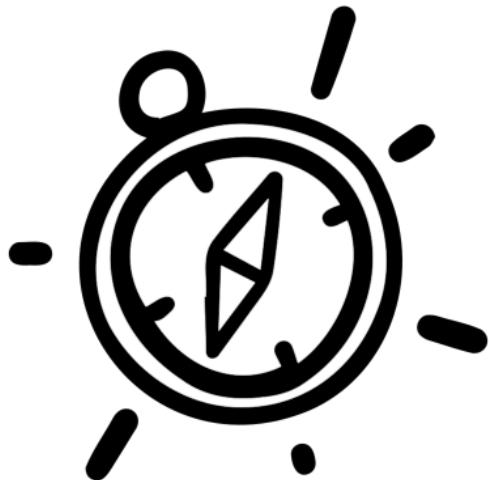


# ZÁVĚREČNÁ STUDIJNÍ PRÁCE

## dokumentace

### LOTR - Location TRacking System



**Autor:** Štěpán Balner  
**Obor:** 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  
se zaměřením na počítačové sítě a programování  
**Třída:** IT4  
**Školní rok:** 2025/26



## **Poděkování**

Děkuji těm, jimž poděkováno zatím nebylo. Také děkuji Současné době za to, že jsou již LLM na takové úrovni, že vytvoření tohoto projektu bylo možné i za sníženého pracovního nasazení.

## **Prohlášení**

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým a prezentačním účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 1. 1. 2026

.....  
Podpis autora

## **Anotace**

Cílem této práce je návrh a realizace sledovacího systému LOTR (LOcation TRacking System), který se skládá z fyzického hardwareového zařízení, serverové infrastruktury a klientské mobilní aplikace.

Hardwareová část je postavena na platformě ESP32 s využitím modemu SIMCOM A7670 pro komunikaci přes síť LTE a modulu GPS pro získávání polohy. Firmware je optimalizován pro nízkou spotřebu energie a spolehlivý přenos dat. Serverová část, realizovaná v prostředí Node.js s databází MySQL, zajišťuje sběr telemetrických dat, správu uživatelů a poskytuje REST API pro komunikaci s koncovými zařízeními. Součástí systému je také nativní aplikace pro OS Android vyvinutá v jazyce Kotlin, která slouží jako plnohodnotná softwarová náhrada hardwareového trackeru.

Výsledkem projektu je prototyp IoT řešení umožňující sledování polohy v reálném čase, historii pohybu a vzdálenou konfiguraci připojených zařízení.

## **Klíčová slova**

GPS, IoT, sledovací systém, ESP32, LTE, Node.js, MySQL, Android, Kotlin, REST API

# Obsah

<b>ÚVOD</b>	<b>3</b>
<b>1 FYZICKÉ ZAŘÍZENÍ „TRACKER“</b>	<b>5</b>
1.1 ÚVOD A KONCEPCE . . . . .	5
1.2 POUŽITÉ TECHNOLOGIE . . . . .	5
1.2.1 MIKROKONTROLÉRY A SoC (ESP32) . . . . .	5
1.2.2 MOBILNÍ KOMUNIKACE V IoT A AT PŘÍKAZY . . . . .	5
1.2.3 GLOBÁLNÍ NAVIGAČNÍ SYSTÉMY (GPS) . . . . .	5
1.2.4 OPERAČNÍ SYSTÉMY REÁLNÉHO ČASU (FREERTOS) . . . . .	6
1.2.5 SOUBOROVÝ SYSTÉM LITTLEFS . . . . .	6
1.3 NÁVRH HARDWARE - VÝBĚR KOMPONENT . . . . .	7
1.3.1 LILYGO T-CALL v1.5 . . . . .	7
1.3.2 MULTI-GNSS L76K MODUL . . . . .	8
1.3.3 BATERIE, TP4056 + MT3608 . . . . .	8
1.3.4 ŘÍZENÍ NAPÁJENÍ A POWER LATCH MODUL . . . . .	9
1.4 IMPLEMENTACE FIRMWARE . . . . .	11
1.4.1 ARCHITEKTURA A POUŽITÉ KNIHOVNY . . . . .	11
1.4.2 PRACOVNÍ CYKLUS . . . . .	12
1.4.3 PARALELNÍ ZPRACOVÁNÍ A OBSLUHA PŘERUŠENÍ (FREERTOS) . . .	12
1.4.4 ŘÍZENÍ NAPÁJENÍ A POWER LATCH MODUL . . . . .	12
1.4.4.1 Režim hlubokého spánku (Deep Sleep) . . . . .	13
1.4.5 ZPRACOVÁNÍ GPS DAT A PARSOVÁNÍ NMEA . . . . .	15
1.4.6 SPRÁVA DATOVÉHO ÚLOŽIŠTĚ (LITTLEFS) . . . . .	15
1.5 KOMUNIKACE A DATA . . . . .	16
1.5.1 KOMUNIKAČNÍ PROTOKOL A ZABEZPEČENÍ . . . . .	16
1.5.2 STRUKTURA DAT A HANDSHAKE . . . . .	16
1.5.3 PRÁCE S MODEMEM A AT PŘÍKAZY . . . . .	18
1.6 KONFIGURACE A SERVISNÍ REŽIM . . . . .	18
1.6.1 OTA REŽIM A WEBOVÉ ROZHRANÍ . . . . .	19
1.6.2 MANUÁLNÍ A VZDÁLENÁ KONFIGURACE ZAŘÍZENÍ . . . . .	20
1.7 SEZNAM SOUČÁSTEK + OBRÁZKY . . . . .	20
<b>2 SERVEROVÁ ČÁST</b>	<b>25</b>
2.1 ÚVOD A KONCEPCE SYSTÉMU . . . . .	25

2.1.1	ROLE SERVERU V SYSTÉMU LOTR . . . . .	25
2.1.2	MONOLITICKÁ ARCHITEKTURA A MVC VZOR . . . . .	25
2.1.3	RELAČNÍ DATABÁZE A ORM (MYSQL + SEQUELIZE) . . . . .	26
2.1.4	PRINCIPY AUTENTIZACE A OAUTH 2.0 . . . . .	26
2.1.5	REST API ARCHITEKTURA . . . . .	26
2.2	NÁVRH A IMPLEMENTACE BACKENDU . . . . .	26
2.2.1	DATABÁZOVÁ VRSTVA (MYSQL A SEQUELIZE) . . . . .	26
2.2.1.1	ORM Sequelize a definice modelů . . . . .	27
2.2.1.2	API pro HW tracker (ESP32) . . . . .	28
2.2.1.3	API pro APK klient (Android) . . . . .	29
2.2.1.4	Validace vstupních dat (Express Validator) . . . . .	30
2.2.1.5	Dokumentace API (Swagger) . . . . .	31
2.2.2	SPRÁVA UŽIVATELŮ A AUTORIZACE . . . . .	32
2.2.2.1	Autorizace a role (authorization.js) . . . . .	32
2.2.2.2	Session a cookie politika . . . . .	32
2.2.2.3	Správa identit (Passport.js) . . . . .	33
2.2.2.4	Hashování hesel (Bcrypt.js) . . . . .	33
2.2.2.5	ROOT účet . . . . .	34
2.2.2.6	E-mailová komunikace (Nodemailer) . . . . .	34
2.3	PREZENTAČNÍ VRSTVA (FRONTEND) . . . . .	34
2.3.1	ŠABLONOVACÍ SYSTÉM EJS A STRUKTURA POHLEDŮ . . . . .	35
2.3.2	VIZUALIZACE DAT A MAPOVÉ PODKLADY . . . . .	35
2.3.2.1	Dynamická aktualizace polohy (AJAX) . . . . .	35
2.4	NASAZENÍ A PROVOZ (DEPLOYMENT) . . . . .	36
2.4.1	DOCKER A KONTEJNERIZACE . . . . .	37
2.4.2	ORCHESTRACE KONTEJNERŮ (DOCKER COMPOSE) . . . . .	38
<b>3</b>	<b>APLIKACE PRO ANDROID</b> . . . . .	<b>45</b>
3.1	KONCEPT A CÍLE APLIKACE . . . . .	45
3.1.1	NÁHRADA HARDWAROVÉHO TRACKERU . . . . .	45
3.1.2	PRINCIP OFFLINE-FIRST A ODOLNOST PROTI VÝPADKŮM . . . . .	45
3.2	TEORETICKÁ VÝCHODISKA A POUŽITÉ TECHNOLOGIE . . . . .	45
3.2.1	JAZYK KOTLIN A ASYNCHRONNÍ PROGRAMOVÁNÍ (COROUTINES) . . . . .	45
3.2.2	SYSTÉMOVÉ KOMPONENTY (FOREGROUND SERVICE, WORKMANAGER) . . . . .	45
3.2.3	LOKÁLNÍ PERSISTENCE (ROOM DATABASE) . . . . .	46
3.2.4	ZABEZPEČENÉ ÚLOŽIŠTĚ (ENCRYPTEDSHAREDPREFERENCES) . . . . .	46
3.3	ARCHITEKTURA APLIKACE . . . . .	46
3.3.1	VRSTEVNATÁ ARCHITEKTURA A REAKTIVNÍ MODEL (STATEFLOW) . . . . .	46
3.3.2	VLASTNÍ IMPLEMENTACE HTTP KLIENTA . . . . .	46
3.4	IMPLEMENTACE KLÍČOVÝCH FUNKCÍ . . . . .	46
3.4.1	SBĚR POLOHY NA POZADÍ (LOCATIONSERVICE) . . . . .	46

3.4.2	DÁVKOVÁ SYNCHRONIZACE DAT (SYNCWORKER) . . . . .	47
3.4.3	ŘÍZENÍ SPOTŘEBY A VZDÁLENÉ VYPÍNÁNÍ (POWERCONTROLLER) .	47
3.4.4	HANDSHAKE PROTOKOL A KONFIGURACE . . . . .	47
3.5	UŽIVATELSKÉ ROZHRANÍ . . . . .	47
3.5.1	HLAVNÍ OVLÁDACÍ PANEL (DASHBOARD) . . . . .	47
3.5.2	DIAGNOSTICKÁ KONZOLE PRO TERÉNNÍ TESTOVÁNÍ . . . . .	47
<b>ZÁVĚR</b>		<b>49</b>
<b>SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ</b>		<b>51</b>
<b>SEZNAM OBRÁZKŮ</b>		<b>55</b>
<b>SEZNAM TABULEK</b>		<b>55</b>
<b>PŘÍLOHY</b>		<b>55</b>



# ÚVOD

Můžeme říci, že již od začátku cílem projektu bylo vytvořit kompletní a hlavně uzavřený systém, co by mohl žít svým vlastním životem a jehož použití by bylo možné nasadit v reálných podmínkách.

Již kdysi jsem se zabýval "sběrnými"nebo "sledovacími"systémy. Poohlížel jsem se z variabilních důvodů po možnostech jak by šlo sestrojit zařízení schopné nahrávat video a ukládat na SD kartu nebo jiné uložiště. Sledování polohy je jenom dalším krokem v tomto směru. Možnost že bych měl krabičku jež bych mohl dát někomu do auta, nebo jej připnout k nápravě kamionu a sledoval jeho pohyb do doby vybití baterie byla, a stále je, velmi přitažlivá.

Bylo jasné, že již existují řešení a "GPS-trackery"jsou běžně dostupné a velmi dobré kvality a výdrže baterie. Avšak protože jsem dumal nad tématem pro zavěrečný projekt, výroba a řešení konstrukce vlastního, sic nízkokvalitního a dosti poruchového, zařízení se ukázal jako vhodný nápad. Existovala možnost vytvořit opravdu pouze "krabici"trackeru s ukládáním dat na pevné uložiště , avšak rozhodl jsem se to spojit s vývojem serveru jež by umožňoval sledování v reálném čase. Na tomto "webovém"serveru by tak šlo zobrazovat a spravovat data posílaná zařízením přes mobilní síť. To se pak přirozeně rozvinulo do plného systému s uživatelskými účty a možností spravovat více zařízení na jednom z nich. Již z počátků pak bylo také na výhledni ,že pokud by server měl fungovat a přijímat data spolehlivě, tak jeho jádrem by muselo být responzivní a hlavně "robustní"API. Jeho struktura se však, hlavně kvůli postupnému přehodnocuvání principu komunikace mezi serverem a zařízeními, dosti drasticky měnila. Fyzické zařízení totiž bylo hlavním pilířem celého systému a jeho konstrukce, složení, a hlavně samotný kod, udávaly směr vývoje všeho ostatního.

Jako méně důležitou, spíše doplňkovou, ale přesto užitečnou součást systému jsem pak vyděl souběžný vývoj mobilní aplikace pro systém Android. Ta měla sloužit jako plnohodnotná náhrada hardwarového zařízení a možná jako náhrada více schopná.



# 1 FYZICKÉ ZAŘÍZENÍ „TRACKER“

## 1.1 ÚVOD A KONCEPCE

## 1.2 POUŽITÉ TECHNOLOGIE

### 1.2.1 Mikrokontroléry a SoC (ESP32)

V oblasti vestavěných systémů (embedded systems) dnes dominují tzv. SoC (System on Chip) řešení, která integrují procesor, paměť a bezdrátová rozhraní do jediného pouzdra. Zástupcem této kategorie je čip **ESP32** od společnosti Espressif Systems. Jedná se o 32-bitový mikrokontrolér postavený na architektuře Xtensa LX6. Mezi jeho klíčové vlastnosti pro bateriově napájená zařízení patří pokročilá správa napájení. Čip disponuje několika režimy spánku, přičemž v tzv. *Deep Sleep* režimu je spotřeba redukována na řádově desítky mikroampérů, zatímco RTC (Real Time Clock) časovač zůstává aktivní a dokáže čip probudit.

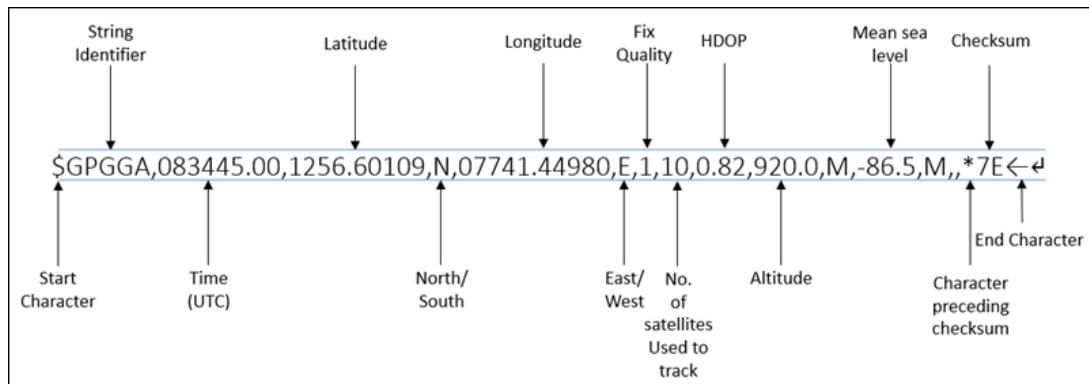
### 1.2.2 Mobilní komunikace v IoT a AT příkazy

Pro přenos telemetrických dat na velké vzdálenosti se využívají celulární sítě. V kontextu IoT se často setkáváme s technologiemi jako LTE-M nebo NB-IoT, avšak pro univerzální použití je stále relevantní standardní LTE (Long Term Evolution). Komunikace mezi řídícím mikrokontrolérem a LTE modemem probíhá tradičně prostřednictvím sériové linky (UART) s využitím sady **AT příkazů** (Hayes command set). Jedná se o textový protokol, kde každý příkaz začíná prefixem "AT".

### 1.2.3 Globální navigační systémy (GPS)

Systém GPS (Global Positioning System) funguje na principu měření zpoždění signálu vysílaného družicemi na oběžné dráze. Přijímač musí mít přímou viditelnost na minimálně 4 družice, aby mohl vypočítat svou trojrozměrnou polohu (zeměpisná šířka, délka, nadmořská výška) a přesný čas. Data z GPS modulů jsou standardně poskytována v textovém formátu **NMEA 0183**. Tento protokol definuje různé typy vět (sentences), z nichž nejpoužívanější je \$GPRMC (Recom-

mended Minimum Specific GPS/TRANSIT Data), obsahující klíčové navigační údaje.



Obrázek 1.1: NMEA sentence

#### 1.2.4 Operační systémy reálného času (FreeRTOS)

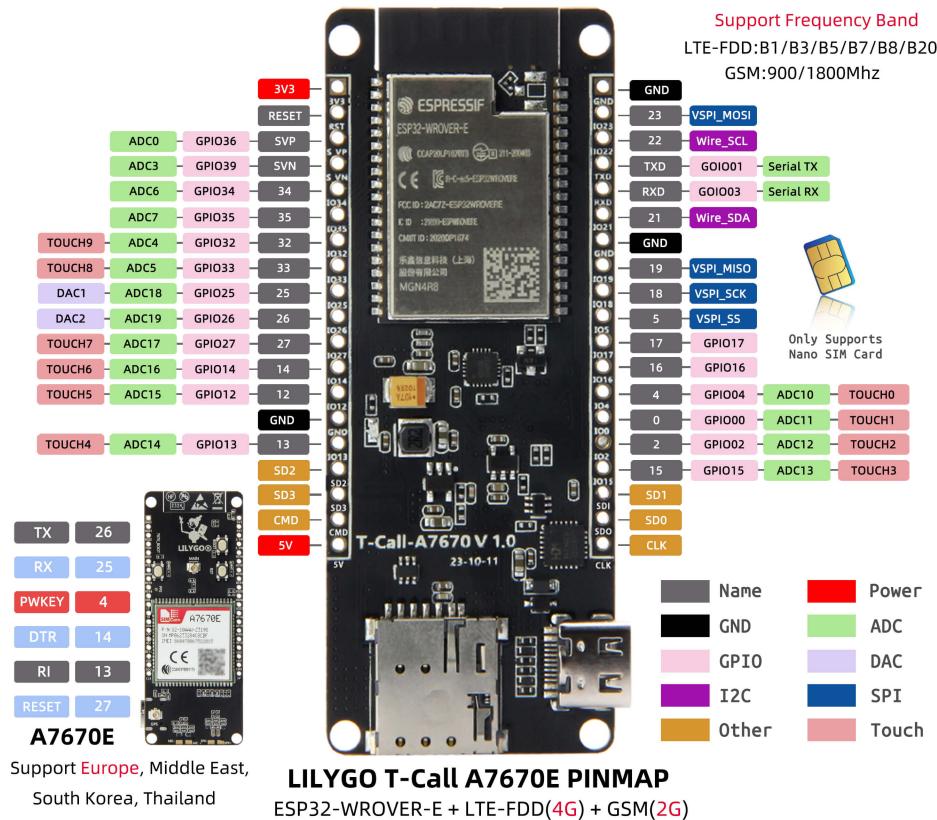
S rostoucí složitostí firmware přestává být dostačující klasická smyčka `while(1)`. Pro zajištění deterministického chování se využívají operační systémy reálného času (RTOS). **FreeRTOS** je open-source jádro, které umožňuje rozdělit aplikaci do samostatných úloh (Tasks) s definovanou prioritou. Plánovač (Scheduler) pak zajišťuje přepínání mezi těmito úlohami, což umožňuje "souběžný" běh komunikace, sběru dat a správy napájení.

#### 1.2.5 Souborový systém LittleFS

Pro ukládání konfiguračních dat a mezipaměti (cache) naměřených poloh přímo v paměti mikrokontroléru je využit souborový systém **LittleFS**. Jedná se o moderní souborový systém navržený speciálně pro mikrokontroléry s flash pamětí. Oproti staršímu SPIFFS (SPI Flash File System) přináší zásadní výhodu v podobě odolnosti vůči výpadkům napájení (power-loss resilience). LittleFS využívá techniku *Copy-on-Write*, kdy se data při úpravě nejprve zapíší na nové místo a teprve po úspěšném zápisu se aktualizuje ukazatel. To zaručuje, že ani při náhlém vybití baterie nedojde k poškození souborového systému a ztrátě dat.

## 1.3 NÁVRH HARDWARE - VÝBĚR KOMPONENT

Návrh hardwarové části systému vychází z požadavku na vytvoření energeticky nezávislého zařízení schopného provozu na baterii.



Obrázek 1.2: LilyGO T-Call V1.5 - integrované řešení ESP32 + LTE modem

### 1.3.1 LilyGO T-Call v1.5

Srdcem celého zařízení je vývojová deska **LilyGO T-Call** (verze V1.5), která v sobě integruje výkonný mikrokontrolér ESP32 a komunikační modem. Tato volba byla učiněna na základě následujících kritérií:

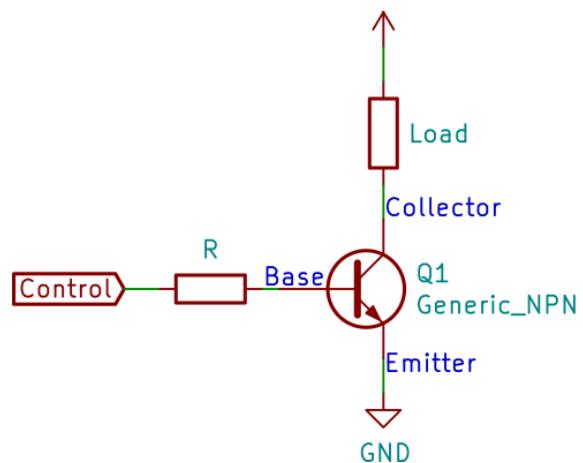
- **Integrace:** Spojení MCU a modemu na jedné desce eliminuje nutnost složitého propojování a snižuje riziko chyb v zapojení.
- **Konektivita:** Použitý modem SIMCOM A7670 podporuje moderní síť LTE (4G), což zajišťuje lepší pokrytí a nižší latenci než zastaralé 2G moduly (např. SIM800L).
- **Podpora komunity:** Pro platformu ESP32 existuje rozsáhlá dokumentace a knihovny, což výrazně urychluje vývoj.

### 1.3.2 Multi-GNSS L76K modul

Pro získávání polohy byl zvolen externí modul **Multi-GNSS L76K** (případně kompatibilní NEO-6M). Tento modul podporuje příjem signálu z více navigačních systémů současně (GPS, GLONASS, BeiDou, QZSS), což zvyšuje přesnost a rychlosť fixace polohy (TTFF - Time To First Fix), zejména v městské zástavbě nebo v Budovách. Deska LilyGo T-call sice disponuje možností zisku GPS přes vestavěný modem, avšak externí modul nabízí lepší citlivost a rychlosť zisku dat (dle testů až 4x rychleji), také to, že takový modul byl ve vlastnictví již před zahájením vývoje projektu, a tudíž nebylo nutné jej dokupovat prosadilo jeho použití.



(a) Externí modul Multi-GNSS L76K



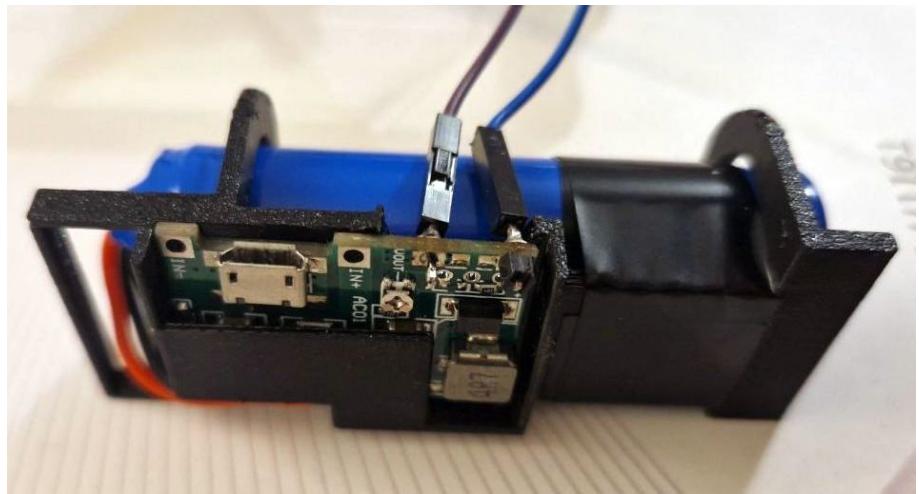
(b) Ovladání GND

Obrázek 1.3: Modul GPS a zapojení pro řízení napájení

Kvůli lepším možnostem ovladání spotřeby je modul GPS připojen přes tranzistorový spínač (NPN), který umožňuje mikrokontroléru zcela odpojit napájení GPS modulu, když není potřebný v zapnutém stavu. Aplikován je princip řízeného GND, bylo rozvažováno nad stabilnější možností ovládat VCC, avšak z důvodu absence PNP tranzistoru bylo zvoleno toto řešení.

### 1.3.3 Baterie, TP4056 + MT3608

Napájení zajišťuje Li-Ion článek typu 18650, který je dobíjen pomocí modulu s čipem **TP4056**. Pro zvýšení napětí z baterie (3.7V) na úroveň potřebnou pro stabilní provoz některých periferií (5V) je použit DC-DC step-up měnič **MT3608**.

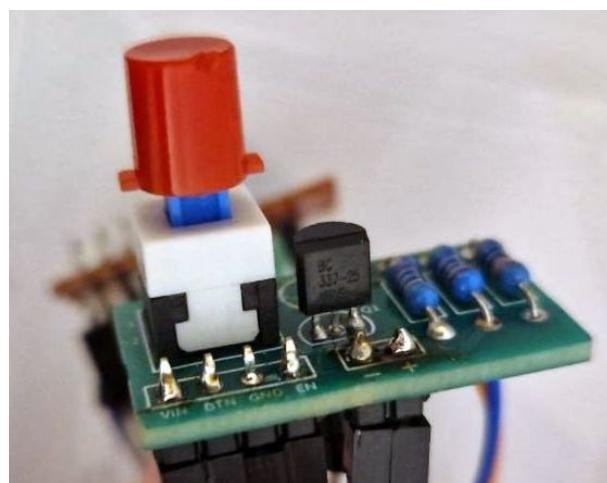


Obrázek 1.4: Baterie + nabíjecí modul TP4056 a step-up měnič MT3608 již v pouzdře

### 1.3.4 Řízení napájení a Power Latch modul

V kontextu komplexního systému, jakým je tracker, nestačí pouhé "tvrdé" odpojení od baterie vypínačem. Systém potřebuje čas na bezpečné ukončení procesů (uzavření souborů, odhlášení ze sítě). K tomuto účelu slouží obvod **Power Latch** (samodržný obvod). Jeho primárním cílem je umožnit mikrokontroléru převzít kontrolu nad vlastním napájením. Navržen byl dle rady a schématu od známeho v oboru Elektroniky. deska byla navržena v EasyEDA a vyrobena přes službu JLCPCB. Pro úsporu peněz byla osazena součástkami ručně (cena osazení cca. 4x převyšovala cenu součástek). Proto byly součástky vybrány dle požadavků na možnost ručního osazení (např. místo SMD vybrán mosfet v pouzdře TO-220 etc.).

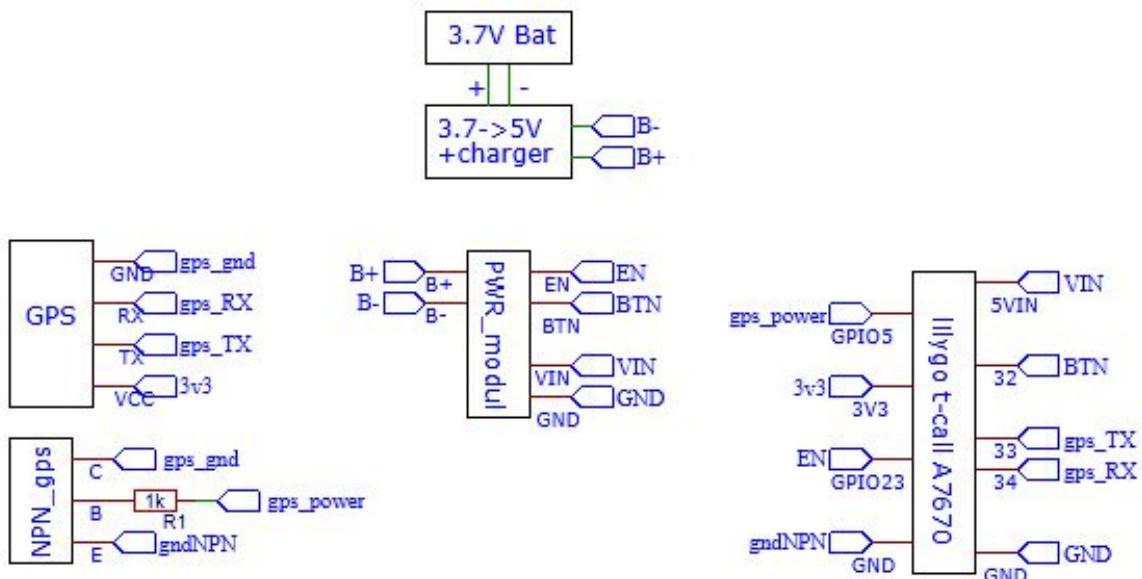
Viz. implementace ve firmware, sekce 1.4.



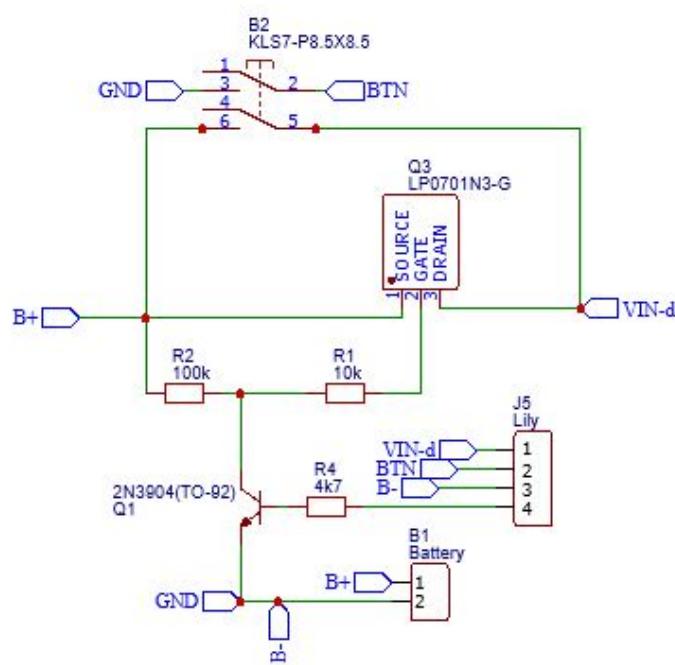
Obrázek 1.5: Osazená deska

Obrázek 1.6: Power Latch modul - Power Modul

## VÝSLEDNÁ SESTAVA - SCHÉMA



Obrázek 1.7: Blokové schéma zapojení hardwarového trackeru [easyEDA]



Obrázek 1.8: Schéma power-modul [easyEDA]

## 1.4 IMPLEMENTACE FIRMWARE

Software pro mikrokontrolér ESP32 je vyvíjen v jazyce C++ s využitím frameworku Arduino. Jako vývojové prostředí bylo zvoleno **Visual Studio Code** s rozšířením **PlatformIO**, které poskytuje pokročilé nástroje pro správu závislostí, kompliaci a nahrávání kódu.

### 1.4.1 Architektura a použité knihovny

Projekt je koncipován modulárně, což usnadňuje orientaci v kódu a případné budoucí rozšiřování. Zdrojový kód je rozdělen do samostatných jednotek (hlavičkové a zdrojové soubory) podle funkčních celků:

- `main.cpp`: Hlavní smyčka programu, řízení stavového automatu.
- `gps_control`: Obsluha GPS modulu, parsování NMEA zpráv, řízení napájení GPS.
- `modem_control`: Komunikace s LTE modemem přes AT příkazy, správa GPRS připojení a HTTP požadavků.
- `file_system`: Abstrakce nad souborovým systémem LittleFS, ukládání a načítání konfigurace a offline dat.
- `power_management`: Řízení spotřeby, ovládání Power Latch obvodu, přechod do Deep Sleep.
- `ota_mode`: Implementace servisního režimu, webového serveru a OTA aktualizací.

Mezi klíčové knihovny třetích stran, na kterých je firmware postaven, patří:

- **TinyGSM**: Univerzální knihovna pro komunikaci s GSM/LTE modemy. Pro tento projekt byl použit profil TINY\_GSM\_MODEM\_A7670.
- **TinyGPS++**: Efektivní parser NMEA dat z GPS modulu.
- **ArduinoJson**: Knihovna pro serializaci a deserializaci JSON objektů, používaná pro komunikaci s API serveru.
- **DeepsSleep.h**: Umožnuje použití "hlubokého spánku" v ESP32
- **Wifi.h**: knihovna pro připojení k WiFi síti, použitá v OTA režimu
- **FreeRTOS , LittleFS**: již zmíněny v sekci 1.2

## 1.4.2 Pracovní cyklus

Protože je použit textbfDeepSleep funkcionality, tak veškeré činnosti zařízení se odehrávají v `setup()`. Funkce `loop()`, obyčejně užita pro hlavní program, je nevyužita.

Standardní pracovní cyklus probíhá v následujících krocích:

1. **Probuzení a inicializace:** Po přivedení napájení (tlačítkem nebo časovačem) se provede inicializace periferií a načtení konfigurace ze souborového systému.
2. **Získání polohy (GPS Fix):** Zapne se GPS modul. Mikrokontrolér čeká na platná data o poloze. Pokud není fix získán do stanoveného limitu (timeout), pokus se ukončí.
3. **Uložení dat:** Získaná data (nebo informace o chybě) jsou uložena do interní paměti (LittleFS).
4. **Odeslání dat (Upload):** Aktivuje se modem. Zařízení se připojí k mobilní síti a pokusí se odeslat dávku uložených záznamů na server.
5. **Synchronizace:** Server potvrdí přijetí dat a případně pošle novou konfiguraci (např. změnu intervalu sledování).
6. **Uspání (Deep Sleep):** Po dokončení všech úloh se zařízení odpojí od sítě, vypne periferie a přejde do režimu hlubokého spánku na dobu definovanou v konfiguraci.

## 1.4.3 Paralelní zpracování a obsluha přerušení (FreeRTOS)

Vzhledem k tomu, že ESP32 disponuje dvěma jádry a běží na něm operační systém FreeRTOS, je možné efektivně rozdělit úlohy. Zatímco hlavní smyčka `loop()` řeší sekvenční logiku (GPS -> Modem -> Sleep), kritické události, jako je stisk tlačítka pro vypnutí, jsou řešeny asynchronně pomocí přerušení (ISR) a samostatné úlohy (Task).

Následující ukázka z `power_management.cpp` ukazuje inicializaci této logiky. Funkce `xTaskCreatePinnedToCore` vytvoří novou úlohu `ShutdownTask`, která běží na pozadí a čeká na signál z přerušení tlačítka.

## 1.4.4 Řízení napájení a Power Latch modul

V kontextu komplexního systému, jakým je tracker, nestačí pouhé "tvrdé" odpojení od baterie vypínačem. Systém potřebuje čas na bezpečné ukončení procesů (uzavření souborů, odhlášení ze sítě). K tomuto účelu slouží obvod **Power Latch** (samodržný obvod). Jeho primárním cílem je umožnit mikrokontrolérovi převzít kontrolu nad vlastním napájením.

Princip funkce je následující:

1. Uživatel stiskne tlačítko, čímž přivede napájení do systému.

```

void power_init() {
    if (shutdownTaskHandle != nullptr) {
        return; // Already initialized
    }
    pinMode(PIN_BTN, INPUT_PULLUP); // Use of internal pull-up so button can pull the line low

    // Ensure LED reflects normal run state (solid ON)
    status_led_set(true);

    // Create the FreeRTOS task for button handling
    xTaskCreatePinnedToCore(
        ShutdownTask,           // Task function
        "ShutdownTask",         // Name of task
        4096,                  // Stack size (bytes)
        NULL,                  // Parameter to pass to function
        configMAX_PRIORITIES - 1, // Task priority (highest)
        &shutdownTaskHandle, // Task handle
        0                      // Core where the task should run (Core 0)
    );

    // Attach interrupt to the button pin
    attachInterrupt(digitalPinToInterruption(PIN_BTN), on_button_isr, FALLING);
    DBG_PRINTLN(F("[POWER] Button interrupt and shutdown task initialized."));
}

```

Obrázek 1.9: funkce power\_init()

2. ESP32 se nastartuje a okamžitě nastaví pin PIN\_EN na úroveň HIGH. Tím "přemostí" tlačítko a drží se pod napětím i po jeho uvolnění.
3. Když chce zařízení přejít do vypnutého stavu (Deep Sleep nebo úplné vypnutí), provede potřebné úkony a následně nastaví PIN\_EN na LOW, čímž se samo odpojí.

Následující ukázka kódu demonstruje funkci graceful\_shutdown(), která zajistí uje bezpečné vypnutí. Všimněte si, že před samotným shozením pinu PIN\_EN (které fyzicky odpojí baterii) se nejprve vypne modem a GPS, aby nedošlo k poškození periferií nebo ztrátě dat.

#### 1.4.4.1 Režim hlubokého spánku (Deep Sleep)

Kromě úplného vypnutí (Power Off) využívá zařízení také režim hlubokého spánku. Tento režim je klíčový pro cyklické probouzení trackeru bez nutnosti interakce uživatele. V režimu Deep Sleep je vypnuto CPU, RAM i většina periferií. Napájen zůstává pouze RTC (Real Time Clock) řadič a malá část paměti (RTC Slow Memory). Spotřeba čipu v tomto stavu klesá na jednotky mikroampérů.

Probuzení z tohoto režimu může nastat dvěma způsoby:

1. **Časovač (Timer Wakeup):** Po uplynutí nastaveného intervalu (např. 10 minut).
2. **Externí signál (Ext0 Wakeup):** Stiskem tlačítka (změna logické úrovně na pinu GPIO32).

```

void graceful_shutdown() {
    DBG_PRINTLN(F("\n[POWER] Shutdown requested - starting graceful power-off..."));
    DBG_FLUSH();

    // Detach interrupt to prevent any further triggers during shutdown
    detachInterrupt(digitalPinToInterrupt(PIN_BTN));

    g_shutdown_requested = true;
    power_status_mark_off();
    status_led_set(false);

    // Ask long-running tasks to abort before we tear down shared resources
    gps_request_abort();

    // Call functions from other modules to power down peripherals
    modem_disconnect_gprs(); // Defined in modem_control.cpp
    modem_power_off(); // Defined in modem_control.cpp
    // Wait briefly for GPS loops to notice the abort request
    for (int i = 0; i < 50 && gps_is_active(); ++i) {
        vTaskDelay(pdMS_TO_TICKS(10));
    }
    gps_close_serial(); // Defined in gps_control.cpp
    gps_power_down(); // Defined in gps_control.cpp
    fs_end(); // Defined in file_system.cpp

    // Finally, cut power to the ESP32
    pinMode(PIN_EN, OUTPUT);
    digitalWrite(PIN_EN, LOW);
    delay(100); // Give some time for power to cut

    // Fallback: if EN is not a true "latch", enter deep sleep indefinitely
    esp_deep_sleep_start();
}

```

Obrázek 1.10: funkce graceful\_shutdown()

Následující funkce enter\_deep\_sleep ukazuje konfiguraci těchto budících zdrojů před uspáním procesoru.

```

void enter_deep_sleep(uint64_t seconds) {
    DBG_PRINT(F("[SLEEP] Entering deep sleep"));

    // Detach interrupt before sleeping to prevent issues on wake
    detachInterrupt(digitalPinToInterrupt(PIN_BTN));

    // Enable wakeup by timer
    esp_sleep_enable_timer_wakeup(seconds * 1000000ULL); // microseconds
    // Enable wakeup by button (on LOW level)
    esp_sleep_enable_ext0_wakeup(static_cast<gpio_num_t>(PIN_BTN), 0);

    status_led_set(false);

    esp_deep_sleep_start();
}

```

Obrázek 1.11: funkce enter\_deep\_sleep()

#### 1.4.5 Zpracování GPS dat a parsování NMEA

Komunikace s GPS modulem L76K probíhá přes hardwarovou sériovou linku (UART1). Modul v pravidelných intervalech (1 Hz) odesílá textová data ve formátu NMEA 0183. Pro efektivní zpracování tohoto proudu dat je využita knihovna **TinyGPS++**.

Klíčovou částí implementace je funkce `gps_get_fix`, která v cyklu čte znaky ze sériového portu a předává je parseru pomocí metody `gps.encode()`. Jakmile parser detekuje platnou větu a aktualizuje souřadnice, zkontroluje se, zda jsou splněny podmínky pro platný fix (platná poloha, čas a minimální počet satelitů).

```
bool gps_get_fix(unsigned long timeout) {
    unsigned long startTime = millis();
    unsigned long lastPrintTime = 0;
    gpsFixObtained = false;
    gpsAbortRequested = false;
    gpsLoopActive = true;

    while (millis() - startTime < timeout) {
        if (gpsAbortRequested) {
            DBG_PRINTLN(F("[GPS] Fix attempt aborted."));
            break;
        }
        while (SerialGPS.available() > 0) {
            if (gpsAbortRequested) break; // Immediate exit if requested inside the read loop

            if (gps.encode(SerialGPS.read())) {
                if (gps.location.isUpdated() && gps.location.isValid() &&
                    gps.date.isValid() && gps.time.isValid() &&
                    gps.satellites.isValid() && (gps.satellites.value() >= minSatellitesForFix)) {
                    gps_display_and_store_info();
                    gpsFixObtained = true;
                    break;
                }
            }
            if (gpsFixObtained) {
                break;
            }
        }
        gpsLoopActive = false;
        return gpsFixObtained;
    }
}
```

Obrázek 1.12: funkce `gps_get_fix()`

#### 1.4.6 Správa datového úložiště (LittleFS)

Jedním z klíčových požadavků byla schopnost pracovat i v oblastech bez signálu GSM. Firmware proto implementuje robustní systém cachování dat. Naměřené polohy nejsou odesílány okamžitě, ale jsou nejprve serializovány a uloženy do souboru `/gps_cache.log` v paměti flash (LittleFS). Při každém úspěšném připojení k internetu se zařízení pokusí odeslat dávku nejstarších záznamů (FIFO - First In, First Out). Teprve po potvrzení serverem (success: true).

Následuje příklad funkce `textttfs_init()` jež zajišťuje nabootovaní souborového systému.

```

bool fs_init() {
    FsLockGuard lock;
    if (!lock.isLocked()) {
        DBG_PRINTELN(F("[FS] Failed to acquire FS lock during init."));
        return false;
    }
    if (!LittleFS.begin()) {
        DBG_PRINTELN(F("[FS] An Error has occurred while mounting LittleFS"));
        return false;
    }
    DBG_PRINTELN(F("[FS] LittleFS mounted successfully."));
    preferences.begin(PREFERENCES_NAMESPACE, false); // false for read/write
    DBG_PRINTELN(F("[FS] Preferences initialized."));
    return true;
}

```

Obrázek 1.13: funkce `fs_init()`

## 1.5 KOMUNIKACE A DATA

### 1.5.1 Komunikační protokol a zabezpečení

Komunikace mezi zařízením a serverem probíhá přes protokol HTTP(S). Všechna data jsou formátována jako JSON objekty. Z důvodu bezpečnosti a integrity dat je preferováno šifrované spojení (HTTPS).

Pro navázání zabezpečeného spojení využívá modem knihovnu SSL/TLS. Následující ukázka kódů z funkce `modem_send_post_request` demonstruje inicializaci HTTPS relace a nastavení cílové URL.

### 1.5.2 Struktura dat a handshake

Komunikace se serverem probíhá prostřednictvím REST API rozhraní. Zařízení odesílá data ve formátu JSON (JavaScript Object Notation), který je dnes standardem pro výměnu dat v IoT aplikacích díky své čitelnosti a široké podpoře.

Detailní specifikace jednotlivých endpointů, struktura přenášených zpráv a návratové kódy jsou podrobně popsány v kapitole věnované serverové části (viz sekce ??).

Pro operace a práci s odesílanými a přijímanými JSON daty je využita efektivní knihovna **ArduinoJson**. Následující ukázka kódů z funkce `modem_perform_handshake` demonstruje sestavení JSON objektu pro úvodní "pozdrav"(handshake) a zpracování odpovědi.

```

DBG_PRINTLN(F("[MODEM] beginning HTTPS session."));
g_modem.https_begin();
| DBG_PRINT(F("[MODEM] Set URL: "));
| DBG_PRINTLN(fullUrl);
| g_modem.https_set_url(fullUrl.c_str());

| DBG_PRINTLN(F("[MODEM] Set Content-Type header..."));
| g_modem.https_set_content_type("application/json")

| DBG_PRINTLN(F("[MODEM] Sending POST request..."));
int statusCode = g_modem.https_post(payload);

response_body = g_modem.https_body();

| DBG_PRINTLN(F("[MODEM] End HTTPS session."));
| g_modem.https_end();

DBG_PRINTLN(F("[MODEM] Response Body:"));
return response_body;

```

Obrázek 1.14: funkce modem\_send\_post\_request()

```

bool modem_perform_handshake() {
    JsonDocument payloadDoc;
    payloadDoc["device_id"] = deviceID;
    payloadDoc["client_type"] = CLIENT_TYPE;
    payloadDoc["power_status"] = power_status_to_string(power_status_get());

    String payload;
    serializeJson(payloadDoc, payload);

    DBG_PRINTLN(F("[MODEM] Performing device handshake..."));
    int statusCode = 0;
    String response = modem_send_post_request(RESOURCE_HANDSHAKE, payload, &statusCode);

    if (statusCode == 404) {
        DBG_PRINTLN(F("[MODEM] Handshake responded 404 - device not registered."));
        fs_set_registered(false);
        return false;
    }
}

```

Obrázek 1.15: funkce modem\_perform\_handshake()

### 1.5.3 Práce s Modemem a AT příkazy

Komunikace s LTE modemem SIMCOM A7670 probíhá prostřednictvím sériové linky (UART) pomocí sady standardizovaných AT příkazů. Firmware využívá knihovnu **TinyGSM**, která tuto nízkoúrovňovou komunikaci abstrahuje do vysokoúrovňových metod (např. `gprsConnect`, `https_post`).

Přesto je v některých případech nutné zasahovat do komunikace přímo, například při inicializaci modemu nebo diagnostice. Následující ukázka z funkce `modem_initialize` ukazuje sekvenci AT příkazů pro ověření dostupnosti modemu a jeho zapnutí.

```
bool modem_initialize() {  
  
    // initialization sequence ...  
  
    // Initialize SerialAT immediately  
    SerialAT.begin(115200, SERIAL_8N1, MODEM_RX_PIN, MODEM_TX_PIN);  
    delay(100);  
  
    bool modemReady = false;  
  
    // Check 1: Already ON? (Quick check)  
    if (g_modem.testAT(500)) {  
        DBG_PRINTLN(F("[MODEM] Modem responded to AT. It is already ON."));  
        modemReady = true;  
    } else {  
        DBG_PRINTLN(F("[MODEM] No response. Performing Power-On sequence (Attempt 1)..."));  
    }  
}
```

Obrázek 1.16: funkce `modem_initialize()`

Knihovna TinyGSM následně překládá volání funkcí na konkrétní AT příkazy. Například volání `g_modem.getSignalQuality()` odešle příkaz AT+CSQ a vyparsuje odpověď ve formátu +CSQ: <rss>, <ber>.

## 1.6 KONFIGURACE A SERVISNÍ REŽIM

Pro prvotní nastavení nebo servisní zásahy v terénu disponuje zařízení speciálním režimem "OTA"(Over-The-Air). Tento režim se aktivuje dlouhým stiskem ovládacího tlačítka při startu zařízení (cca 2 sekundy).

Detekce tohoto stavu probíhá v nejranější fázi funkce `setup()`, ještě před inicializací ostatních periferií. Pokud je detekován dlouhý stisk, zařízení nespustí standardní pracovní cyklus, ale přejde do nekonečné smyčky servisního režimu.

---

```

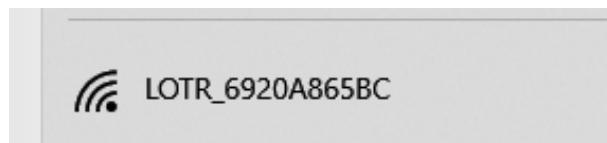
// 2. Start WiFi AP
DBG_PRINTLN(F("Starting WiFi AP..."));
WiFi.disconnect(true);
WiFi.mode(WIFI_AP);
bool apStarted = ota_password.length() == 0
    ? WiFi.softAP(ota_ssid.c_str())
    : WiFi.softAP(ota_ssid.c_str(), ota_password.c_str());
if (apStarted) {
    if (ota_password.length() == 0) {
        DBG_PRINTLN(F("WiFi AP started as open network."));
    } else {
        DBG_PRINTLN(F("WiFi AP started with configured credentials."));
    }
} else {
    DBG_PRINTLN(F("[OTA] Failed to start WiFi AP with configured credentials;
}
IPAddress apIP = WiFi.softAPIP();
DBG_PRINT(F("AP IP address: "));
DBG_PRINTLN(apIP);

```

Obrázek 1.18: konfigurace Wifi APN pro OTA režim

### 1.6.1 OTA režim a webové rozhraní

V servisním režimu se ESP32 přepne do role Wi-Fi přístupového bodu (Access Point) s názvem lotrTrackerOTA\_<DeviceID> a spustí webový server na adrese 192.168.4.1. Uživatel se může připojit pomocí mobilního telefonu nebo notebooku a přes webové rozhraní provádět údržbu.



Obrázek 1.17: wifi\_apn

Webové rozhraní nabízí následující funkce:

- **Registration:** Spárování nového zařízení s uživatelským účtem.
- **Settings:** Konfigurace APN, adresy serveru a portu.
- **Test GPRS & Test Serverové konektivity:** Okamžité ověření konektivity modemu.
- **Firmware Update:** Nahrání nové binární verze firmware přímo z prohlížeče.
- **Clear Cache:** Pokud je potřeba, umožňuje vymazat uložená data v paměti.(např. pokud jsou data poškozena či ve špatném formátu a nelze je odeslat).

## 1.6.2 Manuální a Vzdálená konfigurace zařízení

Konfigurace, at' již nastavena v /settings (OTA režim) nebo samotné nastavení posílané Ser-verem, je trvale uložena v paměti pomocí knihovny Preferences. Mezi klíčové parametry patří:

- apn, gprsUser, gprsPass: Nastavení mobilní sítě.
- server, port: Cílová adresa backendu.
- sleepTimeSeconds: Interval mezi probuzeními.
- minSatellitesForFix: Minimální počet satelitů pro validní fix.
- etc...

## 1.7 SEZNAM SOUČÁSTEK + OBRÁZKY

- LilyGO T-Call V1.5 (ESP32 + LTE modem A7670)
- Externí GPS modul Multi-GNSS L76K (alternativně NEO-6M)
- LTE mikroSIM karta, datový tarif
- GPS anténa (u.FL/SMA dle modulu)
- LTE anténa (u.FL/SMA dle desky)
- Li-Ion 18650 článek
- Nabíjecí modul TP4056 + Step-up měnič MT3608
- SEMTECH BC337-25 bipolární NPN tranzistor
- INFINEON IRF4905PBF P-Channel MOSFET
- Rezistory (1 k $\Omega$ , 330  $\Omega$ , 100 k $\Omega$ )
- KLS 7-P8.0x8.0-0 non lock tlačítko do DPS, 2 póly, ON-(ON)
- LED dioda 3mm zelená

```

[BOOT] Long press detected. Entering OTA mode.
--- OTA Service Mode Activated ---
[POWER] Button interrupt and shutdown task initialized.
[FS] LittleFS mounted successfully.
[FS] Preferences initialized.
[FS] Configuration loaded from Preferences.
Device ID (last 10 of MAC): 6920A865BC
[OTA] Initializing modem synchronously...
[MODEM] Initializing modem...
[MODEM] No response. Performing Power-On sequence (Attempt 1)...
[MODEM] Resetting modem...
[MODEM] Toggling PWRKEY...
[MODEM] PWRKEY toggled. Waiting for boot...

[MODEM] AT command responded.
[MODEM] Initializing modem with modem.init()...
[MODEM] Modem init successful.
[MODEM] Modem Info: Manufacturer: SIMCOM INCORPORATED Model: A7670E-FASE Revision: A7670M7_V1.11.1 IMEI: 867255079769784
[MODEM] Waiting for network... success
[MODEM] Connecting to GPRS: internet.t-mobile.cz success
[MODEM] GPRS IP: 100.109.85.14
[OTA] Modem ready and connected. Performing handshake...
[MODEM] Performing device handshake...
[MODEM] Performing HTTPS POST to: /api/devices/handshake
[MODEM] Payload: {"device_id":"6920A865BC","client_type":"HM","power_status":"ON"}
[MODEM] Set URL: https://iotr-system.xyz/api/devices/handshake
[MODEM] Set Content-Type header...
[MODEM] Sending POST request...
[MODEM] Response Status Code: 200
[MODEM] Reading response body...
[MODEM] End HTTPS session.
[MODEM] Response Body:
{"registered":true,"config":{"interval_gps":20,"interval_send":5,"satellites":7,"mode":"batch"},"power_instruction":"NONE"}
[FS] Server set GPS interval to: 20
[FS] Server set batch send threshold to: 5
[FS] Server set minimum satellites to: 7
[FS] Server set operation mode to: batch
[OTA] Handshake success. Reloading config.
[FS] Configuration loaded from Preferences.
Starting WiFi AP...
WiFi AP started as open network.
AP IP address: 192.168.4.1
OTA Web Server started. Waiting for connections...

```

Obrázek 1.19: log z OTA režimu

**Device Service Mode**

Device ID: 6920A865BC

GPRS Status: Connected

Registration: Registered

---

**Register Device**

If this device is not registered, enter your account details below.

Username:

Password:

**Register Device**

[Settings](#) | [Firmware Update](#)

**Device Settings**

**GPRS Configuration**

GPRS test connection successful.  
APN:

GPRS User:

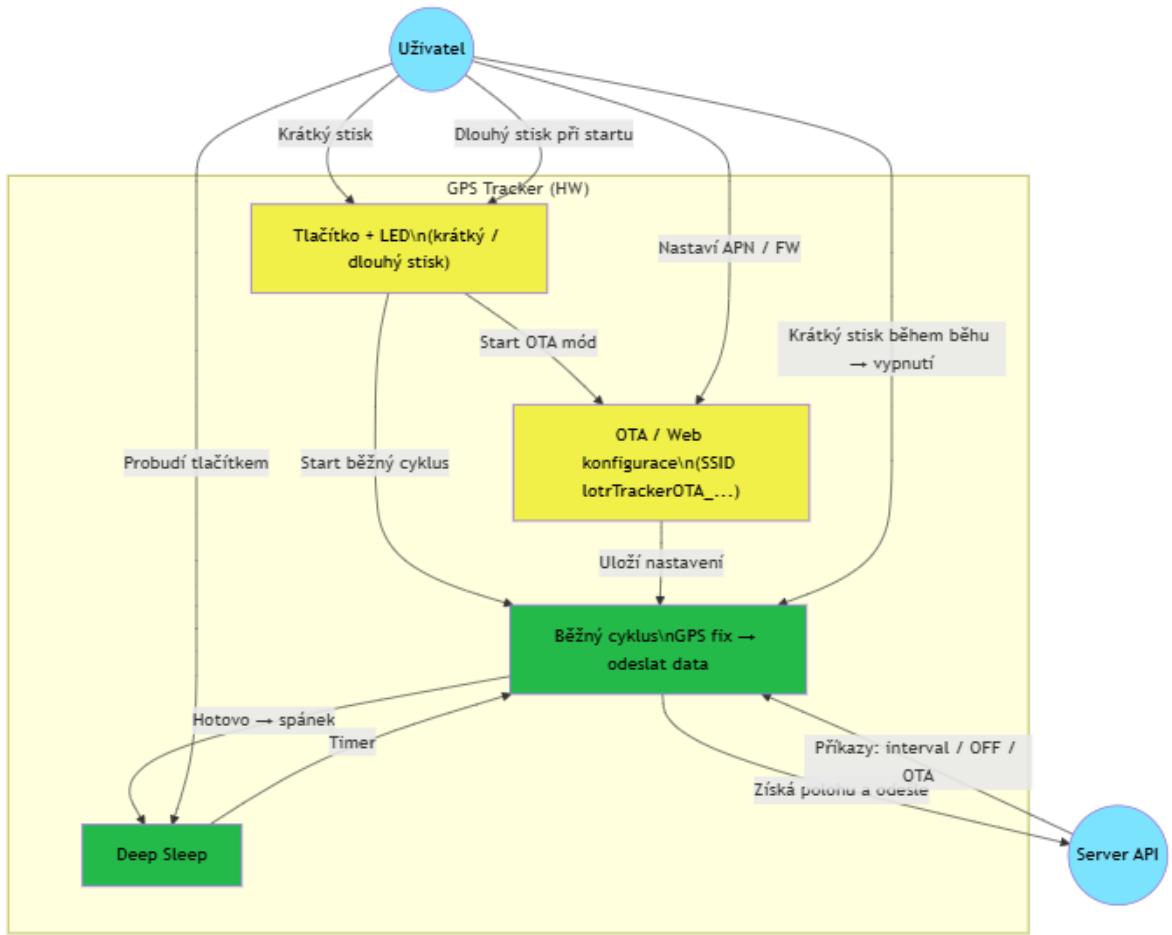
GPRS Password:

Confirm GPRS Password:

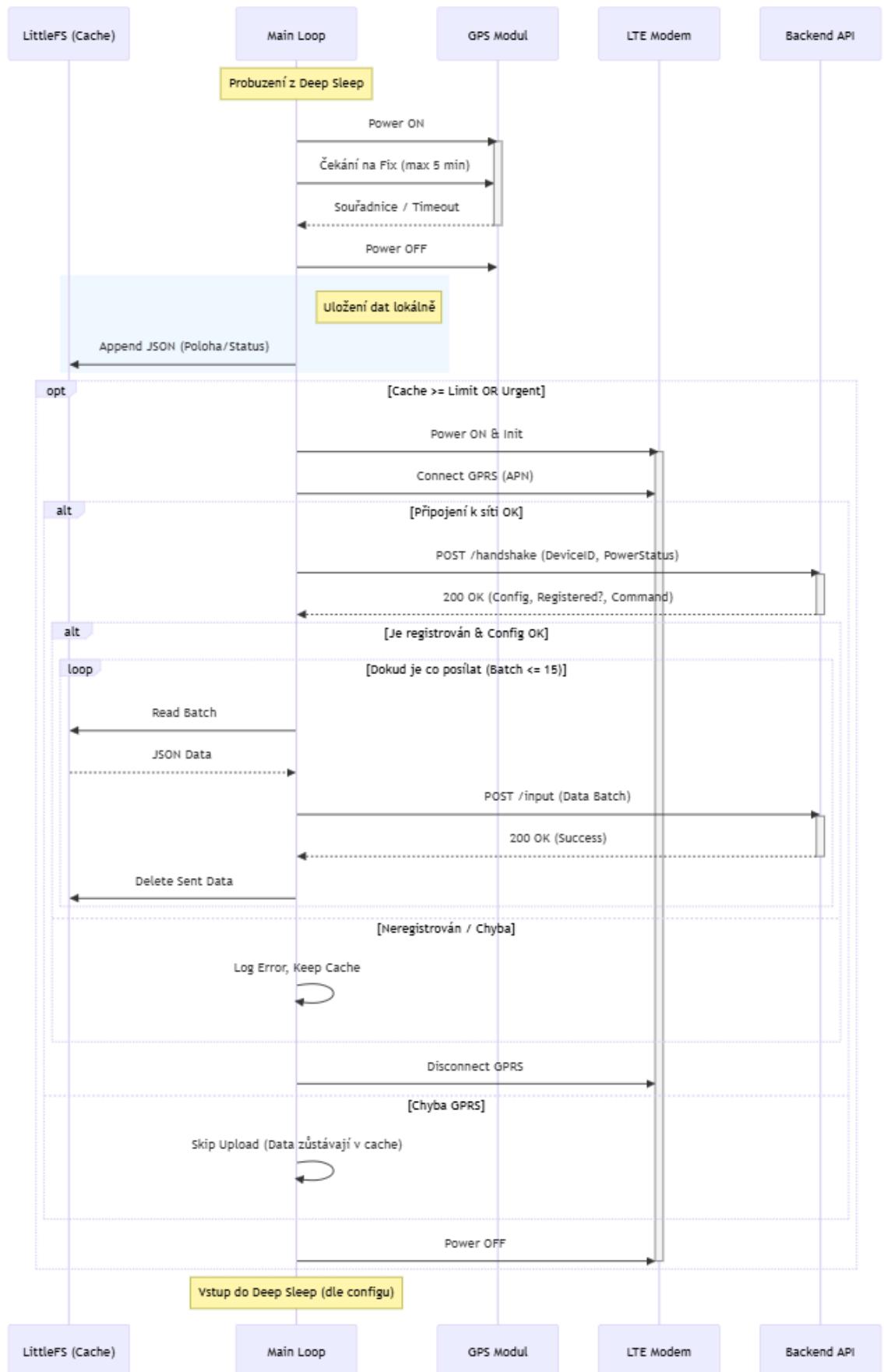
(a) registrace

(b) OTA režim - webové rozhraní

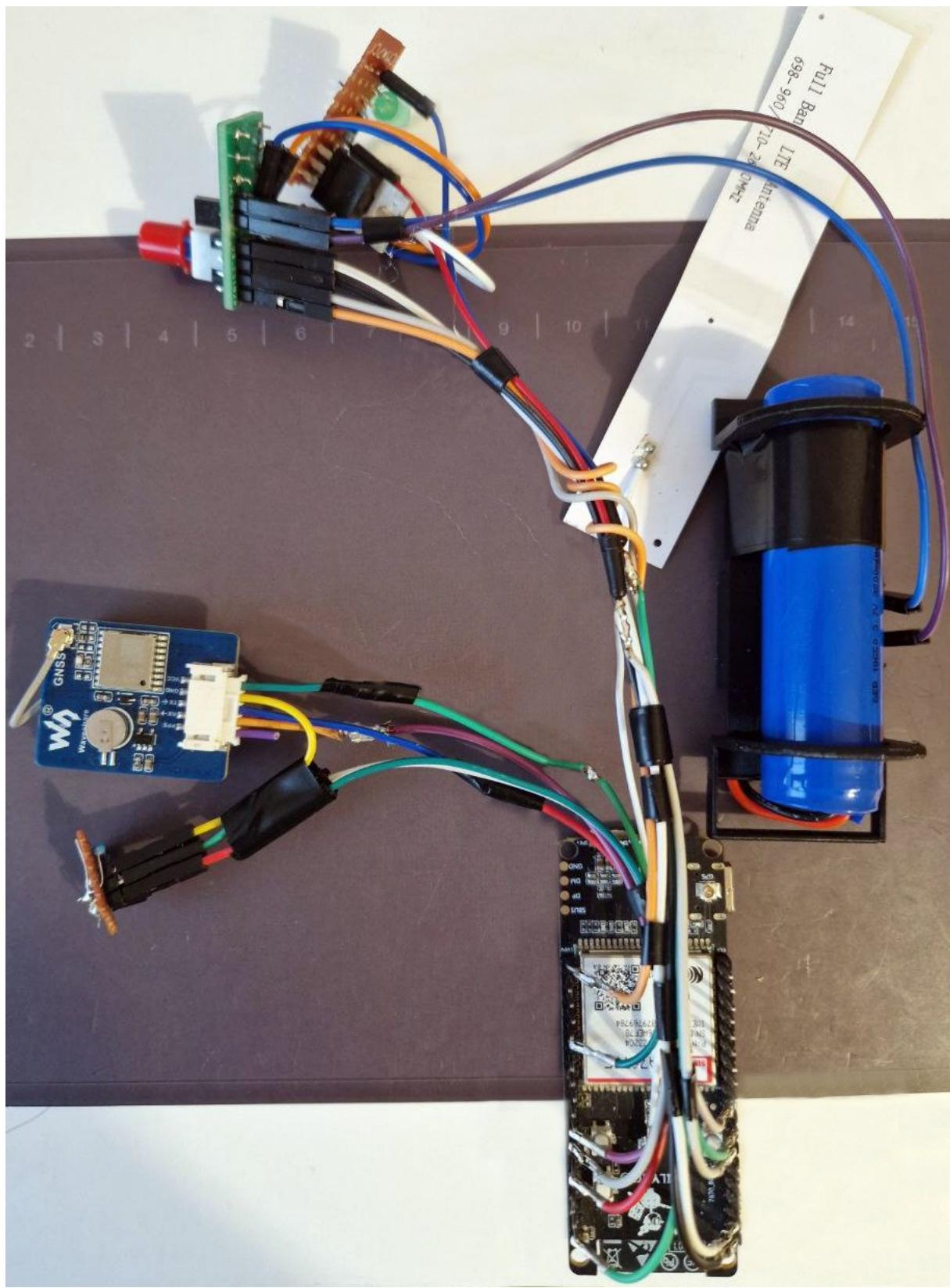
Obrázek 1.20: OTA režim - webové rozhraní



Obrázek 1.21: Usecase diagram



Obrázek 1.22: sekvenční diagram životního cyklu



Obrázek 1.23: "vnitřnosti" zařízení

## 2 SERVEROVÁ ČÁST

### 2.1 ÚVOD A KONCEPCE SYSTÉMU

Serverová část systému LOTR představuje centrální bod celého systému. Zajišťuje komunikaci s hardwarovými jednotkami, trvalé ukládání telemetrických dat, správu uživatelských účtů a poskytování uživatelského rozhraní pro vizualizaci polohy a stavu zařízení.

#### 2.1.1 Role serveru v systému LOTR

Hlavním úkolem serveru je agregace dat z jednotlivých GPS trackerů. Hardwarové jednotky odesírají data (poloha, stav baterie, síla signálu) prostřednictvím mobilní sítě na definované API endpointy. Server tato data validuje, zpracovává a ukládá do relační databáze. Druhým klíčovým úkolem je obsluha klientských požadavků. Uživatelé přistupují k systému prostřednictvím webového prohlížeče. Server zajišťuje autentizaci uživatelů a generuje dynamické HTML stránky zobrazující mapové podklady s aktuální polohou sledovaných objektů.

#### 2.1.2 Monolitická architektura a MVC vzor

Aplikace je navržena jako monolit, což znamená, že backendová logika i prezentační vrstva (frontend) jsou součástí jednoho projektu a běží v rámci jednoho procesu. Pro organizaci kódu byl zvolen architektonický vzor **MVC (Model-View-Controller)**, který odděluje data, logiku a zobrazení:

- **Model (Model):** Definuje strukturu dat a logiku pro přístup k databázi. V našem případě je tato vrstva realizována pomocí ORM knihovny Sequelize (viz sekce 2.2.1). Modely odpovídají tabulkám v databázi (např. User, Device, Telemetry).
- **View (Pohled):** Stará se o prezentaci dat uživateli. Využíváme šablonovací systém EJS, který umožňuje vkládat data z backendu přímo do HTML struktury.
- **Controller (Řadič):** Přijímá požadavky od uživatele (nebo API), zpracovává je s využitím Modelů a rozhoduje, jaký Pohled se má zobrazit, nebo jaká data se mají vrátit (v případě JSON API).

Tento přístup usnadňuje údržbu kódu a umožňuje paralelní vývoj jednotlivých částí aplikace.

### 2.1.3 Relační databáze a ORM (MySQL + Sequelize)

Pro ukládání strukturovaných dat s jasně definovanými vazbami jsou standardem relační databáze (RDBMS). Systém **MySQL** poskytuje transakční zpracování dat a dodržuje principy ACID (Atomicity, Consistency, Isolation, Durability), což je nezbytné pro zajištění integrity telemetrických dat. Pro interakci s databází se v moderním vývoji často využívá technika **ORM** (Object-Relational Mapping). Knihovna (zde Sequelize) mapuje databázové tabulky na třídy (objekty) v programovacím jazyce. Vývojář tak nemusí psát surové SQL dotazy, ale manipuluje s daty pomocí metod objektů (např. `User.findAll()`).

### 2.1.4 Principy autentizace a OAuth 2.0

Zabezpečení přístupu k API a webovému rozhraní vyžaduje robustní autentizační mechanismus. Kromě klasického ověření jménem a heslem se stále častěji využívá standard **OAuth 2.0**. Tento protokol umožňuje uživateli udělit aplikaci přístup ke svým údajům na jiné službě (např. Google, GitHub) bez nutnosti sdílet své heslo. Aplikace získá pouze tzv. *Access Token*, který slouží k ověření identity. Implementováno pomocí knihovny `passport.js`.

### 2.1.5 REST API architektura

**REST** (Representational State Transfer) je architektonický styl pro návrh síťových aplikací. REST API definuje sadu pravidel pro komunikaci mezi klientem a serverem:

- **Bezstavovost (Stateless):** Server neuchovává stav klienta mezi požadavky. Každý požadavek musí obsahovat všechny potřebné informace (např. autorizační token).
- **Jednotné rozhraní:** Zdroje (data) jsou identifikovány pomocí URL a manipuluje se s nimi pomocí standardních HTTP metod (GET pro čtení, POST pro vytvoření, PUT pro úpravu, DELETE pro smazání).

## 2.2 NÁVRH A IMPLEMENTACE BACKENDU

### 2.2.1 Databázová vrstva (MySQL a Sequelize)

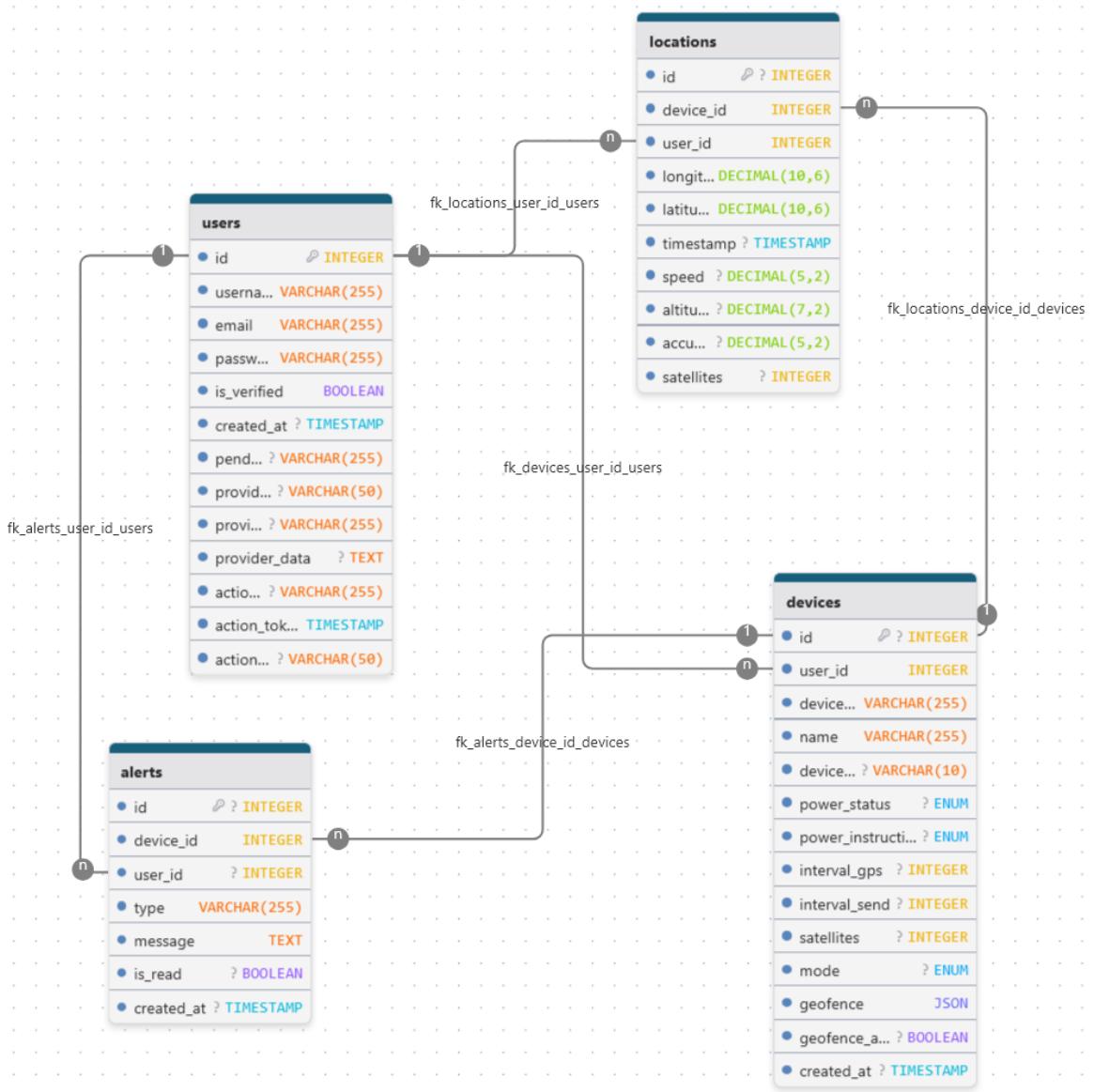
Databázová vrstva je postavena na relační databázi MySQL. Pro komunikaci s databází je využita knihovna `mysql2` ve spojení s ORM frameworkm Sequelize.

### 2.2.1.1 ORM Sequelize a definice modelů

Sequelize abstrahuje SQL dotazy do JavaScriptových objektů. V projektu jsou definovány následující klíčové modely:

- **User:** Ukládá informace o uživatelích (jméno, email, hash hesla, OAuth ID).
- **Device:** Reprezentuje hardwarové jednotky (tracker). Obsahuje unikátní identifikátor, název a vazbu na vlastníka (User).
- **Location:** Uchovává historická data o poloze (zeměpisná šířka, délka, čas, rychlosť, stav baterie).
- **Alert:** Záznamy o bezpečnostních událostech (např. opuštění geofence zóny).

Vztahy mezi modely jsou definovány jako 1:N (jeden uživatel má více zařízení, jedno zařízení má více záznamů polohy).



Obrázek 2.1: ER diagram databázových modelů

### 2.2.1.2 API pro HW tracker (ESP32)

HW klient se autentizuje pouze pomocí device\_id registrovaného u uživatele. Nevyužívá session/cookie, neodesílá uživatelská hesla.

Endpoint	Metoda	Popis
/api/devices/register	POST	Registrace HW zařízení k účtu. Payload: client_type="HW", device_id, username, password, volitelné name.
/api/devices/handshake	POST	Periodická synchronizace stavu a konfigurace. Payload: device_id, client_type="HW", power_status. Odezva obsahuje registered, config, power_instruction.
/api/devices/input	POST	Příjem telemetrie. Payload: jeden objekt nebo pole objektů se souřadnicemi, časem, rychlostí, power_status.

Tabulka 2.1: HW API endpointy

1. Každý záznam musí obsahovat device, latitude, longitude, timestamp, může obsahovat power\_status.
2. Server uloží lokace, aktualizuje last\_seen; pokud power\_status potvrzuje instrukci, vynuluje power\_instruction.
3. Odezva "success": true; až poté klient smaže data z bufferu.

Chybové stavy (HW):

- 404 / registered=false: zařízení není známo; má přejít do konfiguračního režimu.
- 409: device\_id patří jinému účtu; zařízení musí zastavit akce.
- 500+: klient retry s backoff; data v bufferu se nemažou.

### 2.2.1.3 API pro APK klient (Android)

APK klient používá uživatelské přihlášení (session cookie) a odděleně identifikuje zařízení pomocí device\_id = installationId. Telemetrie i handshake sdílí endpointy s HW, ale autentizační vrstva je odlišná.

Endpoint	Metoda	Popis
/api/apk/login	POST	Přihlášení uživatele; vyžaduje ověřený účet ( <code>user.is_verified=true</code> ); vydá HTTP-only cookie <code>connect.sid</code> .
/api/apk/logout	POST	Zrušení session; odhlášení APK klienta.
/api/devices/register	POST	Registrace zařízení typu APK. Payload: <code>client_type="APK"</code> , <code>device_id</code> (UUID), <code>name</code> ; vyžaduje platnou session.
/api/devices/handshake	POST	Synchronizace konfigurace a power instrukcí. Payload: <code>device_id</code> , <code>client_type="APK"</code> , <code>power_status</code> , <code>reason</code> .
/api/devices/input	POST	Odeslání telemetrije z APK (batch array). Stejný formát jako HW, navíc může obsahovat <code>accuracy</code> .
/api/devices/coordinates	GET	Vrací poslední známé polohy všech zařízení přihlášeného uživatele (používá se pro mapu ve webu/APK).

Tabulka 2.2: APK API endpointy

APK Handshake (stejný endpoint, odlišný kontext): APK spouští handshake při startu služby, periodicky (např. 15 min) a po odeslání dávky. Server vrací `registered`, `config` a `power_instruction`; APK musí respektovat `TURN_OFF` zastavením služby.

APK Input: Telemetrie se odesílá dávkově z lokální SQLite (store-and-forward). Každý záznam obsahuje `device`, `latitude`, `longitude`, `timestamp`, `power_status`, případně `speed` a `accuracy`. Server po `success:true` dovolí klientovi smazat dávku.

Autentizace a chyby (APK):

- 401/403: neplatná nebo chybějící session; APK musí zneplatnit lokální session (`logout broadcast`).
- 404 `registered=false`: zařízení není registrováno; klient zastaví službu a vyžádá novou registraci.
- 500+: retry s exponential backoff; data v lokální DB zůstávají.

#### 2.2.1.4 Validace vstupních dat (Express Validator)

Pro všechny vstupy (HW i APK) platí validace přes `express-validator`. Chybné payloady jsou odmítnuty kódem 400 ještě před zpracováním.

```

#HW_REQUEST
{
    "device_id": "ABC1234567",
    "client_type": "HW",
    "power_status": "ON",
}
#SERVER_RESPONSE
{
    "registered": true,
    # if registered true :
    "config": {
        "interval_gps": 60,
        "interval_send": 3,
        "satellites": 7,
        "mode": "batch",
    },
    "power_instruction": "TURN_OFF" | "NONE"
}

#INPUT
{
    "device": "ABC1234567",
    "latitude": 50.12345,
    "longitude": 14.45678,
    "speed": 35.6,
    "satellites": 8,
    "timestamp": "2025-11-07T10:20:00Z",
    "power_status": "OFF", # pokud se změnil
}

```

(a) HW Handshake tok

(b) HW/APK Input tok

### 2.2.1.5 Dokumentace API (Swagger)

Swagger (swagger-jsdoc + swagger-ui-express) generuje strojově čitelnou dokumentaci z komentářů controllerů a je dostupný na /api-docs.

Obrázek 2.3: Ukázka Swagger dokumentace API

## 2.2.2 Správa uživatelů a Autorizace

Používáme uživatelské účty s možností registrace, přihlášení a správy jejich údajů. Pro bezpečnou autentizaci a autorizaci je implementován middleware `authorization.js` (popsáno výše).

### 2.2.2.1 Autorizace a role (`authorization.js`)

Middleware `authorization.js` vynucuje přihlášení, odděluje běžné uživatele, administrátory a blokuje nevhodné akce ROOT účtu, a zároveň autentizuje HW/APK zařízení na základě `device_id`:

- `isAuthenticated/isApiAuthenticated`: vyžadují platnou session pro web i API.
- `isUser/isRoot/isNotRootApi`: směrují podle role, zakazují ROOT na běžných API a naopak.
- `authenticateDevice`: pro HW/APK čte `device_id`, ověří vazbu na User a naplní `req.device`, `req.user`, `req.clientType`.

Role / middleware	Webové routy	API routy
<code>isAuthenticated</code>	Zobrazí HTML nebo přesměruje na <code>/login</code> .	Vrací 401 JSON, neprovádí přesměrování.
<code>isUser</code>	Povolen běžný uživatel, root přesměrován do administrace.	Kombinuje se s <code>isNotRootApi</code> pro blokaci root na uživatelských API.
<code>isRoot</code>	Otevřá <code>/administration</code> .	Chrání <code>/api/admin/*</code> .
<code>authenticateDevice</code>	N/A	Povinné pro <code>/api/devices/input handshake</code> , váže <code>device_id</code> k uživateli.

Tabulka 2.3: Rychlá orientace v rolích a middleware

### 2.2.2.2 Session a cookie politika

Session jsou spravovány pomocí `express-session`. Klíč `SESSION_SECRET` se načítá z prostředí, cookie je `httpOnly` s `maxAge` 6 hodin. Flag `secure` se zapíná, pokud je `NODE_ENV=using_ssl`, aby se cookie přenášela jen přes HTTPS. Webové routy používají `isAuthenticated` a případně přesměrování, API vrací JSON přes `isApiAuthenticated`.

```

const isAuthenticated = (req, res, next) => {
  if (hasUserSession(req)) {
    return next();
  }
  req.flash?('error', 'Please log in to view this page.');
  res.redirect('/login');
};

```

Obrázek 2.4: příklad jedné z funkcí v authorization.js

### 2.2.2.3 Správa identit (Passport.js)

Knihovna passport.js zajišťuje flexibilní autentizaci. V systému jsou implementovány tři strategie:

- **Local Strategy:** Přihlášení pomocí emailu a hesla.
- **Google Strategy:** OAuth 2.0 přihlášení přes Google účet (passport-google-oauth20).
- **GitHub Strategy:** OAuth 2.0 přihlášení přes GitHub účet (passport-github2).

Po úspěšném přihlášení je uživatelská relace (session) uložena a identifikována pomocí cookie.

```

passport.use(new GoogleStrategy({
  clientID: process.env.GOOGLE_CLIENT_ID,
  clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  callbackURL: process.env.GOOGLE_CALLBACK_URL,
  scope: ['profile', 'email']
}, async (accessToken, refreshToken, profile, done) => { ... });

passport.use(new GitHubStrategy({
  clientID: process.env.GITHUB_CLIENT_ID,
  clientSecret: process.env.GITHUB_CLIENT_SECRET,
  callbackURL: process.env.GITHUB_CALLBACK_URL,
  scope: ['user:email']
}, async (accessToken, refreshToken, profile, done) => { ... });

```

Obrázek 2.5: Passport.js autentizační strategie

### 2.2.2.4 Hashování hesel (Bcrypt.js)

Hesla uživatelů nejsou nikdy ukládána v otevřené podobě. Při registraci je heslo prohnáno hashovací funkcí bcrypt se solí (salt). Při přihlášení se zadанé heslo zahashuje a porovná s uloženým hashem. To chrání uživatele i v případě úniku databáze.

## 2.2.2.5 ROOT účet

Z bezpečnostních i provozních důvodů systém startuje s hardcoded administrátorským účtem ROOT. Tento účet má vlastní administraci, která umožnuje přímí přístup do DB a manipulaci s jejími daty.

The screenshot shows the 'LOTR System' admin interface. It has three main sections:

- Users (3):** A table with columns: ID, Username, Email, Verified, Provider, Provider ID, Pending Email, Password Hash, Devices, Created At, Actions. It lists three users: 'spora' (verified, local provider), 'iotr' (verified, google provider), and 'root' (verified, local provider).
- Devices (4):** A table with columns: Device ID, Name, Owner, Power Status, Instruction, GPS Interval, Send Interval, Satellites, Mode, Last Seen, Geofence, Actions. It lists four devices: 'Google sdk\_phone64\_x86\_64', 'NED-0M\_A7670E', 'samsung SM-S901B', and 'Samsung'.
- Latest Locations (50):** A table with columns: Device ID, Latitude, Longitude, Speed, Altitude, Accuracy, Satellites, Timestamp. It lists 50 location entries for device 'A7F8B0902' at coordinates (37.421998, -122.084000) with speed 0.00 and accuracy 5.00.

Obrázek 2.6: Administrátorský panel

## 2.2.2.6 E-mailová komunikace (Nodemailer)

Pro interakci s uživatelem mimo webové rozhraní slouží knihovna nodemailer. Využívá se v následujících scénářích:

- Verifikace emailu:** Po registraci je odeslán unikátní kód pro ověření existence emailové schránky.
- Reset hesla:** Odeslání odkazu pro obnovu zapomenutého hesla.
- Bezpečnostní alerty:** Upozornění uživatele, pokud jeho zařízení opustí nastavenou bezpečnou zónu (Geofence).

## 2.3 PREZENTAČNÍ VRSTVA (FRONTEND)

Frontendová část aplikace je navržena s důrazem na jednoduchost a rychlou odezvu. Kombinuje server-side rendering (SSR) pro základní strukturu stránky a klientský JavaScript pro dynamické aktualizace dat v reálném čase.



Obrázek 2.7: E-mailové scénáře: transakční zprávy a bezpečnostní upozornění

### 2.3.1 Šablonovací systém EJS a struktura pohledů

Pro generování HTML stránek na straně serveru je použit šablonovací systém \*\*EJS (Embedded JavaScript)\*\*. Ten umožňuje vkládat JavaScriptovou logiku přímo do HTML kódu. Struktura pohledů je modularizována pomocí tzv. *partials* (dílčích šablon), což zabraňuje duplicitě kódu. Typická stránka se skládá z:

- `_head.ejs`: Meta tagy, importy CSS stylů a externích knihoven.
- `_navbar.ejs`: Navigační lišta s odkazy a informacemi o přihlášeném uživateli.
- **Obsah stránky**: Unikátní obsah pro daný pohled (např. `index.ejs` pro mapu, `settings.ejs` pro nastavení).
- `_footer.ejs`: Patička stránky a importy JavaScriptových souborů.

Data z backendu (např. seznam zařízení, chybové hlášky) jsou do šablon předávána při vykreslování v controlleru.

### 2.3.2 Vizualizace dat a mapové podklady

Klíčovou funkcí frontendu je vizualizace polohy trackerů na mapě. Pro tento účel byla zvolena open-source knihovna \*\*Leaflet.js\*\*, která je lehká a flexibilní. Jako zdroj mapových podkladů (dlaždic) slouží služba \*\*OpenStreetMap\*\*.

#### 2.3.2.1 Dynamická aktualizace polohy (AJAX)

Aby uživatel viděl aktuální polohu zařízení bez nutnosti obnovovat celou stránku, využívá aplikace technologii \*\*AJAX\*\* (Asynchronous JavaScript and XML) prostřednictvím moderního Fetch API.

1. Při načtení stránky se inicializuje mapa a vykreslí se poslední známé polohy.

2. Klientský skript (index.js) spouští v pravidelných intervalech (nastaveno na 5 sekund) požadavek na API endpoint /api/devices/coordinates.
3. Server vrátí aktuální data ve formátu JSON.
4. JavaScript na klientovi porovná nová data s existujícími markery na mapě a aktualizuje jejich pozici, případně obsah informačních bublin (tooltipů).

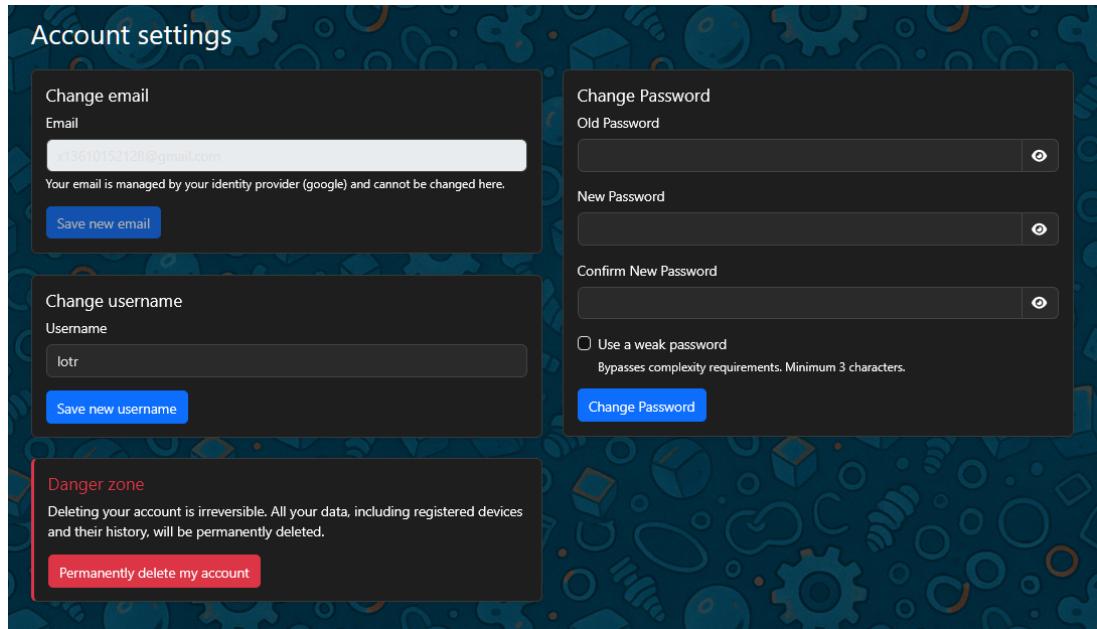
The screenshot displays a web-based application interface for managing a device. On the left, there's a sidebar with sections for Device List, Device Settings (Mode: Simple, GPS Fix Interval: 00:00:00, Minimum Satellites: 7), Power Control (Power OFF button), and Device Information (Device ID: 3333337789, Device Type: HW, Registration Date: 05. 12. 2025 15:41:26). The main area is divided into two parts: 'Live Map' and 'Location History'. The 'Live Map' shows a map of Prague with several blue dots representing device locations. A tooltip for one dot provides details: Time: 05. 12. 2025 15:49:19, Speed: 50.00 km/h, Altitude: 210.00 m. Another tooltip for a cluster of three points shows: Cluster (3 points), From: 05. 12. 2025 15:49:19, To: 05. 12. 2025 15:49:19, To: 05. 12. 2025 15:46:24. The 'Location History' table lists the following data:

Time	Latitude	Longitude	Speed	Altitude	Accuracy	Satellites
05. 12. 2025 15:49:19 - 05. 12. 2025 15:49:19	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05. 12. 2025 15:46:24 - 05. 12. 2025 15:46:24	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05. 12. 2025 15:46:24	50.098040	14.430760	50.00 km/h	210.00 m	5.00 m	12
05. 12. 2025 15:45:17 - 05. 12. 2025 15:45:17	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05. 12. 2025 15:45:17	50.098040	14.430760	50.00 km/h	210.00 m	5.00 m	12
05. 12. 2025 15:44:38 - 05. 12. 2025 15:44:38	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05. 12. 2025 15:44:38	50.098040	14.430760	50.00 km/h	210.00 m	5.00 m	12

Obrázek 2.8: Webové rozhraní - Správa zařízení

## 2.4 NASAZENÍ A PROVOZ (DEPLOYMENT)

Pro zajištění konzistentního běhového prostředí a snadného nasazení na různé servery (vývojový, produkční) je celý systém kontejnerizován pomocí technologie **Docker**. Kontejnerizace eliminuje problémy typu "u mě to funguje", protože aplikace si nese veškeré své závislosti (Node.js runtime, knihovny) s sebou.



Obrázek 2.9: Webové rozhraní - Settings

### 2.4.1 Docker a kontejnerizace

Obraz (image) serverové aplikace je definován v souboru `Dockerfile`. Jako základ je použit oficiální odlehčený obraz `node:24-slim`, který minimalizuje velikost výsledného kontejneru. Proces sestavení obrazu zahrnuje:

1. Nastavení pracovního adresáře na `/app`.
2. Kopírování definice závislostí (`package.json`) a jejich instalace pomocí `npm install`.
3. Kopírování zdrojových kódů aplikace.
4. Expozice portu 5000, na kterém server naslouchá.
5. Definice spouštěcího příkazu `node server.js`.

```
FROM node:24-slim
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5000
CMD ["node", "server.js"]
```

Obrázek 2.10: Ukázka Dockerfile pro serverovou aplikaci

### 2.4.2 Orchestrace kontejnerů (Docker Compose)

Protože systém vyžaduje ke svému běhu nejen aplikační server, ale i databázi, využíváme nástroj Docker Compose pro definici a spouštění více kontejnerů současně. Konfigurace je uložena v souboru `docker-compose.yml`, který definuje dvě hlavní služby:

- **app:** Samotná Node.js aplikace. Služba je nakonfigurována tak, aby čekala na plné spuštění databáze (`depends_on`). Skrze proměnné prostředí (environment variables) jsou předány konfigurační údaje jako přístup k DB, tajné klíče pro sessions a nastavení CORS.
- **mysql:** Databázový server MySQL verze 8.0. Data jsou ukládána do trvalého svazku (volume) `mysql-data`, což zajišťuje, že data přežijí i restart nebo smazání kontejneru. Při prvním spuštění se automaticky provede inicializační skript `init-db.sql`.

Obě služby běží ve společné virtuální síti `gps-network`, což jim umožňuje vzájemnou komunikaci pomocí názvů služeb (hostname), aniž by musely být vystaveny do veřejného internetu (s výjimkou aplikačního portu 5000).

**(a) Registrace**

**Registration**

Username  
lotr

Email  
stepan.balner@proton.me

Password  
••••••••

Password must be at least 6 characters long and contain at least one uppercase letter, one number, and one special character.

Confirm Password  
••••••••

Use weak password (min. 3 characters)

**Register**

Do you already have an account? [Login](#)

**(b) Přihlášení**

**Login**

Username or Email

Password

[Forgot Password?](#)

**Login**

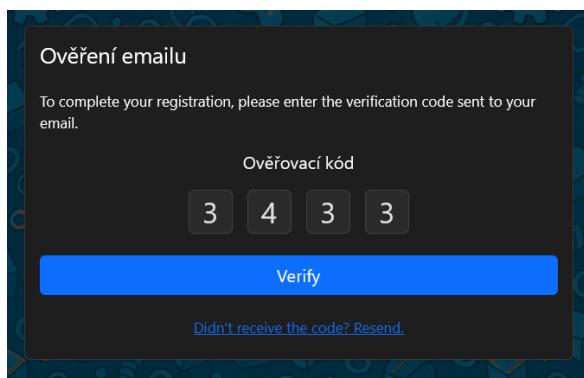
OR

**Continue with Google**

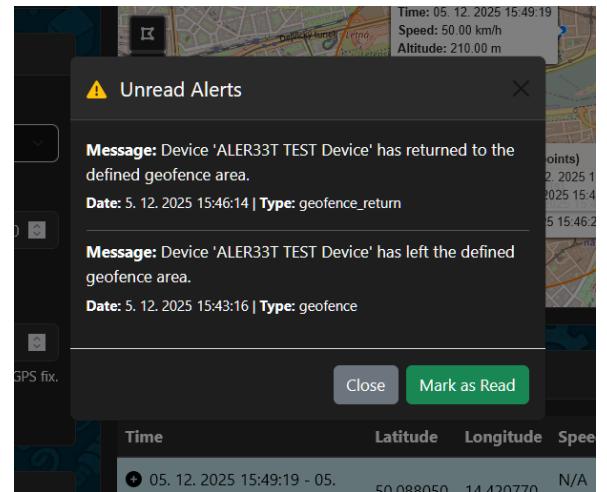
**Continue with GitHub**

Don't have an account? [Register](#)

Obrázek 2.11: Webové rozhraní: proces registrace a přihlášení

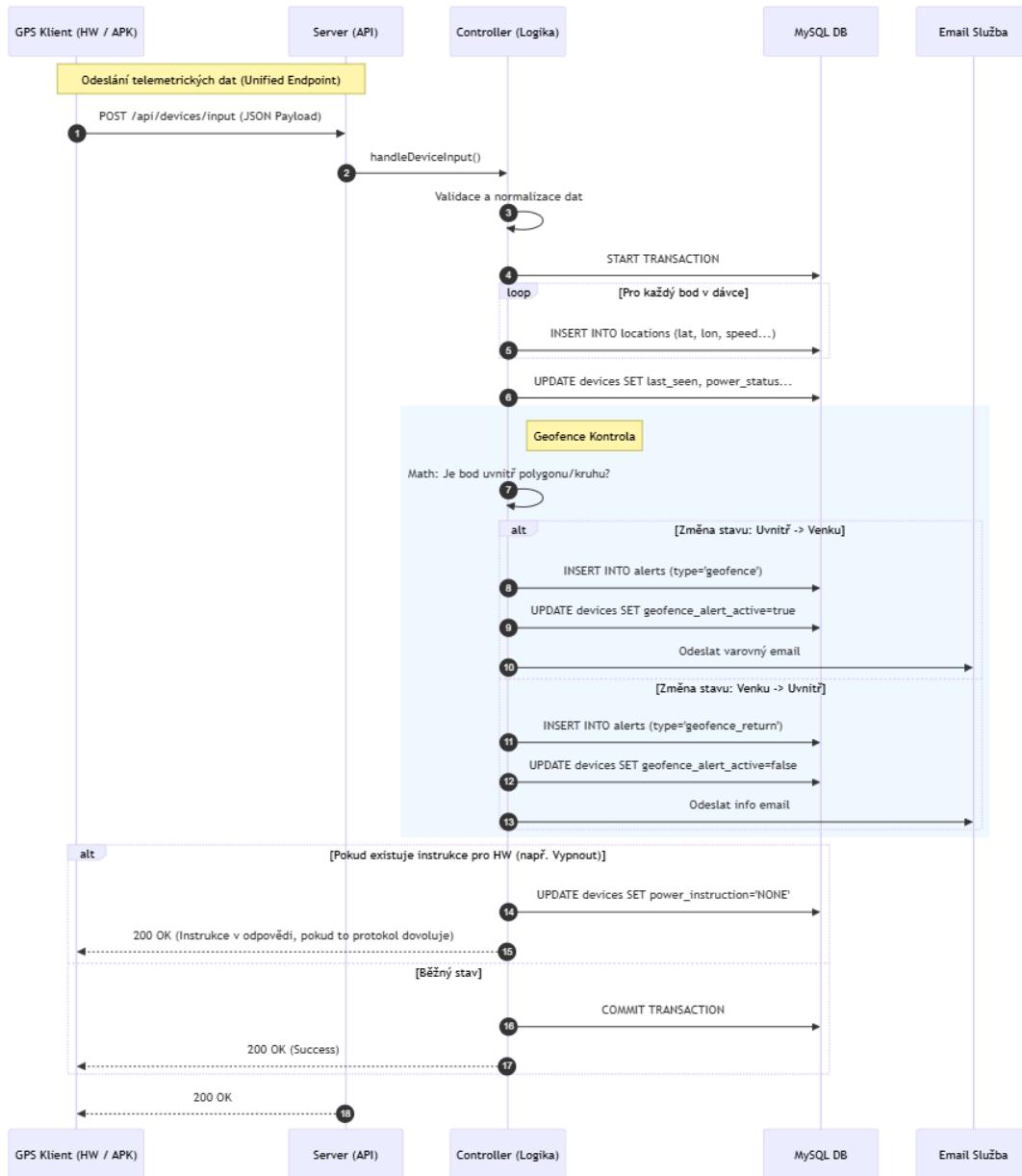


(a) Verifikace e-mailu

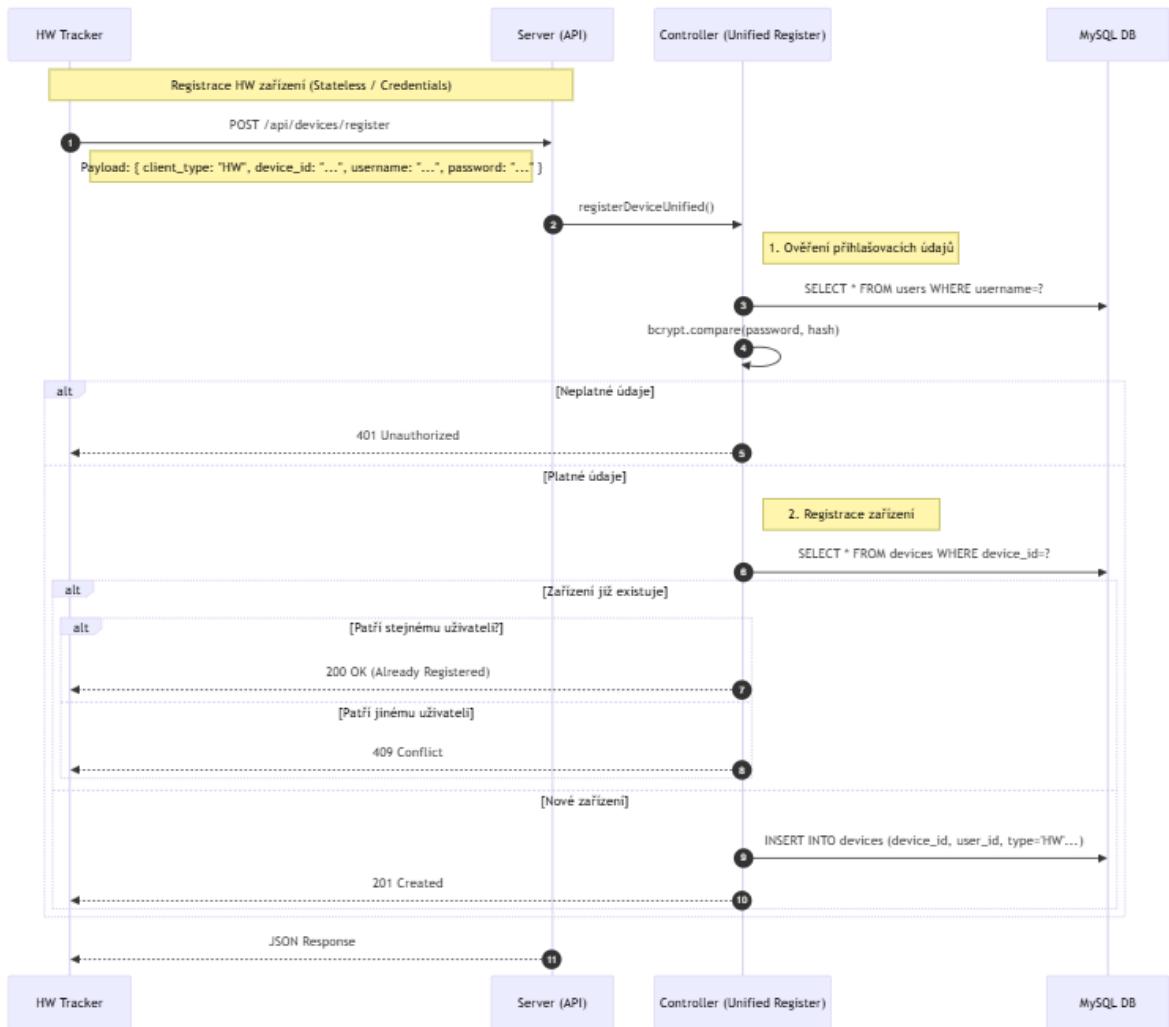


(b) Bezpečnostní upozornění (modal)

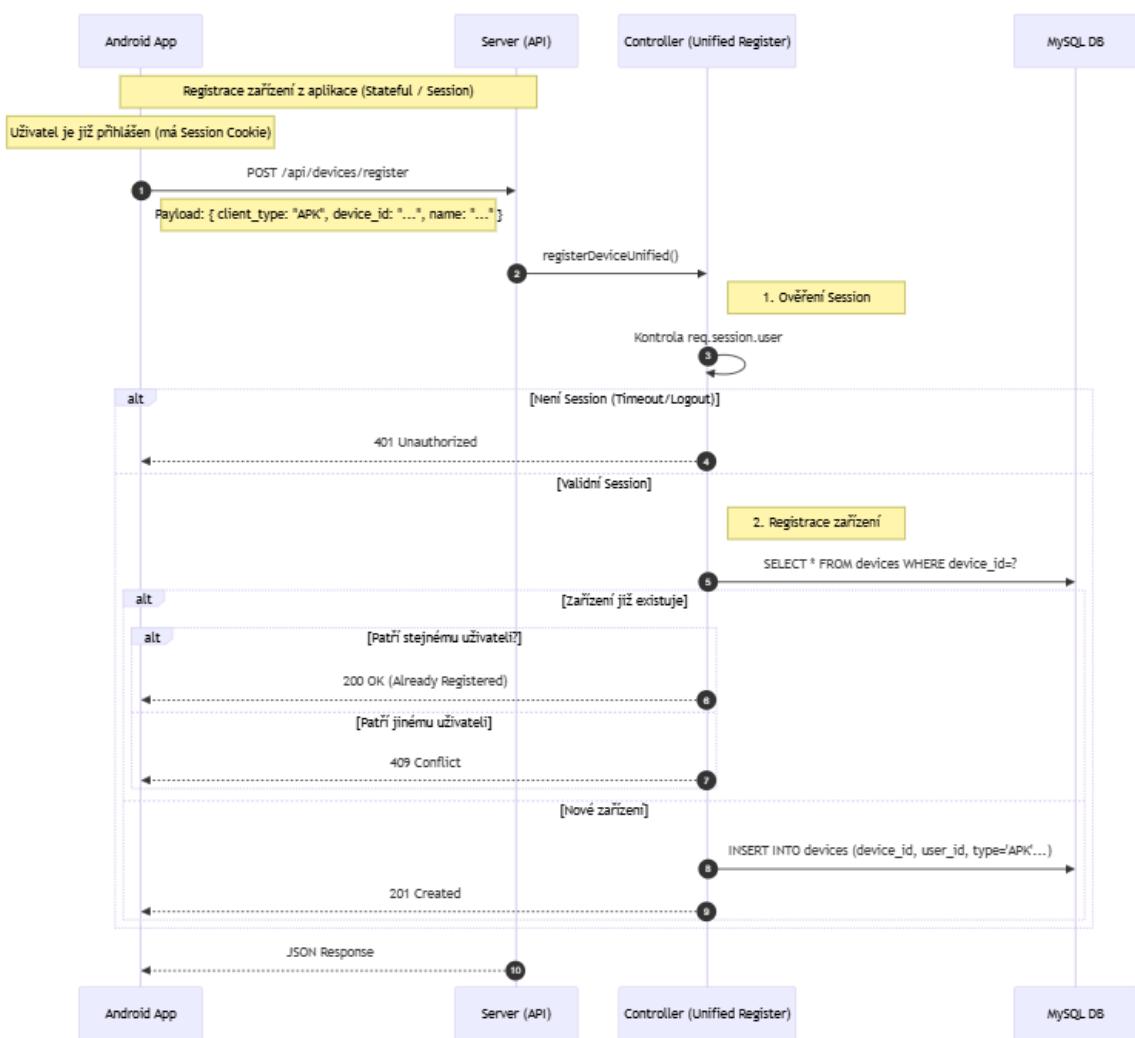
Obrázek 2.12: Webové rozhraní: verifikace e-mailu a bezpečnostní upozornění



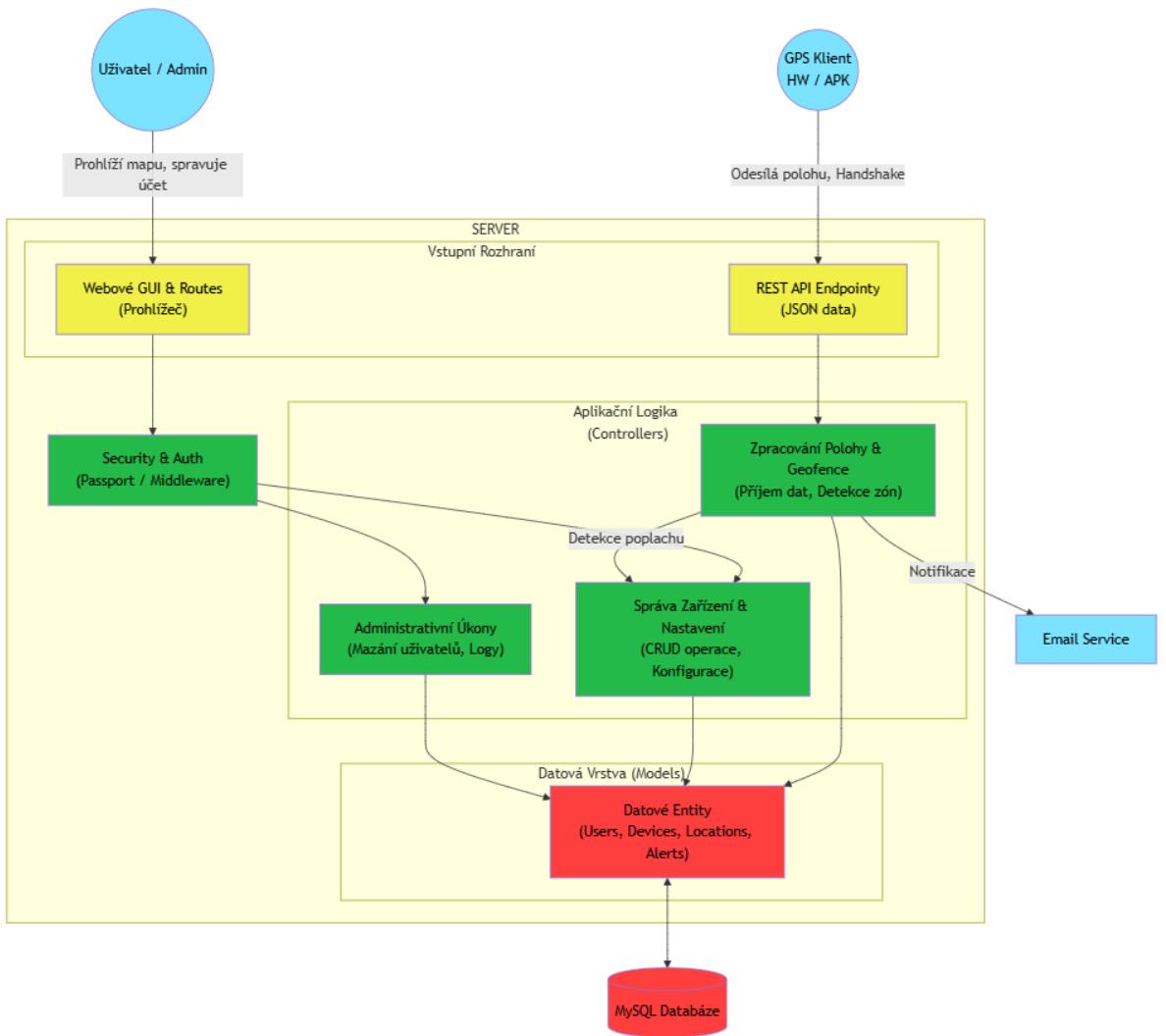
Obrázek 2.13: Sekvenční diagram: tok sledování zařízení (handshake, input, aktualizace polohy)



Obrázek 2.14: Sekvenční diagram: registrace HW zařízení (ESP32) do systému



Obrázek 2.15: Sekvenční diagram: registrace APK klienta (Android) a navázání session



Obrázek 2.16: Use-case diagram systému LOTR: aktéři a hlavní případy použití



# **3 APLIKACE PRO ANDROID**

## **3.1 KONCEPT A CÍLE APLIKACE**

### **3.1.1 Náhrada hardwarového trackeru**

### **3.1.2 Princip Offline-First a odolnost proti výpadkům**

## **3.2 TEORETICKÁ VÝCHODISKA A POUŽITÉ TECHNOLOGIE**

Vývoj pro mobilní platformu Android s sebou nese specifické výzvy, zejména v oblasti správy životního cyklu aplikací a úspory energie. Volba správných nástrojů je proto klíčová pro stabilitu výsledného řešení.

### **3.2.1 Jazyk Kotlin a asynchronní programování (Coroutines)**

**Kotlin** je staticky typovaný programovací jazyk běžící na virtuálním stroji Java (JVM). Od roku 2019 je společností Google doporučován jako preferovaný jazyk pro vývoj Android aplikací. Oproti starší Javě přináší moderní prvky, jako je *Null Safety* (typový systém odlišuje hodnoty, které mohou být null), což eliminuje časté pády aplikace na chybu `NullPointerException`. Pro řešení asynchronních operací (sítová volání, databázové dotazy) Kotlin využívá koncept **Coroutines**. Na rozdíl od klasických vláken (Threads), která jsou náročná na systémové zdroje, jsou korutiny "lehká vlákna". Umožňují psát asynchronní kód sekvenčním stylem (pomocí klíčového slova `suspend`), což výrazně zvyšuje čitelnost a usnadňuje správu chyb.

### **3.2.2 Systémové komponenty (Foreground Service, WorkManager)**

Operační systém Android agresivně ukončuje aplikace běžící na pozadí, aby šetřil baterii. Pro aplikace typu tracker je však nutné běžet nepřetržitě.

- **Foreground Service (Služba na popředí):** Jedná se o službu, která dává systému na-

jevo, že vykonává pro uživatele důležitou činnost. Musí zobrazovat trvalou notifikaci ve stavovém řádku. Díky tomu ji systém neukončí ani při nedostatku paměti.

- **WorkManager:** Pro úlohy, které nejsou okamžité, ale musí se zaručeně provést (např. odeslání dat na server, až bude dostupný internet), se využívá knihovna WorkManager. Ta inteligentně plánuje spuštění úloh s ohledem na stav baterie a sítě.

### 3.2.3 Lokální persistence (Room Database)

Pro ukládání strukturovaných dat přímo v zařízení se využívá databáze **SQLLite**, která je součástí Androidu. Práce s ní pomocí surových SQL dotazů je však náchylná k chybám. Proto byla zvolena knihovna **Room**, která slouží jako abstrakční vrstva (ORM) nad SQLite. Umožňuje definovat databázové tabulky jako datové třídy (Entity) a přístup k nim definovat pomocí rozhraní (DAO - Data Access Object). Room automaticky kontroluje správnost SQL dotazů již při komplikaci aplikace.

### 3.2.4 Zabezpečené úložiště (EncryptedSharedPreferences)

Ukládání citlivých dat (přihlašovací tokeny, API klíče) do běžných textových souborů (SharedPreferences) představuje bezpečnostní riziko, zejména na zařízeních s root oprávněním (tzv. rootnutých). Knihovna **EncryptedSharedPreferences** řeší tento problém tím, že data před uložením automaticky šifruje. Šifrovací klíče jsou uloženy v **Android Keystore System**, což je speciální hardwarově chráněné úložiště, ke kterému nemá přístup ani samotný operační systém, natož malware.

## 3.3 ARCHITEKTURA APLIKACE

### 3.3.1 Vrstevnatá architektura a reaktivní model (StateFlow)

### 3.3.2 Vlastní implementace HTTP klienta

## 3.4 IMPLEMENTACE KLÍČOVÝCH FUNKCÍ

### 3.4.1 Sběr polohy na pozadí (LocationService)

### 3.4.2 Dávková synchronizace dat (SyncWorker)

**3.4.3 Řízení spotřeby a vzdálené vypínání (PowerController)**

**3.4.4 Handshake protokol a konfigurace**

## **3.5 UŽIVATELSKÉ ROZHRANÍ**

**3.5.1 Hlavní ovládací panel (Dashboard)**

**3.5.2 Diagnostická konzole pro terénní testování**



## ZÁVĚR A ZHODNOCENÍ

Seznam nedokonalostí či nedodělků je docela dlouhý. Jako hlavní nedostatek vidím absenci 3D tisknutého pouzdra pro HW tracker. V počátcích vývoje jsem sice vymodeloval prototyp, ale ukázalo se, že s tím jak byly postupně přidávány prvky nebo měněny stávající, nestačil model udržovat krok, jediná použitelná část zůstal držák na baterii + její nabíječku, ten se ukázal jako užitečný. Dále celý základní princip API a identifikace mezi zařízeními a serverem je dosti primitivní a náchylný k chybám. V budoucnu by bylo vhodné přejít na nějaký propracovanější systém autentizace a autorizace zařízení, místo pouhého ID a uživatelských údajů (je to takto dosti jednoduché na falšování požadavků). Také možnosti řazení nebo třídění dat v uživatelském rozhraní chybí, a tak bych mohl pokračovat...

Co však dokončeno bylo, alespoň základně či přízemně, je samotné jadro systému. I když bez pokročilého zabezpečení tak API funguje a zařízení dokáží se serverem komunikovat a mají schopnosti se tak či onak vypořádat s vypadky sítě (cachování) a HW-tracker přestože s obtížemi by přece jenom šlo naložit do auta a sledovat jeho pohyb do doby vybití baterie (přes snahy o nízkou spotřebu však výdrž není úplně optimální). Nejlépe asi použít jako GPS alarm, kdy by se tracker hlasil s periodou cca. 1 dne a v případě změny by uživatel dostal hlášení. Celkově tedy systém funguje v urovni možností a schopností prototypu, ale pro reálné nasazení by bylo potřeba ještě hodně práce a vylepšení.



## **SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ**



## LITERATURA

- [1] ESPRESSIF SYSTEMS. *ESP32 Technical Reference Manual* [Online]. 2024. Dostupné z: <https://www.espressif.com/en/support/documents/technical-documents>
- [2] SIMCOM WIRELESS SOLUTIONS. *A7600 Series AT Command Manual* [Online]. V1.01, 2021.
- [3] U-BLOX AG. *NEO-6 u-blox 6 GPS Modules Data Sheet* [Online]. 2011. Dostupné z: <https://www.u-blox.com>
- [4] AMAZON WEB SERVICES. *FreeRTOS Kernel Developer Guide* [Online]. Dostupné z: <https://www.freertos.org>
- [5] SHYMANSKYY, Volodymyr. *TinyGSM: A small Arduino library for GSM modules* [Software]. GitHub, 2023. Dostupné z: <https://github.com/vshymanskyy/TinyGSM>
- [6] OPENJS FOUNDATION. *Node.js Documentation* [Online]. Dostupné z: <https://nodejs.org/en/docs/>
- [7] EXPRESS. *Express - Node.js web application framework* [Online]. Dostupné z: <https://expressjs.com>
- [8] SEQUELIZE. *Sequelize v6 Reference Manual* [Online]. Dostupné z: <https://sequelize.org>
- [9] AGAFONKIN, Vladimir. *Leaflet: an open-source JavaScript library for mobile-friendly interactive maps* [Online]. Dostupné z: <https://leafletjs.com>
- [10] JETBRAINS. *Kotlin Documentation* [Online]. Dostupné z: <https://kotlinlang.org/docs/home.html>
- [11] GOOGLE DEVELOPERS. *Android Developers: Guide to App Architecture* [Online]. Dostupné z: <https://developer.android.com/jetpack/guide>
- [12] GOOGLE DEVELOPERS. *Android WorkManager* [Online]. Dostupné z: <https://developer.android.com/topic/libraries/architecture/workmanager>
- [13] DOKULIL Jakub. *Šablona pro psaní SOČ v programu LATEX* [Online]. Brno, 2020. Dostupné z: [https://github.com/Kubiczek36/SOC\\_sablon](https://github.com/Kubiczek36/SOC_sablon)

- [14] GOOGLE DEEPMIND. *Gemini 3: Multimodal Generative AI Model* [Online]. 2025.  
Asistoval při generování a strukturování dokumentace.

# Seznam obrázků

1.1	NMEA sentence . . . . .	6
1.2	LilyGO T-Call V1.5 - integrované řešení ESP32 + LTE modem . . . . .	7
1.3	Modul GPS a zapojení pro řízení napájení . . . . .	8
1.4	Baterie + nabíjecí modul TP4056 a step-up měnič MT3608 již v pouzdře . . . . .	9
1.5	Osazená deska . . . . .	9
1.6	Power Latch modul - Power Modul . . . . .	9
1.7	Blokové schéma zapojení hardwarového trackeru [easyEDA] . . . . .	10
1.8	Schéma power-modul [easyEDA] . . . . .	10
1.9	funkce power_init() . . . . .	13
1.10	funkce graceful_shutdown() . . . . .	14
1.11	funkce enter_deep_sleep() . . . . .	14
1.12	funkce gps_get_fix() . . . . .	15
1.13	funkce fs_init() . . . . .	16
1.14	funkce modem_send_post_request() . . . . .	17
1.15	funkce modem_perform_handshake() . . . . .	17
1.16	funkce modem_initialize() . . . . .	18
1.18	konfigurace Wifi APN pro OTA režim . . . . .	19
1.17	wifi_apn . . . . .	19
1.19	log z OTA režimu . . . . .	21
1.20	OTA režim - webové rozhraní . . . . .	21
1.21	Usecase diagram . . . . .	22
1.22	sekvenční diagram životního cyklu . . . . .	23
1.23	"vnitřnosti" zařízení . . . . .	24
2.1	ER diagram databázových modelů . . . . .	28
2.3	Ukázka Swagger dokumentace API . . . . .	31
2.4	příklad jedné z funkcí v authorization.js . . . . .	33
2.5	Passport.js autentizační strategie . . . . .	33
2.6	Administrátorský panel . . . . .	34
2.7	E-mailové scénáře: transakční zprávy a bezpečnostní upozornění . . . . .	35
2.8	Webové rozhraní - Správa zařízení . . . . .	36

2.9	Webové rozhraní - Settings . . . . .	37
2.10	Ukázka Dockerfile pro serverovou aplikaci . . . . .	37
2.11	Webové rozhraní: proces registrace a přihlášení . . . . .	38
2.12	Webové rozhraní: verifikace e-mailu a bezpečnostní upozornění . . . . .	39
2.13	Sekvenční diagram: tok sledování zařízení (handshake, input, aktualizace polohy)	40
2.14	Sekvenční diagram: registrace HW zařízení (ESP32) do systému . . . . .	41
2.15	Sekvenční diagram: registrace APK klienta (Android) a navázání session . . . . .	42
2.16	Use-case diagram systému LOTR: aktéři a hlavní případy použití . . . . .	43

## Seznam tabulek

2.1	HW API endpointy . . . . .	29
2.2	APK API endpointy . . . . .	30
2.3	Rychlá orientace v rolích a middleware . . . . .	32