VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VZDÁLENĚ SPRAVOVANÝ NÍZKOENERGETICKÝ INFORMAČNÍ DISPLEJ

REMOTELY MANAGED LOW-ENERGY INFORMATION DISPLAY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ZDENKO PETRUŠKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ GOLDMANN**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Petruška Zdenko**
Program: Informační technologie
Název: **Vzdáleně spravovaný nízkoenergetický informační displej**
Remotely Managed Low-Energy Information Display
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s moderními zobrazovacími jednotkami (LED, OLED, E-ink). Dostupné technologie porovnejte a zjistěte jakou mají energetickou náročnost.
2. Seznamte se s problematikou internetu věcí. Prostudujte a sumarizujte informace o platformě ESP32, především se zaměřte na bezdrátovou komunikaci a možnosti řízení spotřeby.
3. Navrhněte dálkově řízený vizualizační panel s E-Ink displejem a platformou ESP32. Pro správu zařízení navrhněte server, který bude se zařízením komunikovat prostřednictvím TCP/IP. Vizualizační panel bude umožňovat zobrazení obrázku a textu.
4. Implementujte firmware pro platformu ESP32 a serverovou aplikaci. Serverovou část implementujte jako multiplatformní aplikaci v libovolném programovacím jazyce.
5. Proveďte otestování a zároveň realizujte experimenty zaměřené na zhodnoceni energetické náročnosti vizualizačního panelu.

Literatura:

- CHOU, Timothy. Precision-Principles, Practices and Solutions for the Internet of Things. McGraw-Hill Education, 2017.
- SPANULESCU, Sever. ESP32 Programming for the Internet of Things. Lulu.com, 2018.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

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

Vedoucí práce: **Goldmann Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cieľom tejto práce je vytvorenie vizualičnej jednotky a aplikácie pre jej správu. Vizualizačná jednotka sa skladá z displeja založeného na technológií E-ink a platformy ESP32. Jednotka umožňuje zobrazenie grafického aj textového obsahu. Jednotka sa periodicky prepína medzi režimom spánku a aktívnym režimom, v ktorom si stiahne a zobrazí aktuálne dáta. Jednotka a aplikácia komunikujú bezdrôtovo. Webová aplikácia pre správu je implementovaná v jazyku Python. Aplikácia umožňuje správu viacerých jednotiek, medzi jednotkami je možné obsah zdielať.

Abstract

The main goal of this thesis is to create visualization panel and an app used for management of content on this panel. Visualization panel is composed of a platform ESP32 and an E-ink display. The panel and the app communicate with each other using wireless connection. The app is written using Django web framework. The app provides management of multiple panels and displayed content can be shared by panels.

Kľúčové slová

Internet vecí, ESP32, ESP8266, E-ink, EPD, Arduino, Vstavané systémy, Web, Webová aplikácia, HTTP, Django

Keywords

Internet of things, IoT, ESP32, ESP8266, E-ink, EPD, Arduino, Embedded systems, Web, Web application, HTTP, Django

Citácia

PETRUŠKA, Zdenko. *Vzdáleně spravovaný nízkoenergetický informační displej.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Vzdáleně spravovaný nízkoenergetický informační displej

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Tomáša Goldmanna. Uviedol som všetky literárne pramene, publikácie a dalšie zdroje, z ktorých som čerpal.

.....
Zdenko Petruška
28. mája 2020

Poďakovanie

Týmto by som sa rád poďakoval svojmu vedúcemu práce za jeho odborné vedenie, rady a možnosť konzultácie vždy, keď som to potreboval pre vypracovanie práce.

Obsah

Kapitola 1

Úvod

Vstavané systémy sa dnes stávajú viac populárnymi ich aplikácie uľachčujú bežný život. Občas je treba zobraziť statickú informáciu, ktorú je však potrebné niekedy aktualizovať. Toto môže uľahčiť navrhovaný sýstém, ktorý bude túto informáciu zobrazovať na displeji a informáciu bude možné aktualizovať pomocou webovej aplikácie. Aby bolo toto riešenie možné aplikovať v praxi, nemalo by byť závislé na pripojení do elektrickej siete. Preto by malo toto riešenie mať takú spotrebu, aby ho bolo možné napájať pomocou batérií, prípadne solárneho panela. Táto práca bude venovananá návrhu a implementácií takéhoto riešenia. Niektoré firmy už dnes ponúkajú podobné riešenia s použitím displeja založeného na techológií elektronického papiera, napríklad pre zobrazovanie cestovných poriadkov na autobusových zastávkach.

Práca sa v druhej kapitole zaoberá zobrazovacími technológiami, ktoré sa používajú pri výrobe plochých displejov (*flat panel display*). Ploché displeje sú vhodné pre vizualizačný panel. V kapitole sú vymenované parametre, ktorými je vhodné riadiť sa pri výbere displeja pre cieľovú aplikáciu. Ďalej sú tu objasnené princípy najpoužívanejších zobrazovacích technológií a skonzultované oblasti možného použitia danej technológie spolu s faktormi, ktoré ovplyvňujú spotrebu. Tiež sú tu ukázané príklady displejov ponúkaných na trhu, u ktorých sú uvedené ich parametre vrátane spotreby.

V tretej kapitole sú vysvetlené základné pojmy spojené s problematikou internetu vecí. Ďalej sa kapitola venuje oblasti vstavaných systémov, konkrétne určovaniu ich spotreby. Sú tu objasnené základné technológie použité v implementačnej časti. V kapitole sú tiež spracované informácie o platformách *ESP32* a *ESP8266* potrebné k implementácií vizualizačnej jednotky vrátane informácií o rôznych režimoch spotreby týchto platforiem.

Vo štvrtej kapitole je popísaný návrh vizualizačnej jednotky spolu so serverovou aplikáciou pre jeho správu. Tiež je tu definovaná komunikácia aplikácie s vizualizačnou jednotkou a popísané použité technológie a knižnice. V piatej kapitole sú popísané detaily implementácie aplikácie a jednotky.

V šiestej kapitole je zhodnotené riešenie a tiež sú tu popísané experimenty pre určenie spotreby vizualizačnej jednotky. Na základe vykonaných experimentov je popísaný pracovný cyklus jednotky a jej spotreba v jednotlivých fázach cyklu. Ďalej sú navrhnuté možnosti pre optimalizáciu energetickej náročnosti, niektoré z nich sú následne aplikované.

Kapitola 2

Technológie pre zobrazovanie obrazu

Existuje množstvo zobrazovacích technológií, ktoré sa neustále vyvýjajú. Niektoré zobrazujú dvoj-dimenzionálne projekcie, iné troj-dimenzionálne (napr. 3D projektory). Pri implementácií vizualizačnej jednotky sa počíta s použitím displeja s plochou obrazovkou (*Flat panel display*), preto bude nasledujúca kapitola venovaná tejto kategórií. Technológie displejov s plochou obrazovkou možno rozdeliť do dvoch rodín – na tie, ktoré využívajú podsvietenie a tie, ktoré ho nevyužívajú. V nasledujúcich podkapitolách sú tieto rodiny popísané bližšie.

Nasledujúci prehľad parametrov zobrazovacích technológií vychádza z [23] a [6]. Podľa týchto publikácií má každá technológia svoje limity jednotlivých parametrov. Parametre je potrebné zohľadníť pri výbere displeja pre cieľovú aplikáciu. Medzi základné parametre možno zaradiť:

1. **Veľkosť a pomer strán**. Veľkosť je typicky definovaná ako vzdialenosť protiľahlých rohov displeja. Často býva vyjadrená v palcoch. Pomer strán býva vyjadrený ako pomer šírky a dĺžky, napr. 4:3.

2. **Rozlíšenie**. Displeje sa bežne skladajú z matice bodov (pixely). Ich množstvo definuje rozlíšenie, alebo kvalitu obrazu. U farebných displejov je každý pixel rozdelený na 3 subpixely (RGB).

3. Hodnota **Pitch** je hodnota vzdialenosti susedných pixelov typicky udávaná v milimetroch. Tiež je ukazateľom kvality obrazu.

4. **Jas a farba**. Pre správnu čitateľnosť by mal mať displej práve taký jas, aký majú objekty ním zobrazované aj v skutočnosti. Množstvo farieb, ktoré dokáže displej zobraziť, je tiež dôležité pri jeho výbere pre konkrétnu aplikáciu. V súvislosti s jasom je dalej možné definovať si kontrast – rozdiel medzi svietivosťou najsvetlejšieho a najtmavšieho bodu, ktorú dokáže zobrazovacia jednotka zobraziť.

5. **Pozorovací uhol** (*viewing angle*). Podľa jednej z častých interpretácií pozorovací uhol definuje, v akom uhle musí stáť pozorovateľ, aby rozpoznal informáciu zobrazovanú na displeji. Konkrétne ide o hodnotu uhla spojnice stredu displeja s jeho pozorovateľom a normály plochy displeja pretínajúcej jeho stred. Výsledný pozorovací uhol bude tak dvojnásobkom hodnoty definovanej predchádzajúcou vetou.

6. **Spotreba** displeja je tiež dôležitým faktorom pri jeho výbere. U niektorých displejov rastie spotreba lineárne s ich veľkosťou. U iných technológií rastie spotreba neúmerne k veľkosti a tak sú tieto technológie vhodné len pre displeje menších rozmerov.

2.1 Technológie nevyužívajúce podsvietenie

U týchto technológií vyžarujú svetlo jednotlivé body (*emissive family*), takže nie je potrebné vyvýjať energiu pre podsvietenie. Problematickou sa však stáva distribúcia energie pre jednotlivé body.

Typickými predstaviteľmi rodiny, u ktorej sa nevyužíva podsvietenie, sú technológia katódovej trubice CRT (*cathode ray tube*), technológia diódy vyžarujúcej svetlo LED (*Light-emitting diode*) a organické elektroluminiscentné diódy OLED.

2.1.1 Technológia katódovej trubice

Tento odsek vychádza z [6]. Technológia katódovej trubice je založená na princípe generovania svetla excitovaním luminiscentého materiálu. Hlavnými komponentami sú elektrónová tryska (*electron gun*), vychylovacia cievka (*deflection coil*) a vrstva katódoluminiscentného fosforu. Najskôr elektrónová tryska vyšle elektrón pomocou vychylovacej cievky smerom k forsforovej vrstve. Potom katódoluminicentný fosfor premení energiu elektrónu na svetlo. Oblasť medzi elektrónovou tryskou a fosforom je vyplnená vákuom pre zníženie mechanického napätia. Elektróny sú na katóde generované pomocou odporového ohrievača pri teplote približne 600 °C. Elektrónový lúč je ďalej zosilnený pomocou niekoľkých Einzelových šošoviek a ďalej nasmerovaný pomocou vychylovacej cievky. Vychylovacia cievka cievka ohýba trajektóriu elektronu pomocou magnetického pola. Vychylovacia cievka je tiež navrhnutá tak, aby zabránila degradácií ostrosti spôsobenej dopadaním elektrónu na vrstvu fosforu vo veľkých uhloch. Rôzne typy používaného fosforu majú rôznu životnosť a líšia sa tiež svojou granularitou, ktorá má vplyv na kvalitu výsledného obrazu.

Technológia CRT sa vyznačuje vysokou hĺbkou farieb a kontrastom, no pre jej vysokú energetickú náročnosť je dnes na ústupe. Na jej základoch vznikajú dnes aj nové technológie, ktoré sľubujú veľmi nízku energetickú náročnosť. Jedným z príkladov je technológia FED, ktorá je založená na princípe matice vyžarujúcich bodov (*Field emission display*). Táto technológia narozdiel od technológie CRT využíva milióny elektrónových trysiek miesto jednej. Vďaka tomu by mohla v budúcnosti ponúknuť displeje, ktoré budú disponovať vysokou hodnotou maximálneho pozorovacieho uhla, malou hrúbkou a nízkou spotrebou.

Tento odsek vychádza z [26]. Technológia FED pracuje s maticou zarovnaných kovových kužeľov mikroskopickej veľkosti. Kužele môžu byť vyrobené z kremíku, molybdénu, prípadne uhlíkových nanotrubiek (CNT FED). Jeden pixel zastupujú tisíce takýchto kužeľov. Každý je rozdelený na pásiky – čierny, zelený a červený (schéma RGB). Spotreba displeja závisí na zobrazovanom obsahu, t.j. aktivujú sa len vybrané pixely. Spodnú vrstvu tvorí matica kužeľov. Na nej sa nachádza sústava elektród, ktoré generujú elektrický prúd (tzv. *gate elektrode*). Elektróny smerujú k anóde, ktorá sa nachádza medzi vrstvou fosforu a vrchným sklom. Ďalšie detaily o použití nanotrubiek v tejto technológií sú diskutované v [30] a ďalšie možnosti vývoja a využitia v rámci článku [26].

2.1.2 Technológie LED a OLED

Tento odsek vychádza z [6]. LED diódy sa používajú ako podsvietenie pre LCD displeje. Okrem toho existujú displeje, ktorých pixely sú reprezentované trojicou veľmi malých LED diód, každá jednej z farieb červená-zelená-modrá (podľa modelu RGB). Výsledné RGB-LED displeje disponujú potom veľmi dobrou čitateľnosťou pri rôznych svetelných podmienkach, rovnako kontrastom a aj vysokou hodnotou maximálneho pozorovacieho uhla.

Tento odsek vychádza z [26] a [6]. Technológia organických elektroluminiscentných diód (OLED) sa skladá zo subtrátu a tenkých organických vrstiev amorfných štruktúr vložených medzi dva vodiče. Pre substrát je typicky použitý plast, sklo alebo tenká vrstva kovu. Technológia OLED využíva princíp elektroluminiscencie, kedy prenášaný elektrický náboj medzi dvoma vodičmi generuje svetlo. Vďaka amorfným štruktúram možno vytvárať aj displeje veľkých rozmerov, prípadne rôznych tvarov. Displeje OLED typicky využívajú adresovanie pomocou aktívnej matice. Pre ovládanie intenzity, akou má pixel svietiť, sa používa kondenzátor. Displeje sa vyznačujú tiež nízkou celkovou spotrebou, pretože svietia len tie pixely, ktoré sú aktívne. Čierne pixely nepotrebujú žiadnu energiu. Pri vačších rozmeroch spotreba však rapídne stúpa.

2.2 Technológie využívajúce podsvietenie

Tieto technológie modulujú svetlo prechádzajúce displejom (*Non-emissive family*). Toto svetlo môže byť buď získané z okolia, alebo je vyžadované podsvietenie. Výsledný príkon displeja P je teda definovaný nasledovne:

$$P_{displej} = P_{panel} + P_{podsvietenie}$$

2.2.1 Technológia LCD

Nasledujúca časť vychádza z [23] a [6]. Technológia LCD (*Liquid crystal display*) využíva vlastnosti tekutých kryštálov, ktoré dokážu modulovať svetlo. Každý pixel sa typicky skladá z vrstvy niekoľkých molekúl tekutých kryštálov vložených medzi 2 priehľadné elektródy a polarizačné filtre. Elektrickou energiou možno kontrolovať, koľko svetla sa prepustí cez daný pixel a tým možno získať rôzne odtiene šedej. Molekuly tekutých kryštálov sa pri nepretejakúcom prúde usporiadajú do približne homomorfnej štruktúry. V prípade, že kryštálmi preteká prúd, usporiadajú sa do štruktúry pripominajúcej pyramídu, ktorá potom vedie svetlo. Vo vrchnej časti sa nachádza tenká vrstva, ktorá zvyšuje pozorovací uhol, tzv. WV film (*wide-view film*). Farebné filtre sú zodpovedné za stratu väčšiny svetla. Monochromatické filtre pohlcujú až 3x menej svetla, preto sú energeticky menej náročné.

Kedže LCD pre svoju funkcionalitu len modulujú svetlo a nevyžarujú, bývajú buď podsvietené (*backlit type*) alebo odrážajú svetlo z okolia (*reflective type*). Nevýhodou displejov s podsvietením je možná znížená čitateľnosť v prípade, že v okolí sa nachádza príliš veľa svetla. Displeje odrážajúce svetlo z okolia sú oproti displejom s podsvietením dobre čitateľné aj za vysokých svetelných podmienok. Tiež majú nižšiu spotrebu, keďže nie je potrebná energia na podsvietenie. Na druhej strane ak na displej dopadá svetla z okolia málo, displej odrážajúci svetlo sa stáva nečitateľným. Ako podsvietenie sa bežne používajú mikro-flourescentné trubky, prípadne svietivé LED diódy. Podsvietenie sa dáva po stranách displeja, prípadne zozadu. Často sa používa panel, ktorý uniformne rozloží svetlo po celej ploche displeja. Pre minimalizáciu vyššie spomínaných nevýhod vznikajú hybridné displeje.

Každy pixel takéhoto displeja sa skladá z dvoch subpixelov – jeden prepúšťajúci svetlo a druhý odrážajúci. Pomer velkostí jednotlivých subpixelov je potom možné optimalizovať vzhľadom na cieľovú aplikáciu.

Dipleje môžu byť implementované ako segmentové, ktoré zobrazujú fixné množstvo číslic alebo iných symbolov. Tieto displeje majú potom pre každý segment nezávislé elektródy. Inak bývajú displeje implementované ako matice elektród, kedy adresovanie prebieha sekvenčne po maticiach. Na jednej sú eletródy prepojené po riadkoch, na druhej po stĺpcoch. Ďalej možno displeje rozdeliť podľa spôsobu adresovania na LCD displeje ovládané aktívnou maticou – AMLCD (*active matrix LCD*) a displeje ovládané pasívnou maticou – PMLCD (*passive matrix LCD*). U displejov ovladaných pasívnom maticou si každý pixel udržuje svoju hodnotu medzi aktualizáciami. Pixely v tomto prípade fungujú podobne ako kondenzátory, kedy sa pri aktualizácií hodnoty nabijú a následne sa vybíjajú. Pri aplikovaní tohto spôsobu adresovania je potom potrebné vziať do úvahy čas, ktorý je potrebný na aktualizáciu každého pixelu a čas, za ktorý pixel vybledne. Podľa [23] môže byť vhodné použitie displejov ovládaných pasívnou maticou u aplikácií, kedy je množstvo používaných pixelov nižšie a kde je potrebné minimalizovať spotrebu, keďže je potrebné menšie množstvo energie na podsvietenie. Pri ovládaní displeja aktívnou maticou je každý pixel aktualizovaný nezávisle na ostatných. Bežne sa pre tento spôsob adresovania používa technológia tranzistorov v tenkej vrstve (*thin-film transistor*, TFT). Tým, že sú pixely adresované nezávisle, možno získať oveľa vyššiu frekvenciu aktualizácie.

2.2.2 Elektronický papier

Elektronický papier, *electronic paper* (*E-paper*), zastupuje technológie, ktoré nevyžarujú svetlo, no ani nevyužívajú podsvietenie. Pre zobrazovanie informácie modulujú odrazené svetlo z okolia. Dnes najpoužívanejšou je technológia *E-ink* (elektronický atrament). Táto technológia moduluje obsah displeja pomocou elektrického pola. V článku [2] sú spomenuté iné prespektívne technológie na báze elektronického papiera.

Nasledujúca časť textu vychádza z patentu [34]. Displeje *E-ink* sa skladajú z mikrokapsúl, ktoré obsahujú kladne nabité biele častice a záporne nabité čierne častice rozpustené v priehľadnej tekutine. Prvé displeje *E-ink* boli implementované v 70. rokoch. Tieto displeje obsahovali farebné guľôčky, ktoré boli z jednej strany čierne a druhej biele, pri čom sa po aplikovaní elektrického pola otáčali. Pri takto voľne rorzptýlených časticiach vznikal problém, že sa posúvali kvôli gravitácií, prípadne sa zoskupovali pri elektródach. Pri ďalších generáciach *E-ink* sa preto farebné častice vkladajú do mikrokapsúl.

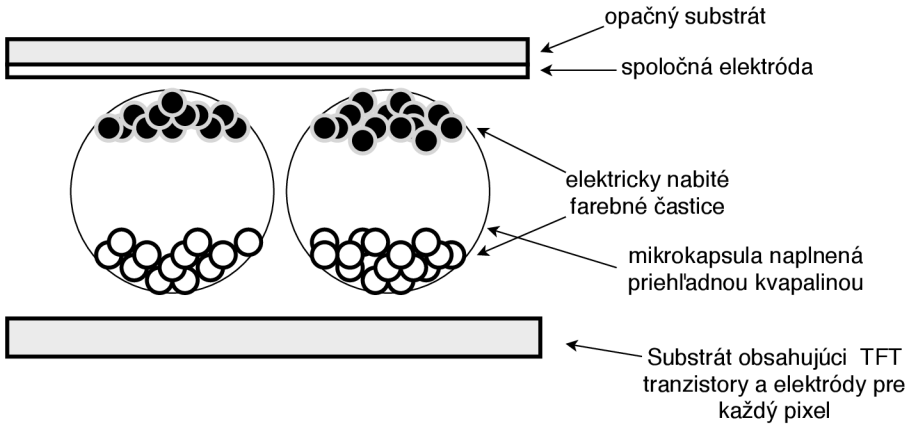
Aktuálna technológia sa skladá z niekoľkých vrstiev - substrátu, zobrazovacieho média a opačného substrátu (*opposite substrate*). Substrát obsahuje maticu s aktívnym adresovaním. Zobrazovacie médium je tvorené farebnými časticami. Opačný subtrát je priehľadný a tvorí hornú vrstvu displeja.

Substrát sa skladá z vlastného substrátu, skenovacích a dátových riadkov (*scan line*, *data line*) a TFT tranzistorov (*thin-film transistor*). Substrát je typicky vyrobený zo skla alebo plastu. Skenovacie a dátové riadky sú použité pre adresovanie jednotlivých pixelov, presnejšie aktiváciu elektródy zodpovedajúcej každému pixelu pomocou TFT tranzistoru. Každý pixel používa vlastný TFT tranzistor a elektródu (*pixel electrode*). Tranzistor a elektróda sú oddelené dielektrickou vrstvou. Vrstva tiež zabraňuje interferencií jednotlivých elektród.

Opačný substrát sa skladá zo samotného substrátu a spoločnej elektródy. Spoločná elektróda (*common electrode*) je priehľadná, vodivá vrstva a táto elektróda je spoločná pre všetky pixely. Usporiadanie vyššie uvedených vrstiev je zobrazené na obrázku 2.1.

Zobrazovacie médium (*display medium*) je minimálne bistabilné. Existujú rôzne typy zobrazovacích médií:

- Médium sa skladá zo svetlých a tmavých častíc, ktoré sú rozptýlené v priehľadnej (transparentnej) kvapaline. Pri prechádzajúcom prúde medzi spoločnou elektródou a elektródou zodpovedajúcou danému pixelu častice jednej farby vystúpia do popredia a častice druhej farby zase ustúpia do pozadia v závislosti na zmene smeru elektrického pola. Takýto typ média je použitý na obrázku 2.1.
- Médium je tvorené časticami svetlej farby, ktoré sú rozptýlené v tmavej kvapaline. Svetlé častice potom menia svoju pozíciu v závilosti na smere elektrického pola. Takto sú viditeľné buď svetlé častice, alebo tmavá kvapalina.



Obr. 2.1: Schéma prierezu displeja *E-ink*. Nákres vychádza z [34].

Médium – častice s kvapalinou bývajú ďalej zabalené do mikrokapsúl. Jednému pixelu typicky zodpovedá mnoho takýchto mikrokapsúl. Publikácia [19] hovorí o použití média tvoreného jedným typom častíc a ofarbenej kvapaliny. Pri tejto implementácií bolo možné pomocou ladenia hodnoty napätia odlíšiť až 4 hodnoty *grayscale*. Problémom tejto implementácie bol dlhý čas potrebný aktualizáciu a pomerne vysoké napätie (až 18 V). Dnes spoločnosť *Waveshare* [31] ponúka riešenie, ktoré podporuje 3 farby.

Jednou zo slabín technológie *E-ink* je čas potrebný pre aktualizáciu. Podľa [19] má na rýchlosť presunu častíc vplyv hlavne veľkosť častíc, veľkosť náboja v častici, elektrické pole vizkozita kvapality. Spotreba veľmi závisí aj na zobrazovanom obsahu. Podľa [21] sa až 75% energie minie na ovládanie kondenzátorov pri adresovaní hodnôt jednotlivých pixelov. Tento kondenzátor sa využije len v prípade, že pixel mení svoju farbu. V prípade aktualizácie len pixelov, ktoré zmenili svoju farbu, hovoríme o čiastočnej aktualizácií (*partial refresh*). Opačným prípadom je plná aktualizácia (*full refresh*).

Výhodou technológie *E-ink* je okrem nízkej spotreby aj dobrá čitateľnosť za akýchkoľvek sveteľných podmienok. Vďaka tomu, že displej neobsahuje vrstvu polarizačných filtrov, tiež poskytuje maximálny pozorovací uhol takmer 180°.

2.3 Vybrané nízkoenergetické displeje

V tejto časti budú popísané technické detaily o vybraných displejoch vhodných pre vizualizačný panel. Technológie, na ktorých tieto displeje pracujú, sú popísané v kapitole 2. Všetky vybrané displeje sú monochromatické.

2.3.1 Elektronický papier *Waveshare*

Tento odsek vychádza z [32]. Displej využíva techológiu *E-ink* (elektronický papier). Je veľmi vhodný pre aplikácie, kedy je potreba zobrazovať statický obsah s ohľadom na nízku spotrebu.

technické parametre	
uhlopriečka	4,2 palca
operačné napätie	3,3 V
operačná teplota	0 – 50 °C
veľkosť pixelu	0,212 mm × 0,212 mm
hodnota dot pitch	0,212 mm × 0,212 mm
rozlíšenie	400 × 300 px
farby	čierna, biela
doba úplnej aktualizácie	4 s
spotreba pri aktualizácií	40 mW
spotreba pri udržiavaní obsahu	0,017 mW
komunikačné rozhranie	SPI

Tabuľka 2.1: Technické parametre displeja *Waveshare 4.2 inch e-paper*.

Niektoré technické údaje o tomto displeji sú uvedené v tabuľke 2.1. Z údajov o príkone v rôznych režimoch fungovania možno spočítať približnú priemernú hodnotu príkonu potrebného pri reálnej aplikácií. Nech displej aktualizuje svoj obsah každých 10 minút pomocou kompletnej aktualizácie, ktorá trvá približne 4 sekundy. Tým pádom bude mať displej 10 minút aktuálny príkon 0,017 mW a po 4 sekundy príkon približne 40 mW. Potom displej pri takejto aplikácií spotrebuje približne 0,28 mWh.

Podľa [32] bežný scenár aktualizácie obsahu displeja vždy pozostáva z nasledujúcich bodov: zapnutia zobrazovacieho panelu, nastavenia panelu, nastavenia rozlíšenia, vloženia obrázku, aktualizácie obsahu a nastavenia ohraničenia (*border*). Pri implementácií riešenia je potrebné brať ohľad na nasledovné obmedzenia displeja:

1. Raz za niekoľko čiastočných aktualizácií je potrebné urobiť aktualizáciu celého displeja. Inak môže ostať displej trvalo poškodený.

2. Displej nemôže ostávať zapnutý po príliš dlhý čas, musí sa prepnúť do režimu spánku, prípadne vypnúť. V opačnom prípade môže ostať trvalo poškodený.

3. Výrobca odporúča interval aktualizácie 24 hodín až 10 dní. Inak môžu vzniknúť tiene, ktoré budú trvalé.

2.3.2 Grafický LCD displej *Crystalfontz*

Nasledujúci odsek vychádza z [5]. Tento displej využíva technológiu tekutých kryštálov popísanú v podkapitole 2.2.1. Ide o grafický displej, pre komunikáciu je použité paralelné rozhranie pre LCD moduly typu 8080. Ďalšia špecifikácia displeja je uvedená v tabuľke 2.2.

technické parametre	
uhlopriečka	4,84 palca
operačná teplota	-20 – 70 °C
veľkosť pixelu	0,56 mm × 0,56 mm
hodnota dot pitch	0,60 mm × 0,60 mm
rozlíšenie	320 × 240 px
farby	čierna, biele pozadie
doba úplnej aktualizácie	200 – 300 ms
napájacie napätie pre logiku	5,5 V
napájacie napätie pre LCD	19,8 V
napájací prúd	0,50 mm
komunikačné rozhranie	paralelné rozhranie 8080 LCM

Tabuľka 2.2: Technické parametre LCD displeja od spoločnosti *Crystalfontz*.

V tabuľke 2.2 možno tiež nájsť informácie o napájacom napätí a napájacom prúde, z ktorých možno určiť príkon pre jednotlivé komponenty: 275 mW pre logiku, 990 mW pre samotný displej. Celkový potrebný príkon bude teda približne 1,266 W.

2.3.3 Grafický OLED displej *EastRising*

Nasledujúci odsek vychádza z [8]. Tento displej používa technológiu OLED vysvetlenú v podkapitole 2.1.2. Podľa údajov v tabuľke 2.3 možno určiť príkon zobrazovacieho panelu – približne 1,25 W.

technické parametre	
uhlopriečka	5,2 palca
operačná teplota	-40 – 80 °C
veľkosť pixelu	0,50 mm × 0,50 mm
hodnota dot pitch	0,53 mm × 0,53 mm
rozlíšenie	256 × 64 px
farby	zelená, čierne pozadie
napájacie napätie	5 V
napájací prúd	250 mA
komunikačné rozhrania	paralelné rozhrania LCM 6060 a 8080, rozhranie SPI

Tabuľka 2.3: Technické parametre LCD displeja od spoločnosti *EastRising*.

2.3.4 Zhodnotenie vybraných displejov

Technológie OLED a LCD ponúkajú pomerne rýchlu odozvu – rádovo v stovkách milisekúnd. Preto sú displeje založené na týchto technológiach vhodné pre aplikácie, kde je potrebné zobrazovať dynamický obsah. U displeja technológie *E-ink* môže trvať aktualizácia až 4 sekundy pri plnej aktualizácií obsahu, pri parciárnej aktualizácií je táto hodnota nižšia.

Preto je vhodnejší pre aplikácie, kde nie je aktuálnosť informácií až tak kritická, respektíve majú informácie viac statický charakter. *E-ink* zobrazovací panel má mnohonásobne nižší príkon, čím sa stáva vhodným pre nízkoenergetické (*ultra low power*) aplikácie. Vybrané OLED a LCD displeje používajú pre zníženie spotreby režimy spánku pamät pre uloženie aktuálneho obsahu.

Kapitola 3

Internet vecí a vstavané systémy

V tejto kapitole bude definovaný pojem internetu vecí, jeho vznik, protokoly ktoré sú v tejto oblasti využívané, ďalšie smerovanie oblasti a problémy, ktorým táto oblasť čelí. V druhej podkapitole je preberaná problematika vstavaných systémov a ich spotreby. V poslednej časti sú predstavené 2 mikrokontroléry, ktoré sú vhodné pre riadenie vizualizačnej jednotky.

3.1 Internet vecí

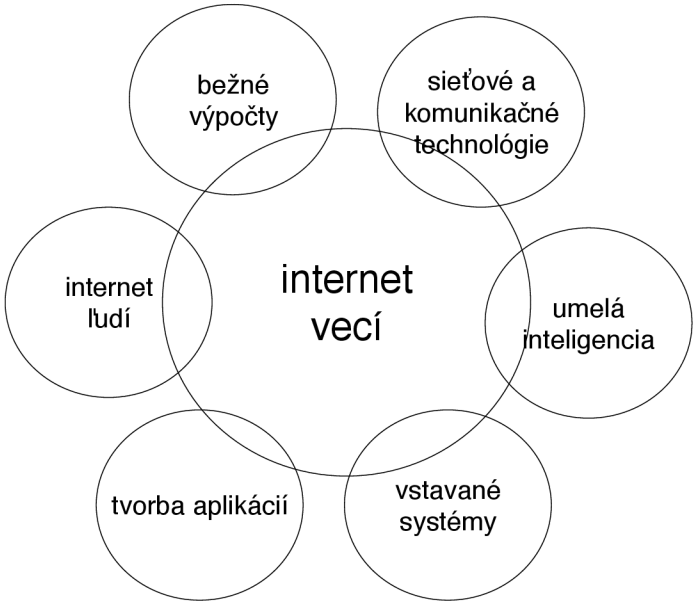
Podľa [33] je internet vecí alebo ang. IoT (*Internet of things*) konceptom, ktorý sa snaží prepojiť objekty každodennej potreby cez internet pomocou vstavaných systémov. Takéto riešenie potom tvorí distribuovaný systém, v ktorom budú tieto objekty komunikovať s človekom a ostatnými objektami.

Nasledujúca časť tejto podkapitoly vychádza z [29]. Táto publikácia rozširuje vyššie spomenutú definíciu internetu vecí. Komunikujúce objekty môžu byť fyzické alebo virtuálne. Tieto objekty sa aktívne účastnia procesov v oblastiach obchodu, informačných a sociálnych procesov. Objekty by mali byť schopné komunikovať medzi sebou a autonómne reagovať na podnety reálneho sveta bez priameho ľudského zásahu. Pri implementácií by mal byť braný dôraz na bezpečnosť a súkromie.

Pozornosť na oblasť internetu vecí priniesol rozvoj a klesajúca cena oblastí rádio frekvenčnej identifikácie (RFID), senzorových systémov, vysokofrekvečných bezdotykových spojení s krátkym dosahom NFC (*Near-field communication*), bezdrôtovej komunikácie a v neposlednom rade webových technológií. Vďaka týmto technológiam je možné určovať aktuálny stav fyzických objektov. Spolu so sadou dát a ich spracovaním umožňuje okamžitú odpoveď na zmeny reálneho sveta. Tieto systémy sa stávajú plne interaktívnymi, čím sa stávajú zaujimávymi a dostupnými pre investorov a koncových zákazníkov. Publikácia [18] označuje IoT ako generáciu internetu a popisuje úzku spojitosť rozvoja internetu s rozvojom internetovej služby *Web*.

Obrázok 3.1 zobrazuje oblasti, ktoré sa prekrývajú s internetom vecí podľa predošlých definícií, avšak nie sú ich výlučnou súčasťou. Napríklad sieťové a komunikačné technológie sú použité pri prepájaní systémov v rámci internetu vecí. Vstavané systémy nie sú výlučne súčasťou IoT, no ich prepojenie s internetom poskytuje viac možností využitia. Aplikácie sú v IoT potrebné pre komunikáciu s užívateľom. Umelá inteligencia otvára ďalšie možnosti spracovania dát. Internet ľudí je podľa [18] predchodcom internetu vecí.

IoT sa dnes používa v mnoho oblastiach. Napríklad v leteckom a automobilovou priemysle, telekomunikáciach, oblasti inteligentných budov, spracovateľskom priemysle, škols-



Obr. 3.1: Oblasti, ktoré sa prekrývajú s problematikou internetu vecí, diagram vychádza z [29].

tve, bezpečnosti a monitorovaní. V budúcnosti by sa malo posilniť postavenie oblasti internetu vecí pri strategickom rozhodovaní a plánovaní v podnikoch. IoT umožňuje zbierať a vyhodnocovať dáta v reálnom čase, čo je pri plánovaní podstatné. Ďalej by mali ďalej zlepšovať algoritmy spracovania dát, pričom sa počíta s použitím strojového učenia. Ďalšou výzvou pojednávanou v [24] je bezpečnosť IoT zariadení. V tejto publikácií sú za najväčšie problémy považované slabá podpora existujúcich zariadení, slabé zabezpečenie prístupu, neschopnosť predpovedať útoky a predchádzať im, nízke zabezpečenie privátnosti dát, zabezpečenie správneho spracovania dát, zabezpečenie inteligentných domácností a autonómnych vozidiel.

3.2 Koncové zariadenia IoT

Vstavané systémy (*Embedded Systems*) sú dôležitou súčasťou Internetu vecí, pretože sú často koncovými zariadeniami, teda prvkom, ktorý prepája objekty reálneho sveta a umožňuje ich vzájomnú interakciu. S možnosťou bezdrôtovej komunikácie vstavaných systémov spolu s možnosťami ich napájania stúpa škálovateľnosť týchto systémov a ich uplatnenie v praxi. V tejto podkapitole sú rozobrané rôzne možnosti napájánia vstavaných systémov. Niektoré spôsoby napájania poskytujú len obmedzené množstvo energie a tak sú v druhej časti podkapitoly popísané faktory, ktoré majú najväčší vplyv na spotrebu vstavaných systémov.

3.2.1 Možnosti napájania vstavaných systémov

Tento odsek vychádza z [27]. Spôsoby napájania vstavaných systémov možno rozdeliť do troch kategórií:

- napájanie vstavaných systémov z elektrickej siete,

- napájanie pomocou jednorázovej alebo znovunabíjateľnej batérie,
- napájanie pomocou systémov získavajúcich energiu z okolia.

Tento odsek vychádza z [27]. U vstavaných systémov napájaných z elektrickej siete je spotreba energie najmenej kritickou vďaka jej relatívne neobmedzenému množstvu okamžite k dispozícií. Nezanedbateľná časť energie je pri takomto spôsobe spotrebovaná pre prevod z obojsmerného prúdu, ktorý je privádzaný zo siete, na jednosmerný prúd vyžadovaný vstavaným systémom. Tiež je potrebné previesť vyššie napätie z elektrickej siete na nižšie vyžadované vstavaným systémom. Pri tomto prevode platí, že so znižujúcim sa pomerom výstupného napätia k vstupnému napätiu klesá aj efektivita využitia energie.

Tento odsek vychádza z [27]. Vstavané systémy napájané pomocou jednorázovej (*primary battery*) alebo znovunabíjateľnej batérie (*secondary battery*) majú narozdiel od systémov napájaných z elektrickej siete relatívne obmedzené množstvo energie k dispozícií a preto sa ich spotreba stáva kritickou. Jednorázové batérie majú vyššiu kapacitu a menšie straty elektrického náboja v čase. Jednorázové batérie sú vhodné pre aplikácie, kedy je potrebná dlhá životnosť batérie a mikrokontrolér sa prepína periodicky medzi režimom spánku a aktívnym režimom. Na druhej strane sa znovunabíjateľné batérie dokážu poskytovať stabilnejšie napätie počas životného cyklu a tiež pri nižších teplotách. Znovunabíjateľné batérie sú vhodné, ako záložný zdroj energie pri výpadkoch elektrickej siete, prípadne u aplikácií, u ktorých ich použitie vedie použitie znovunabíjateľných batérií k zníženiu nákladov. Pre aplikáciu vizualizačného panelu sú vhodné jednorázové alkalické alebo lítiové batérie. Podľa experimentov popísaných v [27] sa alkalické batérie vyznačujú dobrým výkonom aj pri nízkych teplotách a nárazovo vysokých odberoch. Litiové batérie podporujú stabilné napätie vo vyššom rozsahu teplôt a pri rovnakej veľkosti dokážu poskytnúť viac energie. Oba typy batérií majú nízke straty energie v čase, ich nevýhodou je znižujúce poskytované napätie počas celého životného cyklu. Pre všetky typy batérií platí, že so zvyšujúcou teplotou sa zvyšujú straty energie a pri striedavej záťaži (prepínanie medzi aktívnym režimom a režimom spánku) možno získať viac celkovej energie z batérie. Pri použití prevodníka napätia a prepínania medzi aktívnym režimom a režimom spánku systému je potrebné dať pozor na účinnosť takéhoto prevodníku, pretože môže významne ovplyvniť spotrebu systému v režime spánku.

Tento odsek vychádza z [27] a [25]. Vstavané systémy môžu dnes získavať energiu (*energy harvesting*) zo svetla (solárne panely), rozdielu teplôt (napr. Peltierov článok), pohybu a vibrácií (dynamo), vody alebo prúdenia vzduchu. Zariadenia zbierajú energiu buď neustále alebo v krátkych časových intervaloch. Ak ju zbierajú v krátkych časových intervaloch, vstavané systémy musia byť navrhnuté tak, aby vykonali obsluhu v čo najkratšom čase. Ak zbierajú energiu neustále, túto energiu ukladajú pre neskoršie použitie. Podľa [27] ju často ukladajú v kondenzátoroch. Kondenzátory poskytujú len obmedzené množstvo energie a dochádza u nich k rýchlemu vybíjaniu, takže energiu nemožno dlhodobo skladovať bez prísunu vstupnej energie. Systémy využívajúce neustálu energiu sú potom navrhnuté tak, aby zotrvali v aktívnom režime čo najkratšie a boli prepínané do aktívneho režimu čo najmenej často, ako to konkrétna aplikácia dovolí. U systémov napájaných zariadeniami zbierajúcimi energiu je potrebné tiež zabezpečiť vhodný mechanizmus pre zotavenie po vypnutí kvôli nedostatku energie.

3.2.2 Spotreba mikrokotroléru

Väčšina vstavaných systémov sa dnes skladá zo zdroja napätia, výpočetného jadra a použítých periférií v danej aplikácií. Výpočetné jadro využíva moduly pre generovanie taktovacej frekvencie, komunikuje s pamäťami typu RAM, ROM a aplikačne závislými perifériami pomocou vstupno-výstupných rozhraní. V nasledujúcej časti textu sú spomenuté faktory, ktoré ovplyvňujú spotrebu energie najviac. Informácie o týchto faktoroch a prístupoch vhodných pre zníženie celkovej spotreby vychádzajú z [27] a [25].

Výpočetné jadro

Spotrebu výpočetného jadra možno rozdeliť na statickú a dynamickú. Statická zložka predstavuje straty prúdu a táto zložka nie je závislá na taktovacej frekvencií. Dynamická zložka je závislá na taktovacej frekvencií procesora a tvorí majoritnú časť spotreby. Dynamická aj statická zložka spotreby sú závislé na vstupnom napätí. Ak by sme spotrebu hodnotili, ako spotrebu energie za jednu operáciu, pre minimalizáciu spotrebu by bolo potrebné pre danú hodnotu napätia využiť maximálnu taktovaciu frekvenciu. Po zohladnení vstupného napätia je najvýhodnejšie maximalizovať taktovaciu frekvenciu pri použití minimálneho možného napätia. Dnes je vo vstavaných systémoch často implementované dynamické škálovanie vstupného napätia a taktovacej frekvencie na základe aktuálnej záťaže. Pri dynamickom škálovaní vstupného napätia je potrebné počítať s efektivitou prevodníku napätia. Pri napájaní systému z elektrickej siete treba zohľadniť aj efektivitu prevodníka na jednosmerný prúd. U reálnych aplíkácií je potrebné pri výbere taktovacej frekvencie prihliadnuť na interakciu výpočetného jadra s perifériami, ktoré majú vplyv na utilizáciu jadra.

Pamäť typu RAM

Pamäť s priamym prístupom (*Random Access Memory*) sa používa pre dočasné uloženie dát v rámci vstavaného systému. Pamäť typu RAM je najefektívnejšia z pohľadu spotreby. Tiež poskytuje konštatný čas prístupu pre ktorýkoľvek bajt. Jej nevýhodou je, že je volatilná, teda neudržuje v sebe dáta po odpojení od elektrickej energie.

Pamäť typu ROM

Pernamentná pamäť (*Read-Only Memory*) má výhodu, že je nevolatilná, teda sú v nej uložené dáta aj po odpojení elektrickej energie. Tieto dáta však nemožno prepísať, prípadne pre je pre prepisovanie potrebný značný čas a energia (elektricky mazateľná pamäť ROM – EEPROM). U pamäte typu EEPROM sa dáta zapisujú po stránkach, pre zápis je potrebné vyššie napätie, ako pri iných operáciach, čím sa zápis stáva energetický náročným. Táto pamäť podporuje len obmedzený počet zápisov počas svojej životnosti. Pamäť typu ROM sa často používa pre uloženie programu.

Vstupno-výstupné rozhrania

Periférie pripojené na vstupno-výstupné rozhrania tiež ovplyvňujú spotrebu. Pre minimalizáciu spotreby [27] odporúča pripojiť periférie najnižším možným napätím a komunikovať s nimi najnižšou možnou rýchlosťou, akú dané komunikačné rozhranie umožňuje. Periférie by mali byť vypnuté, ak sa aktuálne nevyužívajú. Za veľkú časť spotreby je zodpovedný bezdrôtový prenos. Pre jeho optimalizáciu [25] odporúča nechávať *transmitter* spustený len, keď je to nevyhnutné. Tiež je odporúčané minimalizovať množstvo prenášaných dát.

3.2.3 Meranie spotreby vstavaných zariadení

Tento odsek vychádza z [4]. Meranie reálnej spotreby vstavaných systémov je pomerne problematická záležitosť. Pre meranie je vhodné oddeliť každú komponentu, aby bolo možné merať jej spotrebu. Pri meraní spotreby sa meria aktuálny prúd, ktorý sa následne vynásobí hodnotou použitého napätia a tým získame hodnotu aktuálneho príkonu. Následne je potrebné určiť priemerný príkon za určitý čas a po jeho vynásobení časom dostávame spotrebu zariadenia v mWh. Pre meranie je tiež problematický rozdiel v prúde potrebnom pri rôznych režimoch spotreby jednotlivých komponént. Pre tento veľký rozsah meraných hodnôt prúdu sa stáva použitie osciloskopu problematickým. Multimeter je o niečo vhodnejší, ten poskytuje ustálenejšie hodnoty, ktoré navzorkuje a zobrazená hodnota je potom ich pseudopriemerom. Takýto prístup na druhej strane nezachytí nárazové zmeny prúdu. Určovanie spotreby pomocou zisťovania aktuálneho prúdu pomocou osciloskopu sa tiež stáva problematickým kvôli jeho vysokej premenlivosti. Preto je vo [4] navrhované použiť špecializované zariadenia pre meranie spotreby vstavaných systémov.

3.3 Vybrané mikrokontroléry pre koncové zariadenia

V tejto podkapitole sú predstavené platformy *ESP32* a *ESP8266*, ktoré budú použité pri implementácií vizualizačnej jednotky. O týchto platformách sú popísané dôležité informácie získané z technickej špecifikácie. Dôraz je kladený na režimy spotreby a možnosti bezdrôtovej komunikácie s použitím Internetového protokolu (IP). Vďaka podpore rôznych režimov spotreby a bezdrôtovej komunikácie sa tieto mikrokontroléry stávajú vhodnými najmä pre aplikácie napájané batériou, prípadne zariadením získavajúcim energiu z okolitého prostredia. Použiť ich možno napríklad pre periodické získavanie hodnôt zo senzorov. Získané dáta sa môžu okrem okamžitého odoslania na centrálnu jednotku aj sami vyhodnocovať a spracovávať (*Sensor hub*). Vďaka veľkosti čipov sú tiež vhodné pre nositeľné aplikácie (*wearable applications*). Modul WiFi podporuje okrem módu stanice BSS aj mód prístupového bodu, prípadne promiskuitný mód, takže môže byť použitý pre analýzu sieťovej prevádzky (*sniffer*).

3.3.1 Platforma *ESP32*

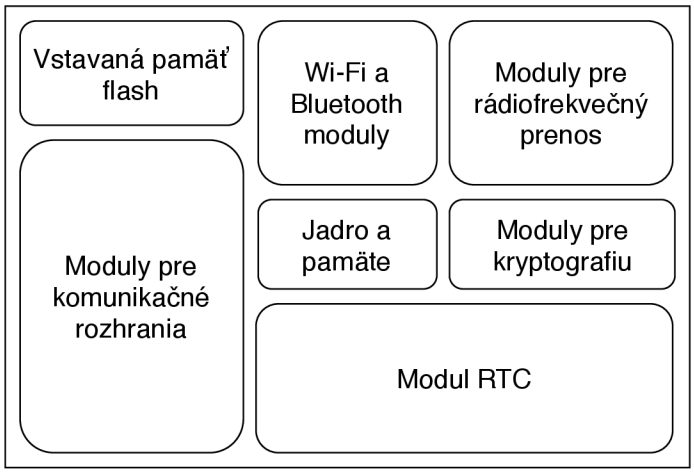
Pre riadenie vizualizačnej jednotky je vhodný generický modul *ESP-WROOM-32* od firmy *Expressif*. V implementácií je tento modul osadený do vývojovej dosky *ESP32 devkitc V4*.

Nasledujúca časť textu vychádza primárne z *datasheet* pre vývojovú platformu *ESP32* [13]. *ESP32* je kombinovaný čip (*combo chip*), ktorý obsahuje vstavané moduly pre WiFi a *Bluetooth*. Pri tvorbe čipu bola použitá nízkoenergetická (*ultra-low-power*) 40 nm technológia. Tento čip bol spolu s obsahujúcimi modulmi navrhnutý pre mobilné, nositeľné (*wearable*) a IoT aplikácie, u ktorých je potrebné optimalizovať spotrebu. Platforma *ESP32* je veľmi rozšírená a je podporovaná v mnohých vývojových prostrediach. Podľa [22] je platforma podporovaná vo vývojovom prostredí *Arduino*, výrobca platformy odporúča v [15] vlastné vývojové prostredie *ESP-IDF*.

Obrázok 3.2 zobrazuje funkčné bloky v rámci *ESP32*:

- *ESP-WROOM-32* podľa [15] disponuje dvomi jadrami s 32 bitovou architektúrou. Nachádzajú sa tu dve pamäte - 448 KB pamäť ROM, ktorá slúži pre bootovanie a hlavné funkcie jadra. Ďalej 520 KB pamäť, kde sa ukladajú dáta a inštrukcie.

- *ESP32* podporuje základné štandardné komunikačné rozhrania, ktorými sú SPI, I2S, I2C a UART. Poskytuje 34 pinov pre všeobecné použitie (GPIO), Analógovo-digitálne prevodníky (ADC, DAC), dotykové senzory, ethernetové MAC rozhranie a moduly pre pulzne širkovú moduláciu s podporou motorov aj LED.
- WiFi modul plne podporuje protokol *802.11bg* a *802.11n* (pre 2,4 GHz pásmo).
- Na platforme sa nachádzajú moduly pre HW podporu kryptografie - podporujúce štandard AES, generátor náhodných čísel (RNG) a algoritmy RSA, SHA-2 a ECC (eliptická krivka).
- RTC modul *ESP-WROOM-32* podľa [15] obsahuje dve 8 KB pamäte, ktoré sú dostupné počas režimu hlbokého spánku. Rýchla pamäť (*RTC Fast*) je dostupná pre hlavnú *CPU jednotku*, pomalá pamäť (*RTC Slow*) je dostupná pre ko-processor. Modul tiež obsahuje ULP ko-processor, ktorý možno použiť pre obsluhu periférií aj v režimu hlbokého spánku.
- *ESP-WROOM-32* podľa [15] tiež obsahuje 4 MB nevolatilnú flash pamäť, ktorá je pripojená k pinom cez rozhranie SPI.



Obr. 3.2: Bloková schéma obsahujúca funkčné bloky *ESP32*. Schéma vychádza z [13].

Režimy spotreby platformy *ESP32*

Nasledujúci odsek vychádza z [14] a [13]. Platforma *ESP32* ponúka mnoho možností pre nastavovanie spotreby tak, aby bola vhodná pre nízkoenergetické aplikácie. Za týmto účelom má platforma niekoľko preddefinovaných režimov spotreby (*power modes*). Podľa [14] možno okrem použitia preddefinovaných režimov spotreby aj manuálne nastaviť, ktoré moduly majú byť napájané. V rámci platformy *ESP32* sú preddefinované nasledujúce režimy spotreby:

1. Pri **aktívnom režime** (*active mode*) je spustený modul pre bezdrôtovú komunikáciu a tak možno prijímať/odosielať dáta. Spotreba v tomto režime je závislá od využívania bezdrôtovej komunikácie.

2. V rámci **režimu spánku modemu** (*modem-sleep mode*) beží CPU jednotka, no moduly pre rádiovú komunikáciu, WiFi a *Bluetooth* sú vypnuté. Taktovacia frekvencia je v tomto režime variabilná podľa aktuálnej záťaže procesora a aktívnych periférií, pri čom sa mení automaticky. Spotreba platformy je pri frekvencií 20 MHz približne 3 mA a pri 80 MHz približne 30 mA. Na spotrebu má ďalej vplyv počet využívaných jadier.

3. V **režime ľahkého spánku** (*light sleep mode*) je pozastavená činnosť CPU jednotky, jej kontext je zachovaný. V tomto režime je ďalej spustený RTC modul, je možné použiť RTC rýchlu a pomalú pamäť a ULP ko-procesor (*Ultra Low Power Co-processor*) je možné periodický spúšťať. S potreba je v tomto režime približne 800 µA. Prebudenie trvá do 1 ms.

4. Pri použití **režimu hlbokého spánku** (*deep sleep mode*) bude stratený kontext jednotky CPU. V tomto režime je však možné použiť RTC pamäť pre uloženie konfigurácie WiFi, prípadne *Bluetooth* modulu. V tomto režime je možné použiť ULP ko-processor, RTC časovač a niektoré periférie. Spotreba je približne 10 µA a prebudenie trvá do 1 ms. Pri použití monitorovania periférií v režime spánku sa spotreba zvýši približne na 100 µA, pri použití ULP ko-processoru na 150 µA.

5. V režime **hybernácie** (*hybernation mode*) ostáva spustený len RTC časovač a niektoré periférie pre všeobecné použitie (GPIO). Spotreba v tomto režime je približne 5 µA.

Režimy hlbokého spánku a hybernácie sú vhodné pre aplikácie, kedy sa mikrokontrolér periodicky prepína prepína medzi režimom spánku a aktívnym režimom, v ktorom zozbiera dáta a pošle ich pomocou bezdrôtovej komunikácie.

Bezdrôtová komunikácia pomocou platformy *ESP32*

Platforma *ESP32* obsahuje moduly pre *Bluetooth* aj WiFi komunikáciu. Jednotlivé typy bezdrôtovej komunikácie a ich podpora na platforme je popísaná nižšie.

Bezdrôtová komunikácia pomocou *Bluetooth* Informácie o podpore *Bluetooth* platformou *ESP32* sú získané z [13]. Modul pre *Bluetooth* integruje fyzickú (*Bluetooth baseband*) a linkovú (*Bluetooth link controller*) vrstvu rozhrania *Bluetooth* a ďalšie nízkoúrovňové rutiny. Modul ďalej podporuje rôzne protokoly pre detekciu a opravu chýb v dátach, zvukové kodeky, rôzne intenzity signálov a protokoly pre modulovanie dát. Podporuje tiež kryptografický štandard AES. Modul podporuje Bluetooth verziu 4.2 BR/BRE a kolekciu služieb BLE (nízkoenergetický Bluetooth).

Základné informácie o technológií Bluetooth sú získané z [20]. Technológia Bluetooth slúži pre komunikáciu s krátkym dosahom. Komunikácia Bluetooth prebieha štandardne na princípe master-slave. Pomocou Bluetooth môžu spolu komunikovať 2 zariadenia, prípadne môže komunikovať až 8 zariadení (sieť *Piconet*). V prípade siete typu Piconet ide o komunikáciu maximálne 7 periférií s 1 centrálnym zariadením. Všetky zariadenia v sieti Piconet komunikujú na jednej frekvencií, centrálne zariadenie odlišuje komunikuje s rôznymi perifériami na základe časového multiplexu. Skratky BR a EDR zastupujú základnú (*Basic Rate*) a rozšírenú rýchlosť dát (*Enhanced Data Rate*).

Informácie o protokoloch BLE sú získané z [28]. Nízkoenergetický Bluetooth BLE je kolekcia služieb, ktoré poskytujú podporu Bluetooth komunikáciu s dôrazom na minimálnu spotrebu. Zariadenia sa delia na centrálne, ktoré zbierajú a spracúvajú dáta, a periférie,

ktoré dáta vysielajú. Vytvorenie spojenia prebieha pomocou protokolu GAP (generický profil prístupu). Proces prebieha tak, že periférie najskôr ponúkajú dáta (*advertisment*). Správy typu *advertisment* vysielajú v pravidelných intervaloch. Centrálne zariadenie môže potom požiadať o ďalšie dáta (*scanning data*). Protokol GATT potom ďalej popisuje samotnú komunikáciu po nadviazaní spojenia. Komunikácia prebieha na princípe master-slave. Centrálne zariadenie, GATT client, začína transakciu požiadavkou na dáta a dostáva odpoveď od periférie, GATT serveru. Služby implementované nad BLE sú identifikované pomocou jedinečnej hodnoty UUID. V rámci platformy *ESP32* je ďalej implementovaná podpora viacerých spojení, adaptívna zmena frekvencie, hodnotenie kanálu a šifrovanie na linkovej vrstve.

Bezdrôtová WiFi komunikácia Informácie o protokole 802.11 sú získané z [1] a podpore tohto protokolu z [13]. V rámci WiFi modulu je implementovaná fyzická (*WiFi baseband*) aj linková vrstva (*WiFi MAC*). Fyzická vrstva podporuje kompletne protokol *802.11 n/b/g*, prenášanie dát vo formáte MCS32 a prenosovú rýchlosť až do 150 mb/s. Zariadenia pomocou WiFi komunikujú medzi sebou na určitej frekvencií. Správy medzi sebou posielajú s určitou periódou a je dôležité zabrániť kolízií posielaných rámcov medzi jednotlivými objektami. Jednotlivé komunikujúce objekty sa nazývajú bezdrôtové stanice (STA). Každá stanica STA má priradenú adresu. Základnú komunikáciu definuje mód základného ustanovenia služby (BSS, *Basic Station Mode*). Tento mód umožňuje vytvorenie lokálnej siete (LAN). Mód BSS definuje komunikáciu medzi dvoma rovnocennými stanicami, prípadne komunikáciu medzi stanicou a prístupovým bodom (AP) do inej siete. Dve prepojené stanice tvoria distribučný systém a tieto systémy sú ďalej prepojené pomocou prístupových bodov. Členstvo v distribučnom systéme je dynamické. Modul WiFi MAC v rámci platformy *ESP32* podporuje mód základného ustanovenia služby (*BSS mode*), mód prístupového bodu (*SoftAP mode*) a promiskuitný mód. Modul tiež podporuje agregáciu protokolových dátových jednotiek (A-MPDU), použitie mechanizmov k zabráneniu kolízií medzi posielanými rámcami, prenášanie médií s rôznou kvalitou služieb (*WiFi multimedia*), rôzne protokoly pre zabezpečenie komunikácie a 4 virtuálne WiFi rozhrania.

Informácie o podpore WiFi platformou *ESP32* sú získané z [9]. Pre používanie WiFi modulu je implementované aplikačné programové rozhranie (API), prípadne tento modul využívajú funkcie rozhrania BSD schránok popísaných v 3.3.1. Každá funkcia aplikačného rozhrania pre ovládanie WiFi detekuje chyby návratovými hodnotami funkcií. Rozoznávajú sa 2 typy chýb:

- zotaviteľné (*recoverable*) možno vyriešiť pomocou opakovaného volania funkcie s časovým odstupom,
- nezotaviteľné (*non-recoverable*) jediným možným spôsobom reakcie je vypísanie/oznámenie chybového kódu, [9] navrhuje vo vývojovej fáze ošetriť tieto návratové hodnoty pomocou funkcie `assert()`.

Pre ovládanie WiFi modulu možno použiť API, prípadne využiť systém udalostí (*event*). Udalosti možno potom ovládať pomocou funkcie `esp_event_register()`. Údaje o API pre WiFi modul sú vybrané z [10]. Toto rozhranie poskytuje funkcie pre vytvorenie a používanie klientskej stanice WiFi, vytvorenie prístupového bodu (AP) a použitie zabezpečení WPA, WPA2, WEP. Ďalej podporuje skenovanie prístupových bodov a odchytávanie komunikácie v promiskuitnom móde.

Aplikačné rozhranie pre komunikáciu pomocou internetového protokolu

Nasledujúci odsek vychádza z [11]. Platforma *ESP32* používa *IwIP*[1] implementáciu balíka internetových protokolov *TCP/IP*, tzv. *TCP/IP stack*. Implementácia *IwIP* implementuje aplikačné programové rozhranie *BSD sockets*. Táto implementácia je menej náročná na zdroje a tak sa stáva vhodnou pre vstavané systémy. Napriek tomu obsahuje základnú podporu protokolov potrebných pre tvorbu bežných aplikácií:

- *Internetový protokol* (*Internet protocol*) - IPv4 aj IPv6,
- protokoly ICMP pre IPv4 a IPv6, protokol IGMP (*Group management protocol*) pre IPv4 a tiež MLD, ND (*Neighbour discovery*) pre IPv6,
- TLS (*transport layer security*), ktorý poskytuje zabezpečuje dáta prenášané pomocou transportného protokolu TCP,
- protokoly PPPoS a PPPoE pre sériovú komunikáciu nad IP protokolom.

Pri tvorbe projektu na platforme *ESP32* je potrebné použiť upravenú implementáciu *IwIP*, ktorá je dostupná vo vývojovom prostredí *ESP-IDF*. Táto implementácia obsahuje okrem bežne dostupných funkcií tiež podporu:

- odosielania *ICMP ping* správ pomocou *ICMP echo*,
- protokolu SNTP (*simple network time protocol*),
- služby mDNS (*multicast DNS*), ktorá slúži pre lokálne mapovanie hostname na IP adresy.

Implementácia *IwIP* implementuje väčšinu bežných funkcií potrebných pre inicializáciu a vykonávanie sieťovej komunikácie vrátane funkcie `gethostbyname()`, ktorá využíva službu DNS k dohladaniu IP adresy. Chyby funkcií sú indikované pomocou návratových hodnôt. Konkrétnu chybu možno dohladať pomocou `errno`, prípadne funkcie `select()`.

Nasledujúci odsek vychádza z [12]. Vývojové prostredie ďalej obsahuje knižnice, ktoré implementujú rozhrania pre rôzne aplikačné protokoly ako HTTP, MQTT alebo *WebSockets*. Rozhranie `esp_http_client` poskytuje funkcie pre implementáciu klientskej HTTP aplikácie. Komunikácia HTTP klienta prebieha pomocou nasledovných funkcií:

1. Funkcia `esp_http_client_init()` nainicializuje komunikáciu kontextom klienta pomocou štruktúry `esp_http_client`.
2. Funkcia `esp_http_client_perform()` je blokujúca a vykoná komunikáciu na základe štruktúry inicializovanej predošlou funkciou. Vykonaním komunikácie sa rozumie vytvorenie komunikácie, zaslanie a stiahnutie dát a ukončenie komunikácie pri chybe.
3. Funkcia `esp_http_client_cleanup()` ukončí komunikáciu so serverom a prípadne uvolní pamäť.

Rozhranie `esp_http_client` tiež podporuje perzistentnú komunikáciu (opakované volanie funkcie *perform()*), HTTPS pomocou implementácie protokolu TLS *Transport layer security* nad schránkami, a HTTP autentizáciu.

[1] http://www.nongnu.org/lwip/2_1_x/index.html

3.3.2 Platforma *ESP8266*

Alternatívou platforme *ESP32* je jej predchodca – platforma *ESP8266*. Informácie o tejto platforme sú vytiahnuté z [17] a [16]. Táto platforma podskytuje menšiu funkcionalitu, napriek tomu sú podporované všetky moduly potrebné pre implementáciu vizualizačnej jednotky: modul pre bezdrôtovú komunikáciu, pamäť RTC, nevolatilnú pamäť EEPROM a podporu zohrania SPI.

V rámci použitej vývojovej dosky sa nachádza čip *ESP8266EX*. Jadrom čipu je jednojadrový procesor s 32 bitovou architektúrou. Pre taktovanie procesora možno použiť frekvenciu 80 alebo 160 MHz. Pre uloženie programu sa používa externá pamäť *FLASH*. Čip požaduje napájacie napätie minimálne 2,5 V, maximálne 3,6 V. Čip *ESP8266EX* podporuje rôzne režimy behu, podobne ako *ESP-32-WROOM*:

- aktívny režim (*Active mode*),
- režim spánku modemu (*Modem-Sleep mode*),
- režim ľahkého spánku (*Light-Sleep mode*),
- režim hlbokého spánku (*Deep-Sleep mode*).

V aktívnom režime je aktívny modul pre rádiofrekvenčný prenos oproti režimu spánku modemu. V režime spánku modemu platforma stále udržuje existujúce spojenie WiFi pomocou správ DTIM (*Delivery traffic indication message*). Periférne rozhrania sú v tomto režime aktívne. Potrebný prúd v tomto režime je okolo 15 mA. V režime ľahkého spánku je pozastavené aj vykonávanie programu jednotkou CPU, prúd v tomto režime je okolo 0,9 mA. V režime hlbokého spánku ostane funkčný len modul RTC. Predpokladaná dĺžka zobudenia aj s pripojením k prístupovému bodu WiFi je 1 sekunda. V hlbokom spánku je prúd požadovaný platformou približne 20 µA.

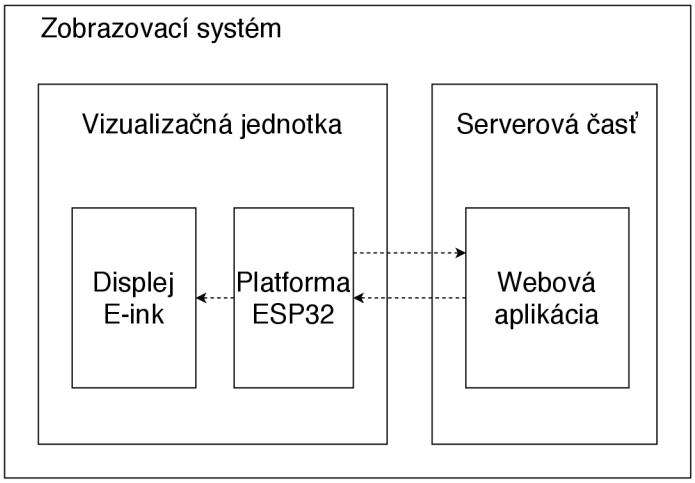
Pre komunikáciu s perifériami čip obsahuje 17 pinov GPIO, AD prevodník (ADC), modul PMW (pulzne-šírková modulácia), rozhrania SDIO (*Secure Digital Input Output Interface*), I2C, I2S a modul pre prenos infračerveným signálom (*IR remote control*).

Modul WiFi pre protokoly *802.11 b/g/n* podporuje 3 módy fungovania – mód stanice BSS, prístupového bodu a promiskuitný mód. Pre zabezpečenie sú podporované protokoly WPA a WPA2. Na SW úrovni je podporovaný je internetový protokol *IPv4*, na transportnej vrstve protokoly TCP aj UDP a tiež aplikačný protokol HTTP. Modul vyžaduje prúd až 170 mA pri odosielaní dát, 56 mA pri prijímaní.

Kapitola 4

Návrh zobrazovacieho systému

Po upresnení požiadavok a zohľadnení možností technológií spomínaných v predošlých kapitolách bol vytvorený výsledný návrh. Obrázok 4.1 zobrazuje blokovú schému navrhovaného systému, ktorý sa bude skladať zo serverovej časti a vizualizačných jednotiek, ktoré s ňou budú komunikovať.



Obr. 4.1: Bloková schéma zobrazovacieho systému.

Vizualizačná jednotka sa skladá z platformy *ESP32*, ktorá bude pripojená k displeju vyrobenému firmou *Waveshare* 2.3.1. S platformou bude komunikovať pomocou rozhrania SPI. Modul displeja už obsahuje ovládač pre toto rozhranie. Raz za určitý časový interval sa platforma zobudí z režimu spánku, aktualizuje obsah displeja a prejde naspäť aj s displejom do režimu spánku. Displej bude zobrazovať textovú aj grafickú časť. Tieto 2 časti budú oddelené. V grafickej časti bude zobrazený obrázok (dvojfarebný, obsahujúci len čiernu a bielu farbu), v textovej časti budú zobrazené bloky textu. Zobrazený obrázok aj bloky textu definuje užívateľ prostredníctvom serverovej aplikácie. Pre implementáciu firmware pre platformu bude použité vývojové prostredie *Arduino*.

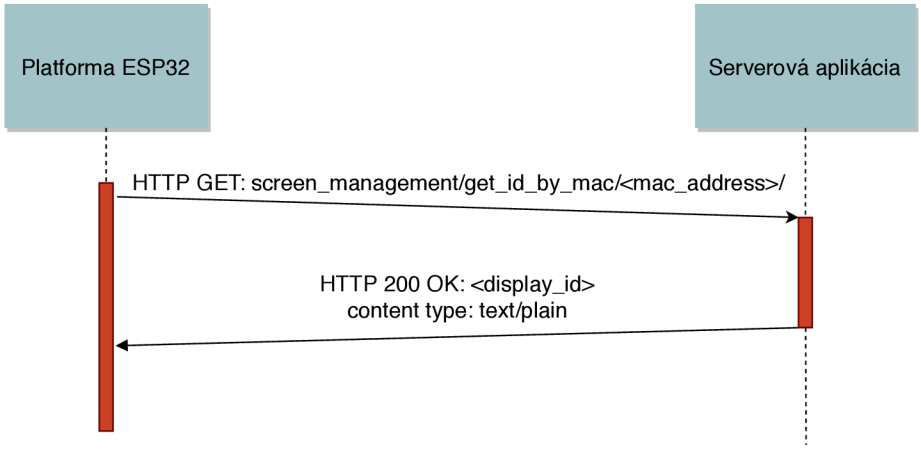
Serverová časť a mikrokontrolér budú spolu komunikovať bezdrôtovo pomocou WiFi a aplikačným protokolom HTTP.

Serverová časť bude implementovaná ako multiplatformná aplikácia v jazyku *Python* verzie 3. Pre implementáciu aplikačného HTTP servera bude používý webový framework *Django*.

Pre začatie komunikácie pomocou protokolu HTTP je potrebné predať platforme *ESP32* meno prístupového bodu WiFi (*WiFi AP*), heslo pre pripojenie, IP adresu servera a číslo portu. Tieto informácie budú do platformy nahrané užívateľom pomocou seriového rozhrania UART s použitím skriptu napísanom v jazyku *Python* verzie 3. Získané parametre sa na platforme uložia do nevolatilnej pamäte typu EEPROM.

4.1 Komunikácia

V tejto podkapitole je popísaná komunikácia medzi firmware nahranom na platforme v rámci zobrazovacej jednotky a serverovou aplikáciou. Serverovou aplikáciou je HTTP server naslúchajúci na IP adrese a porte zadanom užívateľom 4.4. Platforma sa periodicky prepína medzi režimom spánku a provozným režimom. V prípade, že sa platforma prvý krát spustila, musí najskôr získať jedinečný identifikátor inštancie displeja v rámci serverovej aplikácie. Tento identifikátor je displeju priradený systémom po jeho predošlej registrácií v rámci serverovej aplikácie. Mechanizmus získania tohto identifikátora je popísaný v diagrame 4.2.



Obr. 4.2: Získanie identifikátora displeja, diagram správ.

Platforma potrebuje ďalej získať údaje o aktuálnom obsahu – jeho identifikátor, veľkosti a pozície grafickej a textovej časti obsahu. Tieto kontrolné hodnoty sa posielajú vo formáte JSON. Konkrétne ide o objekt (*JSON Object*), ktorý obsahuje potrebné atribúty:

```
{
    content_id: <content id>,
    text_pos_x : <text_pos x>,
    text_pos_y: <text_pos y>,
    text_size_x : <text width>,
    text_size_y : <text heigth>,
    graphics_pos_x : <graphics pos x>,
    graphics_pos_y: <graphics pos y>,
    graphics_size_x : <graphics width>,
    graphics_size_y : <graphics heigth>
}
```

Výpis 4.1: Správa vo formáte JSON obsahujúca údaje o aktuálnom obsahu.

Ak je hodnota aktuálneho identifikátoru zhodná s identifikátorom získanom v predošlej iterácií, nie je potrebné obsah aktualizovať. V opačnom prípade je potrebné ďalej získať aktuálne dáta na zobrazenie – grafickú aj textovú časť. Textová časť sa posiela vo formáte JSON. Textová časť sa skladá z jednotlivých blokov textu, uložených v zázname (*JSON list*). Každý blok má svoje atribúty:

```
[
    {
        text: <ascii text>,
        pos_x : <pos x>,
        pos_y: <pos y>,
        text_size : <font type>,
        font_type : <text size>
    },
    ...
]
```

Výpis 4.2: Formát správy obahujúci textové bloky posielané z webovej aplikácie na zobrazovaciu jednotku.

Posielaný obrázok je vždy čiernobiely. Preto je možné každý jeho pixel kódovať jedným bitom. Výsledný binárny tok (*bit stream*) je tiež požadovaným formátom pre funkciu na vykreslovanie grafického obsahu na displeji. Tvar binárnych dát možno vyjadriť štruktúrou zapísanou v jazyku C:

```
struct {
    uint16_t width;
    uint16_t heigth;
    uint8_t bit_stream[];
} binary_image;
```

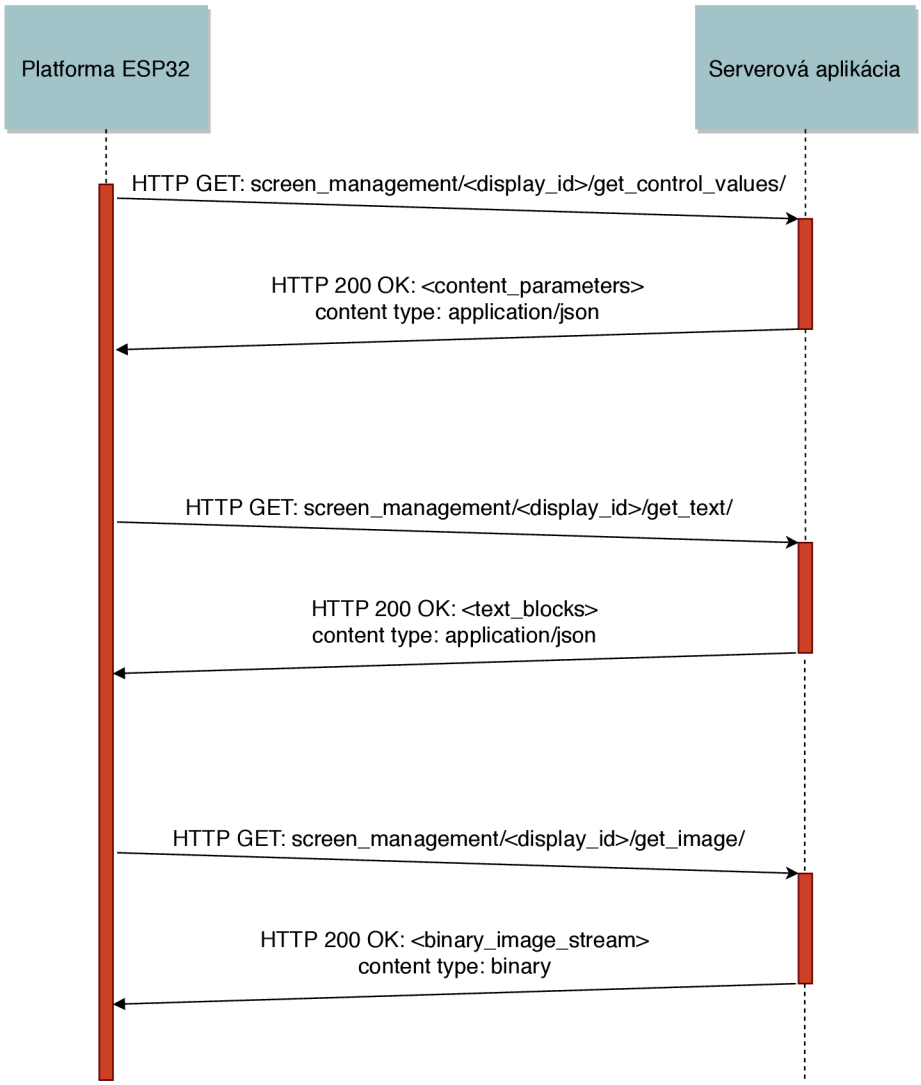
Výpis 4.3: Štruktúra obsahujúca obrázok v tvare bitového toku.

Stiahnutie aktuálnych dát je popísané na diagrame komunikácie 4.3. V prípade, sa v danom obsahu nenachádzajú žiadne textové bloky alebo obrázok, odpovedá serverová aplikácia kódom `204 / No content`. V prípade, že obsah s ID použitým v požiadavke (*HTTP request*) neexistuje, serverová aplikácia odpovie chýbovým kódom `404 / Not found`.

4.2 Zobrazovacia jednotka

Pre vývoj firmware na platforme *ESP32* bude použité prostredie *Arduino*. Toto prostredie obsahuje sadu základných dátových typov, tried a funkcií, ktoré sú vhodné pre implementáciu vstavaných systémov. Okrem toho možno použiť aj knižnice, ktoré sú špecifické pre jednotlivé platformy. Pri implementácií budú použité knižnice na komunikáciu pomocou WiFi, na komunikáciu pomocou protokolu HTTP, knižnica pre spracovanie správ vo formáte JSON, pre prístup do pamäte typu EEPROM a knižníc pre prácu s displejom *E-ink*. V rámci implementácie bude tiež použitá knižnica `esp32-hal-log`, ktorá umožňuje logovanie prostredníctvom sériového rozhrania UART. Pre vypisovanie hlášok slúžia funkcie rôznej dôležitosti (*verbosity level*) logovacej hlášky `log_v()`, `log_d()`, ... `log_e()`.

Platforma sa bude periodicky prepínať medzi aktívnym režimom a režimom hlbokého spánku. Pracovný cyklus sa začne aktiváciou modulu WiFi, pripojením k prístupovému



Obr. 4.3: Aktualizácia obsahu displeja, diagram správ.

bodu a získaním údajov o aktuálnom obsahu. V prípade, že sa zobrazované dáta nezmenili, platforma sa uspí. Inak platforma stiahne aktuálnu textovú aj grafickú časť. Po stiahnutí oboch častí je deaktivovaný modul WiFi. Následne sa z režimu spánku zobudí displej, dáta sa spracujú, zobrazia a displej sa uloží späť do režimu spánku. Potom prejde do režimu hlbokého spánku aj platforma. Cyklus je popísaný v prílohe C.1.

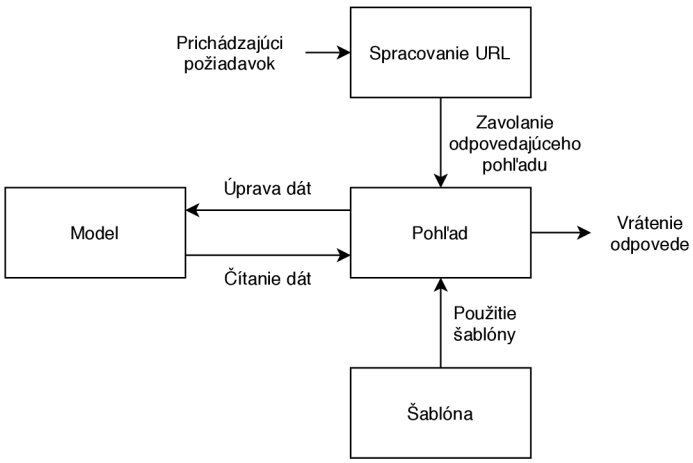
Pre komunikáciu so serverovou aplikáciou bude použitá trieda `HTTPClient`. Táto knižnica umožňuje posielanie requestov *GET* a *POST*. Odpoveď (*HTTP response*) môže byť prijatá ako text, prípadne aj inom formáte pomocou funkcie `writeToStream()`. Trieda `HTTPClient` využíva triedu `WiFiClient`, ktorá zapuzdruje aplikačné rozhranie pre prácu so schránkami (*BSD sockets*).

Pre zobrazovanie dát na displeji bude použitá knižnica `GxEPD`. Táto knižnica je vytvorená pre displeje, ktoré využívajú technológiu elektronického papiera (*Electronic paper displays*). Podporované sú displeje, ktoré využívajú rozhranie SPI, od *Waveshare* a *Dalian*

Good Display. Knižnica `GxEPD`[1] využíva v implementácií knižnicu `AdafruitGFX library`[2]. Knižnica `AdafruitGFX` je grafická knižnica vhodná predovšetkým pre platformy používané vo vstavaných systémoch. Rozhranie knižnice umožňuje zobrazovanie textového aj grafického obsahu.

4.3 Serverová aplikácia

Pre implementáciu serverovej aplikácie bude použitý webový framework *Django*. Informácie o tomto frameworku sú získané z [3] a oficiálnej dokumentácie projektu Django [7]. Podľa [3] architektúra frameworku Django vychádza z návrhového vzoru *MVC* (Model-View-Controller). Jeho základnými prvkami sú model (*Model*), pohľady (*Views*), šablóny (*Templates*) a modul pre mapovanie cesty k zdroju na užívateľom definované funkcie (*URL dispatcher*).



Obr. 4.4: Základná schéma odbavenia požiadavku v rámci frameworku Django. Schéma vychádza z [3].

Obrázok 4.4 zobrazuje základnú schému odbavenia požiadavku. Po prijatí požiadavku sa pomocou komponenty *URL Dispatcher* daná cesta k zdroju priradí k pohľadu – funkcií, ktorá tento požiadavok odbaví. Táto funkcia obrdrží daný požiadavok, spracuje dáta a vráti odpoveď. Pri spracovaní dát môže pristúpiť do databázy pre čítanie/zápis. Model definuje túto databázu. Pri vytváraní odpovede môže daný pohľad použiť šablónu. Šablóna definuje kód, ktorý sa vykonáva/interpretuje na strane klienta. Framework *Django* ďalej obsahuje rôzne pomocné moduly a aplikácie pre zjednodušenie tvorby serverových aplikácií, napríklad pre tvorbu a spracovanie formulárov, rôznych kontrol vstupov na strane servera, autentizáciu užívateľov, administrátorské stránky alebo podporu zabezpečenia CSRF. CSRF (*cross-site resource forgery*) zabezpečuje ochranu proti prijímaniu požiadavok od iných stránok.

V jednom projekte vytvorenom frameworkom sa môže nachádzať viac aplikácií (*Django apps*). Pre každý účel by mala byť vytvorená jedna aplikácia. Hlavnou úlohou serveru je spravovanie obsahu zobrazovacích jednotiek, za týmto účelom bude vytvorená aplikácia

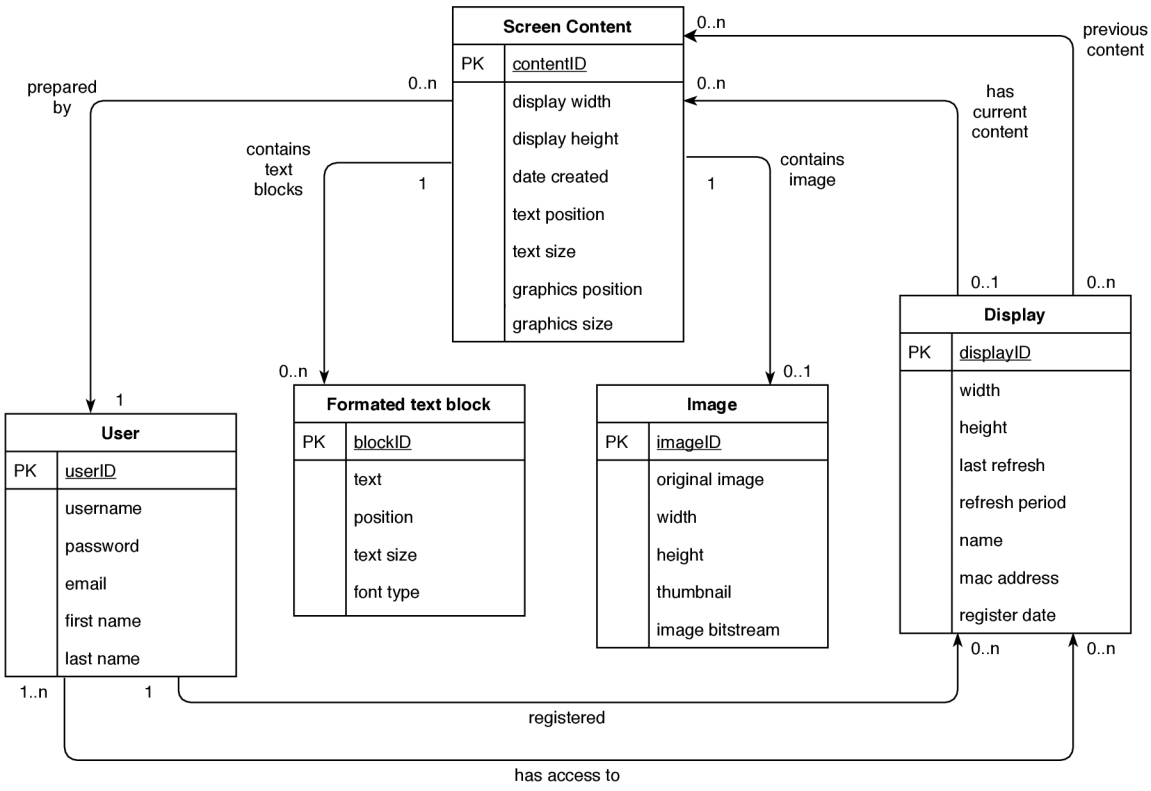
[1] https://github.com/ZinggJM/GxEPD
[2] https://github.com/adafruit/Adafruit-GFX-Library

`screen_management`. Ďalšou použitou aplikáciou bude aplikácia `users_auth` pre správu užívateľov.

Užívateľ sa do systému zaregistruje sám, po registrácií nebude mať prístup k žiadnej vizualizačnej jednotke. K panelu môže získať prístup buď tak, že mu ho udelí majiteľ niektorej existujúcej, prípadne on sám pridá do systému novú vizualizačnú jednotku.

4.3.1 Návrh databázy

Pre reprezentáciu modelu sa využíva systém *objetkovo-relačného mapovania*. Framework *Django* podporuje na pozadí rôzne databázové systémy. Ako databázový systém pre potreby projektu postačuje databázový systém *SQL-lite*. V tomto databázovom systéme je databáza fyzicky uložená v jednom binárnom súbore. *Django* ďalej podporuje prácu s migráciami, ktoré slúžia pre zjednodušenie aplikovania zmien v modeli.



Obr. 4.5: ER diagram systému pre správu vizualizačných jednotiek.

Na obrázku 4.5 je zobrazený ER (*entity-relationship*) diagram systému pre správu vizualizačných jednotiek. Pre vizualizačnú jednotku je potrebné ukladať MAC adresu pre spárovanie, rozmery displeja a periódu aktualizácie obsahu displeja. Užívateľ môže označiť jednotku menom pre jej jednoduchšiu identifikáciu. Ďalej sa preň ukladá časové razítko pri každom obnovení obsahu a tiež dátum registrácie jednotky pre potreby radenia displejov pri vypísaní ich zoznamu. Užívateľ, ktorý jednotku pridal do systému, môže upravovať zoznam užívateľov, ktorí k nej majú prístup. Každý displej zobrazovať maximálne jeden obsah v určitom čase.

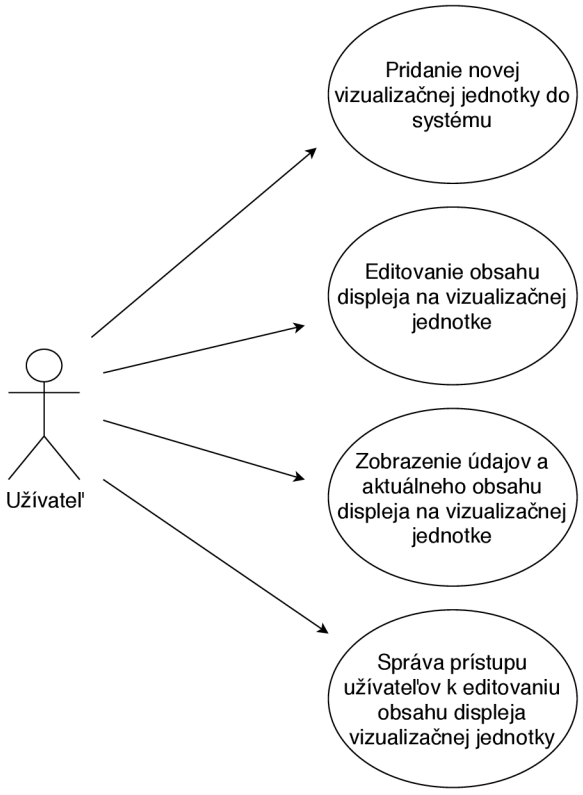
Obsah zobrazuje grafickú a textovú časť. Tieto časti sú oddelené deliacou čiarou, ktorá môže byť buď horizontálna (textová časť sa nachádza nad grafickou alebo naopak), alebo

vertikálna (textová časť sa nachádza vedľa grafickej). Grafická časť obsahuje maximálne 1 obrázok. Obrázok vyplní vždy kompletnú grafickú časť. V textovej časti sa môže nachádzať viacero blokov textu, každému bloku textu možno nastaviť veľkosť textu a typ fontu. Jeden obsah môže byť zobrazený na viacerých displejoch naraz, prípadne nemusí byť zobrazený na žiadnom. Všetky displeje, na ktorých je daný obsah zobrazený, musia mať rovnakú veľkosť.

V prípade pridania obrázku sa vždy nahrá do systému obrázok v pôvodnej veľkosti, po jeho nahratí sa oreže podľa parametrov zadaných používateľom, zmení sa jeho veľkosť podľa veľkosti grafickej časti a prekonvertuje sa na čiernobiely. Takto naformátovaný obrazok sa uloží a bude použitý pri zobrazení daného obsahu, ako ukážka. Ďalej sa z neho vytvorí a uloží obrázok v binárnom formáte, v akom bude predaný vizualizačnej jednotke.

4.3.2 Užívateľské rozhranie

Serverová aplikácia slúži pre správu obsahu displejov pridaných do systému. Užívatelia búdú párovať nové vizualizačné jednotky s touto aplikáciou, v prípade potreby budú môcť upravovať zadané údaje o vizualizačnej jednotke. Ďalej budú pomocou systému pridávať nový obsah displeja. Jeden vytvorený obsah bude môcť byť zdielaný medzi viacerými displejmi. Tento obsah bude možné ďalej upravovať a bude dostupná aj história obsahu zobrazenom na displeji. Okrem obsahu bude môcť užívateľ nastaviť aj periódu aktualizácie. Užívateľ, ktorý do systému vizualizačnú jednotku pridal, bude môcť riadiť prístup ostatných užívateľov k úprave obsahu displeja na jednotke. Tieto prípady použitia užívateľom aplikácie sú zhrnuté v diagrame použitia 4.6.



Obr. 4.6: Diagram prípadov užitia aplikácie pre správu obsahu vizualizačných jednotiek.

Pre automatické formátovanie formulárov je použitá knižnica `django-crispy-forms`[3]. Táto knižnica je závislá na knižnici `Bootstrap 3`[4].

4.3.3 Spracovanie grafického vstupu

Do aplikácie sa tiež bude nahrávať grafický vstup – obrázok. Okrem nahraného obrázka užívateľ zadá aj súradnice, podľa ktorých sa oreže okno. Orezaný obrázok sa zmení jeho veľkosť podľa veľkosti grafickej časti displeja. Ďalej bude tento obrázok prekonvertovaný pomocou metódy *Dithering*[5]. Upravený obrázok sa uloží vo formáte PNG[6] ako ukážka (*thumbnail*) a vo formáte, v ktorom bude predaný vizualizačnej jednotke 4.3. Pre prácu s obrázkami bude použitá knižnica `Pillow`[7], ktorá vznikla z pôvodnej knižnice `PIL` (*fork*). Táto knižnica podporuje väčšinu štandardných formátov obrázkov.

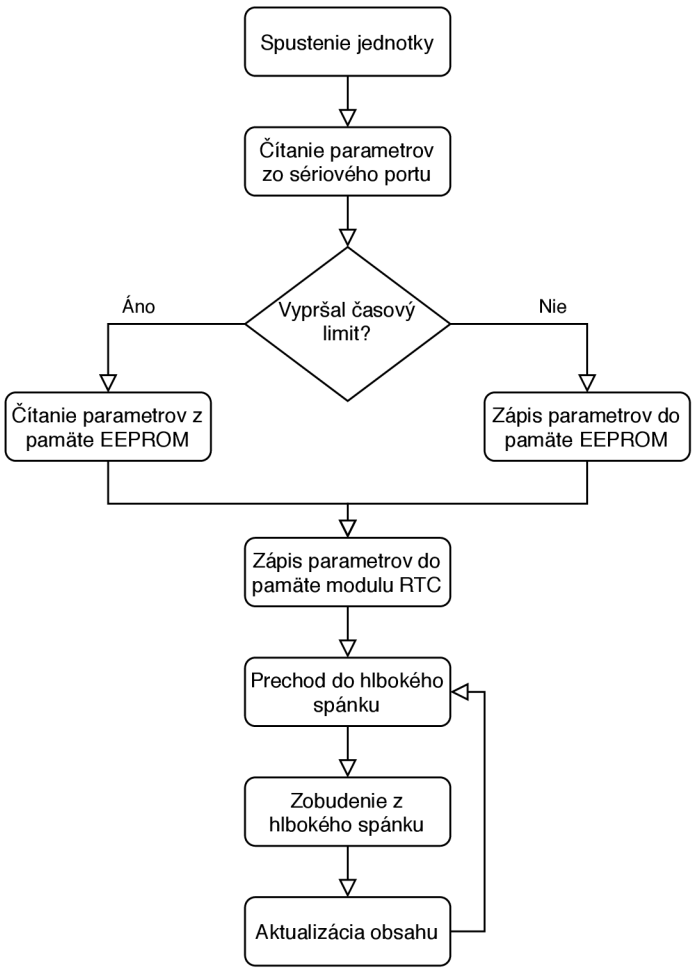
4.4 Nastavenie parametrov zobrazovacej jednotky

Aby mohla zobrazovacia jednotka komunikovať so serverovou aplikáciou, potrebuje získať parametre spomenuté v úvode tejto kapitoly 4.4 – heslo pre pripojenie k prístupovému bodu WiFi, heslo potrebné pre pripojenie k prístupovému bodu, IP adresu servera a komunikačný port. Tieto parametre bude môžné platforme predať pripojením platformy k osobnému počítaču pomocou zbernice USB (*Universal Serial Bus*).

Po pripojení platormy k počítaču užívateľ spustí skript, ktorému predá parametre pre odoslanie platforme a port, na ktorý je platforma pripojená. Skript bude implementovaný v jazyku python. Skript sa najskôr pokúsi pripojiť na sériový port. Potom užívateľ spustí program na platforme pomocou stlačenia tlačidla `EN`. Po spustení platforma vypíše logovacie výpisy, skript ich vloží na štandardný výstup. Skript ďalej predá potrebné parametre platforme a po ich predaní bude ďalej vypisovať logovacie výpisy.

Platforma po spustení bude očakávať prijatie parametrov prostredníctvom rozhrania UART. Prijaté parametre ďalej uloží do pamäte typu EEPROM a do volatilnej pamäte modulu RTC. Platforma potom prejde do režimu hlbokého spánku. Po zobudení používa parametre uložené v pamäti RTC pre aktualizáciu obsahu displeja. Ak platforma po reštarte (*reboot*) neobdrží komunikačné parametre zo sériového portu do časového limitu (*timeout*), prečíta tieto parametre z pamäte EEPROM. Algoritmus získania parametrov je popísaný v diagrame 4.7.

[3] https://github.com/django-crispy-forms/django-crispy-forms
[4] https://getbootstrap.com/docs/3.4/
[5] https://en.wikipedia.org/wiki/Dither
[6] https://en.wikipedia.org/wiki/Portable_Network_Graphics
[7] https://pillow.readthedocs.io/en/stable/



Obr. 4.7: Získanie parametrov potrebných pre komunikáciu so serverovou aplikáciou.

Kapitola 5

Implementácia

V tejto kapitole sú popísané detaily implementácie navrhnutej serverovej aplikácie pre správu vizualizačných jednotiek, firmware pre aktualizáciu obsahu displeja pre platformu *ESP32* a skriptu použitého pre nastavenie konfigurácie vizualizačnej jednotky. Návrh všetkých častí je popísaný v kapitole 4.

5.1 Firmware zobrazovacej jednotky

Firmware bol vytvorený v prostredí *Arduino*. Prostredie a použité knižnice sú popísané v 4.2. V tejto podkapitole sú popísané detaily implementácie komunikácie so serverovou aplikáciou a aktualizácie obsahu pre platformu *ESP32*.

5.1.1 Komunikácia so serverovou aplikáciou

Pre komunikáciu so serverovou časťou je použitá trieda `HTTPClient`. Po vytvorení inštancie tejto triedy je potrebné zavolať metódu `begin()`. Ako argument je do tejto metódy predaná referencia na objekt triedy `WiFiClient`, pomocou ktorého by už malo byť vytvorenie spojenie s daným prístupovým bodom WiFi. Metóda spracuje aj ďalšie argumenty - meno serveru (`hostname`) prípadne IP adresu, port a cestu k zdroju.

Ďalej sa pomocou metódy `GET()` vykoná požiadavok *GET* a príjme odpoveď. V tejto metóde sa tiež vytvorí nové spojenie na úrovni schránok, ak je to potrebné. Po prijatí odpovede metóda vráti kód odpovede (*HTTP response code*). Na základe návratového kódu možno reaguje firmware na platforme *ESP32* týmito spôsobmi:

1. kód *200, OK*: platforma spracuje prijaté dáta štandardne,
2. kód *204 no content*: platforma vynechá spracovanie daných dát,
3. ostatné kódy vrátane *404, not found*: znamenájú, že došlo ku chybe pri komunikácií so serverovou aplikáciou a platforma sa prepne do režimu spánku.

Odpoveď možno prijať v textovej podobe pomocou metódy `getString()`. Táto metóda vracia objekt triedy `String` (trieda implementovaná v rámci prostredia *Arduino*). Metóda `getString()` je v projekte použitá na príjimanie odpovedí, kedy sú dáta prijaté vo formáte JSON, prípadne *text/plain*. V prípade, že sa prijala správa vo formáte JSON, implementoval som jej deserializáciu na základe ukážky v rámci knižnice `ArduinoJson`[1].

[1] https://github.com/bblanchon/ArduinoJson/blob/6.x/examples/JsonHttpClient/JsonHttpClient.ino

Pre príjimanie binárnych dát bola použitá metóda `writeToStream()`. Argumentom tejto metódy je ukazateľ na objekt triedy, ktorá dedí od abstraktnej triedy `Stream`. V implementácií metódy `getString()` je metóda `writeToStream()` volaná s objektom triedy `StreamString`, ako argumentom. Vytvoril som triedu pre binárne dáta `FixedByteStream()`, ktorá dokáže prijať dáta až do veľkosti zadanej v konštruktore. Riešenie s obmedzenou maximálnou velkosťou je v rámci projektu postačujúce, keďže na základe predošlej správy obsahujúcej informácie o obsahu možno určiť dopredu očakávané množstvo prijatých dát.

Po prijatí odpovede zo servera sa zavolá metóda `end()`, ktorá pomocou rozhrania schránok uzavrie spojenie `TCP`. Zamedziť nekonečnému čakaniu na odpoveď od servera možno zamedziť metódou `setTimeout()`. V takom prípade metóda `getStream()` vráti odpovedajúcu chybovú návratovú hodnotu.

5.1.2 Zobrazovanie obsahu

Pre úpravu obsahu displeja je nutné vytvoriť inštancie tried `GxIO_CLASS` a `GxEPD_CLASS`, ktorým je do konštruktoru potrebné ako argument predať piny rozhrania SPI. Najskôr je potrebné inicializovať displej pomocou metódy `init()`, nastaviť farbu pozadia, farbu textu a orientáciu displeja pomocou metód `fillScreen()`, `setTextColor()` a `setRotation()`. Pre vloženie obrázku je použitá metóda `drawBitmap()`, ktorej je potrebné do argumentu zadať polohu, veľkosť obrázka a hodnoty pixelov zadaných v jednorozmernom bitovom poli. Pre vkladanie textu sú použité metódy `setCursor()` a `println()`. Ďalej sa nastavuje font metódou `setFont()`. Použité sú fonty v rámci knižnice `AdafruitGFX`, fonty sú reprezentované ako bitová mapa. V projekte je použitý font *Freesans*, pre každú dostupnú veľkosť je definovaná samostatná bitová mapa. Veľkosť fontu *Freesans* je od približne 1800 do 8130 bajtov pre každú veľkosť. Pre vypisovanie textu je ďalej možné povoliť zalamovanie textu na nový riadok, aby sa predišlo jeho pretečeniu mimo plochu displeja metódou `setTextWrap()`.

Obsah sa aktualizuje len vtedy, ak sa zmenila hodnota `content_id` v rámci získaných informácií o aktuálnom obsahu. Predošlá hodnota `content_id` sa ukladá do pamäte modulu RTC (*RTC slow memory*).

5.1.3 Firmware pre platformu ESP8266

V 6.1.3 je spomenuté, že bolo pre zníženie spotreby vizualizačnej jednotky upraviť firmware pre platformu *ESP8266*. Riešenie si nevyžadovalo väčšie zásahy do návrhu, prostredie *Arduino* obsahuje takmer všetky použité triedy s ekvivalentným rozhraním rozhraniu tried pre platformu *ESP32*. Odlišoval sa najmä spôsob prístupu do pamäte typu EEPROM a pamäte modulu RTC. Tiež bolo potrebné získať informáciu o tom, či bola platforma spustená, alebo len zobudená z režimu hlbokého spánku. Za týmto účelom bola použitá metóda `getResetInfoPtr()`. Okrem toho bolo potrebné pridať explicitné zapínanie a vypínanie modulu WiFi.

5.2 Webová aplikácia pre správu vizualizačných jednotiek

Táto podkapitola obsahuje detaily implementácie webovej aplikácie pre správu jednotiek popísanej v 4.3. V tejto podkapitole je bližie popísaný vytvorený databázový model pomocou objektovo-relačného mapovania, pohľady pre komunikáciu s platformou, užívateľské rozhranie aplikácie a spracovanie grafického vstupu.

5.2.1 Databázový model

Implementovaný databázový model vychádza z návrhu 4.3.1. Jednotlivé tabuľky sú implementované ako triedy v rámci modulu `models`. Pre vytvorenie novej tabuľky v rámci frameworku *Django* je potrebné vytvoriť novú triedu, ktorá dediť od základnej triedy `Model`. Táto trieda bude obsahovať atribúty reprezentujúce políčka – objekty tried, ktoré dedia od základnej triedy `Field`. Pri konštruktore triedy je možné pridať argumenty s predvolenou hodnotou, či hodnota môže byť prázdna, či musí byť unikátna, prípadne pridať validátory vkladaných hodnôt. Ďalej je potrebné u textových reťazcov nastaviť ich maximálnu dĺžku.

Pre vytvorenie cudzieho kľúča je potrebné použiť triedu `ForeignKey`, do konštruktoru sa ako parametre predávajú trieda, na ktorú sa bude kľúč odkazovať, identifikátor pre prístup z triedy, na ktorú sa cudzí kľúč odkazuje a čo sa má urobiť pri zmazaní odkazovaného objektu s objektom, ktorý toto pole obsahuje – môže sa s ním buď zmazať, prípadne sa mu nastaví cudzí kľúč na hodnotu `None`. Pre vytvorenie vzťahu *M:N* nie je potrebné vytvárať ďalšiu tabuľku, možné je použiť triedu `ManyToManyField`. Objekt tejto triedy je množinou objektov, ktoré sú v danom vzťahu.

Množina (`Queryset`) je tiež výsledkom dotazu nad databázou. K objektom danej tabuľky možno získať pomocou atribútu `objects` triedy `Model`. Nad týmto atribútom možno potom voľať funkcie pre získanie všetkých objektov metódou `all()`, získať konkrétny objekt metódou `get()` alebo filtrovať objekty pomocou `filter()`.

Pre komunikáciu s platformou budú implementované potrebné pohľady, ktoré po obdržaní HTTP požiadavku získajú potrebné dáta z databázy a vrátia ich vo formáte definovanom v 4.1. Pre vytvorenie odpovede v potrebnom formáte budú použité objekty tried `JsonResponse` a `FileResponse`. Pre komunikáciu s užívateľom budú pre komunikáciu použité okrem pohľadov aj formuláre (modul `forms`) a šablóny (modul `templates`).

Vizualizačné jednotky sú uložené v triede `Display`. Po vytvorení získa každý displej identifikátor, ktorý používa firmware na platforme *ESP32* pre aktualizáciu obsahu. *Mac adresa* je uložená ako textový reťazec (`CharField`), pre overenie správneho tvaru je použitý validátor pomocou regulárneho výrazu (`RegexValidator`) a tento reťazec musí byť v rámci systému jedinečný. Rozmery displeja sú uložené celočíselne (`IntegerField`). Dátum registrácie a časové razítko sú uložené v triede `DataTimeField`. Pre Periódu aktualizovania obsahu je použitá trieda `DurationField`. Trieda `Display` tiež obsahuje cudzí kľúč pre užívateľa, ktorý tento displej pridal do systému (`ForeignKey`). Okrem toho trieda obsahuje zoznam užívateľov, ktorí majú prístup k displeju pomocou triedy `ManyToManyField` a cudzí kľúč obsahu, ktorý je pre tento displej aktuálny. Tento cudzí kľúč môže byť aj prázdny.

Obsah displeja je uložený v triede `ScreenContent`. Rozmery displeja, pre ktorý je vytvorený a tiež aj rozmery textovej a grafickej časti sú uložené celočíselne (`IntegerField`). Polohy grafickej a textovej časti sú rovnako uložené celočíselne. `ScreenContent` ďalej obsahuje cudzí kľúč, ktorý odkazuje na aktuálny obrázok v tomto obsahu.

Pre textové bloky je vytvorená trieda `FormattedTextBlock`. Text je uložený ako textové pole, pozícia textu je uložená v celočíselných atribútoch. Typ fontu je uložený ako textové pole, veľkosť textu celočíselne. Trieda `FormattedTextBlock` ďalej obsahuje cudzí kľuč obsahu, v ktorom sa nachádza. Ak bude zmazaný obsah, zmažú sa aj textové polia k nemu priradené. Z obsahu možno získať textové polia napríklad pomocou metódy filter:

```
content = get_object_or_404(ScreenContent, pk=content_id)
blocks = FormattedTextBlock.objects.filter(in_content=content)
```

Výpis 5.1: Získanie množiny textových blokov priradených k obsahu displeja s daným `content_id`.

Pre tabuľku obrázok sú rozmery uložené v objektoch triedy `IntegerField`, pre uloženie originálneho obrázku a ukážky sformátovanej ukážky je použitá trieda `ImageField`, pre uloženie obrázku v binárnom tvare je použitý `FileField`. Triedy `ImageField`, `FileField` slúžia na správu súborov. Súbory sú fyzicky uložené mimo databázy. Cesta k súborom uloženým na disku je definovaná premennou `MEDIA_ROOT` v rámci súboru `settings.py`. Každému atribútu je možné ďalej definovať podadresár, v ktorom budú uložené pomocou argumentu `upload_to`:

```
thumbnail_image = models.ImageField(upload_to='thumbnail')
```

Výpis 5.2: Definovanie podadresára kde budú obrázky uložené. Obrázok bude tak uložený v adresári `MEDIA_ROOT/thumbnail`.

5.2.2 Komunikácia s vizualizačnou jednotkou

V rámci serverovej aplikácie je potrebné implementovať pohľady pre komunikáciu s jednotkou v rámci modulu `views` a namapovať ich na zodpovedajúce cesty k zdrojom v module `urls`. Metóda teda prijme požiadavok, získa potrebné hodnoty z databáze a zabalí ich do HTTP odpovede. Nebude potrebné používať šablóny.

Po spustení (*reboot*) musí platforma najskôr získať ID. Pre určenie ID na základe MAC adresy je implementovaná funkcia `get_id_by_mac()`. Tá vráti ID vizualizačnej jednotky, alebo odpovie chybovým kódom `404` v prípade, že vizualizačná jendotka nie je v aplikácií zaregistrovaná.

Pre získanie údajov o obsahu je implementovaná funkcia `get_control_values()`. Najskôr sa získajú údaje o aktuálnom obsahu a uložia sa do slovníka. V prípade, že displeju nie je priradený žiadny aktuálny obsah, všetky vrátené hodnoty budú nulové. Vysledná odpoveď sa vytvorí triedou `JsonResponse` (políčko *content type* bude mať hodnotu *application/json*), do ktorej sa slovník s hodnotami vloží ako argument.

Pre získanie textových blokov je v rámci modulu views definovaná funkcia `get_text()`, ktorá berie ID obsahu ako argument. Pomocou metódy `filter()` sa získajú všetky bloky, ktoré prislúchajú danému obsahu. Pomocou funkcie `serialize()` modulu `serializers` sa vytvorí slovník, pri čom sa vyberú len potrebné polia. Nakoniec sa z tohto slovníka vytvorí odpoveď triedou `JsonResponse`.

Pre získanie obrázku je implementovaná metóda `get_image()`, ktorá berie ako argument ID obsahu. Obrázok sa predáva binárne ako *binary/application*. Keďže knižnica pre spracovanie HTTP odpovede pre platformu *ESP32* vyžaduje vyplnenú veľkosť dát, špočíta sa a pridá táto hodnota explicitne:

```
response = FileResponse(current_image.image_stream)
response['Content-Length'] = image_size
return response
```

Výpis 5.3: Explicitné nastavenie po hodnoty *Content-Length* v HTTP odpovedi vyžadovanej knižnicou `HTTPClient` pre platformu *ESP32*.

5.2.3 Užívateľské rozhranie

Návrh pohľadov pre komunikáciu s užívateľom je definovaný v 4.3.2. V tejto sekcií sú popísané detaily implementácie jednotlivých pohľadov. Jedna časť pohľadov slúži pre správu vizualizačnej jednotky (pridanie nového panelu, obmedzenie prístupu iných užívateľov), druhá časť slúži pre editovanie obsahu displejov.

Uživateľské rozhranie je definované v funkciách modulu `views`. Pre užívateľské rozhranie sú ďalej použíté šablóny a formuláre (modul `forms`).

Šablóny obsahujú kód, ktorý sa pošle v tele HTTP odpovede a je interpretovaný na strane v klienta. V šablónach možno okrem kódu odoslaného jazyka použiť aj príkazy, ktoré slúžia na jeho generovanie. Pre vytvorenie výslednej HTTP odpovede sa použije funkcia `render`, ktorá pre zadaný HTTP požiadavok vytvorí odpoveď. Táto funkcia ďalej berie ako argument použitú šablónu a tiež kontext. Kontext je slovník hodnôt, ktoré sa v šablóne použijú.

Všetky použité šablóny by mali obsahovať rovnakú hlavičku *HTML* súboru, v tele rovnaký nadpis (`header`) a panel pre navigáciu. Navigácia obsahuje základné odkazy – odkaz na hlavnú stránku, odkazy na stránku pre prihlásenie, odhlásenie a registráciu. Pre zdielanie kódu je možné využiť dedičnosť šablón. Hlavička *HTML* súboru, nadpis a panel pre navigáciu budú implementované v základnej šablóne. Táto šablóna bude obsahovať blok `block_content`, ktorý bude prázdny. Od základnej šablóny budú potom dediť ostatné šablóny použitím tagu `{% extends 'template_base.html' %}` na začiatku šablóny. Odvodené šablóny zadefinujú obsah bloku `block_content`.

Formuláre sú triedy, ktoré dedia od základnej triedy `Form`. Vytvorená trieda obsahuje parametre – jednotlivé polia v rámci formulára. Framework *Django* obsahuje triedy pre základné polia. Do konštruktorov tried základných polí sa predávajú argumenty:

- označenie pola (`label`),
- zoznam funkcií pre kontrolu vstupnej hodnoty (`validators`),
- u pola s možnosťou výberu (trieda `ChoiceField`) zoznam možností,
- či má byť pole skryté, využíva sa pre pridanie kontextu.

Trieda odpovedajúca formuláru môže ďalej dediť od triedy `ModelForm`, v tomto prípade je potrebné vnútri definovať metadáta, definovaním triedy `Meta`. V tejto triede sa potom špecifikuje:

- tabuľka, objekt triedy z modulu `models`, ktorému formulár odpovedá,
- zoznam políčok tabuľky, pre ktoré sa má vytvoriť pole vo formulári,
- zoznam mien jednotlivých políčok.

Tiež je možné definovať aj ďalšie polia v rámci tohto formulára. Ak je tabuľke u niektorého políčka definovaný validátor, tiež sa použije pre kontrolu vstupných dát.

Okrem použitia validátoru možno skontrolovať zadané hodnoty atribútov definovaním metódy `clean_<meno_atribútu>()`, kde možno pri validácií použiť všetky hodnoty zadané vo formulári. Táto metóda bude zavolaná automaticky.

Do šablóny sa tento formulár predá pomocou kontextu, v rámci šablóny je ďalej potrebné vytvoriť formulár v jazyku *HTML* tak, ako v 5.4.

```
<form method="post" action="{% url 'screen_management:add_data' %}">
{{ form }}
<input type="submit" value="odoslat">
</form>
```

Výpis 5.4: Použitie formuláru *Django Form* v rámci šablóny. Po vyplnení sa požiadavok predá metóde `add_data()` definovanej v module `views`

Správa vizualizačných jednotiek

Po prihlásení do aplikácie sa užívateľovi zobrazí zoznam vizualizačných jednotiek, ku ktorým má prístup a tlačidlo s možnosťou pridania novej jednotky do systému. Tento zoznam jednotiek je implementovaný vo funkcií `logged_index()` modulu `views`.

Pridanie nového vizualizačnej jednotky je implementované v pohľade `add_display()`. Pre pridanie je použitý formulár založený na tabuľke `Display` definovanej v module `models`, pri čom užívateľ zadáva MAC adresu platformy vo vizualizačnej jednotke, jej meno (ľubovoľné), veľkosť displeja a periódu aktualizácie obsahu. Po kontrole správnosti zadaných údajov sa doplní údaj o majiteľovi displeja a dátume registrácie. Potom je vizualizačná jednotka pridaná do databázy.

Po pridaní vizualizačnej jednotky do systému je užívateľ prepojený na podstránku, kde sa nachádzajú údaje o jednotke. Tiež sa tu nachádza odkaz pre správu obsahu displeja jednotky a pre správu užívateľov, ktorí k nej majú prístup. Táto podstránka je implementovaná v pohľade `display_detail()`. Zoznam užívateľov, ktorí k nej majú prístup, je implementovaný v pohľade `granted_users()`. Majiteľ môže odoberať prístup iným užívateľom, prípadne ho pridať pomocou políčka s možnosťou výberu.

Správa obsahu displejov

V rámci zobrazenia informácií o vizualizačnej jednotke je možné zobraziť aj obsah jej displeja. Okrem toho je možné vytvoriť nový obsah, prípadne použiť niektorý existujúcich z obsahov, ktoré boli použité na iných displejoch s rovnakou veľkosťou displeja a užívateľ má ku ním prístup.

Pre zobrazenie obsahu je implementovaný pohľad `content_detail()`. V rámci pohľadu je zobrazená textová aj grafická časť. Pre textovú časť je zobrazená jej poloha, veľkosť a zoznam textových blokov, ktoré sa v nej nachádzajú. Tiež je tu odkaz na pohľad `add_text_block()`, ktorým je možné pridať ďalši text. Existujúce textové bloky je tiež možné zmazať. Pre grafickú časť je uvedená jej poloha, veľkosť a tiež je zobrazená ukážka (`thumbnail`).

V prípade úpravy existujúceho obsahu sa vytvorí hľbková kópia pre prípad, že by bol tento obsah zdielaný medzi viacerými vizualizačnými panelmi. Tým sa zabráni prepísaniu obsahu na všetkých vizualizačných paneloch. Zobrazenie zoznamu vhodných obsahov je implementované v pohľade `browse_contents()`.

Každý obsah má textovú a grafickú časť, tieto časti sú oddelené. Preto užívateľ pri vytvorení nového obsahu zadá polohu deliacej čiary a jej orientáciu (vertikálne, horizontálne) a tiež vyberie polohu jednotlivých častí. Pri pridaní bloku užívateľ zadá polohu nového bloku textu, veľkosť písma a samotný text. Pri pridaní obrázku vyberie užívateľ zo súboru obrázok v niektorom štandardnom formáte a súradnice pre jeho orezanie. Obrázok bude po pridaní orezaný a jeho veľkosť upravená podľa veľkosti grafickej časti obsahu. Vstupné dáta sú pred nahraním skontrolované.

Spracovanie grafického vstupu

Ako je spomenuté v 4.3.3, vložený obrázok bude spracovaný pomocou knižnice `Pillow`. Obrázok sa otvorí pomocou metódy `open()`, do ktorej sa ako argument predáva objekt s rozhraním súboru (*file like object*). Z pola tabuľky `Image` sa tento objekt získa pomocou atribútu `file` nad daným polom. Otvorený obrázok je potom orezaný na základe súradníc zadaných užívateľom, ďalej je upravená jeho veľkosť na požadovanú veľkosť grafickej časti

obsahu. Následne je tento obrázok prekonvertovaný na binárny (dvojfarebný) pomocou metódy *Dithering* použitím metódy `convert()`.

Výsledný dvojfarebný obrázok je uložený ako ukážka (`thumbnail`). Ukázka je do súborového systému uložená vo formáte PNG. Pre konvertovanie do výsledného formátu je použitá metóda `save()` triedy `Image`. Táto metóda berie ako argument objekt súboru (`file like object`). Ako objekt súboru je použitý objekt triedy `BytesIO`. Tento objekt je ďalej potrebné zapísať do súboru, k čomu slúži trieda `InMemoryUploadedFile` z modulu `uploadedFile` v rámci projektu *Django*. Ako argumenty je potrebné predať objekt triedy `BytesIO`, použitý formát, meno súboru a jeho veľkosť. V rámci *Django* sa trieda `UploadedFile` využíva pre uloženie nahraných obrázkov z formulárov. Referenciu na výsledný objekt tak následne stačí priradiť do atribútu triedy `FormattedImage` z modulu models. Pri uložení výsledného objektu triedy `FormattedImage` pomocou metódy `save()` sa tento obrázok uloží do súborového systému.

Vytvorený dvojfarebný obrázok je ešte potrebné zmeniť na formát požadový firmwarom na platforme. Tento formát je binárny, na začiatku sú zapísané rozmery obrázka ako 2 dvojbajtové neznamienkové hodnoty a potom sú zapísané hodnoty jednotlivých pixelov – každý pixel zastupuje jeden bit. Najskôr sa z čiernobieleho obrázku reprezentovaného triedou `Image` modulu `PIL` získa zoznam hodnôt pixelov mentódou `getdata()`. Ďalej sa vytvorí objekt triedy `Bitstream`, do ktorého je možné postupne vkladať bity a následne tento bitový tok preformátovať na zoznam bajtov. Pre uloženie rozmerov obrázka v bajtoch sa použije metóda `to_bytes()`, ktorá berie ako argumenty požadovanú veľkosť, endianitu a či je hodnota znamienková. Platforma *ESP32* používa *little endian*. Pre uloženie do súborového systému je použitá trieda `ContentFile`.

5.2.4 Správa užívateľov

Pre správu užívateľov sú v rámci frameworku *Django* predimplementované formuláre a iné pomocné triedy/funkcie. Štandardne sa autorizácia implementuje ako oddelená aplikácia (*Django app*) a preto som vytvoril aplikáciu `users_auth`. Pri implementácií pohľadu pre registráciu som vychádzal z ukážky v rámci tutoriálu `calendar-application-tutorial`[2] a pri pouziťí predvoleného prihlasovacieho formulára z tutoriálu `django-auth-tutorial`[3].

V rámci frameworku možno obmedziť prístup k pohľadom pomocou dekorátora funkcie `@login_required`, ktorý berie ako argument adresu URL, kam má užívateľa presmerovať pre prihlásenie. Tiež uloží pôvodnú adresu URL, odkiaľ bol užívateľ presmerovaný, do požiadavku *GET* pod kľúčom `next`.

Registrácia užívateľa je implementovaná v rámci pohľadu `signup_view()` v module `views`. Použitý je formulár `UserCreationForm` implementovaný v rámci projektu *Django*. Pre vytvorenie odpovede je použitá šablóna `signup.html`. Táto šablóna vypíše formulár. Po prijatí a skontrolovaní dát pre registráciu sa nový užívateľ uloží a prihlási.

Pohľad pre prihlásenie je implementovaný vo funkcií `login_view()`. Použitý je existujúci formulár `AuthenticationForm`. Ak sú zadané údaje vo formulári správne, z formulára sa získa užívateľ metódou `get_user()` a prihlási sa. V šablóne sa predáva stránka, kam sa má po úspešnom prihlásení presmerovať, do formuláru prostredníctvom pola `hidden`. Ak sa po spracovaní prihlasovacieho formulára nachádza hodnota pod kľúčom next v slovníku `POST`, užívateľ tam bude presmerovaný. Inak bude presmerovaný na hlavnú stránku.

[2] https://github.com/TheDumbfounds/calendar-application-tutorial/blob/master/calendarproject/myapp/views.py

[3] https://github.com/wsvincent/django-auth-tutorial/blob/master/config/urls.py

V rámci pohľadu `logout_view()` bude užívateľ odhlásený a presmerovaný na hlavnú stránku.

5.3 Nastavenie parametrov zobrazovacej jednotky

Pre komunikáciu so serverovou aplikáciou je najskôr potrebné predať platforme *ESP32* parametre, pomocou ktorých bude môcť s ňou začať komunikovať. Spôsob predania týchto parametrov je navrhnutý v 4.4.

Implementácia v rámci platformy *ESP32*

V rámci platformy je pre prečítanie a uloženie parametrov komunikácie potrebné použiť triedy pre sériovú komunikáciu, čítanie a zápis do pamäte EEPROM, pre nastavenie časovača. Pre sériovú komunikáciu sa v prostredí *Arduino* používa trieda `Serial`. Pred čítaním a zápisom je najskôr potrebné zavolať metódu `begin()`, berie ako parameter hodnotu `baudrate`. Pre odosielanie textových reťazcov sú použité metódy `print()` a `println()`. Pre čítanie je použitá metóda `ReadStringUntil()`, ktorá berie ako parameter koncový znak, po ktorý má čítať. Táto metóda je blokujúca, ukončí sa po prečítaní koncového znaku, prípadne po uplynutí času zadaného metódou `setTimeout()`. Funkcia ďalej vráti prečítaný reťazec v triede `String`, bez koncového znaku. Či sa metóda ukončila prečítaním koncoveho znaku alebo uplynutím časového limitu, nemožno detekovať. Preto bolo je pri implementácií získavania parametrov nutné použiť časovač s generovaním prerušenia. V rámci platformy *ESP32* je implementovaných niekoľko časovačov. Najskôr je potrebné nastaviť alarm časovača. To je možné urobiť vhodnými hodnotami *prescale* a hodnoty, kedy má časovač pustiť alarm. Ďalej je potrebné časovaču predať ukazateľ na funkciu, ktorá obslúži prerušenie a prerušenie povoliť. Parametre komunikácie sú uložené v štruktúre:

```
struct {
    char wifiname[EEPROM_STR_SIZE];
    char wifipasswd[EEPROM_STR_SIZE];
    char serverip[EEPROM_STR_SIZE];
    uint16_t serverport;
} comm_params;
```

Výpis 5.5: Štruktúra pre uloženie parametrov potrebných pre komunikáciu so serverovou aplikáciou.

Pre čítanie a zápis do pamäte EEPROM je potrebné najskôr nainicializovať triedu `EEPROMClass` pomocou metódy `begin()`, ktorá berie veľkosť bloku ako parameter. Následne je možné čítať a zapisovať do pamäte pomocou metód `readBytes()` a `writeBytes()`. Tieto metódy berú ako argumenty počet prečítaných/zapísaných bajtov, ukazateľ na typ `void`, kam majú tieto bajty zapísať/prečítať. Pri funkcií `writeBytes()` sú dáta uložené len vo vyrovnávacej pamäti, do nevolatilnej pamäte sa zapíšu až po volaní metódy `commit()`. Po reštarte tieto parametre nahrajú do pamäte modulu RTC, kde ostanú uložené aj v režime hlbokého spánku. Pre zápis do pamäte modulu RTC stačí definovať globálnu premennú a označiť ju atribútom `RTC_DATA_ATTR`. Takáto premenná bude uložená v pamäti *Slow* modulu RTC.

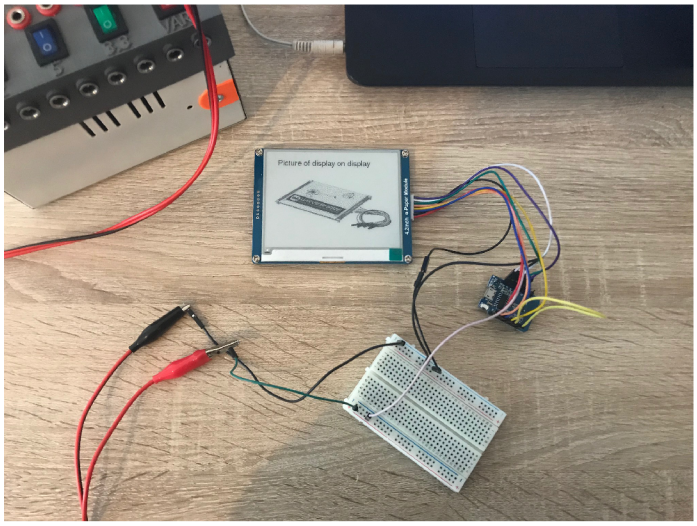
Skript pre nahranie konfigurácie do vizualizačnej jednotky

Pre komunikáciu s plaformou je implementovaný skript `set_comm_properties.py`. V rámci skriptu je najskôr je potrebné nahrať argumenty, pre tento účel bol použitý modul `argparse`. Ďalej sú skontrolované hodnoty hodnoty zadané v argumentoch, pre kontrolu validity IP adresy je použitá funkcia modulu `socket`. Následne sa skript pokúsi pripojiť na sériový port použítím modulu `serial`. Pre začatie komunikácie je potrebné vytvoriť novú inštaciu triedy `Serial`, ako argument je potrebné zadať meno portu (*device name*) a ďalšie parametre komunikácie – *baud rate*, počet stop bitov, veľkosť bajtu v bitoch, a hodnotu *timeout* v sekundách. Následne je použítá metóda `write()` pre odosielanie a metóda `readln()` pre čítanie. Textové reťazce je pri odoslaní potrebné prekódovať do kódovania ASCII pomocou funkcie `bytes()`, prijatý reťazec sa dekóduje pommocou metódy `decode()`.

Kapitola 6

Testovanie a zhodnotenie energetickej náročnosti

V rámci práce bola vytvorená zobrazovacia jednotka, ktorá sa skladá z platformy *ESP32* a displeja vyrobeného firmou *Waveshare*. Tiež bola vytvorená webová aplikácia pre jej správu. Na 6.1 obrázku možno vidieť zapojenie displeja na platformu. Vizualizačná jednotka bola pri testovaní a vykonávaní meraní napájaná pomocou laboratórneho zdroja napätím 5 V. Pre účely testovania som tiež nasadil aplikáciu pre správu pomocou Webového servera *NGINX*[1]. Návod na nasadenie webovej aplikácie sa nachádza v rámci odovzdaných zdrojových súborov. Testovanie systému bolo vykonané ručne. Testovanie prebiehalo s použitím mikrokotroléru *ESP-WROOM-32* aj *ESP8266EX*. Otestovaná bola inicializácia zobrazovacej jednotky aj aktualizácia obsahu. Pri aktualizácií boli otestované všetky možnosti rozloženia obsahu, reakcie na chybové návratové kódy, neexistujúciu inštanciu jednotky v rámci aplikácie pre správu, reakcie na nedostupný server a prístupový bod WiFi.



Obr. 6.1: Fotka zobrazujúca zapojenie vizualizačnej jednotky pri testovaní. Pre meranie bol následne do série pripájaný multimeter.

[1] https://www.nginx.com/resources/wiki/

Po implementácií a otestovaní som odmeral hodnoty prúdu v rôznych fázach behu zobrazovacej jednotky. Po prvom meraní som následne aplikoval niektoré zmeny v riešení 6.1, ktoré viedli k optimalizácií jeho energetickej náročnosti. Následne som vykonal druhé meranie, pri čom som použil riešenie, ktoré tieto zmeny už zahŕňa. Výsledky tohto merania sú popísané v 6.2. V závere som spomenul aj ďalšie zmeny, ktoré by mohli viesť k optimalizácií spotreby 6.3.

6.1 Optimalizácia energetickej náročnosti

Po prvých meraniach som sa pokúsil znížiť spotrebu jednotky v aktívnom režime znížením taktovacej frekvencie procesora a obmedzením času, po ktorý sú aktívne periférne moduly. Ďalej, ako je spomenuté v 6.1.3, pre zníženie spotreby jednotky v režime spánku som použil inú vývojovú dosku, založenú na platforme *ESP8266*.

6.1.1 Taktovacia frekvencia

Veľký vplyv na spotrebu platformy má v aktívnom režime taktovacia frekvencia procesora. Na platorme *ESP32* je možné v aktívnom režime použiť 2 zdroje hodinovej frekvencie, hodiny XTAL a PLL. Hodiny XTAL podporujú taktovaciu frekvenciu 26 MHz a 40 MHz, hodiny PLL 80 MHz, 160 MHz alebo 240 MHz. Pri použití modulu pre rádiofrekvenčný prenos je ale nutné použiť hodiny PLL.

Obrázok 6.2 zobrazuje spotrebu pre rôzne hodnoty taktovacej frekvencie pri použití hodín PLL. Tak ako je spomenuté v 3.2.2, pri zohľadnení metriky *spotreba energie za jednu operáciu* sa odporúča využitie maximálnej možnej taktovacej frekvencie. Pri tejto aplikácií sa však strávi väčšina času aktívnym čakaním na vstupno-výstupné operácie – dátový prenos pomocou modulom WiFi, alebo čakanie na zobrazenie dát na displeji, ktoré trvá až 4 sekundy. S ohľadom na túto skutočnosť som zvolil frekvenciu 80 MHz. Čas aplikácie strávený v aktívnom režime platformy sa výrazne nezmenil.

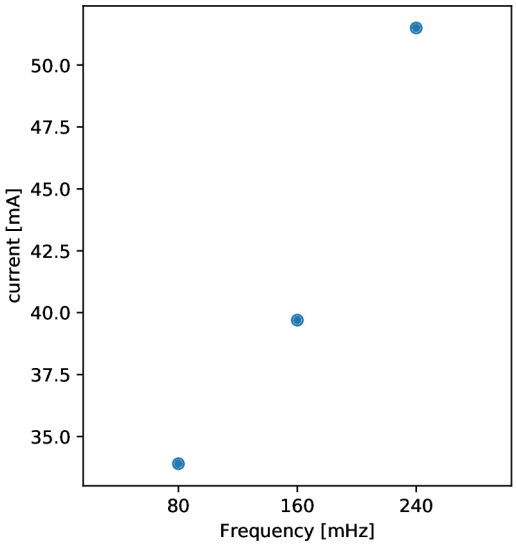
V prípade platformy *ESP8266* možno taktovaciu frekvenciu nastaviť na 80 MHz alebo 160 MHz. Podobne, ako na platforme *ESP32*, som taktovaciu frekvenciu nastavil na 80 MHz.

6.1.2 Obmedzenie času, po ktorý su periférne moduly aktívne

V pôvodnom riešení prebiehalo zobrazenie aktuálnych dát tak, že hneď po stiahnutí aktualizácie boli dáta spracované a zobrazené na displej. Teda počas tohto procesu bol po celý čas spustený aj modul WiFi, aj displej v aktívnom režime. Odoberaný prúd sa pohyboval po celý čas okolo 118 mA. V novom riešení sa najskôr stiahnu dáta, vypne modul WiFi, následne sa dáta spracujú a zobrazia na displeji.

6.1.3 Použitie vývojovej dosky s nižšou spotrebou v režime spánku

Vizualizačná jednotka bola navrhnutá pre vývojovú dosku *ESP32 devkitc v4*. Dosku možno napájať buď privedením na 3,3 V, prípadne 5 V napätia na odpovedajúci pin. Ďalej je možné napájať pomocou *Micro USB* portu, kde sa tiež pre napájanie používa 5 V. V prípade vstupného napätia 5 V je použitý regulátor na 3,3 V. Okrem toho doska obsahuje aj prevodník na UART a svietivú LED diódu pre signalizovanie napájania. Všetky tieto komponenty môžu zvýšiť celkovú spotrebu vývojovej dosky. Zvýšená spotreba sa potom prejaví najmä v režime spánku, kedy samotný čip spotrebuje len minimálne množstvo energie. Po implementácií a následných meraniach som zistil, že celková spotreba vývojovej dosky v režime



Obr. 6.2: Závislosť taktovacej frekvencie a vstupného prúdu využívanom platformou *ESP32* v aktívnom režime. Pri meraní nebol používaný modul WiFi.

hlbokého spánku je až 8,1 mA. V prípade použitia batérie s kapacitou 3000 mAh by vizualizačná jednotka dokázala fungovať len okolo 15 dní. Túto vývojovú dosku možno nahradiť inou, prípadne je možné vytvoriť vlastnú dosku obsahujúcu čip *ESP-WROOM-32*.

S ohľadom na aktuálnu dostupnosť som použil dosku *WEMOS D1 mini*, ktorá obsahuje čip *ESP8266EX*. Referenčná spotreba čipu je spomenutá v 3.3.2, potrebné úpravy kódu vizualizačnej jednotky sú popísané v 5.1.3.

6.2 Zhodnotenie spotreby vizualizačnej jednotky

V tejto kapitole je popísaná spotreba jednotky s použitím dosky *WEMOS D1 mini*. Základné údaje o spotrebe jednotky s použitím vývojovej dosky *ESP32 devkitc v4* sú potom zhrnuté v 6.2.5.

Pre meranie aktuálneho prúdu je použitý multimeter zapojený do série. V prípade tohto spôsobu merania sa dopúšťam istej chyby merania, hodnota prúdu odoberaná platformou totiž kolíše a multimeter zobrazí len aproximovanú hodnotu v istom časovom rámci tak, ako je to spomenuté v 3.2.3. Multimeter ďalej zobrazuje tiež niekoľko hodnôt, ja som si vybral pre výpočet spotreby tú najvyššiu v danom časovom úseku (*worst case*). Pre určenie aktuálneho príkonu je potom získaná hodnota prúdu vynásobená vstupným napätím.

6.2.1 Spotreba platformy *ESP8266* v aktívnom režime

Platforma komunikuje s webovou aplikáciou pomocou modulu WiFi, ktorý sa nachádza na platforme, konkrétne v rámci čipu *ESP8266EX*. Platforma ovláda obsah displeja pomocou rozhrania SPI a podporuje rôzne prevádzkové režimy, z nich sa využíva režim hlbokého spánku a aktívny režim, v ktorom je aktívny aj modul pre rádiofrekvenčný prenos. V ak-

tívnom režime ovplyvňujú najviac spotrebu bezdrôtové prenosy, prúd WiFi môže vystúpiť až na 170 mA pri odosielaní dát, 56 mA pri prijímaní.

Vývojovú dosku možno napájať pomocou *Micro USB* portu alebo pomocou 5 V pinu. V rámci testovania som dosku napájal pomocou pinu, čím sa spotreba znížila. Platforma bude komunikovať s webovou aplikáciou, ktorá beží na serveri pripojenom v lokálnej sieti. Pri použití servera mimo lokálnej siete treba počítať s dlhšou dobou potrebnou pre aktualizáciu, teda aj dlhší čas zotrvania platformy v aktívnom režime s aktivnym modulom WiFi. Merania neboli vykonávané pre proces získania údajov potrebných na pripojenie k webovej aplikácií od užívateľa. Jedná sa totiž o jednorázovú procedúru, ktorá sa nevykovanáva periodicky. Pri aktualizácií obsahu v prípade úspešného získania údajov o aktuálnej verzií obsahu z webového servera možu nastať dve situácie:

- obsah na displeji sa zmenil a je potrebné tento obsah stiahnuť a zobraziť,
- obsah na displeji sa nezmenil od poslednej kontroly aktualizácie.

Keďže sa predpokladá, že displej bude zobrazovať statický obsah, častejšie bude nastávať druhá možnosť. Pre obe možnosti je spotreba odlišná a tak bude popísaná v nasledujúcich častiach.

Spotreba pri aktualizácií, kedy sa obsah zmení

V prípade, že sa stiahne a zobrazí nový obsah, trvá aktualizácia približne 11 sekúnd. Na platforme sa najskôr aktivuje WiFi modul, pripojí sa k prístupovému bodu a pošlú sa 3 požiadavky *GET* – v prvom sa zistí, či je potrebná aktualizácia a v ďalších dvoch sa získa aktuálny obsah, a následne sa deaktivuje modul WiFi. Počas tejto procedúry zobrazoval multimeter hodnotu prúdu približne 71,9 mA, táto hodnota prúdu bola stabilná počas 7 sekúnd. Pri odosielaní dát môže byť hodnota prúdu nárazovo až 170 mA.

Po prijatí dát sa spracuje sa prijatý obsah a zobrazí sa na displeji. Potom sa vykoná úplná aktualizácia displeja (*full refresh*) a displej sa zase uspí. Multimeter tak ukazoval hodnotu prúdu okolo 21,8 mA, približne 4 sekundy.

Spotreba pri aktualizácií, kedy sa obsah nezmenil

V prípade, že sa nestiahne nový obsah, zotrvá platforma v aktívnom režime približne 5 sekúnd. Na platforme sa aktivuje modul WiFi, pripojí sa k prístupovému bodu a zašle požiadavok *GET*. Nameraná hodnota prúdu je porovnateľná s prvou časťou aktualizácie v prípade, že sa stiahne nový obsah.

6.2.2 Spotreba platformy *ESP8266* v režime hlbokého spánku

Podľa *datasheet* by mal čip *ESP8266EX* v režime hlbokého spánku využívať prúd približne 20 µA. *Datasheet* [17] uvádza tento prúd pri napätí 2,5 V, pri čom samotný čip požaduje napätie 2,5 – 3,6 V. Nameraná hodnota prúdu je 161,9 µA. V riešení je použítá vývojová doska *WEMOS D1 mini*, ktorá pre najájanie požaduje napätie 5 V, takže vyššia hodnota prúdu môže byť spôsobená napríklad regulátorom napätia. V prípade použitia vývojovej dosky *ESP32 devkitc v4* bol aktuálny prúd v režime hlbokého spánku 8,1 mA.

6.2.3 Spotreba displeja

Pre meranie je použitý 4,2" displej od spoločnosti *Waveshare*. Súčasťou použitého modulu s displejom je tiež ovládač (*driver board*), ktorý komunikuje s platformou prostredníctvom rozhrania SPI. Pre napájanie displeja má byť použité napätie 3,3 V. Displej sa väčšinu času bude nachádzať v režime spánku. Ako je spomenuté v 2.3.1, podľa *datasheet* má mať v tomto režime príkon približne 16,5 µW. Pre meranie skutočnej spotreby bol pripojený multimeter do série s modulom displeja. V režime spánku displeja bol nameraný prúd v rozsahu 7 až 12 µA. Po vynásobení vstupným napätím dostávame príkon 23,6 až 39,6 µW. Mierne vyššia hodnota príkonu môže byť daná použitím vstavaného ovládača displeja.

V prípade, že je potrebné zobraziť nové dáta, bude displej uvedený do aktívneho režimu. Počas merania displej zotrval v aktívnom režime približne 7 sekúnd. V tomto režime by mal mať podľa *datasheet* príkon typicky 26,4 mW, maximálne však 40 mW. Pri meraní bola nameraná hodnota prúdu v nárazoch maximálne 7,43 mA, čo odpovedá príkonu približne 24,5 mW.

6.2.4 Odhad doby behu pri napájaní pomocou batérie

Na základe hodnôt nameraných v predošlých častiach podkapitoly možno odhadnúť, ako dlho by mohla vizualizačná jednotka fungovať v prípade, že by bola napájaná zdrojom s dopredu danou kapacitou. Pre výpočet som zvolil kapacitu batérie 3000 mAh. Pri výpočtoch sa bude ďalej počítať s tým, že si vizualizačná jednotka pri každej aktualizácií stiahne nové dáta a zobrazí ich na displeji 6.2.1. Pri výpočte najskôr určím, koľko elektrickej energie v [mWh] sa spotrebuje pri aktualizácií. Ďalej je potrebné určiť požadovaný príkon v režime spánku. Po dosadení periódy aktualizácie, ktorú zadáva užívateľ, bude potom možné vypočítať spotrebovanú energiu vizualizačnou jednotkou v rámci jednej aktualizácie a čakania v režime spánku (*duty cycle*). Potom bude možné spočítať počet cyklov, ktoré bude možné vykonať v prípade, že sa bude jednotka napájať pomocou batérie.

Spotrebovaná energia pri aktualizácií

Spotrebovaná energia súvisí s prácou, ktorú je potrebné vykonať pri prenose elektrického náboja. Elektrickú prácu možno vyjadriť ako súčin príkonu a času, po ktorý je tento stály príkon dodávaný:

$$W = P * t[Wh]$$

Počas aktualizácie spotrebuje energiu platforma aj pripojený displej. Výslednú potrebnú elektrickú prácu možno vyjadriť ako ich súčet:

$$\begin{aligned} W_{jednotka} &= W_{platforma} + W_{displej} \\ W_{jednotka} &= W_{platforma} + P_{displej} * t_{zobrazenie} \end{aligned}$$

Proces aktualizácie na platforme sa skladá z dvoch fáz – stiahnutie aktuálnych dát a ich zobrazenie. Výslednú vykonanú prácu možno vyjadriť súčtom elektrickej práce v týchto dvoch fázach:

$$\begin{aligned} W_{platforma} &= W_{stiahnutie} + W_{zobrazenie} \\ W_{platforma} &= P_{stiahnutie} * t_{stiahnutie} + P_{zobrazenie} * t_{zobrazenie} \end{aligned}$$

Po dosadení hodnôt získam výslednú elektrickú prácu, ktorú je potrebné vykonať pre aktualizáciu obsahu displeja. Táto hodnota je približne 0,89792 mWh.

Potrebný príkon vizualizačnou jednotkou v režime spánku

Príkon potrebný pre fungovanie jednotky v režime spánku sa určí, ako súčet potrebného príkonu pre displej a dosku:

$$P_{jednotky} = P_{platforma} + P_{displej}$$

Po dosadení vyšla hodnota príkonu 0,8491 mW.

Výpočet približnej doby behu

Tým, že má displej zobrazovať statické dáta, počítalo sa s pomerne veľkými odstupmi medzi aktualizáciami obsahu. Preto sa počíta s periódou aktualizácie napríklad 30 minút. V takom prípade by vizualizačná jednotka napájaná 3000 mAh batériou vydržal fungovať približne 241 dní. Pri reálnej aplikácií je ďalej potrebné pri určení približnej doby behu zohľadniť aj iné parametre zdroja energie:

- Čip *ESP8266EX* tiež vyžaduje prúd v nárazoch až 170 mA, niektoré batériové zdroje však nemusia byť schopné poskytnúť toto napätie počas celej ich životnosti.
- Pri vybíjaní batérie sa tiež znižuje napätie, ktoré poskytujú.
- Napätie poskytované batériou sa mení s teplotou.
- Batérie sa tiež samovoľne vybíjajú v čase.

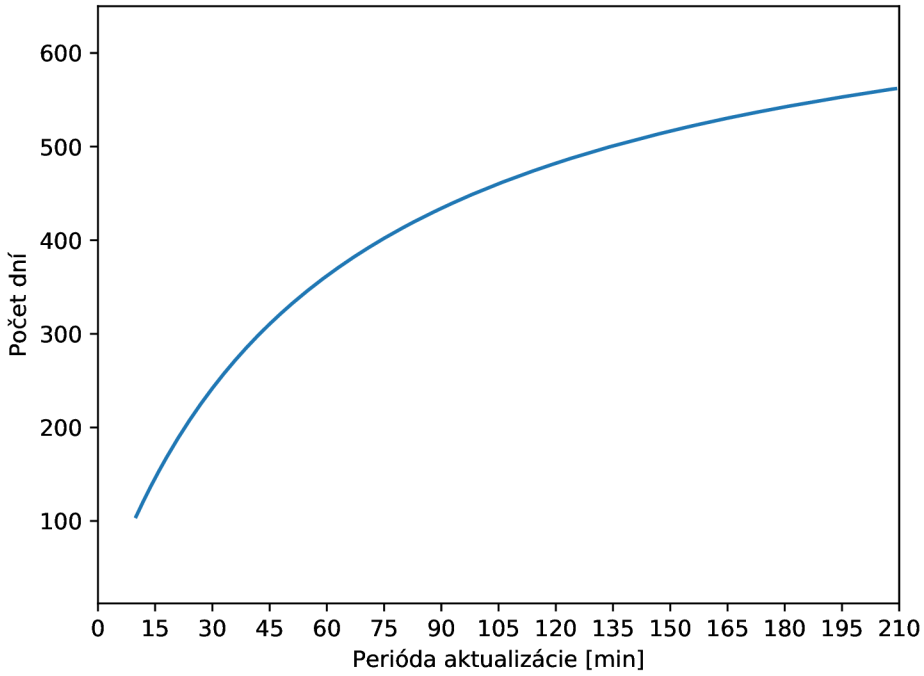
V predošlom odstavci sa počítalo s periódou aktualizácie 30 minút. Užívateľ môže túto hodnotu ďalej meniť pomocou webovej aplikácie pre správu jednotiek a tak je vhodné porovnať, ako sa mení doba behu pri rôznych periódach pri napájaní batériou s rovnakou kapacitou. Graf 6.3 zobrazuje dobu behu pri rôznych hodnotách periódy.

6.2.5 Spotreba platformy *ESP32*

Merania boli vykonané aj v prípade, že vo vizualizačnom paneli bola použitá doska *ESP32 devkitc v4*. V tejto časti sú popísané výsledky meraní po aplikovaní úprav vedúcich k zníženiu spotreby spomenutých v 6.1.2 a 6.1.1. Pracovný cyklus s použitím platformy ESP32 možno podobne ako v prípade platformy ESP8266 rozdeliť na:

- aktívnu fázu, počas ktorej je spustený modul WiFi a sťahujú sa aktuálne dáta,
- fázu, v ktorej je aktívny displej a prijaté dáta sa na ňom zobrazujú,
- fázu, kedy je displej aj platforma v režime hlbokého spánku.

Merania boli vykonané spôsobom, ako je to popísané v úvode tejto podkapitoly. Vývojová doska bola pri meraniach napájaná pomocou Micro USB portu. Prvá fáza, v ktorej sa stiahli aktuálne dáta, trvala približne 4 sekundy, nameraný prúd bol okolo 150 mA. Spracovanie a zobrazenie stiahnutých dát trvalo približne 7 sekúnd a hodnota prúdu bola približne 33 mA. V režime spánku bola nameraná hodnota prúdu až 8.1 mA. Možné odôvodnenie tejto vysokej hodnoty je popísané v 6.1.3. V prípade, že by vizualizačná jednotka aktualizovala svoj obsah každých 30 minút, vydržala fungovať 14 dní a 14 hodín pri použití 3000 mAh batérie pre napájanie. Na obrázku 6.4 je zobrazený výpočet približnej doby pre rôznu periódu aktualizácie.



Obr. 6.3: Závislosť periódy aktualizácie a počtu dní, koľko dokáže vizualizačná jednotka fungovať pri použití dosky *Wemos D1 mini* a napájaní batériou s kapacitou 3000 mAh.

6.3 Ďalšie optimalizácie spotreby

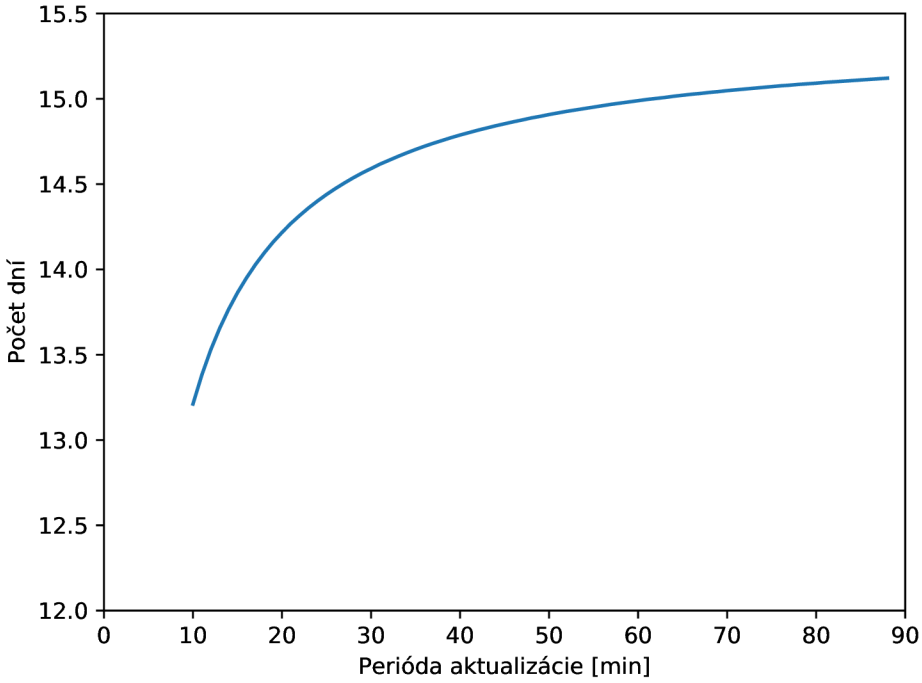
V tejto časti sú spomenuté možnosti, ktorými by bolo možné ďalej optimalizovať spotrebu vizualizačnej jednotky.

6.3.1 Zníženie počtu synchrónnych správ vymenených so serverom

V aktívnom režime sa pomerne veľká časť energie spotrebuje pre komunikáciu s serverom prostredníctvom modulu WiFi. Tak ako je to zobrazené na grafe komunikácie 4.3, platforma pre získanie dát na zobrazenie zašle 2 požiadavky *GET*, jeden pre získanie textového obsahu, druhý pre získanie grafického. V prípade, že by sa tieto dáta zlúčili do jedného požiadavku, skrátil by sa na platforme čas, po ktorý by bol aktívny modul pre komunikáciu WiFi, čo by viedlo k zníženiu spotreby. Rozdiel v spotrebovanej energii by sa potom prejavil výraznejšie v prípade, ak by sa webový server nenachádzal v lokálnej sieti a čas potrebný pre získanie odpovede by sa predĺžil.

6.3.2 Napájanie bez použitia prevodníka napätia

V rámci testovania sa vývojová doska napájala napätím 5 V. V prípade, že by bolo možné dosku napájať použitím napätia 3,3 V bez použitia regulátora napätia, spotreba by sa znížila.



Obr. 6.4: Závislosť periódy aktualizácie a počtu dní, koľko dokáže vizualizačná jednotka fungovať pri použití dosky *ESP32 devkitc v4* a napájaní batériou s kapacitou 3000 mAh.

Kapitola 7

Záver

Cieľom práce bolo navrhnúť vzdialene riadený vizualizačný panel s použitím platformy *ESP32* a displeja založeného na technológií *E-ink*. Ďalej bolo potrebné navrhnúť aplikáciu pre jeho správu a implementovať firmware pre platformu *ESP32* spolu s aplikáciou. Pri implementácií bol použitý *E-ink* displej od firmy *Waveshare*, ktorý obsahuje ovládač s rozhraním SPI a pomocou tohto rozhrania je ovládaný platformou. Plaforma komunikuje s aplikáciou pre správu bezdrôtovo pomocou technológie WiFi. Vizualizačná jednotka zobrazuje obrázok aj textové bloky. Pre správu bola navrhnutá a implementovaná webová aplikácia pomocou frameworku *Django*. Aplikácia komunikuje so zobrazovacou jednotkou pomocou protokolu HTTP. Aplikácia umožňuje správu viacerých vizualizačných jednotiek, pri čom je možné zdielať obsah medzi týmito jednotkami. Každá jednotka je identifikovaná pomocou MAC adresy použitej platformy.

Po implementácií boli vykonané experimenty pre určenie energetickej náročnosti vizualizačnej jednotky. Na základe experimentov bol popísaný pracovný cyklus vizualizačnej jednotky a spotreba v jeho jednotlivých fázach. Ďalej boli definované spôsoby, akými by bolo možné znížiť energetickú náročnosť jednotky. Niektoré z nich boli následne aplikované. Pri implementácií bola použitá vývojová doska *ESP32 devkitc v4*. Problematickou bola jej vysoká spotreba v režime hlbokého spánku, kvôli ktorej by bolo problematické napájanie pomocou batérií. Preto bola s ohľadom na aktuálnu dostupnosť vývojová doska nahradená za dosku *Wemos D1 mini*, ktorá je založená na platforme *ESP8266*. Z tohto dôvodu musel byť firmware pre platformu upravený, dostupné sú verzie pre obe platformy. Spotrebu v režime spánku sa týmto podarilo znížiť.

Pre nasadenie vizualizačného panela bude ďalej potrebné navrhnúť a implementovať spôsob napájania. Pre ďalšie zníženie spotreby jednotky bude vhodné navrhnúť vlastnú dosku založenú na platforme *ESP32*, prípadne *ESP8266*. V rámci aplikácie pre správu je priestor na zlepšenie jej užívateľského rozhrania a zabezpečenia.

Literatúra

[1] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*. March 2012, s. 1–2793. Dostupné z: https://ieeexplore.ieee.org/servlet/opac?punumber=6178209.

[2] *E-paper displays in consumer electronics* [online]. Júl 2014 [cit. 2020-1-7]. Dostupné z: https://circuitdigest.com/microcontroller-projects/programming-esp32-with-arduino-ide.

[3] MDN web docs. *Django introduction* [online]. November 2019 [cit. 2020-4-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

[4] CMicrotek. Low Power Design. 2013. Dostupné z: http://www.cmicrotek.com/LPD_book.pdf.

[5] Crystalfontz. *Graphic LCD Module Datasheet CFAG160128B-TMI-TZ*. September 2017. Dostupné z: https://www.crystalfontz.com/products/document/360/CFAG160128E-TMI-TZDatasheetReleaseDate2017-09-27.pdf.

[6] Datta, A. a Munshi, S. *Information Photonics Fundamentals, Technologies, and Applications*. CRC Press, november 2016. 203-212 s. ISBN 9781315373072.

[7] Django Software Foundation. *Django Documentation*. Revízia 3.0. Apríl 2020. Dostupné z: https://buildmedia.readthedocs.org/media/pdf/django/3.0.x/django.pdf.

[8] EastRising Technology. *ER-OLEDM055-1 Series OLED Display Module Datasheet*. December 2019. Dostupné z: https://www.waveshare.com/w/upload/6/6a/4.2inch-e-paper-specification.pdf.

[9] Espressif Systems. IwIP. *ESP-IDF programming guide* [online]. November 2019 [cit. 2020-1-6]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v4.0-beta2/api-guides/wifi.html.

[10] Espressif Systems. Wi-Fi. *ESP-IDF programming guide* [online]. November 2019 [cit. 2020-1-6]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/protocols/esp_http_client.html.

[11] Espressif Systems. IwIP. *ESP-IDF programming guide* [online]. November 2019 [cit. 2020-1-6]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/lwip.html#socket-error-reason-code.

[12] Espressif Systems. ESP HTTP Client. *ESP-IDF programming guide* [online]. November 2019 [cit. 2020-1-6]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/protocols/esp_http_client.html.

[13] Expressif. *ESP32 Series Datasheet*. Verzia 3.2. Október 2019. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[14] Expressif. *ESP32 Technical Reference Manual*. Verzia 4.1. November 2019. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[15] Expressif. *ESP32-WROOM-32 Datasheet*. Verzia 2.9. September 2019. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[16] Expressif. *ESP8266 Technical Reference*. Verzia 1.4. August 2019. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf.

[17] Expressif. *ESP8266EX Datasheet*. Verzia 6.2. Apríl 2020. Dostupné z: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[18] Guarda, T., Leon, M., Augusto, M. F., Haz, L., de la Cruz, M. et al. Internet of Things challenges. In: *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. June 2017, s. 1–4. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7975936/.

[19] Inoue, S., Kawai, H., Kanbe, S., Saeki, T. a Shimoda, T. High-resolution microencapsulated electrophoretic display (EPD) driven by poly-si TFTs with four-level grayscale. *IEEE Transactions on Electron Devices*. Sep. 2002, roč. 49, č. 9, s. 1532–1539. Dostupné z: https://ieeexplore.ieee.org/document/1027833. ISSN 1557-9646.

[20] Parikh, B. Basics and Working. *Bluetooth protocol* [online]. Jún 2018 [cit. 2020-1-21]. Dostupné z: https://www.engineersgarage.com/article_page/bluetooth-protocol-part-1-basics-and-working/.

[21] Pitt, M. G., Zehner, R. W., Amudson, K. R. a Gates, H. 53.2: Power Consumption of micro-encapsulated Display for Smart Handheld Applications. *SID Symposium Digest of Technical Papers*. 2002, roč. 33, č. 1, s. 1378–1381. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1889/1.1830204.

[22] Raj, A. *Programming ESP32 Board with Arduino IDE* [online]. Október 2018 [cit. 2020-1-2]. Dostupné z: https://circuitdigest.com/microcontroller-projects/programming-esp32-with-arduino-ide.

[23] Rodriguez Fernandez, M., Zalama, E. a González Alonso, I. Review of Display Technologies Focusing on Power Consumption. *Sustainability*. September 2015, roč. 2015, s. 10854–10875. Dostupné z: https://www.researchgate.net/publication/282219145_Review_of_Display_Technologies_Focusing_on_Power_Consumption.

[24] Shah, V. *9 Main Security Challenges for the Future of the Internet Of Things (IoT)* [online]. 2019. Aktualizováno 5. 9. 2019 [cit. 2019-12-30]. Dostupné z: https://readwrite.com/2019/09/05/9-main-security-challenges-for-the-future-of-the-internet-of-things-iot/.

[25] Strba, A. Embedded systems with limited power resources. *Sustainability*. EnOcean GmbH. Január 2009. Dostupné z: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=11&cad=rja&uact=8&ved=2ahUKEwimxO-61vfmAhVlmVwKHXWdBhYQFjAKegQIARAC&url=https%3A%2F%2Fwww.enocean.com%2Ffileadmin%2Fredaktion%2Fpdf%2Fwhite_paper%2Fwp_embedded_systems_en.pdf&usg=AOvVaw3Fwdbd1CJxGHo3e7mWFqy_.

[26] Talin, A. A., Dean, K. A. a Jaskie, J. E. *Solid-State Electronics*. Elsevier, 2001. DOI 10.1016/S0038-1101(00)00279-3. Dostupné z: https://www.researchgate.net/publication/222364608_Field_emission_displays_A_critical_review.

[27] Tanaka, K. *Embedded Systems: Theory and Design Methodology*. InTech, 2012. 407-430 s. ISBN 978-953-51-0167-3.

[28] Townsend, K. A basic overview of key concepts for BLE. *Introduction to Bluetooth Low Energy* [online]. Marec 2014 [cit. 2020-1-21]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/.

[29] Uckelmann, D., Harrison, M. a Michahelles, F. *Architecting the Internet of Things*. Springer-Verlag Berlin Heidelberg, 2011. 1-24 s. ISBN 978-3-642-19156-5.

[30] Wang, Q. H., Setlur, A. A., Lauerhaas, J. M., Dai, J. Y., Seelig, E. W. et al. A nanotube-based field-emission flat panel display. *Applied Physics Letters*. 1998, roč. 72, č. 22, s. 2912–2913. Dostupné z: https://doi.org/10.1063/1.121493.

[31] Waveshare Electronics. *7.5 inch e-paper b Specification* [online]. Júl 2015 [cit. 2019-12-31]. Dostupné z: https://www.waveshare.com/wiki/7.5inch_e-Paper_HAT_(B).

[32] Waveshare electronics. *SPECIFICATION EPD 4.2 inch e-paper*. Marec 2017. Dostupné z: https://www.waveshare.com/w/upload/6/6a/4.2inch-e-paper-specification.pdf.

[33] Xia, F., Yang, L. T., Wang, L. a Vinel, A. Internet of things. *International Journal of Communication Systems*. 2012, roč. 25, č. 9, s. 1101. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.2417.

[34] Yu Chen Hsu, C.-M. W. *E-ink display panel* [online]. Google Patents, február 2008 [cit. 2019-12-31]. United States patent US20080043317A1. Dostupné z: https://patents.google.com/patent/US20080043317A1.

Príloha A

Obsah odovzdaného pamäťového média

Po rozbalení odovzdaného archívu sa v koreňovom adresári nachádzajú tieto súbory:

- podadresár `/management_app/` obsahuje zdrojové texty projektu a súbor obsahujúci databázu,
- podadresár `/visualization_unit/` obsahuje zdrojové texty (*Arduino sketch*) pre vizualizačné jednotky a skripty pre komunikáciu s týmito jednotkami,
- podadresár `/technical_report/` obsahuje zdrojové texty technickej správy,
- `/xpetru15_bp_text.pdf` je výsledný dokument obsahujúci technickú správu,
- súbory obsahujúce licencie použitých knižníc tretích strán.

V adresári `/management_app/` sa nachádza podadresár obsahujúci samotný projekt frameworku *Django* `screen_management_server` a návod `deploy_instructions.txt` pre nasadenie aplikácie v prostredí *Ubuntu / Debian*.

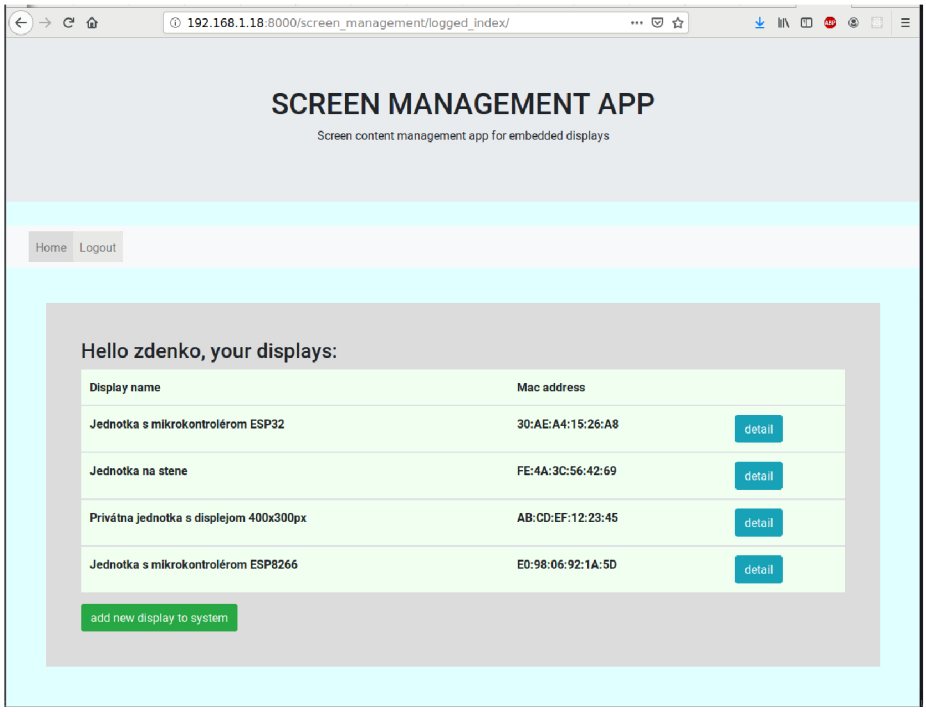
V adresári `/visualization_unit/` sa nachádzajú podadresáre so zdrojovými textami pre jednotlivé platformy `panel_platform_esp32` a `panel_platform_esp8266`. V jednotlivých podadresároch sa nachádzajú aj výsledné binárne súbory pre nahranie na konkrétnu vývojovú dosku. Okrem toho sa v tomto adresári nachádza aj súbor `readme` a podadresár `serial_comm_scripts`. Súbor `readme` obsahuje inštrukcie pre použitie zdrojových súborov v prostredí *Arduino IDE* spolu so zoznamom potrebných knižníc.

V adresári `serial_comm_scripts` sa nachádzajú skripty pre prácu s vizualiačnou jednotkou `set_comm_params.py`, `read_uart_outputs.py`, súbor `requirements.txt` a súbor `readme`, ktorý obsahuje inštrukcie k použitiu skriptov. Skript `set_comm_params.py` slúži pre nainicializovanie vizualizačnej jednotky, `set_comm_params.py` pre zobrazovanie logovacích výpisov. Parametre skriptov možno nájsť v súbor readme, prípadne pomocou prepínača `--help`.

Príloha B

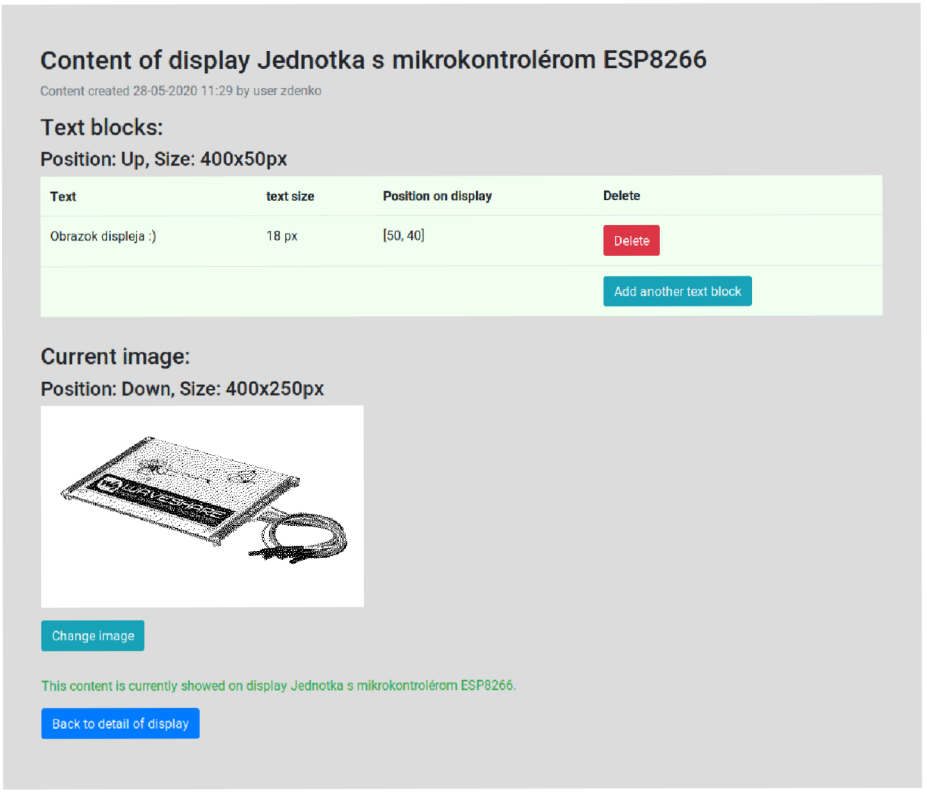
Ukážky užívateľského rozhrania aplikácie pre správu

Táto príloha obsahuje ukážky (*screenshots*) užívateľského rozhrania aplikácie pre správu vizualizačných jednotiek. Ukážka B.1 zobrazuje úvodnú obrazovku. Na ukážke B.2 sú zobrazené detaily displeja, na ukážke B.3 sú zobrazené detaily daného obsahu a na ukážke B.4 zoznam existujúcich obsahov vhodných pre danú vizualizačnú jednotku.

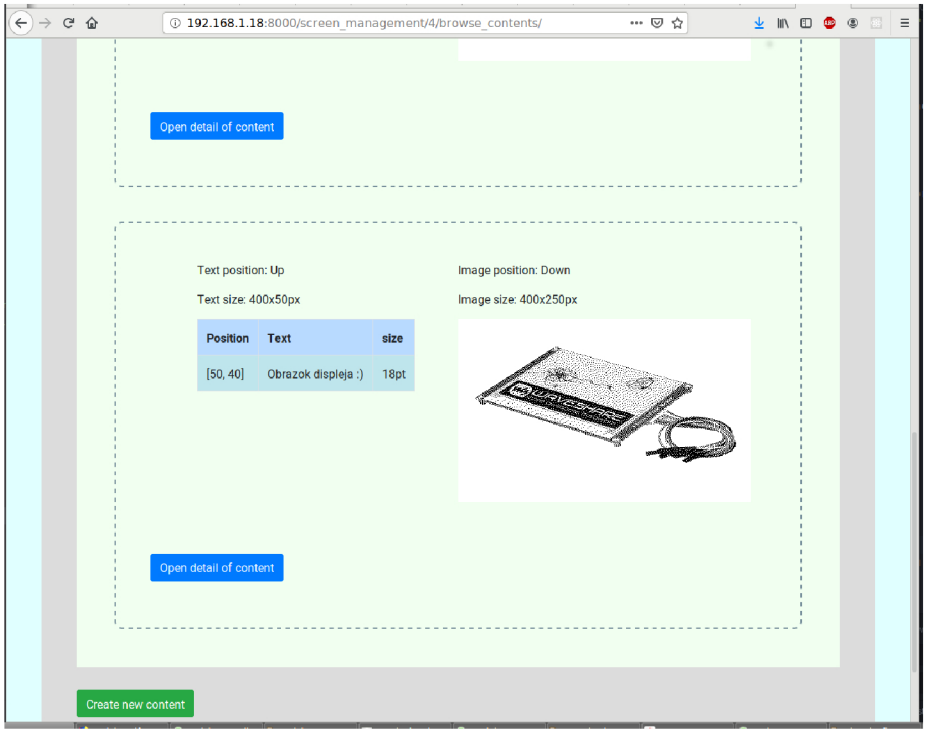


Obr. B.1: Ukážka úvodnej obrazovky obsahujúcej zoznam vizualizačných jednotiek, ku ktorým má užívateľ prístup.

Obr. B.2: Ukážka obrazovky obsahujúcej údaje o danej vizualizačnej jednotke.



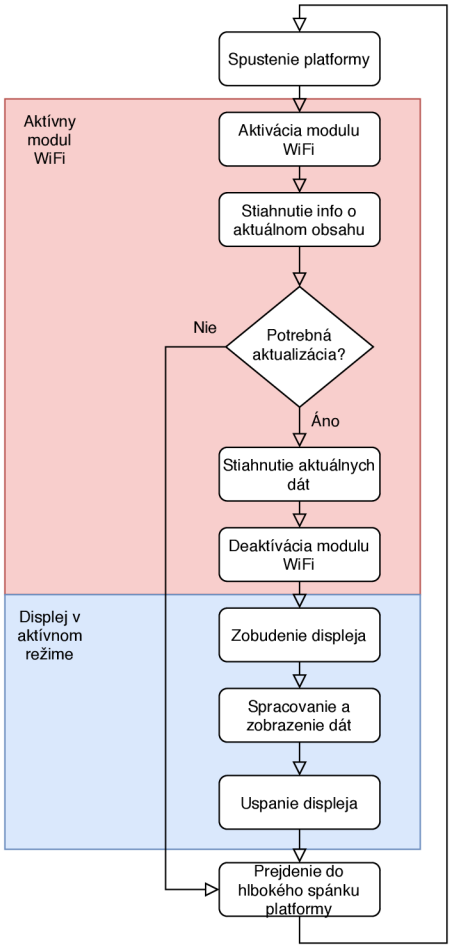
Obr. B.3: Ukážka obrazovky obsahujúcej údaje o obsahu pre vizualizačnú jednotku.



Obr. B.4: Ukážka obrazovky obsahujúcej zoznam existujúcich dostupných obsahov pre vizualizačnú jednotku.

Príloha C

Diagram pracovného cyklu vizualizačnej jednotky



Obr. C.1: Diagram popisujúci pracovný cyklus implementovanej vizualizačnej jednotky. Popis pracovného cyklu sa nachádza v 4.2.