine (M2M) communications is the communication between two or more entities that do not necessarily need any direct human intervention. M2M services intend to automate decision and communication processes." [2]

IETF (The Internet Engineering Task Force) je velká mezinárodní otevřená komunita sdružující profesionály, operátory, dodavatele a vědce z oblasti internetových technologií. Spolupracuje s organizacemi jako W3C nebo ISO/IEC.

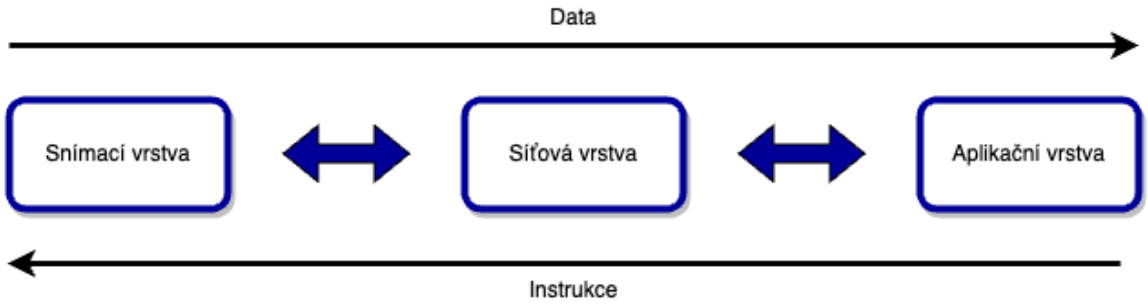
"The basic idea is that IoT will connect objects around us (electronic, electrical, non-electrical) to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and co-operate each other to make the service better and accessible anytime, from anywhere." [2]

Z výše uvedených definic vyplývá, že IoT je globální síť inteligentních prvků, které vzájemně interagují.

3.1.1 Architektura

IoT systém je o třech důležitých prvcích: zařízení, sítě a data. IoT instalace vynikají svou různorodostí, protože jsou používána v různých oblastech lidské činnosti od chytrých měst, precizního zemědělství, logistiky až po oblečení nebo chytrou domácnost. Kromě těchto specifických instalací lze zařadit mezi IoT i tablety, mobily nebo počítače. Souhrnně, lze napsat, že součástí IoT může být jakékoliv zařízení, které má konektivitu a pořizuje nějaká data, z čehož plyne, že data jsou vysoce heterogenní. Při jejich klasifikaci se řídíme pravidlem šesti V [3]:

- velikost (volume),
- hodnota (value),
- datový typ (variaty),
- proměnlivost (variability),
- důvěryhodnost (veracity),
- rychlost jejich vzniku (velocity).



Obrázek 3.2 - Architektura IoT [2]

IEEE definuje architekturu jako třívrstvou, viz obrázek 3.2 [2]:

- Snímací vrstva zahrnuje různé senzory a aktivní prvky, které obvykle snímají nějaké fyzikální jevy, získávají data nebo automatizují vybranou činnost.
- Síťová vrstva zodpovídá za přenos získaných dat do datového úložiště. Datové přenosy se odehrávají mezi aktivními prvky, síťovými branami nebo servery. Často je to komunikace mezi stroji, která rozšiřuje standardní internetové protokoly (http, smtp) o nový typ komunikačního protokolu m2m (machine to machine). K přenosu dat se využívá různých typů sítí. Základní členění je na sítě s dlouhým nebo krátkým dosahem, viz obrázek 3.3.
- Aplikační vrstva jsou servery nebo aplikace pro koncové uživatele. Servery jsou obvykle provozované jako cloudová služba a plní různé úlohy, například řízení síťového provozu, integrátora, datového úložiště nebo poskytují aplikační logiku pro práci s daty.

- Komunikace mezi prvky je obousměrná, kdy jedním směrem putují nasnímaná data a druhým směrem od severů k senzorům se přenáší instrukce, jak se má vybraný aktivní prvek chovat nebo potvrzení o přijetí zprávy. Instrukce může obsahovat i informace o přidání nebo odebrání aktivního prvku z provozovaného řešení.

3.1.2 Typy IoT instalací

Hromadná M2M (massive Machine-Type communication – mMTC) instalace o mnoha nízkonapěťových aktivních prvcích (senzory), které průběžně v pravidelných sekvencích přenáší malé množství dat do datových center. Senzory obvykle snímají nějaký fyzikální jev jako teplota, tlak, vlhkost. Instalace najdeme například v oblasti precizního zemědělství nebo ve správě budov. Síť definují parametry jako nízké náklady, nízká spotřeba elektrické energie, přenášená data mají malou velikost, signál pokrývá velké plochy a existuje mnoho aktivních koncových prvků. Vhodnými síťovými technologiemi jsou například LoRa, Sigfox nebo NB-IoT. [4]

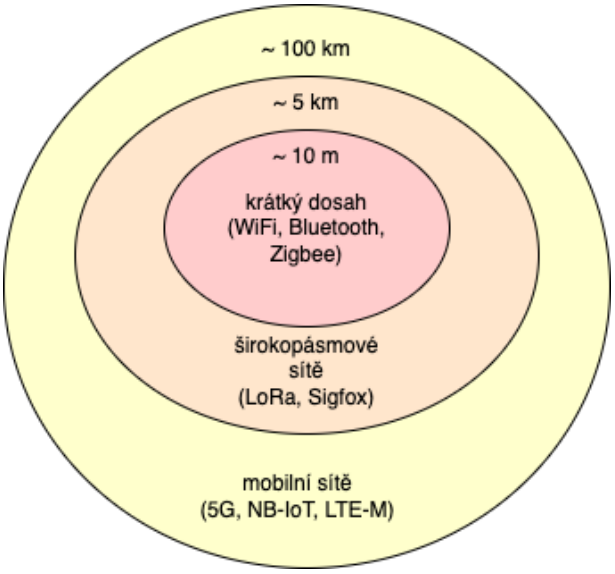
Kritická (ultra-high reliability and ultra-low latency – uRLLC) je protipólem hromadné instalace. Přenáší se větší objemy dat. Je to instalace náročná na minimální odezvu (latency) a je požadována větší šířka pásma oproti mMTC nebo vysoká dostupnost. Typickými příklady instalací jsou roboti nebo autonomní vozidla. [4]

Průmyslová (Industrial IoT – IIoT) zahrnuje instalace v rámci průmyslu 4.0, což je nová éra průmyslu, jeho digitalizace. K optimalizaci výroby bude docházet skrze využití robotizace, automatizace a datové analýzy. Sítě použité v této instalaci musí umět pracovat s průmyslovými protokoly a jsou citlivé na čas a synchronizaci aktivit. [4]

Celulární širokopásmová instalace (Broadband IoT) v porovnání s mMTC má vyšší propustnost dat, menší dobu odezvy (latency) a vyšší spotřebu energie. Zahrnuje tradiční celulární sítě založené na technologiích GSM nebo LTE. Typickými příklady použití jsou augmentová realita, nositelná elektronika, drony nebo senzory, které potřebují vyšší propustnost dat, než kterou nabízí LPWAN. [4]

3.1.3 Síťová vrstva (LPWAN)

V IoT je první hop, typicky je to přenos dat mezi senzorem a branou, bezdrátový. K bezdrátovému přenosu dat, lze využít různé technologie například Bluetooth, Wifi, celulární sítě nebo LPWAN, viz obrázek 3.3. Volba vhodného typu je závislá na různých faktorech jako jsou pokrytí, spotřeba elektrické energie, náklady nebo typ instalace. [5]



Obrázek 3.3 - Typy IoT sítí [5]

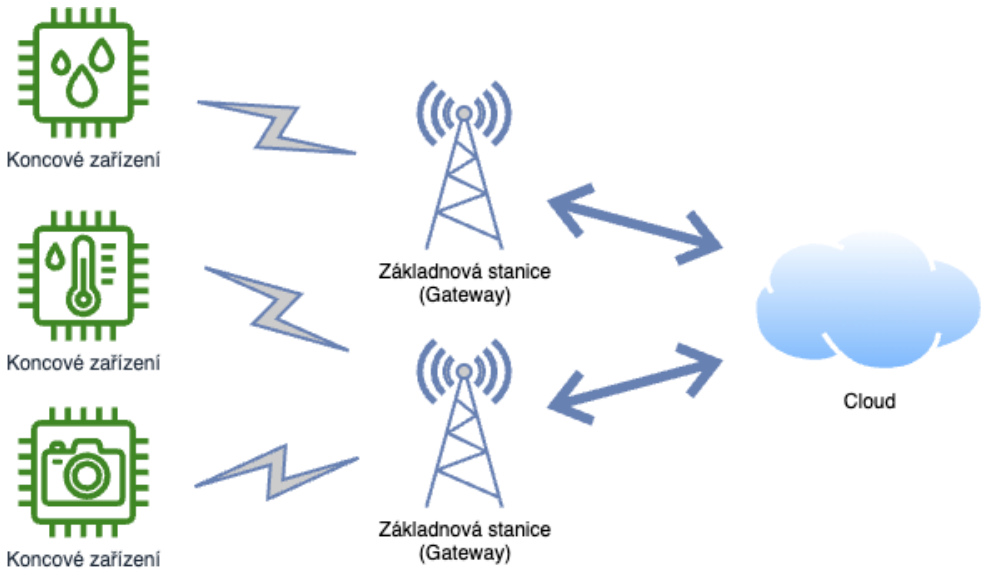
V licencovaném pásmu v celulárních sítích lze uvažovat o dvou variantách přenosu malých dat na velké vzdálenosti [5]:

- K přenosu dat je využita stejná infrastruktura, stejné zdroje a protokoly jako k poskytování hlasových a běžných datových služeb (LTE, 5G).
- K přenosu dat se využívají specializované protokoly nebo dedikované linky (NB-IoT).

V nelicencovaném pásmu se k přenosu malých dat na velké vzdálenosti využívají sítě typu LPWAN (Low Power Wide Area Network). NB-IoT, Sigfox nebo LoRaWAN jsou dominantní technologie a všechny tři umožňují obousměrnou komunikaci, tedy odesílání (uplink) nebo příjem (downlink) dat. LPWAN řešení se obvykle skládá z těchto prvků [5]:

- **Koncová zařízení**: terminátory, které mají autorizovaný přístup do sítě, sbírají data nebo automatizují činnosti (senzory, ovládací prvky, mobilní zařízení, autonomní vozidla, mikroprocesory atd.).
- **Základnová stanice**: hardware instalovaný operátorem, který pokrývá určitou oblast rádiovým signálem. Zodpovídá za modulaci nebo demodulaci zprávy, kterou následně buď pošle skrze IP spojení na servery do cloudu, nebo skrze rádiové vlny na koncová zařízení.

- **Cloud**: část, která poskytuje různé služby související s uložením nebo zpracováním nasnímaných dat, řízením síťového provozu nebo zpřístupňuje data prostřednictvím různých API.



Obrázek 3.4 - LPWAN architektura [5]

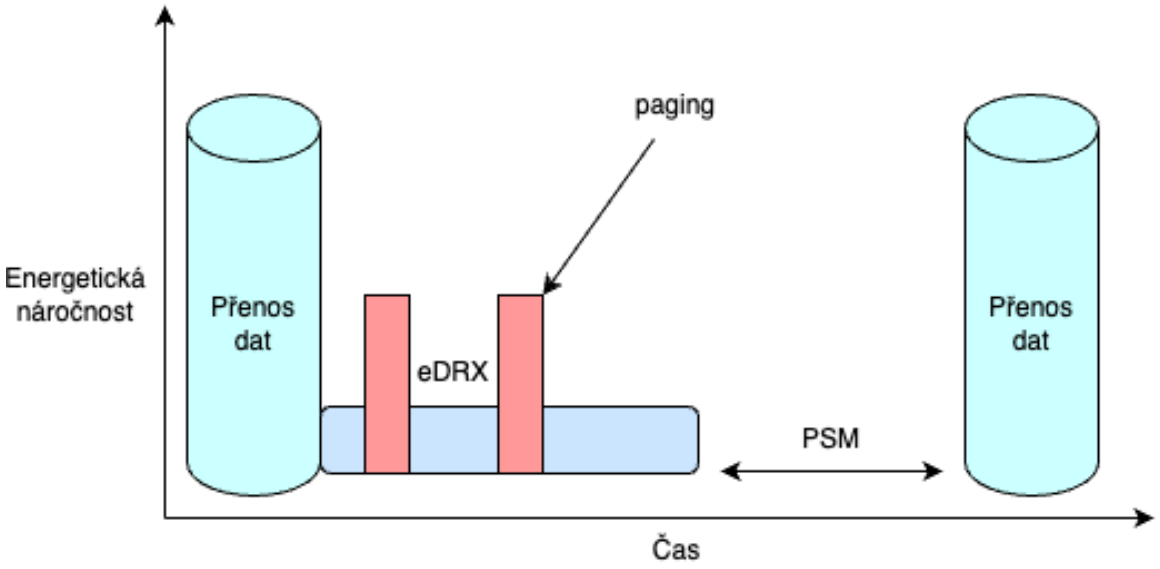
3.1.3.1 NB-IoT (Narrow Band IoT)

Je síť určená pouze pro přenos dat a podporuje IoT instalace typu mMTC a uRLLC. NB-IoT je typem LPWAN sítě, protože vyniká nízkou spotřebou a širokým pokrytím. Rychlost přenosu dat se pohybuje kolem 160–250 kbps. [6] Síť je provozována v licencovaném pásmu, což garantuje kvalitu signálu, protože nedochází k rušení. V roce 2017 se stal prvním provozovatelem sítě v ČR mobilní operátor Vodafone, poté následovaly O2 a T-Mobile. [7]V současné době je NB-IoT signál dostupný po celé ČR a operátoři uvádí, že garantují výborné pokrytí i uvnitř budov, dokonce i ve sklepních prostorech, čehož je dosáhnuto zesílením signálu zhruba o 20 dB. [6] NB-IoT je standardizovaná síť stejně jako GSM nebo LTE, z čehož plyne, že NB-IoT může využívat podobné služby jako LTE, jen při nižších rychlostech a nižší spotřebě energie na koncových zařízeních. Těmito službami může být například roaming, který tak otevírá možnost stavět a realizovat řešení s mezinárodním charakterem. Standard rozvíjí a spravuje organizace 3GPP. Připojení a autorizace zařízení v síti je zprostředkováno prostřednictvím SIM karty. [6]

Tři typy scénářů nasazení NB-IoT řešení [6]:

- Nezávislá implementace (stand-alone): využívá vlastní nezávislé pásmo, které se nepřekrývá s běžným LTE.
- In-Band implementace: sdílí pásmo společně s LTE.
- Guard-band implementace: využívá okrajové frekvence LTE.

Nízké spotřeby je dosaženo díky úzkému pásmu (180 kHz), jednoduchostí technologie (nižší nároky na hardware), PSM (power saving mode) nebo eDRX (extended discontinuous reception) funkcemi. PSM je funkce, která uvede zařízení do hlubokého spánku, kdy ze všech komponent zůstane aktivní pouze časovač. Po uplynutí prahové hodnoty, časovač probudí zařízení, proběhne přenos dat a po dokončení přenosu je zařízení uvedeno zpět do režimu spánku. Maximální hodnota PSM se počítá na stovky dní. eDRX stanovuje délku intervalu mezi jednotlivými synchronizacemi, kdy se terminál ohlašuje síti (paging). Zařízení si v klidovém režimu zachovává síťové spojení a předpokládá se, že pokud bychom posílali každý den zprávu o velikosti 200 B a použili obě výše zmíněné technologie, pak by životnost 5 Wh baterie byla kolem 12,8 let. [6]



Obrázek 3.5 - NB-IoT PSM a eDRX [6]

3.1.3.2 LoRaWAN (Long Range WAN)

Nízkoenergetická síť s dalekým dosahem (LPWAN) jejíž specifikace byla vydána v roce 2012. Pojem LoRaWAN definuje komunikační protokol a síťovou architekturu. [8] Je to otevřená technologie, která využívá bezlicenční pásmo do 1 GHz. V Evropě se používá frekvence 868 MHz, severní Amerika 915 MHz a Asie má přidělenu frekvenci 433 MHz. Použití těchto nízkých frekvencí ji dělá odolnou vůči rušení a umožňuje lepší průchod překážkami. Dosah sítě je v lehce zastavěném terénu okolo 15 km na jednu bránu a v husté zástavbě kolem 5 km. [9] Je to patentovaná technologie, která funguje na principu rozprostřeného spektra (CSS). [9]

Obvykle používanou topologií je hvězda. Zařízení v síti komunikují asynchronně a oboustranně. Data vyslaná koncovými zařízení jsou přenesena na základnové stanice po rádiových vlnách a pro přenos se používají úzká pásma. Šířka pásma je volena dynamicky na základě množství přenášených dat a podmínek bezdrátového spojení, pro každé zařízení individuálně a za volbu odpovídá síťový server. Jedná se o adaptivní nastavení rychlosti přenosu dat a maximální rychlost může být až 50 kbps. [8]

Zabezpečení je zajištěno pomocí asymetrické šifry AES dvěma 128bitovými klíči. Šifrování mezi koncovým zařízením a síťovým serverem je zabezpečeno 128bitovým klíčem Network Session Key (NwkSKey) a mezi síťovým serverem a aplikačním serverem je komunikace zabezpečena 128bitovým klíčem Application Session Key (AppsKey). Koncové zařízení má přidělen 64bitový identifikátor (DevEUI), který je pro každý zařízení jedinečný a na základě tohoto klíče je zařízení autorizováno v síti. [9]

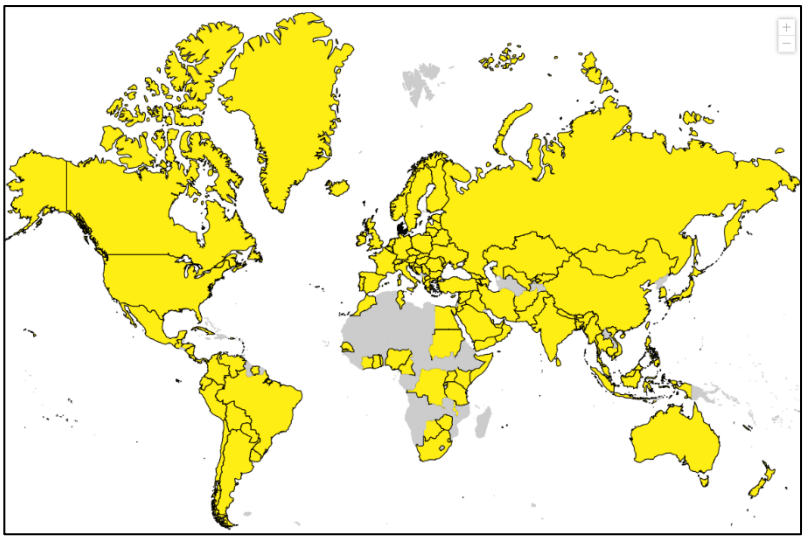
Třída určuje režim práce aktivního prvku v síti:

- **Třída A**: koncové zařízení podporuje obousměrnou komunikaci, každý uplink je následovaný dvěma okny pro příjem krátkých zpráv (downlink). Je to nejvíce úsporný režim z pohledu spotřeby energie, protože aktivní prvek smí přijmout zprávu jen tehdy, když odeslal zprávu.
- **Třída B**: downlink pro příjem dat se otevírá v předem definovaných časových intervalech. Je to přidání dalšího způsobu ke komunikaci třídy A. Během režimu spánku je aktivní časová synchronizace.

- **Třída C**: přijímací okna jsou otevřena téměř nepřetržitě a zavírají se pouze při vysílání. Nejrychlejší odezva za cenu nejvyšší možné spotřeby energie. [9]

Zařízení, které je v režimu spánku udává svou spotřeba v jednotkách mW, proto každý aktivní prvek musí disponovat aspoň třídou A. [9]

Jaký je aktuální stav pokrytí světa signálem LoRa je uvedeno na obrázku 3.6. Obrázek zahrnuje veřejné, komunitní nebo profesionální provozovatele sítí. V ČR disponují dobrým pokrytím například České radiokomunikace (ČRa),



Obrázek 3.6 - Pokrytí LoRa svět

k čemuž od roku 2021 nabízejí roaming a lze tak vytvářet mezinárodní instalace. [10] V případě LoRa se jedná o otevřenou platformu (open source) a lze tak k získání síťové konektivity využít alternativní vybudování vlastní sítě. [9]

3.1.3.3 Sigfox

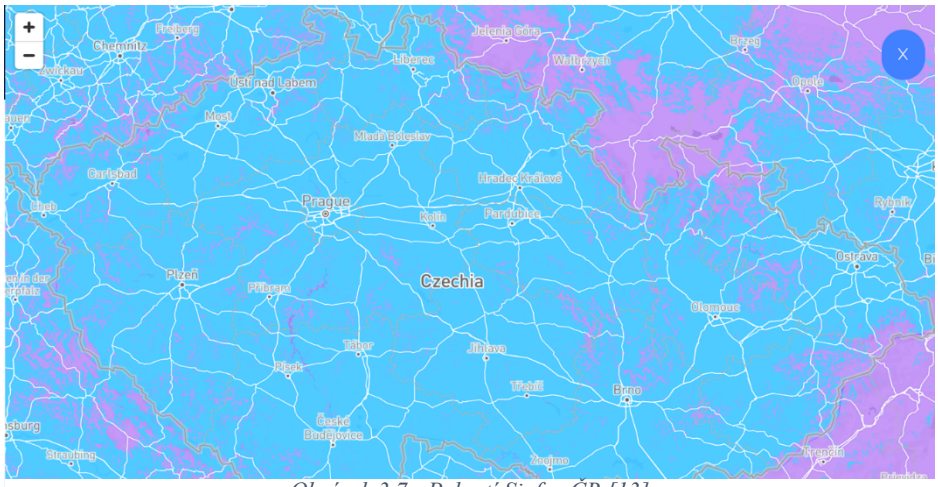
Je to první nízkoenergetická přenosová síť pro přenos malého množství dat na velké vzdálenosti s dosahem i do podzemních prostor, která se objevila v roce 2009. Řešení je vhodné pro IoT instalace typu mMTC. [5] Uvádí se přenosová rychlost 100 bps a dosah sítě v zabydlené oblasti okolo 3–10 km nebo v nezabydlené oblasti okolo 30–50 km. Ze začátku byl přenos dat pouze jedním směrem, konkrétně od sensoru do datového úložiště (uplink) a od roku 2015 je možné data přijímat i vysílat. Připojení do sítě neprobíhá přes SIM karty ani pomocí adresace a není potřeba zařízení párovat. Připojení a autorizace do sítě je možná na základě vnitřního identifikátoru BPSK (Binary Phase Shift keying), který je jedinečný pro

každé zařízení. Odpadá tím autentizace, vytváření spojení a synchronizace (paging). To jsou všechno aspekty energeticky úsporného režimu. [5]

Sigfox pracuje v bezlicenčním pásmu v rozsahu 862–928 MHz. [11] K přenosu dat využívá patentovanou technologii UNB (Ultra Narrow Band) pro vysílání krátkého pulsu. Každá přenášená zpráva v době přenosu zabírá šířku pásma 100 Hz. [11] Protokol sítě Sigfox je záměrně navržen pro přenos malých dat, protože čím menší jsou data, tím menší jsou energetické nároky. Úzké pásmo a malá velikost přenášené zprávy pozitivně ovlivňují i vzdálenost, na kterou lze přenos realizovat a schopnost demodulovat zprávy na úrovni hladiny šumu. Zpráva nemá předem definovanou strukturu. Definice, co se bude posílat a s jakou strukturou je na odesílateli nebo příjemci. Celkový objem přenášených dat v jedné zprávě může být 26 B, z čehož užitečné zatížení (payload) může mít maximálně 12 B pro odeslání (uplink) a 8 B pro zpětné zprávy (downlink). Během 24 hodin může zařízení odeslat maximálně 140 zpráv a 4 zprávy přijmout. Toto omezení předurčuje síť k nasazením typu zapnout/vypnout, otevírání dveří, měření teploty, stav baterie, identifikace pohybu nebo posílání GPS koordinátů. [11]

Zařízení poté co odešle data si samo vybírá, kdy chce poslouchat a přijmout potvrzení nebo reakci na odeslaná data. Výhodou této vlastnosti je velmi nízká energetická náročnost, protože během vysílání se spotřeba energie pohybuje okolo několika desítek mA a v režimu spánku, ve kterém se zařízení nachází většinu času, má spotřebu pouze několik nA. Tento režim připomíná třídu A u technologie LoRa. [12]

Modrá barva na obrázku 3.7 znázorňuje pokrytí signálem Sigfox v rámci ČR. [13]



Obrázek 3.7 - Pokrytí Sigfox ČR [13]

3.2 Cloud computing

Stejně jako u Internetu věcí i zde v Cloud computingu neexistuje jedna definice, která by vymezila pojem cloud. Definic existuje mnoho a kombinují technické nebo obchodní aspekty. Důležitým obchodním aspektem je, že IT je možné dodávat jako utilitu. Rozumějte, že koncový uživatel si dávkuje IT podle potřeby. Analogicky lze toto chování přirovnat ke kohoutku od vodovodu, kdy si vlastní vůlí napustím jen tolik kolik potřebuji a za to platím. Zároveň nejsem vlastníkem zdrojů vody, úpraven nebo vodovodního řádu. To je druhý důležitý obchodní aspekt. Nemusím investovat do vlastního HW nebo SW včetně obsluhy. Mohu tak mít k dispozici velice drahé a pokročilé nástroje pro podporu mého podnikání. Ze všech definic byla vybrána níže uvedená, která dobře ilustruje přednosti cloud computing řešení. Definici vydal National Institute of Standards and Technology (NIST) [14]:

„Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“ [14]

Z technického pohledu je cloud computing spojením trendů v různých oblastech ICT. Za stěžejní lze povatovat vývoj v oblasti výpočetních modelů (utility computing, grid computing, …), virtualizace nebo kontejnerizace (sdílení zdrojů nejen hardwarových) a v počítačových sítích. [14] Síť je pro cloud deterministická, protože cílem je doručovat požadovaný výpočetní výkon, datový prostor nebo aplikace kamkoliv a kdykoliv. K tomu je třeba globální počítačové sítě s dostatečným výkonem a propustností dat. Propustnost dat je klíčový parametr jak pro provozovatele, tak i pro odběratele cloudové služby. [14]

Charakteristika [14]:

- multitenantnost – oddělené sdílení zdrojů mezi uživateli,
- škálovatelnost – uživatel může dynamicky měnit alokaci zdrojů,
- samoobsluha s platbou jen za spotřebované zdroje (pay as you go),
- aktuálnost – poskytovatel garantuje aktuálnost nabízeného řešení,
- monitoring – sledování zdrojů a na základě výstupů dochází k optimalizaci,
- přístup přes heterogenní síť.

Cloudová služba může být realizována jako veřejná služba pro všechny, kteří splní podmínky jejího užívání, nebo privátní služba, což je řešení určené pro danou entitu, veřejnosti je uzavřené a vlastník infrastruktury je zároveň uživatelem. Třetí možností je kombinace obou přístupů v podobě hybridního cloud řešení.

Cloud jsou také služby, které klasifikujeme podle podstaty nabízené technologie. Máme tak různé typy nabízených cloudových služeb jako Software as a Service, Platform as a Service, Infrastructure as a Service, Monitoring as a Service, Desktop as a Service, Security as a Service a řadu dalších. [14] Tři nejvýraznější zástupci, kteří determinují cloud computing popisuji podrobněji níže.

Software jako služba (SaaS) spočívá ve sdílení aplikace. Některé aplikace mohou být příliš drahé, proto je vhodné jejich distribuci širší skupině uživatelů umožnit prostřednictvím jejího sdílení (Google Apps, Office 365, Force.com). Aplikace musí být přístupná přes internet a jsou oddělená data a zdroje pro jednotlivé uživatele. U tohoto řešení kompletně odpadá péče o aplikaci, aplikaci pouze používám. V této variantě převažují aspekty obchodní nad technickými a služba je poskytována na vyžádání za cenu předplatného. Výhodou je, že provozovatel kompletně pečuje o aplikaci, včetně ochrany dat před jejich ztrátou nebo zničením a průběžně aplikaci aktualizuje. Provozovatelem jsou obvykle velké nadnárodní společnosti s velkými datovými centry. [14]

Infrastruktura jako služba (IaaS) je virtualizace hardware bez softwaru. Je to nejzákladnější varianta cloud computing modelu. Pronajímáme hardwarové zdroje (CPU, diskový prostor, paměť) včetně technického zabezpečení v podobě klimatizace, napájení nebo síťové konektivity. Aby se nejednalo jen o virtualizaci, je důležité, aby bylo možné virtualizovanou infrastrukturu škálovat, rozumějte, na základě požadavků koncového uživatele dynamicky podle potřeby měnit její parametry rozsahu a výkonu. Virtuální stroj může běžet na více fyzických strojích, nebo více virtuálních strojů může běžet na jednom fyzickém. Obvykle je služba zpoplatněná po hodinách. Oblíbenou je například Amazon Elastic Compute Cloud. [14]

Platforma jako služba (PaaS) je vyšší úroveň virtualizace oproti IaaS. Výsledkem je platforma, kde k virtualizovanému železu byl přidán i potřebný software, jehož rozsah je určen požadavkem na službu. Požadavek specifikuje uživatel a obvyklým požadavkem je platforma pro vývoj, testování nebo doručování aplikací (webové služby, databázové služby, úložiště, verzování, …). Oblíbenými jsou služby Google App Engine nebo Azure App Service. [14]

Z výše uvedeného plyne, že smyslem cloudu je sdílet výpočetní výkon a zpřístupnit ho kdykoliv a kdekoliv. Z čehož plyne silná závislost na síťové konektivitě, protože bez sítě není dostupná služba. Kvalitu služby vyjadřujeme její dostupností. Dalším omezením tohoto výpočetního modelu je umístění dat. Ve veřejném cloudu jsou data umístěna vždy mimo naši infrastrukturu, v privátním je to závislé na zvoleném řešení. Data mohou být umístěna mimo právní rámec dané země, ve které podnikáme a z toho plyne riziko, které je potřeba zohlednit ve smluvních vztazích a smluvně definovat kvalitu dodávané služby (SLA). [15]

Níže je popsáno několik příkladů cloudových služeb od významných technologických firem se zaměřením na služby typu PaaS nebo IaaS. Cílem je seznámení s principy cloudu na reálných službách.

3.2.1 Google Cloud Platform (GCP)

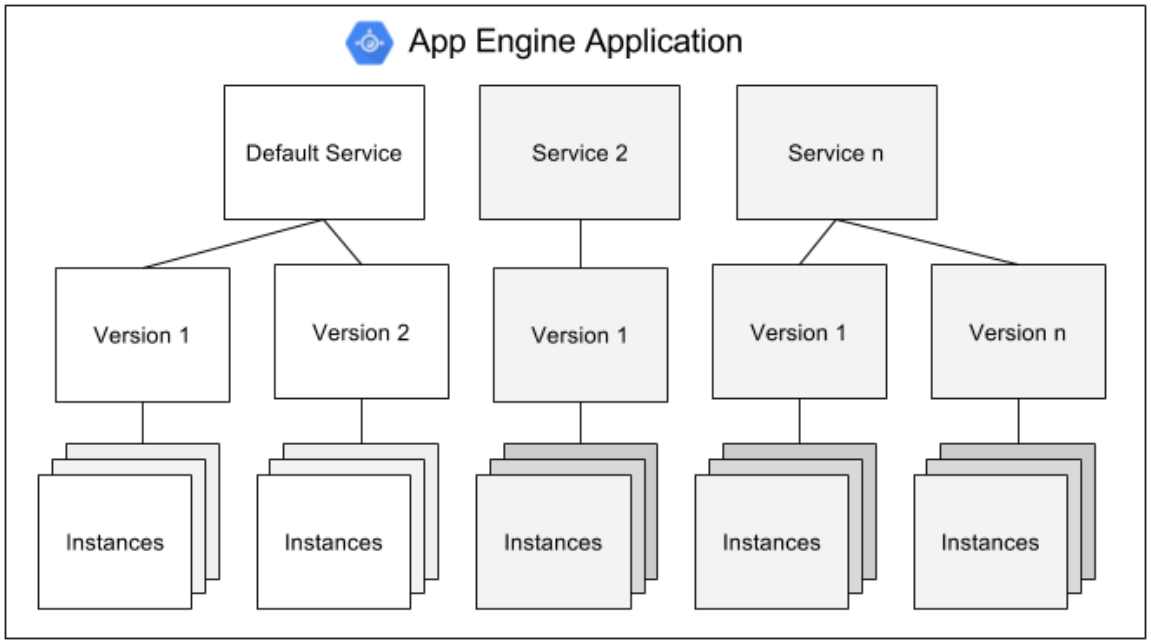
Nabízí cloudová řešení typu IaaS nebo PaaS na vysoce škálovatelné a spolehlivé infrastruktuře, protože využívá stejné zázemí a vybavení jako SaaS řešení G Suite (Gmail, Google Drive, …). GCP nabízí desítky různých produktů z různých oblastí zahrnujících výpočetní výkon v podobě virtuálních počítačů, ukládání dat, NoSQL databáze, analytické nástroje pro big data až k IoT nebo umělé inteligenci. Google zajišťuje dobrou integraci napříč produkty GCP a některé produkty disponují vlastností automatického škálování výkonu na základě zátěže, která roste společně s rostoucím počtem uživatelů nebo množstvím přenesené užitečné informace (payload). Tuto funkci nabízí například Cloud Datastore nebo App Engine. [14]

3.2.1.1 App Engine

Je to PaaS služba, která poskytuje hostingové služby pro webové aplikace nebo statický web. Aplikace mohou být napsané v programovacích jazycích Python, Java, PHP, Node.js, .Net, Ruby nebo Go a jsou přístupné koncovému uživateli buď přes vlastní doménu, nebo přes doménu služby appspot.com. Google App Engine (GAE) vyžaduje použít k ukládání dat Google Bigtable a používat dotazovací jazyk Googlu. [16]

Na výběr je ze dvou typů prostředí. Typ standard je prostředí podporující jazyky Python, Go nebo node.js. Nabízí perzistentní úložiště, load balancing, integraci s dalšími službami Google Cloud Platform a API. Flexibilní typ nabízí větší míru svobody, protože vývojář může používat Docker prostředí a může tak instalovat vlastní software ve vlastním kontejneru. [16]

App Engine je součástí Google Cloud Platform projektu a v rámci projektu dojde k přidělení zdrojů. Z čehož plyne, že je možné pouze horizontální škálování, protože z pohledu projektu se jedná o aplikaci typu monolit. [16]



Obrázek 3.8 - Struktura App Engine [17]

Aplikace (Application) je hierarchicky nejvýše postavená jednotka, kterou představuje kontejner, který má přidělené zdroje.

Služby (Services) reprezentují nějakou logickou část aplikace. Služby mezi sebou vzájemně komunikují. Můžeme nasadit aplikace, které jsou tvořeny pouze jednou službou (monolit), nebo skládající se z mnoha služeb (microservice-set). [16]

Verze (Versions) udržujeme pro každou službu vybrané verze, což umožňuje rychlé přepínání mezi verzemi. [16]

Instance (Instances) jsou závislé na vytíženosti spuštěné služby. Každá verze služby provozuje jednu nebo více svých instancí. Počet je odvislý od potřeb aplikace, kdy se vzrůstajícím počtem uživatelů, roste provoz a s tím velikost přenášených dat (payload). Ze zvýšených požadavků plyne rostoucí počet instancí. App Engine disponuje vlastností automatického škálování výkonu. [16]

3.2.2 Amazon Web Services (AWS)

Amazon provozuje kromě e-shopu datová centra po celém světě a koncovým uživatelům nabízí ICT zdroje, výpočetní výkon nebo datový prostor v podobě cloudových služeb. Provozuje mnoho datových center po celém světě, v různých lokalitách. Jelikož se jedná o jednoho z technologických gigantů, který disponuje rozsáhlou sítí datových center po celém světě, jsou jeho datová centra kategorizována následovně. Region je fyzická lokalita a každý region má několik zón dostupnosti (Availability Zones), což jsou vlastně datová centra se záložním přívodem elektrické energie a síťové konektivity. Platí pravidlo minimálně tří zón dostupnosti na region, což je zapříčiněno ochranou proti výpadku. Například v Evropě je celkem osm regionů a každý region je vždy tvořen aspoň třemi zónami dostupnosti, které jsou na sobě nezávislé, fyzicky oddělené. [18]

Cloudové služby lze ovládat třemi způsoby [18]:

- Management Console, což je webové rozhraní pro práci s AWS.
- REST nebo SOAP API, pro které existuje mnoho knihoven podporujících programovací jazyky Python, PHP, Java, Ruby, .NET.
- CLI ovládání prostřednictvím terminálu.

3.2.2.1 Elastic Compute Cloud (EC2)

Služba typu IaaS, která uživateli umožňuje provozovat virtuální stroje. Využívá celosvětově rozsáhlé infrastruktury, kterou společnost Amazon záměrně postavila pro potřeby svého eCommerce řešení. EC2 je považována za vysoce kvalitní službu, která se svými parametry nejvíce přibližuje představě sdílení výpočetního výkonu prostřednictvím cloudu. Uživatel definuje parametry virtuálního stroje a vybere typ operačního systému z mnoha předpřipravených AMI (Amazon Machine Images). Některé AMI obsahují holý operační systém, jiné obsahují databázové nebo webové aplikace či dokonce kompletně připravené e-shop řešení. [14]

Instance lze provozovat ve více datových centrech tedy více lokalitách, kdy lze na základě lokality vybrat uživateli nejbližší data. Replikace mezi datovými centry zajišťuje služba Content Delivery Network (CDN). [14]

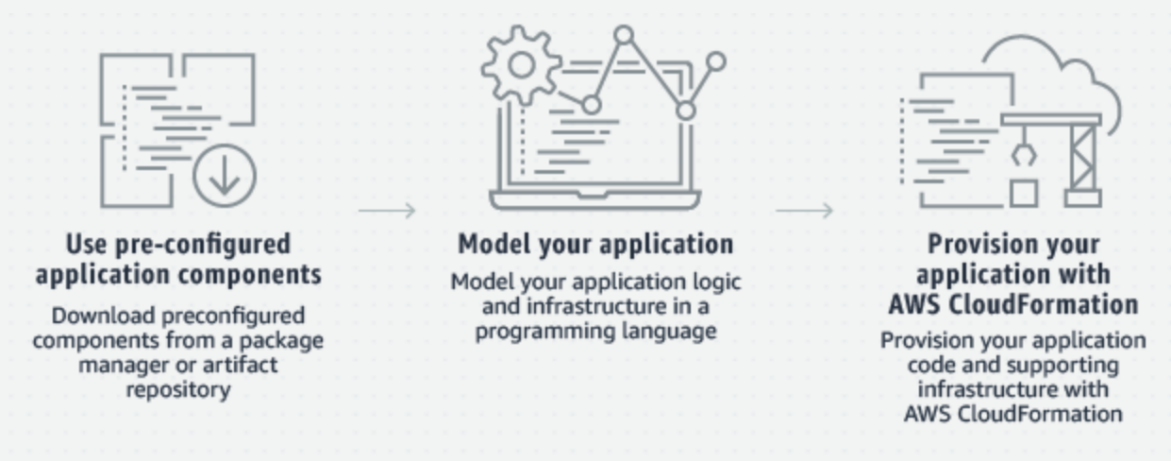
Existují další AWS služby, které vhodně doplňují službu EC2. Služba pro ukládání dat Amazon Simple Storage Service (S3), služba pro rozkládání zátěže a automatické škálování virtuálních strojů Elastic load balancing (ELB), pro sledování stavu a spotřeby cloudových zdrojů Cloud Watch nebo databázová řešení podporující tradiční relační systémy jako Oracle, MSSQL, MariaDB, Postgres nebo NoSQL varianty. [18]

3.2.2.2 Infrastructure as code (IaC)

Paradigma, které umožňuje vytvořit vlastní virtuální infrastrukturu ze sdílených zdrojů deklarativně prostřednictvím programovacího jazyka. Tento přístup má vazbu na cloud computing, protože právě ke cloudovým prostředkům nebo službám se typicky přistupuje prostřednictvím API, oblast vlastní programátorům. Přínosem je, že vlastníkem virtuální infrastruktury jsou vývojáři, infrastruktura je jednoduše reprodukovatelná a transparentní, protože k uložení konfigurací a definicí infrastruktury je využíváno verzovacích systémů (Git). [18]

Framework Cloud Development Kit (CDK) definuje infrastrukturu deklarativní programovací cestou, která je následně nasazena prostřednictvím služby CloudFormation. CDK podporuje programovací jazyky jako TypeScript, JavaScript, Python, Java, .Net a Go.

Taky jsou k dispozici sady předpřipravených struktur nebo komponent, které usnadňují konfiguraci a vytvoření prostředí. [18]



Obrázek 3.9 - AWS CDK [18]

Služba CloudFormation zodpovídá za nasazení komponent definovaných rozhraním CDK, k čemuž využívá šablon ve formátu JSON nebo YAML. Prostředí se spravuje přes CloudFormation HTTP API. Za běhu je možné provádět aktualizace komponent a v případě problému se automaticky provede roll-back (CI/CD pipeline). [18]

3.2.3 Microsoft Azure

Rozsáhlá cloudová platforma od firmy Microsoft nabízející mnoho ICT služeb a výpočetní výkon z vlastních datových center propojených rozsáhlou globální WAN sítí Microsoftu. Na zhruba 200 datových center je rozmístěno po celém světě. [19] Datová centra jsou fyzické budovy se servery a síťovými prvky, které jsou sdružovány do zón dostupnosti. Stejně jako AWS zóna dostupnosti reprezentuje datové centrum s nezávislým napájením, chlazením a komunikační sítí. Microsoft garantuje, že data neopustí zvolený region, například data pořízená v ČR zůstanou na území Evropské unie, čímž se na tato data bude vztahovat legislativa EU. Z výše uvedeného plyne, že globální infrastruktura je rozdělena do regionů a každý region obsahuje aspoň tři zóny dostupnosti pro zajištění vysoké dostupnosti nabízených služeb. V Azure je k dispozici mnoho služeb, jejichž autorem není jen společnost Microsoft. [19]

Výpočetní výkon lze v Azure sdílet několika různými způsoby:

- virtuální stroj (IaaS),
- kontejner,
- Azure App Service (PaaS),
- Service Fabric a další varianty.

Azure Virtual Machine je IaaS služba, kde je možné vytvářet virtuální stroje s operačním systémem Windows nebo Linux. Dalším typem IaaS služby jsou kontejnery, což je typ virtualizace, kde aplikace vyžaduje ke svému běhu pouze vlastní závislosti v podobě knihoven nebo běhových prostředí (runtime), a není nutné instalovat operační systém pro danou aplikaci. Kontejnery jsou vhodnou volbou pro nasazení a provoz cloudových aplikací, protože odpadá starost o operační systém včetně zdrojů, které ke svému běhu operační systém potřebuje. Kontejnery lze provozovat prostřednictvím služby Azure Container Apps, nebo si pronajmout plnohodnotný orchestrátor kontejnerů Kubernetes cluster spravovaný Microsoftem v rámci služby Azure Kubernetes Service (AKS). U AKS platíme pouze za běžící virtuální servery zařazené do clusteru. U Container Apps oproti AKS neřešíme složitosti Kubernetu, ale využíváme jeho funkcí, čímž získáváme serverless prostředí, které umí škálovat od nuly (vypne se v případě nulového provozu). [20]

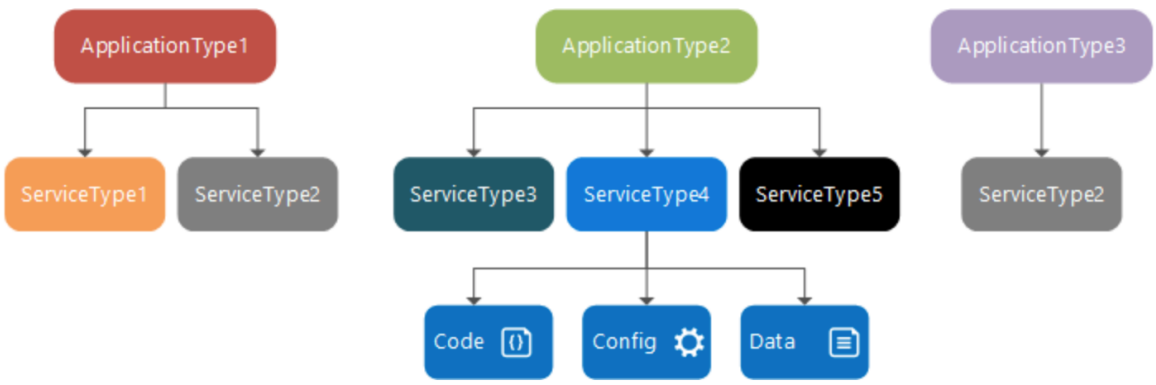
Azure App Service je PaaS implementace jejímž prostřednictvím lze snadno nasadit a provozovat webovou aplikaci, mobilní aplikaci, kontejnery nebo aplikaci, která zpřístupňuje data prostřednictvím REST API. Podporuje programovací jazyky a frameworky .NET, PHP, Python, Node.js nebo Java. Ve výchozím nastavení jsou aplikace přístupné přes url končící na *.azurewebsite.net a je možné toto url změnit na vlastní doménu. Je to služba příbuzná Google App Engine, Microsoft pro nás spravuje prostředí a my provozujeme aplikaci. Z toho plyne, že získáme vysokou dostupnost a nemusíme řešit konfigurace a monitoring clusteru, znalosti ohledně zón dostupnosti, availability setu nebo aktualizace operačních systémů a jeho nástrojů. App Service není ideální pro provozování aplikací postavených na architektuře mikroslužeb, protože není možné nezávisle škálovat jednotlivé služby nebo mít jednoduchou vnitřní komunikaci mezi službami, či určit služby dostupné zvenku a služby provozované jen interně. Pro mikroslužby jsou vhodné jiné varianty jako například řešení Azure Service Fabric. [20]

3.2.3.1 Service Fabric

Service Fabric je řešení pro vývoj a provoz distribuovaných aplikací. K běhu aplikací využívá samotných procesů nebo kontejnerů. Provozované služby lze škálovat horizontálně a vertikálně. Tedy zvyšovat počet instancí nebo zvyšovat hardwarové zdroje jednotlivým službám. Je to řešení pro aplikaci komplexních systémů postavených na architektuře mikroslužeb. Silnou vlastností, která toto řešení odlišuje od ostatních platforem je práce s persistentními daty. Data jsou součástí clusteru, konkrétní služby a pro každou službu je její lokální databáze unikátní a nezávislá. Data jsou ukládána na lokální disk, není třeba sestavovat síťové spojení a kontaktovat databázový stroj. Výhodou toho je vysoká rychlost pro práci s daty. Autorem frameworku je Microsoft, který ho sám používá pro provoz svých klíčových cloudových služeb, například Cosmos DB, Azure SQL, Cortana, Azure IoT Hub, Dynamics 365 a mnoho dalších. [21]

K vývoji a provozu aplikací lze využít jak cloudového řešení přímo z Azure portálu, tak i lokální instalace (on-premis). K zprovoznění vlastní lokální instance clusteru je vyžadován operační systém Windows nebo Linux. Dále pak Service Fabric runtime a SDK. [21]

Platí, že co instance služby to proces nebo kontejner. Architektura služby spočívá v uspořádání služeb do setu zvaného ApplicationType. To je množina všech procesů nebo kontejnerů, které definují aplikaci, kterou lze nasadit. Služba (ServiceType) odkazuje na data, konfigurace nebo samotný programový kód. [21]



Obrázek 3.10 - Service Fabric struktura aplikace [21]

Každou službu lze nasadit několika různými způsoby. Buď je to služba uchovávající stavy, která persistentně uchovává data a označuje se statefull, nebo je to služba, která stavy neuchovává a pak se označuje pojmem stateless. Tyto dva typy služeb jsou vlastní programovacímu modelu Reliable Service. Existují ještě programovací modely kontejner a Actor Model. Každý programovací model má své výhody a své doporučené scénáře užití. [21]

Služba typu Reliable je typ služby, který buď s žádným nebo minimálním API umí efektivně zprostředkovat komunikaci mezi službami, které tvoří aplikaci. Pro sestavení takové komunikace je potřeba definovat posluchače a na druhé strany klienta zabudované reversní proxy. Významnou výhodou je ovšem z pohledu programátora snadná práce s databází, protože k uložení dat na lokální disk využívá Reliable kolekce, které jsou podobné dobře známým kolekcím z rodiny .NET jako je list nebo slovník. Z toho plyne, že s daty se pracuje jako s objekty. Ochrana proti ztrátě dat je zajištěna replikami a jejich počet určuje konfigurace clusteru. [21]

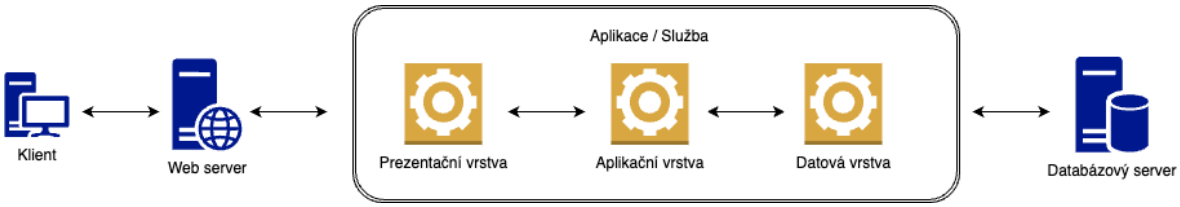
Druhým významným programovacím modelem je Actor model. Je to implementace návrhového vzoru výpočetního modelu Actor. Actor je plně nezávislá výpočetní entita s vlastním vláknem a výhodou tohoto přístupu je paralelní zpracování dat a asynchronní výměna zpráv mezi službami. Je to přístup podobný systémům zpracování zpráv jako například Kafka. [21]

3.3 Architektura mikroslužeb

Aplikace, které jsou vytvářeny agilně mají rychlejší nástup v počtu vytvořených funckionalit, ale časem se vývoj dostane do bodu, kdy systém bude složitější měnit, rozšiřovat nebo aktualizovat. Schopnost rozšiřování systému a tím i délka životnosti aplikace je úměrně závislá na kvalitě softwarového návrhu čili architektuře. [22]

Jedním z dlouhodobě používaných návrhů je Třívrstvá architektura aplikace, kde máme vrstvu prezentační, datovou a vrstvu aplikační logiky. Je to typická architektura pro aplikace typu monolit, kdy vývoj začíná návrhem datové struktury, tedy tvorbou databáze. Charakteristickými rysy monolitické aplikace jsou [22]:

- Programový kód je obvykle uložen v jednom úložišti.
- Vybrané technologie jsou společné pro všechny části aplikace.
- Aplikace je členěna do vrstev.
- Obvykle jedno datové úložiště (databáze) pro celou aplikaci.
- Pouze horizontální škálování.
- Moduly, ze kterých se aplikace skládá nejsou nezávislé, změna v jednom modulu má dopad na celou aplikaci.
- Aplikace se typicky nasazuje a testuje jako celek.
- Aplikace je závislá na mnoha knihovnách, které mají tranzitivní závislosti.



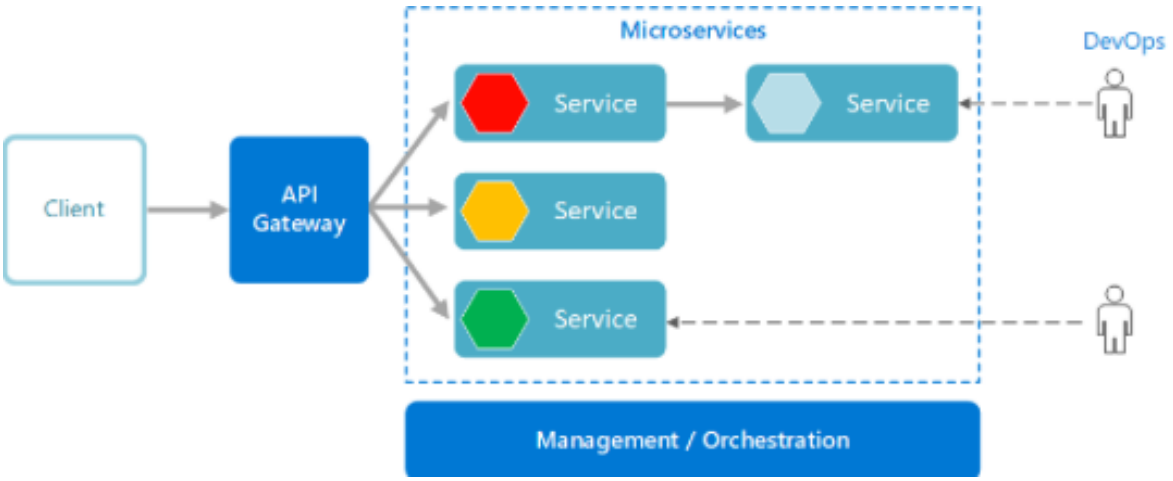
Obrázek 3.11 - Monolit schéma [22]

Škálování monolitických aplikací je složité, protože využívají k ukládání dat jednu databázi a veškerá logika, ať už aplikační nebo datová se odehrává v jednom kontejneru. Škálovat lze pouze horizontálně, kdy je možné zvyšovat výkon aplikace pouze jako celku, přidáváním dalších kopií aplikace (instancí). Za rozdělování zátěže mezi jednotlivé kopie zodpovídá load balancer. Problémem tohoto škálování je omezení v podobě jedné databáze. [21]

Koncept mikroslužeb je návrhový vzor, který aplikaci roztrhá na mnoho menších částí, kde základním stavebním prvkem je služba. S pomocí tohoto vzoru lze budovat odolná, vysoce škálovatelná řešení, kde jednotlivé služby jsou nezávislé a jejich vývoj a aktualizace je tak rychlejší a flexibilnější. Nezávislost spočívá v izolaci běhu programu v čase a prostoru a v datové nezávislosti čili každá služba disponuje vlastní databází. Takové služby lze opakovaně použít. Služby mezi sebou komunikují prostřednictvím aplikačního rozhraní (API) nebo pomocí asynchronních zpráv a každá služba má přesně určenou aplikační doménu (obvykle právě jednu). Takováto myšlenka malých nezávislých programů (služeb), kdy každý dělá dobře právě jednu činnost, je myšlenka realizovaná už v dobách začátků operačního systému Unix, tedy stará desítky let. Tři hlavní výhody konceptu malých služeb (programů) jsou [22]:

- Škálování, které lze přesně zaměřit na problematickou službu.
- Služby jsou nezávislé, tedy havárie nebo aktualizace jedné neovlivní chod ostatních.
- Lze je vyvíjet nezávislé na sobě, tedy v různých programovacích jazycích a v různých týmech. Tým má zodpovědnost za danou službu po celý její životní cyklus.

V souvislosti s mikroslužbami se často objevuje open source, protože nezávislost služeb umožňuje použít optimální technologie pro dosažení daného cíle. Generické části kódu jsou distribuovány a sdíleny prostřednictvím knihoven. Stejně jako knihovny, tak i služby je možné opakovaně použít v různých implementacích a zde je nutné vysvětlit rozdíl v těchto pojmech. Knihovna je funkční závislost programu, kterou program potřebuje ke svému běhu. Je nalinkována přímo do programu, a v rámci programu jsou používaný funkce z vybrané knihovny. Služby tvoří aplikaci a s ostatními komponentami komunikují buď prostřednictvím zasílání zpráv nebo aplikačním programovým rozhraním (API) způsobem požadavek odpověď (request/response). Služby jsou nezávislé a aby mohli spolu vzájemně komunikovat potřebují o sobě vědět, znát adresy dalších služeb. Toto není vhodné řešit statickou konfigurací, proto existují řešení pro automatizaci správy služeb, automatické škálování ať už horizontální nebo vertikální a identifikaci chyb. Typicky používaným řešením jsou kontejnery, které jsou orchestrovány například řešením Kubernetes. [23]



Obrázek 3.12 - Koncept Mikroslužeb [23]

Aplikace postavená na konceptu mikroslužeb je vlastně distribuovaný systém, jehož základní vlastností je vysoká citlivost na kvalitu komunikační sítě. Je to kritický prvek návrhu a distribuované systémy jsou citlivé na dobu odezvy, spolehlivost sítě, její zabezpečení a vlastní správu. Toto bylo dlouho dobu brzdou konceptu mikroslužeb, ale rozvíjející se model cloud computing umožnil získat výpočetní výkon ze špičkových datových center, která nabízejí kvalitní technologii, profesionální správu a snadné škálování výkonu pro námi vytvořené aplikace s vysokou dostupností služeb datového centra a službami maximalizujícími dostupnost našich aplikací (geografický clustering, atd.). [22]

3.3.1 Principy návrhu mikroslužeb

Každá služba má svou oblast zájmu (doménu) za kterou zodpovídá. Princip se nazývá **Single Responsibility** a je součástí návrhového vzoru SOLID, který je vlastní objektovému paradigmatu. Zkratka SOLID je akronym počátečních písmen pěti prvků, na kterých stojí tento návrhový vzor a jsou to [23]:

- Single Responsibility: jedna zodpovědnost.
- Open-Closed Principle: otevřené pro rozšiřování, uzavřené pro změny.
- Liskov Substitution: princip dědičnosti, podtřídy jsou zaměnitelné s jejich nadtřídami.
- Interface Segregation: preferovat specifická rozhraní, před univerzálními.
- Dependency Inversion: konkrétnější musí záviset na abstraktnějším, ne naopak.

Míra dekompozice je o citlivosti a zkušenosti architekta nebo týmu, protože je potřeba zohlednit riziko vysoké komunikační zátěže mezi službami v případě komplexních úloh. [22]

Všechny služby jsou na sobě vzájemně **nezávislé**, jsou samostatné. Vše, co služba potřebuje ke svému běhu je její součástí. Nezávislost podporují technologie jako je kontejnerizace, kdy celá služba je zabalena do kontejneru včetně všech svých závislostí. Kontejner je přenositelný, lze ho samostatně spouštět a pomocí orchestrátoru kontejnery spravovat a aplikaci škálovat. Samostatnost je potřeba udržovat i z pohledu návrhu datového modelu, abychom předešli vzniku sdílené databáze, která by vytvořila silnou závislost mezi službami. S výskytem mnoha samostatných databází se objevuje problém na zajištění správné konzistence dat. Existuje řada řešení například návrhový vzor Saga, který spravuje konzistenci dat napříč mikroslužbami v případě distribuovaných transakcí. [23]

Komunikační rozhraní pro výměnu zpráv mezi službami musí být co nejvíce jednoduché, ideálně nesmí obsahovat žádnou logiku. Martin Fowler tento princip nazval „Smart Endpoints and Dumb Pipes“ [24], kdy komunikační rozhraní je určené pouze pro přenos dat mezi službami bez jakékoliv transformace. Obdobným způsobem pracuje roura (Pipe) v Unixu, která jen přenáší data mezi programy operačního systému.

Aplikační rozhraní (API) je potřeba verzovat a po určitou dobu souběžně podporovat s aktuální verzí i verze starší. Verzování je důležité pro služby, které nezapracovali změny vyplývající z námi změněného API. [24]

3.3.2 Reaktivní manifest

Cílem manifestu je specifikovat základní požadavky na systém, aby ho bylo možné považovat za reaktivní. Reaktivní systémy jsou odolné, lze je škálovat dle potřeb a vždy poskytují uživateli odpověď, a to i v případě chyby systému. Reaktivní manifest definuje tyto čtyři základní principy [25]:

- Reaktivní systém je responsivní. Rychlá odezva zvyšuje důvěru uživatelů ve službu a také je z technického hlediska snazší pracovat se systémy, které reagují okamžitě, kde nemusíme zvažovat délku zpoždění.

- Odolnost systému vůči výpadkům a haváriím. Smyslem této vlastnosti je, aby systém odpověděl vždy za každé situace, a to i v případě, že uživatel nedostane očekávanou odpověď.
- Aplikace musí pružně reagovat na měnící se potřeby výkonu. Cílem této vlastnosti je zacílit potřebný výkon tam, kde je třeba. Povinným požadavkem k zajištění pružného systému je kromě automatického škálování potřeba monitoringu služeb, kterým sledujeme míru zátěže.
- Systém musí být zprávami řízený. Tato vlastnost vede k asynchronnímu zpracování zpráv. Obvykle řešeno další komponentou, která zajišťuje zpracování fronty zpráv mezi službami. Tato vlastnost zajišťuje volnou provázanost mezi službami, jejich nezávislost.

3.4 Domain Driven Design (DDD)

Domain Driven Design (DDD) je přístup k návrhu software, který byl definován už v roce 2003 v knize od Erica Evanse [26], ale na významu téma získalo až o mnoho let později s nástupem cloud computingu a architektury mikroslužeb. Doména je slovo označující oblast působnosti, obor nebo oblast zájmu a každá taková oblast má své vlastní termíny, chování, lidi a jejich role nebo procesy. DDD je soubor principů a postupů jehož cílem je vyvíjet spolehlivé a snadno udržovatelné softwarové aplikace a podporovat komunikaci mezi vývojovým týmem a doménovými experty. DDD je o porozumění všech zúčastněných stran, identifikaci funkčních požadavků a modelování. Tento přístup je obzvlášť vhodný pro komplexní projekty nebo pro projekty s mnoha týmy. [27]

Neznalost domény vede k výskytu jevu, který se nazývá Big Ball of Mud (BBM), což je DDD anti návrhový vzor (antipattern). BBM je označení pro projekty, které jsou z velké části nestrukturované, mezi jednotlivými částmi existuje mnoho skrytých závislostí, je zde velký výskyt duplicit jak v datech, tak i v kódu. Výsledkem je džungle zvaná špagety kód. Je to realita často způsobená finančním nebo časovým tlakem, častými změnami v týmu a v zadání projektu. [27]

Tradiční přístup k vývoji software je, že aplikační logika vzejde z dat (data-centric) a má zhruba následující průběh [23]:

- Identifikujeme uživatelské požadavky na systém nebo aplikaci.
- Sestavíme datový model.
- Z datového modelu a funkčních požadavků se identifikuje aplikační logika, relevantní úlohy k zpracování dat.
- Vytvoříme uživatelské rozhraní.
- Pokud akceptační testy neprojdou, bude zjištěn nesoulad, procházíme celým kolečkem znovu od začátku.

Vývoj software podle DDD je o aplikační logice a rozdíl oproti modelům vycházejících z dat je, že se hodně zaměřuje na fázi analýzy a návrh software s přístupem porozumět logice a business procesům dané domény. Tyto klíčové prvky DDD jsou reprezentovány doménovým modelem. DDD není spojen s žádnou konkrétní metodologií

vývoje software nebo programovacím jazykem. Ze své podstaty má nejblíže k objektovému paradigmatu a agilnímu přístupu. Aby bylo možné aplikovat DDD přístup, je nutné k vývoji software přistoupit iterativně a zástupci všech zúčastněných domén musí být součástí procesu vývoje. [27]

Vývoj software podle přístupu DDD [27]:

- Studujeme doménu, cílem je získat nejen seznam objektů a pravidel, ale pochopit, jak doména uvnitř pracuje, její vlastní dynamiku a logiku. Klíčová je spolupráce s doménovými experty.
- Dekompozice na subdomény.
- Vytvoříme doménový model.
- Programujeme, píšeme kód.

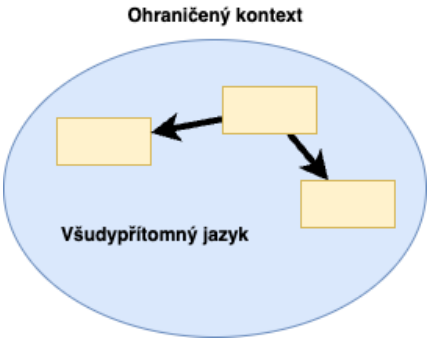
DDD přístup je vhodný pro komplexní složité úlohy. V případě, že implementujeme jednodušší úlohy, které provádí jednoduché CRUD operace, pak je doporučeno využít jiných vhodných architektonických přístupů. DDD není solitérní technika, ale vhodně se doplňuje z dalšími návrhovými vzory pro architekturu systémů. DDD přístup se používá zejména k poznání domény, stanovení komunikačních pravidel mezi týmy a identifikace nezávislých kontextů v rámci celé domény. [26]

Model je abstrakce reálného světa, doména je oblast zájmu a poslední slovo z názvu DDD je design, k jehož vysvětlení si vypůjčím citát:

„Most people make the mistake of thinking design is what it looks like. People think it's this veneer—that the designers are handed this box and told, "Make it look good!" That's not what we think design is. It's not just what it looks like and feels like. Design is how it works." [28]

3.4.1 Strategický návrh (Strategic Design)

Strategický pohled zasazuje jednotlivé objekty do určitého kontextu. Kontext je vlastně ohraničené prostředí, ve kterém se nachází objekty s nějakými vlastnostmi a chováním, což je vlastní objektovému paradigmatu. Strategický návrh to jsou tři důležité aktivity. Rozdělení problémové oblasti na menší celky pomocí návrhového vzoru zvaného Ohraničený kontext (Bounded Context), druhým bodem je definice všudypřítomného jazyka pro konkrétní ohraničený kontext, a nakonec mapování kontextů, což je o vyjádření vztahů mezi kontexty a možnými způsoby integrace. [27]



Obrázek 3.13 - DDD ohraničený kontext [27]

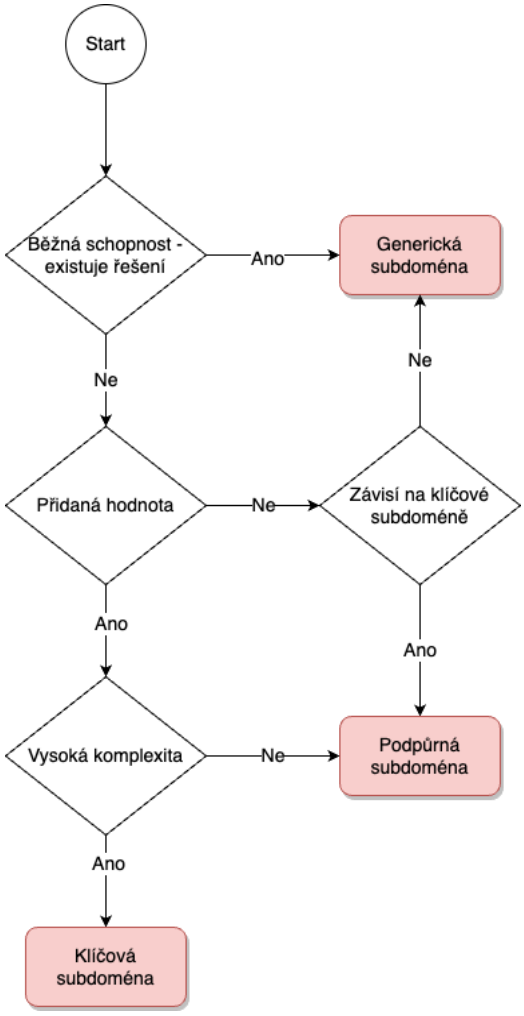
3.4.1.1 Subdomény

Máme celkem tři druhy subdomén v rámci doménového modelu [27]:

- Generické (Generic),
- Klíčové (Core),
- Podpůrné (Support).

Pro **generické** (Generic) subdomény je vlastní, že v této oblasti máme k dispozici řešení, doporučené postupy (best practices). Není zde prostor pro získání obchodní výhody. Příkladem takové subdomény jsou lidské zdroje nebo správa majetku. Řešení nevyvíjíme, ale kupujeme. [27]

Klíčová (Core) subdoména oproti generické přináší konkurenční výhodu. Je definována kolem vlastního know-how a je zde prostor pro vlastní vývoj. V této části je důležité nasadit to nejlepší co máme, protože zde vzniká



Obrázek 3.14 – Subdomény [27]

produkt, který se odlišuje a má obchodní potenciál. Je to dynamická oblast, která se neustále vyvíjí. [27]

Podpůrné subdomény nemají žádný přímý obchodní dopad, ale klíčové subdomény na nich závisí. Jejich obchodní logika není komplexní. Do této domény lze například zařadit zákaznickou podporu. Jelikož jsou známé postupy, ale řešení obvykle není postaveno, pak se doporučuje tuto oblast outsourcovat. Níže uvedený diagram graficky znázorňuje výše popsané vztahy mezi typy subdomén. [27]

3.4.1.2 Ohraničený kontext (Boundary Context)

Kontextem domény rozumíme ohraničenou oblast problému, která definuje společný jazyk, chování, události nebo pravidla. Kontext dává objektům význam a dva stejné objekty v různým kontextu mají různý význam, různé chování a platí pro ně odlišné slovníky. Komplexní systémy mohou být značně komplikovaný a jejich doména rozsáhlá, proto se doména takových systémů dekomponuje na subdomény, kdy cílem je stabilizovat model a jazyk. Když se objeví objekt, který má odlišný význam pro různé týmy, pak dochází k narušení integrity, k nejednoznačnosti, což je impuls pro vytvoření ohraničeného kontextu. Dalším důvodem vzniku ohraničené kontextu je závislost na externích systémech nebo starším kódu čili potřeba integrace. Z výše uvedeného plyne, že ohraničený kontext seskupuje a izoluje prvky, které mají společný jazyk a obchodní pravidla nebo procesy. Generují tak nezávislou implementaci s dobře definovanými objekty a vlastním komunikačním rozhraním. [26]

Zjednodušeně lze napsat, že ohraničené kontexty zajistí [26]:

- Opatření proti nejednoznačnosti.
- Rozdělení domény na menší části, zjednodušení návrhu software.
- Zjednodušení integrace do externích systémů.

Jeden tým může pracovat na více ohraničených kontextech, ale více týmů nesmí pracovat na jednom ohraničených kontextu. Každý kontext má své vlastní datové schéma,

vlastní úložiště zdrojového kódu a vlastní jazyk, je to nezávislá implementace určité části komplexního projektu. [27]

3.4.1.3 Společný všudypřítomný jazyk (Ubiquitous Language)

Cílem je mít společný jazyk pro všechny zúčastněné entity v rámci dané domény, tím se vyhnout nedorozuměním a šířit stejnou kvalitu znalosti domény napříč týmy. Je to sjednocení komunikace mezi softwarovými vývojáři a doménovými experty. Specifické odborné termíny jsou zaznamenány ve sdíleném slovníku, což je obvykle dokument, který je součástí dokumentace projektu a je jedinečný pro každý ohraničený kontext. Definice hesel a založení slovníku obvykle začíná při specifikaci funkčních požadavků. Přístup ke slovníku musí mít všichni členové týmů a za aktualizaci tohoto dokumentu zodpovídá vývojový tým. Vhodnými nástroji ke sdílení jsou Microsoft Teams, Confluence a jim podobné nástroje. Slovník obsahuje různé typy slov, obvyklé jsou slovesa, podstatná jména a idiomy. Snahou je hesla vysvětlovat co nejblíže přirozenému jazyku dané domény. Pro maximalizaci užitku je důležité používat slovníková hesla při každé příležitosti kdykoliv a kdekoliv v mluveném nebo psaném projevu nebo při programování. Při programování je pro DDD přístup kritické, aby pojmenování tříd, metod, tabulek v databázi nebo jmenných prostor bylo v souladu s hesly ve slovníku všudypřítomného jazyka. Z toho je patrné, že změna ve slovníku má zásadní dopad do celého projektu, protože to znamená změnu v modelu a také změnu v programu. Je zakázáno používat synonyma k heslům ze slovníku. [27]

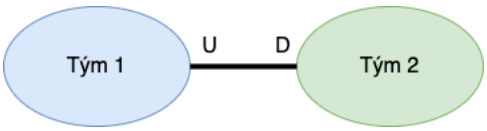
3.4.1.4 Kontextová mapa

Typicky se řešení skládá z více ohraničených kontextů, kdy ohraničený kontext často kopíruje organizační strukturu dané entity, která projekt zadává. Každý ohraničený kontext má svou jedinečnou identitu se svým společným slovníkem a implementací s vlastním technologiemi nebo programovacím jazykem. Například jeden kontext využívá architektury CQRS a další je implementovaný pomocí jednoduché dvouvrstvé datové vazby. K získání pohledu na systém jako celek využíváme kontextových map. Mapa je diagram, který poskytuje komplexní pohled na celý systém. Důležitým aspektem kontextové mapy je vztah,

který vyjadřuje typ komunikace a vzájemnou souvislost mezi jednotlivými kontexty. V níže uvedených typech používám pojmy odběratel (downstream – D) a vydavatel (upstream – U). Oba pojmy označují směr vztahu. Vydavatel má vliv na kontext odběratele, ale naopak to neplatí a vliv může nabývat různých podob, které jsou popsány v následujících odstavcích. [27]

Sdílené jádro (Shared Kernel) je část kontextu, která je společná pro další kontexty, z čehož plyne že jednotlivé týmy nemohou nezávisle měnit společnou část, bez předchozí domluvy se správcem společného kontextu nebo týmy, které jádro sdílí. Je doporučeno, aby jádro bylo malé. [27]

Obrázek 3.15 - DDD sdílené jádro [27]

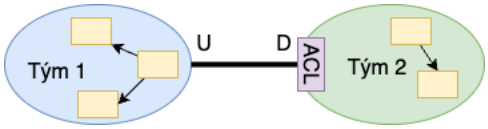


Obrázek 3.16 - DDD zákazník – dodavatel [27]

Zákazník-Dodavatel (Customer-Supplier), kde dodavatel je kontext vydavatele (U) a zákazník je kontext odběratele (D). Dodavatel v tomto vztahu vládne, protože zajišťuje to, co po něm zákazník chce, z čehož plyne, že dodavatel určuje, co a kdy. Je to velmi běžný vztah mezi týmy v rámci jedné organizace. Tým 2 čeká na tým 1. Je důležité pečovat o čistý vztah, rozumějte, že potřeby zákazníků by se měly promítnout do priorit dodavatelů. V tomto vztahu mají odběratelé možnost specifikovat své potřeby. Někdy je ve velkých korporacích autonomní činnost dodavatelů omezována. [27]

Konformista (Conformist) vychází z podobného vztahu jako zákazník-dodavatel s rozdílem, že dodavatel nemá motivaci poskytnout zákazníkovi požadované, odběratel nemůže definovat co potřebuje a je plně závislý na dodavateli. Tento vztah se obvykle vyskytuje, když dodavatel je externí služba, starší kód nebo když se integruje do rozsáhlého modelu. [27]

Antikorupční vrstva (Anticorruption layer – ACL) je defenzivní přístup, kdy jsou jednotlivé týmy zcela odděleny. Je to pevná komunikační vrstva mezi ohraničenými kontexty. ACL izoluje model odběratele



Obrázek 3.17 - DDD ACL [27]

(D) od modelu dodavatele a zajišťuje překlad komunikace mezi nimi. Odběratel (D) pevný komunikační bod mezi ohraničenými kontexty, která je vhodná pro integrace služeb. [27]

Obrázek 3.18 - DDD OHS [27]

Otevřená služba (Open Host Service) definuje rozhraní nebo přístupový protokol do daného ohraničeného kontextu. Každý, kdo se chce integrovat s tímto kontextem využije tento protokol. OHS implementuje API, které je dobře zdokumentované, například REST nebo SOAP. [27]

Veřejný jazyk (Published Language) je dobře dokumentovaný společný jazyk pro jednoduchou komunikaci mezi submodely. Je definován jako XML, JSON schéma nebo je možné využít i jiné formáty jako Protobuf nebo Avro. Často se veřejný jazyk kombinuje s otevřenou službou, což nabízí lepší možnosti integrace pro třetí strany. [27]

3.4.1.5 Realizace propojení kontextů

K integraci s jiným ohraničeným kontextem můžeme využít SOAP RPC, RESTful http nebo prostřednictvím výměny zpráv. Varianty s integrací skrze databázi nebo souborový systém nejsou doporučovány a neměly by se objevovat a pokud je nutné využít tento přístup, pak je nutné zajistit, aby odběratel byl izolován od jiných kontextů prostřednictvím ACL vrstvy. [27]

RPC je zkratka Remote Procedure Call a je jedním z oblíbených přístupů komunikace skrze Simple Object Access Protocol (SOAP). Princip spočívá ve volání lokálních procedur nějakého systému prostřednictvím SOAP požadavku. SOAP požadavek je přenášen sítí k vzdálené službě, kde se vykoná požadovaná procedura. Odpověď je po síti přenesena zpět ke zdroji požadavku. Mezi SOAP RPC a komunikujícími kontexty vznikne silná vazba. Nejlepší variantou implementace je, že kontext, který zveřejňuje své zdroje je služba se zdokumentovaným API a druhý kontext je klient izolovaný proti nechtěným přístupům ACL vrstvou. [27]

RESTful (Representational State Transfer) http je přístupový protokol, který se zaměřuje na přístup ke zdrojům kontextu prostřednictvím čtyřech základních http operací GET, POST, PUT, DELETE. Je to přístup, který umožňuje definovat dobré API pro distribuované systémy. Použití je obdobné jako u SOAP, tedy služba, která zveřejňuje API s veřejným slovníkem, který popisuje zdroje přístupné pomocí výše popsaných metod a k ní se připojuje klient, který je izoluje svůj kontext prostřednictvím ACL vrstvy. Chybou by bylo v kontextu služby vytvářet agregáty v souladu s REST API. Je důležité, aby API poskytovalo data tak jak potřebuje klient, nikoliv jak vypadá model ohraničeného kontextu služby. [27]

Výměna zpráv je asynchronní varianta komunikace mezi kontexty. Je to robustní řešení odolné vůči chybám, na které jsou citlivé SOAP nebo REST API, tedy na výpadek síťového spojení nebo na chyby na serverové straně. Řešení musí obsahovat aspoň dva kontexty, kde jeden je v roli vydavatele (Publisher) a druhý v roli odběratele (Subscriber). Kvalita řešení je přímo závislá na zvoleném systému zpracování zpráv, který by měl minimálně podporovat metodu doručení typu At-Least-Once (aspoň jednou). Je to obvyklá varianta, kdy zpráva je považována za doručenou až když ji odběratel zpracuje. Když proces doručení selže, celý proces se znovu opakuje. Aby nedocházelo k duplicitám na straně odběratele, musí být implementovaný jako idempotentní. [27]

3.4.2 Taktický návrh (Tactical Design)

Taktický návrh zpřesňuje analýzu domény. Zabývá se podrobnostmi implementace softwarového projektu. Pomocí několika nástrojů popisuje model uvnitř ohraničeného kontextu. Nástroje jsou entity, hodnotové objekty, události nebo úložiště. Jejich použití bude vysvětleno a demonstrováno v praktické části práce. Taktický návrh pomáhá definovat přirozené hranice mikroslužeb a využívá k tomu zejména agregáty a entity. Principem je, že mikroslužba by neměla být menší než agregát a větší než ohraničený kontext. [27]

Entita (Entity) je jedinečný objekt, který má své unikátní id (GUID, primární klíč) a je persistentní, rozumějte je uložena v databázi. Obsahuje atributy, které v průběhu času mohou měnit svou hodnotu. Entita může odkazovat na jinou entitu a vytvářet tak vztahy. [27]

Naproti tomu objekty nesoucí hodnotu (Value Object) jsou objekty, které nejsou jedinečný a nemění svůj stav, jsou imutabilní. Pokud chci aktualizovat stav objektu, musím vytvořit jeho novou instanci a tou nahradit instanci původní. Typickým příkladem takového objektu je měna, datum nebo barvy. Hodnotový objekt lze chápat, jako rozšíření datových typů. [27]

Množina entit, které mají spolu něco společného, jsou vzájemně propojené a vše vychází z jedné dané entity (root), se seskupují do množiny, kterou nazýváme agregát (aggregates). Výhodou agregátů je, že zjednodušíme doménový model a podpoříme konzistenci dat. Lze se na to dívat i jako na zapouzdření podobných objektů do většího kontejneru. Součástí agregátů mohou být i objekty nesoucí hodnotu. Agregát musí obsahovat aspoň jednu entitu, a právě tato jedna entita je vždy kořenová (root). Přes kořenovou entitu se přistupuje k ostatním entitám v agregátu. Neexistuje přesná definice nebo postup, který určuje, jak identifikovat entity, které společně definují agregát. Jejich vznik iniciuje analýza obchodních procesů a pravidel nebo nástroje jako Event Storming. [27]

Událost je činnost, která se odehrála v minulosti, známe jejího vlastníka, její datum a čas výskytu. Doménová událost informuje další část systému, že se něco stalo v rámci domény, takže to je událost typu zrušení objednávky, vytvoření objednávky apod. Jsou to události, které obvykle mají vztah k obchodnímu procesu. Doménové události v architektuře mikroslužeb zajišťují synchronizaci a komunikaci mezi jednotlivými službami. K výměně zpráv se využívá asynchronní komunikace. [27]

3.4.2.1 Event Storming

Jedná se o brainstorming, schůzku nebo schůzky mezi doménovými experty a vývojáři, která je zaměřena na obchodní procesy. Vzájemnou spoluprací jsou identifikovány klíčové prvky taktického návrhu jako agregáty, doménové události nebo příkazy. Společně s definicí prvků na úrovni taktického návrhu, dochází v rámci těchto schůzek k zpřesňování nebo definici nových hranic ohraničených kontextů. [27]

Modelování probíhá za pomoci samolepících papírků různých barev. Je nutné vyhnout se UML nebo podobným formálním technikám, protože doménový experti často těmto formám zápisu nerozumí a zpomaluje to tak vzájemné porozumění. Je to hlavní příčina řádného dosažení cíle z důvodu nejasností nebo nedorozumění. Event storming díky své jednoduchosti za pomoci samolepících papírků různých barev, které jsou intuitivní, dokáže nedorozuměním zabránit. Obvykle probíhá v průběhu několika dní s jednou několikahodinovou schůzkou denně. Důležitou zásadou brainstormingu je, že žádná myšlenka nebo nápad se nezavrhuje a nekritizuje. [27]

Event storming se provádí v pěti krocích [27]:

1. Definice doménových událostí (oranžové lístečky)
2. Definice příkazů (modré lístečky)
3. Definice entit a agregátů (žluté lístečky)
4. Zakreslení ohraničených kontextů (fixem)
5. Identifikace pohledů (zelené lístečky)

K těmto základním krokům existují ještě dva druhy aktivit, které mají zastoupení v barevných lístečcích. Jsou to aktivity, které se prolínají všemi jednotlivými kroky a doplňují získanou znalost tím, že zakreslují do diagramu událostí, příkazů nebo entit fialovou barvou lístečku procesy nebo světle žlutým lístečkem aktory. Všechny zde uvedené barvy jsou doporučené, je možné si zvolit vlastní, důležité je všeobecná znalost významu jednotlivých barev u všech členů týmu. [27]

4 Vlastní práce

Vlastní práce spočívá v aplikování teoretického poznání do praxe. Hlavním záměrem je demonstrace návrhových vzorů DDD a architektury mikroslužeb při návrhu komplexního informačního systému z oblasti IoT. Vedlejším produktem práce je využití cloudových služeb při implementaci vybraných částí navrženého řešení.

4.1 Definice zadání

V současné době existuje mnoho senzorů pro měření různých fyzikálních veličin nebo aktivních prvků pro automatizaci procesů v různých oblastech. Od amatérských instalací v podobě automatického otevírání dveří až po profesionální instalace v oblastech precizní zemědělství, průmyslu 4.0, dopravy nebo chytrých měst. Tyto instalace využívají různé varianty konektivity k internetu pro přenos získaných dat. Od sítí krátkého dosahu až po LPWAN sítě. Pro potřeby vlastní práce jsem vybral LPWAN sítě, které jsou vhodné pro rozsáhlé instalace typu mMTC.

Cílem vlastní práce je navrhnout s využitím návrhových vzorů DDD a architektury mikroslužeb informační systém, který uživateli usnadní výběr vhodné LPWAN sítě na základě síly signálu v dané lokalitě, umožní celkovou správu připojených zařízení nebo přenášet data z aktivních prvků připojených do podporovaných sítí do vlastní databáze pro potřeby pozdější analýzy nebo prezentace a přístup k nim bude umožněn skrze vlastní aplikační rozhraní. Z výše uvedeného vyplývá, že se jedná o komplexní systém s různými požadavky na uložení dat, z čehož plyne, že vhodnou variantou pro návrh systému bude architektura mikroslužeb.

K integraci jsem vybral dva druhy sítí od dvou poskytovatelů, kterými jsou České radiokomunikace s technologií LoRa a Sigfox ČR. Zpracování získaných dat, datová analýza nebo například využití umělé inteligence pro získání informace nejsou součástí vlastní práce. Součástí zadání je níže přiložený Business Model Canvas.

Business Model Canvas

Designed for:	*Designed by:*	*Date:*	*Version:*
ČZU	Ladislav Herynek	xx.xx.xxxx	0.01

Key Partners:

Poskytovatelé LPWAN sítí.

Poskytovatel cloud computing prostředí.

Key Activities:

- průzkum trhu,
- návrh produktu + klikatelné demo,
- uzavření smluv s klíčovými partnery,
- marketingové kampaně.

Key Resources:

- softwarový architekt,
- copywriter,
- UX designér,
- vývojáři,
- manažér partnerských vztahů,
- marketingový experti.

Value Propositions:

Pohodlná práce s heterogenním LPWAN prostředím z jednoho bodu.

Jeden účet pro vícero LPWAN sítí, jedna registrace, jedna platba.

Vyjednávací síla zájmové skupiny pro vyjednávání s korporacemi a provozovateli velkých síťových instalací.

IoT komunita, která tvoří vlastní obsah – blog, sociální síť.

Knihovna s odborným obsahem z oblasti IoT.

Customer Relationships:

- samoobslužný portál,
- cílené průzkumy spokojenosti,
- podpora 24x7.

Channels:

- sociální sítě,
- odborná diskusní fóra,
- partneři dodávající do sektoru SMB.

Customer Segments:

Fyzické osoby nebo malé firmy.

Primární zaměření na trh ČR.

Cost Structure

- návrh a vývoj webové aplikace,
- marketing,
- LPWAN sítě,
- cloud computing,
- poplatek partnerům.

Revenue Streams

- instalační balíčky (sensor + síť),
- měsíční paušální platba,
- roční paušální platba.

Obrázek 4.1 - Business Model Canvas

4.2 Domain Driven Design

Je to kapitola věnující se analýze a návrhu. Cílem této části rozdělení komplexního problému na menší nezávislé celky, které usnadní identifikaci mikroslužeb. Vstupem do analýzy je předchozí kapitola, které se věnovala popisu problému z obchodního hlediska. Před zahájením analýzy je důležité pamatovat na to, že DDD je iterativní a živý proces, tedy nikdy není dosažen fixní stav návrhu, protože se neustále vyvíjí. V této práci bude uplatněn následující postup:

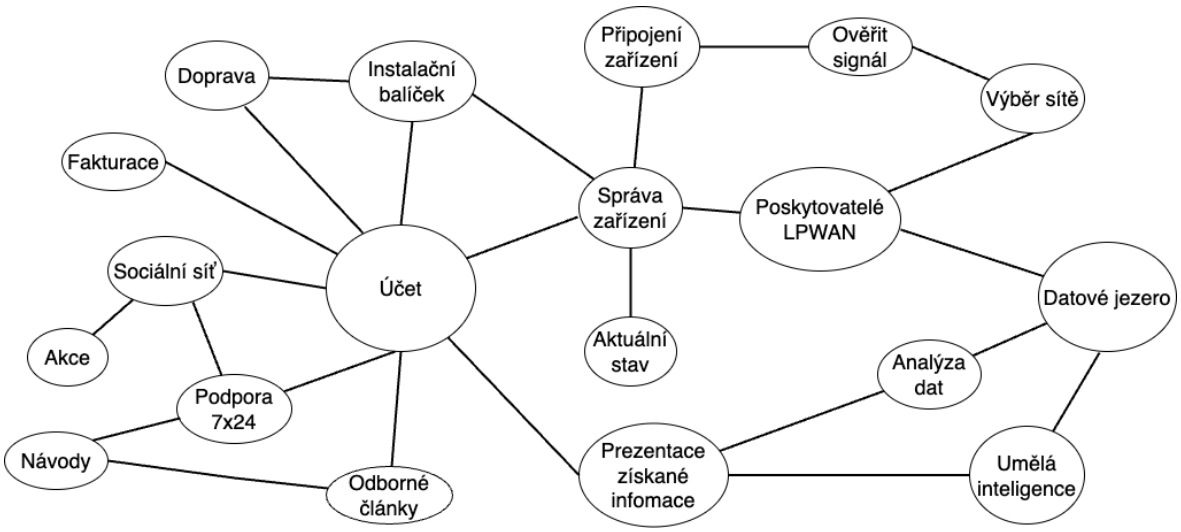
1. analýza domény,
2. určení ohraničených kontextů,
3. identifikace entit a agregátů,
4. identifikace mikroslužeb.

4.2.1 Strategický návrh (analýza domény)

Výstupem je porozumění doméně a identifikované funkční požadavky. Výstup je výsledkem řady schůzek mezi doménovými experty, architekty, případně dalšími potřebnými členy týmu. Na schůzkách je iniciován vznik slovníku všudypřítomného jazyka. Slovník i brainstorming jsou hypotetické události, které demonstrují, jak by v reálném případě probíhala analýza problému vlastní práce nějakého komplexního systému v rámci spolupráce několika lidí z různých týmů.

Vlastní práce byla zahájena fází definice zákazníka. Pro koho je produkt tvořen:

- Subjekty jejichž záměrem je v rámci IoT provozovat aktivní prvky v množství o maximálně několika desítkách kusů.
- Chtějí jeden účet s jednou platbou a provozovat zařízení v různých sítích. Měsíční paušál má nastavené limity v podobě počtu připojených zařízení.
- Hledají portál, kde na jednom místě mohou vybrat z nabídky různých LPWAN sítí na základě parametrů jako síla signálu pro vybranou lokalitu nebo množství přenesených zpráv za den.
- Potřebují systém pro správu IoT aktivních prvků v heterogenním LPWAN prostředí,
- Chtějí sdílet a využívat znalostí a spolupráce v rámci komunity.



Obrázek 4.2 - Analýza domény

Výše uvedený diagram zachycuje klíčové funkce systému a vztahy mezi nimi. Typicky je takový diagram výsledkem výše zmíněných schůzek. V souvislosti s touto prací z diagramu vyplývá, že se jedná o komplexní systém pro správu zařízení, získání poznání z nasnímaných dat a komunitní části obsahující články, návody a sociální síť. Každá část reprezentuje nějaký typ subdomény:

- Účet je jedním z klíčových prvků diagramu. Vždy nějakým způsobem zasahuje do jiných prvků diagramu. Je to klíčová (core) subdoména. Prostřednictvím účtu jsou spravována zařízení, obsah komunitní části nebo jsou realizovány akce na konkrétní množinu uživatelů.
- Prostřednictvím modulu správa zařízení spravujeme zařízení v LPWAN sítích, na základě vybraných parametrů vybíráme optimální síť pro uživatele nebo zjišťujeme stav konkrétního zařízení včetně prezentace aktuálních dat. Je to klíčová (core) doména řešení, prostřednictvím které realizujeme přidanou hodnotu pro uživatele.
- Datový jezero je další klíčová (core) subdoména. Je to zdroj nasnímaných dat pro datovou analýzu, data mining nebo umělou inteligenci. Cílem je hledat složité netriviální vztahy a získat poznání. Tato část je klíčovým prvkem znalostního systému. Datové jezero bude rozděleno na mnoho menších částí, datových skladů a datových tržišť, které budou obsahovat specializované informace například pro analýzu kvality bazénové vody, kvality půdy a mnoho dalších případů užití.

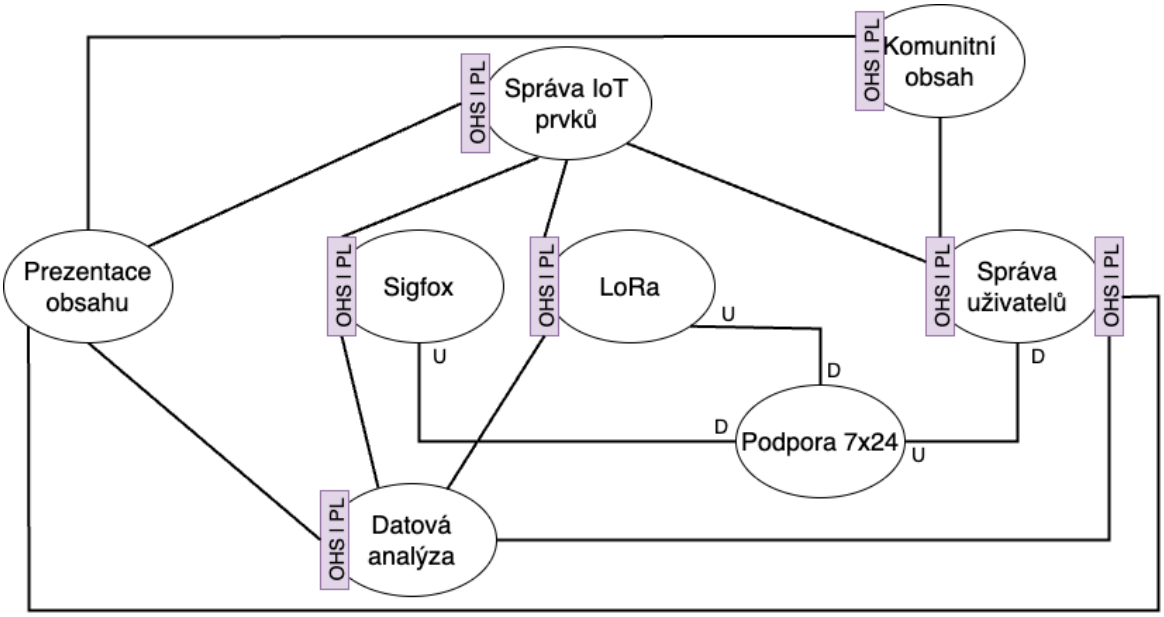
- Analýza dat a umělá inteligence jsou klíčové subdomény, které přináší přidanou hodnotu zákazníkovi. Jejich posláním je získat informace z obrovského množství nasnímaných dat a podporovat tak správná rozhodnutí.
- Instalační balíček a doprava zastupuje předpřipravené sady aktivních prvků, které lze objednat, snadno připojit do sítě a ihned provozovat. Snadno tak lze provozovat vlhkoměr, teploměr nebo nějaký jiný vybraný sensor. Jedná se o podpůrnou doménu.
- Sociální síť, návody nebo odborné články jsou součástí generické subdomény.
- Prezentace získané informace, fakturace a podpora 7x24 jsou podpůrnou doménou, protože přímo podporují klíčovou subdoménu.
- Poskytovatelé LPWAN jsou služby třetí strany (Sigfox, ČRa). Jsou to provozovatelé přenosové soustavy. Je to generická subdoména.

4.2.1.1 Ohraničený kontext

Pro identifikaci ohraničených kontextů neexistuje přesně stanovený postup. Není to věda, je to spíše umění a schopnost správně určit hranice kontextu a tato schopnost úzce souvisí se zkušenostmi a znalostmi v oblasti domény. Pro potřeby vlastní práce jsem z pozice doménového experta na základě vlastních zkušeností a analýzy domény identifikoval níže uvedené výchozí ohraničené kontexty:

- správa zařízení,
- Sigfox (externí systém),
- LoRa ČRa (externí systém),
- datová analýza,
- správa uživatelů,
- správa obsahu,
- prezentace obsahu,
- podpora 24x7 (externí systém).

Níže uvedená kontextová mapa graficky znázorňuje typ vztahů mezi jednotlivými ohraničenými kontexty. Co který vztah znamená je popsáno v teoretické rešerši v části Domain Driven Design. Zakreslené čáry neurčují tok dat nebo informací. Do diagramu nebylo potřeba zanést ACL vrstvy, protože systém neobsahuje žádnou starší (legacy) službu, kterou by bylo nutné pomocí antikorupční vrstvy oddělit. OHS|PL jsou služby, které mají dokumentované API, obvykle OpenAPI.



Obrázek 4.3 - Kontextová mapa

Každý ohraničený kontext vede svůj vlastní slovník pro komunikaci v týmu. V rámci této práce nebude generován slovník všudypřítomného jazyka, protože to je klíčový prvek pro práci v týmu a mezi týmy, což není případ této práce.

4.2.2 Taktický návrh

Pro potřeby další analýzy a následné realizace byl vybrán ohraničený kontext „Správa IoT prvků“. Cílem je identifikace agregátů a entit, jejichž pomocí bude možné přesněji identifikovat hranice jednotlivých mikroslužeb.

4.2.2.1 Případy užití a scénáře

Případy užití (use case) pro ohraničený kontext LPWAN, kdy zákazník očekává možnost:

- Na základě zadaných GPS souřadnic zjistit sílu signálu v dané lokalitě.
- Vznést požadavek na připojení, deaktivaci nebo odebrání zařízení z vybrané LPWAN sítě.
- Zakoupit předpřipravený samoinstalační balíček s předem nakonfigurovaným senzorem.
- Zobrazit všechna vlastní provozovaná zařízení a filtrovat na základě lokality nebo typu sítě.

- Zobrazit poslední stažené zprávy, a to buď pro skupinu zařízení nebo pro jednotlivá zařízení.
- Vzdáleně konfigurovat připojené zařízení.
- Upozornění na příjem nových dat z připojených zařízení.

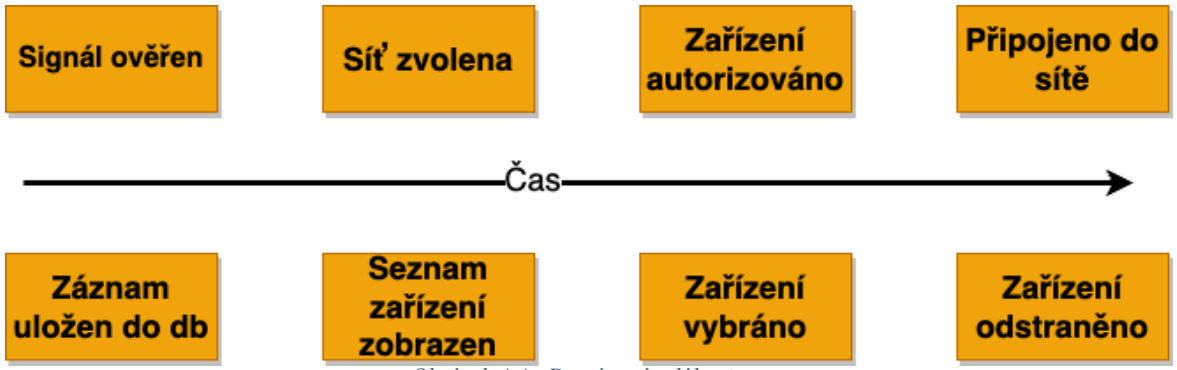
Scénáře pro ohraničený kontext LPWAN, kdy systém:

- Eviduje počet odeslaných a přijatých zpráv za den.
- Eviduje poslední přenesenou zprávu.
- Umožní seskupovat zařízení do skupin, které jsou předem definované, například meteostanice nebo půdní senzory.
- Umožní aktivovat senzor v pravidelných opakujících se intervalech, nebo při dosažení prahové hodnoty, nebo na vyžádání.
- Aktivace senzoru automaticky vyvolá událost pro přenos dat do datového jezera, které se nachází v kontextu datová analýza.

4.2.2.2 Event storming

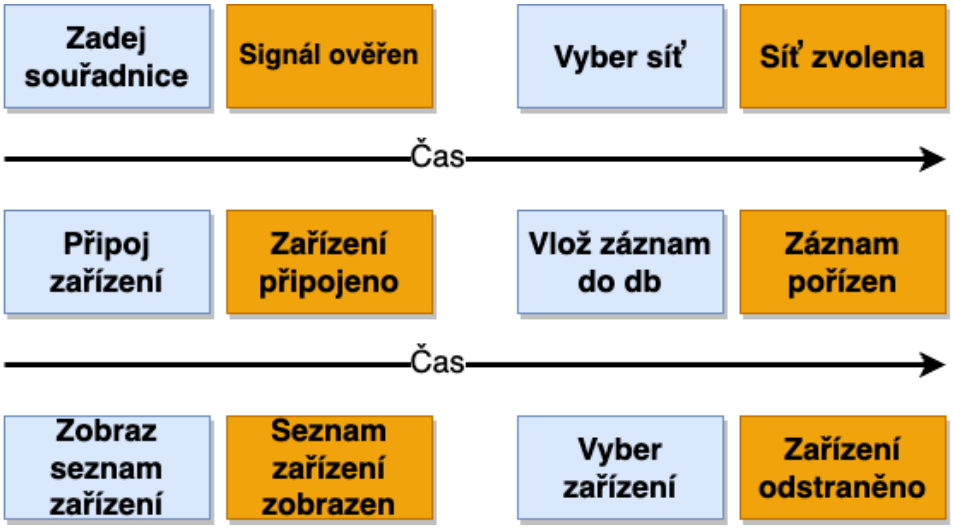
Analyzovány budou procesy přidání nebo odebrání zařízení, ověření síly signálu a zobrazení všech vlastních připojených zařízení.

Krok 1 – generování doménových událostí. Byly identifikovány události, viz obrázek 4.4. Události jsou pojmenovány v minulém čase a jsou časově seřazeny zleva doprava od prvního do posledního výskytu. Nebyly identifikovány žádné paralelní události, které by se řadili vedle sebe ani žádný proces. Výše uvedené doménové události jsou řazeny pod sebe ne z důvodu paralelního výkonu, ale z důvodu prezentace v tomto dokumentu, proto jsou jednotlivé řádky událostí odděleny časem.



Obrázek 4.4 - Doménové události

Krok 2 – generování příkazů (command) je zakresleno na obrázku 4.5 a jsou tam ty, které vedly k výskytu dříve nalezených doménových událostí. Všechny příkazy jsou pojmenovány imperativem a v případě této práce nebyla zaznamenána žádná událost, která by vedla k výskytu události bez příkazu (události podmíněné časovým intervalem). Z důvodů zachování jednoduchosti byly z definic příkazů vynechány významově důležité entity (aktor), které spouští určité události. Zároveň se v průběhu definice příkazů nevyskytl žádný proces, který by navazoval na události nebo příkazy. V průběhu druhého kroku může docházet k definici nových doménových událostí nebo k editaci stávajících, z čehož plyne že se změnil set událostí oproti kroku jedna.

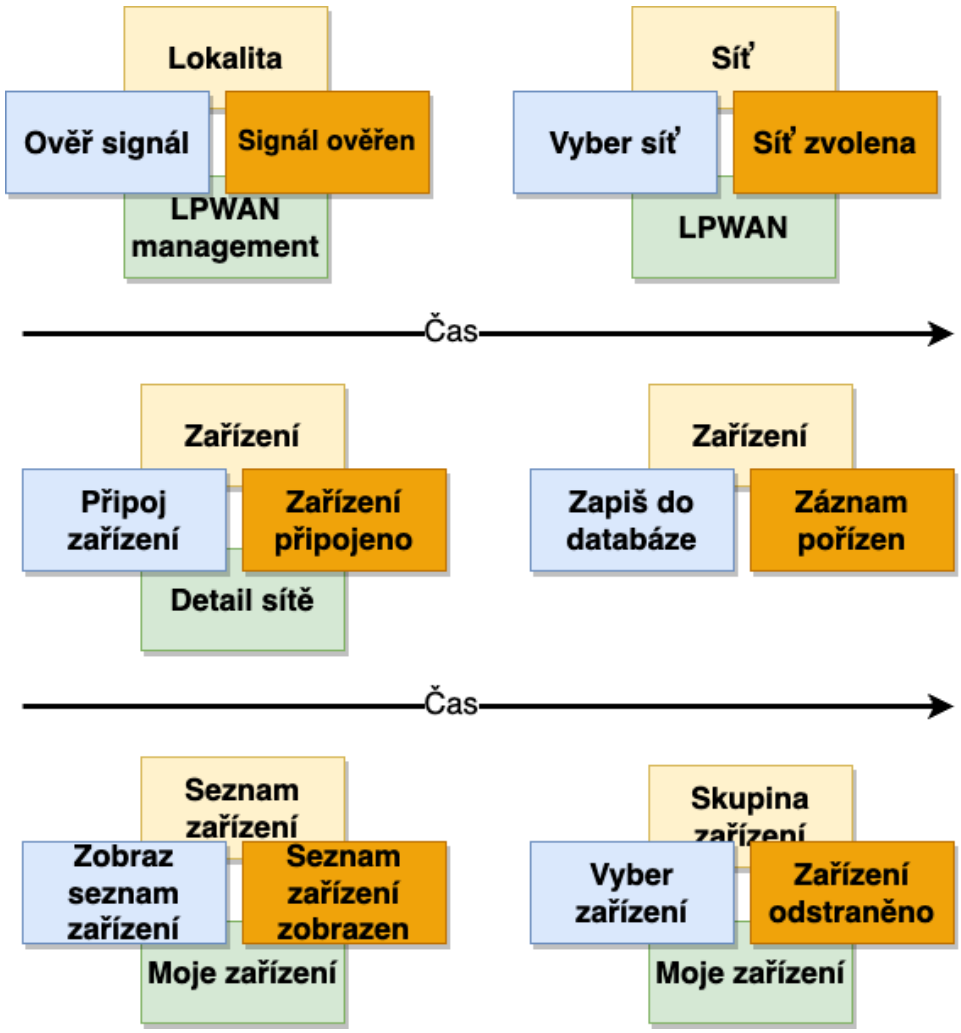


Obrázek 4.5 - Doménové události + Příkazy

Krok 3 – Asociovat entity/agregáty s příkazy a doménovými událostmi. V této části se pracuje s daty, které příkaz vykonává a na výstupu je doménová událost. Výsledek kroku tři je zaznamenán na obrázku 4.6, který je společný s krokem 5. Nebyl nalezen takový agregát, který by byl natolik komplexní, aby byl v diagramu zaznamenán jako proces.

Krok 4 – Zakreslení ohraničených kontextů v tomto případě nebude aplikováno, protože je zpracováván jeden konkrétní ohraničený kontext, který v průběhu Event storming nezaznamenal významnější podnět pro změnu jeho hranic.

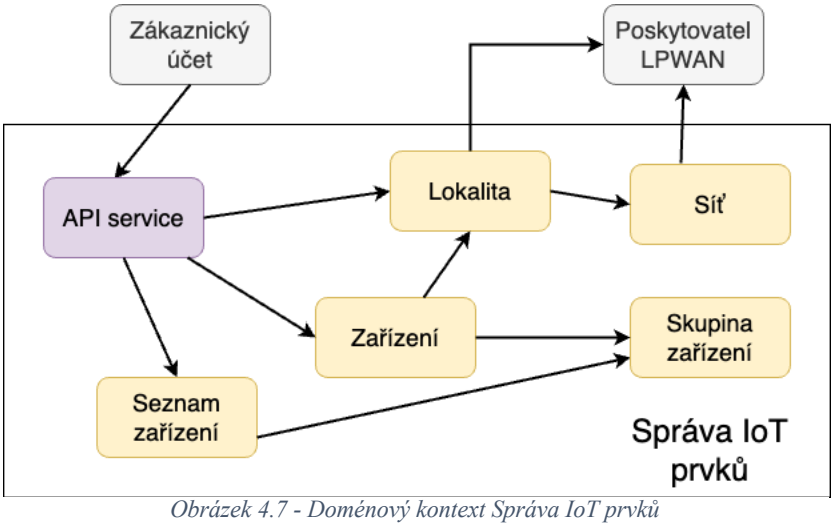
Krok 5 – Identifikace pohledů je krok, kde došlo k zaznamenání pohledů, které zajišťují interaktivitu aktora se systémem. K identifikovaným obrazovkám byly zpracovány návrhy obrazovek formou drátových modelů, které jsou součástí práce a jsou uvedeny v kapitole 4.4.5 Prezentační vrstva, strana klienta.



Obrázek 4.6 - Finální taktický návrh

4.3 Návrh mikroslužeb

Nejdůležitější otázkou v této části práce je, jaká je správná velikost mikroslužby. S nalezením optimálního množství mikroslužeb nám pomohla předchozí kapitola věnující se Domain Drive Design, protože mikroslužba nesmí být větší než ohraničený kontext a nesmí být menší než agregát. Vstupem do návrhu mikroslužeb tak je obrázek 4.7, který je finálním produktem taktické analýzy v rámci DDD. Tento obrázek znázorňuje ohraničený kontext Správa IoT prvků, kde žlutě jsou zvýrazněné entity a agregáty, fialovou barvou doménové služby a šedou barvou vnější ohraničené kontexty. Tým, který vyvíjí a spravuje kontext „Správa IoT prvků“ identifikoval jednu potřebnou doménovou službu, která bude obstarávat veškeré požadavky uživatelů v rámci tohoto kontextu. Jedná se o vstupní bod do kontextu pro vnější entity, které představují externí kontexty nebo klienti. Služba API service neuchovává žádná data, pouze distribuuje požadavky na další služby v rámci aplikace. Přínosem této služby je zvýšení odolnosti systému a jeho responzivity, viz reaktivní manifest.

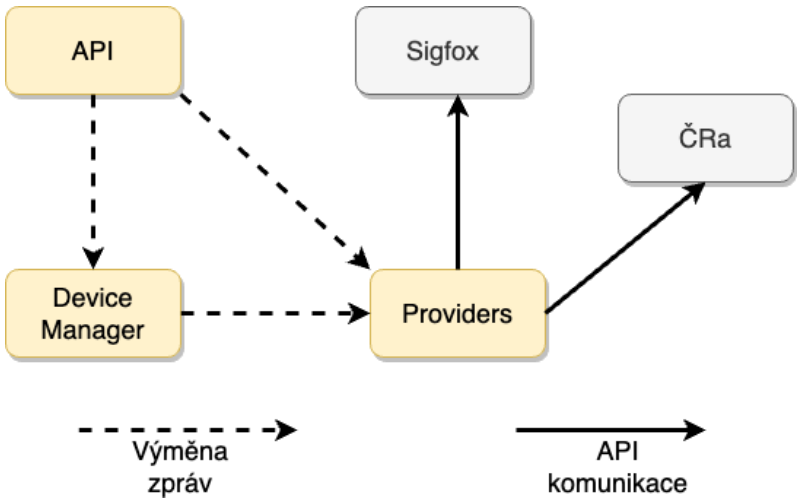


Obrázek 4.7 - Doménový kontext Správa IoT prvků

Ideálními kandidáty na mikroslužby jsou komplexní agregáty nebo doménové služby. Ohraničený kontext definovaný obrázkem 4.7 neobsahuje žádný komplexní agregát s mnoha funkcemi, a tak výsledkem analýzy a návrhu jsou tři mikroslužby, které mezi sebou budou komunikovat prostřednictvím výměny zpráv. První služba „API“, která je derivována z doménové služby „API service“, druhá služba s názvem „DeviceManager“, která bude sloužit k uchování záznamů o vlastních připojených zařízeních a k jejímu vzniku vedli entity „Seznam zařízení, Zařízení a Skupina zařízení“. Poslední služba se bude jmenovat „Providers“, která bude zajišťovat komunikaci s provozovateli LPWAN sítí a k její vznik určila existence externích kontextů v podobě provozovatelů LPWAN sítí.

V rámci vlastní práce budou realizovány služby:

- API, která bude zprostředkovávat komunikaci s klientem, tedy zpracovávat jeho požadavky. Výstupem je zdokumentované API.
- DeviceManager služba, která bude přijímat požadavky klientů formou přenosu zpráv ze služby API. Požadavky představují například připojení zařízení do vybrané sítě nebo odstranění zařízení ze sítě.
- Providers je služba, která bude obsahovat http klienty pro komunikaci s poskytovateli síťového spojení. Komunikace bude často proti zveřejněnému OpenAPI rozhraní typu REST.



Obrázek 4.8 - Mikroslužby Správa IoT zařízení

Při identifikaci těchto mikroslužeb byl brán zřetel na to, aby navrhované služby splňovaly následující:

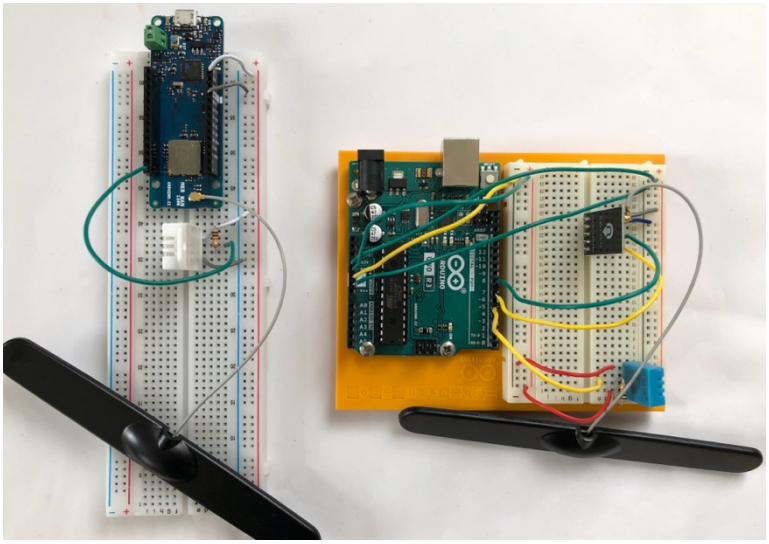
- Princip objektově orientovaného paradigmatu SOLID.
- Byly tak velké, aby na nich mohl pracovat pouze jeden tým zcela nezávisle, protože platí pravidlo, že na jedné službě nesmí pracovat více různých týmů.
- Byly tak velké, aby vzájemná komunikace mezi mikroslužbami nebyla moc upovídaná a nezatěžovala zbytečnou komunikací přenosovou soustavu.
- Systém sestávající se z mikroslužeb byl responsivní, odolný a dobře škálovatelný, tedy vycházel z principu reaktivního návrhu.

4.4 Realizace

Z předchozí kapitoly plyne, že realizace bude spočívat v naprogramování tří služeb. Služba „Api“, kde se budou koncentrovat požadavky klientů, služba „DeviceManager“, která zajistí trvalé uchování dat o vlastních zařízeních a služba „Providers“ jako zprostředkovatele komunikace směrem k poskytovatelům LPWAN sítí. Součástí realizace jsou dva senzory, kdy jeden je kompatibilní se sítí LoRaWAN a druhý lze provozovat v síti typu Sigfox.

4.4.1 Senzory

Reálná data generují dva aktivní prvky, které ke snímání teploty nebo vlhkosti vzduchu využívají senzory DHT11 a DHT22. Senzory se odlišují pouze v citlivosti a rozsahu snímaných veličin. Senzor DHT22 byl osazen na desku Arduino MKR1300 s integrovaným čipem pro komunikaci v síti LoRaWAN. Senzor DHT11 byl osazen na desku Arduino UNO, kterou bylo nutné rozšířit o komunikační modul IoT LPWAN Sigfox Node UART Modem 868 MHz pro přenos dat v síti Sigfox. Arduino v současné době nenabízí MKR desky s integrovaným čipem pro komunikaci v síti Sigfox. Komunikační modul obsahuje roční licenci pro provoz zařízení v síti Sigfox.



Obrázek 4.9 - Senzory

Z pohledu software, byl pro obě zařízení napsán program v jazyce Wiring, který v pravidelných intervalech snímá oba dříve zmíněné jevy a následně skrze LPWAN síť posílá data do cloudového úložiště dat poskytovatele sítě. Součástí implementace nebyla kalibrace senzorů, ani optimalizace výkonu z pohledu nároků na paměť a spotřebu elektrické energie.

4.4.2 LPWAN poskytovatelé

Reálný síťový provoz zajišťují Sigfox ČR a LoRaWAN od Českých radiokomunikací. K používání obou sítí bylo nutné se registrovat v portálech provozovatelů a dohodnout nebo zakoupit potřebný balíček, který umožní provoz dohodnutého počtu zařízení v síti. Oba provozovatelé umožňují přístup do své sítě prostřednictvím dobře zdokumentovaného aplikačního rozhraní typu REST (GET, POST, PUT a DELETE), kde jednotlivé koncové body vrací odpovědi ve formátu JSON. Obě API jsou zabezpečené a je nutné se před použitím jednotlivých koncových bodů autentizovat.

U provozovatele ČRa byl dohodnut speciální tarif pro studijní potřeby, který umožňuje provozovat v síti až 2000 zařízení s objemem 360 zpráv denně odeslaných ze zařízení a 36 zpráv denně přijatých zařízením. [29] Výhodou této sítě je, že je k dispozici široká škála senzorů a mikrokontrolérů, které lze v domácích podmínkách sestavit a v síti provozovat. IoT Backend API je dobře zdokumentované a nabízí širokou škálu koncových bodů od registrace zařízení síti, čtení a odesílání zpráv až po správu uživatelů, skupin nebo MQTT komunikaci. Před použitím jakéhokoliv koncového bodu v rámci IoT Backend API je nutné se autentizovat. Autentizace je realizována prostřednictvím Bearer přístupového tokenu, který je zapotřebí vygenerovat na serveru zpracovávajícím identity na url adrese https://sso.cra.cz/auth/realms/CRA/protocol/openid-connect/token.

V rámci sítě Sigfox byl zakoupen prostor pro připojení dvou zařízení. Stejně jako LoRa od ČRa i síť Sigfox nabízí dobře zdokumentované aplikační rozhraní Sigfox API (2.0) s velkým množstvím koncových bodů pro správu zařízení, účtů, skupin a mnoho dalších funkcí souvisejících s provozem IoT zařízení. V průběhu tvorby této práce bylo identifikováno riziko v podobě úpadku společnosti Sigfox. Společnost Sigfox upadla do konkurzu a trápí ji majetkové poměry. Jaký skutečný vliv na vlastní práci a provoz zařízení do bude mít, bude zjištěno dále v průběhu tvorby. Analýza příčiny úpadku takto významného provozovatele není předmětem této práce. Ve vyjádření firmy je upozorňováno na problém s nedostatkem čipů, který vedl ke zpomalení nástupu internetu věcí a tím k nenaplnění očekávaných finančních příjmů. [30] Dalším faktorem může být zvyšující se konkurence v oblasti IoT, kterou představují velký světový operátoři se sítěmi NB-IoT nebo LTE páté generace. Dalším drobným nedostatkem z pohledu vlastní práce a sítě Sigfox se jeví fakt, že Arduino nedisponuje mikrokontrolérem s přímou podporou komunikačního protokolu Sigfox. Problém byl vyřešen rozšiřujícím modulem, viz stavba senzoru.

Každý koncový bod v Sigfox API (2.0) vyžaduje při jeho volání autentizaci. K tomuto účelu je nutné v rámci Sigfox portálu vytvořit API uživatelský účet. Portál vygeneruje dva klíče s názvem účet a heslo, které se používají pro autentizaci při volání koncových bodů. Takto vytvořenému účtu lze přiřadit uživatelská práva, která mohou opravňovat ke správě zařízení, k příjmu zpráv, správě uživatelů a skupin nebo dalších akcí v rámci Sigfox API. Pokud by někdy došlo ke kompromitaci daného účtu, je možné vygenerovat nové klíče potřebné k autentizaci.

Pomocí aplikace Postman, která slouží ke komunikaci s API, bylo ověřeno vytvoření profilů a autentizace k API u obou provozovatelů sítě. Základní adresa API pro síť LoRaWAN od ČRa je https://api.iot.cra.cz/cxf/api/v1/ a pro Sigfox je to https://api.sigfox.com/v2/. Jelikož pracujeme u obou API s typem REST, pak se za poslední uvozovku, vždy přidá volaný koncový bod, dle dokumentace API. Cílem je získat odpověď z API provozovatelů s http kódem začínajícím číslovkou dva, protože to znamená úspěšně zpracovaný požadavek a validní odpověď čili došlo k úspěšně k autentizaci a autorizaci přístupu a požadavek měl validní formát a získali jsme tak očekávanou odpověď.

4.4.3 Mikroslužby

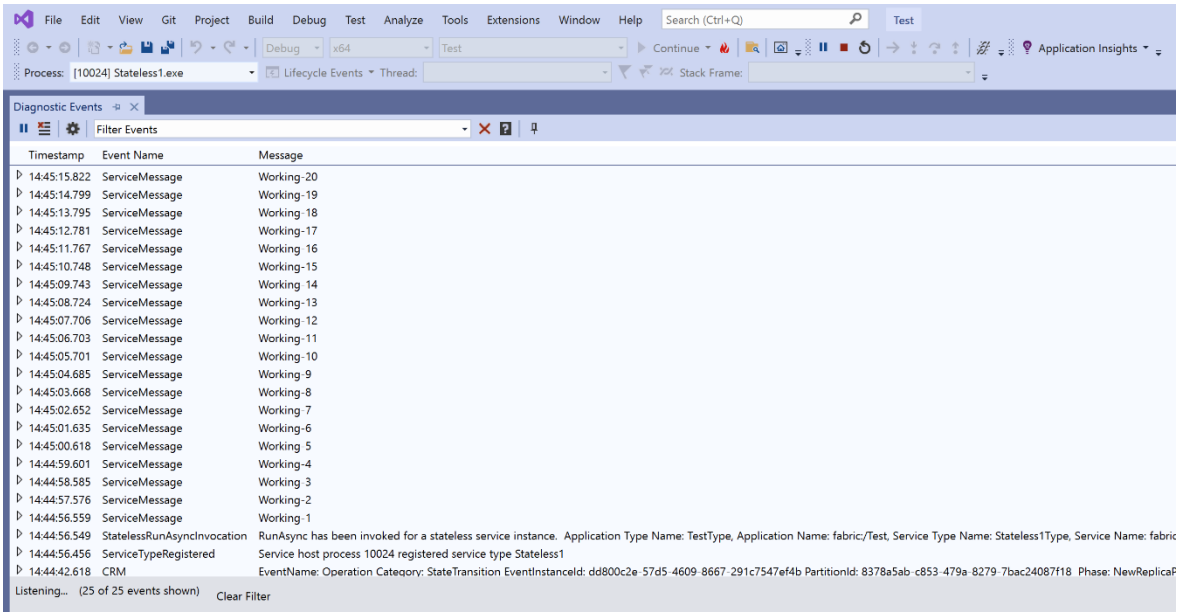
Vývoj vlastních služeb byl realizován na platformě .NET Core a v prostředí Azure, konkrétně na platformě Microsoft Service Fabric. Hlavním kritériem výběru této platformy byly reference, protože sám Microsoft provozuje na platformě Service Fabric vlastní klíčové aplikace, viz teoretická část této práce.

4.4.3.1 Příprava vývojového prostředí

Service Fabric se plně zaměřuje na implementaci a provoz aplikací postavených na bázi architektury mikroslužeb. Systém nasazený v rámci Service Fabric lze provozovat v cloudu, nebo na vlastní infrastruktuře. Obě prostředí ať už lokální nebo cloudové jsou naprosto identické, spouští se ze stejné verze kódu. Lokální vývojové prostředí tak je schopné výborně odladit vyvíjenou aplikaci a připravit ji na ostrý provoz v prostředí cloudu.

Vytvoření vlastního vývojového lokálního prostředí vyžaduje dvě klíčové komponenty. První je integrované vývojové prostředí (IDE), ve kterém budeme vytvářet požadované aplikace. Je podporováno široké množství IDE, jako JetBrains Rider, Visual Studio Code, Notepad++ nebo další populární editory. Z pohledu integrace se jako optimální jeví Microsoft Visual Studio 2019, které má integrované nástroje Service Fabric. Když vytváříme nový projekt ve Visual Studiu 2019 a vybere project type cloud, měli bychom mít k dispozici šablonu Service Fabric Application a pokud tomu tak není, pak je nutné tyto nástroje do Visual Studia přidat přes installer. Druhou důležitou komponentou je lokální instalace komponenty Service Fabric SDK (software development kit). Jsou podporované operační systémy Windows a Linux. Pro MacOS lze využít kontejnerového řešení Docker. Pro vlastní práci byla vybrána optimální varianta, tedy instalace Service Fabric SDK na operační systém z rodiny Windows. Po instalaci Service Fabric runtime a SDK na operační systém Windows byl celý cluster ponechán ve výchozím nastavení, z čehož plyne, že cluster provozuje jeden node. Tento node reprezentuje jeden virtuální počítač s nainstalovanou aplikací Service Fabric runtime, která bude hostovat vlastní vytvářenou aplikaci v rámci této práce.

Připravenost prostředí pro vývoj byla ověřena vytvořením stateless aplikace, která neuchovává data, na základě šablony ve Visual Studiu. K vývoji aplikací na lokálním clusteru je vždy nutné spouštět Visual Studio s oprávněním správce, což je volba, která je dostupná přes pravé tlačítko v rámci kontextové nabídky v navigaci operačního systému. Příčinou je, že Service Fabric cluster na lokálním stroji využívá ke svému běhu systémový účet NetworkService, který má omezená práva a pro potřeby ladění programu jsou potřeba práva vyšší. Připravená šablona byla bez jakýchkoliv úprav přímo spuštěna. Čímž došlo k sestavení aplikace a vytvoření lokální instance clusteru. O výsledku testu připravenosti prostředí nás informuje Diagnostic Events log, viz obrázek níže. Z obrázku je zřejmé, že prostředí je připravené pro vývoj a implementaci mikroslužeb.



Obrázek 4.10 - Service Fabric implementace lokálního clusteru

4.4.3.2 Implementace mikroslužeb

Service Fabric je prostředí přímo určené pro vývoj a provoz aplikací s architekturou mikroslužeb. Pro vlastní tvorbu byl vybrán programátorský přístup „Reliable Services“. Celé řešení se skládá ze tří „Reliable Service“ služeb, které demonstrují vlastnosti a schopnosti Service Fabric clusteru v oblasti tvorby distribuovaných systémů. Níže v popisu jednotlivých služeb se zaměřím na ty části kódu, které demonstrují, jak služby mezi sebou komunikují a jak pracují s daty. K vývoji aplikací byl použit programovací jazyk C#.

Z kolika mikroslužeb se vlastní aplikace bude skládat bylo definováno v rámci návrhu ve výše uvedených kapitolách, zde k těmto jednotlivým službám je přiřazena šablona na základě účelu služby:

- Služba API pro komunikaci s klienty jako rozcestník na další mikroslužby v rámci aplikace. Z pohledu vývoje je realizovaná jako stateless „Reliable“ služba, která neukládá data, neuchovává stavy.
- Služba DeviceManager spravuje a uchovává informace o připojených zařízeních. Bylo vybráno řešení statefull „Reliable“ služba, která trvale ukládá informace o jednotlivých zařízeních a využívá k tomu vlastní lokální databáze, kterou ukládá na lokální diska, což je vlastnost Reliable Service.

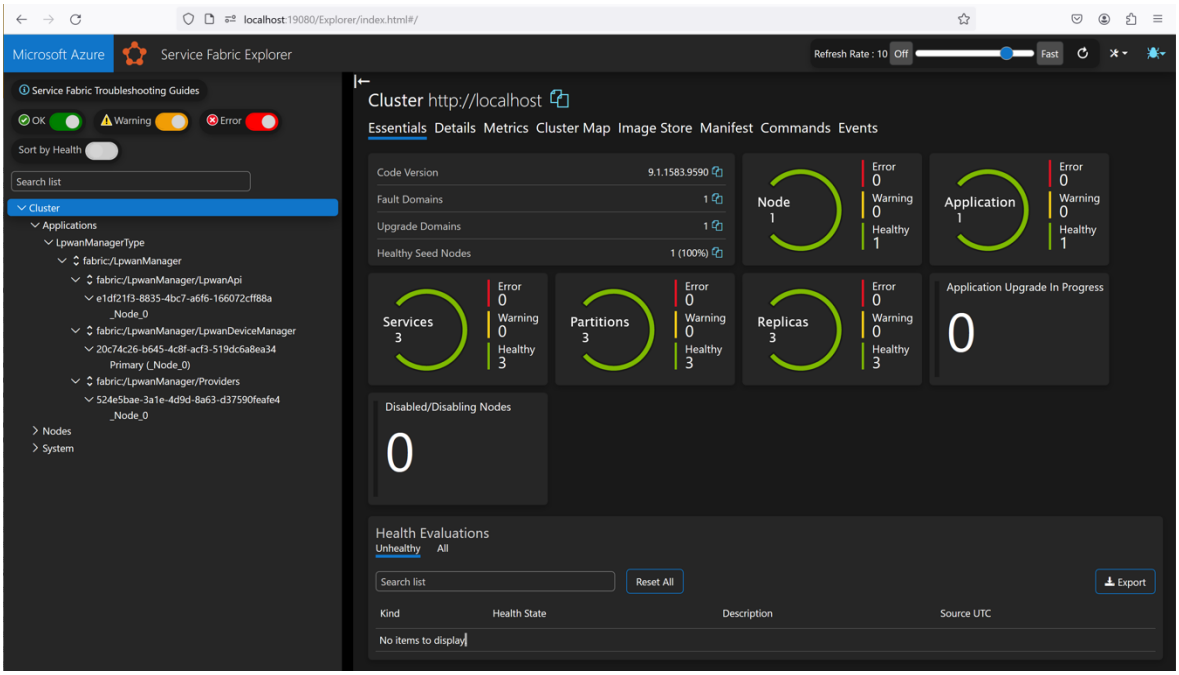
- Služba Providers provozuje klienty, kteří zprostředkovávají komunikaci směrem k API provozovatelů LPWAN sítí. Není třeba uchovávat stavy, služba byla implementována jako stateless.

Volba, která služba bude stavová a která stavy neuchovává je důležitá nejen z pohledu uchování dat, ale je důležité brát i ohled na způsob práce clusteru s danými službami. Služby, které neuchovávají stavy jsou v rámci clusteru na všech nodech aktivní a vlastní logiku clusteru rozhodne, který node vypořádá požadavek. U stavových služeb je pouze jedna na jednom node aktivní a kopie na ostatních nodech jsou v pasivním režimu, aby byla aplikace odolná vůči výpadkům. Tyto faktory byly zohledněny při vývoji aplikace. Níže jsou popsány dva scénáře, které demonstrují jak cluster pracuje s různými typy služeb.

První scénář je, že na službu API přijde požadavek na přidání zařízení do sítě. API kontaktuje službu DeviceManager, služba DeviceManager kontaktuje službu Providers a po úspěšném importu zařízení do sítě poskytovatele uloží služba DeviceManager záznam do databáze na lokální disk. Když cluster rozhoduje, jaký node zvolit pro obsluhu požadavku na straně API, vybere vždy ten node, kde je aktivní stavová služba DeviceManager. Což může být potenciálním úzkým hrdlem.

Druhý scénář je, že na službu API přijde požadavek na ověření dostupnosti signálu pro zadanou lokalitu na základě GPS souřadnic. API kontaktuje službu Providers. Komunikace se odehrává pouze mezi stateless službami čili cluster si může dynamicky rozhodnout na základě vlastní logiky, jaký node pro zpracování požadavku vybere, a to jak pro službu API, tak zvlášť pro službu Providers.

Na obrázku číslo 4.11 je zobrazena plocha pro správu a sledování nasazených mikroslužeb v rámci clusteru. Každá služba obsahuje kód, konfigurace a stavové služby k tomu uchovávají data. Konfigurace služeb je možné realizovat buď v Service Fabric Exploreru nebo přímo v projektu pomocí integrovaného vývojového prostředí, například Visual Studio.



Obrázek 4.11 - Service Fabric Explorer

Cluster a jeho jednotlivé služby se konfigurují skrze Manifest. Manifest clusteru je dostupný přes Service Fabric Explorer a zde bylo zkontrolováno, že reversní proxy v rámci clusteru má nastavený port na hodnotu 19081. Je to výchozí port a výhodou tohoto nastavení je, že pak není nutné uvádět specifický port v konfiguraci reverzní proxy ve vybraných objektech u jednotlivých služeb.

Každá služba obsahuje konfiguraci a kód a stavová služba ještě ukládá data. Důležité části kódu, jak stavová služba pracuje s daty a kde lze konfigurovat vlastnosti služby popíšu níže vždy konkrétně k vybrané službě.

4.4.3.3 Služba DeviceManager

Stavová služba a konzolová aplikace, která obstarává správu zařízení a ukládá informace do databáze na disku. K ukládání dat využívá kolekci Reliable Dictionary, což z pohledu .net programátora je dobře známá kolekce slovník. Tato podobnost zjednodušuje práci s daty.

```
IReliableDictionary<Guid, Device> devices = await
_stateManager.GetOrAddAsync<IReliableDictionary<Guid,
Device>>("devices");
```

Data jsou ukládána na lokální disk, což činí ze Service Fabric výkonné řešení pro práci s daty. Bezpečnost z pohledu ztráty dat je zajištěna replikami a výkon lze optimalizovat rozdělením databáze na části zvané partitions. V případě vlastní práce nebylo potřeba rozdělovat databázi na více částí, protože se pracuje s malým objemem dat, který slouží hlavně k demonstraci práce s mikroslužbami. Přístup k databázi a práci s daty v ní uloženými zprostředkovává ve službě DeviceManager vlastní třída ServiceFabricDeviceRepository, která v rámci konstruktoru implementuje rozhraní IReliableStateManager.

```
public ServiceFabricDeviceRepository(IReliableStateManager
stateManager) => _stateManager = stateManager;
```

Zjednodušeně lze konstatovat, že vytvoření instance IReliableStateManager v rámci konstruktoru je postačující pro plnohodnotný přístup k lokálním datům a práci s nimi. Není třeba otevírat šifrované síťové spojení, autentizovat se vůči databázovému serveru a dotazovat relační nebo grafovou databázi. Je to nejrychlejší možná práce s daty, protože využívá jen lokálního disku a data jsou organizována jako objekty z objektově orientovaného paradigmatu. Všechny operace s daty v rámci Service Fabric vyžadují práci s transakcemi a pracují na stejném principu jako dobře známé transakce z databází nebo ze systémů zpracování zpráv (např. Kafka).

```
using (ITransaction tx = _stateManager.CreateTransaction())
{
      await devices.AddOrUpdateAsync(tx, newDevice.Id, newDevice,
      (id, value) => newDevice);
      await tx.CommitAsync();
}
```

Service Fabric zpracuje trvale data jen když dojde k potvrzení transakce. Pokud v průběhu zpracování dat se vyskytne chyba, data nebudou aktualizována nebo uložena a dojde k odmítnutí požadavku. Pokročilejší práci s daty a jejich asynchronní zpracování nabízí implementovaný návrhový vzor Actor Model v Service Fabric, který z důvodů rozsahu práce nebyl implementován v rámci vlastního řešení. Je to alternativa k systémům asynchronního zpracování zpráv, jakými jsou například Kafka, Nats a mnoho dalších.

Stejně jako Kafka nebo Nats zajišťují asynchronní komunikaci mezi mikroslužbami, tak Service Fabric využívá vlastního runtime prostředí pro výměnu zpráv. Ze tří standardních protokolů WCF, http nebo Service Remoting, byl pro komunikaci mezi vytvořenými službami

vybrán výchozí protokol, kterým je Service Remoting a do projektu ho nainstalujeme jako knihovnu „Microsoft.ServiceFabric.Services.Remoting“.

```
protected override IEnumerable<ServiceReplicaListener>
CreateServiceReplicaListeners()
{
   return new[]
   {
     new ServiceReplicaListener(context =>
     new FabricTransportServiceRemotingListener(
     context, this, new FabricTransportRemotingListenerSettings
      {
           ExceptionSerializationTechnique =
           FabricTransportRemotingListenerSettings
           .ExceptionSerialization.Default}), "ServiceEndpointV2")};}
```

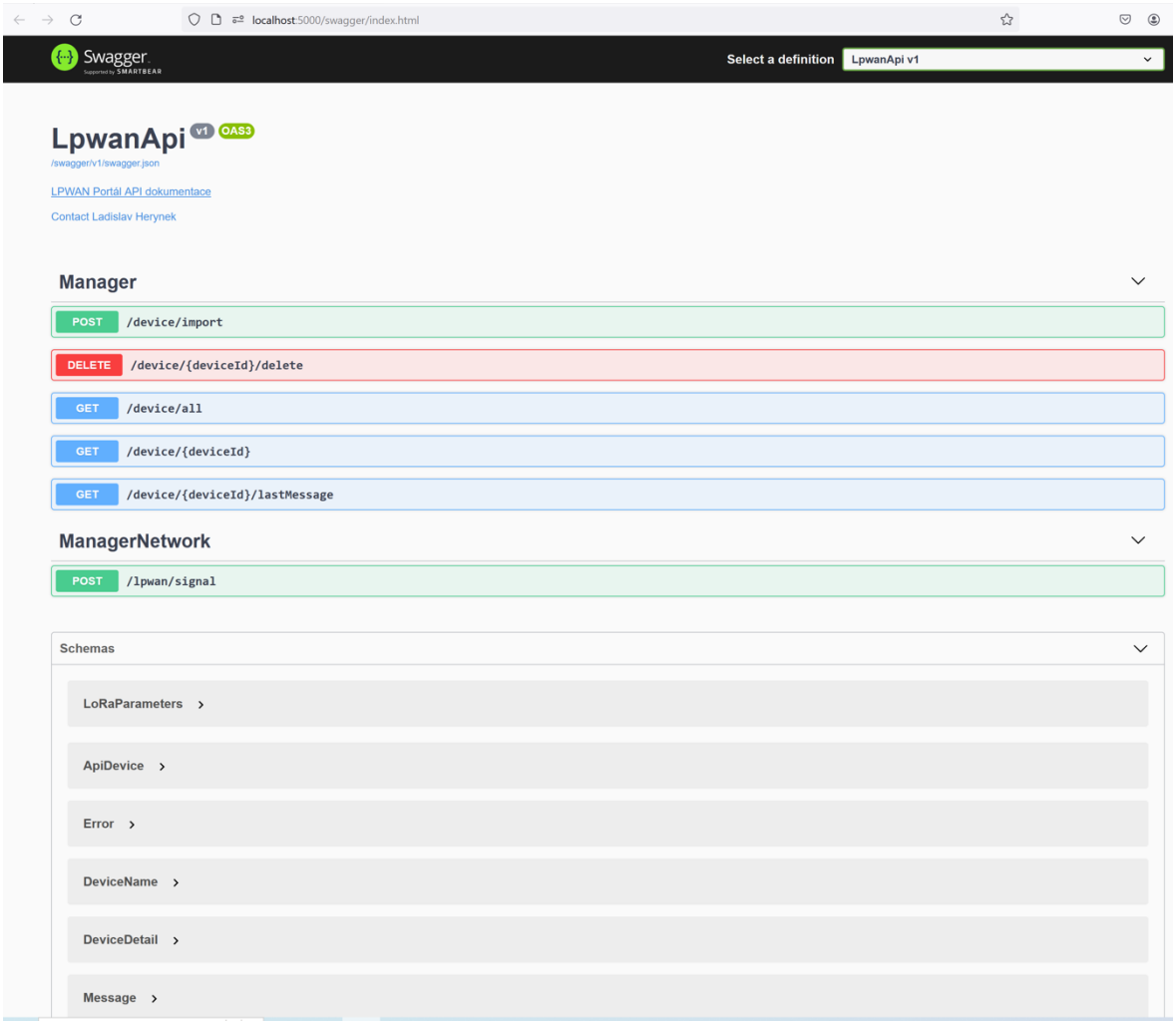
Vstupní branou do implementované služby je třída, které se jmenuje stejně jako služba. V případě této práce se třída jmenuje DeviceManager a výše uvedený kód je součástí této třídy a vytváří takzvaného posluchače pro komunikaci s dalšími službami v rámci clusteru, protože ve výchozím nastavení není nastaven žádný posluchač. Tato třída také určuje chování a vlastnosti služby, což činí metodami nebo vlastnostmi, které jsou zpřístupněné prostřednictvím přístupového modifikátoru public. Příklad takové metody, která je součástí třídy DeviceManager je uveden níže.

```
public async Task<Device[]> GetAllDevicesAsync()
{
     return (await _repo.GetAllDevicesAsync()).ToArray();
}
```

Posluchače nastavujeme na všech službách, které nějakým způsobem interagují s dalšími službami v rámci clusteru. Výše uvedený kód inicializoval jednoho posluchače pro stavovou službu. Stateless služba má podobný postup, ale odlišuje se deklarací výsledného objektu, kde výsledným objektem není ServiceReplicaListener, ale ServiceInstanceListeners. Parametry vytvářeného objektu se liší případ od případu, jiné má konzolová služba a jiné má služba vycházející ze šablony .Net Core API.

4.4.3.4 Služba API

Služba API je branou do ohraničeného kontextu „Správa IoT prvků“. Přijímá, zpracovává a distribuuje požadavky klientů a prezentuje odpovědi. Pro dokumentaci chování nebo vlastností API byla využita OpenAPI specifikace (Swagger), viz obrázek číslo 32. Aby služba mohla přijímat požadavky externích klientů, byl za tímto účelem stejně jako u výše uvedené služby implementován posluchač. Rozdíl oproti službě DeviceManager je v tom, že posluchač iniciuje jiný objekt (CreateServiceInstanceListener) a neuchovává stavy, jedná se tedy o službu typu stateless.



Obrázek 4.12 - OpenAPI specifikace

API distribuuje příchozí požadavky vůči dalším službám clusteru k jejich dalšímu zpracování. Dle typu požadavku buď na službu Providers, nebo na službu DeviceManager. Za tímto účelem byla v obou kontrolérech inicializována vestavěná reverzní proxy clusteru. Níže zveřejněný kód inicializuje proměnnou _service vůči službě Providers.

```
public ManagerNetworkController()
{
   var proxyFactory = new ServiceProxyFactory(c => new
   FabricTransportServiceRemotingClientFactory());
     _service = proxyFactory.CreateServiceProxy<IProviderService>(new
      Uri("fabric:/LpwanManager/Providers"));
}
```

Dříve jsem zmínil, že kromě kódu každá služba obsahuje ještě konfigurace. Konfigurace jsou dostupné skrze manifest služby, který je uložen v souboru ServiceManifest.xml v adresáři PackageRoot. Pro zjednodušení přístupu klientů k API byla změněna konfigurace vstupního portu na hodnotu 5000. Na tomto portu služba poslouchá příchozí požadavky.

4.4.3.5 Služba Providers

Cílem je zprostředkovávat komunikaci s externími kontexty, které představují poskytovatele podporovaných LPWAN sítí. Je to stateless služba, která neuchovává stavy a obsahuje standardní http klienty pro připojení k API třetích stran, na které jsou zasílány požadavky například na připojení zařízení, zjištění síly signálu nebo získání zpráv ze zařízení. Níže uvádím příklad http klienta, který je součástí metody pro ověření síly signálu sítě LoRa.

```
var token = await GenerateTokenAsync(cancellationToken);
_httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer
{token.AccessToken}");
string url = $"{baseUrl}/lora/signal";
var jsonContent = new StringContent(JsonSerializer.Serialize(gps),
Encoding.UTF8,  "application/json");

var response = await _httpClient.PostAsync(url, jsonContent,
cancellationToken);
```

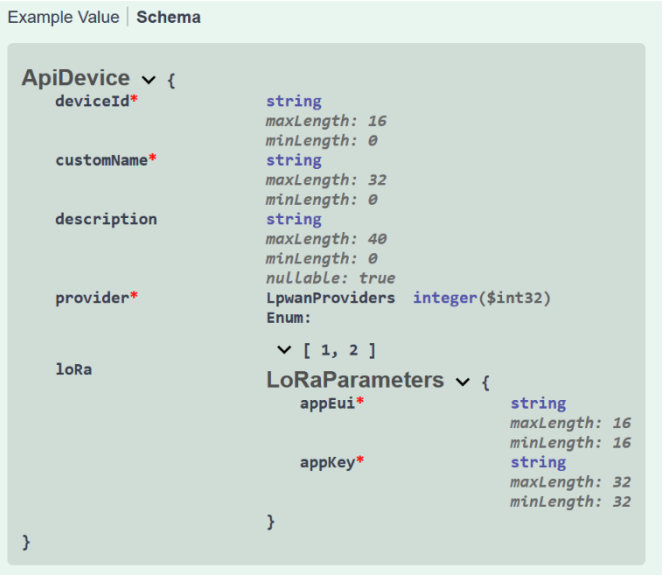
Služba byla realizována jako poslední z řady, a tak se největším problémem k řešení stala optimalizace komunikace tak, aby si v rámci clusteru všechny služby vzájemně nekonkurovaly a bez výpadkově spolu komunikovaly. Tento problém řeší konfigurace posluchačů ve třídě, která má shodný název s názvem služby.

4.4.4 Testování systému

V rámci vývoje minimální funkční aplikace (MVP) byly realizovány funkce pro import zařízení, odstranění zařízení, výpis všech zařízení z databáze, detailní pohled na jedno vybrané zařízení, výpis zpráv z vybraného zařízení a ověření síly signálu. Všechny funkce jsou zobrazeny na obrázku číslo 32 a v průběhu jejich tvorby byly laděny pomocí vývojářských nástrojů, zdali v každém kroku, zejména při výměně informací mezi službami dochází k příjmu nebo odeslání očekáváného modelu dat (payloadu) a zdali logika jednotlivých funkcí pracuje správně se vstupy a vrací očekávané výstupy. K tomuto testování byly využívány nástroje vývojového prostředí Visual Studia a OpenAPI rozhraní Swagger.

Finální otestování aplikace bylo uskutečněno pomocí nástroje Postman, který slouží k testování, sestavování nebo práci s API. Tento všeobecně známý nástroj dobře posloužil pro otestování vstupů a výstupů jednotlivých koncových bodů vytvořeného API vlastní aplikace. Byly otestovány vstupy a výstupy aplikace. Na základě tohoto testu by byl v rámci týmu definován další postup ve vývoji aplikace. Výstup testů by měl vliv na definování rozvojových aktivit a plán následujícího sprintu.

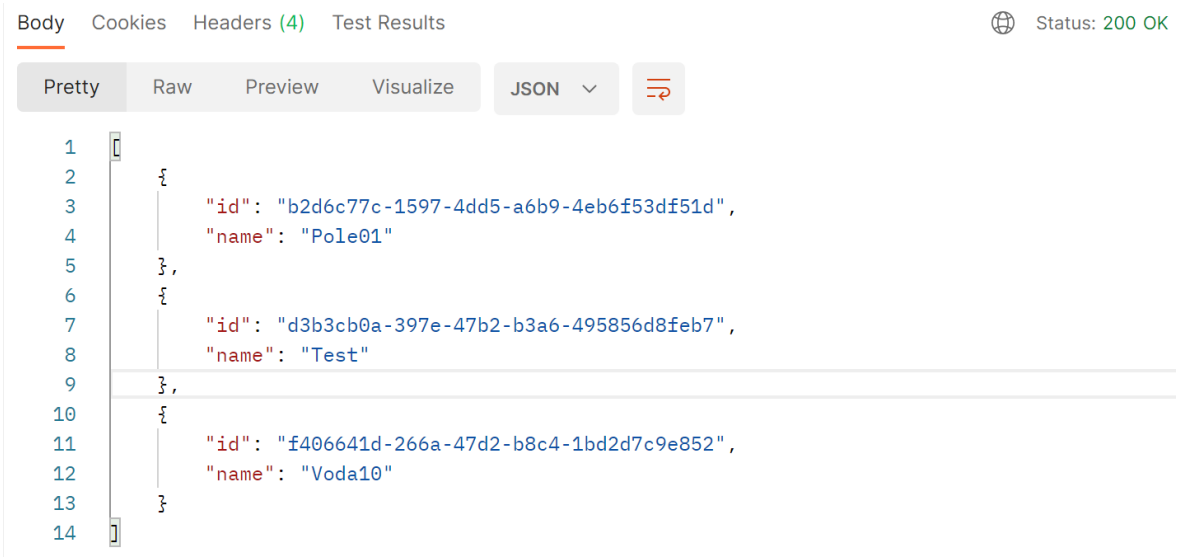
Funkce POST **/device/import** ukládá zařízení do vlastní databáze pro evidenci vlastních zařízení v různých podporovaných sítích a do těchto sítí současně provede import požadovaného zařízení. Obrázek číslo zobrazuje strukturu položek včetně anotací, které definují mandatorní atributy a omezení na vkládané řetězce. Testy bylo identifikováno několik problémů k řešení. První test kombinoval nevhodné řetězce s prázdnými atributy nebo objekty a aplikace na tyto dotazy vždy reagovala chybovým stavem s http kódem 500. Správná varianta je http kód 400 (Bad request) chyba na straně zadavatele požadavku. Další testy probíhali se správnými hodnotami dotazu a bylo zjištěno, že nejsou správně zpracovávány chybové stavy API třetích stran. Rozumějte, vlastní



Obrázek 4.13 - Import zařízení struktura dotazu

aplikace provedla úspěšně import zařízení, záznam byl uložen do vlastní databáze, byla získána http odpověď s kódem 200, ale na straně poskytovatele import skončil chybovým stavem s http kódem 422 (Unprocessable Entity).

Metoda **GET /device/all** vypisuje seznam všech zařízení, která evidujeme ve vlastní databázi. Metoda nemá žádný vstupní parametr, tudíž z pohledu uživatele není, jak ovlivnit výsledek a průběh dotazu.



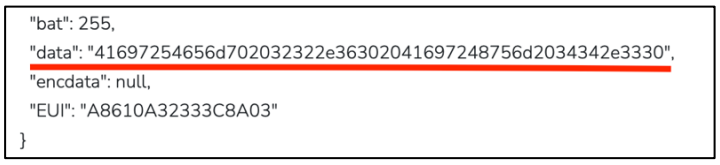
Obrázek 4.14 - Get /device/all

Metoda **GET /device/{id}** vypíše detailní informace o vybraném právě jednom zařízení. Dotaz lze ovlivnit pouze z pohledu nevhodně zadaného id parametru, který je typu Guid a je součástí url cesty. Test spočíval ve vkládání nesmyslných řetězců nebo prázdného řetězce se vždy uspokojivým výsledkem, který spočíval v získání http odpovědi s kódem 400 (Bad request) doprovázený popisem chyby jak „The value is not valid“.

Metoda **DELETE /device/{id}**/ odstraní vybrané zařízení z databáze a LPWAN sítě. Volání má jediný vstupní parametr, a to id typu Guid a výsledkem testu je zjištění, že při různých kombinacích vstupního parametru, který nebyl v souladu s id v databázi, byl vždy vrácen chybový stav s http kódem 400 (Bad request). Při správně zadaném id zařízení, bylo zařízení odstraněno jak z databáze, tak i ze sítě poskytovatele.

Get /device/{id}/message načte přijaté zprávy z IoT zařízení. Testování neodhalilo žádný neobvyklý nebo nový problém. Uživatel může pouze zadat id, které je typu Guid a je vstup ošetřen, tak aby nebylo možné zadat jiný formát řetězce. Níže dvě obrazovky dokazující úspěšné načtení informací. Data jsou šifrována, k dešifrování je nutné implementovat dekodér, který není součástí rozsahu řešení.

Obrázek 4.16 - Postman výsledek dotazu Get Message



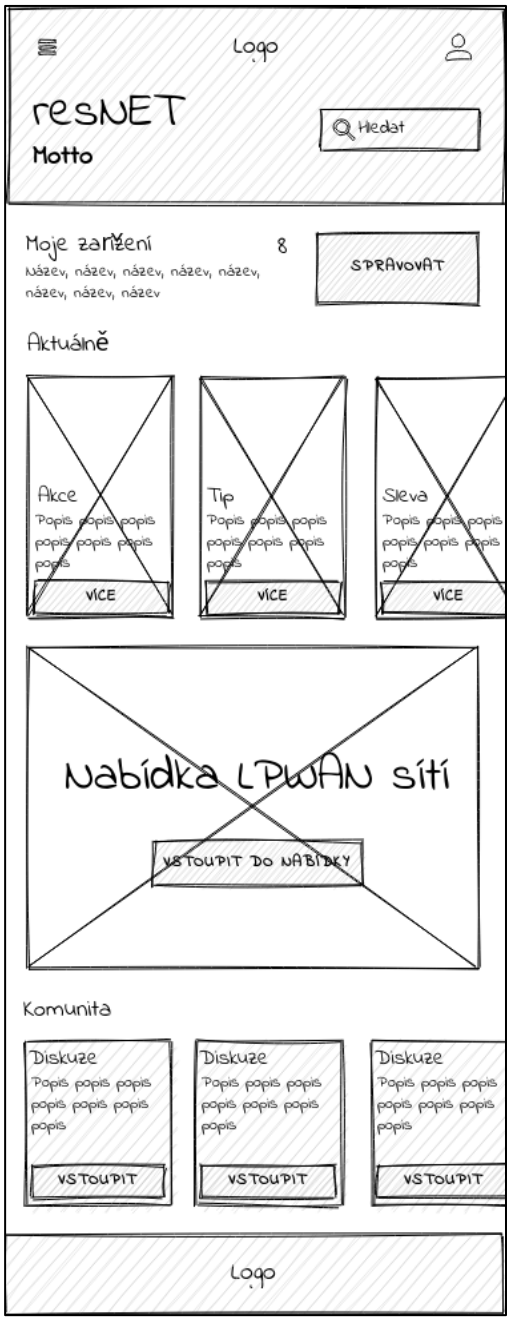
Obrázek 4.15 - Výsledek dotazu Get Message v síti ČRa

Souhrnně lze napsat, že další fáze vývoje aplikace by spočívala v odstranění identifikovaných chyb nebo implementaci definovaných zlepšeních, které spočívají například v:

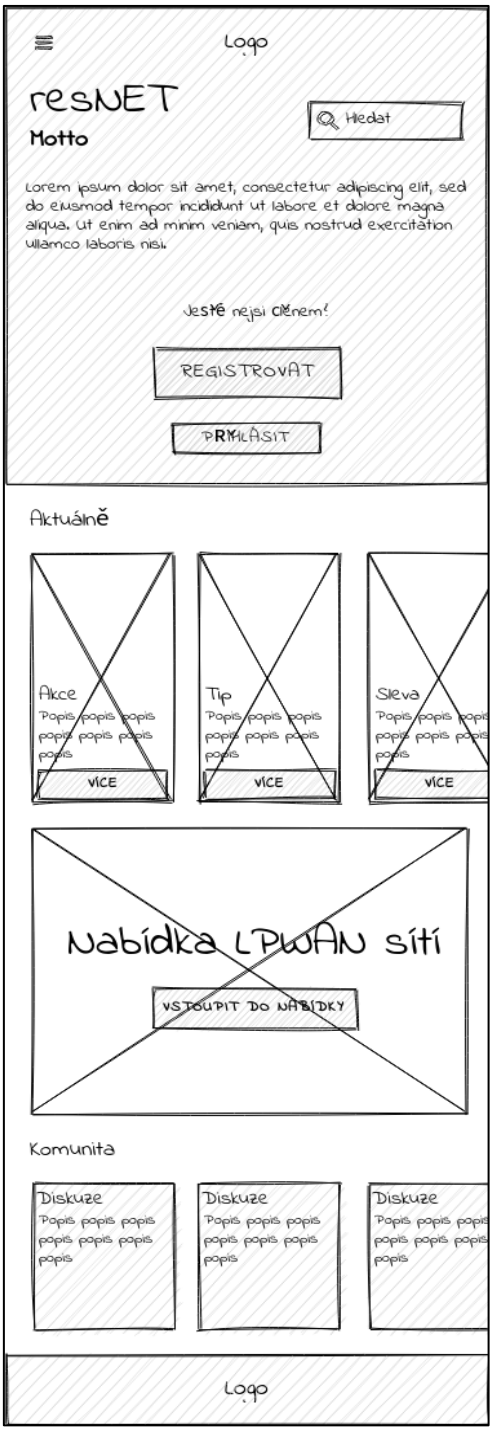
- Zlepšení dokumentace koncových bodů a atributů v OpenAPI.
- Zpracování chybových událostí z API třetích stran,
- Minimalizace počtu chybových stavů s http kódem 500.

4.4.5 Prezentační vrstva, strana klienta

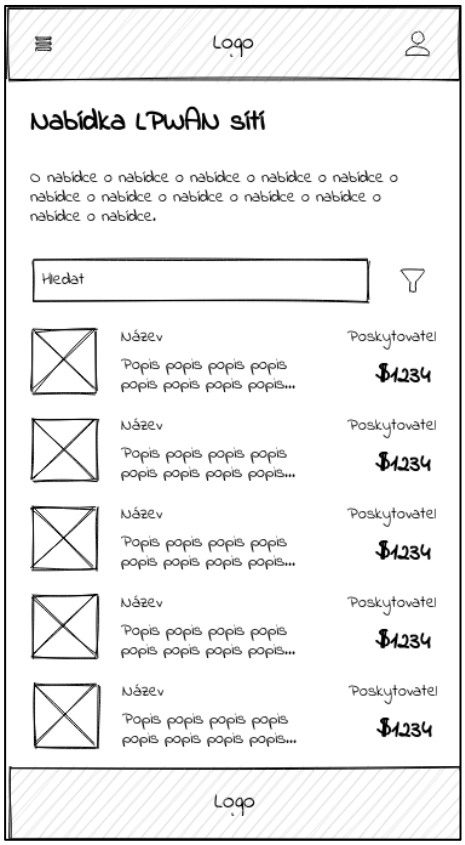
V této kapitole je znázorněn způsob zpracování front-end vrstvy, tedy klientské strany, která prezentuje entity a agregáty (data) z ohraničeného kontextu „Správa IoT prvků“, protože součástí návrhového vzoru DDD je drátový návrh (wireframe) prezentační vrstvy. V kapitole Event Storming byly identifikovány agregáty nebo entity, které byly zakresleny do níže přiložených obrazovek zpracovaných jako drátové modely. Velikost ovládacích prvků je uzpůsobena snadnému ovládání na menší dotykové obrazovce, protože k vývoji klienta bylo přistoupeno metodou „mobile-first“. Což je princip, který spočívá v návrhu obrazovek od menších k větším. Veškerý obsah na stránce je řazen pod sebou v jednom vertikálně scrollovacím prostoru, v němž na vybraných místech je aplikováno i horizontálně scrollovací pole s dlaždicemi. Postranní menu nebude pevně umístěno na stránce, ale bude vyvolatelné tlačítkem vlevo nahoře, a to i ve verzi pro větší obrazovky.



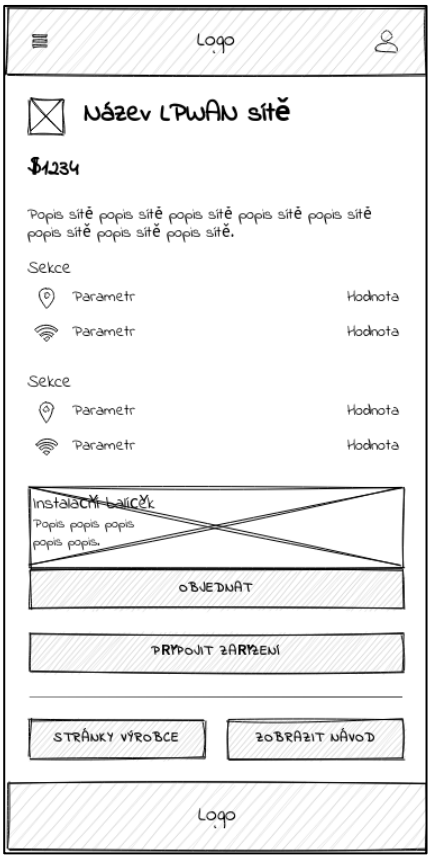
Obrázek 4.18 - WireFrame HomePage



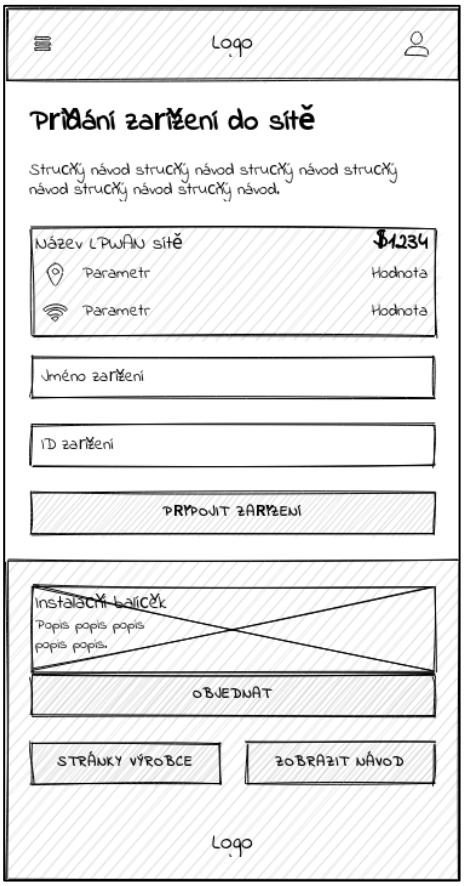
Obrázek 4.17 - WireFrame HomePage rozbalené Menu



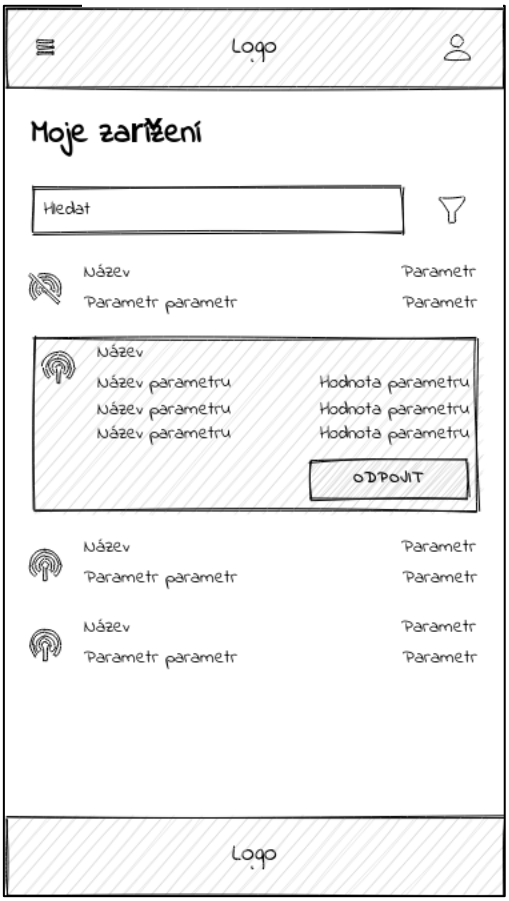
Obrázek 4.21 - WireFrame Seznam sítí



Obrázek 4.20 - WireFrame Detail sítě



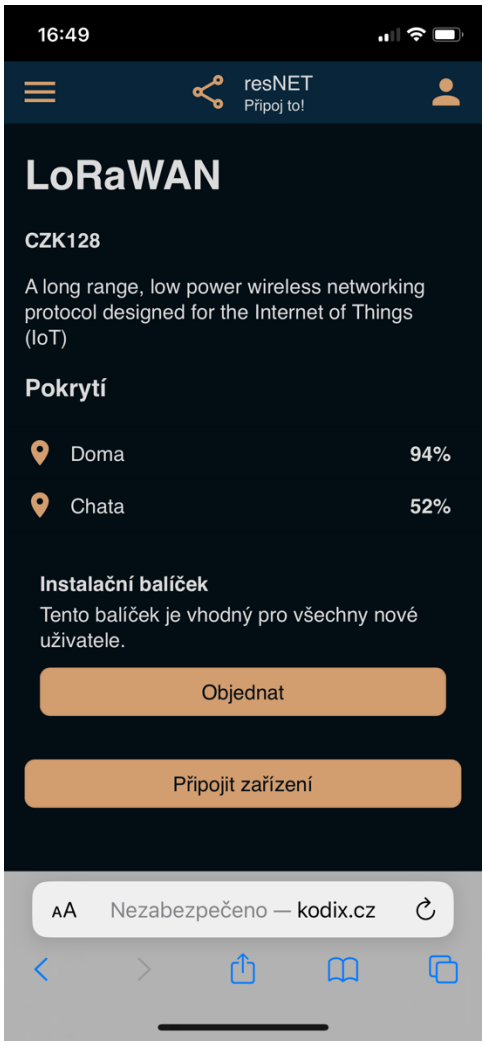
Obrázek 4.22 - WireFrame Přidání zařízení



Obrázek 4.19 - WireFrame Detail zařízení

Všechny vymodelované obrazovky byly implementovány prostřednictvím frameworku React, který pracuje s jazykem JavaScript. Bylo vytvořeno jednostránkové řešení, celá prezentace je přístupná přes jedinou stránku index.html, s odkazy na substránky realizovanými prostřednictvím routingu uvnitř React aplikace. Níže přikládám příklady dvou obrazovek

Obrázek 4.24 - Homepage



Obrázek 4.23 - Detail sítě

5 Výsledky a diskuse

5.1 Hodnocení minimální funkční aplikace (MVP)

Vlastní práce je postavena na třech pilířích. Na analýze, která podrobně definuje zadání a využívá k jeho prezentaci nástroj Business Canvas. Druhým pilířem je návrh, je to stěžejní část práce, která k návrhu systému využívá moderních návrhových vzorů, které komplexní problémy dekomponují na menší části a které jsou v souladu s principy objektového paradigmatu a výpočetní modelu cloud computing. Konkrétně se jedná o návrhové vzory Domain Driven Design a mikroslužby. Oba návrhové vzory spolu úzce souvisí, protože DDD je vstupní branou pro návrhový vzor mikroslužeb. DDD rozděluje velkou a složitou oblast problému z jedné rozsáhlé domény na několik menších ohraničených kontextů, které izolují subdomény a definují tak prostor pro optimální identifikaci mikroslužeb a transparentní vymezení odpovědnosti vůči vývojovým týmům, protože platí pravidlo, že na jedné službě může pracovat pouze jeden tým, ale jeden tým může vytvářet a provozovat více služeb. Oba návrhové vzory se doplňují a jsou v souladu s principy cloud computingu a využívají tak jeho předností. Závěr vlastní práce se věnuje realizaci jednoho vybraného ohraničeného kontextu s definovaným minimálními funkčními požadavky. K realizaci, což je třetí pilíř vlastní práce, byla vybrána cloudová platforma Microsoft Service Fabric, kdy hlavním výběrovým parametrem byla symbióza s výše uvedenými architektonickými návrhovými vzory. Service Fabric je řešení plně podporující vývoj a provoz mikroslužeb, kdy sám Microsoft na tomto systému provozuje řadu svých klíčových aplikací. Vývoj byl realizován na platformě .NET s programovacím jazykem C#. Vývoj software z pohledu DDD nezačíná návrhem datové struktury a tento princip Service Fabric plně podporuje, kdy pro trvalé uložení dat bylo využito jeho vlastních datových kolekcí a vybraného typu služby. Principiálně každá provozovaná služba, která uchovává stavy, ukládá data na lokální disk a využívá k tomu kolekce, které jsou plně v souladu s objektovým paradigmatem. Pro otestování funkčnosti systému byly sestrojeny dva senzory, komunikující prostřednictvím sítě LoRa od Českých radiokomunikací nebo Sigfox ČR. Obě sítě pokrývají téměř celé území ČR kvalitním signálem a mají zveřejněné zdokumentované aplikační rozhraní. Testování aplikace odhalilo řadu chyb a podnětů pro zlepšení, což z pohledu vývoje software jsou úkoly do dalších fází vývoje, které by byly řešeny v rámci nového sprintu, protože vývoj softwaru je nepřetržitý proces odstraňování chyb a zlepšování funkcí vyvíjené aplikace.

5.1.1 Silné stránky MVP

Identifikace optimálního množství mikroslužeb a jejich ohraničení z pohledu řešeného problému jsou dva důležité problémy návrhu, a to je první silná stránka vlastní práce, která využila návrhového vzoru Domain Driven Design k identifikaci mikroslužeb a dekompozici složitého problému na menší části, které lze pak rozdělit mezi jednotlivé týmy k řešení. Druhou výhodou DDD je, že vývoj není řešení od návrhu dat, nýbrž se začíná obchodní analýzou, rozumějte na začátku probíhá intenzivní interakce mezi doménovými experty a softwarovými architekty nebo vývojáři. Je to přístup, který považuji za výhodu, protože dochází k vzájemnému porozumění, které vede k optimálnímu návrhu softwaru z pohledu očekávání a možností.

Druhou významnou silnou stránkou vlastní práce je technologie Microsoft Service Fabric, která nabízí řadu technických i návrhových přístupů, které usnadňují nasazení a vývoj aplikace nebo práci s daty. Service Fabric ukládá data na lokální disk virtuálního počítače, který provozuje námi vyvíjenou aplikace. Je to nejrychlejší možný způsob práce s daty, protože není třeba utvářet žádné síťové spojení a prostřednictvím sítě komunikovat s databázovým strojem. Každá služba má nezávislá data a proti jejich ztrátě se využívá principů distribuovaných systémů v podobě rozdělení dat na aktivní a pasivní části. Poslední výhodou Service Fabric, kterou je nutné zmínit jsou dva programovací přístupy, kterými jsou „Service Reliable“ a „Actor Model“. Silnou stránkou „Service Reliable“ jsou kolekce pro persistentní uložení dat a práci s nimi. Tyto kolekce jsou příbuzné kolekcím z jazyka C# (seznam, slovník) a jsou tak plně v souladu s objektově orientovaným paradigmatem. Pro programátora je práce s kolekcemi přirozená a činí návrh a vývoj datové struktury snazší. A pro asynchronní zpracování dat a vytváření paralelních úloh pro práci s daty se využívá „Actor Model“. Všechny tyto vlastnosti činí Service Fabric silnou platformou pro vývoj a provoz distribuovaných systémů nebo systémů postavených na bázi mikroslužeb.

5.1.2 Slabé stránky MVP

Slabou stránkou je šířka znalostní báze a podpora Microsoft Service Fabric. Podporou je myšleno integrace s vývojovými nástroji, která není rozsáhlá. Konkrétně lze zmínit, že existuje problém najít vhodný nástroj, který umožní zprovoznit lokální vývojové prostředí pro plnohodnotné testování vyvíjené distribuované aplikace, protože jediným podporovým IDE je Visual Studio 2019, které disponuje nástroji a šablonami Service Fabric.

Druhou slabou stránkou je šíře znalostí báze oproti jiným systémům. Neexistuje tolik zdokumentovaných postupů pro optimální konfiguraci clusteru nebo pro optimální konfiguraci jednotlivých služeb včetně ukázek kódů, které demonstrují řešení obvyklých i neobvyklých problémů. V této části byla vlastní práce značně odkázána na přístup pokus omyl. Tento styl vývoje vlastní práci poznamenal zejména při konfiguraci jednotlivých služeb, aby mezi sebou vzájemně komunikovali nebo při rozdělení uložených dat na menší části (partitions).

6 Závěr

Cílem práce bylo navrhnout prostřednictvím moderních návrhových vzorů informační systém, který bude zpracovávat data z oblasti IoT, spravovat připojená zařízení a vytvářet prostor pro IoT komunitu ke sdílení znalostí, postupů nebo zkušeností. Ze tří pilířů analýza, návrh a realizace byl důraz kladen hlavně na oblast návrhu, která spočívala v demonstraci moderního návrhového vzoru Domain Driven Design, který k vývoji software přistupuje odlišným způsobem než tradiční postupy, které začínají specifikací funkčních požadavků a návrhem datové struktury. DDD rozdělí komplexní problém na menší oblasti, které mají přesně definované hranice a svými základními nástroji definuje transparentní komunikaci mezi doménovými experty a vývojovými týmy. Je to nástroj, který přímo podporuje druhý použitý moderní návrhový vzor, konkrétně se jedná o mikroslužby.

Teoretická příprava spočívala v podrobném studiu LPWAN sítí, výpočetní modelu cloud computing a obou výše zmíněných návrhových vzorů. Řazení není náhodné a odpovídá návaznosti na mou bakalářskou práci, která pojednávala o IoT se zaměřením na senzorickou část. Stávající práce posunula poznání do aplikační roviny, kde všechny výše uvedené oblasti se vzájemně doplňují. IoT je komplexní oblast, která je náročná na síťovou konektivitu a zpracování velkého objemu dat. Očekává se, že IoT bude generovat velká datová jezera nebo datové sklady pro pozdější analýzu nasnímaných dat a zde je průnik do světa cloud computingu, kde je pro širokou veřejnost k dispozici velký výpočetní výkon s velkou datovou kapacitou v podobě sdílených datových center a jeho vybavení.

Téměř vše, co si lze ve světě informačních technologií představit lze poskytovat jako službu a čím vím monolitické řešení dekomponuji na menší službu tím přesnější je škálování zdrojů na nejvíce vytížené prvky systému jako celku. O optimálním rozdělení monolitické aplikace na menší části pojednává oblast návrhových vzorů, které byly zmíněny výše. Celá práce je o příběhu, jak z nápadu pomocí moderních trendů vytvořit konkrétní realizaci služby.

Práce naplnila očekávání, protože v jejím průběhu byla dokázána vzájemná symbióza klíčových prvků, což lze shrnout do věty, že IoT potřebuje cloud computing k efektivnímu zpracování velkého množství dat, cloud computing zpřístupňuje pokročilé nástroje pro provoz IoT prvků a pokročilé analytické nástroje pro vlastní zpracování dat a moderní návrhové vzory přímo podporují výpočetní model cloud computing, když rozbíjí komplexní problémy na menší

prvky, které jsou přesněji škálovatelné a vzájemně nezávislé. Toto tvrzení bylo ověřeno vlastní implementací systému, kdy na začátku bylo komplexní zadání a na konci realizace jedné jeho nezávislé části prostřednictvím .NET prostředí s jazykem C#. Klient pro uživatelskou interakci byl vytvořen v prostředí React s jazykem JavaScript.

Vnímám, že tato práce definuje další prostor pro návaznou činnost například v podobě disertační práce. Protože IoT oblast ve spojení s výpočetním modelem cloud computing nabízí velký potenciál pro strojové zpracování dat, rozumějte umělou inteligenci nebo data mining a cílem z dat získat poznání nebo informace.

7 Seznam použitých zdrojů

[1] EVANS, Dave. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. In: *Cisco White Paper* [online]. Amsterdam, The Netherlands: www.cisco.com, 2011 [cit. 2023-02-09]. Dostupné z: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] MINERVA, Roberto, Abyi BIRU a Domenico ROTONDI. Towards a definition of the Internet of Things (IoT). In: *Iot.ieee.org* [online]. Itálie: iot.ieee.org, 2015 [cit. 2020-05-04]. Dostupné z: https://iot.ieee.org/definition.html

[3] FOUAD, Mohamed, Nour OWEIS, Tarek GABER a Maamoun AHMED. Data Mining and Fusion Techniques for WSNs as a Source of the Big Data. In: *ResearchGate* [online]. online: ResearchGate, 2015 [cit. 2023-02-11]. Dostupné z: https://www.researchgate.net/publication/282848334_Data_Mining_and_Fusion_Techniques_for_WSNs_as_a_Source_of_the_Big_Data/link/561e31a008aecade1acb4bef/download

[4] Cellular networks for Massive IoT. In: *Ericsson.com* [online]. Europe: ericsson.com, 2020 [cit. 2023-02-11]. Dostupné z: https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-networks-for-massive-iot--enabling-low-power-wide-area-applications

[5] VAEZI, Mojtaba, Saeed KHOSRAVIRAD, Amin AZARI a Mahdi AZARI. Cellular, Wide-Area, and Non-Terrestrial IoT: A Survey on 5G Advances and the Road Towards 6G. In: *ResearchGate* [online]. online: ResearchGate, 2022 [cit. 2023-03-30]. Dostupné z: https://www.researchgate.net/publication/358523992_Cellular_Wide-Area_and_Non-Terrestrial_IoT_A_Survey_on_5G_Advances_and_the_Road_Towards_6G

[6] MIAO, Yiming. Narrow Band Internet of Things. In: *ResearchGate* [online]. online: ResearchGate, 2017 [cit. 2023-02-13]. Dostupné z: https://www.researchgate.net/publication/319869218_Narrow_Band_Internet_of_Things

[7] NB-IoT: Co všechno stojí za sítí, která dělá z obyčejné budovy chytrou. In: *Vodafone CZ* [online]. online: Vodafone, 2021 [cit. 2023-02-13]. Dostupné z: https://www.vodafone.cz/business-blog/internet-veci/nb-iot-co-vsechno-stoji-za-siti-ktera-dela-z-obyce/

[8] PAUL, Biswajit. An Overview of LoRaWAN. In: *ResearchGate* [online]. online: ResearchGate, 2021 [cit. 2023-02-15]. Dostupné z: https://www.researchgate.net/publication/348464014_An_Overview_of_LoRaWAN

[9] What are LoRa and LoRaWAN. In: *LoRa Developer Portal* [online]. online: LoRa Alliance, 2023 [cit. 2023-02-15]. Dostupné z: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/

[10] IoT síť LoRa od Českých Radiokomunikací zavádí mezinárodní roaming. In: *LUPA.cz* [online]. online: Internet Info, s.r.o., 2021 [cit. 2023-02-15]. Dostupné z: https://www.lupa.cz/aktuality/iot-sit-lora-od-ceskych-radiokomunikaci-zavadi-mezinarodni-roaming/

[11] PUMAMA, Arrizky. SigFox-based Internet of Things Network Planning for Advanced Metering Infrastructure Services in Urban Scenario. In: *ResearchGate* [online]. online: ResearchGate, 2020 [cit. 2023-02-14]. Dostupné z: https://www.researchgate.net/publication/343777129_SigFox-based_Internet_of_Things_Network_Planning_for_Advanced_Metering_Infrastructure_Services_in_Urban_Scenario

[12] *Sigfox support* [online]. online: sigfox.com, 2023 [cit. 2023-02-14]. Dostupné z: https://support.sigfox.com/docs

[13] *Sigfox 0G Technology* [online]. online: sigfox.com, 2023 [cit. 2023-02-14]. Dostupné z: https://www.sigfox.com

[14] BOJANOVA, Irena a San MURUGESAN. *Encyclopedia of Cloud Computing* [online]. 1st. USA: IEEE Press, 2016 [cit. 2023-02-18]. ISBN 9781118821954. Dostupné z: https://ebookcentral.proquest.com/lib/czup/reader.action?docID=4526670

[15] JAMSA, Kris. *Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More* [online]. 1st Edition. United States of America: Jones & Bartlett Learning, 2012 [cit. 2022-05-24]. ISBN 978-1449647391.

[16] *App Engine documentation* [online]. USA: Google, 2023 [cit. 2023-02-19]. Dostupné z: https://cloud.google.com/appengine/docs

[17] Structure of a Google App Engine application. In: *O'Reilly* [online]. USA: O'Reilly, 2023 [cit. 2023-02-19]. Dostupné z: https://www.oreilly.com/library/view/google-cloud-platform/9781788837675/a10eee42-5613-463e-8bd0-d76dd8ef6216.xhtml

[18] *AWS Documentation* [online]. USA: Amazon, 2023 [cit. 2023-02-20]. Dostupné z: https://docs.aws.amazon.com/?nc2=h_ql_doc_do

[19] *Azure* [online]. USA: Microsoft, 2023 [cit. 2023-02-20]. Dostupné z: https://azure.microsoft.com/cs-cz/

[20] *Dokumentace k Azure* [online]. USA: Microsoft, 2023 [cit. 2023-02-20]. Dostupné z: https://learn.microsoft.com/cs-cz/azure

[21] Azure Service Fabric documentation. In: *Microsoft Learn* [online]. USA: Microsoft, 2022 [cit. 2023-03-30]. Dostupné z: https://learn.microsoft.com/en-us/azure/service-fabric/

[22] DUSTDAR, Patricia LAGO, Manuel MAZZARA a Victor RIVERA. *Microservices Science and Engineering* [online]. 1st. Switzerland: Springer, 2020 [cit. 2023-03-30]. ISBN 978-3-030-31646-4. Dostupné z: https://arxiv.org/abs/1706.07350

[23] *Microsoft Learn* [online]. USA: Microsoft, 2023 [cit. 2023-03-30]. Dostupné z: https://learn.microsoft.com/en-us/training/

[24] *Microservices* [online]. online: Martin Fowler, 2014 [cit. 2023-03-30]. Dostupné z: https://martinfowler.com/articles/microservices.html

[25] KUHN, Roland, Brian HANAFEE a Jamie ALLAN. *Reactive Design Patterns* [online]. 1st. online: O'Reilly, 2017 [cit. 2023-03-30]. ISBN 9781617291807. Dostupné z: https://www.oreilly.com/library/view/reactive-design-patterns/9781617291807/

[26] GHOSH, Debasish. *Functional and Reactive Domain Modeling*. 1st ed. United States of America: Manning, 2017. ISBN 9781617292248.

[27] VERNON, Vaughn. *Domain-Driven Design Distilled*. 1st. USA: Pearson Education, 2016. ISBN 978-0-13-443442-1.

[28] MERHOLZ, Peter a Kristin SKINNER. *Org Design for Design Orgs* [online]. 1st. USA: O'Reilly, 2016 [cit. 2023-02-23]. ISBN 9781491938409. Dostupné z: https://www.oreilly.com/library/view/org-design-for/9781491938393/

[29] *CRA IoT portál* [online]. ČR: České radiokomunikace, 2023 [cit. 2023-03-21]. Dostupné z: https://portal.iot.cra.cz/dashboard

[30] SEDLÁK, Jan. Síť internetu věcí Sigfox míří do bankrotu, v Česku má provoz fungovat i nadále. In: *Www.lupa.cz* [online]. ČR: Internet Info, 2022 [cit. 2023-03-21]. Dostupné z: https://www.lupa.cz/aktuality/sit-internetu-veci-sigfox-miri-do-bankrotu-v-cesku-ma-provoz-fungovat-i-nadale/