Česká zemědělská univerzita v Praze

Technická fakulta

Katedra elektrotechniky a automatizace

Diplomová práce

Návrh a realizace kontrolního systému na WiFi síti

Martin Novák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Technická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Martin Novák

Informační a řídicí technika v agropotravinářském komplexu

Název práce

Návrh a realizace kontrolního systému na WiFi síti

Název anglicky

Design and implementation of a control system on a WiFi network

---

Cíle práce

Cílem diplomové práce je provedení analýzy zavedení kontrolního systému na WiFi síti. Výstupem práce bude návrh a realizace struktury zařízení, které bude obsahovat prvky kontroly komunikace jednotlivých připojených uzlů a na základě nastavených pravidel a přijatých dat bude provádět vyhodnocení.

Metodika

Prostudování hardwarových a softwarových možností řešení. Navržení několika variant provedení úlohy. Výběr nejvhodnější varianty s kritickým hodnocením návrhu. Specifikace funkcí systému podle cílů práce.

Doporučený rozsah práce

60stran, bez příloh

Klíčová slova

ESP, WiFi, komunikace, síť, bezpečnost

Doporučené zdroje informací

BELL, Charles A. *Beginning sensor networks with Arduino and Raspberry Pi.* [New York, New York]: Apress, 2013. ISBN 1430258241.

DENNIS, Andrew K. *Raspberry Pi home automation with Arduino : automate your home with a set of exciting projects for the Raspberry Pi!.* Birmingham: Packt Publishing, 2013. ISBN 978-1-78439-920-7.

MORRISS, S. Brian. *Automated manufacturing systems : actuators, controls, sensors, and robotics.* New York: Glencoe, 1995. ISBN 0028023315.

ORR, Ruby Ashby; PITTOCK, Kenny; NEJEDLÁ, Kateřina. *Sto a jedna věc co dělat, když wifi nefunguje.* Praha: Ikar, 2020. ISBN 978-80-249-4280-3.

Předběžný termín obhajoby

2024/2025 LS – TF

Vedoucí práce

doc. Ing. Miloslav Linda, Ph.D.

Garantující pracoviště

Katedra elektrotechniky a automatizace

Elektronicky schváleno dne 29. 01. 2024

doc. Ing. Monika Hromasová, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 01. 03. 2024

doc. Ing. Jiří Mašek, Ph.D.

Děkan

V Praze dne 26. 03. 2025

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a realizace kontrolního systému na WiFi síti" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

Prohlašuji, že jsem nástroje AI využil v souladu s vnitřními předpisy univerzity a principy akademické integrity a etiky. Na využití těchto nástrojů v práci vhodným způsobem odkazuji.

V Praze dne 31.3.2025 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) vedoucímu diplomové práce doc. Ing. Miloslavovi Lindovi, Ph.D. za vedení, užitečné rady, a především za trpělivost. Dále bych chtěl poděkovat své rodině za podporu při psaní této práce.

Návrh a realizace kontrolního systému na WiFi síti

Abstrakt

Tato diplomová práce se zabývá navržením a následnou realizací kontrolního systému, jenž je provozován na Wi-Fi síti. Ten je tvořen hlavním uzlem interagujícím s uživatelem a uzly, které k sobě mají připojeny sensory, akční členy nebo obojí. Hlavní uzel je vytvořen jako počítačový program s vrstvenou architekturou, čím je umožněna modularita a nahrazení části, která nevyhovuje potřebám řešené úlohy. Dále je možné bez ovlivnění zbylé části kódu nahradit komunikační třídu za jinou. Uzly jsou realizovány pomocí vývojových desek NodeMCU pro ESP2688 12E. Kód využívá Arduino knihoven a obsahuje abstrakci, aby bylo možné řešení migrovat na jinou platformu. Ke komunikaci je využíván protokol HTTP a data ve formátu JSON. Při návrhu byl kladen důraz na modularitu a možnost provozovat více systémů současně.

Klíčová slova: ESP, Wi-Fi, komunikace, síť, bezpečnost, C#, JSON, vrstvená architektura, OSI, SoC

Design and implementation of a control system on a WiFi network

Abstract

This diploma thesis deals with the design and subsequent implementation of a control system that operates on a Wi-Fi network. It consists of a main node interacting with the user and nodes that have sensors, actuators or both connected to them. The main node is created as a computer program with a layered architecture, which allows modularity and the replacement of a part that does not meet the needs of the task being solved. It is also possible to replace the communication class with another without affecting the rest of the code. The nodes are implemented using NodeMCU development boards for ESP2688 12E. The code uses Arduino libraries and contains abstraction to allow the solution to migrate to another platform. Communication is based on the HTTP protocol and JSON data format. The design emphasized modularity and the ability to operate multiple systems simultaneously.

Keywords: ESP, Wi-Fi, Communication, network, security, C#, JSON, layered architecture, OSI, SoC

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ADC	Analog-to-Digital Converter
AP	Access Point

API	Application Programming Interface
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
BSSID	Basic Service Set IDentifier
CCK	Complementary Code Keying
CI/CD	Continuous Integration / Continuous Delivery
CPU	Central Processing Unit
CRC	Cyclical Redundancy Checking
CSV	Comma-Separated Values
CVE	Common Vulnerabilities and Exposures
DHCP	Dynamic Host Configuration Protocol
DO	Domain Object
DOM	Document Object Model
DRAM	Dynamic Random Access Memory
DSSS	Direct-Sequence Spread Spectrum
DTO	Data transfer Object
FHSS	Frequency-Hopping Spread Spectrum
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I2C	Inter-lntegrated Circuit
IDE	Integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
IPV4	Internet Protocol version 4
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KRACKs	Key Reinstallation AttaCKs
LAN	Local Area Network
LTS	Long Term Support
MAC	Media Access Control
MCU	MicroController Unit
MIMO	Multiple-Input Multiple-Output

MTU	Maximum transmission unit
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-ViewModel
NAT	Network Address Translation
ODF	OpenDocument Format
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
OLED	organic light-emitting diode
OSI	Open System Interconnection
QAM	Quadrature Amplitude Modulation
ROM	Read Only Memory
RSA	Rivest–Shamir–Adleman
SAE	Simultaneous Authentication of Equals
SoC	System-On-Chip
SPI	Serial Peripheral Interface
SSID	Service Set IDentifier
SSL	Secure Sockets Layer
STA	STAtion
STS	Short Term Support
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UART	universal asynchronous receiver-transmitter
UDP	User Datagram Protocol
URL	uniform resource locator
UWP	Universal Windows Platform
VLSI	Very large-Scale Integration
WAN	Wide Area Network
WEP	Wired Equivalent Privacy
WinForm	Windows Forms

WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WPF	Windows Presentation Foundation
XAML	eXtensible Application Markup Language
XML	eXtensible Markup Language

1 Úvod

V době stále rozrůstajícího se počtu chytrých zařízení, která jsou připojena k internetu nebo počítači, roste také zájem uživatelů o automatizaci různých procesů. Může se jednat například o automatizaci v rámci domácnosti nebo nějakého výrobního procesu. Komplikací je, že mnoho těchto zařízení se nachází v ekosystémech, které nejsou vzájemně kompatibilní. Tato práce se snaží vytvořit modulární řídící systém, který jednak umožňuje připojit zařízení vytvořena pomocí přiložené knihovny, ale také programátorům umožňuje vytvořit modul pro přidání zařízení z již existujícího ekosystému.

V přehledu řešené problematiky budou popsány jednotlivé vrstvy OSI modelu, jenž slouží k popisu síťové komunikace. Dále budou vysvětleny nejdůležitější síťové protokoly a formáty používané k posílání dat. Poté budou popsány principy funkce komunikace v síti Wi-Fi. Následuje vysvětlení dělení obvodů s vysokým stupněm integrace a představení čipu ESP8266. Nakonec budou vysvětleny programátorské techniky související s touto prací.

Vlastní řešení se skládá ze dvou hlavních oblastí. První je hlavní uzel, se kterým uživatel interaguje a jenž řídí komunikaci a vyhodnocování zadané logiky. Při návrhu této části bude kladen důraz především na modulárnost, aby bylo možné jednotlivé moduly nahradit bez ovlivnění zbytku systému. Druhou oblastí jsou jednotlivé uzly sloužící jako vstupy a výstupy systému. Tato oblast je tvořena logikou společnou pro všechny uzly a vytvořením vzorových implementací.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem diplomové práce je provedení analýzy zavedení kontrolního systému na Wi-Fi síti. Výstupem práce bude návrh a realizace struktury zařízení, které bude obsahovat prvky kontroly komunikace jednotlivých připojených uzlů a na základě nastavených pravidel a přijatých dat bude provádět vyhodnocení.

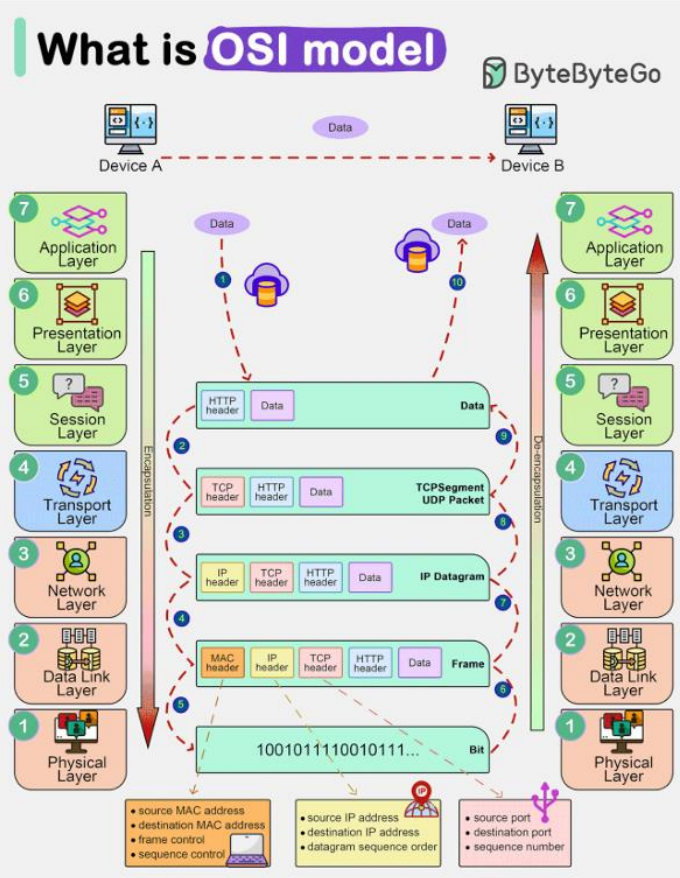
2.2 Metodika

Prostudování hardwarových a softwarových možností řešení. Navržení několika variant provedení úlohy. Výběr nejvhodnější varianty s kritickým hodnocením návrhu. Specifikace funkcí systému podle cílů práce.

3 Přehled řešené problematiky

3.1 OSI model

Model OSI (Open System Interconnection) je teoretickým modelem vyvinutým v roce 1984 mezinárodní organizací pro standardizaci (ISO), definující protokoly pro komunikaci různých zařízení na síti. Jedná se o sedmivrstvou architekturu (viz kapitola 3.6.2), která je vyobrazena na Obr. 1 během posílání HTTP (viz kapitola 3.2.4) dotazu. Výhodou je nezávislost jednotlivých vrstev na konkrétní implementaci ostatních, což usnadňuje případný vývoj nových technologií. Dále se snáze hledá příčina problémů s připojením. Ovšem v praxi se spíše využívá model TCP/IP (Transmission Control Protocol/Internet Protocol) slučující první a druhou vrstvu do síťového rozhraní a pátou až sedmou do aplikační vrstvy. Oproti OSI je postaven na reálných komunikačních protokolech používaných v síťových prvcích. [1–4]

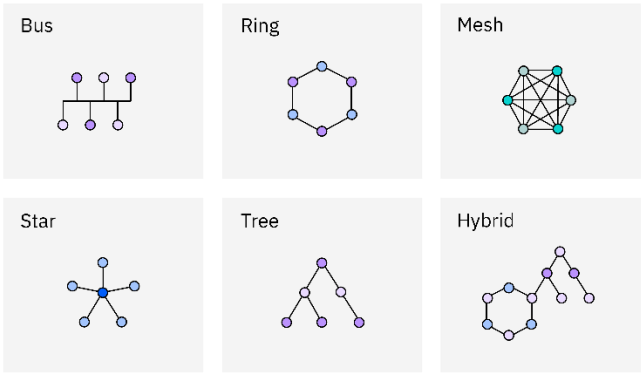


Obr. 1 OSI model[5]

3.1.1 Fyzická vrstva

Na této vrstvě dochází k fyzickému přenosu dat mezi síťovými prvky. Mezi ně se počítají routery, repeatery, switche a huby. Patří sem také metalické a optické kabely nebo radiové vlny, přes která jsou data přenášena.[3]

Tato vrstva je zodpovědná za kódování a dekódování přenášených dat na nosný signál a synchronizaci mezi vysílající a přijímající stranou. Zvolené prvky určují maximální přenosovou rychlost a zda bude komunikace simplexní, polo duplexní nebo plně duplexní. Zvolená topologie sítě (Obr. 2) má vliv na spolehlivost, bezpečnost a škálovatelnost. Je znázorňována jako graf, jehož uzly jsou jednotlivá zařízení. Fyzická topologie je dána strukturou propojení zařízení mezi sebou. Ta se nemusí shodovat s logickou topologii, která je dána datovými toky. Ty mohou být všesměrové, nebo jeden s co nejmenším počtem prošlých uzlů potřebných do cílové destinace.[3, 6]



Obr. 2 Typy topologií[6]

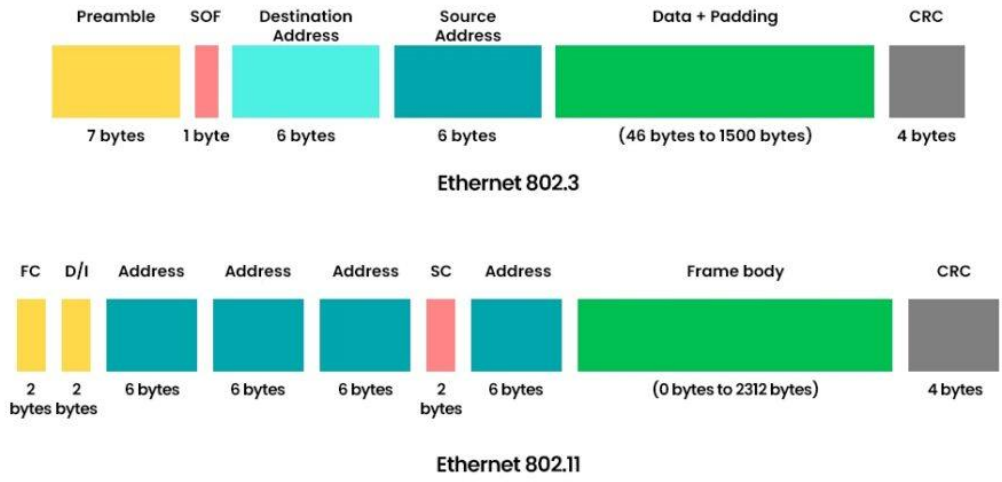
3.1.2 Linková vrstva

Tato vrstva je zodpovědná za to, aby data dorazila do správného koncového zařízení. Kromě toho kontrolují, že data dorazila bez chyb. Toho je docíleno tím, že jsou data zabalena do rámce začínající adresou koncového zařízení a končící výsledkem CRC (Cyclical Redundancy Checking) algoritmu. K adresaci zařízení se využívá MAC (Media Access Control) adresa. Dalším účelem této vrstvy je řízení datového toku. To zahrnuje určování velikosti jednotlivých rámců a určení zařízení, které momentálně řídí komunikaci.[1, 3, 7]

Algoritmus CRC slouží k detekci chyb během datového přenosu. Při odesílání je z dat vytvořen kontrolní součet o fixní velikosti. Po přijetí dat je postup zopakován a výsledek je

porovnán s přijatou hodnotou. Pokud jsou shodné, byl přenos úspěšný. K výpočtu je využíváno dělení binárních polynomů. Mezi hlavní výhody této metody patří snadná implementace a rychlost výpočtu. Dále dokáže detekovat jak náhodné chyby, tak shluky chyb. Tato metoda je oblíbená pro svou robustnost a vysokou přesnost. Nevýhodou je, že se jedná pouze o detekční mechanismus, ale ne o samoopravný kód. Množství chyb, které je možné detekovat, je určeno zvoleným charakteristickým polynomem.[8]

Podoba rámce a velikost jeho datové části je dána použitou fyzickou vrstvou (Obr. 3). Pro metalické kabely se datová část pohybuje od čtyřiceti šesti do patnácti set bytů, zatím co pro Wi-Fi se rozmezí pohybuje od nuly do dvou tisíc tří set dvanácti bytů. Rozdělením paketů ze síťové vrstvy (viz kap. 3.1.3) na menší části se snižuje pravděpodobnost kolize na přenosovém mediu.[9]



Obr. 3 Rámec 802.3 vs 802.11[9]

3.1.3 Síťová vrstva

Úkolem této vrstvy je dostat data z jednoho zařízení do jiného, aniž by se tato zařízení musela nacházet ve stejné síti. Kromě toho také hledá nejkratší cestu, kterou paket musí urazit, aby se dostal do cílové destinace. K adresaci na této vrstvě se nejčastěji využívá IPv4 (Internet Protocol version 4), ale existují i jiné alternativy. Pokud velikost paketu překročí MTU (Maximum Transmitted Unit) specifikovanou linkovou vrstvou, musí být rozdělen na menší části nazývané fragmenty. Na této vrstvě pracují routery a switche. [1, 3, 4, 10]

3.1.4 Transportní vrstva

Tato vrstva na straně odesilatele data vyšší vrstvy rozloží na části nazývané segmenty a na straně příjemce opět složí do původní podoby. Součástí tohoto procesu je kontrola, že všechna data dorazila v pořádku a případné opakování komunikace. Použitý protokol a jeho implementace určují, zda se při chybě bude opakovat celý přenos, pouze jeho část nebo bude chyba tolerována. K adrese síťové vrstvy přidává port, který operačnímu systému říká, které aplikaci má přijatá data předat [11]. Tímto je zajištěno, že stejné spojení může být používáno více aplikacemi současně. Transportní vrstva také řídí rychlost přenosu, aby v případě rozdílných rychlostí připojení na straně příjemce a odesilatele, nebyla jedna strana přehlcena. Jsou rozlišovány dva způsoby komunikace. První je bez navázání spojení (Connection-less Service), kdy jsou data rovnou odeslána příjemci. Tento způsob může vést k chybám, neboť zde není mechanismus, jak odesilateli potvrdit doručení. Druhý způsob je před zahájením komunikace navázat spojení mezi komunikujícími uzly (Connection-Oriented Service). Při tomto procesu je vytvořena definovaná trasa, po které budou pakety posílány. Jelikož spolu uzly udržují spojení, je možné dosáhnout vyšší spolehlivost, avšak může vést k větší vytíženosti sítě. [1, 3, 4, 12, 13]

3.1.5 Relační, prezentační a aplikační vrstvy

Z hlediska komunikace se na poslední tři vrstvy OSI modelu dá nahlížet jako na jednu, tak jak to dělá TCP/IP. Při vzniku OSI modelu se předpokládalo, že pátá a šestá vrstva bude využívána všemi aplikacemi. V praxi se tak nestalo a každá aplikace, která je využívá má vlastní implementaci dle svých potřeb. [14]

Úkolem relační vrstvy je navazování, spravování a ukončování relací mezi zařízeními. Během komunikace jsou zařízení synchronizována a vytváří si záchytné body, takže pokud dojde k přerušení spojení, nemusí opakovat celou komunikaci, ale pouze část od posledního záchytného bodu. Tato vrstva má také na starosti autorizaci a zabezpečení.[1, 3, 4]

Úkolem prezentační (někdy nazývané překladová) vrstvy je příprava dat aplikační vrstvy k odeslání na straně odesilatele a následná uvedení do čitelného stavu na straně příjemce. Toto zahrnuje šifrování, kompresy a přizpůsobení datového formátu.[1, 3, 4]

Tato vrstva je nejblíže uživateli a umožňuje aplikacím volat API endpointy. Samotná aplikace není součástí vrstvy, ale poskytuje protokoly umožňující aplikacím komunikovat

s ostatními zařízeními na síti. Tím je uživateli přenášet soubory, zprávy, ověřovat zařízení, vzdáleně ovládat jiná zařízení a získávat data z databází. [1, 3, 4]

3.2 Protokoly

Protokol je sada pravidel, definující strukturu přenášených dat a průběh komunikace mezi elektronickými zařízeními. Pokud odesílající i přijímající strana používají stejný protokol, je možné zajistit efektivní a spolehlivou komunikaci, protože obě strany interpretují data stejným způsobem a vědí, jak se chovat v případě chybového stavu. Různé způsoby propojení zařízení mají rozdílné protokoly. Často je protokol používaný aplikací zabalen do jednoho či více protokolů sloužícího k přenosu. Například u síťové komunikace je protokol aplikační vrstvy v datové části protokolu transportní vrstvy, který je obalen protokoly síťové a linkové vrstvy. [15]

3.2.1 IP

Protokol IP (Internet Protocol) fungující na třetí vrstvě (viz kap. 3.1.3) OSI modelu, slouží k směrování packetů napříč sítí. Pro tento účel se používá IP adresa, která je pro každé zařízení připojené do dané síti unikátní. V současné době se jako adresa stále využívá IPv4 s dvě na třicátou druhou možných adres, tedy přibližně čtyři miliardy, které jsou zapisovány jako čtveřice čísel v rozsahu 0-255 oddělené tečkou. Od roku 1998 je hotový protokol IPv6 s dvě na sto dvacátou osmou adres, což je přibližně tři sta čtyřicet sextilionů. IPv6 adresa je zapsaná jako osm hexadecimálních čísel v rozsahu 0000-FFFF oddělených dvojtečkou. Ačkoliv s počtem adres je problém již tři desetiletí, změna stále neproběhla, protože by bylo nutné nahradit celou infrastrukturu, což je velice nákladné. [16–18]

Aby mohl internet dále fungovat, bylo potřeba udělat opatření, které sníží počet potřebných adres na internetu. Tím, že ze seznamu všech možných adres se část vyhradí pro podsítě, se umožní, aby se tyto adresy opakovaly. K určení, zda jsou zařízení ve stejné podsíti, slouží masky, které v binárním zápisu v místě, kde se musí shodovat mají jedničku a v části adresy určující konkrétní zařízení nulu. Routery a modemy mají dvě adresy. Jednu pro vnitřní síť (obvykle značenou jako LAN = Local Area Network) a jednu pro vnější síť (obvykle značenou jako WAN = Wide Area Network). Přijde-li packet s adresou odpovídající masce vnitřní sítě, je přesměrován do zařízení nacházejícího se ve stejné síti. V opačném případě je pomocí NAT (Network Address Translation) nahrazena adresa

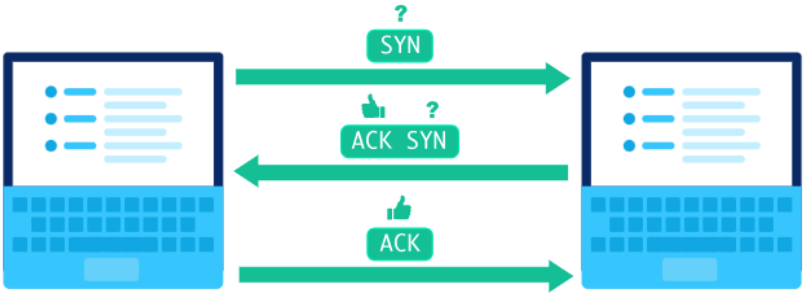
zařízení ve vnitřní síti na vnější adresu routeru a paket je odeslán mimo lokální síť. Kromě umožnění připojení více zařízení, než kolik je IPv4 adres zvyšuje NAT bezpečnost sítě. Jelikož všechna zařízení jsou na WAN viditelná pod jednou adresou, je pro útočníky obtížné zjistit podobu vnitřní sítě. [19, 20]

3.2.2 UDP

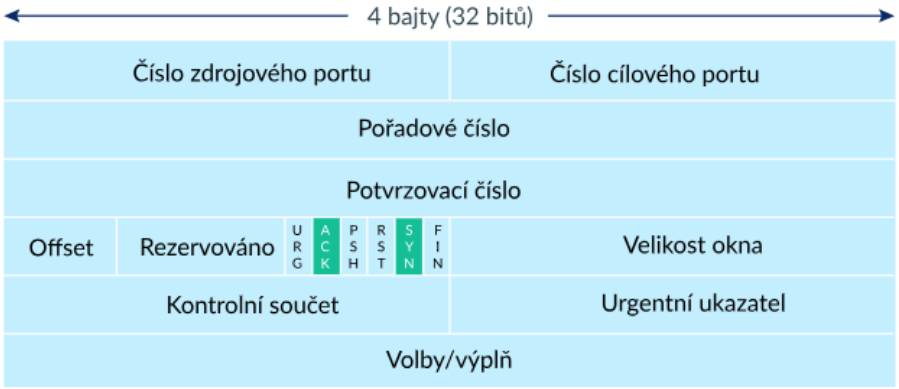
Jeden ze dvou hlavních protokolů čtvrté vrstvy (viz kap. 3.1.4) OSI modelu je UDP (User Datagram Protocol), sloužících ke komunikaci na síti. Má velice jednoduchý princip, kdy pakety pošle do cílové destinace bez navazování spojení, či ověřování, že všechny dorazily v pořádku. Je vhodný především v situacích, kdy je důležitější rychlost než spolehlivost, nebo když se nehodí očekávat odpověď. Typickým příklad použití je přehrávání audia a videa, videohovory a online hry, kdy opakované vysílání ztracených paketů již nemá smysl. Aplikace ovšem musí počítat se situacemi, kdy některé z paketů budou ztraceny, duplikovány, nebo dorazí v jiném pořadí, než byly odeslány. [21]

3.2.3 TCP

Druhým z hlavních protokolů čtvrté vrstvy OSI modelu je TPC (Transmission Control Protocol). Na rozdíl od UDP je zde zajištěno, že když pakety dorazí ve špatném pořadí, budou seřazeny správně. V případě, kdy je paket ztracen, požádá o opakované poslání konkrétního paketu. Před zahájením komunikace je navázáno spojení pomocí třístupňového ověření (anglicky Three-way handshake) kdy, jak je vidět na Obr. 4 jedna strana žádá o navázání spojení, druhá potvrdí žádost a současně požádá o spojení, které první strana potvrdí. Toto je provedeno pomocí bitů *SYN* a *ACK* v TCP hlavičce (viz Obr. 5), kde datová část bývá obvykle prázdná. Jakmile je navázáno spojení, začíná odesílající strana posílat pakety, po jejich obdržení přijímající strana pošle potvrzení. Pokud do stanovené doby není obdrženo potvrzení, předpokládá se, že byl paket ztracen a je zopakováno jeho odeslání. Obdrží-li příjemci paket s vyšším číslem, než které očekává, pošle potvrzení očekávaného, čímž dá odesilateli najevo, že má špatné pořadí a potřebuje znovu poslat chybějící. Příjemce si může podle pořadových čísel obdržené pakety seřadit do správného pořadí a rozeznat duplicity. Pokud chce jedna ze stran spojení ukončit, zopakuje podobný postup jako při navazování spojení, ale místo *SYN* je v logické jedničce bit *FIN*. [22, 23]



Obr. 4 Třístupňové ověřování [22]



Obr. 5 TCP hlavička [22]

3.2.4 DHCP

Protokol DHCP (Dynamic Host Configuration Protocol) umožňuje automatickou konfiguraci IP adres v síti. Bez DHCP by bylo nutné manuálně přidávat a odebírat zařízení ze seznamu adres a na zařízení nastavovat adresu brány (lokální IP adresa routeru), masku sítě a zajistit že adresa zařízení je v síti unikání. DHCP má k dispozici seznam dostupných adres, které přiřazuje nově připojeným zařízením. Když se zařízení odpojí, je adresa opět dostupná a je možné ji přiřadit jinému zařízení. Jelikož je proces automatizovaný, je eliminována lidská chyba a je usnadněna správa sítě. [24]

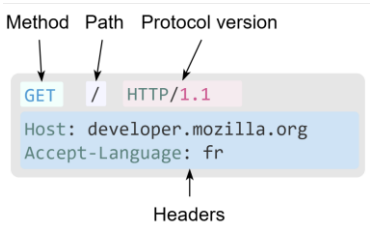
3.2.5 HTTP a HTTPS

Protokol HTTP (Hypertext Transfer Protocol), fungující na sedmé vrstvě OSI modelu, je základem výměny dat na internetu. Jedná se klient-server protokol, kdy klient pošle požadavek na server, který ho zpracuje a pošle zpět odpověď. Byl vyvinut počátkem devadesátých let dvacátého století jako rozšiřitelný protokol, což umožňuje kromě textu

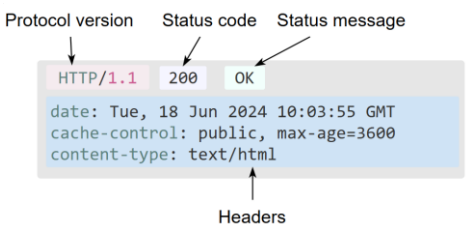
posílat i obrázky, videa a další datové soubory. Nové funkce lze snadno doplnit přidáním nového atributu do hlavičky dotazu. HTTP je bezstavový protokol, ale umožňuje využít cookies soubory, které jsou uloženy u klienta a v případě potřeby mohou být přiloženy k dotazu. Ke komunikaci se využívá TCP (Kap. 3.2.3) protokol, kvůli vytváření spojení. [25]

Mezi klientem a serverem mohou být proxy servery, které dotazy pouze přeposílají, nebo mají jednu či více funkcí. První možnou funkcí je cache, která má uložené odpovědi pro časté dotazy, takže není potřeba zatěžovat server [26]. Druhou je odfiltrování potencionálně škodlivých dotazů. Třetí možnou funkcí je load balancing, kdy klient volá proxy server, který pak podle vytížení jednotlivých serverů zvolí, na který z nich bude dotaz přeposlán, například podle lokace nebo zajištění rovnoměrného rozložení zátěže [27]. Čtvrtou funkcí je autorizace dotazů, aby se ke zdrojům nedostala neoprávněná osoba. Poslední z běžně využívaných funkcí je logování dotazů, které mohou být zpětně použity k analýze (např. kvůli optimalizaci nebo při vyšetřování incidentu). [25]

Verze HTTP/1.1 a starší jsou v podobě, která je čitelná pro lidi. Od verze HTTP/2.0 jsou zprávy zabaleny do rámců, které umožňují kompresy a multiplexing. Struktura zprávy se liší v závislosti na tom, zda se jedná o dotaz, nebo odpověď (viz Obr. 6 a Obr. 7). U dotazu je nutné uvést, jaká metoda se má provést. Nejběžnější jsou GET pro načtení dat a POST pro odeslání dat v těle dotazu. Cesta je adresa od kořenového adresáře k zdroji nebo endpointu, o který klient žádá. Hlavička obsahuje dodatečné informace pro server, jako je například autorizace, očekávaný jazyk, způsob kódování a další. Obdobný význam má hlavička odpovědi pro klienta, ale místo metody a cesty obsahuje status kód a zprávu. Kód je tříciferné číslo, u něhož stovky určují kategorii a zbylé dvě číslice konkrétní stav. Jednička jsou informační zprávy, ale nejsou využívány tak často jako ostatní. Dvojka na začátku znamená, že dotaz byl v pořádku zpracován. Trojka značí přesměrování dotazu jinam. Čtyřka znamená chybu na straně klienta, zatímco pětka je chyba na straně serveru. [25, 28, 29]

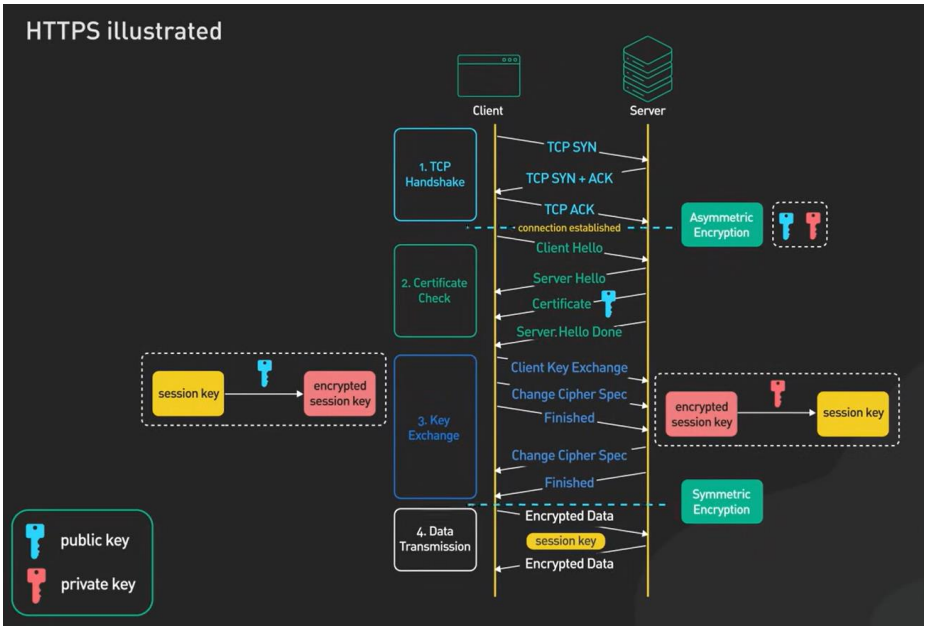


Obr. 6 HTTP-dotaz [25]



Obr. 7 HTTP-odpověď [25]

Jelikož HTTP je nešifrované, je možné komunikaci odchytit a přečíst si obsah. Z tohoto důvodu bylo vytvořeno HTTPS (Hypertext Transfer Protocol Secure), které využívá SSL/TLS (Secure Sockets Layer/Transport Layer Security), jenž jsou založeny na asymetrické kryptografii, kdy data zašifrovaná pomocí veřejného klíče, mohou být dešifrována pouze soukromým klíčem [30]. Ačkoli HTTPS vniklo již v roce 1994, bylo využíváno především pro bankovnictví a v roce 2013 ho používala pouze čtvrtina webů. Díky organizacím jako Let's Encrypt, jež poskytují bezplatné certifikáty, toto číslo v roce 2020 vzrostlo na 80 % a kolem této hodnoty se pohybuje dodnes [31, 32]. Aby bylo možné navázat spojení, musí server mít platný certifikát vystavený nezávislou certifikační agenturou. V opačném případě klient ukončí komunikaci. Během navazování spojení je proveden TLS handshake, který ve verzi 1.2 (viz Obr. 8) probíhá tak, že klient pošle serveru seznam podporovaných šifer a server odpoví, co během komunikace budou používat. Obě tyto zprávy obsahují náhodné číslo, které druhá strana použije k vygenerování klíče, například pomocí RSA nebo Diffie-Hellman algoritmu. Toto číslo dále brání útočníkovi v použití dříve odchycené komunikace. Server poté pošle svůj certifikát obsahující veřejný klíč a klientovu zprávu zašifrovanou soukromým klíčem. Klient použije veřejný klíč certifikační agentury, která měla certifikát vydat k ověření jeho pravosti. Poté klíčem serveru dešifruje zprávu, čímž ověří, že odesilatel disponuje příslušným soukromým klíčem. Server dále pošle zprávu, kterou oznamuje, že poslal všechny potřebné údaje. Klient pošle svůj premaster secret, oznámení konce nešifrované komunikace a zašifrovaný hash dosavadní komunikace. Server také pošle zašifrovaný hash. Pokud se tyto dvě shrnutí liší, znamená to, že někdo sedí uprostřed a další komunikace není bezpečná. Od této chvíle může probíhat bezpečná komunikace. TLS 1.3 tuto výměnu zkracuje a zakazuje použití šifer, které již byly prolomeny, ale mnoho serverů a klientů stále využívá TLS 1.2, které je zpětně kompatibilní se staršími verzemi. [33–38]



Obr. 8 HTTPS komunikace [34]

3.3 Datové formáty

Za běhu programu jsou situace, kde je třeba objekty v paměti uložit či přenést do jiného programu. Tomuto procesu se říká serializace. Dochází během něho k zachycení aktuálního stavu objektu, který je reprezentován kombinací hodnot v jeho vlastnostech. Serializovat je možné pouze data nikoliv metody. Reverzní operaci, kdy je tato reprezentace převedena zpět na objekt, se nazývá deserializace. Nejčastější formáty jsou často součástí systémových knihoven daného jazyka, nebo existuje knihovna třetí strany. Je důležité, aby se shodovala struktura v programu i serializované verze, protože v opačném případě může při deserializace nastat chyba. [39]

Data je možné přenášet a ukládat v binární nebo textové podobě. Při použití binární podoby je zpracování rychlejší, ale obsah je pro člověka nečitelný a všechny zúčastněné strany musí znát význam jednotlivých bitů. Textová podoba je čitelná pro všechny, což umožňuje snadnou editaci a jednodušší hledání příčin chyb, protože si programátor může lehce ověřit, zda mají data očekávanou podobu. Nevýhodou je nutná konverze do příslušných datových typů. [40]

3.3.1 XML

Značkovací jazyk XML (eXtensible Markup Language) popisuje strukturu dat. Oproti některým jiným značkovacím jazykům neobsahuje informaci o jejich významu. Ten musí znát aplikace, což znamená, že XML vytvořený jedním programem, může být pro jiný nečitelný. Na Obr. 9 je ukázka XML s kolekcí psů. Dokumenty obvykle začínají nepovinnou značkou obsahující verzi a kódování. Veškerý obsah musí být zabalen do jednoho kořenového prvku. Jednotlivé značky mohou být rozšířeny o atributy obsahující doplňující informace o textu, který je jimi ohraničen. Nejčastěji se jedná o identifikátory nebo vzhled. Hodnoty jsou zapisovány do uvozovek. XML a značkovací jazyky na něm založené podporují komentáře, které jsou při zpracování ignorovány. Dále je možné přidat sekci *CDATA*, jejíž obsah je ponechán nezměněn, což je využíváno, pokud je třeba uložit text obsahující značkovací jazyk, který byl jinak zpracován. [40]

Při zpracování XML jsou rozlišovány dva základní typy. SAX (Simple API for XML) projde dokument pouze jednou a v závislosti na právě přečtené značce vyvolá příslušnou událost. Tento přístup vyžaduje méně paměti, ale aplikace si musí pamatovat vztahy mezi jednotlivými daty. Oproti tomu DOM (Document Object Model) uchovává celý dokument ve stromové struktuře. Tento přístup potřebuje více paměti, ale kdykoli se dá přistoupit k jakémukoliv prvku včetně jeho kontextu. DOM je využíván například u webových stránek, ODF (OpenDocument Format používaný v OpenOffice), Open XML (využívaný v Microsoft Office od verze 2007), SVG (Scalable Vector Graphics) nebo .NET aplikacích využívajících XAML (eXtensible Application Markup Language). [41]

```
<?xml version="1.0"?>
<psi>
  <pes>
    <jmeno>Rex</jmeno>
    <vek>6</vek>
    <plemeno>Nemecky Ovcak</plemeno>
  </pes>
  <pes>
    <jmeno>Alik</jmeno>
    <vek>5</vek>
    <plemeno>Jack Russel terier</plemeno>
  </pes>
</psi>
```

Obr. 9 Příklad XML

3.3.2 JSON

Formát JSON (JavaScript Object Notation) vznikl původně pro převod objektů v JavaScript do textového řetězce. JSON podporuje pouze základní datové typy, jako jsou textové řetězce, čísla, logické hodnoty a objekty, či pole z těchto typů sestavené. V případě potřeby je možné do objektu vnořit další objekt. Zápis je tvořen páry skládajících se z názvu a hodnoty oddělených dvojtečkou. Jednotlivé páry jsou od sebe odděleny čárkou. Veškerý obsah musí být obalen složenými závorkami označující objekt, nebo hranatými závorkami značící kolekci. JSON byl implementován mnoha jazyky, jako jsou například C, C++, C#, Java, Python a mnoho dalších. Navzdory svému názvu není závislý na konkrétním jazyku, což ho dělá ideální volbou pro sdílení dat mezi programy napsanými v různých technologiích. Za nevýhodu by se dala označit nemožnost používat komentáře. Na Obr. 10 je příklad zápisu kolekce psů. [42, 43]

```
[
 {
  "jmeno": "Rex",
  "vek": 6,
  "plemeno": "Nemecky Ovcak"
 },
 {
  "jmeno": "Alik",
  "vek": 5,
  "plemeno": "jack Russell terrier"
 }
]
```

Obr. 10 Příklad JSON

3.3.3 CSV

Formát CSV (Comma-Separated Values) je používaný k ukládání tabulek. Jedná se o jednoduchý a hojně rozšířený formát pro import a export dat. Každý řádek textu odpovídá jednomu řádku v tabulce. Jak název napovídá, sloupce jsou většinou oddělovány čárkou, ale v některý případech je nutné použít jiný oddělovací znak (obvykle středník nebo svislítko). Příkladem takové situace jsou desetinná čísla, kde se v češtině používá desetinný čárka místo tečky, jež se používá v angličtině. [44]. První řádek se většinou využívá k pojmenování jednotlivých sloupců. Na Obr. 11 je příklad z tabulky psů. Oproti ostatním formátům má výhodu v menší velikosti, neboť význam hodnoty je definován pouze jednou, nikoli pro

každou instanci. Toto s sebou ovšem nese nevýhodu, že jeden soubor může obsahovat pouze záznamy stejného typu, jelikož v opačném případě se nebudou shodovat sloupce. CSV má nezastupitelné využití při exportu dat z databází a v situacích, kdy se předpokládá, že data budou zpracovávána uživatelem například pomocí nástrojů jako je Microsoft Excel. [45]

```
jmeno,vek,plemeno
Rex,6,Nemecky Ovcak
Alik,5,Jack Russell terier
```

Obr. 11 Příklad CSV

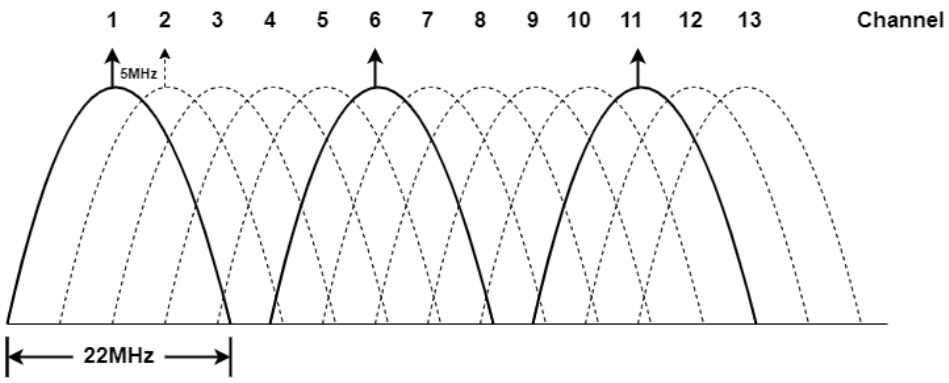
3.4 Wi-Fi

Wi-Fi je uživatelsky přívětivější název pro bezdrátovou síťovou technologii definovanou standardem IEEE (Institute of Electrical and Electronics Engineers) 802.11, popisující první a druhou vrstvu OSI modelu (viz Kap. 3.1), jehož první verze vznikla v roce 1997. Navzdory běžnému omylu se nejedná o zkratku pro „Wireless Fidelity“, ale pouze o snadno zapamatovatelný název [46, 47]. Větší rozšíření Wi-Fi nastalo po uvedení Apple AirPort v roce 1999, který využíval 802.11b. Původní verze Wi-Fi měla maximální šířku pásma pouze 2 Mb/s a využívala frekvenční pásmo 2,4 GHz. 802.11b fungoval na stejné frekvenci, ale zvýšil přenosovou rychlost na 11 Mb/s. Kromě vyšší rychlosti také využíval modulační schéma DSSS/CCK (Direct-Sequence Spread Spectrum/Complementary Code Keying), snižující vliv rušení způsobeného mikrovlnnými troubami, bezdrátovými telefony a jinými zdroji elektromagnetického záření. Ve stejném roce vyšel také standart 802.11a pracující ve frekvenčním pásmu 5 GHz s maximální rychlostí 54 Mb/s. Bylo zde také poprvé představeno OFDM (Orthogonal Frequency-Division Multiplexing). 5 GHz má oproti 2,4 GHz výhodu vyšší rychlosti, ale za cenu kratšího dosahu. V roce 2003 byl představen standart 802.11g přinášející technologie představené v 802.11a také na 2,4 GHz síť. V následujících letech přibyly další verze (viz Tab. 1) s vyšší přenosovou rychlostí, dosahem a pokročilými technologiemi, umožňující vyšší spolehlivost, bezpečnost a komunikaci více zařízení současně. [48–50]

Tab. 1 Verze Wi-Fi [48, 51]

standart	Wi-Fi	rok	Frekvence [GHz]	Přenosová rychlost (teoretická)	Šířka kanálu [MHz]
802.11	Wi-Fi 0	1997	2,4	2 Mb/s	20
802.11a	Wi-Fi 2	1999	5	54 Mb/s	20
802.11b	Wi-Fi 1	1999	2,4	11 Mb/s	20
802.11g	Wi-Fi 3	2003	2,4 + 5	54 Mb/s	20
802.11n	Wi-Fi 4	2009	2,4 + 5	600 Mb/s	20, 40
802.11ac	Wi-Fi 5	2013	5	3,5 Gb/s	20, 40, 80, 160
802.11ax	Wi-Fi 6	2019	2,4 + 5	9,6 Gb/s	20, 40, 80, 160
802.11ax	Wi-Fi 6E	2021	2,4 + 5 + 6	9,6 Gb/s	20, 40, 80, 160
802.11be	Wi-Fi 7	2024	2,4 + 5 + 6	46 Gb/s	20, 40, 80, 160

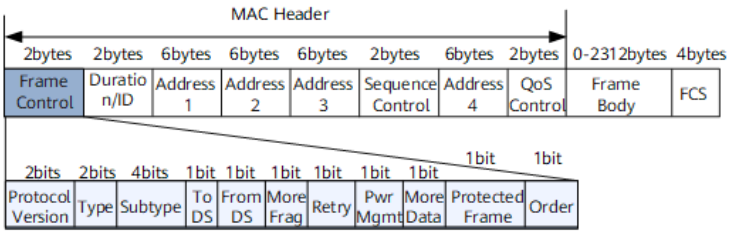
Wi-Fi má v daném frekvenčním pásmu vymezený určitý rozsah frekvencí, které jsou rozděleny na 22 MHz úseky nazývané kanály. Dostupnost těchto kanálů se liší v závislosti na regulacích telekomunikačních úřadů jednotlivých států. Wi-Fi v pásmu 2,4 GHz může teoreticky mít až čtrnáct kanálů. V České republice je stejně jako ve většině Evropy a Spojených státech možné využít třináct kanálů, odpovídající frekvencím od 2,4000 do 2,4835 GHz. Jednotlivé kanály mají však rozestupy pouze 5 MHz, což znamená, že sousední čtyři kanály na obě strany jsou vzájemně rušeny (viz Obr. 12). Pro vyšší datovou propustnost je možné zvolit i jinou šířku (viz Tab. 1), ale za cenu ztráty zpětné kompatibility se staršími zařízeními a menší počet vzájemně nerušených kanálů. [52, 53]



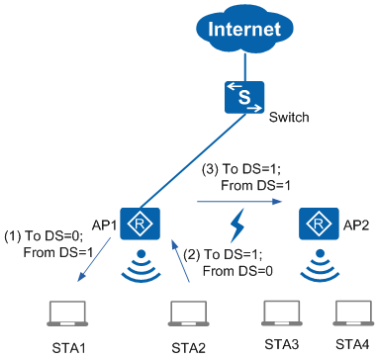
Obr. 12 Překryv kanálů 2,4 GHz [52]

K realizaci bezdrátové sítě neboli WLAN (Wireless Local Area Network) je potřeba zařízení nazývané AP (Access Point). Jedná se zařízení vysílající bezdrátový signál, který mohou zachytit koncová zařízení (označovaná jako stanice či zkráceně STA) v dosahu. K připojení do této sítě je potřeba znát SSID (Service Set IDentifier) a heslo (pokud není síť nezaheslovaná). SSID je možné zadat ručně, pokud ho uživatel zná předem, nebo ho získat ze speciálních paketů nazývaných beacon (někdy také SSID broadcast), které AP pravidelně vysílá na všech kanálech. Obvykle bývá součástí routeru, ale může se jednat i o samostatné zařízení. Síť může být tvořena jedním či více AP, která jsou propojena kabelem. [54, 55]

Oproti rámci pro Ethernet (IEEE 802.3), kterému k úspěšnému doručení stačí pouze dvě MAC adresy (viz Obr. 3), obsahuje Wi-Fi rámec (jehož podoba je detailněji popsána na Obr. 13) čtyři adresy. O jejich významu rozhodují devátý a desátý bit hlavičky, které obsahující informaci o směru toku dat (viz Obr. 14). V závislosti na situaci se může jednat o MAC adresu zařízení, nebo BSSID (Basic Service Set Identifier [56]) sítě vysílané určitým AP (viz Tab. 2). Rámce mohou mít několik významů, které určují třetí až osmý bit hlavičky. Bit *More Frag* slouží jako indikátor, zda byl paket rozdělen na více rámců (viz Kap. 3.1.2). IEEE 802.11 obsahuje také úsporný režim, kdy koncové zařízení vypne napájení antény za účelem úspory energie. V případě změny tohoto stavu posílá koncové zařízení AP rámec, který neobsahuje žádná data. Bit *Pwr Mgmt* říká, zda po odvysílání tohoto rámce bude zařízení aktivní, nebo v úsporném režimu. S tím souvisí i další bit určující, zda má být rámec odvysílán, nebo uložen do doby, než bude cílové zařízení probuzeno. [57]



Obr. 13 Rámec Wi-Fi [57]



Obr. 14 Význam DS bitů [57]

Tab. 2 Význam adres v Wi-Fi rámci [57]

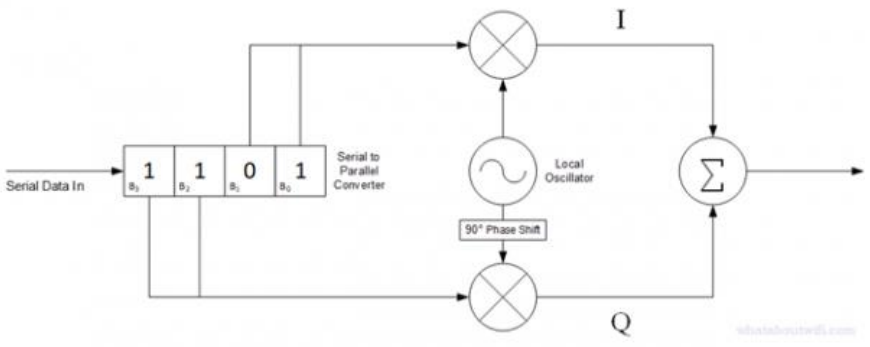
To DS	From DS	ADR 1	ADR 2	ADR 3	ADR 4	Situace na Obr. 14
0	0	Cílová adresa	Zdrojová adresa	BSSID	-	Beacon
0	1	Cílová adresa	BSSID	Zdrojová adresa	-	(1) AP1 posílá data STA1
1	0	BSSID	Zdrojová adresa	Cílová adresa	-	(2) STA2 posílá data AP1
1	1	Cílová BSSID	zdrojová BSSID	Cílová adresa	Zdrojová adresa	(3) AP1 posílá data AP2

Od 802.11ax se pro zjednodušení pro koncové uživatele místo značení verze pomocí standardu používá číslo generace (viz Tab. 1). Ačkoliv finální verze IEEE standardu vyšla až v roce 2021, Wi-Fi Aliance vydávala certifikáty již od roku 2019. Primárním cílem nové verze je zvýšení schopnosti současně komunikovat s více uživateli v prostředích s velkým množstvím zařízení, jako jsou například sportovní stadiony a dopravní uzly. Díky technologii OFDMA (Orthogonal Frequency Division Multiple Access) je možné jednotlivé subcarriery (viz Kap. 3.4.4) rozdělit na menší úseky, které mohou být přiřazeny jednotlivým zařízením. Zavedením plánování komunikace je snížen počet kolizí na síti, čímž je zvýšena datová propustnost, neboť není potřeba opakovat vysílání. Zlepšena byla také bezpečnost použitím WPA3 využívajícího SAE (Simultaneous Authentication of Equals). Kromě toho se také prodloužila výdrž baterií napájených zařízení, neboť nyní místo soustavného kontrolování, zda mu někdo něco neposílá, je probuzeno až v případě potřeby. 802.11ax obsahuje také Wi-Fi 6E pracující v nově uvolněném frekvenčním pásmu 6 GHz umožňující přenášet větší množství dat. [49, 58–60]

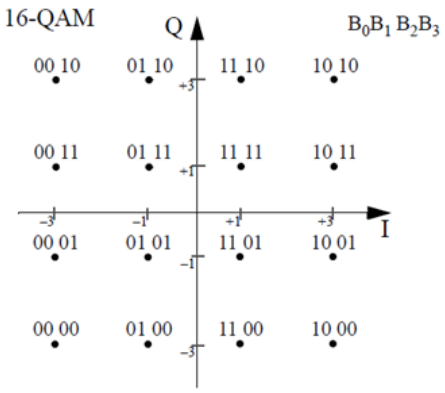
3.4.1 QAM

Modulace QAM (Quadrature Amplitude Modulation) je způsob, jak do bezdrátového signálu zakódovat více informací. Tato modulace mění amplitudu a fázi signálu. Během modulace jsou data rozdělena na polovinu. Obě tyto části jsou modulovány pomocí sinusoid,

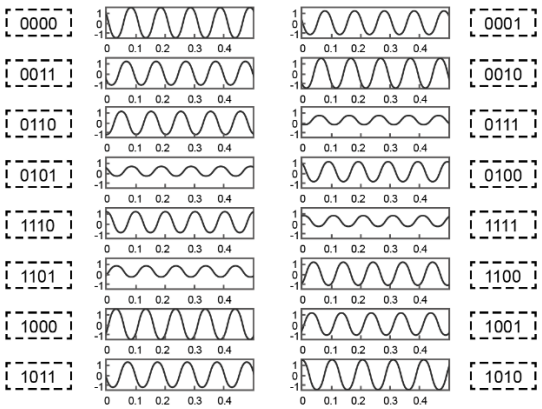
které jsou vůči sobě posunuty o devadesát stupňů (viz Obr. 15). Polovina obsahující LSB (Least Significant Bit) není fázově posunuta a označuje se proto jako I (In phase). Polovina obsahující MSB (Most Significant Bit), jenž je fázově posunuta se označuje jako Q (Quadrature). Obě tyto sinusoidy jsou poté sečteny, čímž je získán výsledný signál k odvysílání (viz Obr. 17). Možné stavy se dají znázornit pomocí dvourozměrného grafu, kde na ose x je I a na ose y je Q (viz Obr. 16). Množství možných hodnot je přímo v názvu použité modulace. Například 256-QAM znamená, že signál může nabývat dvě stě padesát šest různých stavů, tedy přenáší osm bitů. Z tohoto značení je výjimkou QPSK, který odpovídá 4-QAM. Ovšem s vyšším počtem možných stavů se zvyšuje také jejich hustota, což znamená, že v případě rušení nemusí být symbol správně rozpoznán. Proto vyšší QAM je možné použít pouze na kratší vzdálenosti.[61]



Obr. 15 Schéma QAM modulátoru **[61]**



Obr. 16 Graf 16-QAM **[61]**



Obr. 17 Signál 16-QAM **[62]**

3.4.2 Šifrování komunikace

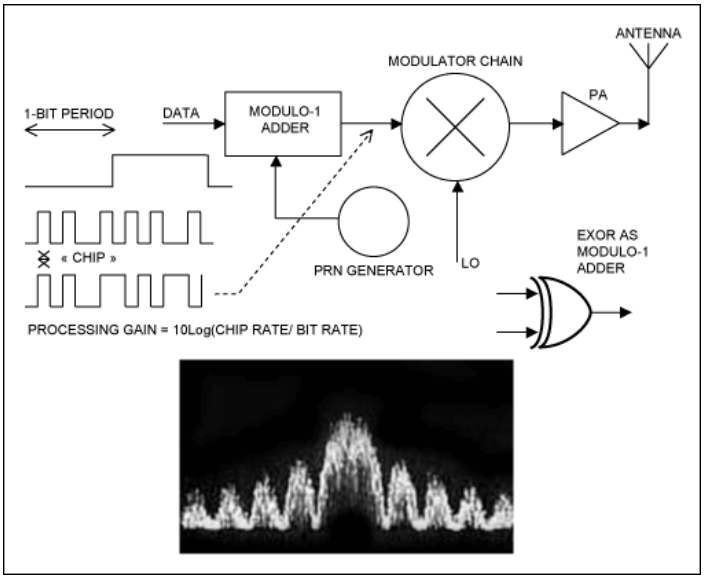
Jelikož data nejsou data vysílána do zařízení přímo, ale je možné je zachytit jinou anténou poblíž, je nutné provádět šifrování, aby k nim nezískala přístup neoprávněná osoba. Prvním protokolem byl WEP (Wired Equivalent Privacy), který používal šedesáti čtyř nebo sto dvaceti osmi bitový statický klíč. Ten byl stejný pro všechna zařízení, a tudíž chránil pouze před útočníky, kteří nebyli k síti připojeni. Kromě toho nová zařízení mají dostatek výkonu k prolomení šifry a není proto doporučeno WEP nadále používat. V roce 2003 s 802.11g přišlo WPA (Wi-Fi Protected Access), řešící zranitelnost statických klíčů využitím TKIP (Temporal Key Integrity Protocol). Ten je generován pro každý paket. Jelikož je klíč jednorázový, má útočník méně informací použitelných k jeho zjištění. Dále WPA obsahuje mechanismus k ověření integrity dat v případě, že s nimi bylo manipulováno. O rok později bylo představeno WPA2 využívající AES (Advanced Encryption Standard). Navíc byla vylepšena i autorizace, kdy kromě hesla používaného v soukromém režimu, přibyl enterprise režim využívající EAP (Extensible Authentication Protocol), který ověřuje identitu vůči serveru. Nejnovější verze WPA3 z roku 2018 vnikla kvůli objeveným zranitelnostem WPA2 (CVE-2017-13077 až CVE-2017-13088 souhrnně označované jako KRACKs [63]). Klíče jsou nyní unikátní pro každý přenos a mají sto devadesát dva bitů v osobním režimu a dvě stě padesát šest v enterprise režimu. [64]

3.4.3 Spektrální rozprostření

Jedná se o metody používané k snížení vlivu rušení a zvýšení bezpečnosti bezdrátově přenášených signálů. Základem je výzkum z roku 1941, ve které se herečka Hedy Lamarr a pianista George Antheil snažili najít způsob, jak zabránit rušení signálu pro radiem řízená torpéda. Americká armáda toto řešení však odmítla. Dnes je využíváno pro Wi-Fi, Bluetooth, mobilní sítě a GPS (Global Positioning System). Hlavní myšlenkou je rozprostřít signál přes více frekvencí, díky čemuž je potřeba menší energie, neboť v případě interference nejsou ovlivněny všechny a není tedy nutné, aby byl signál silnější než šum. S tím souvisí složitější odposlech komunikace, protože bez znalosti příslušných frekvencí se signál od šumu nedá odlišit (platí obzvláště pro DSSS). Toto navíc umožňuje ve stejném frekvenčním pásmu vysílat více signálů. [53, 65, 66]

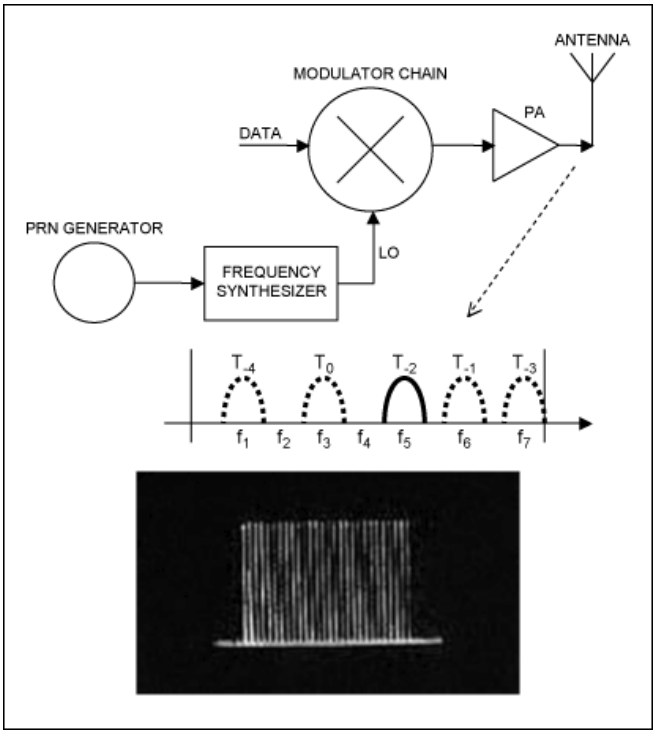
Metoda rozprostření DSSS (Direct Sequence Spread Spectrum) kombinuje data s pseudonáhodným kódem o vyšší frekvenci, než jsou data. Jednotlivé hodnoty tohoto kódu

se nazývají chipy. Jeden bit je přenášen pomocí jedenácti chipů, které jsou s daty zkombinovány pomocí funkce XOR (viz Obr. 18). Oproti ostatním spektrálním rozprostřením má výhodu vyšší odolnosti proti šumu, ale za cenu potřeby širšího frekvenčního pásma, kvůli čemuž má méně dostupných kanálů. [53, 65]



Obr. 18 DSSS [65]

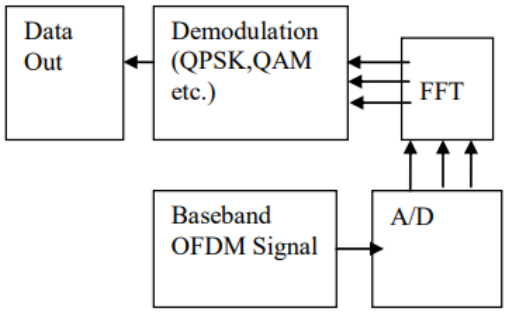
Metodu FHSS (Frequency-Hopping Spread Spectrum), jež vychází z původního patentu, je možné použít jako alternativu k DSSS, či je zkombinovat dohromady. Oproti DSSS nemanipuluje přímo s daty, ale provádí skoky mezi sedmdesáti osmy frekvenčními pásmy (viz Obr. 19). Jelikož každých pár bitů mění frekvenci, útočník tak není schopen zachytit celou zprávu. V případě rušení v daném rozsahu, může díky úzkým kanálům stále využívat ty, které nejsou rušené. Má menší datovou propustnost, protože na přeskok potřebuje více času. Navzdory silné konkurenci v podobě DSSS, jenž má lepší odstup signálu od šumu, je FHSS stále populární především u vysílaček a radiofrekvenčních dálkových ovládání. [53, 65, 66]



Obr. 19 FHSS [65]

3.4.4 OFDM

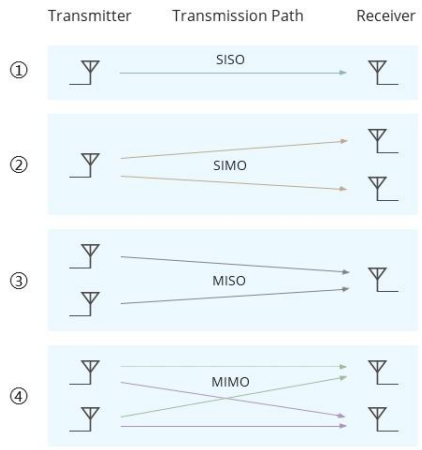
OFDM (Orthogonal Frequency-Division Multiplexing) je způsob, jak vyřešit problém s odrazy signálu, které komplikují rozeznaní jednotlivých bitů, kvůli echu. Data jsou rozdělena mezi více samostatně modulovaných signálů označovaných jako subcarriers, které jsou voleny tak, aby v momentě, kdy je daná frekvence na vrcholu, byly všechny ostatní v nule. Poté co je provedena modulace, jsou signály sečteny a odeslány pomocí antény. Na straně přijímače je signál pomocí rychlé Fourierovi transformace (FFT = Fast Fourier Transform) opět rozložen a demodulován (viz Obr. 20). [67]



Obr. 20 OFDM přijímač [68]

3.4.5 MIMO

Mechanismus MIMO (Multiple-Input Multiple-Output) umožňuje v jeden okamžik na straně vysílače i přijímače využít více antén. Poprvé bylo představeno v 802.11n v podobě SU-MIMO (Single User MIMO). Podobně jako jeho předchůdci, kdy pouze jedna ze stran měla dvě antény (viz Obr. 21), sloužilo k vyšší spolehlivosti. SU-MIMO využívá všechny antény pro stejný signál, tudíž má přijímač více informací umožňující vyčistit data od šumu, nebo pokud má vyšší šanci obdržet data, je-li signál rušen či odražen od překážky. Dále může být anténa navíc využita k zvýšení rychlosti přenosu tím, že jsou data rozdělena na více částí a každá poslána jako samostatný signál. S příchodem 802.11ac byl tento mechanismus rozvinut do podoby MU-MIMO (Multi User MIMO) umožňující každé anténě AP komunikovat s jiným zařízením. Počet souběžných komunikací je limitován schopnostmi AP. Tento údaj je vyjadřován pomocí *MxN*, kde M je počet antén pro vysílání a N je počet antén pro příjem. Obvykle umí jedna anténa plnit obě funkce. [69–71]



Obr. 21 SISO, SIMO, MISO, MIMO [71]

3.5 Jednočipové počítače

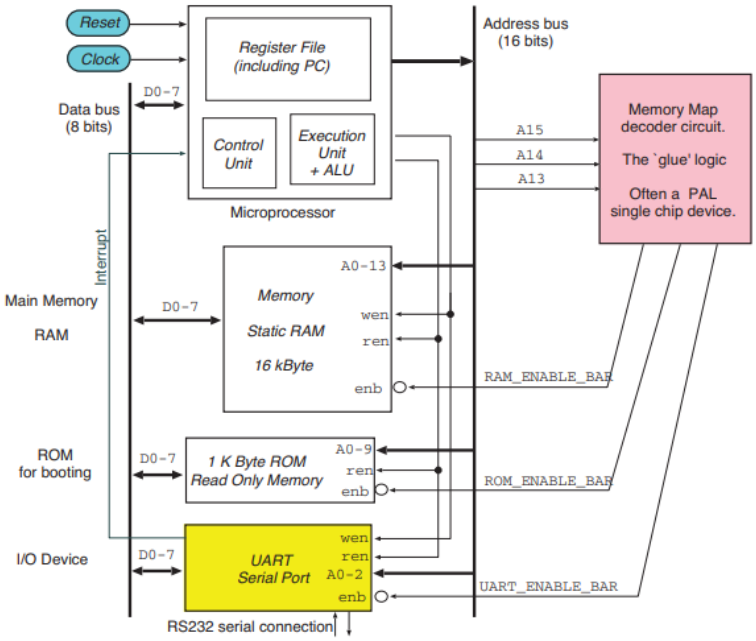
V oblasti obvodů s vysokým stupněm integrace (VLSI = Very large Scale Integration), tedy integrovaný obvod, jenž obsahují více zařízení, je několik pojmů, které si jsou velice blízké a někdy dochází k jejich záměně. [72, 73]

Mikroprocesor je označení integrovaného obvodu obsahující vykonávající a řídící jednotku, které dohromady tvoří CPU (Central Processing Unit). Jejich umístění do jednoho

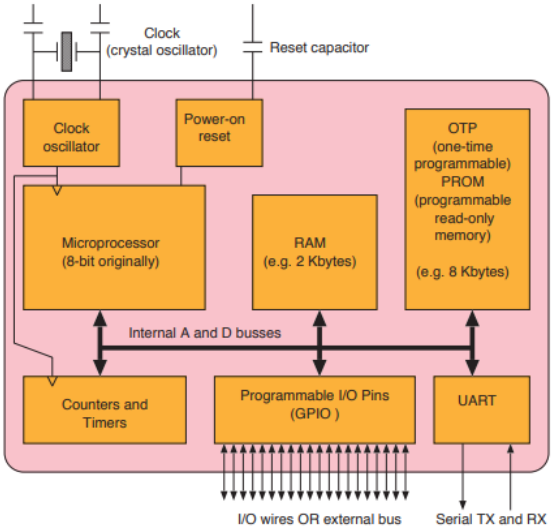
čipu zvyšuje spolehlivost, neboť je sníženo množství míst, kde by mohl nastat problém při kombinaci více čipů. Prvním integrovaným obvodem tohoto typu byl čtyř bitový Intel 4004 z roku 1971 s frekvencí 740 kHz. Na Obr. 22 je vyobrazena struktura jednoduchého mikropočítače, tedy zařízení založeném na mikroprocesoru. [72, 74]

SoC (System-On-Chip) je integrovaný obvod obsahující všechny základní části počítače v jednom pouzdře. Tyto čipy obsahují procesor, cache, paměť a vstupně výstupní obvody. Toto umožňuje zjednodušení výroby a tím snížení nákladů. Dále zařízení využívající SoC mohou mít menší rozměry a spotřebu než ta používající více čipů. Často je využívána von Neumannova architektura, kdy jsou instrukce i data umístěna do jedné paměti, která je přímo adresována CPU a označuje se jako primární nebo hlavní paměť. [72]

Mikrokontrolery neboli jednočipové počítače, často zkracované jako MCU (MicroController Unit), jsou speciálním případem SoC, jenž nevyužívají externí DRAM (Dynamic Random Access Memory). Většinu vstupně výstupních obvodů a ROM (Read Only Memory) s programem, jenž mají vykonávat, obsahují přímo v sobě. Ke své funkci potřebují pouze zdroj hodinového signálu a napájení (viz Obr. 23). Ve většině případů obsahují čítače, časovače a převodníky analogového signálu na digitální (ADC = Analog-to-Digital Converter). Kromě těchto základních obvodů jsou typicky vybaveny sběrnicemi (např. I2C, SPI a další) umožňujícími připojit sensory, které nepoužívají pouze jednu logickou hodnotu, či napětí v rozmezí 0-3 V, ale komunikují pomocí zpráv tvořených jedním či více bajty. Díky těmto vlastnostem jsou ideální pro úlohy vyžadující zpracování signálů v reálném čase. [72, 75]



Obr. 22 Struktura jednoduchého mikropočítače [72]

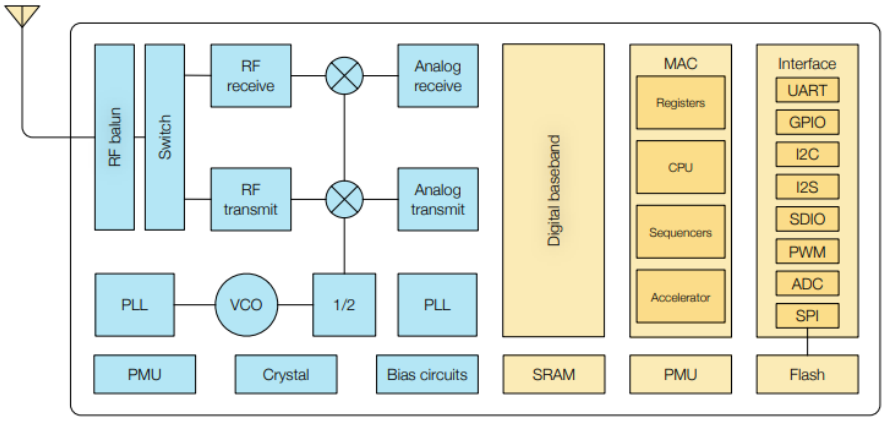


Obr. 23 Struktura mikrokontroleru [72]

3.5.1 ESP8266

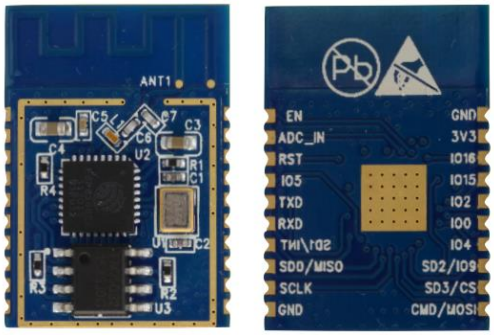
ESP8266EX od společnosti Espressif je SoC s QFN32-pin pouzdrem o rozměrech 5x5 mm kombinující vylepšenou verzi třiceti dvou bitového procesoru Tensilica L106 Diamond series, jehož maximální frekvence může být až 160 MHz, a 2,4GHz Wi-Fi 802.11 b/g/n s rychlostí až 72,2 Mb/s. Je možné ho použít buď samostatně, nebo jako periferii k jinému čipu. K napájení lze využít napětí v rozmezí od 2,5 V do 3,6 V s průměrným odebíraným proudem 80 mA. Odebíraný proud závisí na stavu, v jakém se čip a Wi-Fi

momentálně nachází. V hlubokém spánku, kdy jsou aktivní pouze hodiny, může proud klesnout až na úroveň 20 µA. ESP8266EX (Obr. 24) je vybaveno sedmnácti digitálními GPIO piny, z nich většina má ještě další funkci, na což musí být myšleno při návrhu řešení. Jednotlivé piny mohou být nastaveny jako pull-up nebo pull-down. Pro analogové periferie je možné využít jeden deseti bitový ADC převodník. Ke komunikaci s dalšími čipy je možné využít SPI, I2C nebo UART. ESP8266 má k dispozici dvě SPI schopné fungovat jako master nebo slave s frekvencí 20 MHz. Z toho jedna slouží k připojení externí FLASH paměti. Dále je k dispozici I2C s frekvencí 100 kHz. Pro komunikaci s počítačem je možné využít UART schopné dosáhnout rychlosti 115200 b/s. [76]



Obr. 24 Blokový diagram ESP8266EX [76]

Nejčastěji se dají ESP8266EX sehnat již umístěny na desce plošných spojů společně s anténou, oscilátorem a FLASH pamětí (viz Obr. 25 a Příloha 2). Krystal oscilátoru může mít frekvenci 40, 26 nebo 24 MHz. Tyto hotové moduly se vyrábí v několika provedeních lišících se rozměry, počtem vyvedených pinů a anténou (viz Obr. 26), z čehož jsou nejpopulárnější ESP-01 a ESP-12E. K jejich popularitě výrazně přispívá nízká cena, Wi-Fi a kompatibilita s Arduinem. [77, 78]



Obr. 25 ESP-WROOM-S2 [78]

Obr. 26 Verze modulů [77]

Pro snazší použití jsou moduly připájeny k vývojovým deskám. Ty jsou vybaveny USB konektorem pro nahrávání kódu a napájení. Mezi nejoblíbenější patří NodeMCU využívající ESP-12E. Ten dává programátorovi k dispozici 4 MB FLASH paměti, ADC a jedenáct GPIO. K nahrání kódu jsou využívány převodníky USB-UART CP2101 nebo CH340. Příloha 1 vyobrazuje jaké funkce mají jednotlivé piny této desky. [77]

3.6 Návrhové a architektonické vzory

Návrhové a architektonické vzory jsou léty ověřené techniky pro řešení opakujících se problémů v objektově orientovaném programování. Nejedná se o konkrétní kód, ale jen o koncept. Z tohoto důvodu nejsou svázány s konkrétní technologií a je tak možné je použít v téměř libovolném jazyce. Výhodou takto pojmenovaných a popsaných postupů je, že je zná většina vývojářů po celém světě a při komunikaci stačí říci jaký vzor použít, bez nutnosti vysvětlovat detaily. Tyto dvě skupiny se od sebe liší oblastí, kterou pokrývají. Návrhové vzory se zabývají chováním jedné třídy, vytvářením nových instancí, nebo komunikaci tříd mezi sebou. Oproti tomu architektonické vzory určují sktrukturu celého projektu a mají přímý vliv na jeho modularitu a škálovatelnost. [79–81]

3.6.1 Zapouzdření

Tímto pojmem je obvykle myšlen jeden za základních pilířů objektově orientovaného programování, kdy třída skryje své hodnoty a metody používané pro vnitřní fungování a ostatním přístupní jen ty potřebné ke komunikaci. Tento přístup také pomáhá zajistit konzistenci, protože stav objektu může být upraven pouze zamýšleným způsobem. Toto lze přenést i do většího měřítka, kdy je aplikace rozdělena na více zapouzdřených částí. Aby ostatní části mohly komunikovat nepotřebují znát vnitřní fungování, ale pouze rozhraní.[82]

3.6.2 N-vrstvá architektura

Pro složitější aplikace, nebo tam, kde se očekává potřeba měnit některé celky, se často na základě pokrývané oblasti rozděluje aplikace na části označované jako vrstvy. Obvykle se každá vrstva nachází ve vlastním projektu. Hlavní výhodou je přehledná struktura, ve které se snáze hledá. V kombinaci se zapouzdřením také zvyšuje modularitu a bezpečnost. Jelikož okolní vrstvy vidí pouze rozhraní, a nikoliv konkrétní implementaci, je snadné vrstvu nahradit jinou bez ovlivnění ostatních. Komunikace je obvykle omezena na vrstvy o jednu pod a nad. Případný útočník proto nemůže z nejvyšší vrstvy přistupovat přímo k nejnižší. Rozdělení vrstev sebou však nese komplikaci v podobě komunikace mezi nimi, protože má každá vlastní datové objekty. Pokud se vrstvy nachází fyzicky na jiných serverech, jsou označovány jako tiery. [82, 83]

Nejběžnější je třívrstvá architektura. Nejvyšší vrstva komunikuje s uživatelem a podle typu aplikace se jedná o uživatelské rozhraní, nebo v případě webového API o controller s endpointy. Prostřední a nejdůležitější vrstvou je business logika, která zpracovává požadavky od uživatele. Poslední vrstva se stará o přístup k datům. Tím může být například zápis do databáze, nebo komunikace s jiným systémem.[82]

3.6.3 Dependency injection

Dependency injection je technika, která snižuje závislost třídy na jiné. Toto umožňuje aplikaci být více modulární, lépe testovatelná a snáze upravitelná.[84]

Pokud má třída například zpracovat data a výsledek uložit do databáze, při klasickém přístupu je pevně svázána s konkrétním databázovým systémem. V horším případě obsahuje všechen kód, čímž porušuje Single responsibility principle (S ze SOLID) [85]. V lepším

případě je práce s databází umístěna do vlastní třídy, ale její instance je součástí objektu s logikou, který je zodpovědný za jeho správu. Oba tyto případy komplikují přechod z jednoho typu databáze na jiný a testování je velice obtížné, protože kód očekává připojení k funkční databázi.[84]

Aby se těmto problémům předešlo, je instance této pomocné třídy, která je obvykle označována jako služba, předávána zvenčí. Nyní za správu služby není zodpovědný objekt s logikou, ale injector. Dále třída většinou není závislá na konkrétní třídě, ale na rozhraní definující metody, které je možné zavolat. Díky této abstrakci je možné snadno změnit implementaci. Mimo jiné je takto umožněno místo skutečné implementace použít testovací třídu, která pouze simuluje volání databáze. Služba je nejčastěji vkládána pomocí konstruktoru, ale může být také použita metoda.[84]

Pro drobné projekty může jako injector sloužit prosté zavolání konstruktoru z kódu [84]. Ve většině případů je použit framework, který automaticky řeší vytváření a předávání potřebných instancí. Může se jednat o knihovnu třetí strany, nebo v některých případech přímo o systémovou knihovnu. Od verzí *.NET Core 1.0* a *.NET Framework 4.5* mezi tyto jazyky patří také C#[86]. V závislosti na typu projektu je knihovna již importována, nebo je třeba dodat příslušný NuGet. Při přidávání služby do seznamu je možné definovat životnost instance. První možností je *Transient*, který je při každém zavolání vytvořen nový. Druhou možností je *Singleton*, jehož instance je vytvořena jen jednou. Poslední je *Scoped* využívaný v ASP.NET pro situace, kde je potřeba aby každé zavolání API mělo vlastní instanci. Od .NET 8.0 je přidán atribut *FromKeyedServices* umožňující zaregistrovat více implementace jednoho rozhraní odlišených klíčem a zvolit implementaci podle aktuální potřeby. [87]

3.6.4 Data Transfer Object (DTO)

Data transfer Object je instance třídy sloužící k přenosu dat mezi systémy. Použití speciálních objektů umožňuje skrýt hodnoty používané k vnitřní funkci jedné strany, ale pro druhou stranu zbytečných nebo jejichž přenos by mohl být bezpečnostní hrozbou. Současně je takto snížen objem dat, který je nutné přenášet. Další výhodou je možnost naráz přenést údaje nacházející se na více místech a uspořádat je do vhodné struktury. Tyto objekty slouží k serializaci a deserializaci a neměly by obsahovat žádnou logiku. [88, 89]

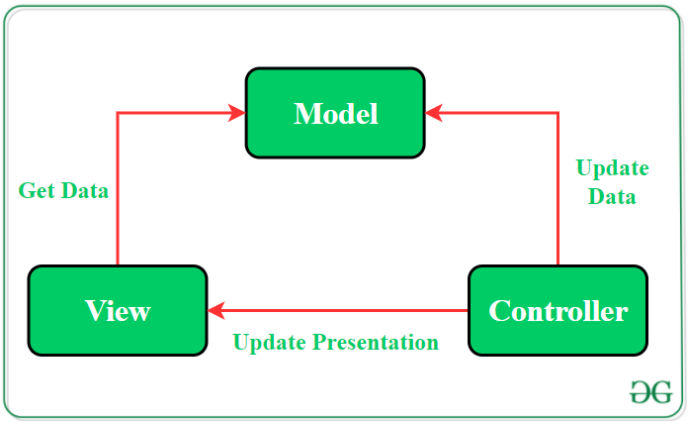
3.6.5 Observer

Jedná se o návrhový vzor využívaný v situacích, kdy je potřeba informovat více objektů (nebo klientů, když aplikace běží na více zařízeních) o změně stavu. K realizaci se využívá kolekce odběratelů (subscribers), kam se mohou dynamicky přidávat a odebírat. Když nastane daná událost, objekt (nebo server) rozešle zprávu všem v této kolekci. Některé jazyky mají tento mechanismus integrovaný přímo do sebe (např. Event v C#). [90]

3.6.6 MVVM

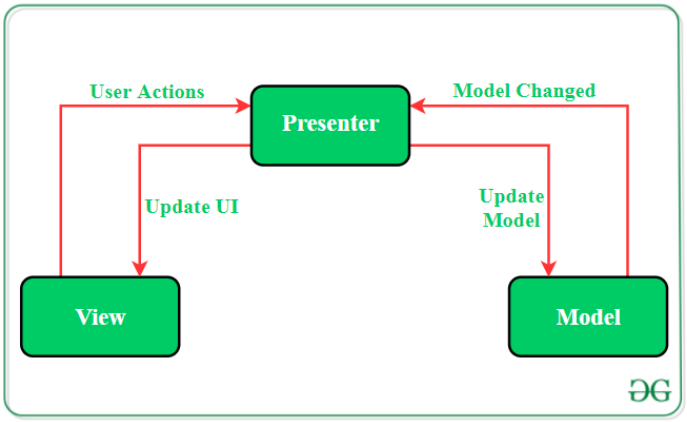
Pro jednodušší vývoj a testování uživatelských rozhraní se využívají návrhové vzory MVC, MVP a MVVM. Všechny tři od sebe oddělují data, vzhled a logiku, čímž usnadňují udržení struktury a umožňují modulárnost aplikace. Liší se v datových tocích a závislostech jedné části na ostatních. [91]

Nejstarším z těchto návrhových vzorů je MVC (Model-View-Controller). Model obsahuje aplikační data a je zodpovědný za komunikaci s databází, serverem, či jinou externí částí aplikace. View má na starosti zobrazování dat z modelu uživateli. Controller reaguje na uživatelské akce a dává modelu a view pokyny k aktualizaci. Jak je vidět na Obr. 27 jednotlivé části jsou úzce provázány, což komplikuje testovatelnost a úpravy. [91, 92]



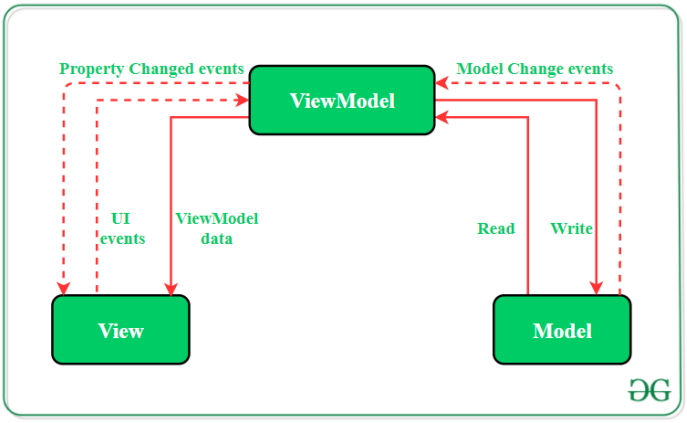
Obr. 27 Datový tok MVC [91]

Většinu problémů MVC řeší MVP (Model-View-Presenter), kde view a model nekomunikují napřímo, ale přes presenter jako prostředníka (viz Obr. 28). Oproti MVC zde na uživatelské akce reaguje view, které informaci předává presenteru. Ten při vracení aktualizovaných dat z modelu může provést další zpracování. Díky většímu oddělení jednotlivých částí usnadňuje testování a úpravy. [91, 92]



Obr. 28 Datový tok MVP [91]

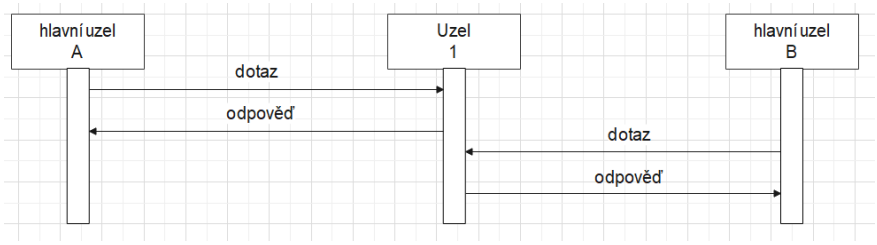
Vzor MVVM (Model-View-ViewModel) je podobný MVP, ale view neobsahuje žádnou logiku a pouze vykresluje data, která dostane z ViewModelu. Svůj obsah aktualizuje na základě eventu OnPropertyChanged (viz Obr. 29). Většina logiky se nachází ve ViewModelu, který má také na starosti stav aplikace. Tento přístup umožňuje, aby více view bylo navázáno na jeden ViewModel. Oproti svým předchůdcům je MVVM modulárnější, testovatelnější a snáze škálovatelný. Avšak za cenu vyšší komplexity tříd. [91, 92]



Obr. 29 Datový tok MVVM [91]

4 Vlastní řešení

Praktickou částí této práce je návrh a realizace řešení, které by umožnilo uživatelům s minimální či žádnou znalostí programovacích jazyků vytvořit automatizovanou úlohu. Řešení je navrženo tak, že existuje jeden či více hlavních uzlů, které vykonávají zadanou úlohu. Jako vstupy slouží data z uzlů se senzory. Jeden uzel může být součástí více úloh, tudíž iniciátorem komunikace je hlavní uzel (Obr. 30).



Obr. 30 Sekvenční diagram: obecná komunikace s více hlavními uzly

Při realizaci je využívána abstrakce a není spoléháno na funkce specifické pro použité technologie, díky čemuž je snazší případná migrace. Dále je kladen důraz na modularitu, aby v případě, že aktuální řešení nevyhovuje potřebám konkrétní úlohy, bylo snadné danou část jednoduše nahradit bez ovlivnění zbytku systému.

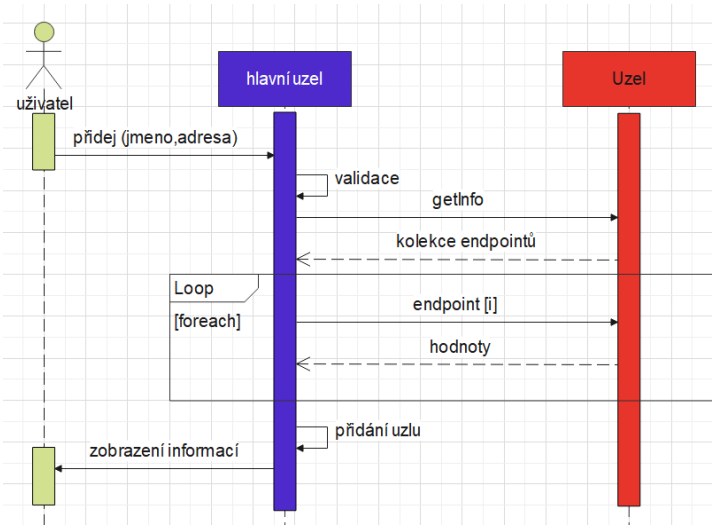
Pro komunikaci byl zvolen protokol HTTP (viz Kap. 3.2.4) s obsahem ve formátu JSON (viz Kap. 3.3.2). Jelikož pro HTTP existuje mnoho nástrojů, je jednoduché otestovat funkčnost uzlu. Dále je možné využití uzlů i mimo tento projekt. Z těchto důvodů byl upřednostněn textový formát před bitovým (viz Kap. 3.3), který vyžaduje funkční program schopný reprezentovat přijatá data. Z běžně využívaných textových formátů CSV vyžaduje tabulku, kde každý záznam musí mít stejný počet sloupců. Jelikož tato podmínka nebude projektem splněna, není pro tuto práci vhodný. XML musí mít pro každou hodnotu otevírací a ukončovací značku, čímž zvětšuje požadavky na množství přenesených dat. Z tohoto důvodu je stejně jako ostatní značkovací jazyky vhodný pro složité struktury psané programátorem (například vzhled uživatelské rozhraní). Pro účely této práce je vhodnější JSON, který má menší datovou stopu a mnoho jazyků má již existující systémové knihovny pro jeho serializaci a deserializaci. Případně je možné použít nástroj třetí strany.

Z důvodů jako jsou například množství přenášených dat a princip fungování sensorů, mohou mít uzly rozdílný počet endpointů. Proto všechny uzly mají endpoint *getInfo*, který vrátí kolekci s informacemi o dostupných endpointech (Obr. 31). Ty obsahují údaje jako

jsou URL, HTTP metoda, zda slouží k načtení hodnot, nebo k jejich nastavení, jaké hodnoty vrací a zda očekává argumenty. V případě pomalých získávání hodnot také může obsahovat údaj o očekávaném zpoždění, aby hlavní uzel věděl, kdy ještě probíhá zpracování a kdy již uběhl časový limit znamenající problém se spojením. U vracených hodnot sdělí jejich název a datový typ. Argumenty navíc obsahují výchozí hodnotu a limity v jakých se může hodnota pohybovat. Proto je tento endpoint volán, když uživatel přidává nový uzel do systému, aby při zadávání logiky věděl, s jakými hodnotami může pracovat. Pro ověření funkčnosti hlavní uzel zkusí, zda je možné všechny zavolat. Tento postup sice brání přidání uzlů bez funkčního spojení, ale odchytí případné problémy již na začátku. Z kontroly jsou vynechány endpointy označené jako výstupní, aby změna hodnoty neměla nežádoucí účinky na systém. Tento postup je znázorněn na Obr. 32 pomocí sekvenčního diagramu.


```
[
  {
    "HTTP": "GET",
    "Type": "EP_TYPE_GET",
    "URL": "/slow",
    "Vals": [
      {
        "Name": "a",
        "Type": "INT"
      }
    ],
    "Args": [],
    "Delay": 1000
  }
]
```


Obr. 31 Příklad odpovědi getInfo



Obr. 32 Sekvenční diagram: přidání uzlu

4.1 Hlavní uzel

Hlavní uzel je realizován jako počítačový program. Řešení je rozděleno na tři části (viz Kap. 3.6.2), které řeší komunikační, logickou a uživatelskou vrstvu. Každá vrstva má referenci jen na vrstvu pod ní. Toto řešení umožňuje snadnou změnu jednotlivých částí s minimálními zásahy do kódu.

Pro realizaci byl zvolen C#, jakožto hlavní programovací jazyk .NET ekosystému, který umožňuje knihovny využít na většině současně využívaných operačních systémech. Jako verze byla zvolena .NET 8.0, protože v době psaní této práce se jednalo o jedinou LTS (Long Term Support). Dále také podporuje *KeyedServices* (viz Kap. 3.6.3) umožňující snáze implementovat více typů uzlů. Tato volba je na úkor podpory starších operačních systémů (např. Windows 7 a verze Windows 10 starší než 21H2) a verzí programů třetích stran, které vyšly před listopadem 2023 [93–95]

Pro paralelní úlohy nebyly explicitně vytvářena vlákna, ale byl použit princip úkolů. Ty jsou v C# reprezentovány pomocí třídy *Task*. Jejich hlavní výhodou je úspora systémových zdrojů díky využití *ThreadPool*, který slouží jako zásobník znovu použitelných vláken. Tímto je ušetřen čas, který by jinak byl nutný k alokaci nového vlákna. Dále jsou poskytnuty mechanismy k počkání na výsledek a předčasnému ukončení. [96–98]

Uživatelem zadaná logika je reprezentována jako kolekce generických objektů obsahujících operaci a instanci třídy reprezentujícího hodnotu uzlu. Jelikož se hodnota předává uvnitř objektu, je možné ji dynamicky měnit během vykonávací smyčky.

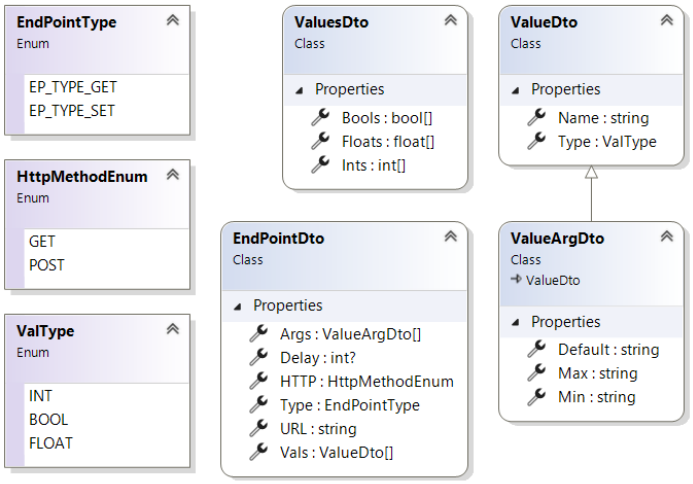
4.1.1 Komunikační vrstva

Úkolem této vrstvy je zajistit komunikaci s uzly a poskytnout abstrakci pro ostatní vrstvy, aby ke své funkci nemusely znát podrobnosti o způsobu komunikace. Tento přístup umožňuje v budoucnu nahradit HTTP jiným protokolem nebo využívat více protokolů či API třetích stran současně.

Vrstva je realizována jako knihovna tříd nazvaná *MainNode.Communication*. Většina tříd, které se zde nacházejí, jsou DTO (viz Kap. 3.6.4). Ty jsou v následující vrstvě, odkud byla metoda, jež je vrací volána, převedena na DO.

4.1.1.1 DTO třídy

Pro komunikaci s uzly slouží třídy plnící funkci DTO (Kap. 3.6.4), jenž se nacházejí v namespace *MainNode.Communication.Dto*, a enumy, které se nacházejí v namespace *MainNode.Communication.Enums*. Ty jsou zobrazeny pomocí diagramu tříd na Obr. 33.



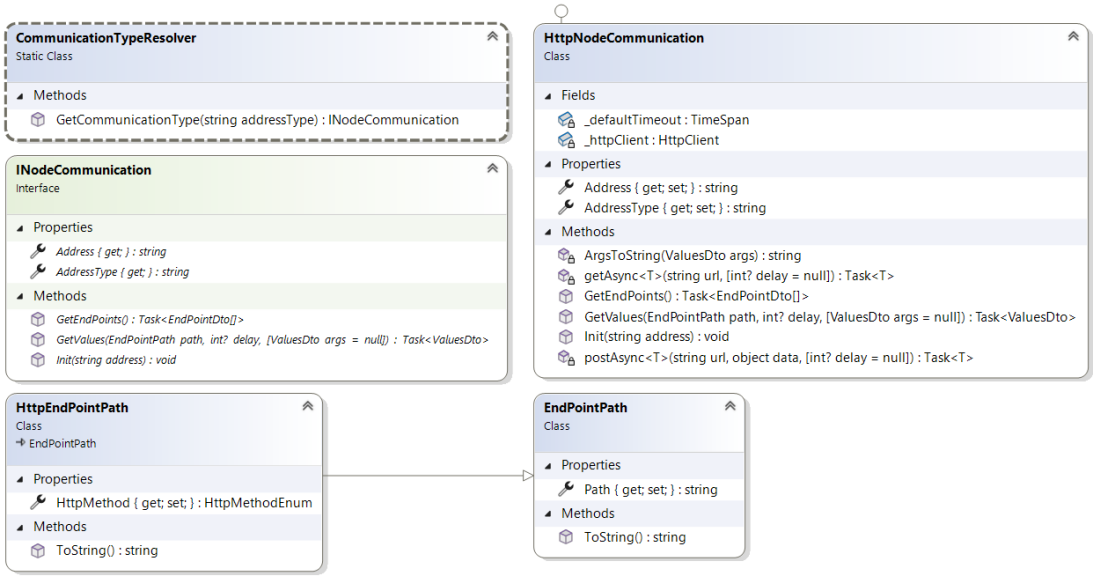
Obr. 33 Diagram tříd MainNode.Communication.Dto a MainNode.Communication.Enums

Třída *ValuesDto* slouží k přenosu hodnot mezi hlavním uzlem a ostatními uzly. Obsahuje pole pro hodnoty každého podporovaného datového typu. Tento přístup byl zvolen za účelem snížení množství přenášených dat při současném umožnění delších sebevysvětlujících názvů hodnot. Jelikož data v objektu reprezentující hodnotu na vyšších vrstvách musí být aktualizována, aniž by byl stávající objekt nahrazen, jsou možnosti buď hledat v kolekci podle jména, nebo spoléhat na jejich pořadí. Z důvodu rychlosti bylo upřednostněno pořadí před vyhledáváním, jelikož pravděpodobnost jeho změny je velice nízká. Jedinými případy jsou změna kódu uzlu po uložení informací o něm nebo kolize dvou enpointů se stejnou adresou.

Třída *ValueDto* obsahuje informace o jednotlivých hodnotách endpointu. Skládá se z názvu a enum *ValType*, jenž udává datový typ. Potomek *ValueArgDto* kromě těchto dvou údajů má navíc výchozí hodnotu a horní a dolní limit. Pro abstrakci protokolu je cesta k endpointu reprezentována pomocí rodičovské třídy *EndPointPath* a jejího potomka *HttpEndPointPath*, jenž kromě textového řetězce s cestou obsahuje také HTTP metodu daného endpointu.

4.1.1.2 INodeCommunication

INodeCommunication je rozhraní pro třídu řešící komunikaci. Obsahuje adresu uzlu, se kterým bude komunikovat, a o jaký typ adresy se jedná, což je využito při deserializaci uloženého seznamu uzlů. Tato hodnota je nastavena instancí, jenž toto rozhraní implementuje. Metoda *Init* slouží k inicializaci objektů potřebných k samotné komunikaci. Tato logika nemohla být umístěna do konstruktoru, protože kvůli využití dependency injection (viz Kap. 3.6.3) je instance v některých situacích vytvořena dříve, než je známa adresa. Asynchronní Metoda *GetEndPoints* slouží k zjištění seznamu dostupných endpointů (první zpráva mezi hlavním uzlem a uzlem na Obr. 32) a vrací kolekci *EndPointDto*, která může být *null*. Jak již bylo řečeno, jedná se o surová data, která jsou zpracována až na vyšší vrstvě. Asynchronní metoda *GetValues* slouží ke komunikaci s uzly. Návratovou hodnotou je *nullable* objekt typu *ValuesDto*. Povinným parametrem je instance třídy *EndPointPath*, či jejich potomků. Dále má nepovinné celé číslo *delay* a instanci *ValuesDto*, jenž jsou obsaženy v objektu, který reprezentuje endpoint ve vyšší vrstvě a byl vytvořen na základě hodnot načtených pomocí *GetEndPoints*.



Obr. 34 Diagram tříd INodeCommunication

HttpNodeCommunication je implementací (viz Obr. 34) právě popsaného rozhraní, které slouží ke komunikaci pomocí HTTP protokolu a dříve popsaných DTO objektů. K tomu využívá instanci třídy *System.Net.Http.HttpClient*, jež je vytvořena v metodě *Init* s hodnotou *Timeout* nastavenou na deset hodin, aby bylo zajištěno, že bude delší

nejpomalejší endpoint. Bylo vycházeno z předpokladu, že uzly budou posílat hodnoty v intervalech, které se pohybují v řádek minut, až milisekund.

Implementace *GetValues* rozlišuje, zda volaný endpoint využívá *GET* nebo *POST* a zavolá podle toho pomocnou metodu. Jedná-li se o *GET*, přidá argumenty do URL adresy. V případě *POST* provede serializaci. V obou případech je před odesláním HTTP dotazu vytvořena instance třídy *System.Threading.CancellationTokenSource*, jenž v případě, že metoda nestihne nedoběhnout dříve, než uplyne stanovený časový limitu, vyhodí *System.Threading.Tasks.TaskCanceledException*. Ta je zachycena a volání je vyhodnoceno jako chyba spojení. K nastavení časového limitu je využit parametr *delay*. Pokud pro daný endpoint není definován, je použita výchozí hodnota 1 s. Tento postup je zvolen, protože *Timeout* může být nastaven pouze jednou, takže by bylo nutné vytvořit samostatného klienta pro každý endpoint. Pokud je volání úspěšné, je odpověď deserializována a vrácena.

4.1.1.3 CommunicationTypeResolver

Jedná se o pomocnou statickou třídu sloužící k vytvoření instance správné implementace rozhraní *INodeCommunication*. K tomu je využívána metoda *GetCommunicationType*, jenž má jako parametr textový řetězec, na jehož základě je rozhodnuto, jakou instanci je třeba vytvořit.

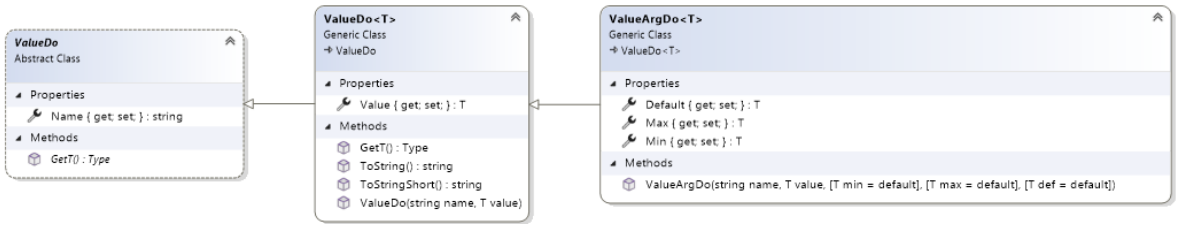
4.1.2 Logická vrstva

Logická vrstva je jádrem této práce, neboť zde probíhá zpracování uživatelem zadané logiky. Ta je zadávána v podobě textu, aby byl proces stejný bez ohledu na to, zda je vyšší vrstva uživatelské rozhraní, nebo ASP.NET aplikace sloužící jako API komunikující s jinou aplikací. Tím je minimalizováno riziko, že by implementace vytvářela neočekávané stavy, protože pokud programátor, který vytváří vyšší vrstvu, neobejde třídu na zpracování vstupu, budou neočekávané operace zachyceny. Je-li uživatelský vstup bez chyb, je vytvořena kolekce datových toků tvořených operacemi obsahujícími objekty reprezentující data z uzlů. Když uživatel spustí smyčku, jsou na začátku každé iterace aktualizovány hodnoty využitých endpoitů. Po jejich aktualizaci je provedeno vyhodnocení datových toků. Na konci smyčky jsou výsledky odeslány na příslušné endpointy. Po uplynutí nastaveného času je spuštěna další iterace smyčky.

Vrstva je realizována jako knihovna tříd nazvaná *MainNode.Logic*. Pro využití ve vyšší vrstvě má definována rozhraní, aby se snížila šance nesprávné manipulace. V této vrstvě jsou využívány repositáře, aby byla data na jednom místě a předešlo se tak více instancím, které mají reprezentovat stejnou informaci, ale mají odlišné hodnoty.

4.1.2.1 DO třídy

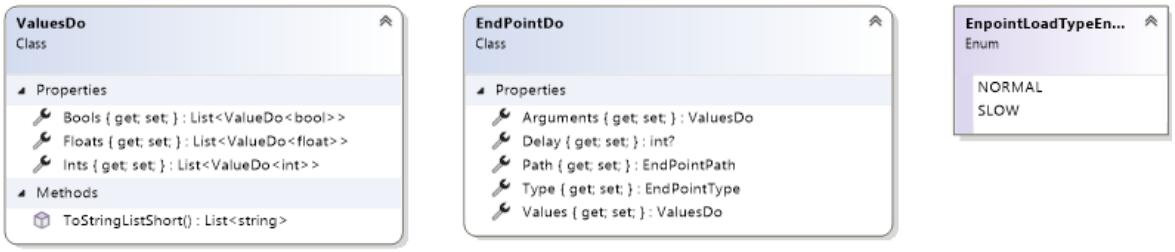
DO na této vrstvě jsou reprezentací DTO z komunikační vrstvy (viz 4.1.1.1), které využívají generiku, aby mohli obsahovat všechny podporované datové typy a poskytnou ostatním objektům přímo danou hodnotou. K převodu DTO na DO slouží statická třída *Mapper* obsahující metody, které se shodně jmenují *Map* a pro DTO parametr vrací jeho ekvivalentní instanci DO. Tento přístup minimalizuje závislost logické vrstvy na komunikační a v případě změny je třeba upravit pouze třídy *Mapper* a *Node*, který s touto vrstvou komunikuje (viz Kap. 4.1.2.3).



Obr. 35 Diagram tříd ValueDo a potomci

Základem je abstraktní třída *ValueDo* (viz Obr. 35) a její stejnojmenný generický potomek reprezentující hodnotu uzlu. Abstraktní rodič je potřebný z důvodu, aby bylo možné tuto třídu použít jako parametr metod a vlastnost ostatních tříd i v situacích, kdy v době kompilace není možné určit datový typ hodnoty. Abstraktní rodič obsahuje pouze jméno *Name* a deklaraci metody *GetT*, která vrací generický datový typ. Potomek obsahuje definici oné metody a generickou vlastnost *Value*, obsahující již zmíněnou hodnotu. Pro účely výpisu a debugu obsahuje přetíženou metodu *ToString* vypisující celé jméno generického typu společně s jménem a hodnotou. Dále obsahuje zkrácenou verzi *ToStringShort* vypisující pouze krátký název typu a jméno. Od generické *ValueDo* dědí

ValueArgDo, která přidává pouze vlastnosti *Default*, *Min* a *Max*, jenž jsou stejného datové typu jako *Value*.



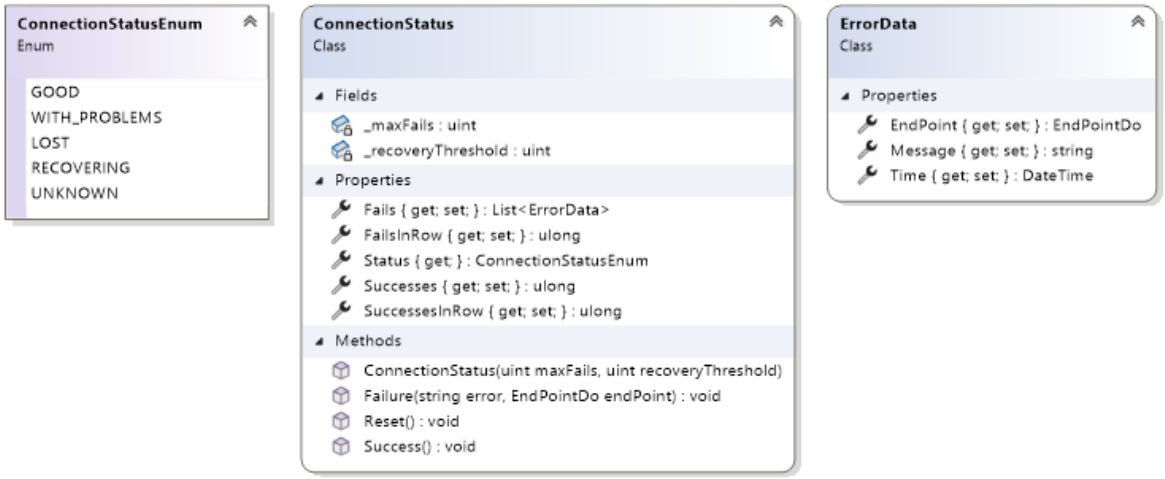
Obr. 36 Diagram tříd EndPointDo

Třída *ValuesDo* (Obr. 36) obsahuje informaci o aktuálních hodnotách endpointu. K tomu jí slouží listy *ValueDo<int>*, *ValueDo<float>* a *ValueDo<bool>*. Při načtení nových dat je aktualizována hodnota *Value* všech elementů v těchto kolekcích. Pro účely výpisu se zde nachází metoda *ToStringListShort*, jež zavolá *ToStringShort* nad každým elementem ve výše zmíněných kolekcích a jejich výsledky spojí do listu textových řetězců.

Třída *EndPointDo* (Obr. 36) uchovává informace o jednotlivých endpointech uzlu. Cesta potřebná k jeho zavolání je uložena ve vlastnosti *Path* typu *EndPointPath*, která slouží jako abstrakce komunikační vrstvy (viz Kap. 4.1.1.1). Hodnoty a argumenty jsou reprezentovány instancemi třídy *ValuesDo*. Vlastnosti *Type* a *Delay* jsou identické jako ve třídě *EndPointDto*.

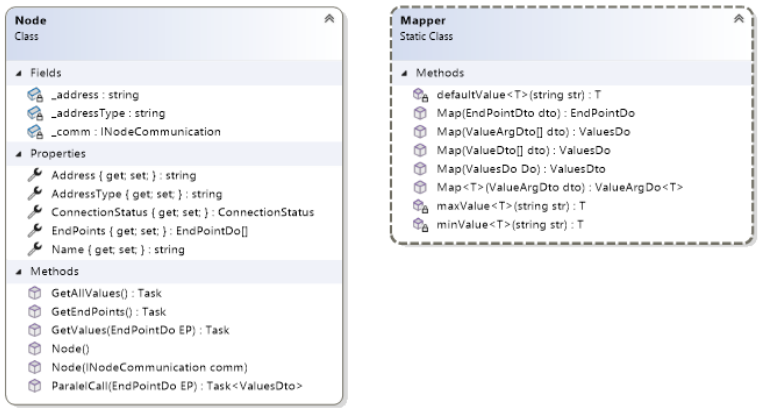
4.1.2.2 ConnectionStatus

Třída *ConnectionStatus* (viz Obr. 37) slouží k sledování stavu spojení s uzlem. K jeho reprezentaci slouží *ConnectionStatusEnum* jenž může nabývat hodnot *GOOD*, *WITH_PROBLEMS*, *LOST*, *RECOVERING* a *UNKNOWN*. Při vytváření instance pomocí konstruktoru jsou nastaveny bezznaménkové celočíselné hodnoty *maxFails* a *recoveryThreshold*. První je počet neúspěšných pokusů v řadě, kdy se ze spojení s problémy stává ztracené spojení. Druhý je kolikrát v řadě musí být volání úspěšné, aby se ze zotavujícího spojení stalo opět dobré. V případě, že neexistují žádané informace o těchto hodnotách, je stav neznámý, což nastává jen v případě načtení uloženého uzlu. Dojde-li k chybnému spojení, je do listu typu *ErrorData* přidán nový objekt, jenž obsahuje informaci o času, volaném endpointu a z jakého důvodu k chybě došlo.



Obr. 37 Diagram tříd ConnectionStatus

4.1.2.3 Node



Obr. 38 Diagram tříd Node

Třída Node (viz Obr. 38) obsahuje veškeré informace o uzlu a slouží jako prostředník mezi logickou a komunikační vrstvou. Toho je dosaženo pomocí private instance rozhraní *INodeCommunication* (viz Kap. 4.1.1.2). Vlastnost *Name* slouží k identifikaci při zadávání logiky a pokud ho uživatel nezmění, jeho výchozí hodnotou je node a pořadové číslo. Aby bylo možné odesílat dotazy, je třeba adresa uložená v proměnné *_address* k niž je přistupováno pomocí vlastnosti *Address*. Při čtení pouze vrátí hodnotu, ale při zápisu současně zavolá metodu Init z komunikačního objektu. Adresa je textový řetězec, protože se jedná datový typ využívaný většinou tříd pro komunikaci. Ať už se jedná síťovou komunikaci s IP adresou a portem, sérovou linku s číslem portu, volání jiného procesu,

databázi s ConnectionStringem nebo další. S adresou souvisí *AddressType*, jenž je také kombinací proměnné a vlastnosti. Ten je využíván při deserializaci aby bylo možné zjistit, která implementace *INodeCommunication* je potřeba vytvořit. Seznam všech endpointů dostupný pro uzel je uložen v kolekci *EndPoints*. Pro reprezentaci stavu spojení slouží instance třídy *ConnectionStatus*, která je vyřazena ze serializace, neboť nemá význam ukládat aktuální stav spojení, neboť při příštím spuštění bude situace zcela odlišná.

Třída Node má dva konstruktory. První má parametr *INodeCommunication*, využívající dependency injection (viz Kap. 3.6.3), volaný při přidávání nového uzlu, kdy je na základě typu implementace nastavena hodnota *AddressType*. Druhý je bezparametrický a je využívaný při deserializaci, kdy není možné volat konstruktor s parametry.

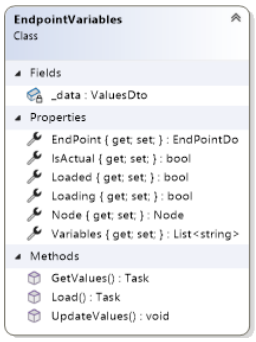
Asynchronní metoda *GetEndPoints* slouží k naplnění kolekce endpointů pomocí zavolání stejnojmenné metody v implementaci *INodeCommunication* a s využitím statické metody Mapper.Map přemapováním vrácených DTO, využívaných v komunikační vrstvě na DO, která jsou používány v logické vrstvě.

Asynchronní metoda *GetValues*, jejíž parametr je právě volaný endpoint, slouží k aktualizaci hodnot. Na začátku je zavolána *INodeCommunication.GetValues* vracející *ValuesDto* obsahující kolekce hodnot rozdělené podle datových typů. Pro každý list hodnot v *EndPointDo* je zavolána rozšiřující metoda *CopyValues*, která nahradí původní hodnoty novými. Tento postup je použit, neboť kvůli zachování referencí nemůže být objekt nahrazen novým. Pokud vše proběhne v pořádku, je v objektu *ConnectionStatus* pomocí metody *Success* komunikace označena za úspěšnou. V opačném případě je zavoláno *Failure*, čímž je aktualizován počet neúspěšných spojení a současně je zaevidován důvod proč se nezdařila. GetValues je využívána jak pro vstupní, tak výstupní endpointy.

Asynchronní metoda *GetAllValues* slouží k ověření dostupnosti všech vstupních endpointů. Toho je dosaženo tak, že ze seznamu vybere ty, jejichž *Type* má hodnotu *EndPointType.GET*. Ty jsou následně pomocí *GetValues* zavolány.

Pro volání pomalých endpointů, které mají nenulovou vlastnost *Delay*, slouží asynchronní metoda *ParalelCall*. Stejně jako jeho obdoba pro běžné volání využívá *INodeCommunication.GetValues*, ale neprovádí aktualizaci hodnot, aby se předešlo situaci, kdy by část výpočtů iterace proběhla se starými a část již s novými hodnotami.

4.1.2.4 EndpointVariables



Obr. 39 Diagram tříd EdpointVariables

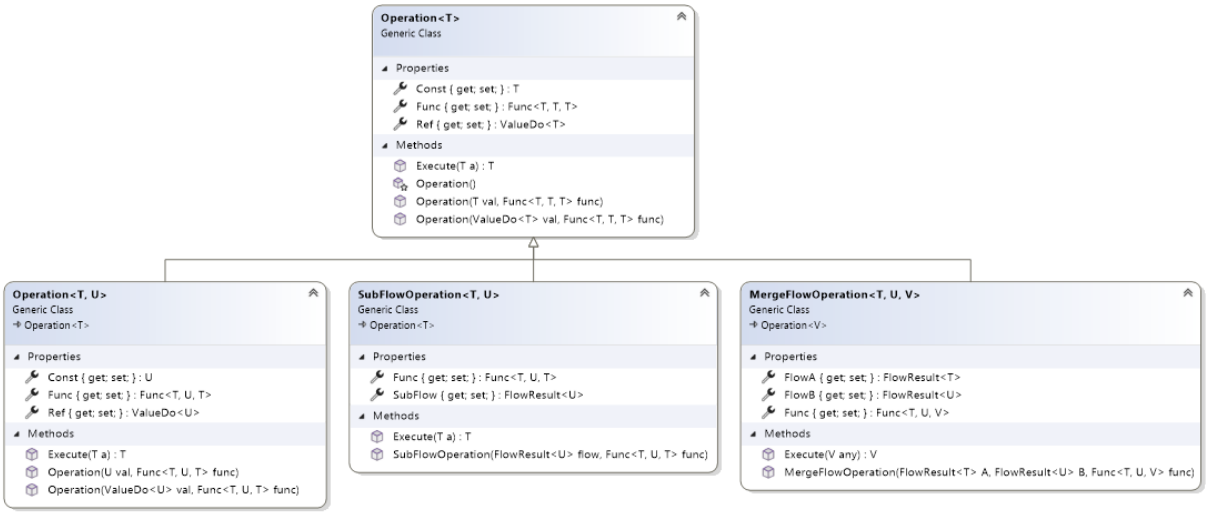
Účelem této třídy (Obr. 39) je sledovat aktualitu hodnot a sledovat stav načítání v případě endpointů, na jejichž odpověď se čeká jednu či více iterací vyhodnocovací smyčky. Dále je takto nevolat endpointy jenž jsou v uzlu sice dostupné, ale k řešení požadované úlohy jsou nepotřebné a jejich volání by pouze prodloužilo načítací proces, zvýšilo množství dat, které je potřeba přenést a uzel by musel vynaložit energii na zpracování dotazu, což je problematické obzvláště v případě bateriového provozu.

Aby bylo vůbec možné hodnoty aktualizovat, jsou potřebné reference na instance Node a EndPointDo. Pro sdělení informace o aktuálnosti hodnot slouží booleovská vlastnost *IsActual*, která je před zahájením načítání pomalého volání nastavena na false. Teprve poté co jsou hodnoty aktualizovány je nastavena na true. Podobnou funkci má *Loaded*, avšak zde je hodnota změněna již v okamžiku dokončení načítání, k čemuž dojde ještě během iterace vyhodnocovací smyčky. Opačný význam má *Loading* znamenající, že načítání teprve probíhá. Tyto dvě hodnoty jsou na sobě nezávislé, protože kdyby byla použita pouze jedna nebo by se jednalo o inverzi, tak by v okamžiku po spuštění smyčky mohlo docházet k nečekaným stavům, jelikož by nebylo možné dosáhnout stavu, kdy hodnota není načítána a současně není dokončeno získávání nové.

Pro načítání hodnoty pomalých endpointů je využívána asynchronní metoda *Load* volající *Node.ParalelCall*. V momentě, kdy je načítání dokončeno, ale čeká se do konec aktuální iterace vyhodnocovací smyčky, jsou nové hodnoty uloženy v proměnné typu *ValuesDto*, odkud budou během příští fáze načítání zkopírovány stejně jako je tomu bezprostředně po načtení u klasických endpointů využívající metodu Node.*GetValues* volanou stejnojmennou metodou této třídy.

4.1.2.5 Třídy datového toku

Aby bylo možno dynamicky měnit vyhodnocovanou logiku na základě uživatelského zadání, je nutné vytvořit objekty obsahující jak požadované operace, tak hodnoty, se kterými se mají provádět. Tuto funkci plní generická třída *Operation* a její potomci (viz Obr. 40). Pro reprezentaci prováděné operace je využíván delegát *System.Func*, jenž má dva parametry. Hodnota pro výpočet je uložena buď ve vlastnosti *Const* nebo *Ref*, v závislosti na tom, zda se jedná o konstantu nebo referenci typu *ValueDo*. Obě tyto hodnoty jsou nullable a nastavovány pomocí konstruktoru. Výsledek je počítán pomocí generické metody *Execute*, jenž jako parametr přijímá hodnotu nalevo od znaménka (Obr. 41).



Obr. 40 Diagram tříd Operation

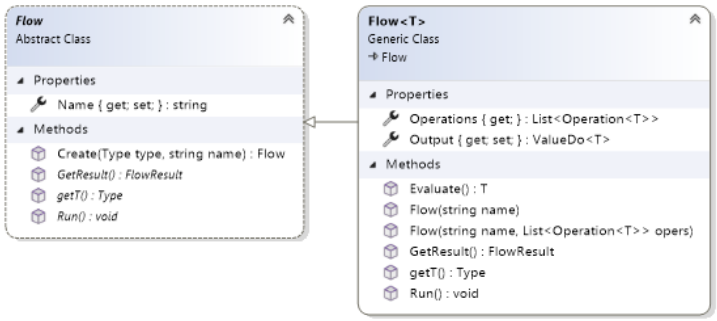
```
public virtual T Execute(T a)
{
    if (Ref != null) { return Func(a, Ref.Value); }
    return Func(a, Const!);
}
```

Obr. 41 Kód metody Execute

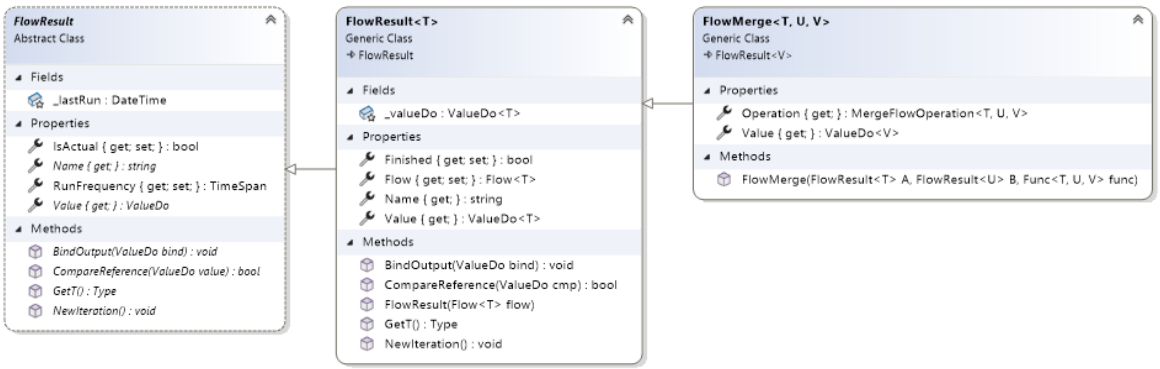
Základní třída očekává, že trojice výsledek, levá a pravá hodnota jsou stejného datového typu. V závislosti na tom, která z vlastností je null dosadí hodnoty do delegáta a vrátí výsledek. Její stejnojmenný potomek má stejné chování, ale pravá hodnota od znaménka je jiného datové typu. Aby nedocházelo k nejasnostem, zda využít vlastnost rodiče nebo potomka, je využita schopnost jazyka C# přidáním klíčového slova *new* v definici překrýt stejnojmennou vlastnost rodiče.

K vytvoření datového toku hodnot slouží instance generické třídy *Flow* (Obr. 42) obsahující list operací. Aby bylo možné uložit všechny datové toky do jedné kolekce, je

nutné vytvořit abstraktního negenerického předka. Pro umožnění vytvoření instance na základě proměnné typu *System.Type* je v této třídě vytvořena statická metoda, jež na základě na této hodnoty vrátí generickou instanci. Každá instance *Flow* má jméno ve vlastnosti *Name*, aby bylo možné se na ní odkazovat při zadávání logiky. K vyhodnocení celého datového toku slouží generická metoda *Evaluate*. Ta na začátku vytvoří výchozí hodnotu datového typu tohoto toku, který slouží jako levá hodnota první operace, a uloží ji do proměnné pro výsledek. Poté projde celou kolekci a pro každý prvek zavolá metodu *Execute* s aktuálním výsledkem jako levou hodnotou. Po projetí celé kolekce vrátí hodnotu, jež se aktuálně nachází v proměnné s výsledkem. Toto řešení sebou nese nutnost používání závorek pro určení pořadí operací. Pokud bude uživatel potřebovat tuto knihovnu implementovánu tak, že bude potřeba získat pouze hodnotu, je možné využít právě popsanou metodu *Evaluate*, ale je počítáno s variantou, kdy jeden datový tok je součástí dalšího a k tomu slouží metoda *Run*. Ta takto vypočtený výsledek vloží do *ValueDo* vlastnosti *Output*, kterou je možné použít jako referenci. Aby bylo možné sledovat, zda v této iteraci vyhodnocovací smyčky již byl výsledek datového toku vypočítán, je využívána třída *FlowResult* (Obr. 43), jejíž instance je vytvářena pomocí metody *GetResult*.



Obr. 42 Diagram tříd Flow



Obr. 43 Diagram tříd FlowResult

Stejně jako v případě *Flow* je *FlowResult* generická třída s negenerickou abstraktní rodičovskou třídou. Booleovská hodnota Finished vyjadřuje, zda v této iteraci již bylo provedeno vyhodnocení datového toku. Jeho výsledek je uložen v proměnné. Pokud při zavolaní konstruktoru *Flow*, pro který je instance vytvářena, nemá nastavenou instanci *ValueDo*, kam bude vkládat výsledek, je vytvořena nová, jež má stejné jméno jako datový tok, ale na konec je doplněno _out. V opačném případě je pouze uložena reference na tento objekt. K získání výsledku slouží vlastnost *Value*, která obsahuje pouze get. Pokud v této iteraci již došlo k vyhodnocení, je rovnou vrácen výsledek. V opačném případě je provedeno vyhodnocení. Poté jsou nastaveny hodnoty *Finished*, *IsActual* a *_lastRun*. Poslední dvě zmíněné jsou využity v případě, kdy je pomocí vlastnosti *RunFrequency* nastaveno, aby vyhodnocení bylo prováděno jednou za určitý časový úsek. V takovém případě IsActual slouží jako indikátor, že výsledek nepochází z této iterace. Na začátku nové iterace vyhodnocovací smyčky je zavolána metoda *NewIteration*, která na základě aktuálního času, *_lastRun* a *RunFrequency*, jejíž výchozí hodnota je 0 µs, nastavuje hodnoty *IsActual* a *Finished*. V případě, že datový tok je současně výstupem systému, je použita metoda *BindOutput*, jež nahradí vlastnost *Flow.Output* hodnotou endpointu.

Pro využití výsledku z datového toku v jiném, slouží třída *SubFlowOperation*, která je potomkem *Operation*. Její konstruktor přijímá *FlowResult* a *System.Func*, jehož levá hodnota a výsledek musí být stejného datového typu. Při výpočtu jako hodnotu na pravé straně operátoru využívá *FlowResult.Value*.

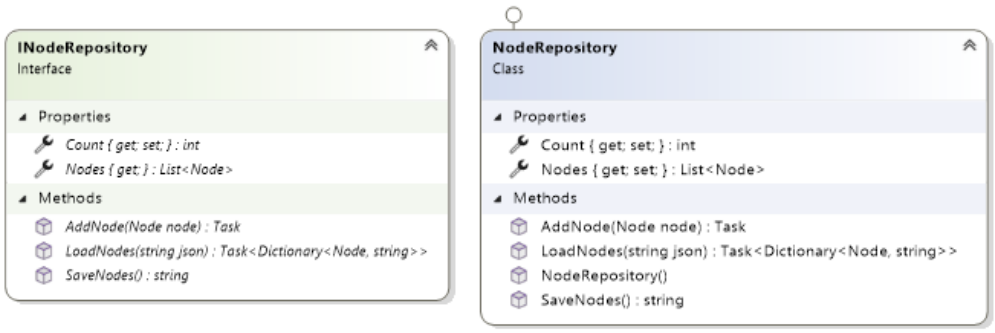
Speciálním případem je *MergeFlowOperation* sloužící k sloučení výsledků dvou datových toků. Oproti ostatním potomkům *Operation* výsledek, levá a pravá strana mohou být rozdílných datových typů. Ačkoliv metoda *Execute* má parametr, je zde pouze kvůli dědičnosti, ale k výpočtu není využit. Aby bylo možné tento výsledek použít pro výpočet, je potřeba současně použít instanci třídy *FlowMerge*, jenž je potomkem *FlowResult*. V konstruktoru je vytvořena na základě dvou *FlowResult* a *System.Func* vytvořena instance *MergeFlowOperation*. Vlastnost *Value* vypadá téměř totožně jako v rodičovské třídě, ale místo datového toku je zde vyhodnocována operace.

4.1.2.6 NodeRepository

Třída *NodeRepository* implementující rozhraní *INodeRepository* (viz Obr. 44), čímž je snížena závislost vyšší vrstvy na konkrétní implementaci, slouží jako globální úložiště

instancí třídy *Node*. Očekává se, že tato třída bude využívána jako singleton. Jádrem této třídy je kolekce *Nodes*, obsahující seznam všech uzlů připojených do systému. Pro přidání nového uzlu slouží asynchronní metoda *AddNode*, jejímž parametrem je přidávaný uzel. Než dojde k přidání, je ověřena nenulovost adresy a jména. Jelikož je jméno využíváno jako identifikátor při zadávání logiky, musí být unikátní. Pokud je některý z těchto požadavků nesplněn, je vyhozena výjimka *System.ArgumentException* s odpovídajícím chybovou zprávou. V opačném případě je zavolána metoda *GetEndPoints*, aby bylo možné získat seznam dostupných endpointů. Po jejím dokončení je pomocí *GetAllValues* ověřena jejich dostupnost (viz Kap. 4.1.2.3). V případě, že je seznam prázdný, je vyhozena výjimka *MainNode.Exceptions.NoEndPointException*, protože nemá smysl přidávat uzel, který nejde zavolat. Pouze pokud nedošlo k žádnému problému, je uzel přidán a změněna hodnota počítadla. Jelikož je počet uzlů použit jako výchozí název uzlu, jedná se o statickou hodnotu, jež není součástí instance. Toto je z důvodu předcházení cyklických referencí.

Pro uložení seznamu slouží metoda *SaveNodes*, vracející serializovanou kolekci uzlů. Třída neřeší ukládání sama, protože *MainNode.Logic* je knihovnou a může být implementována v různých typech aplikace včetně webových. Z tohoto důvodu je zde řešena pouze serializace a deserializace, ale práce s perzistentním úložištěm je přenecháno vyšší vrstvě. Obdobně metoda *LoadNodes* má jako parametr textový řetězec ve formátu JSON. Po deserializaci na *List<Node>* se pokusí pomocí metody *AddNode* tyto uzly přidat. V případě neúspěchu je přidá do *Dictionary<Node, string>*, který je návratovou hodnotou této metody. Klíčem záznamu je přidávaný uzel a hodnotou je chybová hláška, proč se přidání nezdařilo.

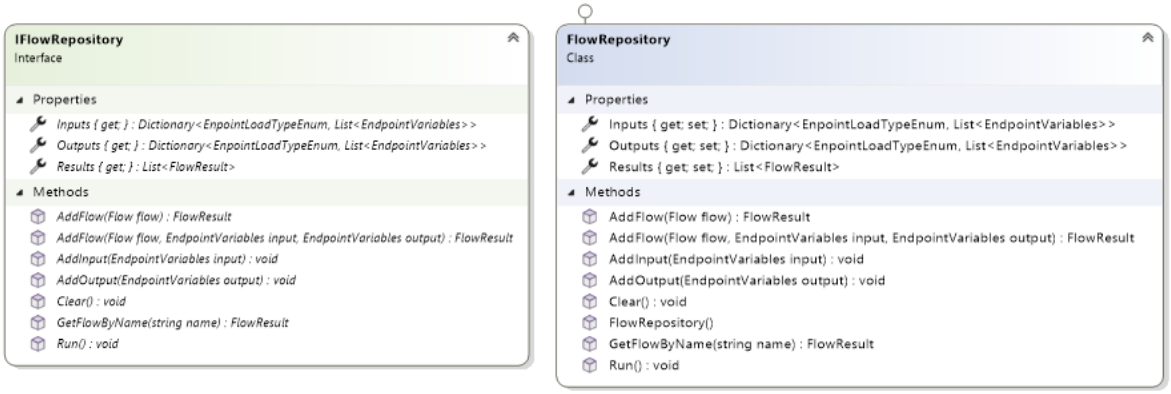


Obr. 44 Diagram tříd NodeRepository

4.1.2.7 FlowRepository

Třída *FlowRepository* implementující rozhraní *IFlowRepository* (viz Obr. 45) uchovává všechny datové toky, vstupy a výstupy potřebné k realizaci dané úlohy na jednom

místě. Jednotlivé datové toky jsou ve formě *FlowResult* (viz Kap. 4.1.2.5) uloženy v kolekci *Results*. Pro uložení vstupních a výstupných hodnot slouží kolekce Inputs a Outputs, které jsou typu *Dictionary<EnpointLoadTypeEnum, List<EndpointVariables>>*. Klíčem je enum určující, zda se jedná o klasický endpoint, nebo o pomalý, jehož odpověď může trvat několik iterací vyhodnocovací smyčky. Hodnotami jsou kolekce *EndpointVariables* (Kap. 4.1.2.4) řešící aktualizaci hodnot.



Obr. 45 Diagram tříd FlowRepository

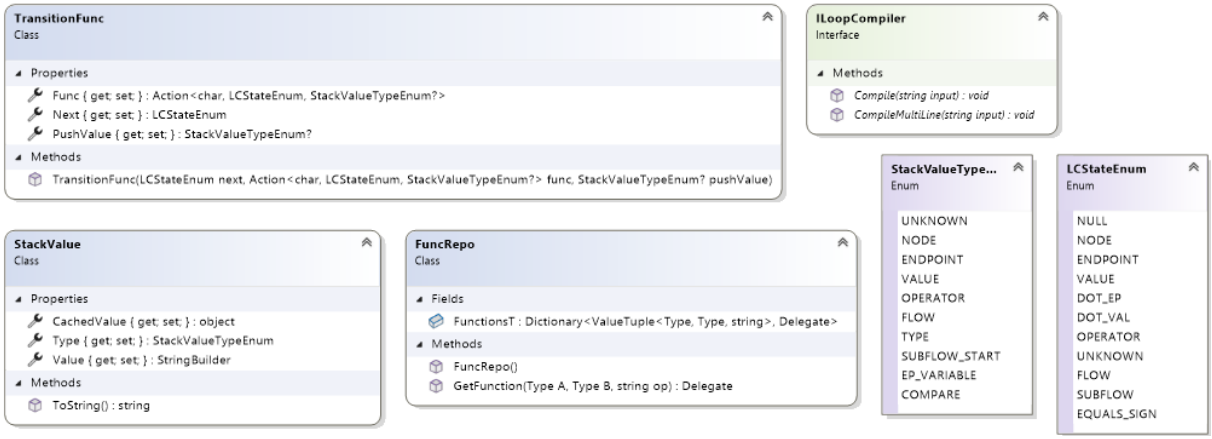
K přidání nového datového toku slouží metoda *AddFlow* s parametrem typu *Flow*, která pomocí *GetResult* vytvoří instanci *FlowResult* a přidá ji do kolekce. Aby bylo možné využít tuto hodnotu jako referenci pro další výpočet, je tato instance současně také vrácena. Pro přidání vstupů a výstupů slouží metody *AddInput* a *AddOutput*, které jsou téměř identické, ale pracují s jinými kolekcemi. Podle toho, zda je hodnota *Delay null* či nikoli, je instance přidána do seznamu normálních nebo pomalých volání.

Pro získání požadovaného datového toku je využívána metoda *GetFlowByName*, jejímž parametrem je jméno tohoto toku. Pokud nebyl nalezen žádný výsledek, nebo naopak je jich více, je vyhozena výjimka System.Exception s příslušnou chybovou hláškou.

Tato třída také řeší provedení vyhodnocovací části iterace smyčky. Slouží k tomu metoda *Run*, která nejprve na všech prvcích zavolá metodu *NewIteration*, čímž dojde k vynulování příznaků, zda v této iteraci již bylo provedeno vyhodnocení. Poté ze všech prvků načte hodnotu vlastnosti *Value*. Pokud ještě nebyl datový tok vyhodnocen, tak k tomu dojde. Byl-li vyhodnocen při získávání některého předcházejícího, je pouze vrácen výsledek.

4.1.2.8 Zpracování uživatelem zadané logiky

Pro zpracování uživatelem zadaného textového řetězce do podoby, kterou je možné vyhodnotit pomocí výše popsaných tříd, byl zvolen deterministický konečný automat. Kombinace stavů jsou reprezentovány pomocí matice. Jelikož při návrhu bylo počítáno s možností v budoucnosti převést toto řešení na jednočipový počítač, které oproti klasickému osobnímu počítači má výrazně méně paměti, byla tomu přizpůsobena struktura. Oproti běžně využívanému řešení, kdy každý znak abecedy automatu má vlastní sloupec (nebo řádek v závislosti na způsobu zápisu), což by v tomto případě, při využití pouze ASCII znaků, vyžadovalo dva tisíce osm set šestnáct možných kombinací, byly znaky se stejným významem seskupeny (např. všechna písmena) a nepotřebné zcela vynechány. Tím se počet možných přechodů snížil na sto čtyřicet tři. Ačkoli je toto řešení v době provádění náročnější na výkon, je ušetřena zbytečně zabraná paměť, což vhledem k tomu, že některá vývojová prostředí pro jednočipové počítače omezují velikost programu na 32 kB nebo méně, je výrazný rozdíl. Pro snížení počtu stavů je využíván zásobník. Na Obr. 46 je diagram tříd, které tento stavový automat a zásobník reprezentují.



Obr. 46 Diagram tříd stavový automat

Zásobník obsahuje objekty typu *StackValue* skládající se z dosud přečtených znaků, StackValueTypeEnum a cache. Enum určuje o jakou hodnotu se jedná (např. konstanta, jméno, uzlu, ...). Pro zaznamenání znaků je využit System.Text.StringBuilder, jenž je schopný dynamicky rozšiřovat svůj obsah bez zbytečného kopírování paměťových bloků, čímž je výrazně ušetřen čas procesoru. Cache je typu *nullable Object* a je využita v případě, že je z přečteného slova je možné vytvořit objekt, ale bez znalosti následujícího ho není

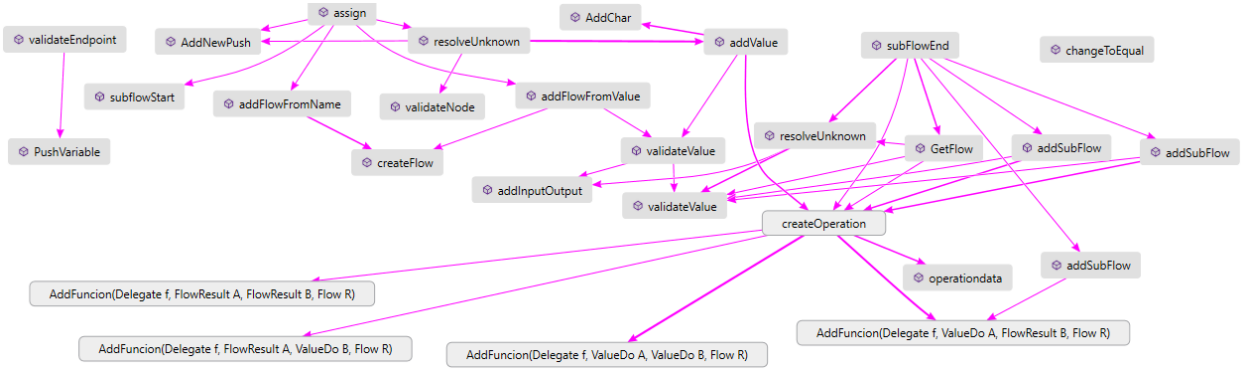
možné dále zpracovat. Takovým případem může být přečtení hodnoty ednpointu, ale neznalost operátoru a druhé hodnoty potřebné k vytvoření operace.

Matice reprezentující stavový automat je dvourozměrné pole *TransitionFunc,* jehož indexy jsou právě čtený znak převeden na číslo pomocí metody *getId* a enum *LCStateEnum* reprezentující stavy (viz Příloha 4 obsahující tabulku přechodů). Třída *TransitionFunc* reprezentuje přechodovou funkci a obsahuje následující stav, delegáta *System.Action<char, LCStateEnum, StackValueTypeEnum?>*, jenž bude proveden při přechodu do nového stavu a jaký typ záznamu bude přidán do zásobníku. Parametry delegáta jsou aktuální znak a stav společně s typem záznamu, jenž je součástí objektu reprezentující přechodovou funkci. Tato hodnota je potřebná, protože metody, na něž se delegáti odkazují, nejsou součástí této třídy.

Delegáti *System.Func* používaní v instancích *Operation* jsou bráni z *FuncRepo,* kde se nachází v *Dictionary<(Type, Type, string), Delegate>*, jenž je naplněn v konstruktoru. Klíčem je trojice datových typů obou hodnot a textové podoby operátoru. Třetí hodnotou je textový řetězec místo znaku, kvůli logickým operátorům, jenž jsou tvořeny dvěma znaky. Hodnota je definována pomocí lambda výrazů. K přistupování ke kolekci slouží metoda *GetFunction*, která v případě neexistujícího klíče vyhodí výjimku se zprávou obsahující informaci, která trojice nebyla nalezena.

Samotný převod logiky z textového řetězce na datové toky probíhá v instanci třídy *LoopCompiler* (viz Příloha 3), která je pro vyšší přehlednost rozdělena na více souborů, jenž mezi názvem třídy a koncovkou mají jakou část převodu řeší. V konstruktoru je zavolána metoda *InitTable*, která naplní tabulku přechodových funkcí. Pro zahájení převodu je zavolána metoda *Compile*, která zpracuje jeden datový tok. Ten může obsahovat další vnořené toky ohraničené závorkou, ale na ně se uživatel nebude moci odkazovat. Aby bylo možné zadat více datový toků, je nutné použít metodu *CompileMultiLine*, která textový řetězec rozdělí podle středníků a nových řádků. Dojde-li k chybě, vyhodí *System.ApplicationException* se zprávou obsahující index datového toku a text výjimky, kterou současně vloží jako *innerException*, aby bylo možné dohledat příčinu jejího vzniku. Na začátku kompilace je vymazán zásobník a na jeho vrchol je vložen inicializační operátor, neboť kvůli principu vyhodnocování (viz Kap. 4.1.2.5) musí být první operací ekvivalent *0+b*. Poté se projde pomocí smyčky for celý textový řetězec. For byl zvolen místo foreach proto, aby bylo možné v chybové hlášce přesně určit místo, kde nastal problém. Na základě indexu vráceného metodou *getId* a číselného vyjádření *LCStateEnum*, který popisuje

aktuální stav konečného automatu, je z tabulky získána přechodová funkce. V případě, že tato hodnota není zadána, je vyhozena *System.ApplicationException* oznamující uživateli, že v tomto místě není takovýto znak očekáván, a je poznačeno místo v řetězci kde se právě nachází. V opačném případě je provedena metoda, na niž ukazuje delegát a aktualizován stav konečného automatu. Po skončení smyčky je provedena funkce, jež se v tabulce nachází na souřadnici označenou prázdným znakem a aktuálním stavem. Na Obr. 47 je mapa kódu zobrazující, jak jsou vzájemně volány metody. Kvůli přehlednosti byli vynechány metody sloužící k přístupu do zásobníku. Dále byly metody *createOperation* umístěny do jedné skupiny. V nejvyšší řadě jsou metody, jejichž delegáti jsou v přechodových funkcích stavového automatu. Ve spodní části se nachází skupiny *AddFuncion*, které obsahují řetězec metod začínající s čistě abstraktními parametry, které jsou postupně nahrazovány generickými. Tento postup umožňuje zjistit datové typy obou hodnot a výsledku a na jejich základě vytvořit správnou instanci *Operation*.

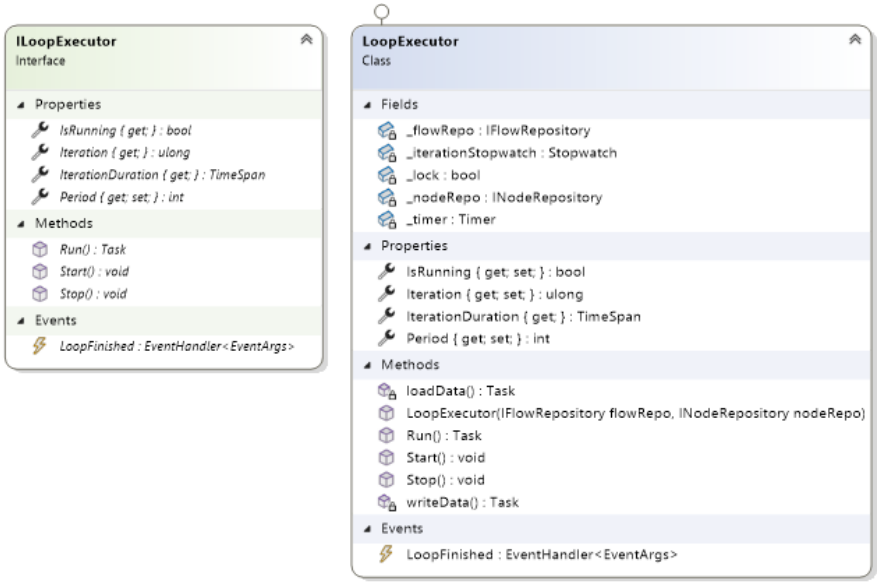


Obr. 47 Mapa kódu LoopCompiler

4.1.2.9 LoopExecutor

Instance třídy LoopExecutor (Obr. 48) je zodpovědná za provádění vyhodnocovací smyčky. Při zavolání metody *Start* je změněna hodnota vlastnosti *IsRunning* na *true* a vytvořena nová instance časovače *System.Threading.Timer*, jenž má nulové zpoždění a periodu nastavenou dle hodnoty vlastnosti *Period*, jejíž výchozí hodnota je 1000 ms. Delegát volaný po uplynutí periody je asynchronní metoda *Run*. Na začátku je ověřen zámek, zajišťující, aby nenastala situace, kdy je metoda spuštěna vícekrát. Před začátkem vyhodnocení je počkáno na provedení asynchronní metody *loadData* starající se o načtení aktuálních hodnot z uzlů. Vyhodnocení je provedeno pomocí metody *FIowRepository.Run* (viz Kap. 4.1.2.7). Poté jsou výsledky odeslány do příslušných uzlů pomocí metody

writeData. Po dokončení je vyvolána událost (viz Kap. 3.6.5) *LoopFinished*, na kterou se mohou napojit třídy z vyšší vrstvy a reagovat tak na dokončení smyčky. Nakonec je do debug terminálu zapsána informace o délce trvání a uvolněn zámek. Metody *loadData* a *writeData* jsou asynchronní, neboť v nich probíhá volání komunikačních metod, jež ze své podstaty musí být asynchronní, ale průběh metody *Run* je pozastaven, dokud nedoběhnou. Obě metody jsou téměř identické, ale pracují s jinou kolekcí. Normální endpointy jsou aktualizovány okamžitě. V případě pomalých je nejprve ověřeno, zda již uzel odpověděl. Pokud ano, proběhne aktualizace hodnot. Jestliže momentálně neprobíhá načítání nových hodnot, je zahájeno.



Obr. 48 Diagram tříd LoopExecutor

4.1.3 Uživatelské rozhraní

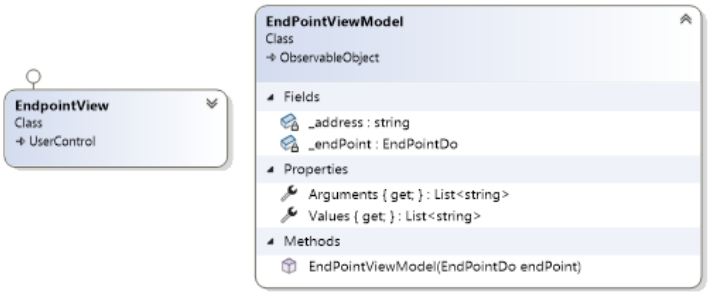
Pro nejvyšší vrstvu, se kterou interaguje uživatel, byla zvolena desktopová aplikace. Při zvažování, jaký typ aplikace zvolit, bylo myšleno na to, aby se dala jednoduše distribuovat a když uživatel změní velikost okna, tak se mu přizpůsobí. Z možností, jež jsou dostupné bez použití knihoven třetích stran, jsou na výběr WinForm (Windows Forms), WPF (Windows Presentation Foundation) a UWP (Universal Windows Platform). WinForm není responzivní a prvky jsou umístěny prostřednictvím absolutních souřadnic, tudíž při změně velikosti okna buď nejsou viditelné, nebo je část okna prázdné. UWP je aplikace určená pro Microsoft Store, kvůli čemuž je pro distribuci nepraktické. Zbývá tedy WPF, jenž je responzivní a distribuuje se jako klasický *exe* soubor. Při realizaci bylo využíváno MVVM (viz Kap. 3.6.6), kdy vzhled je definován pomocí XAML a data s logikou jsou přidány

pomocí provázání s ViewModely. Pro lepší čitelnost ViewModelů byl využit NuGet balíček *CommunityToolkit.Mvvm [99]*, který přidává atributy, díky nimž je logika aktualizace View při změně hodnoty vygenerována vývojovým prostředím do samostatných souborů mimo kód psaný programátorem.

4.1.3.1 Komponenty

Pro lepší čitelnost XAML definujících vzhled oken a možnost znovu použitelnosti na více oknech byly grafické prvky, které jsou tvořeny z více elementů nebo obsahují nějakou logiku, vytvořeny jako samostatné komponenty s vlastními ViewModely.

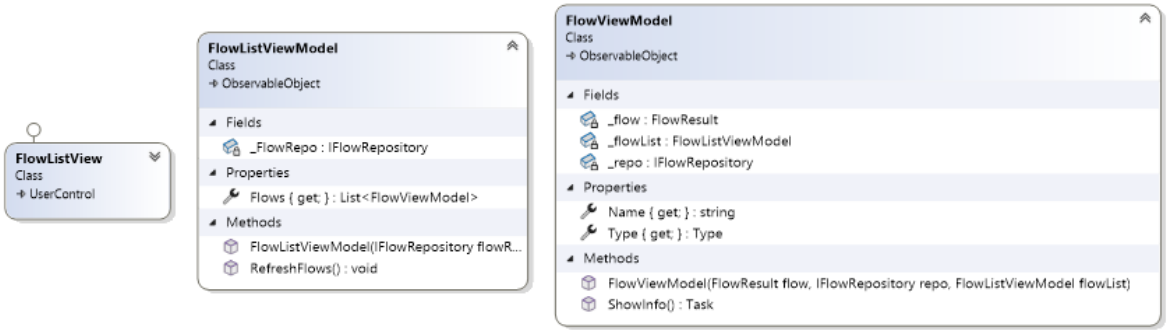
EndpointView (na prostředku Obr. 52) sloužící k zobrazení informací o endpointu je tvořen mřížkou s třemi řádky a dvěma sloupci, jež si rozdělí šířku na polovinu. První řádek o výšce 50 px obsahuje *Label*, jehož obsah je navázán na vlastnost *Address* nacházející se v *EndPointViewModel* (Obr. 49) a zabírá dva sloupce. Na druhém řádku se v obou sloupcích nachází kódem zadané popisy sloupců pro hodnoty a argumentu. Poslední řádek zabere veškeré dostupné místo. Zde se nachází *ListView* jejichž zdrojem dat jsou kolekce *Values* a *Arguments*. Ty jsou získány metodou *ValuesDo.ToStringListShort* zavolanou nad příslušnou kolekcí instance třídy *EndPointDo*. Jednotlivé hodnoty jsou zobrazeny pomocí *TextBlock*, která má menší rozměry a paměťovou stopu než *Label* [100, 101], zabaleném do *Border* se zaoblenými rohy.



Obr. 49 Diagram tříd EndPointViewModel

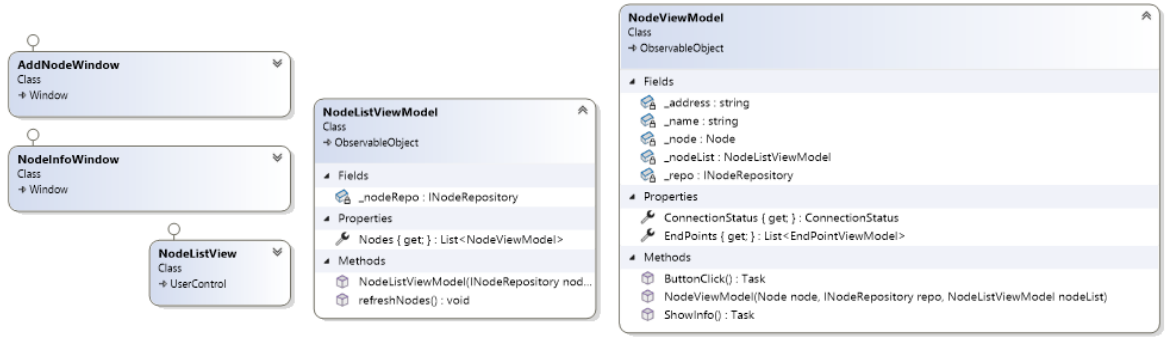
FlowListView (vlevo dole na Obr. 54) je určen k zobrazení seznamu datových toků. Ty jsou reprezentovány třídou *FlowViewModel*, jenž má jméno a datový typ výstupní hodnoty. Tyto objekty se nachází v kolekci *Flows* uvnitř singleton instance *FlowListViewModel* (viz Obr. 50). Ta je vytvářena ze seznamu nacházejícího se

v *FlowRepository* (viz Kap. 4.1.2.7) buď při prvním zavolání nebo jako reakce na použití metody *RefreshFlows*. K zobrazení je využíván *ListView* naplněný *Buttony*, jejichž barva textu a ohraničení je nastavena na základě datového typu a *TypeColorConverter*.



Obr. 50 Diagram tříd FlowViewModel

NodeListView (vlevo nahoře na Obr. 54 a vlevo uprostřed na Obr. 56) složí k zobrazení připojených uzlů a jejich stavu. Má stejné rozložení jako *FlowViewModel*, ale tlačítko je celé podbarvené podle *NodeStatusConverter* a hodnoty *ConnectionStatus* nacházející se v *NodeViewModel* (viz Kap. 4.1.3.2 a Obr. 51). Při kliknutí je proveden *ShowInfoCommand*, jenž otevře nové okno *NodeInfoWindow*.

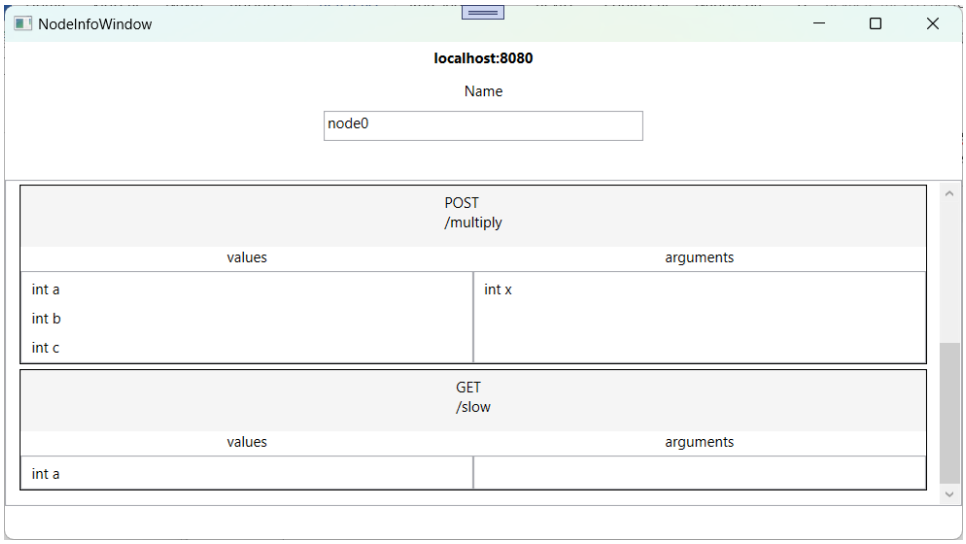


Obr. 51 Diagram tříd NodeViewModel

4.1.3.2 Okna

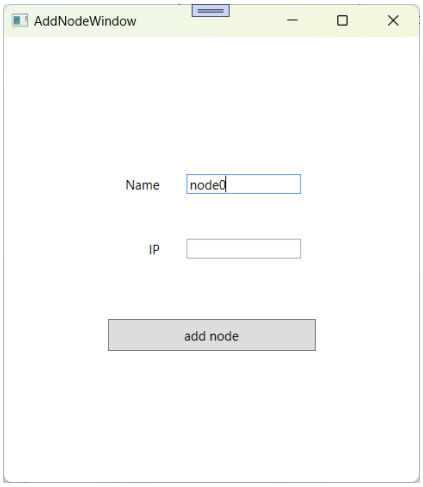
NodeInfoWindow (Obr. 52) slouží k zobrazení informací o uzlu. Jeho ViewModel je instance třídy *NodeViewModel*. V horní části okna se nachází jméno uzlu s výškou 60 px. Pod ním je vložen ListView obsahující komponenty EndpointView s výškou 5*, jehož zdrojem je kolekce *EndPoints*. Mezi elementy a okraji okna se nachází mezery vytvořené

pomocí řádků tabulky. Jejich výška je 0,5* což odpovídá přibližně 7,7 % výšky okna pro každou mezeru a 77 % pro list.



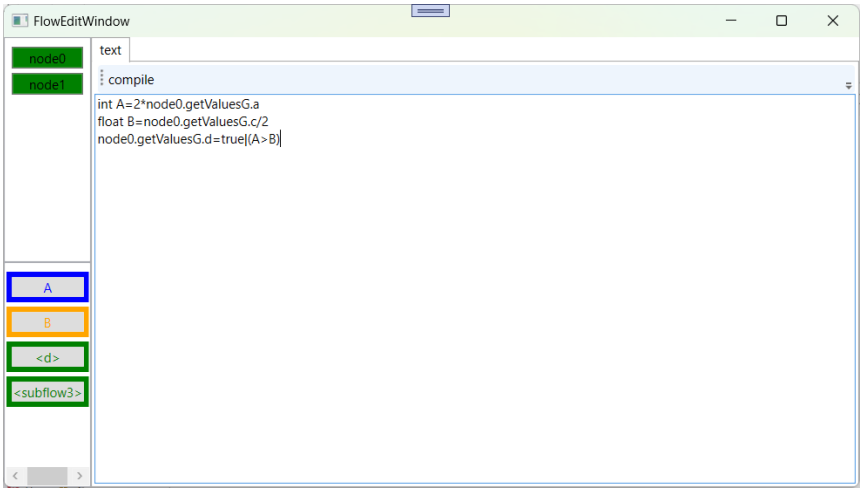
Obr. 52 Okno s informacemi o uzlu

AddNodeWindow (Obr. 53) je okno pro přidání nového uzlu, jehož logika se nachází v *NodeViewModel*. Pro snazší umístění prvků byla použita mřížka. Prvky jsou zarovnány na střed a dohromady na výšku zabírají 170 px a na šířku dvě čtvrtiny velikosti okna. Pro zadání názvu a adresy uzlu slouží dvě textová pole s popisem. Po zadání uživatel klikne na tlačítko, čímž je zavolán *ButtonClickCommand*. Tím je zavolána metoda *NodeRepository.AddNode* (viz Kap. 4.1.2.6), jež ověří validitu hodnot a přidá uzel do seznamu. Pokud je vše v pořádku, zobrazí okno *NodeInfoWindow* a aktualizuje kolekci v *NodeListViewModel*.

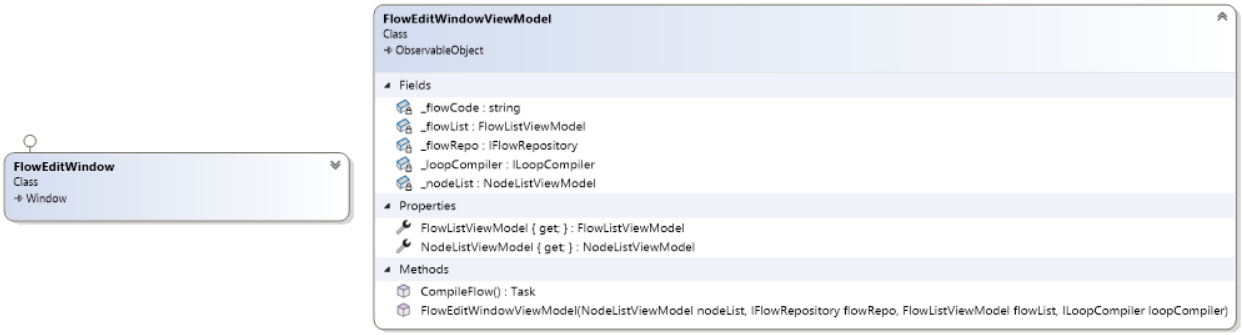


Obr. 53 Okno pro vložení nového uzlu

Pro zadávání logiky se využívá okno *FlowEditWindow* (Obr. 54) jenž využívá ViewModel *FlowEditWindowViewModel* (Obr. 55). V levé části okna se nachází sloupec se šířkou 80 px obsahující komponenty *NodeListView* a *FlowListView*, aby uživatel viděl názvy, které může použít při zadávání logiky. Zbytek okna zabírá *TabControl*, jenž má momentálně pouze jednu záložku, kterou je text. V ní se nachází *ToolBar* a *TextBox* vyplňující celou plochu. Do textového pole je možné psát více řádků, používat tabulátor a pokud se text nevejde na obrazovku, zobrazí se scrollbar. Na liště se nachází tlačítko „compile“, která spouští *CompileFlowCommand*. Ta smaže všechny datové toky a zavolá metodu *LoopCompiler.CompileMultiLine*. V závislosti na výsledku zobrazí buď *MessageBox* se zprávou „compiled successfully”, nebo chybovou hlášku. Poté vyvolá aktualizaci grafické reprezentaci seznamu datových toků.

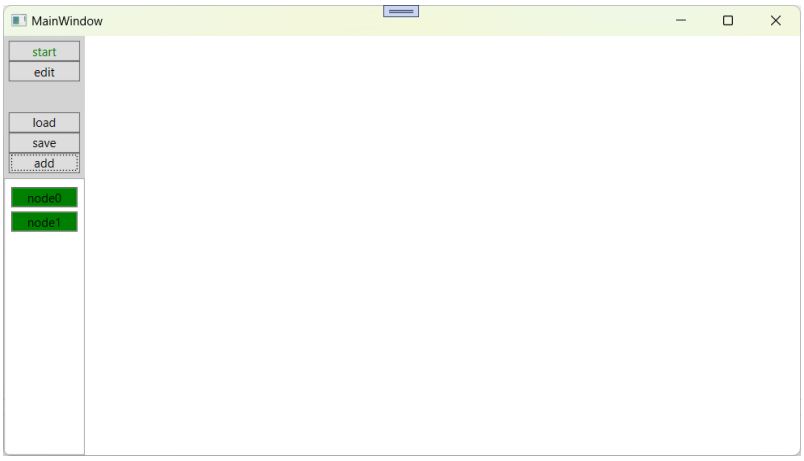


Obr. 54 Okno pro zadávání logiky

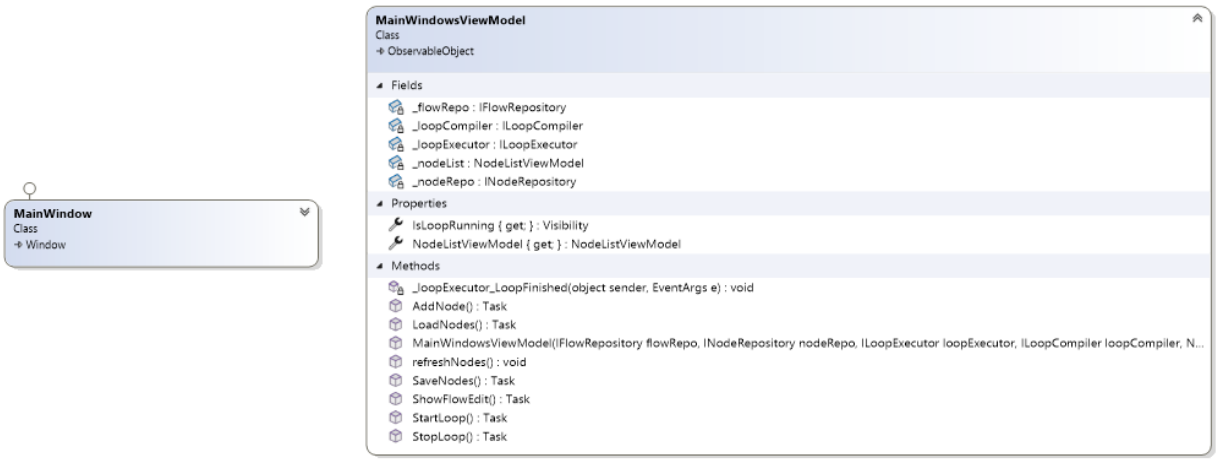


Obr. 55 Diagram tříd FlowEditViewModel

MainWindow (Obr. 56 a Obr. 57) je úvodní obrazovkou aplikace. Momentálně je využíváno pouze menu na levém okraji a zbytek je nevyužit. Zde se nachází dva *StackPanely* a *NodeListView*. První obsahuje tlačítka „start“ a „stop“ ovládající vyhodnocovací smyčku. Jejich viditelnost se mění podle hodnoty vlastnosti *IsLoopRunning* a za běhu aplikace je vidět právě jedno. Pod nimi se nachází tlačítko „edit“ otevírající okno *FlowEditWindow*. Druhý panel obstarává uzly a obsahuje tlačítka pro načtení a uložení uzlů z disku a pro přidání nového, které otevírá okno *AddNodeWindow*. Stav připojených uzlů je aktualizován pomocí metody, jež je napojena na událost vyvolanou ukončením vyhodnocovací smyčky.



Obr. 56 Úvodní obrazovka



Obr. 57 Diagram tříd MainWindowViewModel

4.2 Uzly

Jednotlivé uzly jsou realizovány pomocí vývojových desek NodeMCU pro ESP 8266 (viz kap. 3.5.1) a Arduino IDE. Z důvodu omezeného výkonu zde není striktní hranice mezi

komunikační a logickou vrstvou. Kde je to možné, byla použita abstrakce, aby byl kód přenositelný na mikrokontrolery nepodporované Arduino IDE nebo využívající jiný způsob komunikace. Kvůli využití třídy *Optional*, umožňující existenci proměnných bez hodnoty, je nutné použít kompilátor podporující C++17 nebo novější.

Projekt je strukturován tak, že většina kódu se nachází ve složce *scr*, která je rozdělena na *Abstract, HW, Lib* a složky pro jednotlivé uzly. *Abstract* obsahuje abstrakce knihoven specifických pro Arduino, které využívá zbytek projektu. Složka *HW* obsahuje abstrakci pro práci s jednotlivými senzory, zobrazovacími prvky a akčními členy. V *Lib* se nachází sdílená logika a DTO. V kořenové složce se nachází soubor *ESP.ino*, jenž je vstupním bodem pro Arduino IDE. Dále se zde nachází *secret.h*, sloužící k zapsání klíčů a jiných citlivých údajů. Při vytvoření do něj byly zapsány zástupné hodnoty, aby se po stažení repositáře dal projekt zkompilovat, a následně byl vyřazen z verzování.

4.2.1 Společná část

Pro analytické účely jsou během provozu uzlu posílána data na sériovou linku, což v případě neočekávaného chování usnadní hledání chyby. Po zapnutí je do terminálu odeslána zpráva „boot“ sloužící k označení začátku provozu. Toto je z důvodu, kdyby v terminálu zůstala komunikace z minulého připojení. Dále je vypsán aktuální stav paměti, aby bylo možné odhalit případné problémy z důvodu špatného rozmístění dat mezi zásobník a haldu. Poté proběhne inicializace specifická pro každý uzel. Po jejím dokončení je provedeno připojení k Wi-Fi za využití údajů nacházejících se v souboru *secret.h*, během něhož je vykonána funkce *WaitToConnect*. Ta je definována jako *weak*, což umožňuje, aby uzel měl jiné chování, než je výchozí. Pokud není definováno, je každých 500 ms ověřen stav připojení a dokud nedojde k připojení, je do terminálu posíláno „Connecting...“. Je-li připojování úspěšné, je vypsána IP adresa. Nakonec jsou přidány výchozí endpointy, kterými jsou *getInfo* a zavolání kořenové adresy, a do terminálu je napsáno „Server listening“ následováno časovým razítkem. Od této chvíle až do vypnutí, je v nekonečné smyčce očekáván dotaz určený k zpracování.

4.2.1.1 Abstract

Tato složka obsahuje hlavičkové soubory fungující jako rozhraní. Nachází se zde abstrakce pro komunikaci, logování, serializaci a deserializaci. Soubory *cpp* s implementací

pro Arduino se nachází v příslušně pojmenované podsložce. V případě rozšíření řešení o jinou technologii bude přidána nová složka. Jelikož je jen jedna možnost, tak v tento okamžik nejsou použity preprocesory určující, jaká implementace hlavičkových souborů má být využita.

Aby bylo možné měnit způsob komunikace s uzlem je definována třída *CommunicationHandler*. Metoda *StartListening* slouží k přidání reakce na specifikovaný endpoint. Parametry jsou cesta a ukazatel na funkci, jenž se má zavolat jako reakce na jeho obdržení. K odeslání odpovědi slouží metody *SendOk* a *SendError* jejímž parametrem je textový řetězec. Při přijetí dotazu s argumenty, je využita metoda *GetBody*, která má jako parametry buffer a jeho velikost. Pro zpracování přijatých dotazů slouží metoda *Loop*, jenž je volána z nekonečné smyčky v *main*, které odpovídá Arduino funkce *loop*. V případě ESP je implementací využívána instance třídy ESP8266WebServer [102]. Jedná se o HTTP server poslouchající na portu 80, který je spuštěn v konstruktoru *CommunicationHandler*.

Jelikož každá platforma má vlastní knihovny pro serializaci a deserializaci, byla i pro tuto logiku vytvořena abstrakce, která je pro vyšší přehlednost rozdělena na dva soubory. Protože pří kompilaci C++ jsou hlavičkové soubory zkopírovány do cpp soborů, jenž jsou kompilovány samostatně, je nutné jednoznačně určit datový typ využívaný v dané situaci [103]. Z tohoto důvodu nebylo možné pro serializaci *ValueDto* využít generické funkce a místo toho jsou deklarovány pro každý ze tří podporovaných datových typů samostatně. V závislosti na množství potřebných dat jsou pro hodnoty a endpointy definovány funkce *Serialize, SerializeInfo* a *SerializeValue*. První vypisuje veškeré údaje a je využívána pro logování. Druhá vytváří odpověď pro endpoint *getInfo*. Poslední se používá při dotazu na hodnoty uzlu. Všechny tři právě popsané funkce mají jako parametry ukazatel na instanci *EndPointDto,* buffer a jeho velikost pro serializaci ednpointu a ukazatel na ValueDto pro hodnoty. Pro serializaci celé kolekce je využívána *SerializeEndpoints*. Při přijetí dotazu, jenž obsahuje argumenty, je použita funkce *Deserialize* jejíž parametry jsou *const char** obsahující JSON a ukazatel na *EndPointDto*, kam se mají hodnoty zapsat. Arduino implementace využívá knihovnu *ArduinoJson* ve verzi 7.1.0 [104, 105].

Aby bylo možné provádět analýzu v případě chyby, je nutné za běhu programu někam zaznamenávat stavové informace. K tomu slouží funkce *Log* deklarované v hlavičkovém souboru *Logger.h*, které mají jako parametr buď *const char** nebo buffer a počet znaků. Pro využití v souborech specifický pro Arduino jsou tyto dvě funkce

deklarovány v *LoggerExtend.h* také jako String. Implementací je výpis do terminálu pomocí sériové linky.

4.2.1.2 Lib

V této složce se nachází soubory využívající pouze C++ a hlavičkové soubory z ostatních složek, jenž vytváří abstrakci nad platformě závislých knihovnách. Jedná se především o třídy se strukturou odpovídající DTO v hlavním uzlu (viz Kap. 4.1.1.1) a logiku potřebnou ke komunikaci s ním.

Generická třída *ValueDto* má konstruktor s *const char* name*, jenž slouží k identifikaci hodnoty v hlavním uzlu či jiné aplikaci načítající hodnoty z tohoto uzlu, a generickou *val*, která obsahuje uloženou hodnotu. Metoda *GetType* vracející *ValTypeEnum* je využívána při volání endpointu *getInfo*. Ke své činnosti využívá podmínky obsahující *constexpr is_same_v*, což je výraz porovnávající dva datové typy, jenž je vyhodnocen již během kompilace. Tato funkce vyžaduje kompilátor podporující C++14 nebo novější. Pokud se jedná o jiný datový typ, než je povolený, vyhodí výjimku *runtime_error*.

Třída *EndPointDto* obsahuje informace o endpointu. Pro zaznamenání adresy je využívána proměnná *const char* URL*. Aby bylo možné při zavolání *getInfo* sdělit jaká HTTP metoda má být použita, je třeba zadat hodnotu *HttpEnum*. Jelikož C++ nevyžaduje název enum před jeho hodnotou, bylo před využitím již existujícího enum upřednostněno vytvoření vlastního, neboť by při změně knihovny nebo migraci na jinou technologii mohlo docházet k nečekanému chování, z důvodu rozdílného pořadí hodnot nebo konfliktu s jinou knihovnou. Pokud je k zpracování dotazu více času, je zadána hodnota *Delay*, která zpracovávající aplikaci říká, kdy se ještě nejedná o chybu spojení. Tento údaj je deklarován jako *optional<int>*, čímž je umožněno, aby jinak hodnotový datový typ neměl zadanou hodnotu. Aby hlavní uzel či jiná aplikace pracující s tímto uzlem věděla, zda se jedná o vstupní či výstupní endpoint, je definován enum *EndPointType*. Pro uložení hodnot a argumentů složí *vectory*. Na rozdíl od hlavního uzlu je zde k dispozici omezené množství paměti, takže zde nejsou kolekce hodnot podle účelu sloučeny do dvou objektů, ale nachází se přímo v *EndPointDto* rozlišeny prefixem *Val_* nebo *Arg_*. Kromě bezparametrického konstruktoru má třída dva parametrické, lišící se parametrem *delay*. Povinnými hodnotami jsou HTTP metoda a URL adresa. Nepovinným údajem je EndPointType, jenž v případě nevyplnění má hodnotu *EP_TYPE_GET*, znamenající, že endpoint slouží k získání hodnot.

Hlavičkový soubor *Node.h* obsahuje deklarace funkcí *NodeInit* a *printEndpoint*. *NodeInit* slouží k definici endpointů specifických pro daný uzel a nachází se v souborech pro konkrétní uzel umístěných ve složkách o úroveň výše. Který z nich bude kompilován se vybírá na základě definice nacházející se na začátku *Node.h* a preprocesorů. Funkce *printEndpoint* je využívána jako kontrola při vytvoření nového endpointu a je definována v souboru *NodeShared.cpp* tak, že provede serializaci veškerých údajů a zapíše je do logu.

Pár hlavičkové souboru a zdrojového kódu *SharedHttpEndpoints* obsahují přidání základních endpointů, kterými jsou *getInfo* a volání bez cesty. Funkce, jež se provede při zavolání *getInfo*, zavolá *SerializeEndpoints* výsledek zapíše do logu a odešle jako odpověď. V případě, že se za adresou uzlu nenachází žádná cesta, je do logu zapsána autorizační hlavička a tělo dotazu. Poté je odeslána odpověď „hello world“. Tento endpoint je využíván při testování komunikace, kdy není žádoucí, aby se uzel pokoušel o zpracování právě přijatých hodnot.

4.2.2 Uzel 1

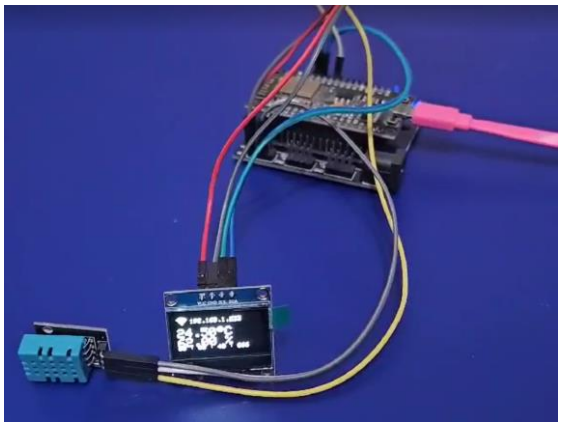
První realizovaný uzel (Obr. 58) je vybaven sensorem DHT11 a monochromatickým OLED displejem. DHT11 od společnosti Adafruit je levný teploměr a vlhkoměr komunikující pomocí protokolu 1wire. Ačkoliv se s přesností teploty ± 2 °C jedná spíše o orientační hodnotu, byl tento sensor ponechán pro své dynamické vlastnosti. Jelikož je po odečtení hodnoty potřeba počkat dalších 2000 ms, než je možno získat další, slouží tento sensor jako ukázka, jak si hlavní uzel poradí s pomalu odpovídajícím uzlem. Pro komunikaci jsou využívány knihovny *DHT sensor library* a *Adafruit Unified Sensor[106, 107]*. Bílý monochromatický OLED displej s rozlišením 128x64 px je řízen čipem SSD1306, který umožňuje komunikaci pomocí I2C. Pro ovládání jsou využívány knihovny *Adafruit GFX Library* a *Adafruit_SSD1306* [108, 109]. [110–112]

Pro abstrakci uvnitř *Node1.cpp* byla vytvořena třída *DhtWrapper*. Kromě snazší přenositelnosti je důvodem k abstrakci fakt, že knihovna po dobu 2000 ms od posledního čtení vrací stejnou hodnotu, ale časové razítko není zvenčí dostupné. Pro získání hodnot slouží metoda *ReadRaw*, jenž získá teplotu a vlhkost ihned po sobě a uloží je do proměnných. Poté porovná své časové razítko a pokud uplynul daný limit, aktualizuje ho. Pro přístup k takto přečteným hodnotám slouží metody *GetTemp* a *GetHumid* a *GetDataAge*. Pokud je

potřeba aby data byla aktuální, je zavolána metoda *WaitForNewestData*, která počká do uplynutí zbývajícího času a poté teprve proběhne četní.

Tento uzel má přepsanou společnou funkci *WaitToConnect*, aby během čekání na připojení nejen vypisovala tečky do terminálu, ale také na displeji blikal symbol Wi-Fi. Po připojení se vedle něj vypíše IP adresa.

Jsou definovány endpointy *getDhtValuesNew* a *getDhtValuesAny*, které vrací teplotu, vlhkost a stáří dat. Liší se tím, zda jsou poslány hodnoty bez ohledu na stáří, nebo je čekáno na čerstvé. Poté co je odeslána odpověď, jsou tyto údaje vypsány na displej včetně časové značky počítané od doby zapnutí napájení.



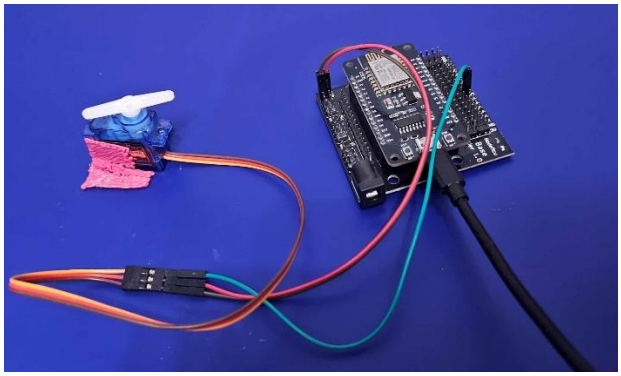
Obr. 58 fotografie uzel 1

4.2.3 Uzel 2

Druhý realizovaný uzel (Obr. 59) je vybaven servomotorem SG-90 s úhlem otočení 180°, jenž je napájen 5 V [113]. Tento modul má simulovat otevírání ventilu, ale nebyl k němu připevněn žádný předmět. K ovládání byla použita ESP8266 implementace Arduino knihovny *Servo* [114]. Ta na základě zadaného úhlu generuje PWM signál potřebný k dosažení požadované polohy.

Pro abstrakci v kódu uzlu byla vytvořena třída *ServoWrapper*, jenž skryje platformě závislý kód. Její konstruktor přijímá pin, kam je servomotor připojen, a vytváří instanci třídy knihovny. Má metody *SetAngle* a *GetAngle*, jenž obalují volání *write* a *read*.

Pro nastavení úhlu natočení servomotoru slouží endpoint *setAngle*, který je typu *EP_TYPE_SET*. Má celočíselný Argument *angle* a vrací odpověď obsahující hodnotu *deg*. Ta slouží pro kontrolu a jedná se o skutečnou hodnotu úhlu v době poslání odpovědi. Druhým endpointem je getAngle, který vrací také hodnotu *deg*, ale oproti předchozímu slouží pouze k získání hodnoty.



Obr. 59 fotografie uzlu 2

4.3 Pomocné projekty

Kromě těchto dvou právě popsaných hlavních částí během vývoje vznikly další dva pomocné projekty, které nejsou součást zadání a jejich účel je usnadnit vývoj. Prvním je *NodeEmulator*, který napodobuje chování uzlů. Druhým je *MainNode.Logic.Test*, jehož úkolem je jedním kliknutím ověřit, že pro daný vstup bude mít aplikace očekávaný výstup.

NodeEmulator je WinForm aplikace, jejímž úkolem je umožnění testování logiky v okamžiku, kdy není možné použít fyzický uzel. Toto je výhodné například v okamžiku, kdy ještě žádný neexistuje nebo se nachází mimo dosah programátora. Projekt má závislost na knihovně *MainNode.Communication*, čímž je zajištěna shodná podoba DTO jako má hlavní uzel. Veškerá logika je zapsána přímo v kódu a ke spuštění není potřeba žádný zásah uživatele. Hodnoty endpointů jsou zobrazeny v okně, kde je možné je měnit. K tomuto virtuálnímu uzlu je možné přistupovat z portu 8080.

MainNode.Logic.Test je projekt typu MSTest a slouží k testování scénářů v logické vrstvě. Momentálně je vytvořeno omezené množství testů, jež se zaměřují na chování třídy *LoopCompiler*. Tyto testy ověřují, zda pro různé kombinace datových typů, operací, konstant a referencí bude po spuštění datových toků navrácena správná hodnota. Názvy testovacích metod mají na začátku první písmena datových typů. Následuje podtržítko a zkoušená operace. Název končí informacemi o tom, zda jsou hodnoty konstanty, reference nebo další datové toky.

5 Výsledky a diskuse

Z existujících řešení je této práci nejpodobnější Node-RED. Jedná se o událostmi řízenou Node.js aplikaci [115, 116]. Hlavním rozdílem ve fungování je způsob komunikace. V případě Node-RED vyhodnocení větve datové toku začíná při obdržení zprávy z periferního zařízení, zatímco v řešení realizované touto prací se hlavní uzel na hodnoty aktivně ptá. To umožňuje existenci dvou souběžně běžících systémů sdílejících stejnou periferii bez nutnosti vytvářet server. Toto je výhodné především pro technicky méně zdatné uživatele, kterým stačí pouze stáhnout a spustit exe soubor. Další výhodou je větší versatilita uživatelského rozhraní, jelikož díky vrstvenému modelu je možné vytvořit nové, aniž by to ovlivnilo logiku aplikace.

Je-li potřeba aby logika byla vykonávána na jednočipovém počítači, který oproti klasickému osobnímu počítači může fungovat na baterii po dobu několika měsíců, je možné Node-RED spustit na Raspberry Pi nebo BeagleBone [117]. Na tytéž vývojové desky je možné s pomocí knihovny *.NET IoT* nasadit i *MainNode* vytvořený v této práci [118]. Dále je dostupný také *.NET nanoFramework*, jenž umožňuje spouštět kód napsaný v .NET na méně výkonných čipech jako jsou ESP32 a STM32F429, avšak kvůli hardwarovému omezení nepodporuje všechny funkce [119, 120]. Řešení bylo navrženo tak, aby bylo možné bez zásahů do logiky ho z C# přepsat do C++. Pokud při realizaci nebyla udělána chyba, mělo by se jednat pouze o rozdíl v syntaxi (např. vlastnosti a lambda výrazy).

Řešení vytvořené v této práci momentálně nenabízí pokročilé funkce jako Node-RED. Přidání vlastních funkcí vyžaduje zásah do *LoopCompiler*. Ačkoliv pro data není vytvořena vizualizace, logická vrstva poskytuje dostatek dat, aby bylo možné ji doplnit. Přidání nového komunikačního protokolu do tohoto řešení je oproti Node-RED jednodušší [121, 122].

6 Závěr

Byl vysvětlen význam jednotlivých vrstev OSI modelu během síťové komunikace a byly popsány nejdůležitější protokoly, které jsou k ní používány. Poté byl vysvětlen rozdíl mezi bitovým a textovým formátem dat a struktura XML, JSON a CSV. Oblast síťové komunikace byla uzavřena vysvětlením principu funkce Wi-Fi, včetně její stručné historie a rozdílů mezi jednotlivými generacemi. Také byl ukázán rozdíl mezi rámcy IEEE 802.3 (Ethernet) a IEEE 802.11 (Wi-Fi).

Teoretická část pokračovala vysvětlením pojmů mikroprocesor, SoC a mikrokontroler. Dále byl představen čip ESP8266, jeho moduly a vývojová deska NodeMCU. Teorie byla zakončena vysvětlením programovacích technik souvisejících s touto prací.

Praktickou část tvoří hlavní uzel, sdílený kód pro uzly realizované pomocí ESP8266, vzorové implementace těchto uzlů a pomocné projekty sloužící pro testování.

Hlavní uzel je realizován jako vrstvený model skládající se ze dvou knihoven a uživatelského rozhraní v podobě WPF aplikace. Jeho účelem je interakce s uživatelem a vykonávání vyhodnocovací smyčky na základě zadané logiky. Vstupem jsou hodnoty získané z ostatních uzlů a po dokončení jsou hodnoty poslány uzlům s akčním členem. Jelikož byl použit vrstvený model, je možné jednotlivé části nahradit bez ovlivnění zbytku aplikace. Současně je řešení navrženo tak, aby bylo možné využít více protokolů současně.

Uzly realizované pomocí ESP8266 mohou fungovat jako vstupní, výstupní nebo oboje současně. Při programování bylo využito Arduino IDE, jenž poskytuje abstrakci od práce s registry daného čipu. Jelikož ne všichni výrobci mají plnou podporu všech funkcí, byla pomocí hlavičkových souborů vytvořena abstrakce, aby kód šlo použít i pro tyto čipy. ESP8266 implementace využívá HTTP server s endpointy pro potřebné operace. Každý uzel má vlastní soubor obsahující definice těchto endpointů. Pro volbu, který soubor bude použit, slouží definice a na ní navázaný preprocesor.

Kvůli nečekaným komplikacím nebyly implementovány všechny plánované funkce, jako je vizualizace a automatické znovu připojení v případě ztráty spojení. Kromě těchto funkcí je do budoucna plánováno přidat mechanismy, řešící situace, kdy se systém dostane do nežádoucího stavu. Dále je zamýšleno přidat možnost definovat vlastní znovupoužitelné funkce a zadávat logiku pomocí grafického rozhraní, aby bylo řešení více přístupné technicky méně zdatným uživatelům. Do komunikační vrstvy je plánováno přidat podporu

dalších protokolů a API pro chytrou domácnost (např. Samsung SmartThings [123]). Jak bylo v této práci několikrát zmíněno, počítá se spuštěním hlavního uzlu na jednočipovém počítači. Pro tuto úlohu jsou zvažovány vývojové desky STM32F429 Discovery s grafickým displejem [124] a dvoujádrové Nucleo STM32H755 s Ethernet rozhraním [125]. Testování bude postupně rozšířeno i na další třídy. Plánuje se zapojit *NodeEmulator* a zprovoznit kompletní CI/CD pipeline.

Celá tato práce je dostupná na GitHub. Verze aktuální v době odevzdání se nachází ve větvi *Release_1_0_0* (https://github.com/pjocesoj/diplomka_git/tree/Release_1_0_0). V souboru README.md jsou na začátku rychlé odkazy. Zdrojový kód se nachází ve složce *prakticka_cast*, kde jsou složky pro obě hlavní části a *NodeEmulator*. Ve složce *output* jsou umístěny exe soubory a markdown verze této práce, které jsou vytvářeny automaticky pomocí GitHub Actions.

7 Seznam použitých zdrojů

[1] What is OSI Model | 7 Layers Explained. *GeeksForGeeks* [online]. [vid. 2025-01-28]. Dostupné z: https://www.geeksforgeeks.org/open-systems-interconnection-model-osi/

[2] Difference Between OSI Model and TCP/IP Model - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-28]. Dostupné z: https://www.geeksforgeeks.org/difference-between-osi-model-and-tcp-ip-model/

[3] MICHAEL GOODWIN a CHRYSTAL R. CHINA. What Is the OSI Model? | IBM. *IBM* [online]. [vid. 2025-01-28]. Dostupné z: https://www.ibm.com/think/topics/osi-model

[4] What is the OSI Model? | Cloudflare. *Cloudflare* [online]. [vid. 2025-01-28]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

[5] *Bytebytego Big Archive System Design 2023* [online]. 2023 [vid. 2025-01-26]. Dostupné z: https://blog.bytebytego.com/p/free-system-design-pdf-158-pages

[6] MICHAEL GOODWIN, GITA JACKSON a TASMIHA KHAN. What Is Network Topology? | IBM. *IBM* [online]. [vid. 2025-01-28]. Dostupné z: https://www.ibm.com/think/topics/network-topology

[7] What Is a Data Link Layer? | Coursera. *Coursera* [online]. [vid. 2025-02-02]. Dostupné z: https://www.coursera.org/articles/data-link-layer

[8] What is Cyclic Redundancy Check (CRC) and How Does it Work? | Lenovo US. *Lenovo* [online]. [vid. 2025-02-02]. Dostupné z: https://www.lenovo.com/us/en/glossary/cyclic-redundancy-check/?orgRef=https%253A%252F%252Fwww.perplexity.ai%252F

[9] Difference Between Packet And Frame - PyNet Labs. *PyNet Labs* [online]. [vid. 2025-02-08]. Dostupné z: https://www.pynetlabs.com/what-is-the-difference-between-packet-and-frame/

[10] SUBHAM DATTA. Definition of Network Units: Packet, Fragment, Frame, Datagram, and Segment | Baeldung on Computer Science. *Baeldung* [online]. [vid. 2025-03-30]. Dostupné z: https://www.baeldung.com/cs/networking-packet-fragment-frame-datagram-segment

[11] What is Ports in Networking? - GeeksforGeeks. *GeeksforGeeks* [online]. [vid. 2025-02-05]. Dostupné z: https://www.geeksforgeeks.org/what-is-ports-in-networking/?ref=header_outind

[12] Connection-less Service - GeeksforGeeks. *GeeksforGeeks* [online]. [vid. 2025-02-05]. Dostupné z: https://www.geeksforgeeks.org/connection-less-service/

[13] Connection-Oriented Service - GeeksforGeeks. *GeeksforGeeks* [online]. [vid. 2025-02-05]. Dostupné z: https://www.geeksforgeeks.org/connection-oriented-service/

[14] PETERKA, Jiří. Lekce 3: Vrstvy a vrstvové modely. In: *NSWI090: Počítačové sítě I (verze 4.0)* [online]. nedatováno [vid. 2025-03-30]. Dostupné z: https://www.ksi.mff.cuni.cz/~svoboda/courses/182-NSWI090/lectures/P%C5%99edn%C3%A1%C5%A1ka-03-Vrstvov%C3%A9-modely.pdf

[15] What is Protocol? A Guide to Understanding | Lenovo US. *Lenovo* [online]. [vid. 2025-02-10]. Dostupné z: https://www.lenovo.com/us/en/glossary/what-is-protocol/?orgRef=https%253A%252F%252Fwww.perplexity.ai%252F

[16] IPv4 vs. IPv6: what are the differences in 2025? - Surfshark. *SurfShark* [online]. [vid. 2025-02-11]. Dostupné z: https://surfshark.com/blog/ipv4-vs-ipv6

[17] What is the Internet Protocol? | Cloudflare. *Cloudflare* [online]. [vid. 2025-02-10]. Dostupné z: https://www.cloudflare.com/learning/network-layer/internet-protocol/

[18] PETERKA, Jiří. Téma 4: Adresy a adresování v TCP/IP, IP adresy verze 4. In: *NSWI045: Rodina protokolů TCP/IP (verze 3)* [online]. nedatováno [vid. 2025-03-30]. Dostupné z: https://www.ksi.mff.cuni.cz/~svoboda/courses/182-NSWI090/lectures/P%C5%99edn%C3%A1%C5%A1ka-09-Adresov%C3%A1n%C3%AD-TCPIP.pdf

[19] KRISTOFER KOISHIGAWA. Subnet Cheat Sheet – 24 Subnet Mask, 30, 26, 27, 29, and other IP Address CIDR Network References. *FreeCodeCamp* [online]. [vid. 2025-02-12]. Dostupné z: https://www.freecodecamp.org/news/subnet-cheat-sheet-24-subnet-mask-30-26-27-29-and-other-ip-address-cidr-network-references/

[20] ADITYAPRATAPBHUYAN. Understanding Network Address Translation (NAT) in Networking: A Comprehensive Guide - DEV Community. *Dev.to* [online]. [vid. 2025-02-13]. Dostupné z: https://dev.to/adityapratapbh1/understanding-network-address-translation-nat-in-networking-a-comprehensive-guide-8bo

[21] What is the User Datagram Protocol (UDP)? | Cloudflare. *Cloudflare* [online]. [vid. 2025-02-14]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/

[22] *Transmission Control Protocol (TCP) (článek)* | *Khan Academy* [online]. [vid. 2025-02-14]. Dostupné z: https://cs.khanacademy.org/computing/informatika-pocitace-a-internet/x8887af37e7f1189a:internet/x8887af37e7f1189a:tcp-protokol/a/transmission-control-protocol--tcp

[23] *What is TCP/IP?* | *Cloudflare* [online]. [vid. 2025-02-14]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/

[24] Dynamic Host Configuration Protocol (DHCP) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-02-13]. Dostupné z: https://learn.microsoft.com/en-us/windows-server/networking/technologies/dhcp/dhcp-top

[25] An overview of HTTP - HTTP | MDN. *Mozilla Developer Network* [online]. [vid. 2025-02-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[26] Caching - IBM Documentation. *IBM* [online]. [vid. 2025-02-19]. Dostupné z: https://www.ibm.com/docs/en/was-nd/8.5.5?topic=discussions-caching

[27] HTTP Load Balancing | NGINX Documentation. *NGINX* [online]. [vid. 2025-02-19]. Dostupné z: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/

[28] HTTP response status codes - HTTP | MDN. *Mozilla Developer Network* [online]. [vid. 2025-02-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[29] RUFAI MUSTAPHA. What is HTTP? Protocol Overview for Beginners. *freeCodeCamp* [online]. [vid. 2025-02-14]. Dostupné z: https://www.freecodecamp.org/news/what-is-http/

[30] How does public key cryptography work? | Public key encryption and SSL | Cloudflare. *Cloudflare* [online]. [vid. 2025-02-17]. Dostupné z: https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/

[31] Let’s Encrypt Stats - Let’s Encrypt. *Let’s Encrypt* [online]. [vid. 2025-03-30]. Dostupné z: https://letsencrypt.org/stats/#percent-pageloads

[32] ILYA GRIGORIK a PIERRE FAR. *Google I/O 2014 - HTTPS Everywhere - YouTube* [online]. [vid. 2025-03-31]. Dostupné z: https://www.youtube.com/watch?v=cBhZ6S0PFCY

[33] ARTHUR BELLORE. The TLS Handshake Explained. *auth0* [online]. [vid. 2025-02-15]. Dostupné z: https://auth0.com/blog/the-tls-handshake-explained/

[34] BYTEBYTEGO. *SSL, TLS, HTTPS Explained - YouTube* [online]. [vid. 2025-02-15]. Dostupné z: https://www.youtube.com/watch?v=j9QmMEWmcfo

[35] COMPUTERPHILE a DR. MIKE POUND. *TLS Handshake Explained - Computerphile - YouTube* [online]. [vid. 2025-02-15]. Dostupné z: https://www.youtube.com/watch?v=86cQJ0MMses&t=4s

[36] How does public key cryptography work? | Public key encryption and SSL | Cloudflare. *Cloudflare* [online]. [vid. 2025-02-15]. Dostupné z: https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/

[37] What is SSL/TLS Certificate? - SSL/TLS Certificates Explained - AWS. *Amazon Web Services* [online]. [vid. 2025-02-15]. Dostupné z: https://aws.amazon.com/what-is/ssl-certificate/

[38] HTTP vs HTTPS - Difference Between Transfer Protocols - AWS. *Amazon Web Services* [online]. [vid. 2025-02-15]. Dostupné z: https://aws.amazon.com/compare/the-difference-between-https-and-http/

[39] BAELDUNG. What Are Serialization and Deserialization in Programming? | Baeldung on Computer Science. *Baeldung* [online]. [vid. 2025-02-22]. Dostupné z: https://www.baeldung.com/cs/serialization-deserialization

[40] SATRAPA, Pavel. Jazyky pro popis dat. In: *Jazyky pro popis dat* [online]. B.m.: TUL, nedatováno [vid. 2025-02-21]. Dostupné z: https://www.nti.tul.cz/~satrapa/vyuka/xml/prednaska01.pdf

[41] SATRAPA, Pavel. API pro XML. In: *Jazyky pro popis dat* [online]. B.m.: TUL, nedatováno [vid. 2025-02-20]. Dostupné z: https://www.nti.tul.cz/~satrapa/vyuka/xml/prednaska12.pdf

[42] ŠIMON RAICHL. Lekce 7 - Formát JSON. *ITnetwork.cz* [online]. [vid. 2025-02-21]. Dostupné z: https://www.itnetwork.cz/javascript/oop/objekty-json-a-vylepseni-diare-v-javascriptu

[43] JSON. *JSON.org* [online]. [vid. 2025-02-21]. Dostupné z: https://www.json.org/json-en.html

[44] PETR SEDLÁČEK. Lekce 2 - REST API, SOAP, GraphQL a JSON. *ITnetwork.cz* [online]. [vid. 2025-02-21]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/rest-api-soap-graph-a-json

[45] Exploring Why CSV is a Popular File Format and How to Manage it | Lenovo UK. *Lenovo* [online]. [vid. 2025-02-21]. Dostupné z: https://www.lenovo.com/gb/en/glossary/csv/?orgRef=https%253A%252F%252Fwww.perplexity.ai%252F

[46] What does Wi-Fi Stand For? | CORSAIR. *CORSAIR* [online]. [vid. 2025-03-28]. Dostupné z: https://www.corsair.com/us/en/explorer/gamer/gaming-pcs/what-does-wi-fi-stand-for/

[47] *What Is Wireless Fidelity and is it the same as WiFi?* [online]. [vid. 2025-03-28]. Dostupné z: https://stl.tech/blog/wireless-fidelity-the-rundown/

[48] Standardy Wi-Fi: IEEE 802.11ac, 802.11ax a standardy bezdrátového připojení | Dell Česká republika. *Dell* [online]. [vid. 2025-02-22]. Dostupné z: https://www.dell.com/support/contents/cs-cz/article/product-support/self-support-knowledgebase/networking-wifi-and-bluetooth/wi-fi-network-standards-overview

[49] IEEE SA - The Evolution of Wi-Fi Technology and Standards. *IEEE* [online]. [vid. 2025-02-23]. Dostupné z: https://standards.ieee.org/beyond-standards/the-evolution-of-wi-fi-technology-and-standards/

[50] Different Wi-Fi Protocols and Data Rates. *Intel* [online]. [vid. 2025-02-23]. Dostupné z: https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html#primary-content

[51] What is WiFi 6E? | TP-Link. *TP-Link* [online]. [vid. 2025-02-24]. Dostupné z: https://www.tp-link.com/us/wifi-6e/

[52] Brief introduction of Wireless Channel, Channel Width and DFS | TP-Link Norway. *TP-Link* [online]. [vid. 2025-02-28]. Dostupné z: https://www.tp-link.com/no/support/faq/4309/

[53] KLEMENT, Doc Phdr Milan. Univerzita Palackého v Olomouci Technologie bezdrátových sítí základní principy a standardy [online]. 2019 [vid. 2025-02-26].

Dostupné
z: https://www.pdf.upol.cz/fileadmin/userdata/PdF/katedry/ktiv/Studijni_materialy/Klement/2019/TBS_2019_skripta.pdf

[54] What is a Wireless Access Point (WAP)? Benefits & How It Works | Lenovo US. *Lenovo* [online]. [vid. 2025-02-28]. Dostupné z: https://www.lenovo.com/us/en/glossary/wireless-access-point/?orgRef=https%253A%252F%252Fwww.perplexity.ai%252F

[55] STA Access - NetEngine AR600, AR6100, AR6200, and AR6300 V300R019 CLI-based Configuration Guide - WLAN-FAT AP - Huawei. *Huawei* [online]. [vid. 2025-02-28]. Dostupné z: https://support.huawei.com/enterprise/en/doc/EDOC1100112363/75acc8a8/sta-access

[56] [MS-TCC]: Glossary | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-03-01]. Dostupné z: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-tcc/477dff81-3c9d-4b84-b002-1a9fe7659e0a#gt_59565412-59a1-4e14-862f-28810f583050

[57] 802.11 Standards - NetEngine AR600, AR6100, AR6200, and AR6300 V300R019 CLI-based Configuration Guide - WLAN-FAT AP - Huawei. *Huawei* [online]. [vid. 2025-02-09]. Dostupné z: https://support.huawei.com/enterprise/en/doc/EDOC1100112363/b1db415/80211-standards

[58] Wi-Fi CERTIFIED 6™ coming in 2019 | Wi-Fi Alliance. *Wi-Fi Alliance* [online]. [vid. 2025-02-24]. Dostupné z: https://www.wi-fi.org/news-events/newsroom/wi-fi-certified-6-coming-in-2019

[59] LINUS TECH TIPS. *Just how FAST is WiFi 6? - YouTube* [online]. [vid. 2025-02-24]. Dostupné z: https://www.youtube.com/watch?v=Mx5-T8ZwxbU

[60] What Is Wi-Fi 6? - Intel. *Intel* [online]. [vid. 2025-02-24]. Dostupné z: https://www.intel.com/content/www/us/en/gaming/resources/wifi-6.html

[61] ANTHONY M. BRUNO. What is Quadrature Amplitude Modulation (QAM)? *CWNP* [online]. [vid. 2025-03-02]. Dostupné z: https://www.cwnp.com/qam-basics/

[62] What Is QAM? How Does QAM Work? - Huawei. *Huawei* [online]. [vid. 2025-03-02]. Dostupné z: https://info.support.huawei.com/info-finder/encyclopedia/en/QAM.html

[63] WPA2 Security (KRACKs) Vulnerability Statement | TP-Link Baltic. *TP-Link* [online]. [vid. 2025-03-02]. Dostupné z: https://www.tp-link.com/baltic/support/faq/1970/

[64] IRMA ŠLEKYTĖ. WEP, WPA, WPA2, and WPA3: Main differences | NordVPN. *NordVPN* [online]. [vid. 2025-03-02]. Dostupné z: https://nordvpn.com/blog/wep-vs-wpa-vs-wpa2-vs-wpa3/

[65] An Introduction to Spread-Spectrum Communications | Analog Devices. *Analog Devices* [online]. [vid. 2025-02-25]. Dostupné z: https://www.analog.com/en/resources/technical-articles/introduction-to-spreadspectrum-communications--maxim-integrated.html

[66] IEEE SA - Actress/Inventor Hedy Lamarr – and How Far Wireless Communication Has Come. *IEEE* [online]. [vid. 2025-03-30]. Dostupné z: https://standards.ieee.org/beyond-standards/hedy-lamarr/

[67] LESLIE A. RUSCH. GEL7114 - Module 4.12 - OFDM introduction. In: *GEL-7114 Digital Communications* [online]. B.m.: Universite Laval, 2020 [vid. 2025-03-01]. Dostupné z: https://www.youtube.com/watch?v=i3LBGw8Yle4

[68] BHARDWAJ, Manushree, Arun GANGWAR a Devendra SONI. A Review on OFDM: Concept, Scope & its Applications. *IOSR Journal of Mechanical and Civil Engineering (IOSRJMCE)* [online]. nedatováno, **1**(1), 7–11 [vid. 2025-03-03]. Dostupné z: www.iosrjournals.orgwww.iosrjournals.org

[69] RF ELEMENTS S.R.O. *Inside Wireless: MIMO Introduction - Multiple Input Multiple Output - YouTube* [online]. [vid. 2025-02-26]. Dostupné z: https://www.youtube.com/watch?v=T7NyrG4_RSI

[70] What Is MIMO? From SISO to MIMO - Huawei. *Huawei* [online]. [vid. 2025-03-02]. Dostupné z: https://info.support.huawei.com/info-finder/encyclopedia/en/MIMO.html

[71] Detailed explanation of MU-MIMO technology and the application of MU-MIMO in WiFi6. *FS* [online]. [vid. 2025-03-02]. Dostupné z: https://www.fs.com/blog/demystifying-mumimo-technology-in-wifi-6-115.html

[72] GREAVES, David J. *Modern System-on-Chip Design on Arm* [online]. B.m.: ARM, nedatováno [vid. 2025-03-03]. ISBN 978-1-911531-37-1. Dostupné z: https://armkeil.blob.core.windows.net/developer/Files/pdf/ebook/arm-modern-soc-design-on-arm.pdf

[73] VLSI | Analog Devices. *Analog Devices* [online]. [vid. 2025-03-05]. Dostupné z: https://www.analog.com/en/resources/glossary/vlsi.html

[74] JOSH SCHNEIDER a IAN SMALLEY. What is a microprocessor? | IBM. *IBM* [online]. [vid. 2025-03-05]. Dostupné z: https://www.ibm.com/think/topics/microprocessor

[75] JOSH SCHNEIDER a IAN SMALLEY. What is a microcontroller? | IBM. *IBM* [online]. [vid. 2025-03-05]. Dostupné z: https://www.ibm.com/think/topics/microcontroller

[76] ESPRESSIF SYSTEMS. *ESP8266EX Datasheet* [online]. 2023 [vid. 2025-03-06]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[77] *Getting Started with ESP8266 NodeMCU Development Board| Random Nerd Tutorials* [online]. [vid. 2025-03-07]. Dostupné z: https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/

[78] ESPRESSIF SYSTEMS. *ESP8266 Hardware Design Guidelines Version 2.8* [online]. 2024 [vid. 2025-03-07]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp8266_hardware_design_guidelines_en.pdf

[79] What's a design pattern? *Refactoring Guru* [online]. [vid. 2025-01-25]. Dostupné z: https://refactoring.guru/design-patterns/what-is-pattern

[80] Why should I learn patterns? *Refactoring Guru* [online]. [vid. 2025-01-25]. Dostupné z: https://refactoring.guru/design-patterns/why-learn-patterns

[81] Difference Between Architectural Style, Architectural Patterns and Design Patterns - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-26]. Dostupné z: https://www.geeksforgeeks.org/difference-between-architectural-style-architectural-patterns-and-design-patterns/

[82] STEVE “ARDALIS” SMITH. *Architecting-Modern-Web-Applications-with-ASP.NET-Core-and-Azure* [online]. 2023 [vid. 2025-01-21]. Dostupné z: https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf

[83] RITVIK GUPTA. Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project | Turing. *Turing* [online]. [vid. 2025-01-26]. Dostupné z: https://www.turing.com/blog/software-architecture-patterns-types

[84] Dependency Injection(DI) Design Pattern - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-19]. Dostupné z: https://www.geeksforgeeks.org/dependency-injectiondi-design-pattern/

[85] Single Responsibility in SOLID Design Principle - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-19]. Dostupné z: https://www.geeksforgeeks.org/single-responsibility-in-solid-design-principle/

[86] NuGet Gallery | Microsoft.Extensions.DependencyInjection 1.0.0. *NuGet* [online]. [vid. 2025-01-23]. Dostupné z: https://www.nuget.org/packages/Microsoft.Extensions.DependencyInjection/1.0.0#supportedframeworks-body-tab

[87] Dependency injection - .NET | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-01-23]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

[88] Create Data Transfer Objects (DTOs) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-01-24]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

[89] BAELDUNG. The DTO Pattern (Data Transfer Object) | Baeldung. *Baeldung* [online]. [vid. 2025-01-24]. Dostupné z: https://www.baeldung.com/java-dto-pattern

[90] *Observer* [online]. [vid. 2023-03-21]. Dostupné z: https://refactoring.guru/design-patterns/observer

[91] Difference Between MVC, MVP and MVVM Architecture Pattern in Android - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2024-11-26]. Dostupné z: https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/

[92] NIMROD KRAMER. Android Architecture Patterns: MVC vs MVVM vs MVP. *daily.dev* [online]. [vid. 2025-01-03]. Dostupné z: https://daily.dev/blog/android-architecture-patterns-mvc-vs-mvvm-vs-mvp

[93] RICH LANDER. core/release-notes/8.0/supported-os.md at main · dotnet/core · GitHub. *GitHub* [online]. [vid. 2025-03-10]. Dostupné z: https://github.com/dotnet/core/blob/main/release-notes/8.0/supported-os.md

[94] . NET and .NET Core official support policy. *Microsoft* [online]. [vid. 2025-03-10]. Dostupné z: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

[95] core/release-notes/6.0/supported-os.md at main · dotnet/core · GitHub. *GitHub* [online]. [vid. 2025-03-10]. Dostupné z: https://github.com/dotnet/core/blob/main/release-notes/6.0/supported-os.md

[96] PETR PUŠ. Poznáváme C# a Microsoft. NET 52. díl – ThreadPool – Živě.cz. *Živě.cz* [online]. [vid. 2025-03-30]. Dostupné z: https://www.zive.cz/clanky/poznavame-c-a-microsoft-net-52-dil--threadpool/sc-3-a-128106/default.aspx

[97] Task-based asynchronous programming - .NET | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-03-30]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming

[98] ThreadPool Class (System.Threading) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-03-30]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=net-9.0

[99] NuGet Gallery | CommunityToolkit.Mvvm 8.2.2. *NuGet* [online]. [vid. 2025-03-16]. Dostupné z: https://www.nuget.org/packages/CommunityToolkit.Mvvm/8.2.2?_src=template

[100] TextBlock Class (System.Windows.Controls) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-03-31]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.windows.controls.textblock?view=windowsdesktop-9.0

[101] Label Class (System.Windows.Controls) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-03-31]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.windows.controls.label?view=windowsdesktop-9.0

[102] esp8266/Arduino: ESP8266 core for Arduino. *GitHub* [online]. [vid. 2025-03-19]. Dostupné z: https://github.com/esp8266/Arduino

[103] *26.1 — Template classes – Learn C++* [online]. [vid. 2024-09-19]. Dostupné z: https://www.learncpp.com/cpp-tutorial/template-classes/

[104] *ArduinoJson: Efficient JSON serialization for embedded C++* [online]. [vid. 2025-03-20]. Dostupné z: https://arduinojson.org/?utm_source=meta&utm_medium=library.properties

[105] *How to upgrade from ArduinoJson 6 to 7 - YouTube* [online]. [vid. 2024-07-21]. Dostupné z: https://www.youtube.com/watch?v=eE6_77YIkzI

[106] *adafruit/Adafruit_Sensor: Common sensor library* [online]. [vid. 2025-03-24]. Dostupné z: https://github.com/adafruit/Adafruit_Sensor

[107] *adafruit/DHT-sensor-library: Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors* [online]. [vid. 2025-03-24]. Dostupné z: https://github.com/adafruit/DHT-sensor-library

[108] adafruit/Adafruit-GFX-Library: Adafruit GFX graphics core Arduino library, this is the „core" class that all our other graphics libraries derive from. *GitHub* [online]. [vid. 2025-03-24]. Dostupné z: https://github.com/adafruit/Adafruit-GFX-Library

[109] adafruit/Adafruit_SSD1306: Arduino library for SSD1306 monochrome 128x64 and 128x32 OLEDs. *GitHub* [online]. [vid. 2025-03-24]. Dostupné z: https://github.com/adafruit/Adafruit_SSD1306

[110] *Overview | DHT11, DHT22 and AM2302 Sensors | Adafruit Learning System* [online]. [vid. 2025-03-24]. Dostupné z: https://learn.adafruit.com/dht

[111] GM electronic | Modul teploměru a vlhkoměru s DHT11. *GME* [online]. [vid. 2025-03-24]. Dostupné z: https://www.gme.cz/v/1508421/modul-teplomeru-a-vlhkomeru-s-dht11

[112] OLED displej 0,96 palce. *GME* [online]. nedatováno [vid. 2025-03-24]. Dostupné z: https://img.gme.cz/files/eshop_data/eshop_data/9/772-153/dsh.772-153.1.pdf

[113] SG-90 servomotor 9g. *GME* [online]. [vid. 2025-03-28]. Dostupné z: https://www.gme.cz/v/1497888/sg-90-servomotor-9g

[114] Arduino/libraries/Servo/src at master · esp8266/Arduino. *GitHub* [online]. [vid. 2025-03-28]. Dostupné z: https://github.com/esp8266/Arduino/tree/master/libraries/Servo/src

[115] Low-code programming for event-driven applications : Node-RED. *Node-RED* [online]. [vid. 2025-03-28]. Dostupné z: https://nodered.org/

[116] Running Node-RED locally : Node-RED. *Node-RED* [online]. [vid. 2025-03-28]. Dostupné z: https://nodered.org/docs/getting-started/local

[117] Getting Started : Node-RED. *Node-RED* [online]. [vid. 2025-03-28]. Dostupné z: https://nodered.org/docs/getting-started/

[118] iot/Documentation/README.md at main · dotnet/iot · GitHub. *GitHub* [online]. [vid. 2025-03-28]. Dostupné z: https://github.com/dotnet/iot/blob/main/Documentation/README.md

[119] NANOFRAMEWORK. GitHub - nanoframework/Home: :house: The landing page for .NET nanoFramework repositories. *GitHub* [online]. [vid. 2025-03-28]. Dostupné z: https://github.com/nanoframework/Home

[120] MICROSOFT IOT DEVELOPERS a LAURENT ELLERBACH. IoT Show: An introduction to .NET nanoFramework - YouTube. *Youtube* [online]. [vid. 2025-03-28]. Dostupné z: https://www.youtube.com/watch?v=TLYqRdmmj5k

[121] Packaging : Node-RED. *Node-RED* [online]. [vid. 2025-03-28]. Dostupné z: https://nodered.org/docs/creating-nodes/packaging

[122] Creating your first node : Node-RED. *Node-RED* [online]. [vid. 2025-03-28]. Dostupné z: https://nodered.org/docs/creating-nodes/first-node

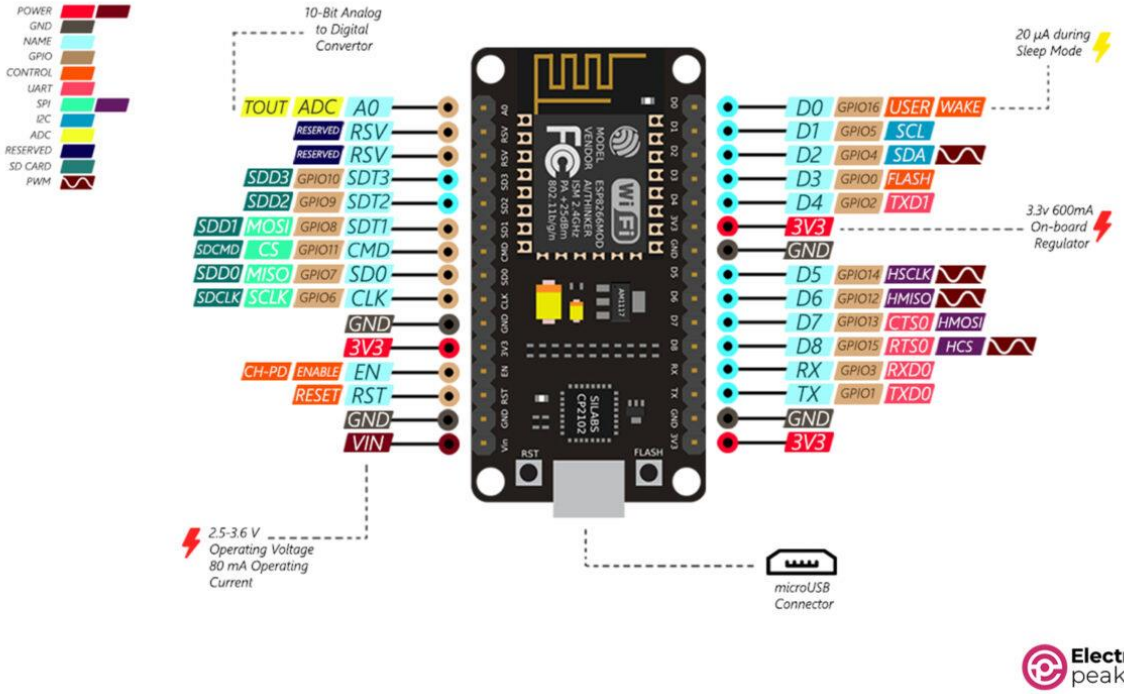
[123] *API | Developer Documentation | SmartThings* [online]. [vid. 2025-03-29]. Dostupné z: https://developer.smartthings.com/docs/api/public

[124] *32F429IDISCOVERY - Discovery kit with STM32F429ZI MCU * New order code STM32F429I-DISC1 (replaces STM32F429I-DISCO) - STMicroelectronics* [online]. [vid. 2025-03-29]. Dostupné z: https://www.st.com/en/evaluation-tools/32f429idiscovery.html

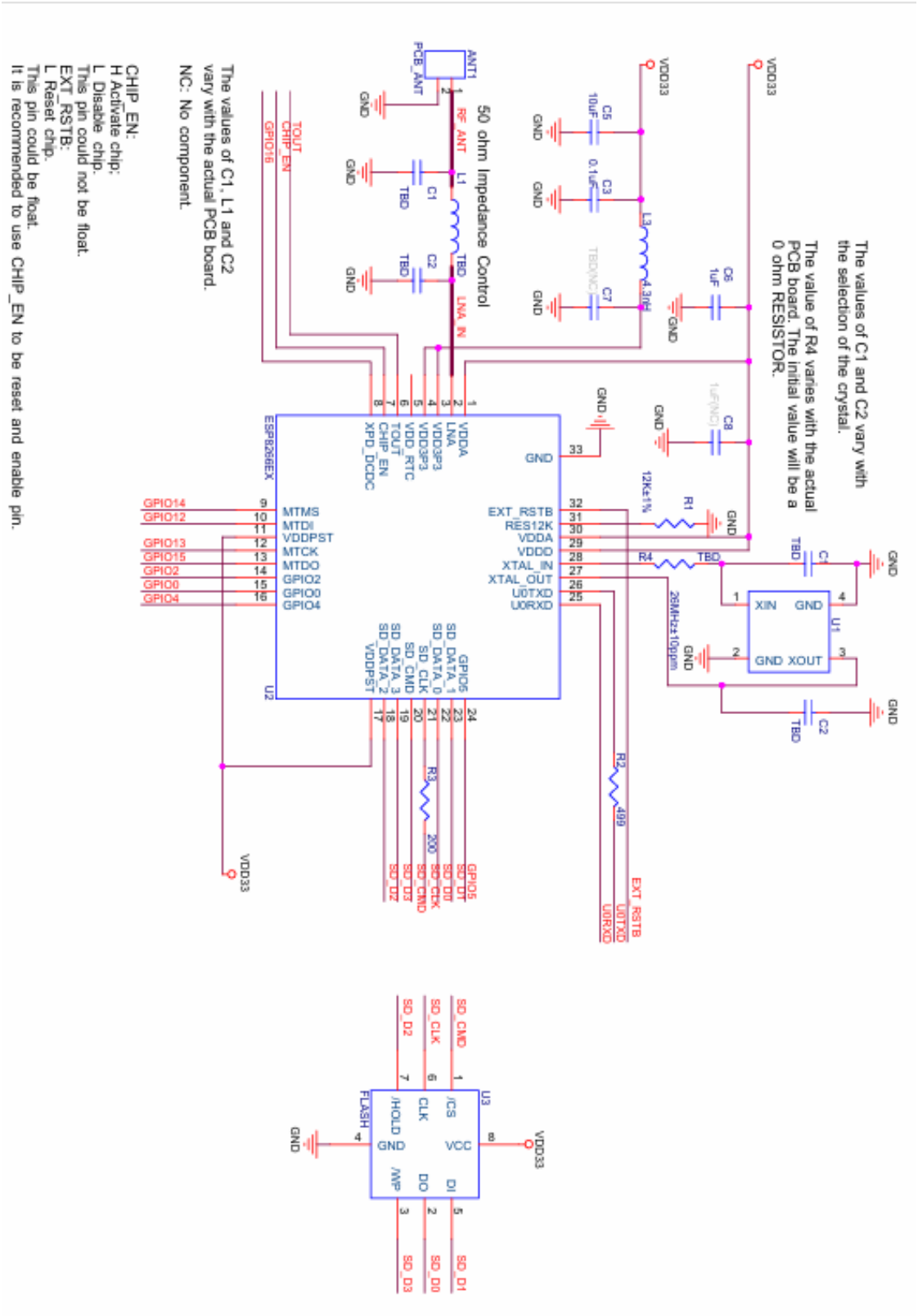
[125] *NUCLEO-H755ZI-Q - STM32 Nucleo-144 development board with STM32H755ZI MCU, SMPS, supports Arduino, ST Zio and morpho connectivity - STMicroelectronics* [online]. [vid. 2025-03-29]. Dostupné z: https://www.st.com/en/evaluation-tools/nucleo-h755zi-q.html

[126] *ESP8266 Pinout Reference: How To Use ESP8266 GPIO Pins* [online]. [vid. 2025-03-19]. Dostupné z: https://electropeak.com/learn/esp8266-pinout-reference-how-to-use-esp8266-gpio-pins/

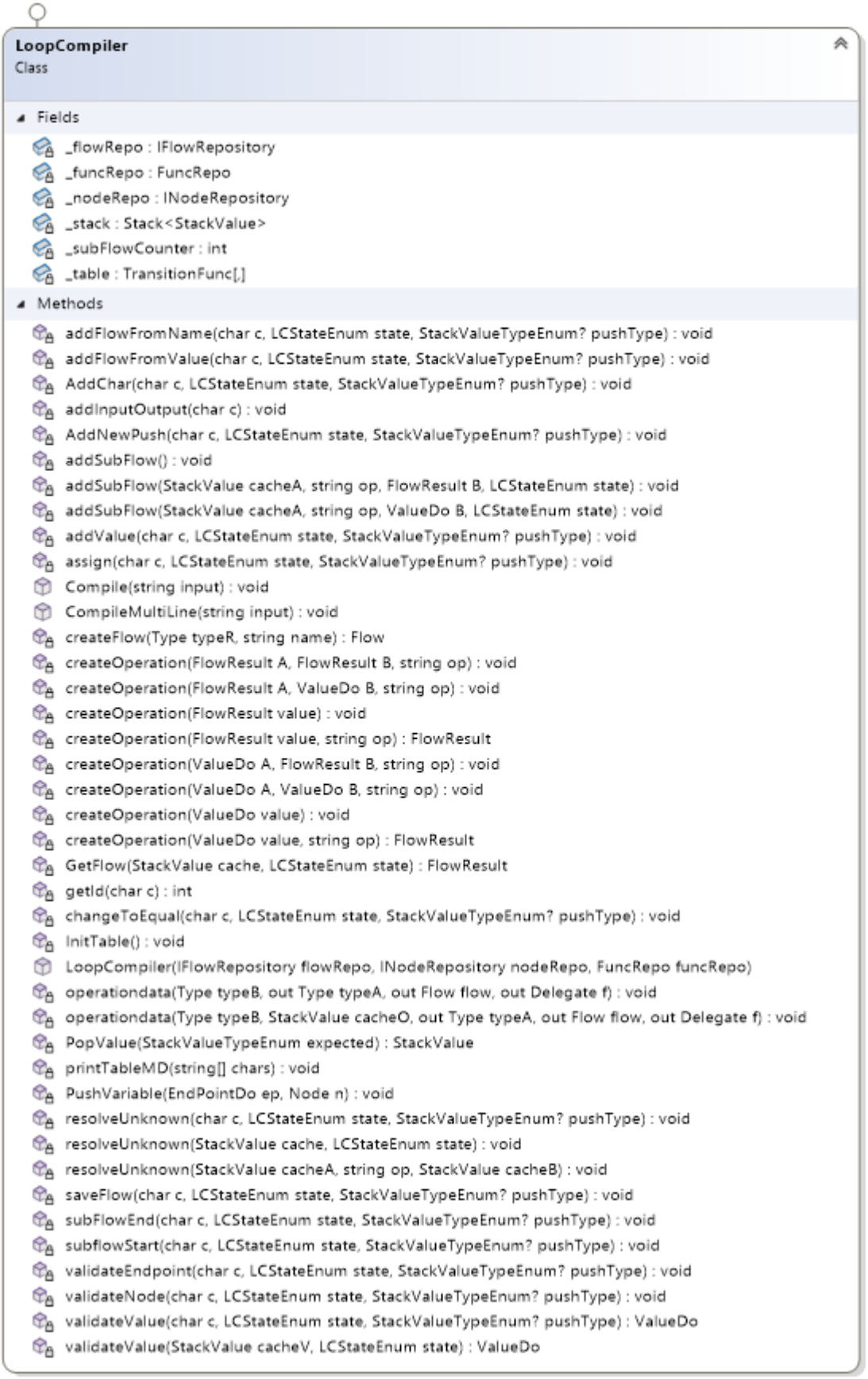
8 Přílohy



Příloha 1 Piny NodeMCU [126]



Příloha 2 Schéma zapojení ESP8266EX [78]



Příloha 3 Diagram tříd LoopCompiler

____	NULL	NODE	ENDPOINT	VALUE	DOT_EP	DOT_VAL	OPERATOR	UNKNOWN	FLOW	SUBFLOW	EQUALS_SIGN
Ø	-	-	-	VALUE	-	-	-	VALUE	-	-	-
A-Z a-z	UNKNOWN	NODE	ENDPOINT	VALUE	ENDPOINT	VALUE	UNKNOWN	UNKNOWN	FLOW	-	UNKNOWN
0-9	VALUE	NODE	ENDPOINT	VALUE	ENDPOINT	VALUE	VALUE	UNKNOWN	FLOW	-	VALUE
.	-	DOT_EP	DOT_VAL	VALUE	-	-	-	DOT_EP	-	-	-
(	NULL	-	-	NULL	-	-	-	NULL	-	-	NULL
)	-	-	-	UNKNOWN	-	-	-	UNKNOWN	-	-	-
+-*/	-	-	-	NULL	-	-	-	NULL	-	-	-
&|	-	-	-	NULL	-	-	-	NULL	-	-	-
!	OPERATOR	-	-	OPERATOR	-	-	-	OPERATOR	-	-	OPERATOR
<	-	-	-	OPERATOR	-	-	-	OPERATOR	-	-	-
>	-	-	-	OPERATOR	-	-	-	OPERATOR	-	-	-
=	-	-	-	EQUALS_SIGN	-	-	NULL	EQUALS_SIGN	EQUALS_SIGN	-	NULL
	-	-	-	-	-	-	-	FLOW	-	-	-

Příloha 4 Tabulka stavů konečného automatu