MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Radim Vališ

Systém pro management ESP vývojových desek

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. RNDr. Tomáš Bureš, Ph.D.
Studijní program: Informatika

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych poděkovat svému vedoucímu práce panu prof. RNDr. Tomáši Burešovi, Ph.D. za jeho trpělivost a cenné rady, které mi pomohly při tvorbě této práce. Dále bych rád poděkoval své rodině za jejich podporu a trpělivost během celého studia.

Název práce: Systém pro management ESP vývojových desek

Autor: Radim Vališ

katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. RNDr. Tomáš Bureš, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: ESP vývojové desky jsou dostupná a výkonná zařízení s širokým spektrem využití. Jejich programování je však poměrně složité, což omezuje okruh potenciálních uživatelů. Tato práce se zabývá návrhem a implementací webové aplikace a knihovny postavené na ESP-IDF, které společně propojují programátory ESP desek s jejich koncovými uživateli. Programátoři mohou prostřednictvím webového rozhraní publikovat svůj firmware a jeho konfigurační formulář, který umožňuje deklarovat proměnné, jejichž hodnoty lze nastavit až při nahrávání firmwaru do desky. Uživatelé vývojových desek mohou registrovat své desky, vyhledávat dostupný firmware, konfigurovat ho pomocí konfiguračního formuláře a firmware s konfigurací vzdáleně nahrávat do desek pomocí mechanismu OTA (Over-The-Air), a to bez nutnosti cokoliv instalovat a programovat. Spojení mezi deskou a webovou aplikací zajišťuje doprovodná knihovna, která se automaticky připojuje k Wi-Fi síti, komunikuje s webovou aplikací přes MQTT a provádí OTA aktualizace.

Klíčová slova: IoT, ESP-IDF, automatizace

Title: Management framework for ESP-based development boards

Author: Radim Vališ

department: Department of Distributed and Dependable Systems

Supervisor: prof. RNDr. Tomáš Bureš, Ph.D., Department of Distributed and Dependable Systems

Abstract: ESP-based development boards are affordable and powerful devices with a wide range of applications. However, their programming is quite complex, which limits the pool of potential users. This thesis deals with the design and implementation of a web application and a library built on ESP-IDF that jointly connect ESP board programmers with end users. Through the web interface, programmers can publish their firmware and its configuration form, which allows them to declare variables whose values can be set when the firmware is flashed onto the board. Development board users can register their boards, search for available firmware, configure it using the configuration form, and upload the firmware with the configuration remotely to their boards using the OTA (Over-The-Air) mechanism, without the need to install and program anything. The connection between a board and the web application is provided by the companion library that automatically connects to the Wi-Fi network, communicates with the web application using MQTT and performs OTA updates.

Keywords: IoT, ESP-IDF, automation

Obsah

1 Úvod **8**
1.1 Cíl práce . . . 8
1.2 Cíloví uživatelé . . . 8
1.2.1 Uživatel vývojové desky . . . 9
1.2.2 Programátor . . . 9

2 Analýza **10**
2.1 Funkční požadavky . . . 10
2.2 Příklad použití . . . 11
2.3 Platforma . . . 11
2.4 Architektura . . . 12
2.5 Uživatelské rozhraní . . . 12
2.5.1 Programátor . . . 12
2.5.2 Uživatel vývojové desky . . . 14
2.6 Hardware . . . 17
2.7 Uživatelský firmware . . . 17
2.8 Konfigurace firmwaru . . . 17
2.8.1 NVS . . . 18
2.8.2 Konfigurační formulář . . . 18
2.8.3 Průběh konfigurace . . . 20
2.8.4 Konfigurace ESP Managera . . . 20
2.9 Nahrávání a aktualizace firmwaru . . . 20
2.9.1 OTA . . . 21
2.10 Síťová komunikace . . . 21
2.11 Flash paměť . . . 21
2.12 Nahrávání výchozího firmwaru . . . 22
2.12.1 esptool-js . . . 23
2.13 Zabezpečení . . . 23

3 Implementace **25**
3.1 Architektura webové aplikace . . . 25
3.2 Backend . . . 26
3.2.1 API . . . 26
3.2.2 Databáze . . . 27
3.2.3 Ukládání souborů . . . 28
3.2.4 Validace konfiguračních souborů . . . 28
3.2.5 Generování NVS . . . 28
3.2.6 Komunikace s ESP Managerem . . . 29
3.2.7 Bezpečnost . . . 30
3.3 Frontend . . . 30
3.3.1 Obrazovky programátora . . . 30
3.3.2 Obrazovky uživatele vývojových desek . . . 31
3.3.3 Navigace . . . 33
3.3.4 Komunikace s backendem . . . 33
3.3.5 Globální úložiště . . . 34

3.3.6 Konfigurační formulář 34
3.3.7 Sériová komunikace s deskou 34
3.4 Společná část 34
3.5 ESP Manager 35
3.5.1 Inicializace 35
3.5.2 Hlavní smyčka 36
3.5.3 Meziprocesová komunikace 37
3.5.4 Nahrávání a aktualizace firmwaru 37
3.5.5 Přepnutí na výchozí firmware 38
3.6 Výchozí firmware 38

4 Práce se systémem **39**
4.1 Administrátor 39
4.1.1 Instalace 39
4.1.2 Certifikáty 39
4.1.3 Konfigurace 39
4.1.4 Spuštění 40
4.2 Programátor 40
4.2.1 Programování firmwaru 40
4.2.2 Psaní konfiguračního formuláře 43
4.2.3 Nahrání firmwaru 44
4.2.4 Aktualizace firmwaru 44
4.2.5 Smazání firmwaru 44
4.3 Uživatel vývojové desky 45
4.3.1 Registrace vývojové desky 45
4.3.2 Nahrání firmwaru do vývojové desky 46
4.3.3 Aktualizace firmwaru ve vývojové desce 47
4.3.4 Smazání desky 47

5 Závěr **48**
5.1 Rozšíření 48

Literatura **51**

Seznam obrázků **52**

1 Úvod

Internet of Things (IoT) je dynamicky se rozvíjející technologie, která nalézá své uplatnění jak v průmyslu, tak stále častěji i v životech obyčejných lidí. Příkladem mohou být chytré termostaty, meteostanice či automatická zavlažování. Zásluhu na rozmachu IoT mají i vývojové desky z rodin Arduino, STM, ESP a další, které tvoří jádro mnoha komerčně úspěšných zařízení. Tam ale jejich využití ani zdaleka nekončí. Jejich hlavní doménou jsou totiž nejrůznější domácí projekty, a to zejména kvůli jejich mimořádné flexibilitě a nízké pořizovací ceně. Každý má tak na dosah nástroj, s jehož pomocí je schopen sestrojit takřka libovolné zařízení a navíc jen za zlomek ceny, za kterou jsou prodávány obdobná zařízení na trhu.

Široká škála schopností těchto desek jde ovšem ruku v ruce s vysokou komplexitou. Proniknout do jejich ekosystémů nebývá vůbec jednoduché, obvykle je nutné instalovat mnoho vývojářských nástrojů, mít pokročilé programátorské dovednosti a obětovat mnoho hodin studiu dokumentace.

1.1 Cíl práce

Cílem práce je navrhnout a implementovat systém pro management vývojových desek prostřednictvím webového rozhraní, který bude propojovat programátory firmwaru s koncovými uživateli vývojových desek. Programátoři budou pomocí systému distribuovat a aktualizovat svůj firmware. K firmwaru bude možné připojit konfigurační formulář umožňující nastavení hodnot vybraných proměnných uvnitř firmwaru.

Uživatelé vývojových desek budou moci přes webové rozhraní zaregistrovat svoji desku, vyhledat firmware, vyplnit konfigurační formulář a nahrát firmware s konfigurací do desky. V případě potřeby bude možné firmware v desce i vzdáleně aktualizovat. Pro zajištění vzdálených aktualizací bude vytvořena doprovodná knihovna, kterou bude nutné integrovat do každého firmwaru nahrávaného do systému.

Spravovat vícero typů zařízení od vícero výrobců by bylo ovšem značně komplikované, proto bude systém omezen pouze na podporu ESP vývojových desek (dále jen *vývojových desek* nebo *desek*) od společnosti Espressif Systems.

Systém bude určen k domácímu nebo komunitnímu použití, zvolená architektura proto nebude optimalizována na vyšší počty uživatelů, větší datové toky a maximální zabezpečení komunikace.

1.2 Cíloví uživatelé

Systém cílí na všechny, kteří vlastní vývojové desky nebo umí vývojové desky programovat. Tato sekce proto obě skupiny blíže představuje a identifikuje problémy, se kterými se při používání vývojových desek potýkají.

1.2.1 Uživatel vývojové desky

Tento uživatel vlastní vývojovou desku a má představu o tom, jak by ji chtěl používat. Naneštěstí často nemá na realizaci svých plánů potřebné, zejména programátorské, dovednosti, které jsou zmíněny v samotném úvodu. Při zprovozňování své vývojové desky, programování a konfiguraci firmwaru tak musí spoléhat na pomoc od zkušenějších kolegů.

1.2.2 Programátor

Programátor je technicky zdatný jedinec, který má zkušenosti s vývojem firmwaru, orientuje se v platformě ESP, a tudíž mu například automatizace vlastní domácnosti nečiní žádný problém. Výzvou pro něj naopak může být nasazení firmwaru na desky ostatních. Zdrojový kód a dokumentaci může zveřejnit v gitovém repozitáři, takový firmware se ale většinou podaří zprovoznit jen dalším programátorům. Druhou, časově však daleko náročnější, možností je se s každým zájemcem o firmware fyzicky setkat a se zprovozněním mu pomoci. Takový postup je ale v případě pokračujícího vývoje firmwaru zcela neudržitelný.

2 Analýza

Tato kapitola se zabývá návrhem celého systému. Úvodní sekce se zaměřují na obecné aspekty, jako je výběr vhodné platformy, základní komponenty systému, jejich role a vztahy mezi nimi. Následující sekce pak analyzují vybrané problémy ve větším detailu a navrhují jejich vhodné řešení.

2.1 Funkční požadavky

Na základě stanovených cílů a překážek, které ztěžují cílovým uživatelům práci s vývojovými deskami, lze definovat klíčové funkce, jež by měl systém poskytovat.

Programátor Programátorům by měl systém přinášet následující funkcionalitu.

- Vytvořit firmware a k němu příslušný konfigurační formulář, prostřednictvím kterého lze nastavit hodnoty vybraných proměnných.
- Nahrát firmware a konfigurační formulář do systému.
- Aktualizovat firmware v systému.
- Smazat firmware a konfigurační formulář ze systému.

Uživatel vývojové desky Tito uživatelé by se měli díky systému stát více soběstačnými, proto by měli mít možnost provádět následující operace, a to bez odhalování technických detailů ze strany systému a nutnosti cokoliv instalovat nebo programovat.

- Registrovat desku do systému.
- Vyhledat v systému firmware, nakonfigurovat jej přes konfigurační formulář a vzdáleně nahrát firmware s konfigurací do desky.
- Vzdáleně aktualizovat firmware nahraný v desce.
- Smazat firmware z desky.
- Smazat desku ze systému.

2.2 Příklad použití

Programátor vytvořil firmware, který v pravidelných intervalech čte údaje o teplotě ze senzoru DHT22 a posílá je do jeho monitorovací aplikace. Po čase se rozhodne do aplikace přizvat své přátele, protože ale ne všichni mají zkušenosti s programováním mikrokontrolérů, použije výše představený systém. Programátor proto provede následující kroky.

- Firmware upraví tak, aby byl kompatibilní se systémem.
- Vytvoří konfigurační formulář pro nastavení uživatelsky specifických parametrů.

Upravený firmware s konfiguračním formulářem nahraje do systému. Zájemci o aplikaci zaregistrují do systému svoji desku, vyhledají v něm firmware, nakonfigurují si ho a potvrdí změny. V ten moment převezme iniciativu systém, který se sám postará o bezdrátový přenos firmwaru a konfigurace do cílové vývojové desky.

Po čase se programátor rozhodne firmware aktualizovat. DHT22 umí měřit vedle teploty i vlhkost vzduchu, proto bude nová verze měřit i tuto veličinu. Systém v okamžiku nahrání nové verze firmwaru upozorní všechny jeho provozovatele na dostupnou aktualizaci. Pakliže ji provozovatel potvrdí, dojde k vzdálené aktualizaci jeho desky. Od té chvíle uvidí v monitorovací aplikaci i vlhkost vzduchu v okolí svého zařízení.

2.3 Platforma

Nezbytnou součástí systému bude backend poskytující API pro správu vývojových desek a firmwarů. Aby uživatelé mohli s tímto rozhraním pohodlně pracovat, je třeba zvolit vhodnou platformu pro implementaci uživatelského rozhraní.

První možností je mobilní aplikace. Výhodou tohoto řešení je, že aplikace by mohla být distribuována přes oficiální obchod (Google Play, App Store), přes který lze aplikace snadno instalovat a aktualizovat. Toto řešení by však nebylo optimální pro programátory, jelikož vývoj firmwaru probíhá převážně na počítači, tudíž by museli při jeho nahrávání do systému řešit jeho přenos z počítače do svého telefonu. Aplikace navíc musí podporovat sériovou komunikaci s deskou přes USB, protože to je jediný způsob, jak do ní při prvním použití nahrát firmware. Zatímco na Androidu by to bylo technicky možné, zařízení s iOS by zřejmě kvůli řadě restrikcí od společnosti Apple musely být vynechány.

Další možností je vytvořit desktopovou aplikaci, která by při sériové komunikaci s deskou neměla žádná omezení a programátoři by mohli firmware nahrávat z počítače přímo do systému. Znamenalo by to však, že i jednoduché operace, jako je např. odstranění desky ze systému, by bylo nutné provádět výhradně na počítači, přestože by byly realizovatelé i pomocí telefonu. Desktopové aplikace navíc mají v porovnání s těmi mobilními podstatně složitější proces instalace a aktualizace.

Nejlepší volbou je proto webová aplikace, která kombinuje výhody obou variant. Je přístupná z libovolného zařízení s webovým prohlížečem, není nutné ji instalovat a aktualizace probíhají z pohledu uživatele automaticky. Nahrávání firmwaru bude možné realizovat pomocí webového formuláře, sériovou komunikaci zas pomocí Web Serial API [1].

2.4 Architektura

Systém bude složen z následujících komponent.

Webová aplikace První částí bude *webová aplikace*, která bude dále členěna na frontend, backend a MQTT broker. Frontend bude single-page aplikace, prostřednictvím které bude možné provádět všechny hlavní úkony v systému. Bude umožňovat registraci vývojových desek, sledování jejich stavu, nahrávání a konfiguraci firmwaru, i provádění aktualizací. Součástí backendu bude databáze pro ukládání management dat, souborový systém pro ukládání firmwarů a konfiguračních souborů a REST API, prostřednictvím kterého bude možné s těmito daty manipulovat. MQTT broker bude zprostředkovávat obousměrnou komunikaci mezi backendem a vývojovými deskami 2.10, důležitou především pro konfiguraci OTA aktualizací 2.9.

ESP Manager *Webovou aplikaci* bude doplňovat knihovna napsaná v ESP-IDF (Espressif IoT Development Framework) [2], která bude označována jako *ESP Manager*, a kterou budou mít programátoři povinnost použít ve všech firmwarech, které nahrají do systému. Úkolem *ESP Managera* se bude automaticky připojit k Wi-Fi, komunikovat s backendem pomocí MQTT a v případě potřeby provést OTA aktualizaci. ESP-IDF není jediný framework pro programování ESP desek, mezi další patří např. Arduino Core nebo MicroPython, ve kterých by bylo možné *ESP Managera* rovněž implementovat. Pro zachování kompatibility je ale nezbytné, aby *ESP Manager* i firmware byly naprogramovány pomocí stejného frameworku. ESP-IDF byl zvolen, protože nabízí největší kontrolu nad projektem, umožňuje psát nejefektivnější kód a má rozsáhlé API pro práci se všemi periferiemi.

Výchozí firmware Třetí součástí systému bude *výchozí firmware*, což bude wrapper nad *ESP Managerem*. Ten bude na desce spuštěn v případech, kdy uživatel nebude chtít nebo mít možnost (např. bezprostředně po registraci desky) spustit uživatelský firmware.

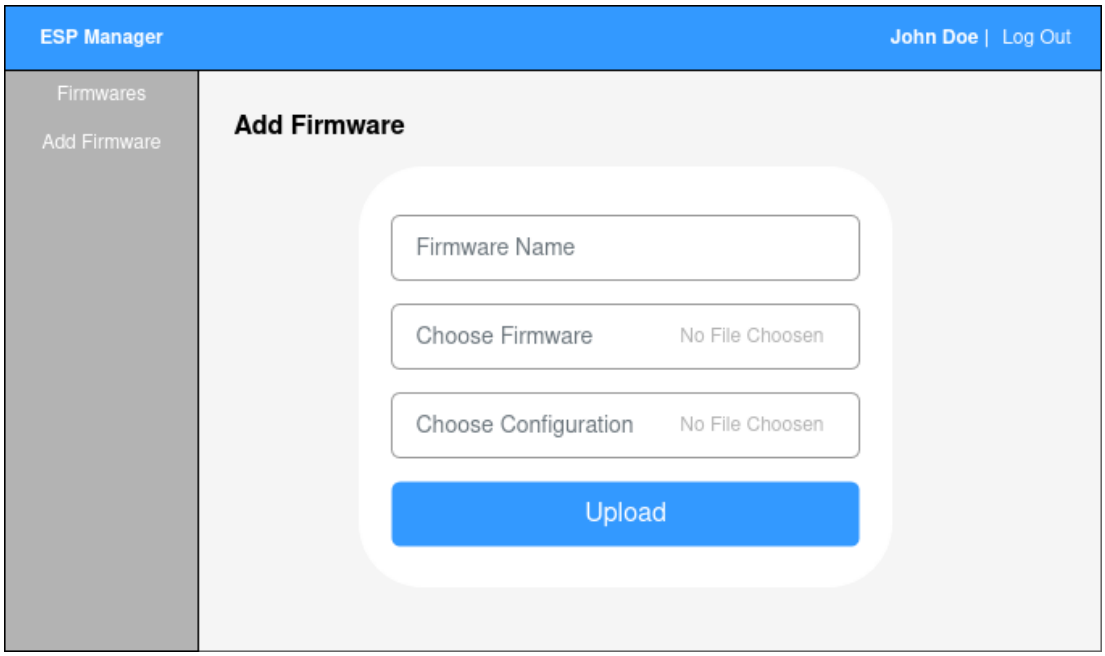
2.5 Uživatelské rozhraní

Tato sekce obsahuje návrh uživatelského rozhraní, který reflektuje funkční požadavky a architekturu systému. Prezentovaný návrh není definitivní, během vývoje může docházet k jeho úpravám (srovnej s implementací v části 3.3.1 a v části 3.3.2).

2.5.1 Programátor

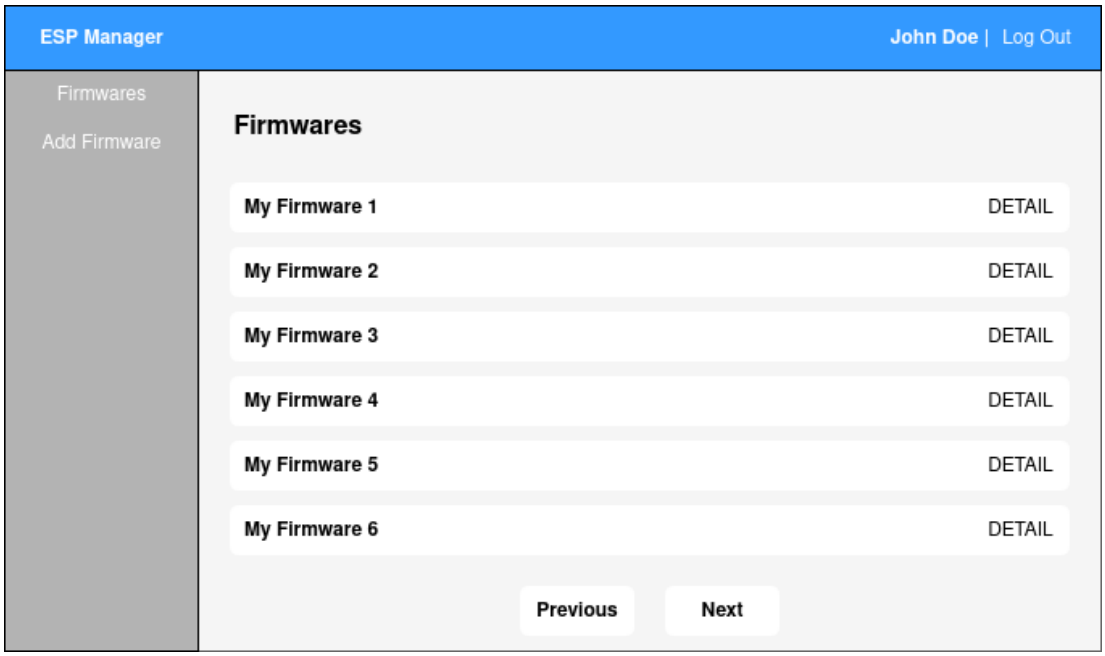
Pro programátory budou určeny následující obrazovky.

Nahrání firmwaru Na této obrazovce programátor zvolí název firmwaru, nahraje firmware a konfigurační formulář a tlačítkem `UPLOAD` odešle vše na backend.



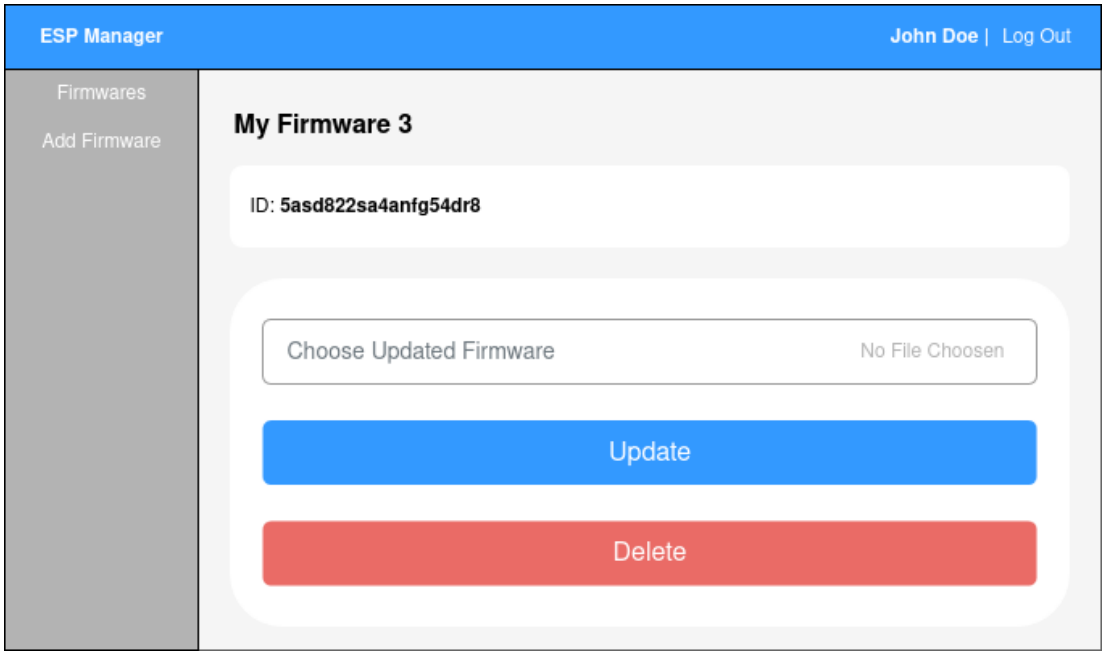
Obrázek 2.1 Nahrání firmwaru

Seznam nahraných firmwarů Tato obrazovka bude zobrazovat seznam všech firmwarů, které programátor do systému nahrál. Ze seznamu firmwarů se bude možné dostat na detaily jednotlivých firmwarů, např. kliknutím na tlačítko, kliknutím na kartu reprezentující daný firmware apod. Obrazovka bude případně obsahovat ovládací prvky umožňující přehlednější zobrazení, např. stránkování, filtrování podle názvu apod.



Obrázek 2.2 Seznam nahraných firmwarů

Detail firmwaru Na této obrazovce programátor uvidí základní informace o daném firmwaru. Dále bude přítomen formulář pro aktualizaci firmwaru a tlačítko pro jeho odstranění ze systému.

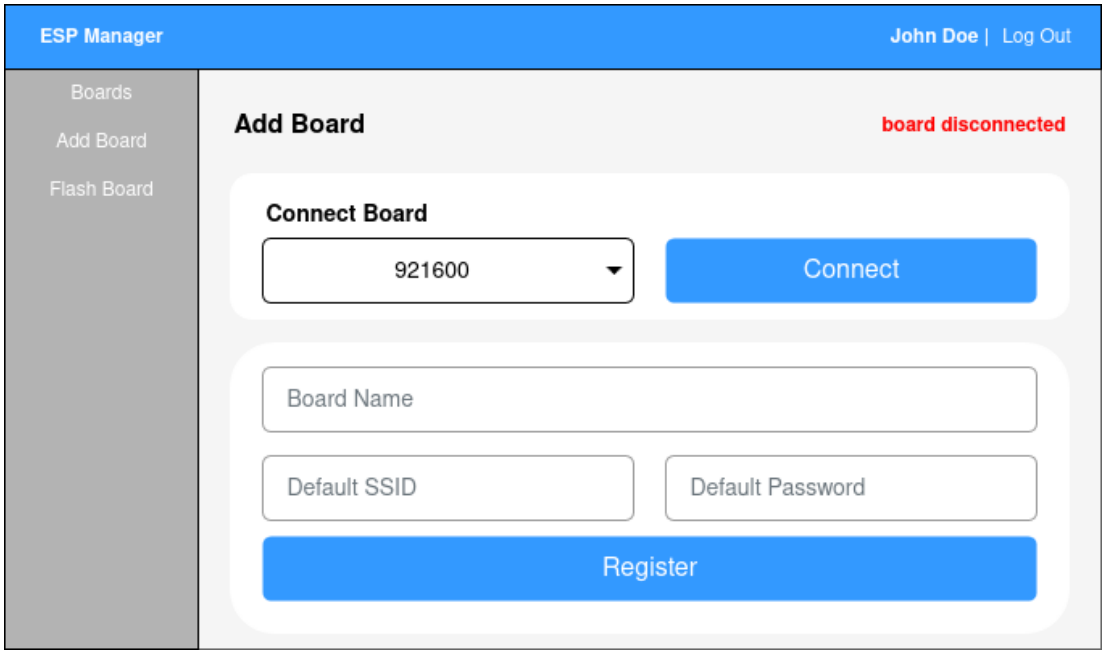


Obrázek 2.3 Detail firmwaru

2.5.2 Uživatel vývojové desky

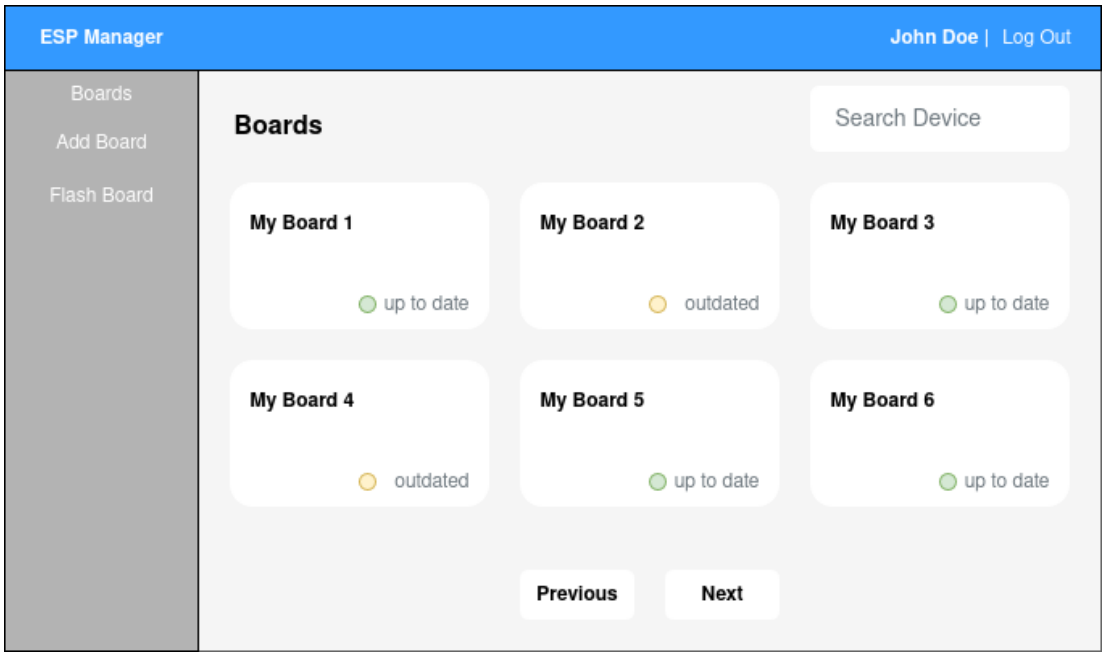
Uživatelé vývojových desek budou používat následující obrazovky.

Registrace vývojové desky Na této obrazovce bude možné zvolit název desky a nakonfigurovat *ESP Managera*, tj. zvolit SSID a heslo WiFi sítě, ke které se bude deska připojovat. Tlačítkem `CONNECT` bude možné navázat sériovou komunikaci s deskou, tlačítkem `REGISTER` bude možné automaticky stáhnout *výchozí firmware* a nahrát jej do desky.



Obrázek 2.4 Registrace vývojové desky

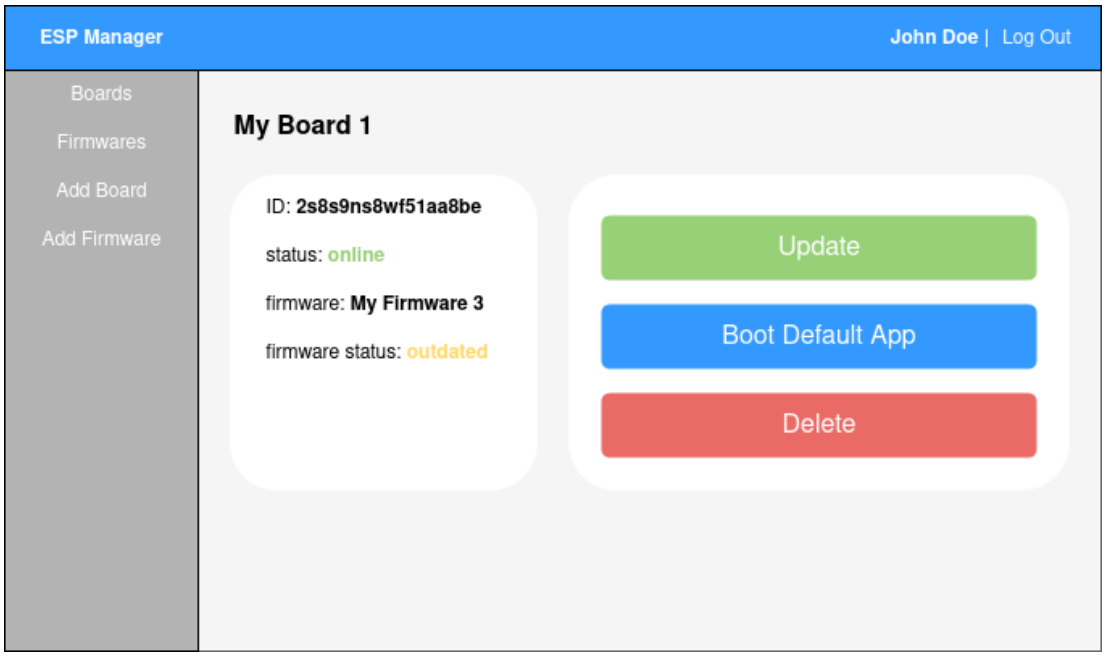
Seznam vývojových desek Seznam registrovaných vývojových desek bude ukazovat všechny desky daného uživatele společně s nejdůležitějšími informacemi, jako je např. stav firmwaru. Opět bude možné se prokliknout na detaily jednotlivých desek.



Obrázek 2.5 Seznam registrovaných vývojových desek

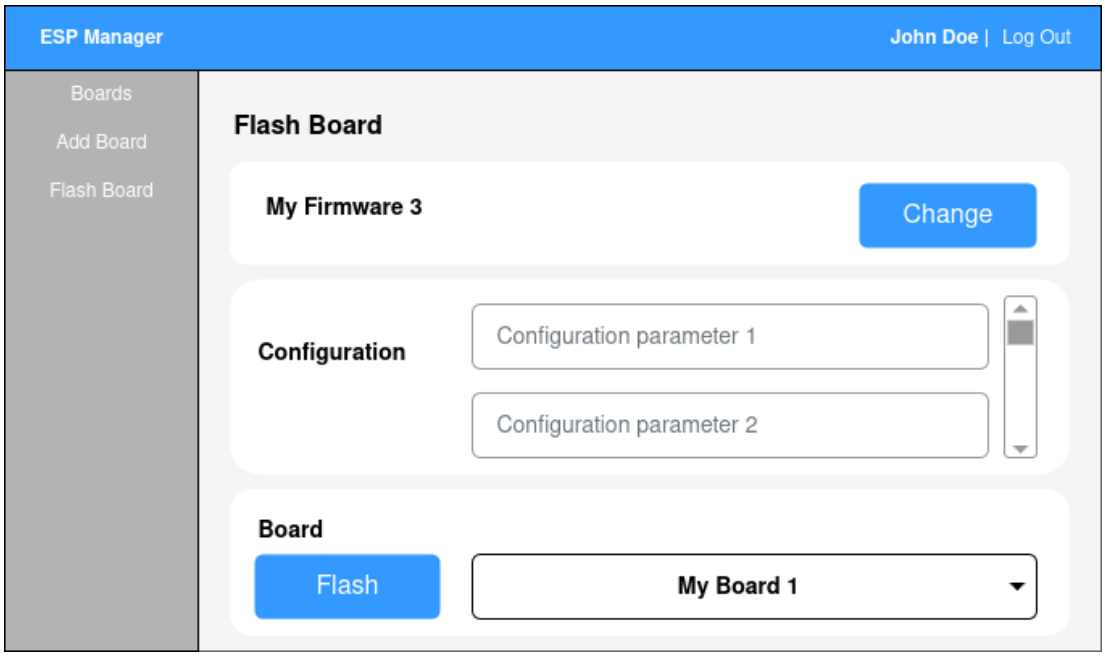
Detail vývojové desky Tento pohled bude poskytovat základní informace o zvolené desce a dále informace, zdali je deska ve spojení s backendem a zdali je nahraný firmware aktuální. Bude-li k dispozici aktualizace, bude ji možné zahájit

stisknutím tlačítka `UPDATE`. Dalšími možnostmi bude opětovné spuštění *výchozího firmwaru* a jeho odstranění ze systému.



Obrázek 2.6 Detail vývojové desky

Nahrávání firmwaru do desky Součástí nahrávací obrazovky bude vyhledávací pole, do kterého bude uživatel zadávat identifikátor firmwaru. Pokud bude daný firmware existovat, bude možné vyplnit jeho konfigurační formulář a zvolit desku, do které se má firmware s konfigurací nahrát.



Obrázek 2.7 Nahrávání firmwaru do desky

2.6 Hardware

Systém bude podporovat pouze desky, jejichž součástí je ESP čip s integrovaným Wi-Fi rozhraním, v jehož důsledku budou možné OTA aktualizace, a USB-UART převodník napojený na USB port, který umožní nahrát *výchozí firmware* a přidružená data. Nebude proto podstatné, zda uživatelé budou používat vývojové desky přímo od Espressif Systems nebo od jiného výrobce, jakým je např. společnost Adafruit, nebo si je zkonstruují svépomocí.

Systém bude v základu podporovat pouze desky osazené čipem ESP32. Série ESP32 je dostatečně všestranná a pravděpodobně i nejrozšířenější, neboť se do prodeje dostala již v roce 2016, tedy jen dva roky po méně výkonné a dnes již nepodporované sérii ESP8266 a čtyři roky před srovnatelnými sériemi ESP32-S2 a ESP32-C3 [3]. S vědomím, že podpora pouze jedné série může být omezující, bude systém navržen tak, aby jeho rozšíření o další série bylo snadné a vyžadovalo jen drobné úpravy v konfiguraci projektu bez nutnosti zasahovat do již existujícího kódu.

2.7 Uživatelský firmware

Uživatelský firmware bude muset dodržovat určitá pravidla vyplývající z architektury systému. Zásadní bude korektní inicializace *ESP Managera*, bez jeho inicializace bude sice možné firmware nahrát do systému a následně jej prostřednictvím OTA aktualizace i nahrát do desky, tím ale dojde k trvalé ztrátě spojení s backendem, a desku tak nebude nadále možné používat. Firmware bude moci využívat Wi-Fi připojení inicializované *ESP Managerem*, nebude však možné manipulovat s parametry připojení, např. přepojit se k jinému přístupovému bodu.

Je důležité, aby programátoři měli při vývoji firmwaru plnou svobodu a mohli využívat libovolné knihovny a komponenty. Z toho důvodu nebude možné, aby programátoři nahrávali do systému zdrojový kód a systém jej pak kompiloval. Bude tedy nutné nahrávat firmware již zkompilovaný. Jelikož systém nebude mít možnost firmware nijak analyzovat, bude zodpovědností programátora nahrávat jen software splňující výše uvedené body.

2.8 Konfigurace firmwaru

Programátoři budou do systému nahrávat firmware určený pro různé účely - meteostanice, termostat, pohybové čidlo atd. Z tohoto důvodu bude nutné zařídit, aby každý firmware mohl disponovat specifickou sadou parametrů, jejichž hodnotu budou uživatelé firmwaru moci nastavit.

Firmware se bude do systému nahrávat zkompilovaný, a není tedy možné, aby systém zapisoval hodnoty parametrů přímo do kódu. Místo toho bude nutné konfiguraci ukládat do jiného oddílu flash paměti a zodpovědností programátora bude implementovat její načítání z tohoto oddílu.

Pro ukládání dat nabízí ESP-IDF možnost vytvořit ve flash paměti souborový systém nebo NVS (Non-Volatile Storage). Při použití souborového systému by bylo nutné vymyslet, v jakém formátu konfiguraci ukládat a zároveň by bylo nutné

přidat do *ESP Managera* API, které by konfiguraci parsovalo. Proto bude systém používat NVS, které má formátování a parsování již vyřešeno.

2.8.1 NVS

NVS je knihovna určena pro práci s dvojicemi klíč-hodnota uložených ve flash paměti. Knihovna je vhodná pro ukládání mnoha malých hodnot, jež mohou být 8bitová až 64bitová čísla se znaménkem i bez znaménka, textové řetězce nebo tzv. `blob` (Binary Large OBject), tedy libovolná binární data. Typy `float` a `double` zatím podporovány nejsou. Každá hodnota je asociována s jedním unikátním klíčem, což je až 15místný ASCII řetězec. Tyto klíče budou reprezentovat parametry firmwaru [4].

Součástí ESP-IDF je knihovna NVS Partition Generator Utility, která dokáže z dvojic klíč-hodnota uložených v CSV souboru vygenerovat binární soubor, tzv. NVS image, který je kompatibilní s NVS architekturou a lze ho nahrát do flash paměti. Pro příklad takového CSV viz Soubor 1.

Soubor 1 Definice NVS klíčů

```
key,type,encoding,value
namespace_1,namespace,,
key_11,data,i8,5
key_12,file,string,/path/to/file
namespace_2,namespace,,
key_21,data,string,string_value
key_22,file,binary,/path/to/another/file
```

2.8.2 Konfigurační formulář

Úkolem programátora bude vytvořit konfigurační formulář, což bude šablona, ze které bude frontend generovat HTML formulář, prostřednictvím kterého budou uživatelé specifikovat hodnoty jednotlivých parametrů firmwaru. Konfigurační formulář bude obsahovat seznam klíčů, které jeho firmware očekává v NVS. Bylo by možné použít podobnou strukturu a formát, který se používá pro specifikaci obsahu NVS oddílu [5], CSV ale není pro konfigurační data vhodný, zejména kvůli neschopnosti reprezentovat strukturovaná data a absenci datových typů. Z tohoto důvodu bude vhodnější JSON, který je navíc dobře použitelný ve webovém prostředí.

Struktura formuláře Formulář bude strukturován jako pole objektů, kde každý objekt bude reprezentovat jeden parametr. Každý objekt bude obsahovat sadu položek, pomocí kterých se budou specifikovat vlastnosti daného parametru. Součástí budou položky, které jsou nezbytné pro převedení formuláře do CSV - `key`, `type` a `encoding`, tak i položky, které umožní programátorovi lépe kontrolovat hodnoty parametrů. Následující sadu položek budou muset, nebo smět obsahovat všechny objekty ve formuláři.

- `key` - Klíč, pod kterým bude hodnota dostupná v NVS. Povinná položka.

- `type` - Způsob zadávání hodnoty do formuláře. Možnými hodnotami budou `data`, kdy uživatel bude muset hodnotu zadat přímo, a `file`, kdy bude jako hodnota použit obsah souboru. Povinná položka.
- `encoding` - Datový typ hodnoty. Povinná položka.
- `isRequired` - Určuje, zda je uživatel povinen zadat hodnotu. Povinná položka.
- `label` - Název, pod kterým uživatel uvidí daný parametr v HTML formuláři. Povinná položka.
- `description` - Popis parametru, který uživatel uvidí v HTML formuláři. Volitelná položka.

Následující položky bude možné použít, pokud bude `type` nastaven na `data` a `encoding` na `string`.

- `options` - Seznam přípustných textových hodnot. Vylučuje současné použití položky `pattern`. Volitelná položka.
- `pattern` - Regulární výraz, který přijímá pouze povolené hodnoty. Výraz musí být kompatibilní s konstruktorem javascriptové třídy `RegExp`. Vylučuje současné použití položky `options`. Volitelná položka.

Pro použití následujících položek bude nutné nastavit `type` na `data` a `encoding` na libovolný číselný typ, např. `u8`, `i64` atd.

- `options` - Seznam přípustných celočíselných hodnot. Vylučuje současné použití položek `minimum` nebo `maximum`. Volitelná položka.
- `minimum` - Hodnota parametru musí být větší nebo rovna dané hodnotě. Vylučuje současné použití položky `options`. Volitelná položka.
- `maximum` - Hodnota parametru musí být menší nebo rovna dané hodnotě. Vylučuje současné použití položky `options`. Volitelná položka.

Poslední položku bude možné použít pouze při nahrávání hodnoty prostřednictím souboru, tedy když bude `type` nastaven na `file`.

- `format` - Media type daného souboru. Volitelná položka.

Přesný formát formuláře bude nutné popsat exaktně, z toho důvodů vznikne schéma pomocí slovníku JSON Schema [6], která bude sloužit jako dokumentace pro programátory. Schéma využijeme i při nahrávání formulářů do systému, kdy bude nezbytná validace na backendu. Ruční definování validačních pravidel by bylo kvůli složité struktuře formulářů nepraktické, použití schématu v kombinaci s vhodným validátorem však celý proces zjednoduší.

2.8.3 Průběh konfigurace

Konfigurace firmwaru bude probíhat těsně před začátkem jeho nahrávání do desky podle následujícího schématu.

1. Backend odešle na frontend konfigurační formulář.
2. Frontend vygeneruje z konfiguračního formuláře HTML formulář.
3. Uživatel vyplní HTML formulář a odešle jej zpět na backend.
4. Backend z příchozích hodnot a konfiguračního formuláře vytvoří CSV, ze kterého pak pomocí NVS Partition Generator Utility vygeneruje NVS image.
5. Spustí se nahrávání firmwaru a konfigurace do desky 2.9.

2.8.4 Konfigurace ESP Managera

Konfigurace *ESP Managera* bude probíhat během registrace desky a bude probíhat stejným způsobem jako konfigurace firmwaru 2.8.3. Jeho konfigurační formulář bude obsahovat přistupové údaje k Wi-Fi, tj. SSID a heslo.

Do *ESP Managera* se bude navíc předávat několik interních parametrů. Tyto parametry nebudou součástí formuláře, místo toho je bude nastavovat backend těsně před zavoláním NVS Partition Generator Utility. Mezi těmito parametry bude identifikátor desky, kterým se bude prokazovat při komunikaci s backendem a MQTT brokerem, a identifikátor spuštěného firmwaru s číslem jeho verze, které boudou sloužit k synchronizaci mezi deskou a backendem.

Tato konfigurace bude uložena v samostatném NVS oddílu, neboť ji narozdíl od konfigurace uživatelského firmwaru nebude nutné aktualizovat.

2.9 Nahrávání a aktualizace firmwaru

Systém uživatelům umožní nahrávat do svých desek firmware od jiných uživatelů. Pokud bude firmware konfigurovatelný, bude systém muset do desky nahrát i NVS image obsahující jeho konfiguraci. Když programátor zveřejní aktualizaci firmwaru, bude možné do desky tuto novou verzi nahrát.

Otázkou zůstává, zda by systém měl umožnit programátorům aktualizovat i konfigurační formulář. Je žádoucí, aby po nahrání firmwaru systém vyžadoval co nejméně akcí ze strany uživatele, aktualizace formuláře by ale vyžadovala jeho opětovné vyplnění. Z tohoto důvodu se aktualizace konfiguračního formuláře nejeví jako vhodné řešení, a systém ji tak nebude podporovat.

Firmware lze do desky nahrát lokálně pomocí USB kabelu a nástroje esptool nebo esptool-js 2.12. Druhou možností je firmware nahrávat vzdáleně pomocí mechanismu OTA (Over The Air), jež je v ESP-IDF k dispozici. Výhodou OTA mechanismu je, že nevyžaduje přerušení běžícího firmwaru ani fyzický přístup k desce, proto pro nahrávání a aktualizace zvolíme právě tuto variantu. Důsledkem této volby je potřeba nepřetržitého spojení mezi *ESP Managerem* a backendem 2.10, který mu bude zasílat pokyny k zahájení OTA aktualizace.

2.9.1 OTA

ESP-IDF nabízí pro OTA aktualizace rozsáhlé API [7]. Každá deska využívající toto API musí mít ve flash paměti jeden datový OTA oddíl pro ukládání metadat a alespoň dva OTA aplikační oddíly pro ukládání firmwaru. Vzhledem k omezené velikosti flash paměti bude systém při registraci desky vytvářet v její paměti jen dva aplikační oddíly - jeden pro aktuální firmware a jeden pro jeho případnou aktualizaci. Výhodou tohoto API je jeho jednoduchost, neboť lze celou aktualizaci provést jedinou funkcí - `esp_https_ota`, která automaticky zvaliduje image, uloží jej do volného oddílu a nastaví jej jako výchozí aplikaci.

Vzdáleně bude třeba nahrávat i NVS image s konfigurací. OTA API ale umí pracovat pouze s OTA aplikačními oddíly, proto bude nutné tuto funkcionalitu implementovat jinak. K tomu využijeme funkci `esp_http_client_read`, která bude image načítat po bajtech, které bude funkce `esp_partition_write` ukládat do příslušného NVS oddílu.

2.10 Síťová komunikace

Pro nahrávání a aktualizace firmwaru byl zvolen mechanismus OTA realizovaný pomocí HTTP. V návaznosti na tuto volbu je nutné rozhodnout, jakým způsobem zasílat desce pokyny k zahájení jedné z těchto operací. Z bezpečnostních důvodů není žádoucí, aby byla deska dostupná z internetu, je proto třeba využít protokol umožňující obousměrnou komunikaci, kterou lze inicializovat ze strany *ESP Managera*. Kandidáty jsou tak protokoly WebSocket a MQTT, pro něž je v ESP-IDF implementován klient.

Přednost dostal MQTT, což je otevřený protokol navržený pro oblast IoT, který implementuje návrhový vzor publisher/subscriber. Klienti tedy spolu nekomunikují přímo, ale prostřednictvím centrálního serveru (brokeru). Server přijímá zprávy od vydavatelů (publishers), třídí je podle témat (topics), jímž je každá zpráva označena, a přeposílá je všem odběratelům (subscribers), kteří jsou k odběru daného tématu přihlášeni. Klient může u libovolného tématu vystupovat jako vydavatel a odběratel zároveň, což je způsob, jakým lze realizovat obousměrnou komunikaci mezi libovolnou dvojicí zařízení.

MQTT má v porovnání s WebSocket nižší systémové nároky a kompaktnější formátování zpráv. Navíc podporuje LWT (Last Will Message and Testament), díky čemuž bude snadné detekovat ztrátu spojení mezi deskou a backendem.

2.11 Flash paměť

V předchozích sekcí bylo rozhodnuto, že flash paměť každé desky bude obsahovat alespoň dva NVS oddíly, dva aplikační OTA oddíly a jeden datový OTA oddíl. Součástí bude muset být i bootloader, který zavádí firwmare z flash paměti do operační paměti. Adresu firmwaru bootloader načítá z tabulky oddílů [8], která je dalším nezbytným oddílem. Tabulka oddílů specifikuje typy dostupných oddílů, jejich velikost a umístění ve flash paměti.

Otázkou zůstává, zda pro *výchozí firmware* vyčlenit samostatný oddíl, nebo jej nahrávat do jednoho z OTA oddílu. Výhodou použití OTA oddílu by bylo efektivnější využití paměti, místo vyhrazené pro *výchozí firmware* by totiž bylo

možné rozdělit mezi dva OTA oddíly, což by zvýšilo limit na maximální velikost uživatelského firmwaru. Samostatný oddíl pro *výchozí firmware* by naopak zajistil jeho trvalou dostupnost na desce. To by bylo výhodné v situacích, kdy by uživatel chtěl desku nechat zaregistrovanou v systému, ale zároveň by na ní nechtěl mít spuštěný žádný uživatelský firmware. ESP-IDF poskytuje API, které umožňuje za běhu změnit bootovací firmware i bez OTA aktualizace a provést restart systému. Z těchto důvodů bude systém využivat druhou variantu – samostatný oddíl pro *výchozí firmware*, přičemž tento oddíl bude mít minimální možnou velikost, aby zbylo co nejvíce místa pro OTA oddíly.

Bude nutné vytvořit pět verzí tabulky, jednu pro každou z možných velikostí flash paměti, která může být 2 MB až 32 MB. Tabulky se zapisují ve formátu CSV a je nutné je před nahráním do desky zkompilovat pomocí nástroje parttool. Například pro verzi tabulky pro 8MB paměť viz Soubor 2. Struktura ostatních tabulek zůstane stejná, pouze se bude měnit velikost OTA oddílů tak, aby byla vždy využita celá paměť.

Soubor 2 Tabulka oddílů pro 8MB flash paměť

```
# Name,    Type, SubType, Offset,    Size
nvs,       data, nvs,     0x9000,    0x5000
otadata,   data, ota,     0xe000,    0x2000
factory,   app,  factory, 0x10000,   0x100000
ota_0,     app,  ota_0,   0x110000,  0x370000
ota_1,     app,  ota_1,   0x480000,  0x370000
config,    data, nvs,     0x7F0000,  0x10000
```

V oddílu `factory` bude uložen *výchozí firmware*, konfigurace *ESP Managera* bude v oddílu `nvs`. Uživatelský firmware se bude střídavě nahrávat do oddílů `ota_0` a `ota_1`, z čehož plyne, že nebude možné mezi těmito oddíly přepínat, jelikož bude přítomen pouze jeden oddíl pro konfiguraci, tj. oddíl `config`.

Umístění oddílů `nvs` a `factory` před aplikačními OTA nebylo zvoleno náhodně. Díky tomuto umístění se totiž nemění jejich offset napříč tabulkami, což výrazně usnadní nahrávání *výchozího firmwaru* 2.12.

V tabulkách není zahrnuta adresa bootloaderu a tabulky oddílů. Tabulka je vždy na offsetu `0x8000`, proto není nutné ji uvádět. Existují ale tři možná usmístění bootloaderu, která se liší napříč jednotlivými architekturami. Abychom nemuseli vytvářet $5 \cdot 3$ různých tabulek, budeme adresu bootloaderu specifikovat až při nahrávání *výchozího firmwaru*.

2.12 Nahrávání výchozího firmwaru

Pro čtení informací z ESP čipu a zápis do jeho flash paměti prostřednictvím sériové linky slouží otevřený a na platformě nezávislý nástroj esptool, napsaný v jazyku Python. V rané fázi projektu se uvažovalo o vzniku *nahrávací aplikace*, jednoduchého wrapperu nad esptool, který by z backendu *webové aplikace* automaticky stáhl všechna potřebná data, tj. bootloader, tabulku oddílů, *výchozí firmware* a NVS image s konfigurací *výchozího firmwaru*, a nahrál je do uživatelovy desky. Nevýhodou tohoto řešení by byla nutnost instalace aplikace na straně uživatele,

což je v rozporu s požadavkem na minimalizaci technických nároků ze strany uživatelů vývojových desek. Uživatelé by navíc museli své desky obsluhovat ze dvou různých aplikací, což by snižovalo uživatelský komfort. Z těchto důvodů *nahrávací aplikace* nebude součástí systému. Místo toho budou její funkce integrovány přímo do uživatelského rozhraní *webové aplikace* pomocí nástroje esptool-js.

2.12.1 esptool-js

esptool-js je javascriptová implementace nástroje esptool, určená pro webové prohlížeče a prostředí Node.js. Implementace je postavena nad experimentálním Web Serial API [1], které umožňuje sériovou komunikaci mezi prohlížečem a deskou. Přestože je Web Serial API podporováno pouze v prohlížečích Chrome, Edge a Opera, je toto omezení akceptovatelné, neboť Chrome je nejrozšířenějším webovým prohlížečem a Edge je výchozím prohlížečem systému Windows, nejrozšířenějšího operačního systému.

Data se do desky nahrávají pomocí metody `writeFlash`. Ta jako parametr přijímá objekt typu `FlashOptions`, kde je nutné specifikovat soubory, které se mají do desky nahrát a offset, na který má být každý ze souborů uložen. Díky vhodně zvolené organizaci flash paměti se budou všechny soubory bez ohledu na její velikost nahrávat vždy na fixní offset. Výjimkou bude bootloader, o jehož adrese bude nutné rozhodnout na základě typu architektury.

Bootloader vyžaduje informaci o velikosti flash paměti, proto `FlashOptions` obsahuje položku `flashSize`, kterou by interní funkce `_updateImageFlashParams` měla využít k nastavení příslušných bajtů v bootloaderu. Při analýze knihovny jsme však odhalili chybu v její implementaci, kvůli které se nelze na položku `flashSize` spolenout. `FlashOptions` požaduje, aby byl každý image včetně bootloaderu předán jako `string`. Na část bootloaderu, ve které je specifikována velikosti flash paměti, je pak aplikována funkce `parseInt`, která v důsledku špatného formátu selhává. Z toho plyne, že na backendu bude muset být uložen zvláštní bootloader pro všechny kombinace čipu a velikosti flash paměti.

2.13 Zabezpečení

Komunikace se systémem bude probíhat prostřednictvím protokolů HTTP a MQTT. V obou případech bude nutné provádět autentizaci. Bude nutné zajistit, aby uživatelé měli přístup pouze ke svým deskám a firmwarům a desky byly oprávněny stahovat pouze konfiguraci, kterou vytvořil jejich provozovatel. MQTT broker zas bude muset kontrolovat, aby se připojovaly pouze desky zaregistrované do systému.

Komunikace mezi frontendem a backendem bude zabezpečena pomocí JWT (JSON Web Token). Token bude vystavován jako `HttpOnly` cookie, což zajistí, že bude automaticky připojen v hlavičce každého požadavku. Tím se výrazně usnadní implementace frontendu a zároveň se tím zvýší bezpečnost, jelikož token nebude přístupný z JavaScriptu.

Pro OTA aktualizace bude dostatečná Basic Access Authentication. Jako uživatelské jméno bude *ESP Manager* používat identifikátor desky, heslo bude nutné vygenerovat a přidat do jeho konfigurace během registrace desky.

Přístup k MQTT brokeru bude rovněž umožněn pouze deskám s platnými přihlašovacími údaji. Uživatelské jméno bude opět identifikátor desky, heslo bude opět vygenerováno během registrace desky. MQTT broker navíc dokáže pomocí tzv. ACL (Access Control List) omezit přístup klientům k vybraným tématům. Backend bude mít přístup ke všem tématům, zatímco každé desce bude pomocí ACL povolen přístup výhradně k jednomu tématu, ke kterému bude mít přístup už jen backend.

Autentizační údaje při Basic Access Authentication nebo data posílaná přes MQTT nejsou šifrována, proto bude nutné přidat šifrování pomocí TLS. Použití TLS vyžaduje, aby MQTT broker i backend disponovaly digitálním certifikátem, který bude zároveň nutné přiřadit ke konfiguraci *ESP Managera*, aby mohl oba servery ověřovat.

3 Implementace

Systém je složen ze tří samostatných projektů. Tou první je *webová aplikace*, která je uložena v adresáři `webapp/`, a její architektura je rozebrána v sekci 3.1. Implementace vybraných komponent *webové aplikace* jsou pak popsány v několika následujících sekcích. Druhým projektem je *ESP Manager*, jež je uložen v adresáři `esp_manager/` a jeho implementaci se věnuje sekce 3.5. Posledním projektem je *výchozí firmware* v adresáři `default_app/`, o kterém pojednává sekce 3.6.

3.1 Architektura webové aplikace

Webová aplikace je složena z pěti Docker kontejnerů a má dva dostupné režimy - vývojářský a produkční, které se liší v počtu spuštěných kontejnerů a jejich nastavením.

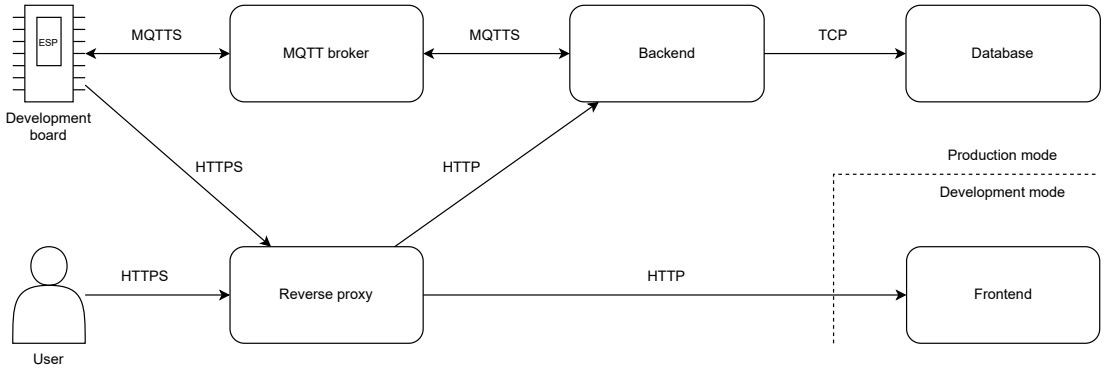
Backend Backend je HTTP server implementován v Node.js, který zajišťuje většinu logiky celého systému. Backend pracuje s databází, do níž ukládá informace o deskách, firmwarech a uživatelích, a se souborovým systémem, kam ukládá firmwary, konfigurační soubory apod. Součástí backendu je rovněž MQTT klient, prostřednictvím kterého komunikuje s deskami zaregistrovanými do systému.

Frontend Uživatelské rozhraní je vytvořené pomocí frameworku Vue.js. Ve vývojářském režimu běží frontend jako samostatný kontejner s vlastním development serverem, v produkčním režimu je aplikace zkompilována pomocí buildovacího nástroje Vite do statických souborů a uložena na reversním proxy.

Reverse proxy Veškeré HTTP požadavky jsou směrovány na reversní proxy, kterou zajišťuje server nginx. Požadavky směrované na API endpointy s prefixem `/api/` jsou přesměrovávány na backend, zbylé požadavky jsou ve vývojářském režimu směrovány na frontend, v produkčním režimu je obsluhuje přímo nginx. Nginx zároveň zajišťuje správu SSL certifikátů pro komunikaci přes HTTPS a úpravu cookie cest před přeposláním požadavků na backend.

MQTT broker Jako MQTT broker byl zvolen Mosquitto, zejména kvůli Dynamic Security Plugin, který nabízí jednoduché API pro programatickou kontrolu autentizace uživatelů a jejich přístupových práv.

Database Pro ukládání management dat byla zvolena relační databáze MariaDB. V rané verzi projektu nebyl pro databázi vyhrazen samostatný kontejner, místo toho se používala databáze SQLite nainstalovaná přímo na backendu, avšak vzhledem k její omezené škálovatelnosti bylo rozhodnuto o přechodu na samostatný kontejner s MariaDB. Databáze se spouští vždy jako první, protože jsou na ní (transitivně) závislé zbylé kontejnery. Z tohoto důvodu se v definici kontejneru (soubor `compose.yaml`) využívá direktiva `healthcheck`, která kontroluje, že databáze je nejen spuštěna, ale i připravena přijímat spojení.



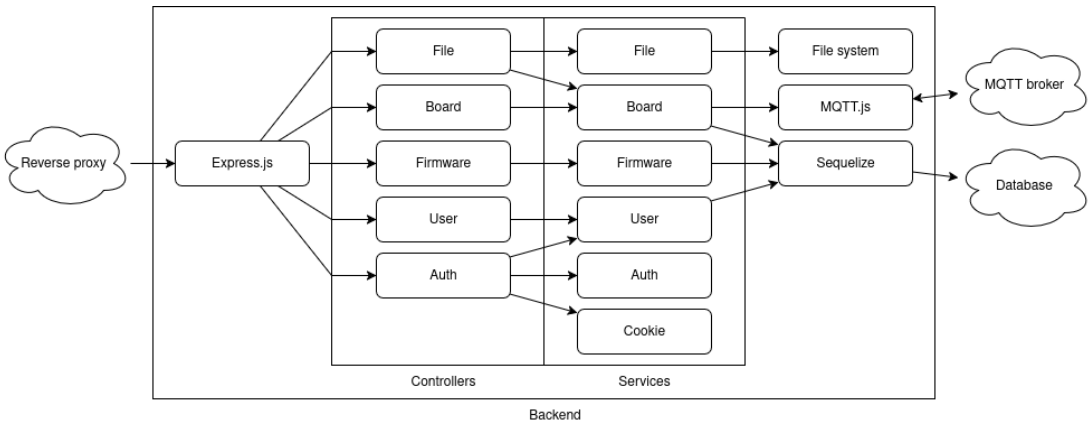
Obrázek 3.1 Architektura webové aplikace

3.2 Backend

Pro obsluhu HTTP požadavků backend používá framework Express.js. Vstupní bod aplikace je uložen v souboru `index.js`, ve kterém probíhá načítání konfigurace a připojování k databázi 3.2.2 a MQTT brokeru 3.2.6. Inicializace Express serveru se odehrává v souboru `api/index.js`, kde jsou namapovány všechny API endpointy 3.2.1 na jednotlivé kontrolery z adresáře `api/controllers`. Kontrolery zpracovávají HTTP požadavky, parsují příchozí data a ta předávají servisům definovaných v adresáři `logic/services`, ve kterých je implementována veškerá aplikační logika. Pro ucelený pohled na architekturu a závislosti jednotlivých komponent viz Obrázek 3.2.

3.2.1 API

Všechny API endpointy jsou definovány ve sdíleném NPM balíčku 3.4. Detailní dokumentace jednotlivých API endpointů, včetně příkladů požadavků a odpovědí, je k dispozici v OpenAPI formátu v příloze práce v souboru `webapp/openapi.yaml`.



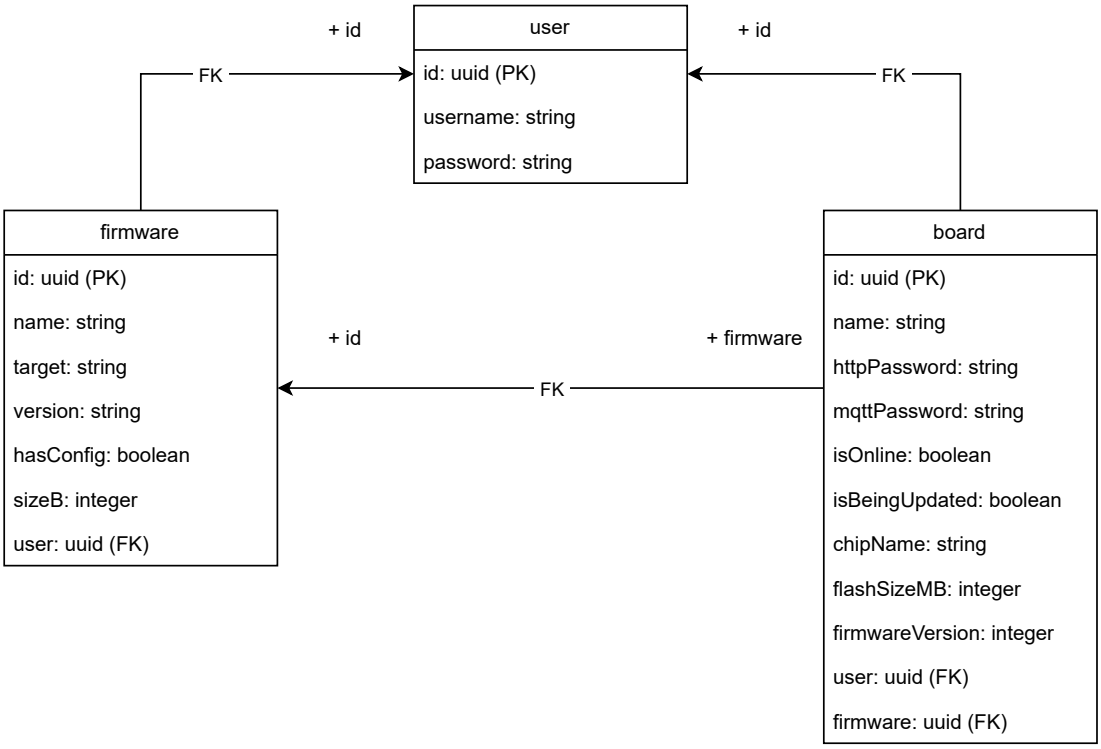
Obrázek 3.2 Architektura backendu

3.2.2 Databáze

Pro komunikaci s databází MariaDB se používá ORM knihovna Sequelize. Sequelize vyžaduje definovat tzv. modely, ze kterých vytvoří tabulky, a pomocí kterých mapuje jejich řádky na javascriptové objekty. Definice všech modelů jsou uloženy v adresáři `db/models/` a vycházejí z databázového schématu na Obrázku 3.3.

Při návrhu databáze bylo nutné vyřešit, jak nakládat s deskami, na kterých běží firmware, který se jeho autor rozhodl odstranit ze systému. Firmware nebylo možné zcela smazat, protože jej někteří uživatelé stále využívají. Z toho důvodu využívá model firmwaru funkci `paranoid`. Sequelize záznamy z takto označených tabulek fyzicky nemaže, ale pouze je označuje jako smazané.

Zároveň však bylo žádoucí, aby se v databázi a souborovém systému nehromadily již nepotřebná data. Proto třída `FirmwareService` obsahuje metodu `tryForceDelete` (viz Program 1), která definitivně smaže záznam z databáze a celý adresář příslušící k firmwaru jen v případě, že tato operace neovlivní žádnou z desek.



Obrázek 3.3 Databázové schéma

Program 1 Metoda pro smazání firmwaru

```
async tryForceDelete(firmwareId) {

    const firmware = await this._db.models.firmware.
        findByPk(firmwareId, { paranoid: false });

    // Must be soft deleted and must not be flashed

    if (firmware &&
        firmware.deletedAt !== null &&
        !await this._isFlashed(firmwareId)) {

        await firmware.destroy({ force: true });

        // No board is using this firmware anymore
        // so we can delete its directory

        await this._file.tryDeleteFirmwareDir(firmwareId);
    }
}
```

3.2.3 Ukládání souborů

Všechny soubory se ukládají do adresáře `data/`, který je dále struktuován do adresářů `firmwares/`, `boards/` a `default/`. Adresář `default/` obsahuje společné soubory, jako jsou např. konfigurační formulář pro *výchozí firmware* a zkompilované *výchozí firmwary* pro jednotlivé architektury.

Se souborovým systémem a soubory pracuje výhradně třída `FileService`. Ta pro každou desku vytváří v adresáři `boards/` adresář `<board id>/`, do kterého ukládá všechny soubory příslušící dané desce, tj. NVS image s konfigurací *výchozího firmaru* atd. Obdobně pro každý firmware vytváří v adresáři `firmwares/` adresář `<firmware id>/`, kde je uložen zkompilovaný firmware a případně i jeho konfigurační formulář.

3.2.4 Validace konfiguračních souborů

Validaci konfiguračních souborů provádí třída `FileService` pomocí JSON validátoru Ajv. JSON Schema popisující strukturu konfiguračního formuláře je uloženo v souboru `data/default/config-form.schema.json`.

3.2.5 Generování NVS

Konfigurační soubory jsou transformovány do formátu kompatibilního s ESP-IDF pomocí metod `createNVS` a `createDefaultNVS`, které jsou definovány ve třídě `FileService`. Obě metody nejprve volají metodu `_saveConfigDataAsCSV`, která kombinuje informace z konfiguračního formuláře (položky `key`, `type` a `encoding`) a hodnoty od uživatele, a ukládá vše do CSV podle předepsaného formátu [5]. Následně je volána funkce `_createNVS`, která převádí CSV na NVS image. Pro převod se využívá utilita NVS Partition Generator Utility, implementována v jazyce

Python, z toho důvodu `_createNVS` musí utilitu spouštět jako externí proces pomocí funkce `spawn`.

`createNVS` generuje NVS image k uživatelským firmwarům, konfiguraci pro *ESP Managera* generuje `createDefaultNVS`, z toho důvodu přidává do konfigurace několik interních parametrů (rozšiřuje objekt `configData`, který drží hodnoty, a konfigurační formulář v proměnné `defaultConfigForm`), např. identifikátor desky nebo adresu MQTT brokeru (viz Program 2).

Program 2 Interní rozšíření výchozího konfiguračního formuláře

```
configData.id = board.id;
defaultConfigForm.push({
    key: "id", type: "data", encoding: "string"
});
```

3.2.6 Komunikace s ESP Managerem

Komunikace s *ESP Managerem* probíhá prostřednictvím knihovny MQTT.js. Mosquitto broker používá Dynamic Security Plugin [9], díky kterému lze programaticky spravovat seznam uživatelů a jejich přístup k jednotlivým tématům. Plugin se ovládá prostřednictím vyhrazeného tématu `$CONTROL/dynamic-security/v1`, do kterého má právo publikovat pouze klient s administrátorskou rolí, která je v našem případě přidělena backendu. Backend se k brokeru připojuje ihned po spuštění pomocí funkce `getMqtt`. Vytváření uživatelů, tj. desek s oprávněním připojit se k brokeru, zajišťuje metoda `_createMqttClient` definovaná ve třídě `BoardService`. Každé desce je pomocí pluginu povoleno komunikovat (publikovat a odebírat) výhradně prostřednictvím témat `<board id>/#`. Backend odesílá deskám zprávy prostřednictvím následujících témat.

- `<board id>/cmd/update` - Pokyn k provedení OTA aktualizace. Zpráva je formátována jako JSON objekt s položkami `firmware_id`, `version`, `firmware_url` a pokud se nahrává firmware vyžadující konfiguraci, tak ještě `config_url`.
- `<board id>/cmd/boot-default` - Pokyn ke spuštění *výchozího firmwaru*.

Desky odesílají na backend zprávy prostřednictvím následujících témat.

- `<board id>/info/online` - *ESP Manager* se připojil k brokeru.
- `<board id>/info/offline` - *ESP Manager* se odpojil od brokeru.
- `<board id>/info/update/ok` - OTA aktualizace proběhla úspěšně, nová verze firmwaru / nový firmware bude spuštěn po restartování desky. Zpráva je formátována jako JSON objekt s položkami `firmware_id` a `version`, které popisují nový firmware.
- `<board id>/info/update/error` - OTA aktualizace se nezdařila, stará verze firmwaru / starý firmware pokračuje v běhu. Zpráva je formátována jako JSON objekt s položkami `firmware_id` a `version`, které popisují starý firmware.

3.2.7 Bezpečnost

S výjimkou API endpointů pro registraci a přihlášení jsou veškeré ostatní endpointy chráněny pomocí některé z middleware funkcí definovaných v adresáři `api/middlewares/`. Tyto funkce zpracovávají každý příchozí požadavek a kontrolují jeho oprávněnost před tím, než je předán příslušnému kontroleru.

Endpointy, na které se dotazuje frontend, jsou zabezpečeny pomocí přístupových a obnovovacích tokenů založených na standardu JWT. Tyto tokeny vydává třída `AuthService`, třída `CookieService` je nastavuje jako `HttpOnly` cookies při sestavování odpovědi. Uživatel tyto tokeny získává při přihlášení nebo registraci. Platnost přístupového tokenu je aktuálně nastavena na 10 minut, platnost nového obnovovacího tokenu činí nyní 24 hodin. K obnově tokenů slouží speciální API endpoint `/auth/refresh`, který v případě validního obnovovacího tokenu vydává novou dvojici tokenů.

Vývojové desky stahují firmware z endpointu `/files/firmware/{firmwareId}` a NVS image z endpointu `/files/nvs/{boardId}`. Tyto endpointy jsou zabezpečeny pomocí metody Basic Access Authentication. Přihlašovací údaje, které *ESP Manager* k tomuto účelu používá, jsou do jeho NVS přidávány jako interní parametr vygenerovaný přímo backendem (viz část 3.2.5).

3.3 Frontend

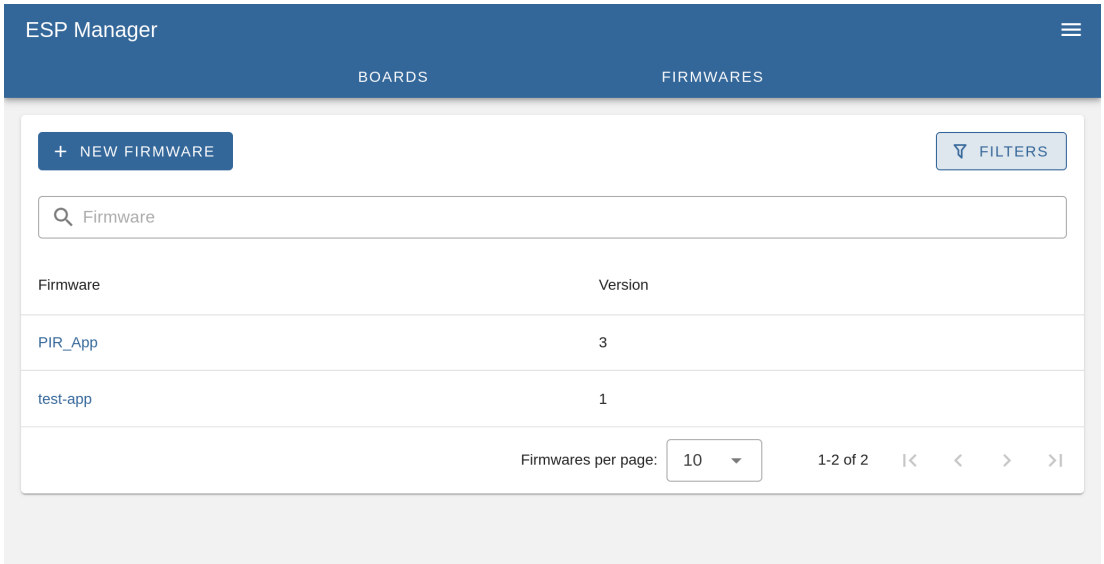
Frontend je realizován jako single-page aplikace vytvořená pomocí frameworku Vue.js, jež je založen na komponentovém systému. Výhodou tohoto frameworku je možnost psát kód v tzv. SFC (Single-File Component) formátu, který umožňuje psát šablonu, logiku a styly dané komponenty do jediného souboru, což zvyšuje jeho přehlednost. Pro vytvoření uživatelského rozhraní byl dále použit komponentový framework Vuetify, který poskytuje předpřipravenou a designově konzistentní sadu komponent, což urychluje vývoj a zajišťuje vizuální jednotnost.

Frontend je struktuován jako standardní Vue.js projekt, tj. adresář `views/` obsahuje hlavní komponenty, které reprezentují jednotlivé stránky aplikace, adresář `components/` obsahuje menší a znovupoužitelné komponenty, ze kterých jsou sestaveny hlavní komponenty.

3.3.1 Obrazovky programátora

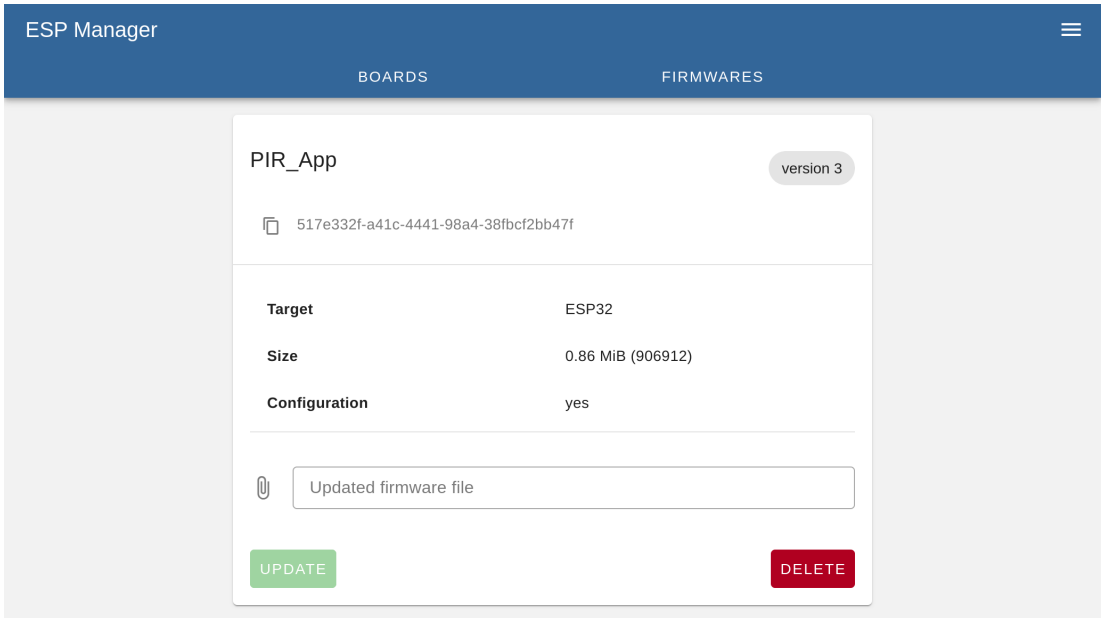
Pro programátory je určena následující sada obrazovek.

Seznam nahraných firmwarů Tato obrazovka zobrazuje seznam všech firmwarů, které daný programátor do systému nahrál. Seznam podporuje filtrování podle názvu firmwaru a stránkování. Jednotlivé položky seznamu jsou interaktivní, kliknutím na název konkrétního firmwaru je programátor přesměrován na detail firmwaru 3.5. Tlačítkem `NEW FIRMWARE` se lze dostat na obrazovku určenou pro nahrání nového firmwaru 4.1.



Obrázek 3.4 Seznam nahraných firmwarů

Detail firmwaru Funkcionalita této obrazovky odpovídá návrhu uvedenému v části 2.5.1.

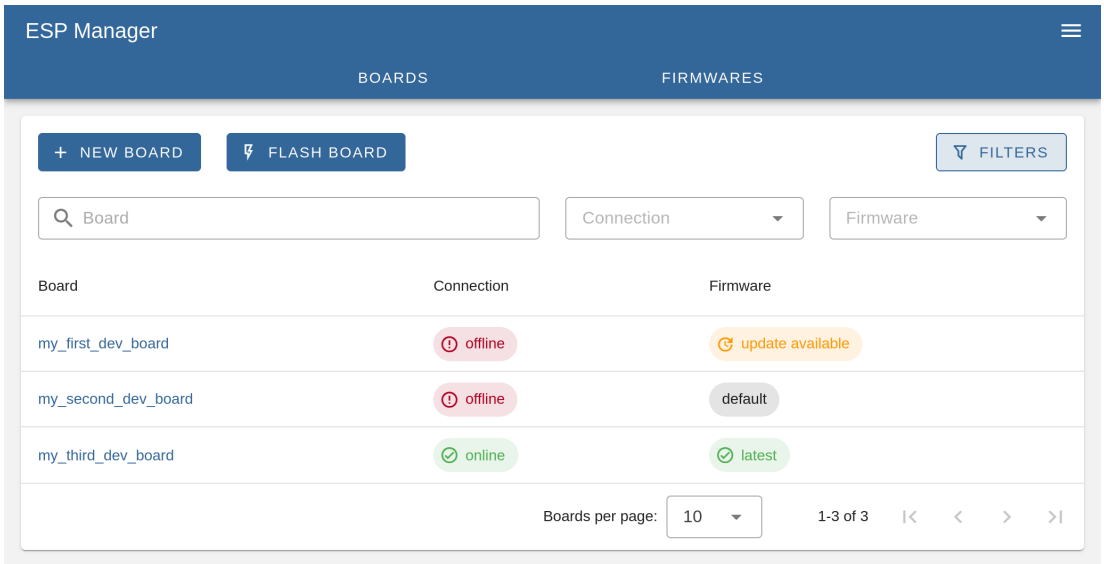


Obrázek 3.5 Detail firmwaru

3.3.2 Obrazovky uživatele vývojových desek

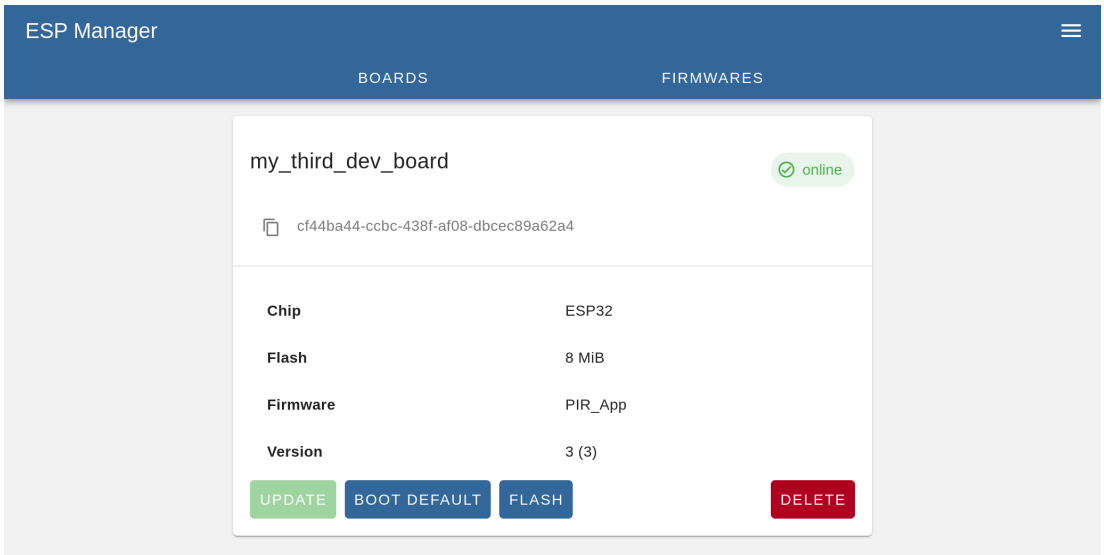
Následující obrazovky jsou relevantní pro uživatele vývojových desek.

Seznam vývojových desek Tato obrazovka zobrazuje seznam všech vývojových desek, které uživatel zaregistroval do systému. U každé desky je zobrazen aktuální stav připojení a informace o firmwaru, který je na ní momentálně nahrán. Desky lze filtrovat podle názvu, stavu připojení a stavu nahraného firmwaru. Kliknutím na název desky je uživatel přesměrován na její detail 3.7. Tlačítkem `NEW BOARD` se lze dostat na obrazovku určenou pro registraci nové desky, tlačítkem `FLASH BOARD` uživatel zahájí nahrávání firmwaru do desky 4.3.



Obrázek 3.6 Seznam vývojových desek

Detail vývojové desky Tato obrazovka odpovídá návrhu z části 2.5.2. Tlačítko `BOOT DEFAULT` je aktivní pouze v případě, že je deska online a současně je na ní nahrán nějaký uživatelský firmware. Tlačítko `UPDATE` je aktivní pouze v případě, že je deska online a současně je na ní nainstalována jiná než aktuální verze nějakého uživatelského firmwaru.



Obrázek 3.7 Detail vývojové desky

Program 3 Definice navigační stráže

```
router.beforeEach(async (to, from) => {

    if (from === START_LOCATION) {

        await useSessionStore().init();
    }

    const isLoggedIn = useSessionStore().isLoggedIn;
    const isAuthRequired = to.meta.isAuthRequired;

    if (isLoggedIn && (!isAuthRequired || to.path === "/")) {

        return { name: "Boards" };
    }

    if (!isLoggedIn && (isAuthRequired || to.path === "/")) {

        return { name: "LogIn" };
    }
});
```

3.3.3 Navigace

Hlavní navigační panel je implementován v komponentě `TheNavigation.vue` a slouží k přechodům mezi částí určenou pro programátory a částí určenou pro uživatele vývojových desek. Navigace mezi jednotlivými stránkami je zajištěna knihovnou `vue-router`.

Instance routeru je vytvořena v souboru `router/index.js`, kde jsou definovány stránky aplikace formou mapování URL cest na komponenty z adresáře `views/`. Každá definovaná cesta je navíc označena položkou `isAuthRequired`, která určuje, zda je nutné se pro zobrazení dané stránky přihlásit. Tuto položku využívá tzv. navigační stráž, která před každým přechodem mezi stránkami kontroluje, zda je uživatel oprávněn si danou stránku zobrazit 3.

3.3.4 Komunikace s backendem

Frontend komunikuje s backendem výhradně prostřednictvím HTTP, a to pomocí požadavků směrovaných na jeho REST API. K odesílání těchto požadavků slouží třída `HTTPClient`, která je postavena nad knihovnou axios. Instanci této třídy využívají třídy definované v adresáři `api/resources/`, jako např. `AuthApi` nebo `BoardApi`. Tyto třídy obsahují metody odpovídající jednotlivým API endpointům a slouží jako abstrakční vrstva mezi komponentami a třídou `HttpClient`. Komponenty přistupují k těmto třídám prostřednictvím třídy `ApiProvider`, která je dostupná v globálním úložišti 3.3.5.

3.3.5 Globální úložiště

Veškerá data týkající se aktuálně přihlášeného uživatele jsou uložena v globálním úložišti, které je implementováno pomocí knihovny Pinia, což je oficiální nástroj pro sdílení stavů napříč komponentami. Frontend používá jedno úložiště, které je definováno v souboru `stores/session.js`.

Důležitou metodou tohoto úložiště je `init`, která inicializuje instanci třídy `ApiProvider` a provádí testovací dotaz na backend pro získání uživatelských dat. Tento dotaz slouží k ověření, zda je uživatel přihlášen, nebo zda je nutné jej přesměrovat na přihlašovací obrazovku. Metoda se volá v navigační stáži při prvním načtení uživatelského rozhraní (viz Program 3).

3.3.6 Konfigurační formulář

Převod z konfiguračního formuláře na HTML formulář zajišťuje komponenta `ConfigForm.vue` Tato komponenta je parametrizovaná samotným konfiguračním formulářem a na základě typu jeho položek (položky `type` a `encoding`) dynamicky generuje odpovídající subkomponenty, např. `ConfigFormFileInput.vue` nebo `ConfigFormNumberInput.vue`. Každá z těchto specializovaných komponent generuje odpovídající HTML vstupní prvek a správnou sadu validačních pravidel.

3.3.7 Sériová komunikace s deskou

Sériová komunikace s deskou probíhá během její registrace a je nezbytná pro nahrání *výchozího firmwaru* do desky. Sériovou komunikaci s deskou zabezpečuje třída `Flasher` definovaná v souboru `/utils/Flasher.js` a je využívána v komponentě `NewBoardView.vue`. `Flasher` je wrapper nad knihovnou esptool-js.

3.4 Společná část

Část kódu byla kvůli jeho současnému použití na backendu a frontendu vyčleněna do samostatného NPM balíčku, který je dostupný v adresáři `shared/`. Tento balíček je do obou kontejnerů zkopírován při jejich inicializaci (soubor `Dockerfile`) a přidán jako lokální závislost pomocí reference v souboru `package.json`. Balíček obsahuje definici API endpointů, které byly během vývoje častým předmětem změn, proto bylo výhodné je definovat na jednom centrálním místě.

Problém při této společné definici představovaly parametrizované endpointy, jako např. `/boards/{id}`, který je parametrizovaný identifikátorem desky. Backendový framework Express.js vyžaduje zápis parametrizovaných endpointů ve formátu `/boards/:id`, zatímco na frontendu je při odesílání požadavku nutné do URL dosadit konkrétní hodnotu. V kódu je to řešeno pomocí lambda funkcí s výchozími parametry, které jsou při použití na frontendu volány s konkrétními hodnotami, které automaticky nahradí parametry v cestě. Na backendu jsou naopak volány bez argumentů, čímž ve výstupu zůstane potřebný placeholder vhodný pro Express.js (viz Program 4).

Program 4 Sepcifikace a použití API endpointů

```
// shared/src/index.js

boards: {

    supportedChips: () => "/boards/chips",
    all: () => "/boards",
    watchAll: () => "/boards/watch",
    one: (id=":id") => `/boards/${id}`,
    watchOne: (id=":id") => `/boards/${id}/watch`,
},

// frontend/src/api/resources/board.api.js

async getOne(boardId) {

    return await this._httpClient.get(endpoint.boards.one(boardId));
}

// backend/src/api/index.js

app.get(endpoint.boards.one(), boardController.getOne);
```

3.5 ESP Manager

Architektura *ESP Managera* je inspirována architekturou vybraných komponent z ESP-IDF, tj. obsahuje sadu inicializačních funkcí, které spouští hlavní smyčku, ve které komponenta reaguje na události z přidružených komponent.

3.5.1 Inicializace

Inicializace komponenty probíhá ve dvou krocích, obdobně jako u mnoha jiných komponent v ESP-IDF. Nejprve je volána funkce `esp_manager_init`, jejíž úkolem je načíst konfiguraci *ESP Managera* do dynamicky alokované struktury `esp_manager_client`. Funkce vrací tzv. opaque pointer na tuto strukturu, která slouží jako kontext pro další operace 5.

Druhým krokem je zavolání metody `esp_manager_start`, která spouští nový task (vlákno v terminologii ESP-IDF). V tomto tasku je spuštěna hlavní smyčka celé komponenty, kterou popisuje část 3.5.2.

Program 5 ESP Manager klient

```
struct esp_manager_client {

    char *id;
    char *wifi_ssid;
    char *wifi_password;
    char *http_password;
    char *mqtt_password;
    char *mqtt_broker_uri;
    char *ca_bundle;
    TaskHandle_t task_handle;
    QueueHandle_t queue_handle;
    esp_mqtt_client_handle_t mqtt_handle;
    esp_manager_client_state_t state;
    bool is_running;
};
```

3.5.2 Hlavní smyčka

Hlavní smyčka komponenty je implementována ve funkci `esp_manager_task`, která běží ve vlastním tasku jako nekonečný cyklus. *ESP Manager* se může nacházet v jednom ze tří stavů definovaných ve struktuře `esp_manager_client_state_t` 6. Každý stav odpovídá jedné fázi životního cyklu komponenty a má přiřazenu funkci, která daný stav obsluhuje.

`STATE_CONNECT_WIFI` V tomto stavu se *ESP Manager* pokouší připojit k Wi-Fi síti.

`STATE_CONNECT_MQTT` Jakmile je *ESP Manager* připojen k Wi-Fi síti, začíná připojování k MQTT brokeru. Při vytváření MQTT klienta je definováno tzv. last will téma `<board id>/info/offline`, díky kterému bude MQTT broker automaticky informovat backend v případě ztráty spojení. Ihned po připojení k brokeru je publikována zprávu do tématu `<board id>/info/offline`, čímž deska signalizuje backendu, že je připravena přijímat zprávy.

`STATE_RUN` V tomto stavu je *ESP Manager* již plně inicializován. Komponenta zpracovává příchozí zprávy od MQTT klienta a provádí OTA aktualizace.

Program 6 Stavy ESP Managera

```
typedef enum {

    STATE_CONNECT_WIFI,
    STATE_CONNECT_MQTT,
    STATE_RUN
} esp_manager_client_state_t;
```

Program 7 Struktura pro realizaci meziprocesové komunikace

```
typedef enum {

    EVENT_WIFI_CONNECTED,
    EVENT_MQTT_CONNECTED,
    EVENT_MQTT_ERROR,
    EVENT_UPDATE,
    EVENT_BOOT_DEFAULT
} esp_manager_event_id_t;

typedef struct {

    esp_manager_event_id_t id;
    void *data;
} esp_manager_event_t;
```

3.5.3 Meziprocesová komunikace

ESP Manager běží v samostatném tasku, aby nedocházelo k blokování uživatelského kódu ve vstupní funkci `app_main`, které běží v hlavním tasku. Tento task se interně připojuje k Wi-Fi síti a k MQTT brokeru, přičemž obě tato připojení vytváří svůj vlastní task. Mezi těmito tasky je nezbytná synchronizace, např. task *ESP Managera* potřebuje od Wi-Fi tasku informaci o úspěšém připojení, aby mohl inicializovat MQTT klienta. Podobně musí MQTT klient přeposílat *ESP Managerovi* všechny přijaté zprávy.

Pro zajištění meziprocesové komunikace bylo třeba zvolit synchronizační mechanismus, který umožňuje přenos libovolných dat, z toho důvodu byl zvolen synchronizační mechanismus fronta (`QueueHandle_t`). Zprávy jsou předávány prostřednictvím struktury `esp_manager_event_t`, která obsahuje identifikátor události a volitelný pointer na související data 7. Wi-Fi a MQTT task figurují v tomto schématu jako producenti událostí, task *ESP Managera* jako konzument.

3.5.4 Nahrávání a aktualizace firmwaru

Proces nahrávání nebo aktualizace firmwaru začíná přijetím MQTT zprávy do tématu `<board id>/cmd/update`. Z pohledu *ESP Managera* nehraje roli, zda se jedná o aktualizaci současného firmwaru nebo o požadavek o nahrání zcela nového firmwaru, oba případy probíhají jednotným způsobem.

Po přeposlání zprávy do tasku *ESP Managera* je její obsah zpracován a následně je zahájen mechanismus OTA pomocí funkce `esp_https_ota`, která dokáže automaticky stáhnout firmware do volného OTA oddílu a označit jej jako bootovací oddíl. URL adresa s umístěním firmwaru je dostupná ve zprávě pod klíčem `firmware_url`.

V případě, že zpráva obsahuje i položku `config_url`, je zahájena aktualizace NVS oddílu. Tato operace vyžaduje manuální zpracování (viz část 2.9.1) a její implementace se nachází v souboru `config_update.c`.

Pokud je stažení všech souborů úspěšné, je na backend odesláno prostřednictvím tématu `<board id>/info/update/ok` potvrzení o úspěšné aktualizaci. Toto potvrzení obsahuje identifikátor nového firmwaru a jeho verzi, které backend

následně uloží do databáze. Tím je z pohledu backendu aktualizace dokončena. Závěrečným krokem je automatický restart desky pomocí funkce `esp_restart`, čímž je aktualizace dokončena i na straně desky.

Pokud nastane při stahování souborů chyba, je na backend odesláno přes téma `<board id>/info/update/error` hlášení o chybě. Zpráva obsahuje identifikátor a verzi aktuálně spuštěného firmwaru, který z důvodu nezdařilé aktualizace pokračuje v běhu.

3.5.5 Přepnutí na výchozí firmware

K přepnutí na *výchozí firmware* dochází po přijetí MQTT zprávy do tématu `<board id>/cmd/update`. V tomto případě postačuje pouze označit factory oddíl jako bootovací oddíl, což provádí funkce `set_factory_as_boot_partition`. Následuje opět odeslání potvrzovací zprávy do tématu `<board id>/info/update/ok` a provedení restartu.

3.6 Výchozí firmware

Výchozí firmware představuje minimální funkční implementaci, která je kompatibilní se systémem. Jeho jediným účelem je inicializovat *ESP Managera*, aby deska byla v každém okamžiku ve spojení s backendem. Neobsahuje tedy žádnou aplikační logiku, vypisuje pouze logovací zprávy, které mohou sloužit k ladění. Zkompilovaná verze `výchozího firmwaru` pro každou podporovanou architekturu je uložen v následujícím souboru.

- `<ESP-Manager>/webapp/backend/data/default/targets/<target>/firmware.bin`

4 Práce se systémem

Tato kapitola je rozdělena do tří tématických částí. První část tvoří administrátorská dokumentace, která popisuje nasazení *webové aplikace*. Následující část je určena programátorům a zabývá se především specifiky vývoje firmwaru využívajícího *ESP Managera*. Závěrečná část slouží jako návod pro uživatele vývojových desek.

4.1 Administrátor

Tato sekce dokumentuje nasazení *webové aplikace*. Prerekvizitou pro následující kroky je počítač s operačním systémem Linux, případně se systémem Windows a prostředím WSL.

4.1.1 Instalace

Na hostující počítač je třeba nainstalovat Docker Compose a naklonovat gitový repozitář systému:

```
> git clone https://github.com/radimvalis/ESP-Manager.git
```

4.1.2 Certifikáty

Komunikace s reverzní proxy a MQTT brokerem je zabezpečena protokolem TLS, tudíž je nutné pro oba servery vygenerovat privátní klíče a digitální certifikáty. *Webová aplikace* očekává všechny klíče a certifikáty v následujících souborech:

- `<ESP-Manager>/webapp/reverse-proxy/certs/server.crt`,
- `<ESP-Manager>/webapp/reverse-proxy/certs/server.key`,
- `<ESP-Manager>/webapp/mqtt-broker/certs/broker.crt`,
- `<ESP-Manager>/webapp/mqtt-broker/certs/broker.key`.

Dále je nutné získat certifikát od autority, která oba certifikáty podepsala, a uložit jej do následujícího souboru:

- `<ESP-Manager>/webapp/ca-bundle/ca-bundle.crt`.

4.1.3 Konfigurace

Webová aplikace je parametrizovaná proměnnými prostředí, jež jsou definovány v souboru `<ESP-Manager>/webapp/.env`. Do proměnné `HOST` je třeba přiřadit IP adresu hostujícího počítače, do proměnných `MQTT_ADMIN_PASSWORD`, `DATABASE_PASSWORD`, `ACCESS_TOKEN_SECRET` a `REFRESH_TOKEN_SECRET` je z bezpečnostních důvodů nutné přiřadit dostatečně dlouhé a náhodné textové řetězce.

4.1.4 Spuštění

Webová aplikace se spouští pomocí skriptu `<ESP-Manager>/webapp/start.sh`. Skript lze spustit v jednom ze dvou dostupných režimů, před prvním použitím je ale nezbytné jej označit jako spustitelný:

```
> chmod +x start.sh
```

Vývojářský režim Vývojářský režim je díky zabudovanému development serveru ideální pro testování a vývoj nových funkcionalit. Aplikace se v tomto režimu spouští příkazem:

```
> ./start.sh --dev
```

Produkční režim V produkčním režimu běží menší počet kontejnerů, používají se pouze nezbytné balíčky a frontend je přeložen do statických souborů, což zlepšuje výkon a snižuje paměťové nároky. Tento režim se spouští příkazem:

```
> ./start.sh
```

4.2 Programátor

Tato část poskytuje návod k programování kompatibilního firmwaru a jeho správě prostřednictvím uživatelského rozhraní *webové aplikace*.

4.2.1 Programování firmwaru

Firmware kompatibilní se systémem musí být naprogramován v ESP-IDF a využívat *ESP Managera*. Návod k instalaci ESP-IDF je dostupný na oficiálních stránkách frameworku [10], *ESP Manager* je k dispozici v repozitáři projektu, v adresáři `<ESP-Manager>/esp_manager/`.

Nový ESP-IDF projekt se vytvoří následujícím příkazem:

```
> idf.py create-project <project name>
```

Příkaz vygeneruje mimo jiné soubor `CMakeLists.txt`, v němž je nutné direktivou `set` specifikovat cestu k lokální kopii *ESP Managera* (viz Soubor 3).

Soubor 3 Ukázková konfigurace souboru `CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.16)

set(EXTRA_COMPONENT_DIRS <ESP-Manager>/esp_manager/)

include($ENV{IDF_PATH}/tools/cmake/project.cmake)

project(<project name>)
```

Posledním krokem v konfiguraci ESP-IDF projektu je volba cílové architektury. V aktuální verzi systému je podporován pouze ESP32, proto je třeba provést příkaz:

```
> idf.py set-target esp32
```

Kód firmwaru musí inicializovat *ESP Managera*, což zahrnuje volání funkcí `esp_manager_init` a `esp_manager_start` (viz Program 8).

Program 8 Minimální firmware kompatibilní se systémem

```
#include "esp_manager.h"

void app_main(void)
{
    // Initialization of ESP manager

    esp_manager_client_handle_t client = esp_manager_init();

    esp_manager_start(client);

    // Your code goes here ...
}
```

Má-li být firmware konfigurovatelný, je nutné vytvořit konfigurační formulář a zároveň zajistit správné načtení konfigurace z NVS oddílu do paměti. Konfigurace je uložena vždy v NVS oddílu s názvem `config`, ve jmenném prostoru `esp_manager`. Například firmware pro komunikaci s pohybovým čidlem, jež by uživatelům umožňoval nastavit proměnné `sensitivity`, `timeout` a `alert_options` (viz konfigurační formulář v Souboru 4), by mohl vypadat jako Program 9.

Program 9 Čtení konfigurace z NVS

```
#include "esp_manager.h"
#include "nvs_flash.h"

void app_main(void)
{
    // Initialization of ESP manager

    esp_manager_client_handle_t client = esp_manager_init();

    esp_manager_start(client);

    // Open NVS partition with stored configuration

    nvs_flash_init_partition("config");

    nvs_handle_t nvs_handle;

    nvs_open_from_partition("config",
                            "esp_manager",
                            NVS_READONLY,
                            &nvs_handle);

    // Read configuration

    size_t size;

    char* sensitivity;
    nvs_get_str(nvs_handle, "sensitivity", NULL, &size);
    sensitivity = malloc(size);
    nvs_get_str(nvs_handle, "sensitivity", sensitivity, &size);

    int32_t timeout;
    nvs_get_i32(nvs_handle, "timeout", &timeout);

    char* alert_options;
    nvs_get_str(nvs_handle, "alert_options", NULL, &size);
    alert_options = malloc(size);
    nvs_get_str(nvs_handle, "alert_options", alert_options, &size);

    // Use configuration here ...
}
```

4.2.2 Psaní konfiguračního formuláře

Ke každému konfigurovatelnému firmwaru je třeba vytvořit konfigurační formulář, což je soubor ve formátu JSON, který obsahuje neprázdné pole objektů, z nichž každý reprezentuje jeden uživatelsky nastavitelný parametr.

V Programu 9 se z NVS oddílu načítají tři hodnoty, proto by i konfigurační formulář měl obsahovat tři objekty (viz Soubor 4). Každý z objektů obsahuje povinné položky `key`, `isRequired`, `label`, `type` a `encoding`, a z dokumentačních účelů také volitelnou položku `description`. Parametr `sensitivity` obsahuje navíc položku `options`, která definuje seznam hodnot, ze kterých bude uživatel firmwaru mít možnost vybírat. Číselným parametrům jsou k dipozici položky `minimum` a `maximum`, což využívá parametr `timeout`. Parametr `alert_options` vhodně kombinuje položky `type` a `format`, čímž bude uživatel donucen za parametr dosadit textový řetězec formátovaný jako JSON.

Soubor 4 Konfigurační formulář

```
[
    {
        "key": "sensitivity",
        "isRequired": true,
        "label": "Sensitivity Level",
        "description": "Sets sensor's sensitivity.",
        "type": "data",
        "encoding": "string",
        "options": [ "low", "medium", "high" ]
    },
    {
        "key": "timeout",
        "isRequired": true,
        "label": "Timeout",
        "description": "Sets sensor's timeout in seconds.",
        "type": "data",
        "encoding": "i32",
        "minimum": 1,
        "maximum": 60
    },
    {
        "key": "alert_options",
        "isRequired": true,
        "label": "Alert Options",
        "description": "Alert options in JSON.",
        "type": "file",
        "encoding": "string",
        "format": "application/json"
    }
]
```

4.2.3 Nahrání firmwaru

Firmware je nutné před nahráním do systému zkompilovat:

```
> idf.py build
```

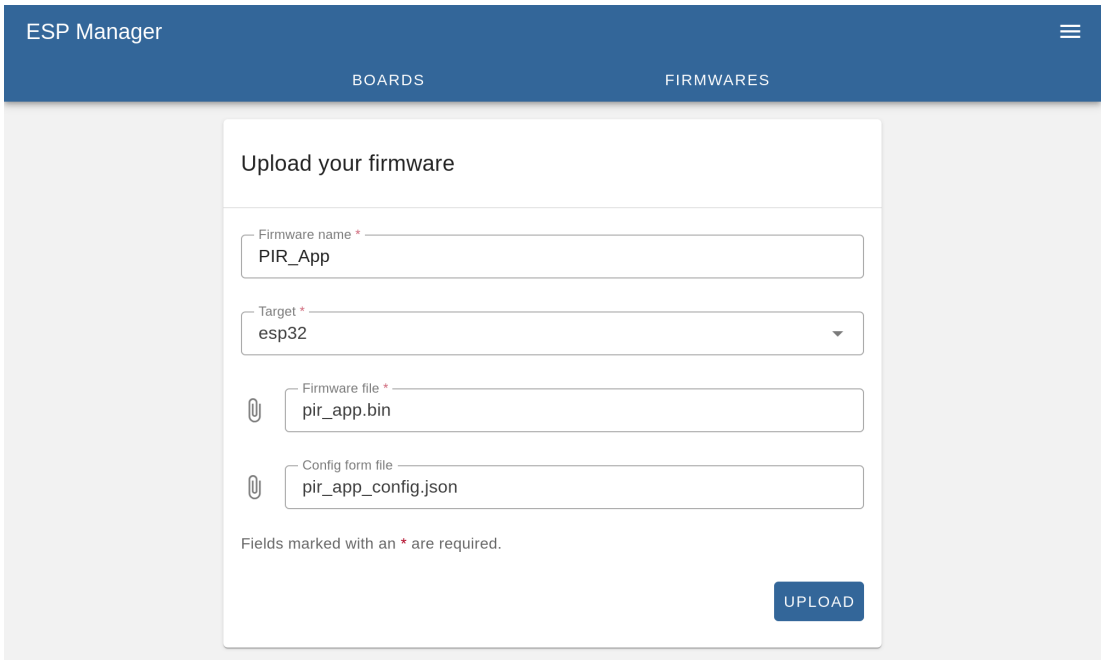
Firmware se nahrává prostřednictvím uživatelského prostředí *webové aplikace*. Nahrávací formulář (viz Obrázek 4.1) je dostupný z detailu desky po stisknutí tlačítka `NEW FIRMWARE`. Ve formuláři je třeba zadat název firmwaru, zvolit cílovou architekturu a připojit firmware (soubor `<project root>/build/<project name>.bin`), případně i konfigurační formulář. Nahrání firmwaru se potvrzuje tlačítkem `UPLOAD`.

4.2.4 Aktualizace firmwaru

Nová verze se nahrává v detailu firmwaru prostřednictvím pole `Updated firmware file`. Po stisknutí tlačítka `UPDATE` se v horní části obrazovky zobrazí informační baner, který uživateli zobrazí výsledek aktualizace. S firmwarem není možné aktualizovat jeho konfigurační formulář, proto je nutné se při úpravách zdrojového kódu vyvarovat takovým změnám, které by vyžadovaly úpravu konfiguračního formuláře.

4.2.5 Smazání firmwaru

Firmware lze smazat v detailu firmwaru prostřednictvím tlačítka `DELETE`. Po provedení této akce není již možné daný firmware vyhledat (viz druhou fázi nahrávání firmwaru do desky 4.3.2), a tedy ani nahrát do žádné desky. Tato operace však neovlivňuje desky, na kterých byl daný firmware nahrán ještě před svým smazáním. Na těchto deskách funguje firmware i nadále a je možné jej aktualizovat na poslední verzi, která byla dostupná před smazáním.



Obrázek 4.1 Nahrání firmwaru a konfiguračního formuláře do systému

4.3 Uživatel vývojové desky

Uživatel vývojové desky pracuje se systémem výhradně prostřednictvím uživatelského rozhraní *webové aplikace*. Tato sekce proto dokumentuje funkce uživatelského rozhraní, jež jsou pro tuto skupinu užitečné.

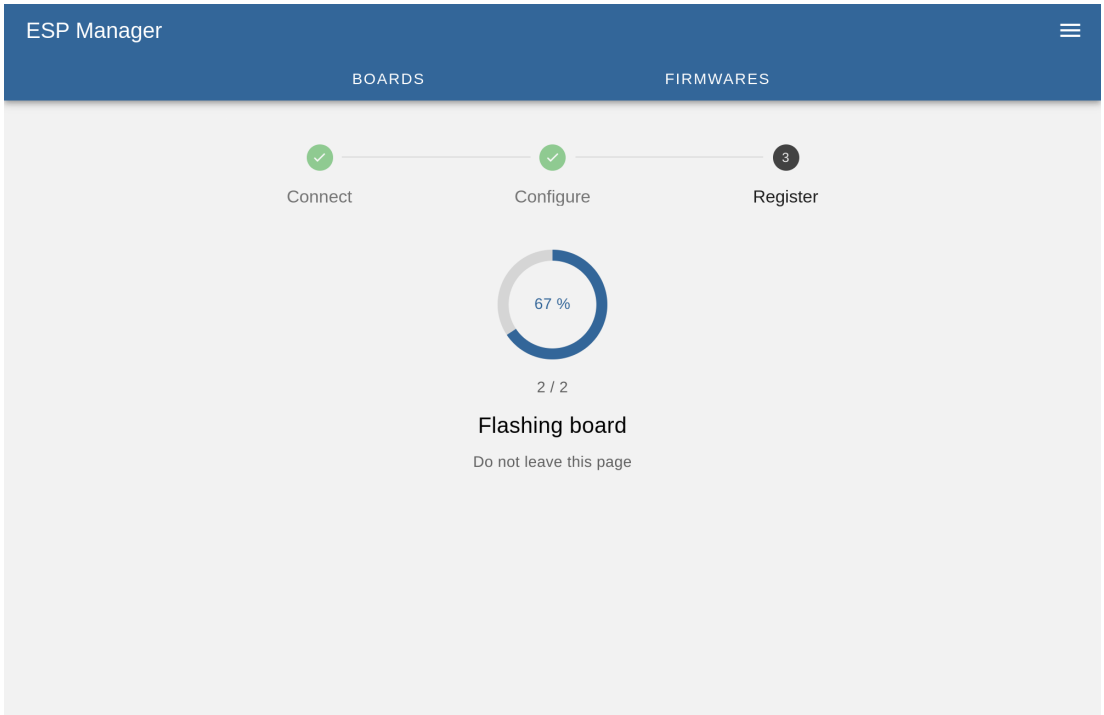
4.3.1 Registrace vývojové desky

Registraci vývojové desky lze provést výhradně v prohlížečích Chrome, Edge a Opera, neboť ty jako jediné podporují Web Serial API, které je nezbytné pro sériovou komunikaci s vývojovou deskou. Registrace se zahajuje v seznamu desek tlačítkem `NEW BOARD` a sestává ze tří fází.

Connect Nejprve je nutné připojit vývojovou desku k počítači prostřednictvím datového USB kabelu. Následně je třeba zvolit sériový port (např. `ttyUSB0`), ke kterému je deska připojena. Výběr se provádí pomocí tlačítka `CONNECT`, které zobrazuje seznam všech dostupných portů. V případě nejasností ohledně názvu sériového portu lze nahlédnout do návodu pro Windows [11], případně pro Linux a macOS [12]. Po potvrzení portu se začne navazovat spojení mezi počítačem a deskou. Jakmile je spojení navázáno, registrace pokračuje další fází. Není-li spojení navázáno během několika sekund, obvykle pomůže desku odpojit, znovu připojit a výběr portu zopakovat.

Configure Od tohoto okamžiku až do skončení registrace musí vývojová deska zůstat připojena k počítači. Uživatel zároveň nesmí opustit aktuální stránku ani přepnout na jinou záložku, neboť by taková akce přerušila spojení s deskou. V této fázi je uživatel vyzván k zadání názvu vývojové desky a přístupových údajů k Wi-Fi síti, ke které se má deska po registraci připojovat. Přístupové údaje nelze po dokončení registrace měnit, je proto důležité zvážit, v jakém prostředí bude deska provozována. Po stisknutí tlačítka `REGISTER` následuje závěrečná fáze.

Register V tomto okamžiku je do desky nahráván *výchozí firmware* společně s parametry, které byly zadány v předchozí části. Nahrávání obvykle trvá několik desítek sekund a jeho průběh je možné sledovat prostřednictvím grafického ukazatele 4.2. Po dokončení nahrávání je uživatel přesměrován na detail právě zaregistrované desky. Tím je registrace dokončena, a je tak bezpečné desku odpojit a přenést na místo určené k provozu.



Obrázek 4.2 Registrace vývojové desky

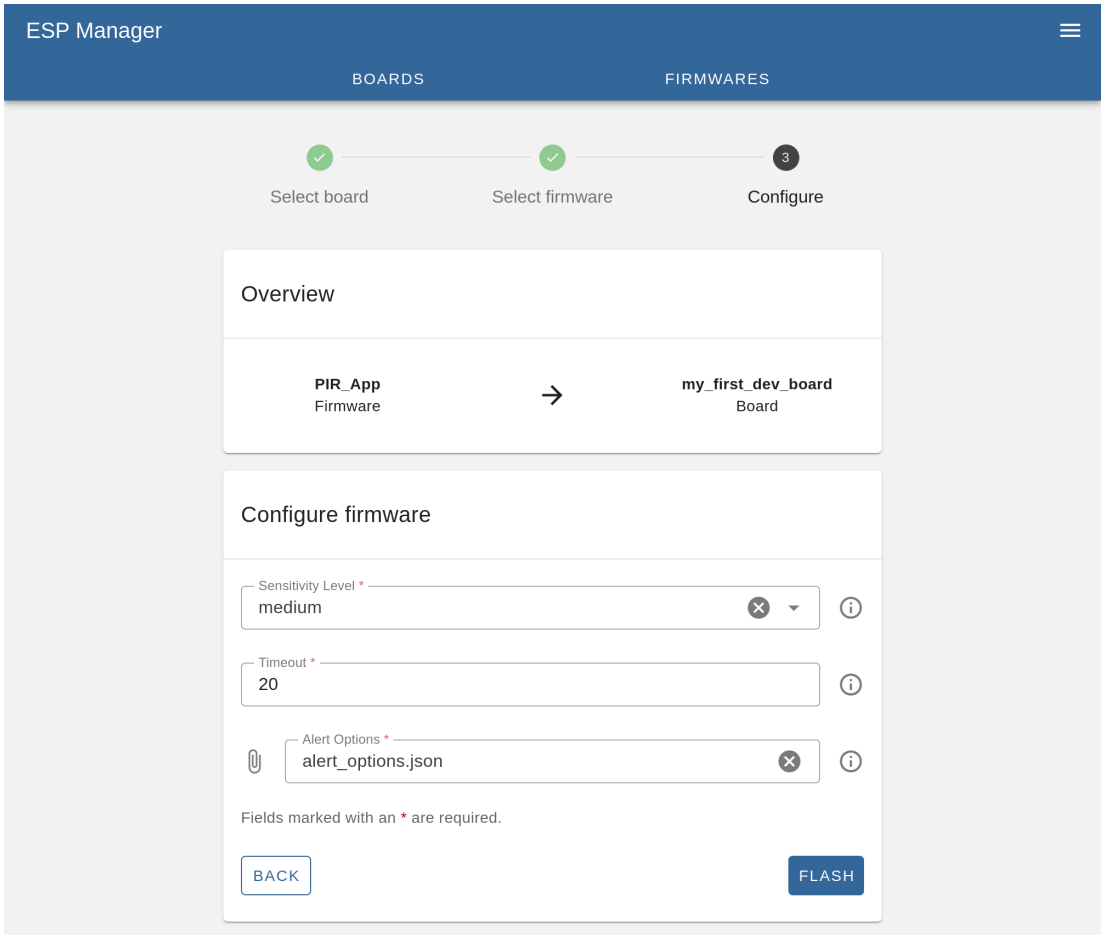
4.3.2 Nahrání firmwaru do vývojové desky

Nahrávání firmwaru probíhá bezdrátově, pomocí mechanismu OTA, tudíž musí být cílová deska po dobu trvání operace online. Operace sestává ze tří fází, přičemž první fáze se zahajuje tlačítkem `FLASH BOARD`.

Select board Cílem této fáze je výběr vývojové desky ze zobrazeného seznamu. V seznamu jsou zahrnuty pouze desky, do nichž je v daný okamžik možné firmware nahrát, tedy takové, které jsou online a současně se neaktualizují. Výběr a přechod do další fáze se potvrzuje tlačítkem `SELECT`.

Select firmware Do vyhledávacího pole je nutné zadat identifikátor požadovaného firmwaru a výběr následně potvrdit tlačítkem `SELECT`. Pokud je firmware s daným identifikátorem nalezen, proces pokračuje do závěrečné fáze. V opačném případě je uživatel informován, že požadovaný firmware neexistuje.

Configure V závěrečné fázi je nutné firmware nakonfigurovat prostřednictvím konfiguračního formuláře 4.3 a pomocí tlačítka `FLASH` zahájit vlastní nahrávání (mechanismus OTA). Pokud firmware konfiguraci nevyžaduje, formulář se nezobrazuje, a nahrávání tak lze zahájit ihned. Po zahájení nahrávání je uživatel přesměrován na detail příslušné desky. Detail desky je během nahrávání firmwaru nahrazen zprávou o probíhající aktualizaci, a práce s deskou tak není až do skončení operace možná. V případě úspěšného nahrání je nový firmware na desce automaticky spuštěn, v opačném případě pokračuje v běhu původní firmware.



Obrázek 4.3 Závěrečná fáze nahrávání firmwaru do desky

4.3.3 Aktualizace firmwaru ve vývojové desce

Všechny desky s neaktuálním firmwarem jsou v seznamu desek označeny oranžovou značkou s nápisem `update available`. Desku s neaktuálním firmwarem lze za předpokladu, že je v daném okamžiku online, aktualizovat v detailu desky pomocí tlačítka `UPDATE`. Proces aktualizace pak probíhá stejným způsobem jako nahrávání firmwaru.

4.3.4 Smazání desky

Desku lze smazat pomocí tlačítka `DELETE`, které je dostupné v detailu desky. Tato operace pouze odstraňuje příslušný záznam z interní databáze *webové aplikace*, nijak tedy nezasahuje do vnitřního stavu dané desky. Na desce tak nadále běží poslední nahraný firmware, avšak vzhledem k jejímu vyřazení ze systému není již možná její komunikace s *webovou aplikací*.

5 Závěr

Tato práce se zabývala návrhem a implementací systému pro správu ESP vývojových desek. V sekci 2.4 bylo rozhodnuto systém rozdělit do tří částí - *webové aplikace*, *ESP Managera* a *výchozího firmwaru*, což bylo úspěšně realizováno.

Webová aplikace byla sestavena z pěti kontejnerů, z nichž je pro nás nejdůležitější frontend, prostřednictvím kterého uživatelé se systémem pracují, a backend, na kterém je koncentrována logika celého systému. Díky knihovně esptool-js se podařilo implementovat nahrávání *výchozího firmwaru* do desky přímo prostřednictvím webového rozhraní. Frontend rovněž umožňuje převod konfiguračního formuláře ve formátu JSON na HTML formulář, čímž nabízí pohodlný způsob konfigurace firmwaru. Díky integrovanému MQTT klientovi a OTA mechanismu v *ESP Managerovi*, kterého využívá každý firmware zveřejněný v systému, je možné prostřednictvím webového rozhraní i sledovat stav připojení desky a provádět její vzdálené aktualizace.

Backend prostřednictvím svého API zprostředkovává všechny soubory potřebné pro zprovoznění vývojové desky, tj. firmwary, bootloadery, NVS image apod. Díky tomu uživatel vývojových desek nemusí instalovat ESP-IDF, a zvládne tak vše vyřídit přes webové rozhraní. Během domácího testování bylo ověřeno, že backend dokáže bez problémů obsloužit desítky paralelních požadavků. Při vyšším počtu souběžných požadavků, které vyžadují práci se souborovým systémem, má však backend výkonnostní problémy, což je způsobeno tím, že Node.js běží pouze v jednom vlákně. S ohledem na zamýšlené domácí / komunitní použití však taková situace nebude nastávat příliš často, a tudíž nepředstavuje závažnější problém. Obsluhu požadavků by mohl pomoci zychlit Node.js modulu Cluster, který umožňuje spustit aplikaci v několika separátních procesech sdílejících stejný síťový port. Druhou variantou by byla replikace celého backend kontejneru, např. pomocí Docker Compose direktivy `deploy` a současného nastavení load balancingu v nginx.

5.1 Rozšíření

Tato sekce poskytuje seznam možných rozšíření systému společně s návrhem jejich implementace.

Podpora dalších čipů Pro rozšíření systému o podporu dalších čipů, jako je např. ESP32-S3, je třeba provést následující kroky.

1. Vytvoří se nový adresář `<ESP-Manager>/webapp/backend/data/default/targets/esp32-s3/`.

2. Nastaví se ESP32-S3 jako cílová architektura.

```
> idf.py set-target esp32-s3
```

3. V adresáři `default_app/` se zkompiluje *výchozí firmware*.

```
> idf.py build
```

4. Výsledný soubor se přejmenuje na `firmware.bin` a přesune se do adresáře vytvořeného v kroku 1.
5. V adresáři `esp32-s3/` se vytvoří adresář `bootloaders/`.
6. Pomocí nástroje `menuconfig` se nastaví velikost flash paměti na `<MB>` megabajtů.

```
> idf.py menuconfig
```

7. Zkompiluje se bootloader.

```
> idf.py bootloader
```

8. Výsledný soubor `build/bootloader/bootloader.bin` se přejmenuje na `bootloader_<MB>MB.bin` a přesune se do adresáře `bootloaders/`.
9. Kroky 6-7 se provádějí pro všechny požadované velikosti flash paměti.

Přepínání firmwaru Při přepnutí na *výchozí firmware* zůstávají původní firmware a jeho konfigurace nahrány v OTA oddílu a příslušném NVS oddílu. Tuto skutečnost by bylo možné využít k implementaci funkcionality „pozastavení firmwaru“, která by uživatelům umožnila dočasně pozastavit firmware bez nutnosti jej při opětovném spuštění znovu konfigurovat a nahrávat prostřednictvím OTA mechanismu.

Konfigurace ESP Managera na úrovni kódu Uživatelé vývojových desek mají možnost částečné konfigurace *ESP Managera* během registrace desky, neboť zadávají SSID a heslo, které komponenta využívá. Programátoři však podobnou možnost nemají, což je může limitovat při integraci *ESP Managera* do jejich firmwaru.

Možným řešením by bylo přidání konfigurační struktury, kterou by přijímala funkce `esp_manager_init`, čímž by programátoři získali větší kontrolu nad chováním komponenty. Toto řešení by bylo užitečné například v případech, kdy je potřeba zajistit připojení k síti jiným způsobem, například prostřednictvím Ethernetu. Programátor by toto připojení provedl sám ve svém kódu a *ESP Manager* by se po svém spuštění začal rovnou připojovat k MQTT brokeru.

Popis firmwaru Uživatel vývojové desky nemá při nahrávání firmwaru do desky k dispozici žádné informace o jeho funkcionalitě či účelu. Řešením tohoto nedostatku by mohla být možnost připojit při registraci firmwaru jeho krátký textový popis. Při omezení na krátké popisy (dlouhé maximálně několik vět) by je bylo možné ukládat přímo do databáze. Bez omezení na délku popisu by bylo vhodné je nahrávat v textových souborech, které by se ukládaly v adresáři daného firmwaru.

Podobným způsobem by bylo možné dokumentovat i jednotlivé aktualizace firmwaru. Díky tomu by se uživatelé mohli lépe rozhodnout, zda se aktualizace vyplatí.

Literatura

1. Mozilla Corporation. *Web Serial API* [online]. [cit. 2025-06-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Serial_API.
2. Espressif Systems. *Espressif IoT Development Framework* [online]. [cit. 2024-12-14]. Dostupné z: https://github.com/espressif/esp-idf.
3. Espressif Systems. *ESP-IDF - Chip Series Comparison* [online]. [cit. 2024-12-07]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/hw-reference/chip-series-comparison.html.
4. Espressif Systems. *Partition Generator Utility* [online]. [cit. 2025-02-22]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/nvs_flash.html.
5. Espressif Systems. *Non-Volatile Storage Library - CSV File Format* [online]. [cit. 2025-02-22]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/nvs_partition_gen.html#csv-file-format.
6. JSON Schema. *JSON Schema - Docs* [online]. [cit. 2025-02-23]. Dostupné z: https://json-schema.org/docs.
7. Espressif Systems. *Over The Air Updates (OTA)* [online]. [cit. 2025-05-13]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html.
8. Espressif Systems. *Partition Tables* [online]. [cit. 2025-05-14]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/partition-tables.html.
9. Eclipse Foundation. *Dynamic Security Plugin* [online]. [cit. 2025-07-08]. Dostupné z: https://mosquitto.org/documentation/dynamic-security.
10. Espressif Systems. *ESP-IDF - Get Started* [online]. [cit. 2025-02-22]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html.
11. Espressif Systems. *Establish Serial Connection with ESP32 - Check Port on Windows* [online]. [cit. 2025-04-06]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/establish-serial-connection.html#check-port-on-windows.
12. Espressif Systems. *Establish Serial Connection with ESP32 - Check Port on Linux and macOS* [online]. [cit. 2025-04-06]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/establish-serial-connection.html#check-port-on-linux-and-macos.

Seznam obrázků

2.1 Nahrání firmwaru 13
2.2 Seznam nahraných firmwarů 13
2.3 Detail firmwaru 14
2.4 Registrace vývojové desky 15
2.5 Seznam registrovaných vývojových desek 15
2.6 Detail vývojové desky 16
2.7 Nahrávání firmwaru do desky 16

3.1 Architektura webové aplikace 26
3.2 Architektura backendu 26
3.3 Databázové schéma 27
3.4 Seznam nahraných firmwarů 31
3.5 Detail firmwaru 31
3.6 Seznam vývojových desek 32
3.7 Detail vývojové desky 32

4.1 Nahrání firmwaru a konfiguračního formuláře do systému 44
4.2 Registrace vývojové desky 46
4.3 Závěrečná fáze nahrávání firmwaru do desky 47