Česká zemědělská univerzita v Praze

Technická fakulta

Katedra elektrotechniky a automatizace



Diplomová práce

Návrh a realizace kontrolního systému na WiFi síti

Martin Novák

© 2024 ČZU v Praze!!!

Místo tohoto textu vložte PŘEDNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem TF.

!!!

!!!

Místo tohoto textu vložte ZADNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem TF.

V případě, že Vaše zadání je na více než 2 strany, vložte i další strany.

!!!

**Čestné prohlášení**

Prohlašuji, že svou diplomovou práci "Návrh a realizace kontrolního systému na WiFi síti" jsem vypracoval(a) 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(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Poděkování**

Rád(a) bych touto cestou poděkoval(a) jméno vedoucího, případně dalších osob, a informace, za co děkujete.

Návrh a realizace kontrolního systému na WiFi síti

**Abstrakt**

Souhrn práce (cca 15 řádek textu).

**Klíčová slova:** klíčová slova (cca 10)

**Design and implementation of a control system on a WiFi network**

**Abstract**

Anglický překlad českého souhrnu

**Keywords**: klíčová slova anglicky

**Obsah**

[1 Úvod 1](#_Toc191129027)

[2 Cíl práce a metodika 2](#_Toc191129028)

[2.1 Cíl práce 2](#_Toc191129029)

[2.2 Metodika 2](#_Toc191129030)

[3 Přehled řešené problematiky 3](#_Toc191129031)

[3.1 ARM 3](#_Toc191129032)

[3.1.1 ESP 8266 3](#_Toc191129033)

[3.2 ISO model 3](#_Toc191129034)

[3.2.1 Fyzická vrstva 4](#_Toc191129035)

[3.2.2 Linková vrstva 4](#_Toc191129036)

[3.2.3 Síťová vrstva 5](#_Toc191129037)

[3.2.4 Transportní vrstva 5](#_Toc191129038)

[3.2.5 Relační Vrstva 5](#_Toc191129039)

[3.2.6 Prezentační vrstva 5](#_Toc191129040)

[3.2.7 Aplikační vrstva 6](#_Toc191129041)

[3.3 Protokoly 6](#_Toc191129042)

[3.3.1 IP 6](#_Toc191129043)

[3.3.2 DHCP 6](#_Toc191129044)

[3.3.3 UDP 7](#_Toc191129045)

[3.3.4 TCP 7](#_Toc191129046)

[3.3.5 HTTP a HTTPS 8](#_Toc191129047)

[3.4 Datové formáty 10](#_Toc191129048)

[3.4.1 XML 10](#_Toc191129049)

[3.4.2 JSON 11](#_Toc191129050)

[3.4.3 CSV 11](#_Toc191129051)

[3.5 Wi-Fi 11](#_Toc191129052)

[3.6 Návrhové a architektonické vzory 11](#_Toc191129053)

[3.6.1 Zapouzdření 12](#_Toc191129054)

[3.6.2 N-vrstvá architektura 12](#_Toc191129055)

[3.6.3 Dependency injection 12](#_Toc191129056)

[3.6.4 Data Transfer Object (DTO) 13](#_Toc191129057)

[3.6.5 MVVM 13](#_Toc191129058)

[4 Vlastní řešení 15](#_Toc191129059)

[4.1 Hlavní uzel 15](#_Toc191129060)

[4.1.1 Komunikační vrstva 15](#_Toc191129061)

[4.1.2 Logická vrstva 15](#_Toc191129062)

[4.1.3 Uživatelské rozhraní 15](#_Toc191129063)

[4.2 Uzly 15](#_Toc191129064)

[4.2.1 Uzel 1 15](#_Toc191129065)

[4.2.2 Uzel 2 15](#_Toc191129066)

[4.2.3 Uzel 3 15](#_Toc191129067)

[5 Výsledky a diskuse 16](#_Toc191129068)

[5.1 Podkapitola úroveň 2 16](#_Toc191129069)

[5.1.1 Podkapitola úroveň 3 16](#_Toc191129070)

[5.1.2 Podkapitola úroveň 3 16](#_Toc191129071)

[5.2 Podkapitola úroveň 2 16](#_Toc191129072)

[6 Závěr 17](#_Toc191129073)

[7 Seznam použitých zdrojů i](#_Toc191129074)

[8 Přílohy v](#_Toc191129075)

**Seznam obrázků**

Odkazovaný seznam obrázků

**Seznam tabulek**

Odkazovaný seznam tabulek

**Seznam použitých zkratek**

Soupis a definování zkratek (vyskytuje-li se jich v textu velké množství)

# Úvod

Text text text text text text text text text text text text text text text text text text text text text text text.

# Cíl práce a metodika

## Cíl práce

Text…

## Metodika

Text…

# Přehled řešené problematiky

## ARM

ARM (Advanced RISC Machine) je

### ESP 8266

Text…

## ISO model

OSI (Open System Interconnection) model 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 7.2), která je vyobrazena na Obr. 1 během posílání HTTP (viz kapitola 5.5) 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. -OSI model[5]

### 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. Fyzická topologie je dána tím, jak jsou zařízení, označovaná jako uzly, vzájemně propojená. Ta se nemusí shodovat s logickou topologii, která je daná datovými toky. Ty mohou být všesměrové, nebo jeden s co nejmenším počtem prošlých uzlů potřebných do cílové destinace.[3, 6]



Obr. -typy topologií[6]

### Linková vrstva

Tato vrstva je zodpovědná za to, aby data dorazili do správného koncového zařízení. Kromě toho kontrolují, že data dorazili 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 využívá MAC (media access control) adresy. Dále tato vrstva má na starosti řízení datového toku, což zahrnuje určování velikosti jednotlivých rámců a určení, které zařízení momentálně řídí komunikaci.[1, 3, 7]

CRC je algoritmus sloužící 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 jednou pouze o detekční mechanismus, ale ne o sebe opravný kód. Množství chyb, které je možné detekovat, je určen zvoleným charakteristickým polynomem.[8]

Podoba rámce a velikost jeho datové část je dána použitou fyzickou vrstvou (viz 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. 4.3) na menší části se snižuje pravděpodobnost kolize na přenosovém mediu.[9]



Obr. rámec 802.3 vs 802.11[9]

### 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. Na této vrstvě pracují routery a switche. [1, 3, 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 pouze 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 [10]. 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. [1, 3, 4]

### Relační Vrstva

Úkolem této 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]

### Prezentační vrstva

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]

### Aplikační vrstva

Tato vrstva je nejblíže uživateli a umožňuje aplikacím volám 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]

## 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). Je nutné dělat kompromisy mezi spolehlivostí a rychlostí přenosu.[11]

### IP

IP (Internet Protocol) je protokol třetí vrstvy (viz kap. 4.3) OSI modelu, sloužící k směrování packetů napříč sítí. Pro tento účel slouží 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 zapisována 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. [12, 13]

Aby mohl internet dále fungovat bylo potřeba udělat opatření, které sníží počet potřebných adres na internetu. Tím že seznamu všech možných adres část vyhradí se pro podsítě, se umožňující, aby se tyto adresy opakovali. K určení, zda jsou zařízení ve stejné podsíti, slouží masky, které v binárním zápisu mají v místě, kde se musí shodovat 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. [14, 15]

### DHCP

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. [16]

### UDP

UDP (User Datagram Protocol) je jeden ze dvou hlavních protokolů čtvrté vrstvy (viz kap. 4.4) OSI modelu, 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, video hovory 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. [17]

### TCP

TPC (Transmission Control Protocol) je druhým z hlavních protokolů čtvrté vrstvy OSI modelu. 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 nedorazí potvrzení, předpokládá se, že byl paket ztracen a je zopakování 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*.[18, 19]



Obr. třístupňové ověřování [18]



Obr. TCP hlavička [18]

### HTTP a HTTPS

HTTP (Hypertext Transfer Protocol) je protokolem sedmé vrstvy (kap. 4.7) OSI modelu, který 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 rozšířitelný protokol, což kromě umožňuje kromě textu posílat i obrázky, videa a další datové soubory. Nové funkce lze snadnou doplnit přidáním nového atributu do hlavičky dotazu. HTTP je bez stavový 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 5.4) protokol, kvůli vytváření spojení. [20]

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 [21]. 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 [22]. 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. [20]

HTTP/1.1 a starší verze 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 o jaká metoda se provést. Nejběžnější jsou GET pro načtení dat a POST pro odeslání dat v těle dotazu. Cesta 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 zprávy. 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 znače přesměrování dotazu jinam. Čtyřka znamená chybu na straně klienta, zatímco pětka je chyba na straně serveru. [20, 23, 24]

|  |  |
| --- | --- |
| Obr. HTTP-dotaz [20] | Obr. HTTP-odpověď [20] |

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 založeno na asymetrické kryptografii, kdy data zašifrovaná pomocí veřejného klíče mohou být dešifrována pouze soukromím klíčem [25]. Aby bylo možné navázat spojení musí server mít platní certifikát vystavený nezávislou certifikační agenturou. V opačném případě klient ukončí komunikaci. Během navazování spojení je proveden TLS handshake, který ve verzi 1.2 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 použití zprávy odchycené v minulosti. 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 soukromím klíčem. Server dále pošle zprávu, kterou oznamuje, že poslal všechny potřebné údaje. Klient pošle svůj premaster secret, oznámení konce nešifrované komunikace a zašifrované shrnutí dosavadní komunikace. Server také pošle zašifrované shrnutí. 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. [26–31]



Obr. HTTPS komunikace [27]

## 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. [32]

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. [33]

### XML

XML (eXtensible Markup Language) je značkovací jazyk, popisující strukturu dat. Oproti některým jiným značkovacím jazykům neobsahuje informaci o jejich významu. Význam musí znát aplikace, což znamená, že XML vytvořený jedním programem, může být pro jiný nečitelný. Dokumenty obvykle začínají nepovinou 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. [33]

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). [34]

### JSON

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 závislí na konkrétním jazyku, což ho dělá ideální volbou pro sdílení dat mezi programy napsaných v různých technologiích. Za nevýhodu by se dalo označit nemožnost používat komentáře. [35, 36]

### CSV

CSV (Comma-Separated Values) je formát 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 (například kvůli českým desetinným číslům s desetinou čárkou místo tečky jako se používá v angličtině) je nutné použít jiný oddělovací znak (obvykle středník nebo svislítko) [37]. První řádek se většinou využívá k pojmenování jednotlivých sloupců. Oproti ostatním formátům má výhodu v menší velikosti, neboť význam hodnoty je definován pouze jednou nikoli pro každou instanci. Toto sebou ovšem nese nevýhodu, že jeden soubor může obsahovat pouze záznamy stejného typu, jelikož v opačném případě se nebudou shodovat sloupečky. 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. [38]

## Wi-Fi

text

## 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, nebo její komunikaci s ostatními. Oproti tomu Architektonické vzory určují sktrukturu celého projektu a mají přímý vliv na jeho modularitu a škálovatelnost. [39–41]

### 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 mohli komunikovat nepotřebují znát vnitřní fungování, ale pouze rozhraní.[42]

### 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 kombinace 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 čili případný útočník nemůže z nejvyšší vrstvy přistupovat přímo k nejnižší. Rozdělení vrstev sebou však nese komplikaci v podobě komunikace mezi nimi.[42, 43]

Nejběžnější je třívrstvá architektura. Nejvyšší vrstva komunikuje s uživatelem a podle typu aplikace se jedná o uživatelské rozhraní, nebo v případě API o 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.[42]

### Dependency injection

Dependency injection je technika, která snižuje závislost třídy na jiné. Toto umožňuje aplikaci být více modulární, lépe testovatelná a snáze upravitelná.[44]

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)[45]. 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.[44]

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.[44]

Pro drobné projekty může jako injector sloužit prosté zavolání konstruktoru z kódu[44]. 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#[46]. 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.[47]

### 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 neměli by obsahovat žádnou logiku.[48, 49]

### 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.[50]

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. 1 jednotlivé části jsou úzce provázány, což komplikuje testovatelnost a úpravy.[50, 51]



Obr. datový tok MVC [50]

Většinu problémů MVC řeší MVP (Model-View-Presenter), kde view a model nekomunikují napřímo, ale přes presenter jako prostředníka (viz Obr. 2). 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.[50, 51]



Obr. datový tok MVP [50]

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. 3). 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.[50, 51]



Obr. datový tok MVVM [50]

# Vlastní řešení

## Hlavní uzel

Hlavní uzel je realizován jako počítačový program. Řešení je rozděleno na tři části, které řeší komunikační, logickou a uživatelskou vrstvu. Každá vrstva má referenci jen na vrstvu pod ní. Toto řešení umožňuje snadnou změnu jednotlivých částí, bez výrazných zásahů do kódu.

### Komunikační vrstva

Text…

### Logická vrstva

Text…

### Uživatelské rozhraní

Text

## Uzly

Jednotlivé uzly jsou tvořeny jednočipovými počítači ESP8266.

### Uzel 1

text

### Uzel 2

text

### Uzel 3

text

# Výsledky a diskuse

## Podkapitola úroveň 2

Text…

### Podkapitola úroveň 3

Text…

### Podkapitola úroveň 3

Text…

## Podkapitola úroveň 2

Text…

# Závěr

Text…

# 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] 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

[11] 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

[12] 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

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

[14] 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/

[15] 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

[16] 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

[17] 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/

[18] *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

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

[20] 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

[21] 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

[22] 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/

[23] 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

[24] 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/

[25] 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/

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

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

[28] 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

[29] 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/

[30] 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/

[31] 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/

[32] 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

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

[34] API pro XML. *TUL* [online]. nedatováno [vid. 2025-02-20]. Dostupné z: https://www.nti.tul.cz/~satrapa/vyuka/xml/prednaska12.pdf

[35] 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

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

[37] 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

[38] 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

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

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

[41] 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/

[42] 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

[43] 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

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

[45] 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/

[46] 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

[47] 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

[48] 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

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

[50] 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/

[51] 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

# Přílohy

Odkazovaný seznam příloh