Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Konfigurovateľné zariadenie pripojené ku IoT cloudovej službe pre správu a vizualizáciu senzorov a akčných členov

Diplomová práca

2022 **Bc. Martin Žingor**

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Konfigurovateľné zariadenie pripojené ku IoT cloudovej službe pre správu a vizualizáciu senzorov a akčných členov

Diplomová práca

Študijný program:	Inteligentné systémy
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	doc. Ing. Peter Papcun, PhD.
Konzultant:	Ing. Pavol Gurbaľ

Košice 2022 **Bc. Martin Žingor**

Abstrakt v SJ

Táto diplomová práca sa zaoberá vytvorením webovej aplikácie, ktorá umožní používateľovi (s minimálnymi informatickými schopnosťami), spravovať konfigurovateľné zariadenia Azure Sphere MT3620 odkiaľkoľvek. Spravovaním sa myslí pridávanie senzorov, akčných členov a jednoduchých automatizácií. Súčasťou práce je aj návrh servisu, ktorý spracúva prichádzajúce dáta od pripojených zariadení, čo môžeme rozumieť hodnoty senzorov alebo zmeny hodnôt akčných členov. Analyzujeme IoT (Internet vecí) služby verejných cloudových poskytovateľov a uvedieme si ktoré služby by nám vyhovovali a ako, pre vytvorenie konfigurovateľného zariadenia pripojeného ku cloudovej službe. Následne analyzujeme riadiace jednotky s možnosťou pripojenia do siete internet a porovnáme ich. Vyberieme riadiacu jednotku a navrhneme spôsob ako sa bude jednotka konfigurovať, čo pod termínom konfigurácia rozumieme a aké služby pre dosiahnutie požadovaného výsledku použijeme. V práci budeme používať rôzne programovacie jazyky a frameworky, nakoľko sa práca skladá z troch častí, čo sú naprogramovanie zariadenia, webovej aplikácie (klient) a serveru. Popíšeme si možnosti pri výbere programovacích jazykov pre každú časť a prečo sme si konkrétne jazyky vybrali. Realizujeme návrh a podrobne popíšeme riešenie, problémy ktoré sme museli vyriešiť a ako sme ich vyriešili. Následne vyhodnotíme implementované riešenie, dosiahnuté výsledky a možnosti pokračovania vývoja na tomto riešení, možné modifikácie, možnosti využitia tohto riešenia v praxi.

Kľúčové slová

IoT, cloud, Azure, Azure Sphere, IoT Hub, MT3620

Abstrakt v AJ

This diploma thesis deals with the creation of a web application that allow the user (with minimal IT skills) to manage configurable Azure Sphere MT3620 devices from anywhere. Management means adding sensors, actuators and simple automations. Design of the service, which processes the incoming data from connected devices is also part of the this work. Incoming data can be understood as sensor values or states of action members, actuators. We will analyze IoT (Internet of Things) public cloud services providers and chose which services would suit us the best, to create configurable device connected to the cloud service. We will analyze micro controllers with the possibility of connecting to the Internet and compare them. We will choose micro controller and determine how the unit will be configured, and what we mean by configuration and what services we will use to achieve the desired result. In this work we will use different programming languages and frameworks, as the work consists of three parts, which are the programming of the device part, web application part (frontend) and server part (backend). We will describe the possibilities when choosing programming languages for each part and why we chose specific languages. We will implement the proposed design and describe the solution in details, the problems we had to solve and how we solved them. Subsequently, we'll evaluate the implemented solution, the achieved results and the possibility of future development on this solution, possible modifications, the areas where to use this solution in acton.

Kľúčové slová v AJ

IoT, cloud, Azure, Azure Sphere, IoT Hub, MT3620

64342

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra kybernetiky a umelej inteligencie

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Inteligentné systémy**

Názov práce:

Konfigurovateľné zariadenie pripojené ku IoT cloudovej službe pre správu a vizualizáciu senzorov a akčných členov

Configurable device connected to IoT cloud service for managing and visualizing sensors and actuators

Študent: **Bc. Martin Žingor**

Školiteľ: **doc. Ing. Peter Papcun, PhD.**

Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**

Konzultant práce: **Ing. Pavol Gurbaľ**

Pracovisko konzultanta:

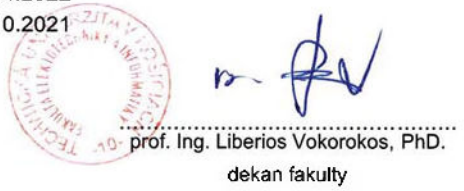
Pokyny na vypracovanie diplomovej práce:

1. Analýza IoT služieb verejných cloudových poskytovateľov
2. Analýza riadiacich jednotiek s možnosťou pripojenia do siete internet
3. Návrh konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe
4. Realizácia konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe
5. Vyhodnotenie implementovaného riešenia
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce (hlavná časť minimálne 60 strán, prílohy – užívateľská a systémová príručka, tlačená forma v nerozoberateľnej väzbe)

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 22.04.2022

Dátum zadania diplomovej práce: 29.10.2021



prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som diplomovú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice 29. 4. 2022 .....

Vlastnoručný podpis

Poďakovanie

Chcel by som sa poďakovať prof. Ing. Ivete Zolotovej, Csc. a doc. Ing. Petrovi Papcunovi, PhD., za možnosť výberu vlastnej témy a externého konzultanta. Docentovi by som chcel poďakovať, že mi pri výbere témy a samotnej implementácie témy nechal voľnú ruku, pripomínal mi dôležité stretnutia a termíny, v prípade otázok pohotovo poradil a tému mi po napísaní skontroloval. V neposlednom rade chcem poďakovať môjmu konzultantovi Ing. Pavlovi Gurbaľovi, ktorý ma počas celého trvania práce usmerňoval, svojimi skúsenosťami naviedol k správnej implementácii riešenia. Taktiež mu chcem poďakovať za čas, ktorý do mňa pri pravidelných stretnutiach investoval a za zabezpečenie hardvéru, ktorý som na vývoj a dokončenie IoT riešenia potreboval.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

IoT Internet of Things

UID Unique Identification number

SaaS Software as a Service

PaaS Platform as a Service

IaaS Infrastructure as a Service

FaaS Function as a Service

AWS Amazon Web Services

MQTT Message Queuing and Telemetry Transport

WSS Websockets Secure

HTTPS Hypertext Transfer Protocol-Secure

LoRaWAN Long Range Wide Area Network

FUOTA Firmware Updates Over-The-Air

API Application Programming Interface

SDK Software Development Kit

MIT Massachusetts Institute of Technology

DDoS Distributed Denial of Service

AMQP Advanced Message Queuing Protocol

RTOS Real-time Operating System

RAM Random Access Memory

USB Universal Serial Bus

GUI Graphical User Interface

MCU Microcontroller Unit

JSON Javascript Object Notation

JWT JSON Web Token

RSA Rivest-Shamir-Adleman encryption

TLS Transport Layer Security

ROM Read-Only Memory

EPROM Erasable Programmable Read-Only Memory

PWM Pulse Width Modulation

GND Ground

IDE Integrated Development Environment

SoC System on a Chip

GPIO General-Purpose Input/Output

SPI Serial Peripheral Interface

I2C Inter-Integrated Circuit

UART Universal Asynchronous Receiver-Transmitter

ADC Analog to Digital Converters

CRUD Create, Read, Update and Delete

DNS Domain Name System

SD Secure Digital

SQL Structured Query Language

LINQ Language-Integrated Query

DOM Document Object Model

HTML HyperText Markup Language

CSS Cascading Style Sheets

DT Device Twin

Úvod

Za posledných pár desiatok rokov ľudstvo zažilo technologický boom, nové technológie, dostupné zdroje, či už finančné alebo energetické zásadne zmenili pracovné podmienky a životný štýl ľudí. Ako v priemyselnej sfére, tak aj v osobnom živote. Do popredia sa dostávajú inteligentné zariadenia a systémy, ktoré by nám mali čo najviac uľahčiť a spríjemniť život, šetriť životné prostredie a efektívne využívať obmedzené zdroje, ako je napríklad voda.

Zariadenia internetu vecí sa nachádzajú všade okolo nás, či sa nachádzame na pracovisku alebo v kuchyni nášho domu. Pri vysokom počte týchto zariadení je často rozhodujúca cena, veľkosť, pripojenie k internetu a hlavne bezpečnosť. Témou diplomovej práce je vytvorenie zariadenia, ktoré spĺňa všetky tieto charakteristiky. Dôraz pri tomto zariadení tiež kladieme na jednoduchosť obsluhy, rozmanitosť použitia a obsluha používateľom, bez potreby špecifických schopností. Našou úlohou je taktiež vytvorenie webovej aplikácie, cez ktorú vieme dané zariadenie konfigurovať.

Pre výber technológií, služieb a zariadenia sme sa opierali o dostupné dokumentácie najznámejších verejných cloudových poskytovateľov. K zariadeniu je dostupná len oficiálna dokumentácia a neexistujú k nemu takmer žiadne návody od jeho používateľov. Pri riešení problémov sme museli na riešenia prísť sami, čo vyústilo do dobrého prehľadu v skúmanej problematike.

V prvej kapitole analyzujeme IoT (Internet of Things) služby verejných cloudových poskytovateľov, porovnáme cenové ponuky týchto služieb. V druhej kapitole sa pozrieme na riadiace jednotky, analyzujeme ich a vyzdvihneme ich prednosti. V tretej kapitole si vyberieme IoT cloudovú službu, zariadenie ktoré k nej pripojíme a urobíme si prvotný návrh. Vyhodnotíme zhotovený prototyp a navrhneme adekvátnu úpravu, finálny návrh architektúry IoT riešenia. Vo štvrtej kapitole sa budeme venovať realizácii IoT riešenia, popíšeme si toky informácií, ako využívame IoT služby, vysvetlíme si vývojové diagramy pre jednotlivé časti IoT riešenia. V poslednej kapitole nás čaká vyhodnotenie implementovaného IoT riešenia a zopakovanie dosiahnutých výsledkov.

1 Formulácia úlohy a cieľ práce

Analýza jednotlivých bodov zadávacieho listu:

1. Analýza IoT služieb verejných cloudových poskytovateľov:
   - definícia IoT a cloudu,
   - rozbor IoT služieb jednotlivých cloudových poskytovateľov,
   - porovnanie a vyhodnotenie cenových ponúk základných IoT služieb.
2. Analýza riadiacich jednotiek s možnosťou pripojenia do siete internet:
   - úvod a rozdelenie riadiacich jednotiek,
   - popis štyroch riadiacich jednotiek s možnosťou pripojenia do siete internet.
3. Návrh konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe:
   - prvotné vnímanie cieľa diplomovej práce,
   - prvý návrh konfigurovateľného zariadenia,
   - návrh back-endu a front-endu webovej aplikácie,
   - vyhodnotenie prvotného návrhu,
   - finálny návrh konfigurovateľného zariadenia.
4. Realizácia konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe:
   - popis využitia Device Twin,
   - komunikácia medzi zariadením a back-endom,
   - dátové objekty komunikácie a databázový model,
   - realizácia konfigurovateľného zariadenia,
   - realizácia back-endu,
   - realizácia front-endu.

5. Vyhodnotenie implementovaného riešenia:
    - vyhodnotenie výberu zariadenia a cloudových služieb,
    - vyhodnotenie použitých technológií,
    - vyhodnotenie návrhu riešenia,
    - vyhodnotenie dosiahnutého riešenia a porovnanie so stanovenou úlohou,
    - perspektíva a možné modifikácie riešenia.
6. Vypracovanie dokumentácie:
    - vypracovanie hlavnej časti diplomovej práce,
    - vypracovanie systémovej príručky,
    - vypracovanie používateľskej príručky.

2 Teoretický rozbor a analýza IoT služieb verejných cloudových poskytovateľov

V tejto časti bude podaný teoretický rozbor a analýza IoT služieb verejných cloudových poskytovateľov. Podrobnejšie definujeme pojem IoT a cloud. Existuje množstvo cloudových poskytovateľov IoT služieb, my sa pozrieme na služby popredných svetových firiem a podrobnejšie ich popíšeme. Služby porovnáme z rôznych hľadísk, ako je napríklad cena, zabezpečenie alebo možnosti pripojenia rôznych IoT zariadení.

2.1 IoT - Internet vecí

Internet vecí (IoT), nazývaný aj internet všetkého, je technologická paradigma predstavovaná ako globálna sieť strojov a zariadení schopných vzájomnej interakcie. IoT je uznávaný ako jedna z najdôležitejších oblastí budúcich technológií a získava veľkú pozornosť zo širokého spektra priemyselných odvetví Lee and Lee (2015). Organizácie využívajú IoT aby fungovali efektívnejšie, lepšie porozumeli potrebám zákazníkom, poskytli zákazníkom lepšie služby, pomohli pri rozhodovaní manažérom podniku a zvyšovali hodnotu podniku. IoT je stále častejšie začleňované do života obyčajného človeka, prvky IoT môžeme nájsť v kuchynských spotrebičoch, práčkach, televízoroch alebo zavlažovacích systémoch. V množstve vecí a procesov okolo nás stále chýbajú prvky IoT, aj preto sa témou IoT oplatí zaoberať. Táto téma poskytuje odpovede na riešenie množstva problémov okolo nás a ponúka príležitosti pre vytváranie nových spoločností, ktoré tieto problémy môžu adresovať. Ako príklad si uvedieme inteligentné skleníky, ktoré šetria energie, zdroje (vodu) alebo náš osobný čas.

Vzájomne prepája výpočtové zariadenia, mechanické a digitálne stroje, predmety, zvieratá alebo ľudí, ktorí sú vybavení jedinečnými identifikátormi (UID) a majú schopnosť prenášať dáta cez sieť. Sú pripojené k internetu a sú jednoznačne identifikovateľné. Vec v IoT môže byť osoba s implantátom srdcového monitora, meračom hodnôt v krvi, hospodárske zviera s transpondérom (čip implantovaný vo zvierati), automobil, ktorý využíva zabudované senzory na upozornenie vodiča, keď je tlak v pneumatikách nízky, alebo akékoľvek iné prírodné alebo človekom vyrobené zariadenie.

Medzi základné prvky IoT patria akčné členy, senzory a riadiace jednotky:

- **Akčné členy** – vykonávacie elementy, preberajú elektrické signály z riadiacich modulov a premieňajú ich na fyzické výstupy. Môžu vykonávať širokú škálu funkcií, od otáčania ventilov, rôzne typy motorov alebo relé.
- **Senzory** – vnemové elementy, ktoré môžu napodobňovať ľudské zmysly (sluch, hmat, zrak alebo čuch). Väčšinou vo forme elektrickej súčiastky, ktorá meria fyzikálne aspekty prostredia a posiela ich formou elektrických signálov do riadiacich modulov, ktoré je schopné ich spracovať. Medzi najbežnejšie senzory patria teplotné, vlhkostné, svetelné, pohybové alebo tlakové senzory.
- **Riadiace jednotky/moduly** – rozhodujúci element, spracováva podnety, elektrické signály zo senzorov a dáta, ktoré do jednotky prichádzajú. Dáta môže aj ukladať a využívať pri riadení (regulátor). Častokrát slúži ako reakčná slučka, čo znamená že vyhodnocuje prijaté dáta a následne sa podľa nich rozhoduje, robí zásahy na akčných členoch.

IoT zariadenie by malo spĺňať aspoň niektoré z nasledujúcich bodov, všetky body by malo spĺňať IoT riešenie ako celok. Tieto body môžeme brať ako bližšiu definíciu IoT zariadení.

1. **Pripojiteľnosť vecí na internet** – možnosť komunikácie s internetom.
2. **Adresovateľnosť** – prijímanie dát, riadenie a nastavovanie parametrov.
3. **Komunikácia a kooperácia** – prepojenie zariadení pomocou sietí.
4. **Lokalizácia** – možnosť určenia fyzickej polohy zariadenia.
5. **Jednoznačná identifikácia** – unikátna identita zariadenia.
6. **Snímanie alebo akčný zásah** – zbieranie dát a vykonávanie akcií.
7. **Spracovanie informácií** – využitie zdrojov zariadenia na vyhodnocovanie.
8. **Programovateľnosť** – prispôsobivosť, adaptácia.
9. **Užívateľské rozhranie** – schopnosť komunikácie s človekom.

2.2 Cloud

Cloud označuje servery, ku ktorým sa pristupuje cez internet, softvér a databázy, ktoré sú spustené na týchto serveroch. Cloudové servery sa nachádzajú v dátových centrách po celom svete. Používaním cloud computingu nemusia používatelia a spoločnosti spravovať fyzické servery alebo spúšťať softvérové aplikácie na vlastných počítačoch.

Hlavný princíp a dôvod využívania cloudu a cloud computingu, je prístup k rôznym typom dát alebo aplikáciám odkiaľkoľvek a z akéhokoľvek zariadenia. Ukladanie dát a výpočtové operácie prebiehajú na serverov v dátových centrách. V nedávnej minulosti museli firmy pre funkčnosť svojich produktov fyzicky prevádzkovať servery, databázy alebo výpočtové prostriedky. Hlavné nevýhody prístupu prevádzkovania vlastných serverov, sú zabezpečenie a starostlivosť o budovu/miestnosť, kde sa tieto prostriedky nachádzajú, servery treba neustále udržiavať a aktualizovať. Dáta sú uložené na jednom mieste a v prípade poruchy alebo znehodnotenia databáz môžeme o všetky dáta prísť alebo budú dáta nejaký čas nedostupné. Tieto problémy boli vyriešené nástupom cloudových technológií.

Cloud computing je možný vďaka technológii nazývanej virtualizácia. Virtualizácia umožňuje vytvorenie simulovaného, digitálneho „virtuálneho“ počítača, ktorý sa správa ako fyzický počítač s vlastným hardvérom. Takýto počítač sa nazýva virtuálny stroj. Virtuálny stroj využíva virtualizačný softvér a fyzický počítač na napodobňovanie ďalších počítačov. Pri správnej implementácii sú virtuálne stroje na tom istom hostiteľskom stroji úplne oddelené, takže medzi sebou nekomunikujú a navzájom sa neovplyvňujú. Súbory a aplikácie z jedného virtuálneho stroja nie sú viditeľné pre ostatné virtuálne stroje.

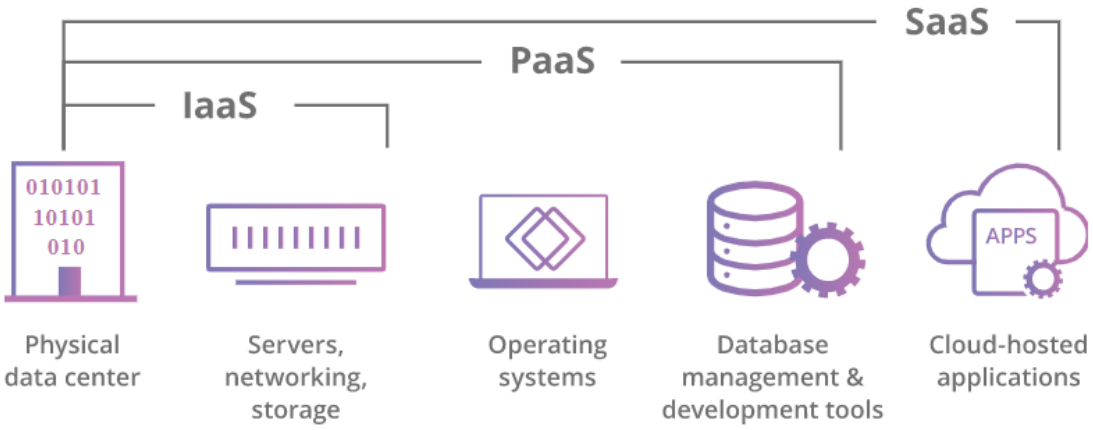
Používaním virtuálnych strojov vieme efektívnejšie využívať hardvér, na ktorom sú nasadené. Spustením viacerých virtuálnych strojov naraz, vedia cloudoví poskytovatelia obsluhovať naraz mnoho služieb rôznych organizácií. Týmto spôsobom môžu ponúknuť využitie svojich serverov za nižšiu cenu naraz oveľa väčšiemu počtu zákazníkov.

Pri poruche jedného alebo viacerých serverov, je služba alebo dáta naďalej dostupná, nakoľko dodávatelia cloudových riešení vo všeobecnosti zálohujú svoje služby

na viacerých serveroch a vo viacerých regiónoch. Používatelia pristupujú k službám prostredníctvom internetového prehliadača alebo prostredníctvom aplikácie, pričom sa ku cloudu pripájajú cez internet.

2.2.1 Modely cloudových servisov

Poznáme 3 základné modely, ktoré môžeme vidieť na obrázku 2 – 1.



Obrázok 2 – 1 Modely cloudových servisov cloudflare (2022)

V nasledujúcich bodoch si v krátkosti opíšeme základné rozdelenie a jeden odvodený model:

- **SaaS** (Software-as-a-Service) – v tomto modeli zákazník platí za existujúci produkt. V podstate si na servise vytvorí svoju vlastnú inštanciu produktu, ktorú si vie sám nastaviť a rozširovať podľa svojich potrieb. Provider sa mu stará a zabezpečenie, ukladanie dát alebo nízko úrovňovú správu. Ako príklad si vieme uviesť Slack, je to nástroj na komunikáciu v rámci alebo medzi firmami. Organizácia si vytvorí účet a sama sa stará o pridávanie používateľov, pridávanie komunikačných kanálov alebo rôznych rozšírení potrebných pri práci.
- **PaaS** (Platform-as-a-Service) – v tomto modeli zákazník platí sa nástroje pre vytvorenie vlastných aplikácií. Zákazník využíva prednastavené servery, na ktoré stačí aplikáciu nasadiť. Ako príklad uvedieme hosťovanie webovej aplikácie, na platformách Heroku alebo Microsoft Azure.

- **IaaS** (Infrastructure-as-a-Service) – v tomto modeli zákazník platí za infraštruktúru, databázy alebo servery. Zákazník má voľnosť pri výbere stavebných blokov na ktorých bude aplikácia nasadená.
- **FaaS** (Function-as-a-Service) – model odvodený zo základných troch Redhat (2020), v ktorom zákazník využíva bez-serverové výpočty. Ako príklad uvedieme automatizáciu posielania emailov, ak sa splní nami určená podmienka.

2.3 Amazon Web Services

Amazon Web Services ponúka služby a riešenia pre pripojenie a správu miliárd IoT zariadení. Zbiera, ukladá a analyzuje údaje IoT zariadení pre priemyselné, spotrebiteľské, komerčné alebo automobilové podniky. Ponúka riešenia pre rôzne záťažové kategórie (objemy dát), až po tie najnáročnejšie. AWS (Amazon Web Services) má veľké množstvo IoT cloudových služieb, my si popíšeme základné, ktoré sa týkajú alebo sa okrajovo dotýkajú témy tejto diplomovej práce. Medzi výhody využívania AWS patria:

- kompletná sada služieb pre rôzne typy IoT projektov,
- jednoduchá škálovateľnosť a integrácia s inými službami spoločnosti AWS,
- spoľahlivosť a elasticita cloudovej infraštruktúry,
- zabezpečenie každej vrstvy od zariadenia až po aplikáciu.

2.3.1 IoT Core

Cloudová služba, ktorá umožňuje pripojeným zariadeniam jednoduchú a bezpečnú interakciu s cloudovými aplikáciami a inými zariadeniami. Dokáže podporovať miliardy zariadení ktoré si môžu vymieňať bilióny správ. Všetky správy sú spoľahlivo a bezpečne spracované a smerované do koncových servisov AWS a do iných zariadení. Služba uľahčuje používanie ostatných AWS služieb, ako napríklad AWS Lambda alebo Amazon DynamoDB Pal et al. (2022c). Pre zjednodušenie používania a integráciu do vlastných projektov existujú SDK (Software Development Kit) pre viacero najpoužívanejších jazykov ako C++, Python, C#, Java, javascript.

Služba IoT Core podporuje najnovšie a najaktuálnejšie technológie pre vytváranie komplexných IoT riešení. Taktiež umožňuje FUOTA (Firmware Updates Over-The-Air) aktualizácie. Pre správu a podporu nasadených zariadení podporuje protokoly:

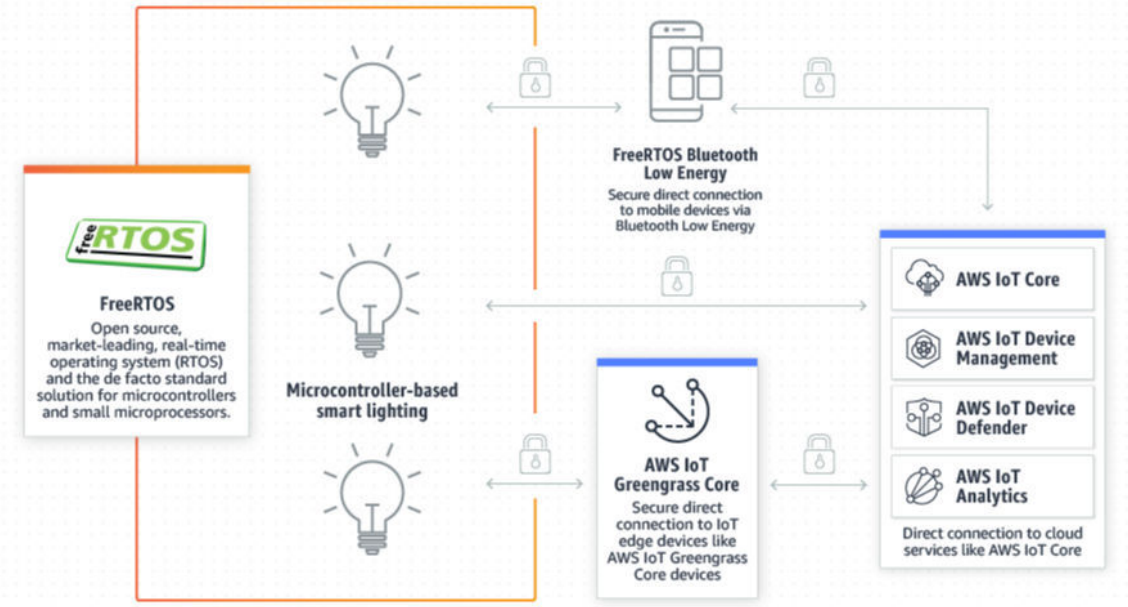
- **MQTT, MQTT cez WSS** (Message Queuing and Telemetry Transport, Websockets Secure) – jednoduchý a široko používaný protokol na odosielanie správ, ktorý je navrhnutý pre zariadenia s obmedzeniami. WSS používa identifikáciu pomocou klientovho ID. Používame ho pre realizáciu komunikácie Pal et al. (2022b).
- **HTTPS** (Hypertext Transfer Protocol-Secure) – posielanie správ pomocou volaní na REST API (Application Programming Interface).
- **LoRaWAN** (Long Range Wide Area Network) – zariadenia na batérie s dlhým dosahom a nízkou spotrebou, ktoré využívajú LoRaWAN komunikačný protokol na prevádzku v bez-licenčnom rádiovom spektre. Umožňuje nízkoenergetickú široko-plošnú komunikáciu medzi zariadeniami. Tieto zariadenia sa dajú pripojiť k AWS IoT Core rovnako, ako by sme pripájali iné zariadenie. Jediným rozdielom je, že medzi zariadením a službou sa nachádza LoRaWan brána, ktorá usmerňuje komunikáciu Pal et al. (2022a).

2.3.2 FreeRTOS

Jeden z najzaujímavejších produktov, vzhľadom na tému tejto diplomovej práce je FreeRTOS, čo je open-source operačný systém pod MIT (Massachusetts Institute of Technology) open-source licenciou pre mikrokontroléry. Tento softvér bol vytvorený s dôrazom na spoľahlivosť a jednoduchosť použitia. Uľahčuje programovanie, nasadenie, zabezpečenie, pripojenie a správu malých zariadení s nízkou spotrebou. Operačný systém zabezpečuje bezpečné pripojenie ku AWS cloudovým službám ako je napríklad IoT Core.

Mikrokontrolér obsahuje jednoduchý procesor s obmedzenými zdrojmi, ktorý možno nájsť v mnohých zariadeniach vrátane senzorov, priemyselnej automatizácie alebo automobilov. Operačný systém bol navrhnutý tak, aby mohol byť nasadený na takýchto mikrokontroléroch a poskytol im bezpečné pripojenie ku cloudovej

službe, mobilnému zariadeniu, lokálnym sieťam alebo edge zariadeniu, ako môžeme vidieť na obrázku 2 – 2.



Obrázok 2 – 2 Schéma použitia FreeRTOS Barry (2022)

V tomto odseku si povieme o ďalších dôležitých vlastnostiach FreeRTOS pri používaní na IoT zariadeniach. AWS ponúka ukážkové aplikácie, pomocou ktorých vieme rýchlo a efektívne začať vyvíjať, čo ušetrí veľa času pri vývoji softvéru zariadenia. Operačný systém je overený a prijatý svetovými spoločnosťami, nakoľko ho mnohé svetové firmy používajú. Softvér podporuje OTA aktualizácie, čo je pri moderných IoT riešeniach nevyhnutnosť. Pre sofvér je garantovaná dlhodobá podpora a kontinuálny vývoj. FreeRTOS je podporovaný na viacerých čipových súpravách a podporuje viac ako 40 architektúr. Medzi najznámejšie kompatibilné vývojové dosky patrí ESP32 a jeho modifikácie *Discover qualified IOT hardware and devices: Build and deliver successful IOT solutions on AWS* (2022).

2.3.3 IoT Analytics

Plne spravovaná služba pre spracovanie dát s možnosťou spúšťania sofistikovaných analýz nad veľkým objemom dát. Eliminuje potrebu vytvorenia zložitej a finančne náročnej analytickej platformy na zobrazovanie a analýzu dát. Analýza dát je veľmi dôležité najmä pre manažérov firmy, ktorí musia robiť dôležité rozhodnutia.

Dáta z IoT zariadení sú často krát zbierané v rôznych podobách (sú neštruktúrované), čo sťažuje ich analýzu pomocou tradičných analytických nástrojov, ktoré sú určené na spracovanie štruktúrovaných údajov. IoT zariadenia zaznamenávajú aj chybové dáta, (ako je teplota, pohyb alebo zvuk) s rôznymi poruchami. Údaje z týchto zariadení môžu často obsahovať časové medzery, poškodené správy alebo falošné hodnoty, ktoré je potrebné pred analýzou vyčistiť. Údaje IoT sú tiež často zmysluplné iba v kontexte dodatočných vstupov údajov tretích strán.

AWS IoT Analytics automatizuje náročné kroky, ktoré sú potrebné na analýzu údajov z IoT zariadení. Filtruje, transformuje a obohacuje dáta pred ich uložením do časových sérií na analýzu. Službu môžeme nastaviť tak, aby zo zariadení zhromažďovala iba tie údaje, ktoré potrebujeme, aplikovala na údajoch matematické transformácie a pred uložením spracovaných údajov obohatila údaje o metadáta špecifické pre zariadenie, ako je typ zariadenia a poloha. Pri analýze môžeme taktiež využiť nástroje strojového učenia a používať vopred vytvorené modely pre bežné prípady analýzy.

2.3.4 IoT Events

Táto služba má za úlohu uľahčiť detekciu a reakciu na sledované vzory dát zo senzorov IoT zariadení. Sledované vzory dát identifikujú komplikovanejšie okolnosti, ako je napríklad narušenie prevádzkovej činnosti vonkajším podnetom, vstupom človeka do zakázanej zóny. Niekedy musela byť rozhodovacia logika v takýchto prípadoch naprogramovaná a musela vedieť na tieto udalosti reagovať. Tento prístup pridával zložitosť do vývoja požadovanej aplikácie. Pomocou IoT Events je jednoduché identifikovať sledované vzory naprieč tisíckami IoT senzorov, ktoré odosielajú rôzne telemetrické údaje. Jednoducho si vyberieme relevantné zdroje údajov, ktoré sa majú vyhodnocovať, definujeme logiku pre každý hľadaný vzor pomocou jednoduchých príkazov if-else a vyberiete vlastnú akciu, ktorá sa spustí, keď hľadaný vzor nájdeme. IoT Events nepretržite monitoruje dáta z viacerých IoT senzorov a aplikácií. Kooperuje s ďalšími službami, ako sú napríklad AWS IoT Core a AWS IoT Analytics, aby zabezpečili včasnú detekciu jedinečných udalostí, pohotový akčný zásah, ktorý si vieme jednoducho naprogramovať pomocou vstavaných blokov, spustenie alarmu, posielanie notifikácií.

2.3.5 IoT 1-Click

Služba ktorá rozširuje funkcionálne možnosti zariadení pripojených do IoT Core. Zariadenia ktoré podporujú službu IoT 1-Click môžu spúšťať funkcie AWS Lambda (FaaS cloudový model servisu). Funkcie lambda môžu pomôcť vykonávať nami definovanú logiku napísanú v jazykoch vrátane Java, Python a C#. Existuje sada preddefinovaných funkcií Lambda pre jednoduché akcie, ako je napríklad odoslanie e-mailu alebo SMS. Veľkou výhodou tohto servisu je bezpečnosť a jednoduchosť nastavenia, nakoľko medzi AWS službami netreba nastavovať bezpečnostné prvky, ako napríklad vytvárať, inštalovať ani spravovať certifikáty. Týmto spôsob sa môžeme vyhnúť vyvíjaniu vlastného softvéru na zariadeniach. Služba uľahčuje správu pripojených IoT zariadení, nakoľko vieme zariadenia spravovať na webovej a mobilnej aplikácii (iOS aj Android). Aplikácia taktiež poskytuje možnosť sledovania stavu, využitia a aktivity na danom zariadení.

2.3.6 Device Defender

Služba, ktorá pomáha pri zabezpečení pripojených IoT zariadení. Nepretržite kontroluje konfigurácie IoT riešenia, objavuje odchýlenie sa od osvedčených bezpečnostných postupov. Uľahčuje údržbu a využíva najefektívnejšie metódy na zabezpečenie zariadení, ako je zabezpečenie identity zariadenia, autentifikácia a autorizácia zariadení a šifrovanie údajov zariadenia. Služba odosiela upozornenia, ak pri kontrole nachádza nejaké medzery, ktoré by mohli predstavovať bezpečnostné riziko. Príklad bezpečnostných rizík môže byť zdieľanie certifikátov identity na viacerých zariadeniach alebo zariadenie so zrušeným certifikátom identity, ktoré sa pokúša pripojiť k službe AWS IoT Core.

Nepretržité monitorovanie bezpečnostných metrík a odchýlok od očakávaného správania zo zariadení je neoddeliteľnou súčasťou každého zdravého IoT riešenia. Možnosťou je predefinovať želané správanie IoT zariadení alebo využiť metódy strojového učenia, ktoré do vyhodnocovania zahŕňa historické dáta. Pri zistení neželaných okolností sa spustí alarm, napríklad pri DDoS (Distributed Denial of Service) útokoch. Pri útokoch na zariadenie, je zariadenie zaradené do karantény.

2.4 Microsoft Azure

Firma Microsoft je jeden z najvýznamnejších hráčov na trhu s cloudovými riešeniami, špecifickými pre IoT odvetvia. Tak ako firma AWS, tiež prepája milióny zariadení a ponúka širokú škálu produktov od zabezpečenia, správu, až po analýzu a spracovanie dát. V nasledujúcich kapitolách si popíšeme najdôležitejšie produkty, ktoré pri vytváraní IoT riešenia môžeme použiť.

2.4.1 IoT Hub

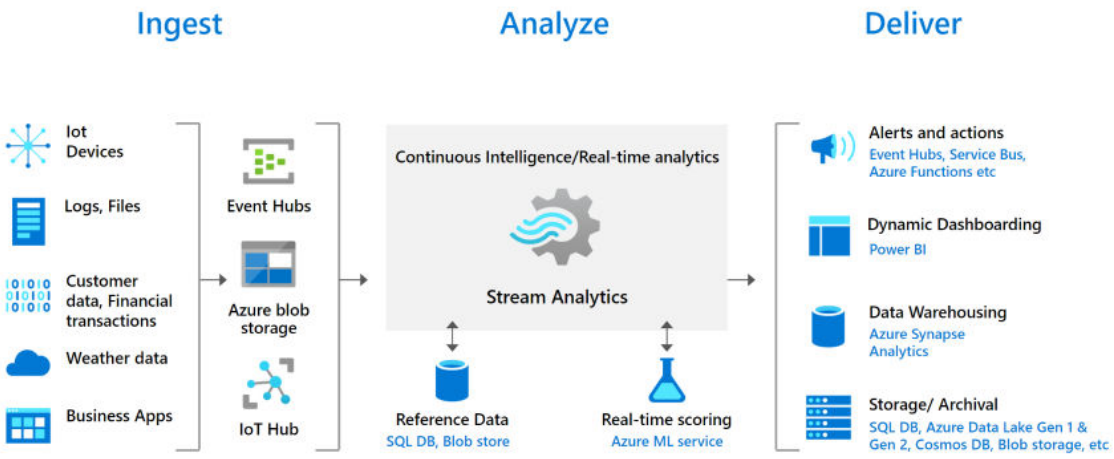
IoT Hub je spravovaná služba s hosťovaním v cloude, ktorá funguje ako centrálne stredisko správ na komunikáciu medzi IoT aplikáciami a jej pripojenými zariadeniami. Spoľahlivo a bezpečne môžete pripojiť veľké množstvá zariadení a ich serverové riešenia. K IoT hubu je možné pripojiť takmer každé zariadenie. Komunikácia s touto službou môže byť realizované pomocou stiahnutia SDK pre rôzne jazyky, čo urýchľuje vývoj, kvalitu a spoľahlivosť IoT riešení. Pri spracovaní prichádzajúcich dát vie IoT Hub naraz obsluhovať milióny zariadení s miliónmi akcií za sekundu. Monitorovanie je súčasťou, pomôže nám sledovať pripojené zariadenia a prípadné poruchy a zlyhania zariadení.

Komunikácia so zariadením a výmena informácií môže prebiehať pomocou viacerých spôsobov posielania správ, napríklad správy cloud-zariadenie, zariadenie-cloud, nahrávanie súborov zo zariadenia na cloud alebo zmena premenných v digitálnom dvojčati zariadenia. Pri komunikácii medzi zariadením a servisom môžeme používať protokoly HTTPS, AMQP (Advanced Message Queuing Protocol), AMQP cez WBS, MQTT, MQTT cez WBS a pre aplikácie ktoré nepodporujú ani jeden zo spomenutých protokolov, si vie zákazník navrhnúť vlastný protokol, pomocou servisu Azure IoT protocol gateway Azure (2019). Podporuje OTA aktualizácie.

IoT Hub môžeme integrovať s viacerými Azure servismi, ktoré si zbežne popíšeme, no pre celkový obraz sú dôležité:

- **Azure Event Grid** – vyberieme zdroj prichádzajúcich správ (IoT Hub), na odber ktorého by sme sa chceli prihlásiť a následne vyberieme obslužný servis alebo koncový bod (WebHook), do ktorého chceme udalosť odoslať, napríklad nami vytvorený servis pre správu a ovládanie zariadení.

- **Azure Logic Apps** – cloudová platforma na vytváranie a spúšťanie automatizovaných pracovných postupov, ktoré integrujú vaše aplikácie, údaje, služby a systémy. Aplikácie Azure Logic Apps zjednodušujú spôsob, akým prepájame staré, moderné a špičkové systémy v cloudových, lokálnych a hybridných prostrediach.
- **Azure Machine Learning** – sprostredkuje učenie na historických údajoch zákazníka, predpovedá budúce správanie, výsledky a trendy. ML Studio je cloudová prediktívna analytická služba, ktorá umožňuje rýchlo vytvárať a nasadzovať prediktívne modely ako analytické riešenia.
- **Azure Stream Analytics** – nástroj navrhnutý na analýzu a komplexné spracovanie veľkých objemov rýchlych streamovaných údajov v reálnom čase, z viacerých zdrojov súčasne (obrázok 2 – 3). Vstupné zdroje môžu byť zariadenia, senzory, streamy, kanály sociálnych sietí a aplikácie. Tieto vzory možno použiť na spustenie akcií, ako je vytváranie upozornení alebo ukladanie transformovaných údajov na neskoršie použitie.



Obrázok 2 – 3 Schéma použitia Stream Analytics Howell (2021)

2.4.2 IoT Central

Je to služba typu PaaS, ktorá ponúka existujúce riešenia pre správu a údržbu podnikových IoT riešení. Výhoda tejto služby je, že zákazník nemusí investovať peniaze

a čas na vytvorenie aplikácie a ani na jej údržbu. Vie si vybrať zo šablón pre rôzne odvetvia vrátane maloobchodu, energetického, štátneho a zdravotníckeho. Pre vytváranie komplexného IoT riešenia má k dispozícii webové používateľské rozhranie, ktoré umožňuje rýchle pripojenie zariadenia, monitorovanie stavu zariadenia, vytváranie pravidiel a spravovanie dát zariadenia. Analýzy a vizualizácie sa do webového rozhrania pridávajú pomocou komponentov, ktoré si stačí nastaviť. K službe je prístup aj z vonku a to pomocou verejného REST API.

2.4.3 IoT Edge

Zabezpečuje presun cloudovej analytiky a vlastnej obchodnej logiky do zariadení na hrane siete. Obchodná logika sa dá nakonfigurovať, nastaviť a následne sa dá pomocou kontajnerov nasadiť na lokálne zariadenia, ktoré vykonávajú nami naprogramovanú funkcionalitu. Priebeh chodu a dáta, sa dajú monitorovať z cloudu. Dôvodom takéhoto prístupu je, že nie všetky analýzy musia byť v cloude. Ak chcete na núdzové situácie reagovať čo najrýchlejšie, môžete spustiť úlohy zisťovania anomálií na hrane siete. Ak chceme znížiť náklady na prevádzke robustnej siete a vyhnúť sa prenosu terabajtov nespracovaných údajov, môžeme údaje vyčistiť a agregovať lokálne a potom len odoslať výsledné dáta na analýzu do cloudu. Iot Edge sa skladá z troch komponentov:

- **Iot Edge moduly** – sú Docker kontajnery, v ktorých sú spustené lokálne inštancie služby Azure (Functions, Stream Analytics, Machine learning), služby tretích strán alebo zákazníkov vlastný kód. Moduly môžu byť postavené na Windows alebo Linux platforme, s použitím rôznych programovacích jazykov. Moduly sa nasadzujú do zariadení IoT Edge, môžu byť nakonfigurované aby medzi sebou komunikovali.
- **Beh programu Iot Edge** – beží na každom zariadení IoT Edge a spravuje nasadené moduly zariadení, zabezpečuje bezpečnostné štandardy zariadení, inštaluje a aktualizuje moduly, odosiela správy o behu zariadenia, zabezpečuje komunikáciu medzi cloudom a zariadeniam, ktoré sú na IoT Edge pripojené.
- **Cloudové rozhranie** – umožňuje vzdialene monitorovať a spravovať zariadenia na hrane siete, upravovať a nasadzovať najnovšie moduly.

2.4.4 RTOS – Real-time Operating System

V podstate veľmi podobný produkt ako ponúka AWS FreeRTOS. Operačný systém ktorý je možné nasadiť na väčšinu najpopulárnejších mikrokontrolérov. Azure RTOS je optimalizované na maximálne využitie obmedzených prostriedkov zariadení (napájané z batérie a majú menej ako 64 KB flash pamäte). Je certifikovaný pre rôzne bezpečnostné štandardy, vrátane certifikácií IEC 61508 SIL 4, IEC 62304 Trieda C a ISO 26262 ASIL D Philmea (2022). Vďaka týmto certifikáciám sa používa v produktoch kritických z hľadiska bezpečnosti, v oblasti medicínskych zariadení, dopravy alebo priemyselných riadiacich zariadení. Medzi ďalšie výhody používania Azure RTOS patria:

- OTA (Over-The-Air) aktualizácie softvéru a aktívny vývoj na RTOS.
- Intuitívny a konzistentný API dizajn, s podrobnou dokumentáciou.
- Komponenty pre zabezpečenie širokej variácie funkcionality, potrebnej pre rôzne typy produktov. Napríklad RTOS FileX, vysokovýkonný FAT kompatibilný súborový systém, ktorý je dostupný pre všetky podporované procesory. Podporuje väčšinu fyzických médií vrátane RAM (Random Access Memory) disku, USBX (Universal Serial Bus) a SD (Secure Digital) CARD.
- Celosvetovo viac ako 6,2 miliardy nasadení, čo je dôkazom o spoľahlivosti, kvality, výkonu, pokročilých funkcií a jednoduchosti používania.
- Integrované s Azure Defender, na zisťovanie hrozieb a včasnú identifikáciu problémov. Všetky bezpečnostné hrozby agreguje do jedného skóre, aby zákazník vedel jednoznačne zaujať postoj na aktuálnu bezpečnostnú situáciu IoT riešenia.

Azure RTOS GUIX Studio poskytuje kompletné prostredie na návrh GUI (Graphical User Interface) aplikácií, ktoré uľahčuje vytváranie a údržbu všetkých grafických prvkov aplikácie. Azure RTOS GUIX Studio automaticky generuje kód v jazyku C kompatibilný s knižnicami RTOS, pripravený na kompiláciu a spustenie v cieľovom zariadení.

Azure RTOS TraceX je analytický nástroj, ktorý vývojárom poskytuje grafický pohľad na systémové udalosti zariadenia v reálnom čase a umožňuje im vizualizovať a lepšie pochopiť správanie ich systémov.

2.4.5 Sphere

Zabezpečená aplikačná platforma, so vstavanými komunikačnými a bezpečnostnými funkciami pre IoT zariadenia pripojené k internetu. Skladá sa zo zabezpečenej jednotky (MCU – Microcontroller Unit), má vlastný operačný systém založený na Linuxe a cloudovú bezpečnostnú službu, ktorá poskytuje nepretržité zabezpečenie s častými aktualizáciami. Hovoríme o fyzickom produkte, čipe, ktorý si zákazníci môžu kúpiť a integrovať ich do svojich nových alebo existujúcich zariadení.

Integruje možnosti spracovania požiadaviek používateľa v reálnom čase, so schopnosťou spúšťať operačný systém na vyššej úrovni. Azure Sphere MCU a operačný systém umožňuje vytváranie zabezpečených zariadení pripojených na internet, ktoré možno aktualizovať (OTA), ovládať, monitorovať a spravovať na diaľku. Sphere MCU poskytuje vylepšené zabezpečenie, ktoré ostatné zariadenia s RTOS nemajú.

Služba Azure Sphere Security Service je neoddeliteľnou súčasťou zariadenia Azure Sphere. Pomocou tejto služby sa zariadenie bezpečne pripájajú ku cloudu a webu. Súčasťou je garancia spustenia zariadenia iba s autorizovanou verziou originálneho, schváleného softvéru. Okrem toho poskytuje zabezpečený kanál, prostredníctvom ktorého môže spoločnosť Microsoft automaticky sťahovať a inštalovať aktualizácie operačného systému pre všetky nasadené zariadenia v teréne. Nevyžaduje sa zásah výrobcu, ani koncového používateľa, čím sa uzatvára klasická bezpečnostná diera IoT zariadení. Azure Sphere je postavené so zameraním na 7 vlastností vysoko zabezpečených zariadení Calaway and Grant (2022):

- **Identita zariadenia v hardvéri** – zariadenie a jeho identita (nefalšovateľný kryptografický kľúč) nemôžu byť oddelené, čím sa predchádza falšovaniu zariadenia. Zabraňuje sa neoprávnenej manipulácii so zariadením od továrne, až po koncového používateľa.
- **Zabezpečenie do hĺbky** – viacero vrstiev zabezpečenia, každá vrstva softvéru overuje, či je vrstva nad ňou zabezpečená.

- **Dôveryhodná výpočtová základňa** – na tejto základni beží iba softvér pre zabezpečenie zariadenia, väčšina softvéru zariadenia zostáva mimo tejto základne.
- **Softvér rozdelený do komponentov** – zabránenie šíreniu narušenia bezpečnosti jedného komponentu na ďalšie komponenty.
- **Overenie bez hesla** – použitie podpísaných certifikátov, overených nefalšovateľným kryptografickým kľúčom, poskytuje oveľa silnejšiu autentifikáciu ako heslá.
- **Hlásenie chýb** – automaticky hlási prevádzkové údaje a chyby do cloudového analytického systému.
- **Obnoviteľné zabezpečenie** – zariadenie sa pri narušení bezpečnosti automaticky aktualizuje, čo si nevyžaduje zásah výrobcu produktu ani koncového používateľa.

2.5 Prehľad iných cloudových poskytovateľov IoT služieb

V tejto časti si uvedieme ďalších cloudových poskytovateľov a okrajovo sa pozrieme na ich IoT služby. Každá spoločnosť ponúka základné služby typu IoT Core, pomocou ktorej IoT zariadenia zaregistrujeme a vieme komunikovať so zariadením, napríklad pomocou REST API. Prídavné služby ako spracovanie a analýza dát alebo modely strojového učenia sa môžu medzi cloudovými produktami líšiť, alebo ich cloudový produkt nemusí vôbec ponúkať. Medzi najznámejších patria firmy: Google s cloudovým produktom Google Cloud, Alibaba - Alibaba Cloud, IBM - IBM Cloud, Oracle - Oracle Cloud, Salesforce - Salesforce Cloud, SAP - SAP Cloud, Rackspace technology - Rackspace Cloud, VMWare - VMWare Cloud.

2.5.1 Google Cloud - IoT Core

Hlavnými komponentmi Cloud IoT Core je správca zariadení a prijímanie správ pomocou dvoch protokolov MQTT a HTTP, ktorými sa zariadenia pripájajú a komunikujú s cloudovou službou. Telemetrické údaje zariadenia sa preposielajú do Cloud

Pub/Sub, potom ich možno použiť na spustenie Cloud Functions, smerovať do servisu tretej strany, poprípade do koncového bodu nášho servisu. K dispozícii je aj služba na analýzu a spracovanie dát Cloud Dataflow.

Pre zabezpečenie bezpečnej komunikácie medzi službou IoT Core a zariadením, máme k dispozícii viacero možností (JWT (JSON Web Token), RSA (Rivest-Shamir-Adleman encryption), TLS (Transport Layer Security) pre MQTT protokol). Google predstavilo SDK pre IoT zariadenia, napísané v C jazyku, vďaka tomu sa dá použiť pre širokú škálu 32-bitových mikrokontrolérov (MCU) naprieč rôznymi operačnými systémami (RTOS), napríklad Zephyr. SDK zabezpečuje:

- Podpora obojsmerného zasielania správ bez prerušenia hlavnej aplikácie zariadenia.
- Autentifikácia a autorizácia prebieha pomocou JSON (Javascript Object Notation) Web tokenov (JWT). Týmto spôsobom sa zmenšuje riziko ohrozenia všetkých zariadení a obmedzuje sa na jedno zariadenie. JWT sa vytvára pre každé zariadenie zvlášť a je obmedzené na určitú dobu platnosti.
- Podpora zariadení s obmedzenými pamäťovými a výpočtovými zdrojmi.
- Podporuje rýchle prototypovanie IoT riešení, ponúka podrobnú dokumentáciu s pred pripravenými ukážkami kódu pre základné typy funkcionalít.

2.5.2 IBM - Watson IoT Platform

Plne spravovaná cloudová služba, ktorá uľahčuje využitie zariadení internetu vecí v praxi. Platforma Watson IoT Platform zabezpečuje bezpečné odosielanie dát do cloudu pomocou protokolu na odosielanie správ MQTT. Pripojené zariadenia môžeme nastaviť a spravovať pomocou online ovládacieho panela alebo zabezpečených rozhraní REST API. Pre ukladanie, riadenie, monitorovanie, spravovanie a analýzu dát môžeme využiť službu Maximo Asset Monitor. Funguje ako SaaS, vďaka čomu vieme bezpečne a jednoducho prepájať rôzne produkty IBM, do jedného zdieľaného rozhrania, ktoré je po nastavení ihneď pripravené spĺňať svoju funkcionalitu. Pripájanie externých služieb prostredníctvom mechanizmu prepájania služieb IBM Cloud nie je podporovaná.

2.6 Porovnanie cenových ponúk základných IoT služieb

V tejto časti si povieme viac o cenách základných IoT služieb verejných cloudových poskytovateľov a skúsime vyčísliť orientačnú cenu v eurách, koľko by mohla stáť prevádzka určitého počtu zariadení. Pri výpočtoch vychádzame z predpokladu, že je zariadenie nonstop pripojené k internetovej sieti a každé zariadenie má podobnú funkcionalitu, čo znamená že sa cena nemení pri rozdielnych prípadoch používania.

Ďalší predpoklad je, že ceny sa každý mesiac zo strany poskytovateľa služieb menia, sú len odhady a nie sú zamýšľané ako skutočné cenové ponuky. Skutočné ceny sa môžu líšiť v závislosti od typu zmluvy uzavretej s rôznymi spoločnosťami, veľkosťami používaných balíčkov alebo dátumu nákupu. Pre získanie presnej sumy, sa vie zákazník spoločnosti spojiť s predajným tímom, ktorý mu na mieru vyčísli náklady pre všetky služby ktoré potrebuje používať.

Za účelom prepočtu budeme uvažovať 10 000 zariadení, pričom každé pošle denne po 10 000 obojsmerných správ, počas obdobia jedného mesiaca (30 dní), čo je 100 000 000 správ za deň. Do cenovej ponuky bude začlenená iba služba typu IoT Core, odosielanie a prijímanie správ z a do zariadení. Neuvažujeme ukladanie, spracovanie alebo analýzu dát. Každá služba má malé rozdiely, ktoré budeme pri prepočte zanedbávať.

2.6.1 Azure IoT Hub

IoT Hub ponúka rôzne balíky, ktoré sa líšia funkcionalitami a počtami správ počas jedného dňa. Pre naše požiadavky musíme uvažovať kúpenie balíka S3, ktorý dovoľuje posielanie 300 000 000 obojsmerných správ veľkosti 4 KB každý deň. My potrebujeme len tretinu tohto počtu, avšak cena je za celý balík, aj keď nie je naplno využitý. V balíku máme zahrnutý Event Hub, ktorý distribuuje správy do nami určených koncových bodov alebo iných služieb. Maximálna veľkosť správy, pre správy odoslané zo zariadenia do cloudu je 256 KB a z cloudu do zariadenia 64 KB. Tieto správy sú merané v 4 KB blokoch, ak zariadenie odošle 16 KB správu, minie si 4 správy v daný deň. Tabuľka 2 – 1 zobrazuje cenu služieb Azure IoT riešenia.

Produkt	100 000 000 správ/deň	300 000 000 správ/deň
Azure IoT Hub	2251.746 €	2251.746 €

Tabuľka 2 – 1 Cena služieb pre Azure IoT riešenie

2.6.2 AWS IoT Core

AWS neponúka balíky, v ich modeli platí zákazník podľa toho, koľko správ celkovo odošle. Taktiež si používateľ musí zvlášť zaplatiť za konektivitu, registrovanie a oznámenia o vnútorných stavoch zariadenia. Maximálna veľkosť pre odosielanie a prijímanie správy je 128 KB. Tieto správy sú merané v 5 KB blokoch, ak zariadenie odošle 8 KB správu, minie si 2 správy v daný deň. V porovnaní s tabuľkou 2 – 1 je výsledná cena dvojnásobne nižšia, avšak pri Azure neplatíme za konektivitu zariadení a môžeme pripojiť koľko chceme zariadení. Ak by sme pri AWS pripojili 100 000 zariadení, cena konektivity by desaťnásobne stúpla. Tabuľka 2 – 2 zobrazuje cenu služieb Azure IoT riešenia.

AWS IoT Core	100 000 000 správ/deň	300 000 000 správ/deň
Konektivita	33.448 €	33.448 €
Správy	330 €	990 €
Twin zariadenia	32 €	96 €
Spolu	395.448 €	1119.448 €

Tabuľka 2 – 2 Cena služieb pre AWS IoT riešenie

2.6.3 Google IoT Core

Google ponúka 3 rôzne balíky, podľa toho aký objem dát za daný mesiac medzi cloudom a zariadením pošleme. Pri objeme dát 250 MB až 250 GB, čo je náš uvažovaný prípad, platí zákazník za MB dát 0.0042 €. Pri výpočte uvažujeme, že jedna správa má 1 KB, čo je štvrtina veľkosti správy oproti službe IoT Hub od Microsoftu. Tabuľka 2 – 3 zobrazuje cenu služieb Google IoT riešenia.

Produkt	100 000 000 správ/deň	300 000 000 správ/deň
Google IoT Core	12304 €	36912 €

Tabuľka 2 – 3 Cena služieb pre Google IoT riešenie

2.6.4 Vyhodnotenie

Z analýzy cien cloudových služieb môžeme pozorovať, že rôzne typy balíkov sa hodia pre rôzne použitia. Google služba je najzaujímavejšia, ak máme obrovské množstvo zariadení a posielame veľké množstvo správ s malou veľkosťou. AWS služba sa oplatí najmenej, ak máme veľmi veľké množstvo zariadení a oplatí sa, ak máme málo zariadení ale chceme posielať veľa správ s veľkou veľkosťou. Microsoft služba sa najviac oplatí, ak potrebujeme neobmedzené množstvo zariadení a zároveň chceme posielať obrovské množstvá dát.

3 Teoretický rozbor a analýza riadiacich jednotiek s možnosťou pripojenia do siete internet

V tejto časti bude podaný teoretický rozbor a analýza riadiacich jednotiek s možnosťou pripojenia do siete internet. V úvode definujeme pojem riadiaca jednotka a podrobnejšie si popíšeme jej vlastnosti. Existuje obrovské množstvo riadiacich jednotiek, ktoré majú možnosť pripojenia k internetu. My sa podrobnejšie pozrieme na najznámejšie, ktoré by nám zároveň vyhovovali pre dosiahnutie cieľa tejto diplomovej práce.

3.1 Mikrokontrolér

MCU (Micro Controller Unit) je malý a lacný mikropočítač, ktorý je určený na vykonávanie špecifických úloh pomocou vstavaných periférií, ako je napríklad prijímanie elektronických signálov. Skladá sa z procesora, pamäte (RAM, ROM – Read-Only Memory, EPROM – Erasable Programmable Read-Only Memory), sériových portov, periférií (časovače, počítadlá). Rozdeľuje sa podľa počtu bitov (8, 16 a 32) s ktorými počas jednej inštrukcie pracuje. 8 a 16 bitový mikrokontolér môžeme použiť na jednoduché výpočty, 32 bitový je vhodný pre komplexnejšie úlohy, ako môže byť automatizácia alebo komunikácia s cloudovou službou. Medzi hlavné výhody používanie mikrokontroléra ako riadiacej jednotky patrí:

- **Cena** – mikrokontolér je malá súčiastka s obmedzenou výpočtovou kapacitou, vďaka čomu je často krát jeho cena nízka. Táto vlastnosť je pri IoT zariadeniach veľmi dôležité, nakoľko častokrát vykonávajú jednoduchú funkcionalitu a ich počet je veľmi veľký (inteligentná žiarovka).
- **Veľkosť** – malá veľkosť a hmotnosť, vďaka čomu môžeme mikrokontrolér používať v malých, prenosných zariadeniach.
- **Nízka spotreba energie** – veľmi nízka spotreba elektrickej energie.
- **Spoľahlivosť** – mikrokontrolér vykonáva jednoduchý program, vďaka tomu eliminuje problémy, ktoré môžu nastať ak by bežal na prídavnej vrstve, operačnom systéme.

3.2 Arduino Uno WiFi

Inak nazvané aj Genuino Uno, je v súčasnej dobe jedna z najpoužívanejších riadiacich jednotiek pre každodenné ale aj zložitejšie vedecké projekty. Je to open-source platforma, založená na ľahko použiteľnom hardvéri a softvéri. Nakoľko bola navrhnutá pre rýchle prototypovanie, má za sebou širokú komunitu priaznivcov a veľmi podrobnú dokumentáciu. My si popíšeme najpoužívanejšiu variantu Arduina Uno s USB napájaním, k dispozícii sú aj výkonnejšie verzie Arduino MEGA, alebo verzia s Bluetooth, Ethernet pripojením.

Na doske nájdeme 8-bitový mikrokontrolér ATmega328P Arduino (2022), kombinuje 32 KB flash pamäte (možnosť zapisovania a čítania), čo je maximálna veľkosť kódu/programu, ktorý je možné do Arduina nahrať. Funguje na napäťovej úrovni 1.8 V až 5.5 V. Frekvencia, operačná rýchlosť je 16 MHz, čo vyjadruje počet jednoduchých operácií za časový úsek jednej sekundy. Pri čítaní analógových signálov, vie čítať s presnosťou 10 bitov, čo je 1024 úrovní. Pri tomto čipe nemáme vstavaný ethernet, čiže pripojenie k internetu musí byť realizované pripojeným externým modulom. Pre zabezpečenie pripojenia k sieti internet je na doske integrovaný WiFi modul ESP8266WiFi, ktorý popri pripojené k internetu umožňuje OTA aktualizácie.

Na plošnej doske Arduina Uno (bližšie popísané v tabuľke 3 – 1) máme k dispozícii 14 digitálnych vstupno výstupných pinov, vrátane 6 PWM (Pulse Width Modulation) výstupov (D11, D10, D9, D6, D5, D3), ktoré slúžia na moduláciu periodického signálu pomocou zmeny šírky impulzu s rozlíšením/presnosťou 8-bitov. K dispozícii máme taktiež 6 analógových pinov s označením od A0 po A5, používajú sa na čítanie analógových hodnôt zo snímačov v rozsahu 0 až 5 V. Na doske sa ďalej nachádzajú viaceré piny uzemnenia s označením GND (Ground), napájacie piny pre senzory a akčné členy 3.3 V a 5 V, vstavané tlačidlo na reštartovanie zariadenia, vstavaná led, USB konektor, napájací konektor, možnosť komunikácie pomocou sériovej linky, SPI (Serial Peripheral Interface) komunikáciu (10, 11, 12, 13), I2C (Inter-Integrated Circuit) komunikáciu (A4, A5) pomocou knižnice Wire.

Pre vývoj na doske Arduino Uno WiFi možeme využiť Arduino IDE (Integrated Development Environment), čo je vývojové prostredie určené pre zariadenia

Arduino. Toto prostredia poskytuje množstvo úsekov kódu pre špecifické funkcionality, ktoré Arduino môže vykonávať alebo možnosť stiahnutia oficiálnych knižníc pre Arduino. Zároveň slúži ako kompilátor a program vie pomocou jedného tlačidla do zariadenia Arduino nahrať. V tomto vývojovom prostredí sa programuje pomocou jazykov C a C++.

Mikrokontrolér	ATmega328P
Napájanie vstupno výstupných pinov	7-12 V
Doporučené napájanie dosky	5 V
Digitálne I/O piny	14 (vrátane 6 PWM výstupov)
PWM piny	6 (D3, D5, D6, D9, D10, D11)
Analógové piny	6 (A0-A5)
Typ CPU	8 - bit
SRAM	2 KB
EPROM	1 KB
Frekvencia mikrokontroléra	16 MHz
Flash pamäť	32 KB
Dĺžka a šírka	68.6 a 53.4 mm
Hmotnosť	25 gramov

Tabuľka 3 – 1 Základné informácie o Arduine Uno Arduino (2022)

3.3 Raspberry Pi

Jedná sa o veľmi lacnú verziu počítača, ktorý má klasickú funkcionalitu ako pripojiteľnosť myši, klávesnice, monitora, ethernetu alebo externého úložiska. Má viacero modelov ako napríklad Raspberry Pi 4, čo je najnovší model a vrámci neho si užívateľ môže vybrať medzi verziami 1, 2, 4 a 8 GB RAM. Zmenšená a slabšia verzia Zero je veľmi podobná, dokážeme k nej pripojiť monitor, má možnosť pripojenia k internetu. Najmenšia riadiaca jednotka zo série je verzia Pico, tá ale nemá integrovaný WiFi modul. K dispozícii sú aj priemyselné verzie, ktoré už sú špecifickejšie na rôzne úlohy. My si popíšeme základnú verziu Raspebrry Pi 4 (tabuľka 3 – 2).

Raspberry Pi 4 je veľmi výkonná riadiaca jednotka, ktorá je zároveň plnohodnot-

ným počítačom. V IoT svete sa používa na výkonovo zložitejšie aplikácie, ako je napríklad snímanie a spracovanie videa, domáci server alebo dátové úložisko. Pre našu potrebu je táto jednotka zbytočne drahá a priveľmi výkonná. O výkon sa stará 4 jadrový 64-bitový procesor ARM Cortex-A27 a SoC (System on a Chip) procesor Broadcom BCM2711, ktorý integruje všetky komponenty (digitálne, analógové vstupy a výstupy) tejto dosky do jednej čipovej jednotky. Pre pripojenie periférií máme k dispozícii Bluetooth 5.0 modul, 2 USB 3.0 a 2 USB 2.0 porty, port pre pripojenie kamery, 2 micro HDMI porty, vďaka ktorím k zariadeniu vieme pripojiť 4K monitor s obnovovacou frekvenciou 60 Hz. K internetu sa vieme pripojiť pomocou WiFi s dvomi frekvenčnými pásmami 2.4 a 5 GHz a gigabitovému ethernetovému portu. Pre napájanie zariadenia môžeme využiť 5 V USB port typu C, GPIO (General-Purpose Input/Output) piny alebo ethernet port.

SoC procesor	Broadcom BCM2711
Procesor	ARM Cortex-A27 64-bit
Počet jadier procesora	Quad-Core (4 jadrá)
Frekvencia jedného jadra procesora	1.5 GHz
RAM	LPDDR4: 1GB, 2GB, 4GB a 8GB
USB porty	2 x USB 3.0, 2 x USB 2.0
Ethernet	Gigabit ethernet
WiFi	2.4GHz, 5GHz (Gigahertz)
Bluetooth	5.0
HDMI	2 x micro HDMI

Tabuľka 3 – 2 Základné informácie o Raspberry Pi 4

Ako úložisko pre operačný systém používame SD kartu (minimálna veľkosť 8 GB) ale k zariadeniu sa tiež dá pripojiť externé úložisko cez USB port. Najpoužívanejším operačným systémom pre Raspberry Pi zariadenia je Raspbian (odvodený z Debianu, čo je linuxová distribúcia), no existujé veľké množstvo ďalších, ako je napríklad upravený Windows. Pre vývojové prostredie na Raspberry Pi môžeme využiť prostredia (Eclipse, Visual Studio Code), ktoré sú určené pre Linux OS. Výhodou vývoja IoT riešení na tejto riadiacej jednotke je široký výber programovacích jazykov, Java, C++, C, Python. Možnosťou je taktiež prevádzkovanie databáz. Nevýhodou je, že program musí zároveň bežať na OS, čo je ďalšia vrstva, v ktorej môžu

pri dlhodobom chode programu nastať problémy.

Dôležitou súčasťou Raspberry Pi 4 sú aj piny, pomocou ktorých vieme pretvoriť zariadenie na IoT riešenie. K dispozícii máme 40 vstupno výstupných pinov (GPIO), uzemnenie (GND) a napájanie (3.3 V a 5 V). Niektoré GPIO piny majú tiež alternatívne funkcie, ktoré im umožňujú prepojenie s rôznymi druhmi senzorov, ktoré používajú protokoly I2C (displeje), SPI alebo UART (Universal Asynchronous Receiver-Transmitter). Ak chceme tieto piny použiť s týmito protokolmi, musíme to povoliť pomocou rozhrania aplikácie Raspberry Pi Configuration.

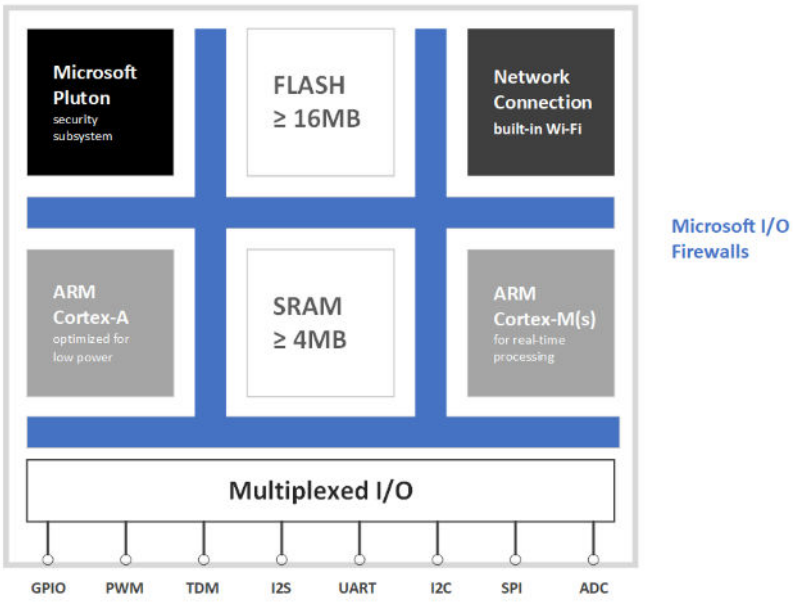
3.4 Azure Sphere MT3620 Development Kit

MT3620 riadiaca jednotka bola navrhnutá v úzkej spolupráci so spoločnosťou Microsoft a ide o prvé riešenie konceptu Microsoft Azure Sphere. V podstate ide o zariadenie, ktoré má unikátnu identitu integrovanú v hardvéri, tým pádom sa zariadenie nemôže zneužiť a ponúka bezkonkurenčné zabezpečenie. MT3620 sa zameriava na širokú škálu aplikácií internetu vecí, vrátane inteligentnej domácnosti, komerčnej, priemyselnej a mnohých ďalších oblastí.

MT3620 má tri jadrá, každé jadro a jeho pridružený subsystém je v inej dôveryhodnej doméne. Najdôveryhodnejšia doména sa nachádza v bezpečnostnom podsystéme Pluton. Každá vrstva architektúry predpokladá, že vrstva nad ňou môže byť ohrozená. Výpočtové a pamäťové zdroje každej vrstvy sú voči iným izolované, čo zvyšuje bezpečnosť zariadenia. Jadrá zariadenia MT3620 (obrázok 3 – 1):

- **Microsoft Pluton bezpečnostný podsystém** – bezpečnostné jadro procesora, ktoré je založené na hardvérovom zabezpečení, identita zariadenia je integrovaná fyzicky v jadre. Zabezpečuje kryptografické služby, hardvérový generátor náhodných čísel, generovanie verejných a súkromných kľúčov, asymetrické a symetrické šifrovanie, spúšťanie softvérových komponentov, spracováva prichádzajúce požiadavky z komponentov a spracováva ich.
- **Jadro aplikácie vyššej úrovne** – založený na ARM Cortex-A procesore, ktorý je optimalizovaný na prevádzku operačného systému a poskytuje oddelenie procesov pomocou dôveryhodných domén.

- **Jadro pre prácu v reálnom čase** – založený na ARM Cortex-M procesore, ktorý je optimalizovaný pre pripojenie k perifériám zariadenia (GPIO) a spúšťanie aplikácií ktoré operujú v reálnom čase. Schopné je spúšťať 'holý' kód alebo RTOS. Toto jadro nemá priamy prístup k internetu, pre túto možnosť komunikuje s jadrom aplikácií vyššej úrovne.



Obrázok 3 – 1 Architektúra hardvéru Calaway and Grant (2022)

Procesory	Arm Cortex-A7, Arm Cortex-M4
Počet jadrier procesorov	1, 2 jadrá
Frekvencia jadier procesorov	500 MHz, 200 MHz
Počet vstupno/výstupných pinov	76
Počet PWM kanálov	12
Počet ADC kanálov	8 (12-bitových)
Ethernet	10 Mbps (Megabits per second)
WiFi	2.4 GHz, 5 GHz

Tabuľka 3 – 3 Základné informácie o Azure Sphere MT3620

Komunikácia s internetom je podporovaná pomocou WiFi (2.4 GHz a 5 GHz) alebo ethernetu (10 Mbps). Internetové pripojenie je pre toto zariadenie veľmi dôležité, nakoľko sa cez neho vykonávajú OTA aktualizácie (OS a aplikácia). Na riadiacej

jednotke (tabuľka 3 – 3) sa nachádzajú 76 programovateľných vstupno výstupných pinov. Väčšina pinov je zdieľaná medzi všeobecnými GPIO a inými funkciami. K dispozícii máme dvanásť PWM kanálov rozdelených do troch skupín (každá skupina má svoj vlastný kontrolér), šesť hardvérových počítadiel, dva bloky I2S, osem 12-bitových ADC (Analog to Digital Converters) a päť blokov sériového rozhrania (I2C, SPI, UART), z ktorých každý obsahuje päť pinov. Na doske sa ďalej nachádza napájanie (3.3 V a 5 V) a uzemnenie.

3.5 ESP8266 12-E NodeMCU

Riadiaca jednotka ktorá môže byť vďaka veľkej FLASH pamäti (4 MB) a jedno jadrovým procesorom s frekvenciou 80 MHz až 160 MHz, použitá pri budovaní IoT riešení. Modul podporuje technológiu OTA aktualizácií, ktorá umožňuje nahrávanie programu aj firmwéru cez bezdrôtovú sieť wifi. Na module je zabudovaný aj USB konektor s prevodníkom, teda programovanie a update softvéru je možný aj cez kábel. Pre toto zariadenia existuje množstvo SDK, napríklad FreeRTOS a Non-OS SDK.

Mikrokontrolér	ESP-8266 32-bit
Frekvencia procesora	80 MHz
Veľkosť Flash pamäte	4 MB
Doporučené napájanie dosky	3.3 V
Digitálne I/O piny	16
Počet ADC kanálov	1 (10-bitový)
WiFi	2.4 GHz

Tabuľka 3 – 4 Základné informácie o ESP8266 12-E NodeMCU

Pri porovnaní ESP s inými WiFi riešeniami na trhu, je táto riadiaca jednotka skvelá voľba pre väčšinu projektov internetu vecí. Pre naše potreby má ale nedostatočné množstvo ADC kanálov a možností pripojenia senzorov cez rôzne sériové rozhrania. Medzi hlavné výhody patrí cena a možnosti jednotku integrovať do pokročilých projektov. Navyše je kompatibilná s Arduino IDE, z ktorého vieme softvér na dosku jednoducho nahrať. Softvér do zariadenia sa dá písať vo viacerých programovacích jazykoch vrátane C, C++ a Python.

ESP8266 je zároveň aj WiFi modul, ktorý využíva iba 2.4 GHz kanál. Zariadenia

neobsahuje Bluetooth modul. Pre pripojenie senzorov a akčných členov môžeme využiť šestnásť GPIO pinov, jeden analógový (ADC) pin s presnosťou 10-bitov a desať PWM pinov. Pre napájanie máme k dispozícii 3.3 V a 5 V napájací zdroj.

3.6 ESP32

ESP32 je nízko nákladový mikrokontrolér od výrobcu riadiacej jednotky ESP8266, toto zariadenie je jeho nástupcom. Existuje v jedno jadrových aj dvoj jadrových variantách 32-bitového mikroprocesora Xtensa LX6 Tensilica Teja (2021) s integrovaným Wi-Fi a Bluetooth. Oproti predchádzajúcej verzii ide o významnú aktualizáciu, vo výkone ako aj v možnosťami pripojenia rôznych periférií.

Softvér pre zariadenie môžeme písať vo viacerých jazykoch (C, C++, python) a môžeme využiť viaceré vývojové prostredia, ako je Arduino IDE, Visual Studio Code alebo MicroPython. Ak ide o vybavenie zariadenia, to opisujeme v tabuľke 3 – 5.

Procesor	Xtensa LX6 Tensilica
Počet jadier procesora	1 a 2 jadrová verzia
Frekvencia procesora	160 až 240 MHz
Veľkosť Flash pamäte	do 16 MB
Digitálne I/O piny	34
Počet ADC kanálov	1 (10-bitový)
Sériové pripojenie	4xSPI, 2xI2C, 2xI2S, 3xUART
WiFi	2.4GHz, 5GHz
Bluetooth	v4.2

Tabuľka 3 – 5 Základné informácie o ESP32

4 Návrh konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe

V tejto kapitole si popíšeme, aká bola prvotná predstava fungovania konfigurovateľného zariadenie, ktorého vytvorenie je cieľ diplomovej práce a aký bol prvý návrh, aké problémy sa pri prvom návrhu vyskytli a ako sme ich vyriešili. Uvedieme výhody a nevýhody prvotného výberu zariadenia. Popíšeme si zmenu prvotného návrhu, uvedieme novú architektúru, ktorá bola s malými úpravami finálna.

Popíšeme si výber druhého zariadenia, ktoré nám pre dosiahnutie cieľa najviac vyhovovalo, spomenieme prostriedky (vývojové prostredia a SDK) s ktorými sme pri vývoji na zariadení a webovej službe pracovali a uvedieme prečo sme ich používali a prečo boli pre našu prácu nevyhnutné. Medzi opis spomenutých častí pridáme odôvodnenie výberu databázy a jej opis.

4.1 Prvotné vnímanie a návrh architektúry konfigurovateľného zariadenia

Predstava, vnímanie, ako má vyzerať výstup po roku práce sa postupom času, riešenia problémov jemne zmenila, aj keď hlavné stanovené body sme splnili a nemuseli sme kvôli nim meniť ich znenie. V predstihu sa nedá na 100% určiť, ako bude vyzerať finálny výsledok. Pri vývoji aplikácií je dôležité zvoliť metodiku, pomocou ktorej určíme postupnosť krokov, ktoré budú viesť k nášmu cieľu, či to je dokončenie diplomovej práce alebo uvedenie hotového produktu na trh.

My sme si vybrali agilnú metodiku SCRUM, ktorej dôraz nie je na naplánovanie všetkých krokov dopredu (inak môžeme tento prístup nazvať aj hádanie) ale plánovanie, práca, ohodnotenie a prispôsobenie v malých časových úsekoch, napríklad 4 týždne. V praxi si môžeme predstaviť nastavenie malého cieľa, ktorý vieme v dohľadnej dobe (4 týždne) dosiahnuť, následne na ňom pracujeme. Po uplynutí stanovenej doby ohodnotíme čo sa podarilo urobiť a aké boli problémy, analyzujeme čo sa dá zlepšiť, navrhneme zmeny. Následne tieto zmeny aplikujeme v ďalšom cykle a týmto spôsobom dosahujeme kontinuálny pokrok smerom k zadanému cieľu. Túto metodiku spomíname, pretože je dôležité vedieť, že sme sa pomocou nej počas celej

práce riadili. Čo v praxi znamená častá zmena architektúry a funkcionality, podľa zistených nových skutočností a problémov, reakcia na podnety pre zlepšenie.

Prvotná predstava bolo vytvorenie softvéru, ktorý umožní akémukoľvek používateľovi (s minimálnymi IT schopnosťami), spravovať konfigurovateľné zariadenie na hrane siete odkiaľkoľvek. Taktiež bude môcť používateľ zobrazovať dáta zo zariadenia a vytvárať jednoduché rutiny, inak môžeme nazvať aj automatizácie, ktoré má zariadenie vykonávať. Ako je napríklad zapnutie ventilátora, ak teplota na konkrétnom senzory presiahne maximálnu povolenú hodnotu. Samozrejmé je taktiež ovládanie akčných členov, pre naše potreby akčné členy predstavujú relé, či už samotnými automatizáciami alebo pomocou webového rozhrania. Pri rozobratí na drobné, sme pri plánovaní mali 4 úlohy:

- **Zariadenie** – vybratie zariadenia, ktoré nám vyhovuje pre komunikáciu s naším back-endom a zároveň má možnosti pre pripojenie dostatočného množstva akčných členov a senzorom. Pri rozhodovaní sme museli pozerať zároveň na SDK, s ktorými si vieme uľahčiť prácu, výber programovacieho jazyka, vývojového prostredia a komunikačného prostriedku pre komunikáciu s webovým rozhraním.
- **Back-end** – pri výbere back-end technológie, sme sa rozhodovali podľa programovacieho jazyka (najlepšie spoľahlivý, robustný, s množstvom knižníc pre uľahčenie a urýchlenie vývoja) a frameworku, ktorý za nás bude riešiť autentifikácie, komunikáciu s databázou.
- **Front-end** – pri výbere front-end technológie, sme sa taktiež rozhodovali podľa programovacieho jazyka v ktorom budeme grafické rozhranie vyvíjať a frameworku, ktorý nám napríklad zabezpečí manažovanie stavov.
- **Databáza** – rozhodovali sme sa podľa skúseností s rôznymi typmi databáz.

4.1.1 Zariadenie

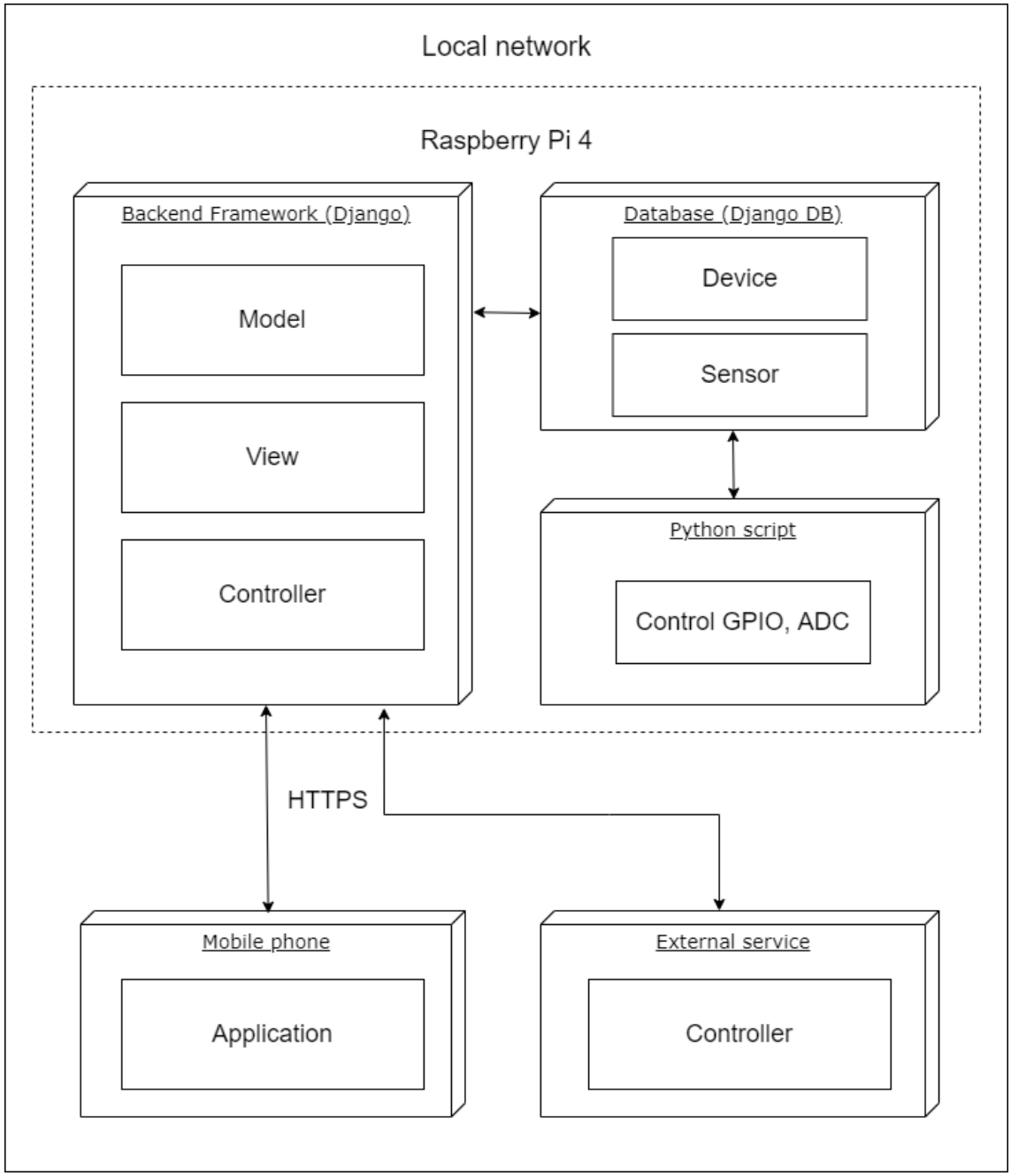
Pri prvom výbere sme zvolili za najvhodnejšiu riadiacu jednotku Raspberry Pi 4, s 4GB RAM konfiguráciou. Bližšie ju popisujeme v analýze riadiacich jednotiek v kapitole 3.3. Medzi kľúčové parametre, ktoré nás presvedčili bol výkon tohto zariadenia, predošlé skúsenosti s prácou na zariadení, výber programovacích jazykov, pomocou

ktorých sa dajú v Raspberry Pi programovať aplikácie. Medzi dôležité argumenty taktiež patrí dobrá dokumentácia zariadenia, veľká komunita nadšencov a používateľov, široká škála voľne dostupných riešení, z ktorých sa dá čerpať a inšpirovať sa.

V Raspberry Pi sa dajú spúšťať aplikácie vo viacerých jazykoch, naša preferencia je Python, nakoľko pre neho existuje množstvo knižníc pre ovládanie GPIO pinov, čítanie z ADC kanálov, knižníc/frameworkov pre spustenie komplexného webového servera s autentifikáciou, autorizáciou, vstavaným pripojením k databáze. Taktiež máme v jazyku Python dlhoročné skúsenosti, nakoľko ho často využívame na školské úlohy. V zariadení sme pre vývoj používali Visual Studio Code.

Ako architektúru sme zvolili prevádzku webového servera, databázy a Python skriptu na zariadení Raspberry Pi 4. Grafickú reprezentáciu architektúry môžeme vidieť na obrázku 4 – 1. Webové rozhranie pre správu zariadenia je nasadené mimo Raspberry Pi a so zariadením komunikuje pomocou HTTPS volaní. Pre podrobnejší opis jednotlivých častí tejto architektúry zariadenia si časti zariadenia rozdelíme na celky:

Webový server - vybrali sme si bezplatný a open source webový framework Django. Pri rozhodovaní sme rozmýšľali aj nad jeho alternatívou, Flask. V čase výberu sme mali s obidvoma nejaké skúsenosti, no rozhodujúce boli porovnania a návody z internetu. Pre porovnanie musíme spomenúť, že Django je framework a Flask je len knižnica a v obidvoch je používaný programovací jazyk python. Pre naše potreby by postačovali obidve technológie, avšak Django je zaujímavejší kvôli jeho robustnosti, vstavaných funkciách a komplexnosti. Django ponúka možnosť jednoduchej integrácie autentifikácie pre bezpečnosť riešenia, pričom pri Flask-u musíme používať rôzne knižnice od rôznych tretích strán. V našom riešení využívame možnosť vytvorenia jednoduchého REST API v Django frameworku, pomocou ktorého bude obojstranne komunikovať Raspberry Pi (REST API) s webovým rozhraním pomocou HTTPS metód, GET, POST, PUT, PATCH a DELETE. Pre naše použitie si vystačíme s GET, POST a DELETE. Správy sú reprezentované pomocou JSON štruktúry.

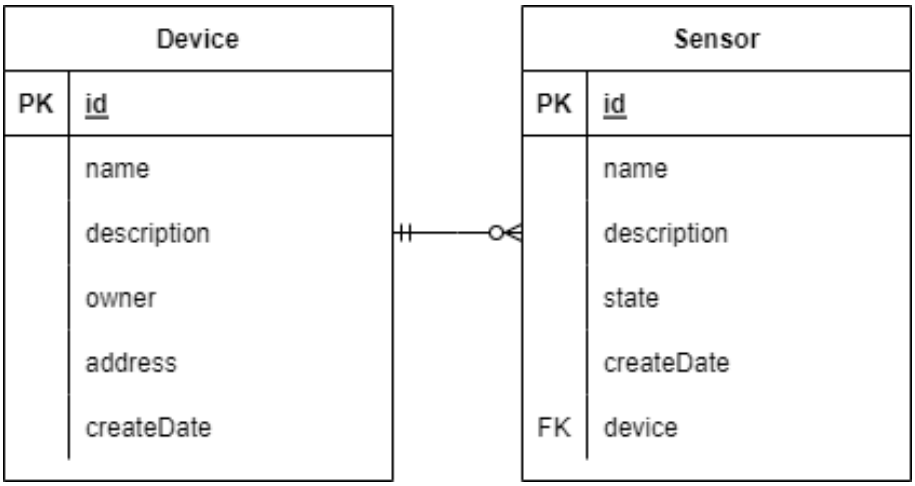


Obrázok 4 – 1 Architektúra zariadenia Raspberry Pi 4

Django architektúra sa drží konvencie MVC (model-view-controler), čo môžeme preložiť ako model-pohľad-kontrolér. My zložku pohľad nepotrebujeme a nevyužívame, nakoľko zariadenie nebude ponúkať žiadne webové prvky, nebude ponúkať žiadny grafický výstup. V modeloch rozumieme dátovú štruktúru objektu, my sme navrhli iba 2 objekty device a sensor, pretože sme túto architektúru po krátkom zvažovaní zavrhli. Kontrolér ponúka koncové body pre CRUD (Create, Read, Update and Delete) akcie, ako 'getAllSensors', 'updateDevice' a následné akcie, ktoré sa zavolaním daného koncového bodu vykonajú. Akcie sa primárne týkajú aktualizovania dát v lokálnej databáze, ktorá je na zariadení nasadená.

Pre testovanie a vývoj je komunikácia zabezpečená len v lokálnej sieti. Pre reálne nasadenie, čo znamená pripojenie a ovládanie zariadenia cez internet, musíme nastaviť presmerovanie portov na routeri a dynamickú DNS (Domain Name System) pre zariadenie. Týmto spôsobom síce zabezpečíme želanú funkcionalitu, ale zároveň zvyšujeme bezpečnostné riziko.

Databáza - vybrali sme si Django MariaDB databázu, ktorá je na zariadení lokálne nasadená. Pre základnú funkcionalitu, sme vytvorili 2 tabuľky (obrázok 4 – 2), Device - zariadenie, Sensor - senzory. Tabuľky obsahujú základné informácie o objektoch a navzájom sú prepojené reláciou jedno zariadenie môže mať 0 alebo viac senzorov. Databáza má obmedzenú veľkosť podľa veľkosti vnútornej pamäte, čo je v našom prípade SD karta.

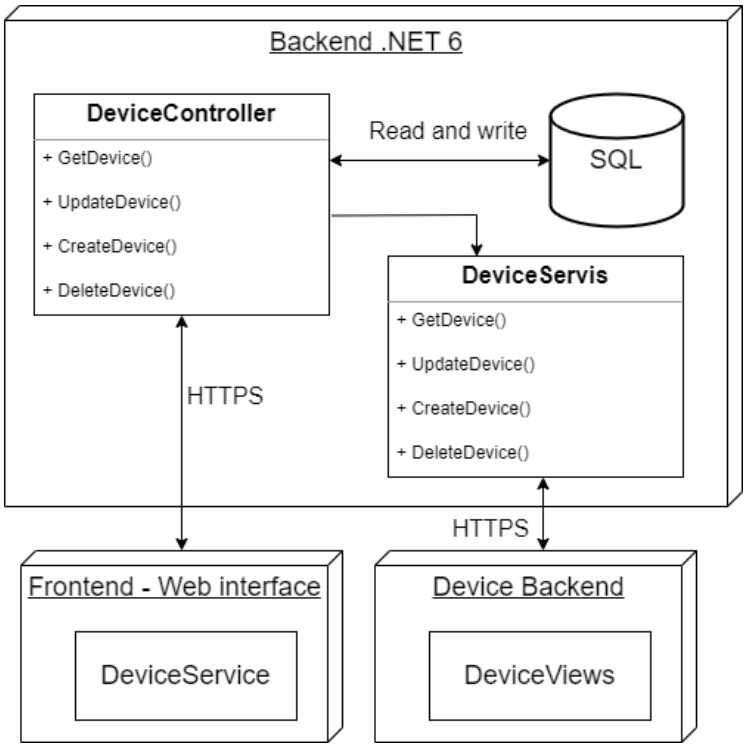


Obrázok 4 – 2 Databázový model Raspberry Pi

Python skript - používame na ovládanie akčných členov, vykonávanie automatizácií a čítanie dát zo senzorov. Skript kontroluje databázové dáta a vykonáva nad nimi želané akcie. Pri zmene stavu, naopak do databázy zmeny zapisuje. Cyklus pre vykonávanie automatizácií sa vykonáva viac krát v priebehu jednej sekundy, čítanie dát z databázy sa vykonáva raz sa sekundu.

4.1.2 Back-end

Pri výbere back-end technológie pre webové rozhranie sme boli rozhodnutí ihneď, aj keď je viacero možností pre technológie (.NET Core, Django, Flask, PHP – Hypertext Preprocessor, Nodejs). Vybrali sme si .NET 6, čo je novšia verzia .NET Core. Vyvíja sa v ňom pomocou programovacieho jazyka C#, v ktorom už máme skúsenosti a taktiež máme skúsenosti s používaním editora Visual Studio. Visual Studio ponúka šablóny pre spojenie rôznych front-end a back-end technológií, čím šetrí používateľovi/vývojárovi množstvo času. Taktiež ponúka možnosť instantného nasadenia súčastí riešenia na platformu Microsoft Azure, ako je aj databázový systém, web server alebo webové rozhranie.



Obrázok 4 – 3 Architektúra back-endu

Prvotný návrh architektúry môžeme vidieť na obrázku 4 – 3, kde sme opísali komunikáciu medzi rôznymi časťami back-endu pre operácie nad zariadením. Ide o jednoduchý prístup, kedy z front-endu príde požiadavka na akciu nad zariadením, zmena sa zapíše do databázy, poprípade sa z databázy dáta vyťahujú. Následne sa cez servisnú triedu prepošlú zmeny do zariadenia. Tento prístup je veľmi krkolomný, nakoľko môžu dáta v databázach stratiť konzistentnosť. Pre zabezpečenie konzistentnosti by sme museli napísať veľké množstvo ošetrení, logiku ktorá zabezpečí dodatočnú aktualizáciu zariadenia v prípade, že bude zariadenie nedostupné (mimo siete internet). Taktiež ukladáme rovnaké dáta na dvoch rôznych miestach, čo zväčšuje rozsah zmien, ktoré musíme pri zmene modelu a funkcionality urobiť.

Pre interakciu s databázou, sme využili nuget balík/knižnicu (Entity framework). Entity fraework znižuje nesúlad impedancie medzi relačným a objektovo orientovaným svetom, čo umožňuje vývojárom písať aplikácie, ktoré interagujú s údajmi uloženými v relačných databázach pomocou .Net objektov (presne určená dátová štruktúra). Objekty predstavujú doménu aplikácie a eliminujú potrebu písania množstva kódu pre prístup k dátam. Ako príklad si uvedieme jednoduchý výber všetkých akčných členov pre zariadenie s id 4. Pričom _context predstavuje objekt databáz. DeviceActionMembers je databázová tabuľka pre akčné členy a senzory zariadenia. Príkaz vracia list všetkých vyhovujúcich objektov:

```
_context.DeviceActionMembers.Where(x => x.DeviceId == 4).ToList();
```

Môžeme si všimnúť, že s databázou pracujeme ako s obyčajným objektom. Je to možné vďaka LINQ (Language-Integrated Query), ktorý predstavuje jednotnú syntax pre prístup k dátam – bez ohľadu na ich zdroj, ktorým môže byť databázové rozhranie, XML súbor alebo obyčajný objekt vo vnútri programu. Uľahčuje transformáciu, triedenie, prepojenie dát a vyhľadávanie v nich.

4.1.3 Front-end

Pri výbere front-end technológie, sme boli hneď rozhodnutí použiť existujúci framework/knižnicu, ktorý nám urýchli a zjednoduší vývoj grafického rozhrania. V dnešnej dobe poznáme 3 najväčšie a najpoužívanejšie, React, Angular a Vue.js. Skúsenosti máme s Angular a React, pričom pri Angular platforme sa dá výhradne programovať len v typescript programovacom jazyku, pri Reacte sa dá vybrať medzi javascript

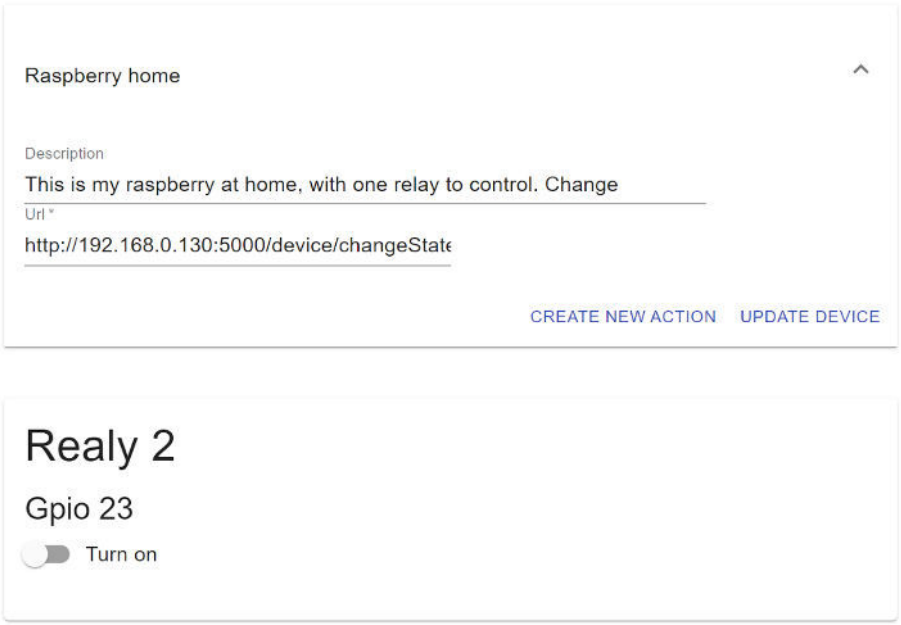
a typescript programovacím jazykom. Typescript jazyk sme si vybrali preto, pretože javascript jazyk rozširuje o statické typovanie a ďalšie atribúty, ktoré poznáme z objektovo orientovaného programovania. Má veľmi podobný syntax ako jazyk C#, pretože ho tiež vytvorila spoločnosť Microsoft.

Ako platformu sme si vybrali React. React je bezplatná a open source front-end knižnica, na vytváranie používateľských rozhraní založených na komponentoch. Spravuje ho Meta, (predtým Facebook) React (2022) a komunita jednotlivcov a spoločností. React možno použiť ako základ pri vývoji jednostránkových alebo mobilných webových aplikácií. Zaoberá sa však iba správou stavu a vykresľovaním daného stavu do DOM (Document Object Model), dôsledkom čoho je pre vytváranie React aplikácií zvyčajne potrebné použitie ďalších knižníc na smerovanie alebo získavanie dát z externých servisov, komunikáciu s našim back-endom.

V našom riešení webového grafického rozhrania využívame iba open source knižnice, ktoré je možné používať bez licencie a kúpenia. V nasledujúcich odrážkach si vymenujeme najdôležitejšie z knižníc, ktoré si treba od React knižníc nezávisle nainštalovať a vysvetlíme si načo ich používame:

- **axios** – promise-based (je predpokladaný výsledok po dokončení alebo zlyhaní asynchrónnej operácie) HTTP klient pre node.js (softvérový systém navrhnutý pre vývoj škálovateľných webových aplikácií). Dokáže natívne transformovať Json dáta na objekty.
- **mui/material** – jednoduchá a prispôsobiteľná knižnica komponentov na rýchlejšie vytváranie estetických a prístupnejších aplikácií React.
- **mui/icons-material** – obsahuje viac ako 2 000 oficiálnych ikoniek prevedených na SvgIcon komponenty, vďaka čomu ich vieme veľmi ľahko používať priamo v typescript kóde.
- **bootstrap** – sada nástrojov kaskádových štýlov pre tvorbu webových aplikácií. Obsahuje šablóny založené na HTML (HyperText Markup Language) a CSS (Cascading Style Sheets), slúžiace pre úpravu typografie, formulárov, tlačidiel, navigácie a ďalších komponentov webových rozhraní. Rýchla pomoc pre vývojárov, ktorí chcú venovať viac času funkcionalite, ako štýlovaniu stránky.

- **lodash** – pomáha písať stručnejší a udržateľnejší kód. Vďaka tejto knižnici vieme zjednodušiť bežné programovacie úlohy, ako je určovanie typu alebo zjednodušenie matematických operácií, obdoba LINQ v jazyku C#.
- **react-router-dom** – pre implementovanie dynamického smerovania vo webových aplikáciách. Zobrazuje stránky a umožňuje implementovanie navigácie medzi komponentami/stránkami webovej aplikácie.
- **recharts** – knižnica na zjednodušenie práce s klasickými grafmi (koláčový, stĺpcový, čiarový) alebo grafmi s dynamickými hodnotami.



Obrázok 4 – 4 Prvotný návrh zobrazenia zariadenia s URL poľom

Architektúra front-endu sa od prvotného návrhu, až po finalizáciu diplomovej práce nijako dramaticky nemenila. Na začiatku sme vytváranie, mazanie, úpravu zariadení a akčných členov/senzorov mali na jednej stránke dokopy, môžeme vidieť na obrázku 4 – 4. Ako zariadenie sme používali Raspberry Pi, ktoré bolo spustené na lokálnej sieti. v boxe pre úpravu zariadenia môžeme vidieť pole Url, kde sme zadali adresu lokálneho zariadenia. Táto adresa sa následne v back-ende používala ako základ url HTTPS volaní, plus koncovka koncového bodu Django servera na zariadení.

4.2 Vyhodnotenie prvotného návrhu - prototypu

Koncept konfigurovateľného zariadenia na hrane siete (predošlý názov diplomovej práce) sme dokončili a úspešne otestovali. Došli sme k záveru, že takéto zariadenie by bolo možné ďalej rozvíjať a pre náš stanovený cieľ by bolo dostatočné, ale ani zďaleka ideálne. Využívanie Raspberry Pi ako riadiacej jednotky, má niekoľko výhod: veľký výpočtový výkon, voľnosť výberu technológií, veľkosť pamäte alebo množstva dokumentácie. Samozrejme sme došli aj k chybám architektúry, problémom a nevýhodám, ktoré si v nasledujúcich bodoch popíšeme:

- **Zabezpečenie** – jedna z najhlavnejších vlastností, ktoré by malo obsahovať každé IoT zariadenie je vysoká bezpečnosť, nakoľko zneužitie informácií, zasahovanie do akcií zariadenia môže spôsobiť veľké majetkové škody. Pre prístup k zariadeniu mimo lokálnej siete potrebujeme otvoriť zariadenie voči internetu. Týmto spôsobom zvyšujeme bezpečnostné riziká, aj keď existujú riešenia pre tento prípad, nakoľko je celkom bežný. Druhá vec je, že by sme museli sami naprogramovať zabezpečenú komunikáciu medzi zariadením a back-endom webovej aplikácie, čo by zobralo veľké množstvo času a zďaleka by to nebolo tak bezpečné ako IoT služby verejných cloudových dodávateľov.

- **OTA** – IoT zariadenia by mali obsahovať možnosť aktualizácie softvéru po sieti, nakoľko sa častokrát nájdu chyby v softvéri po nasadení do produkcie. O túto možnosť by sme pri Raspberry Pi prišli, jediná možnosť by bola taktiež túto funkcionalitu naprogramovať, čo vôbec nie je triviálna záležitosť.

- **Operačný systém** (OS) – čím viac súčastí má IoT zariadenie, tým je väčšia pravdepodobnosť chýb, ktoré pri vykonávaniu programu môžu nastať. OS Raspberry Pi nie je určený len pre IoT zariadenia, spúšťa množstvo procesov, ktoré my ani nevyužívame. Softvér IoT zariadení musí byť čo najviac spoľahlivý, mal by obsahovať predovšetkým funkcie, ktoré zariadenie k plneniu svojej úlohy nevyhnutne potrebuje.

- **Spotreba energie a cena** – veľký výpočtový výkon spotrebuje veľké množstvo energie a taktiež si za výkon musíme priplatiť. Pri IoT zariadeniach vo veľkých množstvách musíme dbať na úsporu energie a cenu jednotlivých zariadení.

Najlepšie je kúpa zariadení, ktoré výkonom zodpovedajú predpokladanému použitiu.

- **Komunikácia medzi časťami riešenia** – bez použitia cloudových služieb musíme všetku komunikáciu IoT riešenia naprogramovať sami, čo tiež ústi k horšiemu zabezpečeniu. Pre našu potrebu je lepšie zaplatiť spravovanú službu, ktorá nám bezpečnú komunikáciu sprostredkuje. Taktiež nám poskytuje manažment zariadení na vyššej úrovni.

- **Synchronizácia** – pri tomto bode budeme písať o dvoch častiach synchronizácie, databázy a dvojčaťa zariadenia (DT). Začneme databázami, pretože v prvotnej architektúre ich využívame dve (pre webovú aplikáciu a zariadenie). V podstate majú obidve databázy obsahovať veľké množstvo spoločných informácií, vďaka čomu môžeme pri chybách stratiť konzistentnosť databáz. Pre zabezpečenie konzistentnosti by bolo potrebné napísať rôzne ošetrenia, čo znižuje čitateľnosť kódu aplikácie a sťažuje zmenu funkcionality kódu. Niektoré cloudové služby poskytujú možnosť využitia DT, v ktorom sa nachádzajú stále aktuálne stavy a nastavenia zariadenia. Týmto vieme zabezpečiť konzistentnosť a včasnú reakciu na zmenu, nakoľko sa vieme na udalosť zmeny pripojiť a pohotovo reagovať. Raspberry Pi takúto funkcionalitu natívne nepodporuje.

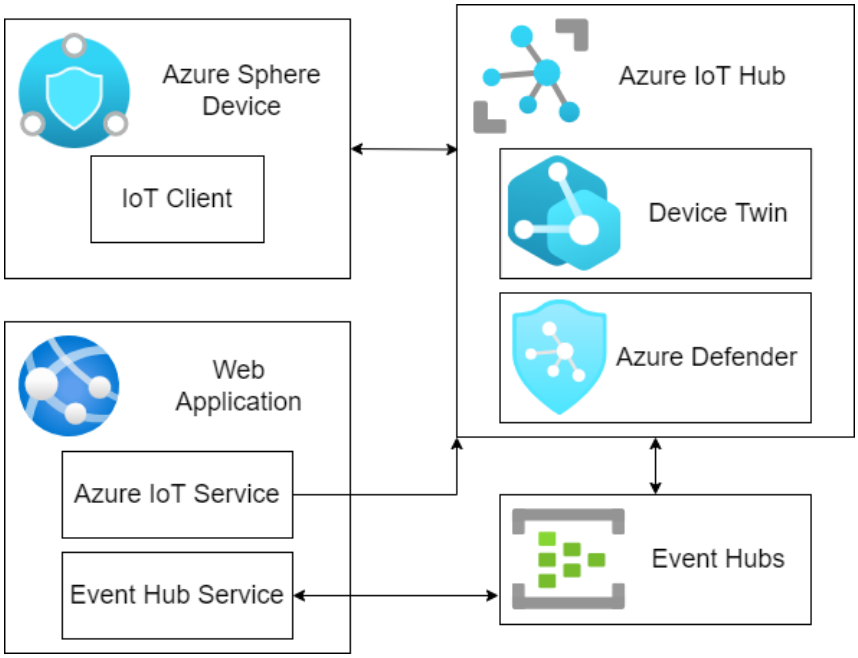
- **Oneskorenie/latencia** – pre synchronizáciu využívame vnútorné servisy zariadenia a webovej aplikácie, kde vznikajú oneskorenia hlavne na strane zariadenia. Aj tento problém sa dá riešiť používaním DT alebo vybudovaním mechanizmu na neustále dopytovanie po zmenách (prihlásením na zmeny), čo taktiež zaberie veľké množstvo času.

Po zvážení plusov a mínusov riešenia, sme sa rozhodli zmeniť zariadenie a prístup komunikácie so zariadením, využitím IoT služby verejného cloudového poskytovateľa. Pri zvážení dostupných zariadení a ioT služieb sme sa rozhodli pre pomerné nové, nepreskúmané zariadenie Azure Sphere, ktorého OS a čip navrhol a naprogramoval Microsoft. Vývojová doska bola skonštruovaná subdodávateľom, pričom ako mikrokontrolér použili Azure Sphere. Ako IoT službu sme si vybrali Azure IoT Hub, ktorá rieši množstvo problémov, ktoré vznikli pri predošlom návrhu architektúry.

Pri takýchto komplexných zmenách, sme taktiež zmenili názov témy, pretože budeme vytvárať konfigurovateľné zariadenie pripojené k IoT službe verejného cloudového poskytovateľa.

4.3 Finálny návrh architektúry IoT riešenia

Pri využití IoT služby pre komunikáciu s IoT zariadením, sa nám výrazne mení architektúra back-endu webovej aplikácie, front-end sa zmení len minimálne. V novej architektúre využívame viaceré služby, ktoré môžeme vidieť na obrázku 4 – 5. Tieto služby sú nevyhnutné pre zabezpečenie požadovanej funkcionality, ktorú máme za cieľ vytvoriť. Niektoré služby ako Azure Defender a Device Twin, sú súčasťou služby IoT Hub. Externe za nich neplatíme a ani ich nemusíme nastavovať, po pripojení zariadenia k IoT Hub službe, vieme funkcionalitu týchto prídavných služieb ihneď používať.



Obrázok 4 – 5 Nová architektúra služieb

Presnú funkcionalitu a potrebné nastavenia všetkých služieb si opíšeme v realizácii. V návrhu môžeme spomenúť, prečo sme sa rozhodli práve pre túto skladbu konkrétnych služieb. Službu Azure Sphere sme opísali v kapitole 2.4.5 a jeho vývojovú dosku v kapitole 3.4. Spomenieme si ale že hlavným dôvodom bola bezkonku-

renčná bezpečnosť a možnosť pripojenia zariadenia k IoT Hub službe, ktorej analýzu sme urobili v kapitole 2.4.1. Ďalšími dôvodmi boli schopnosti tejto vývojovej dosky, podrobná dokumentácia a príklady kódu pre rôznorodé využitie zariadenia.

Azure IoT Hub sme použili kvôli tomu, že s Microsoft službami máme skúsenosti, služba ponúka funkcie šité pre komplexné IoT riešenia a aj kvôli SDK, ktoré Microsoft ponúka ku každej dôležitej službe vo viacerých jazykoch. Aktívne využívame ďalšie dve služby: Device Twin a Event Hub. Každé Azure Sphere zariadenie využíva na pozadí službu pre zabezpečenie zariadenia Azure Defender, avšak s touto službou my nič viac nerobíme.

Device Twin, v preklade dvojča zariadenia, používame primárne na synchronizáciu stavov a nastavení zariadenia, s databázovými hodnotami vo webovej aplikácii. Túto službu si popíšeme do hĺbky neskôr, keď to bude viac relevantné. Event Hub používame na manažment prichádzajúcich správ zo zariadenia alebo zmenu DT (Device Twin). K tejto službe sa pripájame na back-ende webovej aplikácie a odchytávame zmienené udalosti.

5 Realizácia konfigurovateľného zariadenia pripojeného ku IoT cloudovej službe

V tejto kapitole si popíšeme kroky realizácie konfigurovateľného zariadenia pripojeného k IoT cloudovej službe. Do hĺbky si vysvetlíme všetky rozhodnutia, ktoré sme v architektúre komunikácie, front-endu, back-endu a zariadenia museli urobiť. Podrobne sa pozrieme na toky dát medzi aplikáciami, popíšeme si modely týchto dát. Komunikácia so zariadením prebieha pomocou dvoch prístupov, podrobne sa pozrieme na obidva prístupy a uvedieme, ktorý kedy používame. Popíšeme si databázový model webovej aplikácie.

V prvých podkapitolách celku realizácia diplomovej práce si vysvetlíme dôležité časti, ktoré sa budú rôznymi komponentami/časťami IoT riešenia spoločne používať. Medzi jednu z najdôležitejších patrí Device Twin.

5.1 Device Twin – DT

Je to JSON dokument, ktorý uchováva informácie o stave zariadenia vrátane metadát, konfigurácie a podmienok. Azure IoT Hub udržiava Device Twin pre každé zariadenie, ktoré pripojíte k IoT Hub. Vytvorí sa automaticky, keď sa k IoT Hub pripojí nové zariadenia a zanikne po jeho vymazaní. Všeobecne môžeme DT využívať pre:

- Ukladanie metadát špecifických pre dané zariadenie. Napríklad posledná aktualizácia samotného DT alebo súčasnú polohu, kde sa zariadenie nachádza.
- Hlásenie informácií o aktuálnom stave zariadenia, ako sú dostupné možnosti v danom časom okamihu. Napríklad či je zariadenie pripojené k IoT Hubu cez mobilnú sieť alebo WiFi.
- Synchronizuje stav dlhotrvajúcich pracovných postupov medzi aplikáciou zariadenia a IoT Hubom. Napríklad, keď chceme nahrať novú verziu softvéru, špecifikuje novú verziu na inštaláciu a aplikácia zariadenia hlási rôzne fázy procesu aktualizácie.
- Jedna z najdôležitejších funkcií, je získanie súčasných hodnôt senzorov, stavy

akčných členov alebo konfigurácie. Samozrejme to funguje aj naopak, kedy my sami môžeme do DT vkladať hodnoty, ktoré zariadenie zaregistruje a začne používať.

Ako sme už spomínali, DT je JSON dokument, čo znamená že dáta môžu byť organizované do rôznych polí alebo môžu byť agregované do špecifického objektu, ktorý strana ktorá dokument číta vie deserializovať/vložiť dáta do štruktúry. V tomto dokumente máme 4 sekcie s pravidlami:

- **Tags** – tagy, je sekcia JSON dokumentu, kde môže používateľ zapisovať rôzne hodnoty a čítať ich. Zariadeniu tieto hodnoty nie sú poskytnuté. Maximálna veľkosť je 8 KB.
- **Desired properties** – požadované vlastnosti, sa používajú na hlásenie zmien a synchronizáciu konfigurácie zariadenia (v našom prípade automatizácie a používané piny). Používateľ môže tieto vlastnosti meniť, zariadenie ich môže iba čítať. Maximálna veľkosť je 32 KB.
- **Reported properties** – nahlásené vlastnosti, sa taktiež používajú na synchronizáciu. Zariadenie môže tieto vlastnosti meniť, používateľ ich môže iba čítať. Maximálna veľkosť je 32 KB.
- **Device identity properties** – identita zariadenia, tieto vlastnosti nemôže nikto meniť, len čítať. Udávajú kto chce k dátam pristupovať, nakoľko môžeme mať pri zariadení viacej používateľov s rôznymi rolami, ktoré udávajú kto ma k akej vlastnosti prístup.

Takto vyzerá DT nášho zariadenia:


```
{
    "deviceId": "eaddb5fb3106763abbabc20...",
    "status": "enabled",
    "connectionState": "Disconnected",
    "authenticationType": "certificateAuthority",
    "version": 2292,
    "properties": {
        "desired": {
            "GPIO": {
                "length": 0,
                "array": []
            },
            "ADC": {
                "length": 0, "array": []
            }
            "$metadata": {
                "$lastUpdated": "2022-04-09T21:05:27.9325867Z",
                "$lastUpdatedVersion": 140,
                "ADC": { ...
        },
        "reported": {
            "telemetry": {
                "value": true,
                "array": [] }
        }
    },
    "capabilities": {
        "iotEdge": false
    }
}
```

5.1.1 Operácie s Device Twin

Ako sme už spomínali, s DT môžeme robiť rôzne operácie podľa totožnosti strany (v našom prípade zariadenie a back-end webovej aplikácie), ktorá sa ich snaží vykonať. V tejto časti si popíšeme aké akcie môžu tieto strany urobiť. Začnime povolenými operáciami zo strany back-endu a vysvetlime si načo ich používame:

- **Vyhľadanie DT podľa ID zariadenia** – využívame na získanie najnovšej verzie DT.
- **Upravenie časti DT** – umožňuje nám upraviť určitú časť DT, využívame pre jednoduché aktualizácie jedného akčného člena alebo hodnoty jedného tagu. Ak chceme deklaráciu určitej vlastnosti úplne vymazať, musíme jej hodnotu nastaviť na null a IoT Hub túto vlastnosť vymaže.
- **Nahradenie požadovaných vlastnosti** – využívame pre úplnú zmenu všetkých požadovaných vlastností, v tomto prípade sa celá časť JSON dokumentu nahradí za novú.
- **Nahradenie tagov** – taktiež sa všetky tagy nahradia za nové.
- **Odoberanie zmien DT** – jedna z najdôležitejších akcií, je prihlásenie sa na odber zmien DT. Týmto spôsobom odchytávame zmeny, ktoré zariadenie robí pri hlásených vlastnostiach alebo zmien, ktoré vykonáva samotný IoT Hub (notifikácia o nedostupnosti zariadenia).

Povolené operácie pre zariadenie:

- **Získanie DT** – vracia obidve vlastnosti, okrem tagov, ku ktorým zariadenie prístup nemá. Využívame pri reštartovaní alebo aktualizácii zariadenia.
- **Upravenie hlásených vlastností** – používame na ohlásenie zmenenia stavov zariadenia. Nevyužívame na posielanie telemetrie, ako sú hodnoty senzorov.
- **Odoberanie zmien požadovaných vlastností DT** – využívame na zachytávanie zmien požadovaných vlastností DT, čo v našom prípade znamená zmena, vymazanie alebo pridanie nových akčných členov, senzorov alebo automatizácie.

5.2 Posielanie správ medzi zariadením a back-endom

Pri návrhu obojsmernej komunikácie medzi zariadením a back-endom webovej aplikácie máme na výber z viacerých možností. Pri návrhu sme sa sústredili na čo najspoľahlivejšiu synchronizáciu zariadenia s databázou webovej aplikácie a na vyhnutie sa používania DT pre akcie, ktoré neovplyvňujú vnútorné stavy zariadenia. Ak ide o posielanie správ zo zariadenia do webovej aplikácie, máme takéto možnosti:

- **Zariadenie-cloud správy** – tieto správy používame na posielanie telemetrie (hodnoty z ADC pinov), dát ktoré niesu využívané pre synchronizáciu. V aktuálnom riešení sa telemetria odosiela každú jednu sekundu. V budúcnosti by sa tento parameter dal nastavovať dynamicky, dosiahnuť by sme to vedeli pridaním parametru do DT.
- **Zmena DT** – správu o zmene DT zachytí back-end, ktorý aktualizuje databázu. V týchto správach posielame vnútorné stavy aplikácie zariadenie, zabezpečujeme tým synchronizáciu a konzistentnosť databázy webovej aplikácie.
- **Posielanie súborov** – túto možnosť posielania správ nevyužívame, nakoľko sme ju pre žiadnu akciu nepotrebovali. Ide však o väčšie balíky telemetrie, v budúcnosti by sa táto možnosť dala využiť na posielanie zozbieranej telemetrie počas nedostupnosti internetového pripojenia. Taktiež by nám táto možnosť vyhovovala pre veľmi časté čítanie hodnôt senzorov (100 čítaní za sekundu), pričom by sme odosielali celý balík 100 hodnôt každú sekundu.

Pri posielaní dát z back-endu webovej aplikácie do zariadenia máme takéto možnosti:

- **Priama metóda** – tieto typy správ potrebujú okamžité potvrdenie o prijatí od zariadenia. Používame ich na posielanie zmenu stavu akčných členov, na ich ovládanie. Napríklad na zapnutie relé akčného člena. Pri nedostupnosti zariadenie alebo poruche, dostaneme okamžité oznámenie s chybovou hláškou. Týmto spôsobom vieme nanovo zopakovať akciu, chybu uložiť pre neskoršiu analýzu alebo jednoducho chybu zobraziť používateľovi na webovom rozhraní.
- **Zmena DT** – zmenu DT zachytí zariadenie a môže na ňu adekvátne zareagovať. Využívame na ukladanie konfigurácie akčných členov, senzorov a au-

tomatizácií v zariadení. DT stále drží konfiguráciu, ktorá je práve nastavená na zariadení.

- **Cloud-zariadenie správy** – táto možnosť nevracia notifikáciu o úspešnom alebo neúspešnom prijatí. Je to jednosmerná správa, pre naše IoT riešenie nie je relevantná, preto ju nijako nevyužívame.

5.3 Dátové objekty správ medzi zariadením a back-endom

V predošlých kapitolách sme si vysvetlili aké sú možnosti DT a ako ho využívame 5.1. Následne sme si vysvetlili typy správ, ktoré pri komunikácii môžeme využívať 5.2. V tejto časti si popíšeme, aké JSON objekty v jednotlivých typoch správ medzi zariadením a back-endom posielame.

Jednotlivé typy správ si vysvetlíme na príklade, pretože všetky typy správ sa odvíjajú od vnútornej konfigurácie zariadenia. Ako konkrétny príklad sme si zvolili konfiguráciu, ktorú môžeme vidieť v kapitole 5.3.1. Pri uvedení príkladov sa budeme taktiež venovať významu a vysvetleniu jednotlivých parametrov objektov.

5.3.1 Cloud zmení DT

Do požadovaných vlastností, v JSON dokumente DT označované ako *desired*, vkladáme 3 typy objektov: ADC, GPIO a automations. Tieto objekty reprezentujú upravené databázové modely (kapitola 5.4), ktoré dokáže zariadenie spracovať. Hlavná štruktúra každého z týchto troch objektov je rovnaká, mení sa len obsah poľa (môžeme vidieť v kapitole 5.1, názov *array*). Parameter *length* predstavuje dĺžku poľa, používame ho pre zjednodušenie čítania DT zariadením.

Pri objekte **ADC** vkladáme do poľa integer hodnoty. Tento objekt sa používa na určenie, z ktorých ADC pinov zariadenia sa budú čítať hodnoty, na akých pinoch sa nachádzajú senzory:

```
"ADC": {
    "length": 1,
    "array": [1]
}
```

Pri objekte **GPIO** vkladáme do poľa objekty, ktoré určujú na ktorých pinoch sa nachádzajú akčné členy a senzory. Pri každom objekte určujeme typ a pin:

```
"GPIO": {
    "length": 1,
    "array": [
        {
            "type": 0,
            "gpio": 68
        }
    ]
}
```

Pri objekte **automations** vkladáme do poľa objekty automatizácie. Pri každom objekte určujeme typ automatizácie, ADC pin z ktorého čítame hodnoty, hodnotu s ktorom budeme porovnávať, znamienko pre porovnanie a GPIO pin (akčný člen):

```
"automations": {
    "length": 1,
    "array": [
        {
            "type": 0,
            "adc": 1,
            "value": 1000,
            "operator": ">",
            "gpio": 68
        }
    ]
}
```

Pre ucelený pohľad si zopakujeme, čo sme na zariadení nakonfigurovali. Zariadenie číta hodnoty z ADC pinu 1, zaregistrovali sme akčný člen na pine 68, ktorý teraz dokážeme ovládať. Nakoniec sme pridali automatizáciu, ktorá číta hodnoty z ADC pinu, porovnáva ju pomocou znamienka '>' s hodnotou 1000. Ak je podmienka splnená, zmeníme stav akčného člena na pine 68.

5.3.2 Správa cloud-zariadenie

Táto správa sa používa na zmenu stavu (na 0 = false) akčného člena na pine 68:

```
{
    "Pin": 68,
    "State": 0
}
```

5.3.3 Zariadenie zmení DT

Ak je hodnota *value* true, tak sa odosielajú správy typu zariadenie-cloud (kapitola 5.3.4) každú sekundu. V poli *array* máme stavy všetkých akčných členov, v našom príklade máme len jeden:

```
"telemetry": {
    "value": true,
    "array": [
        {
            "gpio": 68,
            "value": true
        }
    ]
}
```

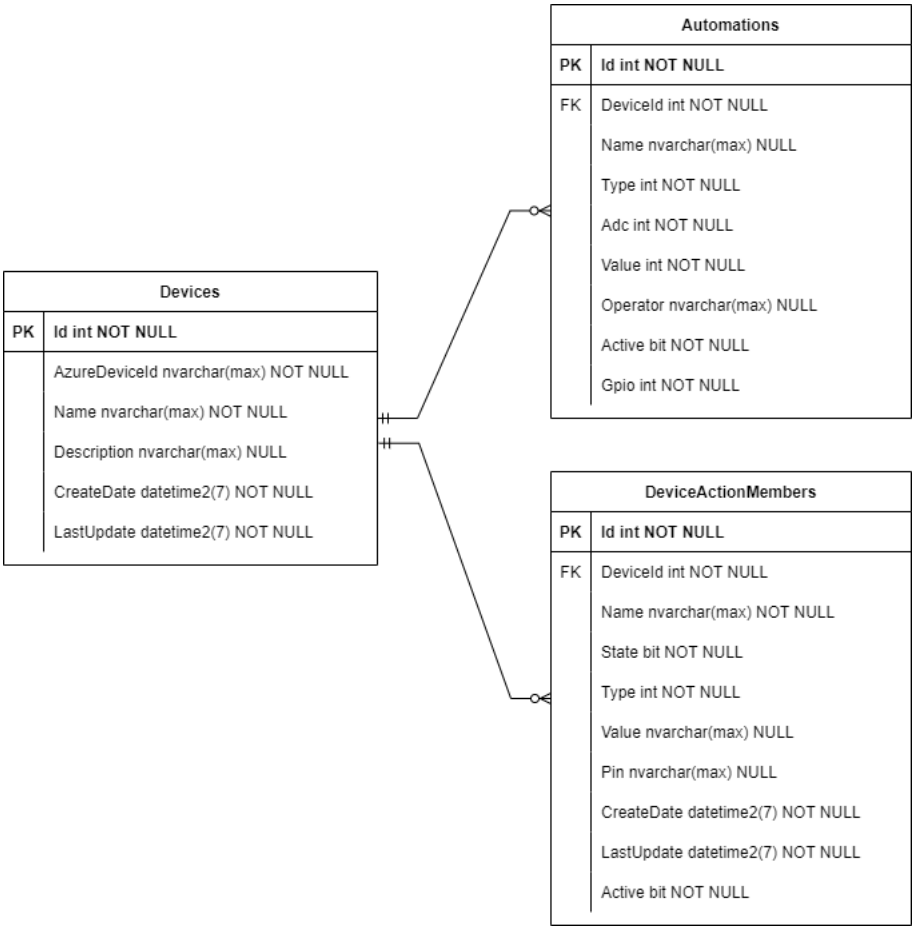
5.3.4 Správa zariadenie-cloud

V tejto správe sa nachádzajú hodnoty pre každý senzor, ktorý je na zariadení zaregistrovaný. Spolu s hodnotami senzorov sa posiela aj čas, kedy boli hodnoty namerané:

```
{
    "array": [
        {
            "pin":1,
            "value":1274
        }],
    "time":1650731017
}
```

5.4 Databázový model a konštanty webovej aplikácie

Používame relačnú databázu, v ktorej sú údaje uložené v tabuľkách a navzájom sú prepojené, musia mať medzi sebou vzťahy, väzby. Tabuľka je súborom údajov, ktoré sa týkajú jednej oblasti (entity), v objektovo orientovaných programovacích jazykoch môžeme túto tabuľku reprezentovať ako objekt. V našom databázovom modeli máme tri tabuľky/objekty, pre entitu Devices-zariadenia, Automations-automatizácie a DeviceActionMembers-senzory a akčné členy (obrázok 5 – 1).



Obrázok 5 – 1 Databázový model

Pri návrhu aplikácie sme sa sústredili na vytvorenie prototypu riešenia, nezaoberali sme sa prípravou IoT riešenia na uvedenie do praxe. Všimnúť si to môžeme na databázovom modeli, kde chýbajú entity pre prihláseného používateľa. Túto entitu používateľa, by sme následne spojili s entitou zariadenia, vzťahom jeden používateľ povinný k viacerým voliteľným zariadeniam (one mandatory to many optional).

V nasledujúcich odsekoch si vysvetlíme, prečo používame spomenuté entity a vysvetlíme si ich parametre. S týmito parametrami v podstate pracujeme v každej časti IoT riešenia.

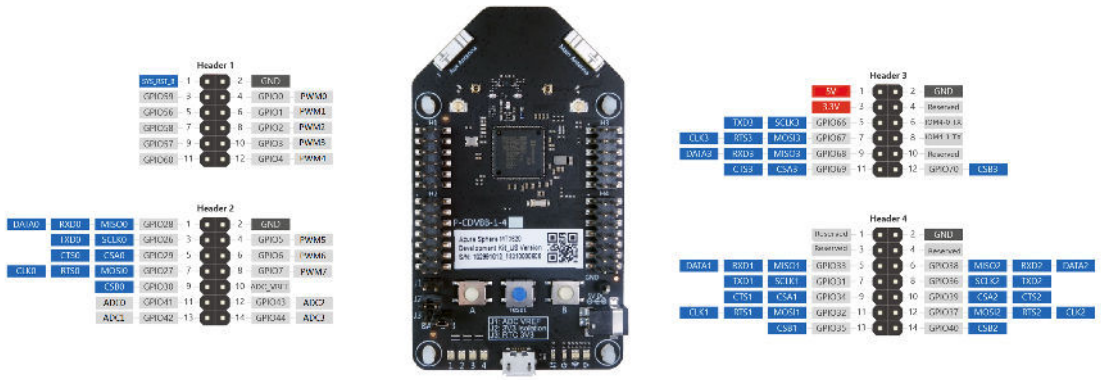
Entitu Devices využívame na opis zariadení, ktoré chceme cez webovú aplikáciu spravovať a ovládať. Parameter *AzureDeviceId* predstavuje unikátny identifikátor, vnútorné Id zariadenia Azure Sphere. Využívame ho na komunikáciu so službou IoT Hub, na volanie akcií spomínaných v kapitole 5.1.1. *Name* používame pre uloženie názvu zariadenia, ktorý si vyberáme sami pri vytváraní zariadenia. *Description* parameter nemusíme vypĺňať, je určený pre uloženie poznámok používateľa. Parametre *CreateDate* a *LastUpdate* sú typu datetime, používame ich na ukladanie časov vytvorenia a úpravy.

Entitu Automations používame na ukladanie automatizácií, s tabuľkou zariadenie je vo vzťahu jedno povinné k viacerým voliteľným automatizáciám, ako cudzí kľúč používame parameter *DeviceId* (predstavuje databázové Id, nie Id zariadenia priradené od IoT Hubu). *Name* predstavuje meno automatizácie, určujeme si ho my pre lepšiu identifikáciu. Pri *Type* určujeme o akú automatizáciu (kapitola 5.4.1 ide o enum *AutomationTypes*). *Adc* určuje ADC pin, z ktorého čítame hodnoty a následne porovnávame. *Value* je hodnota, s ktorou sa porovnáva. *Operator* je znamienko porovnania, *Active* určuje, či si želáme vykonávanie danej automatizácie, môžeme ju kedykoľvek vypnúť. *Gpio* udáva, ktorý pin automatizáciou ovládame.

Entitu DeviceActionMembers používame na ukladanie akčných členov a senzorov zariadenia. S tabuľkou zariadenie je vo vzťahu jedno povinné k viacerým voliteľným akčným členom, ako cudzí kľúč používame parameter *DeviceId* (predstavuje databázové Id, nie Id zariadenia priradené od IoT Hubu). *Name* si určujeme my pri vytváraní, zobrazuje sa nám pri manipulácii s akčným členom na front-ende webovej aplikácie. *State* môže nadobudnúť binárne hodnoty, vypnutý, zapnutý. Pri *Type* určujeme o aký akčný člen ide (5.4.1 ide o enum *TypeOfDeviceActors*). *Value* je hodnota akčného člena, ak ide o senzor, *Pin* je číslo pinu, kde máme akčný člen pripojený. *CreateDate* a *LastUpdate* sú parametre typu datetime, používame ich na ukladanie časov vytvorenia a úpravy. Cez *Active* vieme funkcionalitu akčného člena pozastaviť, ako by na danom pine žiadny akčný člen nebol.

5.4.1 Konštanty webovej aplikácie

Využívame na určenie funkcionality daných pinov zariadenia Azure Sphere a na určenie o aké typy automatizácií a akčných členov ide (pomocou enum). Rovnaké enum triedy máme aj na front-ende webovej aplikácie, využívame ich na zjednodušenie pridávania nových typov a pre lepšie porozumenie kódu. Priradenie funkcionality, čísla alebo Id jednotlivým pinom, sme urobili pomocou tohto rozdelenia (obrázok 5 – 2), kde máme zobrazené naše zariadenie. Piny máme uložené v triede Constants/konštanty na back-ende webovej aplikácie. Dátový model tejto triedy môžeme vidieť na obrázku 5 – 3. V nasledujúcich odsekoch si popíšeme kedy vyberáme aké typy a príklad dátovej štruktúry všetkých pinov zariadenia.

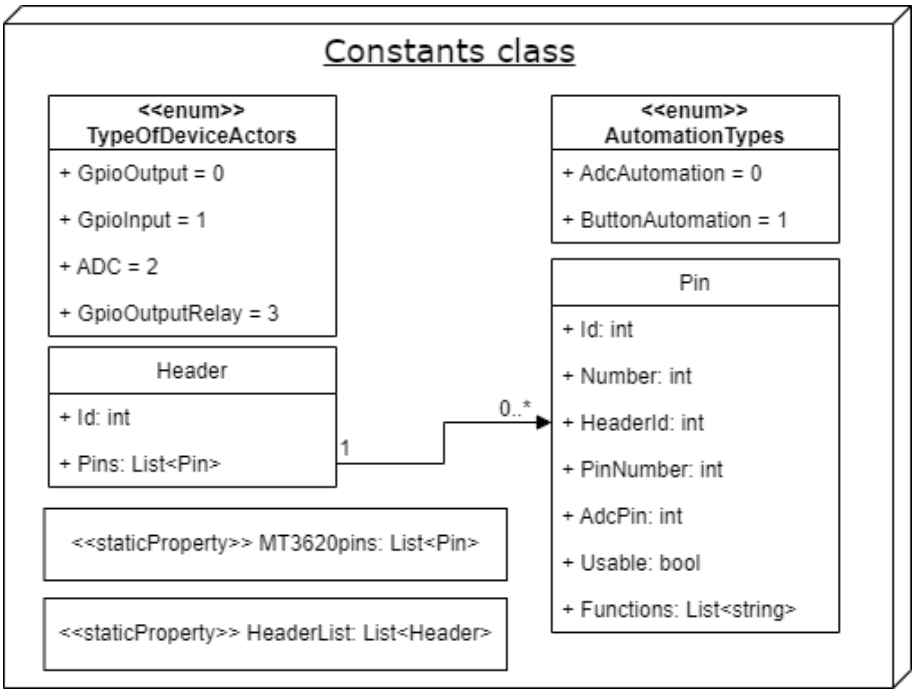


Obrázok 5 – 2 Azure Sphere pinout V-Annene (2022)

Enum TypeOfDeviceActors využívame na určenie o aký typ akčného člena/ senzora ide, pretože v zariadení musíme určiť, či z pinu chceme čítať alebo do pinu zapisovať. *GpioOutputRelay* – ide o prípad, kedy do pinu zapisujeme binárne hodnoty, ide o relé akčné členy. *GpioOutput* – ide o prípad, kedy do pinu zapisujeme binárne hodnoty, ide o akčný člen ledky, kedy musíme pre správnu funkcionalitu zapisovať opačnú hodnotu ako do akčného člena relé. *GpioInput* – z pinu neustále čítame binárne hodnoty, ide o funkcionalitu tlačidla. *ADC* – ide o čítanie hodnoty zo senzorov s presnosťou na 12-bitov.

Enum Automation Types využívame na určenie o aký typ automatizácie ide. Pre prototyp zariadenia máme len dve možnosti automatizácií. *AdcAutomatizácia* sa používa na ovládanie akčné člena pomocou stanovej podmienky, hodnoty z ADC

pinu a *ButtonAutomation* sa používa na zmenu stavu stanoveného akčného člena po stlačení zadaného tlačidla.



Obrázok 5 – 3 Back-end trieda Constants

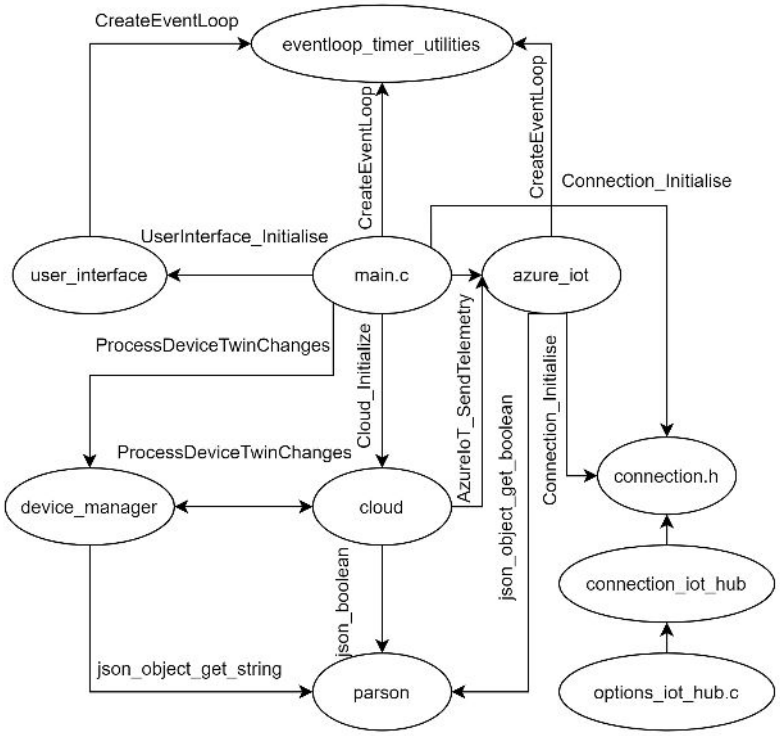
Ako môžeme vidieť na obrázku 5 – 3, k dispozícii máme 2 triedy (Header a Pin) Zariadenie (obrázok 5 – 2) má 4 oddelené časti pinov (header) a každá časť obsahuje list pinov. Statický parameter HeaderList obsahuje 4 Header objekty, každý objekt obsahuje svoje piny (objekt typu Pin). Pre ilustráciu si uvedieme príklad JSON objekt jedného pinu, s označením GPIO41:

```
{"Id":22,"HeaderId":2,"Number":11,"PinNumber":41,"AdcPin":0,
"Usable":true,"Functions":["GPIO41","ADC0"]}
```

Id, *HeaderId* a *Number* používame jedine pre zobrazovanie na webovom rozhraní aplikácie, funkcionálne nemajú žiadnu dôležitú úlohu. *PinNumber* používame napríklad v správach typu zariadenie-cloud (kapitola 5.3.4), kedy chceme akčnému členu na pine 41 zmeniť stav. *AdcPin* je buď vyplnený (pin podporuje ADC), alebo má hodnotu null. *Usable* využívame taktiež len pre zobrazovanie, ak je hodnota false, nedovolíme používateľovi na danom pine vytvoriť žiadny akčný člen alebo senzor. *Functions* zobrazujeme ako popis pinu, aby používateľ vedel načo tento pin môže slúžiť.

5.5 Zariadenie Azure Sphere

V tejto časti sa pozrieme na architektúru zariadenia, popíšeme si jednotlivé súbory, pričom si uvedieme na čo ich používame a najdôležitejšie parametre a funkcie aj krátko popíšeme. Len vecne si prejdeme konfiguračné súbory, ktoré musíme pre správne fungovanie zariadenia spravovať a nastavovať.



Obrázok 5 – 4 Diagram prepojení

Na obrázku 5 – 4 môžeme vidieť súbory, v ktorých sa nachádza všetok náš 'vlastný' kód, obyčajné knižnice (stdio.h, stdlib.h, time.h) sme do diagramu nepridávali, nakoľko nás až tak nezaujímajú a diagram by sa pri terajšej veľkosti stal nečitateľným. Súbory ktoré nemajú príponu, sú spojenie C kódu a hlavičkového súboru (cloud – cloud.c, cloud.h) Pri vývoji kódu sme použili príklad z gitu od Microsoftu, v ktorom je vyriešená základná komunikácia zariadenia s cloudovou službou IoT Hub. Našou hlavnou úlohou bolo vytvorenie modelov dát, ktoré do a zo zariadenia odchádzajú. Následne tieto dáta uložiť a vykonávať pomocou nich akcie (automatizácie). Všetky tieto akcie sme museli vyriešiť genericky, nakoľko sa má používateľ sám rozhodnúť, čo chce mať na akom pine. Taktiež sme museli riešiť ovládanie rôznych typov akčných

členov a senzorov.

5.5.1 Konfiguračné súbory

V tejto podkapitole sa budeme venovať konfiguračným súborom, ktoré sme museli upraviť, aby príklad z gitu fungoval. Súbory nebudeme riešiť do hĺbky, nakoľko ide len o nastavenia. Už sme spomínali, že zariadenie Azure Sphere bolo vytvorené so silným dôrazom na bezpečnosť. Spomínali sme to pri komunikácii medzi zariadením a cloudom. Špecifickosť tohto zariadenia je taktiež v prístupe k ovládaniu pinov, nakoľko musíme piny dopredu zaregistrovať, že na nich budeme vykonávať nejakú špecifickú úlohu. Takto vieme pridať ďalšiu vrstvu zabezpečenia, na úrovni periférií, zamedzením pripojenia nechcených periférií.

Súbor **app_manifest.json** primárne slúži pre zaregistrovanie schopností zariadenia. V našom príklade sme zaregistrovali piny, ktoré môžeme používať, povolili sme komunikáciu s IoT Hubom a pridali sme autentifikáciu pomocou ID vlastníka zariadenia. Existuje veľké množstvo ďalších schopností, ktoré sa dajú zaregistrovať, napríklad SystemEventNotifications, AllowedTcpServerPorts, WifiConfig alebo PWM, UART:

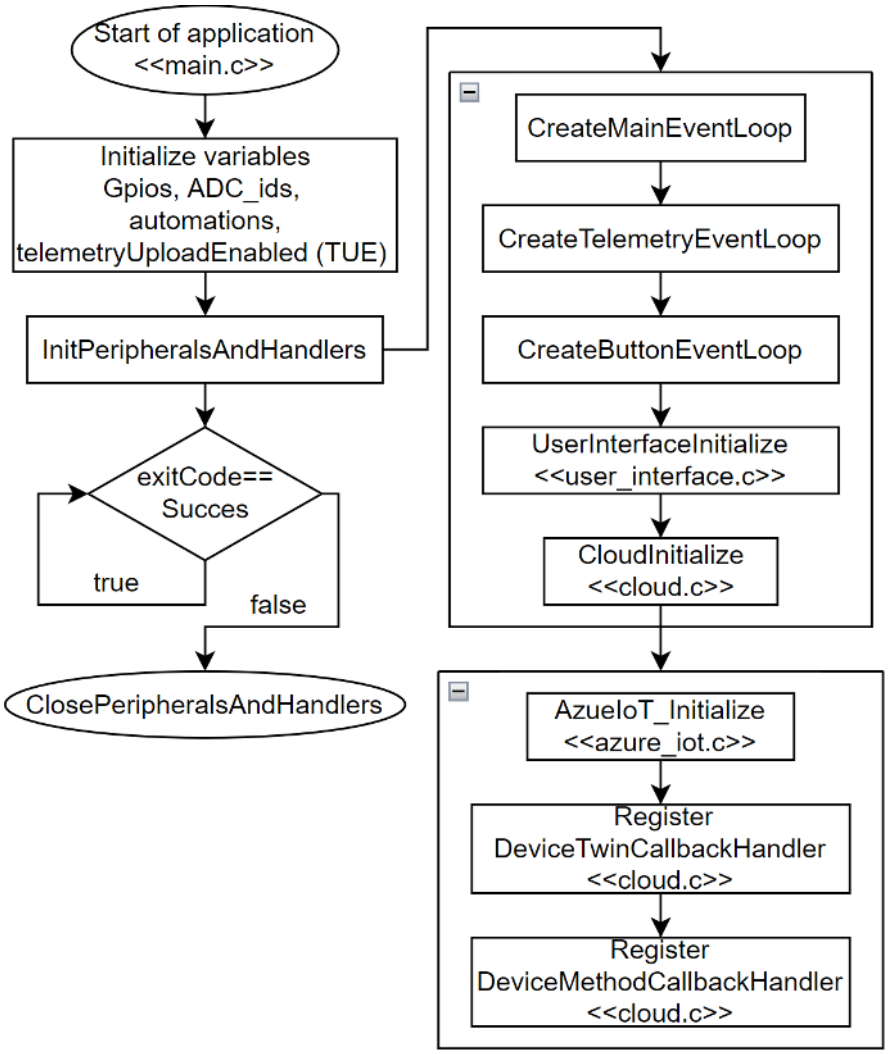
```
"Capabilities": {
    "AllowedConnections": [ "iot_hub_address" ],
    "Gpio": [ "$SAMPLE_LED", "$GPIO66", "$GPIO67" ],
    "Adc": [ "$SAMPLE_POTENTIOMETER_ADC_CONTROLLER" ],
    "DeviceAuthentication": <tenant_id>
}
```

Súbory **sample_appliance.h** a **sample_appliance.json** používame na mapovanie pinov zariadenia na macro, ktoré môžeme následne používať všade v kóde zariadenia. Macro je typu int a jeho hodnota je totožná s parametrom PinNumber (obrázok 5 – 3):

```
{   "Name": "GPIO66",
    "Type": "Gpio",
    "Mapping": "MT3620_GPIO66",
    "Comment": "MT3620 GPIO66"
}
```

5.5.2 Architektúra programu zariadenia

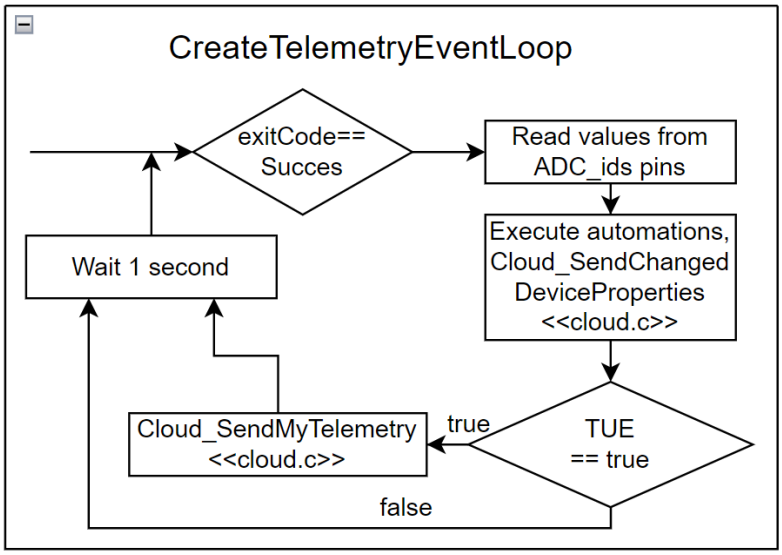
Pre vysvetlenie a priblíženie častí kódu a architektúry programu zariadenia, sme vytvorili stručné vývojové diagramy. Diagramy sa nevenujú každému detailu, zobrazujú zjednodušený pohľad na architektúru.



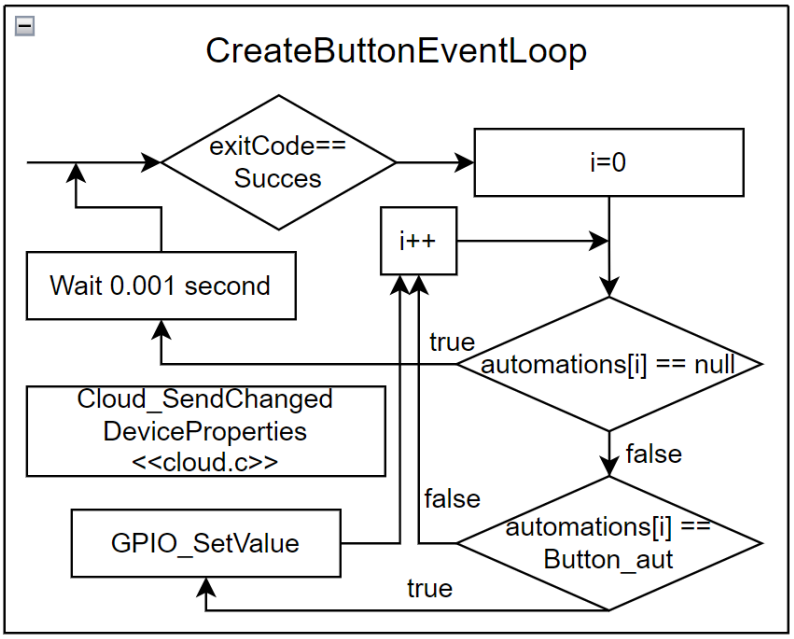
Obrázok 5 – 5 Najvyššia úroveň architektúry

Na obrázku 5 – 5 môžeme vidieť začiatok programu, inicializáciu premenných, ktoré držia údaje (automatizácie, ADC a GPIO piny) získané z DT. Následne vykonáme inicializáciu periférií a zaregistrovanie funkcií na konkrétne udalosti, ako je prijatie cloud-zariadenie správy alebo zmena DT. Program končí, ak hocijaká funkcia z hocijakej časti kódu zariadenia vyprodukuje invalidný exitCode. V tomto prípade

sa zavolá funkcia na vyčistenie zariadenia, odhlásenie sa z odoberaní dát z udalostí alebo zrušenie čítania dát z ADC pinov. V nasledujúcich obrázkoch, si priblížime ako fungujú EvenLoop a Handler pre čítanie, prijímanie a posielanie dát.



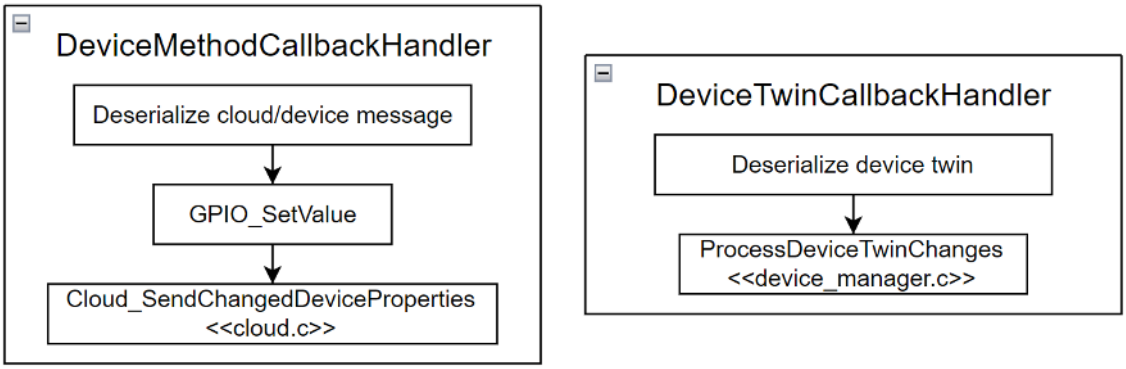
Obrázok 5 – 6 CreateTelemetryEventLoop



Obrázok 5 – 7 CreateButtonEventLoop

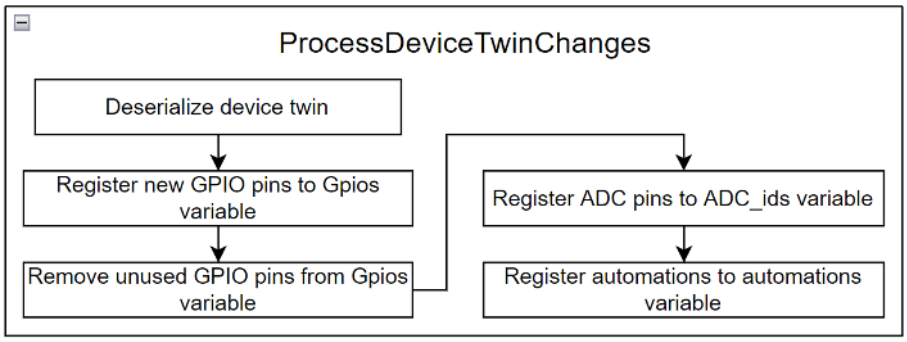
Na obrázkoch 5 – 6 a 5 – 7 môžeme vidieť dva cykly, ktoré sa vykonávajú počas

celého chodu programu. V podstate vykonávajú podobné činnosti, čítanú hodnoty z periférií, vyhodnocujú automatizácie, odosielajú telemetriu na back-end webovej aplikácie a aktualizujú DT počas najnovších stavov zariadenia.



Obrázok 5 – 8 DeviceMethodCallbackHandler a DeviceTwinCallbackHandler

Na obrázku 5 – 8 zobrazujeme dve funkcie, *DeviceMethodCallbackHandler* funkcia prijíma cloud-zariadenie správy (kapitola 5.3.2), spracováva ich, zmení stav zariadenia na danom pine a zmení hlásené parametre DT.*DeviceTwinCallbackHandler* je druhá funkcia, ktorá volá funkciu pre spracovanie a uloženie DT, ktorý bol zmenený zo strany back-endu webovej aplikácie, cloudu. Priebeh spracovania správ môžeme vidieť na obrázku 5 – 9.



Obrázok 5 – 9 ProcessDeviceTwinChanges

Vo vývojových diagramoch sme si uviedli, ktoré súbory využívame na dosiahnutie cieľovej funkcionality zariadenia. Máme aj ďalšie súbory, ktoré využívame na bočné úlohy, ktoré nás pre pochopenie architektúry nezaujímajú. Napríklad *parson.c* využívame na serializáciu a deserializáciu JSON správ, *eventloop_timer_utilities.c* na vytvorenie paralelného vlákna od hlavného vlákna *main.c* (odber zmien DT).

5.6 Back-end webovej aplikácie

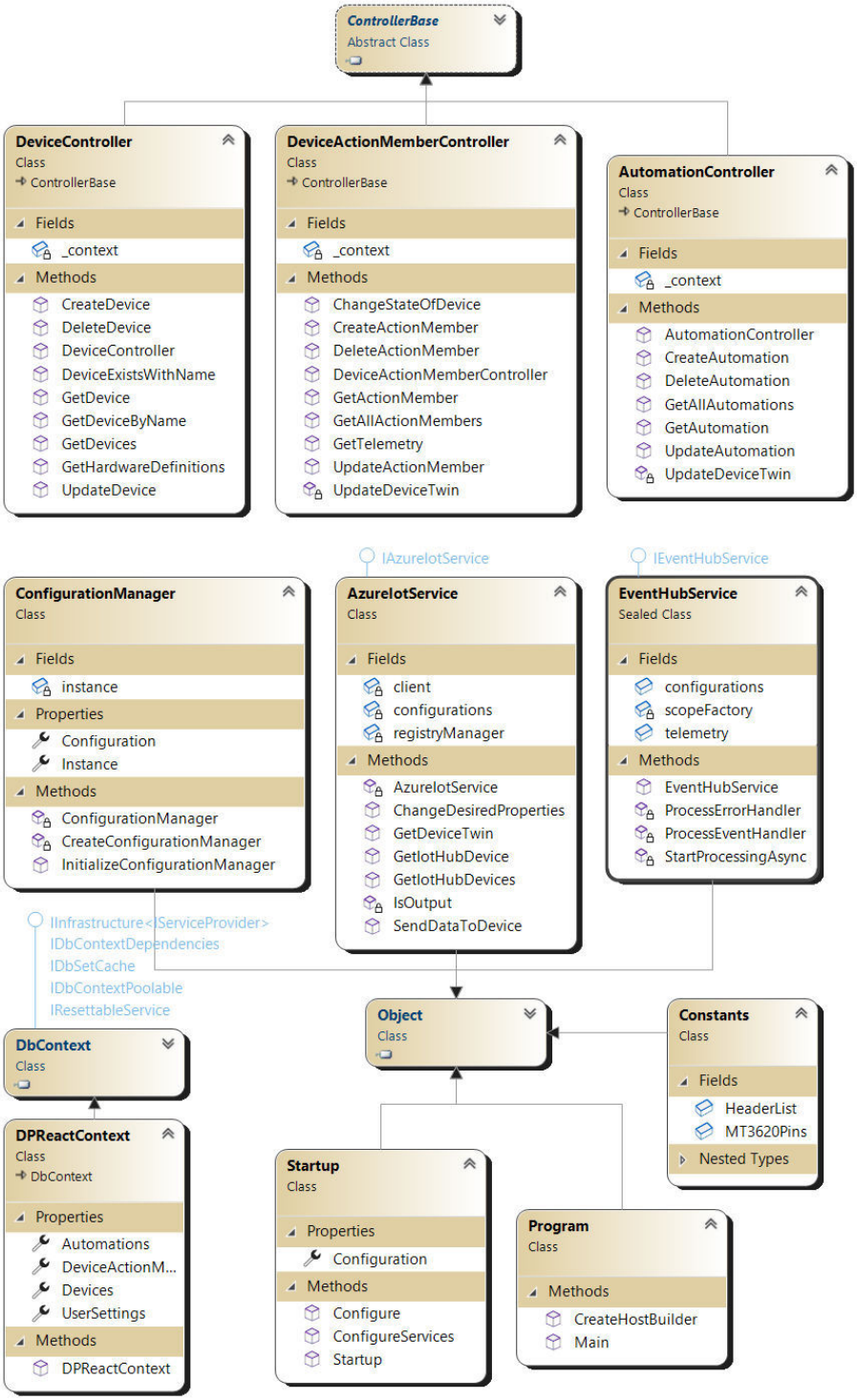
Návrh back-endu sme predstavili v kapitole 4.1.2, pričom po zmene celkovej architektúry sa taktiež čiastočne zmenil back-end. Pre komunikáciu so zariadením sme používali API volania, pri terajšej architektúre 4 – 5 sme prešli na využívanie Microsoft knižníc pre C#. Pri opise tried a nastavení, sa budeme odkazovať na diagram funkcionálnych tried back-end 5 – 10, kde nájdeme všetky parametre a funkcie. Začnime opisom knižníc, načo a kde ich používame:

- **Event Hubs** – táto knižnica slúži na vytvorenie procesora pre prijímanie zmien DT alebo prichádzajúcich správ typu zariadenie-cloud 5.3.4. Následne prichádzajúce správy spracujeme, podľa toho z akého zdroja prichádzajú. Poznáme dva zdroje (twinChangeEvents a Telemetry). Event Hubs ponúkajú veľmi dôležitú funkcionalitu, pretože zbierajú rôzne druhy správ od zariadení, aj keď je náš procesor nedostupný. Po pripojení procesora sa správy začnú znova spracovávať. Po spracovaní správy, ju chceme z Event Hubu vymazať, aby sme ju nedostali na spracovanie viac krát. Robíme to aktualizovaním kontrolného bodu.

- **Storage Blobs** – ide o ukladanie obrovského množstva neštruktúrovaných údajov, BLOB objektov. Neštruktúrované údaje sú údaje, ktoré nezodpovedajú konkrétnemu údajovému modelu alebo definícii, ako sú napríklad textové alebo binárne údaje. V našom prípade túto službu využívame na ukladanie správ Event Hubu. Zbierať dáta môžeme z hocijakých zdrojov, ktoré si k službe pripojíme.

- **Azure devices** – táto knižnica slúži na kompletnú komunikáciu s cloudovou službou Azure IoT Hub (popis nájdeme v kapitole 2.4.1). Túto knižnicu výlučne používame v triede *AzureIoTService*, nachádzajú sa tu funkcie pre odosielanie cloud-zariadenie správ, vytiahnutie DT pre dané zariadenie a zmenu požadovaných parametrov DT.

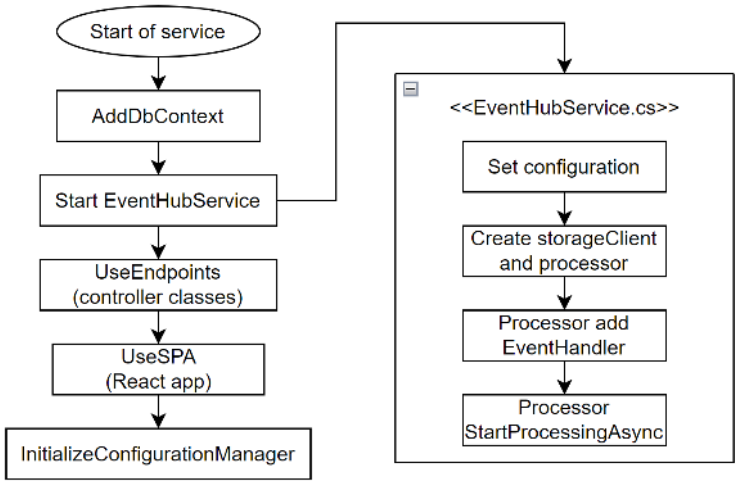
Prihlasovacie údaje do vyššie uvedených služieb a databázy ukladáme cez súbor *appsettings.json*. Momentálne používame lokálnu databázu, avšak v konfigurácii máme taktiež pole pre zadanie connectionString pre Postres SQL (Structured Query Language) databázu (pre tento typ databázy máme stiahnutú knižnicu).



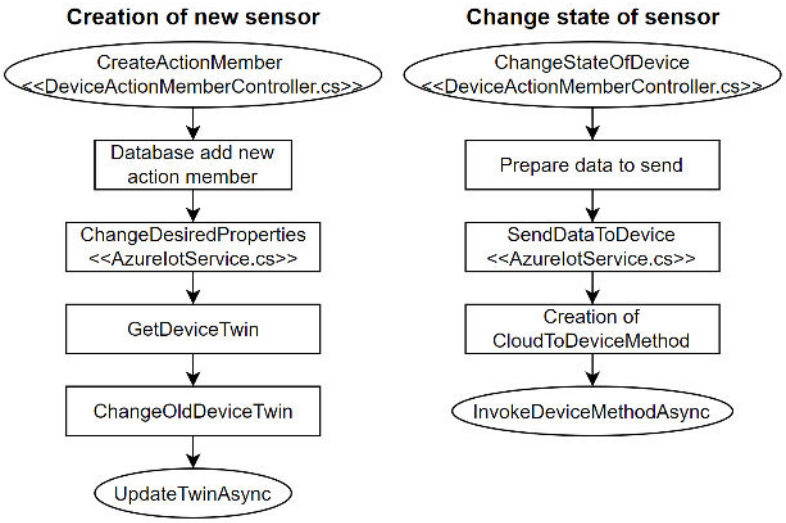
Obrázok 5 – 10 Diagram tried back-endu webovej aplikácie

5.6.1 Vývojové diagramy komunikácie so zariadením

Na nasledujúcich obrázkoch si predstavíme vývojové diagramy pre komunikáciu webovej aplikácie so zariadením a uvedieme si, kedy tieto prípady nastávajú. Diagram 5 – 11 zobrazuje štart webovej aplikácie, kedy zaregistrujeme databázu, koncové body, front-end, spustíme *EventHubService* a inicializujeme *ConfigurationManager* (pre prístup ku konfigurácii mimo kontrolérov). Kontroléry back-endu si predstavovať nebudeme, nakoľko používame štandardnú architektúru .NET 6 aplikácie.



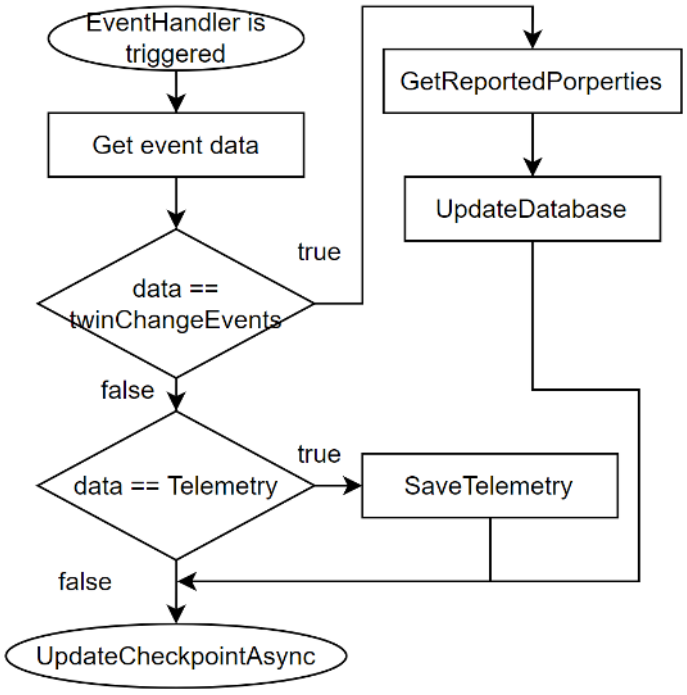
Obrázok 5 – 11 Startup back-endu webovej aplikácie



Obrázok 5 – 12 Príklady posielania správ do zariadenia

Na obrázku 5 – 12 môžeme vidieť dva diagramy, pre posielanie dát webovej aplikácie smerom do zariadenia. Ako prvý si predstavíme príklad vytvorenia nového senzora na zariadení (Creation of new sensor). V tejto správe posielame cloud zmení DT typ správy. Objekt ktorý posielame a popis tejto správy máme popísaný v kapitole 5.3.1.

Druhý diagram predstavuje akciu zmeny hodnoty senzora na zariadení (Change state of sensor). Využívame správu typu cloud-zariadenie 5.3.2. Pre komunikáciu so zariadením používame výhradne triedu *AzureIotService.cs*.



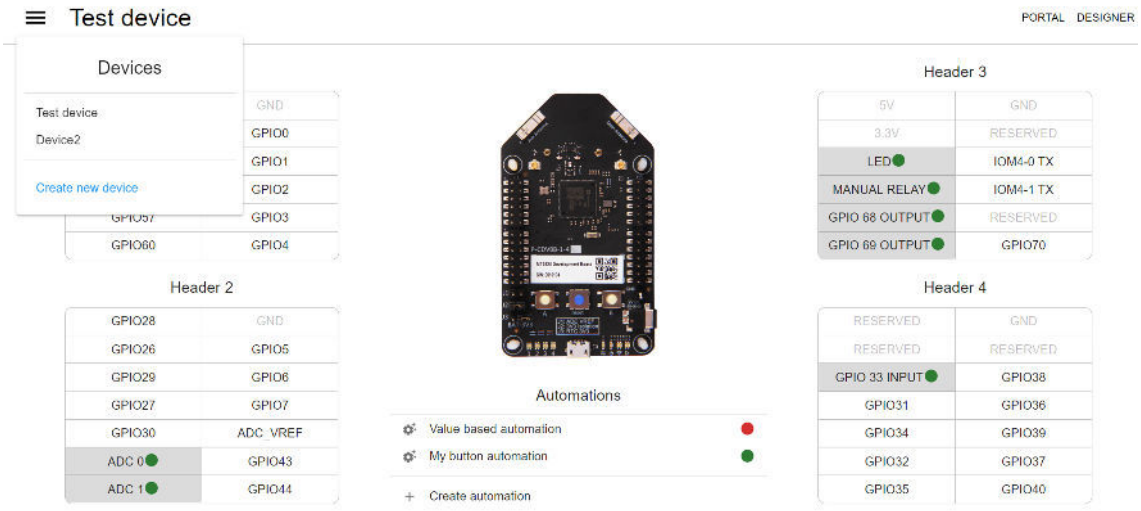
Obrázok 5 – 13 Dáta posielané zo zariadenia

V obrázku 5 – 13 môžeme vidieť dva prípady, prvý je, keď zariadenie zmení DT (vetva kedy sa podmienka data==twinChangeState rovná true). Ohlásené parametre (stavy akčných členov, vnútorné premenné) spracujeme a aktualizujeme databázu. Ide o prípad správy typu zariadenie zmení DT (kapitola 5.3.3). Druhý prípad je posielanie správ typu zariadenie-cloud (kapitola 5.3.4), kedy zariadenie odosiela telemetriu, hodnoty senzorov. Po spracovaní a reakcii na obidva prípady aktualizujeme kontrolný bod, čím dáme vedieť službe Event Hub, že túto správu sme už spracovali a môže ju vymazať z pamäte služby Storage Blobs.

5.7 Front-end webovej aplikácie

Návrh front-endu webovej aplikácie sme predstavili v kapitole 4.1.3, kde sme si uviedli pomocou akej technológie budeme grafický interface vytvárať a aké knižnice pri tom využijeme. Architektúra front-endu nie je nijako zaujímavá, urobili sme ju pomocou klasických prístupov vytvárania webových aplikácií. Front-end má dve hlavné stránky, Designer, pre pridanie, úpravu a vymazanie akčných členov, senzorov a automatizácií. Portal pre zobrazenie hodnôt senzorov a stavov akčných členov a ich ovládanie.

V nasledujúcich obrázkoch si ukážeme ako vyzerá grafické prostredie aplikácie, jednotlivé funkcionality jednotlivých stránok a dialógov, čo sme pri nich museli zabezpečiť.

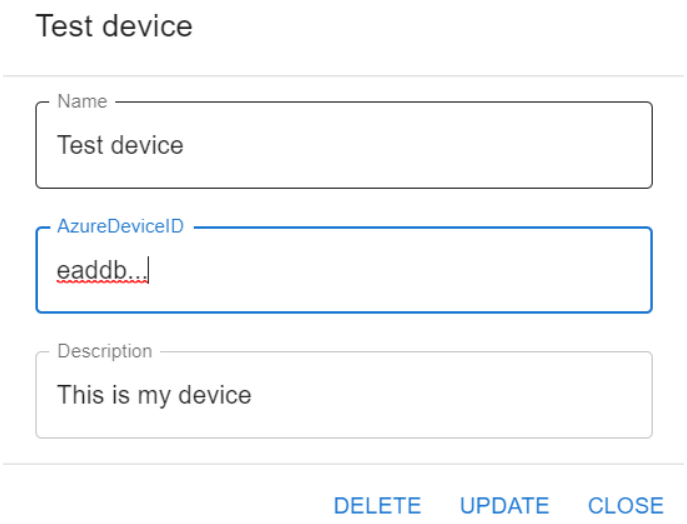


Obrázok 5 – 14 Obrazovka Designer

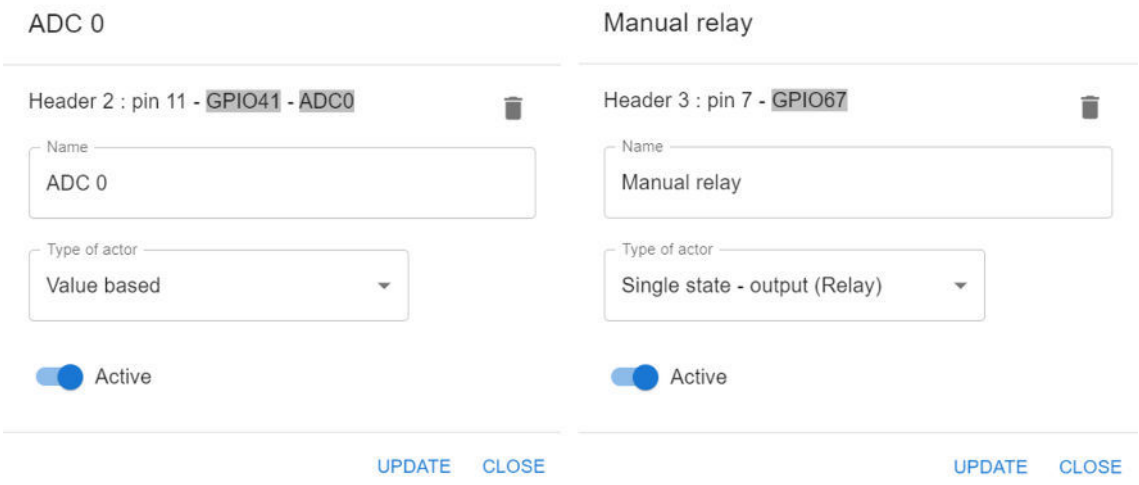
Obrazovka Designer (obrázok 5 – 14) umožňuje používateľovi spravovať zdroje (periférie) zariadenia. Ako prvé si predstavíme hornú lištu, kde sa vieme medzi našimi dvomi stránkami prepínať. Na ľavej strane lišty, máme vyskakovacie menu pre pridávanie nových zariadení a pre výber medzi existujúcimi zariadeniami. Pre tento štýl sme sa rozhodli, kvôli zjednodušeniu používania. Prvotný návrh bola osobitná stránka na výber zariadenia, čo zbytočne pridávalo jeden krok naviac.

Na ľavej a pravej strane Designer obrazovky vidíme rozloženie jednotlivých pinov. Pre nepoužiteľné piny sme zakázali možnosť otvorenia dialógu pre úpravu pinu.

Piny ktoré sú využívané sme pre lepšiu identifikáciu vyfarbili sivou farbou. Môžeme si všimnúť že pod zariadením sa nachádzajú automatizácie, s možnosťou pridania novej. Každý použitý pin a automatizácia zobrazujú meno, ktoré sme im my navrhli. Vedľa mena zobrazujeme farebné bodky, ktoré udávajú, či je akčný člen alebo senzor v aktívny. Po kliknutí na zariadenia nám vyskočí dialóg pre úpravu zariadenia (obrázok 5 – 15).

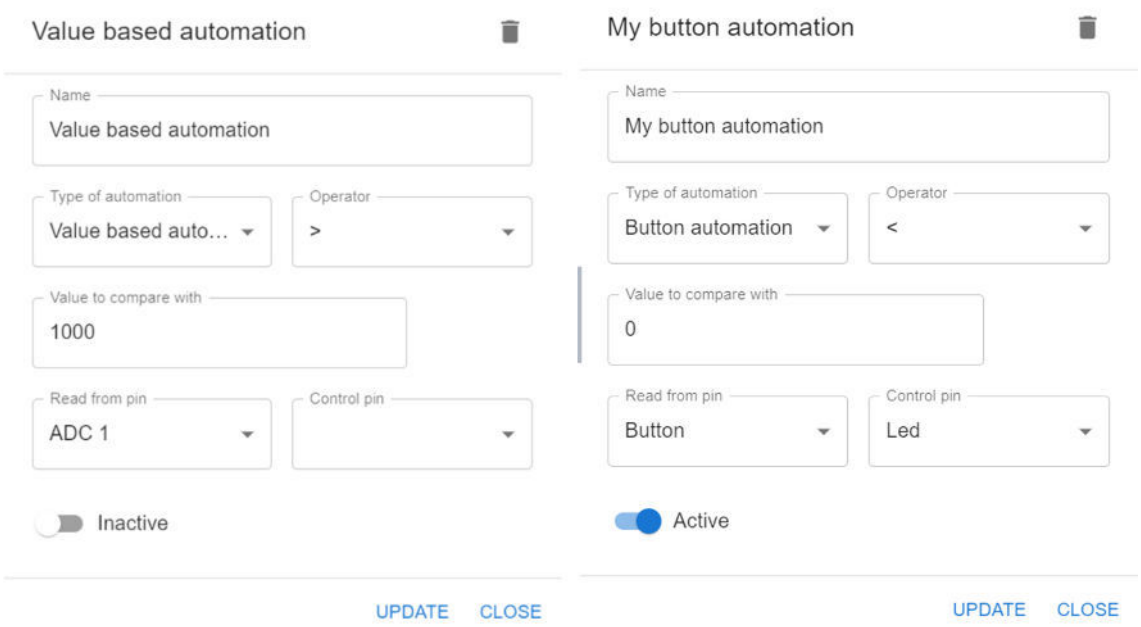


Obrázok 5 – 15 Dialóg pre úpravu zariadenia



Obrázok 5 – 16 Dialóg pre úpravu pinov zariadenia

Na obrázku 5 – 16 zobrazujeme dialógy pre úpravu pinov zariadenia. Konkrétne nastavenie pinu GPIO41 ako senzora, budeme z neho čítať hodnoty a pinu GPIO67 ako akčného člena, konkrétne relé.



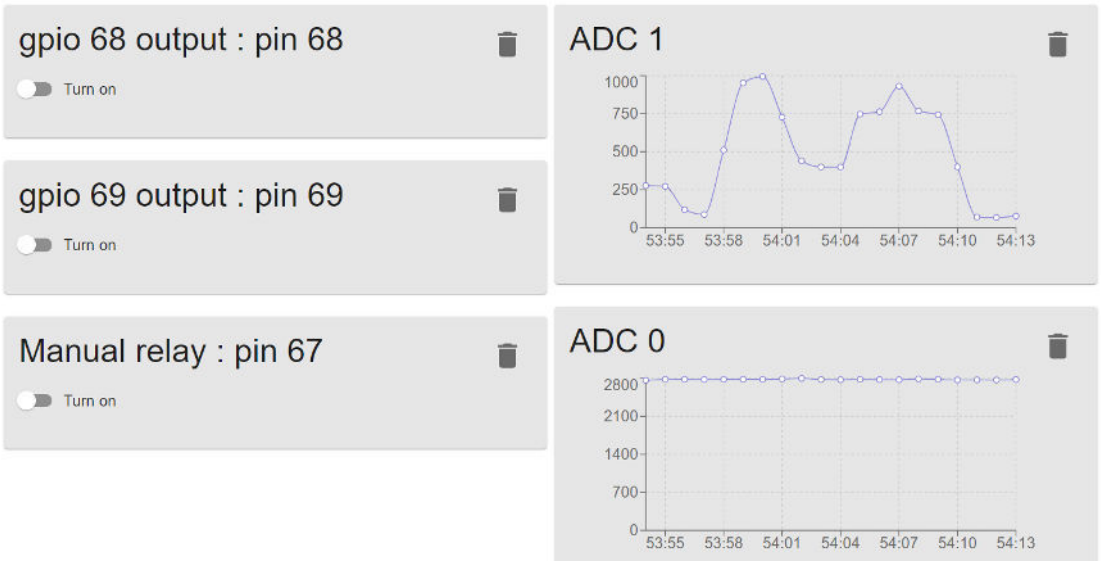
Obrázok 5 – 17 Dialógy pre úpravu automatizácií

Obrázok 5 – 17 zobrazuje dialógy pre úpravu dvoch automatizácií. Dialóg automatizácie sme navrhli tak, aby mal používateľ na výber zo správnych pinov, podľa toho aký typ automatizácie si vyberie. Ak si vyberie *Button automation*, má na výber čítanie len z pinov, ktoré sú typu tlačidla. To isté platí pre hodnotové automatizácie.

Túto funkcionalitu sme navrhli, aby mal používateľ jednoznačný prehľad, čo môže a nemôže urobiť, nakoľko je toto IoT riešenie mierené aj na ľudí, ktorí nemajú rozsiahle informatické poznatky. Slúži to aj ako ošetrenie priamo pri interakcii s webovým rozhraním, bez tejto funkcionality by sme museli z back-endu vraciať hlášky o nekompatibilnosti.

Na obrázku 5 – 18 vidíme Portal obrazovku, z ktorej môžeme sledovať stavy akčných členov, ovládať ich alebo poprípade ich vymazať. Taktiež vidíme hodnoty zo senzorov, ktoré sú reprezentované v čiarovom grafe. Hodnoty pre graf získavame

každú sekundu, nakoľko v takomto intervale posiela zariadenie telemetriu na cloud. Funkcionalita pre nastavenie frekvencie posielania telemetrie by sa dala jednoducho pridať, cez nový parameter pre model zariadenia. Dáta zobrazujeme pre 20 za sebou idúcich hodnôt.



Obrázok 5 – 18 Obrazovka Portal

6 Vyhodnotenie dosiahnutých výsledkov

V práci sme už raz vyhodnotenie urobili a to pre prvotný návrh 4.2. V odrážkach sme si vypísali, aké boli nevýhody prvého návrhu a v druhom sme tieto nedostatky adresovali pomocou zariadenia Azure Sphere a Azure IoT služieb. Novou architektúrou sme sa taktiež 'zbavili' potreby budovania komplexného IoT manažéra zariadení, na správu zariadení a komunikáciu využívame overené knižnice. Týmto spôsobom sme zabezpečili veľkú časť overeného kódu, ktorý nemusíme spravovať a riešiť s ním problémy. Architektúra nám poskytuje možnosti vývoja na akýchkoľvek zariadeniach, nie len Azure Sphere, nakoľko aj RTOS využívajú na pripojenie službu IoT Hub.

Medzi ďalšie výhody architektúry je rýchlosť a spoľahlivosť doručenia všetkých správ medzi zariadením a webovou aplikáciou. Tieto výhody zabezpečuje back-end webovej aplikácie. Sme veľmi spokojní s využitím Entity Frameworku pre uľahčenie práce s databázou, využitím .NET 6 ako základu back-endu. Vďaka spomenutým kombináciám technológií a služieb dosahujeme instantnú komunikáciu (zariadenie-webové rozhranie). Ak sa jedná o správy z cloudu do zariadenia, latenciu správ udávame na úrovni desiatok milisekúnd. Pri opačnom smere správ je oneskorenie o niečo vyššie. Medzi zariadením a cloudom je komunikácia takmer okamžitá, problém nastáva pri aktualizácii zobrazenia najnovších stavov na front-ende. Spôsobené to je tým, že webové rozhranie aktualizujeme každú jednu sekundu. Táto 'nepríjemnosť' sa ale dá jednoducho vyriešiť a to zvýšením počtu dopytov pre aktualizáciu (zmena jedného parametru na front-end strane). V našom prípade nám ale táto implementácia postačuje.

S front-end technológiami sa nám taktiež robilo veľmi dobre, nakoľko sme namiesto vytvárania a štýlovania vlastných komponentov, použili knižnice s optimalizovanými a konzistentnými komponentami. Konzistentnosť spočíva v spoločnom štýle (veľkosť, farba) pre celú knižnicu. Vďaka týmto knižniciam, sme sa mohli viacej venovať aktualizáciám stavov a vytváraniu funkcionality. S jazykom typescript sme taktiež spokojní, nakoľko je podobný jazyku C#, môžeme v ňom používať typy.

Celkovo sa s využitými knižnicami, technológiami a službami robilo počas práce

veľmi dobre. Väčšina má dobre napísanú dokumentáciu, aktívny vývoj, veľkú základňu používateľov a riešení, na rôzne typy problémov.

Osobitné vyhodnotenie by sme mali venovať taktiež práci s Azure Sphere zariadením, nakoľko nie je populárne ako napríklad Raspberry Pi a je pomerne nové a špecifické. Ak sa jedná o dostupnú dokumentáciu, Microsoft všeobecne ponúka excelentnú dokumentácie pre svoje produkty a výnimkou nebola ani dokumentácia pre toto zariadenie. Ako základ kódu sme použili open-source príklad od Microsoftu, ktorý bol dobre zdokumentovaný, jasne štrukturovaný, s jasnými názvami premenných a funkcií.

Pre vývoj na zariadení sme museli využívať Azure Sphere SDK. Počas vývoja sa vyskytla chyba s týmto SDK, ktorú sme oznámili Microsoftu a nakoniec sa nám ho podarilo opraviť. Z pohľadu zdroja chyby, to nebola vôbec príjemná skúsenosť, nakoľko sme nad tým strávili pomerne veľa času (3 celé dni). Z pohľadu podpory Microsoftu, to bola príjemnejšia skúsenosť, podporný tím nám poskytol okamžitú pomoc. V konečnom dôsledku nás podporný tím nasmeroval správnym smerom a chybu sme opravili.

V konečnom dôsledku všetky časti zariadenia, či už fyzická časť, programová časť alebo dokumentácia, tvoria slušný základ pre vývoj produktov s využitím Azure Sphere. S Azure Sphere sme nad mieru spokojní, pre školské potreby ho neodporúčame, nakoľko vývojový kit sa v produkcii používať nesmie. V produkcii si musí zákazník vytvoriť vlastnú vývojovú dosku s mikroprocesorom Azure Sphere, v tomto prípade používanie zariadenia odporúčame pre aplikácie, kde je bezpečnosť najväčšia priorita. Odhliadnuc od používania tejto konkrétnej riadiacej jednotky, určite odporúčame pripojenie akýchkoľvek iných zariadení (RTOS) ku cloudovej službe IoT Hub.

6.1 Zhodnotenie písomnej časti

Pre ucelenú definíciu dosiahnutých výsledkov, v tejto časti v odrážkach jasne a stručne zhrnieme dôležité body, ktoré sme v práci dosiahli. Začneme vymenovaním teoretických bodov:

- definovali sme cloud a internet vecí,

- urobili sme podrobný rozbor služieb dvoch najznámejších verejných poskytovateľov IoT služieb a okrajovo sme spomenuli služby iných poskytovateľov,
- porovnali sme cloudové služby a vyhodnotili cenové ponuky,
- porovnali sme mikroprocesor a mikrokontrolér,
- urobili sme analýzu piatich najznámejších riadiacich jednotiek,
- podrobne sme si popísali nami vybranú riadiacu jednotku Azure Sphere.

Praktické body:

- vytvorili sme prvotné IoT riešenie s riadiacou jednotkou Raspberry Pi,
- navrhli sme back-end a front-end webovej aplikácie,
- vyhodnotili sme prvotné riešenie a zdôvodnili výber inej riadiacu jednotku,
- vytvorili sme databázový model webovej aplikácie,
- navrhli sme architektúru nového riešenia so zariadením Azure Sphere,
- do hĺbky sme popísali Device Twin a jeho využitie v IoT riešení,
- popísali sme zmenu back-endu webovej aplikácie,
- definovali sme toky dát (aj štruktúry) medzi zariadením a cloudom,
- popísali sme realizáciu konfigurovateľného zariadenia,
- popísali sme realizáciu back-endu,
- popísali sme realizáciu front-endu,
- vytvorili sme funkčné konfigurovateľné zariadenie,
- vyhodnotili sme implementované riešenie.

7 Záver

Môžeme povedať, že sme plnili všetky body zadávacieho listu. Urobili sme analýzu riadiacich jednotiek s možnosťou pripojenia do siete internet a IoT služieb verejných cloudových poskytovateľov. Navrhli sme konfigurovateľné zariadenie a návrh sme úspešne realizovali. Prišli sme na to, že existuje množstvo kombinácií, ako sa služby dajú kombinovať navzájom vrámci jedného cloudového poskytovateľa, čo nám môže priniesť komplexné a lacné riešenie. Získali sme vedomosti o riadiacich jednotkách a ich možnostiach, vďaka čomu by sme si teraz vedeli vybrať zariadenie na mieru podľa špecifikácie úlohy, ktorú majú riešiť.

Zistili sme, že IoT riešenia netreba vytvárať od základu, riešiť komunikáciu, operačný systém (RTOS), manažment zariadení, dvojča zariadenia. Na svetovom trhu IoT cloudových riešení sa tejto problematike venuje množstvo firiem, ktoré ponúkajú špičkové zabezpečenie, spoľahlivosť a takmer hotové riešenia. Tento prístup firiem otvára neobmedzené možnosti využitia jednoduchých aj zložitých IoT riešení. Trend pripájania inteligentných IoT zariadení už začal, avšak potenciál ešte zďaleka nedosiahol. V budúcnosti bude čoraz väčší dôraz na efektívne využívanie zdrojov, zbieranie informácií na optimalizáciu procesov. Kvôli tomu si myslíme, že IoT svet je veľmi perspektívna cesta a oplatí sa ňou zaoberať.

S našou implementáciou sme sa nedotkli všetkých dôležitých konceptov moderných IoT zariadení, napríklad OTA aktualizácií. V téme by sa dalo pokračovať v smere vylepšení na zariadení ale aj na webovej stránke. Ku zariadeniu by sa dali pripojiť ďalšie senzory (rôzne zbernice), ktoré sme kvôli krátkosti času vyskúšať nemohli. Taktiež by sa dali implementovať rôzne automatizácie, ktoré by používateľ v praxi mohol využiť. Napríklad zapínanie akčných členov, podľa času a dátumu.

Na stránke by sa dalo pokračovať vo vylepšení dizajnu, vytvorení prihlasovania pre rôznych používateľov. Taktiež by sa dalo pracovať na funkcionalite pre vytváranie univerzálnych automatizácií. Pri grafickej reprezentácii dát, by sa dal vylepšiť dizajn, zobrazovanie dát za rôzne časové obdobia, zobrazovanie rôznych typov grafov. Jedna z komplexnejších úloh, by mohlo byť spracovanie a použitie dát pre analýzy a vytváranie modelov strojového učenia a následné nasadenie týchto modelov v reálnej situácii.

Literatúra

Arduino, T. (2022). Arduinoboarduno.
URL: *https://www.arduino.cc/en/main/arduinoBoardUno*

Azure (2019). Azure-iot-protocol-gateway/readme.md at master · azure/azure-iot-protocol-gateway.
URL: *https://github.com/Azure/azure-iot-protocol-gateway/blob/master/README.md*

Barry, R. (2022). Using the freertos real time kernel: Pic32 edition.
URL: *https://aws.amazon.com/freertos/*

Calaway, R. and Grant, J. (2022). What is azure sphere.
URL: *https://docs.microsoft.com/en-us/azure-sphere/product-overview/what-is-azure-sphere*

cloudflare (2022). Cloud.
URL: *https://www.cloudflare.com/en-gb/learning/cloud/what-is-the-cloud*

Discover qualified IOT hardware and devices: Build and deliver successful IOT solutions on AWS (2022).
URL: *https://devices.amazonaws.com/search*

Howell, J. W. (2021). Introduction to azure stream analytics.
URL: *https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction*

Lee, I. and Lee, K. (2015). The internet of things (iot): Applications, investments, and challenges for enterprises, *Business Horizons* **58**(4): 431–440.
URL: *https://www.sciencedirect.com/science/article/pii/S0007681315000373*

Pal, S., Diaz, V. G. and Le, D.-N. (2022a). Iot: Aws iot core for lorawan.
URL: *https://docs.aws.amazon.com/iot/latest/developerguide/connect-iot-lorawan.html*

Pal, S., Diaz, V. G. and Le, D.-N. (2022b). Iot: Mqtt.
URL: *https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html*

Pal, S., Diaz, V. G. and Le, D.-N. (2022c). Iot: What is aws iot?
URL: *https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html*

Philmea (2022). What is microsoft azure rtos?
URL: *https://docs.microsoft.com/en-us/azure/rtos/overview-rtos*

React (2022). React – a javascript library for building user interfaces.
URL: *https://reactjs.org/*

Redhat (2020). What is faas?
URL: *https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas*

Teja, R. (2021). Introduction to esp32: Specifications, esp32 devkit board, layout,.
URL: *https://www.electronicshub.org/getting-started-with-esp32/*

V-Annene (2022). Mt3620 reference development board (rdb) user guide - azure sphere.
URL: *https://docs.microsoft.com/en-us/azure-sphere/hardware/mt3620-user-guide*

Zoznam príloh

Príloha A CD médium

Príloha B Používateľská príručka

Príloha C Systémová príručka

Príloha D Dokumentácia podľa pokynov výskumnej skupiny IKS- https://tukesk.sharepoint.com/sites/Archiv_ZP_IKS_KKUI