UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Systém inteligentního parkování za využití platformy mini PC

Diplomová práce

Autor: Bc. Jan Dian

Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Hradec Králové 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 18. 3. 2019 Bc. Jan Dian

Poděkování:

Děkuji vedoucímu diplomové práce Mgr. Josefu Horálkovi, Ph.D. za veškerou podporu a pomoc při zpracování práce.

Anotace

Tato diplomová práce si klade za cíl navrhnout a vytvořit systém inteligentního parkování s možností sledování aktuální obsazenosti jednotlivých parkovacích míst, automatickou detekcí dostupného parkoviště a možností komunikovat s celým systémem pomocí mobilní aplikace.

Čtenář je postupně seznámen s problematikou detekce vozidla pomocí magnetometrických čidel, principy IQRF technologie využité k jejich propojení do sítě, fungování Firebase databáze pro ukládání informací o parkovišti a služby geofencing pro automatickou notifikaci o dostupnosti parkoviště.

Práce také popisuje samotný návrh řešení, včetně zapojení elektronických obvodů, návrhu mobilní aplikace pro operační systém Android a dalšího programového vybavení, realizovaného v programovacích jazycích JavaScript a Wiring.

Abstract

The objective of the following diploma thesis is the proposition and creation of an intelligent parking system that utilizes currently available technology, including a (parking units) tracking option and an automated parking space detection system that can interact with a mobile phone application.

The reader will be familiarized with the general knowledge and issues associated with vehicle detection using magneto-metrical sensors, IQRF technology principles used for their in-network installation, the Firebase database functioning used for storing information about the car park, and the geofencing services required for the automatic car park availability notifications.

This thesis also specifically describes the components of the proposed solution including the necessary electronic circuitry, a basic mobile phone application for the Android platform, and the additional software necessary for this implementation using the JavaScript and Wiring programming languages.

Obsah

Seznam obrázků VIII

1. Úvod 1

2. Rešerše stávajících řešení a přístupů 2

2.1 Vědecké přístupy 2

2.2 Stávající řešení 3

2.2.1 Výtahové rotační systémy 3

2.2.2 Použití chytrých senzorů v ČR 3

2.2.3 Využití technologie IQRF 3

3. Návrh struktury systému 4

3.1 Funkční diagram navrženého řešení 5

3.2 Schéma komunikace 8

4. Popis jednotlivých částí systému 9

4.1 Magnetometrická čidla 9

4.2 Technologie IQRF 11

4.2.1 Struktura sítě 11

4.2.2 DPA protokol 12

4.2.3 Custom DPA Handler 12

4.2.4 FRC 12

4.2.5 Vytvoření sítě 13

4.3 Platforma Mini PC 17

4.3.1 Instalace OS 17

4.3.2 IQRF Gateway 18

4.4 Programovací prostředí NodeRED 20

4.5 Realtime databáze Firebase 21

4.6 Android Geofencing 23

5. Vlastní řešení 24

5.1 Detekce vozidla 24

5.2 Ovládání IQRF sítě 25

5.3 Android aplikace 30
5.3.1 Automatická detekce parkoviště v dosahu 30
5.3.2 Navigace k parkovišti 31
5.3.3 Informace o obsazenosti parkoviště 32
5.3.4 Vytvoření rezervace parkovacího místa 34
5.3.5 Otevření vjezdové / výjezdové závory 35
5.4 Obsluha parkoviště pomocí NodeRED 36
5.4.1 IQRF_Request 36
5.4.2 Gate 37
5.4.3 Reservations 39
6. Testy 41
7. Ekonomický pohled na navrhované řešení 42
8. Další vývoj aplikace 43
9. Reference 44
10. Přílohy 45
10.1 Mobilní aplikace 45
10.1.1 MainActivity.java 45
10.1.2 InfoActivity.java 55
10.1.3 Constants.java 57
10.1.4 CarInPark.java 57
10.1.5 Reservation.java 58
10.1.6 GeofenceTransitionsIntentService.java 59
10.2 Arduino 62
10.3 NodeRED JavaScript 63
10.3.1 Request 63
10.3.2 GateEntranceOpen 64
10.3.3 GateEntranceClose 64
10.3.4 GateExitOpen 64
10.3.5 GateExitClose 65

10.3.6 JSON 65
10.3.7 IQRFtoJSON 65
10.3.8 GetIO 66
10.3.9 GetNumberOfFreePlaces 67
10.3.10 GlobalContext 67
10.3.11 FirebaseConvert 67
10.3.12 FirebaseConvert - Reservations 68
10.3.13 GlobalContext 68
10.3.14 ResesrvationsConvert 68

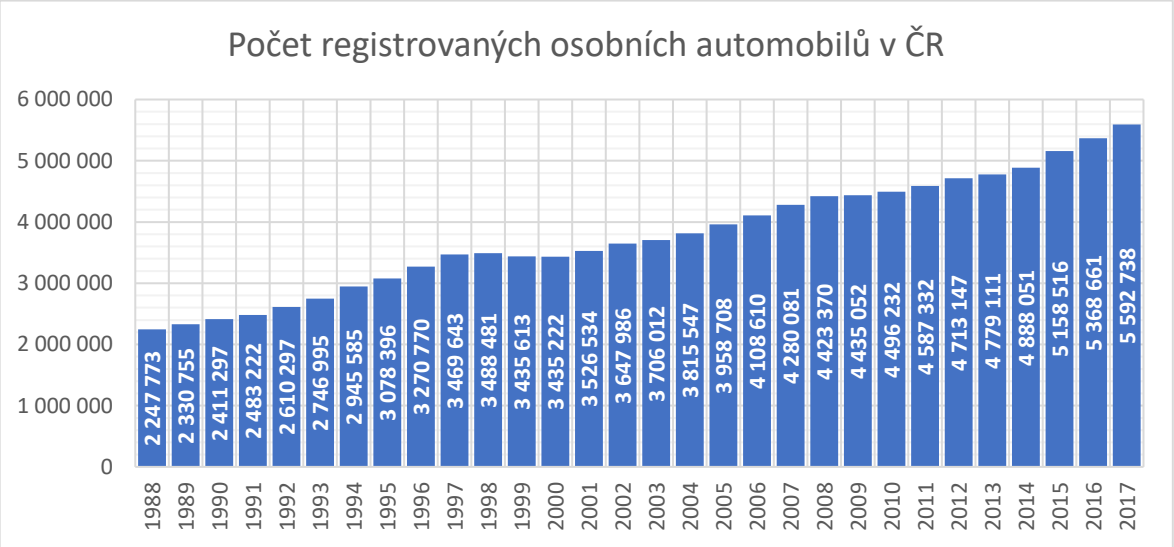
Seznam obrázků

Obrázek 1 - Rostoucí trend počtu registrovaných osobních vozidel v ČR [autor] 1
Obrázek 2 - Návrh struktury systému [autor] 4
Obrázek 3 - Funkční diagram navrženého řešení – zjištění dostupnosti s obsazenosti parkoviště [autor] 5
Obrázek 4 - Funkční diagram navrženého řešení – ovládání vjezdové a výjezdové závory [autor] 6
Obrázek 5 - Funkční diagram navrženého řešení – vytvoření rezervace [autor] 7
Obrázek 6 - Schéma komunikace [autor] 8
Obrázek 7 - Blokové schéma senzoru MPU-9250 [10] 9
Obrázek 8 - Rozdíly naměřených hodnot při absenci vozidla [autor] 10
Obrázek 9 - Rozdíly naměřených hodnot ovlivněných přítomností vozidla [autor] 10
Obrázek 10 - Ukázka komunikace mezi koordinátorem a transceiverem N2 [autor] 11
Obrázek 11 - Struktura DPA paketu [autor] 12
Obrázek 12 - Struktura příkazu FRC (SEND) [autor] 13
Obrázek 13 - Struktura FRC (SEND Selective) příkazu [autor] 13
Obrázek 14 – Založení nového projektu [autor] 14
Obrázek 15 – Vložení souborů do projektu [autor] 14
Obrázek 16 - Ukázka nastavení HWP pro "NOD" transceiver [autor] 15
Obrázek 17 - Zapojení testovacího modulu DK-EVAL-04A [13] 15
Obrázek 18 - Ukázka topologie sítě v IQMESH Network Manageru [autor] 16
Obrázek 19 - Blokové schéma jednodeskového počítače UpBoard [14] 17
Obrázek 20 - Propojení IQRF koordinátoru s UpBoardem [15] 18
Obrázek 21 - Volba SPI Interface [autor] 19
Obrázek 22 - Restart IQRF Daemon service [autor] 19
Obrázek 23 - Ukázka prostředí NodeRED [autor] 20
Obrázek 24 - Firebase - ukázka struktury ukládaných dat [autor] 21
Obrázek 25- Firebase - tabulka událostí [18] 22
Obrázek 26 - Firebase - autorizační metody [18] 22
Obrázek 27 - Detekce událostí pomocí geofencingu [19] 23
Obrázek 28 - Blokové schéma detektoru vozidla [autor] 25
Obrázek 29 - Odeslání požadavku do IQRF sítě [autor] 26
Obrázek 30 - Zpracování požadavku z IQRF sítě [autor] 27
Obrázek 31 - Uložení informací o stavu a počtu jednotlivých parkovacích míst do databáze [autor] 29
Obrázek 32 - Hlavní obrazovka mobilní aplikace [autor] 29
Obrázek 33 - Notifikace po vstoupení do geofence [autor] 31
Obrázek 34 - Zobrazení obsazenosti parkoviště [autor] 33

Obrázek 35 - Ukázka uložení rezervace do databáze [autor] 34
Obrázek 36 - Záznam vozidla přítomného na parkovišti [autor] 35
Obrázek 37 - Programová flow IQRF_Request [autor] 36
Obrázek 38 - Programová flow Gate [autor] 37
Obrázek 39 - Správa rezervací parkoviště [autor] 39

1. Úvod

Počet vozidel v České republice neustále stoupá. Podle informací Ústředního automotoklubu České republiky je v ČR registrováno více jak 5,5 miliónu automobilů [1]. Od roku 1989 se jejich počet zvýšil 2,4x (viz následující graf).



Obrázek 1 - Rostoucí trend počtu registrovaných osobních vozidel v ČR [autor]

S přibývajícím počtem vozidel rostou problémy s parkováním. Především při sportovních, kulturních akcích, ale i před úřady a bankami. Parkování ve městech během dopravní špičky bývá obvykle také obrovský problém. Neznalost počtu volných parkovacích míst může řidiče zavést na obsazené parkoviště, což nutně znamená přesun a hledání volného parkovacího místa na některém z dalších parkovišť. Řidič tak nejen plýtvá svým časem, ale i pohonnými hmotami, zhoršuje se dopravní situace a to má špatný vliv i na životní prostředí.

Cílem práce je navrhnout a implementovat systém inteligentního parkoviště založeného na platformě mini PC, které by řešilo výše popsaný problém. Práce popisuje stávající řešení a vědecké přístupy, na jejich základě jsou stanoveny požadavky na nový systém. V jednotlivých kapitolách práce je popsán princip řešení, architektura systému, výběr vhodného hardware a jeho zapojení, mobilní aplikace určená pro ovládání parkoviště a také způsob komunikace mezi hardwarovým vybavením a mobilní aplikací.

2. Rešerše stávajících řešení a přístupů

Tato kapitola popisuje vědecké přístupy a stávající komerční řešení ve městech v České republice.

2.1 Vědecké přístupy

Článek [2] popisuje využití mobilní aplikace a Internet of Things (IoT) technologie k realizaci parkovacího systému. Je zde popsán proces vyhledání parkoviště, rezervace parkovacího místa, navigace k parkovišti a indoor navigace v rámci vnitřního parkoviště. Dalším tématem je vhodný algoritmus pro výpočet priority doporučení konkrétního parkoviště na základě vzdálenosti řidiče od parkoviště, počtu volných míst a poplatku za parkování, preference jednotlivých položek je nastavitelná uživatelsky. Po výběru parkoviště má řidič možnost rezervovat parkovací místo pomocí mobilní aplikace. K navigaci a výpočtu vzdálenosti od parkoviště jsou využity GPS souřadnice získané mobilní aplikací pomocí Google API. Detekce přítomnosti vozidla na parkovacím místě je realizována pomocí ultrazvukového senzoru a přenášena Wi-Fi modulem k serveru. Popisované řešení je primárně určeno pro vnitřní parkoviště a indoor navigaci zajišťují Bluetooth vysílače iBeacon. Přítomnost řidiče na parkovišti je detekována pomocí RFID čipu.

Článek [3] popisuje architekturu Smart Parking systému založeném na IoT (Internet of Things) technologii. Senzory umístěné na parkovacích místech detekují přítomnost vozidla a tyto informace odesílají do Cloudu pomocí mini PC (Raspberry Pi) umístěném na parkovišti. Autor popisuje využití GPIO pinů Raspberry Pi, ke kterým může být připojeno 26 senzorů, jejich počet může být dále rozšířen pomocí vhodného multiplexoru. Toto řešení využívá IBM MQTT server, kam Raspberry odesílá informace ze senzorů pomocí MQTT zpráv. Uživatel komunikuje se systémem pomocí mobilní aplikace napsané v Apache Cordova, která ke komunikaci se serverem využívá zprávy ve formátu JSON. Uživatel může v mobilní aplikaci sledovat počet volných míst na parkovišti, provést rezervaci parkovacího místa a zaplatit poplatek za parkování.

Problematiku podélného parkování přímo na komunikaci řeší článek [4], který popisuje detekci podélných parkovacích míst pomocí mobilních senzorů umístěných na vozidle, na straně spolujezdce. Senzory jsou umisťovány na vozidla MHD, taxislužby, případně na vozidla dobrovolníků, kteří se často pohybují v inkriminovaných oblastech měření. Systém využívá ultrazvukové čidlo pro detekci zaparkovaných vozidel a volných míst podél komunikace, ve spojení se systémem GPS a použitím Map Matchingu (porovnání detekovaných míst s mapovým podkladem) tak vzniká mapa volných parkovacích míst, která je k uživatelům šířena pomocí mobilní aplikace anebo webové stránky. Podle provedených měření se ukázalo, že pro zmapování stejného počtu parkovacích míst je potřeba méně mobilních senzorů než těch pevných, umístěných přímo na parkovacích místech. Úspěšnost detekce se pohybovala v rozmezí od 76 % do 94 %, v závislosti na přesnosti systému GPS.

2.2 Stávající řešení

2.2.1 Výtahové rotační systémy

Další možností jak řešit nedostatek parkovacích míst jsou výtahové rotační systémy, kterými se v ČR zabývá například firma Smart Parking CZ s.r.o. [5]. Tyto systémy umožňují zaparkovat na dvou parkovacích místech až 16 vozidel, čímž výrazně redukují prostor obsazený zaparkovanými vozidly. Vhodným zakomponováním do okolní výstavby zároveň také nepůsobí rušivým dojmem, jako například velké parkovací domy.

2.2.2 Použití chytrých senzorů v ČR

Výrobou chytrých senzorů pro detekci zaparkovaných vozidel se v ČR zabývá firma CITIQ s.r.o. [6], která využívá bezdrátové magnetické detektory komunikující prostřednictvím sítí IQRF, SIGFOX, nebo LoRA. Detekce zaparkovaného vozidla je realizována pomocí kontinuálního měření geomagnetického pole a dosahuje přesnosti až 97 % (v místech s tramvajovým provozem se přesnost detekce pohybuje okolo 90 %). Senzory jsou napájeny z baterií s garantovanou životností 5 let.

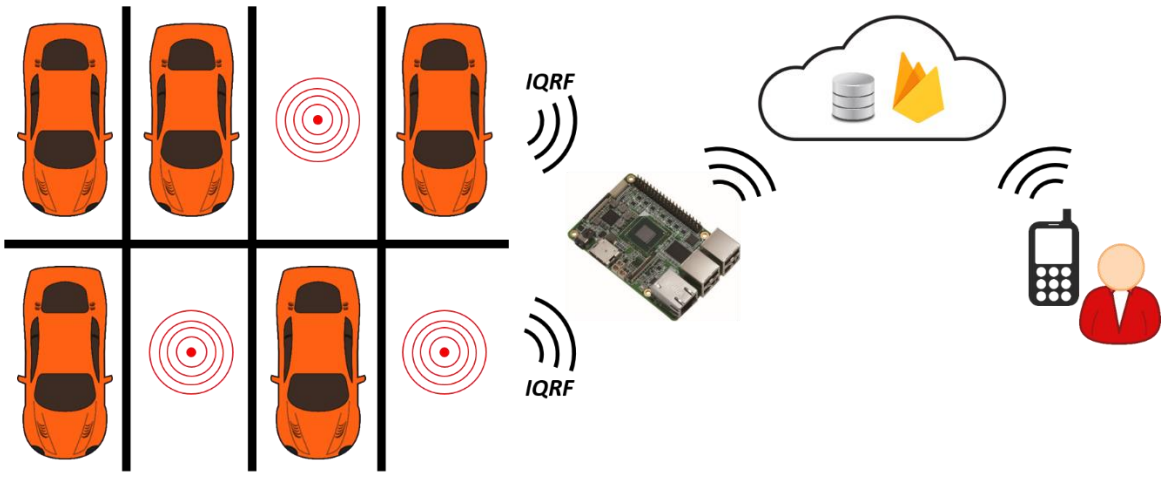
Firma CITIQ s.r.o. spustila v Praze projekt pro detekci obsazenosti parkovacích míst pro tělesně postižené občany, kteří mají díky mobilní aplikaci neustálý přehled o tom, kde můžou zaparkovat. Dalším realizovaným projektem je monitoring 75 parkovacích míst v Brně, které informují řidiče o aktuální obsazenosti a zároveň poskytují magistrátu informaci o platební morálce. Největším projektem firmy je 240 parkovacích senzorů umístěných v šesti částech Liberce.

2.2.3 Využití technologie IQRF

Zde popisované řešení využívá ke komunikaci mezi jednotlivými parkovacími senzory IoT technologii IQRF. Tato technologie je vyvíjena českou firmou MICRORISC s.r.o. [7]. Jak bylo posáno v předchozí kapitole, technologii využívá například firma CITIQ s.r.o. ve svých parkovacích senzorech, ale využití této technologie je mnohem širší a to v celosvětovém měřítku. Konkrétní případy užití lze najít na webových stránkách IQRF aliance [8]. Patří mezi ně například monitoring kvality vody a ovzduší v jihoafrických drůbežárnách, kontrola úrovně CO_2 v brněnských mateřských školách, monitoring dopravy v pražských ulicích, dálkové ovládání lopatek 1,5 MW turbíny v Polsku a mnoho dalších.

3. Návrh struktury systému

Parkovací systém popisovaný v této práci by měl být cenově dostupným systémem pro detekci volných parkovacích míst na menším venkovním školním (pracuji jako učitel elektro oborů na střední průmyslové škole – pozn. autora) parkovišti, založeným na platformě mini PC. Parkoviště využívají učitelé školy a také návštěvy, převážně z řad rodičů. Bohužel se parkoviště nevyhne neukázněným řidičům, kteří ho i přes zákaz také využívají. Tento systém by měl tomuto nešvaru zamezit a dále by měl sloužit jako učební pomůcka. Názorná ukázka praktického využití současných technologií s možností nahlédnutí do zdrojových kódů a komunikačních protokolů. Vzhledem k cenové dostupnosti jsou nevhodná řešení pomocí rotačních výtahových systémů popisovaných v bodě 2.2 této práce. Stejně tak detekce zaparkovaných vozidel pomocí ultrazvukových, nebo infra čidel ve venkovních prostorech není vhodná, proto zde popisované řešení využívá magnetometrická čidla, která jsou propojena do MESH sítě. Mini PC v pravidelných intervalech zjišťuje obsazenost jednotlivých pakovacích míst a tyto informace ukládá do real-time databáze. Řidič je informován prostřednictvím mobilní aplikace pro platformu Android, díky které získá informaci o aktuálním počtu volných míst na parkovišti, a to buď manuálně nebo automaticky, s využitím služby Geofencing.

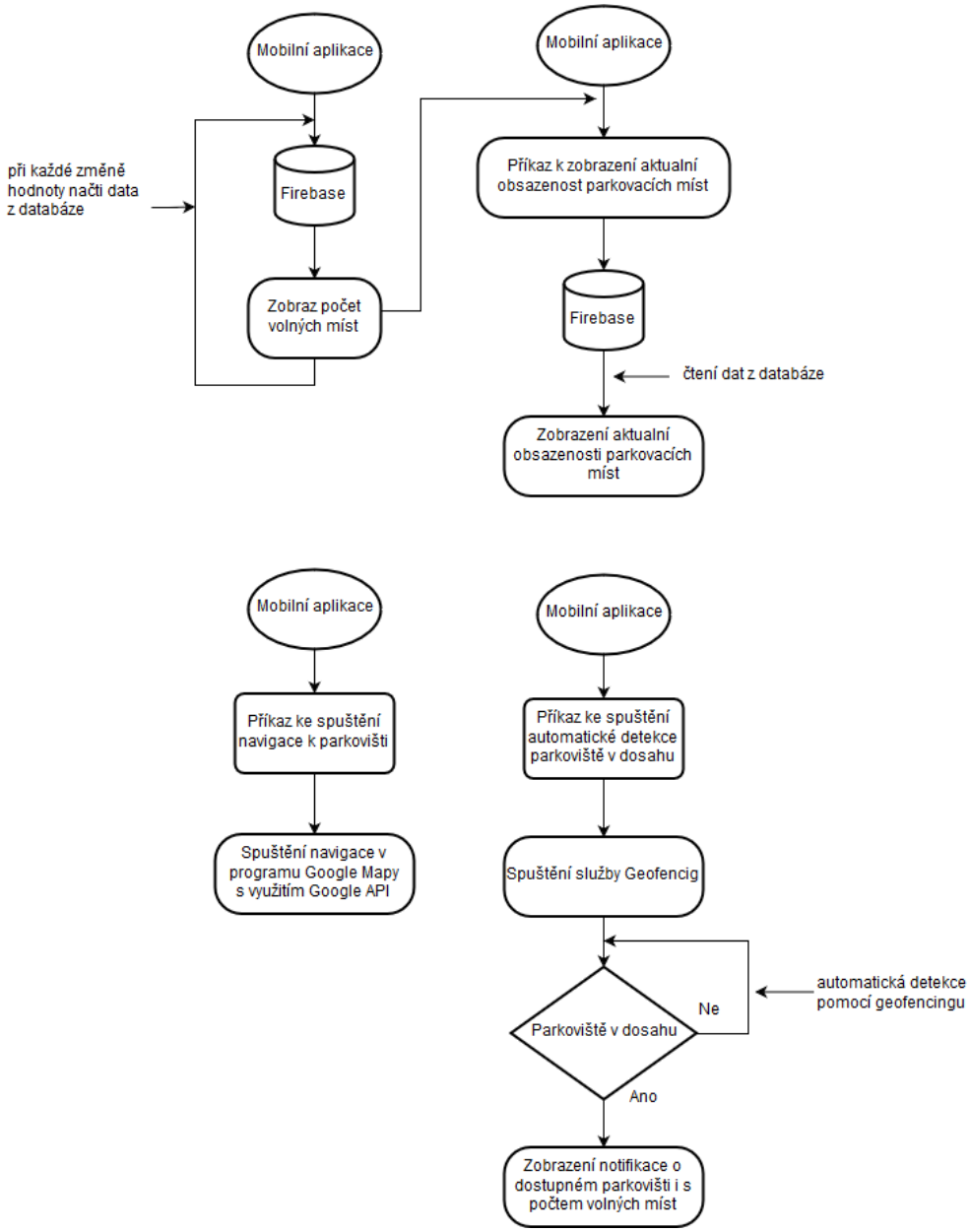


Obrázek 2 - Návrh struktury systému [autor]

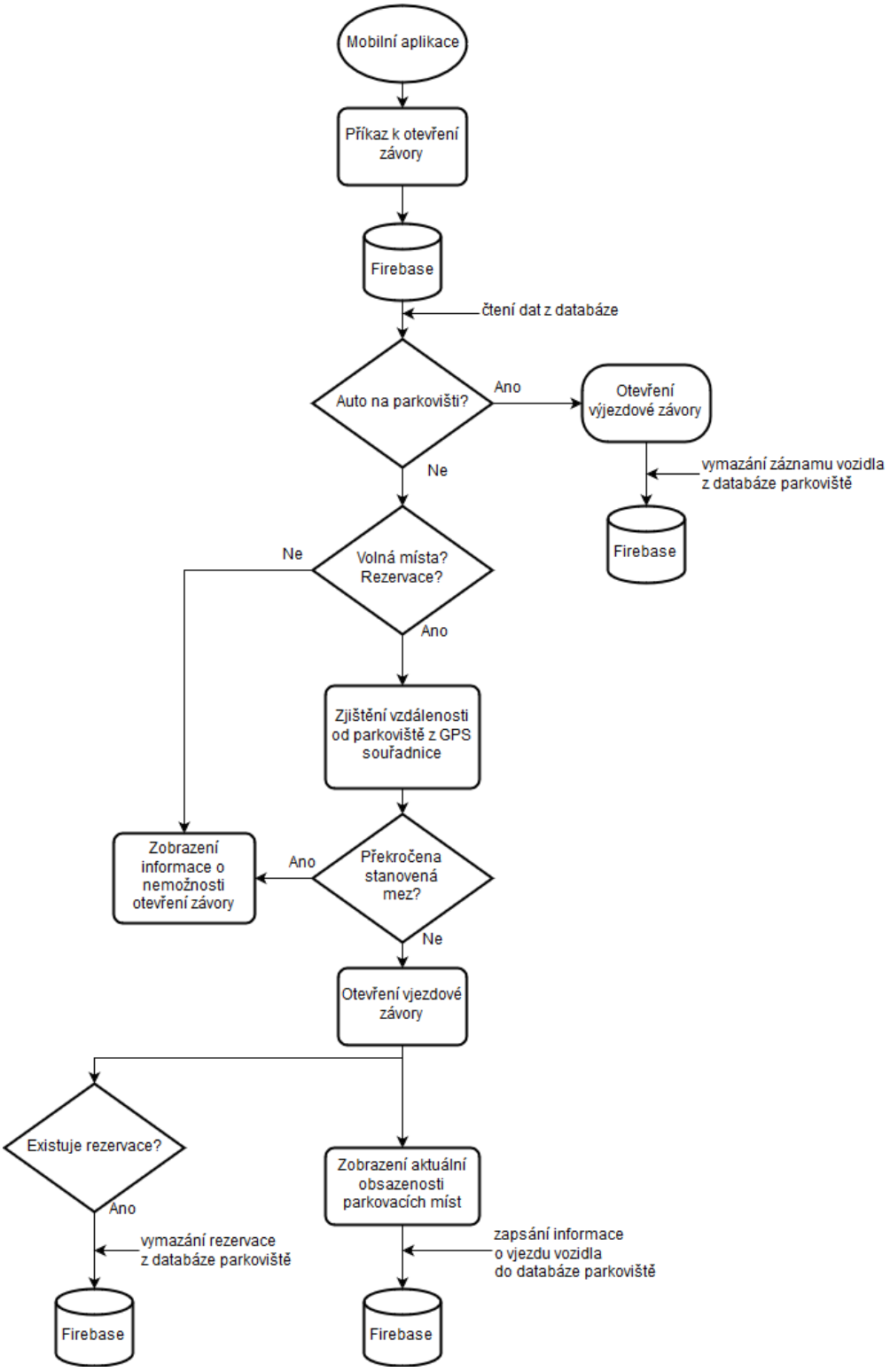
Následně má možnost si jedno parkovací místo na 60 min zarezervovat. Pokud během této doby na parkoviště nedorazí, rezervace je automaticky vymazána. Popisované přístupy v bodě 2.1 využívají pro identifikaci řidiče RFID čipy. Zde popisované řešení umožňuje otevření závory parkoviště přímo z mobilní aplikace pomocí zápisu do real-time databáze. Na základě tohoto zápisu mini PC provede příslušné kroky. Pro kontrolu přítomnosti řidiče (mobilního telefonu) na parkovišti zkontroluje před zápisem mobilní aplikace na základě GPS souřadnice vzdálenost telefonu od vjezdu do parkoviště. Pokud tato vzdálenost překročí stanovenou mez, přístup není povolen.

Díky tomuto řešení odpadá nutnost použití RFID čipu a pro využití parkoviště postačí pouze mobilní telefon s nainstalovanou aplikací připojený k internetu. Identifikace telefonu probíhá na základě vygenerování jedinečného ANDROID ID (64 bitové číslo).

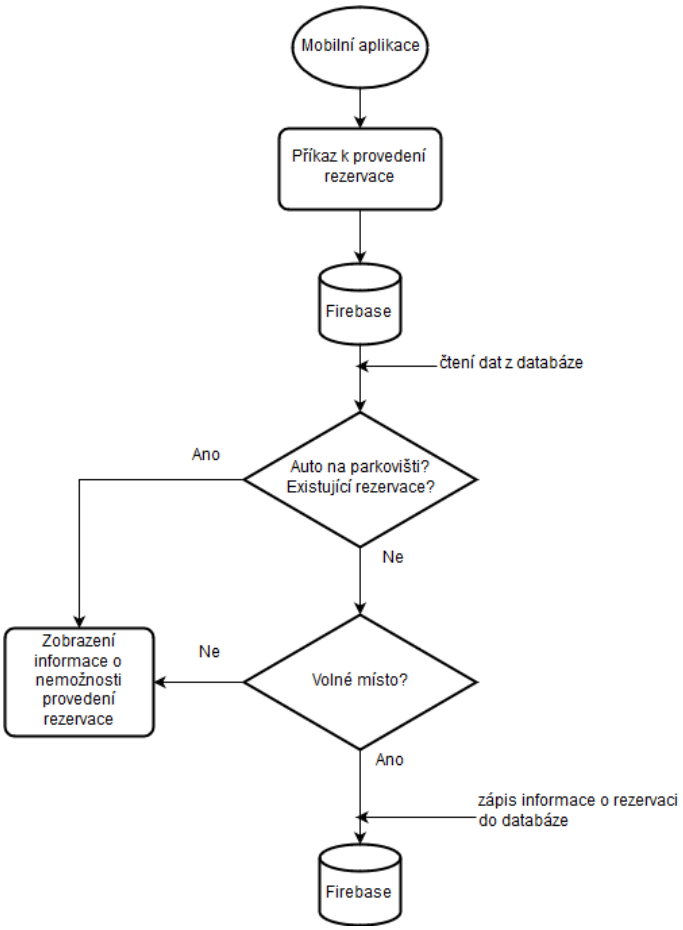
3.1 Funkční diagram navrženého řešení



Obrázek 3 - Funkční diagram navrženého řešení – zjištění dostupnosti a obsazenosti parkoviště [autor]



Obrázek 4 - Funkční diagram navrženého řešení – ovládání vjezdové a výjezdové závory [autor]



Obrázek 5 - Funkční diagram navrženého řešení – vytvoření rezervace [autor]

Předcházející obrázky popisují jednotlivé funkce navrženého řešení. Obrázek (Obrázek 3) popisuje tři základní funkce:

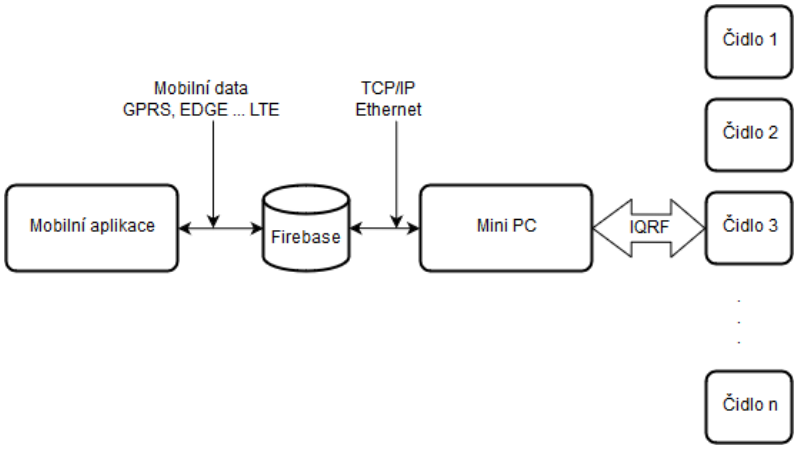
- ✓ Zobrazení počtu volných míst na parkovišti a jejich aktuální obsazenosti,
- ✓ spuštění navigace k parkovišti,
- ✓ spuštění automatické notifikace o dostupném parkovišti.

Mobilní aplikace sleduje položku databáze s počtem volných míst a při každé změně aktualizuje stav notifikační ikony na hlavní obrazovce aplikace. Po kliknutí na tuto ikonu dojde k načtení aktuální obsazenosti jednotlivých parkovacích míst z databáze a zobrazení této informace na displeji mobilního telefonu. Kliknutím na příslušnou ikonu na hlavní obrazovce dojde ke spuštění navigace k parkovišti. Aplikace s využitím Google API spustí navigaci v aplikaci Mapy Google. Podrobnější informace jsou dostupné v kapitolách 5.3.2 a 5.3.3. Pro spuštění automatické notifikace o dostupném parkovišti a počtu volných míst využívá aplikace službu Geofencing, která je popsána v kapitolách 4.6 a 5.3.1.

Další obrázek (Obrázek 4) popisuje kroky potřebné k otevření závory parkoviště. Aplikace nejprve kontroluje databázi zaparkovaných vozidel, aby zjistila, zda automobil na parkoviště vjíždí, nebo ho opouští. Při vjezdu na parkoviště je dále kontrolováno, zda na něm jsou volná místa, případně jestli řidič žádající otevření závory nemá v databázi uloženou rezervaci. Pokud je vše v pořádku, následuje kontrola vzdálenosti od parkoviště, aby se zamezilo nechtěnému, nebo úmyslnému otevření závory z větší vzdálenosti od parkoviště. Pokud není některá z podmínek splněna, obdrží řidič informaci nedostupnosti parkoviště. V opačném případě dojde k otevření vjezdové závory a následně je proveden zápis o vjíždějícím vozidle do databáze parkoviště. Současně dochází také ke kontrole platné rezervace řidiče, která je případně z databáze vymazána. Při výjezdu z parkoviště systém otevře výjezdovou závoru a odstraní záznam vyjíždějícího vozidla z databáze parkoviště. Podrobnější informace obsahuje kapitola 5.3.5. Následující obrázek (Obrázek 5) popisuje vytvoření rezervace. Při pokusu o vytvoření rezervace systém nejprve zkontroluje, jestli se vozidlo nenachází na parkovišti nebo jestli už nebyla pro toto ID rezervace vytvořena, následuje kontrola volných míst. Pokud jsou všechny podmínky splněny, je proveden zápis rezervace do databáze parkoviště. V opačném případě je řidič na displeji mobilního telefonu informován o nemožnosti vytvoření rezervace. Délka trvání rezervace je omezena na 60 minut. Z tohoto důvodu mini PC umístěné na parkovišti v minutových intervalech prochází databázi a kontroluje překročení tohoto časového limitu. Pokud k překročení dojde, je záznam z databáze odstraněn. Další informace obsahují kapitoly 5.3.4 a 5.4.3.

3.2 Schéma komunikace

Jak je vidět na následujícím obrázku (Obrázek 6) společným úložištěm dat pro celý systém je Firebase databáze. Komunikace mezi databází a mini PC je zajištěna pomocí TCP/IP protokolu, který je detailně popsán v knize Kabelové a Dostálka [9]. Komunikace mezi databází a mobilní aplikací využívá datové přenosy v mobilních sítích, které popisuje ve svém článku Peterka [8]. Komunikace mezi mini PC a jednotlivými senzory je zajištěna pomocí technologie IQRF a je popsána v kapitolách 4.2, 4.3.2 a 5.2.

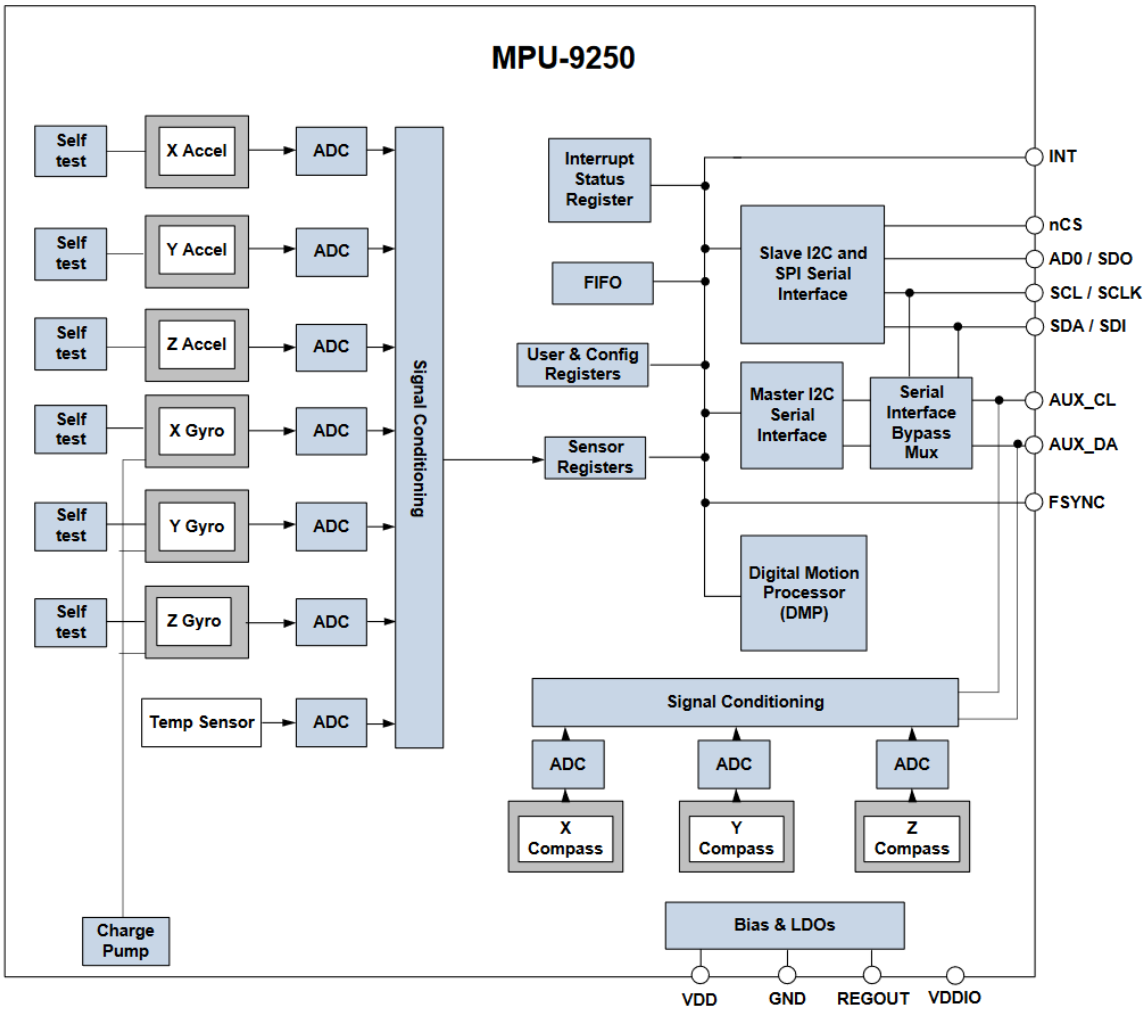


Obrázek 6 - Schéma komunikace [autor]

4. Popis jednotlivých částí systému

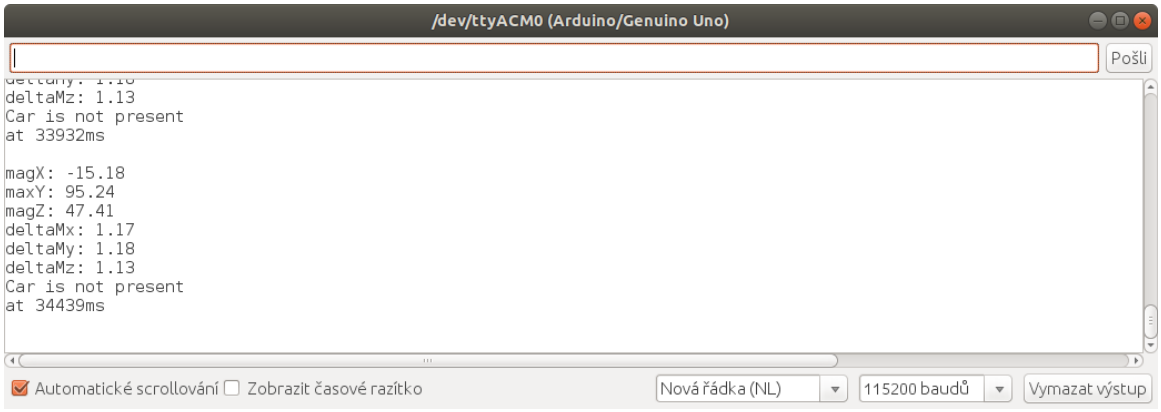
4.1 Magnetometrická čidla

K detekci přítomnosti vozidla na parkovacím místě využívají řešení popisovaná ve druhé kapitole této práce infračervená nebo ultrazvuková čidla přiblížení. Ta jsou vhodná pro vnitřní parkoviště, jelikož povětrnostní vlivy jako sníh, bláto, listí apod. by znemožnily správnou detekci vozidla. V některých případech je čidlo umisťováno nad vozidlo, aby se minimalizovala možnost zakrytí čidla a nesprávné detekce. Toto řešení je opět vhodnější pro vnitřní parkoviště, kde se tato čidla umisťují na strop nebo na speciální konstrukci zavěšenou od stropu. Z výše popisovaných důvodů řešení popisované touto prací využívá k detekci vozidla na parkovacím místě magnetometrické čidlo MPU-9250 [10], které detekuje geomagnetické pole země ve třech osách pomocí magnetického senzoru založeném na Hallově efektu. Měřící rozsah čidla je ±4800μT. Čidlo dále funguje jako akcelerometr a gyroskop, jak ukazuje blokové schéma na následujícím obrázku (Obrázek 7). Tyto funkce však nejsou využity.



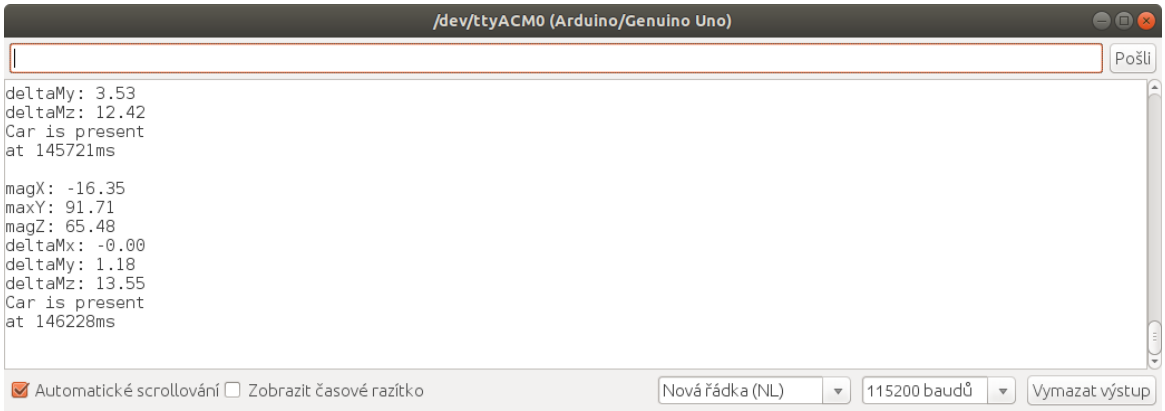
Obrázek 7 - Blokové schéma senzoru MPU-9250 [10]

Čidlo funguje jako kompas ve třech osách a hodnoty každé z nich lze získat jako šestnáctibitové číslo po komunikační sběrnici SPI nebo I2C. Propojením, například s platformou Arduino, lze s využitím vhodné knihovny tyto hodnoty posílat například po sériovém portu a následně je zobrazit v terminálu, jak ukazuje obrázek (Obrázek 8).



Obrázek 8 - Rozdíly naměřených hodnot při absenci vozidla [autor]

Po připojení napájení, čidlo detekuje geomagnetické pole země v jednotlivých osách, pokud je umístěno na stále stejném místě a je znemožněno jeho pohybu, zobrazované hodnoty se téměř nemění. V okamžiku, kdy nad toto čidlo najede automobil, který obsahuje spoustu kovových částí, dojde k ovlivnění měřených hodnot a rozdíl oproti „klidovým“ hodnotám se výrazně změní (Obrázek 9).



Obrázek 9 - Rozdíly naměřených hodnot ovlivněných přítomností vozidla [autor]

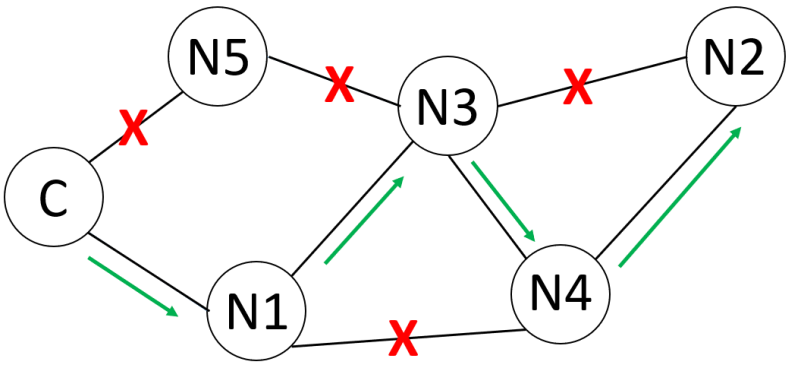
Tento princip je využit k detekci vozidla. Informace je reprezentována binárními hodnotami a předána modulu IQRF.

4.2 Technologie IQRF

4.2.1 Struktura sítě

Samotná parkovací místa musí být schopná pomocí výše popsaného senzoru detekovat, zda je místo obsazené, či nikoliv, ale také by měla být zapojena do vhodné sítě tak, aby byla schopna komunikovat s řídícím mini počítačem. K tomuto účelu je v řešení, popisovaném v této práci, využita technologie IQRF [11].

IQRF je platforma určená pro bezdrátový přenos dat. Komunikace probíhá na frekvencích 868 MHz a 433 MHz. Platforma umožňuje snadnou implementaci do nejrůznějších druhů systémů. IQRF využívá pro bezdrátovou komunikaci speciální transceivery, které si navzájem předávají data. Transceivery IQRF se vyznačují velice nízkou spotřebou (12,3 mA při komunikaci a 380 nA ve sleep módu) a díky podporované topologii MESH lze dosáhnout velkého komunikačního dosahu. Tyto parametry jsou ideální volbou pro IoT technologie. Bezdrátové sítě IQRF využívají protokol IQMESH. Ten využívá toho, že jsou v prostoru rozmístěny transceivery IQRF tak, aby vždy byly minimálně dva vzájemně ve vysílacím dosahu. Maximální vzdálenost mezi dvěma komunikujícími transceivery je přibližně 500 m ve volném prostoru. Transceivery vysílají synchronizovaně, a díky tomu se při vysílání navzájem neruší. Komunikaci řídí tzv. koordinátor, který vyšle data, transceivery v komunikačním dosahu data přijmou, v okamžiku svého time-slotu vyšlou dále a takto se data postupně šíří celou sítí. Díky tomuto principu je celá síť velice spolehlivá a úspěšnost doručení dat je vysoká. Díky duplikovaným cestám mezi jednotlivými transceivery data dorazí do cíle i při přerušení několika komunikačních cest současně (Obrázek 10).



Obrázek 10 - Ukázka komunikace mezi koordinátorem a transceiverem N2 [autor]

Koordinátor C posílá data pro transceiver N2. Přerušené komunikační cesty, například vlivem rušení, představují červené křížky. Informace dorazí do cíle pomocí transceiverů N1, N3, N4, jak naznačují zelené šipky.

4.2.2 DPA protokol

Každý transceiver obsahuje hardwarový profil (HWP). Nahrání HWP do transceiverů umožňuje jejich řízení zprávami na základě DPA (Direct Peripheral Access) protokolu [12]. Díky tomuto protokolu je možné vybudovat síť složenou z 240 transceiverů (včetně koordinátoru), dále je možné transceivery řídit odesláním dat v přesně stanoveném formátu (Obrázek 11).

Adresa zařízení	ID periferie	ID příkazu	ID skupiny	Data
NADR (Node Address)	PNUM (Pheripheral Number)	PCMD (Pheripheral Command	HWPID (Hardware Profile ID)	PDATA Pheripheral Data)
[2B]	[1B]	[1B]	[2B]	[0-56B]

Obrázek 11 - Struktura DPA paketu [autor]

- **NADR** – adresa nodu – 0x00 pro koordinátor, 0x01 – 0xEF pro ostatní transceivery,
- **PNUM** – číslo periferie – 0x03 EEPROM, 0x08 SPI, 0x0C UART apod.,
- **PCMD** – číslo příkazu – záleží na použité periferii,
- **HWPID** – číslo HW profilu jednoznačně určuje funkcionalitu periferního zařízení, pokud je uvedeno číslo 0xFFFF tak se příkaz provede na jakémkoliv HW profilu,
- **PDATA** – pole 56 Byte pro doplňující data příkazu – toto pole je nepovinné.

Vzhledem k obsáhlosti protokolu DPA jsou zde popsány jen základní informace nutné pro pochopení struktury paketu. Kompletní popis protokolu je dostupný v dokumentaci [12], případně online na [www.iqrf.org/DpaTechGuide].

4.2.3 Custom DPA Handler

Custom DPA Handler je kód zapsaný v jazyce C, který transceiveru vytváří vlastní logiku. Pomocí tohoto kódu lze definovat vlastní uživatelskou periferii a nastavit její chování při přijetí DPA příkazu s ID této periferie a ID určitého příkazu. Pomocí DPA handleru lze také rozšířit množinu výše popisovaných HW skupin a umožnit tak filtrování řídících zpráv pro skupiny periferií určitého typu. Konkrétní využití custom DPA handleru včetně zdrojového kódu je popsáno v kapitole 5.2.

4.2.4 FRC

FRC (Fast Response Command) je speciální DPA periferie koordinátoru, která mu umožňuje odeslat do sítě příkaz, který zpracují všechny transceivery v síti. V okamžiku, kdy transceiver zpracuje příkaz, uloží odpověď na specifickém místě ve zprávě a ta pak putuje přes celou síť společně s daty a sbírá odpovědi jednotlivých transceiverů. V případě že potřebujeme ze všech transceiverů získat stejnou informaci (např. stav magnetometrického čidla detekujícího přítomnost vozidla na parkovacím místě) je použití FRC příkazu velice výhodné. Není potřeba obesílat každý transceiver zvlášť, ale stačí odeslat pouze jeden příkaz.

To má pozitivní vliv na množství přenášených dat, ale i na čas, který je potřebný k získání odpovědi. FRC příkaz odesíláme pomocí DPA protokolu, jak ukazuje následující obrázek (Obrázek 12).

Adresa zařízení	ID periferie	ID příkazu	ID skupiny	Data	
0x00	0x0D	0x00	0xFFFF	ID	User data
Koordinátor	FRC	SEND	Všechny skupiny	ID příkazu	Data pro zpracování FRC příkazu

Obrázek 12 - Struktura příkazu FRC (SEND) [autor]

Položka User data není vyžadována všemi FRC příkazy a pokud není použita musí být nahrazena dvoubajtovou hodnotou 00.00. Příkaz SEND může být nahrazen příkazem SEND Selective (0x02), který umožňuje odeslat příkaz jen na vybrané transceivery, které jsou definovány v poli data mezi položkami ID a User data (Obrázek 13).

Adresa zařízení	ID periferie	ID příkazu	ID skupiny	Data		
0x00	0x0D	0x02	0xFFFF	ID	Vybrané transceivery	User data
Koordinátor	FRC	SEND Selective	Všechny skupiny	ID příkazu	Vybrané transceivery pro provedení FRC příkazu	Data pro zpracování FRC příkazu

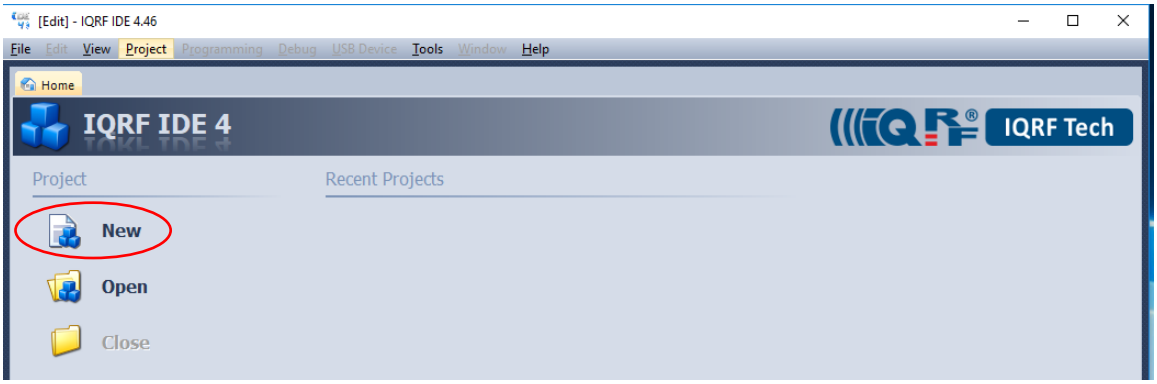
Obrázek 13 - Struktura FRC (SEND Selective) příkazu [autor]

Vybrané transceivery lze specifikovat pomocí binární informace o velikosti 30 bajtů. V IQRF síti může být maximálně 240 transceiverů a to přesně odpovídá třiceti bajtům po osmi bitech. Pokud má příslušný bit hodnotu 1, bude příkaz na příslušný transceiver odeslán, v opačném případě (hodnota 0) nebude transceiver do zpracování příkazu zařazen.

4.2.5 Vytvoření sítě

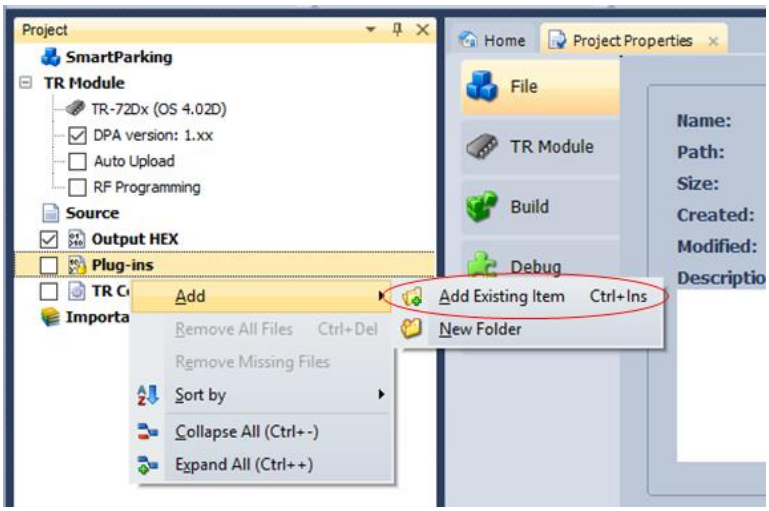
K prvotnímu nakonfigurování transceiverů a vytvoření IQRF sítě je nutné použít programátor CK – USB – 04K dodávaný jako příslušenství k transceiverům IQRF a také vývojové prostředí IQRF IDE, které je volně ke stažení na stránkách výrobce [11]. Dále je potřeba ze stránek výrobce stáhnout hardwarové profily pro koordinátor i jednotlivé nody. Nejvýhodnější je stažení startUp balíčku, který obsahuje všechny potřebné soubory pro konfiguraci a sestavení sítě.

Založení nového projektu se po instalaci a spuštění vývojového prostředí IQRF IDE provede kliknutím na volbu New Project na úvodní obrazovce (Obrázek 14).



Obrázek 14 – Založení nového projektu [autor]

Dále je do projektu nutné přidat HW profily a DPA Custom Handler pro FRC příkaz na zjištění stavu jednotlivých čidel na parkovacích místech. HW profily patří do sekce Plug-ins a DPA Custom Handler do sekce Source. Přidání souboru do příslušné sekce se provede kliknutím pravého tlačítka na příslušnou sekci a zvolením „Add => Add Existing Item“ (Obrázek 15).



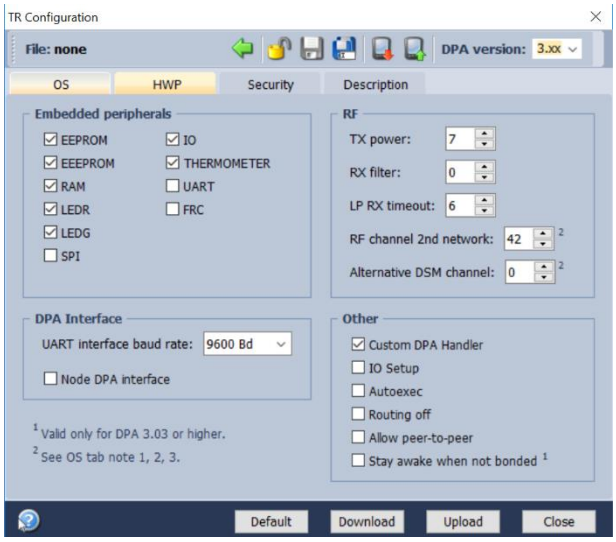
Obrázek 15 – Vložení souborů do projektu [autor]

Vzhledem k tomu, že vložený DPA Handler je ve formátu zdrojového souboru v jazyce C, je potřeba tento soubor zkompilovat do *.hex formátu, aby bylo možné tento soubor nahrát do transceiveru. To lze provést označením příslušného souboru a stiskem klávesy F10 nebo kliknutím na tlačítko build. Po vložení prvního transceiveru do programátoru a následném připojení do USB portu počítače lze v okně „Project“, dvojitým kliknutím na sekci „TR Configuration“, zobrazit konfigurační možnosti pro vložený transceiver. Pro všechny komunikující transceivery je nutné na záložce „OS“ nastavit stejný komunikační kanál (standardně 52), dále na záložce HWP pro koordinátor zvolit možnost zpracování FRC příkazu (Obrázek 16) a pro každý nod zvolit využití Custom DPA Handleru. Na kartě „Security“ lze nastavit heslo a šifrování komunikace. Po úpravě konfigurace je nutné úpravy uložit!

Upload všech označených souborů

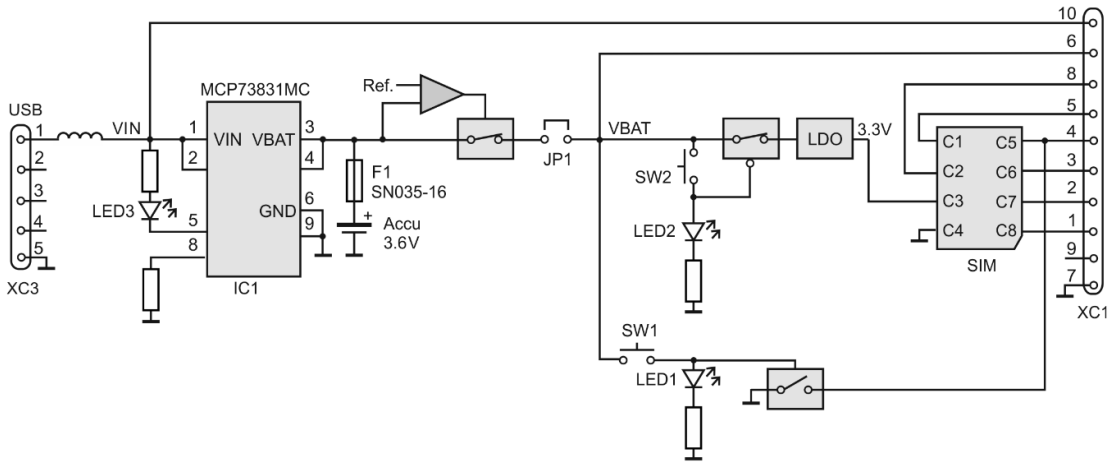
- NOD – HWP, TR Configuration, překompilovaný DPA Handler ve formátu *.hex
- KOORDINÁTOR – HWP, TR Configuration

je proveden stisknutím klávesy F5 nebo kliknutím na tlačítko Upload.



Obrázek 16 - Ukázka nastavení HWP pro "NOD" transceiver [autor]

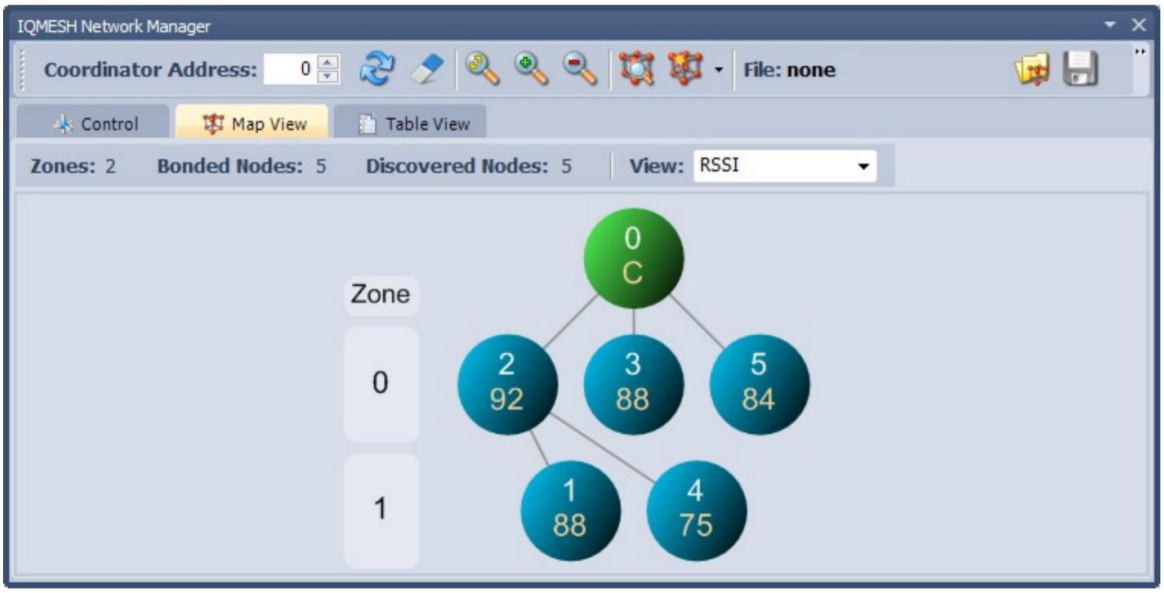
Po naprogramování všech transceiverů a jejich vložení do testovacích modulů DK-EVAL-04A, případně zapojením do vlastních modulů, které musí být uzpůsobeny podle schématu zapojení transceiveru [13] (Obrázek 17), aby byl transceiver napájen příslušným napětím a měl dostupná tlačítka pro RESET a párování (bondování), je nutné připojit koordinátor do programátoru CK – USB – 04K. Následně lze provést vytvoření vlastní IQRF sítě.



Obrázek 17 - Zapojení testovacího modulu DK-EVAL-04A [13]

Pokud po připojení napájení na testovacích modulech bliká červená LED, znamená to, že nemají v paměti uložené žádné předchozí párování (bondování).

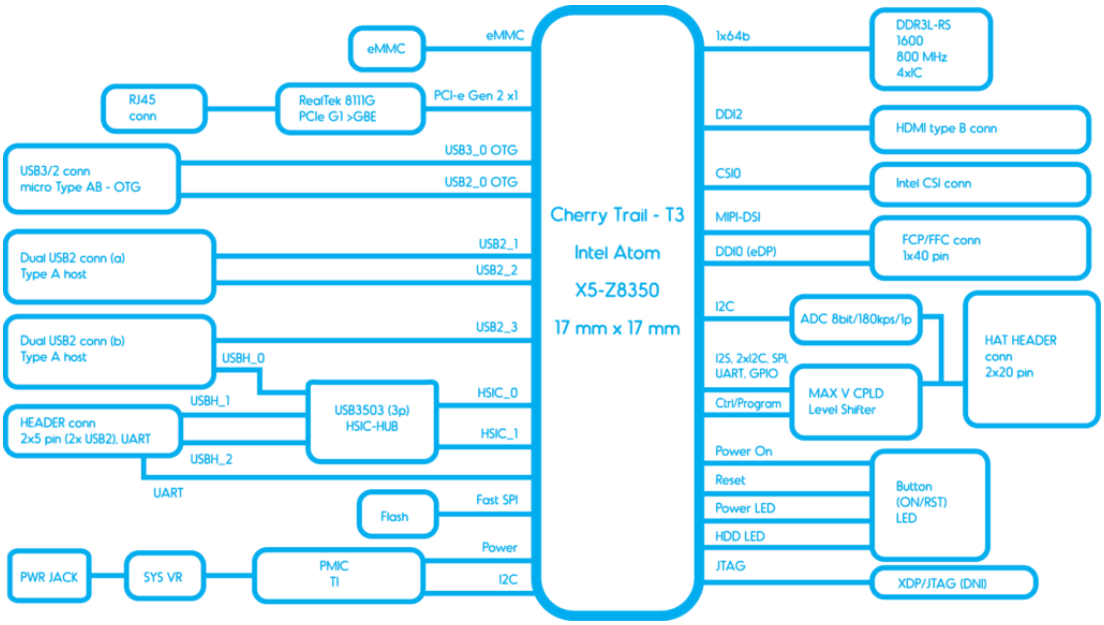
V opačném případě je nutné nody ručně odbondovat stisknutím resetovacího a uživatelského tlačítka na testovacím modulu, následným uvolněním resetovacího tlačítka, po probliknutí zelené LED okamžitým uvolněním uživatelského tlačítka. Pro vymazání informací o párování z koordinátoru je nutno použít IQMESH Network Manager, který je součástí programu IQRF IDE, použitím tlačítka „Clear All Bonds“. Následně stisknutím tlačítka „Bond“ začne koordinátor vyhledávat nový nod a párování je nutno do deseti vteřin potvrdit uživatelským tlačítkem na příslušném testovacím modulu. Opakováním této procedury dojde ke spárování všech nodů s koordinátorem. Všechny nody musí být v době párování v komunikačním dosahu koordinátoru. Po dokončení párování a rozmístění jednotlivých nodů na jejich finální pozice dojde kliknutím na tlačítko “Discovery“ v IQMESH Network Manageru k vytvoření topologie IQMESH sítě, jejíž podobu lze prohlédnout na záložce „MapView“ (Obrázek 18).



Obrázek 18 - Ukázka topologie sítě v IQMESH Network Manageru [autor]

4.3 Platforma Mini PC

Vytvořená IQRF síť potřebuje bránu (gateway) pro odesílání dat do databáze/cloudu pomocí sítě Internet. Z důvodu co největší kompatibility a technické podpory, řešení navrhované v této práci využívá jednodeskový počítač UpBoard [14], dodávaný jako příslušenství k IQRF transceiverům. Jehož blokové schéma je na následujícím obrázku (Obrázek 19). Oproti konkurenčnímu Raspberry Pi nabízí výkonnější mikroprocesor Intel Atom, vyšší kapacitu paměti a hlavně odpadá nutnost instalovat operační systém na externí paměťovou kartu, která ve spojení s mechanickým konektorem může představovat potencionální problémy.



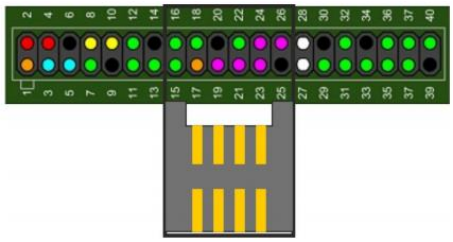
Obrázek 19 - Blokové schéma jednodeskového počítače UpBoard [14]

4.3.1 Instalace OS

Vybraný mini počítač je nutné opatřit operačním systémem. S přihlédnutím na dostupnost technické podpory je pro aplikaci popisovanou v této práci vybrán operační systém Linux (resp. jeho verze Ubilinux) [11], jehož instalační image je dostupný na stránce [https://emutex.com/products/ubilinux]. Pro vytvoření instalačního flash disku lze použít např. program Etcher [https://www.balena.io/etcher/]. Pro instalaci OS je potřeba k UpBoardu připojit klávesnici, myš, monitor a do volného USB portu vložit instalační flash disk. Po připojení napájení pokračovat podle pokynů na obrazovce.

4.3.2 IQRF Gateway

Mini počítač slouží k ovládání sítě IQRF a také k propojení celé aplikace prostřednictvím sítě Internet s databází. Z tohoto důvodu je nutné propojit IQRF koordinátor pomocí dodávané redukce s GPIO piny UpBoardu (Obrázek 20). Dále je nutné doinstalovat příslušné utility pro komunikaci s IQRF sítí a databází [15]. Instalace může probíhat lokálně nebo vzdáleně, například pomocí programu PuTTY [https://www.putty.org/].



Obrázek 20 - Propojení IQRF koordinátoru s UpBoardem [15]

Nejprve je nutné provést update systému Ubilinux,

```
sudo apt-get update && sudo apt-get -y full-upgrade
```

dále nainstalovat MQTT Broker pro vzájemné zasílání zpráv mezi UpBoardem a IQRF sítí,

```
sudo apt-get install -y mosquitto mosquitto-clients
```

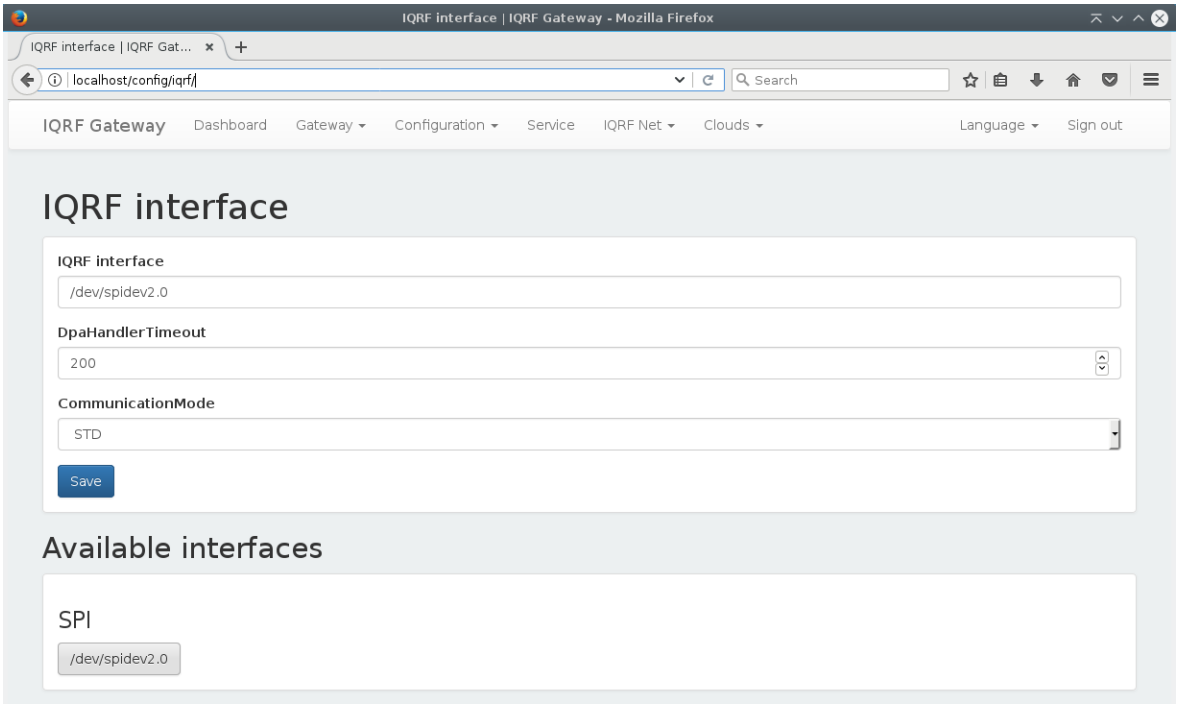
IQRF Gateway Daemon,

```
sudo apt-get install -y dirmngr

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
9C076FCC7AB8F2E43C2AB0E73241B9B7B4BD8F8E

echo "deb http://repos.iqrfsdk.org/debian stretch stable" | sudo tee -a
/etc/apt/sources.list.d/iqrf-daemon.list

sudo apt-get update && sudo apt-get install -y iqrf-daemon
```

webovou aplikaci pro IQRF Daemon,

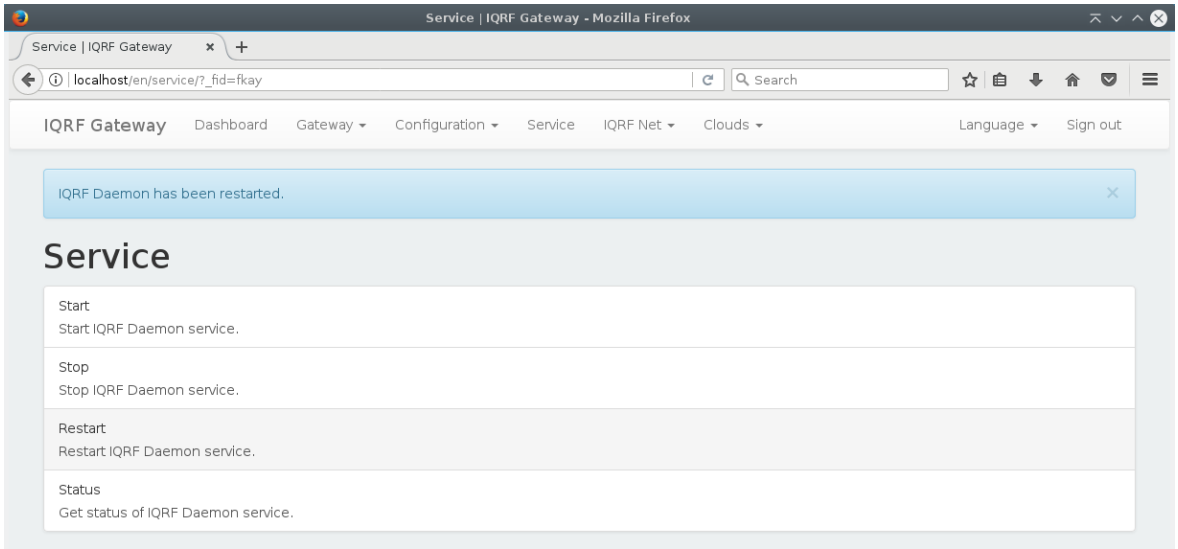
```
cd /home/ubilinux

git clone https://github.com/iqrfsdk/iqrf-daemon-webapp.git

cd iqrf-daemon-webapp/install/

sudo python3 install.p -d debian -v 9
```

ve webovém prohlížeči otevřít adresu [http://localhost/en/config/iqrf] (pro vzdálený přístup je nutné „localhost“ nahradit IP adresou přidělenou UpBoardu) a zvolit dostupný interface (Obrázek 21). Dále na adrese [http://localhost/en/service] restartovat službu IQRF Daemon (Obrázek 22).



Obrázek 21 - Volba SPI interface [autor]

Po zvolení SPI interface je důležité konfiguraci uložit kliknutím na tlačítko „Save“!



Obrázek 22 - Restart IQRF Daemon service [autor]

Následuje instalace Node.js,

```
cd /home/ubilinux

git clone https://github.com/iqrfsdk/iot-starter-kit.git

cd iot-starter-kit/install
```

```
sudo cp etc/lsb-release-debian /etc/lsb-release

sudo apt-get install curl

curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -

sudo apt-get install nodejs

sudo cp etc/lsb-release-ubilinux /etc/lsb-release
```

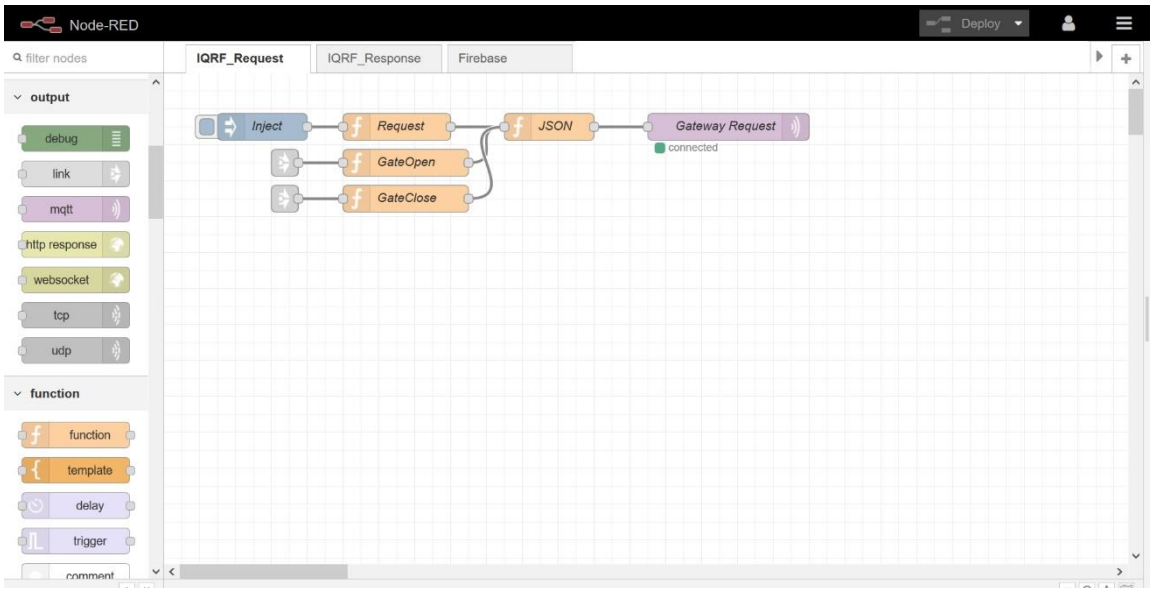
a na závěr proběhne instalace programovacího prostředí NodeRED, které bude sloužit k programování obsluhy IQRF sítě a přístupu do databáze.

```
sudo npm install -g --unsafe-perm NodeRED

sudo npm install -g pm2

cd /home/ubilinux

pm2 start /usr/bin/NodeRED

pm2 save

pm2 startup
```

Programovací prostředí je dostupné na adrese [http://localhost:1880].

4.4 Programovací prostředí NodeRED

NodeRED [16] je online webový programovací nástroj, který funguje na bázi programových flow. Programování probíhá propojováním jednotlivých nodů z palety. Ty poté tvoří celou programovou flow. Paleta obsahuje nody pro MQTT komunikaci, email, úložiště, webové služby, sociální sítě, databáze atd.



Obrázek 23 - Ukázka prostředí NodeRED [autor]

Další nody lze využít pro konverzi mezi datovými formáty, jako například XML, JSON apod. Paleta obsahuje i nody pro tvorbu vlastních funkcí, které lze psát v programovacím jazyce JavaScript. Na obrázku (Obrázek 23) je vidět na levé straně paleta s nody a na hlavní ploše ukázka programové flow pro komunikaci s IQRF Gateway pomocí MQTT zpráv.

4.5 Realtime databáze Firebase

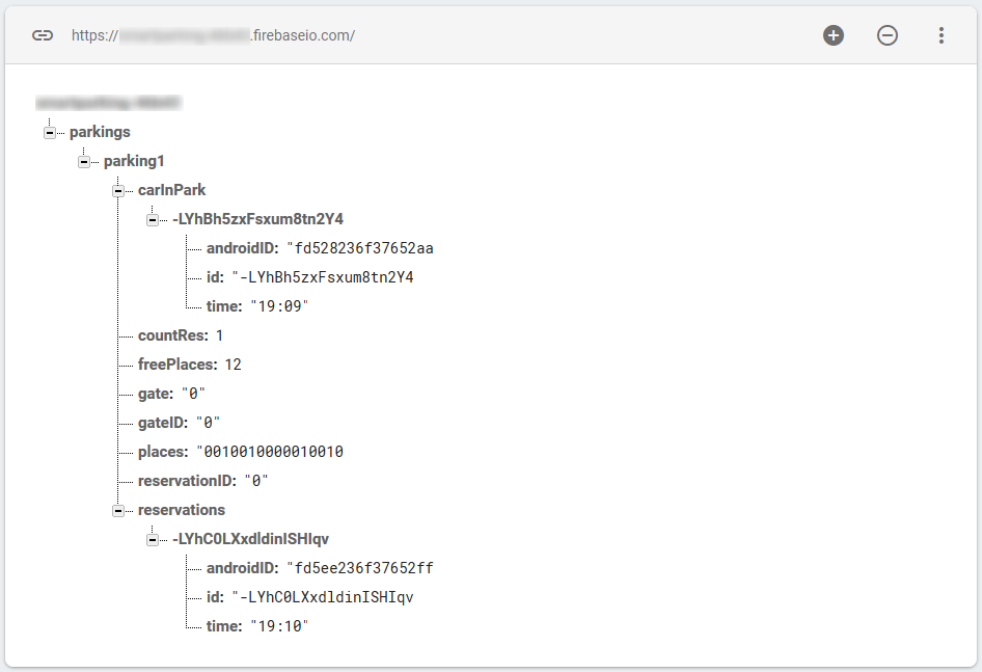
Firebase [17] od společnosti Google v sobě obsahuje užitečné nástroje využitelné např. při vývoji mobilních, webových aplikací apod. Jedním z produktů je NoSQL real-time databáze, kterou využívá řešení popisované v této práci. Základní funkcionalita je pro vývojáře zdarma a obsahuje 100 MB úložiště, maximálně 50 současně připojených klientů a datový přenos 5 GB měsíčně, což při velikosti přenášených zpráv vyhovuje zde popisovanému řešení. Firebase nabízí také celou řadu placených produktů.

Jak už bylo uvedeno, jedná o NoSQL databázi, ta proto neobsahuje žádné tabulky. Data se ukládají ve stromové struktuře, obdobně jako u datového formátu JSON (Obrázek 24). Posloupnost záznamů se ale neukládá do polí jako u JSONu, ale pomocí seřazených unikátních klíčů, u kterých je garantováno pořadí:

- ✓ Aritmetické pro číselné klíče,
- ✓ lexikografické pro ostatní.

Odkazy na data jsou realizovány pomocí syntaxe URL, kde se vnořené položky oddělují lomítky. Následující odkaz například představuje hodnotu `12`:

`https://****************.firebaseio.com/parkings/parking1/freePlaces`



Obrázek 24 – Firebase – ukázka struktury ukládaných dat [autor]

Data lze do databáze zapisovat metodou `set()`, mazat pomocí metody `remove()`. Případně lze využít i metodu `push()`, která nejprve vygeneruje automatický klíč a poté pod něj data uloží [18].

Ke sledování změn v databázi slouží metoda `on()`, která nastaví listener na některou z následujících událostí (Obrázek 25), který následně upozorní na změnu v databázi.

UDÁLOST	POPIS
`value`	Změna libovolných dat ve sledované položce či libovolné podpoložce
`child_added`	Přidání nového potomka v daném umístění
`child_removed`	Odebrání potomka v daném umístění
`child_changed`	Změna v datech libovolného potomka
`child_moved`	Změna v datech libovolného potomka

Obrázek 25- Firebase - tabulka událostí [18]

Databáze je ve výchozím nastavení přístupná všem uživatelům, kteří v ní můžou dělat libovolné změny. Toto nastavení není v reálném nasazení vhodné, proto Firebase nabízí několik autorizačních metod (Obrázek 26). Díky těmto metodám je možné spravovat bezpečnostní pravidla pro skupiny uživatelů.

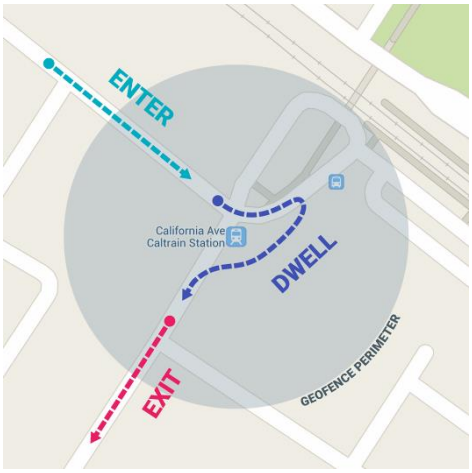
FUNKCE	POPIS
`authWithOAuthPopup`	Přihlášení pomocí OAuth přes Facebook, GitHub, Google či Twitter. Dialog proběhne ve vyskakovacím okně.
`authWithOAuthRedirect`	Jako v předchozím případě, ale dialog proběhne po přesměrování na stránce poskytovatele
`authWithOAuthToken`	Přihlášení pomocí OAuth tokenu (stejní poskytovatelé jako výše)
`authWithPassword`	Přihlášení jménem a heslem; uživatelské účty automaticky spravuje Firebase
`authWithCustomToken`	Autorizace proti vlastní databázi uživatelů pomocí JSON Web Tokens
`authAnonymously`	Dočasné *falešné* anonymní přihlášení s náhodně zvoleným identifikátorem klienta

Obrázek 26 - Firebase - autorizační metody [18]

Dokumentace k databázi je velice obsáhlá, proto tato kapitola obsahuje pouze základní informace pro pochopení problematiky. Kompletní dokumentace je k dispozici na následující webové adrese [https://firebase.google.com/docs/guides/]. Konkrétní metody a techniky pro práci s databází použité v této práci budou vysvětleny v následujících kapitolách.

4.6 Android Geofencing

K upozornění řidiče na dostupnost parkoviště používá mobilní aplikace službu Android Geofencing, která využívá možnosti mobilního telefonu ke zjištění jeho aktuální polohy: GPS, dostupnost známých wifi sítí, vzdálenost od BTS mobilního operátora na základě intenzity signálu mobilní sítě. Pro využití služby je nutné nejprve zadat zeměpisnou šířku a délku konkrétního místa (parkoviště) a poloměr kružnice se středem v této souřadnici [19]. Takto vytvořený kruh, kterému se říká geofence, poté slouží jako plocha pro detekci notifikací, případně jiných akcí, jako například zapnutí



Obrázek 27 - Detekce událostí pomocí geofencingu [19]

Bluetooth, vypnutí vyzvánění telefonu apod. Detekce rozlišuje tři stavy, jak ukazuje obrázek (Obrázek 27). Notifikace může být nastavena pro situaci, kdy telefon vstoupí do vymezeného kruhu (tuto notifikaci využívá řešení popisované touto prací), případně pokud kruh opustí, nebo v něm setrvá určitou dobu. Každý uživatel se systémem Android může mít takto zaregistrováno až 100 geofencí (kruhů) v rámci všech aplikací ve svém telefonu. Pokud se telefon nachází v průniku několika geofencí, může provádět akce samostatně pro každou z nich, případně lze zaslat notifikaci, která upozorní na dostupnost více geofencí (parkovišť) i se vzdáleností ke středu každé z nich, vypočítanou z rozdílu dvou GPS souřadnic (telefon, geofence).

5. Vlastní řešení

5.1 Detekce vozidla

Jak bylo popsáno v kapitole 4.1, detekce vozidla je realizována pomocí magnetometrického čidla MPU-9250, které je propojeno pomocí sběrnice I2C s vývojovou mikroprocesorovou jednotkou Arduino Mini (Obrázek 28), která byla vybrána z důvodu malých rozměrů, snadného ladění aplikace, nízké spotřeby elektrické energie a nízké ceny. Pro komunikaci s čidlem je využita knihovna MPU9250_asukiaaa.h, která umožňuje snadnou obsluhu magnetometrického čidla. Po prvotním ustálení jsou naměřené hodnoty uloženy jako referenční a následně jsou v půlsekundových intervalech porovnávány s aktuálně naměřenými hodnotami (v době inicializace musí být parkovací místo prázdné). Naměřené hodnoty jsou odesílány na sériový port z důvodu možnosti kontroly funkčnosti čidla a naměřených hodnot (Ukázka 1).

```
void loop() {
  if(start){delay(5000);}
  mySensor.magUpdate();
  mX = mySensor.magX();
  mY = mySensor.magY();
  mZ = mySensor.magZ();
  if(start){initSensor(); start = false;}
  Serial.println("magX: " + String(mX));
  Serial.println("maxY: " + String(mY));
  Serial.println("magZ: " + String(mZ));
  testSensor();
  Serial.println(""); // Add an empty line
  delay(500);
}
```

Ukázka 1 - Hlavní program pro komunikaci s magnetometrickým čidlem [autor]

Pokud se díky přiblížení nějakého kovového předmětu (automobilu) hodnoty v některé z os změní a velikost změny překročí zadanou mez, je na digitálním výstupu signalizována přítomnost vozidla. Po každé detekci vozidla a jeho následném opuštění parkovacího prostoru, jsou nové klidové hodnoty uloženy jako referenční. Tím se sníží počet samovolných nebo chybných detekcí, z důvodu dlouhodobých změn intenzity magnetického pole (Ukázka 2).

```
void testSensor(){
  deltaMx = abs(normalMx-mX);
  deltaMy = abs(normalMy-mY);
  deltaMz = abs(normalMz-mZ);
  Serial.println("deltaMx: " + String(deltaMx));
  Serial.println("deltaMy: " + String(deltaMy));
  Serial.println("deltaMz: " + String(deltaMz));
```

```
  if(deltaMx > 7 || deltaMy > 7 || deltaMz > 7){
    Serial.println("Car is present");
    digitalWrite(13,HIGH);
    detect = true;
  }
  else
    {Serial.println("Car is not present");
    digitalWrite(13,LOW);
    if(detect){start = true; detect = false;}
  }
}
```

Ukázka 2 - Detekce vozidla pomocí změny magnetického pole [autor]

Digitální výstup modulu Arduino je propojen se vstupem IQRF transceiveru a tím je zajištěno propojení jednotlivých čidel do IQMESH sítě (Obrázek 28).

Obrázek 28 - Blokové schéma detektoru vozidla [autor]

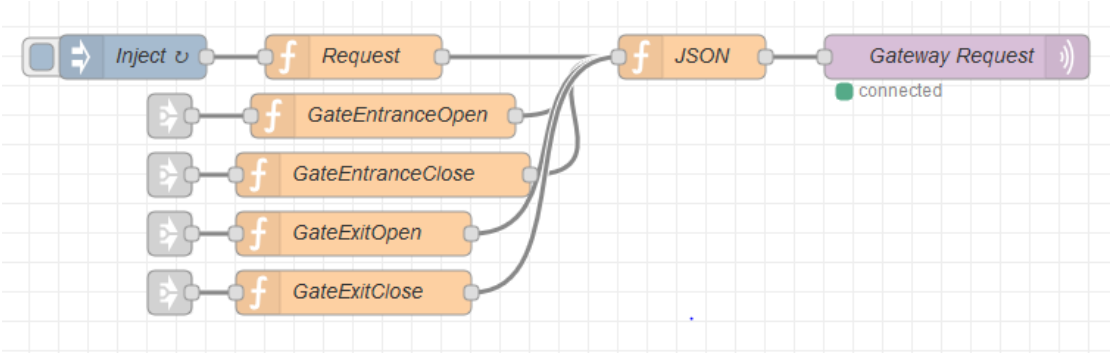
5.2 Ovládání IQRF sítě

Kapitola 4.2 popisuje strukturu a vytvoření IQRF sítě, při kterém byl do transceiverů nahrán DPA custom handler. Ten využívá řešení popisované v této práci ke zjištění stavu digitálních vstupů na IQRF transceiverech, na kterých jsou připojena magnetometrická čidla. Hlavní část handleru je vidět v následující ukázce (Ukázka 3).

```
bit CustomDpaHandler(){
  // Handler presence mark
  clrwdt();
   // Return 1 if IQRF PORTB.4 == 1 (Car is present)
  if ( GetDpaEvent() == DpaEvent_FrcValue  &&  _PCMD == FRC_USER_BIT_FROM  &&  carIsPresent )
    responseFRCvalue.1 = 1;
   return FALSE;
}
```

Ukázka 3 - DPA custom handler [autor]

Veškerá komunikace s IQRF sítí je následně ovládána pomocí programovací utility NodeRED, popsané v kapitole 4.4. Zjišťování stavu jednotlivých senzorů zapojených do IQMESH sítě probíhá na základě programové flow, kterou ukazuje následující obrázek (Obrázek 29).



Obrázek 29 - Odeslání požadavku do IQRF sítě [autor]

Základní funkcionalitu zajišťují programové nody v první řádce (Obrázek 29), význam ostatních je vysvětlen v kapitole 5.4.1.

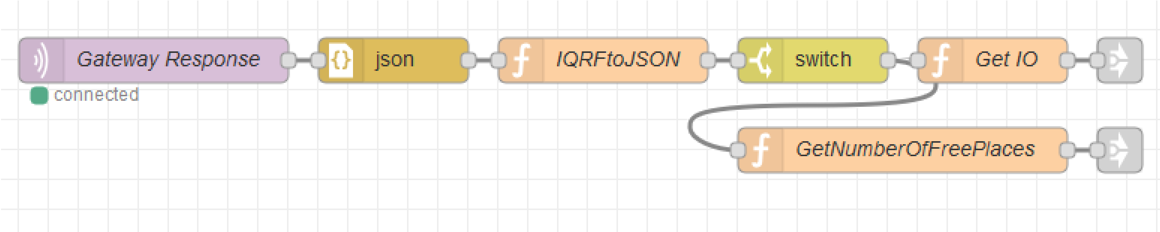
- ✓ **Inject** – tento nod zajistí, aby se provedly operace, které jsou připojeny za ním (Request, JSON, Gateway Request) a to buď kliknutím na modrý čtverec v pření části nodu anebo opakovaně (zde nastaven interval 5 s),
- ✓ **Request** – vlastní funkce napsaná v jazyce JavaScript, která zajišťuje formátování FRC příkazu pro IQRF Gateway (Ukázka 3),
- ✓ **JSON** – nod, který převádí data DPA příkazu do formátu JSON,
- ✓ **Gateway Request** – nod zajišťující propojení s technologií IQRF (koordinátorem připojeným k GPIO portům mini počítače UpBoard).

Pomocí tohoto flow je každých 5 s do IQRF sítě odeslán FRC příkaz ke zjištění stavu digitálního vstupního portu u všech transceiverů připojených v síti.

```
var data={
    type: "raw",
    request: {
        nadr: "0x0000",
        pnum: "0x0D",
        pcmd: "0x00",
        hwpid: "0xFFFF",
        pdata: "0x400000",
    },
    timeout: 1000
}
msg.payload=data;
return msg;
```

Ukázka 4 - Vytvoření FRC příkazu pomocí jazyka JavaScript [autor]

Po odeslání příkazu do sítě je zjištěn stav jednotlivých digitálních vstupů a odpověď zpracovává následující flow (Obrázek 30).



Obrázek 30 - Zpracování požadavku z IQRF sítě [autor]

Po získání odpovědi (Ukázka 4) se následně připravují data pro zápis do databáze. Položka „response" obsahuje data z jednotlivých čidel.

- ✓ **Gateway Response** – nod zajišťující propojení s technologií IQRF (koordinátorem připojeným k GPIO portům mini počítače UpBoard),
- ✓ **json** – nod zajišťující konverzi odpovědi z formátu JSON, pro další zpracování,
- ✓ **IQRFtoJSON** – vlastní funkce pro rozdělení (parser) JSON dat,
- ✓ **switch** – větví program podle typu odpovědi (připraveno pro případné rozšíření funkcionality),
- ✓ **Get IO** – vlastní funkce, která formátuje data pro zápis do databáze (Ukázka 5),
- ✓ **GetNumberOfFreePlaces** – vlastní funkce, která ze získaných dat o stavu jednotlivých čidel vypočítá počet volných míst na parkovišti a připraví tuto informaci pro zápis do databáze (Ukázka 6),
- ✓ **Link 1 a 2** – propojení na další část programu - zápis dat do databáze (Obrázek 31).


```
{
    "ctype": "dpa",
    "type": "raw",
    "timeout": 1000,
    "request": "00.00.0d.00.ff.ff.40.00.00",
    "response":
"00.00.0d.80.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00",
    "status": "STATUS_NO_ERROR"
}
```


Ukázka 5 - Formát odpovědi z IQRF koordinátorun [autor]

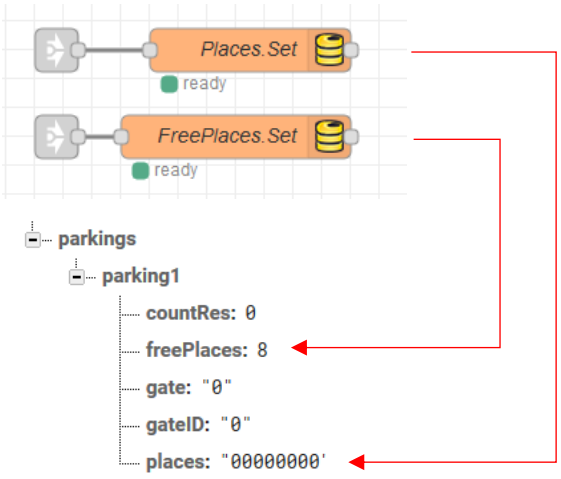
```
var res=msg.payload.response.pdata.split(".");
var inp = "";
var z = 0;
for(j=0;j<z+1;j++){
    var io =  parseInt(res[33 + j],16);
    for(i=0;i<8;i++){
            var m = io%2;
            var io = parseInt(io/2);
            if (j!==0  && i===0) inp = m + "." + inp;
            else inp = m + inp;
    }
}
msg.payload = inp;
return msg;
```

Ukázka 6 - Konverze získaných dat o stavu jednotlivých parkovacích míst pro zápis do databáze [autor]

```
var res=msg.payload.response.pdata.split(".");
var z = 0;
var c = 0;
for(j=0;j<z+1;j++){
    var io =  parseInt(res[33 + j],16);
    for(i=0;i<8;i++){
            var m = io%2;
            if(m===0){c++;}
            var io = parseInt(io/2);
    }
}
msg.payload = c;
return msg;
```

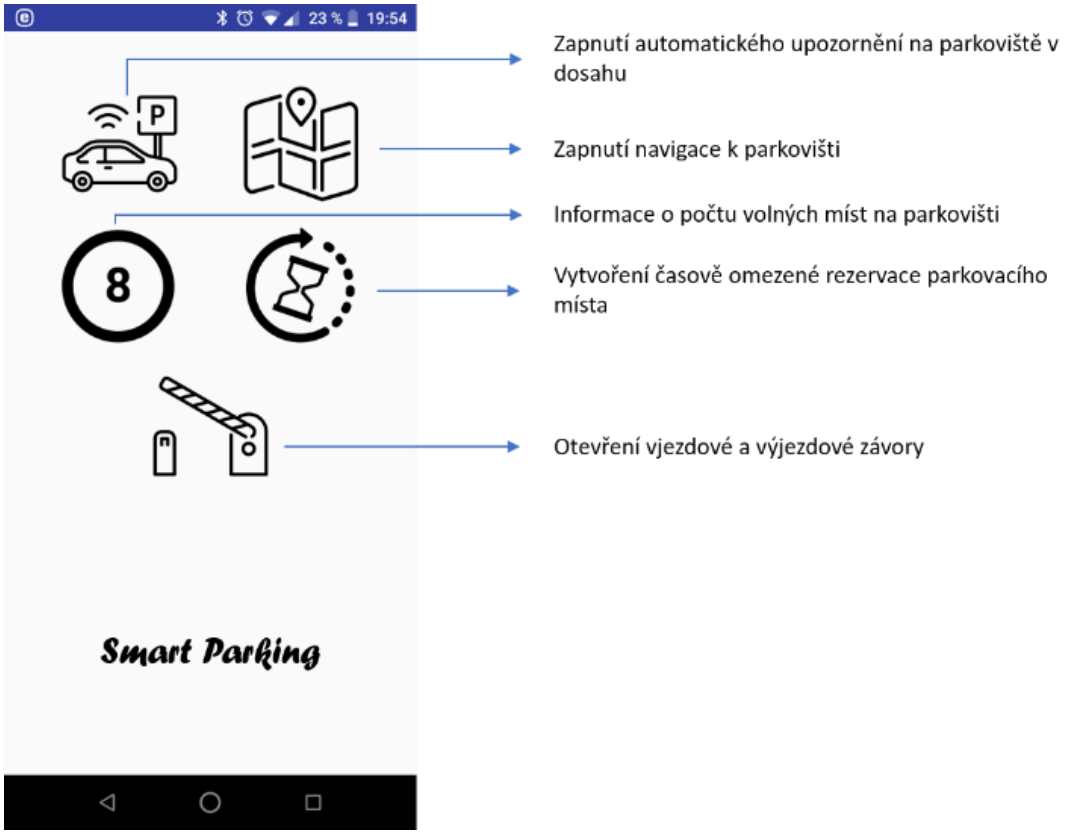
Ukázka 7 - Výpočet počtu volných míst na parkovišti [autor]

Následuje část programu pro zápis dat, která obsahuje pouze linky na předchozí část programu popsanou výše a vlastní nody pro zápis dat do Firebase databáze.



Obrázek 31 - Uložení informací o stavu a počtu jednotlivých parkovacích míst do databáze [autor]

K takto uloženým datům následně přistupuje mobilní aplikace pro telefony s operačním systémem Android, pomocí které uživatel přistupuje k celému systému. Ta mu umožňuje nejen zjistit počet volných míst na parkovišti, ale i provést rezervaci parkovacího místa, zapnout automatické upozornění na parkoviště v dosahu, spustit navigaci k parkovišti a povolit vjezd (otevřít závoru).



Obrázek 32 - Hlavní obrazovka mobilní aplikace [autor]

5.3 Android aplikace

Android aplikace umožňuje ovládání všech funkcí na hlavní obrazovce tak, aby bylo pro řidiče co nejpřehlednější a nejjednodušší. Proto se funkce zapínají „kliknutím“ na jednoduché ikony, aby řidič za volantem nebyl nucen pročítat uživatelskou nabídku. Přehled všech funkcí a význam jednotlivých ikon představuje obrázek (Obrázek 32).

5.3.1 Automatická detekce parkoviště v dosahu

Jak je popsáno v kapitole 4.6, k automatické detekci parkoviště v dosahu používá mobilní aplikace službu geofencing [19]. K tomu slouží třída Constants, která obsahuje parametry pro vytvoření geofence (GPS souřadnici a rádius). Pomocí hash mapy lze vytvořit několik geofencí, přidáním názvu a souřadnice, pomocí dalšího příkazu `LANDMARKS.put(……));` (Ukázka 8).

```
public class Constants {
    public static final float GEOFENCE_RADIUS_IN_METERS = 1000;
    public static final HashMap<String, LatLng> LANDMARKS = new HashMap<String, LatLng>();
    static {
        // Parking 1
        LANDMARKS.put("Parking 1", new LatLng(50.420860, 16.185796));
    }
}
```

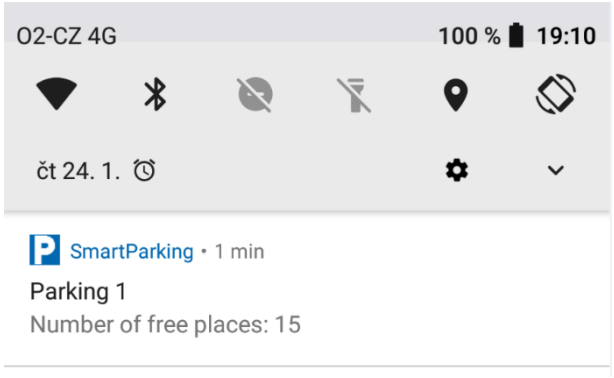
Ukázka 8 – Nastavení parametrů geofence [autor]

Zavoláním metody `populateGeofenceList()` v `MainActivity` dojde k vytvoření vlastní geofence (Ukázka 9).

```
public void populateGeofenceList() {
        for (Map.Entry<String, LatLng> entry : Constants.LANDMARKS.entrySet()) {
            mGeofenceList.add(new Geofence.Builder()
                    .setRequestId(entry.getKey())
                    .setCircularRegion(
                            entry.getValue().latitude,
                            entry.getValue().longitude,
                            Constants.GEOFENCE_RADIUS_IN_METERS
                    )
                    .setExpirationDuration(Geofence.NEVER_EXPIRE)
                    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER)
                    .build());
        }
    }
```

Ukázka 9 - Vytvoření geofence [autor]

Třída GeofenceTransitionsIntentService následně vytvoří notifikační kanál, pomocí kterého informuje řidiče notifikací o dostupném parkovišti, při vstupu do vytvořené geofence (Obrázek 33).



Obrázek 33 - Notifikace po vstoupení do geofence [autor]

5.3.2 Navigace k parkovišti

Ke spuštění navigace, aplikace využívá Google API, resp. Mapy Google pro systém Android [20], které umožňují spustit mapu v následujících režimech:

- ✓ Zobrazení mapy na určitém místě a s určitou úrovní zvětšení,
- ✓ vyhledání místa a jeho zobrazení na mapě,
- ✓ navigace s výběrem typu dopravy (autem, na kole, pěšky),
- ✓ zobrazení panoramatických snímků ve službě Google Street View.

Ke spuštění mapy je nejprve potřeba vytvořit objekt „Intent“ a v něm specifikovat v jakém režimu se mapa spustí. Intent obsahuje speciální řetězec (URI), který přesně specifikuje požadovanou akci. Po vytvoření Intentu se aktivita spustí metodou `startActivity()`.

```
public void startNavigationButtonHandler(View view){
        Uri gmmIntentUri = Uri.parse("google.navigation:q= 50.475367, 16.179489");
        Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
        mapIntent.setPackage("com.google.android.apps.maps");
        startActivity(mapIntent);
}
```

Ukázka 10 - Vytvoření Intentu a spuštění Map Activity [autor]

Jak je vidět na předchozí ukázce (Ukázka 9), Intent je vytvořen pomocí URI, který definuje typ akce jako navigaci k nastavené GPS souřadnici. Dále je do Intentu přidán balíček, který zajistí jeho zpracování pomocí aplikace Mapy Google.

5.3.3 Informace o obsazenosti parkoviště

Informace o počtu volných a obsazených míst parkoviště ukládá NodeRED do Firebase databáze, jak bylo popsáno v kapitole 5.2. V Android aplikaci je vytvořena instance databáze: `FirebaseDatabase database = FirebaseDatabase.getInstance()` a reference na její konkrétní položky:

`DatabaseReference myRef… = database.getReference("parkings/parking1/……");` Následně také listener, který provádí příslušné akce pokaždé, když dojde ve sledovaných položkách ke změně hodnoty.

```
public void readFreePlaces(){
     myRefFplaces.addValueEventListener(new ValueEventListener() {
         @Override
         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
             places = dataSnapshot.getValue().toString();
             int pl = Integer.parseInt(places);
             int cr = (int) (pl - countRes);
             Fplaces.setText(String.valueOf(cr));
         }
          @Override
         public void onCancelled(@NonNull DatabaseError databaseError) {
             places = "Error";
         }
});
```

Ukázka 11 - Nastavení listeneru na změnu počtu volných míst na parkovišti [autor]

Metoda popsaná výše (Ukázka 11) při každé změně počtu volných míst nastaví obsah `textView` na hlavní aktivitě pomocí metody `setText` tak, aby tlačítko (Obrázek 32) vždy obsahovalo aktuální počet volných míst na parkovišti. Při kliknutí na toto tlačítko se spustí nová aktivita, která obsahuje `listView` a to informuje řidiče o konkrétní obsazenosti jednotlivých parkovacích míst na parkovišti. Stejná aktivita se spouští také při vjezdu na parkoviště, po požadavku na otevření vjezdové závory, aby měl řidič při vjezdu na parkoviště přehled o volných místech. K tomu aplikace stejně jako v předchozím případě využívá data uložená v databázi, jejichž změnu hlídá listener. Informace o obsazenosti jednotlivých míst jsou v databázi uložena pomocí NodeRED ve formátu řetězce `„0000110100110010“`, který obsahuje šestnáct informací `„1“` nebo `„0“`. Pokud je na příslušné pozici `„1“`, znamená to, že je toto místo obsazené. V opačném případě je místo volné. Metoda `getOccupancy()` (Ukázka 12) v této aktivitě naplní pole `OCCUPANCY[]` hodnotami `„FREE“`, nebo `„OCCUPIED“`. Tyto hodnoty jsou potom v adaptéru, který využívá `listView`, předány do jednotlivých `textView` (Ukázka 13), jak ukazuje obrázek (Obrázek 34).

```
public void getOccupancy(){
        for(int i=0;i<8;i++){
            System.out.println(places.substring(i,i+1));
            if(places.substring(i,i+1).equals("0")){OCCUPANCY[i]="FREE";}
            if(places.substring(i,i+1).equals("1")){OCCUPANCY[i]="OCCUPIED";}
            if(places.substring(i+8,i+9).equals("0")){OCCUPANCY[i+8]="FREE";}
            if(places.substring(i+8,i+9).equals("1")){OCCUPANCY[i+8]="OCCUPIED";}
            NUMBERS[i] = i + 1;
        }
    }
```

Ukázka 12 - Naplnění textového pole na základě dat z databáze [autor]

```
public View getView(int i, View view, ViewGroup viewGroup) {
            view = getLayoutInflater().inflate(R.layout.customlayout,null);
            TextView tvNumbers = view.findViewById(R.id.tvNumbers);
            TextView tvOccupancy = view.findViewById(R.id.tvOccupancy);
            TextView tvOccupancy1 = view.findViewById(R.id.tvOccupancy1);
            tvNumbers.setText(String.valueOf(NUMBERS[i]));
            tvOccupancy.setText(String.valueOf(OCCUPANCY[i]));
            tvOccupancy1.setText(String.valueOf(OCCUPANCY[i+8]));
            if(OCCUPANCY[i].equals("FREE")){tvOccupancy.setTextColor(Color.parseColor("#00ff00"));}
if(OCCUPANCY[i].equals("OCCUPIED")){tvOccupancy.setTextColor(Color.parseColor("#ff0000"));}
if(OCCUPANCY[i+8].equals("FREE")){tvOccupancy1.setTextColor(Color.parseColor("#00ff00"));}
if(OCCUPANCY[i+8].equals("OCCUPIED")){tvOccupancy1.setTextColor(Color.parseColor("#ff0000"));}
            return view;
        }
    }
```

Ukázka 13 - Naplnění textových polí a nastavení barvy písma na základě obsahu [autor]

Obrázek 34 - Zobrazení obsazenosti parkoviště [autor]

5.3.4 Vytvoření rezervace parkovacího místa

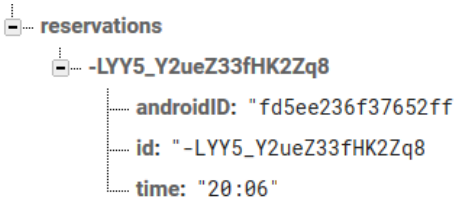
Každý řidič má možnost vytvořit krátkodobou rezervaci jednoho parkovacího místa. K tomuto účelu aplikace využívá třídu `Reservation.java`, která obsahuje tři atributy:

- ✓ **id** – ID položky vygenerované databází Firebase,
- ✓ **androidID** – unikátní 64 bitové číslo vygenerované systémem Android,
- ✓ **time** – čas vytvoření rezervace.

Před vytvořením rezervace aplikace na základě Android ID zkontroluje, zda už není uživatel v databázi uložen (už má platnou rezervaci nebo už je na parkovišti) (Ukázka 14) a zda jsou na parkovišti ještě nějaká volná místa k zarezervování (počet volných míst – počet platných rezervací). Pokud je rezervace možná, požádá databázi o `id` nové položky `id = myRefRes.push().getKey();` vytvoří novou instanci třídy `Reservation reservations = new Reservation(id, androidID, time );` a následně tuto nově vytvořenou instanci uloží do databáze `myRefRes.child(id).setValue(reservations);` (Obrázek 35).

```
public void onDataChange(DataSnapshot snapshot) {
    countRes = snapshot.getChildrenCount();
    updateCountReservation();
    resIndicator = false;
    for (DataSnapshot postSnapshot: snapshot.getChildren()) {
        Reservation post = postSnapshot.getValue(Reservation.class);
        String resCheck = post.getAndroidID();
        if(resCheck.equals(androidID)){resIndicator = true;}
    }
}
```

Ukázka 14 - Zjištění stavu rezervace uživatele [autor]



Obrázek 35 - Ukázka uložení rezervace do databáze [autor]

Stav rezervací v pravidelných intervalech sleduje také mini PC prostřednictvím NodeRED a pokud dojde k překročení časového limitu rezervace, je tato rezervace automaticky z databáze odstraněna. K odstranění rezervace dojde také v případě, že řidič s platnou rezervací dorazí na parkoviště a vydá povel k otevření vjezdové brány. Postupy při odstraňování rezervace z databáze jsou podrobněji popsány v následujících kapitolách.

5.3.5 Otevření vjezdové / výjezdové závory

K otevření vjezdové / výjezdové závory, aplikace opět využívá Firebase databázi, která obsahuje dvě položky:

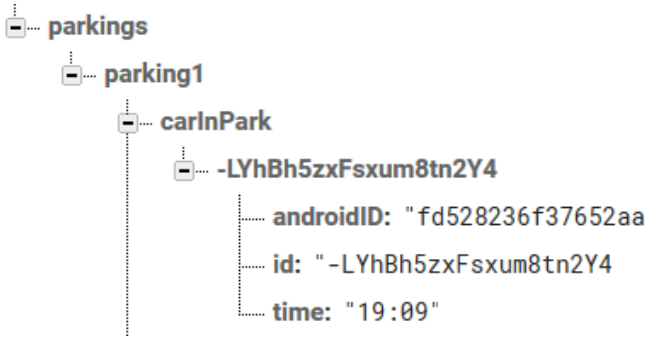
- ✓ **gate** – může obsahovat dvě hodnoty 1 a 0 (otevření a zavření vjezdové závory),
- ✓ **gateID** – v případě, že je auto přítomno na parkovišti, uloží aplikace `androidID` uživatele, který chce otevřít výjezdovou závoru.

Aplikace obsahuje třídu `CarInPark.java`, která obsahuje, podobně jako třída `Reservation.java`, tři atributy:

- ✓ **id** – ID položky vygenerované databází Firebase,
- ✓ **androidID** – unikátní 64 bitové číslo vygenerované systémem Android,
- ✓ **time** – čas příjezdu na parkoviště.

Stejně jako při požadavku na vytvoření rezervace, tak i při požadavku na otevření závory, aplikace nejprve zjišťuje na základě `androidID`, zda je vozidlo na parkovišti (má záznam v databázi). Dále je také pomocí vzdálenosti dvou GPS souřadnic kontrolováno, zda není překročena povolená vzdálenost mobilního telefonu od vjezdové závory. Pokud jsou podmínky splněny, zapsáním „`1`“ do položky `gate` Firebase databáze, NodeRED zajistí otevření vjezdové závory. Následuje požadavek na nové `id` pro uložení položky do databáze `String id = myRefCar.push().getKey();`, vytvoření nové instance třídy `CarInPark carInPark = new CarInPark(id, androidID, time);` a následně je položka zapsána do databáze `myRefCar.child(id).setValue(carInPark);` (Obrázek 36). Po otevření závory je pomocí NodeRED smazána případná rezervace řidiče, který právě vjíždí na parkoviště.

Pokud je vozidlo na parkovišti (`androidID` uživatele je v databázi), dojde k zápisu `androidID` do položky `gateID` a na základě tohoto zápisu NodeRED otevře výjezdovou bránu a odstraní z databáze záznam příslušného uživatele. Postupy při otevírání a zavírání závor parkoviště a odstraňování záznamů z databáze pomocí NodeRED jsou podrobněji popsány v následujících kapitolách.



Obrázek 36 - Záznam vozidla přítomného na parkovišti [autor]

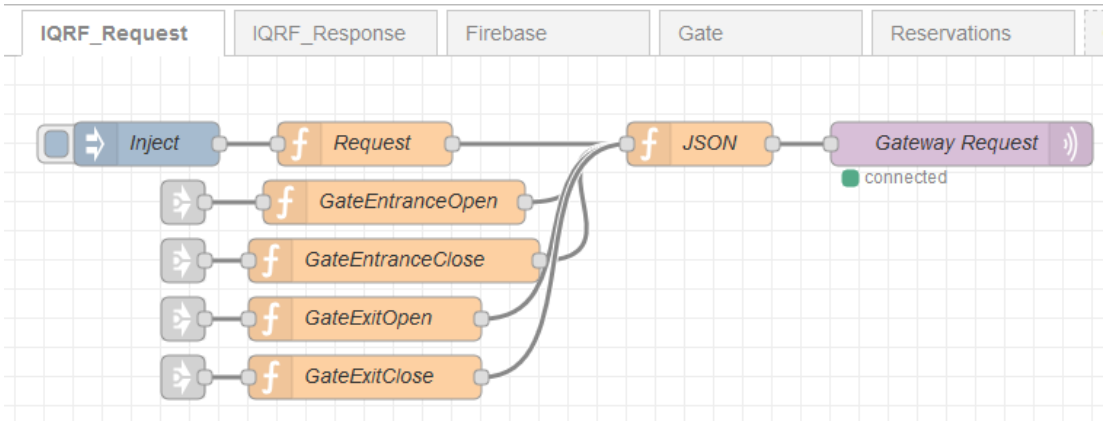
5.4 Obsluha parkoviště pomocí NodeRED

Jak už bylo zmíněno v předchozích kapitolách, s databází komunikuje nejen mobilní aplikace, ale i mini PC a to prostřednictvím NodeRED. Pro přehlednost jsou jednotlivá programová flow rozdělena do samostatných záložek:

- ✓ **IQRF_Request** – odesílá příkazy do IQRF sítě,
- ✓ **IQRF_Response** – zpracovává odpovědi z IQRF sítě,
- ✓ **Firebase** – komunikuje z Firebase databází,
- ✓ **Gate** – obsluhuje vjezdovou a výjezdovou závoru
- ✓ **Reservations** – spravuje rezervace uživatelů.

5.4.1 IQRF_Request

Programová flow IQRF_Request (Obrázek 37) byla částečně popsána v kapitole 5.2. Programové nody na druhém až pátém řádku odesílají požadavky na otevření či zavření vjezdové, nebo výjezdové závory. Požadavky přicházejí z ostatních záložek pomocí propojení (šedé šipky na začátcích řádků).



Obrázek 37 - Programová flow IQRF_Request [autor]

Jednotlivé požadavky jsou realizovány pomocí vlastních funkcí napsaných v programovacím jazyce JavaScript. Jedná se o vytvoření DPA příkazů pro IQRF síť (Ukázka 15). Vzhledem k fyzické absenci závory jsou za účelem ladění aplikace povely simulovány rozsvícením a zhasnutím červené LED na IQRF koordinátoru (vjezdová brána) a zelené LED na IQRF koordinátoru (výjezdová brána). Ve skutečnosti LED nahradí digitální výstup, který odešle povel do závory.

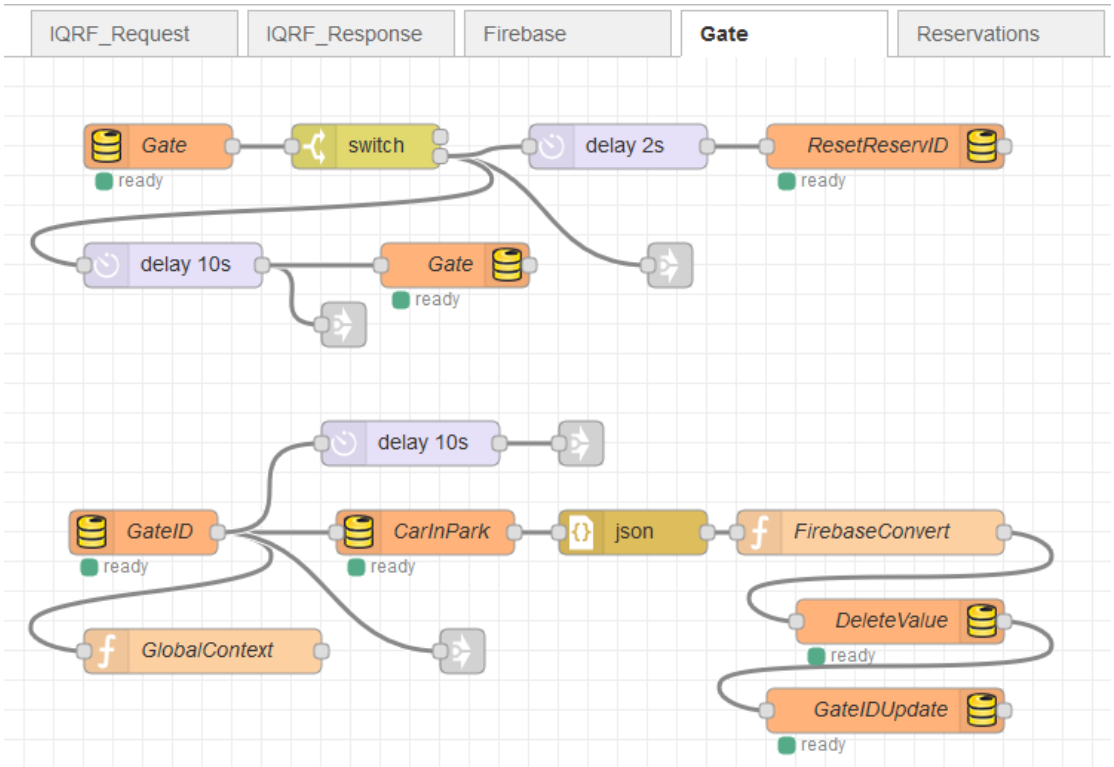
```
var data={
    type: "raw",
    request: {
        nadr: "0x0000",
        pnum: "0x06",
        pcmd: "0x01",
        hwpid: "0xFFFF",
        pdata: "",
    },
    timeout: 1000
}
msg.payload=data;
return msg;
```

Ukázka 15 – Vytvoření požadavku pro IQRF síť [autor]

Programové flow IQRF_Response a Firebase byly kompletně popsány v kapitole 5.2, proto zde nejsou uvedeny.

5.4.2 Gate

Programová flow Gate je rozdělena na dvě části. První část ovládá vjezdovou závoru a druhá výjezdovou (Obrázek 38).



Obrázek 38 - Programová flow Gate [autor]

První flow začíná nodem `Gate,` což je vlastně listener, který hlídá změnu položky `gate` v databázi. Následuje `switch`, který při zápisu hodnoty „1“ (otevření závory) pokračuje druhým výstupem na `link` vedoucí k nodu `GateEntranceOpen` (kapitola 5.4.1) na záložce `IQRF_Request`. Dalším nodem je časová prodleva na projetí vozidla (senzory detekující vozidlo v prostoru závory jsou její součástí) a flow pokračuje linkem na `GateEntranceClose` a zápisem hodnoty „0“ do položky `gate` v databázi. Switch v tomto případě zajistí, aby nedošlo k opětovnému otevření závory po změně hodnoty v databázi tím, že v případě přečtení hodnoty „0“ pokračuje výstupem 1, na který není připojen žádný další programový nod. Při otevření vjezdové závory se do položky `reservationID` ukládá id telefonu, který o otevření požádal. Toto id je poté porovnáno se záznamy v `reservations` a pokud je nalezena platná rezervace, je vymazána. O reset položky `reservationID` se po uplynutí časové prodlevy postará nod `ResetReservID`.

Druhá flow začíná nodem `GateID`, který hlídá změnu položky `gateID` databáze. Do této položky mobilní aplikace ukládá ID telefonu, který je registrován na parkovišti (má záznam v `carInPark`) a žádá o otevření výjezdové závory. Program pomocí linku pokračuje na záložku `IQRF_Reqest`, kde pomocí nodu `GateExitOpen` otevře výjezdovou závoru. Po uplynutí časové prodlevy na projetí vozidla je pomocí linku na `GateExitClose` závora opět zavřena. Flow pokračuje načtením položek `CarInPark` z databáze a jejich převedením do formátu JSON. Tato data jsou předána funkci `FirebaseConvert` (Ukázka 16), která podle ID telefonu, uloženého v `GlobalContext`, vyhledá záznam v databázi a pomocí nodu `DeleteValue` ho vymaže. Posledním krokem programu je reset položky `gateID`.

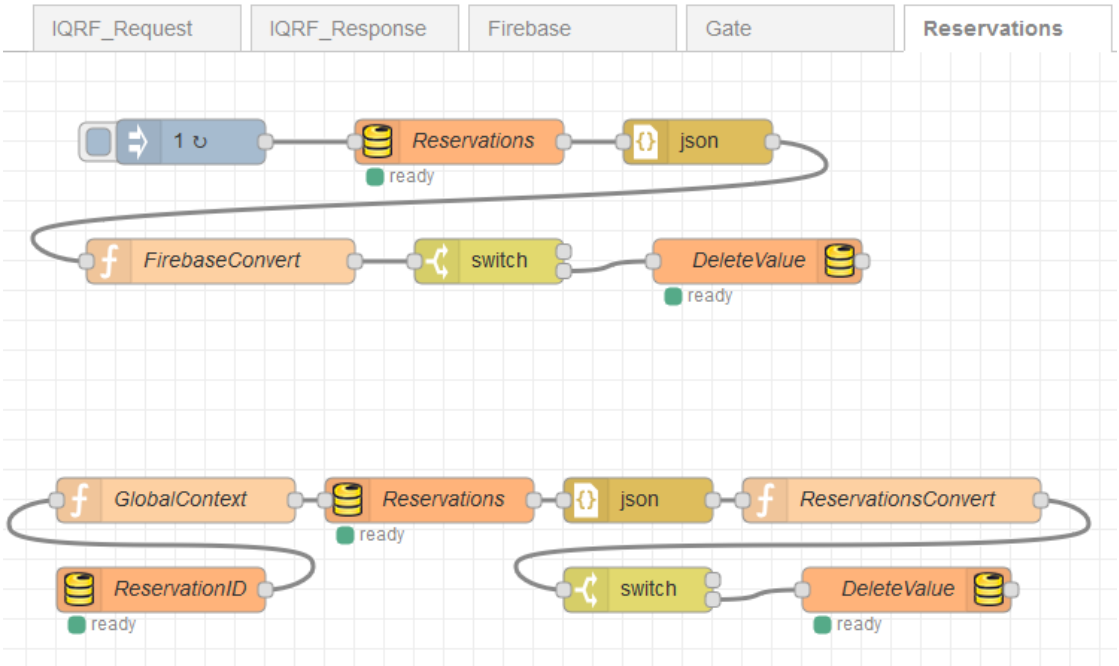
```
var response = msg.payload;
var gateID = global.get("gateID");
var data = "";
var data1 = response.split("{").toString();
var l = data1.length;
for(i=0;i<l;i++){
    var s = data1.substring(i,i+1);
    if(s == "\""){}
    else if(s == "{"){}
    else if(s == "}"){}
    else {data = data + s;}
}
var allmsg = data.split(",-");
var al = allmsg.length;
for(i=1;i<al;i++){
var data3 = allmsg[i].split(",");
var dl = data3[1].length;
var data2 = data3[1].substring(10,dl);
var cesta = "parkings/parking1/carInPark/" + data3[2].substring(3,data3[2].length);
```

```
if(data2==gateID){
    node.send({childpath:cesta});
    }
}
return;
```

Ukázka 16 - Vyhledání ID položky a její předání k odstranění [autor]

5.4.3 Reservations

I v případě této flow došlo k rozdělení programu na dvě části (Obrázek 39). První část každou minutu kontroluje délku rezervací a v případě překročení časového limitu je odstraňuje a druhá kontroluje rezervace řidičů vjíždějících na parkoviště, a pokud takový záznam existuje, odstraní ho.



Obrázek 39 - Správa rezervací parkoviště [autor]

Nod `inject` spouští každou minutu načtení rezervací z databáze a jejich konverzi do formátu JSON. Data jsou předána funkci `FirebaseConvert` (Ukázka 17), která zkontroluje délku každé rezervace a v případě překročení časového limitu, předá příslušnou rezervaci k vymazání. Pokud je nalezena rezervace k vymazání, switch pokračuje druhým výstupem, v opačném případě směruje program do prvního výstupu, ve kterém program končí.

Jak bylo zmíněno v předchozí kapitole, při otevření vjezdové závory dochází k zápisu ID telefonu do položky `reservationID`. Změnu jejího stavu hlídá nod `ReservationID`, kterým začíná druhá část této flow. ID telefonu je uloženo do globální proměnné. Dále jsou z databáze načteny všechny rezervace, převedeny do formátu JSON a předány funkci `ReservationConvert` (Ukázka 18), která porovnává záznamy v databázi s uloženým ID telefonu. Při nalezení shody je záznam vymazán. Switch plní stejnou funkci jako v předchozím případě.

```
var response = msg.payload;
var data = "";
var nic = 1;
var data1 = response.split("{").toString();
var l = data1.length;
for(i=0;i<l;i++){
    var s = data1.substring(i,i+1);
    if(s == "\""){}
    else if(s == "{"){}
    else if(s == "}"){}
    else {data = data + s;}
}
var allmsg = data.split(",-");
l = allmsg.length;
for(i=1;i<l;i++){
var data1 = allmsg[i].split(":");
var fh = parseInt(data1[4]);
var fm = parseInt(data1[5]);
var date = new Date();
var h = date.getHours()+1;
var m = date.getMinutes();
var delta = (h*60+m)-(fh*60+fm);
    if(delta > 60){
        var cesta = "parkings/parking1/reservations/-" + data1[0];
        node.send({childpath:cesta});
    }
    else{
        node.send({payload:nic});
    }
}
return;
```

Ukázka 17 - Kontrola překročení časového limitu rezervace [autor]

```
var response = msg.payload;
var reservationID = global.get("reservationID");
var data = "";
var data1 = response.split("{").toString();
var l = data1.length;
for(i=0;i<l;i++){
    var s = data1.substring(i,i+1);
    if(s == "\""){}
    else if(s == "{"){}
    else if(s == "}"){}
    else {data = data + s;}
}
```

```
var allmsg = data.split(",-");
var al = allmsg.length;
for(i=1;i<al;i++){
    var data3 = allmsg[i].split(",");
    var androidID = data3[1].substring(10,data3[1].length);
    var path = "parkings/parking1/reservations/" +
    data3[2].substring(3,data3[2].length);
    if(androidID == reservationID){
    node.send({childpath:path});

    else {node.send({payload:1});}

}
return;
```

Ukázka 18 - Vymazání rezervace řidiče vjíždějícího na parkoviště [autor]

6. Testy

Testování detekce vozidla pomocí magnetometrického čidla probíhalo ve dvou fázích. V první fázi proběhl test na samovolné detekce a ve druhé test detekce vozidel různých velikostí.

Pro první fázi testu byl upraven program v modulu Arduino tak, aby počítal detekce vozidel. Test probíhal po dobu 24 hodin, dvakrát za vteřinu. Senzor byl umístěn a zafixován, aby byl znemožněn jeho pohyb jakýmkoliv směrem. Za 24 hodin proběhlo 172 800 testů a pouze čtyřikrát došlo k chybné detekci (Tabulka 1).

Počet testů	Chybné detekce	Chybovost
172 800	**4**	**0,02315 ‰**

Tabulka 1 - Test samovolných detekcí vozidla [autor]

Ve druhé fázi byla testována správná detekce na pěti vozidlech různé velikosti. Každé vozidlo bylo třicetkrát umístěno nad magnetometrické čidlo a výsledky testu jsou v následující tabulce (Tabulka 2).

Vozidlo	Počet testů	Chybné detekce	Chybovost
Toyota Yaris	30	1	3,3 %
Škoda Roomster	30	0	0 %
Škoda Octavia kombi	30	0	0 %
Citroën Jumpy	30	0	0 %
Nissan Navara	30	0	0 %
Celková chybovost	**150**	**1**	**0,66 %**

Tabulka 2 - Test detekce vozidel různých velikostí [autor]

Jak je vidět z tabulky (Tabulka 2), úspěšnost detekce byla vysoká, pouze u nejmenšího vozidla byla zaznamenána jedna chybná detekce.

Test automatické detekce parkoviště v dosahu probíhal po dobu dvou týdnů. V prvním týdnu probíhaly testy pro zvolení vhodného rádiusu geofence tak, aby řidič dostal informaci o dostupném parkovišti včas a byl schopen bez problémů na tuto informaci reagovat. Po týdenním testování byl jako nejvhodnější rádius zvolen 1000 m. Díky dříve popisovanému zpoždění dostával řidič v městském provozu informace v průměru 500 m – 800 m před parkovištěm. V druhém týdnu, po nastavení vhodného rádiusu, probíhal test notifikací na trase dlouhé 10 km, kde bylo nastaveno 5 geofencí. Deset řidičů tuto trasu projelo celkem desetkrát, takže bylo provedeno celkem 500 testů. Ve třech případech nebyla notifikace obdržena a ve čtyřech případech řidič obdržel notifikaci ve vzdálenosti menší než 200 m před parkovištěm. Z toho vyplývá, že chybovost byla 1,4 % (Tabulka 3).

Celkový počet testů	Notifikace v pořádku	Notifikace nedoručena	Opožděná notifikace	Chyba
500	493	3	4	1,4 %

Tabulka 3 - Test notifikací [autor]

7. Ekonomický pohled na navrhované řešení

Tato práce si klade za cíl najít cenově dostupné řešení inteligentního parkovacího systému pro účely uvedené v kapitole 3. S tímto ohledem byly také vybírány jednotlivé komponenty systému, aby cenová náročnost řešení nebyla vysoká. Přibližné náklady na jednotlivé komponenty systému shrnuje následující tabulka (Tabulka 4).

Komponenta	Cena
MPU-9250	60 Kč
Arduino Mini	45 Kč
IQRF TR-72D	325 Kč
Baterie	500 Kč
UpBoard - mini PC	2250 Kč
Závora	16000 Kč

Tabulka 4 - Náklady na jednotlivé komponenty systému [autor]

Z tabulky (Tabulka 4) vyplývá, že při použití dvou závor (vjezdová / výjezdová) a mini PC, jsou fixní náklady na jedno parkoviště přibližně 34 000 Kč. Dále je nutné připočítat náklady na senzory jednotlivých parkovacích míst, které se skládají z modulů MP-9250, Arduino Mini, IQRF TR-72D a baterie, což představuje přibližně 1 000 Kč na jedno parkovací místo. Pro konkrétní parkoviště zmíněné v kapitole 3, jsou při šestnácti parkovacích místech přibližné náklady 50 000 Kč.

8. Další vývoj aplikace

Jak bylo popsáno v kapitole 3, této práce, navrhované řešení není primárně určeno ke komerčním účelům, ale za předpokladu vhodného rozšíření funkcionality by mohlo být nasazeno i do komerčního sektoru. V současném stavu, vzhledem k tomu, že při vjezdu automobilu na parkoviště se do databáze ukládá čas, je aplikace schopna vypočítat délku parkovaní a pokud by byly doplněny i cenové tarify, bylo by možné vypočítat také cenu za parkování. Pro použití v komerční sféře by bylo nutné systém doplnit o možnost zaplacení parkovného, a to buď platebním automatem umístěným přímo na parkovišti, anebo doplněním aplikace o propojení na platební bránu. Dále je nutné vytvořit aplikaci i pro operační systém iOS, aby i majitelé mobilních telefonů iPhone mohli systém využívat. Další možností, jak zpřístupnit používání aplikace všem uživatelům, je vytvoření webové aplikace pro komunikaci se systémem. Aplikace by se spouštěla přímo ve webovém prohlížeči mobilního telefonu a systém by tak byl multiplatformní. Některé části systému a mobilní aplikace jsou navrženy tak, aby bylo možné funkcionalitu do budoucna dále rozšiřovat. Struktura uložených dat v databázi umožňuje přidávat další parkoviště, stejně tak i automatická detekce dostupnosti parkoviště umožňuje zobrazovat notifikace z vícero parkovišť současně. Systém by tak po několika úpravách mohli využít například provozovatelé parkovišť. Vzhledem k nízkým nákladům popsaným v kapitole 7 by navrhovaný systém mohl nejen vyřešit řidičům problémy s hledáním volných parkovacích míst a zlepšit tak dopravní situaci ve městech, ale na druhou stranu by omezil problémy s neplatiči parkovného, kteří na otevřených parkovištích s parkovacími automaty často parkují bez zaplaceného poplatku za parkování.

9. Reference

[1] „V ČR je 5,59 mil. osobních aut". [Online]. Dostupné z: http://www.uamk.cz/aktuality/2186-v-cr-je-5-59-mil-osobnich-aut. [Viděno: 06-lis-2018].

[2] M.-F. Tsai, Y. C. Kiong, a A. Sinn, „Smart service relying on Internet of Things technology in parking systems", *J. Supercomput.*, roč. 74, č. 9, s. 4315–4338, zář. 2018.

[3] A. Khanna a R. Anand, „IoT based smart parking system", in *2016 International Conference on Internet of Things and Applications (IOTA)*, Pune, India, 2016, s. 266–270.

[4] C. Roman, R. Liao, P. Ball, S. Ou, a M. de Heaver, „Detecting On-Street Parking Spaces in Smart Cities: Performance Evaluation of Fixed and Mobile Sensing Systems", *Ieee Trans. Intell. Transp. Syst.*, roč. 19, č. 7, s. 2234–2245, čvc. 2018.

[5] „Smart Parking CZ". [Online]. Dostupné z: http://www.smart-parking.cz/. [Viděno: 12-lis-2018].

[6] „CITIQ s.r.o. - detektory pro městskou dopravu a parkování". [Online]. Dostupné z: http://www.citiq.cz/. [Viděno: 12-lis-2018].

[7] „MICRORISC". [Online]. Dostupné z: https://www.microrisc.com/cs. [Viděno: 11-bře-2019].

[8] „IQRF Alliance". [Online]. Dostupné z: https://www.iqrfalliance.org/case-studies/. [Viděno: 11-bře-2019].

[9] A. Kabelová a L. Dostálek, *Velký průvodce protokoly TCP/IP a systémem DNS*. Brno: Computer Press, 2008.

[10] „MPU-9250 | TDK". [Online]. Dostupné z: https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/. [Viděno: 14-úno-2019].

[11] „IQRF - Technology for wireless". [Online]. Dostupné z: https://www.iqrf.org/. [Viděno: 13-lis-2018].

[12] IQRF Tech s.r.o, „IQRF DPA Framework Technical Guide". 10-2018.

[13] „DK-EVAL-04A User's guide". IQRF Tech s.r.o., 09-lis-2017.

[14] „UP Specifications – UP Bridge the Gap". [Online]. Dostupné z: https://up-board.org/up/specifications/. [Viděno: 13-lis-2018].

[15] „install/up-board/GW-SbS-INSTALL.md · master · IQRF Alliance / IoT Starter kit · GitLab". [Online]. Dostupné z: https://gitlab.iqrf.org/alliance/iot-starter-kit/blob/master/install/up-board/GW-SbS-INSTALL.md. [Viděno: 04-bře-2019].

[16] „NodeRED". [Online]. Dostupné z: https://nodered.org/. [Viděno: 09-pro-2018].

[17] „Firebase", *Firebase*. [Online]. Dostupné z: https://firebase.google.com/. [Viděno: 09-pro-2018].

[18] „Firebase: krátké seznámení - Zdroják". [Online]. Dostupné z: https://www.zdrojak.cz/clanky/firebase-kratke-seznameni/. [Viděno: 09-pro-2018].

[19] „Create and monitor geofences | Android Developers". [Online]. Dostupné z: https://developer.android.com/training/location/geofencing#java. [Viděno: 12-led-2019].

[20] „Google Maps Intents for Android | Maps URLs | Google Developers". [Online]. Dostupné z: https://developers.google.com/maps/documentation/urls/android-intents. [Viděno: 12-úno-2019].

10. Přílohy

10.1 Mobilní aplikace

10.1.1 MainActivity.java

```
package cz.uhk.dianja1.smartparking;


import android.Manifest;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Settings;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AlertDialog;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import android.provider.Settings.System;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.Geofence;
import com.google.android.gms.location.GeofencingClient;
import com.google.android.gms.location.GeofencingRequest;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import static android.widget.Toast.*;

public class MainActivity extends Activity implements
        com.google.android.gms.location.LocationListener,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
        ResultCallback<Status> {

    protected ArrayList<Geofence> mGeofenceList;
    protected GoogleApiClient mGoogleApiClient;
    private Button mAddGeofencesButton;
    private GeofencingClient mGeofencingClient;
    private PendingIntent mGeofencePendingIntent;
    private FusedLocationProviderClient mFusedLocationProviderClient;
    private String androidID;
    private FirebaseDatabase database = FirebaseDatabase.getInstance();
    private ArrayList<Reservation> reservList = new ArrayList();
    private long countRes = 0;
    private boolean resIndicator = false;
    private boolean carIndicator = false;
    private boolean fullIndicator = false;
    private String places;
    private String place;
    private double lat;
    private double lng;
    private double distance;
    protected LocationRequest mLocationRequest;
    private GeofenceTransitionsIntentService geo;
    private TextView Fplaces;
    private int LOCATION_PERMISSION_CODE = 1;
    private DatabaseReference myRefCar = database.getReference("parkings/parking1/carInPark");
    private DatabaseReference myRefGate = database.getReference("parkings/parking1/gate");
    private DatabaseReference myRefGateID = database.getReference("parkings/parking1/gateID");
    private DatabaseReference myRefRes = database.getReference("parkings/parking1/reservations");
    private DatabaseReference myRefResID =database.getReference("parkings/parking1/reservationID");
    private DatabaseReference myRefCount = database.getReference("parkings/parking1/countRes");
    private DatabaseReference myRefPlaces = database.getReference("parkings/parking1/places");
    private DatabaseReference myRefFplaces = database.getReference("parkings/parking1/freePlaces");
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    buildGoogleApiClient();
    setContentView(R.layout.activity_main);
    mAddGeofencesButton = findViewById(R.id.add_geofences_button);
    mGeofenceList = new ArrayList<Geofence>();
    Fplaces = findViewById(R.id.tvPlaces);
    androidID = System.getString(this.getContentResolver(), Settings.Secure.ANDROID_ID);
    mGeofencingClient = LocationServices.getGeofencingClient(this);
    readFreePlaces();
    populateGeofenceList();
    readReservations();
    readCarInPark();
    checkPlaces();
}

public void distance() {
    double rlat1 = Math.toRadians(lat);
    double rlat2 = Math.toRadians(50.475367);
    double rlng1 = Math.toRadians(lng);
    double rlng2 = Math.toRadians(16.179489);

    distance = Math.acos(
            Math.cos(rlat1) * Math.cos(rlng1) * Math.cos(rlat2) * Math.cos(rlng2) +
                    Math.cos(rlat1) * Math.sin(rlng1) * Math.cos(rlat2) * Math.sin(rlng2) +
                    Math.sin(rlat1) * Math.sin(rlat2)) * 6372.795;

    distance = distance * 1000;
}

public void readFreePlaces(){
    fullIndicator = false;
    myRefFplaces.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            places = dataSnapshot.getValue().toString();
            int pl = Integer.parseInt(places);
            int cr = (int) (pl - countRes);
            if(cr == 0) {fullIndicator = true;}
            Fplaces.setText(String.valueOf(cr));
        }
```

```
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            places = "Error";
        }
    });
}

public void Info(View v){
    Intent in = new Intent(this, InfoActivity.class);
    startActivity(in);
}

public void readReservations() {
        myRefRes.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot snapshot) {
            countRes = snapshot.getChildrenCount();
            updateCountReservation();
            resIndicator = false;
            for (DataSnapshot postSnapshot: snapshot.getChildren()) {
                Reservation post = postSnapshot.getValue(Reservation.class);
                String resCheck = post.getAndroidID();
                if(resCheck.equals(androidID)){resIndicator = true;}
            }
        }

        @Override
        public void onCancelled(DatabaseError error) {
            showText("Failed to read value.");
        }
    });
}

public void updateCountReservation(){
    myRefCount.setValue(countRes);
    readFreePlaces();

}
public void readCarInPark(){
    myRefCar.addValueEventListener(new ValueEventListener() {

        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            Date d = new Date();
            carIndicator = false;
```

```
            for (DataSnapshot postSnapshot: dataSnapshot.getChildren()) {
                CarInPark post = postSnapshot.getValue( CarInPark.class);
                String carCheck = post.getAndroidID();
                if(carCheck.equals(androidID)){carIndicator = true;}

            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
        }
    });
}

public void populateGeofenceList() {
    for (Map.Entry<String, LatLng> entry : Constants.LANDMARKS.entrySet()) {
        mGeofenceList.add(new Geofence.Builder()
                .setRequestId(entry.getKey())
                .setCircularRegion(
                        entry.getValue().latitude,
                        entry.getValue().longitude,
                        Constants.GEOFENCE_RADIUS_IN_METERS
                )
                .setExpirationDuration(Geofence.NEVER_EXPIRE)
                .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER)
                .build());
    }
}

private void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    createLocationRequest();
}

private void createLocationRequest() {
    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(5000);
    mLocationRequest.setFastestInterval(1000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

```
    protected void startLocationUpdates() {
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {

            requestLocationPermission();
        } else {

            LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
        }
    }




    @Override
    protected void onStart() {
        super.onStart();
        if (!mGoogleApiClient.isConnecting() || !mGoogleApiClient.isConnected()) {
            mGoogleApiClient.connect();
        }
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mGoogleApiClient.isConnecting() || mGoogleApiClient.isConnected()) {
            mGoogleApiClient.disconnect();
        }
    }



    public void addGeofencesButtonHandler(View view) {
        addGeofences();
    }

    public void addReservationButtonHandler(View view) {

        if(resIndicator || fullIndicator || carIndicator){
            if(resIndicator){showText("You already have reservation!!!");}
            if(fullIndicator){showText("Car park is full!!!");}
            if(carIndicator){showText("You are already on car park!!!");}
            return;
        }
```

```
    SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm");
    Date date = new Date();
    String time = dateFormat.format(date);
    String id = myRefRes.push().getKey();

    Reservation reservations = new Reservation(id, androidID, time );
    myRefRes.child(id).setValue(reservations);
    showText("Your reservation was accepted");
}

public void openGateButtonHandler(View view){
    distance();
    if(!carIndicator){
        if(!fullIndicator || resIndicator){
            if(distance < 15) {
                SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm");
                Date date = new Date();
                String time = dateFormat.format(date);
                String id = myRefCar.push().getKey();
                CarInPark carInPark = new CarInPark(id, androidID, time);
                myRefCar.child(id).setValue(carInPark);
                myRefGate.setValue(1);
                myRefResID.setValue(androidID);
                showText("Entrance gate is opening");
                Intent in = new Intent(this, InfoActivity.class);
                startActivity(in);
            } else{
                showText("You are too far from carpark!!!");
            }
        }
        else{ showText("Car park is full!!!");}
    }

    else {
        myRefGateID.setValue(androidID);
        showText("Exit gate is opening");}
}

public void startNavigationButtonHandler(View view){
    Uri gmmIntentUri = Uri.parse("google.navigation:q= 50.475367, 16.179489");
    Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
    mapIntent.setPackage("com.google.android.apps.maps");
    startActivity(mapIntent);

}
```

```
    public void checkPlaces(){
        myRefPlaces.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                place = dataSnapshot.getValue().toString();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });

    }

    private void requestLocationPermission(){
        if(ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.ACCESS_FINE_LOCATION)){

            new AlertDialog.Builder(this)
                    .setTitle("Permission needed")
                    .setMessage("We need this permission for automatic car park detection and " +
                            "for check your position when entrance gate is opening")
                    .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialogInterface, int i) {
                        ActivityCompat.requestPermissions(MainActivity.this, new String[]
                    {Manifest.permission.ACCESS_FINE_LOCATION}, LOCATION_PERMISSION_CODE);
                        }
                    })
                    .setNegativeButton("CANCEL", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialogInterface, int i) {
                            dialogInterface.dismiss();
                        }
                    })
                    .create().show();

        } else {
            ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, LOCATION_PERMISSION_CODE);
        }
    }
```

```
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        if (requestCode == LOCATION_PERMISSION_CODE) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                showText("Permission GRANTED");
            } else {
                showText("Permission DENIED");
            }
        }
    }


    private void addGeofences(){
        if (ContextCompat.checkSelfPermission( this,
android.Manifest.permission.ACCESS_FINE_LOCATION )
                != PackageManager.PERMISSION_GRANTED &&
                ContextCompat.checkSelfPermission( this,
android.Manifest.permission.ACCESS_COARSE_LOCATION)
                        != PackageManager.PERMISSION_GRANTED) {


            requestLocationPermission();




        }
        //showText("You have allready granted this permission");
        mGeofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
                .addOnSuccessListener(this, new OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {
                        // Geofences added
                        showText("Car park detection was enabled");
                    }
                })
                .addOnFailureListener(this, new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        // Failed to add geofences
                        showText("Error");
                    }
                });
    }


    private void showText(CharSequence text) {
        makeText(this, text, Toast.LENGTH_SHORT).show();


    }
```

```
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(mGeofenceList);
    return builder.build();
}

private PendingIntent getGeofencePendingIntent() {
    if (mGeofencePendingIntent != null) {
        mGeofencePendingIntent.cancel();
        //return mGeofencePendingIntent;
    }

    Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);
    intent.putExtra("Places", places);

    mGeofencePendingIntent = PendingIntent.getService(this, 0, intent,
            PendingIntent.FLAG_UPDATE_CURRENT);
    return mGeofencePendingIntent;
}


@Override
public void onConnected(Bundle connectionHint) {
    startLocationUpdates();
}

@Override
public void onConnectionFailed(ConnectionResult result) {

}

@Override
public void onConnectionSuspended(int cause) {
    mGoogleApiClient.connect();
}


@Override
public void onResult(@NonNull Status status) {

}
```

```
    @Override
    public void onLocationChanged(Location location) {
        lat = location.getLatitude();
        lng = location.getLongitude();

    }
}
```

10.1.2 InfoActivity.java

```
package cz.uhk.dianja1.smartparking;
import android.graphics.Color;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class InfoActivity extends AppCompatActivity {
    private int[] NUMBERS = new int[8];
    private String[] OCCUPANCY = new String[16];
    private String places;
    private FirebaseDatabase database = FirebaseDatabase.getInstance();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_info);
        final ListView listView = findViewById(R.id.listView);
        final CustomAdapter customAdapter = new CustomAdapter();
        DatabaseReference refPlaces = database.getReference("parkings/parking1/places");
        refPlaces.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                places = dataSnapshot.getValue().toString();
                getOccupancy();
                listView.setAdapter(customAdapter);

            }
```

```
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
            }
        });
    }

    class CustomAdapter extends BaseAdapter{

        @Override
        public int getCount() {
            return NUMBERS.length;
        }

        @Override
        public Object getItem(int i) {
            return null;
        }

        @Override
        public long getItemId(int i) {
            return 0;
        }

        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
            view = getLayoutInflater().inflate(R.layout.customlayout,null);

            TextView tvNumbers = view.findViewById(R.id.tvNumbers);
            TextView tvOccupancy = view.findViewById(R.id.tvOccupancy);
            TextView tvOccupancy1 = view.findViewById(R.id.tvOccupancy1);

            tvNumbers.setText(String.valueOf(NUMBERS[i]));
            tvOccupancy.setText(String.valueOf(OCCUPANCY[i]));
            tvOccupancy1.setText(String.valueOf(OCCUPANCY[i+8]));
            if(OCCUPANCY[i].equals("FREE")){tvOccupancy.setTextColor(Color.parseColor("#00ff00"));}
if(OCCUPANCY[i].equals("OCCUPIED")){tvOccupancy.setTextColor(Color.parseColor("#ff0000"));}
if(OCCUPANCY[i+8].equals("FREE")){tvOccupancy1.setTextColor(Color.parseColor("#00ff00"));}
if(OCCUPANCY[i+8].equals("OCCUPIED")){tvOccupancy1.setTextColor(Color.parseColor("#ff0000"));}
            return view;
        }
    }
```

```
    public void getOccupancy(){
        for(int i=0;i<8;i++){
            System.out.println(places.substring(i,i+1));
            if(places.substring(i,i+1).equals("0")){OCCUPANCY[i]="FREE";}
            if(places.substring(i,i+1).equals("1")){OCCUPANCY[i]="OCCUPIED";}
            if(places.substring(i+8,i+9).equals("0")){OCCUPANCY[i+8]="FREE";}
            if(places.substring(i+8,i+9).equals("1")){OCCUPANCY[i+8]="OCCUPIED";}
            NUMBERS[i] = 8 - i;
        }
    }
}
```

10.1.3 Constants.java

```
package cz.uhk.dianja1.smartparking;
import com.google.android.gms.maps.model.LatLng;
import java.util.HashMap;

public class Constants {
    public static final long GEOFENCE_EXPIRATION_IN_MILLISECONDS = 12 * 60 * 60 * 1000;
    public static final float GEOFENCE_RADIUS_IN_METERS = 1000;
    public static final HashMap<String, LatLng> LANDMARKS = new HashMap<String, LatLng>();
    static {
        // Parking 1
        LANDMARKS.put("Parking 1", new LatLng(50.475367, 16.179489));
    }
}
```

10.1.4 CarInPark.java

```
package cz.uhk.dianja1.smartparking;
public class CarInPark {
    private String id;
    private String androidID;
    private String time;

    public CarInPark(String id, String androidID, String time) {
        this.id = id;
        this.androidID = androidID;
        this.time = time;
    }

    public CarInPark() {
    }

    public String getId() {
        return id;
    }
```

```
    public void setId(String id) {
        this.id = id;
    }

    public String getAndroidID() {
        return androidID;
    }

    public void setAndroidID(String androidID) {
        this.androidID = androidID;
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }
}
```

10.1.5 Reservation.java

```
package cz.uhk.dianja1.smartparking;
public class Reservation {
    private String id;
    private String androidID;
    private String time;

    public Reservation() {
    }

    public Reservation(String id, String androidID, String time) {
        this.id = id;
        this.androidID = androidID;
        this.time = time;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
```

```
    public String getAndroidID() {
        return androidID;
    }

    public void setAndroidID(String androidID) {
        this.androidID = androidID;
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }
}
```

10.1.6 GeofenceTransitionsIntentService.java

```
package cz.uhk.dianja1.smartparking;
import android.app.IntentService;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.TaskStackBuilder;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v4.app.NotificationCompat;
import android.text.TextUtils;
import android.util.Log;
import android.widget.Toast;
import com.google.android.gms.location.Geofence;
import com.google.android.gms.location.GeofenceStatusCodes;
import com.google.android.gms.location.GeofencingEvent;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
import java.util.List;
import static android.widget.Toast.makeText;
```

```
public class GeofenceTransitionsIntentService extends IntentService {
    protected static final String TAG = "GeofenceTransitionsIS";
    private final String CHANNEL_ID = "personal_notifications";
    private final int NOTIFICATION_ID = 001;
    private int k;
    private String places;
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    public GeofenceTransitionsIntentService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {

        GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
        if (geofencingEvent.hasError()) {
              Log.e(TAG, "GeofencingEvent Error: " + geofencingEvent.getErrorCode());
            return;
        }

        String description = getGeofenceTransitionDetails(geofencingEvent);
           places = intent.getStringExtra("Places");
            sendNotification(description);
 }

    private static String getGeofenceTransitionDetails(GeofencingEvent event) {
        String transitionString =
                GeofenceStatusCodes.getStatusCodeString(event.getGeofenceTransition());
        List triggeringIDs = new ArrayList();
        for (Geofence geofence : event.getTriggeringGeofences()) {
            triggeringIDs.add(geofence.getRequestId());
        }
        return String.format("%s: %s", transitionString, TextUtils.join(", ", triggeringIDs));
    }

    private void createNotificationChannel(){
        if(Build.VERSION.SDK_INT>= Build.VERSION_CODES.O){
            CharSequence name = "Personal notifications";
            String descrption = "Include all personal notifications";
            int importance = NotificationManager.IMPORTANCE_DEFAULT;

            NotificationChannel notificationChannel = new NotificationChannel(CHANNEL_ID, name,
importance);
            notificationChannel.setDescription(descrption);
```

```
            NotificationManager notificationMnager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
            notificationMnager.createNotificationChannel(notificationChannel);
        }
    }

    private void sendNotification(String notificationDetails) {

        createNotificationChannel();

        // Create an explicit content Intent that starts MainActivity.
        Intent notificationIntent = new Intent(getApplicationContext(), MainActivity.class);

        // Get a PendingIntent containing the entire back stack.
        TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
        stackBuilder.addParentStack(MainActivity.class).addNextIntent(notificationIntent);
        PendingIntent notificationPendingIntent =
                stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);

        // Get a notification builder that's compatible with platform versions >= 4
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID);

        for(int i=0; i < notificationDetails.length(); i++){
            String sep = notificationDetails.substring(i,i+1);
            if(sep.equals(":")){
                k = i;
            }
        }

        int l = notificationDetails.length();
        builder.setColor(Color.rgb(0,104,177))
                .setContentTitle(notificationDetails.substring(k+2, l))
                .setSmallIcon(R.drawable.ic_stat_local_parking1)
                .setContentText( "Number of free places: " + places)
                .setContentIntent(notificationPendingIntent)
                .setAutoCancel(false);

        // Fire and notify the built Notification.
        NotificationManager notificationManager =
                (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(0, builder.build());
    }
}
```

10.2 Arduino

```
#include <MPU9250_asukiaaa.h>
#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 26
#define SCL_PIN 25
#endif

MPU9250 mySensor;

uint8_t sensorId;
float aX, aY, aZ, aSqrt, gX, gY, gZ, mDirection, mX, mY, mZ;
float normalMx, normalMy, normalMz;
float deltaMx, deltaMy, deltaMz;
boolean start = true;
boolean detect = false;
int carDetection = 0;

void setup() {
  while(!Serial);
  Serial.begin(115200);
  Serial.println("started");

#ifdef _ESP32_HAL_I2C_H_ // For ESP32
  Wire.begin(SDA_PIN, SCL_PIN); // SDA, SCL
#else
  Wire.begin();
#endif

  mySensor.setWire(&Wire);
  mySensor.beginMag();

  sensorId = mySensor.readId();
  pinMode(3,OUTPUT);
}

void loop() {
  if(start){delay(5000);}
  mySensor.magUpdate();
  mX = mySensor.magX();
  mY = mySensor.magY();
  mZ = mySensor.magZ();
  if(start){initSensor(); start = false;}
  Serial.println("magX: " + String(mX));
  Serial.println("maxY: " + String(mY));
  Serial.println("magZ: " + String(mZ));
```

```
  testSensor();
  Serial.println(""); // Add an empty line
  delay(500);
}

void initSensor(){
  normalMx = mX;
  normalMy = mY;
  normalMz = mZ;
}

void testSensor(){
  deltaMx = abs(normalMx-mX);
  deltaMy = abs(normalMy-mY);
  deltaMz = abs(normalMz-mZ);
  Serial.println("deltaMx: " + String(deltaMx));
  Serial.println("deltaMy: " + String(deltaMy));
  Serial.println("deltaMz: " + String(deltaMz));
  if(deltaMx > 7 || deltaMy > 7 || deltaMz > 7){
    Serial.println("Car is present");
    digitalWrite(3,HIGH);
    detect = true;}
    else {Serial.println("Car is not present");
    digitalWrite(3,LOW);
    if(detect){start = true; detect = false;}}

}
```

10.3 NodeRED JavaScript

10.3.1 Request

```
var data={
type: "raw",
request: {
nadr: "0x0000",
pnum: "0x0D",
pcmd: "0x00",
hwpid: "0xFFFF",
pdata: "0x400000",
},
timeout: 1000
}
msg.payload=data;
return msg;
```

10.3.2 GateEntranceOpen

```
var data={
type: "raw",
request: {
nadr: "0x0000",
pnum: "0x06",
pcmd: "0x01",
hwpid: "0xFFFF",
pdata: "",
},
timeout: 1000
}
msg.payload=data;
return msg;
```

10.3.3 GateEntranceClose

```
var data={
type: "raw",
request: {
nadr: "0x0000",
pnum: "0x06",
pcmd: "0x00",
hwpid: "0xFFFF",
pdata: "",
},
timeout: 1000
}
msg.payload=data;
return msg;
```

10.3.4 GateExitOpen

```
var data={
type: "raw",
request: {
nadr: "0x0000",
pnum: "0x07",
pcmd: "0x01",
hwpid: "0xFFFF",
pdata: "",
},
timeout: 1000
}
msg.payload=data;
return msg;
```

10.3.5 GateExitClose

```
var data={
type: "raw",
request: {
nadr: "0x0000",
pnum: "0x07",
pcmd: "0x00",
hwpid: "0xFFFF",
pdata: "",
},
timeout: 1000
}
msg.payload=data;
return msg;
```

10.3.6 JSON

```
out = ('000'+Number(msg.payload.request.nadr).toString(16)).slice(-2);
out += "."+(('000'+Number(msg.payload.request.nadr).toString(16)).slice(-4)).substr(0,2);
out += "."+('0'+Number(msg.payload.request.pnum).toString(16)).slice(-2);
out += "."+('0'+Number(msg.payload.request.pcmd).toString(16)).slice(-2);
out += "."+('000'+Number(msg.payload.request.hwpid).toString(16)).slice(-2);
out += "."+(('000'+Number(msg.payload.request.hwpid).toString(16)).slice(-4)).substr(0,2);
if (typeof msg.payload.request.pdata !== 'undefined' && msg.payload.request.pdata !== null)
{
len = msg.payload.request.pdata.length;
count = -len;
for(i=len;i>2;i=i-2)
{
count = count + 2;
out += "."+(('000'+Number(msg.payload.request.pdata).toString(16)).slice(count)).substr(0,2);
}
}
msg.payload= {ctype:"dpa", type: msg.payload.type, request: out, timeout: msg.payload.timeout};
return msg;
```

10.3.7 IQRFtoJSON

```
var response=msg.payload.response.split(".");
var request=msg.payload.request.split(".");
var data={
type: "",
request: {
nadr: "",
pnum: "",
pcmd: "",
hwpid: ""
},
```

```
response: {
nadr: "",
pnum: "",
pcmd: "",
hwpid: "",
errn: "",
dpa: "",
pdata: "",
},
timeout: "",
result: "",
time: new Date()
}
data.type = msg.payload.type;
data.request.nadr=parseInt(request[0], 16) + (parseInt(request[1], 16)*256);
data.request.pnum=parseInt(request[2], 16);
data.request.pcmd=parseInt(request[3], 16);
data.request.hwpid=parseInt(request[4], 16) + (parseInt(request[5], 16)*256);
data.response.nadr=parseInt(response[0], 16) + (parseInt(response[1], 16)*256);
data.response.pnum=parseInt(response[2], 16);
data.response.pcmd=parseInt(response[3], 16);
data.response.hwpid=parseInt(response[4], 16) + (parseInt(response[5], 16)*256);
data.response.errn=parseInt(response[6], 16);
data.response.dpa=parseInt(response[7], 16);
data.response.pdata=msg.payload.response.slice(24).trim();
data.timeout=msg.payload.timeout;
data.result=msg.payload.result;
msg.payload=data;
return msg;
```

10.3.8 GetIO

```
var res=msg.payload.response.pdata.split(".");
var inp = "";
var z = 1;
for(j=0;j<z+1;j++){
var io = parseInt(res[33 + j],16);
for(i=0;i<8;i++){
var m = io%2;
var io = parseInt(io/2);
inp = m + inp;
}
}
msg.payload = inp;
return msg;
```

10.3.9 GetNumberOfFreePlaces

```
var res=msg.payload.response.pdata.split(".");
var z = 1;
var c = 0;
for(j=0;j<z+1;j++){
var io = parseInt(res[33 + j],16);
for(i=0;i<8;i++){
var m = io%2;
if(m===0){c++;}
var io = parseInt(io/2);
}
}
msg.payload = c;
return msg;
```

10.3.10 GlobalContext

```
var gateID = msg.payload;
global.set("gateID", gateID);
return;
```

10.3.11 FirebaseConvert

```
var response = msg.payload;
var gateID = global.get("gateID");
var data = "";
var data1 = response.split("{").toString();
var l = data1.length;
for(i=0;i<l;i++){
var s = data1.substring(i,i+1);
if(s == "\""){}
else if(s == "{"){}
else if(s == "}"){}
else {data = data + s;}
}
var allmsg = data.split(",-");
var al = allmsg.length;
for(i=1;i<al;i++){
var data3 = allmsg[i].split(",");
var dl = data3[1].length;
var data2 = data3[1].substring(10,dl);
var cesta = "parkings/parking1/carInPark/" + data3[2].substring(3,data3[2].length);
if(data2==gateID){
//node.send({payload:cesta});
node.send({childpath:cesta});
}
}
return;
```

10.3.12 FirebaseConvert - Reservations

```
var response = msg.payload;
var data = "";
var nic = 1;
var data1 = response.split("{").toString();
var l = data1.length;
for(i=0;i<l;i++){
var s = data1.substring(i,i+1);
if(s == "\""){}
else if(s == "{"){}
else if(s == "}"){}
else {data = data + s;}
}
var allmsg = data.split(",-");
l = allmsg.length;
for(i=1;i<l;i++){
var data1 = allmsg[i].split(":");
var fh = parseInt(data1[4]);
var fm = parseInt(data1[5]);
var date = new Date();
var h = date.getHours()+1;
var m = date.getMinutes();
var delta = (h*60+m)-(fh*60+fm);
if(delta > 60){
var cesta = "parkings/parking1/reservations/-" + data1[0];
node.send({childpath:cesta});
}
else{
node.send({payload:nic});
}
}
return;
```

10.3.13 GlobalContext

```
var reservationID = msg.payload;
global.set("reservationID", reservationID);
msg.payload = 1;
return msg;
```

10.3.14 ResesrvationsConvert

```
var response = msg.payload;
var reservationID = global.get("reservationID");
var data = "";
var data1 = response.split("{").toString();
var l = data1.length;
```

```
for(i=0;i<l;i++){
var s = data1.substring(i,i+1);
if(s == "\""){}
else if(s == "{"){}
else if(s == "}"){}
else {data = data + s;}
}
var allmsg = data.split(",-");
var al = allmsg.length;
for(i=1;i<al;i++){
var data3 = allmsg[i].split(",");
var androidID = data3[1].substring(10,data3[1].length);
var path = "parkings/parking1/reservations/" +
data3[2].substring(3,data3[2].length);
if(androidID == reservationID){
node.send({childpath:path});
}
else {node.send({payload:1});}
}
return;
```