

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 urovni takové kdy kódování tohoto projektu bylo možné i za sníženého nervového vypětí.

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

Tato práce obsahuje návrh a realizaci sledovacího systému LOTR (LOcation TRacking System), který se skládá z fyzického hardwarového zařízení, serverové infrastruktury a (doplňkově) klientské mobilní aplikace.

Hardwarová část je postavena na platformě ESP32 s využitím modemu SIMCOM A7670 pro komunikaci přes síť LTE a modulu GPS pro získávání polohy. Firmware využívá přerušovacího systému a zápisu dat do flash paměti. 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. Doplňkovou součástí systému je také nativní aplikace pro OS Android vyvinutá v jazyce Kotlin, která slouží jako testovací nahradou hardwarového trackeru.

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

Klíčová slova

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

Abstract

This work presents the design and implementation of the LOTR (LOcation TRacking System), consisting of a physical hardware tracker, a server infrastructure, and (optionally) a client mobile application.

The hardware component is built on the ESP32 platform, using the SIMCOM A7670 modem for LTE connectivity and a GPS module for location acquisition. The firmware employs an interrupt-driven approach and writes telemetry to flash memory. The server component, implemented in Node.js with a MySQL database, collects telemetry data, manages users, and exposes a REST API for communication with devices. An additional native Android application written in Kotlin serves as a testing substitute for the hardware tracker.

The outcome is an IoT prototype enabling location tracking, movement history, and remote configuration of connected devices.

Keywords

GPS, IoT, tracking system, ESP32, LTE, Node.js, MySQL, Android, Kotlin, REST API

Obsah

ÚVOD	3
1 FYZICKÉ ZAŘÍZENÍ „TRACKER“	5
1.1 ÚVOD A KONCEPCE	5
1.2 POUŽITÉ TECHNOLOGIE	5
1.3 NÁVRH HARDWARE - VÝBĚR KOMPONENT	6
1.4 IMPLEMENTACE FIRMWARE	10
1.5 KOMUNIKACE A DATA	15
1.6 SEZNAM SOUČÁSTEK + OBRÁZKY	19
2 SERVEROVÁ ČÁST	23
2.1 ÚVOD A KONCEPCE SYSTÉMU	23
2.2 NÁVRH A IMPLEMENTACE BACKENDU	24
2.3 PREZENTAČNÍ VRSTVA (FRONTEND)	31
2.4 NASAZENÍ - DOCKER	33
3 APLIKACE PRO ANDROID	39
3.1 ÚVOD A KONCEPT	39
ZÁVĚR	41
SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ	43
SEZNAM OBRÁZKŮ	48
SEZNAM TABULEK	48

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 (naivní to myšlenka) nebo dát pouze do zásilky 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 nutně dosti poruchového, zařízení se ukázal jako vhodný nápad. Existovala možnost vytvořit opravdu pouze "krabici"trackeru s ukládáním dat napevné 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.

Jako méně důležitou, spíše doplňkovou, ale přesto užitečnou součást systému jsem pak viděl vytvoření mobilní aplikace pro systém Android. Avšak z důvodu, že by pak byl projekt rozšířen na až mnoho frontách věnoval jsem ji menší pozornost. Tato aplikace nemá důležitosti jako ostatní komponenty, jelikož byla koncipována jako testování API a pro neznalost Kotlinu většina (přes 90%)jejího kódu byla generována LLM (chcete-li "AI", konkrétně Gemini). Tato dokumentace ji proto nezmiňuje, pouze popisuje uživatelské prostředí.

Proto je tento Dokument rozdělen do tří hlavních kapitol.

1. Popisuje samotné zařízení, jeho návrh a konstrukci i příkladné použití.
2. Popisuje kód serveru, jeho návrh, API a většinu možností uživatelského rozhraní.
3. V krátkosti popisuje Aplikaci pro Android, její uživatelské rozhraní.

1 FYZICKÉ ZAŘÍZENÍ „TRACKER“

1.1 ÚVOD A KONCEPCE

Tracker je základní součást systému, vše ostaní se zrodilo z jeho potřeb o rozšíření, proto je to hlavní část projektu. Jeho hlavním úkolem je pravidelně zaznamenávat polohu a odesílat ji na server, dle požadavků a jednotlivých nastavení uživatelem. Hlavními požadavky na zařízení jsou:

1.2 POUŽITÉ TECHNOLOGIE

1.2.1 Mikrokontroléry (ESP32)

Jako řídící jednotka byl zvolen čip **ESP32** od společnosti Espressif Systems. Jedná se o 32-bitový mikrokontrolér (architektura Xtensa LX6) s integrovanou Wi-Fi a Bluetooth konektivitou. Pro tento projekt je klíčová jeho pokročilá správa napájení, konkrétně režim *Deep Sleep* viz 1.4.4.

1.2.2 Mobilní komunikace a AT příkazy

Pro přenos dat využíváme síť LTE (4G). Komunikace mezi mikrokontrolérem a modemem probíhá po sériové lince (UART) pomocí standardizované sady **AT příkazů** (Hayes command set).

1.2.3 Globální navigační systémy (GPS)

Pro získávání polohy je využit systém GPS. Modul komunikuje s mikrokontrolérem pomocí textového protokolu **NMEA**. Z celého proudu dat jsou parsovány především věty \$GPRMC (Recommended Minimum Specific GPS/TRANSIT Data), které obsahují klíčové údaje o poloze, času a rychlosti.

1.2.4 Operační systémy reálného času (FreeRTOS)

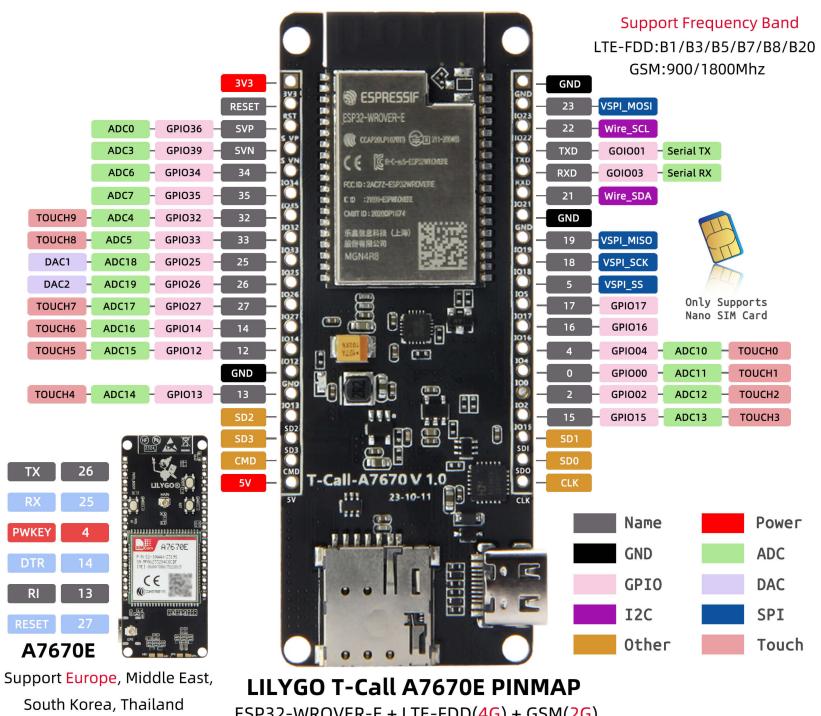
Pro zajištění deterministického chování a správy souběžných procesů (komunikace, sběr dat) je využit operační systém reálného času **FreeRTOS**. Ten umožnuje rozdělit aplikaci do samostatných úloh (Tasks) s definovanou prioritou, které jsou spravovány plánovačem.

1.2.5 Souborový systém LittleFS

Pro ukládání konfigurace a offline dat (cache) je využit souborový systém **LittleFS**. Byl zvolen pro svou odolnost vůči výpadkům napájení (power-loss resilience), což je u bateriového zařízení kritické. Díky mechanismu *Copy-on-Write* nedochází k poškození souborového systému ani při náhlém vypnutí.

1.3 NÁVRH HARDWARE - VÝBĚR KOMPONENT

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



Obrázek 1.1: LilyGO T-Call V1.5 - integrované řešení ESP32 + LTE modem

1.3.1 LilyGO T-Call v1.5

Základem 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 kvůli následujících kritérií:

- **Integrace:** Spojení MCU a modemu na jedné desce eliminuje nutnost složitého propojování (bylo možné použít oddělené desku ESP32 a modem SIM800L, avšak zde by vznikly problémy s napájením. Modem totiž ve špičkách odebírá až 2A.)
- **Konektivita:** Použitý modem SIMCOM A7670 podporuje moderní síť LTE (4G), což zajišťuje lepší pokrytí a nižší latenci než zastaralé 2G moduly (např. SIM800L).

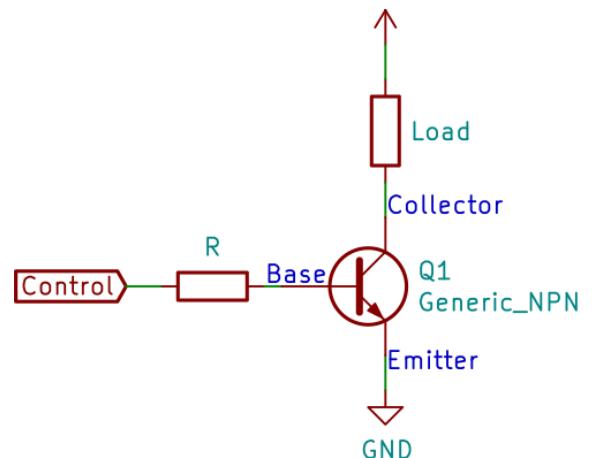
V rané fázi projektu bylo zkoušeno používat desku LilyGO SIM800L (s podporou pouze 2G sítí), avšak kvůli vzniklému zkratu (typická to chyba) a nemožnosti opravy desky ručně (bylo by potřeba mikropájení), přešli jsme na desku T-call v1.5 s modelem A7670 (podpora LTE sítí).

1.3.2 Multi-GNSS L76K modul

Pro získávání polohy byl zvolen externí modul **Multi-GNSS L76K** (hlavně kvůli dostupnosti). 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 1.2: 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

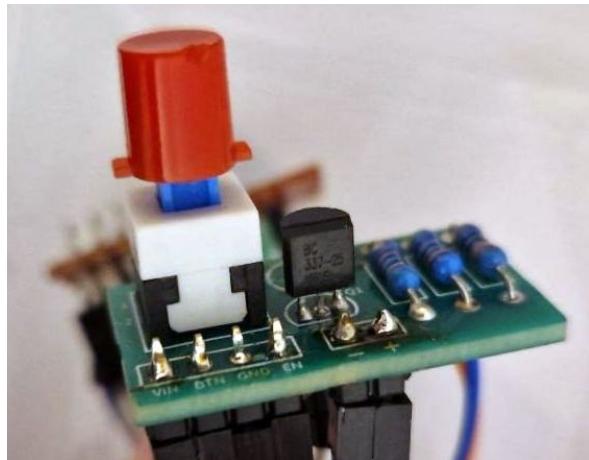
potřebný v zapnutém stavu. Aplikován je princip řízeného GND, sice bylo rozvažováno nad stabilnější možností ovládat VCC (takto se může v krajních případech dostat do modulu napětí), avšak z důvodu absence vhodného PNP tranzistoru bylo zvoleno toto řešení.

1.3.3 Baterie, TP4056 + MT3608

Napájení zajišťuje Li-Ion článek typu 18650 (kapacita 3200 mAh zvolena kvůli ceně), který je dobíjen pomocí modulu s čipem **TP4056**. Pro zvýšení napětí z baterie (3.7V) na úroveň potřebnou pro stabilní provoz desky a GPS modulu(5V) je použit DC-DC step-up měnič **MT3608**.



(a) TP4056 + MT3608 v pouzdře



(b) Osazená deska - Power Latch modul

Obrázek 1.3: Baterie a nabíjecí modul / power latch modul - osazená deska

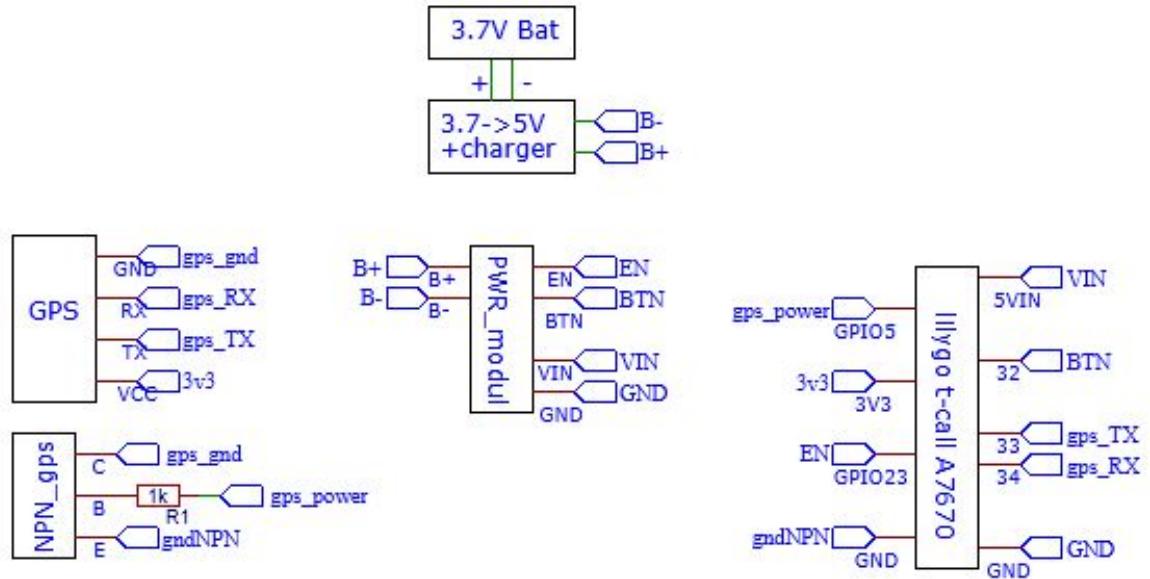
1.3.4 Řízení napájení a Power Latch modul

V případě zařízení 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.).

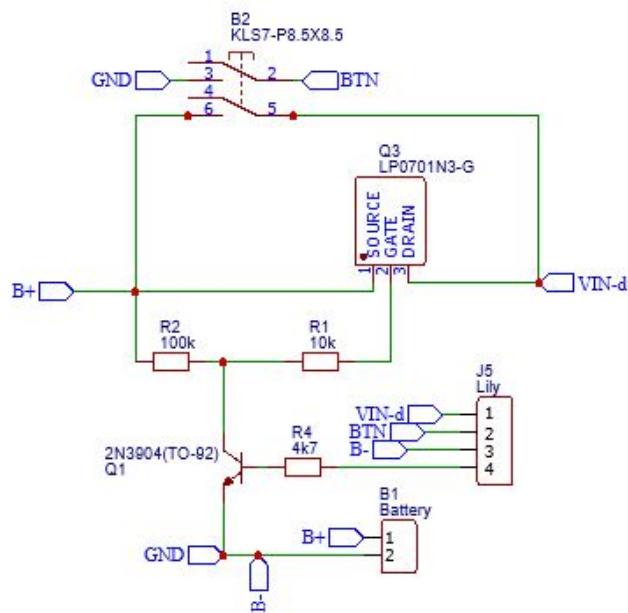
Viz. implementace ve firmware, sekce 1.4.4.

VÝSLEDNÁ SESTAVA - SCHÉMA ZAPOJENÍ - PINY

GPIO piny byly přiřazeny dle dostupnosti na desce, například pro GPS byly použity piny UART1 linky (GPIO16 - RX, GPIO17 - TX) Pro vstup tlačítka byl použit pin GPIO32, který umožňuje vstupu i z režimu Deep Sleep (Ext0 Wakeup). etc.



Obrázek 1.4: Blokové schéma zapojení hardwarového trackeru [easyEDA]



1.4 IMPLEMENTACE FIRMWARE

Software pro mikrokontrolér byl vyvíjen v jazyce C++ s využitím frameworku Arduino. Jako vývojové prostředí bylo zvoleno **Visual Studio Code** s rozšířením **PlatformIO** kvůli největší známosti.

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 dat z GPS modulu.
- **ArduinoJson**: Knihovna pro serializaci a deserializaci JSON objektů, používaná pro komunikaci s API serveru.
- **DeepSleep.h**: Umožnuje použití "hlubokého spánku" v ESP32
- **Wifi.h**: knihovna pro připojení k WiFi síti, použitá v OTA režimu
- **FreeRTOS , LittleFS**: již zmínены v sekci 1.2

1.4.2 Pracovní cyklus

Kvůli potřebě minimalizovat spotřebu baterie používá firmware funkcionalitu ESP32 : **DeepSleep** (hluboký spánek). viz 1.4.4. Protože je použita tato funkcionalita, veškeré činnosti zařízení se odehrávají v `setup()`. Funkce `loop()`, obyčejně užita pro hlavní program, je nevyužita. Veškerý kód je proto umístěn ve funkci `setup()`.

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 používáme na něm operační systém FreeRTOS, který umožňuje 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 `xTaskCreate` vytvoří novou úlohu `ShutdownTask`, která běží na pozadí a čeká na signál z přerušení tlačítka.

1.4.4 Řízení napájení a Power Latch modul

Princip celého systému napájení je následující:

1. Uživatel stiskne tlačítko, čímž přiveze napětí do systému.
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.

```

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(digitalPinToInterrupt(PIN_BTN), on_button_isr, FALLING);
    DBG_PRINTE(F("[POWER] Button interrupt and shutdown task initialized."));
}

```

Obrázek 1.6: funkce power_init()

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 zajišťuje bezpečné vypnutí.

Režim hlubokého spánku (Deep Sleep)

Kromě úplného vypnutí (Power Off) využívá zařízení také režim hlubokého spánku. Tento režim používáme pro cyklické probouzení trackeru (jinak by bylo nutné pokaždé stisknout tlačítko pro zapnutí, což je očividně nepoužitelné). 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).

Následující funkce enter_deep_sleep ukazuje konfiguraci těchto budících zdrojů před uspáním procesoru.

```

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

    // Detach interrupt to prevent any further triggers during shutdown
    detachInterrupt(digitalPinToInterruption(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 1.7: funkce graceful_shutdown()

```

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

    // Detach interrupt before sleeping to prevent issues on wake
    detachInterrupt(digitalPinToInterruption(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 1.8: 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 dle dokumentace) odesílá textová data ve formátu NMEA. Pro zpracování těchto dat to rozumné podoby jsme využili knihovnu **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 detektuje platnou větu a aktualizuje souřadnice, zkонтroluje se, zda jsou splněny podmínky pro platný fix (validní 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_PRINTE(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 1.9: funkce `gps_get_fix()`

1.4.6 Správa datového úložiště (LittleFS)

Jedním z klíčových požadavků byla schopnost pracovat i v oblastech bez signálu mobilní sítě nebo v případě výpadku připojení nebo selhání modemu. Firmware proto implementuje ca-chování dat. Naměřené polohy nejsou odesílány okamžitě, ale jsou nejprve serializovány a uloženy do souboru `/gps_cache.log` v paměti flash (LittleFS). Pro tento učel musel být systém inicializován pomocí vytvoření složky **data/** ve složce firmwaru. Při každém úspěšném připojení k internetu se pak zařízení pokusí odeslat dávku nejstarších záznamů (pravidlo : FIFO - First In, First Out). Teprve po potvrzení serverem (success: true) jsou zaznamy z cache smazány.

Následuje příklad funkce texttfs_init() jež zajišťuje nabootovaní souborového systému.

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

Obrázek 1.10: funkce fs_init()

1.5 KOMUNIKACE A DATA

1.5.1 Komunikační protokol a zabezpečení

Komunikace mezi zařízením a serverem probíhá přes protokol HTTPS (modem SIMCOM A7670 umožnuje použití šifrování narozdíl třeba od SIM800L).

Pro navázání zabezpečeného spojení využívá modem knihovnu SSL/TLS. Následující ukázka kódu z funkce modem_send_post_request ukazuje 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).

Detailní specifikace jednotlivých endpointů, struktura přenášených zpráv a návratové kódy jsou podrobně popsány v kapitole věnované serverové části (viz sekce 2.2.1).

Pro operace a práci s odesílanými a přijímanými JSON daty je využita knihovna **ArduinoJson**. Následující ukázka kódu 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 1.11: 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 1.12: 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 umožňuje využití vysokoúrovňových metod (např. `gprsConnect`, `https_post`) místo čistých AT příkazů.

Přesto je v některých případech nutné používat AT příkazy přímo, například při inicializaci modemu. 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 1.13: funkce `modem_initialize()`

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

1.5.4 OTA režim a konfigurační rozhraní

Pro prvotní registraci, přenastavení konfigurace nebo nahrání nového firmwaru (v BIN formě), zařízení disponuje 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 smyčky OTA režimu.

V OTA režimu se ESP32 přepne do role Wi-Fi přístupového bodu (Access Pointu) s názvem `lotrTrackerOTA_<DeviceID>` (získano z MAC adresy) a spustí webový server na

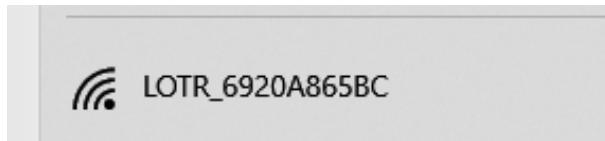
```

// 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 1.15: konfigurace Wifi APN pro OTA režim

adrese 192.168.4.1. Uživatel se může připojit na tuto wifi síť a přes webové rozhraní provádět změny.



Obrázek 1.14: Ukázka Wi-Fi

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 a dostupnost serveru.
- **Firmware Update:** Nahrání nové binární verze firmware přímo z prohlížeče. (s následným restartem zařízení)
- **Clear Cache:** Pokud je potřeba, umožňuje vymazat uložená data v paměti.(např. pokud jsou data poškozena či ve špatném formátu a nelze je odeslat). používá funkce LittleFS.

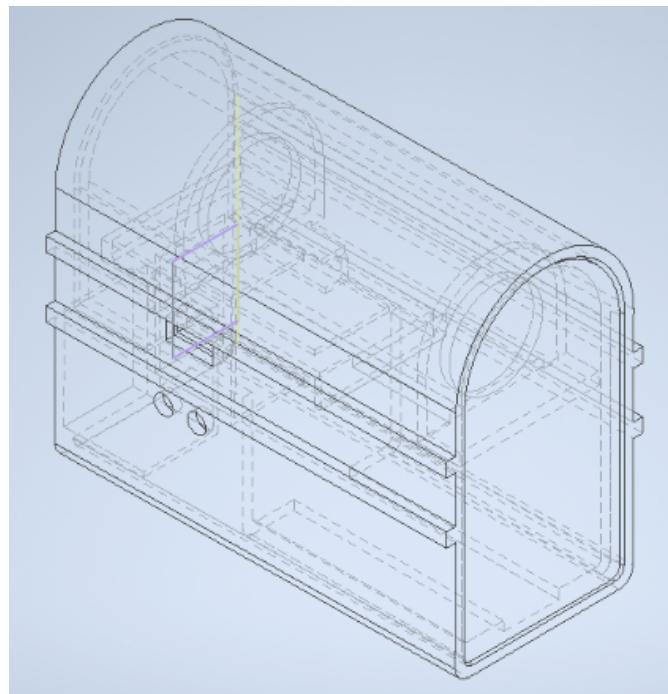
1.5.5 Uložení konfigurace zařízení

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...

1.6 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 (díky minimálnímu toku dat bylo za celé období testování využito do 1MB dat)
- 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 Ω , 100 $k\Omega$)
- KLS 7-P8.0x8.0-0 non lock tlačítko do DPS, 2 póly, ON-(ON)
- LED dioda 3mm zelená



Obrázek 1.16: prototyp krabičky - průhled

(a) registrace

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](#)

(b) OTA režim - webové rozhraní

Device Settings

GPRS Configuration

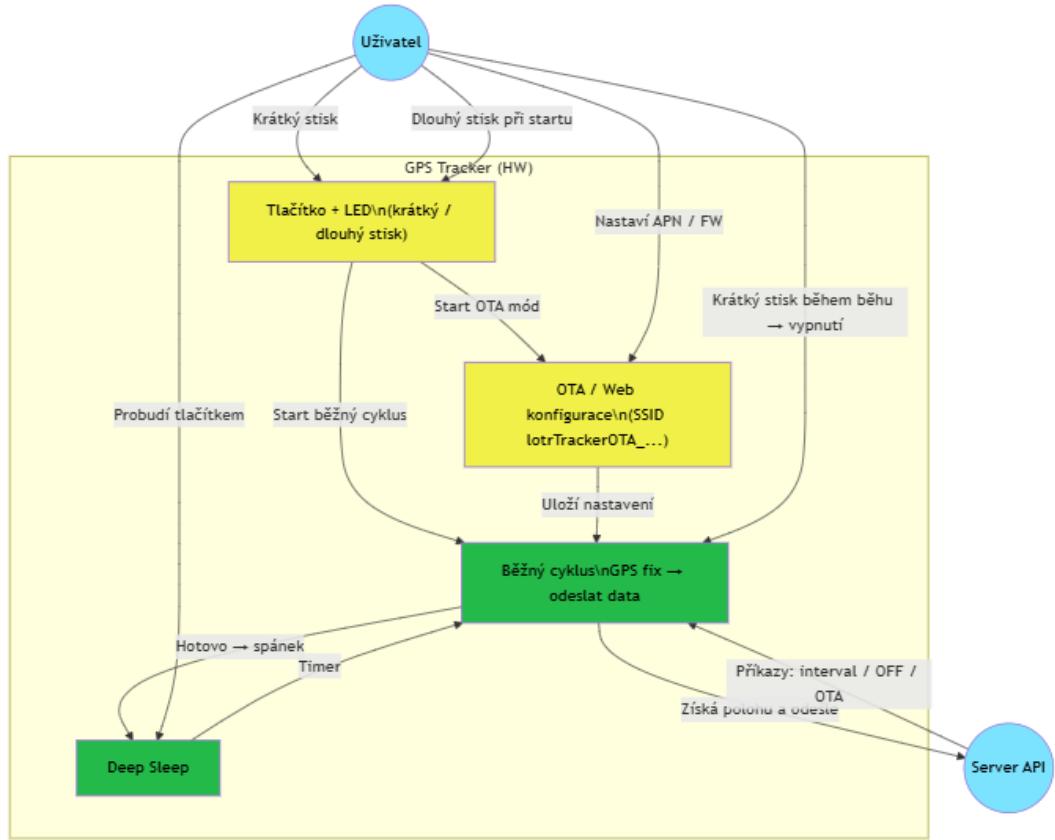
GPRS test connection successful.
APN:

GPRS User:

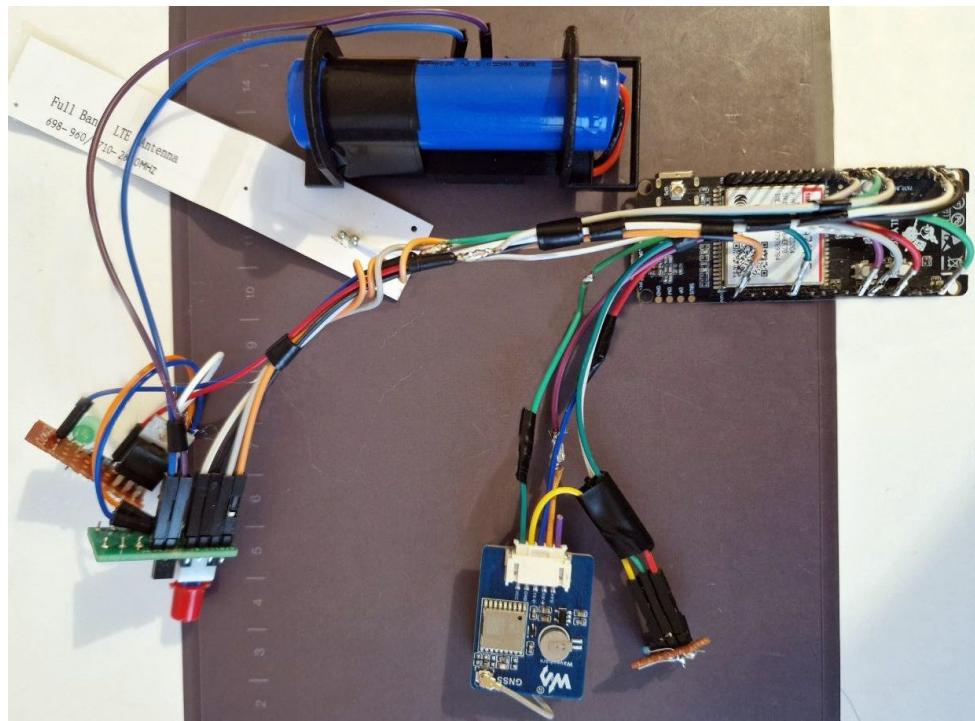
GPRS Password:

Confirm GPRS Password:

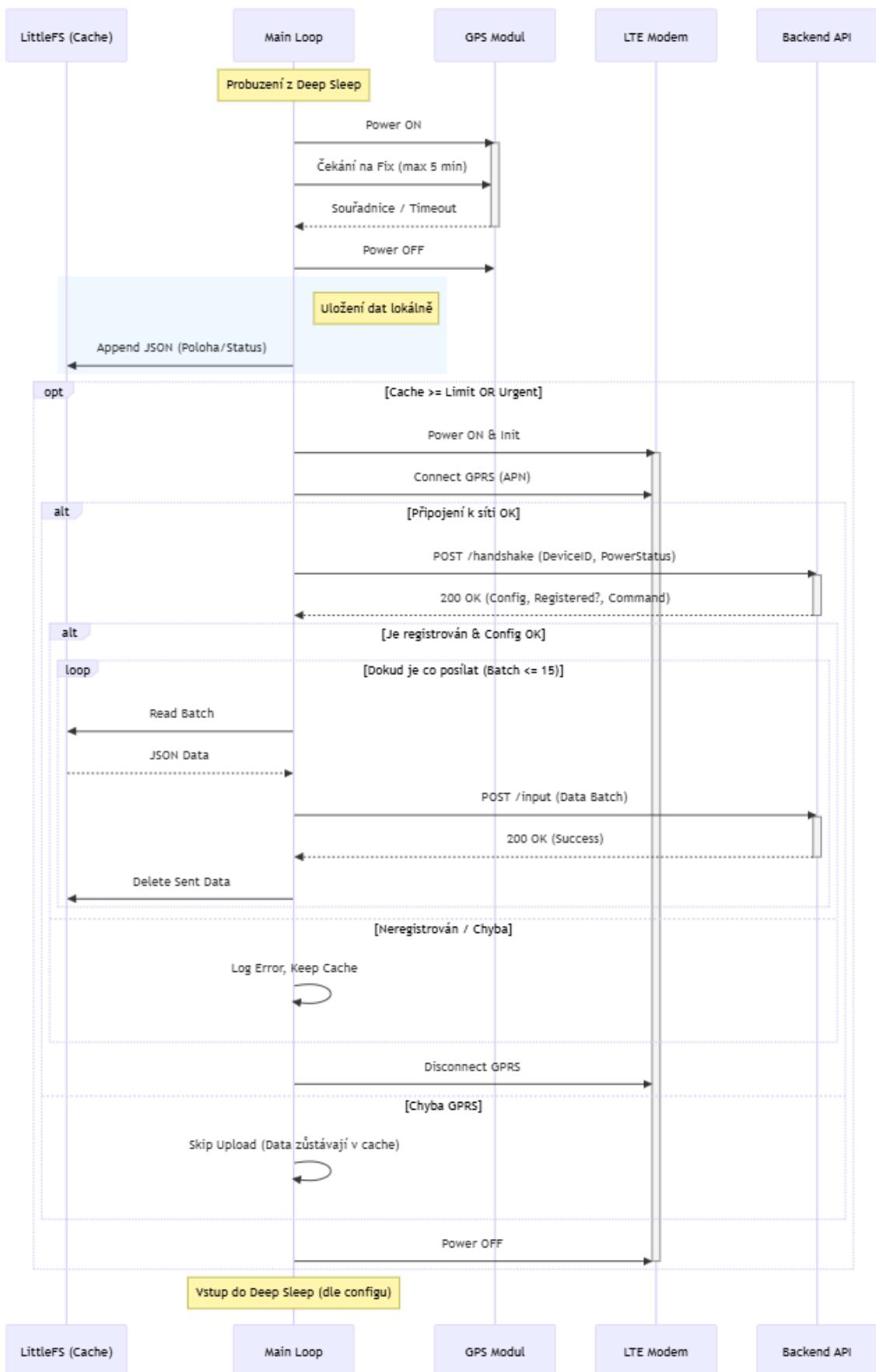
Obrázek 1.17: OTA režim - webové rozhraní



Obrázek 1.18: Usecase diagram



Obrázek 1.19: "vnitřnosti" zařízení



Obrázek 1.20: sekvenční diagram životního cyklu

2 SERVEROVÁ ČÁST

2.1 ÚVOD A KONCEPCE SYSTÉMU

Server zastává centrální roli v projektu. Zajišťuje komunikaci s hardwarovými jednotkami, trvalé ukládání telemetrických dat, správu uživatelských účtů a poskytování uživatelského rozhraní pro vizualizaci polohy a stavu zařízení, i jejich konfiguraci.

Hlavním úkolem serveru je agregace dat z jednotlivých GPS trackerů. Hardwarové jednotky odesírají data (poloha, 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.

2.1.1 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 (models/):** 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 (views/):** Stará se o prezentaci dat uživateli. Využíváme šablonovací systém EJS, který umožňuje vkládat data z backendu přímo do HTML struktury.
- **Controller (controllers/):** 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 umožnuje lepší organizaci kódu a rozšiřování o nové funkce.

2.1.2 Relační databáze a ORM (MySQL + Sequelize)

Jako databázový systém bylo zvoleno relační **MySQL**. Pro interakci s databází je využita technika **ORM** (Object-Relational Mapping) prostřednictvím knihovny Sequelize. Ta mapuje databázové tabulky na JavaScriptové objekty, což zjednoduší manipulaci s daty.

2.1.3 Principy autentizace a OAuth 2.0

Pro zabezpečení přístupu a možností third-party-autentizace je implementován mechanismus využívající knihovnu `passport.js`. Kromě lokálního přihlášení (email/heslo) je tak podporován i standard **OAuth 2.0** pro přihlášení přes externí poskytovatele (v našem případě: Google, GitHub).

2.1.4 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, ale v našem případě je toto použito pouze pro komunikaci s HW. Web a APK klienti využívají **sessions**.
- **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)

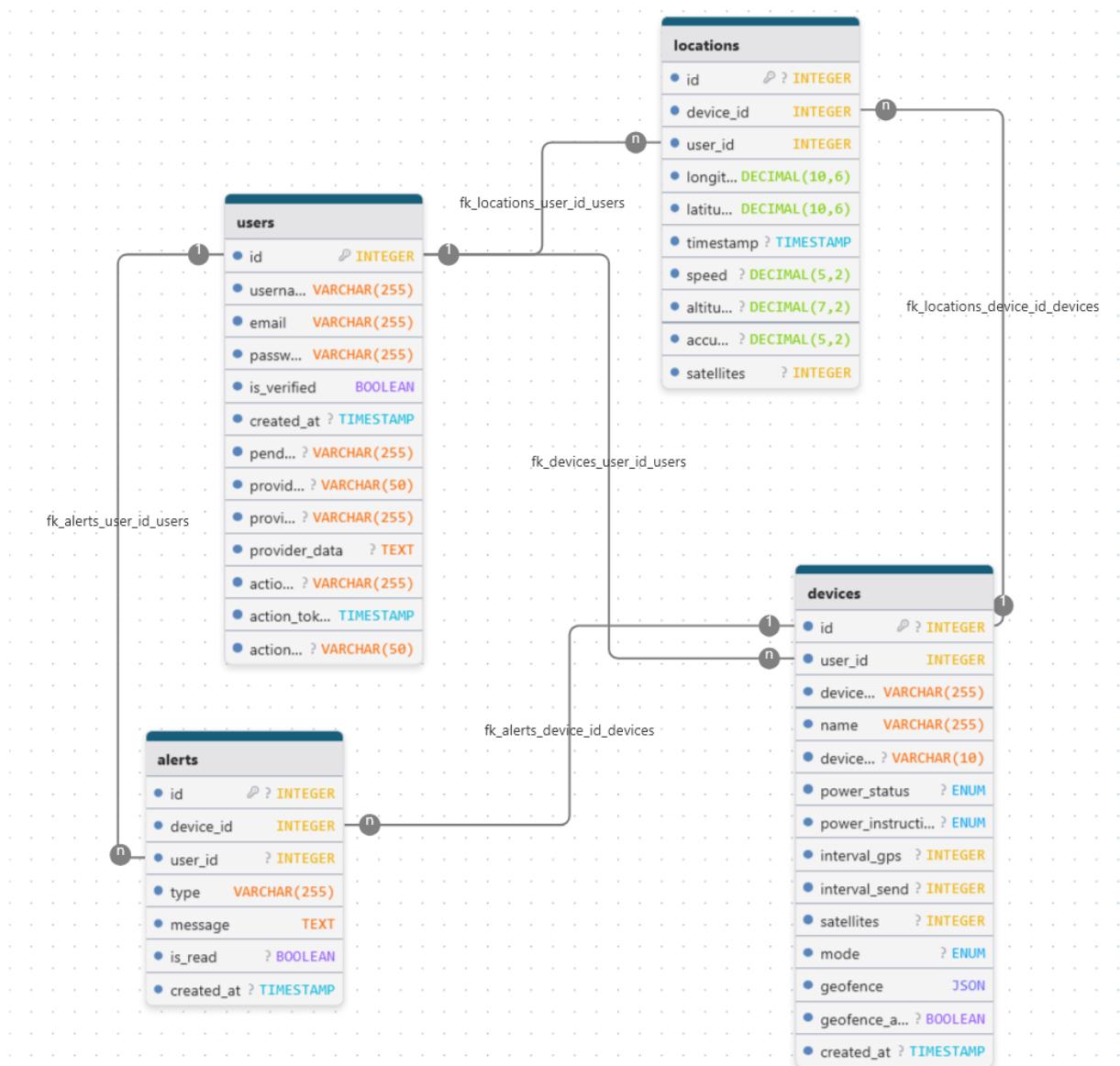
Databázová vrstva je postavena na relační databázi MySQL. Pro komunikaci s databází je využita knihovna `mysql2` ve spojení s ORM frameworkm Sequelize. pro iniciální konfiguraci databáze používáme soubor **init-db.sql**, který vytvoří potřebné tabulky a indexy.

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).
- Alert: Záznamy o bezpečnostních událostech (např. opuštění geofence zóny nebo nárvat do ní).

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



Obrázek 2.1: ER diagram databázových modelů

API - HW tracker (ESP32)

HW klient se při komunikaci autentizuje pouze pomocí `device_id` (získaného z MAC adresy) které je zaregistrováno pokud uživatel vloží správné uživatelské údaje v registrační části OTA režimu. Nevyužívá session/cookie.

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

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.

API - 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> .

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: Telemetrije 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.

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"
}

```

(a) HW Handshake tok

```

#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
}

```

(b) HW/APK Input tok

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).

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>	Otevří <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

Sessions

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

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

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 zadané heslo zahashuje a porovná s uloženým hashem. To chrání uživatele i v případě úniku databáze.

ROOT účet

Z 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' administrators panel. It includes sections for 'Database Administration', 'Users (3)', 'Devices (4)', 'Latest Locations (50)', and 'Alerts (0)'. The 'Users' section lists three users: 'spora' (local), 'lotr' (google), and 'root' (local). The 'Devices' section lists four devices: 'Google sdk_phone4_x86_64', 'NEO-SM-A7370E', 'samsung SM-S901B', and 'Samsung'. The 'Latest Locations' section shows 50 location entries for device ID AFJFB0002, all at coordinates 37.42/1998. The 'Alerts' section is currently empty.

Obrázek 2.6: Administrátorský panel

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 dělaná s myšlenkou ujednoduchosti. Kombinuje "server-side rendering"(EJS) pro základní strukturu stránky a klientský JavaScript pro dynamické aktualizace dat v reálném čase.

```

const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});

```

(a) Použití Nodemailer.js

(b) Upozornění na opuštění geofence zóny

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**.

Dynamická aktualizace (AJAX)

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

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 control panel for a device named 'ALER33T TEST'. The interface is divided into several sections:

- Device List:** Shows the device name and its last known coordinates (05.12.2025 15:49:19). It includes edit and delete buttons.
- Device Settings:** Includes settings for Mode (Simple), GPS Fix Interval (00:00:00), and Minimum Satellites for Fix (7). A 'Save Settings' button is present.
- Power Control:** A red button labeled 'Power OFF' and a green 'ON' button.
- Device Information:** Displays the Device ID (3333337789), Device Type (HW), Registration Date (05.12.2025 15:41:26), and an 'Export GPX' button.
- Live Map:** A map of Prague showing the device's location. A blue marker indicates the current position with the following data in a tooltip: Time: 05.12.2025 15:49:19, Speed: 50.00 km/h, Altitude: 210.00 m. An orange circle represents a cluster of 3 points from different locations. Another tooltip for the cluster shows: Cluster (3 points), C From: 05.12.2025 15:49:19, F To: 05.12.2025 15:49:19, To: 05.12.2025 15:46:24.
- Location History:** A table listing the device's movement history with the following columns: Time, Latitude, Longitude, Speed, Altitude, Accuracy, and Satellites.

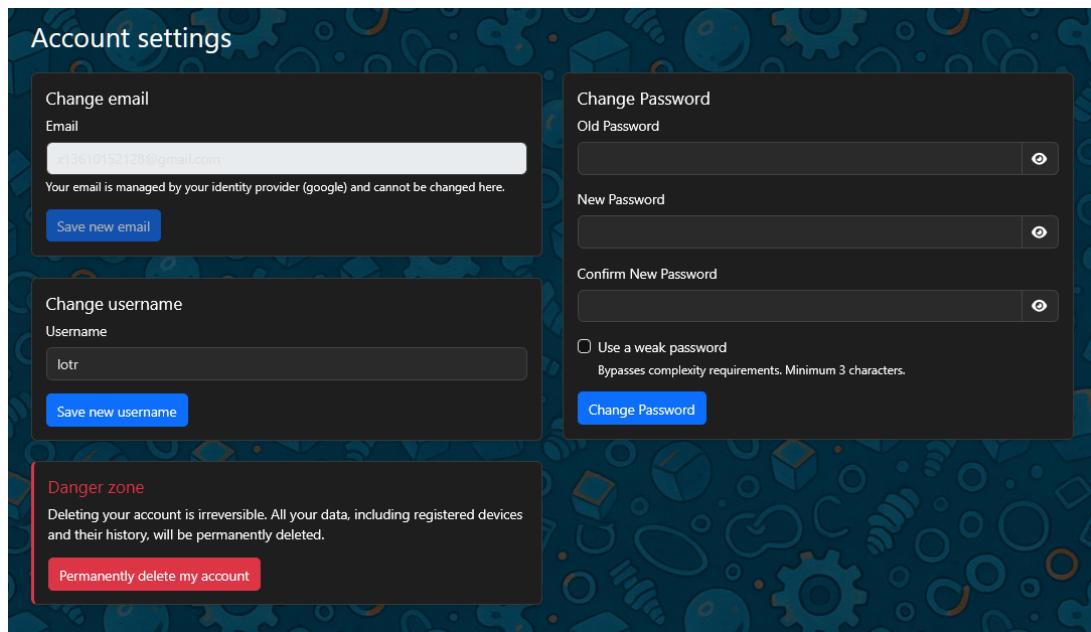
Time	Latitude	Longitude	Speed	Altitude	Accuracy	Satellites
05.12.2025 15:49:19 - 05.12.2025 15:49:19	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05.12.2025 15:46:24 - 05.12.2025 15:46:24	50.088050	14.420770	N/A (Cluster)	N/A	Less 135 m	3 points
05.12.2025 15:46:24	50.098040	14.430760	50.00 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 - DOCKER

Pro zajištění snadného nasazení na různé servery je celý systém kontejnerizován pomocí technologie **Docker**. 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.



Obrázek 2.9: Webové rozhraní - Settings

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

2.4.1 Orchestrace - Docker Compose

Protože systém vyžaduje ke svému běhu nejen aplikační server, ale i databázi, využíváme **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).

- **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).

Registration

Username
lotr

Email
stepan.balner@proton.me

Password

Confirm Password

Use weak password (min. 3 characters)

Register

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

(a) Registrace

Login

Username or Email

Password

[Forgot Password?](#)

Login

OR

Continue with Google

Continue with GitHub

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

(b) Přihlášení

Ověření emailu

To complete your registration, please enter the verification code sent to your email.

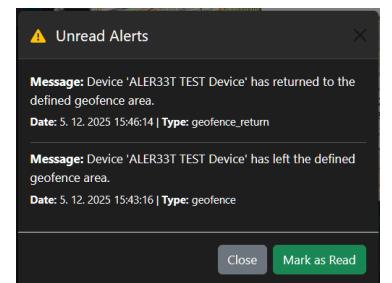
Ověřovací kód

3	4	3	3
---	---	---	---

Verify

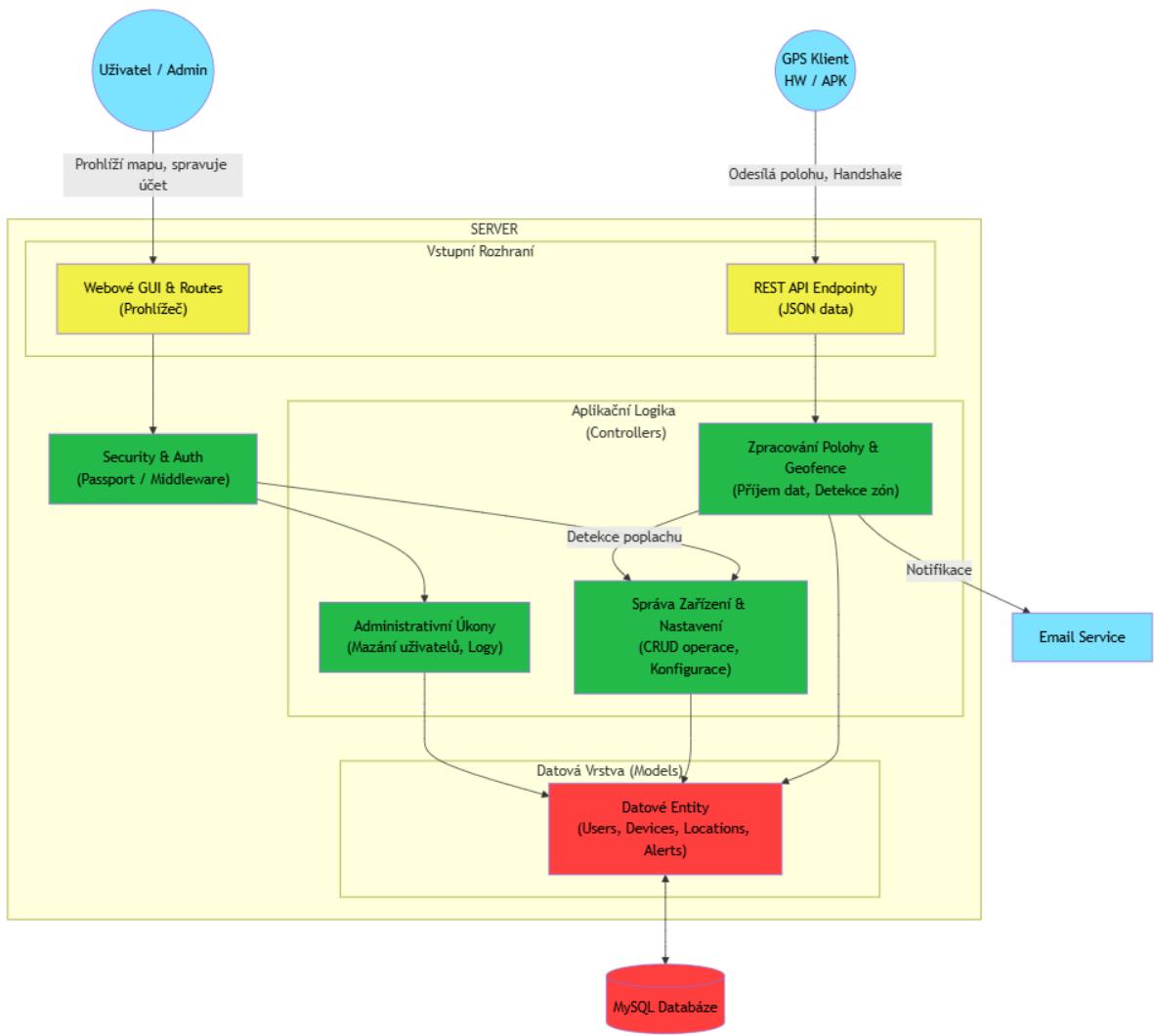
[Didn't receive the code? Resend](#)

(c) Verifikace e-mailu

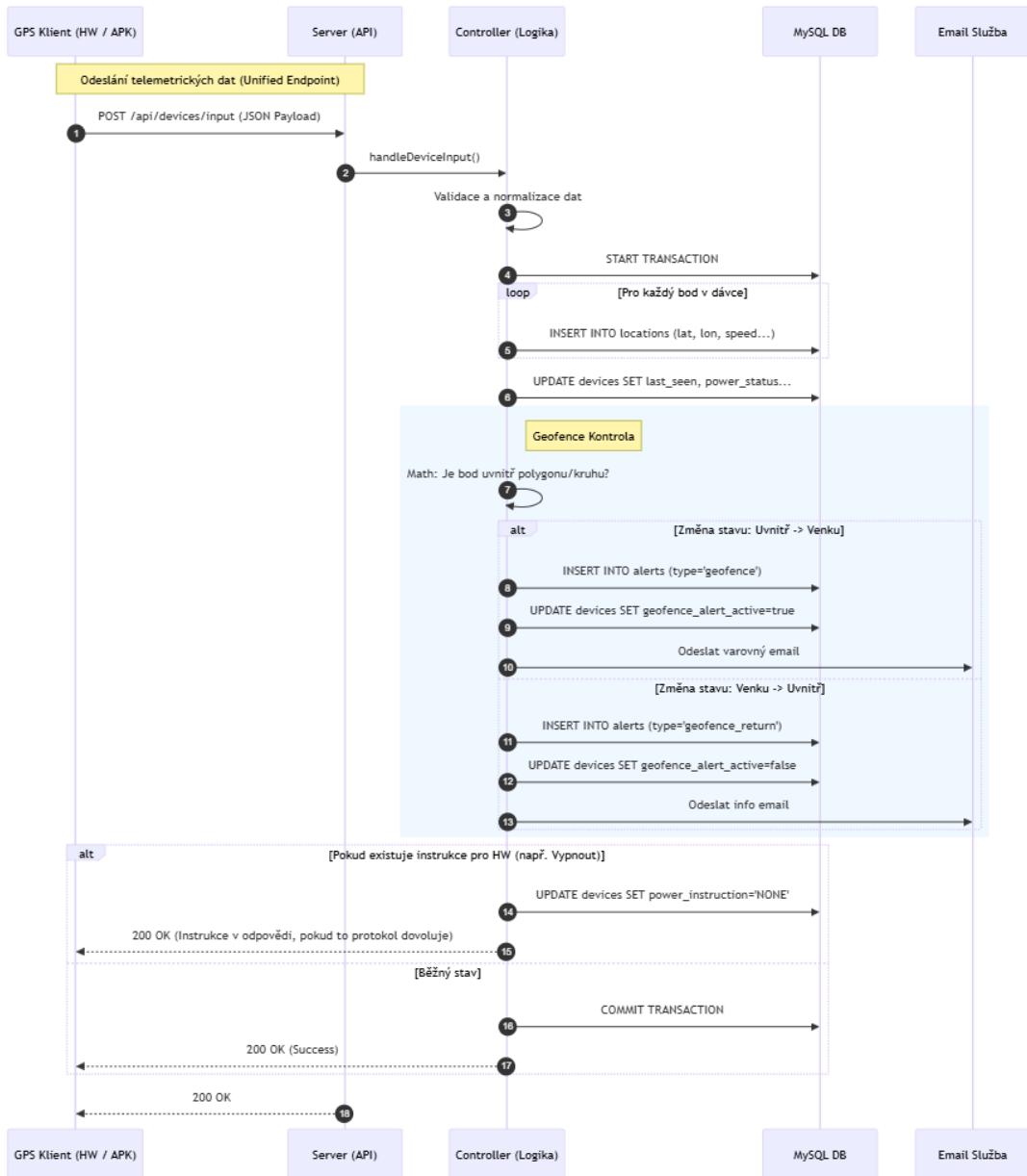


(d) Bezpečnostní upozornění (modal)

Obrázek 2.11: Webové rozhraní: registrace, přihlášení, verifikace e-mailu a bezpečnostní upozornění



Obrázek 2.12: Use-case diagram systému LOTR: aktéři a hlavní případy použití



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

3 APLIKACE PRO ANDROID

3.1 ÚVOD A KONCEPT

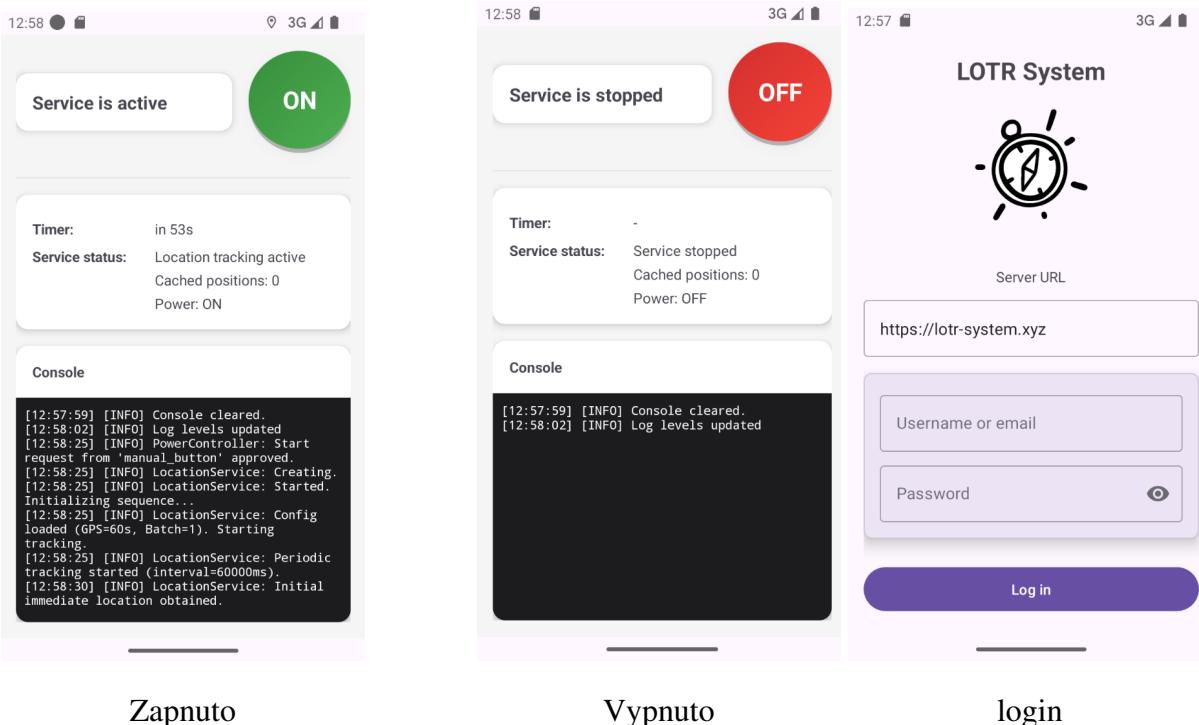
Aplikace pro Android je brána jko aplikace testovací (hlavně pro testování API), ale zároveň ji lze použít jako plnohodnotného klienta pro sběr a odesílání GPS dat na server. Je však skutečností, že kód byl vytvořen pomocí LLM tudíž neručíme za jeho kvalitu a bezpečnost. Byla sice zamýšlena jako plnohodnotná součást projektu, ale kvůli prioritám na Server a HW části (rozprostření zájmů by bylo veliké) byla odsunutá a její vývoj nechán na LLM "Agentu".

Proto také dokumentace nezmiňuje detailey implementace, a omezuje se pouze na ukázku uživatelského rozhraní a jeho mžností.

3.1.1 Hlavní ovládací panel (Dashboard)

Po přihlášení se zobrazí hlavní obrazovka (MainActivity), která poskytuje:

- **Indikátor stavu:** Velká ikona a text informující, zda služba bží, je zastavena, nebo čeká na GPS signál.
- **Ovládací prvky:** Tlačítka START a STOP pro manuální řízení služby.
- **Statistiky:** Počet nasbíraných bodů v lokální databázi a čas posledního úspěšného odeslání dat.



Obrázek 3.1: Hlavní ovládací panel



Obrázek 3.2: Stav čekání na GPS signál



Obrázek 3.3: Diagnostická konzole

ZÁVĚR A ZHODNOCENÍ

Seznam nedokonalostí či nedodělků může být dosti dlouhý. Jako hlavní nedostatek vidím abseaci 3D tisknutého pouzdra pro HW tracker. V počátcích vývoje jsem sice vymodeloval prototyp (viz. příloha modelu), ale ukázalo se, že s tím jak byly postupně přidávány prvky nebo měněny stávající, nestačil model udržovat krok, jediná použitelná část z tohoto modelu zůstal držák na : baterii a její nabíječku, ten se ukázal jako užitečný pro jednoduší manipulací s celou konstrukcí. Dále také 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 (moožná MQTT mezi-server obstarávající autentifikaci ?), místo pouhého ID a uživatelských údajů (je to takto dosti jednoduché na falšování požadavků). Také nějaké více komplikované možnosti výpisu či třídění dat v uživatelském rozhraní chybí, a tak bych mohl pokračovat...

Co však dokončeno bylo, alespoň základně či přízemně, je samotné jádro 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 do doby vybití baterie používat v rámci možností (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 je zde mnoho místo pro zlepšení a rozšíření.

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] ESPRESSIF SYSTEMS. *ESP-IDF Programming Guide* [Online]. 2024. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/latest/>
- [3] ESPRESSIF SYSTEMS. *Arduino core for the ESP32* [Software]. GitHub, 2024. Dostupné z: <https://github.com/espressif/arduino-esp32>
- [4] SIMCOM WIRELESS SOLUTIONS. *A7670 Series AT Command Manual* [Online]. 2022. Dostupné z: <https://www.simcom.com/product/A7670E.html>
- [5] SIMCOM WIRELESS SOLUTIONS. *A7670 Series Hardware Design* [Online]. 2022. Dostupné z: <https://www.simcom.com/product/A7670E.html>
- [6] LILYGO. *LilyGO T-SIM A7670(E) — ESP32 + LTE Cat-1 Development Board* [Online]. GitHub, 2023. Dostupné z: <https://github.com/Xinyuan-LilyGO/LilyGo-T-SIM-A7670>
- [7] U-BLOX AG. *NEO-6 u-blox 6 GPS Modules Data Sheet* [Online]. 2011. Dostupné z: https://content.u-blox.com/sites/default/files/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf
- [8] U-BLOX AG. *u-blox 6 Receiver Description including Protocol Specification* [Online]. 2011. Dostupné z: https://content.u-blox.com/sites/default/files/u-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-12013%29_Public.pdf
- [9] AMAZON WEB SERVICES. *FreeRTOS Kernel Developer Guide* [Online]. 2024. Dostupné z: <https://www.freertos.org>
- [10] SHYMANSKYY, Volodymyr. *TinyGSM: A small Arduino library for GSM/LTE modules* [Software]. GitHub, 2023. Dostupné z: <https://github.com/vshymanskyy/TinyGSM>
- [11] HART, Mikal. *TinyGPS++: NMEA parser for Arduino* [Software]. GitHub, 2023. Dostupné z: <https://github.com/mikalhart/TinyGPSPlus>

- [12] BLANCHON, Benoit. *ArduinoJson* [Software]. 2024. Dostupné z: <https://arduinojson.org/>
- [13] OPENJS FOUNDATION. *Node.js Documentation* [Online]. Dostupné z: <https://nodejs.org/en/docs/>
- [14] EXPRESS. *Express — Node.js web application framework* [Online]. Dostupné z: <https://expressjs.com>
- [15] SEQUELIZE. *Sequelize v6 Reference Manual* [Online]. Dostupné z: <https://sequelize.org>
- [16] ORACLE. *MySQL Documentation* [Online]. Dostupné z: <https://dev.mysql.com/doc/>
- [17] AUTH0. *jsonwebtoken — JWT for Node.js* [Software]. GitHub, 2024. Dostupné z: <https://github.com/auth0/node-jsonwebtoken>
- [18] DcodeIO. *bcrypt.js — bcrypt in JavaScript* [Software]. GitHub, 2024. Dostupné z: <https://github.com/dcodeIO/bcrypt.js>
- [19] MOTDOTLA. *dotenv — Loads environment variables* [Software]. GitHub, 2024. Dostupné z: <https://github.com/motdotla/dotenv>
- [20] EXPRESSJS. *cors — CORS middleware for Express* [Software]. GitHub, 2024. Dostupné z: <https://github.com/expressjs/cors>
- [21] IETF. *The OAuth 2.0 Authorization Framework*. RFC 6749. 2012. Dostupné z: <https://www.rfc-editor.org/rfc/rfc6749>
- [22] IETF. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750. 2012. Dostupné z: <https://www.rfc-editor.org/rfc/rfc6750>
- [23] IETF. *JSON Web Token (JWT)*. RFC 7519. 2015. Dostupné z: <https://www.rfc-editor.org/rfc/rfc7519>
- [24] AGAFONKIN, Vladimir. *Leaflet: an open-source JavaScript library for mobile-friendly interactive maps* [Online]. Dostupné z: <https://leafletjs.com>
- [25] OPENSTREETMAP FOUNDATION. *OpenStreetMap — Copyright and License* [Online]. Dostupné z: https://wiki.openstreetmap.org/wiki/Legal_FAQ
- [26] OPENSTREETMAP FOUNDATION. *Tile Usage Policy* [Online]. Dostupné z: <https://operations.osmfoundation.org/policies/tiles/>

- [27] DOKULIL, Jakub. *Šablona pro psaní SOČ v programu L^AT_EX* [Online]. Brno, 2020. Dostupné z: https://github.com/Kubiczek36/SOC_sablon
- [28] GOOGLE DEEPMIND. *Gemini 3: Multimodal Generative AI Model* [Online]. 2025. Asistoval při generování a strukturování dokumentace. Dostupné z: <https://deepmind.google/>

Seznam obrázků

1.1	LilyGO T-Call V1.5 - integrované řešení ESP32 + LTE modem	6
1.2	Modul GPS a zapojení pro řízení napájení	7
1.3	Baterie a nabíjecí modul / power latch modul - osazená deska	8
1.4	Blokové schéma zapojení hardwarového trackeru [easyEDA]	9
1.5	Schéma power-modul [easyEDA]	9
1.6	funkce power_init()	12
1.7	funkce graceful_shutdown()	13
1.8	funkce enter_deep_sleep()	13
1.9	funkce gps_get_fix()	14
1.10	funkce fs_init()	15
1.11	funkce modem_send_post_request()	16
1.12	funkce modem_perform_handshake()	16
1.13	funkce modem_initialize()	17
1.15	konfigurace Wifi APN pro OTA režim	18
1.14	Ukázka Wi-Fi	18
1.16	prototyp krabičky - průhled	20
1.17	OTA režim - webové rozhraní	20
1.18	Usecase diagram	21
1.19	"vnitřnosti" zařízení	21
1.20	sekvenční diagram životního cyklu	22
2.1	ER diagram databázových modelů	25
2.3	Ukázka Swagger dokumentace API	28
2.4	příklad jedné z funkcí v authorization.js	30
2.5	Passport.js autentizační strategie	30
2.6	Administrátorský panel	31
2.7	E-mailové scénáře: transakční zprávy a bezpečnostní upozornění	32

2.8	Webové rozhraní - Správa zařízení	33
2.9	Webové rozhraní - Settings	34
2.10	Ukázka Dockerfile pro serverovou aplikaci	34
2.11	Webové rozhraní: registrace, přihlášení, verifikace e-mailu a bezpečnostní upozornění	35
2.12	Use-case diagram systému LOTR: aktéři a hlavní případy použití	36
2.13	Sekvenční diagram: tok sledování zařízení (handshake, input, aktualizace polohy)	37
3.1	Hlavní ovládací panel	40
3.2	Stav čekání na GPS signál	40
3.3	Diagnostická konzole	40

Seznam tabulek

2.1	HW API endpointy	26
2.2	APK API endpointy	27
2.3	Rychlá orientace v rolích a middleware	29