VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU LORA ZAŘÍZENÍ

SYSTEM FOR LORA DEVICES MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP ŠTOLFA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

145559

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Štolfa Filip**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Systém pro správu LoRa zařízení**
Kategorie:	Informační systémy
Akademický rok:	2022/23

Zadání:

1. Prostudujte oblast internetu věcí (*Internet of Things*).
2. Prostudujte principy LoRa zařízení, Lora Network Servers (LNS), způsobu komunikace koncových zařízení s LNS a integrace LNS s aplikačními cloudy. Porovnejte existující LNS.
3. Analyzujte způsob registrace a správy koncových zařízení v minimálně dvou LNS (The Things Network a další vybraný). Definujte požadavky firmy Logimic na registraci a správu LoRa zařízení.
4. Navrhněte architekturu cloudového modulu pro centrální registraci koncových zařízení a jeho komunikaci s LNS. Navrhněte webovou aplikaci pro přístup k LNS API.
5. Navržené řešení implementujte.
6. Proveďte testování funkčnosti a použitelnosti implementace na platformě Smart City firmy Logimic.

Literatura:

- Greengard, S. (2015). *The Internet of Things*. MIT Press.
- Kirimtat, A., Krejcar, O., Kertesz, A., & Tasgetiren, M. F. (2020). Future trends and current state of smart city concepts: A survey. *IEEE access*, *8*.
- Moravčík, T. (2022). *Zpracování a ukládání IoT dat se zaměřením na LoRA senzorové sítě*. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Brno. Dostupné z: https://www.fit.vut.cz/study/thesis/24980/.
- Lora Alliance. (2022). *Lora Alliance*. Dostupné z: https://lora-alliance.org/.
- Interní dokumentace firmy Logimic.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	18.10.2022

Abstrakt

Cílem této práce je zjednodušit správu IoT zařízení v ekosystému LoRaWAN a umožnit bezproblémově přecházet mezi dvěma centrálními servery sítě LoRaWAN. V této práci jsou popsány dva servery sítě LoRaWAN – ChirpStack a TheThingsStack. Na základě požadavků uživatelů systému pro správu chytrých měst je navržena a implementována aplikace, která umí s těmito servery komunikovat a nabízí uživatelům jedno rozhraní, pro správu zařízení na těchto serverech. Data zařízení jsou nejprve uložena v databázi aplikace a poté jsou synchronizována s vybraným serverem LoRaWAN sítě pomocí poskytnutého aplikačního rozhraní. Výsledné řešení je implementováno jako webová aplikace pomocí aplikačního rámce Angular. Serverová část je implementována pomocí aplikačního rámce Express.js a platformy AWS Lambda.

Abstract

The main goal of this thesis is to simplify the management of IoT devices in the LoRaWAN ecosystem and enable seamless migration between two central LoRaWAN servers. Two LoRaWAN network servers—ChirpStack and TheThingsStack—are described in this thesis. Based on user requirements for a smart city management system an application is designed and implemented that can communicate with these servers and offers users a single interface to manage devices on these servers. The device data is first stored in the application database and then synchronized with the selected LoRaWAN server using the provided application interface. The solution is implemented as a web application using the Angular application framework. The backend is implemented using the Express.js application framework and the AWS Lambda platform.

Klíčová slova

LoRaWAN, IoT, Angular, LoRaWAN Network Server, The Things Stack, ChirpStack

Keywords

LoRaWAN, IoT, Angular, LoRaWAN Network Server, The Things Stack, ChirpStack

Citace

ŠTOLFA, Filip. *Systém pro správu LoRa zařízení*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Systém pro správu LoRa zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D. Další informace mi poskytl pan Ing. Michal Valný, Ph.D. a Ing. Petr John z firmy Logimic. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . .
Filip Štolfa
9. května 2023

Poděkování

Chtěl bych poděkovat vedoucímu práce, Ing. Jiřímu Hynkovi, Ph.D. za pomoc, rady a časté konzultace. Dále bych chtěl poděkovat panu Ing. Michalu Valnému, Ph.D. a panu Ing. Petru Johnovi za odbornou pomoc při vypracovávání této práce.

Obsah

Kapitola 1

Úvod

Chytrá zařízení se dnes rozšiřují do všech oblastí života. Výjimkou nejsou ani města, ve kterých se začíná prosazovat automatizace, která vede k úspornějším a chytřejším městům. Počet zařízení ve městech připojených do sítě se v posledních letech značně zvyšuje. Kvůli velkému počtu zařízení a jejich umístění je potřeba využít bezdrátové připojení. Jednou z bezdrátových technologií, které se za tímto účelem používají je LoRaWAN, která umožňuje bezdrátovou komunikaci s dosahem v řádech kilometrů.

K provozování LoRaWAN sítě je potřeba mít přístup k serverům implementujícím centrální LoRaWAN funkcionalitu. V dnešní době existuje několik poskytovatelů těchto serverů, kteří nabízí cloudové či lokální řešení. Nyní má každý poskytovatel či implementace serverů LoRaWAN svůj proces registrace chytrých zařízení, který je nekompatibilní s ostatními. Každý má své rozhraní a není možné jednoduše přesunout větší množství zařízení na jiný server. Přechod k jinému poskytovateli je často značně manuální proces. To je jednak časově náročné a také je při takovém postupu velmi jednoduché udělat chybu.

Cílem této práce je vytvořit aplikaci, která poskytne jednotné rozhraní pro práci s několika poskytovateli komunikačních služeb na bezdrátové síti LoRaWAN (TheThingsStack a ChirpStack). Aplikaci by mělo být jednoduché upravit pro podporu dalších poskytovatelů. Měla by umožnit správcům chytrých měst nespoléhat se pouze na jednoho konkrétního poskytovatele. Budou moci jednoduše přejít k jinému, či využívat několik různých poskytovatelů.

V kapitole 2 je rozebrána problematika internetu věcí. V sekci 2.1 je popsána obecná architektura těchto systémů. Sekce 2.2 je věnována výzkumu a vývoji řešení problémů v oblasti chytrých měst. Dále jsou v sekci 2.3 porovnány komunikační protokoly se kterými je možné se setkat v systémech internetu věcí. Kapitola 3 se věnuje komunikační technologii LoRaWAN. Je probrána architektura těchto sítí (sekce 3.1) a v sekci 3.2 jsou rozebrány a porovnány dvě implementace LoRaWAN Netwrok Serverů (TheThingsStack a ChirpStack). V kapitole 4 je provedena analýza problému a jsou definovány požadavky na řešení. Kapitola 5 se zabývá návrhem řešení na základě vytvořených požadavků. Následně je v kapitole 6 popsán proces implementace řešení. Kapitola 7 se zabývá testováním výsledného řešení a kapitola 8 obsahuje závěr této práce.

Kapitola 2

Internet věcí

Internet věcí (anglicky *Internet of Things*, zkráceně IoT) je možné definovat jako síť fyzických zařízení, která spolu komunikují, sbírají a vyměňují si data, na základě kterých mohou upravovat svoji činnost [15]. Tato zařízení jsou většinou velmi jednoduchá a sama dokáží vykonávat pouze základní funkce. Až následným propojením těchto malých částí vznikají užitečné systémy. Nejpodstatnější myšlenkou IoT je automatizace nejrůznějších prvků fyzického světa a získávaní dat a informací, na základě kterých je možné vyvodit nové poznatky o monitorovaném prostředí.

Internet věcí se využívá v mnoha odvětvích, mezi která patří například chytré domácnosti, chytrý průmysl (anglicky *Industrial IoT*, zkráceně IIoT), chytré řízení provozu, chytré zdravotnictví a automatizace zemědělství [2]. Systémy z těchto odvětví se mohou překrývat ve funkcionalitě a také často úzce spolupracovat. Účel chytrých domácností je co nejvíce zjednodušit a automatizovat každodenní život obyvatel domácnosti. Příkladem je roztáhnutí žaluzií a zhasnutí světel, pokud je na základě venkovních senzorů dost světla ze slunce, čímž je ušetřena elektrická energie za svícení, ale také je vytvořeno zdravější prostředí pro obyvatele domácnosti. Další úspory jsou možné pomocí automatické regulace topení, nebo díky chytré lednici, která upozorní na jídlo, které je nutné spotřebovat a zamezí se tak plýtvání jídlem.

Zajímavé využití automatizace domácnosti je také usnadnění každodenního života stárnoucí populaci. Správně navržená chytrá domácnost může snížit jejich závislost na pomoci ostatních, ale také monitorovat jejich zdravotní stav a v případě potřeby přivolat pomoc [2]. Nejrozšířenějším druhem senzorů fyzického stavu člověka jsou fitness monitory, v podobě náramků či chytrých hodinek. Podle [19] se v posledních letech zvýšil zájem o nasazení IoT ve zdravotnictví. Cílem řady projektů je vytvořit senzory, které by mohly dlouhodobě monitorovat zdravotní stav člověka a sloužit tak nejen pro monitorování při zdravotních potížích, ale i pro sbírání dat určených pro prevenci.

Oblastí, která spojuje velké množství odvětví IoT jsou *chytrá města*. Dle článku [11] procento populace žijící ve městech v posledních letech značně roste a s tím roste i spotřeba energií. Mezi způsoby jak se vypořádat s rostoucí spotřebou měst pomocí IoT patří například chytré pouliční osvětlení, větší integrace obnovitelných zdrojů elektrické energie do měst a využití takzvané chytré elektrické sítě. Chytrá města se dále zaměřují na zvýšení kvality života obyvatel, díky monitorování kvality ovzduší, monitorování kvality vody, zamezením dopravních zácp nebo zvýšením bezpečnosti. Systémy chytrých měst mohou umožnit obyvatelům nahlásit poruchy a problémy ve městě, které budou následně zkontrolovány a vyřešeny. Kvůli velkému množství zařízení v chytrých městech musí být systémy schopné zpracovávat velké množství dat. Za tímto účelem jsou často využívány cloudové platformy,

určené na analýzu *velkých dat* (anglicky *big data*). Bez schopnosti získat z naměřených dat užitečné informace nemá smysl provozovat systémy chytrých měst a není možné dosáhnout požadovaných cílů.

Využití nachází IoT také v průmyslu, kde se využívání systémů internetu věcí nazývá „čtvrtou průmyslovou revolucí“ nebo také *Průmysl 4.0* [25]. Automatizace není novým konceptem v průmyslové výrobě, internet věcí ale umožňuje získat podrobnější informace o celém procesu výroby, které mohou být využity pro ještě větší zvýšení efektivity. Propojením několika separátních systémů pomocí IoT technologií je možné například materiály označené pomocí RFID značek přemístit autonomním přepravním zařízením ze skladu na místo spotřeby, kde budou dále zpracovány. Interní systém skladu je informován o spotřebě materiálu a upozorní na případný nedostatek naskladněného zboží. Proces výroby je monitorován pomocí sítě senzorů, která může sloužit jak k zajištění kontroly kvality, tak k zastavení výrobního procesu v případě poruchy a nahlášení této poruchy příslušným pracovníkům [25]. V oblasti energetiky může IoT, mimo snížení spotřeby zmíněné u chytrých měst, pomoci s monitorováním a se zvýšením spolehlivosti elektráren a díky tomu snížit náklady na jejich provoz [17].

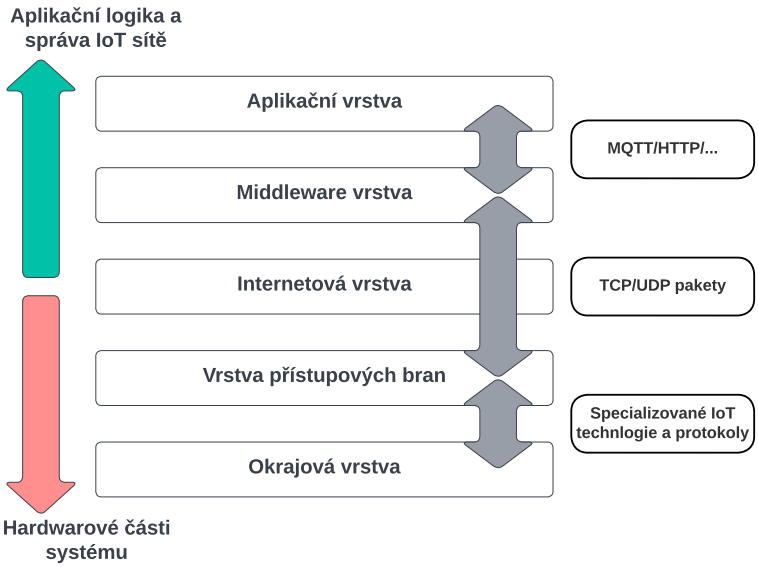
2.1 Architektura internetu věcí

Přesná architektura IoT systémů se může lišit podle konkrétní aplikace. Ovšem většinu systémů je možné rozdělit na dvě hlavní části, které jsou navzájem propojené internetovou vrstvou [2]. Tato architektura je vyobrazena na obrázku 2.1. První dvě vrstvy zpracovávají, zobrazují a poskytují data v užitečné podobě uživatelům. Zatímco spodní dvě vrstvy získávají data z reálného světa, nebo jej nějakým způsobem ovlivňují. Na nejnižší úrovni, nazývané okraj sítě (anglicky *edge*), se nachází vlastní chytrá zařízení – senzory a aktuátory. Tato zařízení komunikují s druhou spodní vrstvou, kterou tvoří přístupové brány, nebo směrovače. Spolu se specializovanými síťovými technologiemi se tato vrstva stará o přenos a směrování zpráv mezi servery na vyšších vrstvách a zařízeními na okraji sítě. Většinou zde dochází k překladu zpráv ze specializovaného protokolu používaného v IoT síti na běžnější protokoly založené na internetovém protokolu.

Následující vrstva, tzv. *middleware*, spojuje spodní hardwarové vrstvy s aplikační vrstvou. Dochází zde ke správě zařízení, agregaci dat, zabezpečení IoT sítě (hlídání přístupových práv zařízení do sítě, ale také práv přístupu aplikací k datům), filtraci dat a zajištění kvality služeb sítě. Tato vrstva je tou nejdůležitější z pohledu funkčnosti sítě. Z pohledu uživatele IoT systému je ale nejdůležitější nejvyšší a poslední vrstva, tedy aplikační vrstva. S tou uživatel interaguje a ta mu umožňuje využít celý systém. Aplikační vrstva se bude nejvíce ze všech vrstev lišit, podle konkrétního typu IoT systému. Zdravotnický personál potřebuje zcela jiné funkce než obyvatel chytré domácnosti [2].

2.2 Vývoj v oblasti chytrých měst

Toto odvětví IoT spojuje dohromady problematiku velkých IoT systémů na makro úrovni měst s problematikou velkého množství malých, částečně nezávislých systémů v podobě chytrých domácností. Komunikace mezi těmito úrovněmi IoT systémů je stále v raných fázích vývoje a pro úspěšnou implementaci propojeného města je stále potřeba provádět výzkum a zkoušet nové nápady a technologie.



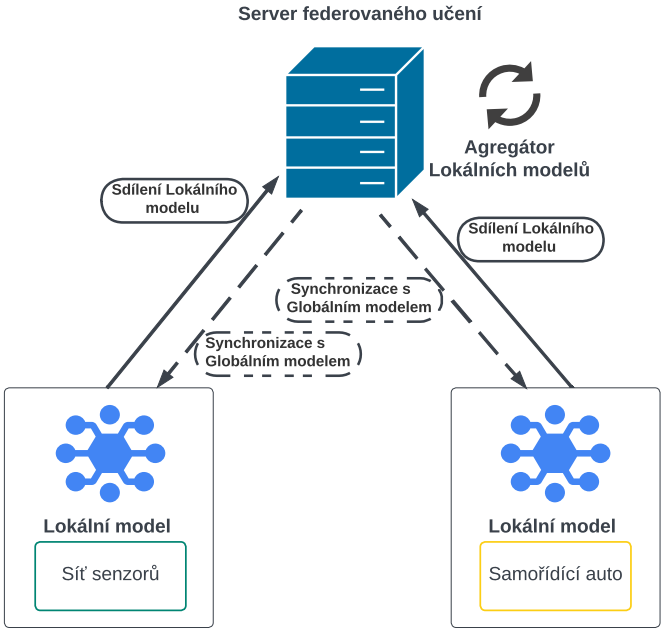
Obrázek 2.1: Běžné vrstvy architektury internetu věcí, převzato a upraveno z [2]. Zobrazuje rozdělení vrstev dle využívaných komunikačních technologií a protokolů – specializované IoT technologie a běžné TCP/IP sítě. Po pravé straně jsou vypsány nejčastěji využívané komunikační protokoly pro dané vrstvy.

Kvůli masivnímu množství dat produkovaných IoT zařízeními v chytrých městech je těžké provádět učení modelů umělé inteligence na centrálním serveru. Kromě toho tento způsob zpracování dat také nese značná bezpečnostní rizika a riskuje narušení soukromí uživatelů, jelikož na server provozovaný poskytovatelem dané služby jsou přenášena nezpracovaná data, která nesou značné množství osobních informací. Problémem je nejenom možný únik těchto dat, ale i to, že poskytovatel služby má přístup ke všem datům uživatelů.

Jedním z aktuálně zkoumaných řešení těchto problémů je takzvané *federované učení*. Podle článku [18] se jedná o metodu vytváření modelů strojového učení lokálně na koncových zařízeních a sdílení těchto modelů s centrálním serverem za účelem vzájemného vylepšení vytvořených modelů. V systému vystupují dva typy entit: koncová zařízení a agregátor modelů. Koncová zařízení mohou být chytré telefony, samořídící auto, nebo řídící server sítě senzorů. Agregátorem modelů je centrální server, který komunikuje se všemi zařízeními a řídí celý proces učení.

Tvorba a vylepšování modelů probíhá v iteracích. Prvním krokem je rozeslání nového modelu z agregátoru na koncová zařízení. Tento model je následně trénován na jednotlivých zařízeních, které pro učení využívají svoje lokální data, vzniká tak lokální model [18]. Modely od všech zařízení jsou poté zaslány zpět na agregátor, který na jejich základě vytvoří nový, vylepšený globální model. Ten je rozeslán na zařízení a proces se opakuje. Jelikož vlastní data zůstávají pouze na koncovém zařízení, dochází ke zvýšení bezpečnosti a soukromí. Jedním z hlavních problémů tohoto přístupu je výpočetní náročnost strojového učení, vzhledem k omezené výpočetní kapacitě mnoha IoT zařízení. Je tedy nutné brát ohled na to, jaké zařízení je vhodné zapojit do federovaného učení. Jedním z řešení těchto problémů je využití hardwarového akcelerátoru pro strojové učení.

Dalším, dnes velmi důležitým tématem, je integrace chytrých zařízení různých výrobců do jednoho IoT systému. Tento problém je nejvíce vidět v oblasti chytrých domácností. Kvůli



Obrázek 2.2: Komunikační proces federovaného učení, převzato a upraveno z [18]. Centrální server (zobrazen nahoře) sbírá vytrénované lokální modely od koncových zařízení. Provede jejich agregaci a sestaví nový, vylepšený globální model. Ten poté rozešle ke všem koncovým zařízením.

snaze výrobců přimět zákazníky kupovat pouze jejich zařízení dochází ke značné fragmentaci IoT prostoru. Tato zařízení je možné ovládat pouze pomocí aplikace konkrétního výrobce a není možné využít data ze senzoru výrobce A k automatizaci zařízení výrobce B, což značně omezuje jejich užitečnost [4]. Vyřešit tento problém je cílem specifikace aplikačního rozhraní pro IoT zařízení *Matter*. Tento standard je spravován skupinou CSA, jejíž členy jsou například firmy Google a Apple. Verze `1.0` byla publikována v říjnu roku 2022 a zatím není příliš mnoho zařízení, které tuto specifikaci podporují. Dalším možným řešením je využití open source IoT platformy *Home Assistant*. Ta podporuje zařízení od více než 1000 různých výrobců, dá se velmi snadno přizpůsobit potřebám konkrétních uživatelů a je plně lokální, bez nutnosti připojení systému k internetu. Nevýhodou pro méně technicky zdatné uživatele může být komplikovanější nasazení a správa systému.

2.3 Komunikační technologie a protokoly

Komunikaci, jakožto nejdůležitější schopnost chytrých zařízení, lze realizovat pomocí celé řady technologií. Dle požadavků konkrétního IoT systému je nutné vybrat tu správnou kombinaci metod pro spojení zařízení do jednotné sítě. První kritérium, které je možné použít pro klasifikaci komunikačních technologií je typ připojení – zda se jedná o drátové či bezdrátové připojení. I v menších IoT systémech se nachází značně velký počet zařízení, které je všechny potřeba připojit do sítě. Z tohoto důvodu se velmi často využívají bezdrátové komunikační technologie. Drátové připojení má nevýhodu v nutnosti dostat k zařízení kabel, což může být často komplikované, či dokonce nemožné. Na druhou stranu má ale

značnou výhodu ve spolehlivosti připojení, což je velmi důležité, zejména u kritických částí systémů. Zástupcem drátového připojení je například ethernet a zástupcem bezdrátového připojení je například hojně rozšířená technologie WiFi.

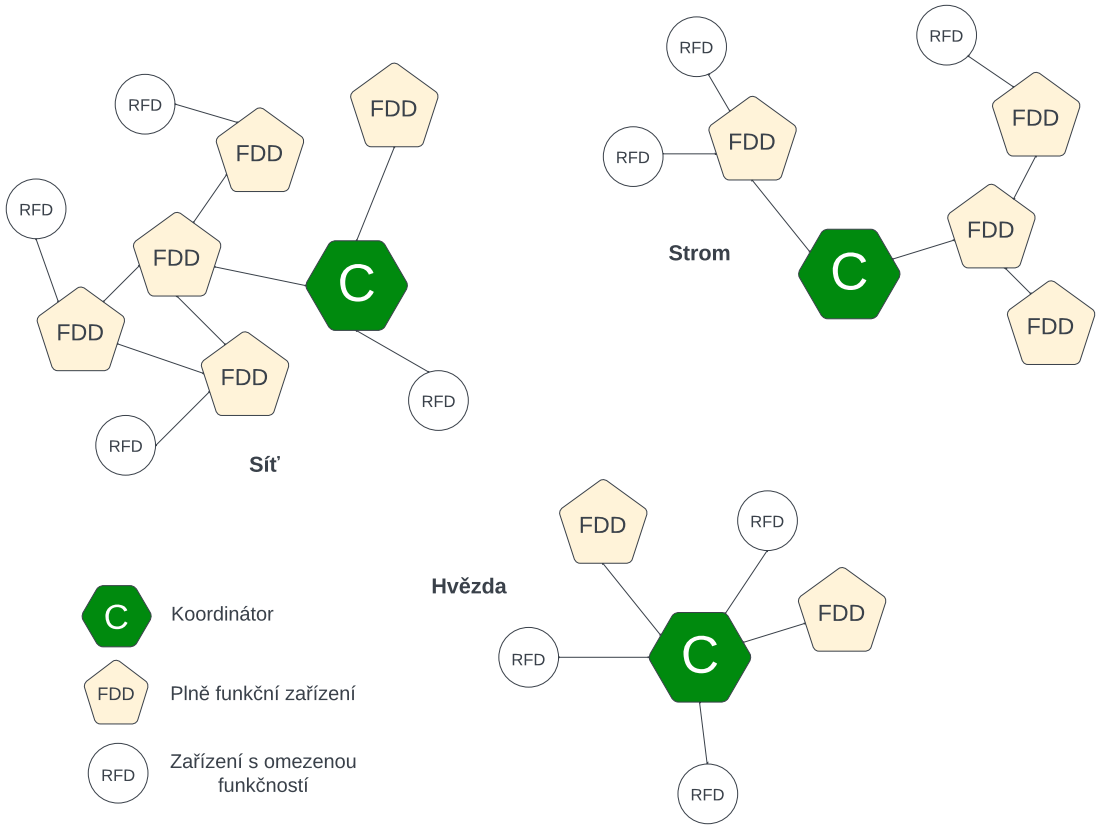
Dalším kritériem při výběru komunikační technologie je vzdálenost a prostředí, ve kterém se IoT systém bude provozovat – venkovní, či vnitřní. Pro vnitřní internetové bezdrátové sítě se nejčastěji používá již zmíněná WiFi. I přes svoji rozšířenost však není velmi dobrým řešením pro IoT sítě. Jedná se o velmi energeticky a časově náročnou rádiovou komunikaci, což je problém zejména u malých zařízení na baterii, kde je potřeba co nejvíce snížit spotřebu při běhu zařízení a také co nejvíce času strávit ve stavu spánku. Tyto problémy se snaží řešit řada rádiových technologií speciálně navržených pro IoT systémy. Například *ESP-NOW* využívá stejné frekvenční pásmo 2,4 GHz jako WiFi, na rozdíl od ní ovšem nevyžaduje dlouhý proces připojování zařízení do sítě. To umožňuje zařízením po probuzení ze spánku velmi rychle poslat naměřená data a znovu přejít do stavu spánku. Zařízení jako jsou chytré hodinky typicky využívají rádiovou technologii *Bluetooth* pro komunikaci s mobilním telefonem uživatele.

Zigbee je standard definující bezdrátovou komunikaci vhodnou pro přenos malého objemu dat s velmi malou energetickou spotřebou [21]. Dosah této technologie je mezi 10 až 50 metry, dle podmínek prostoru. Tento dosah by značně limitoval využitelnost technologie, Zigbee ovšem umožňuje tvorbu *mesh* sítí (česky topoloie síť). Tedy sítí, ve kterých mohou jednotlivá zařízení komunikovat přímo mezi sebou a přeposílat zprávy od okraje až ke středu sítě. Mimo tuto topologii také podporuje síťové topologie typu hvězda a strom. Ukázky jednotlivých topologií je možné vidět na obrázku 2.3. Standard vznikl roku 2006 a je spravován skupinou Connectivity Standards Alliance[1]. Skládá se ze tří vrstev: fyzická, síťová a aplikační. Fyzická vrstva na které Zigbee staví je definována standardem IEEE 802.15.4. V Zigbee síti existují tři druhy zařízení: koordinátor, zařízení s plnou funkčností (FFD) a zařízení s omezenou funkčností (RFD). Koordinátor je centrálním prvkem sítě, spravuje všechna ostatní zařízení a přeposílá zprávy mezi venkovní a Zigbee sítí. FFD slouží jako směrovače sítě a vysílaným signálem prodlužují dosah sítě. Tato zařízení musí být neustále aktivní, není možné je provozovat na baterii. RFD mohou v síti vystupovat pouze jako koncová zařízení, jelikož většinu času tráví ve stavu spánku a neumí rozšiřovat síť.

Pro venkovní IoT systémy je potřeba pracovat s dosahem nejméně v řádu stovek metrů, nejlépe v řádech kilometrů. Síťový protokol *DASH7* je určen pro krátké až středně dlouhé venkovní vzdálenosti. Je vyvíjen skupinou DASH7 Alliance a jeho poslední verze `1.2` byla publikována v lednu 2019 [8]. S průměrným dosahem 500 m je vhodný pro hustě zastavěné oblasti a průmyslové haly. Jeho hlavní výhodou je velmi malá energetická náročnost, zároveň však umožňuje kontaktovat zařízení ze strany serveru s průměrnou odezvou 1 sekunda. Právě zahájení komunikace ze strany serveru bývá častým problémem IoT sítí. Tento nedostatek má i komunikační technologie pro velmi dlouhé vzdálenosti LoRa a s ní často používaný protokol LoRaWAN. I přes tento nedostatek je velmi populární, obzvlášť kvůli maximálnímu dosahu až 15 km. Více je tato technologie popsána v kapitole 3. Tabulka 2.1 porovnává vlastnosti často používaných komunikačních technologií v IoT systémech.

Některé komunikační technologie (například Zigbee) definují své vlastní komunikační protokoly, ty jsou ale užitečné pouze uvnitř sítě dané technologie. Aby mohla zařízení komunikovat se zařízeními které využívají jinou technologii, nebo aby mohla komunikovat s IoT aplikacemi, je potřeba zprávy přeložit do univerzálnějšího protokolu. To probíhá ty-

[1] https://csa-iot.org/all-solutions/zigbee/



Obrázek 2.3: Schémata tří různých topologií Zigbee sítě. Topologie síť vytváří robustní a rozsáhlou síť, díky schopnosti FDD zařízení komunikovat mezi sebou. Topologie strom si zachovává schopnost rozšířit síť pomocí FDD, ale zároveň vytváří méně složité propojení, ve kterých je snažší najít cestu k centrálnímu koordinátoru. Topologie hvězda je nejjednodušší topologií.

picky na koncentrátoru, či routeru, který propojuje síť zařízení se zbytkem lokální sítě, případně s venkovním internetem. Jedním z nejvíce rozšířených protokolů je MQTT. Často je využíván nejen ke komunikaci zařízení, ale také komunikaci různých aplikací mezi sebou. Tento protokol je blíže popsán v podsekci 2.3.1. Mimo něj se často také vyskytuje komunikace přes HTTP protokol a REST API. Tento druh komunikace je častý pro správu zařízení, nebo získávání již zpracovaných dat od centrálního serveru pro zobrazení v aplikaci. Dalším používaným protokolem pro komunikaci různých částí IoT systému je *AMQP*, který je podobný MQTT protokolu, ale je více zaměřený na komunikaci velkých systémů aplikací, zatímco MQTT je vhodnější pro nespolehlivé sítě a malá zařízení.

2.3.1 MQTT

Protokol MQTT vznikl v roce 1999 ve firmě IBM. Dnes se jedná o otevřený standard organizace OASIS. Je to jednoduchý komunikační protokol postavený nad TCP/IP. Existuje ale i upravená verze protokolu MQTT-SN (MQTT for sensor netowrks, česky MQTT pro sítě senzorů), která lze použít v sítích, které nepoužívají TCP/IP komunikační protokoly [10]. Protokol je určen pro zařízení, které mohou mít nekvalitním a pomalé připojení. Umožňuje

Technologie	Maximální dosah	Druh sítě	Topologie	Maximální přenosová rychlost
Bluetooth	<50 m	PAN	Scatternet	1 Mbit/s
WiFi	20–100 m	LAN	Hvězda/Síť	3466.8 Mbit/s
Zigbee	10–50 m	PAN	Hvězda/Síť/Strom	250 kbit/s
DASH7	5 km	Medium range	Hvězda/Síť/Strom	166 kbit/s
LoRaWAN	15 km	LPWAN	Hvězda hvězd	27 kbit/s

Tabulka 2.1: Porovnání bezdrátových komunikačních technologií

zprávy posílat s několika úrovněmi kvality služeb (*QoS*). Používá se ale i pro komunikaci mezi různými aplikacemi, které jsou součástí IoT systému a pro přeposlání zpráv z middleware vrstvy aplikační vrstvě (viz obrázek 2.1).

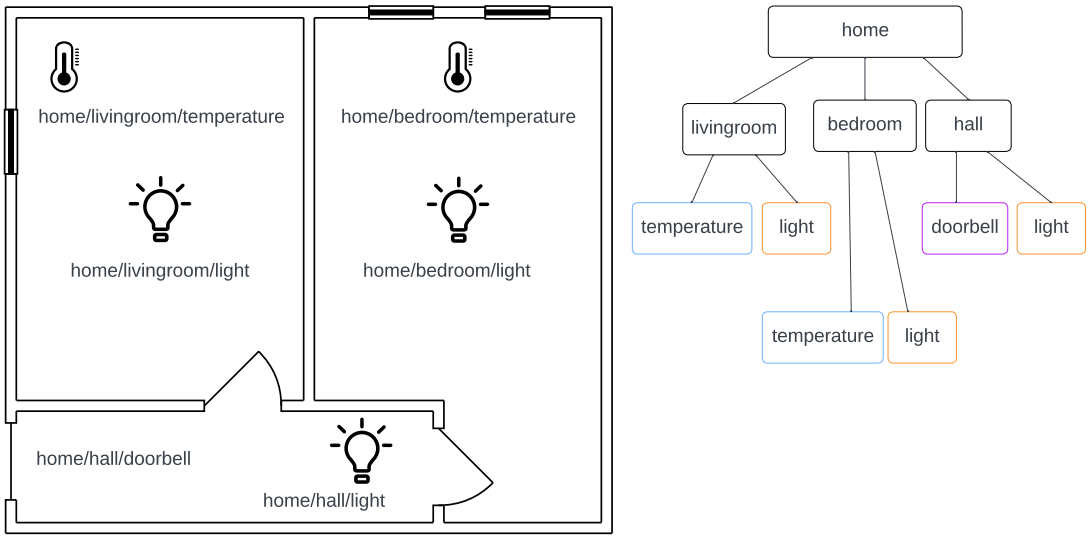
Architektura MQTT je klient-server a výměna zpráv je založena na principu publish-subscribe. Zařízení (klient) se připojí na centrální server, také nazývaný *broker*, který přijímá a přeposílá zprávy od všech klientů. Koncová zařízení se přihlásí k odběru tzv. témat (anglicky topic), na které poté jiná zařízení mohou zasílat zprávy. Broker přijme zprávu od zařízení a přepošle ji všem klientům, kteří jsou přihlášení k tomuto tématu.

Témata jsou reprezentována jako řetězce, kde speciální znak `/` (lomítko) slouží k oddělení jednotlivých podtémat. Vytváří tak stromovou strukturu, která umožňuje se velmi snadno přihlásit k odběru specifických informací. I když by se mohlo nabízet použít jako kořen stromu znak `/` (jako je zvykem například v systémech UNIX), není to potřeba. Kořenové téma je možné zvolit libovolně. Každý výskyt znaku `/` vytváří novou úroveň zanoření tématu. Zařízení se mohou přihlásit k odběru buď kompletního tématu, nebo použít tzv. *wildcard* znaky `+` a `#`. Znak `+` zastupuje jednu konkrétní úroveň tématu. Jestliže se klient například přihlásí k odběru tématu `koren/+/list`, budou mu doručeny zprávy publikovány na obě následující témata: `koren/a/list`, `koren/b/list`. Zprávy publikované na téma `koren/a/c/list` mu ovšem doručeny nebudou. Aby se přihlásil k odběru i těchto zpráv, může použít znak `#`, který nahrazuje všechny podúrovně.

Na obrázku 2.4 je vidět příklad rozvržení témat pro IoT systém v domácnosti. Jako kořenové téma zde bylo zvoleno `home`. Pokud by se tento systém rozšířil například i na zahradu, bylo by možné tuto část systému seskupit pod kořenové téma `garden`. Dále je vytvořeno podtéma pro každou místnost. V rámci místností jsou poté definována témata pro koncová zařízení. Pro odběr zpráv o teplotě v obývacím pokoji se klient přihlásí o odběr tématu `home/livingroom/temperature`. Pokud chce dostávat zprávy od všech teploměrů v systému, přihlásí se k odběru tématu `home/+/temperature`. Všechny zprávy, které nějak souvisí s obývacím pokojem, bude klient dostávat po přihlášení k tématu `home/livingroom/#`.

Témata by také bylo možné uspořádat nejdříve podle funkce a poté rozlišit jejich umístění. Teplota v obývacím pokoji by tedy byla dostupná na `home/temperature/livingroom`. Uspořádání témat je čistě na uživateli a záleží na konkrétní situaci.

Protokol MQTT má dvě hojně používané verze – verzi `3.1.1` [3] vytvořenou v roce 2014 a verzi `5.0` [1] z roku 2019. Od verze `5.0` je možné vytvořit alias pro téma, kdy je řetězce identifikující téma nahrazen číslem. Dále například zavádí možnost nastavit datum expirace zpráv. Pokud není zpráva doručena klientovi do tohoto data, je smazána. Obě verze podporují tři úrovně kvality služeb (QoS).



Obrázek 2.4: Diagram možného přiřazení MQTT témat místnostem v domácnosti (vlevo) a stromová struktura kterou tyto témata vytvoří (vpravo). Témata jsou organizována podle místností.

- `QoS 0` – nejvíce jednou. Zpráva s touto úrovní QoS je zařízením odeslána pouze jednou. Její doručení není nijak kontrolováno. Je tedy možné, že se ztratí při přenosu.
- `QoS 1` – alespoň jednou. Po odeslání zprávy odesílatel čeká na potvrzení přijetí. Pokud do určité doby potvrzení nedostane, je zpráva poslána znovu. Toto se opakuje než je potvrzeno přijetí. Je ale možné, že původní zpráva dorazí až po dalším pokusu o odeslání, což má za následek, že je doručena vícekrát.
- `QoS 2` – právě jednou. Mezi odesílatelem a příjemcem dojde k výměně několika paketů pro potvrzení navázání komunikace, díky tomu je možné zaručit doručení zprávy, bez opakování.

Maximální délka řetězce identifikujícího téma je u obou verzí 65536 bajtů. Délka přenášených dat je ve verzi `3.1.1` omezena na 256 MB. Ve verzi `5.0` není délka přesně definována, záleží na velikosti proměnlivé hlavičky konkrétní zprávy.

Pro přenos MQTT zpráv je typicky využíván protokol TCP, je však možné využít i protokol *websockets*, který staví na TCP. Tento protokol se využívá zejména u webových aplikací, kde umožňuje vytvořit oboustranný komunikační kanál mezi klientem a serverem. Pro MQTT má význam, právě když je potřeba, aby webová aplikace komunikovala s jiným zařízením přímo přes MQTT, bez nutnosti využití serveru jako prostředníka.

Některé implementace MQTT na IoT zařízeních nemusí podporovat odesílání zpráv s vyšší kvalitou služeb než `QoS 0`. Případně mohou podporovat pouze standardní přenos MQTT přes TCP. Verze `5.0` je podporována méně častěji než verze `3.1.1`. Jelikož IoT zařízení mají většinou omezenou operační paměť, některé implementace omezují velikost zprávy při odesílání, což jde ale většinou změnit v konfiguraci.

Kapitola 3

LoRaWAN

LoRaWAN je otevřený standard vyvíjený neziskovou asociací LoRa Alliance [13]. Staví nad proprietární rádiovou komunikací *LoRa* (název vytvořený ze slov long range, česky velký dosah) a definuje MAC a aplikační vrstvu. Tyto dvě technologie dohromady vytváří síť druhu *LPWAN* (low-power wide-area network, česky rozlehlá síť s nízkou energetickou spotřebou) [12]. Právě díky velkému dosahu a nízké spotřebě je tato technologie vhodná pro rozlehlé IoT systémy, jako jsou například chytrá města.

Rádiová komunikace LoRa funguje na principu *rozprostřeného spektra*, což je technika rozprostření signálu do více frekvencí, za účelem zvýšení odolnosti proti rušení a šumu. Konkrétně se jedná o proprietární verzi rozprostřeného spektra, založenou na technice *chirp spread spectrum* (CSS). Chirp (česky čerp) je druh signálu, jehož frekvence se snižuje nebo zvyšuje s časem [20]. LoRa je vysílána v bezlicenčních frekvenčních pásmech, což značně usnadňuje její nasazení. Přehled používaných frekvencí v různých částech světa je vidět v tabulce 3.1. Dosah této technologie je kolem 5 km v městských oblastech a kolem 15 km v méně zastavěných oblastech [22]. Rádiovou komunikaci LoRa lze využívat i bez komunikačního protokolu LoRaWAN. Článek [5] pojednává o možnostech využití LoRa s jinými protokoly, nebo s upravenými verzemi LoRaWAN protokolu, za účelem vytvoření sítí s odlišnými vlastnostmi než mají standardní LoRaWAN sítě. Kromě běžného nasazení LoRaWAN sítí ve venkovních prostranstvích je možné provozovat tyto sítě i ve výrobních či skladových halách a velmi efektivně tak vytvořit IoT síť pro aplikace chytrého průmyslu [9].

Region	**Frekvenční Pásmo**
Evropa	863–873 MHz/433 MHz
Jižní Amerika	915–928 MHz
Severní Amerika	902–928 MHz
Indie	865–867 MHz
Asie	915–928 MHz
Celosvětově	2.4 GHz

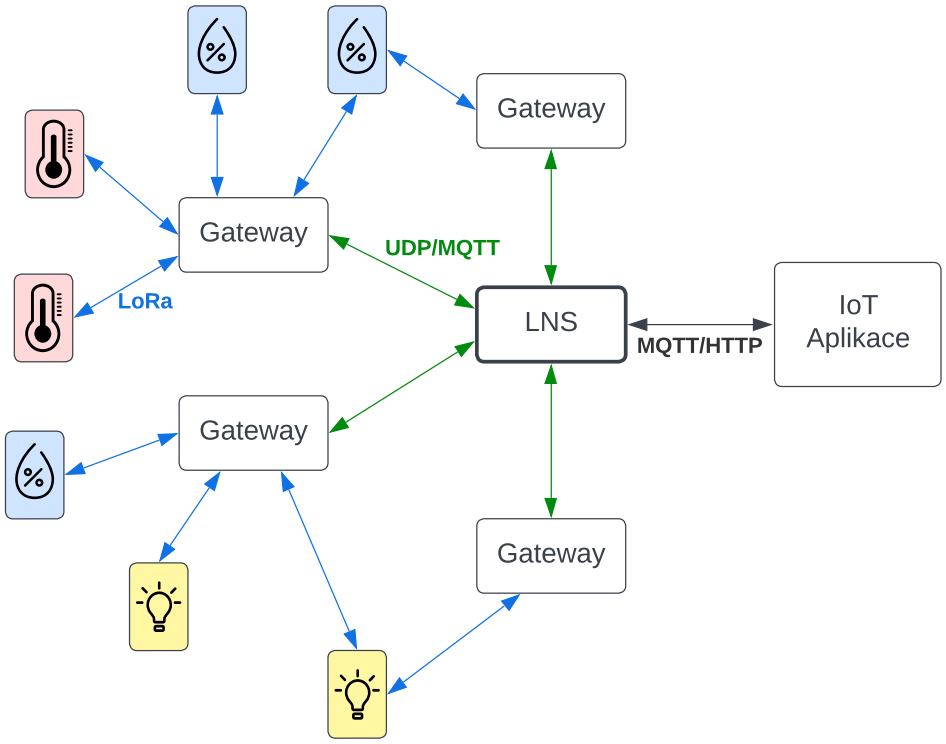
Tabulka 3.1: Bezlicenční frekvenční pásma podle regionu

Specifikace LoRaWAN má několik současně používaných verzí. Nejnovější je verze `1.0.4` z roku 2020. Často používané jsou ale i verze `1.0.3` a `1.0.2`. Jelikož většina zařízení je vyrobena pro jednu určitou verzi LoRaWAN protokolu, i starší verze jsou podporovány dlouho po jejich vydání. Existuje také verze `1.1` z roku 2017, která přináší řadu nových funkcí. Ty jsou pomalu začleňovány i do LoRaWAN řady `1.0.x`. Příkladem je přejmenování

`appEui` na `joinEui`, k čemuž došlo ve verzi `1.1`, ale do `1.0.x` řady verzí se tato změna dostala až v poslední verzi `1.0.4`.

3.1 Architektura sítě LoRaWAN

Sítě využívající komunikační protokol LoRaWAN mají topologii zvanou hvězda hvězd (viz obrázek 3.1) [5]. Koncová zařízení komunikují s bránami, tím je vytvořena první hvězda, zatímco brány komunikují s jedním centrálním serverem, což tvoří druhou hvězdu. Zařízení nemohou komunikovat přímo mezi sebou, stejně tak brány nikdy nekomunikují s jinou bránou, jak by tomu bylo například v topologii sít (*mesh*). Zpráva z jednoho zařízení může být přijata více bránami, pokud jsou v dosahu, tyto duplikované zprávy jsou vyfiltrovány centrálním serverem. Tento server dále poskytuje data uživatelským aplikacím.



Obrázek 3.1: Architektura sítě LoRaWAN. Síť má topologii hvězda hvězd. Centrálním prvkem sítě je LoRaWAN Network Server (LNS), který komunikuje s bránami a vytváří tak vnitřní hvězdu. Brány komunikují s koncovými zařízeními, čímž je tvořena vnější hvězda.

3.1.1 Servery

Středem celé sítě LoRaWAN je *LoRaWAN Network Server* (zkráceně LNS). Funkcionalita LNS může být rozprostřena mezi několik samostatných aplikací (*komponent*), které mohou být rozmístěny na rozdílné fyzické servery [23]. Tyto komponenty LNS dohromady slouží jako prostředník mezi uživatelskými aplikacemi, které běží na klasickém internetu a chyt-

rými zařízeními, která jsou součástí LoRaWAN sítě. Zároveň se také starají o zabezpečení celé sítě. Podle zodpovědnosti jednotlivých komponent je možné rozdělit je následovně:

Gateway Server či někdy *Gateway bridge* se stará o komunikaci mezi zbytkem LNS a bránami, které poté komunikují s koncovými zařízeními. Zajišťuje přenos dat ze serveru na zařízení a naopak. Toho docílí překladem UDP paketů, nebo LoRa Basics paketů z LoRa Basics station na MQTT pakety, které jsou poté dále rozeslány do zbytku LNS.

Network Server implementuje vlastní LoRaWAN protokol. Za pomoci dalších komponent se stará o validaci identity koncových zařízení a vybírá konkrétní bránu, která bude použita pro komunikaci s konkrétním zařízením. Dále se stará o deduplikaci zpráv, které byly přijaty více bránami najednou.

Application Server většinou poskytuje aplikační rozhraní pro uživatelské aplikace. Zpřístupňuje uživatelům přijaté zprávy od zařízení, které odšifruje a dekóduje. Stejně tak dokáže zpracovat zprávy od uživatele k zařízením, které před odesláním zase zakóduje a zašifruje. Dekódování a zakódování zpráv do správného formátu je řešené pomocí uživatelem definovaných formátovacích funkcí, nazývaných *payload formatters*. Komunikaci s uživateli většinou zajišťuje protokol MQTT, ale je možné nastavit i jiný způsob komunikace, jako například HTTP, AMQP nebo nechat LNS zprávy rovnou zapsat do *InfluxDB* databáze.

Join Server je komponenta, která primárně slouží pro bezpečné připojování zařízení do sítě. Dostane požadavek na připojení zařízení od Network Serveru, který buď přijme, nebo zamítne. V případě, že zařízení má přístup do sítě, vygeneruje session klíč, který poté používá Network Server a Application Server pro zabezpečení komunikace. Funkcionalita Join Serveru může být součástí jiné komponenty systému, její oddělení však umožňuje přesunout tuto část na servery, které vlastní uživatel a má je pod úplnou kontrolou, zatím co zbytek systému může být nasazen na cloudové platformě.

Toto rozdělení architektury není žádným způsobem vyžadováno specifikací LoRaWAN, je ale užitečné pro popsání jednotlivých funkcí, které musí každý LNS poskytovat. Nejvíce se tato obecná architektura podobá architektuře The Things Stack LNS [23], který je více popsán v podsekci 3.2.1. Nejčastějšími změnami je spojení funkcionality popsaných komponent, například Join Serveru a Identity Serveru či spojení Network Serveru a Application Serveru. Tyto změny jsou vidět na architektuře ChirpStack LNS [6], který je dále popsán v podsekci 3.2.2.

3.1.2 Brány

Brány (angl. *gateway*) zprostředkovávají komunikaci s koncovými zařízeními pomocí LoRa rádiové komunikace a přeposílají zprávy LoRaWAN Network Serveru, ke kterému jsou připojeny pomocí klasického internetu. Jednu zprávu od zařízení může přijmout více bran najednou. Taková zpráva bude na LNS doručena vícekrát, ten se ale postará o to, že dále bude zpracována pouze jednou. Pro doručení zprávy ze serveru na zařízení vybírá LNS tu bránu, která má nejlepší připojení k danému zařízení (více viz podsekci 3.1.1).

Brány LoRa většinou umí poslouchat a odesílat zprávy na 8 kanálech najednou, díky čemu je možné zamezit rušení a ztrátě zpráv v LoRa síti. Koncová zařízení náhodně mění kanál, na kterém budou vysílat, aby se využití sítě co nejvíce rozložilo. Je možné se setkat

i s bránami, které podporují pouze jeden kanál, ty však nejsou tolik spolehlivé jako více kanálové brány, jelikož je u nich značně zvýšené riziko ztráty paketů. To je způsobeno vysíláním několika zařízení v ten stejný okamžik bez možnosti rozložení zpráv na více kanálů. Z tohoto důvodu nejsou jednokanálové brány ve specifikaci LoRaWAN povoleny. Některé implementace LNS tyto brány vůbec nepodporují.

3.1.3 Koncová zařízení

Když se mluví o LoRaWAN zařízeních, většinou jsou tím myšlena právě koncová zařízení. Ta se vyznačují tím, že nějakým způsobem interagují s okolním světem. Aby mohlo zařízení komunikovat se zbytkem LoRaWAN sítě, musí se chovat podle jedné z následujících specifikací, které jsou nazývány třídy zařízení. Třídy specifikují, jakým způsobem bude zařízení komunikovat se sítí, jak často musí být dostupné pro komunikaci, jak dlouho může být v úsporném režimu kdy s ním nelze komunikovat a další. Základní třídou je třída A, kterou musí implementovat i zařízení nadřazených tříd [24].

Třída A je vhodná pro zařízení, která jsou většinu času neaktivní. Komunikace je vždy zahájena ze strany zařízení, většinou při změně nějaké monitorované hodnoty. Po odeslání hodnoty na server (tzv. *uplink*) zařízení po krátké prodlevě (zvané *RX1 delay*), začne čekat na příjem zprávy ze serveru. Pokud server během specifikované doby (většinou jedné sekundy) zprávu nepošle, zařízení se na krátkou dobu uspí. Po uplynutí druhé stanovené prodlevy (počítané od ukončení přenosu uplink, zvané *RX2 delay*) znovu začne poslouchat a čeká na zprávu ze serveru. Pokud ani v tomto okně nedostane žádnou zprávu, uspí se až do další změny stavu, tedy do další uplink zprávy ze strany zařízení na server. Stejně tak se zařízení uspí, pokud dostane zprávu od serveru již v prvním komunikačním okně.

Třída B Zařízení třídy B pracují velmi podobě jako zařízení třídy A. Změnou je pravidelné otvírání komunikačních oken, ve kterých může server kontaktovat zařízení. Aby zůstala komunikační okna synchronizovaná na serveru i zařízeních, vysílají brány synchronizační signál (tzv. *beacon*). Hlavní výhodou je, že server může zahájit komunikaci se zařízením (v přesně daných intervalech), na rozdíl od třídy A, kde server může komunikovat pouze poté, co zařízení zahájí komunikaci. Protože zařízení musí být aktivní častěji i když nemá co odeslat na server, je výdrž baterie u zařízení třídy B menší.

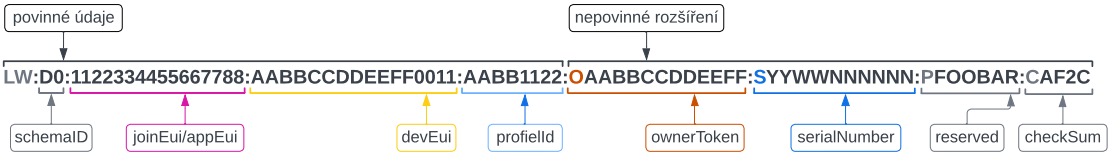
Třída C Zařízení třídy C jsou pořád ve stavu, kdy čekají na zprávu od serveru, tedy kromě odesílání zpráv na server. To znamená, že musí být pořád aktivní a tudíž potřebují mnohem více energie než předchozí dvě třídy. Proto se s touto třídou většinou setkáváme u zařízení, která nejsou napájena z baterie, nýbrž jsou připojena do elektrické sítě. Využití má tato třída u jakéhokoliv zařízení, které musí být schopno během velmi krátké chvíle reagovat na zprávy ze serveru. To jsou například chytré ventily či pouliční osvětlení.

Identifikátory koncových zařízení

Každé koncové zařízení má unikátní identifikátor zvaný *devEui*. Jedná se o 64bitový identifikátor, který je zařízení přiřazen výrobcem a měl by být globálně unikátní [22]. Tento identifikátor se zobrazuje jako osm párů hexadecimálních číslic. Stejný formát má také *joinEui* (dříve zvané *appEui*). To identifikuje server, který zařízení kontaktuje při aktivaci.

Každé zařízení podporuje jeden ze dvou způsobů aktivace. *ABP* (activation by personalization) je starší a méně bezpečná verze aktivace, která vyžaduje pevně dané a neměnné bezpečnostní klíče. Novější a doporučovanou metodou je *OTAA* (over-the-air activation). Během této aktivace jsou dynamicky vytvořeny bezpečnostní klíče. Pro šifrování prvotních zpráv aktivace je využit *appKey*. Následně jsou již použity nově vytvořené klíče [23].

Za účelem zjednodušení registrace nových zařízení zavedla LoRa aliance v roce 2020 specifikaci pro QR kódy obsahující identifikátory koncových zařízení LoRaWAN sítě [14]. Struktura dat uložených v tomto QR kódu je vidět na obrázku 3.2. První dvě písmena `LW` slouží pro identifikaci následující sekvence dat jako sady LoRaWAN identifikátorů. Hodnoty *schemaID*, *joinEui*, *devEui* a *profileId* jsou povinné v každém QR kódu splňujícím tuto specifikaci. Zbylé hodnoty se v datech vyskytovat nemusí, pokud se ale vyskytují, jejich první znak musí být vždy stejný jako je uvedeno na příkladu v obrázku 3.2. Tedy `O` pro *ownerToken*, `S` pro *serialNumber*, `P` pro proprietární rozšíření formátu dat a `C` pro *checkSum*. Tyto písmena slouží jako klíč pro rozpoznání o jakou nepovinnou hodnotu se jedná. Na rozdíl od povinných, nemají nepovinné hodnoty pevně určené pořadí, už jen z důvodu, že se nemusí vyskytovat všechny.



Obrázek 3.2: Struktura dat uložených v QR kódu který identifikuje koncové zařízení. Jednotlivé sekce dat jsou odděleny znakem dvojtečka (`:`), který se nesmí vyskytovat nikde jinde, než právě jako oddělovač částí. Levá polovina sekcí je povinná a musí se vyskytovat v každém QR kódu. Zbylé sekce jsou volitelné. Pokud se ale vyskytyjí, musí jejich první znak být právě ten, který je vidět na tomto schématu a je barevně označen.

3.2 Platformy LoRaWAN

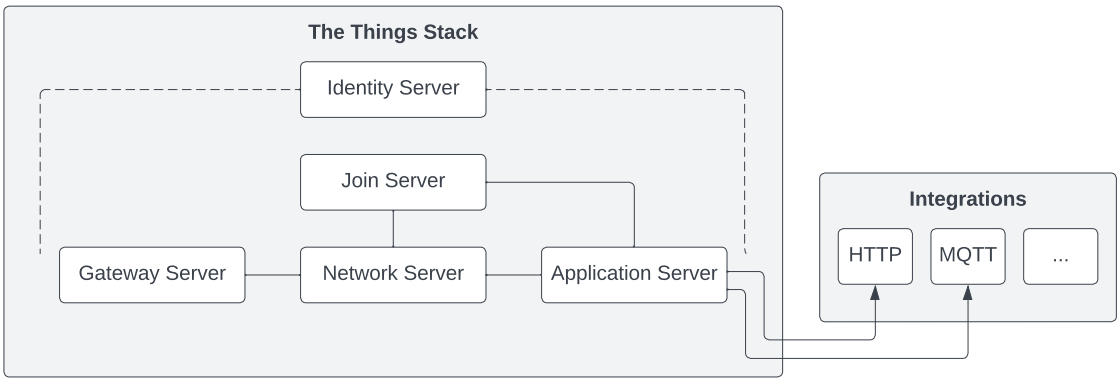
Pro provozování LoRaWAN sítě je potřeba implementace všech potřebných částí LoRaWAN Network Serveru. Často jsou tyto servery poskytovány jako cloudové služby. V této sekci jsou popsány dvě různé implementace LNS. V podsekci 3.2.1 je popsána platforma The Things Stack a následně je v podsekci 3.2.2 popsána open source implementace LNS ChirpStack.

3.2.1 The Things Stack

Jednou z nejvíce používaných implementací LNS je v dnešní době *The Things Stack* (zkráceně TTS) vyvíjený firmou The Things Industries (zkráceně TTI) [23]. TTS je dostupný jako placená cloudová platforma, která je cílená zejména na firmy. Jako alternativa pro jedince či menší firmy provozuje firma The Things Industries také platformu *The Things Network* (zkráceně TTN), která je postavena na TTS, má ale určitá omezení. Mezi omezení patří omezený počet zařízení a bran a neumožnění nasazení na vlastní servery. U placené verze TTS, kterou se tato práce zabývá, je možné si vybrat mezi čtyřmi způsoby nasazení LNS.

- *Cloud* – celý LNS je nasazen na serverech TTI, veškerou správu serverů má na starosti TTI
- *Dedikovaný Cloud* – nasazeno na oddělených serverech, správu má na starosti TTI
- *AWS Launcher* – šablona pro nasazení TTS na AWS serverech
- *Vlastní servery* – TTS je nasazeno na serverech pod kontrolou uživatele

Tento LNS podporuje všechny třídy koncových zařízení (viz sekci 3.1.3) a také podporuje obě nejnovější verze LoRaWAN protokolu (verzi `1.0.4` a `1.1`). Spravovat LNS je možné pomocí webového rozhraní, dále také pomocí aplikace pro příkazovou řádku nebo lze použít HTTP nebo gRPC aplikační rozhraní, díky kterému je možné integrovat správu TTS do jiných aplikací. The Things Industries nabízí kromě LNS také vlastní modely bran, určené jak na venkovní, tak na vnitřní instalaci. Brány je možné použít s kteroukoliv implementací LNS, nejen TTS. Architektura TTS je velmi podobná obecné architektuře popsané v podsekci 3.1.1. Skládá se z *Application Server*, *Network Server*, *Join Server*, *Gateway Server* a *Identity Server* (viz obrázek 3.3). Rozdílem od obecné architektury je Identity Server, který slouží pro uchování základních identifikátorů a informací o zařízeních. S tímto serverem mohou v případě potřeby komunikovat všechny ostatní servery.



Obrázek 3.3: Architektura TheThingsStack (TTS) LoRaWAN Network serveru. Převzato z [23]. TTS se skládá z několika serverů. Každý z nich plní odlišnou roli. Na obrázku je znázorněno, který server komunikuje s kterým. Speciální je Identity Server, který může komunikovat se všemi ostatními. Hlavním serverem pro interakci s jinými systémi a aplikacemi je Application Server.

Registrace zařízení

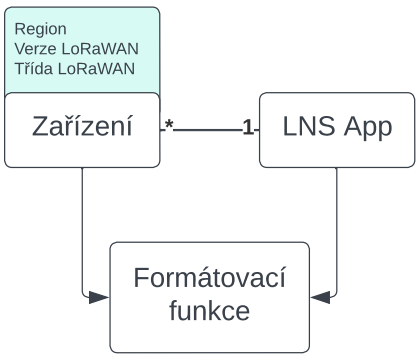
Každé zařízení zaregistrované do The Things Stack (TTS) je nutné přiřadit k předem vytvořené aplikaci. Aplikace shlukují zařízení stejného či podobného typu. Mimo základní organizační strukturu slouží aplikace také k definování formátovacích funkcí přenášených dat mezi LNS a koncovými zařízeními. Bez těchto funkcí by LNS nebyl schopný interpretovat data od zařízení a správně je přeposlat dál do systému. Je možné použít předdefinované formátovací funkce, které byly výrobcem zařízení umístěny do TTS repozitáře zařízení. Pokud pro určité zařízení neexistuje formátovací funkce definovaná výrobcem, nebo pokud existující funkce nemá jako výstup vyžadovaný formát dat, je možné vytvořit novou formátovací funkci pomocí jazyku JavaScript. Je potřeba dát pozor na to, že TTS podporuje

pouze verzi ECMAScript 5.1 z roku 2009, která se značně liší od moderních verzí jazyka. Vždy je potřeba mít pro každou aplikaci definovanou jak downlink tak uplink formátovací funkci. Pokud je potřeba mít v jedné aplikaci více různých druhů zařízení, je možné definovat formátovací funkce na úrovni jednotlivých zařízení. Pokud jsou specifikovány funkce jak u aplikace, tak u zařízení, zařízení má vyšší prioritu. Diagram struktury entity je zobrazen na obrázku 3.4.

Jelikož je TTS složen z několika komponent – Network Server, Application Server, Join Server a Identity Server (které jsou blíže popsány v podsekci 3.1.1) – je nutné při registraci zařízení poslat přes API data o novém zařízení každé z těchto komponent zvlášť. Posloupnost volání rozhraní jednotlivých komponent je při registraci následující (pro odregistrování je přesně opačná):

1. Identity Server,
2. Join Server,
3. Network Server,
4. Application Server.

Prvotní POST požadavek na Identity server nese identifikátory zařízení, identifikátor aplikace pod kterou zařízení patří a adresy Join, Network a Application serverů, ke kterým bude následně zařízení registrováno. Další kroky registrace jsou provedeny pomocí PUT požadavků. Join server ukládá základní identifikátory zařízení, adresy Network a Application serverů a hodnotu *appKey*, která slouží pro zabezpečení první aktivace zařízení. Network server vyžaduje zaslat stejné identifikátory jako Identity server a dále ukládá informace o verzi LoRaWAN kterou zařízení podporuje, regionálních parametrech pro LoRa rádio, zda zařízení podporuje i jiné LoRaWAN třídy než *A* (viz podsekci 3.1.3) a zda zařízení podporuje *OTAA* aktivaci (viz podsekci 3.1.3). Application server stačí informovat o existenci zařízení a předat mu identifikátory zařízení.



Obrázek 3.4: Struktura TTS entit, do kterých jsou ukládány informace o zařízeních. Každé zařízení patří k právě jedné aplikaci. Formátovací funkce je možné definovat jak na úrovni aplikace, tak na úrovni zařízení. Jestliže jsou definovány na obou úrovních, zařízení má prioritu. Zařízení v jedné aplikaci mohou teoreticky podporovat rozdílné verze specifikace LoRaWAN a podporovat rozdílné třídy LoRaWAN, jelikož tyto informace jsou ukládány zvlášť pro každé zařízení.

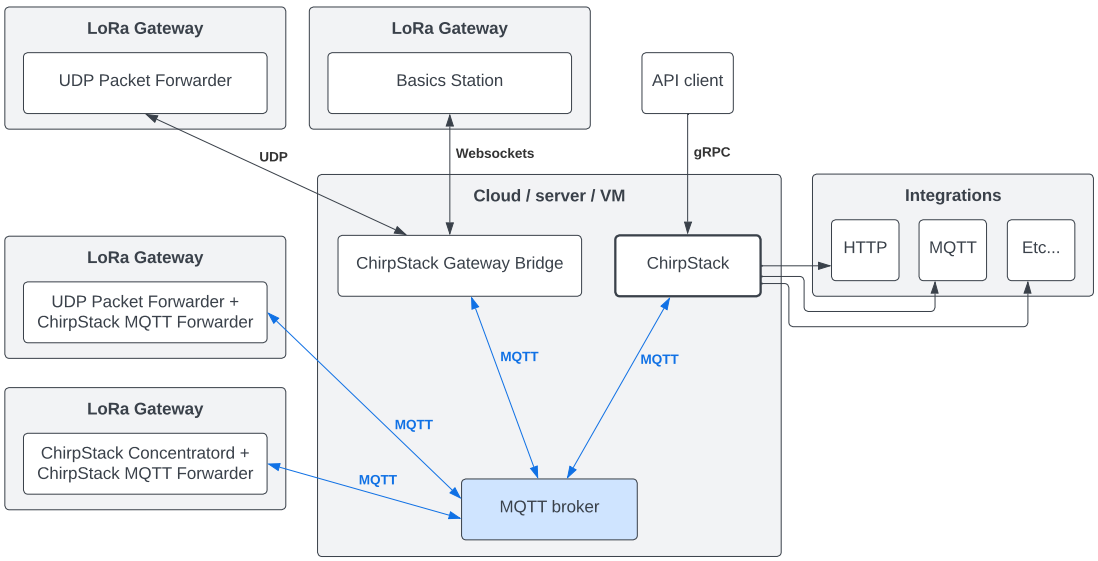
3.2.2 ChirpStack

ChirpStack je plně open source implementace LoRaWAN Network Serveru. Hlavní komponenty LNS jsou implementovány v programovacím jazyce Rust. Je vyvíjen pod vedením zakladatele Orne Brocaar.

O nasazení serveru ChirpStack se musí plně postarat uživatel. Není poskytováno žádné cloudové řešení. Instalace je možná na jakýkoliv linuxový server pomocí *Docker*[1] kontejneru. Dále je k dispozici balíček pro Debian či Ubuntu distribuce a v neposlední řadě je možné nasadit celý LNS na Raspberry Pi pomocí speciální distribuce linuxu.

Nejnovější verze ChirpStack je `v4`, která nahrazuje předchozí verzi `v3`. Mezi změny které ChirpStack `v4` přináší patří například změna formátu identifikátorů aplikací a uživatelů, dále drobné změny v gRPC API a také odstranění implicitní podpory HTTP API. Tu je ale pořád možné explicitně povolit pomocí aktivování komponenty, která překládá REST požadavky na gRPC požadavky [7].

Na obrázku 3.5 je vidět diagram architektury serveru ChirpStack. Tato architektura se liší od té popsáné v podsekci 3.1.1 spojením většiny funkcionality LNS do jedné komponenty. *Network Server*, *Application Server* a *Join Server* jsou implementovány dohromady hlavní částí nazývanou *ChirpStack* (tento název může značit jak vlastní implementaci hlavní funkcionality LNS, tak celý projekt, který zahrnuje i další komponenty). Pro překlad LoRaWAN paketů, které jsou od zařízení do LNS přeposlány bránami, na MQTT pakety slouží *ChirpStack Gateway Bridge*.



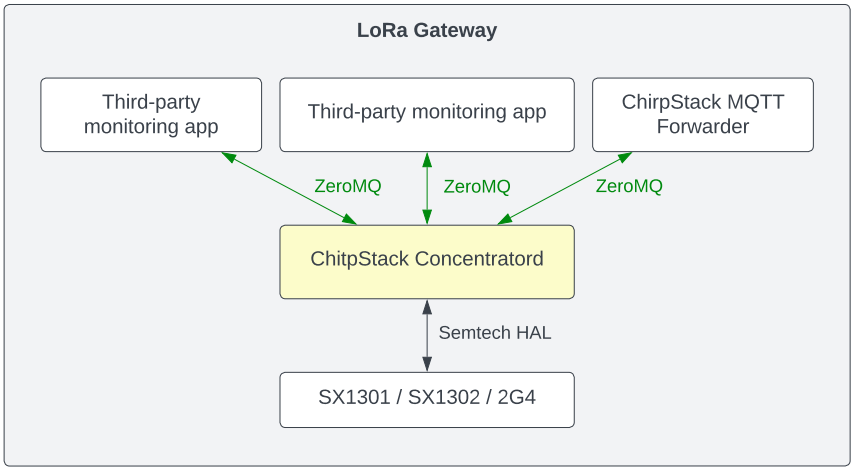
Obrázek 3.5: Architektura ChirpStack LoRaWAN Network serveru. Převzato z [7]. Kromě hlavní komponenty ChirpStack (označené tučným rámečkem), která implementuje hlavní funkcionality LoRaWAN Netwrok Server, projekt ChirpStack také obsahuje komponenty pro překlad paketů bran na MQTT zprávy. Tento překlad může probíhat buď až na serveru, nebo již na vlastní bráně.

Mimo LoRaWAN Network Server projekt ChirpStack také nabízí open-source abstrakční vrstvu pro brány LoRa s názvem *ChirpStack Concentratord*, která je kompatibilní s LoRa

[1] https://www.docker.com/

modemy `SX1301/8` a `SX1302`. Nabízí rozhraní založené na protokolu *ZeroMQ*, ke kterému se mohou připojit jiné aplikace běžící na bráně. Jednou z takových aplikací může být například *ChirpStack MQTT Forwarder*. Díky ní je možné připojit bránu přímo na MQTT broker LNS a vynechat tak ChirpStack Gateway Bridge. Architektura brány která používá ChirpStack komponenty je vidět na obrázku 3.6. Na ZeroMQ API je možné připojit i vlastní aplikaci, která může například sloužit k monitorování stavu brány [6].

Kromě MQTT API přes které mohou uživatelské aplikace dostávat zprávy od LNS, poskytuje ChirpStack řadu možností jak dostávat či reagovat na příchozí zprávy. Patří mezi ně například HTTP API, RabbitMQ API, *If This Than That* (IFTTT) integrace, nebo přímý zápis do *InfluxDB* databáze či *PostgreSQL* databáze. Všechny způsoby integrace je možné najít v ChirpStack dokumentaci [6].



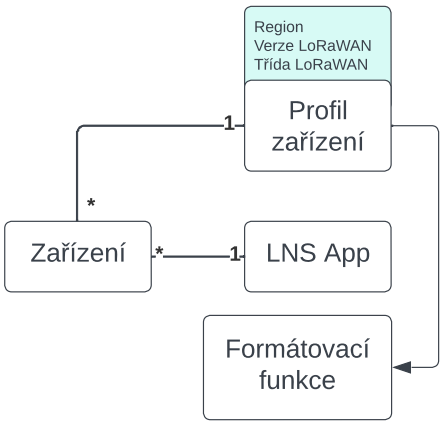
Obrázek 3.6: Architektura ChirpStack brány. Převzato z [7]. Centrálním prvkem brány je implementace hardwarové abstrakční vrstvy ChirpStack Concentratord. Ta komunikuje s rádiem LoRa a přeposílá zprávy dál do systému protokolem ZeroMQ.

Registrace zařízení

Před registrací je třeba mít vytvořenou aplikaci, ke které bude následně zařízení přiřazeno. Aplikace zde slouží primárně pro vytvoření organizace zařízení a pro nastavení integrací LNS s externími aplikacemi. Formátovací funkce nelze definovat na úrovni aplikací, ale pouze na úrovni jednotlivých zařízení.

Další entitou, kterou je nutné před registrací zařízení vytvořit, je takzvaný *profil zařízení* (angl. *device profile*). Ten je možné popsat jako šablonu pro určitý model zařízení. Obsahuje všechny informace, které budou mít všechna zařízení tohoto modelu stejné. Obsahuje především informace o hardware zařízení a nastavení rádia LoRa. Mimo to se v profilu zařízení také definují formátovací funkce. Ty je možné vytvořit v jazyce JavaScript verze ECMAScript 5.1. Profily zařízení není potřeba vytvářet ručně, ChirpStack obsahuje velké množství profilů pro různá zařízení. Vlastní zařízení poté obsahuje pouze informace, které se pro dvě zařízení stejného modelu mohou lišit. Jsou to zejména identifikátory, informace o stavu zařízení, nebo statistiky přenesených dat. Diagram struktury entit je vidět na obrázku 3.7.

Pokud jsou splněny všechny prerekvizity pro registraci nového zařízení (existuje aplikace a profil zařízení), je možné registraci provést jedním voláním API na koncový bod `/api/devices`. V něm je třeba předat jméno zařízení, unikátní identifikátor zařízení (*devEui*), identifikátor aplikace pod kterou bude zařízení registrováno a identifikátor profilu zařízení. Údaj *appEui* (viz podsekci 3.1.3) není potřeba při registraci vůbec poskytnout. ChirpStack získá tuto hodnotu při první komunikaci se zařízením a tak ji nevyžaduje zadat od uživatele. Pro uložení hodnoty *appKey*, která je využita při aktivaci zařízení stylem *OTAA* (viz podsekci 3.1.3) je potřeba provést další POST požadavek na koncový bod `/api/devices/devEui/keys`.



Obrázek 3.7: Struktura ChirpStack entit, do kterých jsou ukládány informace o zařízeních. Každé zařízení patří k právě jedné aplikaci. Aplikace slouží pouze pro organizaci zařízení. Všechny neměnné informace o zařízení jsou uloženy v profilu zařízení. Ten také definuje formátovací funkce pro dané zařízení.

Kapitola 4

Analýza

Správci systémů chytrých měst, ve kterých je použita technologie LoRaWAN, musí momentálně mít relativně dobrou znalost konkrétního LoRaWAN Network Serveru, jež jejich systém využívá. LNS poskytují grafické rozhraní pro správu zařízení, většinou ve formě webové aplikace, které je ale specifické pro danou implementaci LNS. Tato rozhraní se také snaží zprostředkovat všechny možné operace nejen se zařízením, ale i dalšími entitami v systému LNS. Pro získání užitku ze systému chytrého města je potřeba i aplikace na aplikační úrovni architektury IoT systému (viz obrázek 2.1). Nutnost využívání rozdílných aplikací specifických pro daný LNS ubírá na uživatelské přívětivosti systému pro správu chytrých měst. Znemožňuje praktické využívání více LNS implementací v rámci jednoho IoT systému a také brání správě zařízení uživatelům systému, kteří nejsou odborníci na daný LNS.

Tato práce je vypracovávána ve spolupráci s firmou Logimic[1], která vyvíjí systém pro správu chytrých měst. Tento systém sbírá data z různých senzorů, která následně umožňuje zobrazovat a analyzovat. Jedná se o webovou aplikaci, která se skládá z uživatelského rozhraní a serverové části, která implementuje aplikační logiku systému.

4.1 Uživatelé systému

Typickými uživateli tohoto systému jsou správci chytrých měst, kteří mají různou úroveň technických znalostí. Nejčastější operací prováděnou v rámci systému je registrace nových koncových zařízení LoRaWAN. Tato činnost bude nejčastěji prováděna na mobilním zařízení během instalace koncových zařízení v terénu. Od těchto uživatelů by neměla být vyžadována téměř žádná znalost LNS. Měli by pouze rozumět tomu, jak jsou zařízení v systému Logimic seskupena pod takzvané modely zařízení a jaké identifikátory zařízení jsou potřeba pro registraci.

Od uživatelů, kteří se zabývají správou již registrovaných koncových zařízení, se očekává, že budou mít větší znalosti technických detailů týkajících se LNS. Tito uživatelé budou systém používat především na větších zařízeních, jako jsou stolní počítače. I když tito uživatelé mají o něco lepší znalosti o konkrétním systému LNS, nejsou odborníky na tyto technologie. Zaměřují se spíše na pochopení platformy Logimic a způsob jejího ovládání. Vědí více o tom, jak modely zařízení souvisejí s aplikacemi LNS a jak tyto entity v systému nastavit.

Problematickou operací pro správce je migrace zařízení na jiný LNS. Pro provedení této operace je nutné, aby uživatel uměl pracovat a měl přístup k aplikacím jednotlivých

[1] https://logimic.com

LNS. Proces přenosu zařízení je pomocí webových aplikací možný pouze po jednom. Pro přenos většího množství zařízení najednou by uživatel musel vytvořit vlastní skript, který by komunikoval se servery pomocí jejich API. Tato úroveň technických znalostí není u cílových uživatelů systému Logimic očekávána.

4.2 Aktuální stav aplikace pro správu chytrých měst

Systém vyvíjený firmou Logimic slouží pro správu zařízení v chytrých městech. Cloudová část systému sbírá data z řady senzorů, která jsou poté pomocí webové aplikace prezentována uživateli. Systém dále například umožňuje nastavit upozornění na změny hodnot senzorů. To je konfigurovatelné pomocí již zmíněné webové aplikace. IoT zařízení, se kterými systém často pracuje jsou například chytré pouliční osvětlení, senzory kvality ovzduší, teploměry a senzory stavu vody v nádržích.

Pro přidání nového zařízení je momentálně nutné pracovat s aplikací poskytovatele LNS a některé specifické informace pro systém Logimic se musí poté manuálně přidat do databáze. V případě změny poskytovatele LNS je nyní zapotřebí u něj manuálně zrušit registraci zařízení a poté je zaregistrovat u nového poskytovatele. Prototyp mobilní aplikace, která dokáže naskenovat QR kód zařízení a přihlásit jej na server TheThingsStack, je popsán v práci [16]. Na základě tohoto prototypu se firma Logimic rozhodla pro rozšíření jejich stávajícího systému o tuto funkcionalitu.

4.3 Požadavky na řešení

Nejdůležitější schopností řešení je přesunutí veškerých potřebných operací pro správu koncového zařízení do stejné aplikace, ve které je spravován i zbytek systému chytrého města. Aplikace by měla nabízet jednotné rozhraní, které umožní práci s více implementacemi LNS. Je důležité, aby uživatel měl přehled o všech svých zařízeních, nehledě na LNS platformu. Dále by měl uživatel být co nejvíce odstíněn od rozdílů jednotlivých platforem. Tam, kde to není možné, by měly rozdíly být jasně vysvětleny. Řešení musí poskytovat možnost přesunout jedno nebo více zařízení z jedné implementace LNS na jinou, aby nebyli uživatelé systému vázáni na konkrétní LNS.

S aplikací budou hodně pracovat zaměstnanci města v terénu během instalace zařízení. Je potřeba, aby byla plně responzivní a optimalizovaná pro mobilní zařízení. S tím také souvisí zjednodušení nejčastěji prováděných akcí, zejména při použití z mobilního zařízení. Jedná se především o registraci nového zařízení. Za tímto účelem musí aplikace umožňovat registraci pomocí naskenování QR kódu zařízení, který je poskytnutý výrobcem. Pokud není možné provést registraci pomocí QR kódu, musí být možné zadat všechny potřebné údaje do intuitivního formuláře.

4.4 Zhodnocení

Aplikace LNS spolu nejsou kompatibilní a je možné spravovat pouze daný server. Žádná z nich nenabízí jednoduchý způsob přesunu zařízení na jiný LNS a znemožňují tak snadnou migraci na jiný systém. Server TTS umožňuje registrovat nové zařízení pomocí QR kódu, ale jsou předvyplněny pouze některé informace nutné pro registraci, zbytek musí uživatel vyplnit manuálně. ChirpStack registraci pomocí QR kódu neumožňuje. Systém firmy Logi-

mic neumožňuje spravovat koncová zařízení na LNS serverech vůbec. Je ale responzivní a architektura systému umožňuje jej rozšířit o požadovanou funkcionalitu.

Všechny popsané LNS mají odlišnou strukturu entit. S každým z nich je možné pracovat pomocí jeho konkrétního API. I když se interní struktura entit i způsob práce s API liší, všechny tyto LNS nabízí velmi podobnou funkcionalitu a je tedy možné navrhnout aplikaci, která bude fungovat jako abstrakční vrstva nad těmito API a která bude splňovat požadavky na řešení.

Kapitola 5

Návrh řešení

Během celé fáze návrhu bylo důležité mít na paměti dva hlavní případy užití. Tedy registraci nových zařízení, která bude nejčastěji probíhat na menším mobilním zařízení, a správu existujících zařízení, která bude většinou probíhat na větší obrazovce, například na stolním počítači. Začal jsem tím, že jsem nakreslil drátěné modely (tzv. *wireframy*) možných rozložení UI prvků pro oba hlavní případy použití. Podle nejlepších variant jsem poté vytvořil náhledy uživatelského rozhraní (tzv. *mockups*) v nástroji *Figma*[1], který umožňuje snadné experimentování s rozložením UI a také vytvoření interaktivního návrhu, kde lze nastavit, která tlačítka přepnou zobrazení na kterou obrazovku návrhu a simulovat tak průchod uživatelským rozhraním. Po vytvoření návrhů uživatelského rozhraní bylo snazší zjistit, které koncové body API budou potřeba a jaká data potřebuje uživatelské rozhraní, což bylo užitečné při následném návrhu API. Návrh mobilního rozložení uživatelského rozhraní je vidět na obrázku 5.1.

S ohledem na odlišnou strukturu vnitřních entit jednotlivých LNS (podrobněji popsáno na obrázcích 3.7 a 3.4) jsem navrhl datový model, který slouží jako sjednocující bod mezi těmito různými strukturami a zároveň se dobře integruje do vnitřní datové struktury systému Logimic. Podrobnosti o datovém modelu jsou popsány dále v sekci 5.2.

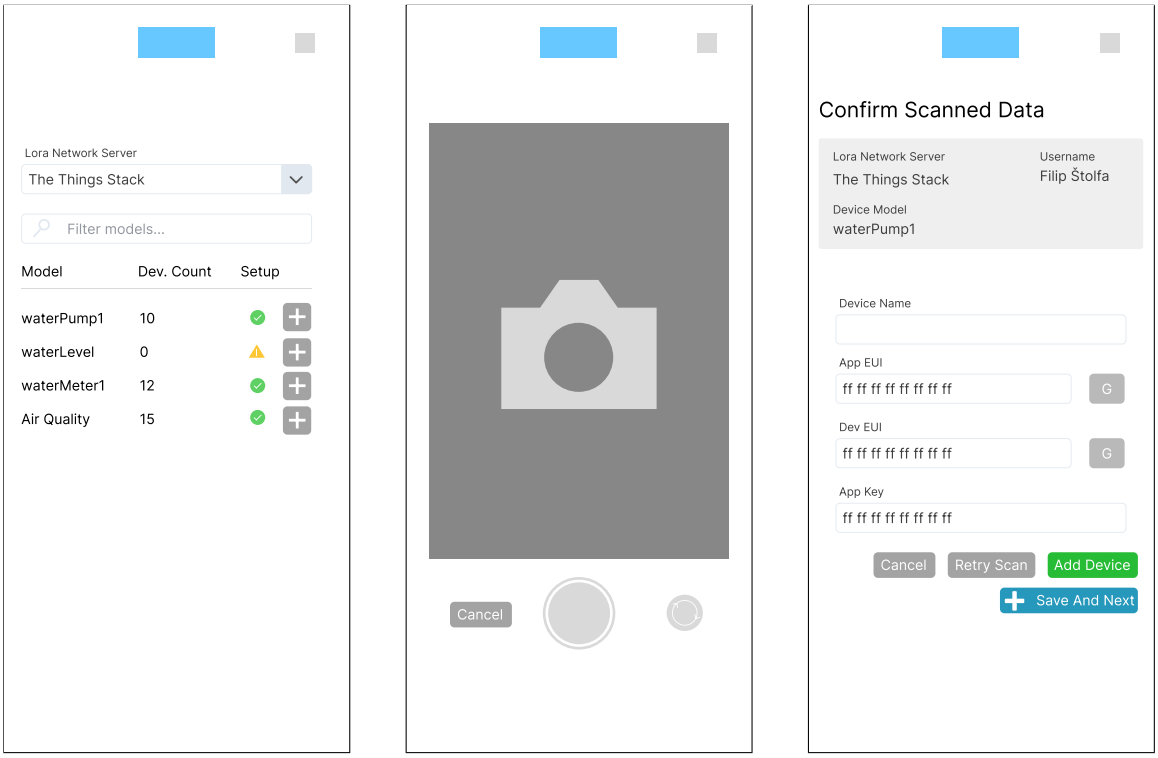
5.1 Uživatelské rozhraní

Jak vyplývá z analýzy uživatelských požadavků, registrace nových zařízení musí být co nejjednodušší s co nejmenšími možnostmi vnášení chyb do systému. Jediné, o co by se měl uživatel při registraci starat, je výběr správného LNS a správného modelu zařízení.

Prvním krokem, který uživatel při registraci nového zařízení provede, je výběr LNS. Většina systémů nebude nikdy připojena k více než pěti různým LNS, proto jsem se rozhodl použít jako mechanismus výběru na mobilních zařízeních rozbalovací nabídku. Volby budou tvořeny zobrazovanými názvy různých LNS, protože ponesou všechny informace, které uživatel potřebuje k identifikaci požadovaného LNS. Rozbalovací výběrová nabídka se stává tím méně použitelnou, čím více je v ní možností, ale v tomto případě to nebude problém.

Po výběru LNS musí uživatel vybrat, k jakému typu zařízení patří zařízení, které se chystá zaregistrovat. Na rozdíl od výběru LNS by zde rozbalovací výběrový prvek nebyl dobrou volbou. Počet možných typů zařízení bude narůstat s tím, jak bude systém přidávat podporu nových zařízení. A protože i nová verze téhož zařízení má v systému svůj

[1] https://www.figma.com/



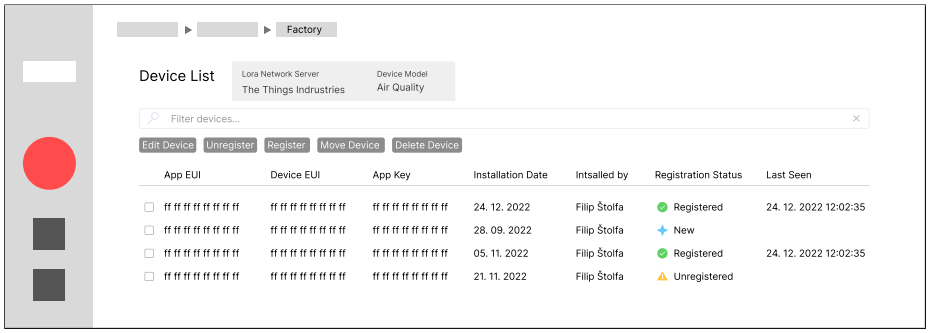
(a) Seznam modelů zařízení (b) Skener QR kódu (c) Potvrzení skenu

Obrázek 5.1: Návrh UI procesu registrace zařízení na mobilním zařízení. Obrazovka 5.1a zobrazuje seznam modelů zařízení pro vybraný LNS. Na obrazovku 5.1b s otevřenou kamerou zařízení je možné se dostat stisknutím tlačítka se symbolem plus v řádku požadovaného modelu. Na obrazovce 5.1c je zobrazen formulář, který slouží pro potvrzení naskenovaných dat.

vlastní model zařízení, může se seznam v budoucnosti poměrně hodně rozrůst. Proto by jeho zobrazení formou jednoduché rozbalovací nabídky velmi ztížilo nalezení požadovaného modelu zařízení a nebylo by uživatelsky přívětivé. Místo toho jsem se rozhodl, že seznam dostupných modelů zařízení bude zobrazen jako jednoduchá tabulka, která bude zobrazovat alespoň název modelu zařízení, počet zařízení registrovaných pod tímto modelem a zda je plně nastaven pro použití s vybraným LNS. Zobrazení těchto údajů ve formě tabulky umožňuje jejich snadné rozšíření o další informace v budoucnu. Použití tabulky také umožňuje snadnou implementaci filtrování a řazení, což může výrazně usnadnit vyhledání požadovaného modelu zařízení.

Po výběru požadovaného LNS a modelu zařízení může uživatel zahájit proces registrace otevřením čtečky QR kódů. Pokud uživatel neudělí oprávnění pro přístup ke kameře nebo dojde k jiné chybě bránící otevření kamery zařízení, otevře se místo toho formulář pro ruční registraci. Po naskenování platného QR kódu se otevře formulář vyplněnými naskenovanými údaji, což uživateli umožní ověřit platnost těchto údajů a také doplnit registrační formulář o informace, které nejsou obsaženy v QR kódu. Uživatel pak může zařízení zaregistrovat a zavřít registrační formulář, nebo může pokračovat v registraci jiného zařízení stejného modelu. K dispozici tedy budou dvě odlišná tlačítka, jedno pro jednorázovou registraci a druhé pro provádění hromadných registrací.

Pokud není prováděna registrace nového zařízení, jsou úvodní dva kroky použití uživatelského rozhraní stejné – výběr z dostupných LNS a výběr modelu zařízení. Místo otevření okna registrace může uživatel otevřít detail vybraného modelu zařízení, který zobrazí tabulku zařízení, jež patří pod vybranou kombinace LNS a modelu zařízení. Návrh této obrazovky je vidět na obrázku 5.2. Tato tabulka zobrazuje jedinečné identifikátory každého zařízení, jeho název a stav registrace. Kromě toho se zobrazí datum instalace a datum poslední komunikace se zařízením. Po zobrazení této tabulky zařízení bude moci uživatel vybrat jedno nebo více zařízení a provést následující operace: odhlásit zařízení z aktuálního LNS, zaregistrovat zařízení do aktuálního LNS, přesunout zařízení do jiného LNS a odstranit zařízení. Odstranění se liší od zrušení registrace, protože při smazání dojde k odstranění jak z LNS, tak z databáze. Tuto operaci na rozdíl od zrušení registrace nelze vrátit zpět. Protože všechny tyto operace mohou být potenciálně destruktivní, musí aplikace před jejich provedením požádat uživatele o potvrzení. Tato tabulka musí umožňovat filtrování a řazení podle všech sloupců.



Obrázek 5.2: Návrh tabulky všech zařízení pro zvolený LNS a model zařízení. Nad tabulkou se nachází tlačítka pro provádění operací s vybranými zařízeními. Zařízení lze vybrat pomocí tlačítka v prvním sloupci tabulky.

5.2 Datový model

Většina LNS sdružuje zařízení do tzv. aplikací. TheThingsStack LNS (TTS) umožňuje nastavení formátovacích funkcí na úrovni aplikace, kdy každé zařízení bude používat stejné formátovací funkce. ChirpStack LNS zachází s formátovacími funkcemi jinak. Jsou specifikovány v tzv. profilech zařízení, které si lze představit jako šablony zařízení. Obsahují veškeré informace o zařízení, které se mezi různými zařízeními stejného modelu nebudou měnit. TTS takový koncept profilů zařízení nemá, místo toho se všechny tyto informace opakují a jsou uloženy v každé entitě zařízení.

Systém Logimic má podobný koncept jako profily zařízení, nazývá se modely zařízení. Každé zařízení, se kterým systém pracuje, je přiřazeno k modelu zařízení. Uložení informací týkajících se sítě LoRaWAN do modelu zařízení by omezilo systém Logimic na práci pouze se zařízeními LoRaWAN. Místo toho budou tyto informace uloženy v tabulce nazvané `LnsApp`. Každá aplikace `LnsApp` bude patřit přesně jednomu modelu zařízení a jednomu LNS. Pokud má tedy systém nastaveny dva LNS, bude mít každý model zařízení dvě aplikace `LnsApp`. Tabulka LnsApp obsahuje identifikátor aplikace, která se nachází na konkrétním LNS. Obsahuje také informace o verzi LoRaWAN a regionu daného modelu zařízení. Zařízení je reprezentováno tabulkou `LoraDevice`. V ní jsou uloženy informace, které se pro každé zaří-

zení stejného modelu zařízení liší, například jeho jedinečné identifikátory. Když je zařízení zaregistrováno, je spojeno s aplikací `LnsApp`, která jej také spojuje s konkrétním LNS. Pokud chce uživatel toto zařízení přesunout na jiný LNS, systém se pokusí najít aplikaci `LnsApp` pro model zařízení v cílovém LNS. Pokud ji nalezne, zařízení se odhlásí z aktuálního LNS a zaregistruje se na cílovém LNS. V databázi je to označeno propojením zařízení s nalezenou aplikací LnsApp.

Zařízení může být také ve stavu, kdy není registrováno na žádném LNS. To je reprezentováno příznakem `registered` uloženým u každého zařízení. Toto zařízení bude stále propojeno s aplikací `LnsApp`, ke které bylo naposledy registrováno, ale nebude přítomno na LNS.

Při srovnání se strukturami entit obou LNS diskutovaných v této práci, je tato struktura databáze podobnější té, kterou implementuje LNS ChirpStack. Tabulka `LnsApp` slouží k podobnému účelu jako entita profilu zařízení v systému ChirpStack. V TTS je třeba tyto informace opakovat v každém jednotlivém zařízení.

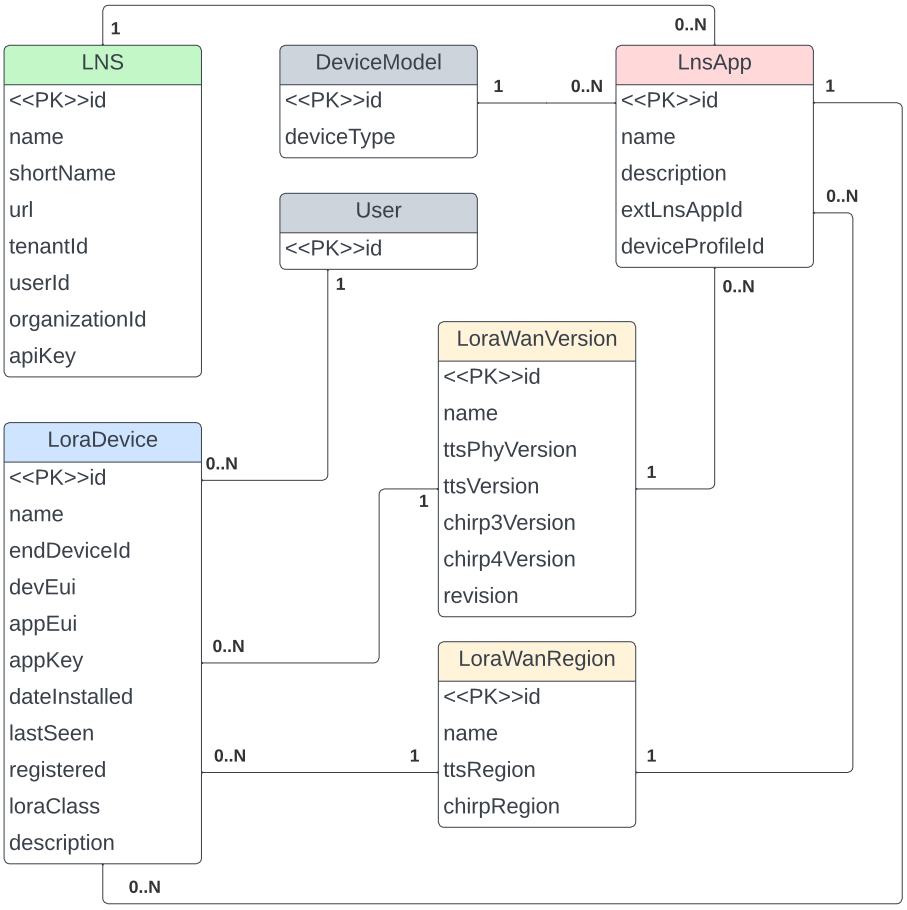
Tabulka `Lns` obsahuje informace o jednotlivých instancích LNS se kterými může daný systém pracovat. Aby bylo možné pracovat s několika instancemi stejného LNS, v tabulce se ukládá URL a API klíč pro danou instanci LNS. Tento klíč umožňuje systému pracovat s API daného LNS. Je důležité, aby klíč API byl dostupný pouze na serverové části aplikace a nikdy nebyl přenesen přes REST API. Tabulka také obsahuje sloupec `shortName`, který slouží pro identifikaci typu LNS a k výběru obslužných funkcí LNS, jež budou použity pro komunikaci s tímto LNS. Systém podporuje práci s TTS verze 3, ChirpStack verze 3 a ChirpStack verze 4. Diagram datového modelu je vidět na obrázku 5.3.

5.3 Komunikace s LoRaWAN Network Servery

Pro každý LoRaWAN Network Server je potřeba implementovat funkce, které budou provádět potřebné operace a dohromady budou poskytovat stejné veřejné rozhraní, tak aby došlo k odstínění rozdílů jednotlivých LNS. Operace, které musí být pro každý LNS implementovány jsou:

1. registrace zařízení,
2. odregistrace zařízení,
3. úprava zařízení,
4. výpis všech aplikací,
5. výpis všech zařízení,
6. výpis všech zařízení pro danou aplikaci.

S oběma LNS bude komunikace probíhat pomocí REST aplikačního rozhraní. To bude podle zvoleného LNS volat konkrétní implementace operací a také bude pracovat s databází a udržovat tak systém v synchronizovaném stavu. Pokud dojde k tomu, že stav uložený v databázi neodpovídá stavu na LNS, systém provede synchronizaci obou stran. Pokud se v databázi nachází zařízení, které by podle hodnoty `registered` mělo být registrováno na LNS, ale není, budou upraveny informace v databázi. Zvolil jsem tuto variantu, místo provedení registrace, jelikož k tomuto stavu může dojít pouze tak, že uživatel s přístupem na LNS odregistruje zařízení přímo přes ovládací aplikaci daného serveru. Informace o stavu registrace zařízení je v databázi měněna pouze tehdy, pokud opravdu dojde k registraci/odregistraci zařízení. Není tedy možné, aby nastala situace, kdy registrace na LNS



Obrázek 5.3: ER diagram systému. `DeviceModel` a `User` představují již existující tabulky v databázi firmy Logimic, jsou tedy zobrazeny jen relevantní atributy.

selže a v databázi bude přesto zařízení označeno jako registrované. Stejně tak pokud bude v databázi uvedeno, že zařízení není registrováno k LNS, ale ve skutečnosti je, pak bude opravena hodnota v databázi. Jestliže bude na LNS objeveno zařízení, které není zavedeno v systému, provede se uložení tohoto zařízení do databáze. Přehled situací, kdy není stav databáze a LNS stejný, je společně s jejich řešením znázorněn v tabulce 5.1.

Databáze	**LNS**	**Operace**
Zařízení registrováno	Zařízení neregistrováno	Úprava databáze
Zařízení neregistrováno	Zařízení registrováno	Úprava databáze
Zařízení nevedeno v systému	Zařízení registrováno	Úprava databáze
Rozdílné jméno, popisek, či appKey		Úprava LNS dle databáze

Tabulka 5.1: Situace, kdy stav databáze a LNS není synchronizovaný a jejich řešení

Kapitola 6

Implementace

Výsledné řešení je implementováno jako rozšíření webové aplikace firmy Logimic. Díky využití webových technologií je aplikaci možné používat na široké škále zařízení, z čehož nejdůležitější je použitelnost aplikace na mobilních zařízeních pro registraci koncového zařízení v terénu. Pro implementaci byl použit programovací jazyk *TypeScript* a to jak pro uživatelské rozhraní, tak pro serverovou část aplikace. Právě možnost implementace obou částí systému ve stejném jazyce je jednou z velkých výhod jazyka TypeScript.

Uživatelské rozhraní je implementováno v aplikačním rámci *Angular*, který umožňuje rozdělení aplikace do znovupoužitelných komponent, což nejenom značně usnadňuje vývoj a následnou údržbu aplikace, ale také vede k rychlejší aplikaci, jelikož Angular je při změnách stavu aplikace schopný rozpoznat, které části aplikace je potřeba překreslit a které není nutné měnit.

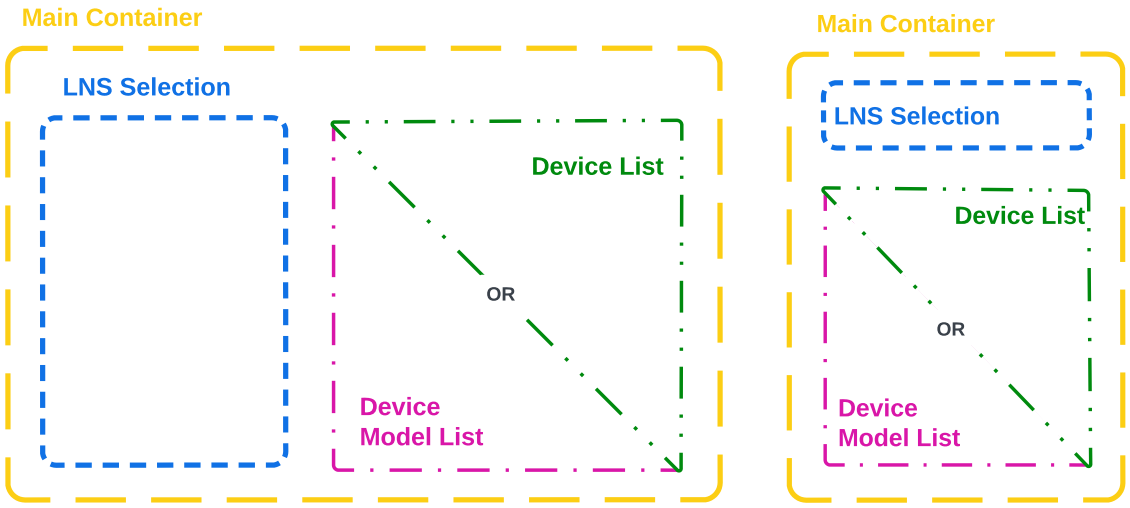
Aplikační logika je implementována tak, že je ji možné provozovat dvěma způsoby: na cloudové platformě *AWS Lambda*[1], nebo na klasickém serveru pomocí aplikačního rámce *Express.js*[2].

Data jsou ukládána v *PostgreSQL*[3] databázi, kterou je možné provozovat jak na platfomě *AWS* díky službě *AWS RDS* tak na vlastním, lokálně provozovaném serveru. Během procesu implementace byla databáze spouštěna lokálně jako *Docker*[4] kontejner, serverová část byla spouštěna jako vývojová verze Express.js a uživatelské rozhraní bylo možné testovat díky lokálnímu vývojovému serveru rámce Angular.

6.1 Uživatelské rozhraní

Komponenty jsou v aplikačním rámci Angular tvořeny z několika částí, jež jsou často rozděleny zvlášť do souborů. Logika komponenty je definována jako třída jazyka TypeScript s dekorátorem `@Component`. Uvnitř tohoto dekorátoru se také definuje takzvaný *selector*, tedy identifikátor, pod kterým bude komponenta dostupná ve zbytku aplikace. Dále se zde definují `templateUrl` a `styleUrls`. Hodnota `templateUrl` určuje, jaký soubor obsahuje HTML šablonu pro danou komponentu a `styleUrls` ukazují na CSS soubory, ve kterých jsou definovány styly.

[1] https://aws.amazon.com/
[2] https://expressjs.com/
[3] https://www.postgresql.org/
[4] https://www.docker.com/



(a) Rozložení na stolním počítači (b) Mobilní rozložení

Obrázek 6.1: Ilustrace hierarchie hlavních komponentů uživatelského rozhraní, zobrazuje jejich uspořádání na stolním počítači 6.1a a na mobilním zařízení 6.1b

Jako základní stavební bloky pro tvorbu uživatelského rozhraní jsou využity komponenty z knihovny *PrimeNG*[5], které obsahují jak předem definované styly, tak plně funkční logiku pro často využívané prvky UI. Příkladem je komponenta tabulky pro zobrazování dat. Ta implementuje běžné operace nad daty v tabulce (řazení, filtrace, ...), takže pro běžné situace stačí komponentě poskytnout data pro zobrazení. Zároveň jsou tyto komponenty však navrženy tak, aby šlo jejich chování jednoduše upravit.

Přehled hlavních komponent je zobrazen na obrázku 6.1. Prvním krokem při práci s aplikací je výběr LNS, o což se stará komponenta *Lns Selection*. Po výběru LNS se zobrazí seznam modelů zařízení, který je zobrazen v tabulce *Device Model List*. Jakmile uživatel vybere s jakým modelem zařízení chce pracovat, je místo tohoto seznamu zobrazen seznam zařízení, která patří pod tento model. Tento seznam implementuje komponenta *Device List*. Nadřazená komponenta *Main Container* rozhoduje o tom, který seznam bude zobrazen. Na obrázku 6.1b je vidět, jak se změní rozložení na mobilním zařízení. Hlavní komponenty jsou dále popsány více do detailů.

O udržování stavu aplikace se stará služba implementovaná třídou `FactoryStateService` v souboru *factory-state.service.ts*. Ke službě je možné přistoupit z jakékoliv komponenty a následně registrovat funkci, která bude volána při každé změně dat. Pokud komponenta potřebuje na základě nových dat provést další operace, je to možné právě v této funkci. Aktuálně vybraný LNS a model zařízení jsou uloženy v URL jako parametry dotazu. Díky tomu je možné sdílet URL, které přímo zobrazí seznam zařízení pro daný LNS a model, případně si uložit záložku pro konkrétní LNS. Příklad 1 ukazuje kód, který provede změnu parametrů dotazu.

Změna stavu aplikace je provedena voláním metody `navigate` služby `router` rámce Angular. Ve službě `FactoryStateService` je následně zaregistrovaná funkce, která je zavolána při každé změně parametrů dotazu. Podle druhu změny provede tato funkce potřebné operace pro aktualizaci stavu.

[5] https://primeng.org/

```
this.router.navigate(
  ['.'],
  {
    relativeTo: this.route,
    queryParams: { lns: "1" },
    queryParamsHandling: "merge"
  });
```

Příklad 1: Ukládání stavu do URL aplikace pomocí služby `router` rámce *Angular*. Tento příklad nastaví parametr dotazu `?lns=1`. Díky `['.']` je specifikována navigace na aktuální stránku, tedy nedojde ke změně hlavní části URL. Parametry dotazu se nastaví dle objektu `queryParams`.

6.1.1 Výběr LNS

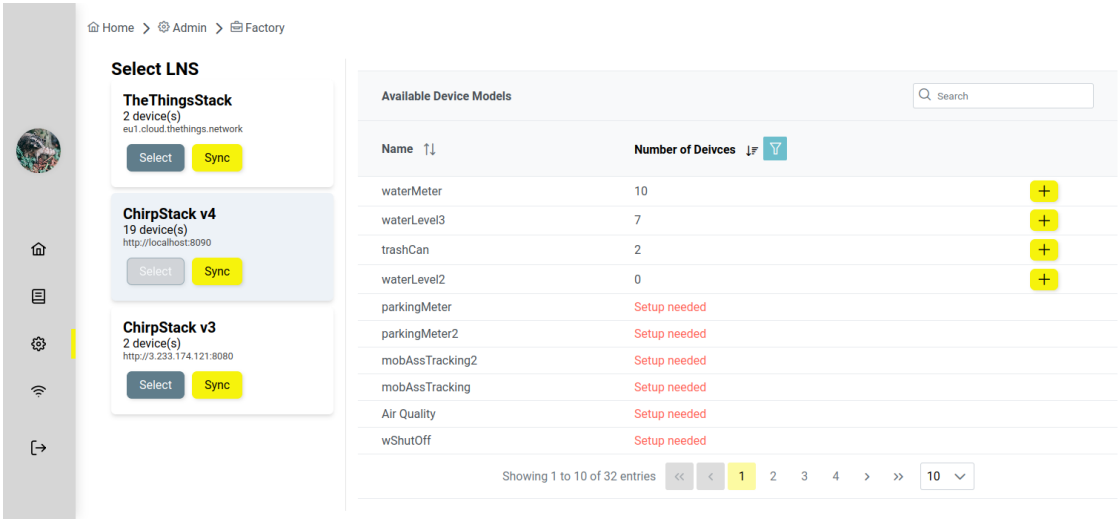
V mobilním rozložení je výběr LNS proveden pomocí jednoduché rozbalovací nabídky, aby bylo co nejvíce ušetřeno místo. Na větších obrazovkách není potřeba šetřit místem a je možné jej využít pro zobrazení více informací. LNS bude v systému většinou maximálně pět. Na větších obrazovkách jsou proto zobrazeny jako seznam jednoduchých karet, které obsahují informace jako název serveru, URL serveru, počet zařízení přiřazených k tomuto serveru a tlačítko pro synchronizaci serveru se stavem uloženým v databázi. Toto tlačítko je zobrazeno pouze pokud uživatel má oprávnění označené řetězcem `admin/factory/sync`. Jelikož na mobilním zařízení se nejčastěji uživatel potřebuje dostat ke skeneru QR kódů pro registraci zařízení, nejsou tyto dodatečné informace tolik užitečné. Zatímco na větších obrazovkách bude uživatel chtít provádět správu již registrovaných zařízení a je vhodné mu poskytnout co nejvíce informací o jednotlivých serverech. Seznam karet LNS je vidět na levé straně snímku obrazovky 6.2.

Změna zobrazené komponenty pro výběr LNS je provedena pomocí CSS pravidel `media queries`, která umožňují aplikovat rozdílné styly podle velikosti obrazovky. Obě komponenty mají přiřazené rozdílné CSS třídy, pomocí kterých je na malých obrazovkách seznam karet LNS skryt nastavením CSS stylu `display: none`. Na obrazovkách, které mají alespoň `992px` je takto skryta naopak rozbalovací nabídka.

6.1.2 Seznam modelů zařízení

V systému pro správu chytrých měst firmy Logimic patří každé zařízení k jednomu určitému modelu zařízení. Při registraci nového LoRa zařízení je nutné nejprve zvolit cílový LNS a poté model zařízení. Po zvolení LNS se zobrazí tabulka se všemi dostupnými modely zařízení. Tato tabulka zobrazuje jméno modelu a počet zařízení v systému, která patří tomuto modelu na daném LNS. Aby mohl systém registrovat zařízení na LNS, musí být ke každému modelu přiřazena aplikace LNS. Všechna zařízení stejného modelu jsou registrována pod stejnou aplikaci LNS. Pokud není k tomuto modelu v systému evidována aplikace LNS pro daný server, v tabulce se zobrazí varování o nutnosti přiřazení aplikace. Výběr modelu se provede kliknutím na řádek v tabulce. Po zvolení modelu s přiřazenou aplikací se místo seznamu modelů zobrazí tabulka zařízení. Pokud model aplikaci nemá, zobrazí se komponent pro nastavení aplikace LNS (viz podsekci 6.1.3). V posledním sloupci tabulky je tlačítko, které umožňuje registraci zařízení pro daný model. Registraci je možné provést i z tabulky zařízení, to ovšem obnáší načtení dalších, pro tuto operaci zbytečných, dat z databáze.

V tabulce je možné vyhledávat a řadit modely podle jména a počtu zařízení. Pro filtrování na základě přiřazené aplikace LNS bylo nutné implementovat vlastní filtrovací funkci. Jedná se totiž o hodnotu, kterou je nutné odvodit od dat, která nejsou přímo zobrazena v tabulce a žádná z dostupných filtrovacích funkcí z knihovny PrimeNG nebyla vyhovující. Knihovna PrimeNG umožňuje vytvářet vlastní filtrovací funkce pomocí služby `FilterService`. Ta má metodu `register`, která vyžaduje jméno nové filtrovací funkce jako první argument a poté vlastní funkci, jež bude volána pro každý řádek tabulky a na základě výsledku `true` nebo `false` určí, zda daný řádek zobrazit či ne.



Obrázek 6.2: Snímek obrazovky seznamu dostupných modelů zařízení pro vybraný LNS ChirpStack. U modelů které jsou připravené k použití s tímto LNS je zobrazen jejich počet zařízení, pokud model není připraven k použití, je tato informace uvedena místo počtu zařízení.

6.1.3 Konfigurace aplikací LNS

Tato komponenta zobrazuje tabulku dostupných aplikací na vybraném LNS. Po zvolení aplikace z tabulky se zobrazí formulář s informacemi které budou uloženy do databáze. Některé informace nelze měnit, jelikož jsou vázány na zvolenou externí aplikaci LNS. Dále jsou ve formuláři pole pro informace, které nejsou na LNS uloženy v entitě aplikace. Jelikož jsou tyto informace stejné pro všechna zařízení v dané aplikaci, rozhodl jsem se je uložit společně s ostatními informacemi o aplikaci. Blíže je struktura dat popsána v sekci 5.2. Konkrétně se jedná o verzi specifikace LoRaWAN, frekvenci rádia LoRa a pokud to LNS vyžaduje, také identifikátor profilu zařízení.

Seznam možných hodnot verzí LoRaWAN a frekvencí rádia LoRa je načten z databáze, aby byla zajištěna konzistence mezi uživatelským rozhraním a aplikační logikou. Až vyjde nová verze specifikace LoRaWAN, stačí ji přidat do databáze a bude možné s ní pracovat v celém systému.

V tabulce externích aplikací jde vyhledávat a filtrovat podle všech sloupců. Pokud požadovaná externí aplikace na LNS neexistuje a uživatel ji musí na LNS nejprve vytvořit, je možné aktualizovat seznam externích aplikací bez nutnosti aktualizace celé stránky, aby uživatel nepřišel o doposud zadané informace.

6.1.4 Seznam zařízení

Jakmile je zvolen LNS a model zařízení, je možné zobrazit tabulku zařízení. Každé LoRa zařízení má přiřazené dva identifikátory, jeden zvaný *devEui* a druhý *endDeviceId*. Kromě těch je v tabulce zobrazeno jméno zařízení, datum posledního kontaktu se zařízením, datum instalace a stav registrace. Aby neměla tabulka příliš mnoho sloupců, jsou dvě dodatečné hodnoty zařízení zobrazeny pouze po rozbalení detailu zařízení. Konkrétně se jedná o hodnoty *appKey* a *appEui*, které slouží pro proces prvotní aktivace zařízení na daném LNS.

Rozbalení řádku tabulky je implementováno komponentou `rowexpansion` z knihovny PrimeNG. Každý řádek tabulky obsahuje tlačítko s direktivou `pRowToggler`, pomocí které je spojeno s šablonou `rowexpansion`. V této šabloně je možné zobrazit libovolný obsah, který bude mít přístup k datům z řádku tabulky, ke kterému patří. Dodatečné informace o zařízení jsou zobrazeny pomocí malé tabulky vložené do zmíněné šablony `rowexpansion`.

Datum instalace a datum poslední komunikace se zařízením je formátováno pomocí funkce `date` z rámce Angular, kterou je možné využít přímo v HTML šabloně komponenty. Tato funkce se stará o zobrazení správného formátu data dle nastavení uživatelova prohlížeče.

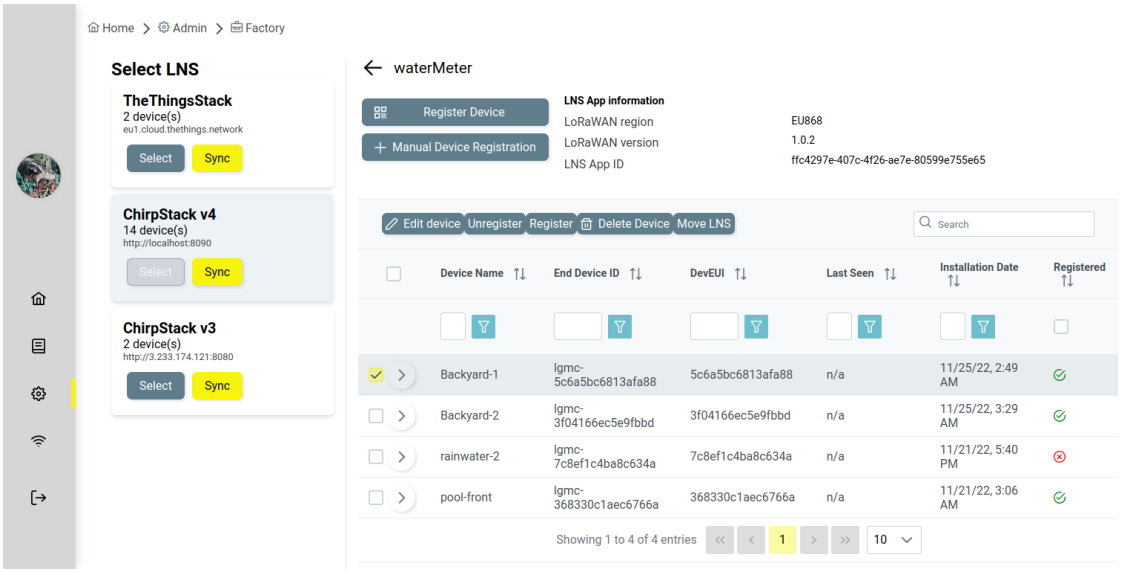
Jako ostatní tabulky aplikace lze data filtrovat a řadit podle všech sloupců. Aby šlo v tabulce efektivně vyhledávat, je možné kromě globálního vyhledávacího pole využít také vyhledávací pole pro konkrétní sloupec a sestavit tak přesnější filtry. Toho je docíleno přidáním druhého řádku do hlavičky tabulky, který v každém sloupci obsahuje komponentu `p-columnFilter` z knihovny PrimeNG. Ta je atributem `field` spojena s daty tabulky, nad kterými bude provádět filtrování. Atribut `type` této komponenty určuje datový typ sloupce, na základě čehož je zobrazen odpovídající prvek pro filtrování. Pro řetězec je zobrazen textový vstup, pro booleovskou hodnotu je zobrazeno zaškrtávací políčko a podobně. Mimo zadání hledané hodnoty, je také možné specifikovat způsob filtrování. Zda se musí vyhledávaná hodnota rovnat přesně, má být menší nebo stačí například jen výskyt podřetězce.

V záhlaví tabulky se nachází tlačítka pro editaci zařízení, odregistrování z LNS, registrování na LNS, kompletní smazání zařízení a přesun na jiný LNS. Tyto operace mají právo provádět pouze uživatelé s oprávněním `admin/factory/edit`. Pokud uživatel toto oprávnění nemá, jsou tlačítka skryta pomocí `ngIf` direktivy.

Všechny operace se zařízením je nutné potvrdit ve vyskakovacím okně. To je implementováno pomocí služby `ConfirmationService` z knihovny PrimeNG. Po stisknutí tlačítka je možné v obslužné metodě komponenty zavolat metodu ze služby `ConfirmationService`, které je předán text jež má být zobrazen ve vyskakovacím okně. Následně jsou metodě předány dvě funkce, první se zavolá pokud uživatel potvrdí operaci a druhá v případě že zamítne. Všechny operace (mimo editaci zařízení) je možné provádět hromadně. Operace je provedena pro všechna vybraná zařízení. Výběr je proveden pomocí zaškrtávacího políčka v prvním sloupci tabulky. Pokud operace pro některé zařízení selže, zůstane vybrané a informace o chybě jsou uživateli prezentovány pomocí vyskakovací zprávy. Tato komponenta je vidět na obrázku 6.3.

6.1.5 Registrace zařízení

Celý proces registrace zařízení je zaštiťován jednou komponentou, která dle výběru uživatele může otevřít skener QR kódu, nebo formulář pro manuální vložení informací. Aby mohl uživatel registrovat nová zařízení, musí mít oprávnění `admin/factory/register`. Pokud si uživatel vyžádá skenování QR kódu, otevře se vyskakovací okno, které se pokusí spustit kameru zařízení a naskenovat QR kód. Pro skenování je využita knihovna *@zxing/ngx-*



Obrázek 6.3: Snímek obrazovky seznamu dostupných zařízení, které patří pod model *waterMeter* na serveru ChirpStack verze 4. Nad tabulkou je tlačítko pro registraci zařízení k tomuto modelu. Zařízení v prvním řádku je vybrané pomocí tlačítka v prvním sloupci. V posledním sloupci je zobrazen stav registrace jednotlivých zařízení.

scanner[6]. Tato knihovna poskytuje komponentu `zxing-scanner`. Ta se stará jak o získání oprávnění pro otevření kamery, tak o vlastní vyhledání a přečtení QR kódu v obraze. Při úspěšném naskenování QR kódu tato komponenta vyšle zprávu s naskenovanými daty.

Na zprávu s daty reaguje nadřazená komponenta, která provede kontrolu formátu naskenovaných dat. Zpracování dat probíhá v metodě `parseQrPayload()`. Nejprve je vstupní řetězec rozdělen na jednotlivé části, které jsou odděleny dvojtečkou (`:`). První kontrolou formátu dat je počet nalezených částí. Pokud jich je méně jak 5, nejedná se o platný QR kód. Dále je ověřena správná hodnota identifikátoru formátu dat (první část), která musí být rovna `LW`. Následně je ověřena správná délka ostatní částí a formát hodnot *devEui* a *appEui/joinEui* je ověřen pomocí regulárního výrazu. Podrobněji je formát dat QR kódu rozebrán v podsekci 3.1.3. V případě správného formátu vrátí funkce objekt se získanými daty, jinak vrátí informace o zjištěné chybě.

Pokud byl naskenován platný QR kód, získané informace se zobrazí ve formuláři pro registraci, kde je možné zkontrolovat správnost těchto informací a doplnit ty informace, které nejsou obsaženy v QR kódu. Zejména se jedná o jméno zařízení, *endDeviceId* a *appKey*. Jméno zařízení a *endDeviceId* můžou být libovolně zvoleny uživatelem. Pro ulehčení procesu zadávání informací je *endDeviceId* předvyplněno a odvozeno od *devEui*. Pokud byly informace naskenovány špatně nebo byl naskenován špatný QR kód, uživatel se může vrátit zpět na krok skenování. Stejně tak pokud se nedaří kód naskenovat, je možné rovnou otevřít formulář pro manuální registraci. Po úspěšné registraci zařízení se okno s formulářem uzavře. Pokud uživatel chce registrovat více zařízení naráz, může využít tlačítko `Uložit a další`, což po úspěšné registraci vrátí uživatele zpět na okno s kamerou a může pokračovat v registraci dalšího zařízení.

Pro lepší přehlednost je toto okno na mobilních zařízeních zobrazeno přes celou obrazovku. Aby bylo možné nastavit styly okna pro roztáhnutí přes celou obrazovku, je v této

[6] https://github.com/zxing-js/ngx-scanner

komponentě využita služba rámce Angular `BreakpointObserver`. Ta umožňuje přihlásit se k odběru a reagovat na změny velikosti okna. Definuje se minimální/maximální rozměr okna, jehož překročením dojde k provedení přihlášené funkce. V té je poté možné reagovat na tuto změnu. Jedná se o stejný princip jako u CSS `media queries`, ale umožňuje na základě těchto událostí provádět libovolné operace.

Komponenta obsahující formulář pro registraci zařízení musí být schopna informovat nadřazenou komponentu o stavu validace prvku `form`. Aby byl tento prvek, který je definován v HTML šabloně komponenty dostupný i v TypeScript kódu, je použit dekorátor `@ViewChild`. Ten je pomocí zvoleného identifikátoru spojen s prvkem v HTML a do proměnné kterou dekoruje, vloží odkaz na tento prvek.

6.2 Aplikační logika

Serverová část řešení je implementována tak, aby ji bylo možné zprovoznit jak na vlastním serveru pomocí aplikačního rámce *Express.js*, tak jako serverless aplikaci na platformě *AWS Lambda*. Implementace je tedy rozdělena na následující části:

- operace s externími *LoRaWAN Network Servery* (LNS),
- databázové operace,
- obslužné funkce koncových bodů API,
- definice koncových bodů REST API,
- definice koncových bodů REST API – AWS Lambda.

Kvůli nutnosti definovat koncové body API zvlášť pro Express.js a pro AWS Lambda je nutné co nejvíce omezit replikaci kódu. V obou případech jsou funkce definující koncové body API co nejjednodušší a pouze specifikují požadované parametry dotazu. Jedinou operací v těle těchto funkcí je volání funkce implementující databázové operace nebo obslužné funkce API. Tímto způsobem je nutné duplikovat pouze volání dané funkce. Pokud daný koncový bod představuje operaci, která pouze vyčítá data z databáze, je volána přímo funkce implementující tuto databázovou operaci. V případech, kdy je nutné pro daný požadavek provádět komplikovanější operace, je volána funkce z modulu obslužných funkcí. Tyto funkce mohou interně volat funkce databázových operací a provádět operace s externími LNS. Díky tomuto rozdělení lze snadno rozpoznat, kdy odpověď na požadavek vyžaduje složitější a potenciálně pomalejší operace, nebo pouze práci s databází.

Data přenášená přes aplikační rozhraní REST mezi uživatelským rozhraním a serverem mají definované typy v souboru `api/factory/factoryTypes.d.ts`. Systém firmy Logimic využívá specifikaci *OpenAPI* pro udržování konzistence mezi těmito dvěma částmi aplikace. Tato specifikace je automaticky generovaná podle definic koncových bodů REST API. Z této vygenerované specifikace rozhraní jsou poté dále automaticky generovány funkce ve frontend části pro provádění HTTP dotazů. Ukázka takového páru funkcí je vidět na příkladu 2. Pro usnadnění podpory OpenAPI specifikace slouží aplikační rámec *tsoa*[7], který staví nad Express.js.

[7] https://tsoa-community.github.io/docs/

```
// ===== Server ======
@Get('get/lns')
@Middlewares([RahErrorMiddleware.lazy])
async getLnsList(
  @Header() conname: string,
  @Query() devCount?: boolean
): Promise<MnLns[]> {
  return await this.factoryDbService.getLnsList(devCount);
}

// ===== Frontend ======
this.factSrvc.getLnsList({conname: this.connName, devCount: true})
```

Příklad 2: Definice koncového bodu API (nahoře), podle kterého je na frontendu vygenerována funkce, která provede daný HTTP požadavek (dole). Typy parametrů jsou zachovány mezi serverovou a frontendovou funkcí.

6.2.1 Databáze

Pro práci s databází je využita knihovna *Typeorm*[8], která poskytuje *ORM* (objektově relační mapování/zobrazení) funkcionalitu, což umožňuje definovat tabulky v databázi pomocí TypeScript tříd. Data z databáze jsou automaticky převedena na instance daných tříd a není nutné provádět manuální kontrolu, zda data získané z databáze mají tvar, který očekáváme. Kromě převodu dat na objekty umí Typeorm také automaticky z definic tříd vytvořit tabulky v databázi, což je užitečné zejména při vývoji, kdy může docházet ke změnám v tabulkách. Tyto třídy jsou definovány v souboru `FactoryEntities.ts`. Struktura databáze je implementována podle ER diagramu 5.3.

Operace s databází jsou implementovány v souboru `FactoryDbService.ts`. Typeorm umožňuje vytvářet SQL dotazy pomocí řetězení metod. Díky tomu není třeba znát přesné detaily o SQL syntaxi dané databáze.

Funkce implementující tyto operace vracejí typy požadované REST rozhraním. Převod mezi Typerorm typy a REST API typy je prováděn metodami `copyFrom()` a `copyTo()`, jež jsou implementovány třídami v souboru `FactoryDbTypesImpl.ts`. Tyto třídy jsou označeny jako implementace REST API typů pomocí klíčového slova `implements`. Typový systém jazyka TypeScript tak může upozornit na chybné či chybějící atributy.

6.2.2 Operace s externími LNS

Pro každý LoRaWAN Network Server jsem implementoval obslužné metody ve třídě, která implementuje rozhraní popsané v příkladu 3. Při prvotním spuštění serveru jsou vytvořeny a registrovány instance obslužných tříd pro každý LNS, který je uložen v databázi. Každé instanci jsou předány informace o LNS, které obsahují URL a klíč API daného LNS, podporované regiony a verze LoRaWAN. Tyto informace jsou nastaveny voláním metody `setLnsInfo()`.

Instance obslužných tříd LNS jsou poté uloženy do hashovací tabulky, která je v jazyce TypeScript implementovaná třídou `Map()`. Klíčem tabulky je `id` LNS z databáze. Při

[8] https://typeorm.io/

obsluhování požadavku REST API je nutné poskytnout `id` LNS, dle kterého je následně vyhledáno ve zmíněné hashovací tabulce.

V případě, že nastane chyba při práci s API LNS, je tato chyba dále propagována pomocí výjimek. Na ty je možné zareagovat, pokud lze chybový stav opravit, avšak většinou jsou tyto chyby propagovány až do obslužné funkce REST API, kde jsou dále poslány jako chybová odpověď na požadavek. Takže je možné chybové hlášky zobrazit uživateli na frontendu pomocí vyskakovacích zpráv.

```
interface ILns {
  registerDevice: (device: MnLoraDevice, app: MnLnsApp) => Promise<boolean>;
  unregisterDevice: (device: MnLoraDevice, app?: MnLnsApp) => Promise<boolean>;
  listAllApps: (offset?: number, limit?: number) => Promise<MnLnsApp[]>;
  listAllAppDevices: (
    appId: string,
    inclAppKey?: boolean,
    offset?: number,
    limit?: number
  ) => Promise<MnLoraDevice[]>;
  editDevice: (device: MnLoraDevice, app: MnLnsApp) => Promise<boolean>;
  listAllDevices: (offset?: number, limit?: number) => Promise<MnLoraDevice[]>;
  setLnsInfo: (
    lns: LnsWithApiKey,
    versions: CatLoraWanVersion[],
    regions: CatLoraWanRegion[]
  ) => void;
}
```

Příklad 3: Rozhraní, které musí být implementováno pro každý LNS, se kterým umí systém pracovat. Každá implementace tohoto rozrhaní musí umět převést interní datové typy na univerzální typy, se kterými pracuje zbytek systému.

ChirpStack

Systém podporuje jak verzi `3`, tak verzi `4` serveru ChirpStack. Rozdíly v API obou verzí jsou dostatečně velké na to, aby bylo nutné pro každou verzi implementovat obslužnou třídu zvlášť. Liší se především strukturou objektů přenášených přes API a formátem identifikátorů aplikací LNS.

Operace se zařízením je většinou možné provést jedním HTTP požadavkem. Veškeré informace o zařízení jsou uloženy na jednom serveru. Pro identifikaci zařízení v HTTP požadavcích se využívá *devEui*. Identifikátory aplikací, profilů zařízení a jiných entity jsou automaticky generovány serverem ChirpStack a není možné je měnit.

Při převádění na univerzální formát zařízení (viz entita `LoRaDevice` z obrázku 5.3) je kromě objektu zařízení z koncového bodu `devices/{devEui}` potřeba poskytnout i objekt profilu zařízení. V tom je uloženo, jakou verzi specifikace LoRaWAN zařízení podporuje a jakou využívá frekvenci. Hodnotu `appKey` je nutné získat z koncového bodu `devices/{devEui}/keys`. Server ChirpStack nazývá hodnotu `appKey` jako `nwkKey`, pokud se jedná o zařízení podporující verzi specifikace LoRaWAN z řady `1.0.x`. Pro zařízení

verze `1.1` je tato hodnota uložena pod názvem `appKey`. Při nastavování této hodnoty je tedy nutné poslat objekt se správným klíčem dle verze zařízení.

TheThingsStack

Zařízení jsou při práci s aplikačním rozhraním TTS identifikována pomocí *endDeviceId*. Tento identifikátor je vytvořen uživatelem při registraci zařízení. Uživatel musí zajistit unikátnost identifikátoru.

Při registraci zařízení na TTS je nutné provést 4 volání aplikačního rozhraní (viz podsekci 3.2.1). Každé vyžaduje jiný formát dat a jiné informace o zařízení. Pro získání všech informací potřebných při převádění na univerzální formát zařízení (viz entita `LoRaDevice` z obrázku 5.3) je nutné získat informace ze tří serverů TTS: *Identity Server*, *Network Server*, *Join Server*.

TTS vyžaduje přesně specifikovat, jaké informace o zařízení mají být poskytnuty při práci s API. Slouží k tomu parametr dotazu `?field_mask`. Například pro získání jména a data poslední komunikace zařízení je nutné nastavit parametr dotazu na `?field_mask=name, last_seen_at`.

Kapitola 7

Testování

Tato kapitola popisuje testování během celého procesu návrhu a implementace řešení. V sekci 7.1 je popsáno testování uživatelského rozhraní. Sekce 7.2 popisuje testování aplikační logiky. Následně jsou v sekci 7.3 popsány závěry z testování a návrhy na zlepšení.

7.1 Testování uživatelského rozhraní

Uživatelské rozhraní je komplikované na testování, jelikož cílem není pouze ověřit funkcionalitu aplikace, ale také uživatelskou přívětivost. To, jestli je uživatelské rozhraní přívětivé, nelze hodnotit kompletně objektivně. Záleží například na zkušenostech a znalostech uživatele, ale také na osobních preferencích. Proto je vhodné začít s touto částí testování co nejdříve a získat více názorů.

Testování uživatelského rozhraní začalo již ve fázi vytváření návrhu. Návrh rozhraní byl vytvářen pomocí nástroje *Figma*[1]. V tomto nástroji je možné udělat interaktivní návrh uživatelského rozhraní, kterým je možné se proklikávat a testovat tak průchod uživatelským rozhraním. Díky stylu vytváření návrhu je možné snadno a rychle návrh upravovat a zkoušet tak alternativní rozložení. Tento návrh byl poskytnut firmě Logimic a na základě zpětné vazby byl upravován.

Na základě poznatků z testování návrhu byl implementován prototyp uživatelského rozhraní. Během této fáze implementace nebyla vůbec implementována aplikační logika a prototyp pracoval pouze se statickými daty. Díky tomuto prototypu bylo jednodušší navrhnout potřebné koncové body aplikačního rozhraní.

Responzivita uživatelského rozhraní byla při implementaci testována pomocí vývojářských nástrojů v prohlížečích *Chrome* a *Firefox*. Tyto nástroje umožňují dynamicky měnit velikost okna webové stránky a simulovat tak zařízení s různými rozměry obrazovek. Kromě toho byla využita možnost zpřístupnit vývojový server *Angular* a *Express.js* do lokální sítě, díky čemu je možné aplikaci otevřít na mobilních zařízeních, které jsou připojené do lokální sítě. Aplikaci je tak možné otestovat na opravdových zařízeních. To je důležité pro testování použitelnosti uživatelského rozhraní pomocí dotykové obrazovky, což není možné kompletně simulovat pomocí vývojářských nástrojů.

Funkčnost skeneru QR kódů jsem testoval pomocí vytištěných kódů různých velikostí. Příklad QR kódu zařízení je vidět na obrázku 7.1. Tyto QR kódy obsahovaly náhodně vygenerované identifikátory a různé kombinace nepovinných rozšíření dat QR kódu (viz podsekci 3.1.3). Pro otestování skeneru na mobilním zařízení je potřeba k aplikaci přistupovat přes

[1] https://www.figma.com/

protokol `https`. Jinak není z bezpečnostních důvodů umožněno používat kameru zařízení. Rámec Angular umožňuje spustit vývojový server s přepínačem `-ssl`, který vygeneruje certifikát a zpřístupní aplikaci přes protokol `https`. Po otevření stránky v prohlížeči se sice zobrazí hláška o neplatném certifikátu, tu je však možné odkliknout.

Velikost nejmenšího QR kódu, který je možný naskenovat, záleží na kvalitě kamery zařízení. Testoval jsem skenování pomocí HD web kamery, několika Android mobilních telefonů a telefonu iPhone 13. Nejmenší QR kód, který byl čitelný všemi zařízeními, byl kód o straně čtverce 1,5 cm.

Obrázek 7.1: Příklad QR kódu který identifikuje koncové zařízení LoRaWAN sítě. Kód obsahuje data, která jsou podrobněji popsána a zobrazena na obrázku 3.2. Tento kód obsahuje i nepovinné informace, bez kterých by mohl mít menší rozměry.

7.2 Testování aplikační logiky

Během vývoje bylo REST API testováno pomocí nástroje *Insomnia*[2]. Tento nástroj umožňuje opakovaně provádět HTTP požadavky, nastavit jejich parametry, cookies a těla požadavků. Insomnia je vhodná jak na rychlé experimentování s API během prvotního návrhu, tak na následné sestavení testů, ve kterých je možné specifikovat tvar očekávané odpovědi, případně očekávanou chybovou zprávu.

Stejným nástrojem jsem testoval i správnost práce s externími LoRaWAN Network Servery. Insomnia umožňuje v jednom testu provést několik HTTP požadavků za sebou. Je tedy možné navrhnout test, který se nejdříve pokusí zaregistrovat zařízení na daný LNS pomocí mého rozhraní a následně provede dotaz na daný LNS pomocí jeho rozhraní a ověří, že požadované zařízení bylo skutečně správně registrováno. Pro účely testování jsem implementoval funkci, která generuje náhodné identifikátory zařízení, ze kterých jsem sestavil fiktivní zařízení a testoval jsem s nimi různé operace.

V prvotních fázích implementace aplikační logiky byl pro testování komunikace s LNS využíván server ChirpStack verze 4, nasazený na Raspberry Pi v lokální síti. Později byla přidána podpora pro TheThingsStack a pro její testování sloužila komunitní edice TheThingsStack na platformě TheThingsNetwork. V této fázi bylo možné testovat přesouvání zařízení z jednoho serveru na druhý. Testování této operace odhalilo velké množství chyb a pomohlo doladit návrh datového modelu. Jako poslední byla přidána podpora serveru ChirpStack verze 3, která byla otestována na serveru firmy Logimic.

[2] https://insomnia.rest/

7.3 Výsledky testování

Na základě testování aplikace odpovídá a splňuje stanovené požadavky na řešení. Aplikace ulehčuje registraci koncových LoRa zařízení na LNS a poskytuje jednotné rozhraní pro dva různé LNS. Díky tomuto sjednocení má správce zařízení snadnější přehled o stavu na jednotlivých LNS. Aplikace je použitelná jak na mobilním zařízení, tak na větších obrazovkách. Uživatelské rozhraní na mobilním zařízení je optimalizované pro registraci zařízení.

Z testování uživatelského rozhraní vyplynulo několik podnětů k vylepšení a dalšímu vývoji. Prvním z podnětů jsou lepší chybová hlášení na straně uživatelského rozhraní. Dále rozložení tabulky a formuláře při vytváření nové aplikace LNS. Tato část je problematická obzvlášť na mobilních zařízeních, kde není po vybrání aplikace z tabulky jasné co dělat dál, jelikož nově zobrazený formulář není vidět bez posunutí stránky dolů. Možným řešením je automaticky posunout stránku pomocí TypeScript kódu, minimalizování tabulky po výběru nebo zobrazení formuláře ve vyskakovacím okně.

Kapitola 8

Závěr

Cílem této práce bylo vytvořit jednotné rozhraní pro správu LoRa zařízení na dvou rozdílných centrálních serverech sítě LoRaWAN. Toto rozhraní bylo implementováno jako rozšíření systému pro správu chytrých měst firmy Logimic.

V teoretické části jsem probral problematiku internetu věcí – architektury těchto systémů, různé odvětví IoT a často používané komunikační technologie. Více do detailu jsem rozebral komunikační technologii LoRa a LoRaWAN. Nastudoval jsem architekturu sítí těchto technologií a potřebné komponenty pro provozování těchto systémů. Zaměřil jsem se hlavně na koncová zařízení a centrální servery. Podrobně jsem nastudoval dva konkrétní servery pro správu LoRaWAN zařízení – TheThingsStack a ChirpStack. Porovnal jsem rozdíly v jejich struktuře a způsobu, jakým ukládají data o registrovaných zařízeních. Rozebral jsem formát QR kódů, které slouží pro rychlou registraci zařízení. Identifikoval jsem potřeby uživatelů a nejčastější případy užití. Na základě těchto požadavků jsem vytvořil návrh uživatelského rozhraní, které je vhodné jak pro mobilní zařízení, tak pro stolní počítače. Na základě potřeb, které vyplynuly z návrhu uživatelského rozhraní jsem navrhl aplikační rozhraní REST serverové části aplikace.

Dle návrhu jsem implementoval responzivní webovou aplikaci pomocí aplikačního rámce Angular. Aplikační logiku a serverovou část systému jsem implementoval pomocí kombinace technologií Express.js, aplikačního rámce tsoa a databázové knihovny Typeorm. Poskytované HTTP aplikační rozhraní umožňuje pracovat s různými servery LoRaWAN sítě. Zprávy aplikačního rozhraní mají stejný tvar nehledě na typ serveru LoRaWAN sítě. Uživatel tohoto rozhraní nemusí rozumět rozdílům v práci s jednotlivými implementacemi serverů. Během implementace jsem systém testoval jak manuálními testy a používáním aplikace na různých zařízeních, tak pomocí jednotkových testů pro aplikační rozhraní a databázové funkce.

Výsledná aplikace, která byla integrována do systému firmy Logimic, ulehčuje registraci a správu zařízení na dvou odlišných centrálních serverech LoRaWAN sítě pomocí jednotného rozhraní. Uživatel nemusí pro správu zařízení v ním spravovaném systému využívat několik odlišných aplikací a manuálně udržovat tyto systémy synchronizované. Možným rozšířením aplikace je automatická tvorba aplikací na serverech LoRaWAN sítě, přidání podpory pro další servery, případně umožnění správy integrací se službami třetích stran.

Literatura

[1] Andrew Banks, K. B. a Gupta, R. *MQTT Version 5.0* [online]. [cit. 2023-25-01]. Dostupné z: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[2] Bandyopadhyay, D. a Sen, J. Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Personal Communications*. Springer Science and Business Media LLC. apr 2011, sv. 58, č. 1, s. 49–69. DOI: 10.1007/s11277-011-0288-5.

[3] Banks, A. a Gupta, R. *MQTT Version 3.1.1* [online]. [cit. 2023-25-01]. Dostupné z: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[4] Broring, A., Schmid, S., Schindhelm, C.-K., Khelil, A., Kabisch, S. et al. Enabling IoT Ecosystems through Platform Interoperability. *IEEE Software*. Institute of Electrical and Electronics Engineers (IEEE). jan 2017, sv. 34, č. 1, s. 54–61. DOI: 10.1109/ms.2017.2.

[5] Centelles, R. P., Freitag, F., Meseguer, R. a Navarro, L. Beyond the Star of Stars: An Introduction to Multihop and Mesh for LoRa and LoRaWAN. *IEEE Pervasive Computing*. Institute of Electrical and Electronics Engineers (IEEE). apr 2021, sv. 20, č. 2, s. 63–72. DOI: 10.1109/mprv.2021.3063443.

[6] ChirpStack. *ChirpStack Documentation* [online]. [cit. 2023-25-01]. Dostupné z: https://www.chirpstack.io/docs/.

[7] ChirpStack. *ChirpStack v4 breaking changes* [online]. [cit. 2023-29-01]. Dostupné z: https://www.chirpstack.io/docs/v4-breaking-changes.html.

[8] DASH7 Alliance. *DASH7 Alliance* [online]. [cit. 2023-15-02]. Dostupné z: https://www.dash7-alliance.org/.

[9] Haxhibeqiri, J., Karaagac, A., Abeele, F. V. den, Joseph, W., Moerman, I. et al. LoRa indoor coverage and performance in an industrial environment: Case study. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, Sep 2017. DOI: 10.1109/etfa.2017.8247601.

[10] Hunkeler, U., Truong, H. L. a Stanford Clark, A. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*. 2008, s. 791–798. DOI: 10.1109/COMSWA.2008.4554519.

[11] Kirimtat, A., Krejcar, O., Kertesz, A. a Tasgetiren, M. F. Future Trends and Current State of Smart City Concepts: A Survey. *IEEE Access*. Institute of Electrical

and Electronics Engineers (IEEE). 2020, sv. 8, s. 86448–86467. DOI: 10.1109/access.2020.2992441.

[12] LoRa Alliance. *About LoRa Alliance* [online]. [cit. 2023-30-01]. Dostupné z: https://lora-alliance.org/about-lora-alliance/.

[13] LoRa Alliance. *LoRaWAN Specification v1.1* [online]. [cit. 2023-30-01]. Dostupné z: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/.

[14] LoRa Alliance. *TR005 LoRaWAN® Device Identification QR Codes* [online]. [cit. 2023-18-03]. Dostupné z: https://lora-alliance.org/resource_hub/tr005-lorawan-device-identification-qr-codes/.

[15] Merriam Webster (n.d.). *Internet of Things*. [cit. 2023-23-01]. Dostupné z: https://www.merriam-webster.com/dictionary/Internet%20of%20Things.

[16] Moravčík, T. *Zpracování a ukládání IoT dat se zaměřením na LoRA senzorové sítě*. Brno, CZ, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/24980/.

[17] Motlagh, N. H., Mohammadrezaei, M., Hunt, J. a Zakeri, B. Internet of Things (IoT) and the Energy Sector. *Energies*. MDPI AG. jan 2020, sv. 13, č. 2, s. 494. DOI: 10.3390/en13020494.

[18] Nguyen, D. C., Ding, M., Pathirana, P. N., Seneviratne, A., Li, J. et al. Federated Learning for Internet of Things: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials*. Institute of Electrical and Electronics Engineers (IEEE). 2021, sv. 23, č. 3, s. 1622–1658. DOI: 10.1109/comst.2021.3075439.

[19] Qadri, Y. A., Nauman, A., Zikria, Y. B., Vasilakos, A. V. a Kim, S. W. The Future of Healthcare Internet of Things: A Survey of Emerging Technologies. *IEEE Communications Surveys & Tutorials*. Institute of Electrical and Electronics Engineers (IEEE). 2020, sv. 22, č. 2, s. 1121–1167. DOI: 10.1109/comst.2020.2973314.

[20] Reynders, B. a Pollin, S. Chirp spread spectrum as a modulation technique for long range communication. In: *2016 Symposium on Communications and Vehicular Technologies (SCVT)*. IEEE, Nov 2016. DOI: 10.1109/scvt.2016.7797659.

[21] Safaric, S. a Malaric, K. ZigBee wireless standard. In: *Proceedings ELMAR 2006*. 2006, s. 259–262. DOI: 10.1109/ELMAR.2006.329562.

[22] Semtech. *LoRa Developer Portal* [online]. [cit. 2023-22-01]. Dostupné z: https://lora-developers.semtech.com/.

[23] The Things Industries. *The Things Stack Documentation* [online]. [cit. 2023-25-01]. Dostupné z: https://www.thethingsindustries.com/docs.

[24] The Things Network. *The Things Network—LoRaWAN Device Classes* [online]. [cit. 2023-28-01]. Dostupné z: https://www.thethingsnetwork.org/docs/lorawan/classes/.

[25] Wan, J., Tang, S., Shu, Z., Li, D., Wang, S. et al. Software-Defined Industrial Internet of Things in the Context of Industry 4.0. *IEEE Sensors Journal*. 2016, sv. 16, č. 20, s. 7373–7380. DOI: 10.1109/JSEN.2016.2565621.