VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCIA PRÍTOMNOSTI SUBJEKTOV V STRÁŽENOM PRIESTORE

DETECTION OF SUBJECTS PRESENCE IN PROTECTED AREA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM MÚDRY**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÁCLAV ŠIMEK**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Múdry Adam**
Program: Informační technologie
Název: **Detekce přítomnosti subjektů ve střeženém prostoru**
Detection of Subjects Presence in Protected Area
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s problematikou zabezpečovacích systémů pro domácí použití. Zpracujte přehled existujících řešení a stručně shrňte jejich klíčové vlastnosti.
2. Proveďte analýzu možností detekce přítomnosti osob ve střeženém prostoru z pohledu využití vhodných senzorů i algoritmů na bázi umělé inteligence.
3. Podrobně se zabývejte vestavěnou platformou ESP32. Zaměřte se na její technické parametry, principy tvorby aplikací a dostupné vývojové nástroje.
4. Navrhněte architekturu zabezpečovacího systému umožňujícího detekci přítomnosti osob ve střeženém prostoru. Identifikujte klíčové funkční celky a zvolte vhodné prostředky pro jejich realizaci.
5. S využitím platformy ESP32 proveďte realizaci potřebných technických prvků zabezpečovacího systému dle bodu 4 zadání. Vytvořte k nim obslužný firmware.
6. Implementujte softwarovou část zabezpečovacího systému zodpovědnou za zpracování senzorických dat a jejich vyhodnocení.
7. Vhodným způsobem demonstrujte funkčnost realizovaného řešení jako celku. Následně zhodnoťte dosažené výsledky a pokuste se navrhnout případná rozšíření či vylepšení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 4 zadání.

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

Vedoucí práce: **Šimek Václav, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cieľom tejto diplomovej práce bolo navrhnúť a implementovať systém slúžiaci na detekciu prítomnosti subjektu v stráženom priestore pomocou vstavanej platformy ESP32. Navrhnutý systém je schopný využívať viacero rôznych senzorov, umožňujúcich detekciu pohybu, zvuku, snímanie fotografii a podobne. Riešenie práce bolo realizované taktiež s pomocou platformy Raspberry Pi ako ovládacieho serveru a voliteľne využíva aj cloudové služby pre väčšiu flexibilitu nasadenia.

Abstract

The aim of this thesis was to design and implement a system for detecting the presence of the subject in the protected area using the embedded platform ESP32. The proposed system is able to use several different sensors, allowing the detection of motion, sound, allowing to take photos, etc. The solution was also implemented with the help of the Raspberry Pi platform as a control server and optionally uses cloud services for greater deployment flexibility.

Kľúčové slová

detekcia prítomnosti, vstavané systémy, ESP32, Raspberry Pi, WiFi, automatizácia, cloudové služby, Internet vecí, IoT, zabezpečovací systém

Keywords

presence detection, embedded systems, ESP32, Raspberry Pi, WiFi, automatization, cloud services, Internet of Things, IoT, security system

Citácia

MÚDRY, Adam. *Detekcia prítomnosti subjektov v stráženom priestore*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Detekcia prítomnosti subjektov v stráženom priestore

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Václava Šimka. Dalšie informácie mi poskytol pán Martin Vychodil. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Adam Múdry
19. mája 2021

Poďakovanie

Rád by som sa poďakoval vedúcemu mojej bakalárskej práce pánovi Václavovi Šimkovi a externému konzultantovi pánovi Martinovi Vychodilovi za ich odbornú pomoc, vedenie, pripomienky a čas strávený nad konzultáciami, ktoré mi poskytli počas riešenia tejto práce.

Obsah

Kapitola 1

Úvod

Svet, v ktorom od počiatku žijeme je spojený aj s negatívnymi javmi, medzi ktoré patria rôzne hrozby, nebezpečenstvá, škodlivé vplyvy, krádeže a podobne. Polícia každý deň zaznamenáva a rieši množstvo nových kriminálnych prípadov, čo je časovo, fyzicky a administratívne náročné. Možnou príčinou tejto skutočnosti je, že majetok alebo iné hodnoty poškodeného neboli v týchto prípadoch chránené vôbec, alebo zabezpečovací systém nebol dostatočne spoľahlivý pred odcudzením, poškodením, zničením alebo pred iným spôsobom narušenia.

Ochrana majetku je proces navodenia stavu bezpečnosti (istoty) s využitím ochranných opatrení za účelom znemožnenia akejkoľvek činnosti alebo udalosti, ktorá je v rozpore so záujmami vlastníka tohto majetku. Tento proces je úzko spojený s bezpečnosťou. Vo väzbe na ochranu hmotného majetku hovoríme o fyzickej bezpečnosti a vo väzbe na ochranu nehmotného majetku hovoríme o ochrane informačnej bezpečnosti.

Na trhu v Slovenskej a Českej republike existuje množstvo zabezpečovacích systémov, ktoré poskytujú rôzne štandardy ochrany. Každý takýto systém je tvorený sústavou elektrických, elektronických, mechanických alebo iných súčiastok, ktoré buď zabraňujú vstupu neželanej osoby, zvieraťa, dopravného prostriedku do stráženého priestoru alebo na chránené miesto, prípadne detegujú a indikujú prítomnosť, vstup alebo pokus o vstup narušiteľa do stráženého priestoru a následne signalizujú narušenie na určitom mieste stráženého objektu alebo priestoru. Spoľahlivé zabezpečenie by malo byť v záujme každého majiteľa nehnuteľnosti.

Táto bakalárska práca sa venuje detekcii narušenia stráženého priestoru s identifikáciou narušenej časti priestoru (a snahou identifikácie narušiteľa) a následným oznámením tejto informácie používateľovi. Zároveň je súčasťou väčšieho riešenia skladajúceho sa z čiastkových prác troch študentov VUT FIT v Brne, ktorých zadávateľom je firma Espressif.

Teoretická časť práce (kapitola 2) sa zaoberá prehľadom hardvérových a softvérových technológií, ktoré sú v práci využívané a terminológiou s nimi spojenou. Ich znalosť je potrebná pre pochopenie ďalších častí práce. Návrh systému obsahujúci ESP32 zariadenia a senzory je popísaný v kapitole 3, jeho implementácia v kapitole 4 a testovanie funkčnosti v kapitole 5. Diskusia (kapitola 6) zhodnocuje problémy spojené s tvorbou práce a navrhuje jej možné vylepšenia.

Kapitola 2

Použité technológie

V tejto kapitole uvediem a stručne vysvetlím technológie a pojmy dôležité na pochopenie ďalších kapitol mojej bakalárskej práce, kde ich využívam. Kvôli lepšej prehľadnosti je kapitola rozdelená do dvoch sekcií: hardvérové technológie a softvérové technológie. Vysvetlím termíny spojené s vstavanými systémami, konceptu Internetu vecí a spomeniem hardvér a softvér, ktorý budem v práci využívať.

2.1 Hardvérové technológie

2.1.1 Vstavaný systém

Vstavaný systém (ang. *Embedded system*) by sa dal charakterizovať ako kombinácia počítačového hardvéru a softvéru (nazývaný tiež *firmvér* [4]), prípadne prídavných mechanických alebo iných častí, navrhnutý na vykonávanie určenej špecifickej funkcie. V niektorých prípadoch sú vstavané systémy súčasťou väčšieho systému alebo produktu, ako napríklad v prípade protiblokovacieho brzdového systému v automobile. [2]

Vstavané systémy poháňajú veľký rozsah zariadení, ako sú napríklad digitálne hodinky, MP3 prehrávače, práčky, semafory a mnohé iné.

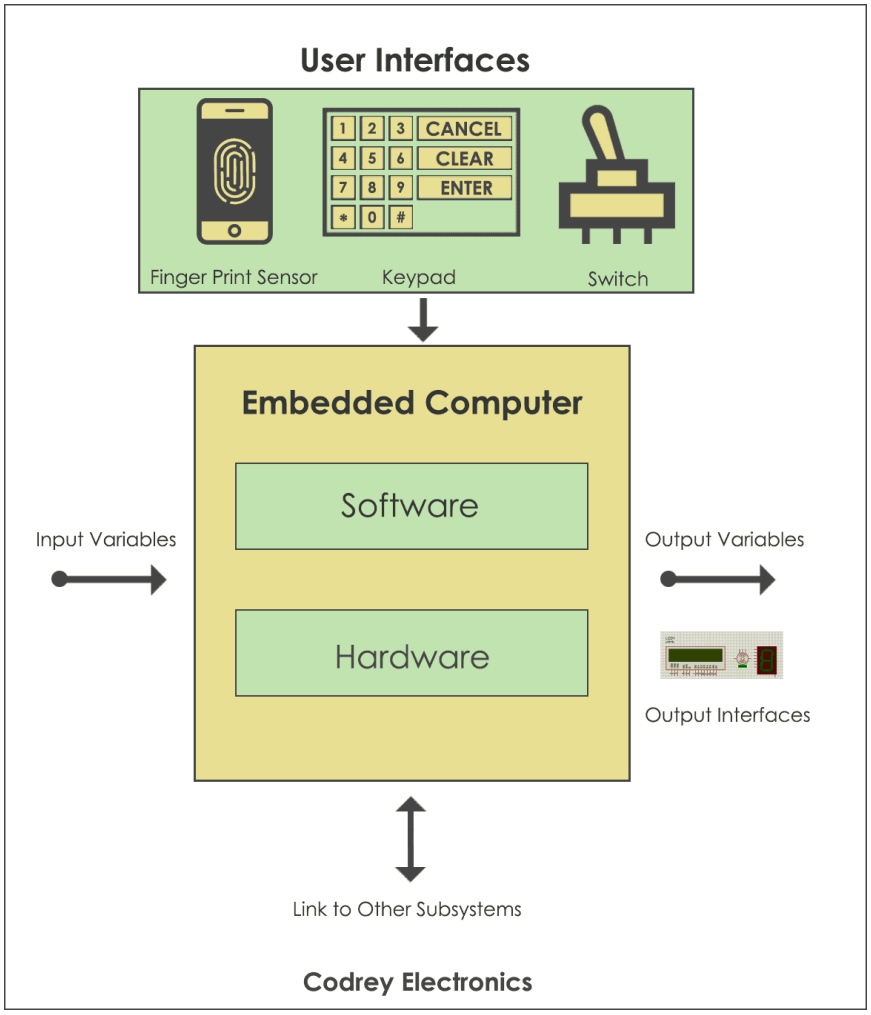
Vstavané systémy na svoju funkciu využívajú mikroprocesor alebo mikrokontrolér (viac v sekcii 2.1.3).

Real-time systém

Real-time systém (systém reálneho času) je akýkoľvek počítačový systém, ktorý má požiadavku byť včasný. Takýto systém musí mať schopnosť uskutočniť svoje výpočty alebo rozhodnutia v určitom časovom rozmedzí, aby stihol takzvaný „*deadline*“. Ak sú následky nestihnutia závažné (ako napr. zlyhanie misie alebo strata života), ide o **hard real-time** systém. Ak systém potrebuje spĺňať časové termíny, avšak závažnosť ich nesplnenia nie je kritická, ide o **soft real-time** systém. [3] Vstavané systémy sú často používané ako real-time systémy kvôli ich špecializácii na jednu určitú úlohu.

Real-time operačný systém (RTOS)

Real-time operačný systém (ďalej **RTOS**) je operačný systém určený pre zložitejšie vstavané systémy, ktoré plnia viacero úloh zároveň. Jeho úlohou je zaobstaranie multi-taskingu (aj na jedno-jadrových/jedno-vláknových procesoroch) – toto rieši časť operačného systému zvaná plánovač (ang. *scheduler*) ktorá rozhoduje, ktorý program bežiaci na zariadení má



Obr. 2.1: Jednoduché znázornenie vstavaného systému. Prevzaté z [4]

momentálne vykonávať svoju funkciu. Prepínanie medzi programami je veľmi rýchle a vytvára tak ilúziu súčasného vykonávania. [16]

2.1.2 System on a chip (SoC)

System on a chip (ďalej **SoC**) je integrovaný obvod (inak „čip“), ktorý integruje všetky alebo väčšinu súčastí počítača alebo iného elektronického systému. V porovnaní so vstavaným systémom sa skôr podobá tradičným počítačom (lebo má univerzálne využitie a ovláda ho obyčajný operačný systém) s tým rozdielom, že všetky jeho hlavné komponenty sa nachádzajú integrované na jednom waferi[1] alebo mikročipe. [5] SoC využívajú napríklad mini-počítače *Raspberry Pi* (viac v sekcii 2.1.6) alebo všetky moderné smartfóny, u ktorých najznámejšie používané SoC sú čipy *Snapdragon* of firmy Qualcomm, čipy *Exynos* od fimy Samsung alebo čipy z *A série* od firmy Apple. Väčšina vysokovýkonných SoC je založených na architektúre **ARM** kvôli tomu, že je vhodnejšia pre prenosné zariadenia z dôvodu nižšej spotreby energie ako je architektúra **x86**, využívaná najmä v dnešných stolových počítačoch, laptopoch alebo serveroch.

[1] substrátový disk

2.1.3 Mikrokontrolér (MCU)

Mikrokontrolér (ang. *microcontroller* alebo *microcontroller unit*, ďalej **MCU**) je vysoko integrovaný mikroprocesor navrhnutý špeciálne pre použitie vo vstavaných systémoch (sekcia 2.1.1). MCU zvyčajne obsahujú integrovaný procesor (mikroprocesor), pamäť (malé množstvo pamäte RAM, ROM alebo obidve, prípadne pamäť FLASH) a ďalšie rôzne periférie na rovnakom čipe (I^2C, SPI, UART, atď.). [2] Ich výhodou je, že cena ich dizajnu a hardvéru samotného je nízka oproti cene mikroprocesoru.

Pri vývoji vstavaného systému môže vývojár použiť „vývojovú dosku“ (ang. *development board*), pomocou ktorej sa vie naučiť ako programovať dané MCU a prípadne ju využiť na prototypovanie pri vývoji vlastného produktu.

General-purpose input/output (GPIO)

Vývojové dosky majú na sebe vyvedené „univerzálne vstupné/výstupné“ piny (ang. *general-purpose input/output*, ďalej **GPIO**), ktoré sú programovateľné a vývojár ich vie použiť na zapojenie rôznych iných zariadení alebo senzorov.

FTDI prevodník

Používa sa ako most medzi USB pripojením z PC a UART (sériový port) pripojením na doske s mikrokonrolérom v prípade, že daná vývojová doska neobsahuje USB port. Vývojár vie s jeho pomocou nahrať do pamäte nový firmvér alebo sledovať výpisy zariadenia na sériovom porte.

2.1.4 Internet vecí (IoT)

Internet vecí (ang. *Internet of Things*, ďalej **IoT**) popisuje sieť fyzických „objektov“ (zariadení), ktorých súčasťou sú senzory, softvér a iné technológie, ktorých účelom je pripojenie a výmena dát s inými zariadeniami a systémami cez internet.

Tradičné oblasti vstavaných systémov (sekcia 2.1.1), bezdrôtových senzorových sietí, riadiacich systémov, automatizácie (vrátane automatizácie domov a budov) a iné prispievajú k umožneniu IoT. Na spotrebiteľskom trhu je technológia IoT najviac stotožňovaná s produktami vzťahujúcimi sa na koncept „inteligentnej domácnosti“ vrátane zariadení a spotrebičov (ako sú napríklad osvetľovacie zariadenia, termostaty, domáce bezpečnostné systémy, kamery a ďalšie domáce spotrebiče), ktoré podporujú jeden alebo viac ekosystémov a je možné ich ovládať prostredníctvom zariadení spojených s týmto ekosystémom, ako sú napríklad smartfóny alebo inteligentné reproduktory. [9]

2.1.5 ESP32

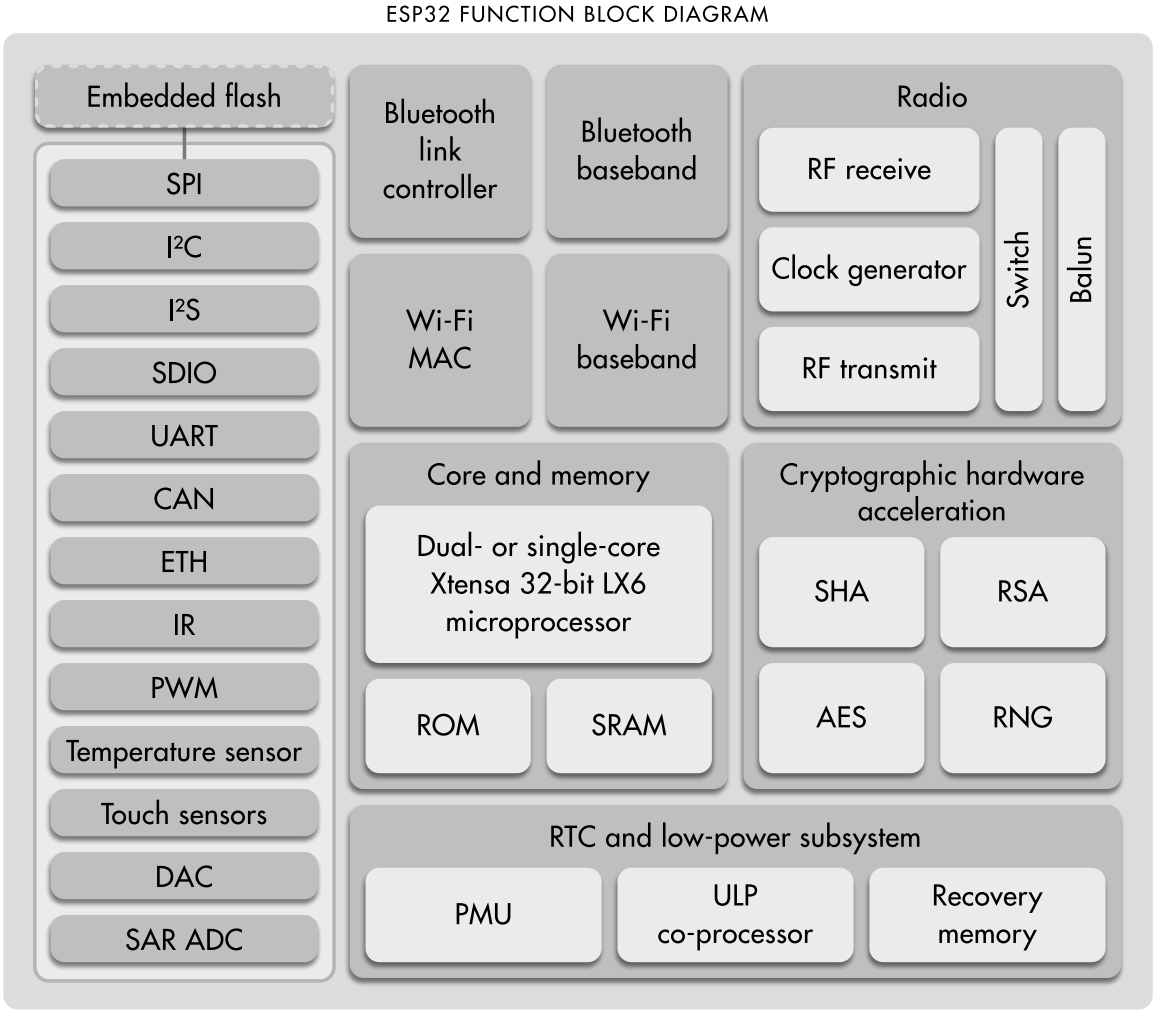
ESP32 je nízko-nákladový a energeticky úsporný SoC (sekcia 2.1.2) mikrokontrolér od firmy Espressif s integrovanou WiFi a Bluetooth technológiou vydaný v roku 2016. Je to nástupca obľúbeného ESP8266 mikrokontroléru.

Medzi jeho periférie patria: 34 programovateľných GPIO pinov, 12-bitový A/D[2] prevodník, 2 8-bitové D/A[3] prevodníky, 10 senzorov dotyku, rozhrania SPI, UART, I^2C, I^2S, PWM[4] pre motor a LED diódy, rozhranie ovládačov pre SD karty a Ethernet. Obsahuje

[2] analógovo-digitálny
[3] digitálno-analógový
[4] impulzová šírková modulácia

taktiež ULP[5] koprocesor pre režim hlbokého spánku, šifrovanie FLASH pamäte, generátor náhodných čísiel, hardvérovú akceleráciu kryptografických algoritmov AES, SHA-2, RSA a ECC a možnosť OTA[6] aktualizácie. [7][8]

V súčastnosti je používaný spoločnosťami v komerčných produktoch ale aj osobných projektoch nadšencov hlavne ako IoT zariadenie kvôli jeho malej veľkosti, konektivite, nízkej spotrebe energie a pomerne vysokému výkonu vzhľadom ku ostatným už spomínaným vlastnostiam.



Obr. 2.2: Blokový diagram ESP32 architektúry. Prevzaté z [7]

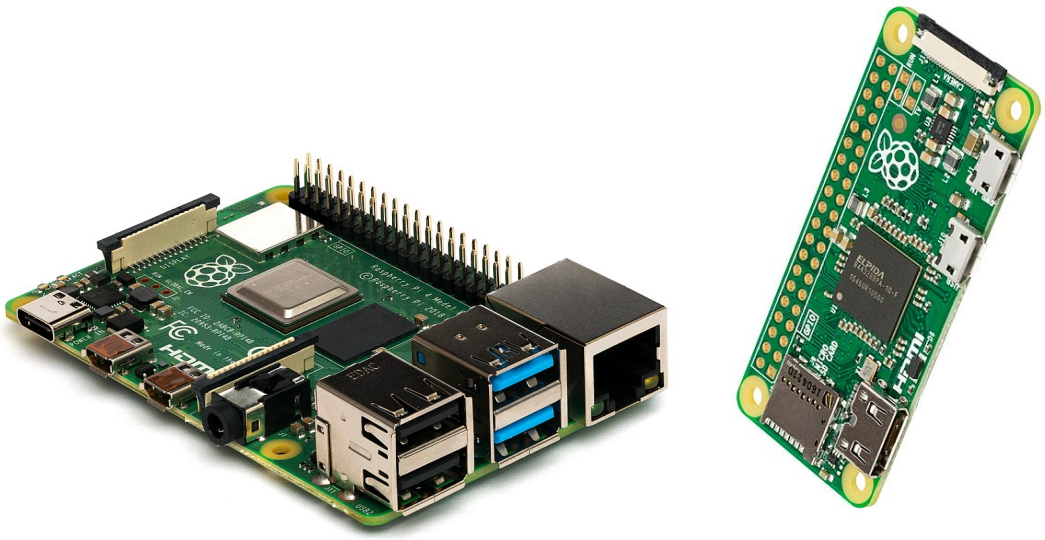
2.1.6 Raspberry Pi

Raspberry Pi je séria populárnych mini-počítačov vytvorená Raspberry Pi nadáciou. Určená bola najskôr pre žiakov do škôl na výuku informatiky, no obľúbili si ju počítačoví nadšenci po celom svete. Hlavý model je veľkosti podobnej kreditnej karte. Procesory sú architektúry ARM a oficiálny operačný systém týchto zariadení sa volá Raspberry Pi OS (kedysi Raspbian), ktorý bol vytvorený z Linuxovej distribúcie Debian.

[5] ultra low power = veľmi nízka spotreba
[6] over-the-air = na diaľku

Najnovšia verzia hlavného modelu je Raspberry Pi 4B, ktorý obsahuje štvor-jadrový procesor, 1 až 8GB pamäte RAM, 2x micro HDMI výstup, audio 3,5mm jack port, 2x USB 2.0, 2x USB 3.0, Ethernet RJ45 port, konektor pre displerj a pre kameru, port pre microSD kartu, GPIO piny a USB-C na napájanie. [15]

Ukážka Raspberry Pi mini-počítačov je na obrázku 2.3.



Obr. 2.3: Raspberry Pi 4B a Raspberry Pi Zero. Prevzaté z [15]

2.2 Softvérové technológie

2.2.1 Cloud computing

Cloud computing („výpočtové mračno“) môžeme chápať ako ukladanie, spracovanie a využívanie údajov prostredníctvom internetu. To znamená, že užívatelia majú k dispozícii neobmedzený počítačový výkon na požiadanie bez veľkých kapitálových investícií a majú prístup ku svojim údajom kdekoľvek, kde je internetové pripojenie. [1] Síce sa stále jedná o nejaký hardvér, na ktorom reálne bežia naše programy, no ten fakt je často pred používateľom skrytý pod určitou softvérovou abstrakciou.

Kľúčové vlastnosti cloud computingu sú:

- samoobslužnosť podľa potrieb,
- prístup cez internet,
- zdieľanie zdrojov,
- vysoká elasticita,
- merateľnosť.

Servisné modely cloud computingu sú:

- softvér ako služba (SaaS),
- platforma ako služba (PaaS),
- infraštruktúra ako služba (IaaS),

2.2.2 JSON

JSON alebo *JavaScript Object Notation* je štandardný textový formát na reprezentáciu štruktúrovaných údajov založený na syntaxe pre objekt v programovacom jazyku JavaScript. Bežne sa používa na prenos údajov vo webových aplikáciách. [11] JSON môže reprezentovať čísla, logické *Boolean* hodnoty, reťazce, polia, objekty a hodnotu *null*. Ak je potrebné, aby JSON predstavoval ďalšie dátové typy, ich hodnoty je potreba transformovať – **serializovať**. Pre získanie pôvodných dát z JSONu je potreba tieto hodnoty **deserializovať**. [10]

```
{
    "zamestnanci":
    [
        {"meno": "Ján", "priezvisko": "Čierny", "vek": 42, "bonus": true},
        {"meno": "Anna", "priezvisko": "Malá", "vek": 20, "bonus": false},
    ]
}
```

Obr. 2.4: Znázornenie obsahu JSON súboru

2.2.3 MQTT

MQTT (*Message Queuing Telemetry Transport*) je klient-server „*publish/subscribe*“ (publikovať/odoberať) protokol určený pre transport správ. Je nenáročný, otvorený, jednoduchý a dizajnovaný na ľahkú implementáciu. Tieto charakteristiky ho tvoria ideálny pre použitie v prostrediach ako je napríklad komunikácia medzi IoT (2.1.4) zariadeniami, kde je treba šetriť veľkosťou kódu a/alebo šírka pásma siete, po ktorej sa komunikuje, je minimálna. [14]

MQTT téma

V MQTT slovo téma (ang. *topic*) odkazuje na reťazec zložený z UTF-8 znakov, ktorý broker používa na filtrovanie správ pre každého pripojeného klienta. Téma sa skladá z jednej alebo viacerých úrovní. Každá úroveň témy je oddelená lomkou (viď. obrázok 2.5).

V porovnaní s frontou správ je používanie MQTT témy výpočetne menej náročné. Klient nemusí vytvárať požadovaný tému skôr, ako do neho správu publikuje alebo ju začne odoberať. Broker akceptuje každú platnú tému bez akejkoľvek predchádzajúcej inicializácie. [13]

Použiteľné zástupné znaky (ang. *wildcards*, iba pre odberateľov):

- $+$ – nahradí 1 úroveň témy,
- $\#$ – nahradí všetky nasledujúce úrovne témy.

MQTT klient

MQTT klient je akékoľvek zariadenie (od mikrokontroléru (2.1.3) po plnohodnotný server), ktoré dokáže spustiť MQTT knižnicu a pripojiť sa cez sieť na MQTT broker. Obaja „odberateľ“ (ang. *subscriber*) aj „vydavateľ“ (ang. *publisher*) správ sú MQTT klientmi. [12]

Obr. 2.5: Znázornenie MQTT témy. Prevzaté z [13]

MQTT broker

MQTT broker („sprostredkovateľ“) má na zodpovednosti prijímanie všetkých správ, ich filtrovanie, rozhodovanie, aký klient odoberá aké správy (akú odoberá tému) a odosielanie týchto správ odberateľom. Broker taktiež drží údaje o súčasnej relácii všetkých klientov, ktorí majú perzistentné relácie, spolu so zoznamom odoberaní a zmeškaných správ. Ďalšia zodpovednosť brokera je autentifikácia a autorizácia klientov. [12]

Kvalita služby (QoS)

Kvalita služby (ang. *Quality of Service*, ďalej **QoS**) je dohoda medzi odosielateľom a príjemcom správy, ktorá definuje záruku doručenia konkrétnej správy. Pri QoS v MQTT sa musia brať do úvahy dve strany doručovania správ:

1. Doručenie správy brokerovi od klienta, ktorý správy publikuje.
2. Doručenie správy od brokera klientovi, ktorý správy odoberá.

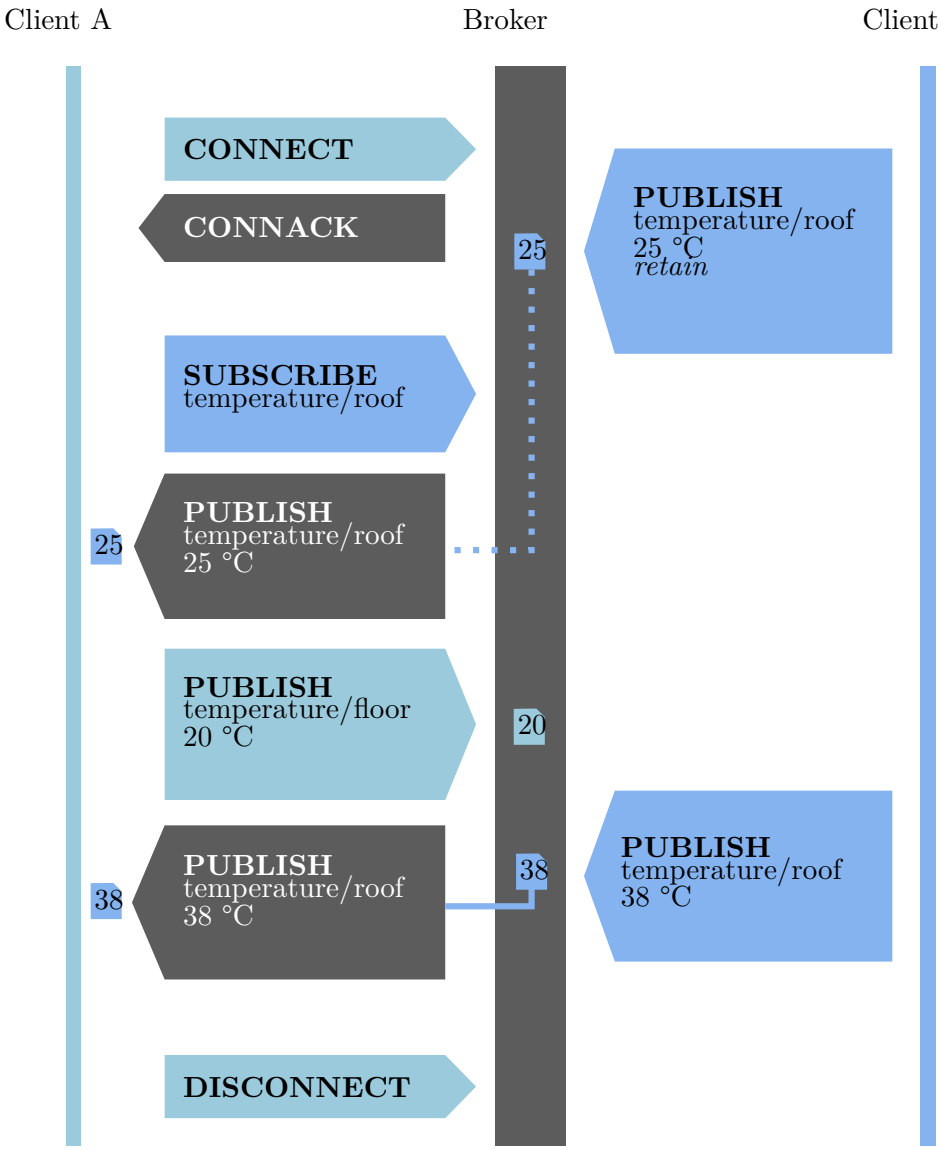
V MQTT sú 3 úrovne QoS:

- maximálne raz (0),
- aspoň raz (1),
- presne raz (2).

Typy správ

V MQTT protokole existujú 3 typy správ:

- CONNECT – použité klientmi na odoslanie požiadavky o pripojenie na broker,
- PUBLISH – použité klientom/odosielateľom na odoslanie správ na broker,
- SUBSCRIBE – použité klientom/odosielateľom na doručenie správ z brokera.



Obr. 2.6: Znázornenie chodu MQTT správ[7]

[7]Prevzaté z https://en.wikipedia.org/wiki/MQTT

Kapitola 3

Návrh riešenia

Táto kapitola sa venuje návrhu systému detekcie prítomnosti v stráženom priestore. Systém sa skladá zo vstavaných systémov, ktoré budú použité ako moduly (sekcia 3.4), z centrálneho ovládača (sekcia 3.3), ktorý bude tieto moduly obsluhovať, z databázy pre uchovávanie perzistentných dát (sekcia 3.5) a z komunikácie medzi jednotlivými časťami systému (sekcia 3.6). Na začiatku kapitoly spomeniem už podobné existujúce riešenia alebo produkty (sekcia 3.1), celkový návrh systému (sekcia 3.2) a ďalej rozpíšem návrh jednotlivých častí systému a ich rozhranie (sekcia 3.7).

3.1 Existujúce riešenia

Zabezpečenie domáceho priestoru je v dnešnej dobe možno riešiť dvoma typmi zabezpečovacích systémov – klasickými „dumb“ alebo modernými „smart“ systémami.

Klasické zabezpečovacie systémy určené pre domácnosť sa skladajú zvyčajne z kamier, ktoré neustále snímajú určitú časť pozemku. Ich snímanie býva nahrávané na úložisko kde je uložené do vymazania, ktoré môže byť manuálne alebo automatické, napríklad po určitom čase od nahratia. Ich nevýhodu je, že používateľ si musí škodu na svojom pozemku vykonanú zlodejmi sám všimnúť, prezrieť záznam a dúfať, že je daný čin zachytený a dostatočne kvalitný na získanie informácii.

V dnešnej dobe sa začína trh zapĺňať aj inteligentnejšími kamerovými systémami s existujúcou mobilnou aplikáciou, v ktorej si používateľ vie nastaviť aj pokročilejšie funkcie ako napríklad mobilné upozornenia pri nasnímaní pohybu alebo prezeranie živého záznamu na diaľku a podobne.

Medzi najmodernejšie systémy patrí systém *Ring*[1], ktorý ponúka inteligentné bezpečnostné kamery, video-zvončeky, senzory na pohyb, kontakt, atď. Ich produkty sú pripojené na internet, ovládateľné cez mobilnú aplikáciu a majú prívetivé grafické rozhranie. Nevýhoda takéhoto riešenia je, že pre pokročilé funkcie je potrebné neustále pripojenie k internetu a pre prístup ku všetkým funkciám platformy používatelia navyše musia platiť mesačné poplatky okrem nemalej ceny za produkty samotné.

3.2 Celkový návrh

Na úvod musím konštatovať, že v súčastnosti sú na trhu zabezpečovacie systémy pre domácnosti, ktoré spĺňajú vysoké štandardy bezpečnosti. Podľa môjho úsudku by mal moderný

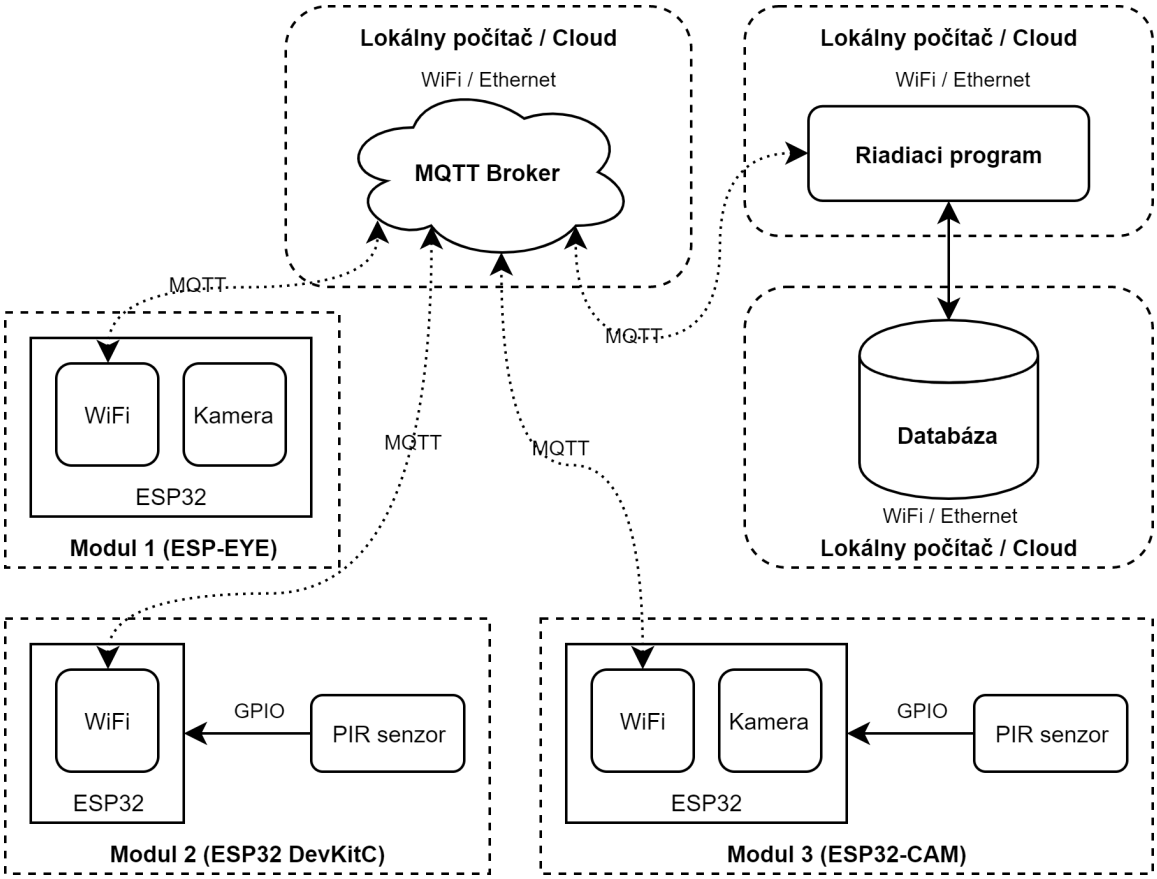
[1] Zdroj: https://eu.ring.com

zabezpečovací systém byť „inteligentný“ (ang. „smart“), nastaviteľný podľa požiadavok používateľa, modulárny, ľahko rozšíriteľný, opraviteľný a cenovo dostupný.

Návrh systému na detekciu prítomnosti subjektov v stráženom priestore sa dá poňať viacerými spôsobmi. Moja voľba bola systém rozdeliť na viacero častí, ktorými sú:

- *moduly* – moduly sú zložené zo vstavaných systémov využívaných ESP32 mikrokontrolér a senzorov ako je napr. pasívne infračervené čidlo (PIR), senzor mikrovĺn alebo senzor nárazového zvuku,
- *riadiaca jednotka (centrálny ovládač)* – obstaráva väčšinu funkcionality systému, ´prijíma správy od modulov a vyhodnocuje ich, zjednodušuje interakciu ostatných častí systému s používateľom,
- *databáza* – ukladá dáta, ktoré môžu zostať uchované medzi jednotlivými spusteniami systému bez potreby ich odznova nastavovať pri každom reštarte systému,
- *MQTT broker* – používa sa na komunikáciu, prepája moduly, riadiacu jednotku a potenciálne používateľa alebo rozšírenia systému.

Schéma môjho návrhu je znázornená na obrázku 3.1. Návrh systému sa spolieha na prítomnosť WiFi pripojenia a neprerušovaného signálu.



Obr. 3.1: Bloková schéma návrhu systému

Systém má tri stavy:

- *nezabezpečený/odomknutý,* – riadiaca jednotka ignoruje správy o narušení zabezpečovaného priestoru, ktoré sú zasielané modulmi,
- *zabezpečený/zamknutý* – systém je v stave stráženia a sleduje správy o narušení zabezpečeného priestoru z modulov,
- *napadnutý* – senzory v moduloch boli aktivované počas zabezpečeného stavu systému a to je riadiacou jednotkou vyhodnotené ako narušenie stráženého priestoru a stav systému sa zmení na napadnutý.

Grafické prostredie nie je súčasťou môjho návrhu a to z dôvodu, že moja diplomová práca je súčasťou väčšieho riešenia a túto časť spolu so zobrazovaním a vyhodnocovaním jednotlivých udalostí mal riešiť iný študent vo svojej diplomovej práci.

3.3 Riadiaca jednotka

Riadiaca jednotka alebo centrálny ovládač je časť systému, s ktorou komunikujú moduly pomocou MQTT protokolu cez MQTT broker. Drží v sebe stav systému a na uloženie nastavení využíva databázu, čím sa systém po reštarte nemusí nastavovať vždy odznova.

Hlavnou úlohou tejto časti je riešenie komplexnejšej funkcionality systému. Tento prístup zjednodušuje návrh a implementáciu firmvéru pre moduly a dáva mi možnosť ľahšej implementácie logiky systému na plnohodnotnom PC vo vyššom programovacom jazyku, ktorý je bližší pseudo-kódu. Tým sa pridáva časť systému navyše oproti riešeniu všetkej logiky vo firmvéri vstavaného systému, ale na druhú stranu získavam vopred spomínané výhody. Z toho dôvodu som riadiacu jednotku navrhol ako počítačový program, ktorý je nezávislý od toho, kde bude spustený – môže to byť na lokálnom PC, na mini-počítači Raspberry Pi alebo na vzdialenom serveri. Používateľ si sám zvolí, kde bude mať systém nasadený, čo je výhodou tohto riešenia. Riadiaca jednotka sa správa ako ďalší MQTT klient pripojený na MQTT broker, pričom sníma a analyzuje každú odoslanú správu v systéme a reaguje na ňu.

Komplexnejšia funkcionalita sa dá chápať ako sprostredkovanie abstrakcií nad samotnými modulmi, ako je určenie ich prezývok/označení (ang. label), lokality v priestore, zaradenie ich do rôznych používateľom definovaných skupín, či nastavenie správania systému po príchode MQTT správ z modulov.

Pri nasadení na cloud spolu aj s MQTT brokerom používateľ nemusí vlastniť okrem modulov žiadne zariadenia navyše a ani mať nonstop zapnutý PC používaný ako riadiacu jednotku. Nevýhoda tohto riešenia je závislosť na internetovom pripojení.

Pri lokálnom nasadení napr. na už spomínané Raspberry Pi spolu aj s MQTT brokerom používateľ získa úplnú kontrolu nad svojim systémom bez závislosti na internete, no systém beží iba v lokálnej sieti – pre ešte väčšie pohodlie by mohol byť implementovaný „most“ medzi lokálnou sieťou a internetom, ktorý by medzi nimi vytváral tunel a používateľ by vedel zistiť stav systému aj zvonku. Toto je podľa môjho názoru ideálne riešenie.

3.4 Moduly

Modul v mojej práci chápem ako spojenie vstavaného systému a senzoru. Môže sa jednať o kameru, senzor s digitálnym binárnym výstupom alebo analógovým výstupom. Použitý vstavaný systém používa mikrokontrolér ESP32.

Modul je pripojený na sieť pomocou vstavanej WiFi a komunikuje s okolitým svetom (najmä riadiacou jednotkou) pomocou MQTT protokolu navrhnutého pre IoT zariadenia. Každý modul má vlastný firmvér s predurčenými nastaveniami, pričom reaguje iba na správy ktorým rozumie – požiadavka na vyhotovenie fotografie može byť spracovaná iba modulom, ktorý obsahuje kameru. Modul bez kamery takúto správu ignoruje – avšak bez manuálneho zásahu by takúto správu ani dostať nemal.

Modul na plnú funkcionalitu potrebuje riadiacu jednotku, bez nej funguje iba odosielanie a prijímanie MQTT správ na jeho unikátnu MQTT tému.

3.4.1 Porovnanie ESP32 s inými populárnymi vývojovými doskami

Základné parametre	Arduino Uno	Raspberry Pi Pico	ESP32
Pracovné napätie	5V	3.3V	3.3V
Procesor (CPU)	ATmega328P (AVR)	ARM Cortex M0+	Tensilica Xtensa LX6
Typ CPU	8-bit	32-bit	32-bit
Frekvencia CPU	20MHz	133MHz	160/240 MHz
Počet jadier	1	2	1-2
SRAM	2KB	264KB	520KiB
Pamäť FLASH	32KB	2MB	4-16MB
GPIO piny	20	26	34
WiFi/Bluetooth	nie/nie	nie/nie	áno/áno
Cena[2]	23$	4$	11$ (DevKitC)

Tabuľka 3.1: Tabuľka porovnania vývojových dosiek s mikrokontrolérmi

Pri pohľade na cenu samostatných vývojových dosiek (viď. tabuľku 3.1) by sa mohlo zdať, že Raspberry Pi Pico sa z ekonomických dôvodov oplatí použiť viac než ESP32. To však nie je pravda, keďže pri tejto práci používam na komunikáciu WiFi, ktorá nie je na Raspberry Pi Pico prítomná a musel by sa dokúpiť externý modul, čo by v konečnom dôsledku pridalo na cene a aj zložitosti riešenia. Aj keby bola cena porovnateľná s ESP32, tak ESP32 má viac SRAM, Flash pamäte a vyšší výkon. Arduino Uno je pre moje účely príliš drahé, nevýkonné, bez žiadaných možností pripojenia na sieť a je teda všeobecne nepraktické.

3.4.2 Napájanie

Moduly sú napájané pomocou USB konektora nachádzajúceho sa na vývojovej doske. V prípade, že sa na doske nenachádza, je použitý 5V pin pre napájanie zariadenia zo zdroja s napätím 5V alebo 3.3V pre pin pre napájanie zo zdroja s napätím 3.3V. Samotný ESP32 mikrokonrolér používa 3.3V, avšak na väčšine vývojových dosiek sa nachádza regulátor napätia.

3.4.3 Senzory

Moduly, ako som už spomínal, sú zložené zo vstavaného systému a senzoru. Modul by mal byť použiteľný s rôznymi typmi senzorov. Väčšina senzorov sa delí na dve kategórie.

[2]Cena bola určená z oficiálnych stránok produktu, prípadne zo stránok lokálnych predajcov ak na oficiálnej stránke nebola uvedená – ku dňu: 2. máj 2021

Výstup senzorov je digitálny – binárny, 0 alebo 1, alebo analógový – hodnota výstupu je súčasťou určeného rozmedzia hodnôt, napríklad 0 až 1023, t.z. spolu 1024 hodnôt. Rozmedzie výstupných hodnôt analógového senzora je spojené so špecifikáciou analógovo-digitálneho prevodníka na doske vstavaného systému.

Pri riešení tejto práce som mal prístup ku viacerým senzorom, no navrhujem použitie najmä ďalej spomenutých:

- *kamera* – od ostatných senzorov sa líši tým, že je zabudovaná v samotnej ESP32 vývojovej doske a nepripája sa externe pomocou GPIO pinov,
- *PIR* – pasívne infračervené čidlo, reaguje na pohyb a má digitálny výstup, môže byť využitý ako senzor pohybu, používa infračervené žiarenie a preto nemôže mať blokovaný výhľad,
- *mikrovlnný senzor* – používa mikrovlnné žiarenie, reaguje na pohyb, reaguje na pohyb a má digitálny výstup, môže byť taktiež použitý ako senzor pohybu, má výhodu oproti PIR senzoru že vie snímať aj po zakrytí materiálmi ktoré prepustia mikrovlnné žiarenie,
- *senzor nárazového zvuku* – meria rozdiely v dB zvuku, má digitálny výstup, môže byť vhodný ako senzor pri oknách snímajúci hlasné zvuky ako roztrieštenie skla,
- *piezoelektrický snímač vibrácií* – jeho ohýbanie generuje zmenu v napätí, má digitálny výstup, je ho možné použiť ako senzor čo sníma otvorenie dverí, ktoré mali byť zamknuté.

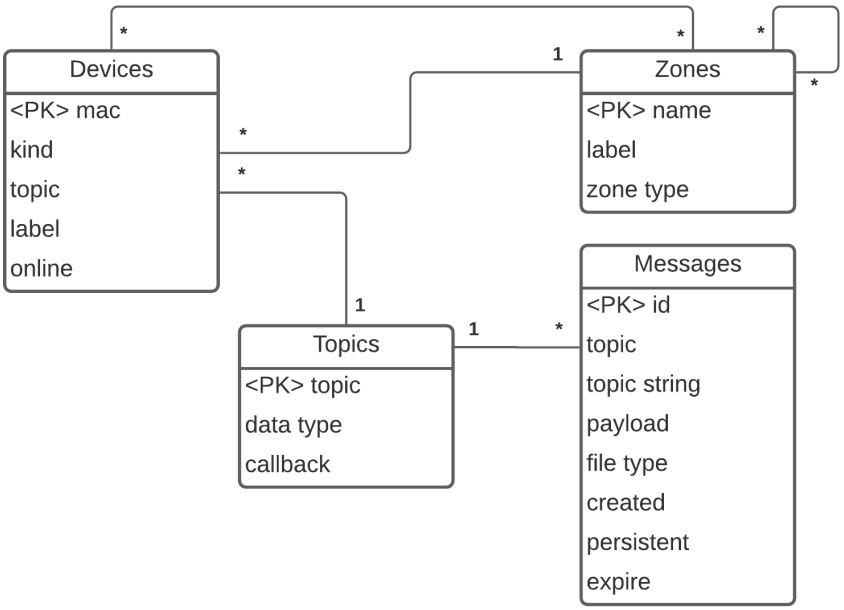
3.5 Databáza

Databáza obsahuje údaje o zariadeniach, takzvaných zónach (abstrakcia nad lokáciami, skupinami zariadení a typmi zariadení), zaregistrovaných MQTT témach a logovaných MQTT správach. Logovanie správ je voliteľné a závislé na implementácií funkcie, ktorá obsluhuje MQTT tému, na ktorú bola táto správa odoslaná. ER diagram zobrazujúci návrh databázy je na obrázku 3.2.

3.6 Komunikácia

Komunikácia medzi zariadeniami a riadiacou jednotkou prebieha pomocou MQTT protokolu. Všetky zariadenia sú pripojené na MQTT broker ako klienti. Každý modul má určenú svoju unikátnu MQTT tému obsahujúcu jeho MAC adresu. Pomocou nej vie riadiaca jednotka, sledujúca všetku komunikáciu idúcu cez MQTT broker určiť, s ktorým zariadením komunikuje. Obsah správ môže byť obyčajný Unicode text, text v JSON formáte alebo binárne dáta.

Obyčajný text a JSON sa používajú na bežnú komunikáciu, pričom binárne dáta sú odosielané v prípadoch, ako je napríklad odoslanie fotografie zachytenej modulom s kamerou. To, aký typ správ sa posiela na danú MQTT tému je určené v nastavovacom súbore pre riadiacu jednotku, ktorá všetky tieto správy spracováva.



Obr. 3.2: ER diagram návrhu databáze

Riadiaca jednotka teoreticky môže komunikovať s databázou napríklad pomocou určeného REST[3] API, ak by bola uložená na cloude. V mojej implementácii je uložená na tom istom systéme ako riadiaca jednotka, takže k nej pristupuje ako k obyčajnému súboru, pričom pri načítaní sa uloží do cache nachádzajúcej sa vo vyrovnávacej pamäti RAM.

3.7 Rozhranie

Keďže komunikácia systému je založená na MQTT protokole, je veľmi jednoduché s ním komunikovať.

Skúsený používateľ môže využiť príkazový riadok a používať nástroj ako je *Mosquitto*[4], konkrétne *Mosquitto_pub* na publikovanie správ obsahujúcich príkazy ako napríklad príkaz pre zaznamenanie fotografií zo všetkých kamier, príkaz na reštart zariadenia, príkaz na vykonanie aktualizácie firmvéru na diaľku a podobne.

Obyčajnými používateľmi by mal byť použitý systém s grafickým rozhraním, ktorý by mal zaznamenávať všetky správy a udalosti, vyhodnocovať a vykresľovať ich. Takýto systém mal byť súčasťou väčšieho riešenia a mal mať schopnosť prepojiť sa s mojim systémom.

Pre účely demonštrácie bez tejto ďalšej práce som navrhol jednoduché grafické rozhranie ktoré je spustené riadiacou jednotkou a využíva webový server, pričom rozhranie je samotná HTML web stránka naplňaná informáciami z riadiacej jednotky a databázy. Zobrazuje základné informácie ako je stav systému: priestor je nestrážený, priestor je strážený a priestor je napadnutý.

[3] Representational State Transfer — architektúra rozhrania, navrhnutá pre distribuované systémy využívajúca HTTP protokol

[4] Zdroj: https://mosquitto.org

Kapitola 4

Implementácia

Táto kapitola popisuje hlavné časti implementácie jednotlivých dielov práce s použitím vybavenia, ktoré som mal k dispozícii. Najskôr popisujem celkovú štruktúru projektu (sekcia 4.1) a neskôr jeho jednotlivé časti ako sú: databáza (sekcia 4.2), riadiaca jednotka (sekcia 4.3), moduly (sekcia 4.4), MQTT broker (sekcia 4.5) a zabezpečenie (sekcia 4.6).

4.1 Štruktúra projektu

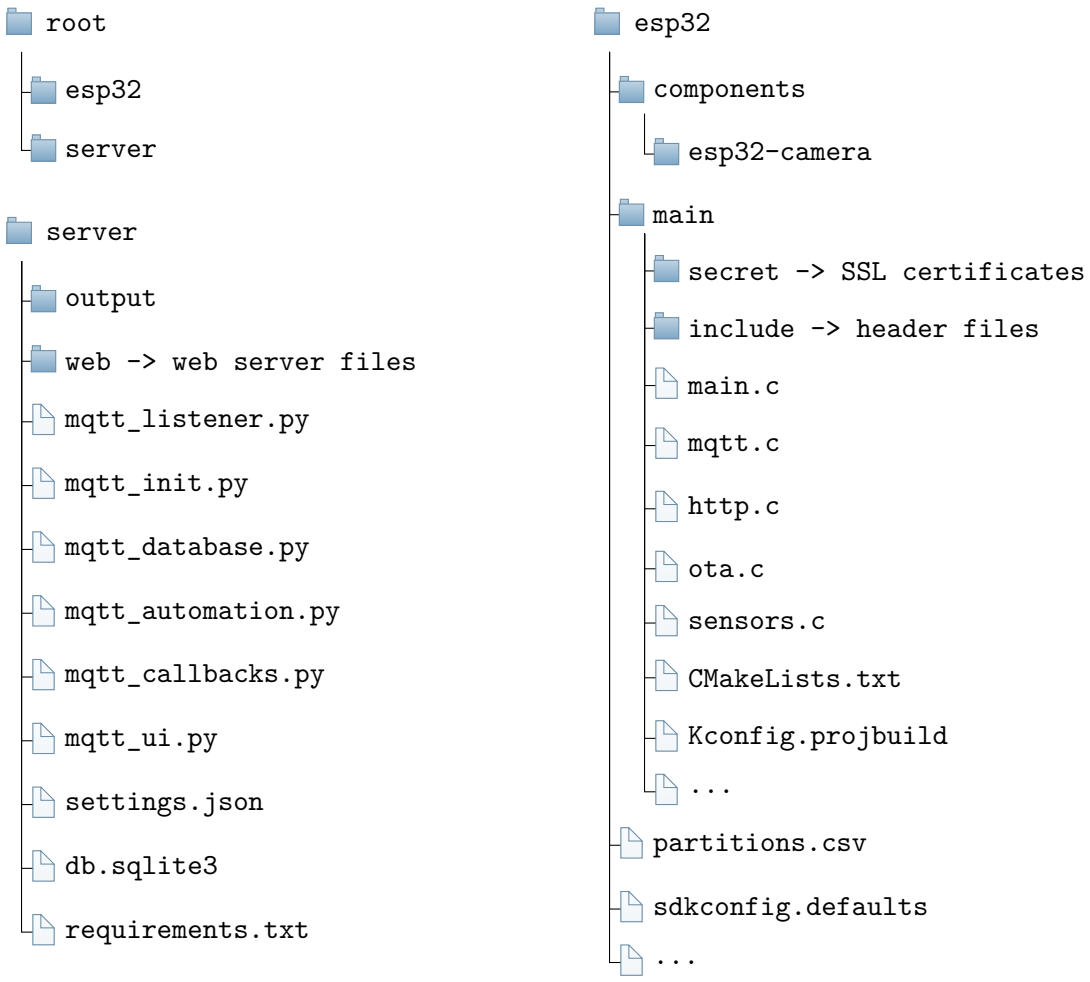
Práca je rozdelená do dvoch hlavných častí (respektíve zložiek) – *esp32*, kde sa nachádza firmvér pre moduly a *server*, kde sa nachádza kód pre riadiacu jednotku a databázu. Štruktúra implementačných častí práce je zobrazená na obrázku 4.1.

Priečinok *esp32* obsahuje:

- zdrojový kód programu bežiaci na ESP32 zariadeniach sa nachádza v zložke *main*, kde sú vložené taktiež SSL certifikáty, ktoré sa pribalia do binárneho súboru firmvéru pri kompilácii,
- priečinok *components* obsahuje externé ESP-IDF komponenty, ktoré využívam v mojej práci a to je konkrétne projekt *esp32-camera*[1], ktorý mi umožňuje používanie kamier v spojení s ESP32 mikrokontrolérom,
- súbor *sdkconfig.defaults*, kde sú nastavené predvolené nastavenia projektu pre ESP-IDF (viac v sekcii 4.4 a časti Firmvér),
- súbor *partitions.csv* definuje tabuľku rozdelenia FLASH pamäte zariadenia na tieto hlavné časti: NVS[2], pamäť pre pôvodnú verziu programu a 2 pamäte využívané na OTA aktualizácie,
- súbor *CMakeLists.txt* v priečinku *main* obsahuje potrebnú konfiguráciu a vyčlenenie súborov a komponentov, ktoré sú potrebné pri preklade programu,
- súbor *Kconfig.projbuild* v taktiež priečinku *main* definuje vlastné menu dostupné z príkazu aplikácie na príkazovom riadku EPS-IDF toolchainu *idf.py menuconfig*, ktorým sa ESP-IDF projekty konfigurujú (viac v sekcii 4.4).

[1] Zdroj: https://github.com/espressif/esp32-camera

[2] non-volatile storage = energeticky nezávislé úložisko (jeho obsah zostane zachovaný aj po reštarte zariadenia)



Obr. 4.1: Znázornenie hlavných zložiek a súborov práce

Priečinok *server* obsahuje:

- program riadiacej jednotky, ktorý sa skladá z viacerých Python súborov. Hlavný súbor je *mqtt_listener.py*, ktorý využíva ostatné Python súbory (viac v sekcii 4.3) – program je takto rozdelený kvôli praktickosti a priečiok taktiež obsahuje pár testovacích Python súborov ktorých názov začína slovom „test“,
- priečinok *output*, kde sa prípadne ukladajú súbory poslané v MQTT správe (napr. kamera odošle fotografiu po tom, ako zachytila pohyb PIR senzorom – fotografiu však musí uložiť callback funkcia v riadiacej jednotke, ktorá správy z danej témy obsluhuje),
- súbor *requirements.txt* obsahuje požadovanú verziu Pythonu 3 a Python balíčky, ktoré je potrebné nainštalovať,
- súbor *db.sqlite3* obsahuje databázu využívanú riadiacou jednotkou – ak neexistuje, automaticky sa vytvorí,

- súbor *settings.json*, kde sa nachádzajú počiatočné nastavenia pre riadiacu jednotku a databázu.

4.2 Databáza

Databázová časť je úzko spätá s centrálnym ovládačom, ktorý k nej jediný priamo pristupuje. Načítava do nej nastavenia z JSON (viď. sekcia 2.2.2) súboru *settings.json* a uchováva informácie o lokáciach, skupinách, druhoch zariadení, zariadeniach samotných a vzťahmi medzi nimi.

Peewee[3] ORM[4] som si vybral ako nástroj na jednoduchšiu prácu a prepojenie môjho Python kódu s databázou. Zaisťuje rovnakú syntax pre rôzne druhy SQL databáz (MySQL, PostgreSQL, SQLite, CockroachDB). V mojom prípade používam databázu typu SQLite 3. Pomocou tejto knižnice môžem písať *SQL queries* („SQL dopyty“) rovno v Pythone a lepšie ich tak integrovať s ostatným kódom, pričom nemusím používať priamo jazyk SQL.

Databáza je definovaná v súbore *mqtt_database.py*. Používa 64MB cache, čo je veľkosť tabuliek z databázy, ktoré udržuje neustále v pamäti kvôli čomu je pristupovanie do databázy veľmi rýchle – pri mojej veľkosti databáze je sú zaplnené stotiny tejto hodnoty a nemusím sa báť spomalenia. Primárne kľúče vyberá Peewee automaticky.

Tabuľky v databáze

- *Topic* – obsahuje predom zaregistrované MQTT témy (viď. sekciu 2.2.3),
- *Zone* – obsahuje zóny, t.z. lokácie, skupiny a druhy modulov (viac v sekcii 4.3.1),
- *ZoneToZone* – rieši hierarchiu medzi zónami (many-to-many problém),
- *Device* – obsahuje zaregistrované zariadenia v databáze,
- *DeviceZone* – obsahuje informácie o tom, v akých zónach sa zariadenie nachádza,
- *Message* – obsahuje archivované MQTT správy.

Súbor s nastaveniami

Súbor s nastaveniami *settings.json* je vo formáte JSON a jeho časť reprezentuje základnú štruktúru databázy.

Nastavenia pod *settings* určujú správanie načítavania do databázy – má sa databáza pri zapnutí centrálneho ovládača a načítania nastavení najprv premazať alebo donačítať nové položky? Tieto nastavenia určujú kolónky textitsave_zones a textitsave_devices.

```
1 {
2     "settings": {"save_zones": true, "save_devices": true}
3 }
```

[3] Zdroj: https://docs.peewee-orm.com

[4] ORM (Objektovo-relačné mapovanie) je programovacia technika, ktorá zaisťuje automatickú konverziu dát medzi relačnou databázou a objektovo orientovaným programovacím jazykom

```python
class Topic(BaseModel):
  topic = pw.CharField(unique=True, null=False)
  datatype = pw.CharField(unique=False, null=False, default="string")
  callback = pw.CharField(unique=False, null=True)

  @classmethod
  def get_topics_and_callbacks(cls):
    query = cls.select().where(Topic.callback.is_null(False))
    return [(t.topic, t.callback) for t in query]

  def __repr__(self):
    return (f"{self._pk}) Topic(topic='{self.topic}',"
            f"datatype='{self.datatype}', callback={self.callback})")
```

Obr. 4.2: Implementácia tabuľky Topic v jazyku Python s pomocou Peewee knižnice

Locations konfiguruje lokácie a *groups* užívateľom definované skupiny, ktoré bude systém poznať. Lokácie vedia medzi sebou vytvoriť hierarchiu (byť previazané medzi sebou many-to-many vzťahom pomocou pomocnej spojovacej tabuľky). V konfiguračnom súbore sa táto väzba definuje pod kolónkou textitparent. Lokácie, skupiny a aj druhy zariadení sú v databáze zjednotené pod *zóny*. Pri prehľadávaní databáze tak viem pomocou prieniku zón nájsť zariadenia – napríklad zariadenie, ktoré je zároveň v spálni, na prízemí a je to senzor pohybu. Túto vlastnosť využíva centrálny ovládač a pomocou nej môžem adresovať zariadenia cez „virtuálne“ MQTT témy. Viac v sekcii 4.3.1.

```
{"locations":
        [{
          "name": "house",
          "label": "House",
          "parent": null
        },{
          "name": "groundfloor",
          "label": "Ground Floor",
          "parent": ["home"]
        },{
          "name": "wc",
          "label": "Toilet",
          "parent": ["home", "groundfloor"]
        }],
    "groups":
        [{
          "name": "night-lights",
          "label": "Night lights"
        }]
}
```

Cez *topics* si definujem MQTT témy spolu s callback funkciou, ktorá bude danú tému „obsluhovať“ – registruje s pri zapínaní centrálneho ovládača a bude volaná miesto štandardnej funkcie ktorá sa spúšťa pri príchode MQTT správy. Ďalej si určujem typ dát, ktoré má callback očakávať. Možné typy dát sú *string*, *json* a *binary*.

```
{"topics":
        [{
            "topic": "device/register",
            "datatype": "json",
            "callback": "device_register"
        }]
}
```

Ukážka celého príkladu konfiguračného súboru je v prílohe A.

4.3 Riadiaca jednotka

Riadiacu jednotku alebo centrálny ovládač som implementoval ako program v programovacom jazyku Python 3. Kvôli tomu nezáleží kde je spustený – môže to byť doma na lokálnom PC, na Raspberry Pi alebo kľudne aj na cloude (to isté platí aj o MQTT brokerovi a databáze). V dobe testovania bol skúšaný na lokálnom PC a na mini-počítači Raspberry Pi.

Program sa skladá z viacerých Python súborov:

- *mqtt_listener.py* – hlavný súbor, obsahuje logiku MQTT klienta,
- *mqtt_init.py* – načítavanie konfiguračnéhu súboru do databázy,
- *mqtt_database.py* – definovaná databáza,
- *mqtt_callbacks.py* – implementácie callback funkcií pre jednotlivé MQTT témy,
- *mqtt_automation.py* – logika automatizácie, napr. vyhľadávanie zariadení podľa zón, kde sa nachádzajú.

Centrálny ovládač pri spustení načíta súbor s konfiguráciou „settings.json“, pripraví databázu (viď. sekcia 4.2) sa pripojí na MQTT broker ako klient, kde sleduje všetky MQTT témy. Každá téma však môže obsluhovaná inou callback funkciou, čo je definované v konfiguračnom súbore.

4.3.1 Virtuálne MQTT témy

Moduly je možné rozdeliť do kategórii: lokácie, kde sa nachádzajú, skupiny zariadení, kam sú priradené a typy zariadení (t.z. v databáze spoločne do zón), pričom typ zariadenia je daný samotným zariadením (obsahuje ho firmvér zariadenia). Moduly majú určené MQTT témy na MQTT brokeri, z ktorých odoberajú správy. Tam patria všeobecné témy, ktoré sú spoločné pre všetky zariadenia, ako napríklad téma na ktorú príjmu MQTT správu, ktorou ich centrálny ovládač žiada o poslanie registračnej JSON správy s informáciami o module.

Každé zariadenie má svoju vlastnú unikátnu MQTT tému, ktorej súčasťou je MAC adresa zariadenia – napríklad *device/e868e722c1d8*. Na tejto MQTT téme modul načúva

```python
def default_callback(client, userdata, msg):

  if mqtt_listener.debug_print:
    print(f"{msg.topic} - DEFAULT on_message() callback")

  IGNORE_TOPICS = {"device/"}
  if any(x not in msg.topic for x in IGNORE_TOPICS):
    devices = mqtt_automation.parse_unknown_topic_get_devices(msg.topic)
    for d in devices:
      client.publish(f"device/{d.device.mac}", msg.payload)
```

Obr. 4.3: Ukážka implementácie štandardnej callback funkcie v súbore *mqtt_callbacks.py*

na správy (príkazy) ako sú: reštart, zapnutie/vypnutie provisioning módu (nastavenie WiFi pripojenia modulu pomocou mobilnej aplikácie), spustenie OTA aktualizácie či vytvorenie fotografie a poslanie jej späť.

Funkcia na obrázku 4.3 zoberie každú MQTT správu, ktorá nebola poslaná na tému už s vopred zaregistrovanou callback funkciou alebo nie je medzi ignorovanými témami a pokúsi sa tému, na ktorú bola poslaná, vyhľadať ako názov zóny v databáze. Ak takéto zóny existujú a vo všetkých zadaných zónach sa zariadenie nachádza (prienik medzi nimi), tak sa mu táto správa prepošle na jeho unikátnu tému, na ktorú naslúcha.

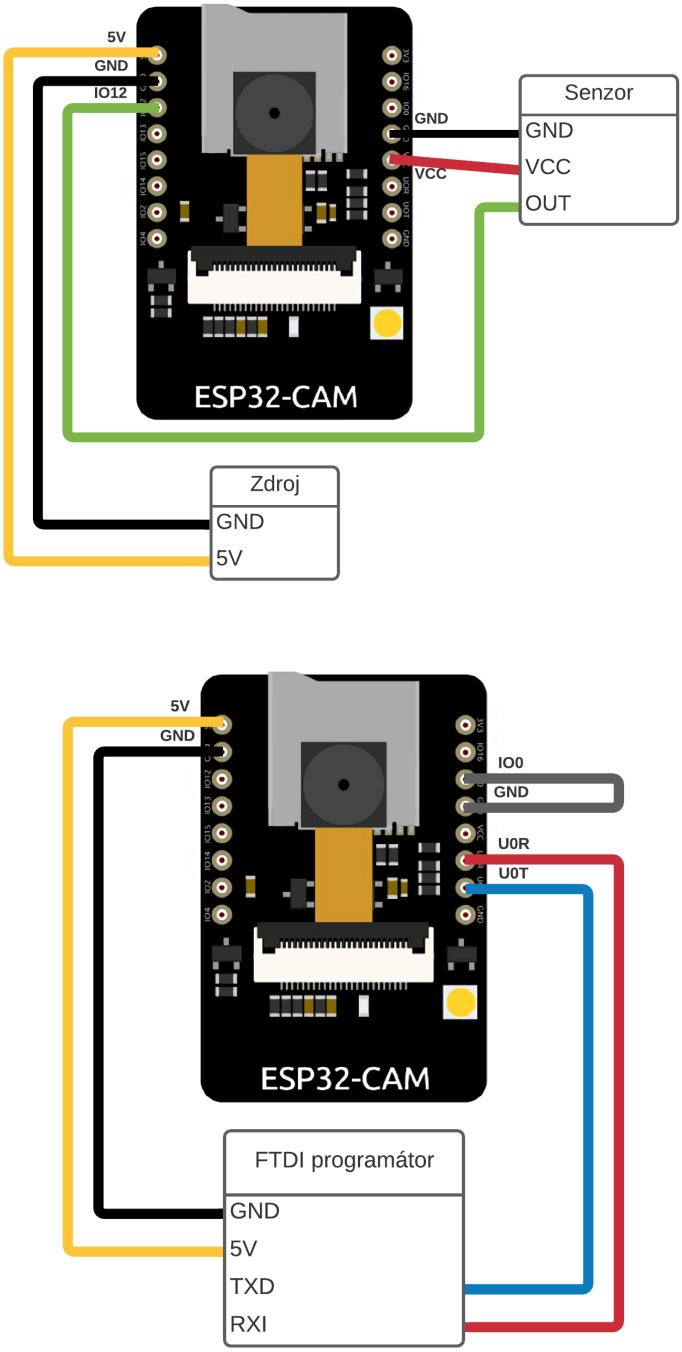
Príklad:

Správa je odoslaná na tému *home/kitchen* – ak existuje zariadenie, ktoré sa nachádza v týchto zónach v databáze, tak sa mu správa prepošle. Ak je správa poslaná na tému *home/pir-sensor/kitchen/group2*, tak sa zariadenie musí nachádzať v lokáciach aj *home*, aj *kitchen* a aj v skupine *group2* a musí byť typu *pir-sensor*. Na poradí zón nezáleží. Ak žiadne zariadenie nespĺňa daný prienik zón, správa sa zahodí.

Týmto spôsobom zariadenia nemusia danú tému naozaj odoberať, lebo rezolúciu za ne rieši centrálny ovládač a vzniká tak téma „virtuálna“. Navyše týmto prístupom systém získava výhodu, že na poradí úrovní tém nezáleží.

4.4 Moduly

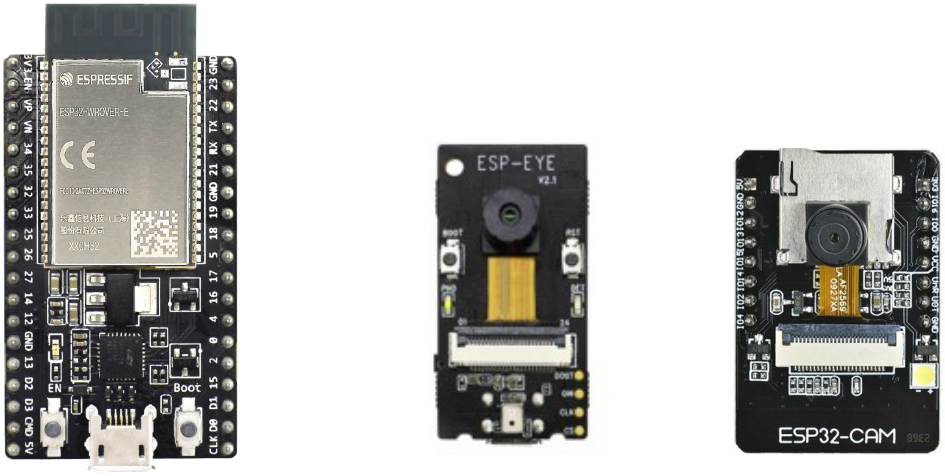
Pri implementácii vstavaných systémov (modulov) som využil 3 rôzne vývojové dosky, ktoré mi boli prístupné: ESP-EYE, ESP32-CAM a ESP32 DevKitC. Všetky využívajú čip ESP32. Prvá spomenutá doska neobsahuje GPIO piny, avšak má v sebe zabudovanú kameru. Kameru aj GPIO (viď. sekcia 2.1.3) piny má aj druhá spomenutá doska, pričom posledná doska obsahuje iba GPIO piny, na ktoré je možné pripojiť rôzne senzory. V mojom prípade som na obe ESP32-CAM a ESP32 DevKitC pripojil PIR senzor na snímanie pohybu, mikrovlnný na snímanie pohybu, senzor nárazového zvuku a piezoelektrický snímač vibrácií. Vývojové dosky sú znázornené na obrázku 4.5. Ich zapojenie je znázornené na obrázku 4.4.



Obr. 4.4: Hore obecná schéma zapojenia modulu (ESP32 vývojovej dosky a senzoru), dole zapojenie FTDI programátoru na ESP32-CAM za účelom nahrania firmvéru.
Originálny obrázok ESP32-CAM prevzatý z https://randomnerdtutorials.com

Konektivita a komunikácia

Moduly komunikujú s centrálnym ovládačom bežiacim na Raspberry Pi, inom lokálnom PC alebo cloude cez MQTT protokol, pričom sú do siete pripojené pomocou WiFi. Pri prvom



Obr. 4.5: Použité ESP32 vývojové dosky (zľava ESP32 DevKitC, ESP-EYE a ESP32-CAM)

spustení sa moduly dostanú do takzvaného „provisioning“ módu, počas ktorého sa správajú ako samy ako WiFi prístupové body. V tomto režime sa na ne dá pripojiť a pomocou mobilnej aplikácie **ESP SoftAP Prov**[5] nastaviť prihlasovacie údaje pre pripojenie reálny na prístupový bod WiFi. Tieto údaje sa uložia do perzistentnej pamäte NVS na zariadení a po reštarte sa automaticky použijú. Ak sa zariadenie ani po niekoľko pokusoch nevie pripojiť na prístupový bod, prejde do provisioning režimu a bude ho potrebné znova nastaviť. V normálnom režime sa pri spustení alebo na požiadanie registrujú cez centrálny ovládač do databázy.

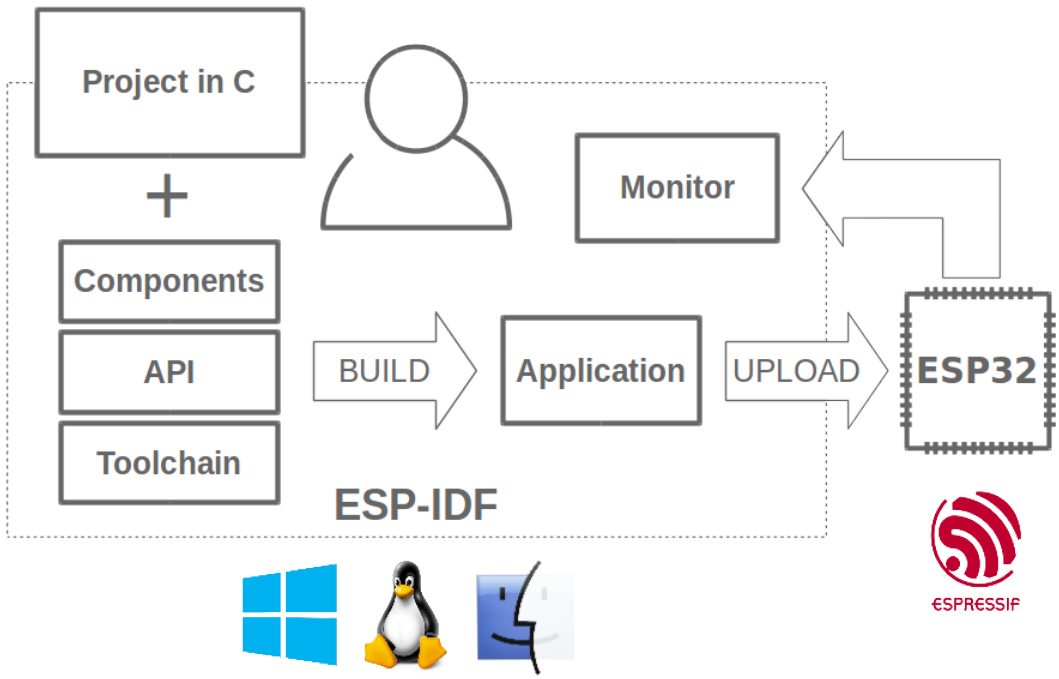
Firmvér

Firmvér modulov je programovaný v jazyku C s pomocou oficiálneho frameworku **ESP-IDF** (verzia „release/v4.2“). [6] Pri vyberaní programovacieho jazyku a nástrojov pre vývoj softvéru som zvažoval aj iné možnosti, ako sú skeče a jazyk pre Arduino platformu, ktorý je medzi nadšencami veľmi obľúbený, MicroPython pre jeho jednoduchosť a jazyk Nim[6] s knižnicou Nesper[7], kvôli jeho výborným vlastnostiam, prívetivej syntaxe a kompilácii do jazyku C s deterministickým garbage kolektorom, ktorý je vhodný aj pre hard real-time systémy. Nakoniec som si však vybral ESP-IDF kvôli tomu, že je to oficiálny nástroj poskytovaný Espressifom, generuje rýchly a kompaktný kód a je zrelý aj na používanie v produkcii – čo sú všetko vlastnosti, ktoré predošle spomínané možnosti nespĺňali.

ESP-IDF má taktiež veľmi dobrú dokumentáciu, knižnice a ukážky kódu, ktoré som pri riešení využil a inšpiroval sa nimi. ESP-IDF ako svoj základ používa open-source RTOS (viď. sekcia 2.1.1) s názvom FreeRTOS, ktorý je pozmenený na umožnenie používania viacjadrových procesorov na mikrokontroléroch. ESP-IDF toolchain používa na stavanie projektu program CMake.

V mojej práci využívam stabilnú verziu **release/v4.2**, pretože pri som narazil na chyby ktoré mi bránili v práci – nefunkčnosť MQTT.

[5] Aplikácia je open-source a je dostupná pre Android a iOS
[6] Zdroj: https://nim-lang.org
[7] Zdroj: https://github.com/elcritch/nesper



Obr. 4.6: Znázornenie funkcie ESP-IDF. Prevzaté z ESP-IDF dokumentácie [6]

OTA aktualizácie

Moduly majú schopnosť aktualizovať sa na diaľku pomocou OTA aktualizácii. Priestor v ich FLASH pamäti je rozdelený tak, že obsahuje partíciu *factory*, kde je nahraná základná verzia programu. Ďalej obsahuje dve OTA partície: *ota_0* a *ota_1*, ktoré sú používané pri takomto spôsobe aktualizácie. Firmvér je stiahnutý do *ota_1* partície a následne nahraný do *ota_0* partície. Skontroluje sa integrita dát a ukončí sa proces aktualizácie. Po reštarte zariadenia sa firmvér načíta z *ota_0* partíce, ktorá obsahuje jeho novú aktualizovanú verziu. Starý firmvér zostáva zachovaný vo *factory* partícii.

Príkazy

Moduly vedia cez MQTT správy prijímať viacero rôznych príkazov. Zasielané sú na unikátnu MQTT tému konkrétneho modulu. Ich funkcionalita je implementovaná vo funkcii *mqtt_data_event_handler(esp_mqtt_event_handle_t event)* v súbore *mqtt.c* v priečinku *esp32 / main*.

Ide o príkazy:

- *restart* – reštartuje modul,
- *get_photo* – modul zašle podlednú odfotenú fotografiu ako binárne dáta na MQTT tému *device/<MAC adresa modulu>/photo-output*,
- *take_photo* – modul vyhotoví novú fotografiu a zašle ju ako binárne dáta na MQTT tému *device/<MAC adresa modulu>/photo-output*,

- *ota_update* – modul sa pokúsi stiahnuť nový firmvér z predkonfigurovanej URL adresy a aktualizovať sa,
- *trigger-provisioning-next-boot* – po reštarte modulu modul zabudne svoje nastavené prihlasovacie údaje do WiFi siete a
- *untrigger-provisioning-next-boot* – zruší účinok predošlého príkazu, pokiaľ sa zariadenie ešte nereštartovalo.

Nastavenie firmvéru

Firmvér rôznych modulov je skompilovaný z tých istých zdrojových súborov. Pred prekladom ho je potrebné nastaviť tak, aby bol vygenerovaný binárny súbor správny. ESP-IDF projekt sa nastavuje pomocou príkazu *idf.py menuconfig*, ktorý otvorí grafický nástroj pre modifikáciu nastavení projektu – relevantné menu nastavení sú zobrazené na obrázku 4.7.

Momentálne sú implementované moduly:

- ESP32 modul bez kamery s digitálnym senzorom,
- ESP32 modul s kamerou a digitálnym senzorom a
- ESP32 modul iba s kamerou.

Obsluha analógových senzorov nie je implementovaná preto, lebo som žiaden analógový senzor nemal k dispozícií. PIR senzor, mikrovlnný senzor, senzor nárazového zvuku a aj piezoelektrický snímač vibrácií boli všetko digitálne senzory.

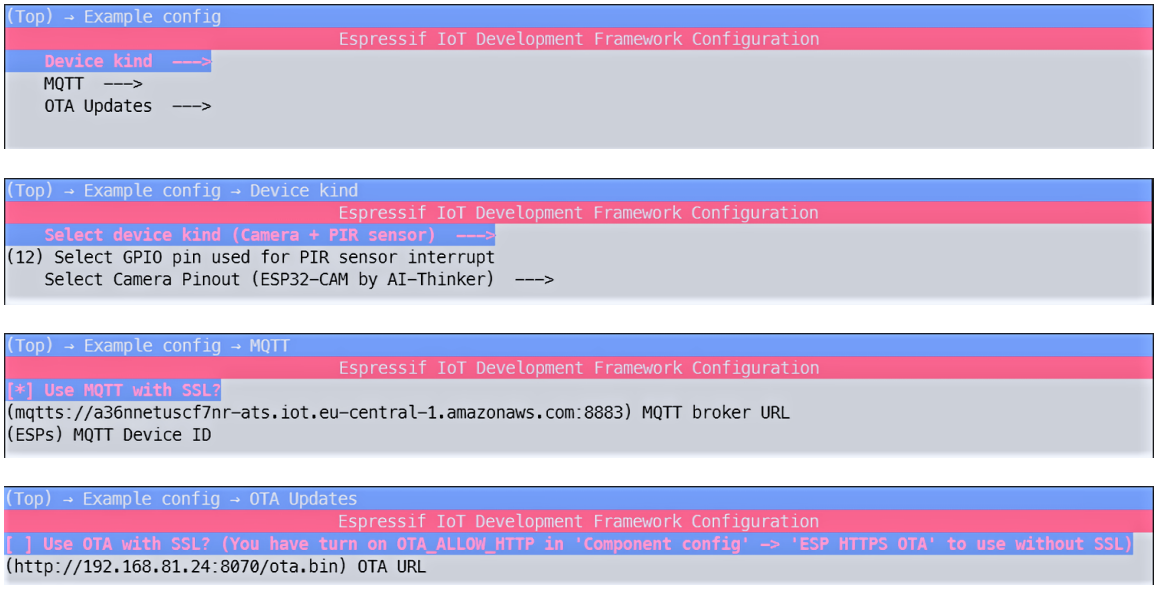
GPIO pin, na ktorý je senzor pripojený má predvolenú hodnotu nastavenú na 13, avšak ten je možný zmeniť pomocou *idf.py menuconfig* v Example config menu tohto projektu. Pred kompiláciou je potrebné nastaviť adresu MQTT brokera, na ktorý sa bude zariadenie pripájať a adresu URL adresu servera, ktorý bude poskytovať aktualizované verzie firmvéru pre OTA aktualizáciu. Nastavenie WiFi prebieha pomocou provisioning módu a mobilnej aplikácie, čo je popísane v podsekcii Konektivita a komunikácia. Predvolené nastavenia celého ESP32 modulu sa nachádzajú v súbore *sdkconfig.defaults*.

4.4.1 ESP-EYE modul

Modul s ESP-EYE je najvhodnejší ako bezpečnostná kamera, ktorá je neustále v pohotovostnom režime. Je to preto, lebo ako som už spomínal vyššie, neobsahuje žiadne GPIO piny, na ktoré by sa dal pripojiť senzor alebo externé napájanie. Kvôli tomu musí byť modul napájaný z USB portu, ktorý sa na doske nachádza. Absencia senzorov taktiež vylučuje režim spánku, pretože nemôže dojsť k externému prerušeniu a zobudeniu zariadenia. Modul je schopný na povel zaslaný cez MQTT vyhotoviť fotografie a cez MQTT ich poslať späť. Nevie zmeniť stav systému.

4.4.2 ESP32 DevKitC modul

Na modul s ESP32 DevKitC je možné pripojiť viacero senzorov, keďže má vyvedených až 34 GPIO pinov. V tomto ohľade je teda najuniverzálnejší. Je možné napájať ho aj cez USB, aj cez GPIO piny. Vhodný je na miesta kde nie je potrebná kamera.



Obr. 4.7: Ukážka menu nastavenia ESP-IDF projektu a jeho podmenu

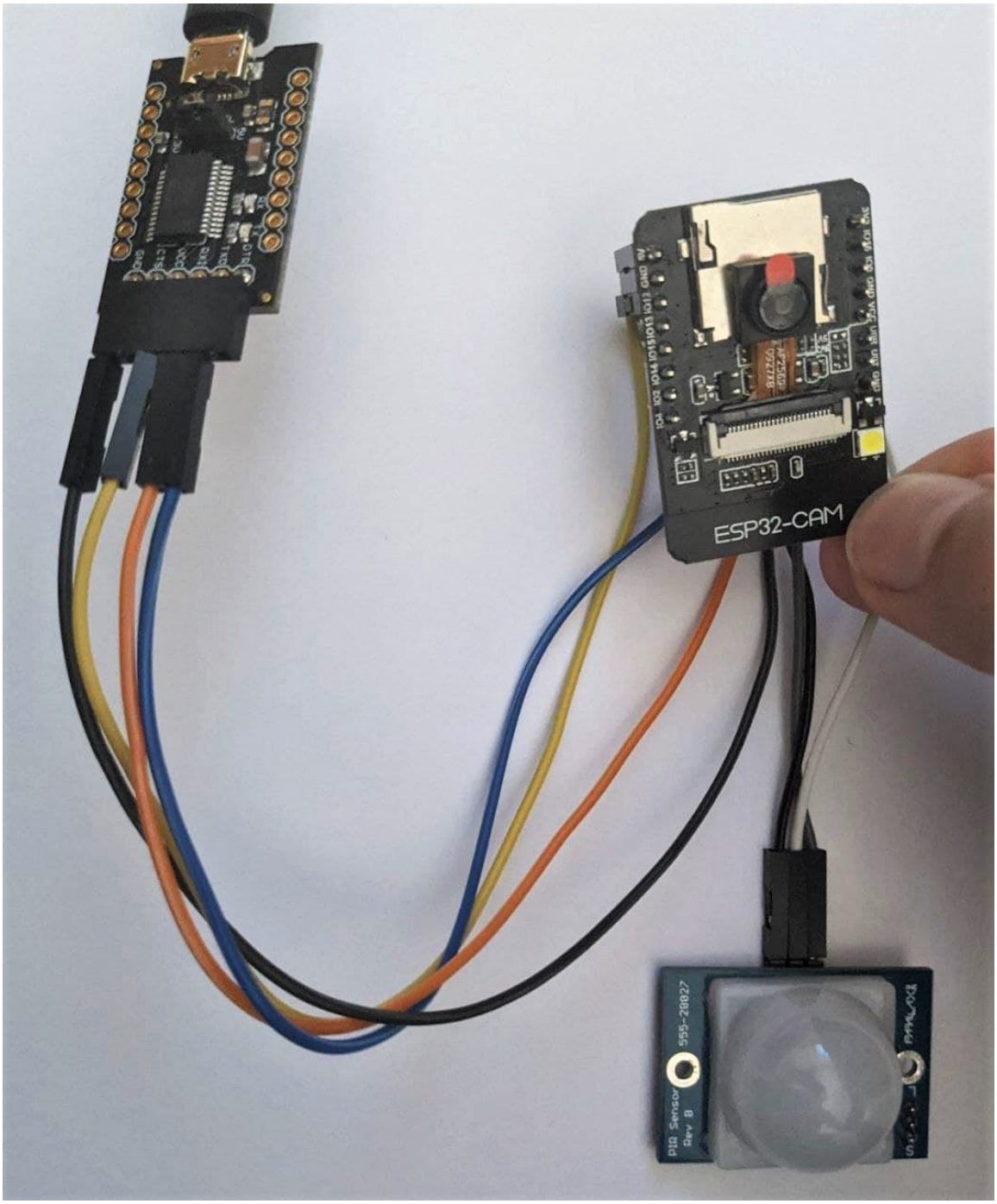
4.4.3 ESP32-CAM modul

Modul s ESP32-CAM je zlatá stredná cesta medzi dvoma predošle spomenutými modulmi. Obsahuje kameru a zároveň aj 16 GPIO pinov. Neobsahuje však USB, takže je potrebné ho obsluhovať s FTDI programátorom, ktorý tvorí most medzi modulom a PC pri programovaní alebo monitorovaní modulu. Neprítomnosť USB znamená aj to, že sa musí napájať pomocou 3.3V alebo 5V pinu na doske. Zvládne to isté čo modul s ESP-EYE (ohľadne kamery) a navyše má možnosti spojené s prípadne pripojenými senzormi. Je možné ho vložiť do režimu spánku kvôli menšej spotrebe energie a zobudiť ho prerušením zo senzoru na GPIO pine. Avšak v tomto režime sa bude chovať viacmenej ako samostatná jednotka, ktorá sa vie zobudiť iba na prerušenie, odoslať informácie do centrálneho ovládača a znovu sa uspať – nebude možné ho ovládať na diaľku. Tento režim nie je implementovaný. Zapojený ESP32-CAM s PIR senzorom sa nachádza na obrázku 4.8. Obrázky ESP32-CAM modulu so zapojenými inými senzormi sa nachádzajú v prílohe B.

4.5 MQTT broker

Ako MQTT broker som využil Amazon cloud (viď. sekcia 2.2.1) služby **AWS**, konkrétne službu *IoT core*. Urobil som to preto, aby som sa mohol do môjho systému pripojiť z celého sveta a nie iba z lokálnej siete. AWS som si vybral aj preto, lebo sú jeden z najznámejších poskytovateľov takýchto služieb a majú veľký výbery poskytovaných cloud služieb, ktoré by som v budúcnosti alebo ďalšej implementácii mohol použiť – napríklad hosting celého systému na cloude (teda nie len MQTT broker, ale aj riadiaca jednotka a databáza). AWS odporúča použitie TLS šifier ECDHE-ECDSA-AES128-GCM-SHA256 alebo ECDHE-RSA-AES128-GCM-SHA256, no ak ich zariadenie nepodporuje, algoritmus vyjedná inú slabšiu šifru.

Na autentifikáciu používa asymetrickú kryptografiu (kryptografiu verejným kľúčom) a pre každé pripojené zariadenie sú vydané vlastné certifikáty, ktoré sú pri moduloch vložené do jeho firmvéru pri kompilácii pričom sa musia nachádzať v priečinku *esp32 / main*



Obr. 4.8: ESP32-CAM modul s pripojeným PIR senzorom a FTDI programátorom (používaný ako most pre napájanie cez USB)

/ secret / amazon_certs mať mená *amazon_cert.pem.crt* pre SSL/TLS certifikát, *amazon_private.pem.key* pre privátny kľúč a *amazon_root_ca.pem* pre certifikát certifikačnej autority.

Ak riadiaca jednotka využíva asymetrickú kryptografiu, jej kľúče sú uložené v priečinku *server / secret / amazon_certs* s názvami *amazon_root_ca.pem*, *server_cert.pem.crt* a *server_private.pem.key*.

Šifrovaná MQTT komunikácia prebieha na porte 8883, zatiaľ čo obyčajná nešifrovaná previeha na porte 1883.

Pri vývoji systému som taktiež používal lokálny MQTT broker bežiaci na Raspberry Pi 4B, ktorý na autentifikáciu používal menu a heslo spolu so TLS/SSL pripojením. Pri použití takéhoto typu zabezpečenia sa meno a heslo napíše ako súčasť URL MQTT brokera v nastaveniach cez *idf.py menuconfig* nástroj.

4.6 Zabezpečenie

V tejto sekcii budem vysvetľovať zabezpečenie jednotlivých častí systému, ktoré sú rozdelené do podsekcií nižšie. Zabezpečenie nie je primárny cieľ tejto práce a preto som ho riešil len okrajovo. Pri reálnom nasadení by bolo treba bezpečnosť kódu prekontrolovať, prehodnotiť a prípadné chyby vyriešiť. ESP-IDF používa knižnicu mbedTLS na sprostredkovanie TLS/SSL funkcionality pre EPS32 mikrokontrolér.

OTA aktualizácie

Aktualizácie na diaľku sú zabezpečené pomocou SSL certifikátu, ktorý sa vloží do samotného firmvéru modulov a je použitý na autentifikáciu HTTPS serveru, ktorý zdieľa aktualizačný súbor. Toto zabezpečenie je možné vypnúť v nastaveniach pred stavbou firmvéru *idf.py menuconfig* a dovoliť modulu prijať nový firmvér aj z obyčajného HTTP serveru, ale na iné ako testovacie účely to neodporúčam.

Komunikácia

Komunikácia medzi modulmi a riadiacou jednotkou prebieha pomocou MQTT protokolu cez MQTT broker, ktorý je na Amazon AWS cloude. Bezpečnosť komunikácie a pripojenia sa na broker je taktiež riešená pomocou SSL certifikátov, ktoré sú vydané AWS cloudom riadiacej jednotke a jednotlivým modulom, kde sú pribalené do ich firmvéru.

Pri lokálom brokeri je zabezpečenie možno riešiť aj iným spôsobom a to je pomocou TLS a autentifikácie pomocou mena a hesla, ktoré by bolo taktiež pribalené do binárneho súboru v jednotlivých moduloch.

Takýmto spôsobom sa na MQTT broker nevie pripojiť cudzie zariadenie, ktoré by mohlo spôsobiť škody alebo odpočúvať komunikáciu medzi zariadeniami.

Moduly a riadiaca jednotka sú však stále schopné pripojiť sa aj na nezabezpečený MQTT broker, pričom danú možnosť opäť neodporúčam.

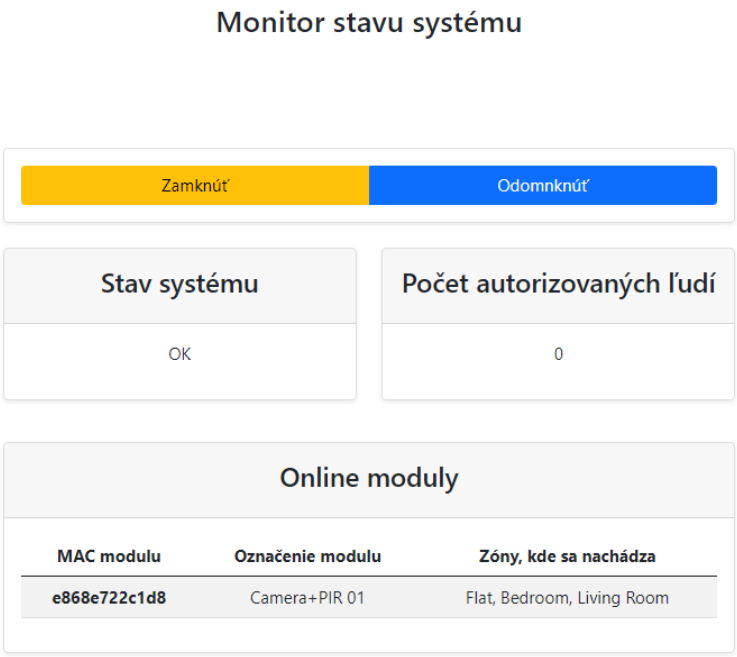
4.7 Grafické rozhranie

Moja práca neobsahuje návrh grafického rozhrania, pretože táto časť práce mala byť súčasťou diplomovej práce iného študenta a naše práce mali spolu spolupracovať. Preto implementujem iba jednoduché grafické rozhranie určené pre samostatnú demonštráciu systému. Znázornené je na obrázkoch 4.9 a 4.10.

Na tvorbu grafického rozhrania som sa rozhodol použiť webový framework **Flask**[8] pre Python, čo mi umožňuje jednoduchú integráciu so zvyškom systému a tým pádom je samotné rozhranie obyčajným HTML dokumentom. Pre estetický a efektný vzhľad používam sadu nástrojov kaskádových štýlov **Bootstrap**[9]. Riadiaca jednotka pri svojom štarte spustí lokálny web server na adrese *localhost:5000*, ktorý hostuje toto grafické rozhranie. Na pripojenie z iných zariadení je potrebné zameniť *localhost* za IP adresu zariadenia, na ktorom je program riadiacej jednotky spustený. Zdrojový kód sa nachádza v súbore *mqtt_ui.py*.

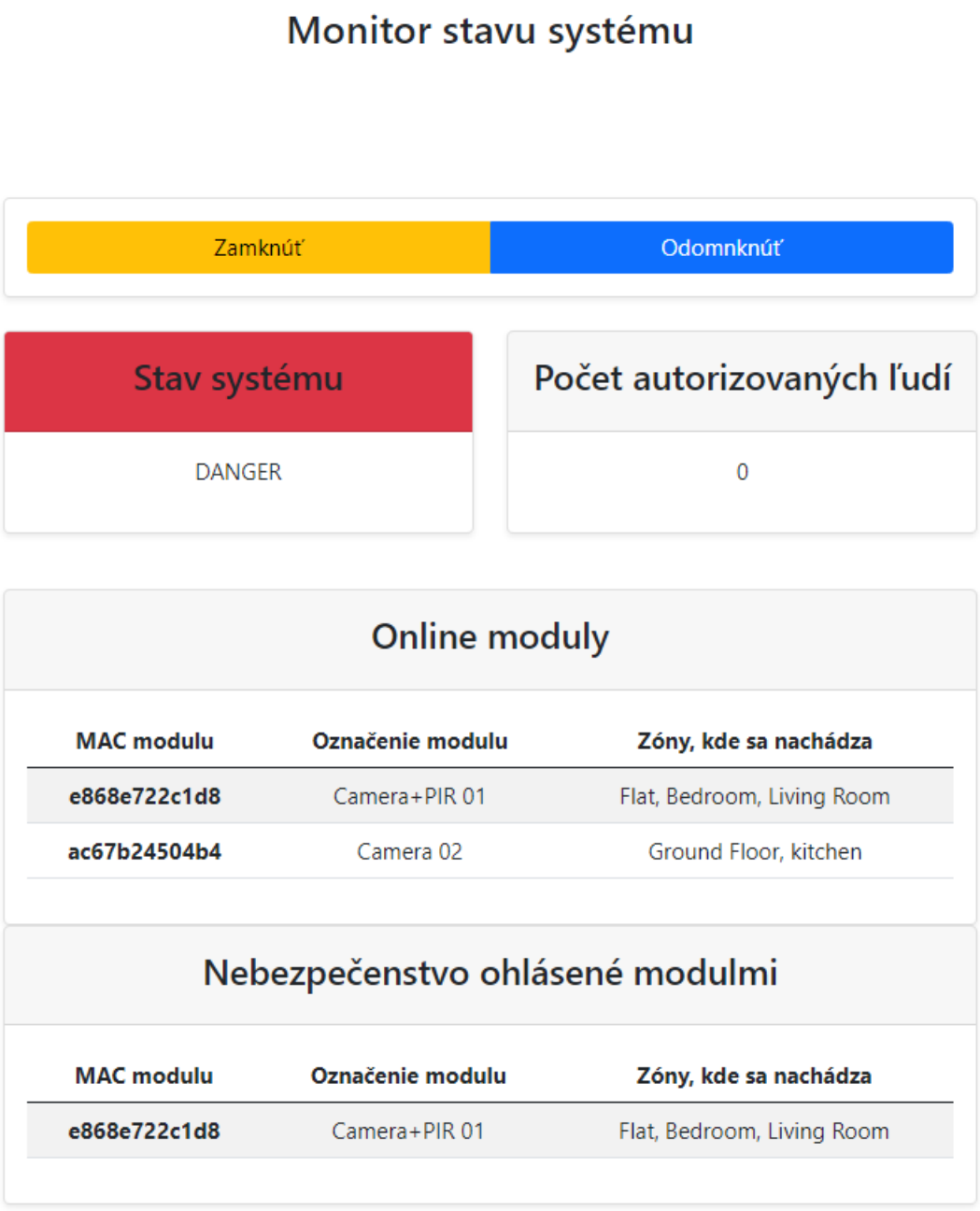
Webové grafické rozhranie vie:

- zmeniť stav systému na odomknutý,
- zmeniť stav systému na zamknutý,
- zobraziť stav systému (nezabezpečený = OK, zabezpečený = ALERT a napadnutie = DANGER),
- zobraziť, ktoré zariadenia stav napadnutia vyvolali a kde sa nachádzajú,
- zobraziť, koľko ľudí sa autentifikovalo pomocou prepojeného systému z diplomovej práce „Inteligentní přístupový terminál na platforme ES32“, ktorá je súčasťou väčšieho riešenia.



Obr. 4.9: Ukážka webového rozhrania v odomknutom stave systému

[8]Zdroj: https://flask.palletsprojects.com
[9]Zdroj: https://getbootstrap.com



Obr. 4.10: Ukážka webového rozhrania v napadnutom stave systému

Kapitola 5

Testovanie

Táto kapitola sa zaoberá testovaním návrhu systému a jeho jednotlivých implementovaných častí.

Databáza

Testovanie databázy prebiehalo počas jej implementácie najprv manuálne, neskôr som si vytvoril pomocné Python skripty, ktoré mi databázu naplnili testovacími dátami. Testoval som správnosť návrhu databázy, t.z. čo všetko majú dané tabuľky obsahovať, ako majú byť previazané. Databáza si prešla viacerými iteráciami implementácie než som sa dostal ku súčasnej. Testovací súbor pre databázu je *test_database.py* a *test_init.py*.

Komunikácia

Testovanie komunikácie a interakcie riadiacej jednotky a modulov som vykonával počas vývoja týchto častí systému. Keďže komunikácia medzi nimi prebieha cez MQTT broker hostovaný na Amazon AWS IoT core cloude, na jej sledovanie som používal integrovaný MQTT klient na AWS web stránke, pričom som odoberal správy pomocou wildcard MQTT témy „#“, pomocou ktorej som dostával správy publikované na každej MQTT téme.

Automatizácia

Testovanie prvku automatizácie, konkrétne funkčnosti virtuálnych MQTT tém spomenutých v sekcii 4.3.1 je možné pomocou súboru *test_automation_topic_parser.py*, ktorý berie argument pri spustení, čo by mala byť požadovaná virtuálna MQTT téma, napríklad *house/kitchen*, a skript vypíše na STDOUT všetky zariadenia z databázy, ktoré sa v zónach „house“ a zároveň „kitchen“ nachádzajú.

Moduly samotné som testoval nie len ich monitorovaním pomocou príkazu *idf.py monitor* na PC, ale snažil som s ich testovať aj v prostredí domácnosti zapojené na batériu. Ich testovanie bolo jednoduché vzhľadom k tomu, že všetky využívané senzory v moduloch majú digitálny (binárny) výstup.

Riadiacu jednotku som testoval a upravoval za behu rovno počas jej vývoja, takže chyby som mohol nájsť prakticky hneď.

Testovacie Python skripty sa sa nachádzajú v priečinku *esp32* a ich názov sa začína slovom *test*.

Kapitola 6

Diskusia

V tejto kapitole píšem o problémoch, na ktoré som pri práci narazil a navrhujem vylepšenia systému.

6.1 Problémy pri riešení práce

Práca sa samozrejme nezaobišla bez pár problémov, na ktoré som popri jej riešeniu natrafil. Zväčša išlo o problémy spojené so softvérovou stránkou práce, kde som nájdené nedostatky alebo chyby, alebo vzniknuté nedorozumenia z nedostatočného pochopenia dokumentácie používaného softvéru riešil priebežne.

Problém s mesh sieťou

Počas návrhu riešenia som sa zaoberal s myšlienkou využiť na riešenie mojej práce technológiu mesh sietí, kde by moduly takúto sieť tvorili. V mesh sieti sa zariadenia stávajú uzlami siete, sú prepojené medzi sebou a vytvárajú určitú topológiu. V takejto sieti existuje jeden koreňový uzol, ktorý je pripojený do klasickej počítačovej siete a všetka komunikácia v mesh sieti a komunikácia mimo mesh siete prechádza cez neho. Výhodou takejto siete je, že iba koreňový uzol musí byť v pripojený na klasickú počítačovú sieť (t.z. pri použití WiFi musí byť v jej dosahu), zatiaľ čo ostatné uzly nemusia, lebo komunikujú medzi sebou. Firma Espressif ponúka upravenú verziu svojho frameworku ESP-IDF nazvanú ESP-MDF (Mesh development framework), ktorý práve umožňuje takúto funkcionalitu. Pri snahe implementovať takýto systém som narazil na obmedzenia v samotnom ESP-MDF a to konkrétne v obmedzení fragmentácie posielaných dát medzi jednotlivými uzlami. Prenos dát väčších ako 8kB nie je podporovaný a vyriešenie tohto obmedzenia naprogramovaním kódu, ktorý rieši tento problém mi prišilo ako potenciálne veľká časová strata s prihliadnutím na to, že vyžitie mesh siete nemá až také veľké výhody od klasického riešenia.

Výhoda súčasného klasického riešenia je aj to, že každé zariadenie má vlastnú IP adresu a teda v prípade potreby viem k nemu pristupovať priamo. Taktiež nemá obmedzenie na veľkosť prenášaných dát, čo je pri prenášaní väčších súborov ako sú práve fotografie z modulov s kamerami nevyhnutnosťou.

6.2 Návrh vylepšení

eFuse, Secure Boot a šifrovanie FLASH pamäte

ESP32 mikrokontroléry obsahujú takzvanú 1024-bitovú eFuse, čiže v preklade elektronickú poistku, ktorá slúži ako perzistentný register s hodnotou buď 1 alebo 0. Je rozdelená do blokov v ktorých vedia byť uložené dáta. Po „vypálení“ (t.z. zmene) bitu z 0 na 1 ho už nejde zmeniť späť, keďže toto vypálenie je fyzické v hardvéri.

V eFuse sú často uložené kľúče napr. pre secure boot zariadenia a šifrovanie pamäte flash. V komerčnej verzii systému podobnému tomu, ako som v tejto práci vytvoril, by bolo ich použitie nutné pre dosiahnutie najvyššej možnej bezpečnosti zariadení.

Brána na internet

Aby bola modularita nasadenia systému kompletná, je potrebné v prípade nasadenia celého systému lokálne implementovať bránu (ang. gateway) na umožnenie komunikácie lokálneho MQTT brokera s internetom. Takýmto spôsobom môže mať používateľ bezpečnostného systému celý systém nasadený lokálne v jeho dome a zároveň využívať benefity pripojenia na internet, t.z. napríklad dostávanie notifikácií kdekoľvek na svete. Toto je zatiaľ možné len s verziou nasadenia systému, kde je MQTT broker mimo lokálnu sieť bežiaci na cloude, ktorý má verejnú IP adresu.

Lepšie UI a/alebo mobilná aplikácia

Tento návrh mi príde ako samozrejmosť, keďže moja súčasná implementácia grafického rozhrania je z väčšej časti improvizačná, vhodná hlavne na demonštráciu funkcionality systému a nie na každodenné používanie koncovým používateľom, ktorý by si mal zaobstarať môj systém ako funkčný produkt.

Využitie microSD karty

Vývojová doska ESP32-CAM obsahuje okrem kamery aj čítačku microSD kariet, ktorá by sa dala použiť ako lokálne úložisko fotografií, prípadne video záznamov z kamery pri vylepšenej verzii systému. Takto má modul používajúci túto funkcionalitu možnosť mať lokálne uložené ohromné množstvo dát, čo by bola veľká výhoda oproti terajšej implementácii, kde môžu mať zariadenia naraz uloženú len jednu fotografiu.

Kapitola 7

Záver

Cieľom diplomovej práce bolo vytvoriť systém na detekciu subjektov v stráženom priestore s využitím ESP32 mikrokontrolérov. Zámer práce splnený bol, systém je funkčný a preto riešenie diplomovej práce považujem za úspešné. Vždy je však miesto pre zlepšenie a to platí aj pre moju prácu, o čom som písal v kapitole Diskusia.

Systém bol vytvorený s myšlienkou modularity a jednoduchosti jeho rozšírenia, používajúci technológie určené priamo pre IoT zariadenia ako je napríklad komunikácia pomocou MQTT protokolu cez MQTT broker. Implementované boli tieto časti: moduly, ktoré informujú systém o udalostiach a sú spojením ESP32 mikrokontroléru a senzoru, riadiaca jednotka, teda program, ktorý dostáva informácie od modulov a vyhodnocuje ich – je to srdce systému, databáza, ktorá ukladá systémové dáta a jednoduché webové rozhranie na prezentáciu práce. Systém má viacero stavov, ktoré sú vyhodnocované riadiacou jednotkou. Automatizovaná je registrácia modulov do systému, odosielanie meraní zo senzorov a reakcia riadiacej jednotky zmenou stavu systému. Skúsený používateľ môže jednoducho pozmeniť správanie systému úpravou obslužných funkcií registrovaných MQTT tém alebo posielať manuálne modulom príkazy.

Pri realizácií riešenia som použil rôzne technológie, z hardvérových ESP32 mikrokontroléry a ich vývojové dosky, mini-počítač Raspberry Pi 4B, využíval som cloudové služby a používal rôzny softvér ako SQL databázu, programovacie jazyky C a Python, ESP-IDF framework, web framework Flask a iné. Vyriešil som mnoho problémov, pričom o existencii niektorých z nich som pred riešením diplomovej práce ani nevedel. Riešením tejto práce som sa teda získal skúsenosti z rôznych IT odvetí, za čo som veľmi povďačný. Skúsenosti som získal aj z časti riešenia práce, ktorá sa nedostala do finálnej verzie a to je napríklad práca s mesh sieťami. Každým rokom sa rapídne zvyšuje počet inteligentných IoT zariadení a systémov a preto si myslím, že zaoberať sa touto témou je nesmierne dôležité.

V tejto práci by som rád pokračoval aj vo voľnom čase, nakoľko ma téma veľmi zaujíma a v kvalitných IoT systémoch vidím v súčasnosti a v budúcnosti veľkú perspektívu.

Literatúra

[1] Hallová, M. *Cloud computing – definícia, výhody a nevýhody.* Slovenská poľnohospodárska univerzita v Nitre, 2013 [cit. 2021-05-01]. ISBN 978-8-055-20983-8. Dostupné z: http://www.slpk.sk/eldo/2013/zborniky/024-13/978-80-552-0983-8.pdf.

[2] Bar, M. a Ganssle, J. *Embedded Systems Glossary* [online]. [cit. 2021-05-01]. Dostupné z: https://barrgroup.com/embedded-systems/glossary.

[3] Bar, M. a Massa, A. J. Programming embedded systems: with C and GNU development tools. In:. O'Reilly., 2006, kap. Introduction [cit. 2021-05-01]. ISBN 978-0-596-00983-0. Dostupné z: https://barrgroup.com/embedded-systems/books/programming-embedded-systems.

[4] Swaroop. *Embedded Systems Introduction* [online]. Codrey Electronics, 25. mája 2020 [cit. 2021-05-01]. Dostupné z: https://www.codrey.com/embedded-systems/embedded-systems-introduction.

[5] Wikipedia contributors. *System on a chip* [online], 23. apríla 2021 [cit. 2021-05-01]. Dostupné z: https://en.wikipedia.org/wiki/System_on_a_chip.

[6] Espressif. *ESP-IDF Programming Guide* [online]. [cit. 2021-05-09]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en.

[7] ESP32net. *The Internet of Things with ESP32* [online]. [cit. 2021-05-07]. Dostupné z: http://esp32.net.

[8] Wikipedia contributors. *ESP32* [online], 6. mája 2021 [cit. 2021-05-07]. Dostupné z: https://en.wikipedia.org/wiki/ESP32.

[9] Wikipedia contributors. *Internet of Things* [online], 30. apríla 2021 [cit. 2021-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Internet_of_things.

[10] Mozzila. *JSON - MDN Web Docs Glossary* [online]. [cit. 2021-05-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/JSON.

[11] Mozzila. *Working with JSON* [online]. [cit. 2021-05-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

[12] HiveMQ team. *MQTT Client and Broker and MQTT Server and Connection Establishment Explained* [online], 17. júla 2019 [cit. 2021-05-02]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment.

[13] HiveMQ team. *MQTT Topics Best Practices* [online], 20. augusta 2019 [cit. 2021-05-02]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices.

[14] OASIS Open. *MQTT Version 5.0: OASIS Standard* [online], 7. marca 2019 [cit. 2021-05-01]. Dostupné z: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[15] Wikipedia contributors. *Raspberry Pi* [online], 6. mája 2021 [cit. 2021-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi.

[16] FreeRTOS. *Why RTOS and What is RTOS?* [online]. [cit. 2021-05-01]. Dostupné z: https://www.freertos.org/about-RTOS.html.

Príloha A

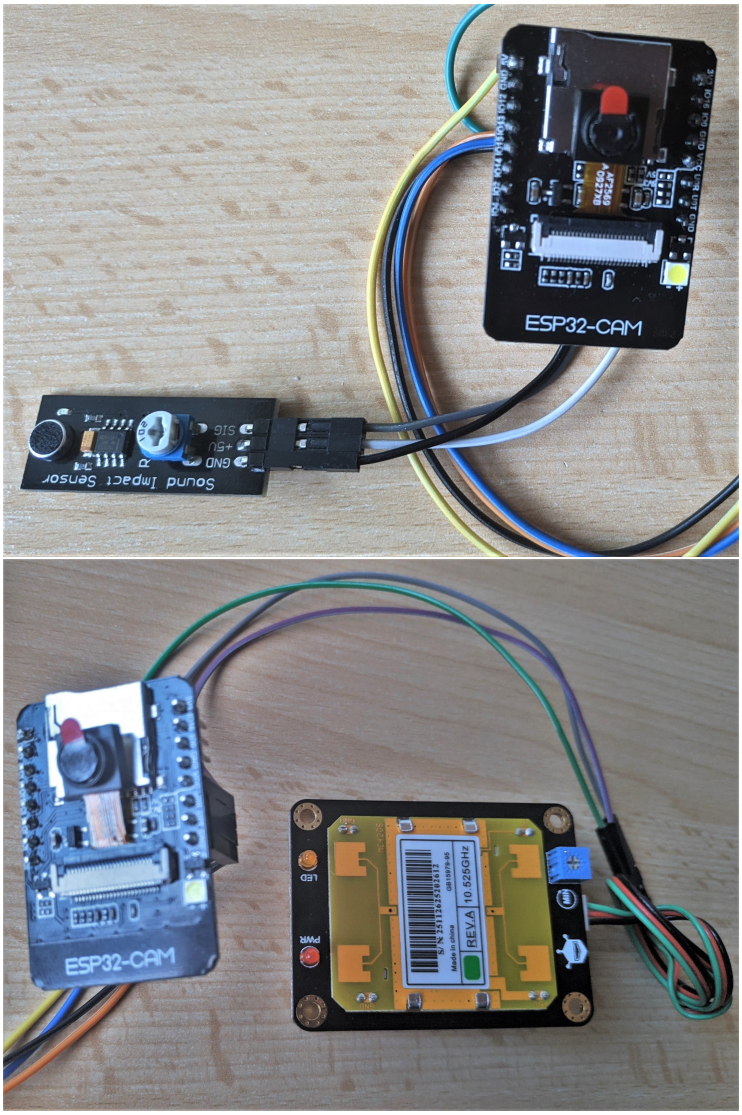
Konfiguračné súbory

```
{
    "settings": {
        "save_zones": true,
        "save_devices": true
    },
    "locations":
        [{
          "name": "home",
          "label": "Home",
          "parent": null
        },{
          "name": "groundfloor",
          "label": "Ground Floor",
          "parent": ["home"]
        }],
    "groups":
        [{
          "name": "night-lights",
          "label": "Night lights"
        }],
    "topics":
        [{
            "topic": "device/register",
            "datatype": "json",
            "callback": "device_register"
        },{
            "topic": "device/+/photo-output",
            "datatype": "binary",
            "callback": "photo_save"
        }]
}
```

Obr. A.1: Ukážka konfiguračného súboru pre centrálny ovládač a databázu

Príloha B

Obrázky zapojených modulov



Obr. B.1: Hore modul zložený z ESP32-CAM a senzoru nárazového zvuku, dole modul zložený z ESP32-CAM a mikrovlnného senzoru